Constructing 3D Faces from Natural Language Interface

Salman Ahmad
Department of Computer Science
Loughborough University

A Doctoral Thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University

ABSTRACT

This thesis presents a system by which 3D images of human faces can be constructed using a natural language interface. The driving force behind the project was the need to create a system whereby a machine could produce artistic images from verbal or composed descriptions. This research is the first to look at constructing and modifying facial image artwork using a natural language interface.

Specialised modules have been developed to control geometry of 3D polygonal head models in a commercial modeller from natural language descriptions. These modules were produced from research on human physiognomy, 3D modelling techniques and tools, facial modelling and natural language processing.

This work uses two main methods sequentially for synthesising 3D facial images from natural language descriptions:

1. Use of a fuzzy truth maintained blackboard system to interpret and translate linguistic data which produces parameters for free form deformation modifiers to parameterise and control pre-constructed 3D head models.
2. A commercially available 3D modelling system which has pre prepared scripts to access and control head templates and modifiers obtained from measurements of 3D human heads.

A novel method of abstracting standard face images, modifiers and hedges is described. Base head templates are obtained by distilling out the modifiers, modifiers are obtained by differencing the modified object from a base template.

After an initial description, amplifying statements may be added to refine the facial image and are blended with the original description.

The interpretation of the statements is based on baseline head models and modifiers taken from measurements of human heads and so the library of head geometry and modifiers provides context within which the statements are given form. By changing the set of head models and associated descriptions, the context may be changed. The interpretation of the natural language is thus based on the experience of the machine; and may arguably be termed "artistic". The resultant facial images are consistent with the descriptions although it has proved difficult to obtain detailed descriptions of faces that result in a recognisable match. The work has shown that it is possible to derive images that match the descriptions but that the descriptions used are insufficient to completely describe a given face. The derived templates and modifiers influence the set of faces produced from any given set of descriptions, and form the basis by which the system interprets the natural language statements.

Keywords: Natural Language, 3D Facial Images, 3D Modelling, Interface, Uncertainty, TMS, Blackboard, Fuzzy Logic, Facial Modelling

Acknowledgements

This work has been carried out by the Author under the supervision of Dr. Chris J. Hinde in the Department of Computer Science at Loughborough University.

I would like to thank Chris for the past years of help and encouragement. He has been more like a colleague and friend, guiding and helping me whenever and wherever I needed it. Without his skill and determination this thesis would not be possible.

I would also like to thank Dr. Helmut Bez for guiding me in the construction of the 3D head models and many other stages throughout the development process. I would like to thank other colleagues in the computer science department namely; John, Irene, Jo, Judith, Margaret and Steve for their help and support.

I would like to thank my family for their continuous support and encouragement, especially my parents who convinced me to perform a Ph.D. and have been a source of inspiration and strength.

Lastly I would like to thank my friends Tom, Manu, Anita and Aziz with whom I immensely enjoyed working and playing.

Contents

Chapter 1

Introduction

Abstract

This chapter provides a description of the research aim and outlines the methodology, approach, and process used to tackle the research problem. The chapter outlines the remaining chapters of the thesis with a brief note on what each chapter entails.

Keywords: Facial Composite, Fuzzy Logic, Human Computer Interface, Natural Language.

1.1 Introduction

Almost 20 years ago an interface redesign revolutionised the way computers would be used. The WIMP (Windows Icons Menus Pointers) environment replaced the command line interface and ushered a new era of mass computer usability. With time the WIMP environment evolved into advanced Graphical User Interface (GUI) that is commonly found in popular computer systems today. The drive behind the redesign and improvement of Human Computer Interface (HCI) was primarily "Ease of use". Work continues on improving existing graphical user interfaces to find the ultimate HCI set-up.

Whereas improvements in the area of HCI have been in the form of advanced GUI, hardware and software solutions. An important area of study is to make computers understand our language and communicate in a natural albeit structured form. This is where Natural Language Processing (NLP) comes in and it is hoped that break through in NLP will lead to implementation of a natural language interface for controlling all functions of a computer system.

1.2 Thesis Aim

We as humans, have a fantastic ability for recognising visual patterns. At a single glance we can absorb and process a huge amount of information about our present environment. We can look at an object, instantly recognise it and perhaps even give a good verbal description of it. But if recognition is one side of the coin, then generation is the other side. And yet how many of us have the ability to successfully generate recognisable images of objects, specifically to sketch them? Furthermore, how many of us are capable of determining which aspects of an object are information bearing in terms of recognition and then reducing them into a collection of lines.

Consider the human face. Humans can distinguish quite well among faces, even though all faces have the same basic features that appear in roughly the same relative position. Nevertheless those among us who lack artistic talent would be hard pressed to sketch a recognisable image of a face.

A broad, albeit vague, definition of artificial intelligence is anything done by a computer which would be called intelligent if done by a human. The ability to perform an image generation task or to instruct one through such a task is intelligent behaviour. Thus if a machine could be made to perform like functions via instructions in words by end users it would be demonstrating a form of AI. Genesereth and Nilsson contend that "Artificial intelligence is the study of intelligent behaviour. Its ultimate goal is a theory of intelligence that accounts for the behaviour of naturally occurring intelligent entities, and that guides the creation of artificial entities capable of intelligent behaviour" (Genesereth and Nilsson, 1987).

This thesis explores the problem of generating recognisable 3D facial images through natural language descriptions. The research comprises of developing a system which can generate 3D facial models not using the usual WIMP control environment as seen in the widely available hybrid modellers such as *Alias Wavefront*, *SoftImage*, *3D Studio Max*, *Unigraphics* etc but rather a Natural Language Interface.

Humans have a remarkable ability to recognise objects, features, faces they see and can reasonably describe them verbally. Whereas the professional modelling packages listed above are adequate tools for the artistically talented and technically skilled individuals, they are by no means useful for people inept with using graphic packages.

A good example of this can be observed with the 3D rendered image shown in Figure 1.1, generated using *SoftImage*. The Artist Jeremy Birn in his tutorial demonstrates the painstaking process involved in generating the image.

The aim of this thesis is to explore whether, firstly 3D human face models can be constructed and modified using a rudimentary natural language interface and secondly the facial images constructed can pass as recognizable human faces. To this end the work carried out and the system developed is reported in this thesis.

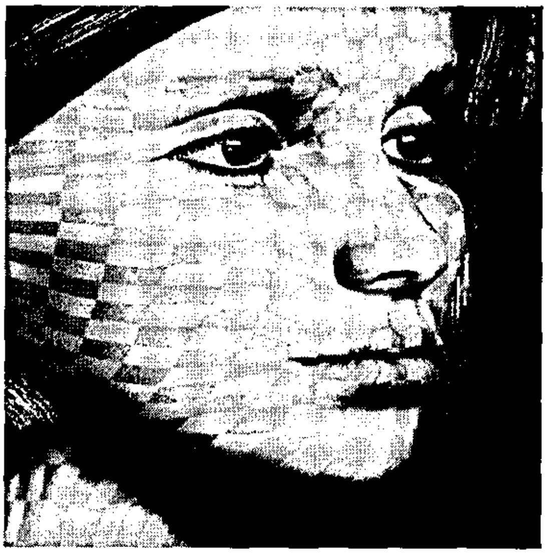


Figure 1.1 "This is a head I modelled in the spring of 1996. The grid shows the UV parameterisation of the NURBS surface", Birn J.

Source: (http://www.3drender.com//jbirn/ea/HeadModel.html)

1.3 Approach

Given the conceptual framework outlined in section 1.2. Development of the entire system was divided into three broad areas of research or study.

1. Natural language Interface Module using a Fuzzy Truth Maintained Blackboard System
2. Head Engine Module using fuzzy logic to translate linguistic data
3. Facial Image Generation Module

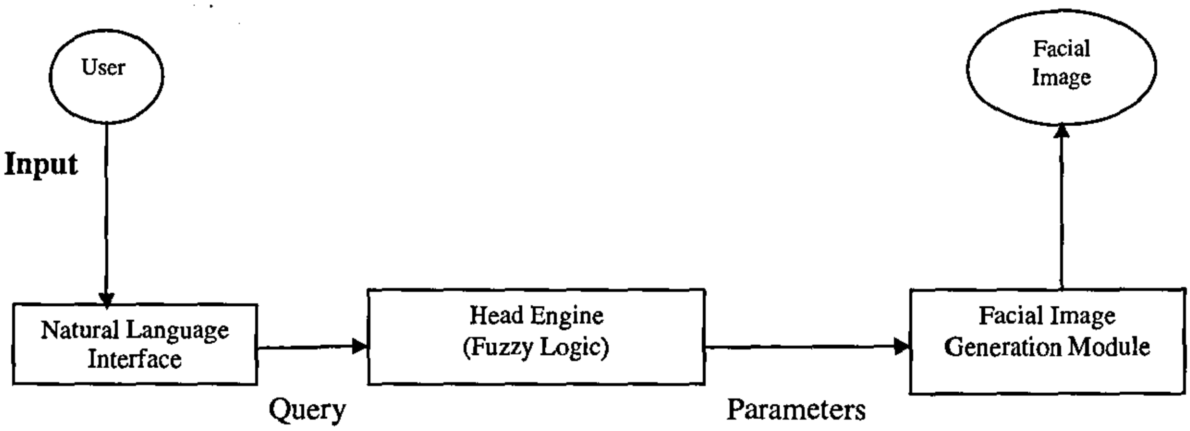

User
Input
Natural Language Interface
Query
Head Engine (Fuzzy Logic)
Parameters
Facial Image Generation Module
Facial Image

Figure 1.2 Diagram of a broad overview of the proposed research aim.

The diagram in Figure 1.2 shows the flow of data and processes required to successfully achieve the project aims.

Starting with the user entering a textual description of a face, the back end of the Natural Language Interface (NLI) processes the English descriptions to smaller descriptive phrases. These phrases filter through a Fuzzy Truth Maintenance System (TMS) that converts the descriptors to numerical parameters. The parameters are finally read by a 3D modelling script called Head Generator Script (HGS), which generates the 3D model of the requested face.

1.3.1 Natural Language Interface (NLI)

Natural Language Processing (NLP) is both a modern computational technology and a method of investigating and evaluating claims about human language itself. Some prefer the term Computational Linguistics in order to capture this latter function, but NLP is a term that links back into the history of Artificial Intelligence (AI), the general study of cognitive functions by computational processes, normally with an emphasis on the role of knowledge representations, that is to say the need for representations of our knowledge of the world in order to understand human language with computers.

NLP as a technology covers computer systems that require no knowledge of how they work to understand what they do: Machine Translation (MT) systems translate from one language to another; Information Extraction (IE) pulls facts and structured information from the content of large text collections; Human-computer conversation systems allow relatively straightforward communication with machines in English by means of speech or typing. These major systems require a set of rather similar subsystems, which are at the heart of NLP, with names like parsing, tagging, aligning, interpreting which can be carried out by methods that may be knowledge and rule-based, or based on statistics or some combination of the two.

The natural language interface module consists of a simple graphical user interface at the front end. The back end of the natural language interface is responsible for processing the natural language sentence of face descriptions. It handles the arduous task of interpreting and parsing sentences using knowledge base of lexical or vocabulary and rules of English grammar. English sentences are interpreted using a Truth Maintained Blackboard system. The interpreted sentence is stored as a PROLOG list of descriptors and processed by a heads engine constructed to translate and convert the descriptors to numerical parameters. The NLI and the TMS is discussed in detail in chapter 6.

1.3.2 Fuzzy Logic

Rules used by people use “linguistic” variables such as “much lower”, “a lot”, “a little”, which we need to interpret more precisely. For this we need to develop the idea of a fuzzy number. An approximation to a fuzzy number is such a method. By approximating a normal distribution with the view that almost any “reasonable” interpretation will give us “reasonable” results then we could take a much simpler approximation and be just as right or wrong (Zadeh L.A., 1965). The fuzzy membership function, distribution diagram in Figure 1.3 shows such an approximation.

Given the fuzzy membership function in Figure 1.3 as a definition of the concept “Wide” for eye spacing we can read off a grade of membership given a width and also read back a width given a grade of membership.

The concept of “Hedges” within the topic of fuzziness is an important one and highly relevant to the project in discussion. Apart from distributions such as “large”, “small”, “medium”, “wide”, etc. there could be other distributions derived from these such as “very wide” and “fairly wide”. These adjectives “very” and “fairly” are known as hedges and modify the distributions they are applied to.

The use of hedges enables finer distinctions in the sets to be derived and so allow better judgements to be made about which set something should be a member of. These sets are very useful in the area of fuzzy control and enable input values to be mapped onto fuzzy sets. Figure 1.4 to 1.6 show how hedges can affect the distribution of a set by strengthening or weakening the set

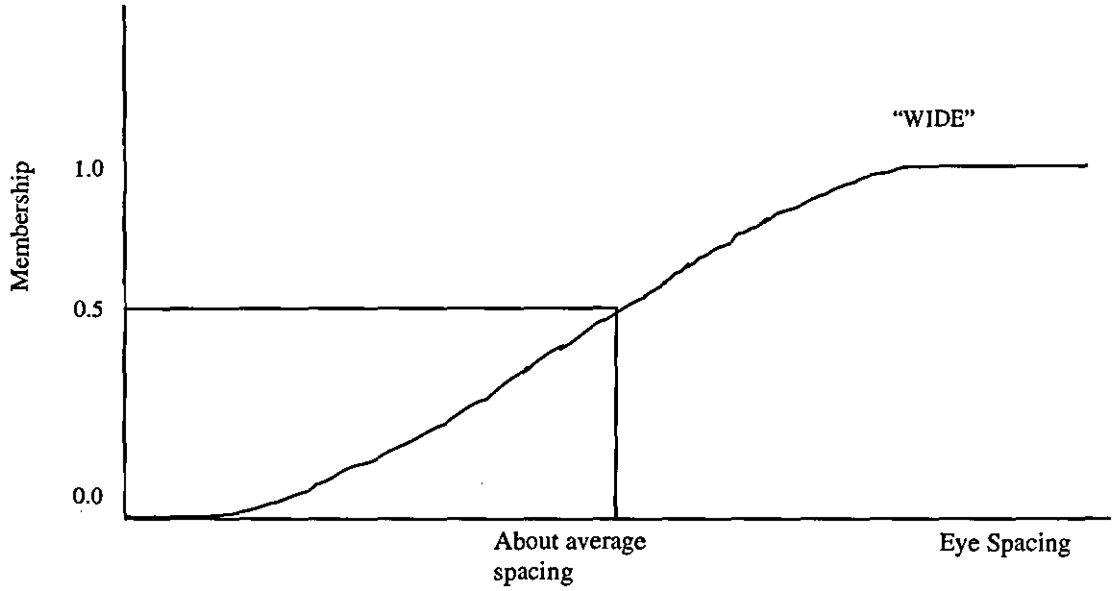


Figure 1.3 Shows a denotation of the membership function "WIDE" for eye spacing

The membership function for "VERY WIDE" might look like Figure 1.4 with "WIDE" shown for comparison.

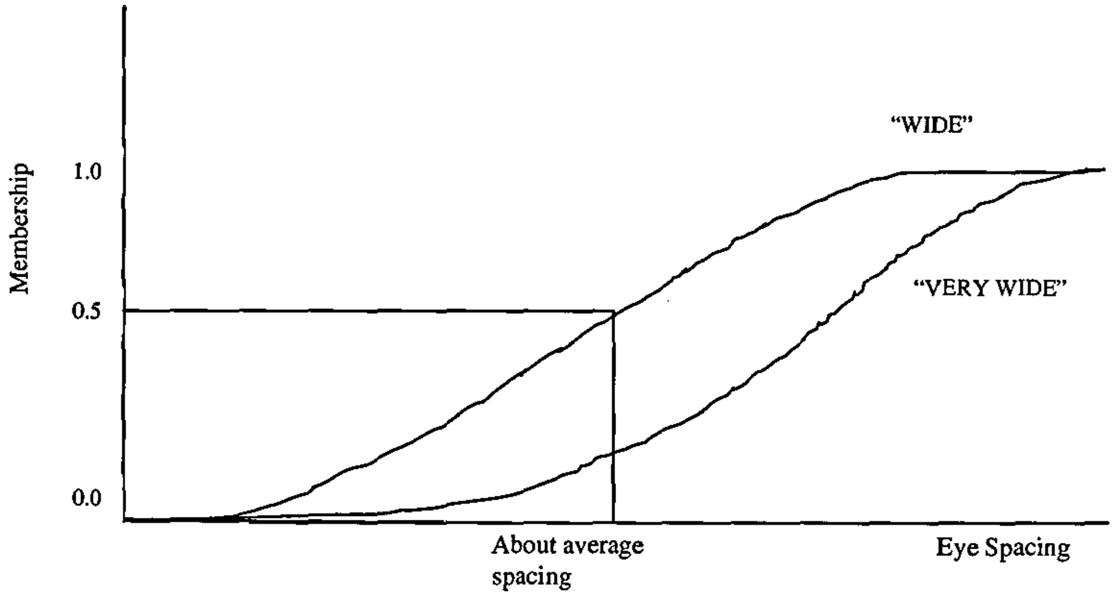


Figure 1.4 Shows membership function of sets "WIDE" and "VERY WIDE"

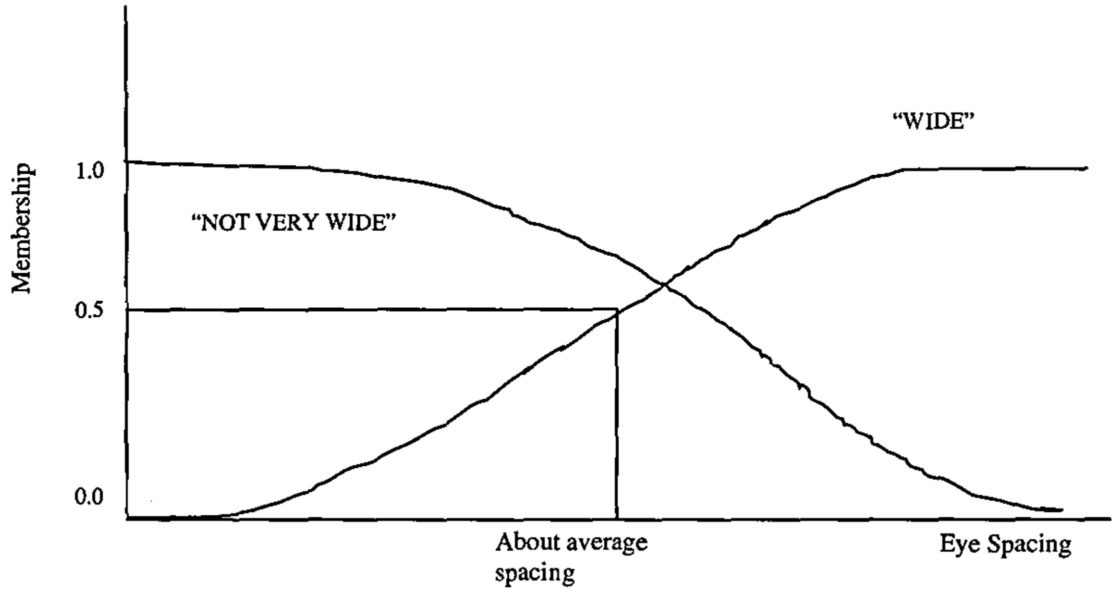


Figure 1.5 Shows the transformation NOT on "VERY WIDE". The intersection would constitute the set "WIDE" but "NOT VERY WIDE"

Using the intersection of "WIDE" and "NOT VERY WIDE" gives the fuzzy set corresponding most closely to "WIDE" so some one with the width within the triangle would have a description of "WIDE" but "NOT VERY WIDE". Similarly the set "WIDE" can be weakened to "FAIRLY WIDE" by applying "FAIRLY" membership hedge. Figure 1.6 shows the result of applying the hedge "FAIRLY" to the set "Eye Spacing".

Since these sets are difficult to describe accurately and precisely it is usual and computationally efficient to use triangular sets. As fuzzy distributions are generally used to describe vague and approximate concepts this is a reasonable decision with respect to the operation of the fuzzy system. Fuzzy set theory and other techniques for handling uncertainty are explained in greater detail in chapter 5.

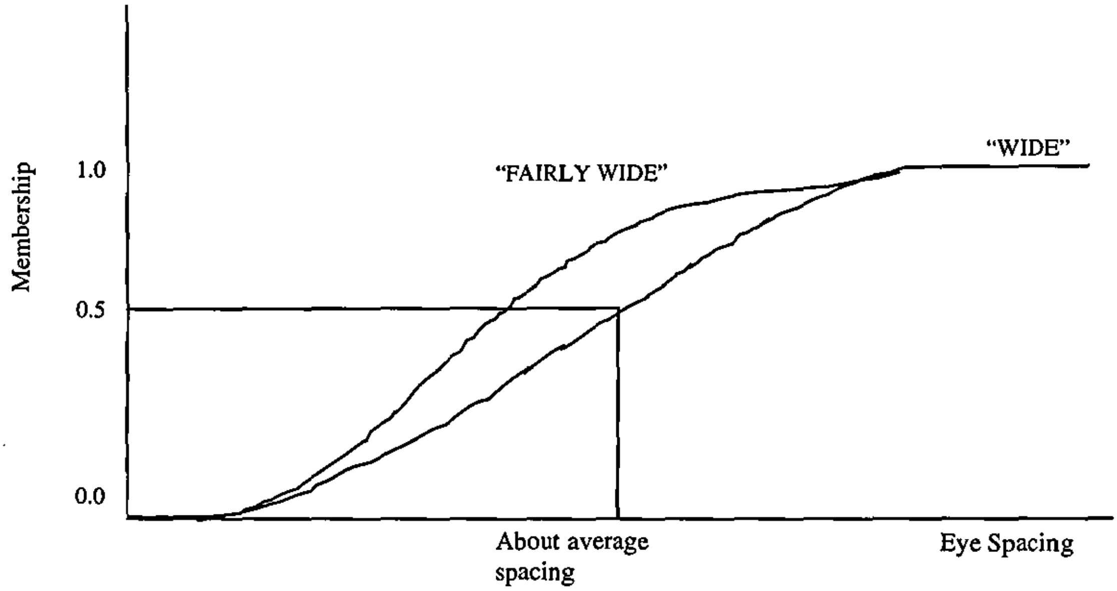


Figure 1.6 Shows the effect of the dilation operator "FAIRLY" on the set "WIDE"
Examining the transformation implied by the two sets "WIDE" and "VERY WIDE" could derive the operator "VERY". These sets are very useful in describing objects in a concise manner by selecting the most appropriate descriptor.

The heads engine uses fuzzy hedges to modify parameters of features that are influenced by classification of hedges. Chapter 6 describes how the head engine modifies parameters using hedges.

1.3.3 Facial Image Generation Module

A lot of research and time was spent on this module. The only sensible way to tackle the project was to build the facial image generation module first and then see how the remaining modules need to be planned to make it all fit together. Proceeding on a bottom up approach, extensive study of the human face, modelling procedures and modelling tools was undertaken. Details on human physiognomy, modelling techniques and tools are explained in chapter 2 and 3 respectively. Chapter 4 describes development of the facial image generation module.

1.4 Contributions of the Thesis

This thesis provides a novel approach to 3D image generation within the context of intelligent machine based control. There has been a great deal of work on facial image composition and generation in both 2D and 3D, however little or no research has been done on generating facial images via natural language descriptions. Most of this has been in the domain of forensic science "mug-shot search problem" (Cutler *et al.*, 1988; Baker E. & Seltzer M., 1997) and facial animation.

Substantive research in the real-time animation of faces for telecommunication and for the synthesis of computer interface "agents" is being conducted at Apple Computer, Inc. (Advanced Technology Group, Cupertino), Hitachi, Ltd. (Hitachi Central Research Laboratory, Japan), NTT (Human and Multimedia Laboratory, NTT Human Interface Laboratories, Japan), and Sony (Information Systems Research Center, Sony Corporation, Japan).

A number of companies are in the business of vending computer systems and services for making facial image composites ("identikit" police identification tools, point-of-purchase video preview for cosmetic makeovers or cosmetic surgery, and one class of systems for estimating the aged appearances of missing children), 3D digitization of faces, 3D reconstructive surgery preview and manufacture of facial prosthetics, 3D digitization of teeth for the manufacture of dental appliances, and 2D and 3D facial animation.

Other important current applications are in the entertainment industry; the use of graphical face models in advertising, for movie special effects, etc.

The interface for facial composition systems like the mug-shot search problem vary from an image based interface like CAFIIR (Wu *et al.*, 1994), which allows the user to construct a face from a database of feature parts by blending each part onto a facial image, to natural language text based where natural language queries are used to search a database of Images (Wu J. K., 1988).

None of the facial Image composition, animation and composite based Image retrieval systems have explored the idea of generating facial composites via a natural language Interface. Rama Bindiganavale reports work on altering agent behaviors using natural language instructions (Bindiganavale *et al.*, 2000). Although this is conceptually close to this research aims, it fails to offer any clues on how 3D head geometry of avatars or agents may be generated and controlled using a natural language interface. This thesis provides the methods and techniques, drawn from the knowledge available on existing work on facial animation and natural language processing, to develop a working model for the synthesis of 3D facial Imagery via natural language descriptions using a fuzzy logic based truth maintenance system.

As mentioned earlier the aim of this thesis is to explore whether firstly 3D human face models can be constructed and modified using a rudimentary natural language interface and secondly whether the facial images constructed can pass as recognizable human faces.

1.5 Outline of Thesis

Chapter 2 provides an insight into what makes faces recognisable; it starts with a description of the term human physiognomy, followed by explanation of the structure of human face and its aspects and features that make faces unique and complex. It moves on to look at existing research on how people attend to faces and describe a face and see which features they focus most in facial recognition and description. It finally looks at the language average people use to describe faces by analysing descriptions gathered from surveys and questionnaires.

Chapter 3 presents background and related work to facial modeling, facial image recognition and composition, techniques and procedures for constructing 3D facial geometry. This chapter provides an exposition of the tools and techniques used to establish a solid framework for the facial image generation system. The human head models developed using the tools and techniques defined in this chapter form the backbone for the automated facial image generation system. The chapter has been divided into three parts – the first dealing with facial modelling, existing research and applications, the second with representation and acquisition techniques available for 3D facial image composition and the third discusses the tools and technology used for developing the human head models in 3D.

In chapter 4 the 3D facial image generation module is described. It begins with describing the procedure for modelling a human head using representation techniques discussed in chapter 3, namely; NURBS, Bezier Patches and Polygon Meshes. This is followed by a description of the finalised baseline head and implementation of deformation controllers through out the geometry to control structure and conformation of the face and its features. Finally described is the parameterisation of the head model and method for influencing the parameters using Maxscript to form the facial image generation system.

In Chapter 5 is discussed the single major problem faced by designers and engineers of AI solutions i.e. Uncertainty. What uncertainty entails has been discussed within the domain of Knowledge-Based systems and techniques available for handling uncertainty. Furthermore the chapter looks in detail at the two main systems namely Fuzzy Logic and Truth Maintenance to deal with uncertainty in natural language descriptions.

Chapter 6 provides details on how natural language descriptions are interpreted and how we have dealt with translating linguistic datum to parameters for the facial image generation module. Obvious reference is made to techniques discussed in chapter 5 on handling uncertainty in natural language using Truth Maintenance System and fuzzy logic.

In Chapter 7 the overall system architecture is defined along with a working model with extensive test results showing if the system works, how effective it is and what are its shortfalls. Finally Chapter 8 offers a summary of thesis conclusions and outlines some outstanding research questions as well as suggests future research work.

Chapter 2

Describing Faces – Human Physiognomy, Facial Recognition and Verbal Descriptions

Abstract

This chapter provides an insight into what makes faces recognisable; it starts with a description of the term human physiognomy, followed by explanation of the structure of the human face and the aspects and features that make faces unique and complex. It moves on to look at existing research on how people attend to faces and describe them, and see which features they focus most in facial recognition and description. It finally looks at the language average people use to describe faces from photo-realistic images and examine the results to compile a list of most commonly used descriptors for the lexical database in the natural language processing engine.

Keywords: Conformation, Description, Facial Structure, Language, Physiognomy, Recognition.

2.1 Introduction

No other object in the visual world is quite so important to us as the human face. Not only does it establish a person's identity, but also, through its paramount role in communication, it commands our almost continuous attention. The significance of the face has long been a topic for speculation by philosophers and artists concerned with character and aesthetics. When William Hogarth wrote in his "Analysis of Beauty" (1753) that "The face is the index of the mind", he was voicing a fairly common belief of the time. But Hogarth also acknowledged another aspect of faces and our ability to discriminate them when he advocated a "methodical enquiry" into the observation that "out of the great number of faces that have been formed since the creation of the world, no two have been so exactly alike, but that the usual and common eye would discover the difference between them" (Davies G. M., 1981).

Our aptitude to remember and recognise faces is an amazing ability; however an interesting discovery is that most people struggle to recall the facial characteristics with enough detail to provide an accurate composite (Penry Jacques, 1971). In light of this knowledge it makes sense to use visual cues and images to aid the use of facial composite systems like Photofit (Davies G. M., 1981), E-Fit (Aspley, 1993), and Identi-kit (Laughery & Fowler, 1980) as used by law enforcement agencies. Implementation of visual cues and images to aid users is evident in systems like CAFIIR (Wu, Ang, Lam, Loh, & Desai, 1994), Mac-a-Mug (Cutler B. L., 1988), and Photobook (Baker & Seltzer, 1998). Where as the use of visual aids and images is a good practise to aid a person in recalling facial characteristics the bottom line is that people still use words to describe the basic elements (Christie & Ellis, 1981). Therefore it would make sense to research a system that allows words to be used to composite a human face. The application of such a system is not as limited or, in the case of this research, targeted as an ID Kit or Photofit system. Instead it is aimed to find out weather such a system is feasible.

In order to develop a query engine based on the language people use to describe faces a study of the human face, people's recognition and consequently linguistics in describing faces is important. This chapter explores the three aforementioned problems; it includes results of surveys carried out to compare descriptions of different people and compilation of vocabulary most commonly used when describing faces.

2.2 Human Physiognomy

A face consists of many parts and details. The term "physiognomy" refers to features of the face, especially so when these features are used to infer the relatively enduring character or temperament of an individual. In this thesis, this term connotes a simpler meaning, i.e., it refers to facial features that change slowly and relatively little over time and constitutes the structure and conformation of a face. Such features have the bony structure as their basis, from which experts can fairly accurately reconstruct the fleshy features.

Topics related to physiognomy have a very long history in human cultures. In China and other Asian cultures, formal systems of face reading techniques developed sometime in the first millennia, integrated with religious beliefs such as Confucianism. Substantial confidence in such methods developed in these cultures, and physiognomic inferences included descriptions of character, suitability for certain positions, and predictions about life and death. In Western cultures, the association of facial features with a person's characteristics also has a history, first noted in the writing of the ancient Greeks. Much later, several pseudo-scientific and cult movements exploited the inference of character from physiognomic features. The physiognomy movement (which cultivated the narrow connotation for the term) was Phrenology, popularized by the 18th century Swiss philosopher Lavater (FaceData, 1990).

The face, despite recent advances in assessing identity such as biometrics and DNA testing, remains paramount in ordinary experience for identifying an individual person. The relatively permanent features of the face convey most of the information about identity, although styles in the production of more transient signals and other body shapes and sizes may also contribute to identity information.

2.3 Properties and attributes of human physiognomy

2.3.1 Physical structures

The face is a complex biological structure. The overall shape of the face is determined by the underlying bone shapes of the skull and the mandible (jaw bone). The bones are generally considered to be rigid in most applications of facial modelling; however it is obvious that changes in shape must be accounted for in any application concerning modelling of children or of the growth process. From a physical point of view, it is also commonly noted in the medical community that soft tissue always shapes hard tissue -

that is to say that if bone is compressed by muscle actions, the bone will eventually be reshaped in response (Pelachaud, Badler, & Viaud, 1994).

The medical term "joint" refers to any region where two distinct bones come together. Several bone masses make up the skull, but by adulthood they have fused together to the extent that the jaw is the only feature of the face which fits our common sense definition of a joint as seen in other parts of the body. The jaw is referred to as the temporo-mandibular joint (TMJ). To a first approximation, the TMJ can be treated as a hinge joint. However, in practice it is important that the muscles control the lower jaw in all six degrees of freedom (this is particularly useful for producing grinding actions in chewing). Several layers of soft tissue cover the bones of the face. Although the tissues can be categorised by function and material content, in vivo the difference between layers of tissue is less distinct (in any given volume of tissue, there may be muscle fibers interspersed with the collagen network of the dermis).

The muscles of facial expression tend to be of the flat, diffuse variety-more like the smooth muscles of the gut than the cylindrical muscles used for locomotion and manipulation in the arms and legs. Whereas the cylindrical muscles have well defined origin and insertion points, the muscles of facial expression have broad attachment areas integrated in the tissue. There may be several layers of muscle fibers connected to the same part of the anatomy (for instance the levator labii and the risorius muscles both insert at the corner of the mouth and are involved in raising it, but they differ in origin). Such muscles may or may not always be independently controllable.

The mechanical behaviour, particularly the *Poisson effect* and the *elasticity*, of the skin and soft tissue is one of the primary determinants of the change of appearance with facial expressions. The Poisson effect describes the tendency of the material to preserve its volume when changing length. Since much of the mass in the soft tissue is water, the soft tissue is nearly incompressible. Thus when muscles cause a contraction along one axis, the face must bulge along another; since the underlying hard tissue forms a firm foundation, facial actions almost always cause the skin to bulge out from the face. This change in the surface becomes visible through changes in the silhouette edge of the face and through changes in the surface shading of the face. The other major mechanical effect, elasticity, is visible in expression through the displacement of features. When a muscle causes a movement at a particular point of the face (say the corner of the lip is raised), the tissue in the surrounding area is displaced also. The amount of displacement of a particular point is determined by its distance from the point being moved, the elasticity of surrounding tissue, and the influence of boundary conditions (such as a rigid attachment to hard tissue). In general, the Poisson effect and the elasticity of the soft tissue (represented mathematically by Poisson's ratio and Young's modulus, see Appendix A) will be different depending on the material being examined. They also may depend on the orientation of motion with respect to, for example, the underlying orientation of fibre of the tissue. Therefore, these values should be considered to be multiple valued functions of spatial location.

The detailed response of the facial soft tissue to muscle action is determined by the distribution of types of material and the orientation of the fibres. In the absence of physical trauma or surgery, these conditions are determined by growth and ageing processes. Obviously, the general shape of the face and the locations of facial features are determined by the developmental process. For an individual, there will be natural areas where a crease in the skin occurs, such as at a naso-labial fold. These locations are characterised physically as areas where the fibrous structure in the tissue is preferentially aligned along the axis of the fold. Similar asymmetric alignments of fibers may arise over time due to the mechanical breakdown of the tissue: age lines and wrinkles. These features of the face occur along lines that are repeatedly exercised during facial activities. The process of wrinkle formation is similar to the fatiguing process in metals and other materials. Scars are characterised by a denser fibre structure and asymmetric fibre alignment (Pelachaud, Badler, & Viaud, 1994).

2.3.2 Primary facial features

The following features are identified as relevant in modelling the human face (Parke F. I., 1982; Faigin G., 1990). The relevance of these features comes from their role in facial conformation, movement, and communication.

1. Nose
2. Eyebrows
3. Eyes
4. Ears
5. Mouth
6. Teeth
7. Tongue and Vocal Tract
8. Cheeks
9. Chin
10. Neck
11. Hair
12. Accessories

Nose

Nose movement usually conveys an emotion of disgust. In addition, nostril movements are observed during deep respiration and inspiration. The size of the nose varies among people with different origins. Nose shape contributes significantly to identification.

Eyebrows

Eyebrow actions play a vital role both in verbal and non verbal communication. They are predominantly visible in emotions such as "surprise", "fear", and "anger'. They may also be used to accentuate a word, or to emphasise a pause or a sequence of words.

Eyes

Eyes are a crucial source of expressive information. When looking at a picture of a person, people tend to devote the greatest attention to the eyes. The eye movement may reveal "interest", or "attention" of a person. Eye blinks may occur to keep the eyes wet, or to emphasise speech, or to show an emotional state-hesitation, nervousness etc. The shape, size, and colour of the eyes provide cues in recognising individuals. The modelling of eyes should include the eyeballs and eyelids and their actions.

Ears

A face without ears looks like a mask. Ears have an intricate structure and shape. Modelling the detailed shape of ears may not be necessary, depending on the application. However, the simplification of ear shape changes the appearance of a complete face. Ear movement is extremely rare in humans.

Mouth

The mouth is a highly articulate facial zone. Lips articulate elaborately during speech. Modelling of lip motions should be able to open the mouth, stretch the lips, protrude the lips etc., to produce the phonemes and basic emotional expressions. The form and shape of lips is generally different for men and women. In addition, they provide attributes to distinguish different individuals.

Teeth

Teeth define the structure of a face as much as do the other bones; however, teeth are visible. Teeth modelling is needed for aesthetic, identification and dental surgery.

Tongue and Vocal Tract

Tongue movement is explicit, particularly in the context of verbal communication, in the formation of phonemes such as "ll", "dd", etc. The motion of the tongue often becomes obscured by the mouth motion. However, incorporation of tongue movement has immense importance for precise simulation of speech. The vocal tract is an important anatomical structure for speech production. This is of concern to clinicians.

Cheeks

Cheek movement is visible in many emotional states. Generally, cheek movements supplement other movements which may include the mouth or lower part of the eyes. The zygomatic muscles generate cheek movements while extending the corners of the lips when smiling or laughing. Actions such as the puffing and sucking of cheeks may provide emphasis for certain emotions. They reveal characteristic movements during sucking or whistling.

Chin

The movement of the chin is mainly associated with jaw motion. However, the chin is distinctively deformed to indicate "disgust" and "anger" with the lips tightened. The shape of chin also plays an important role when conforming facial models to individuals.

Neck

The neck permits the movement of the entire head, such as nodding, turning, rolling etc. As the neck moves, it can change its width or it may elongate.

Hair

To complete the modelling of a face it is essential to include hair. The colour and style of head, hair is often an indicator of gender, race, and individuality. Hair modelling and animation is an active subject of research with tremendous relevance to facial modelling. Facial hair, including eyebrows, eyelashes, moustaches, beards, and nose hairs, is also important.

Accessories

When relating to specific individuals, it is important to model accessories worn on the face and head, such as glasses, makeup, hats and hairpieces, and jewellery. People tend to see such accessories as identification marks.

2.4 Determining parameters for facial model construction

Developing a parameterised model consists of two distinct tasks:

1. Developing appropriate parameters and
2. Developing image synthesis models based on these parameters.

The first step is to determine the appropriate set of facial parameters – a nontrivial task. Ideally, one would develop a complete parameter set for creating and specifying any possible face. The possibility of developing such a set is an open question. How is a facial parameter set developed? One approach is to simply observe the surface properties of faces and develop ad hoc sets that allow these observed characteristics to be specified parametrically. A second approach involves studying the underlying structure, or facial anatomy, and developing a set of parameters based on it.

The models developed by Platt and Badler (Platt & Badler, 1981), for example, deal directly with the underlying structures that cause facial expression. Their work uses a notational system to encode the actions performed by the face. The notation drives a model of the underlying muscle structure, which in turn determines the facial expression.

There are two broad categories of parameters: those controlling the *conformation* (Parke, 1982; 1984) or structure, of an individual face, and those controlling its *expression*, or emotional content. To a certain extent, these two categories overlap, but conceptually they can be considered distinct.

2.4.1 Conformation parameters

Changes in the conformation of faces (those aspects that vary from individual to individual and make each person unique) require a different set of parameters. Again, the ideal set is unknown. The following parameters continue to be used in current models, and although this parameterisation is clearly not complete, it does allow for a wide variety of facial conformation within the implied limits.

Some conformation parameters apply globally to the face. In addition to skin colour and the aspect ratio of the face (height to width), these global parameters include a transformation, suggested by other researchers (Tod, Mark, Shaw, & Pittenger, 1980), that attempts to model facial growth. Conformation parameters control the colour (and the texture in more elaborate models) of the eyebrows, eyelashes, iris, lips, and other features.

Other conformation parameters use relative size (scale), shape, and positioning information to control

2 Neck length and shape;

3 Chin, forehead, cheek, and cheekbone shape;

4 Eyelid, eyeball, and iris size and the position and separation of the eyes (Figure 6)

5 Size and shape of ears

6 Jaws width;

7 Nose length and the width of the bridge and the end of the nose; and

8 Chin and forehead scale and the scale of the mouth-to-eyes portion of the face with the rest of the face (Figure 2.1)

9 Tongue and Vocal Tract

10 Hair

The development of a truly complete conformation parameter sets appear very difficult. Little in the way of theory exists to support their development, and the variations in facial structure from one individual to another are far less understood than the ways in which a given structure varies from one expression to another.

An important factor determining what is regarded as an acceptable facial image (synthesised by a computer system) is the *viewer's expectation.* Sensitive to subtle variations in expression and conformation, we all continually observe faces and develop very clear expectations about them. Facial expression is an important communication channel; in some contexts, it takes priority over other channels (words for example).

"An interesting and slightly frustrating phenomenon we observed while developing our models suggests the following rule: The closer the images get to reality, the more critical the viewer becomes. If the images are clearly perceived as artificial or synthetic, the viewer seems willing to be somewhat forgiving and accept them as such." (Parke, 1982)

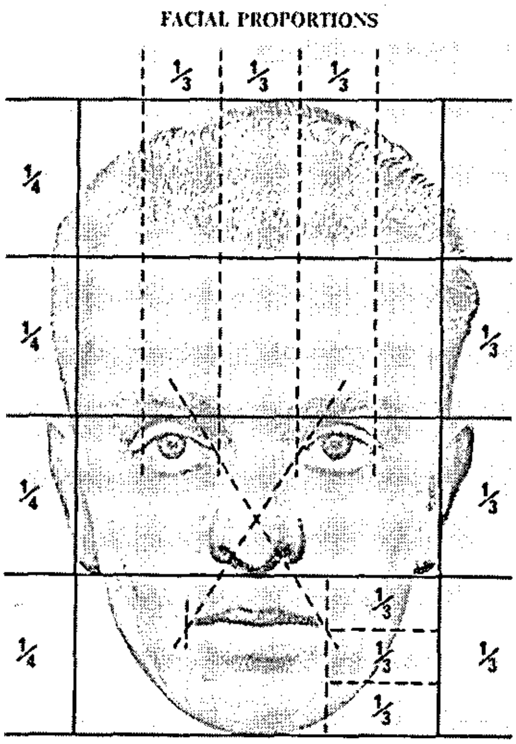


Figure 2.1 Facial Feature Proportions (from Penry J.,1971)

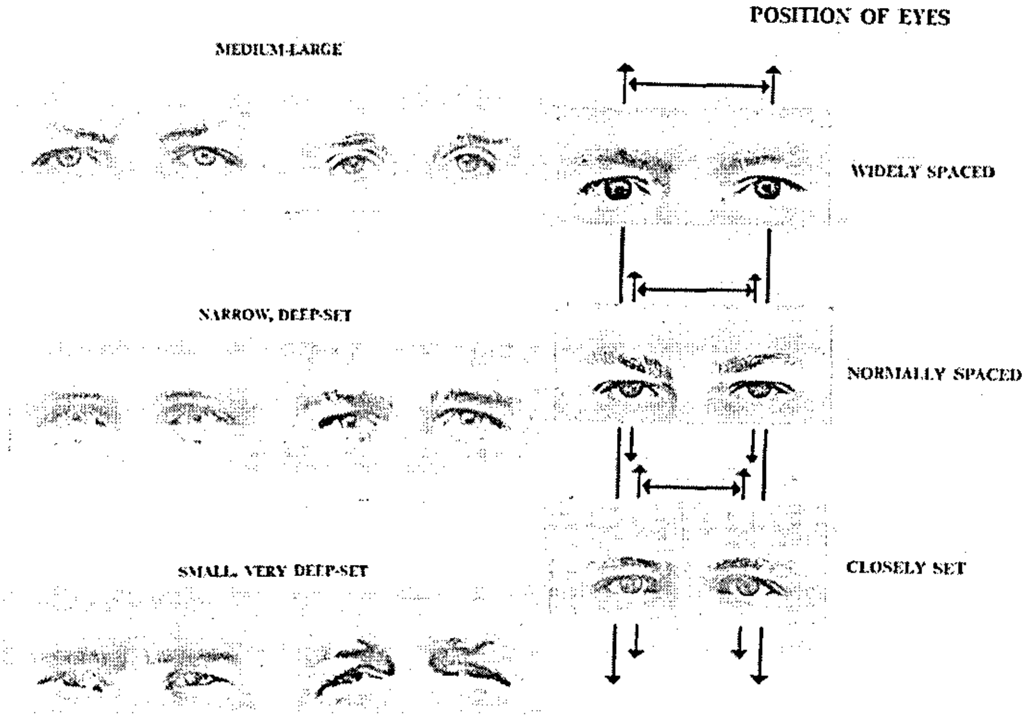


Figure 2.2 Eye Types & Eye Positioning (from Penry J.,1971)

2.5 Facial recognition and verbal description

In recognizing an object, not all aspects or attributes of the stimulus receive the same attention: certain elements appear to be more critical for identification purposes than other (Dodwell, 1971). This is almost certainly true for faces. An opinion poll carried out in the Sunday Times (Jones B, 1977) asked respondents: "What facial features draw your glance and hold you attention?" Eyes(62%) were the overwhelming choice followed by hair (22%) and mouth (8%) with the remaining 8% distributing their choices over a variety of other features.

A study performed to provide information on cue saliency in faces (Shepherd, Davies, & Ellis, 1981) looked at different experimental techniques of which verbal descriptions was one. Due to the relevance of this experiment with regards to this thesis it has been discussed. It provides information on the areas of the face people tend to focus most with greater detail when describing faces of present people (or from images) and absent people (from memory).

2.5.1 Verbal descriptions

A simple method of exploring how people attend to faces is to ask them to describe a face and see which features they mention.

This technique was adopted by Ellis *et al.* (1975) as a convenient method for examining any differences in feature extraction by Europeans and Africans looking at both white and black faces. It was also more extensively employed by Shepherd *et al.* (1977) in a series of studies of facial feature saliency.

In the first experiment 40 subjects were each asked to write descriptions from black and white prints of white male faces. There were 100 faces ranging in age between 16 and 60 years, and each subject wrote descriptions of 10 of them. The resulting 400 descriptions were then tabulated and frequency counts made of the number of times each feature was mentioned, the number of faces for which each feature was mentioned, and the number of subjects who used the particular feature description.

Thirteen facial features were identified in this way. In order of frequency they were: hair, eyes, nose, eyebrows, face shape, chin, lips, mouth, ears, face lines, complexion, forehead and cheeks. The total number of times hair occurred as a descriptor was 1135; at the other end of the continuum, cheeks were mentioned 53 times. Similarly, the category hair was divisible into 10 subsections (e.g. length, colour, texture), whereas cheeks subdivided into three description classes.

The frequency tables indicated that upper face features attracted more attention than did others. Hair, forehead, eyebrows and eyes together accounted for almost half of the total number of feature descriptions given. Not surprisingly, most subjects gave a hair- or eye-related description, and most faces attracted at least one description of their eyes and hair. Figure 2.3 illustrates the relative frequencies with which different features were mentioned.

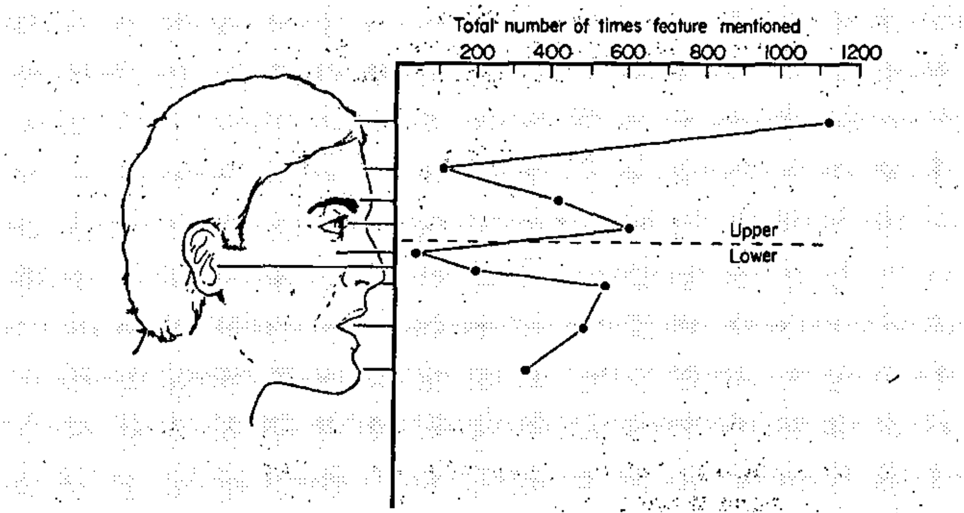


Figure 2.3 Relative frequencies with which the principal facial features were mentioned in free descriptions (from Shepherd *et al.*, 1977).

It could be argued that the distribution of descriptions across facial features depicted in Fig. 2.3 is limited in generalizability. The faces were shown as black and white prints and descriptions were made in the presence of the pictures. In the next experiment (Ellis et al., 1980) colour prints of just two male faces were employed. Subjects were required to make a description of one face immediately following a 20-second inspection period and the other face after a delay of an hour, a day or a week.

The detailed results of this experiment need not concern us here. What is striking about the data, shown in Table 2.1, is the fact that the proportions of features in the descriptions made from memory of just two faces is remarkably like those derived from descriptions made of 100 faces while each was present.

There is considerable agreement between the two sets of figures shown in Table 2.1, and it may therefore be reasonable to infer that there is a consistent pattern of attention to different facial features. Regardless of whether the face is described from a picture, or from some sort of memory image, upper face features attract more attention.

Interestingly, this pattern was not found for descriptions of faces given by black African subjects (Ellis *et al.*, 1975). Presumably then, we learn to attend to distinguishing facial features. In Caucasian faces hair and eyes vary among individuals sufficiently for reliable discriminations to be made largely on the basis of these features alone. Negroid faces, however, are less easily differentiated by hair colour and texture and eye colour and so Africans may develop a more diffuse deployment of attention across more areas of the face.

Feature	Faces absent (from Ellis *et al.*, 1980)	Faces present (from Shepherd *et al.*, 1977)
Hair	0.27	0.24
Eyes	0.14	0.13
Nose	0.14	0.12
Face structure	0.13	0.9
Eyebrows	0.8	0.9
Chin	0.7	0.7
Lips	0.6	0.6
Mouth	0.3	0.4
Complexion	0.2	0.4
Cheeks	0.1	0.1
Forehead	0.1	0.2
Others	0.4	0.9

Table 2.1
Proportion of descriptors allocated to various facial features in two different experiments

2.6 Language used by people to describe faces

The results of the experiment by Shepherd *et al.* (1981) discussed in section 2.5 offers valuable information on the approach needed to structure and conduct surveys in acquiring data on language (vocabulary, phrases) used by ordinary people to describe faces.

The survey was planned such that volunteers would be shown an image of a face and asked to describe it. Depending on the level of detail attained by the descriptions the surveyor might push the volunteer to focus on certain areas of the face and provide description in greater detail. The sample of 12 images used in the experiment is shown in Table 2.2. A web site with the survey was also developed. Since this would entail viewers having to describe faces without support and advice the survey form was divided into 7 sections, each section dealing with a specific area or feature of the face. Volunteers could easily access the web site at their convenience, select a sample from the main page, fill in the descriptions and send the data through. The web based survey forms can be found in Appendix A.

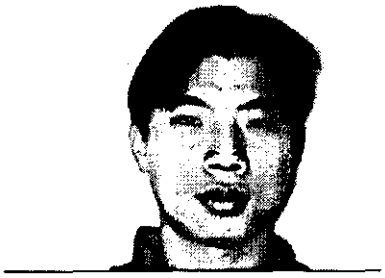
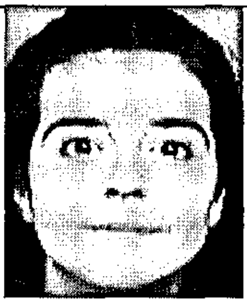
 Sample 01	 Sample 02	 Sample 03
 Sample 04	 Sample 05	 Sample 06
 Sample 07	 Sample 08	 Sample 09
 Sample 10	 Sample 11	 Sample 12

Table 2.2 Sample images, taken from the database of faces from Yale University and AT&T Labs.

2.7 Survey Results and Language Analysis

The descriptions obtained from the survey were simple but diverse. A majority of descriptions were about size and shape details closely accompanied by surface and texture details. Table 2.3 lists the descriptions given by a population of 12 volunteers, the descriptions have been divided into nine categories relating the head and features. Descriptions of the head and features collected from the survey results are listed down the table for each sample image.

Table 2.3 Descriptions obtained from survey

	Sample 01
Head	"Head broad, large skin coarse." "Large round overweight shaped face , dark smooth skinned." "Large head, giving a very round shape to the face. Dark skin, somewhat shiny, with an even texture. Prominent bone structure above the eyes and in the cheek bones."
Hair	"Woolly" "Short, dark curly hair receeding from forehead." "Short curly black hair with a hairline high on the forehead."
Eyes	"Eyes oval, wide spaced eyebrows well defined, arched." "Heavy looking eyes thickness under lower lid, bright large, dark eyes, well spaced with thick curved dark eyebrows." "Fairly small eyes, wide apart on the face. Eyebrows very faint."
Nose	"Nose broad flat large." "Large flat nose with large nostrils central to face." "Large nose, wide at the bottom with large nostrils."
Mouth	"Mouth broad, large." "Wide mouth with thick lips normal type for Africans." "Medium-sized mouth, with full and rounded lips. Mouth positioned quite close to the base of the nose."
Chin	"Chin round." "Slight stubble on chin." "Broad, rounded chin."
Cheek	"Cheek high." "Noticeable cheeks." "Full cheeks."
Jaws	"Jaw oval." "Round heavy jaw line." "Indistinct jaw line."
Ears	"Ears small." "Fairly small close to head."

	"Ears look relatively small and not very prominent, compared to the size of the head."
Sample 02	
Head	"Large oval head" "Head is average sized, face seem to be sightly elongated"
Hair	"Straight hair left partitioned" "Hair is straight"
Eyes	"Small eyes drooping down on the inside, widely spaced" "Eyes are slanted somewhat and seem to be quite highly placed on the face."
Nose	"Long nose, wide at the base nostrils showing" "Nose is centrally placed in face, seems to be of average size and shape"
Mouth	"Small mouth with thick lips" "Mouth seems a bit small, but lips are quite thick"
Chin	"Oval chin, perhaps jutting profile" "Oval chin"
Cheek	"Fairly full cheeks with high cheek bone" "Average cheeks"
Jaws	"Broad jaw line" "wide jaw"
Ears	"Small hidden behind hair" "Ears not really visible"
Sample 03	
Head	"Oblong shaped face, high forehead. Smooth, pale textured skin."
Hair	"Short, cut around ears curly light coloured."
Eyes	"Small oval light coloured eyes with heavy eyebrow and fairly light textured eyebrows."
Nose	"Medium sized with a bend to the right , small nostrils."
Mouth	"Thin upper lip with wider lower lip."
Chin	"Rounded chin."
Cheek	"Non prominent cheeks."
Jaws	"Smooth outline of jaws, slightly pointed jaw."
Ears	"Medium sized protruding ears central to head"
Sample 04	
Head	"Long oblong shape head" "Face seems quite long skin texture not clear"
Hair	"Middle partitioned slightly wavy long hair" "Hair long and wavy"
Eyes	"Squinted medium eyes, closely set" "Eyes quite narrow, eyebrows very close to eyes"
Nose	"Long thin nose, average sized bridge and nostrils flared out" "Nose quite long and pointed"
Mouth	"Small thin lipped mouth"

	“Mouth seems to be average shape and size”
Chin	“Fairly horizontal/squared chin receding chin” “Chin quite pointed”
Cheek	“Average chin with wrinkles and lines” “Cheek bones quite high”
Jaws	“Slightly wide jaw” “Broad jaw line”
Ears	“Not visible” “Not visible”
Sample 05	
Head	“Small round head shape pointy at the bottom” “Head seems quite small, somewhat rounded”
Hair	“Short wavy hair” “Hair is thick and tending to curl”
Eyes	“Small slightly closed eyes average spaced” “Eyes and eyebrows very average size and shape”
Nose	“Wide nose, pointy at the end” “Nose at centre fo face, quite small”
Mouth	“Small thin lined mouth” “Mouth average”
Chin	“Long extended chin” “Chin seems a little pointed”
Cheek	“Average cheeks, well define outline” “Cheeks quite broad”
Jaws	“Broad jaws” “Jaw quite broad”
Ears	“Small protruding ears” “Ears seem to stick out a little”
Sample 06	
Head	“Large head” “Large square-shaped head on a large neck; pale white skin with a number of blemishes.”
Hair	“Wavy” “Light coloured straight hair, quite long.”
Eyes	“Eyes small round eyebrow well marked, straight” “Quite small eyes, widely spaced; dark patches under the eyes,”
Nose	“Nose large blunt” “Large and wide nose, bulbous at the tip.”
Mouth	“Mouth quite large, well shaped” “Broad-lipped mouth, quite narrow compared to size of face.”
Chin	“Chin pointed” “Angular, protruding chin.”
Cheek	- “Cheeks full”

Jaws	- "Jaw bone very angular"
Ears	"Ears medium" -
Sample 07	
Head	"Long head"
Hair	-
Eyes	"Large open eyes oval shaped"
Nose	"Long nose"
Mouth	"Wide thin lipped mouth"
Chin	"Fairly straight/horizontal chin"
Cheek	"Slightly puffed cheeks"
Jaws	"Smooth jaw line"
Ears	"Long protruding ears"
Sample 08	
Head	"Round shaped head, pale fairly smooth skin."
Hair	"Straight dark coloured hair, quite thick, a little wispy."
Eyes	"Dark eyes, fairly round in shape and widely spaced."
Nose	"Medium sized nose with small nostrils, in the middle of the face."
Mouth	"Lipped mouth, fairly small and thin in outline."
Chin	"Rounded chin."
Cheek	"Smooth cheeks."
Jaws	"Normal jaw-line."
Ears	"Partially concealed ears, appear to be set quite low on the head."
Sample 09	
Head	"Small round head with clear skin"
Hair	"Hair wavy"
Eyes	"Eyes large oval wide spaced"
Nose	"Broad large flat"
Mouth	"Mouth wide narrow lipped"
Chin	-
Cheek	"Rounded cheeks "
Jaws	"Squarish jaw"
Ears	"Small ears"
Sample 10	
Head	"Round angular head"
Hair	"Middle parted wavy hair"
Eyes	"Large open eyes"
Nose	"Medium sized, average width nose compared to head size"
Mouth	"Fairly wide mouth thin lips"
Chin	"Squared chin"

Cheek	“Sunken cheek with “
Jaws	“Wide jaw with distinct jaw line “
Ears	“Medium sized ears protruding at the top”
Sample 11	
Head	“Large round head”
Hair	“Short”
Eyes	“Medium, average space with thin eyebrows”
Nose	“Large nose with thick bridge, nostrils showing”
Mouth	“Medium sized mouth lips not visible”
Chin	“Straight chin profile with a round chin”
Cheek	“Full and puffed cheeks”
Jaws	“Broad and wide jaw”
Ears	“Ears small and close to head”
Sample 12	
Head	“Oval head”
Hair	“Side parted, messy”
Eyes	“Small, squinted eyes”
Nose	“Small nose, wide at the base”
Mouth	“Small mouth”
Chin	“Fairly fat, round chin”
Cheek	“Cheeks full”
Jaws	“Broad jaw”
Ears	“Long ears close to head.”

Analysis of the survey results led to the compilation of distinct words or vocabulary that is commonly used to describe a face and its features. These descriptions form the basis for the lexicon used by the natural language parser to identify words and process them to generate parameters for the facial image generation module. Both the facial image generation module and the natural language interface module are described in greater detail in chapters 4 and 6 respectively. Table 2.4 lists the descriptors used in the lexical database, compiled from the survey results given in table 2.3.

Feature		Descriptor
1	Head shape	Round, oval, small, large, long
2	Forehead	Receding, vertical, bulging
3	Eyebrow	Thin, narrow, medium, thick, bushy
4	Eyes	Narrow, squinted, medium, open, large, small, round, oval
5	Eye separation	Close, medium, wide
6	Nose width	Small, medium, average, wide, large
7	Nose length	Short, small, medium, long
8	Nose tip	Bulbous, downward, hooked, pugged
9	Lips	Thin, average, medium, thick
10	Mouth width	Small, medium, average, wide, large
11	Chin	Oval, horizontal, squared
12	Ear length	Short, medium, long, large
13	Ear protrusion	Slight, medium, top, bottom
14	Cheeks	Sunken, average, full, puffed
15	Cheek bones	High, extruding, low
16	Jaw	Narrow, medium, average, wide, broad
17	Hair length	Short, average, long
18	Hair texture	Straight, wavy, curly

Table 2.3 Table of facial descriptors – compiled from survey results

2.8 Conclusion

In this chapter we have closely examined the human facial structure; this examination has, in part looked at the medical definitions of facial structure such as the bone and muscle that give faces structure and allow facial expressions. In greater detail we have identified the physical parts of a face that make faces recognisable. The work of Fredric Parke (Parke, 1982) has been acknowledged for his pioneering work on facial animation and defining techniques to parameterise faces for artificial composition and animation. Recent work by artists like Faigin (1990) has also been acknowledged.

We have also looked at the work of Ellis (Ellis *et al.*, 1975) and Shepherd (Shepherd *et al.*, 1977) on facial recognition and verbal descriptions. This has provided beneficial insight into what areas of a face people usually remember and recall most frequently. This information helped in planning and executing surveys necessary to acquire important data on the language ordinary people use to describe faces.

Finally we examined the survey results and compiled a list of most commonly used descriptors for the lexical database in the natural language processing engine.

Chapter 3

Tools, Techniques and Technology for 3D Facial Modelling

Abstract

This chapter provides an exposition of the tools and techniques used to develop 3D geometry specifically to construct 3D head models and the facial image generation system. The human head models developed using the tools and techniques defined in this chapter are the backbone for the automated facial image generation system. The chapter is in three parts; the first dealing with facial modelling, existing research and applications, the second with representation and acquisition techniques available for 3D facial image composition and the third discusses the tools and technology used for developing the human head models in 3D.

Keywords: 3D, Facial Models, Tools, NURBS, Beziers, Polygons, Spline, Surface, Geometry, FFD.

3.1 Introduction

The complexity of the human face makes it a challenging subject for modellers. Facial modelling has been an active area of research in the computer graphics field for more than two decades. It benefits from and can contribute to the larger field of human body modelling. Facial modelling is also relevant in other fields, such as medicine and engineering. It is, in fact, a multidisciplinary effort.

A facial model is a mathematical abstraction that captures to some degree of accuracy the form and function of a face, whether human or otherwise, in a way that makes the model useful for specific applications. State-of-the-art facial models for computer animation attempt to represent the geometry, photometry, deformation, motion, etc., of the various organs and features associated with the face, as well as with the rest of the head and neck. These models rely on data from various sources (shape, colour, elasticity, control, etc.). Typically, the models are designed to produce meaningful facial images.

3.2 Existing Research

The human face is an important and complex communication channel. It is a very familiar and sensitive object of human perception. The facial animation field has increased greatly in the past few years as fast computer graphics workstations have made the modelling and real-time animation of hundreds or thousands of polygons affordable and almost commonplace. Many applications have been developed such as teleconferencing, surgery, information assistance systems, games, and entertainment (Facial Animation, 1997). To solve these different problems, different approaches for both animation control and modelling have been developed.

Substantive research in the real-time animation of faces for telecommunication and for the synthesis of computer interface "agents" is being conducted at Apple Computer, Inc. (Advanced Technology Group, Cupertino), Hitachi, Ltd. (Hitachi Central Research Laboratory, Japan), NTT (Human and Multimedia Laboratory, NTT Human Interface Laboratories, Japan), and Sony (Information Systems Research Center, Sony Corporation, Japan).

A number of companies are in the business of vending computer systems and services for making facial image composites ("identikit" police identification tools, point-of-purchase video preview for cosmetic make over or cosmetic surgery, and one class of systems for estimating the aged appearances of missing children), 3D digitization of faces (Cyberware, 1990), 3D reconstructive surgery preview (Delinguette *et al.* 1994) and manufacture of facial prosthetics, 3D digitization of teeth for the manufacture of dental appliances, and 2D and 3D facial animation.

Another important current interest is in the entertainment industry; the use of graphical face models in advertising, for movie special effects, etc.

3.3 Applications

Typical models of the human face are relevant to a variety of applications, such as education, entertainment, medicine, telecommunications, etc. The amount of detail that the model captures is likely to vary from application to application.

3.3.1 Education

In an educational environment, a major use of the face is in communicating ideas. For example, a model that captures the physics and anatomy of the human face may be used in teaching medical students about faces (Thalmann & Thalmann, 1994). Another important application is artificial agents or avatars that take students on tours of historical sites or museums. There is some work done on using avatars in networked environments such as CSWG these can be both in field of education and industry (Capin, 1998)

3.3.2 Entertainment

The use of faces for entertainment often requires the elicitation of empathy and human emotion towards computer generated characters. The synthesis of facial expressions is important in this context (Thalmann & Thalmann, 1995).

3.3.3 Medicine

Preoperative simulation of corrective plastic surgery and dental treatment are of great interest to both practitioners and patients alike. Such applications demand precise models of particular individuals based on the bone and soft tissue of the head. A computerised system, which incorporates an anatomically complete model of the head and face, would provide surgeons with the capability to plan, and even rehearse, complex operations without undertaking costly and potentially dangerous exploratory surgery (Delinguette *et al.* 1994).

3.3.4 Narration

Speech is an integral component of human communication. A face model which incorporates speech synthesis capabilities could prove to be useful for the deaf and hard-of-hearing. (Roberts and Storey, 1986; Marigny, Adjoudani, and Benoit, 1994).

3.3.5 Telecommunication

Researchers are developing facial models for use in videophones (such as portable videophones) that must transmit facial images over low-bandwidth channels. A photorealistic model of the speaker is captured and transmitted to the receiving station where it is reconstructed at low bit-rates to produce a realistic animated image of the speaker's face (Ohya, 1995).

3.3.6 Criminology

Recognition and identification of faces is an important aspect in criminal investigations. Here, representing the appearance of a wide variety of faces is particularly important (Carey & Diamond, 1977; Turk & Pentland, 1991).

3.3.7 Forensic Medicine and Anthropology

Reconstruction of realistic faces from skeletal remains is of immense interest in forensic medicine and archaeology (Vanezis, 1999). Facial reconstruction can be employed to assist in identifying a victim from only a few clues. A computer-based system would require a complete model of the face in order to mimic the manual process.

3.3.8 Advertising

A major objective of the use of the face in advertising is to give the audience an unambiguous message. This requires accurate modelling of facial behaviours.

3.4 3D Facial Modeling

3.4.1 Brief history

The first work in developing facial models was done in the early 70's by Parke at the University of Utah (Parke, 1972a, 1972b, 1974, 1975) and Gillenson at Ohio State (Gillenson, 1974). Parke developed the first interpolated and the first parametric three dimensional face models while Gillenson developed the first interactive two dimensional face models. In 1971, Chernoff (1971, 1973) proposed the use of simple 2D computer generated facial images to present n-dimensional data. In the early 80's, Platt and Badler at the University of Pennsylvania developed the first muscle action based facial models (Platt, 1980, 1985; Platt & Badler, 1981). These models were the first to make use of the Facial Action Coding System (Ekman & Friesen, 1978; Ekman & Oster, 1979) as the basis for facial expression control.

Between mid 80’s and early 90’s there was considerable activity in the development of facial models and related techniques. Waters and Terzopoulos developed a series of physically based pseudo-muscle driven facial models (Waters, 1986, 1987, 1988; Waters & Terzopoulos, 1990, 1992; Terzopoulos & Waters, 1990b). Magnenat-Thalmann, Primeau, and Thalmann (1988) presented their work on Abstract Muscle Action models in the same year as Nahas, Huitric and Sanintourens (1988) developed a face model using B-spline surfaces rather than the more common polygonal surfaces. Waite (1989) and Patel and Willis (1991) have also reported facial model work. Techniques for modeling and rendering hair have been the focus of much recent work (Yamana & Suenaga, 1987; Watanabe & Suenaga, 1992). Also, surface texture mapping techniques to achieve more realistic images have been incorporated in facial models (Oka *et al.*, 1987; Williams, 1990; Waters & Terzopoulos, 1991; Anjyo, 1992; Yacoob, 1994).

In early models, modelling was done by digitising sculptures of the face with various expressions (different lip shapes and expressions) and storing them in a library (Walczak, 1988). Animation was performed by linear interpolation between given stored expressions. Such a method is really tedious and time consuming since it is not automatically adaptable to any other new model.

The ability to synchronize facial actions with speech was first demonstrated by Parke in 1974 (Parke, 1974, 1975). Several other researchers have reported work in speech animation (Pearce *et al.*, 1986; Lewis & Parke, 1987; Hill *et al.*, 1988; Wyvill, 1989). Pelachaud has reported on work incorporating co-articulation into facial animation (Pelachaud, 1991). Work modeling the physical properties of human skin have been reported by Komatsu (1988), Larrabee (1986), and Pieper (1989, 1991).

3.4.2 Current models

Essentially all of the current face models produce rendered images based on polygonal surfaces. Some of the models make use of surface texture mapping to increase realism. The facial surfaces are controlled and manipulated using one of three basic techniques: 3D surface interpolation, *ad hoc* surface shape parameterization, and physically based with pseudo-muscles.

By far the most common technique is to control facial expression using simple 3D shape interpolation. This is done by measuring (Cyberware Laboratory Inc., 1990; Vannier *et al.*, 1991) the desired face in several different expressions and interpolating the surface vertex values to go from one expression to the next. One extension on this approach is to divide the face into regions and interpolate each region independently (Kleiser, 1989).

Ad hoc parameterized facial models have been developed primarily by Parke (1982). These models are based on a set of parameters, which affect not only facial expressions (opening of the mouth, raising eyebrows, etc.) but also facial conformation (long nose, short forehead, etc.). These parameters are only loosely physically based. These parametric models (Pearce *et al.*, 1986; Ohmura, 1988; Patel, 1991) are the only ones to date that allow facial conformation control, i.e., changes from one individual face to another. The separation between conformation parameters (Parke, 1982; Platt & Badler, 1981; Faigin, 1990) and expression parameters forces the independence between facial features and the production of an expression.

Physically based models attempt to model the shape changes of the face by modeling the properties of facial tissue and muscle actions. Most of these models are based on spring meshes or spring lattices with muscle actions approximated by various force functions. These systems describe the skin as an elastic spring mesh where unit actions are simulated by forces. The deformations are then performed by solving the dynamic equations (Waters, 1988). Muscle movement propagation is intrinsic to the model. Various layers of facial tissue are integrated (Turk & Pentland, 1991). It succeeds in producing subtle facial actions with realism

3.4.3 Facial Recognition – Complementing Facial Generation

There is a long history of research into face recognition and interpretation. Much of the work in computer recognition of faces has focused on detecting individual features such as the eyes, nose, mouth, and head outline, and defining a face model by the position, size, and relationships among these features. Beginning with Bledsoe's (1966) and Kanade's (1973, 1977) early systems, a number of automated or semi-automated face recognition strategies have modeled and classified faces based on normalized distances and ratios among feature points such as eye corners, mouth corners, nose tip, and chin point (e.g. Goldstein *et al.*, 1971; Kaya & Kobayashi, 1972; Cannon *et al.*, 1986; Craw *et al.*, 1987). Lately this general approach has been continued and improved by the work of Yuille and his colleagues (Yuille, 1991). Their strategy is based on "deformable templates", which are parameterized models of the face and its features in which the parameter values are determined by interactions with the image.

Such approaches have proven difficult to extend to multiple views, and have often been quite fragile, requiring a good initial guess to guide them. In contrast, humans have remarkable abilities to recognize familiar faces under a wide range of conditions, including the ravages of aging. Research in human strategies of face recognition, moreover, has shown that individual features and their immediate relationships comprise an insufficient representation to account for the performance of adult human face identification (Carey & Diamond, 1977). Nonetheless, this approach to face recognition remains the most popular one in the computer vision literature.

In contrast, latest approaches to face identification seek to capture the configurational, or gestalt-like nature of the task. These more global methods, including many neural network systems, have proven much more successful and robust. For instance, the eigenface (Turk & Pentland, 1991) technique has been successfully applied to "mugshot" databases as large as 8,000 face images (3,000 people), with recognition rates that are well in excess of 90% (Pentland, 1992), and neural networks have performed as well as humans on the problem of identifying sex from faces (Golomb et al., 1991).

3.5 Representation Techniques

Input for shape reconstruction may be drawn from photographs and/or scanned data. Among the variety of ways of representing a face geometrically, the choice should be one that allows for precise shape, effective animation and efficient rendering.

Two broad categories may be distinguished: volume representation and surface representation. Volume representation may be based on constructive solid geometry (CSG) primitives or volume elements (voxels) from medical images. However, volume representation has not been widely adopted for facial animation because CSG primitives are too simple to produce reasonable face shapes. Voxels are high resolution data, need to be segmented from a huge voxel map and require data thinning. Largely for these reasons, the animation using volumic data is computationally intensive.

Surface primitives and structures are currently the preferred geometrical representations for faces. Among surface description techniques are polygonal surfaces, parametric surfaces, and implicit surfaces. In a polygonal surface representation, a face is a collection of polygons, regularly or irregularly shaped. The majority of existing models use polygonal surfaces, primarily because of their simplicity and the hardware display facilities available for polygons on most platforms. The parametric surfaces use bivariate parametric functions to define surfaces in three dimensions, e.g. bicubic B-spline surfaces. The advantage of these models is that they have smooth surfaces and are determined using only a few control points. However, local high-density details for eyes and mouth are difficult to add. Hierarchical B-splines developed by Forsey and Bartels (1990) enable more local detail without the need to add complete rows or columns of control points. Wang (1991) has used the hierarchical B-splines for modeling and animating faces. An implicit surface is an analytic surface defined by a scalar field function. Interaction with implicit surfaces is difficult with currently available techniques, and these have not yet been used for facial modeling.

We will now discuss the five main representation techniques available for modeling human heads. Techniques described here have been used to model the baseline head for the facial image generation system

- Polygonal or Mesh based modelling
- Spline based modelling
- Bezier patches
- NURBS (Non Uniform Rational B-Splines)
- Implicit surfaces

3.5.1 Polygonal/Mesh Modelling

Polygonal modelling consists of a topological and geometric structure of interconnecting triangles, called facets, of various sizes and orientations. By arranging the facets a very simple 3D model can be built up to a very complex model. Polygonal models are also easily animated. Further, by altering the size and orientation of the facets, simple animations can be produced, such as bends or twists, or more complex animations, such as morphing.

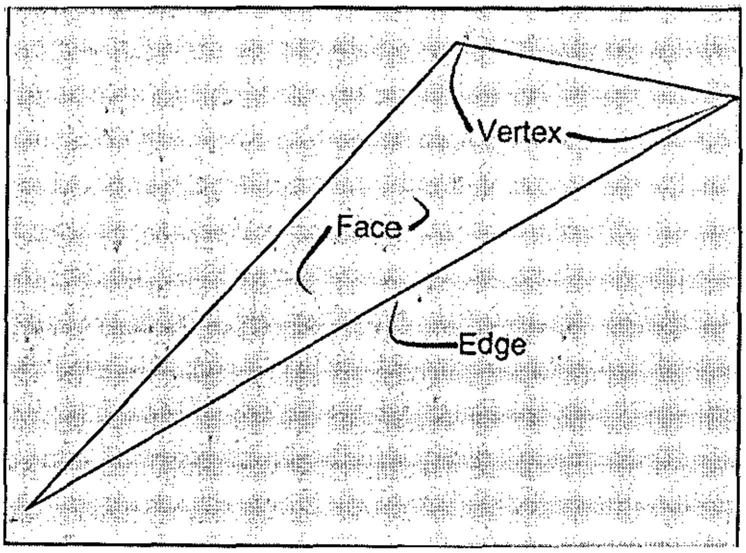


Figure 3.1 The basic building blocks of polygonal modelling (scanned image: Bell, 1998).

The principle of detail is straightforward: the more facets or polygons in a given location, the more detailed it will be. Polygons can be used to model just about anything, furthermore they are used at the lowest level by many commercial rendering systems. With enough detail any surface can be created. There is one major drawback though and that is, detail on model objects requires more polygons. As facet count increases, performance begins to degrade. The increasing number of facets also reduces the ability to edit detailed models. Due to the large number of facets in detailed areas of a polygonal model, making small changes can often be a significant challenge. On the other hand the presence of a large number of facets does allow small detailed changes to be made easily.

Polygons are usually used for creating objects that are planar in nature and not particularly organic. Example buildings, road intersection etc. For rendering reasons, polygons are often used. Polygonal modelling works best for low detail or more rigid looking models. Patches and NURBS work well for more complex and organic models. However there are certain texturing and shading techniques that can reduce the faceted look of polygonal models. (Foley, 1996; Bell, 1998; Peterson, 1997; Boardmann and Hubbell, 1998).

Texturing/shading models (multi-spectral)

Mesh facial models (either polygonal or parametrically based) may be given realism or texture by means of surface mapping and shading. Shading can smooth a polygonal model. Various methods are available and they may be applied alone or in combination depending on the desired appearance of the model:

- Flat shading: the pixels in a polygon are all the same colour with no variation. If the model is faceted, each facet will be distinguishable. Flat shading is useful only as a low-cost rendering method (Foley, 1996; Bell, 1998).

- Gouraud (smooth): This is a shade-interpolating method of shading that will make the object appear smooth, instead of faceted. This method doesn't work well with highlights or local light sources and one can often still see polygonal edges on the object (Foley, 1996; Bell, 1998).

- Phong: This is a normal-interpolating method. The object appears very smooth. This method goes a step further than Gouraud. A new shade is computed for each point, point by point before it shades (Foley, 1996; Peterson, 1997, Boardmann and Hubbell, 1998)

- Bump mapping: is another method for producing maps of rough or textured surfaces, but it does not have the edge or shadow accuracy of displacement mapping (Foley, 1996; Peterson, 1997; Bell, 1998; Boardmann and Hubbell, 1998).

- Displacement mapping: is a method for distorting a surface to produce an embossed or debossed surface that produces geometry with accurate edges and shadows. The displacement map specifies how the surface is to be moved before being mapped (Bell, 1998).

- Reflection mapping: gives the illusion of reflection or a mirrored effect (Boardmann and Hubbell, 1998)..

- Environment mapping: is a method by which the model's surface reflects the environment on its surface (Boardmann and Hubbell, 1998).

- Opacity mapping: involves using the grayscale of a 2D object to define an object's transparency or opacity (Bell, 1998; Boardmann and Hubbell, 1998).

- Transparency mapping: gives the illusion of transparency, like looking through glass (Foley, 1996; Bell, 1998). This is particularly useful for skin pallor, as it is semi-transparent. For example, (Kalra *et al.* 1993) includes an emotion model which expresses emotion through the vascular system, such as paleness due to fear or blushing due to embarrassment.

- Texture mapping: is the process by which the bitmap is applied, on to the geometric model. Textures may be applied as either 2D bitmaps or scans (Lewis & Parke, 1987; Waters & Terzopoulos, 1992). Photographs may be applied to mesh models as maps (Bell, 1998; and Hubbell, 1998). The mapped textures may also be shaded in accordance with the lighting and surface geometry.

3.5.2 Splines

For very smooth surface, a variety of spline-based surface patch methods can be used. They are great for creating any type of object that has a profile or shape that can be lofted or extruded. Example bananas, phone handles bottles, etc.

- *B-splines*: (Ohmura, 1988) The face is modelled using B-splines. Deformations are performed by moving groups of controls points.

- *Cardinal splines & springs*: (Waters, 1987) A cardinal spline representation is coupled with a spring network. Muscle deformations are generated by applying forces to the spring network. For each rest state of the spring network, the spline surface is recalculated to create discontinuities and bulges: tangencies are computed to keep the arclength of the spline segment identical at rest and under compression.

- *Hierarchical splines & springs*: The face is modelled using hierarchical splines (Forsey & Bartels, 1990). Muscles are defined by forces, the definition points of which belong to the surface. Muscle definition follows any face transformation. Additional effects such as wrinkles are provided by behaviour maps.

3.5.3 Bezier Patches

Patch surfaces consist of a series of control points connecting each other, the surface is controlled and deformed through a deformation lattice and smoothed using parametric polynomials or Bezier Tangent Handles.

Patches rely upon the principles of Bezier splines to deform the surface and although it creates a smooth surface it is still an approximation. A patch is moved and distorted by changing the shape of the lattice, either directly or by means of a Bezier vertex at each corner.

A patch surface is made up of two parts: the surface and the deformation lattice (see Figure. 9). The deformation lattice is a series of connected points along the surface of the patch. Each point of the lattice is a control point that has control over the associated area of the patch. Adjusting a lattice point adjusts an area of the patch surface, not just a single point as in vertex editing in a mesh. The lattice acts as the vertices in a Bezier spline and deforms the surface along a Bezier curve, instead of creating a liner curve. (Peterson, 1997; Boardmann and Hubbell, 1998; Foley, 1996).

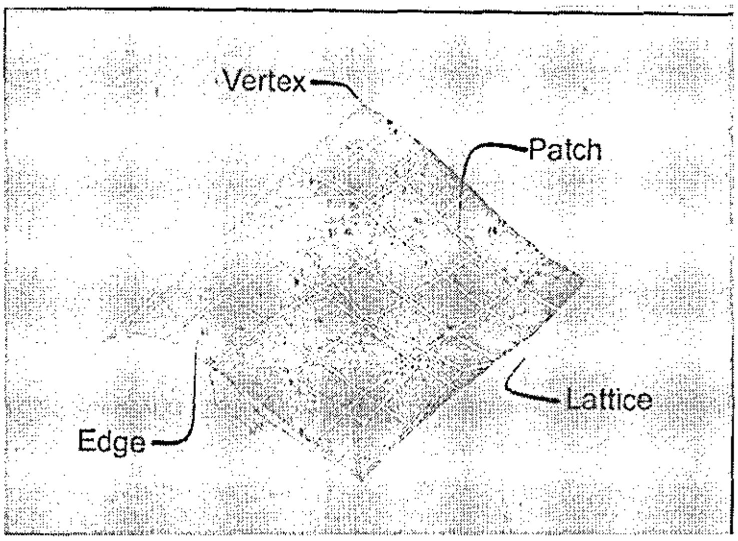


Figure 3.2 Illustrating a patch surface (scanned image: Bell, 1998).

Patches can be joined together by the usual vertex welding method, the drawback is that it can be difficult to get the edges of patches to line up correctly to form large patches. Alternatively a better way is to grow a patch off an existing one. This is done in 3D Studio Max by picking an edge of a patch and then adding a quad or tri patch, this adds the new patch to the existing patch and blends them together.

Patches are used to create somewhat organic surfaces that require fairly precise control over the curvature of the surface e.g. face, animal's etc.

Patches rely upon the principles of bezier splines to deform the surface although it creates a smooth surface it is still an approximation. In Patch modelling it can be difficult to get the edges of patches to line up correctly to form large patches.

3.5.4 NURBS

NURBS (Non-Uniform Rational B-Spline) modelling is probably the most powerful modelling method for creating complex surfaces available today. With NURBS, there are two basic approaches to modelling. One is to create NURBS splines and create surfaces between splines. The other is to create NURBS surfaces and adjust the surfaces or create blends between surfaces (Bell, 1998; Peterson, 1997).

NURBS curves are created out of either points or control vertices (see Figure 3.3). The difference between the two is how the curve is interpreted around the vertices. When using points, the curve passes directly through the control points. When using control vertices (CV), the points act more as a deformation lattice.

NURBS give both smooth, contoured surfaces and keep mesh detail relatively low. Characters or human faces tend to be very complex so using NURBS can significantly increase performance versus the same model in polygonal forms.

A NURBS object is one or more curved lines in three dimensional space with varying properties (weights) that can be rationally defined mathematically (Foley, 1996).

- Non Uniform means that different areas along NURBS objects (curves or surfaces) can have different properties (weights) and are not completely uniform, i.e. the blending functions are no longer the same for each interval, but rather vary from curve segment to curve segment.
- Rational means of the form $A/_B$ where A,B are polynomials (locally). Rational curve segments are ratios of polynomials:

$$x(t)=\frac{X(t)}{W(t)},y(t)=\frac{Y(t)}{W(t)},z(t)=\frac{Z(t)}{W(t)} \tag{3.1}$$

where *X(t), Y(t), Z(t),* and *W(t)* are all cubic polynomial curves whose control points are defined in homogenous co-ordinates. Rational curves are useful because they are invariant under rotation, scaling, translation and perspective transformation of the control points (non-rational curves are invariant under only rotation, scaling, and translation). This means that the perspective transformation needs to be applied to only the control points, which can then be used to generate the perspective transformation of the original curve.

- B-splines consist of curve segments whose polynomial coefficients depend on just a few control points. This is called *local control.* Thus, moving a control point affects only a small part of the curve. In addition, the time needed to compute the coefficients is greatly reduced. B-splines have the same continuity as natural splines, but do not interpolate their control points.

A rational B-spline curve is defined by a set of four-dimensional control points:

$$P_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i) \tag{3.2}$$

The perspective map of such a curve in three-dimensional space is called a rational B-spline curve:

$$P(u) = \sum_{i=o}^{n} P_1^w B_{i,k}(u) \tag{3.3}$$

$$= \frac{\sum_{i=0}^{n} P_i w_i B_{i,k}(u)}{\sum_{i=0}^{n} w_i B_{i,k}(u)}$$

$$= \sum_{i=0}^{n} P_i R_{i,k}(u) \tag{3.4}$$

where:

$$R_{i,k}(u) = \frac{B_{i,k}(u) w_i}{\sum_{j=0}^{n} B_{j,k}(u) w_j} \tag{3.5}$$

and if:

$w_i = 1$ for all i

then:

$$R_{i,k}(u) = B_{i,k}(u) \tag{3.6}$$

The w_i associated with each control point are called **weights** and can be viewed as extra shape parameters. The curve is pulled towards a control point P_i if w_i increases. If w_i is decreased the curve moves away from the control point.

The greatest advantage NURBS have over other modelling techniques is the attribute of NURBS surfaces to adjust themselves in order to maintain their defining curves. NURBS therefore produce extremely smooth and organic models closely resembling human skin thus ideal for facial modelling.

In contrast to polygonal modelling NURBS can give both smooth contoured surfaces and keep mesh detail relatively low. Character modelling can be very complex, so using NURBS can significantly increase performance versus the same model in polygonal form.

All NURBS surfaces consist of three sub-objects: points or control vertices (CV), curves (which are determined by their control vertices) and surfaces (which are controlled by either curves or their own control vertices). Points lie precisely on the surface or on the curve they affect, almost exactly like a standard vertex. However, unlike a standard spline vertex they cannot use Bezier, corner or Bezier corner manipulation (Foley et al. 1996). They behave very much like a smooth spline vertex. CVs are points that control the amount and placement of curvature in a surface or curve but do not lie on the surface or curve they control. CVs have a weight parameter, which influences the curve or surface.

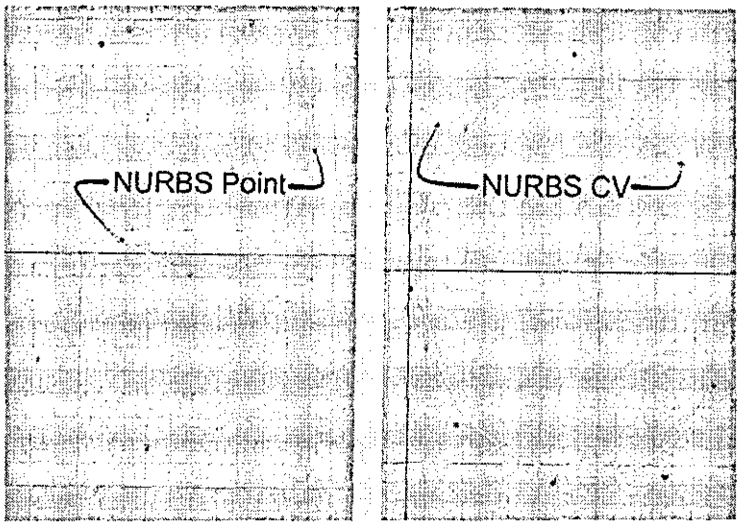


Figure 3.3 The figure on the left shows a NURBS curve constrained to pass through a set of three points, whereas the figure on the right shows a NURBS curve controlled by a set of points which do not pass through the curve but "attract" it according to their associated weight (scanned image: Bell, 1998).

The higher the weight, the more a surface curve is drawn toward a CV's position; the lower the weight, the smaller the influence a CV has over the curve or surface. Weights however are relative, this means that if all of the CVs of a particular surface were set to a high value, there would be no change because the influence over the surface is equal. If a CV has a higher or lower relative weight than its neighbouring CV then a difference can be seen.

Control vertices and points are the basis for everything in NURBS. They are, however mutually exclusive. A surface or curve is made up of one or the other, never both.

In addition to the differences between point and CV objects, surfaces and curves may also be either dependent or independent. A dependent curve or surface has no active CVs or points of its own. Instead it is controlled by a combination of curves and/or surfaces. The benefit of a dependent object is that it will always attempt to maintain a smooth curve or surface across the independent objects that determine how it's formed, allowing for extremely organic animation. The disadvantage of a dependent object is that it cannot be manipulated or sculpted by itself.

3.5.5 Implicit surfaces

Implicit surfaces usually create very expressive models. This technique has also been used to generate symbolic descriptions of an object by fitting simple primitives to range data of the object. First a primitive is given, then an energy function which measures the difference between the range data and the model is minimised each time a new primitive is added (Muzekari, 1986).

3.5.6 Rendering

Rendering is the process of taking a geometric model, applying lighting, selecting a point of view (camera position) to the scene, and then creating a 2D image (bitmap or rasterization), or "snapshot" of that model. Basic rendering algorithms include wireframe, polygon shading, ray tracing, and radiosity.

3.6 Facial data acquisition

Face models rely on data from various sources for shapes, color, texture, etc. In constructing geometrical descriptions, two types of input should be distinguished: three-dimensional and two-dimensional.

3.6.1 Three-dimensional input

Use of a 3D digitizer/scanner (Cyberware Laboratory Inc, 1990) is the most direct method for acquiring the geometry of faces. A 3D digitizer involves moving a sensor or locating device to each surface point to be measured. With this method, 128,000 range and reflectance samples may be obtained in a few seconds. Cylindrical projection is used for the measurement of faces. Yacoob (1994) created facial models from measured data, and animated it. 120,000 samples are typically too much for rendering and animation use, so they should be represented by a simpler model. Fitting the obtained samples to a generic facial model is efficient for the facial animation. Waters and Terzopoulos (Waters & Terzopoulos, 1992) proposed a physics-based technique to reduce these samples to coarser, non-uniform meshes (see also Lewis & Parke, 1987). There are several types of 3D digitizers employing different measurement techniques (mechanical, acoustic, electromagnetic). Polhemus, an electromagnetic digitizer, has been used by many researchers for modeling faces . In other cases a plaster model has been used for marking the points and connectivities. This procedure is not automatic and is very time consuming.

Laser based scanners, such as Cyberware, can provide both the range and reflectance map of the 3D data in a few seconds. The range data produce a large regular mesh of points in a cylindrical coordinate system. The reflectance map gives color and texture information. One of the problems with this method is the high density data provided. Another is that the surface data from laser scanners tend to be noisy, and have missing points. Some post processing operations are required before the data can be used. These may include relaxation membrane interpolation for filling in the missing data, filter methods, e.g. hysteresis blur filters, for smoothing data, and adaptive polygon meshes to reduce the size of the data set for the final face model. One disadvantage of the laser scanner is that the equipment is relatively expensive. Another is that no human subject can be used for scanning due to inherent danger to eyes from laser depending on the class and strength of laser utilized by the scanning device. The process usually involves scanning a plaster cast of the face (Lee, 1995).

Another 3D digitising method uses 3D trackers. With this method, meshes are drawn on a face and the 3D co-ordinates of vertices are digitised using an electro-magnetic 3D digitizer. This procedure is not automatic and therefore is time consuming. The advantage of the method is that the polygonal mesh is designed according to the topology of the face, and then optimised (few polygons for a good definition of the shape). "Tony de Peltrie" from the University of Montreal, Marilyn Monroe and Humphrey Bogart, from Daniel Thalmann and Nadia Magnenat-Thalmann (Magnenat-Thalmann, 1987) were created with this method.

CT (Computer Tomography) and MRI (Magnetic Resonance Imaging) are usually used in the field of medicine. These methods can capture not only the facial surface, but also inner structure such as bones or muscles. These additional structures will be useful for more accurate facial modelling and animation, as well as medical applications such as a medical operation simulation.

As an alternative to measuring facial surfaces, models may be created using interactive methods like sculpturing. With this the face is designed and modeled by direct and interactive manipulation of vertex positions or surface control points. This, however, presupposes design skills and sufficient time to build the model. When constructing a clone, relying on subjective visual impressions may not be accurate or rapid enough. Arbitrary facial models (such as imaginary faces or faces of historical person) can be designed. However, it requires time and design skill because faces have very complex structures. Commercial geometric modellers have been used for the face and body design of the figures in "Little Death" (Elson, 1996; Parke, 1975) has used interactive deformation techniques such as the "ball and mouse" metaphor (Lee, 1993) for face and body design.

3.6.2 Two dimensional input

There are a number of methods for inferring 3D shape from 2D images. Photogrammetry of a set of images (generally two) can be used for estimating 3D shape information (Parke, 1974) . Typically, the same set of surface points are located and measured in at least two different photographs. This set of points may even be marked on the face before the pair of photograph is taken. The measurement can be done manually or using a 2D digitizer. A better method takes account of perspective distortion by using a projection transformation matrix determined by six reference points with known 3D coordinates.

Another approach is to modify a canonical or generic face model to fit the specific facial model using information from photographs of the specific face (Williams, 1990). This relies on the fact that humans share common structures and are similar in shape. The advantages here are that no specialized hardware is needed and that the modified heads all share the same topology and structure and hence can be easily animated.

Parametric animation models make use of local region interpolation, geometric transformations, and mapping techniques to manipulate the features of the face. These transformations are grouped together to create a set of parameters. Sets of parameters can apply to both the conformation and the animation of the face.

In pseudo-muscle based models, muscle actions are simulated by abstract notions of muscles, where deformation operators define muscle activities. The dynamics of different facial tissues is not considered. The idea here is not to simulate detailed facial anatomy exactly but to design a model with a few parameters that emulate muscle actions (Waters, 1987).

There are no facial animation models yet, based on complete and detailed anatomy. Models have, however, been proposed and developed using simplified structures for bone, muscle, fatty tissue and skin. These models enable facial animation through particular characteristics of the facial muscles. Platt and Badler (1981) used a mass-spring model to simulate muscles. Waters (1987) developed operators to simulate linear and sphincter muscles having directional properties. A physically-based model has been developed where muscle actions are modeled by simulating the tri-layer structure of muscle, fatty tissues and the skin. Most of these methods do not have real-time performance

The interaction between the various layers of the face generates the complexity of facial deformations. Therefore, it is difficult to isolate representation techniques from deformation techniques: each model is an association between a geometric representation and some deformation tools. In some cases, as with the finite element method, the geometry of the model is strongly linked to the deformation method. On one hand, techniques depend on the desired application. For example, the requirements for medical applications may be drastically different from the requirements for animation. On the other hand, it is often desirable to get as complete a simulation as possible of the entire structure (bones, muscles, skin, and internal actions leading to deformation are important). In certain cases, the visual effect (deformation of the external layer) is all that matters and the issues may be computation time and manipulation tools.

3.7 3D Modelling Tools and Applications

A prerequisite of the image generation system was existence of 3D face models. This criterion required either the acquisition of 3D face geometry, as discussed in section 3.6, or development of facial geometry using commercially available geometric modellers. Difficulties in acquiring 3D face data via digitisers or scanners led to the decision to construct a 3D head model using existing geometric modelling tools. Research on available modelling tools narrowed down the list to a choice of four professional surface and solid modellers existing at Loughborough University. These packages were as follows:

1. Duct
2. SoftImage
3. Unigraphics
4. 3D Studio Max

Each of the four packages were tested with respect to performance, usability, scripting facilities and modelling technologies. Duct and Unigraphics had support for Beziers but did not support Nurbs or B-Splines. SoftImage and 3D Studio Max both had an intuitive interface and provided a range of surface modelling techniques including Nurbs and spline curves. The choice came down to using the software that provided B-spline or Nurb curves technology and also included a scripting facility that would allow the entire modelling process to be automated.The software chosen for constructing the human head models was 3D studio Max by Discreet.

3.8 Examining 3D Studio Max

3D Studio Max is a powerful modelling and animation software tool developed by Discreet, a division of Autodesk (http://www.discreet.com). 3D Studio Max incorporates three different modelling technologies in the basic package, namely: Polygonal, Patch, and NURBS. The software can be further extended with plug-ins. The software provides a graphical user interface, which is easy to use. It is possible to import curves from Autocad to convert them in NURBS. The rendering is efficient, almost as fast as polygons.

The 3D Studio Max renderer includes features such as selective ray tracing, analytical antialiasing, motion blur, volumetric lighting, and environmental effects. Lights can be created with various properties to illuminate the scene. Lights can cast shadows, project images, and create volumetric effects for atmospheric lighting. Cameras in a scene have real-world controls for lens length, field of view, and motion control such as truck, dolly, and pan.

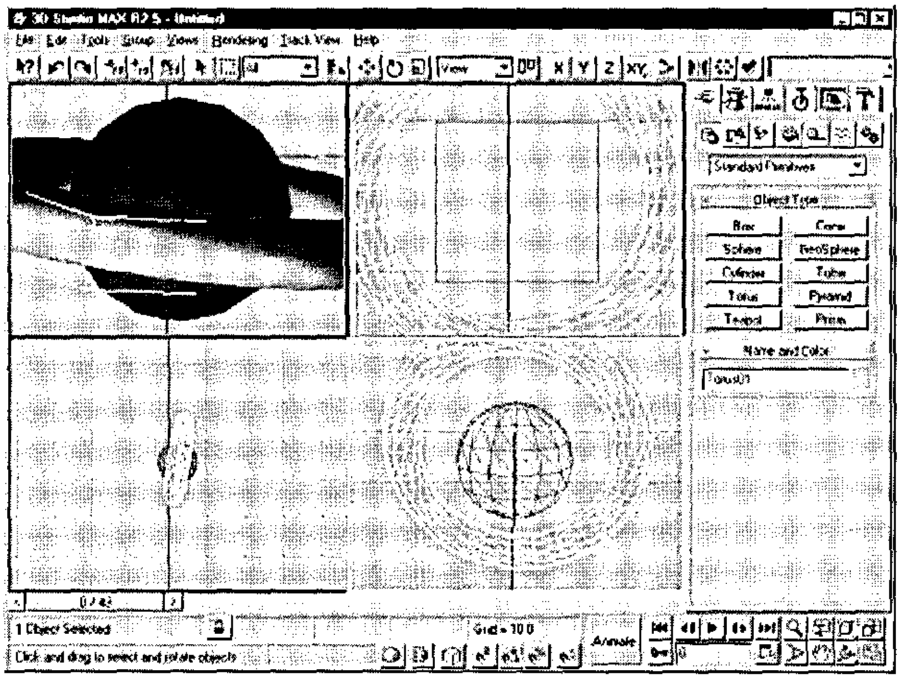


Figure 3.4 3D Studio Max Interface

3.8.1 MAX Script

Maxscript is a programming language, like Basic, Pascal, C or C++. The structure of Maxscript is similar to C. Like a computer program, a script consists of a series of instructions that affect elements on the screen. Maxscript provides access to the core functions of 3D Studio Max. Most of the tools available via the user interface are available via Maxscript (Bell, 1998). This scripting feature of 3D Studio Max allows the facial model to be manipulated and morphed into new heads such as female, male, elderly, child etc. Special commands and functions allow changes to be made to the position, scale, dimension, texture, lighting and shading of the objects in the viewport scene.

3.9 Free Form Deformation (FFD) Modifiers in 3D Studio Max

FFD's are usually used in computer animation but can be used for modelling as well. The FFD modifier surrounds the selected geometry with a lattice box. By adjusting the control points of the lattice (see Figure 3.5), the enclosed geometry can be deformed. In 3D Studio Max there are three FFD modifiers, each providing a different lattice resolution: 2x2x2, 3x3x3, and 4x4x4. The 3x3x3 modifier, for example, provides a lattice with three control points across each of its dimensions or nine on each side of the lattice. There are also two FFD-related modifiers FFD(Box) and FFD(Cyl) that provide supersets of the original modifiers. The FFD(Box/Cyl) modifiers can be used to create box-shaped and cylinder-shaped lattice free-form deformation objects and the number of points in the lattice can be set which makes them more powerful than the basic FFD modifier.

The source lattice of an FFD modifier is fitted to the geometry it's assigned in the stack. This can be a whole object, or a sub-object selection of faces or vertices. FFD modifiers can be controlled at three different levels:

- Control Points
- Lattice
- Set Volume

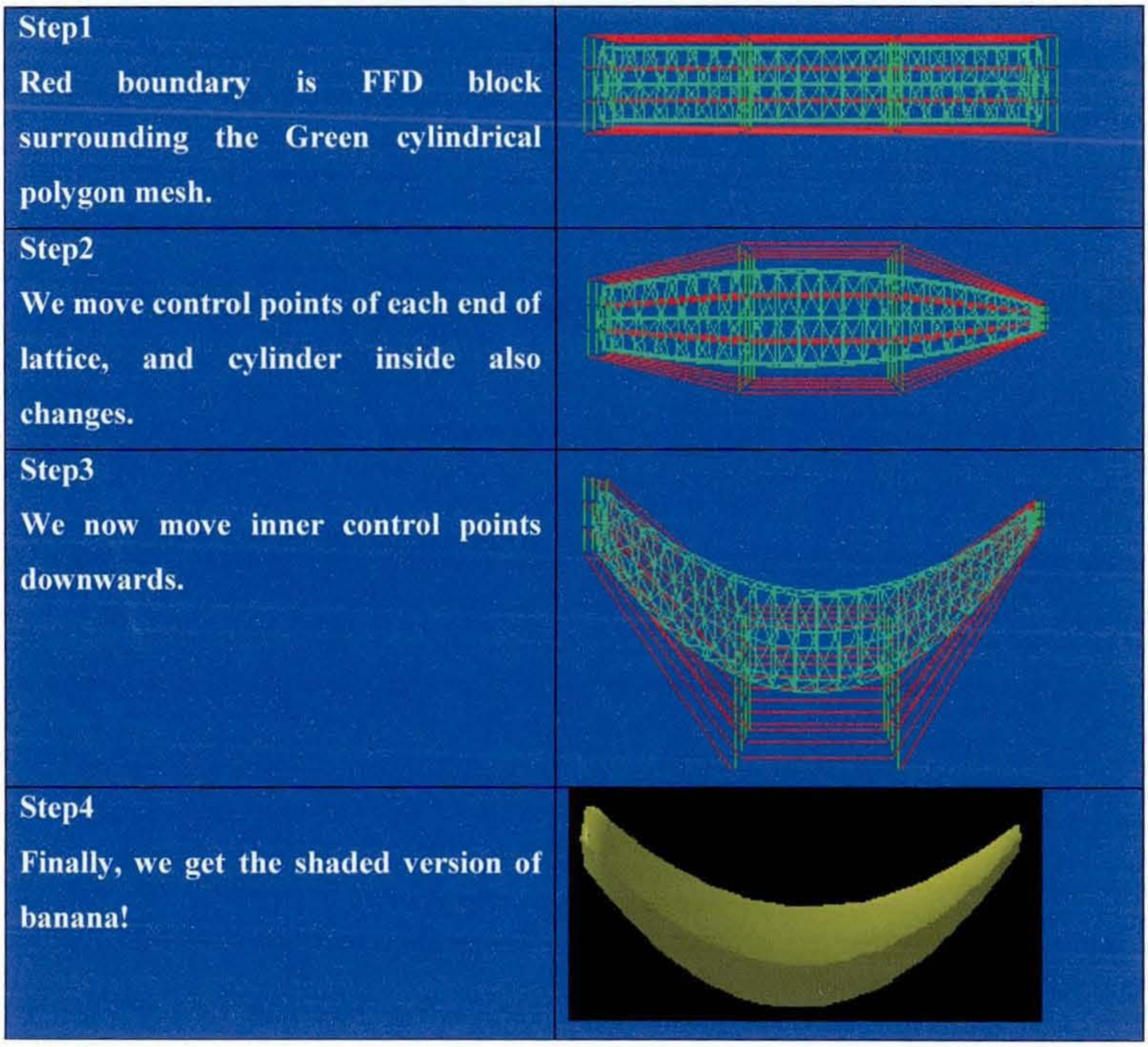
example, provides a lattice with three control points across each of its dimensions or nine on each side of the lattice. There are also two FFD-related modifiers FFD(Box) and FFD(Cyl) that provide supersets of the original modifiers. The FFD(Box/Cyl) modifiers can be used to create box-shaped and cylinder-shaped lattice free-form deformation objects and the number of points in the lattice can be set which makes them more powerful than the basic FFD modifier.

The source lattice of an FFD modifier is fitted to the geometry it's assigned in the stack. This can be a whole object, or a sub-object selection of faces or vertices. FFD modifiers can be controlled at three different levels:

- Control Points
- Lattice
- Set Volume

Control Points: At this sub-object level, control points of the lattice can be selected and manipulated, one at a time or as a group. Manipulating control points affects the shape of the underlying object. Standard transformation methods can be used with the control points to affect underlying geometry.

Lattice: At this sub-object level, the lattice box can be positioned, rotated, or scaled separately from the geometry. When the FFD is first applied, its lattice defaults to a bounding box surrounding the geometry. Moving or scaling the lattice so that only a subset of vertices lie inside the volume makes it possible to apply a localized deformation.



Step1
Red boundary is FFD block surrounding the Green cylindrical polygon mesh.

Step2
We move control points of each end of lattice, and cylinder inside also changes.

Step3
We now move inner control points downwards.

Step4
Finally, we get the shaded version of banana!

Figure 3.5 A simple illustration of FFD used to model a banana from a cylinder object

Set Volume: At this sub-object level, the deformation lattice control points can be selected and manipulated without affecting the underlying object. This control level allows the lattice to be fitted more precisely to irregular shaped objects, permitting finer deformation control (see Figure 3.6).

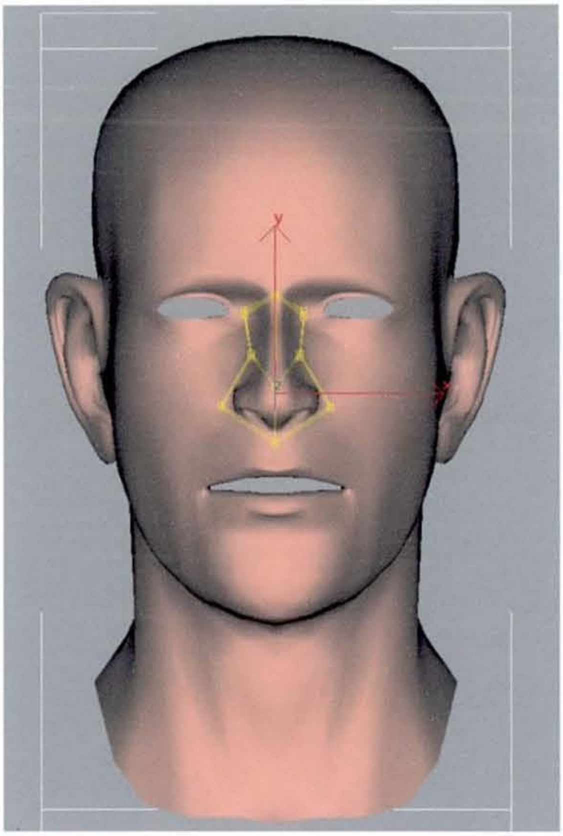


Figure 3.6 FFD(3x3x3) lattice with volume set modified to fit area of the nose precisely.

FFD modifiers have been used extensively through out the geometry of the baseline head models. FFD is the principal technology behind manipulation and control of geometry of the 3D face and its features effectively controlling the structure and conformation of the face. Details of the appliance and operation of this technology on the baseline head models have been discussed in detail in chapter 4. Since this chapter is concerned with the technology, tools and techniques used for development of 3D face models we will move on to describe the technology behind FFD.

3.10 The Technology behind FFD

Free-Form Deformation or FFD, can be thought of as a method for sculpturing solid models. Indeed, the sculpturing metaphor is stronger for solids than for surfaces because a lump of clay or a block of marble is a solid. Several researchers have promoted this sculpturing metaphor for geometric modeling, noting that it is a natural and familiar mode of thought for a designer or stylist. For example, Parent

(1977) discusses a "computer graphics sculptor's studio" for defining polygonal objects, and Brewer (1977) describes a planar shaping tool for manipulating sculptured surfaces. Other "lump-of-clay" modeling techniques are surveyed in Cobb (1984).

FFD involves a mapping from R^3 to R^3 through a trivariate tensor product Bernstein polynomial. An earlier use of $R^3$to R^3 mapping is found in Barr's innovative paper on regular deformations of solids (Barr, 1984). While not a free-form modeling technique, Barr's idea of twisting, bending and tapering of solid primitives is a powerful and elegant design tool. Brief mention of deformation is also made in Sabin (1970) and in Bezier (1974). Trivariate hyper-patches also are an R^3 - R^3 map, but the result is a distorted cube with six four sided faces.

FFD is a remarkably versatile tool. It can be applied to CSG based solid models as well as those using Euler operators. It can sculpt solids bounded by *any* analytic surface: planes, quadrics, parametric surfaces patches, or implicit surfaces. Furthermore, its application is not restricted to solid models, but it can also sculpt surfaces or polygonal data.

FFD can be applied locally while maintaining derivative continuity with adjacent, undeformed regions of the model, It can also be applied hierarchically, with each application being analogous to a sweep of the sculptor's hands. Constraints can be placed on the FFD to control the degree to which the volume of the solid changes, and in fact, there exist free-form deformations which are perfectly volume preserving.
Veenman (1982) suggests that the free-form surfaces used in practical engineering design fall into four categories: Aesthetic surfaces (the main design requirement is visual appearance); fairings or duct surfaces (a surface transition between two other surfaces of different cross-section); blends and fillets (smooth the intersection of two other surfaces}; and functional or fitted surfaces (high geometric constraint imposed to satisfy some functional requirement, such as a turbine blade). FFDs can create aesthetic surfaces and fairings. It is also possible to synthesize fillets in certain situations, but a general fillet and blending capability is not claimed. However, FFD can be used in conjunction with any fillet and blend formulation, such as those

discussed in Hoffmann (1985), Middleditch (1985) and Rockwood (1988). Functional surfaces are not discussed, although Sabin (1970) reports that a type of small displacement FFD is useful in the design of airplane wings.

3.10.1 FORMULATING FREE-FORM DEFORMATIONS

A good physical analogy for FFD is to consider a parallelepiped of clear, flexible plastic in which is embedded an object, or several objects, which we wish to deform. The object is imagined to also be flexible, so that it deforms along with the plastic that surrounds it.

Figure 3.5 illustrates this analogy using a cylindrical object embedded in clear, flexible plastic. The plastic has been deformed and the embedded cylinder is deformed in a manner that is intuitively consistent with the motion of the plastic.

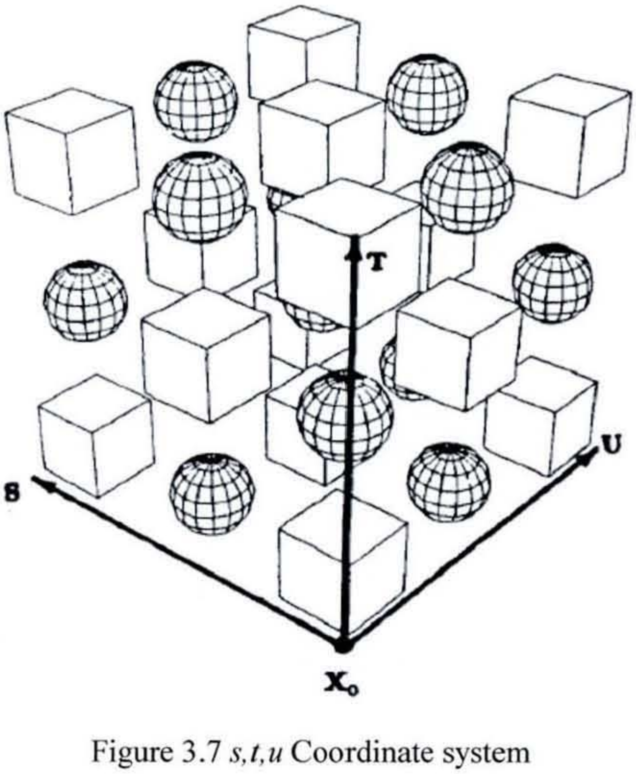


Figure 3.7 *s,t,u* Coordinate system

Mathematically, the FFD is defined in terms of a tensor product trivariate Bernstein polynomial. We begin by imposing a local coordinate system on a parallelepiped region, as shown in Figure 3.7. Any point *X* has (*s,t,u*) coordinates in this system such that

X= X_0+ sS + t T + uU.

(3.7)

The *(s,t,u)* coordinates of X can easily be found using linear algebra. A vector solution is

$$s = \frac{T \times U \cdot (X - X_0)}{T \times U \cdot S}, t = \frac{S \times U \cdot (X - X_0)}{S \times U \cdot T}, u = \frac{S \times T \cdot (X - X_0)}{S \times T \cdot U}$$

(3.8)

Note that for any point interior to the parallelepiped that 0<s <1, 0 <t <l and *O<u<l.*

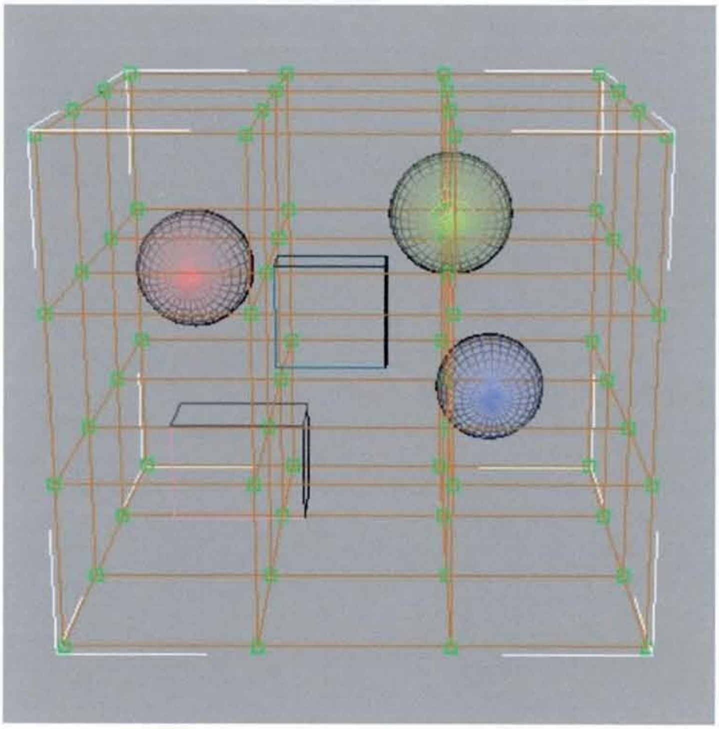


Figure 3.8 Undisplaced Control Points

We next impose a grid of control points P_{ijk} on the parallelepiped. These form *l*+1 planes in the S direction, *m*+l planes in the T direction, and *n*+l planes in the U direction. In Figure 3.8, *l*=1, *m=2,* and n=3. The control points are indicated by small green diamonds, and the brown bars indicate the neighbouring control points. These points lie on a lattice, and their locations are defined

$$P_{ijk} = X_0 + \tfrac{i}{l}S + \tfrac{j}{m}T + \tfrac{k}{n}U$$

(3.9)

The deformation is specified by moving the P_{ijk} from their undisplaced, lattice positions. The deformation function is defined by a trivariate tensor product Bernstein polynomial. The deformed position $X_{\int d}$ of an arbitrary point X is found by first computing its (s,t,u) coordinates from equation (1), and then evaluating the vector valued trivariate Bernstein polynomial:

$$X_{\int d} = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^{i} \left[\sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^{j} \left[\sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^{k} P_{ijk} \right] \right]$$

(3.10)

where $X_{\int d}$ is a vector containing the Cartesian coordinates of the displaced point, and where P_{ijk} is a vector containing the Cartesian coordinates of the control point.

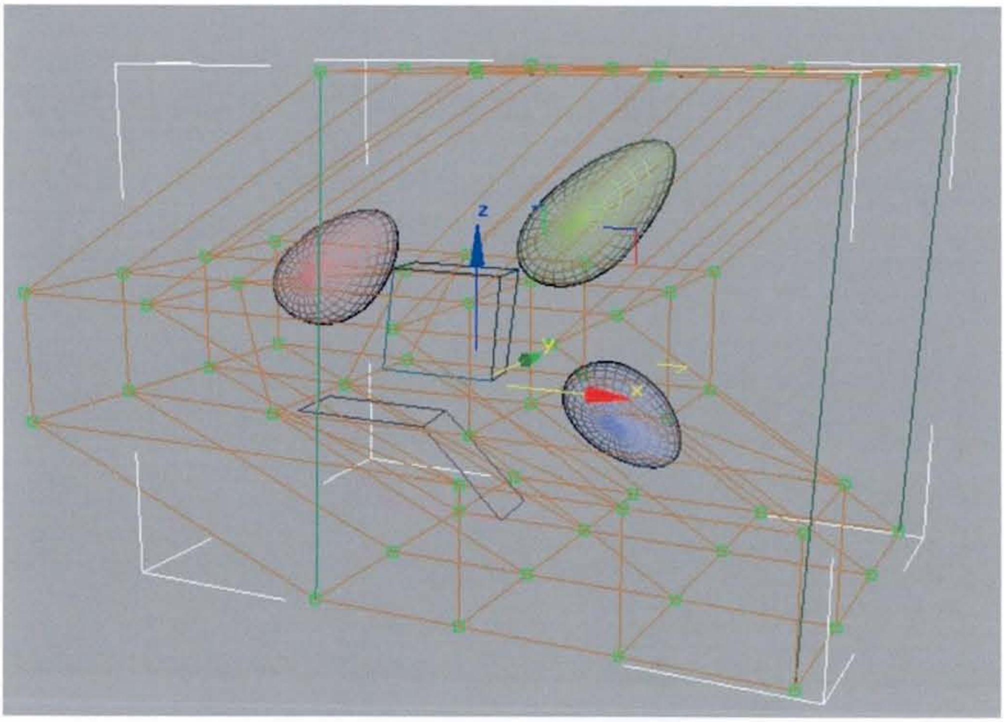


Figure 3.9 Control Points in Deformed Position

The control points P_{ijk} are actually the coefficients of the Bernstein polynomial. As in the case of Bezier curves and surface patches, there are meaningful relationships between the deformation and the control point placement. Note from Figure. 3.9 that the 12 edges of the parallelepiped are actually mapped into Bezier curves, defined by the control points which initially lie on the respective edges. Also, the six planar faces map into tensor product Bezier surface patches, defined by the control points that initially lie on the respective faces.

This deformation could be formulated in terms of other polynomial bases, such as tensor product B-splines or non-tensor product Bernstein polynomials.

3.10.2 Deformation Domain

FFD can be applied to virtually any geometric model. Figures 3.10 and 3.11 show deformed polygonal data. Only the polygon vertices are transformed by the FFD, while maintaining the polygon connectivity. Deformation of polygonal data is discussed more thoroughly in (Sederberg, 1986).

The FFD can be applied with equal validity to parametric and implicit surface representations. A very important characteristic of FFD is that a deformed parametric surface remains a parametric surface. This is easy to see. If the parametric surface is given by $x = f(\alpha, \beta), y = g(\alpha, \beta)$ and $z = h(\alpha, \beta)$ and the FFD is given by $X_{fd} = X(x, y, z)$, then the deformed parametric surface patch is given by $X_{fd}(\alpha, \beta) = X(f(\alpha, \beta), g(\alpha, \beta), h(\alpha, \beta))$.

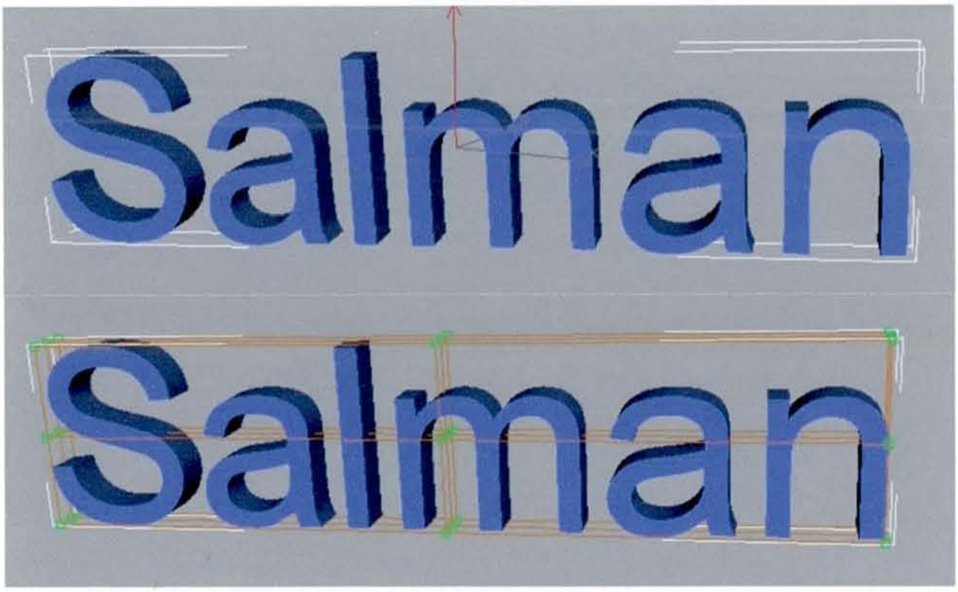


Figure 3.10 Undeformed Polygons

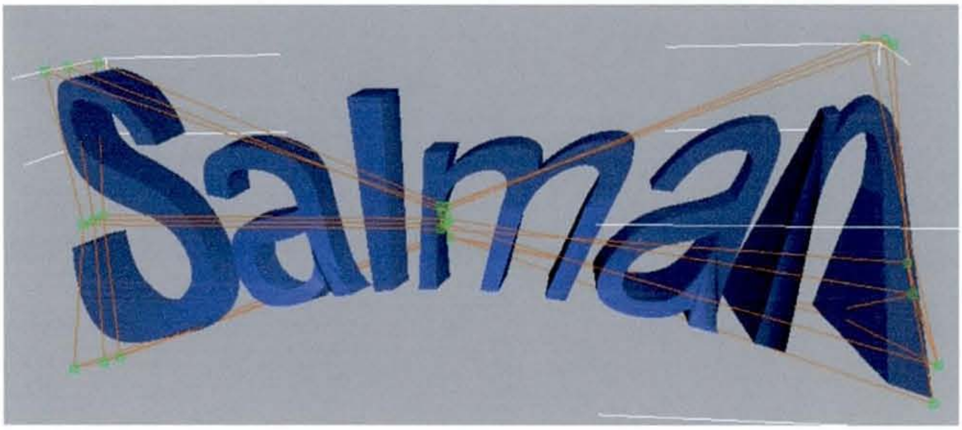


Figure 3.11 Deformed Polygons

This fact suggests important possibilities for solid modeling. For example, if one performs FFD in a CSG modeling environment only after all boolean operations are performed, and the primitive surfaces are planes or quadrics, then all intersection curves would be parametric, involving rational polynomials and possibly square roots.

Quadrics and planes make excellent primitives because they possess both implicit and parametric equations. The parametric equation enables rapid computation of points on the surface, and the implicit equation provides a simple point classification test - is a point inside, outside, or on the surface. To classify a point on a deformed quadric, one must first compute its s,t,u coordinates and substitute them into the implicit equation. The s,t,u coordinates can be found by subdividing the control point lattice, or by trivariate Newton iteration (see Parry, 1986). This inverse mapping requires significant computation, and can be a source of robustness problems, especially if the Jacobian of the FFD changes sign.

3.10.3 Local Deformations

A special case of continuity conditions enables local and isolated deformations to be performed. In this case, we might imagine that the neighbouring FFD with which we wish to maintain C^k is simply an undeformed lattice. We consider the problem of maintaining C^k along the plane where one face of the FFD intersects the geometric model. It is easy to show that sufficient conditions for a C^k local deformation are simply that the control points on the k planes adjacent to the interface plane are not moved. This is illustrated in Figures 3.12 and 3.13. Of course, C^k can be maintained across more than one face by imposing these conditions for each face that the surface intersects.

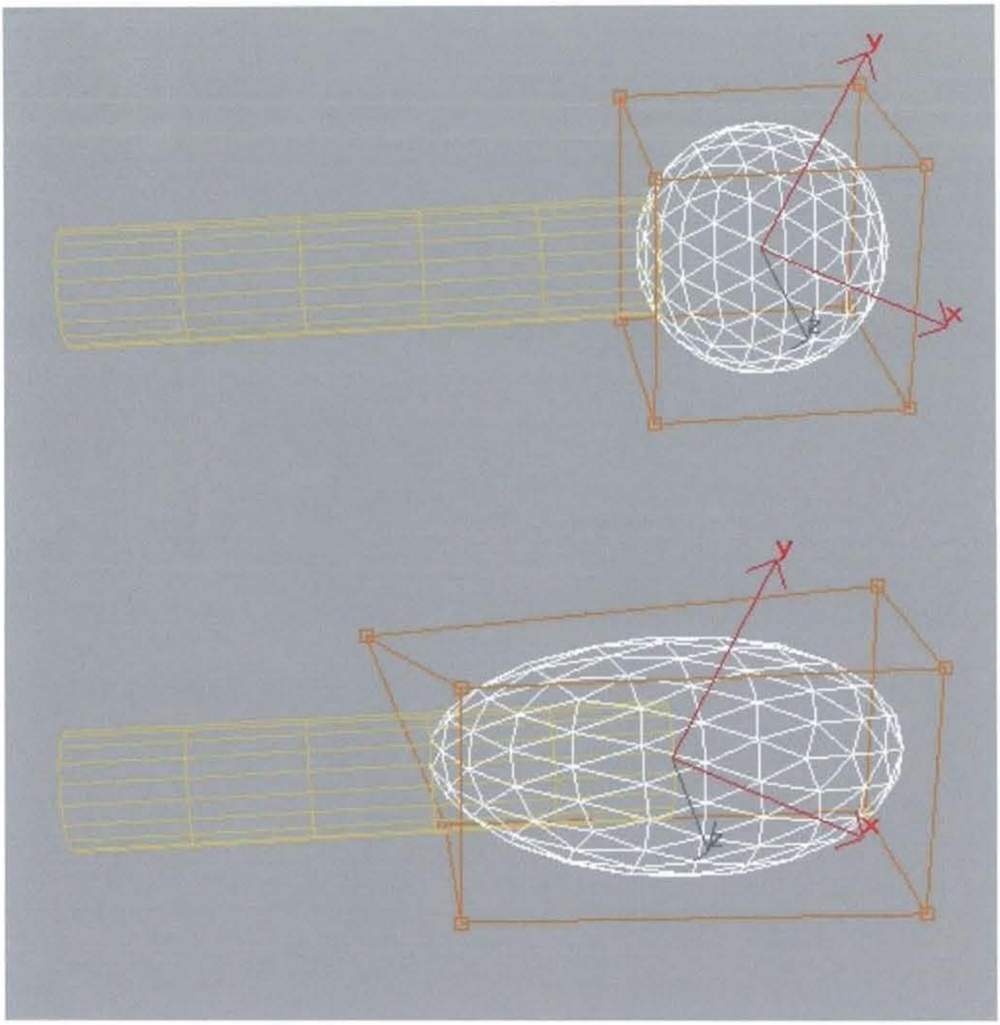


Figure 3.12 Local C^k Control Points

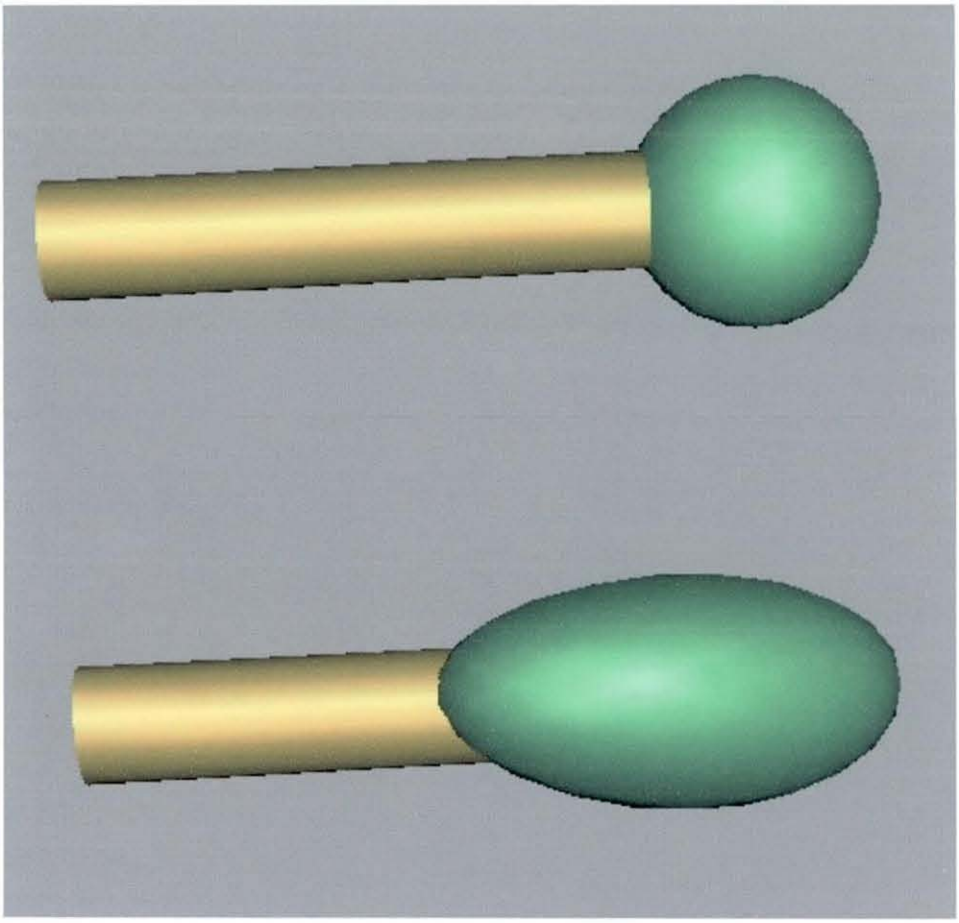


Figure 3.13 C^0 and C^1 Local Deformations

This local application lends to the FFD a capability that makes the technique strongly analogous to sculpting with clay. These local deformations can be applied hierarchically, which imparts exceptional flexibility and ease of use to the technique.

To summarize, FFDs strength and versatility can be listed as follows:

1. It can be used with any solid or surface modeling scheme.
2. It works with surfaces of any formulation or degree.
3. It can be applied locally or globally, and with derivative continuity.
4. It is very easy to use. The informal response of some professional stylists is that the strong sculpturing metaphor seems natural and familiar to them.
5. In addition to solid and surface modeling, it can be applied to polygonal models.

6. It provides indication of the degree of volume change, and a class of FFDs are even volume preserving.
7. Parametric curves and surfaces remain parametric under FFD.
8. It can be used for aesthetic surfaces, many fairing surfaces, and probably many functional surfaces.

Every technique has its limitations .and shortfalls, the following identify limitation of FFD:

1. It cannot perform general filleting and blending.
2. Local FFD forms a planar boundary with the undeformed portion of the object. To create an arbitrary boundary curve, one would have to begin with a FFD which is already in a deformed orientation, and then deform it some more. This would be quite costly.
3. Operations on trivariate Bernstein polynomials, such as subdivision, are much more costly than operations on bivariates.

3.11 Conclusion

We have provided an exhaustive examination of the tools and techniques available for 3D modelling. This chapter has investigated three main areas.

1. Facial modelling - existing research and applications.
2. Representation techniques available for 3D modelling and technology available for acquiring facial data
3. Tools and technology available for constructing 3D human head geometry.

Among the various representation techniques available for constructing 3D facial geometry, three (namely; NURBS, Bezier patches, and Ploygons) have been selected for experimentation with creating 3D head models. Details on constructing a human head using these techniques are presented in chapter 4. This chapter has also looked at some other technologies in the domain of 3D modelling such as FFD that has been critical in the development of a 3D facial image generation module.

Chapter 4

Development of 3D Facial Image Generation System – Procedures and Implementation

Abstract

This chapter describes the 3D facial image generation module. It begins with describing the procedure for modelling a human head using representation techniques discussed in chapter 3, namely; NURBS, Bezier Patches and Polygon Meshes. Following this is a description of the finalised baseline head and implementation of deformation controllers through out the geometry to control structure and conformation of the face and its features. We finally describe parameterisation of the head model and method for influencing the parameters using Maxscript to form the Head Generator Script.

Keywords: Geometric Modelling, NURBS, Beziers, Polygons, FFD, parameterisation, MaxScript, Image Generation, Facial Image, Human Head.

4.1 Introduction

Difficulties entailed in acquiring 3D facial geometry led to the decision of constructing 3D head models using 3D Studio Max, a commercial geometric modeler with an impressive library of 3D modeling tools and an efficient rendering engine (See chapter 3, section 3.9 for more details on 3D Studio Max).

The remainder of this chapter will concentrate on how the human heads were modeled using NURBS, Beziers and Polygons and parameterization of the finalized baseline head model for the 3D facial image generation module.

4.2 Constructing 3D Head Models

The 3D face model was constructed from a generic/canonical 3D face using two orthogonal photographs, a front and a side view.

Three modelling techniques were explored for the development of human head models.

1) NURBS
2) Bezier patches using a combination of quad and tri surface patches
3) Polygonal mesh

In all three modelling techniques front and profile images of a human face were used as reference (see Appendix B) for the construction of the basic spline outline. This section describes the modelling procedures for constructing the baseline head. It also examines the benefits and drawbacks of each technique and evaluates the suitability of the techniques for the end project.

4.2.1 NURBS Modelling

The exploration of head modelling began by creating a point spline, simply because its easier to use and can be converted to a C.V. spline if required (3D Café, http://www.3dcafe.com). The axis of the spline was set at the top of the head. Then the profile spline was cloned and rotated about five degrees. Each new profile spline was edited before moving on to creating a new clone, Figure 4.1.

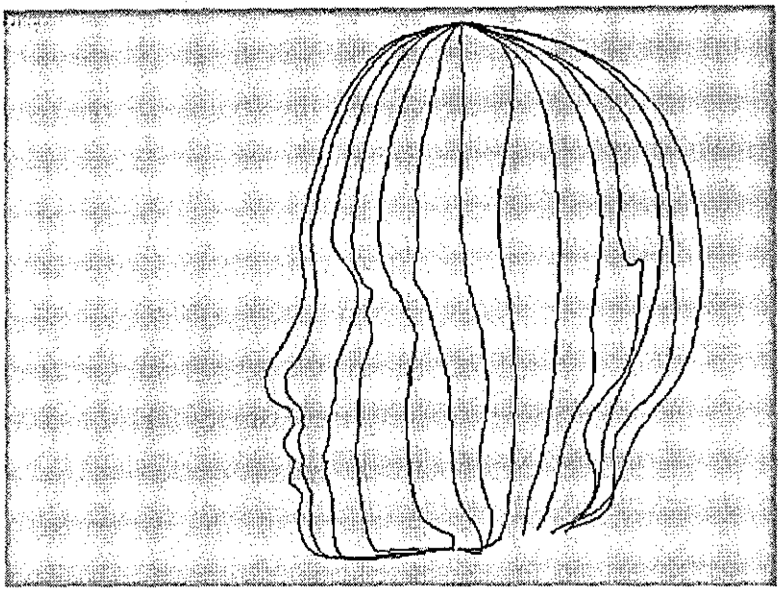


Figure 4.1 The model at point of cloning and editing spline profiles .

The points around the eyes were brought in and the lips made smaller. This was hard and took some time and effort. The procedure was repeated until the area around the ear was reached. Then the back spline was used following the same procedure of cloning and editing until the ear was reached. One thing to note is that the splines do not have the same number of points. Around the cheek a lot of points were erased because less detail was required in this area.

After the splines were created, they were attached and then collapsed to a point surface. It is important to collapse to a point surface in order to avoid dealing with a bunch of CV points. The profile spline was then started and a u-loft surface created between the splines resulting in an extremely ugly looking head. The real editing of the face could only be achieved by converting the head to a cv-surface but taking advantage of the point surface attributes the model could be cleaned up.

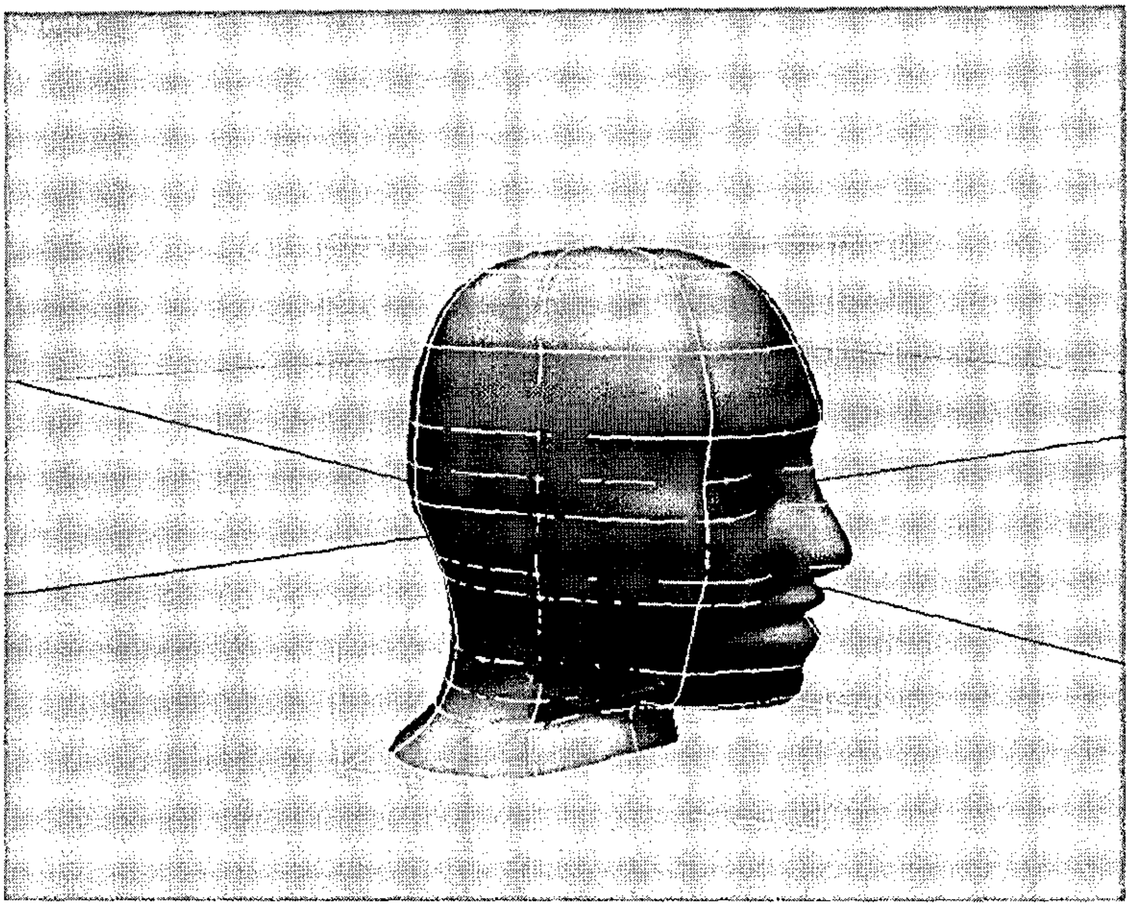


Figure 4.2 The first attempt at creating a head model using NURBS.

The most complex task of the modelling procedure began after making the surface independent thus converting the head to a cv-surface. The CV points needed a lot of editing before feature details began to show. A number of tools in 3D Studio Max were used for the editing namely *Affect Region* and CV *Point weight* (Bell 1998; Peterson, 1997; Boardman, 1998). *Affect region* allowed a region of points to be moved by moving just a single point and by increasing a point's weight the model could be given more detail (e.g. corners of the nose and mouth).

Beginning by editing the nose a row of points was added to produce the nostril of the nose. A point in the middle of the nostril was then selected and brought up in the z direction by increasing its weight. This pulled the nostril up more. Finally the points on the outside of the nose were edited and their weights increased.

The eyes were a challenge and took some time. One thing that helped was to look at the lattice and its direction. Often it is important to look at the points and the lattice and not the surface. A couple of rows of CV points were needed to add more detail but even with a lot of effort spent editing this region satisfactory results were not obtained.

The mouth came next, in order to edit this region of the face 3 rows of CV points were added, increasing their weights in the middle of the mouth and corners.

After half of the head was done it was cloned and mirrored, then joined to the other side. Joining the halves was not easy and no matter what was done the join function always flipped the normals of one side. This was solved by creating a blend between the two halves, making the blend independent and joining all 3 surfaces together. The drawback of the procedure is that the centre column of points cannot be erased. The model was difficult to construct and even after spending long hours at editing it a suitable head could not be constructed.

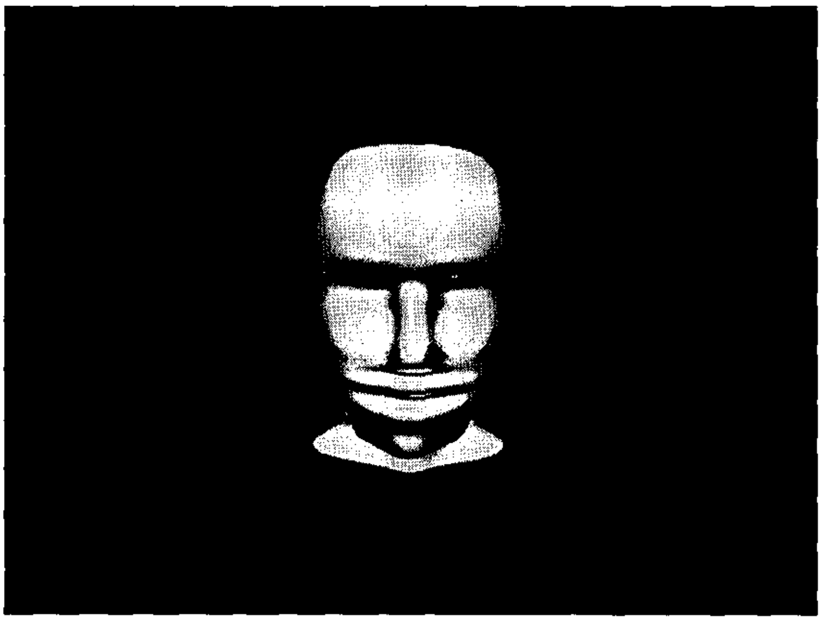


Figure 4.3 The final unsatisfactory head model constructed using NURBS.

4.2.2 Bezier Patch Modelling

The first step in preparing this model involved creating a spline layout of the head using the side and front head images as reference. Using the line tool the front and side layout views were created. Further lines were added for the main features of the face such as sides of the nose, outline of the lip, curve of the cheek bone etc. This was the most important part of the modelling process since it was to form the basis of the template. The quality of the final head is strongly dependent on the quality of the template model.

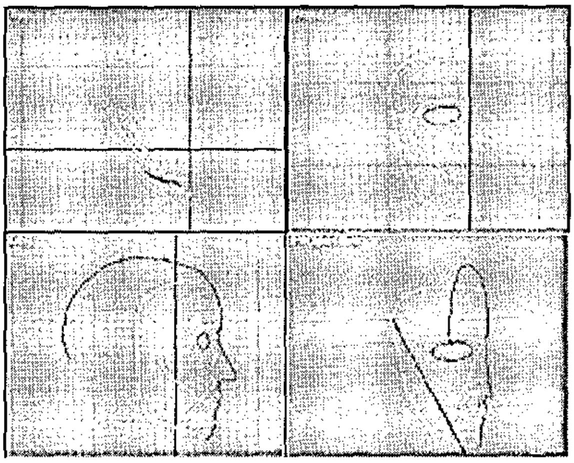


Figure 4.4 Four views of the spline lay out for the patch model

The next step involved adding patches to the head template. This began by creating a small patch by the chin. Editing the patch involved switching between vertex and edge modes. In vertex mode it is possible to move the vertices and Bezier handles to match up the patch with the outline. The patch handles were adjusted until a suitably round chin shape was obtained. Then after switching back to edge mode to view the patch laid out, another quad patch was added to the left edge leading to the cheek area. The next step involved successive adding and editing of patches, controlling the vertices to match up with the outline. Patching up the nose outline proved harder in comparison to the rest of the face. The curved contours of the nose meant quad patches were not going to work so tri patches were used. Tri patches can be useful for filling in tight curved shapes because, not only are they made up from triangular faces but, when added to another patch form a triangular shape.

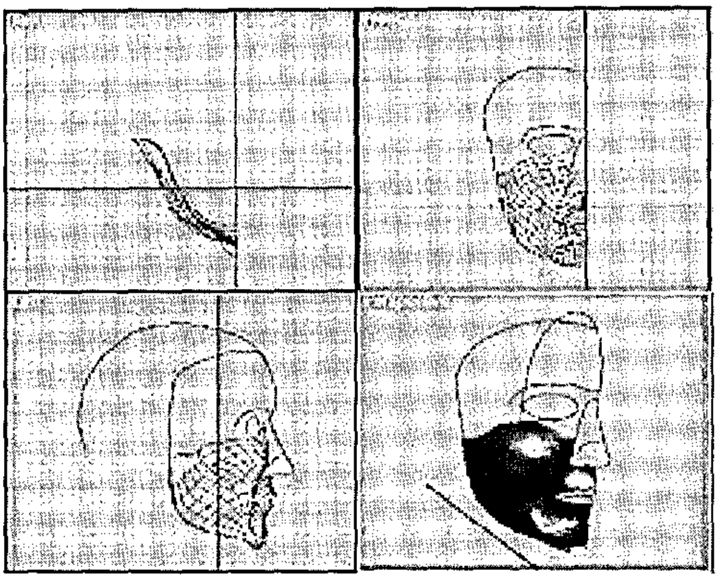


Figure 4.5 Showing the emerging patch structure.

The final step involved cloning and mirroring the half head. The clone object function in the Edit modifier tool set (Peterson, 1997; Boardman, 1998) of 3D Studio Max was used to achieve this. The two patches had to be attached next, this required welding the vertices down the middle of the head. The best option available was to select one set at a time and perform the weld between the adjacent vertices. The final result was a head constructed using Bezier patches.

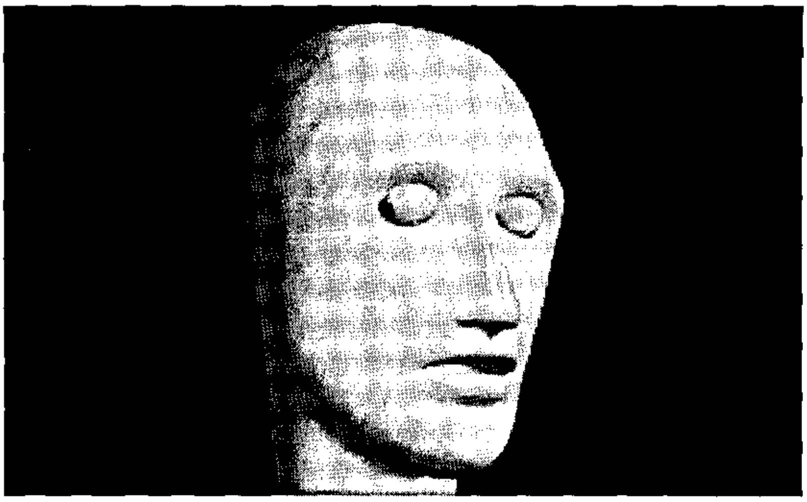


Figure 4.6 A head constructed using Bezier patches

4.2.3 Polygonal Modelling

The first step in preparing the model involved creating a spline layout of the head using the side and front head images as references. Once the proper contours were laid down *Create Line* and *Refine* options (Bell 1998) were used to fuse together all the main lines, while making sure that each section of the face was divided into Quad or Tri sections.

After connecting and unifying the spline cage, all of the vertices were selected and converted to *cornered vertices*. This was very important because it simplified the next step, which was to pull out the flat spline cage to give it another dimension. Before proceeding to pull out the vertices, the viewport was configured to show the left and front view of the spline cage. The front viewport was used for selecting the appropriate vertices and the left viewport for pulling.

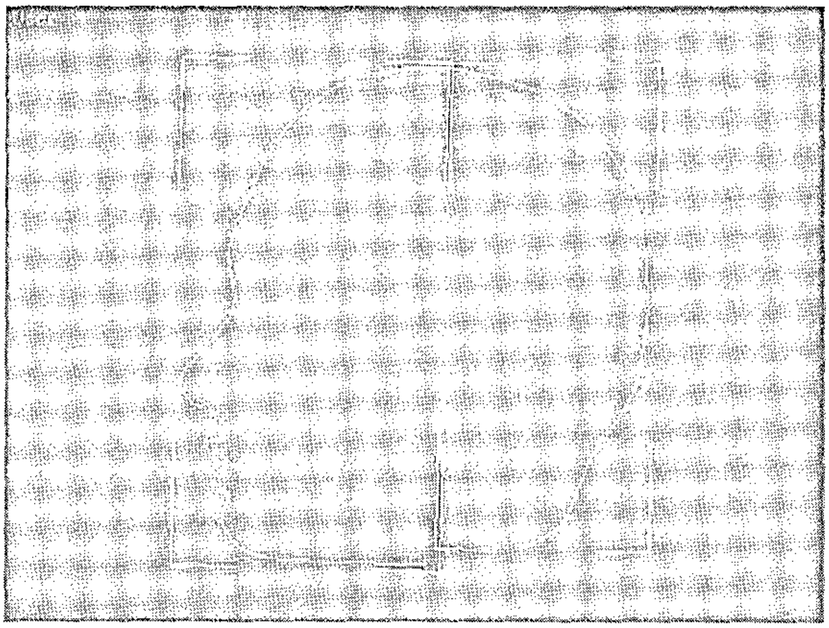


Figure 4.7 Spline lay out of the head for polygonal modelling.

Once all the vertices on the right edge of the front viewport had been selected, the left viewport was activated, and the selected vertices were pulled out along the X axis. This way the structure of the face could be preserved whilst editing the profile of the face by pulling or pushing the desired vertices along the X-axis.

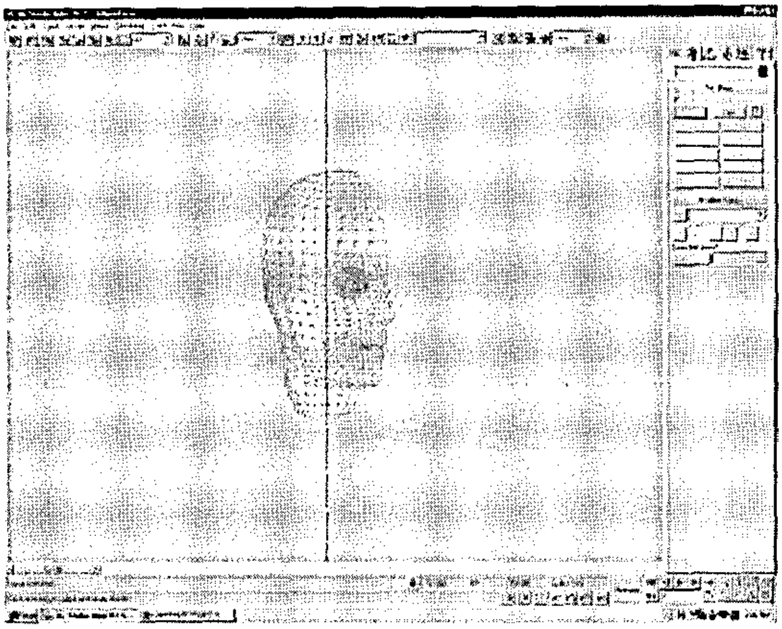


Figure 4.8 The half mesh of the developing head.

Once everything had been pulled out accordingly a surface modifier was applied to collapse the surfaced spline cage and turn it into an editable mesh. The viewport was configured to show the object with Edge Faces turned on as well as Mesh Smoothing and Highlight. When working with complex meshes it is helpful to have Edge Faces on since it shows the actual contours of the wireframe which, in turn makes it easier to modify and edit the mesh

In order to see how the face looks as a whole, an Instanced [an interdependent copy of MAX object] copy of the control mesh was made so that whatever modifications were made with the original mesh, the instanced version would always update accordingly. This is very important since, to see if the face looks reasonably realistic, it will always need to be seen in its entirety, and not just the halved section..

A *Meshsmooth* modifier was then applied onto the mesh. *Meshsmooth* is a built-in function of 3D Studio Max that adds faces to the mesh like Tessellate does and softens the edges. In effect, it refines the topology of the mesh. To refine the control mesh the Cut tool was used to edit areas such as the eyes, the nose, and the lips to carve out extra feature detail on the head.

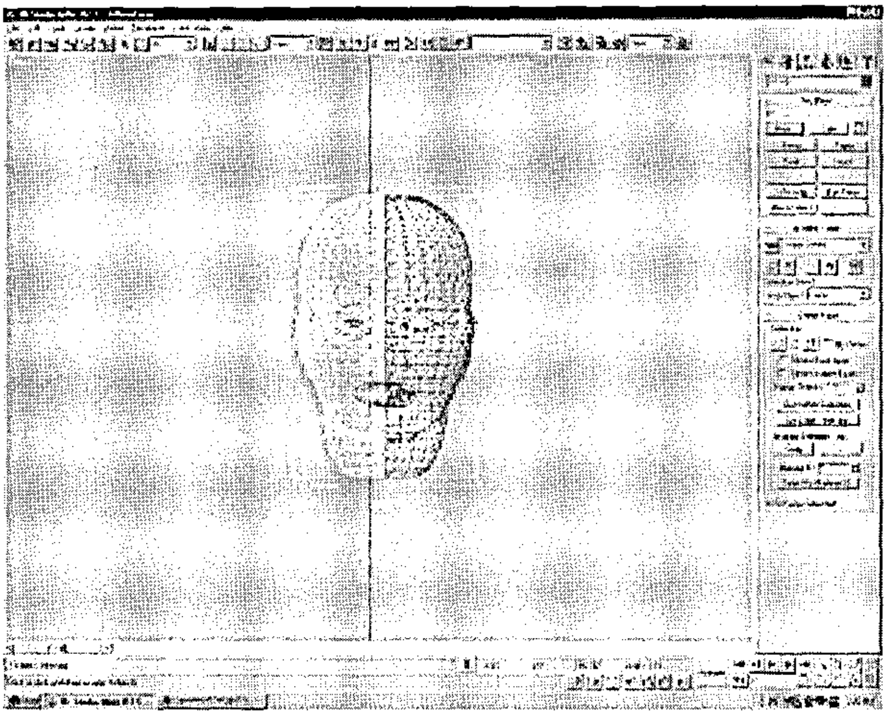


Figure 4.9 The full head mesh.

On completion of the refinement and detail work the Instanced copy of the original mesh was deleted, and then re-mirrored again as a copy version. The reason being that an instanced version of the mesh can not be attached to the original mesh. In order to get the whole head the mirrored copy or clone had to be attached to the original mesh. Once everything was ready, and the mirrored mesh attached, the very last step was to Weld all the vertices that meet between the edges of the two halves to complete the full head model.

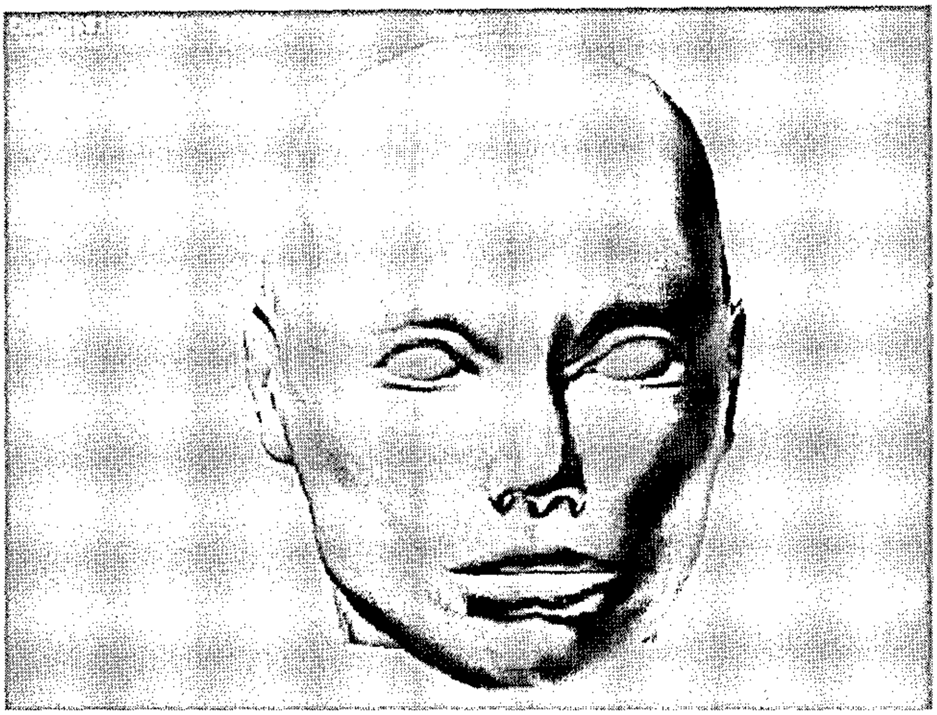


Figure 4.10 A basic human head modelled using polygonal mesh.

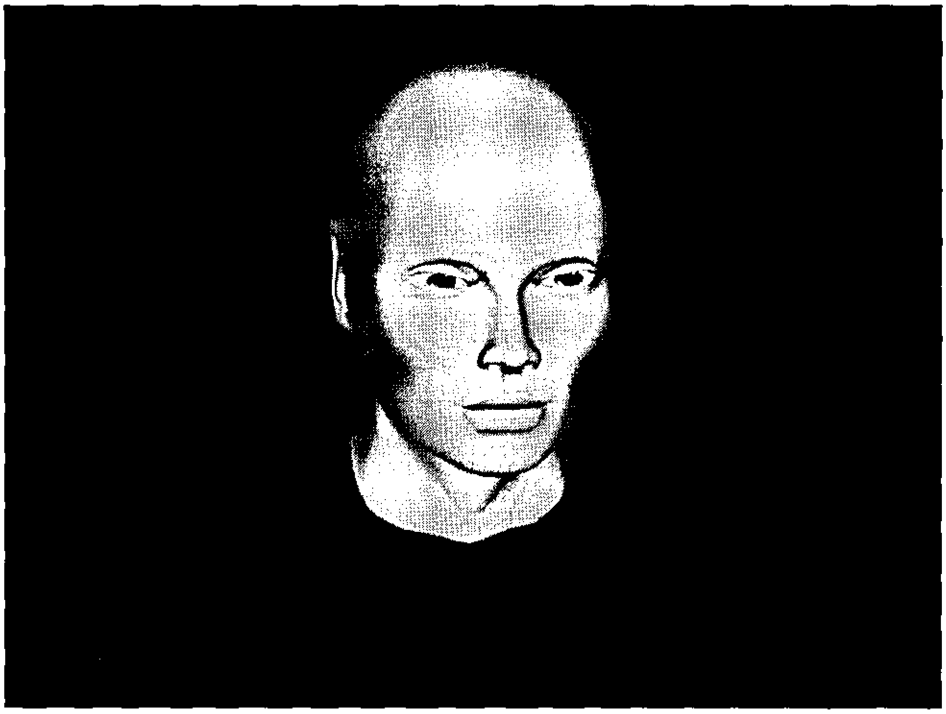


Figure 4.11 Rendered Image: Final baseline head achieved editing and adding greater detail to the basic polygon mesh model

4.3 Evaluation of Modelling Procedures and Results

The head models achieved from the three different techniques varied in quality and geometric detail. Of the three models produced the polygonal mesh model was by far the finest. NURBS is an excellent tool for organic modelling, like constructing faces or characters, however difficulty in using the NURBS tools made the task complicated and tedious. Modelling using patches required a simple but tedious procedure of building from the foundation outward. The result was a head that faired better than the NURBS head but required tremendous amount of editing to line up the patches accurately. The polygon head was easy to edit and once the head was built, application of mesh optimisation, mesh smoothing, and selective shading techniques allowed for a smooth rendered surface. Another aspect of modelling with polygons is that many 3D digitising systems output polygonal data which may then be subsequently matched to Bezier patches or NURBS, however the original representation is often in polygons.

A very important observation noted during the construction of 3D head models is that while there is a large volume of data and experimentation available regarding the mathematical theories and application in construction of complex scenes and objects in 3D. There is also a huge amount of material available for artists on how to develop 3D objects as complex as the human head. There is however little or no reference available with regards to how inexperienced users and non artists could approach the task of modelling a human head or other shapes of equivalent complexity. We believe there is a need to develop alternative means of constructing imagery other than the complicated and often repetitive tools offered by the GUI of existing software.

4.4 Baseline Head Models

The heads constructed using polygons were by far the best effort and thus selected as the baseline model for the system (see Figures 4.12 and 4.13). The baseline head models comprise of tri and quad facets and have vertex counts of roughly 4500 - 5000, and polygon counts of around 7500. An optimisation modifier built in to the modelling tool can easily reduce this count depending on where and how the head models need to be used (Peterson,1997).

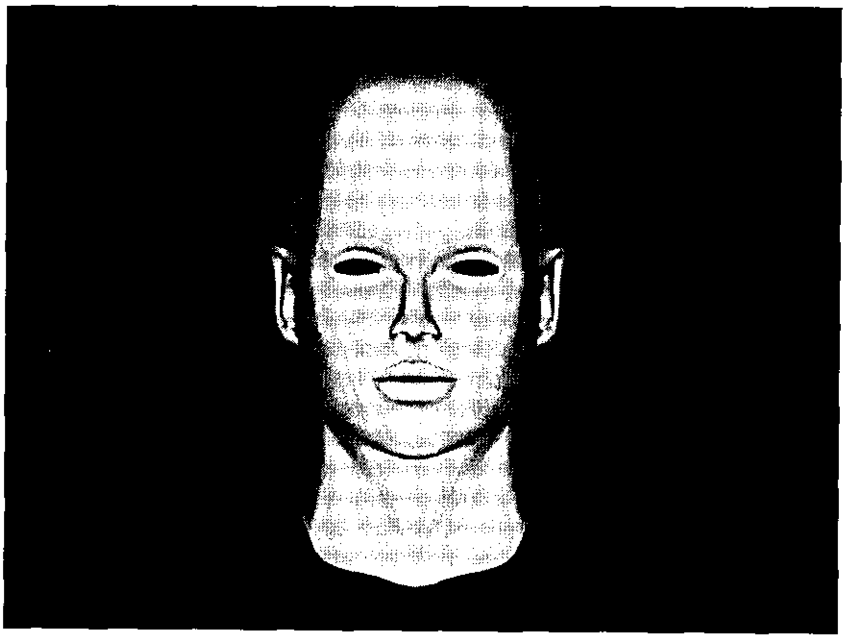


Figure 4.12 Female Baseline Head Model

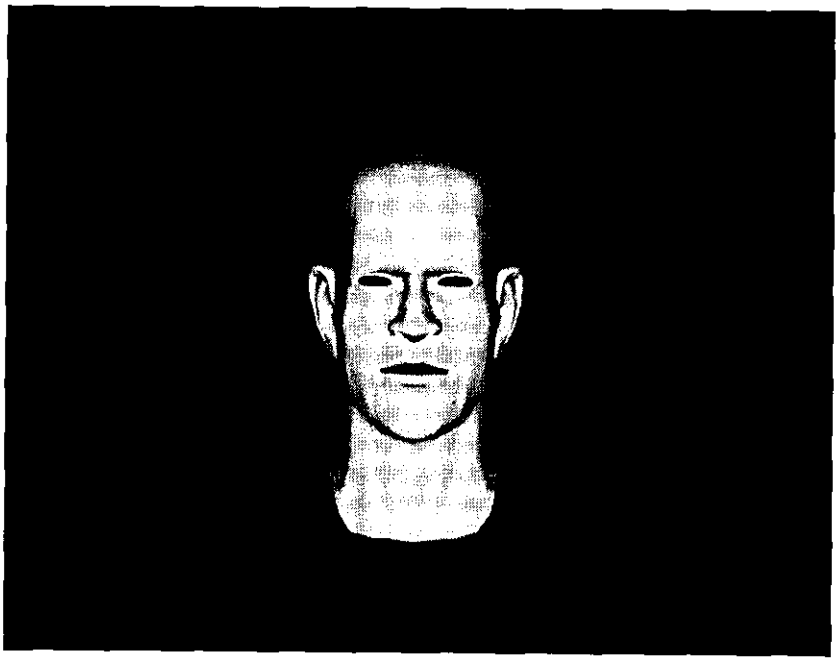


Figure 4.13 Male Baseline Head Model

4.5 Applying FFD Modifiers to Baseline Head

Construction of a basic parametric head model was the building block for an automated 3D facial image generation system. To this effect once a baseline 3D head was constructed, the next phase involved assigning deformation control groups (FFD) through out the geometry of the head model. These deformation groups would allow transformation of specific areas of the geometry effectively executing non-uniform scale, skew, rotation and translation.

Previous work on parametric models (Parke, 1972; Pearce *et al.*, 1986; Ohmura, 1988; Patel, 1991) are the only ones to date that allow facial conformation control, i.e., changes from one individual face to another. The work in this thesis does not replicate the techniques developed by Parke, Pearce or Patel for creating parametric heads. Instead the head models developed here are given parametric properties using deformation control groups applied through out the head geometry and for each group its respective control points are assigned variables/parameters to control the underlying geometry.

The head and its features are controlled by a deformation mesh, the mesh can be regarded as a deformation lattice with control points or handlers that allow local transformation of vertices, effectively performing scale, translate, skew and rotation of the associated geometry. The deformation group controls the basic head shape by pulling vertices towards an imaginary ellipsoid. The Scaling group performs a non-uniform scaling in each direction. The Skew controls cause the scaling to vary as a function of position. Other deformation control groups affect the facial features like nose, eyes, ears, cheeks, jaws, and forehead in terms of size and shape. The deformation modifiers applied to the baseline head geometry are known as FFD modifiers in 3D Studio Max.

The FFD modifier surrounds the selected geometry with a lattice box. By adjusting the control points of the lattice (see Figure 3.5), the enclosed geometry can be deformed. In 3D Studio Max there are three FFD modifiers, each providing a different lattice resolution: 2x2x2, 3x3x3, and 4x4x4. The 3x3x3 modifier, for example, provides a lattice with three control points across each of its dimensions or nine on each side of the lattice. There are also two FFD-related modifiers FFD(Box) and FFD(Cyl) that provide supersets of the original modifiers. The FFD(Box/Cyl) modifiers can be used to create box-shaped and cylinder-shaped lattice free-form deformation objects and the number of points in the lattice can be set which makes them more powerful than the basic FFD modifier.

FFD modifiers in 3D Studio Max allow a great deal of flexibility in the control of the deformation lattice. Volume of the deformation lattice can be edited and modified allowing precise fitting of the lattice over the underlying geometry. FFD control points can be edited manually or via MaxScript using transformation functions to affect the vertices of the underlying geometry consequently changing the structure of the confined area.

4.5.1 Applying FFD modifiers to the Head and Features

Head: A deformation modifier with a 4x4x4 lattice resolution was attached to the outline head geometry to control aspects like the head width, height and depth using non-uniform scaling and skewing (see Figure 4.14). Other control points were assigned to areas such as the forehead and face to allow adjustment to forehead slope and face compression (squash in or pull outwards). The FFD is denoted as Head_Modifier with single or grouped control points assigned variables as follows: x_pull, y_pull, z_pull, head_widh, head_depth, height height, head_width_skew, head_depth_skew, head_height_skew, face_squash, and forehead_slope.

Nose: A deformation modifier with a 3x3x3 lattice resolution was attached to the nose and area around it to control aspects like nose width, length, bridge, hook or pug amount and pull up amount (see Figure 4.15). The FFD is denoted as Nose_Modifier with single and grouped control points assigned variables as follows: nose_width, nose_length, nose_bridge, nose_pullup, and nose_hook.

Chin: A deformation modifier with a 2x2x2 matrix was applied to the chin and area around it to control aspects like chin extent, tilt and acccent amount (see Figure 4.16). The FFD is denoted as Chin_Modifier with single and grouped control points assigned variables as follows: chin_extent, chin_tilt_amount, and chin_accent.

Jaw: A deformation modifier with a 3x3x3 lattice was attached to the area around the jaw to control jaw width (see Figure 4.17). The FFD is denoted as Jaw_Modifier with grouped control points assigned variables as follows: jaw_width and jaw_width_uniformity.

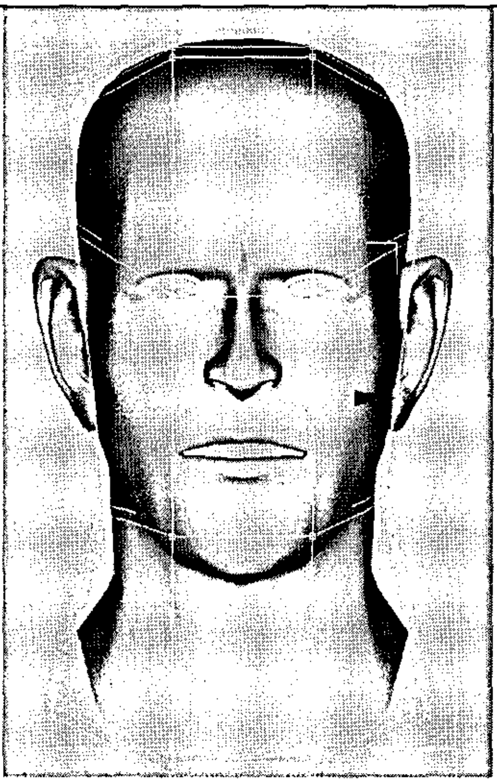


Figure 4.14 FFD(4x4x4) Applied to outline Head Model

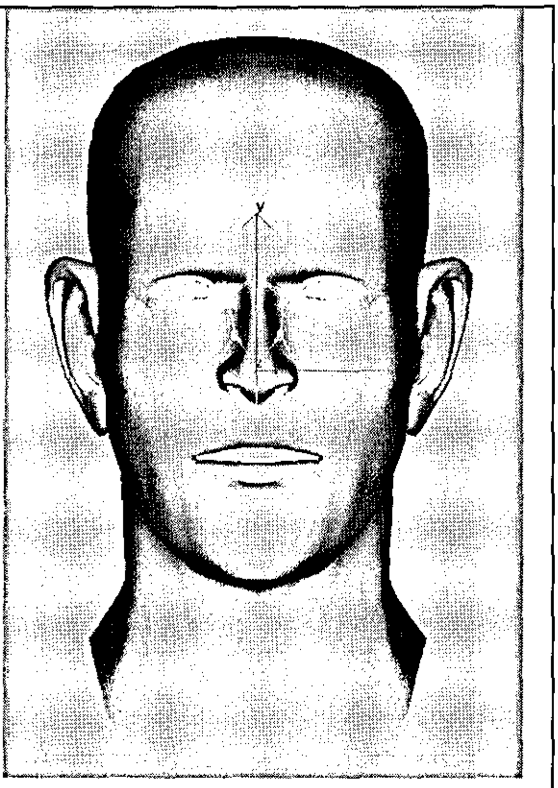


Figure 4.15 FFD(3x3x3) Applied to Nose

Cheek: A deformation modifier with a 3x3x3 lattice resolution was attached to the cheek and area around it to control aspects like cheek bones extrusion, cheek bone position and cheek curvature (see Figure 4.18). The FFD is denoted as Cheek_Modifier with single and grouped control points assigned variables as follows: cheekbones_extrusion, cheekbones_z_pos, cheek_curvature, cheek_curvature_z_falloff, and cheek_curvature_y_falloff.

Eyes: A deformation modifier with a 2x2x2 matrix was attached to each eye to control aspects like eye separation, eye roundness and rotation (see Figure 4.19). The FFD is denoted as Eye_Modifier with grouped control points assigned variables as follows: eye_separation, eye_rotation, eye_bottom_roundness and eye_top_roundness.

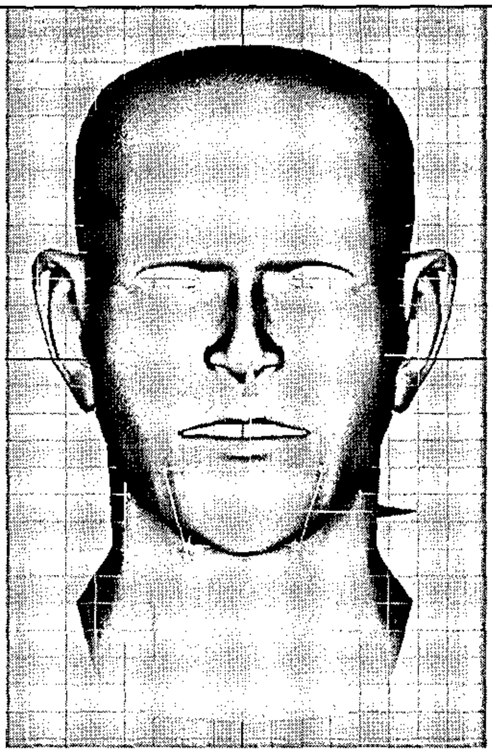


Figure 4.16 FFD(2x2x2) Applied to Chin

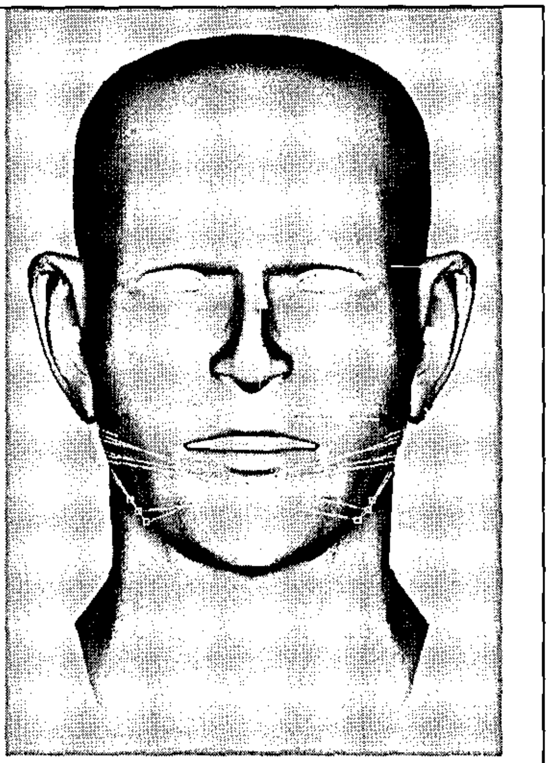


Figure 4.17 FFD(3x3x3) Applied to Jaw

Ears: Deformation modifiers with a 3x3x3 lattice resolution were attached to each ear to control aspects like ear height, depth, rotation and lobe length (see Figure 4.20). The FFD is denoted as Ear_Modifier with grouped control points assigned variables as follows: ear_height, ear_depth, ear_rotation and ear_lobe_length.

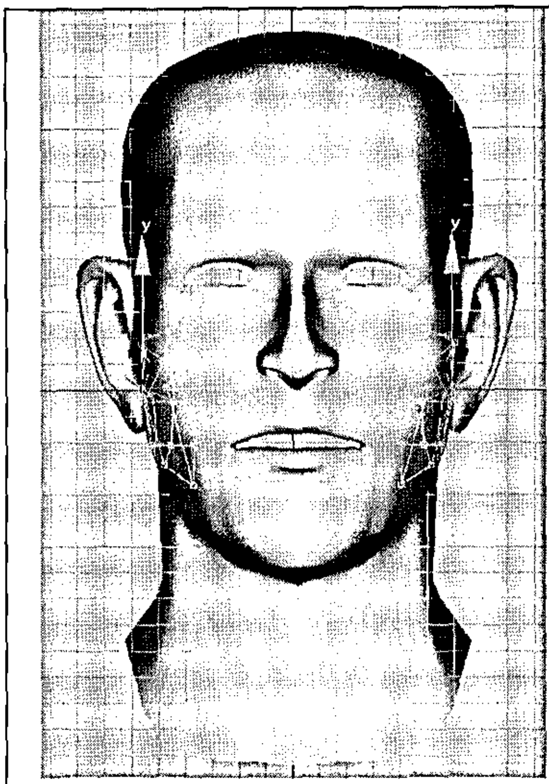


Figure 4.18 FFD(3x3x3) Applied to Cheek

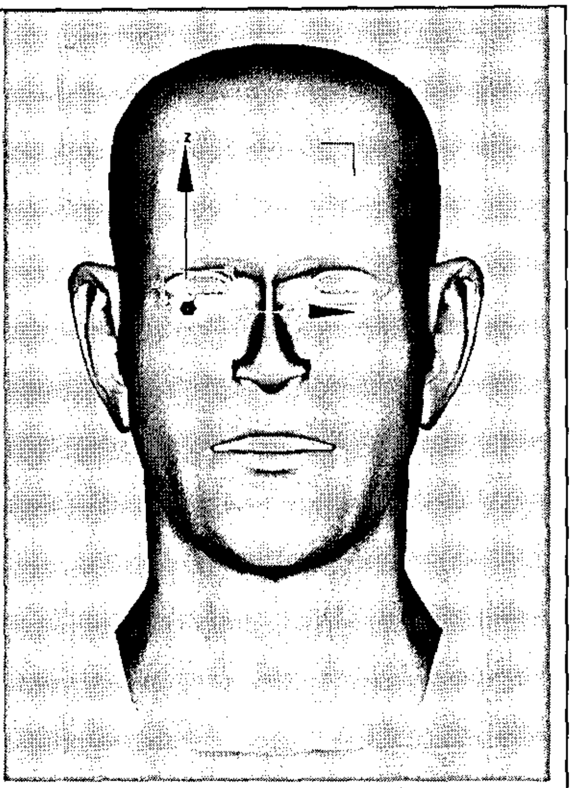


Figure 4.19 FFD(2x2x2) Applied to Eyes

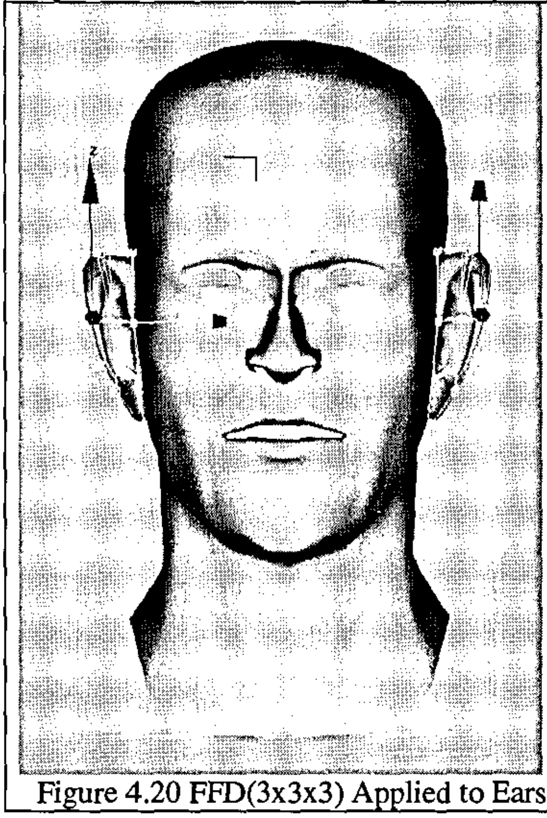


Figure 4.20 FFD(3x3x3) Applied to Ears

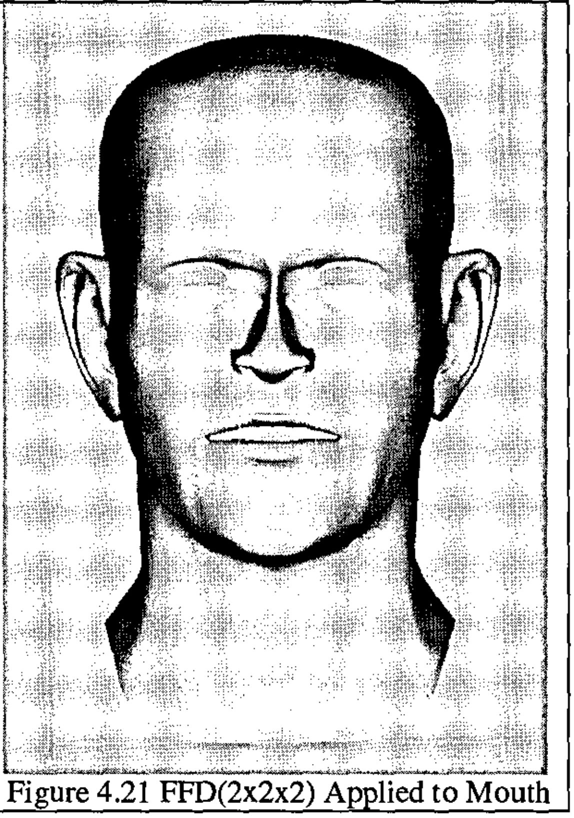


Figure 4.21 FFD(2x2x2) Applied to Mouth

Mouth: A deformation modifier with a 3x3x3 lattice resolution was attached to the mouth and area around it to control aspects like mouth width and mouth protrusion (see Figure 4.21). The FFD is denoted as Mouth_Modifier with single and grouped control points assigned variables as follows: mouth_width and mouth_protrude.

4.6 Control of Head Geometry via MaxScript.

3D Studio Max incorporates a powerful scripting tool called Maxscript. Maxscript is a programming language like Basic, C or C++. The structure of Maxscript is similar to C it consists of a series of instructions that affect elements on the screen. Maxscript provides access to the core functions of 3D Studio Max. Most of the tools available via the user interface are available via Maxscript (Bell, 1997; Peterson, 1998). This scripting feature of 3D Studio Max is used to edit the head model using mathematical prescriptions for adjusting and controlling various organic features. The script is used to automate the 3D head geometry modification procedure. It handles the task of reading the parameters produced by the Natural Language Processing engine (NLP) and passes them on to the appropriate command function to create or edit the human head model.

The head and its features are controlled by deformation modifiers (FFD) that allow local transformation of vertices, effectively performing scale, translate, skew and rotation of the associated geometry. The deformation group controls the basic head shape by pulling vertices towards an imaginary ellipsoid. The Scaling group performs a non-uniform scaling in each direction. The Skew controls cause the scaling to vary as a function of position. Other parameters affect the facial features like nose, eyes, ears, cheeks, jaws, and forehead in terms of size, shape and orientation.

The head parameters associated to the deformation modifiers are adjusted through script code called Head Generator Script or HGS (See Appendix B) using a complete set of predefined variables as mentioned in section 4.5. A comprehensive list of the variables and corresponding parameters are listed in Table 4.1.

Parameter	Variable Name	Type	Lower Limit	Upper Limit	Default
Head Type	head_type	integer	0	100	N/A
Deformation X-Pull	x_pull	float	1.0	100.0	100.0
Deformation Y-Pull	y_pull	float	1.0	100.0	100.0
Deformation Z-Pull	z_pull	float	1.0	100.0	100.0
Width Scaling	head_width	float	0.1	1000.0	1.0
Width Skew 1	head_width_skew_1	float	-1.0	1.0	0.0
Width Skew 2	head_width_skew_2	float	-1.0	1.0	0.0
Depth Scaling	head_depth	float	0.1	1000.0	1.0
Depth Skew	head_depth_skew	float	-1.0	1.0	0.0
Height Scaling	head_height	float	0.1	1000.0	1.0
Height Skew	head_height_skew	float	-1.0	1.0	0.0
Face Compression	face_squash	float	0.0	20.0	1.0
Forehead Slope	forehead_slope	float	-1.0	1.0	0.0
Nose Width	nose_width	float	0.0	2.0	1.0
Nose Length	nose_length	float	0.0	3.0	1.0
Nose Pullup	nose_pullup	float	0.0	2.0	1.0
Nose Bridge	nose_bridge	float	-1.0	1.0	0.0
Nose Hook/Pug Amount	nose_hook	float	-1.0	1.0	0.0
Chin Extent	chin_extent	float	0.0	2.0	1.0
Chin Tilt Amount	chin_tilt_amount	float	0.0	2.0	1.0
Chin Accent	chin_accent	float	-1.0	1.0	0.0
Jaw Width	jaw_width	float	-1.0	1.0	0.0
Jaw Width Uniformity	jaw_width_uniformity	float	-1.0	1.0	0.0
Cheek Bones Extrude	cheekbones_extrude	float	-1.0	1.0	0.0
Cheek Bones Z Position	cheekbones_z_pos	float	-1.0	1.0	0.0
Cheek Curvature	cheek_curvature	float	-1.0	1.0	0.0
Cheek Curvature Z Falloff	cheek_curvature_z_falloff	float	0.0	1.0	0.5
Cheek Curvature Y Falloff	cheek_curvature_y_falloff	float	0.0	1.0	0.5
Eye Separation	eye_separation	float	0.0	2.0	1.0

Parameter	Variable Name	Type	Lower Limit	Upper Limit	Default
Eye Top Roundness	eye_top_roundness	float	-1.0	1.0	0.0
Eye Bottom Roundness	eye_bottom_roundness	float	-1.0	1.0	0.0
Eye Rotation	eye_rotation	float	-1.0	1.0	0.0
Ear Height	ear_height	float	0.0	2.0	0.0
Ear Lobe Length	ear_lobe_length	float	0.0	1.0	0.0
Ear Depth	ear_depth	float	-1.0	2.0	0.0
Ear Rotation	ear_rotation	float	-1.0	1.0	0.0
Mouth Protrude	mouth_protrude	float	-1.0	1.0	0.0
Mouth Width	mouth_width	float	-1.0	1.0	0.0

Table 4.1: Parameters and corresponding variables for the Parametric Heads

The Head Type parameter defined as an integer variable loads a head definition file from the existing baseline heads. These can range from 0 to 100, currently only two head definition files exist denoted as Head Type: 1 (Male Head) and Head Type: 2 (Female Head). Further baseline heads can be created and added to the database of head definition files to increase the choice of heads to work with.

Deformation X Pull, Y Pull, and Z Pull control the amount of influence in each direction

The Scaling group performs a non-uniform scaling in each direction. The Skew controls cause the scaling to vary as a function of position.

Head Width, Head Depth, and Head Height scale the head in each direction i.e. (x,y,z respectively – world coordinate system).

Head Width Skew1, Width Skew2, Height Skew, and Depth Skew varies the amount of skew (see Figures 4.22 and 4.23).

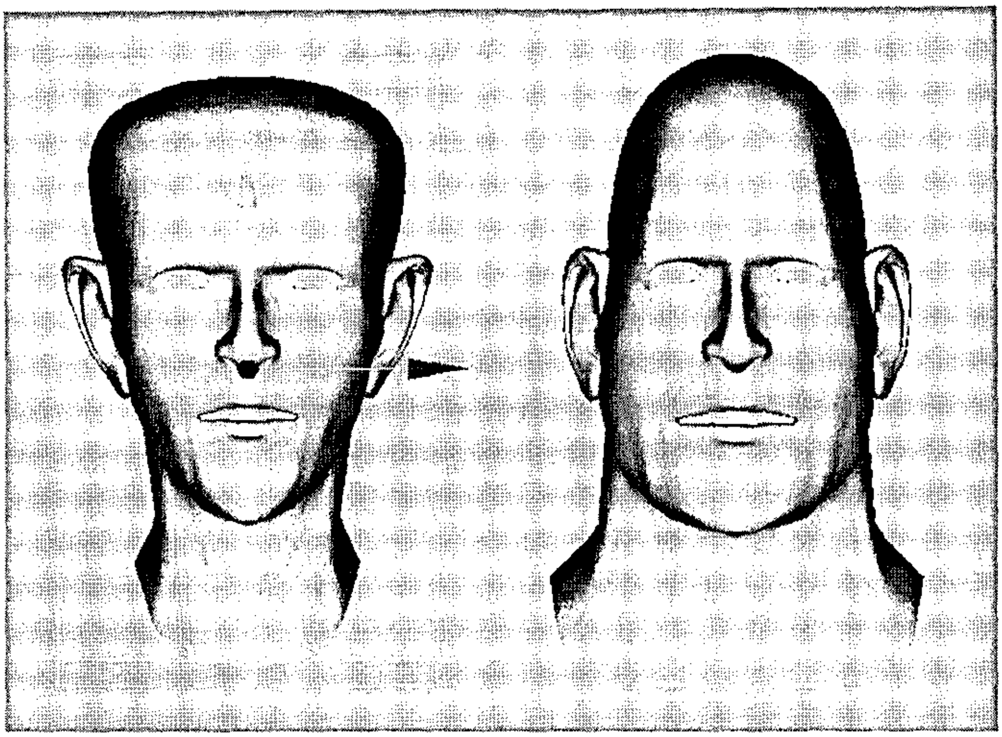


Figure 4.22 Width Skew1 Positive and Negative

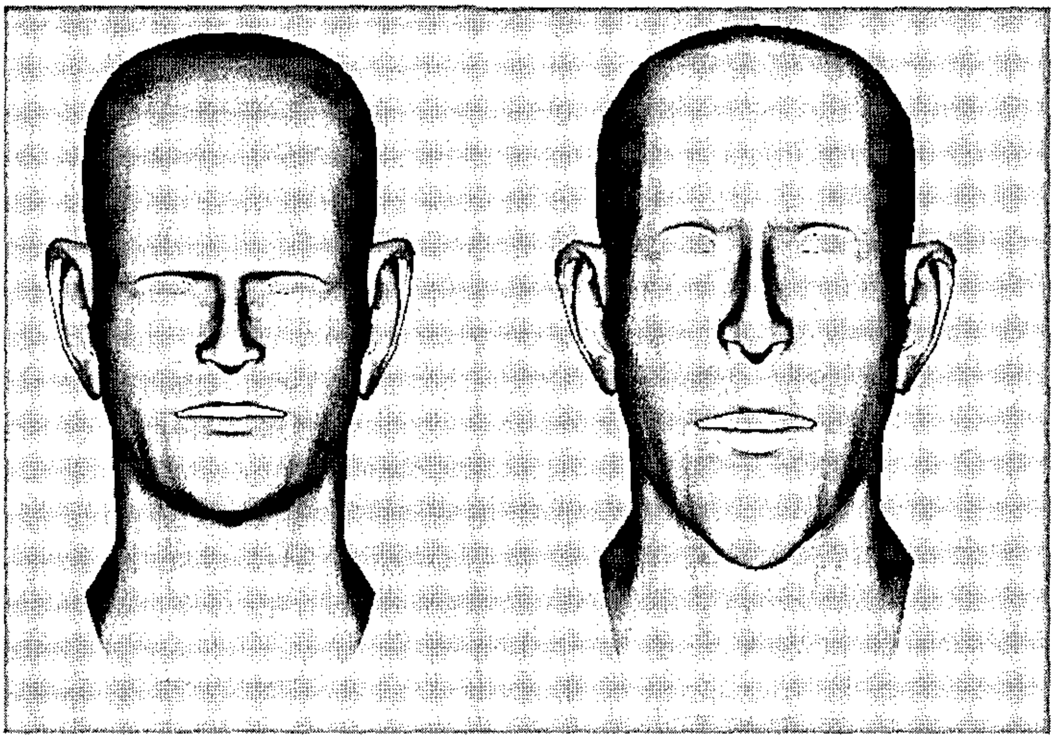


Figure 4.23 Height Skew Positive and Negative

Compress Face parameter squashes the face inwards with values less than 1.0 whereas values greater than 1.0 pull the face outwards. Forehead Slope as the name suggests controls the slope of the head. Values greater than 0.0 slope the forehead back, while values less than 0.0 slope it forward (see figure 4.24).

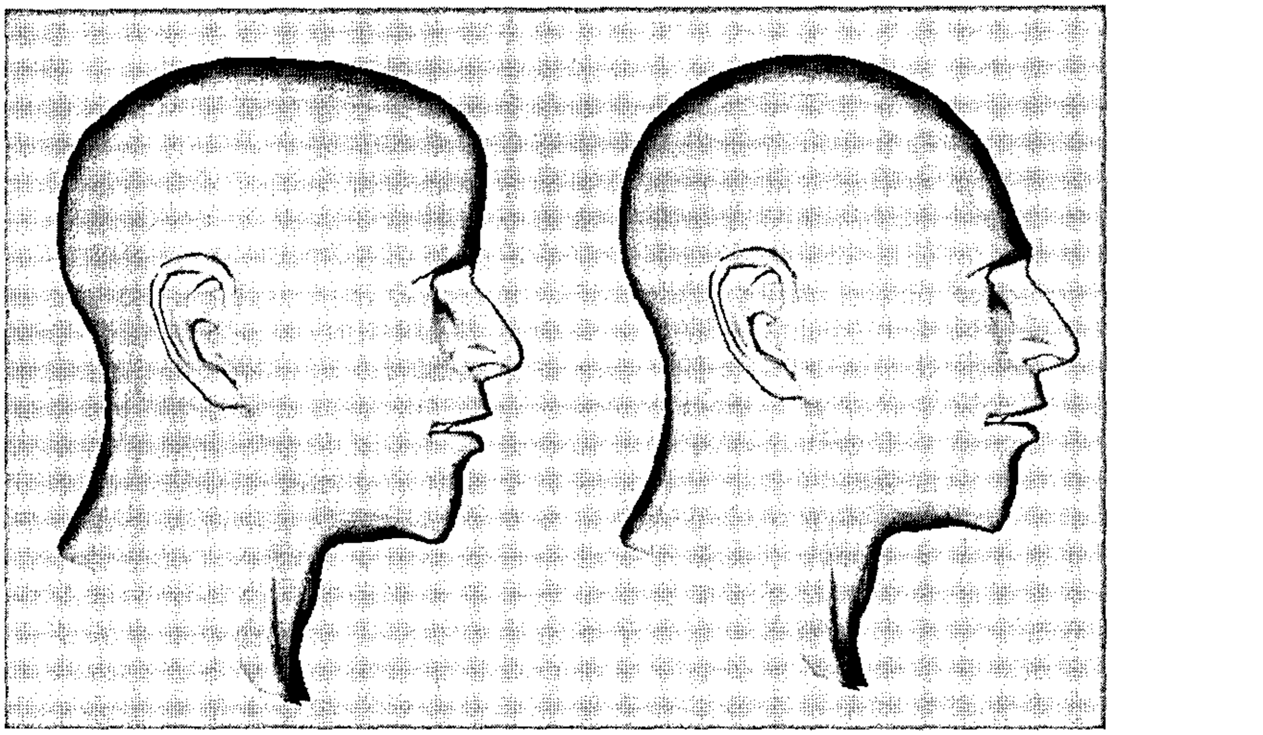


Figure 4.24 Forehead Slope Negative and Positive

The Nose Width variable controls the width of the nose and affects the area around it. Nose Width values greater than 1.0 widen the nose, whereas values less than 1.0 make it narrow (see Figure 4.25). Nose Length controls the length of the nose with a value greater than 1.0 stretches the nose outwards and values less than 1.0 squash it inwards. Nose Bridge changes the slope of the Nose Bridge (see Figure 4.26).

Nose Pull Up pulls the nose upwards by compressing it from the bottom when the pullup is greater than 1.0 and lengthens the nose vertically if the Pull Up is less than 1.0. The Nose Bridge parameter changes the slope of the nose bridge. Nose Hook/Pug amount hooks the nose downwards for values greater than 0.0, values less than 0.0 twist it upwards to form a pug nose (see Figure 4.27).

Chin Extend pulls the chin in or out. Values greater than 1.0 pull the chin out. Values between 0 and 1.0 push it inwards (Figure 4.28). Chin Tilt produces a rotation of the chin and Chin Tilt Amount influences how much effect there is at the end of the chin. Chin Accent sharpens the chin for values greater than 0.0 and widens the chin for values less than 0.0

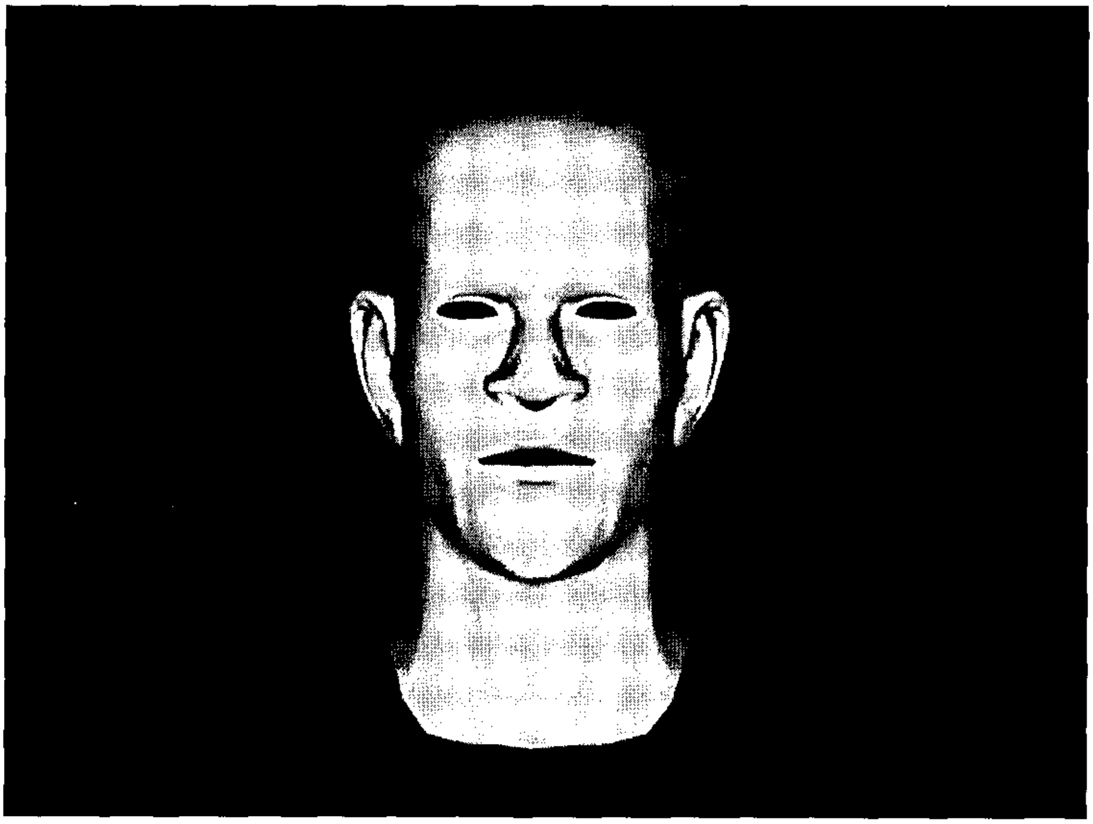


Figure 4.25 Nose Width Increased nearing upper limit – Rendered Image

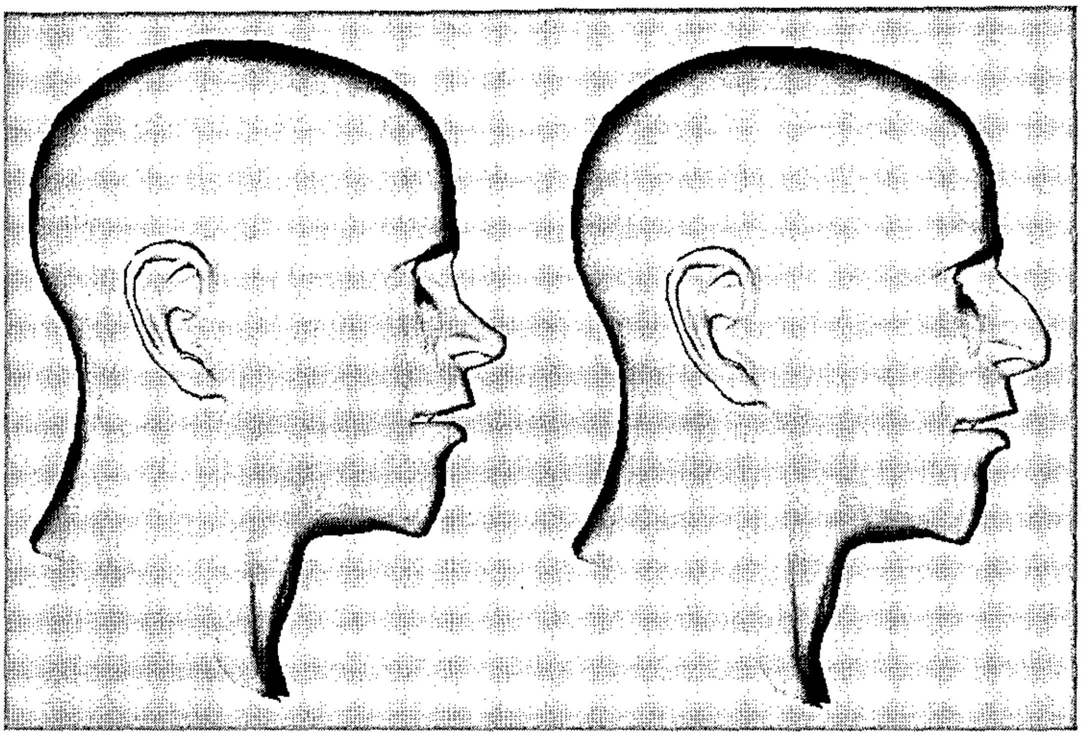


Figure 4.26 Nose Bridge Negative and Positive

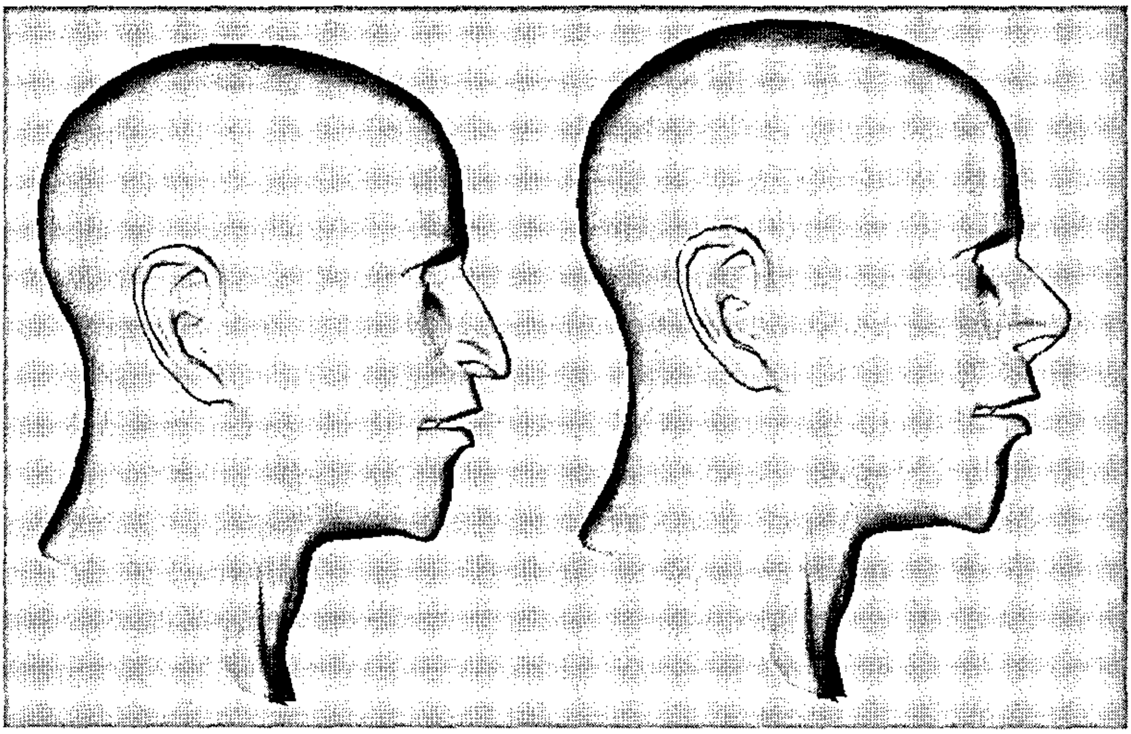


Figure 4.27 Left – Nose Hook/Pug greater than 0.0, Right – Nose Hook less than 0.0

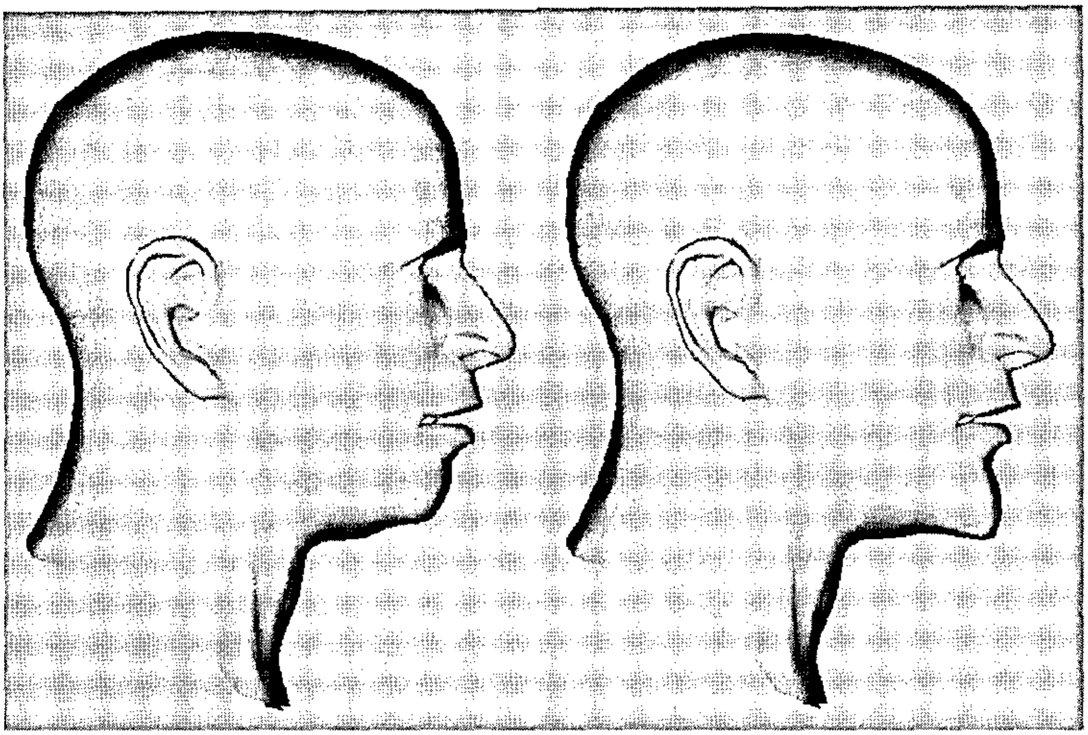


Figure 4.28 Left - chin extend set at 0.0 (pushed inwards), Right – chin extend set at 1.5 (pulled outwards)

Jaw Width widens the jaw with values greater than 0.0 widens the jaw up to the maximum limit of 1.0 (see Figure 4.29). The Jaw Width Uniformity control extends from the back of the jaw to the chin. This control determines where the widening occurs most. If the Uniformity is greater than 0.0, then the width influences the area towards the chin more. If the Uniformity is less than 0.0, then the influence is more towards the back of the jaw.

Cheek Bones Extrude extrudes the cheekbones outwards or pushes them inwards. Values greater than 0.0 pulls them out, values less than 0.0 pushes them in. CheekBones Z-Pos moves the cheekbones up and down (z direction translation). The Cheek Curvature parameter pulls the cheek inwards for values greater than 0.0 and puffs them outwards for values less than 0.0 (see Figure 4.30 and 4.31). Cheek Curvature Z Falloff controls the outward curvature. If the falloff is 0.0, then the cheek will be puffed out substantially all along the vertical direction. If the falloff is 1.0, then the falloff in the vertical direction is sharp, and the puffiness is more localized vertically. Cheek Curvature Y Falloff also controls the outward curvature. If the falloff is 0.0, then the cheek will be puffed out substantially all along the Y direction (going from the mouth to the ears). If the falloff is 1.0, then the falloff in the Y direction is sharp, and the puffiness is more localized.

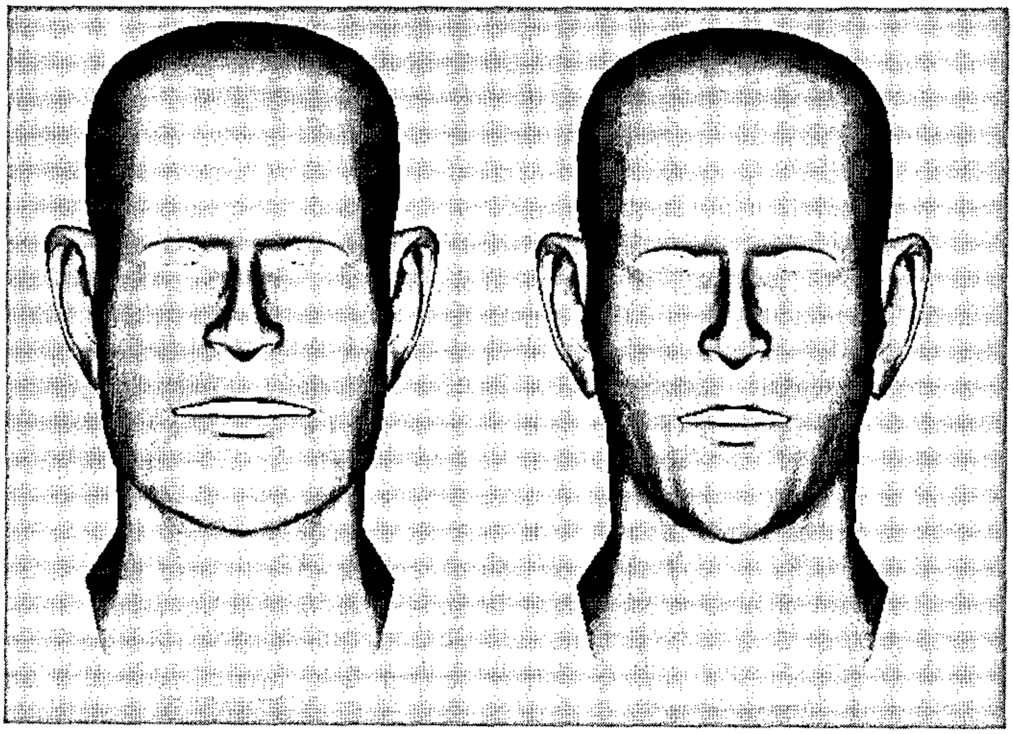


Figure 4.29 Left - Jaw Width set to 1.0, Right – Jaw Width set below 0.0

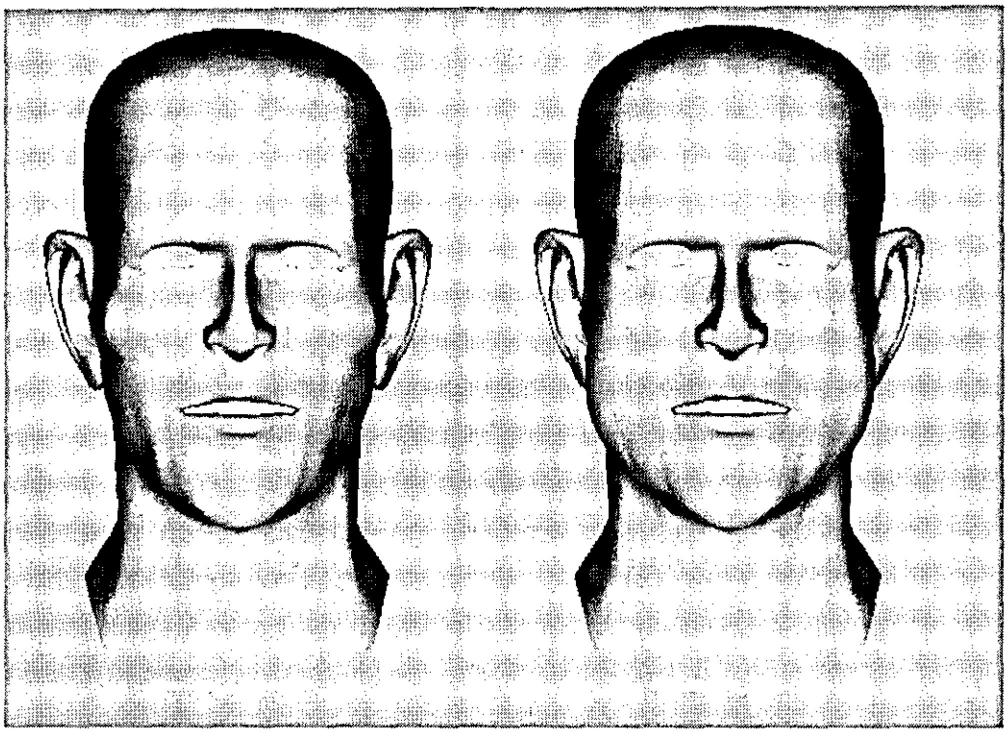


Figure 4.30 Left – Cheek Bones extruded, Right – Cheek Bones not extruded and Curvature less than 0.0

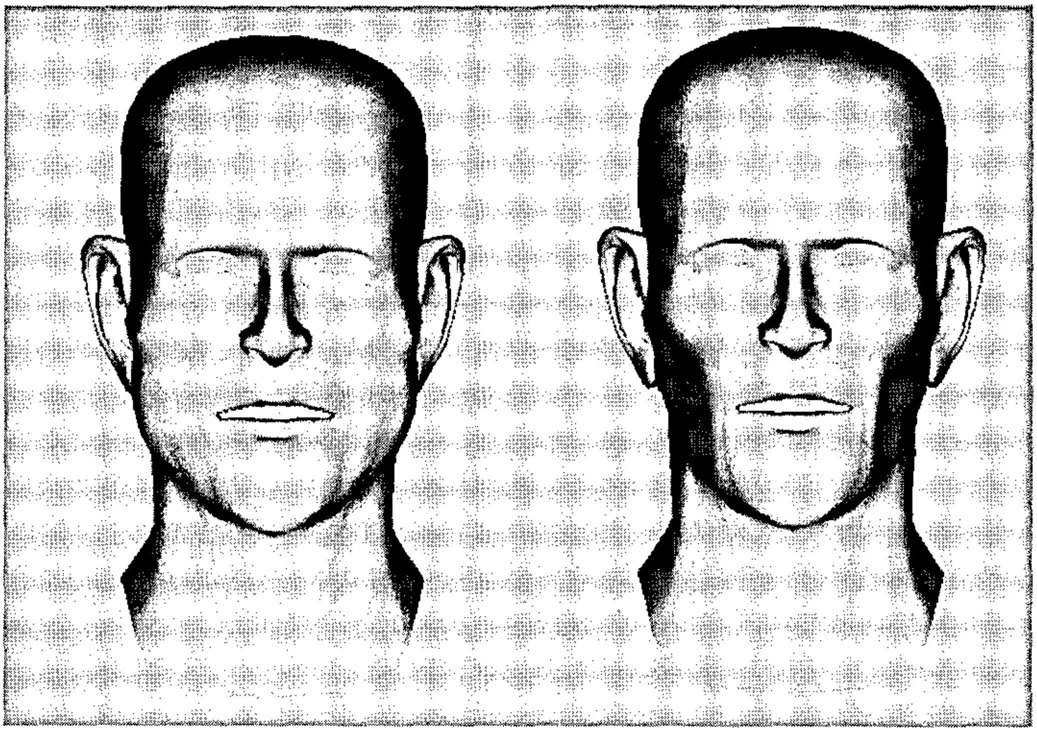


Figure 4.31 Left – Cheek Bones extruded and Curvature less than 0.0, Right – Cheek Bones extruded and Curvature greater than 0.0

Eyes Separation controls the distance between the eyes with values greater than 1.0 pull the eyes apart, values between 0.0 and 1.0 push them closer (see Figure 4.32). Eyes Top Roundness and Eyes Bottom Roundness parameters control the roundness of the top and bottom half of the eye respectively. Values greater than 0.0 make the eye sockets more round. Values less than 0.0 make the eye sockets more squinted. Eyes Rotation parameter rotates the eyes inwards and outwards (Figure 4.33).

Ear Height increases the ear height, ear depth increases the ear depth (grows in the backwards direction). Ear Rotation rotates the ear around the joint to the head where negative values rotate the ear towards the head and positive values rotate the ears away from the head. Lobe Length lengthens the ear downwards (see Figure 4.34).

Mouth Protrude controls the amount by which the mouth extends out from the face. Values less than 1.0 pull the mouth inward whereas values greater than 1.0 push it out (see Figure 4.35). Mouth width controls the width of the mouth with values greater than 0.0 widen the mouth and values less than 0.0 shrink it (see Figure 4.36).

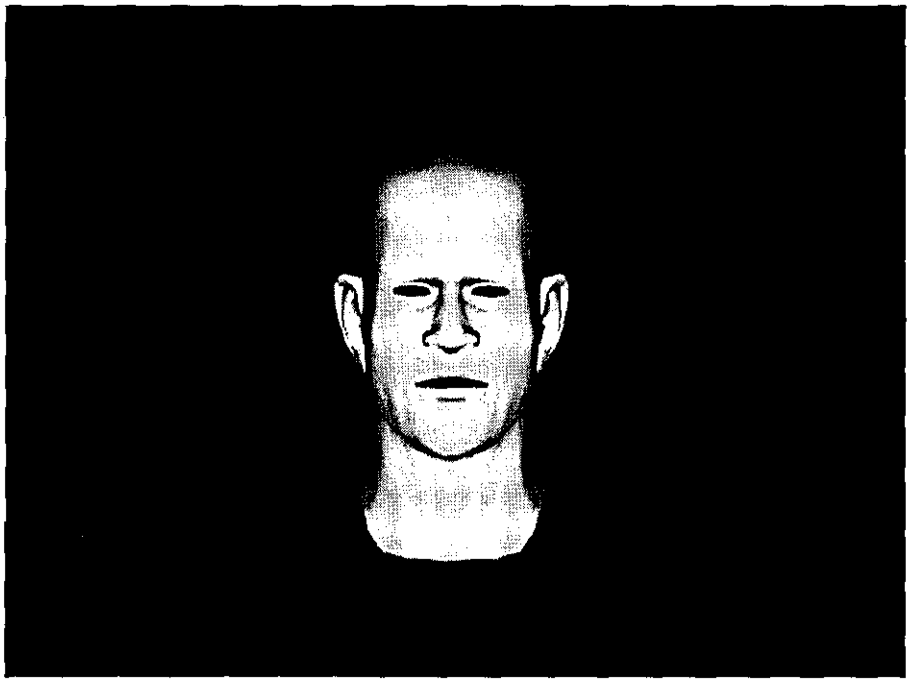


Figure 4.32 Eye Separation set at 1.0 making eyes closely set – Rendered Image

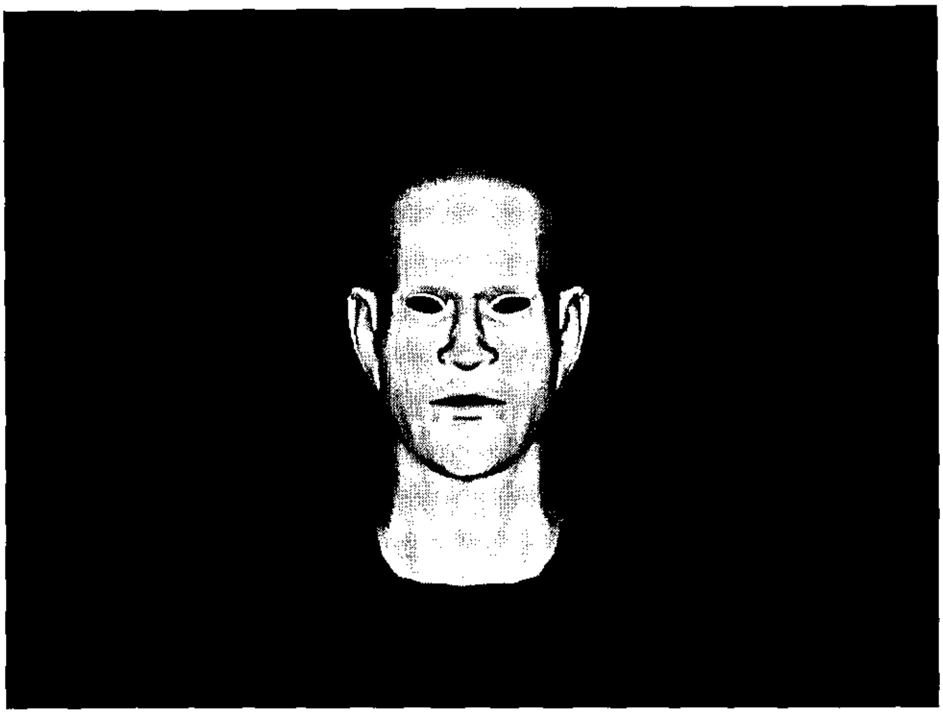


Figure 4.33 Eyes Rotation and Top Roundness set greater than 0.0 – Rendered Image

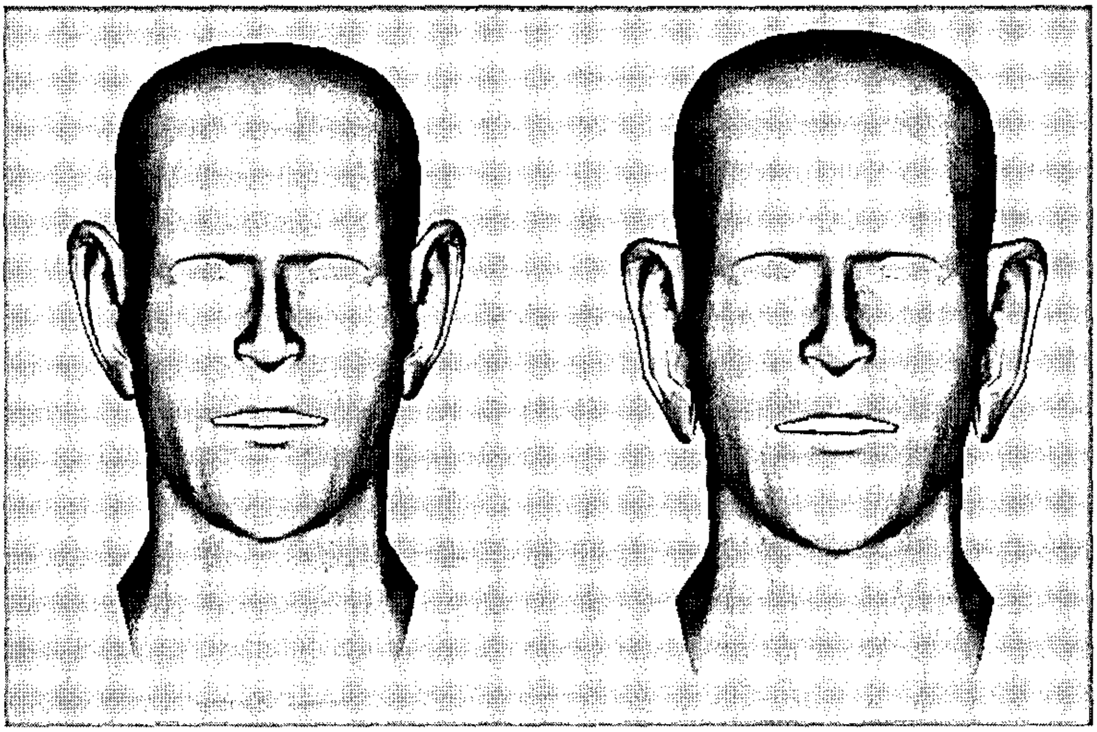


Figure 4.34 Left – Ears Height greater than 0.0, Right – Ears Lobe Length greater than 0.0

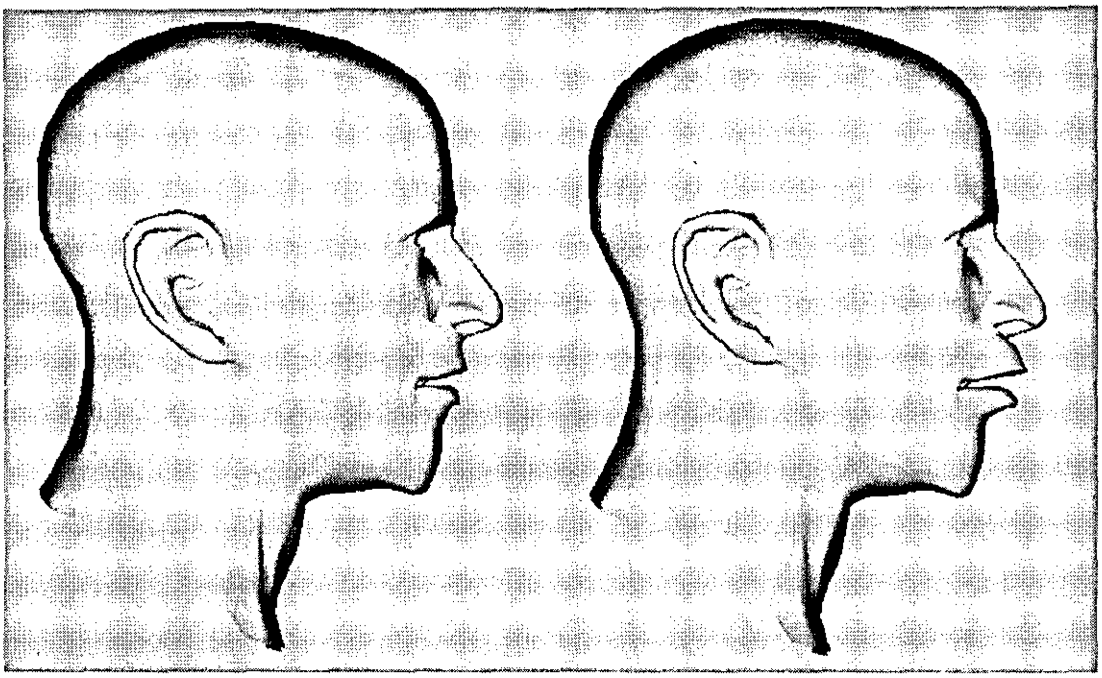


Figure 4.35 Left – Mouth Protrude between 0.0 and 1.0, Left – Mouth Protrude between 1.0 and 2.0

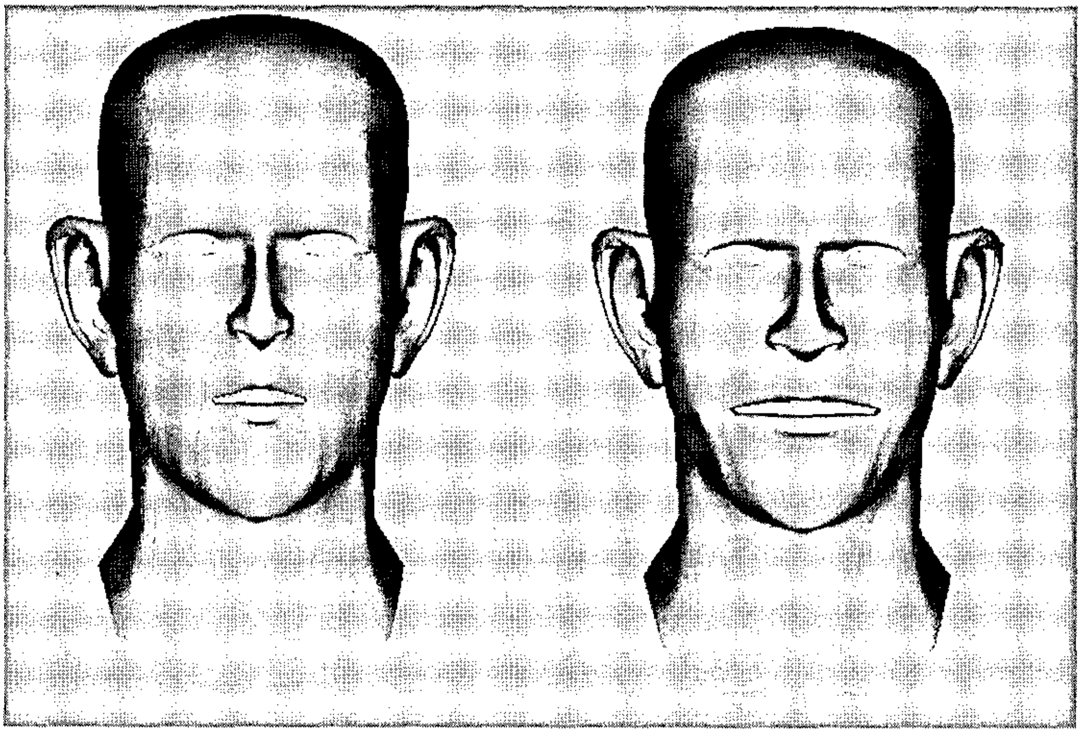


Figure 4.36 Left – Mouth Width less than 0.0, Right – Mouth Width greater than 0.0

4.6.1 Parameterisation and Facial Image Generation Script.

The Head Generator Script (HGS) edits the parameters of the 3D head geometry by assigning floating point values listed in the *Heads Parameter File* generated by the NLP. These parameters are passed as variables of the deformation control modifiers to affect changes to the geometry of the head in the manner set in the support header file for each deformation control modifier. The parameters need only be assigned to the correct variables in the script; the header file handles the arduous task of ensuring the parameters assigned to the variables edit the correct modifier control point/s by the amount specified in the variables. Figure 4.37 gives an overview of the processes involved in the Facial Image Generation Module.

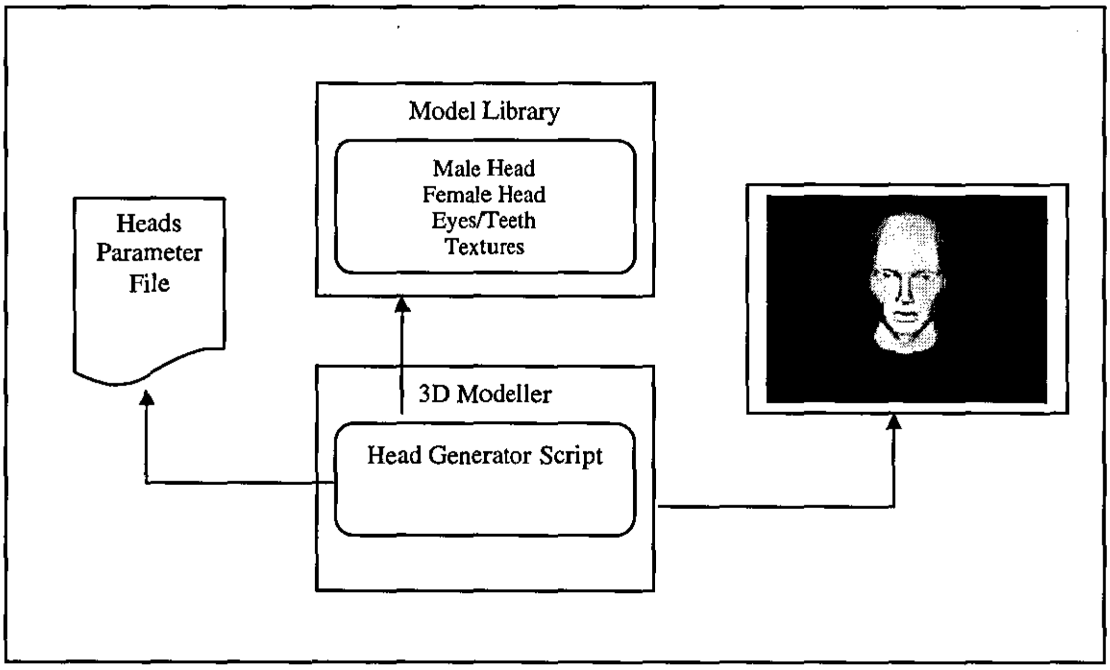


Figure 4.37 Flowchart of processes involved in Facial Image Generation Module

Let us take the example of the nose and how the parameters applied to Nose_Modifier can control the structure of the nose. As mentioned earlier in section 4.5 "Applying FFD Modifiers to Baseline Heads", the 3x3x3 matrix FFD applied to the nose has a total of 27 control points C_0 - C_{26}. Each of these control points can be controlled selectively or in control points C_0 - C_{26}. Each of these control points can be controlled selectively or in groups to perform transformation of the underlying geometry. In order to increase or decrease the width of the nose two distinct control points (C_1 and C_7) need to be modified. Figures 4.38 identifies these control points along with other control points on the FFD structure.

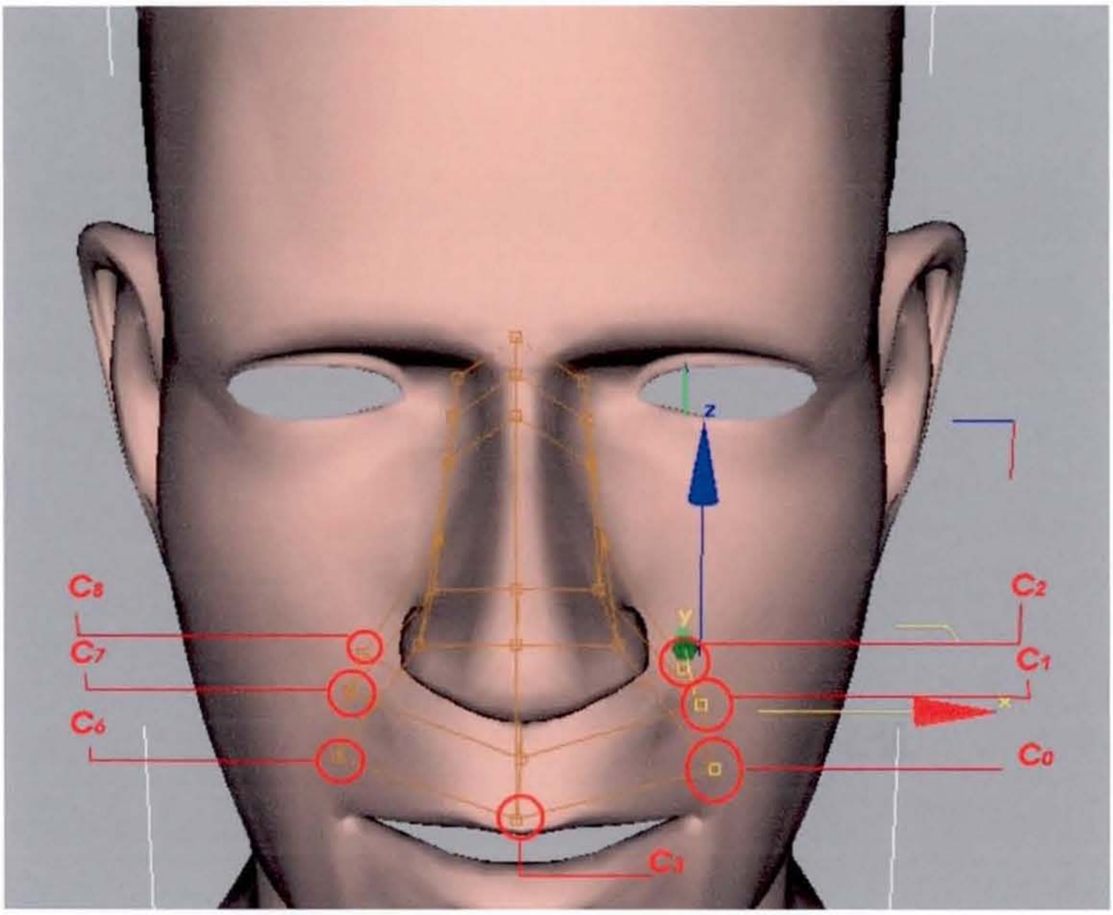


Figure 4.38 Nose_Modifier with control points C_0 - C_8 outlined by red circles

In order for the nose width parameter to be edited the nose_width variable is computed and passed to the Nose_Modifier FFD for changes to take effect. The following pseudo code provides an explanation of how the script code works.

```
head_model.nose_width = [ P]   -- nose width is set to value P assigned by NLP
head_model.nose_length = [Q]   -- nose length is set to value Q assigned by NLP
```

```
nose_width()            -- nose_width function called by FIGS
    (
        Param_Range = #(0,0.1,0.2,.......................,2.0)
  -- array of parametric  values, range pre-defined (see Table II)
        x_coord, y_coord, z_coord = 0.0
  -- initialize x, y, z coordinate variable

        coordinate_location = findItem Param_Range [P]

-- Does a '==' comparison between elements in the array Param_Range and the target
value P and then returns the index of the first occurrence of the given value in the array
or zero if the value is not in the array.

        CPtransform_corordinate = Param_Range[coordinate_location]

-- Find the transformation co-ordinate. So if P was 1.60 then CPtransform_coordinate
will be 16 points. However since the nose width must be increase or decreased
uniformly in both directions, the amount by which the control points must be moved is
half in each direction.

        coordinate = CPtransform_coordinate/2

        x_coord = coordinate

        Nose_Modifier.deformType = 1

-- Integer  default: 0  deformType = 0 - Only In Volume; 1 - All Vertices

        Nose_Modifier.lattice_transform  SubAnim

-- Enable lattice and it's control points to be transformed and animated

        animateVertex Nose_Modifier.control_point_1.position
[x_coord,y_coord,z_coord]

        animateVertex Nose_Modifier.control_point_7.position [-(coordinate),0.0,0.0]

-- Applies transformation to the specified control points of the FFD modifier
'Nose_Modifier', here the control point specified is transformed by positioning or moving
the control point in the x-axis direction by amount specified in variable 'coordinate'.
```

A similar process is repeated for the parameter/variable nose length with the difference that the Param_Range is set between 0.0 – 3.0 and only one control point is edited, C_3. Figure 4.39 shows the control points transformed by the script to affect the nose. Note the *visible* Nose_Modifier matrix over the nose (which is otherwise hidden) to show the new position of control points after nose width and length is increased.

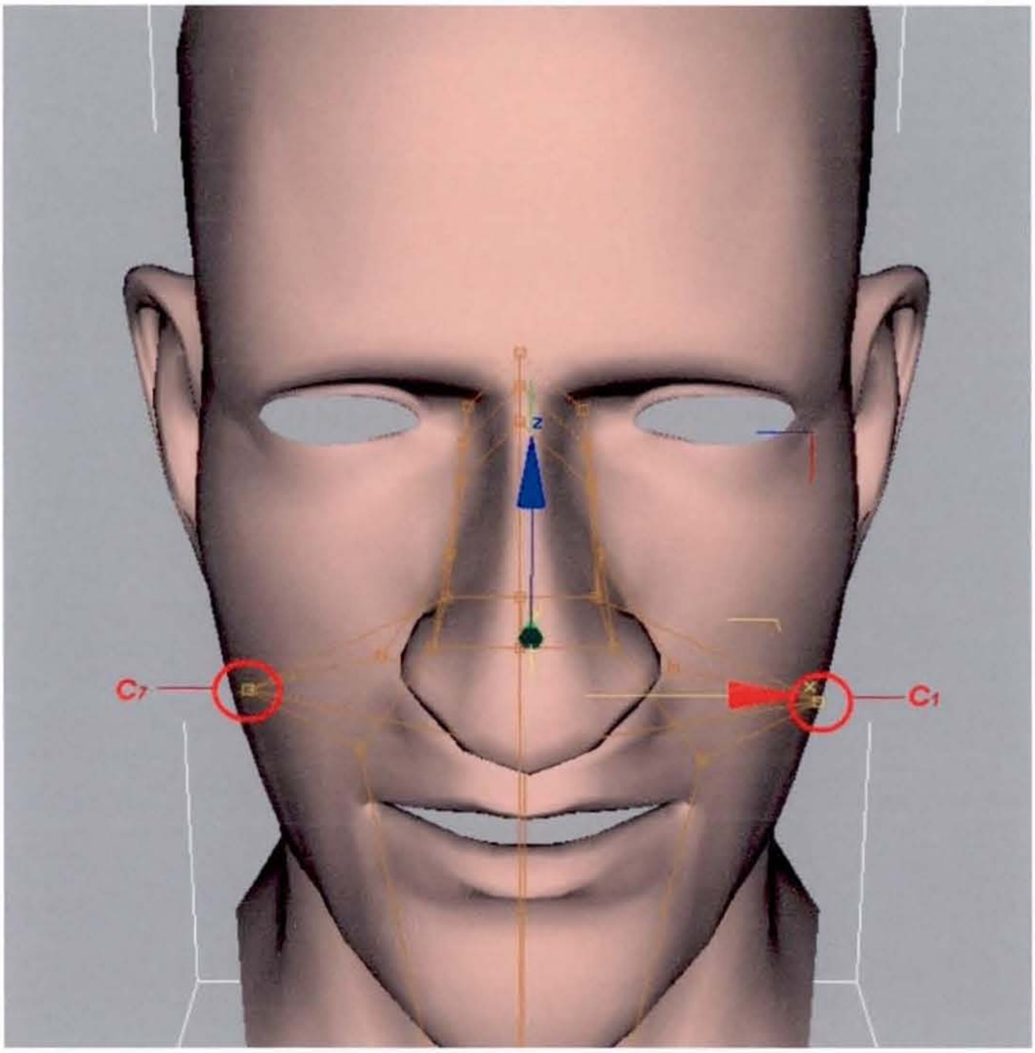


Figure 4.39 Shows new position of control points C_1 and C_7 in the x direction to increase nose width and C_3 extended in the z direction beyond view to increase nose length.

4.7 Testing the Facial Image Generation Module

The facial image generation module was tested by scripting a routine that assigned random values to parameters of the baseline head. The random values were allocated by a random number generator within the range specified in Table 4.1 for each parameter. This simple test labelled 'Crowd Generator' provided a comprehensive method for evaluating the different combinations and variations of heads and features that could be produced by the facial image generation system. Figure 4.40 and 4.42 show a set of 12 heads produced by the 'Crowd Generator' script. Figure 4.40 shows the various combinations of heads and features possible by the FIG module using the male baseline head (Figure 4.41/4.13) and Figure 4.42 shows the heads produced using the female baseline head (Figure 4.43/4.12)

Careful observation of the heads generated by the Crowd Generator script reveals the flexibility and capacity of the FIG module to produce vast variations in the head and features. The heads produced by the FIG module may not be photorealistic and this can be attributed to a number of factors such as quality of textures, lighting and shading and rendering configurations but a more noticeable factor is the absence of accessories like hair that considerably lowers recognition detail. The system can be configured to display more realistic heads by working on the rendering configurations and improving the textures, shading and lighting details but that usually requires time and experience both as an artist and user of the modelling application. Besides aesthetical improvements to the model is beyond the scope and aim of this thesis. We aim to show that geometric models of human faces can be controlled and defined by natural language instruction and that is what chapters 6 and 7 hope to demonstrate. The system as it stands is not adequate as an ID-Kit or E-fit system however it could be used to produce heads suited to applications like entertainment or character creation such aliens, demons and comic characters.

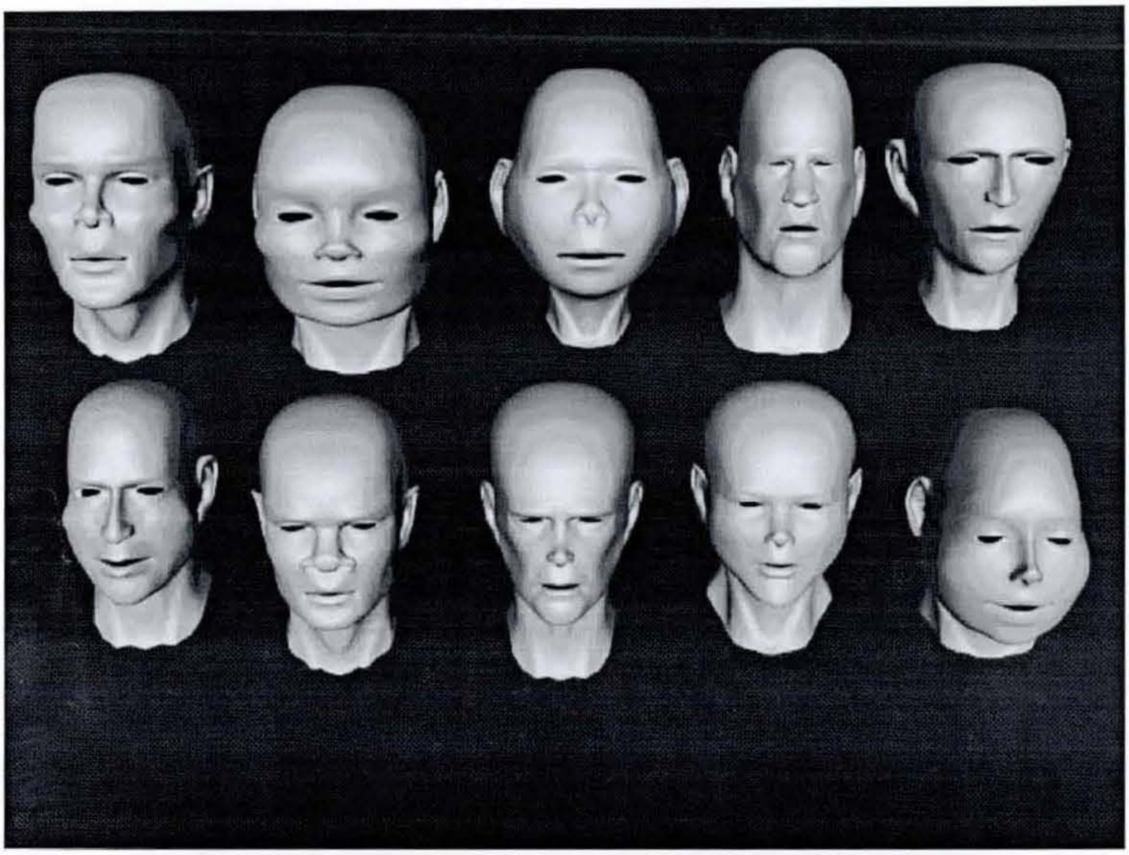


Figure 4.40 Heads produced by the Crowd Generator script using the male baseline head.

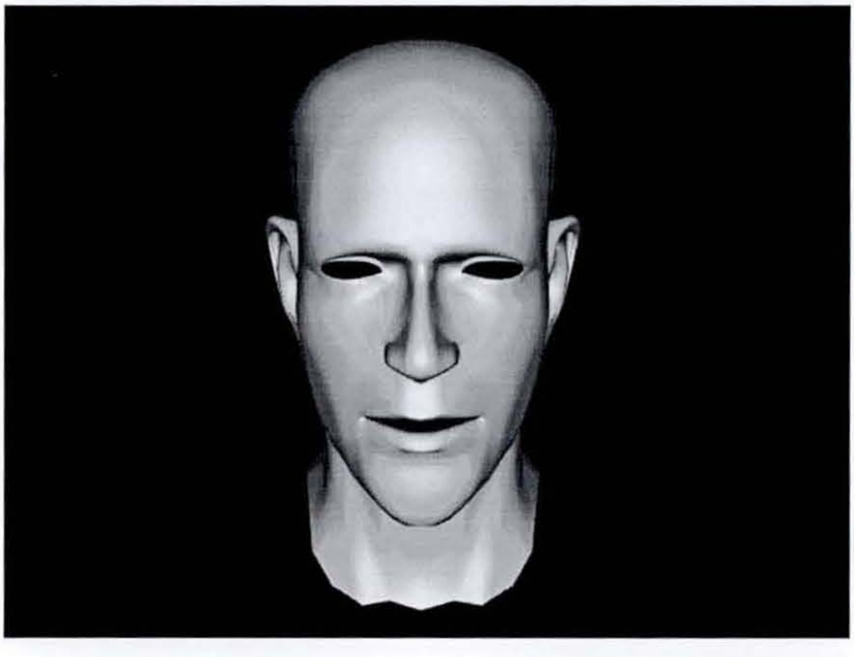


Figure 4.41 Male baseline head model

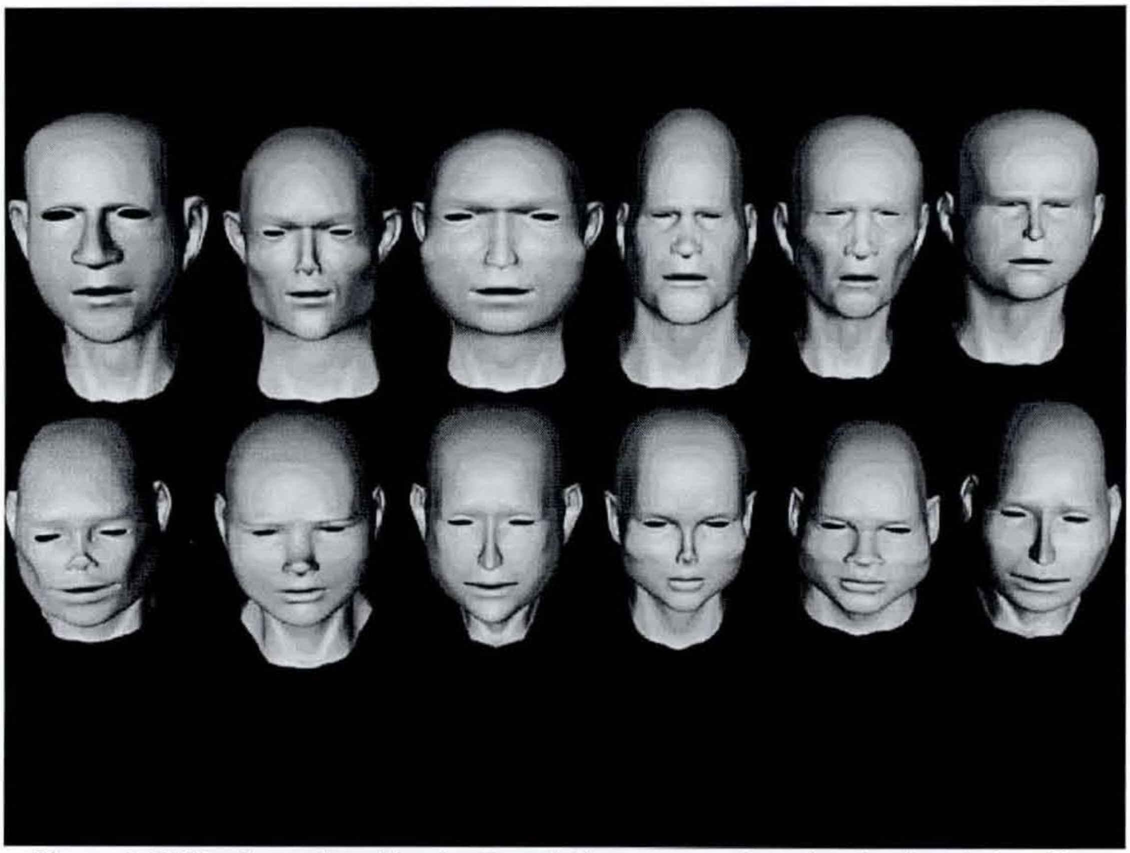


Figure 4.42 Heads produced by the Crowd Generator script using the female baseline head

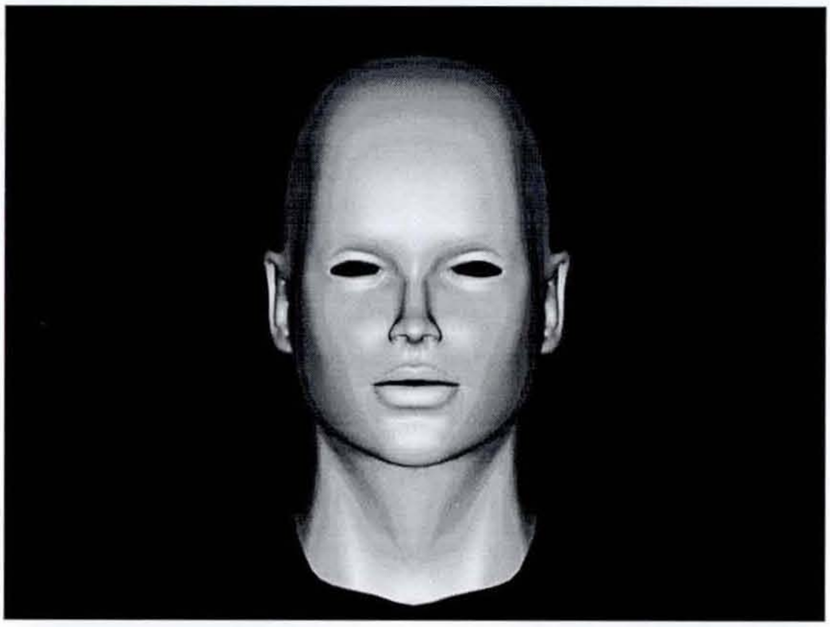


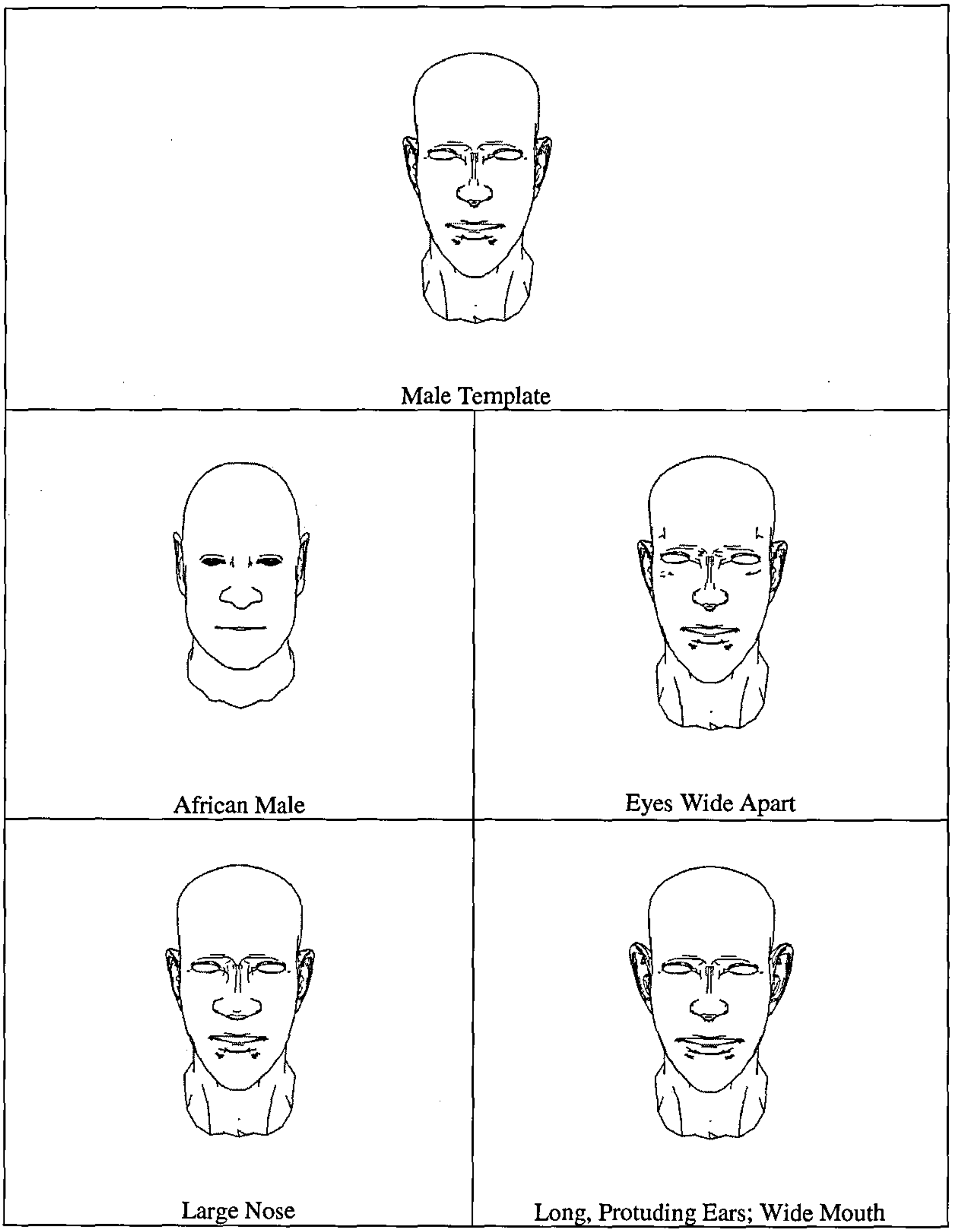
Figure 4.43 Female baseline head model

4.8 Deriving Modifier Parameters from Template Head Parameters

Currently the database of templates contains two entries; male template (Figure 4.41) and female template (Figure 4.43) representing geometry data of the two baseline heads constructed and described in the earlier sections. These files hold the default parameters for the baseline head model. Modifiers are sets of parameters that affect the head geometry when applied to the baseline head.

Modifier parameters are calculated by differencing the parameters of a modified head from the baseline head. A simple code routine was developed in Visual Basic called "head comparator" (see Appendix B) that compared geometric data of modified heads with the template head and calculated the difference. The difference calculated is saved in a new file as set of parameters in a library of modifier files. The library consists of a comprehensive collection of files each one referring to a specific qualifier or description such as; "wide", "long", "big", "fat", etc.

Prior to calculating modifiers, 3D head geometry had to be edited and modified to represent a target description. For example to represent a head of African origin, the baseline heads had to edited inside the 3D modeller application until the feature set of both the male and female baseline heads resembled an African. Figure 4.44 shows the modified African head derived by manually modifying the male template. The images in Figure 4.44 show an outline illustration without highlights or textures to amplify the shape and structure of features on the face. The most obvious differences noticeable are in the size and shape of the head, nose, ears and mouth. Similarly other modified heads were created to experiment with other descriptors like wide - nose, mouth, jaw; long - nose, ears, chin, head; wide apart - eyes; fat – cheek, nose, head etc. Figure 4.44 also shows modified heads representing faces with "large nose", "eyes wide apart", "long protruding ears", "wide mouth" derived from the male template head by manually editing the 3D head model in 3D Studio Max. The modified 3D head files are examined by the comparator and modifier parameters are calculated and written to file for the heads engine to use.



Male Template

African Male

Eyes Wide Apart

Large Nose

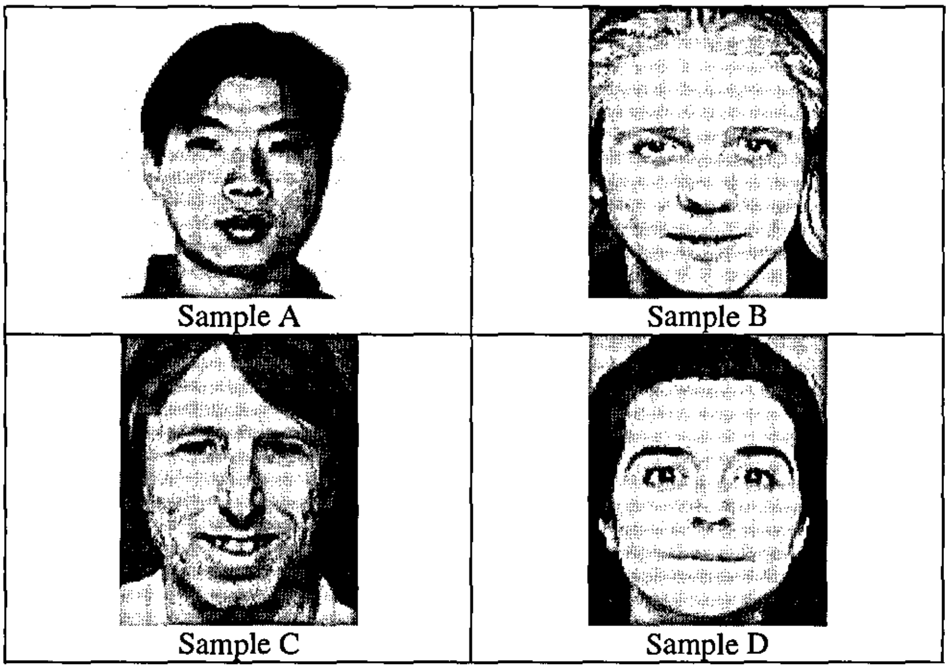
Long, Protuding Ears; Wide Mouth

Figure 4.44 Modified heads derived by editing the male baseline head/male template

This experiment also looked at biometric data from 2D images of real people. The measurements stipulated parameters correlating descriptions, provided from survey results (see chapter 2), to modifier parameters.

4.8.1 Extract biometric data from facial image data set

A small sample of 4 images was selected from the AT&T database of facial images. All images were front poses with minimal tilt and turn to ensure feature measurement is consistent. Table 4.2 shows the sample of images selected, all images were normalised to have the same dimension and resolution. This was necessary for the image measurement software to calculate head and feature dimensions coherently and accurately.



Sample A

Sample B

Sample C

Sample D

Table 4.2 Sample of 4 photo realistic facial images used as target images for reconstruction

Numerous studies have been carried out in the past for measuring facial features (Bisson, 1965a; a965b; Sakai *et al*, 1972; Bromley, 1977; Batten & Rhodes, 1978). Most of these have been in the area of facial recognition and is based on the assumption that certain points in a facial image can be located with accuracy. Once located, the positions of each point can be recorded in a suitable coordinate system. Points typically located during measurements are the corners and pupils of the eyes; the rightmost, leftmost, and lowest points on the nose; the corners, highest, and lowest points on the sides of the face. Some of these points, such as those on the sides of the face, are difficult to locate with reproducible accuracy.

Some of the earliest work in measurement of facial features was done by Bisson (1965a; 1965b). The first of these reports, which describes efforts to determine the outside corners of the eyes, illustrates both the methods and the difficulties in such image processing. Processing was done by locating the front, bottom and sides of the iris; points along the upper and lower eyelids were found; parabolas were fitted to the eyelid lines; and the intersection s of these parabolas were found and used as corners of the eyes. This approach generally requires some initial estimates about the size and positions of the components to be determined. Such estimates are usually easy to make when dealing with facial images.

The first successful automatic measuring algorithm appeared to be that of Sakai *et al.* (1972). Line images were produced by thresholding the "9 × 9 Laplacian" of the image. This simple technique produces very good line images. Facial features are then located using a signature technique with *R(y)* equal to number of dark pixels across the strip. Features are located in the following order: top of the head; sides of the face; nose, mouth, and chin; then the chin contour. Once these have been determined, some refinements are made, and the positions and dimensions of various features are determined.

Bromley (1977) developed a similar feature-measuring algorithm. It is based on a line detection scheme using an optimum filter for detecting edges in images. This filter happens to be a cascade of the Laplacian operator with a low-pass filter, so it is related to the one used by Sakai *et al.* (1972). The order of processing the features is different from that of Sakai *et al.* First the left and right sides of the face and the face centre line are determined. The signature (similar to that of Sakai *et al.*) along the centre line is used to find the top of the head, the hairline, the mouth position, and the chin-line. Eyebrow and eye positions are located using signatures along lines positioned to the left and right of the centre line. The algorithm locates the tip of the nose and the end points of the mouth, and then determines the facial outline by searching outward in various regions of the face.

Batten and Rhodes (1978) describe a man-machine system used to obtain measurements from several thousand images. The system comprises a computer-controlled projector, a digitizing tablet, and a mini computer with sufficient disk storage to save the measurements. Images placed on top of digital tablets are measured by the coordinates of the stylus placed in the image area. The coordinates, in digital form, are transmitted to the computer for processing.

A simpler approach to facial measurements is coding facial features in terms of distances, angles, areas and other mathematical functions. The basic elements for geometric information are coordinates of points. For example, the feature "length of nose", is the distance between "top of nose" and "bottom of nose". Most existing systems use trained people to locate these points.

A system which uses geometric coding usually combines basic measurements into features which summarize information about the images. Two of the early facial pattern recognition studies, Bledsoe (1964, 1966) and Kaya and Kobayashi (1972), used geometric coding of features; the latter used 10 distances to 9 features, each feature being a distance divided by a referent, the nose length. Townes (1976) used a similar set of distances shown in Figure 4.45. Instead of scaling each distance to a single referent such as nose length, he considered all possible ratios less than one and selected those which had the best correlation with his target image.

The facial measurements used by Townes have been used as a guide line for measuring facial features of our sample target images. Figure 4.46 shows the distances measured to compile the basic list of measurements needed to construct a frontal pose composite similar to the target face. Table 4.3 lists the measurements calculated for the 4 facial images shown in Table 4.2.

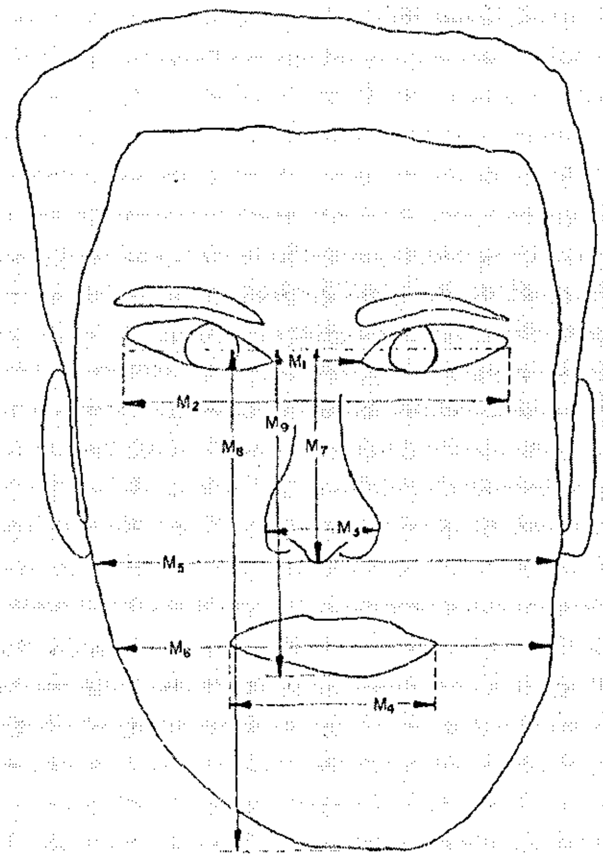


Figure 4.45 Facial Measurements used by Townes (Image scanned from Townes,1976)

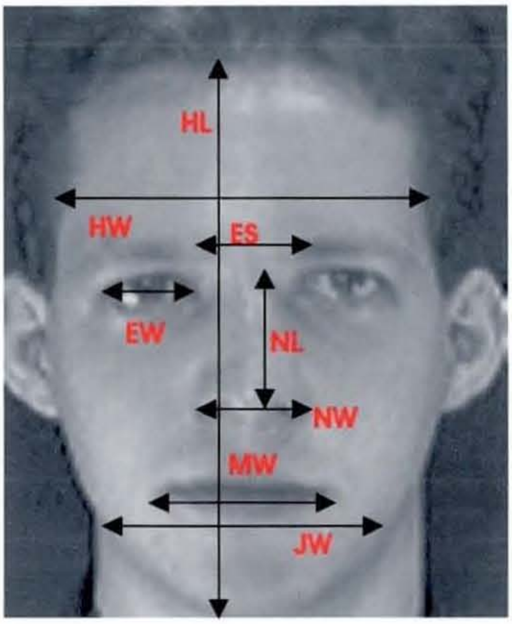


Figure 4.46 Facial Measurements used for Experiment

Features (cm) / Images	NL	NW	EW-L	EW-R	ES	MW	JW	HW	HL
A	2.68	2.60	1.60	1.68	2.44	3.26	7.86	8.90	11.40
B	1.92	1.74	1.27	1.32	1.37	2.23	4.22	5.95	7.17
C	2.40	1.44	1.33	1.47	1.12	2.32	3.00	4.93	6.80
D	1.58	1.43	1.26	1.37	1.36	2.33	4.52	5.66	6.70

Table 4.3 Measurements of the 4 Target Faces shown in Table 4.2

Key:

NL	=	Nose Length
NW	=	Nose Width
EW-L	=	Eye Width, Left Eye
EW-R	=	Eye Width, Right Eye
ES	=	Eye Spacing
MW	=	Mouth Width
JW	=	Jaw Width
HW	=	Head Width
HL	=	Head Length/Height

4.8.2 Mapping feature measurements on to 3D face parameters

The next stage of the experiment required mapping the measurements from Table 4.3 to parameters of the parameterised head models. This required some sort of mapping or fitting function to map the large range of real values onto the limited range real number values for the head parameters. One immediate solution was to experiment with the parameters of the 3D head until a near to accurate representation of the target image was achieved. Not only was this technique tedious but crude and inefficient. Another solution was to mathematically solve this problem. Sigmoid function was selected as a suitable and in many respects an efficient mathematical solution.

A sigmoid function is an S-shaped "squashing function" (see Figure 4.47) which maps a real value, which may be arbitrarily large in magnitude (positive or negative), to a real value which lies within some narrow range. The mathematical form for the particular sigmoid function used in this simulation, and commonly used in many other neural network simulations, is the following:

$$f(x) = \frac{1}{1 + e^{-ax}} \tag{4.1}$$

Where e^{-ax} is exponential **e** raised to the power **(-ax)**. The result of this sigmoid function lies in the range 0 to 1. In the neural computation literature, the sigmoid is sometimes also referred to as the **logistic function**.

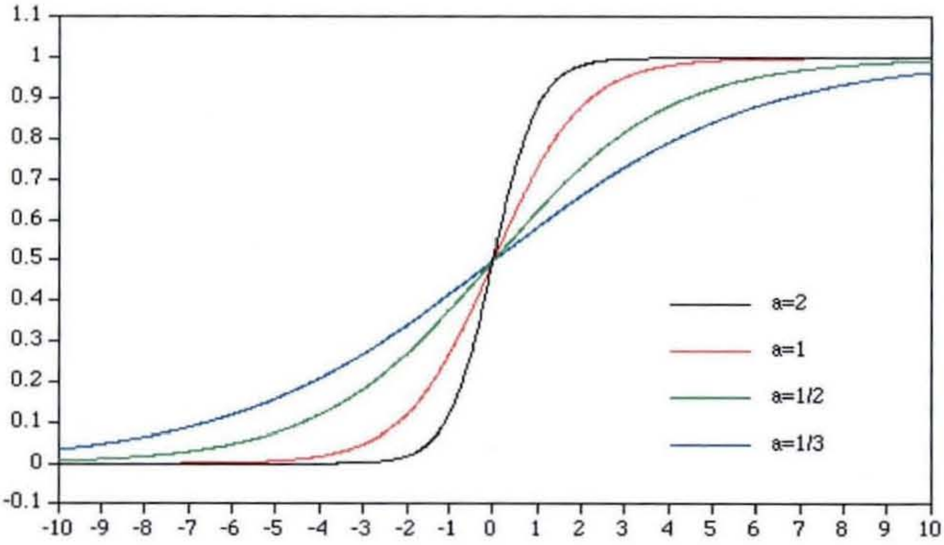


Figure 4.47 S Shaped Curves Produced by Sigmoid Function of varying values for **a**

4.9 Conclusion

The facial image generation module has been described in this chapter. It started by describing the modelling process using NURBS, Beziers and Polygons. Modelling a human head is not an easy task by any means, especially where artistic skills are lacking. As mentioned early on NURBS was a difficult tool to master and consequently the head sculpted using the technology was less than satisfactory. Polygonal modelling however was simpler and easier to work with especially with the vast array of tools and utilities built inside the modelling package for polygon creation and editing. The baseline heads constructed using polygon meshes formed the foundation for the facial image generation module. The baseline heads were parameterised using FFD modifiers attached to the head geometry. Each FFD modifier was catalogued and assigned variables acting as parameters that could be edited using Maxscript. FFD modifiers in 3D Studio Max allow a great deal of flexibility in the control of the deformation lattice. FFD control points are edited via MaxScript using transformation functions to affect the vertices of the underlying geometry consequently changing the structure of the confined area.

The chapter concludes with a test that assesses the efficiency and capability of the FIG module in its capacity to generate heads of different shapes, sizes and features. The heads generated by the FIG module revealed its flexibility and capacity to produce vast variations in the head and features. The heads generated by the facial image generation module, although not photorealistic, were good enough to interface the module to a natural language processing engine that would control and produce a head in par with the descriptions offered to the natural language interface.

Chapter 5

Dealing with Uncertainty – Theories and Techniques

Abstract

In this chapter we discuss the single major problem faced by designers and engineers of AI solutions – Uncertainty. We will discuss what uncertainty entails within the domain of Knowledge-Based systems and techniques available for handling uncertainty. Furthermore the chapter looks in detail the two main systems namely Fuzzy Logic and Truth Maintenance to deal with uncertainty in natural language descriptions.

Keywords: Uncertainty, Probability, Dempster Shaeffer Theorem, Fuzzy Logic, Mass Assignment, Semantic Unification, Truth Maintenance Systems.

5.1 Introduction

The problems with which Artificial Intelligence is concerned are inherently uncertain – it is the lack of certainty, the need to make sense of incoherent or incomplete information, which gives rise to the need for "intelligent" problem-solving behaviour. (Hinde, 1985; 1986).

In modelling the real world uncertainty abounds; it can be broken into two main categories: (1) uncertainty arising from lack of knowledge relating to concepts that in the sense of classical logic may be well defined and (2) uncertainty due to inherent vagueness in concepts. Uncertainty can manifest itself in the problem data, in facts and in rules (Fox, 1986). Kodratoff *et al.* (1988) describes these sources and types of uncertainty:

- Unreliability of data due to symbolic noise (vagueness or ambiguity in the meaning of a term) or uncertainty in the measure of an attribute

- Human induced errors: assigning wrong values to attributes, misclassifying examples or giving too many or too few descriptors
- Omission of necessary examples from a training set
- Noise in background knowledge
- Deficiencies in the description language used
- Uncertainty in the problem domain.

The uncertainty which is inherent in a system can be distinguished from the uncertainty introduced when modelling it using a particular representation system, which arises from vagueness in our perception and judgement of it. These distinct types of uncertainty may be best handled by different methods, numeric methods being more appropriate for the former, and symbolic methods for the latter. (Wise, 1986).

5.2 Approaches to Handling Uncertainty

Two different approaches can be adopted to model intelligent (human) behaviour: the understanding-oriented approach, aimed at duplicating the way in which humans operate, and the performance-oriented approach, aimed at producing the same results as a human would produce by whatever methods seems most effective. (Spiegelhalter, 1986). The various approaches which have been developed for dealing with uncertainty reflect this division as well as the differences between types of uncertainty which arise in different problem domains.

Humans often use vague, ill-defined terms when describing their reasoning processes; the difficulties involved in translating vague expressions into numeric terms without introducing an unjustifiable level of precision can be circumvented by using a symbolic approach, but also to reason with or about uncertainty (Fox, 1986; Hinde, 1986).

Expert systems often employ IF..Then rules obtained from human experts, with associated certainty factors which may show various forms of bias: people's estimates of probabilities tend to be influenced by such factors as the ease with which they can recall or imagine an event (which leads to bias towards specifics rather than generalities) and the degree of 'representative-ness' which an event appears to display. For example, if a coin is to be tossed six times, 'HHTHTH' will be judged a more probable outcome than 'HHHHHH'. If the biases can be recognized, it should be possible to remove or reduce their effects. The results obtained will then be more accurate, but less 'human'. The main advantage of using such rule-based systems is the ease with which their conclusions can be explained to the user.

Performance-oriented approaches are frequently based on probability theory or an extension, simplification or adaptation of it. Probability theory is the oldest and most widely used method of handling uncertainty, and is derived from a formal description of rational behaviour. Probabilities are a function of two things: the proposition under consideration, and the evidence at hand. Their precise magnitude is usually less important than the reasoning behind it, the context in which it applies and the sources of information which would cause it to change. Probability theory is unique in its ability to process context-sensitive beliefs, and it has been shown in (Perl, 1988) that for any reasonable scoring rule, any scalar measure of uncertainty is either worse than or equivalent to it. However, its use does present some problems: there may be insufficient data available to allow a full probability distribution to be specified accurately. With traditional probability theory ignorance cannot be distinguished from uncertainty and if approximations and simplifications have to be made the results obtained may not be accurate.

The need to express ignorance as opposed to uncertainty has led to the development of methods based on intervals. The range of probabilities which could be assigned to a hypothesis is given, with the lower limit of the interval based on the weight of the evidence supporting the hypothesis, and the upper limit calculated from the weight of evidence against it, or the support for its negation. The width of the interval represents the degree of ignorance, or lack of evidence.

There is a clear difference between the concept of probability and the concept of truth. A probability of 0.5 attached to a hypotheses does not means that it is half-true; hypothesis are either true or false, and probabilities can be regarded merely an estimate of the relative likelihood of these two alternatives. The idea of reasoning with truth rather than with probability – or with belief, as the truth or falsehood of a hypotheses will, in general, not be known – has led to the development of truth maintenance systems. Truth maintenance systems are used to establish sets of mutually consistent hypotheses and also to manage inconsistent hypotheses. Truth maintenance can be linked with uncertainty methods, the use of a preference ordering of assumptions will ensure that the 'most probable' solutions to a problem are explored first. (Hinde et al., 1989).

5.3 Numeric Methods

5.3.1 Probability Theory

For a long time probability theory was the only way of expressing uncertainty. Various schools of probability exist including subjective probability (based upon the view that probability is a logic of degrees of belief) and frequentist probability (based upon counting). Within probability theory a form of knowledge representation is used that allows uncertainties to be represented by numbers; the frame of discernment. Each attribute (variable) in the knowledge base is defined over a set of possible values (its universe of discourse). A probability distribution is associated over the set of possible values for any variable. This would say that one value is more likely than another. Various rules of inference exist within probability theory including Bayes Rule. Probability theory, while being an intuitive way of representing uncertainty, does not cater directly for other areas of incompleteness in knowledge representation such as ignorance and inconsistency. As a result the new fields such as belief theory (Dempster 1967; Shafer 1976), mass assignment theory (Baldwin 1991) and fuzzy set theory (Zadeh 1965) have evolved that address these shortcomings.

If an event has yet to occur, and there is more than one possible outcome, there is clearly some uncertainty about its outcome, and we need a method to deal with this uncertainty. Probability theory gives us one way of handling simple uncertainties such as this. Probability gives us a measure of the likelihood of an event resulting in one possible outcome under one set of conditions.

The outcome itself is restricted to a binary state {*true, false*}. Given some history of previous outcomes for this type of event we can determine a measure of the probability of this event being true when it occurs.

Mutually exclusive events

Take for example tossing a coin. The universe over which outcomes are defined is {*head, tail*}. The two possible outcomes of the toss are {*head,not tail*} and {*not head, tail*}. For simplicity we reduce this to the mutually exclusive outcomes head and tail. Since these outcomes are mutually exclusive, when *head* is true *tail* is false, and vice-versa. Tossing the coin twice may generate the count of each possible outcome, *head* : 1 *tail* : 1. Probability theory assigns probabilities Pr(*head*) = 0.5 and Pr(*tail*) = 0.5 for the next toss of the coin, where the probability of outcome *P* is the count of outcomes were *P* is true divided by the total number of outcomes so far encountered. For an event with possible outcomes {*Pl, P2,........,Pn*}, the probability restriction $\sum_{i=1}^{n} \Pr(P_i) = 1$ must hold.

If there is no history of previous outcomes and we have no insight into the event itself, we have total uncertainty with regard to the event outcome. This complete uncertainty is represented by the uniform a priori probability distribution. A uniform a priori probability can be assigned to each of the possible outcomes. The uniform a priori probability for all outcomes of an event is the reciprocal of the total number of possible outcomes of that event. For a fair six-sided dice the possible outcomes are {1, 2,3,4,5,6} and the uniform a priori probabilities are therefore Pr(l) = 1/6, Pr(2) = 1/6, Pr(3) = 1/6, Pr(4) = 1/6, Pr(5) = 1/6, Pr(6) = 1/6.

In almost all cases the uniform a priori probabilities are unrepresentative of the actual outcome probabilities. A better method of obtaining these probabilities is by taking a frequency of occurrence approach where we assume the number of times the event is encountered tends to infinity. This limit approach is more accurate than a uniform a priori approach but requires a large history of event outcomes.

The probability of mutually exclusive events can be combined to give a measure of the probability of the disjunction of a number of outcomes. Eqn. 5.1 shows the simple additive combination of probabilities to give the disjunctive probability $\Pr(P_1 \vee P_2 \vee P_3)$.

$$\Pr(P_1 \vee P_2 \vee P_3) = \Pr(P_1) + \Pr(P_2) + \Pr(P_3) \qquad (5.1)$$

Conditional events

These simple probability approaches are useful for many simple cases, but a more complicated problem arises when events are not mutually exclusive.

In these cases conditional probabilities can be calculated from Eqn. 5.2, the rule of conditional probabilities. Pr(A|B) is the conditional probability that A is true given that B is true.

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)} \qquad (5.2)$$

In one way the conditional probability equation gives us some elementary reasoning under uncertainty. We are uncertain if A is true, but since we know that B is true and we have Eqn. 5.2 we can at least estimate the probability Pr(A|B).

The rule of conditional probability is extended to give the rule of total probabilities. This is shown in Eqn. 5.3.

$$\Pr(B) = \Pr(B \mid A).\Pr(A) + \Pr(B \mid \overline{A}).\Pr(\overline{A}) \qquad (5.3)$$

The rule of total probabilities gives us more power in reasoning about discrete events which are not mutually exclusive. As we will later see, the rule of total probabilities is important in evaluating the support logic inference rule.

5.3.2 Bayes Theorem

Bayes theorem extends the rules of conditional probability and total probability. It provides a method of dealing with inference and belief updating in uncertainty situations.

Eqn. 5.4 defines Bayes theorem. It defines a method of calculating the conditional probability Pr(*H*|*E*) from known probability Pr(*E*|*H*) and prior probabilities Pr(*E*) and Pr(*H*).

We read Pr(*H*|*E*) as "the probability that hypothesis *H* is true given observed evidence *E*" and Pr(*E*|*H*) as "the probability of observing evidence *E* given hypothesis *H*".

Bayes theorem enables us to update the probability distribution across all independent and mutually exclusive H_i given new evidence *E*.

$$P(H_i \mid E) = \frac{P(E \mid H_i) \cdot P(H_i)}{\sum_{n=1}^{k} P(E \mid H_n) \cdot P(H_n)} \tag{5.4}$$

where,

$P(H_i \mid E)$ = probability that H_i is true given evidence *E*

$P(E \mid H_i)$ = probability of observing *E* given hypothesis H_i

$P(H_i)$ = *a priori* probability of hypothesis H_i being true

k = number of hypotheses

It is important to note that Eqn. 5.4 applies to cases where all evidence *E* is independent. This independent assumption is referred to as *naive Bayes.*

If on the other hand we encounter new evidence e, and E and e are not independent, we need to take into account conditional joint probabilities in order to calculate $P(H \mid E,e)$. This is shown in Eqn. 5.5.

$$P(H \mid E,e) = P(H \mid E) \cdot \frac{P(e \mid E,H)}{P(e \mid E)} \qquad (5.5)$$

where,

$P(H \mid E)$ = probability that H is true given evidence E

$P(H \mid E,e)$ = probability that H is true given E and new evidence e

$P(e \mid E,H)$ = probability of observing e given H and E

$P(e \mid E)$ = probability of observing e given E

The problem with modifying simple Bayes theorem (Eqn. 5.4) to the conditional evidence case (Eqn. 5.5) is in calculating the joint probabilities. For n pieces of evidence there are 2^n joint probabilities to be calculated. For reasons of computational speed, storage and knowledge acquisition, the conditional evidence case of Bayes theorem is frequently intractable.

Bayes theorem adds no more insight into complete uncertainty than basic probability theory. Given no information we still must assume a uniform a priori distribution across possible outcomes.

Newer approaches to reasoning under uncertainty fall into two camps:

- Theorems based on Bayes theorem or simplified Bayes. These include Bayesian networks and Dempster-Shafer theory.

- Approaches tackling inference under uncertainty without Bayes theorem. These include Fuzzy set theory.

5.3.3 Dempster-Shafer Theory

The Dempster-Shafer theory of evidence was designed to handle cases where the probability distribution is not completely known; it has the ability (which traditional probability theory lacks) to distinguish between uncertainty and ignorance.

Dempster-Shafer theory (Shafer, 1976) takes a slightly different approach to the theories and reasoning methods derived from probability by representing data using belief and plausibility measures.

Dempster-Shafer theory also adds a third measure, the probability assignment m, based on belief and plausibility.

Belief measure, *Bel*

Given a universe *X*, a belief measure is defined on the power set of *X*, *P(X)* as shown in Eqn. 5.6.

$$Bel : P(X) \rightarrow [0, 1] \tag{5.6}$$

such that,

$$Bel(\emptyset) = 0$$
$$Bel(X) = 1$$
$$Bel(A_1 \cup A_2 \cup ... \cup A_n) \geq \sum_j - \sum_{j<k} Bel(A_j \cap A_k) + ... + (-1)^{n+1} Bel(A_1 \cap A_2 \cap ... \cap A_n)$$

The third condition applying to Eqn. 5.6 yields the conclusion $Bel\ (A) + Bel\ (\overline{A}) \leq 1$ given that only A and $\overline{A}$ are possible and $n = 2$.

Plausibility measure, *Pl*

Plausibility is the dual of belief, and is usually defined in terms of belief, as in Eqn. 5.7.

$$Pl(A) = 1 - Bel(\overline{A}), \forall A \in P(X) \qquad (5.7)$$

Plausibility can also be defined independently, given a power set of universal set *X*, *P*(*X*), as shown in Eqn. 5.8

$$Pl : P(X) \rightarrow [0, 1] \qquad (5.8)$$

such that,

$Pl(\emptyset) = 0$

$Pl(X) = 1$

$Pl(A_1 \cup A_2 \cup ... \cup A_n) \geq \sum_j - \sum_{j<k} Pl(A_j \cap A_k) + ... + (-1)^{n+1} Pl(A_1 \cap A_2 \cap ... \cap A_n)$

In the dual of belief, the third condition applying to Eqn. 5.8 yields the conclusion $Pl(A) + Pl(\overline{A}) \geq 1$ for the condition that only *A* and $\overline{A}$ are possible and $n = 2$.

Probability assignment *m*

The probability assignment m defined by Dempster and Shafer attempts to relate the measures *Bel* and *Pl* directly to probability theory.

The Dempster-Shafer probability assignment *m* is unlike the basic probability distribution, which is defined over the universe *X*, in that *m* is defined over the *power set* of *X*, *P*(*X*).

$$m\!: P(X) \rightarrow [0, 1] \qquad (5.9)$$

such that,

$m(\emptyset) = 0$

The focal elements *A* of *P*(*X*) are defined as those elements of *P*(*X*) which have non-zero probability assignment. Clearly = cannot be a focal element.

Clearly if we take a subset of *P(X)* containing only the singleton sets, $\{e\}\forall e \in X$, then this is analogous to the basic probability density function. The basic probability density function is therefore a restricted case of the Dempster-Shafer probability assignment.

Given that *m* is defined over the power set of *X*, the quantity *m(p)* is interpreted as the belief that is currently assigned to the exact set of hypotheses *p*.

It is important to note that the definition of *m* does not require that *m(X)* = 1 (as the basic probability density function does) or that *m(A)* ≤ *m(B)* when $A \subset B$. The second of these two cases is important because it gives us more representation power then the basic probability density function.

Bel, *Pl* and *m* are related by Eqns. 5.10 and 5.11, where *A* is a subset of *P*(*X*)

$$Bel(A) = \sum_{B \subseteq A} m(B) \qquad (5.10)$$

$$Pl(A) = \sum_{B \cap A \neq \phi} m(B) \qquad (5.11)$$

This clearly gives rise to the condition, *Pl*(*A*) ≥ *Bel*(*A*).

There are two special conditions to note for all of *Bel*, *Pl*, and *m*. These are *total ignorance* and *absolute certainty*. The absolute certainty cases are shown in Eqns. 5.12, 5.13, 5.14.

$$Bel(\{A\}) = 1 \text{ and } Bel(B) = 0, \forall A \in X, B \neq \{A\}, B \in P(X) \quad (5.12)$$

$$Pl(\{A\}) = 1 \text{ and } Pl(B) = 0, \forall A \in X, B \neq \{A\}, B \in P(X) \qquad (5.13)$$

$$m(\{A\}) = 1 \text{ and } m(B) = 0, \forall A \in X, B \neq \{A\}, B \in P(X) \qquad (5.14)$$

The total ignorance cases are shown in Eqns. 5.15, 5.16, and 5.17.

$$Bel(X) = 1 \text{ and } Bel(A) = 0,\ \forall A \neq X, A \in P(X) \tag{5.15}$$

$$Pl(\emptyset) = 0 \text{ and } Pl(A) = 1,\ \forall A \neq X, A \in P(X) \tag{5.16}$$

$$m(X) = 1 \text{ and } m(A) = 0,\ \forall A \neq X, A \in P(X) \tag{5.17}$$

Dempster evidence combination

Dempster's evidence combination method combines two different bodies of evidence, expressed as probability assignments. Eqn. 5.18 defines the method to combine probability assignments m_1 and m_2 to give a joint probability assignment m_3.

$$m_3(z) = \frac{\sum_{X \cap Y = Z} m_1(X) \cdot m_2(Y)}{1 - \sum_{X \cap Y = \phi} m_1(X) \cdot m_2(Y)} \tag{5.18}$$

In practice it is easier to examine an application of this rule in table form. Take, for example, the set shown in Eqn. 5.19 with power set shown in Eqn. 5.20.

$$X = \{a,b,c\} \tag{5.19}$$

$$P(X) = \{\emptyset,\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\} \tag{5.20}$$

If we now have evidence expressed as $m_1(\{a,b\}) = 0.\ 7$ and $m_2(\{b,c\}) = 0.\ 3$ we can calculate the combined evidence m_3 from the table in Table 5.1. Note that, from Eqn. 5.9, the empty set is only permitted to have a zero probability assignment and that the sum of all probability assignments must be one. Given these two condition the remainder of the probability assignment for m_1 must be allocated to the universal set *X*. Thus $m_1(X) = 0.3$. Likewise for m_2 the remainder of the probability assignment (0.7) is assigned to *X*. Thus $m_2(X) = 0.7$

			m_2			
			$\{b,c\}$	0.3	X	0.7
m_1	$\{a,b\}$	0.7	$\{b\}$	0.21	$\{a,b\}$	0.49
	X	0.3	$\{b,c\}$	0.09	X	0.21

Table 5.1: Table for m_3

The new probability assignment m_3 is expressed by the following table.

m_3			
$\{b\}$	$\{a,b\}$	$\{b,c\}$	X
0.21	0.49	0.09	0.21

Note that $\sum_{A \in P(X)} m_3(A) = 1$.

Now let us consider what happens as more evidence, m_4, is presented. If m_4 ($\{c\}$) = 0.7 we must find the joint probability assignment of m_3 and m_4 in order to assimilate this new evidence. The table for this new Dempster combination is shown in Table 5.2.

			m_4			
			$\{c\}$	0.7	X	0.3
m_3	$\{b\}$	0.21		0.21	$\{b\}$	0.49
	$\{a,b\}$	0.49		0.09	$\{a,b\}$	0.21
	$\{b,c\}$		$\{c\}$		$\{b,c\}$	
	X		$\{c\}$		X	

Table 5.2: Table for m_3

Taken directly from Table 5.2 the new probability assignment m_5 is expressed as the following distribution

$m_5(\{b\}) = 0.063$

$m_5(\{ a, b\}) = 0.\ 147$

$m_5(\{b,c\}) = 0027$

$m_5(\{c\}) = 0.21$

$m_5(X) = 0.063$

$m_5(\emptyset) = 0.49$

This distribution shows a probability assignment of 0.49 has been assigned to the empty set. This indicates that m_3 and m_4 define conflicting evidence. Dempster's rule decrees that this assignment must now be distributed among the other members of the assignment. This is achieved by dividing all other assignments by 1 - $m_5(\emptyset)$ = 1 - 0.49 = 0.51. The re-scaled assignment $m_5^{'}$ is now shown as the following distribution.

$m_5^{'}(\{b\}) = 0.1235294118$

$m_5^{'}(\{a, b)) = 0.2882352941$

$m_5^{'}(\{b, c\}) = 0.0529411765$

$m_5^{'}(\{c\}) = 0.4117647059$

$m_5^{'}(X) = 0.1235294118$

$m_5^{'}(\emptyset) = 0$

The belief distribution represented by the probability assignment m_3 has been revised in light of the evidence in m_4 to give a final probability assignment $m_5^{'}$.

The renormalisation of the final probability assignment to redistribute probability assigned to the empty set is a contentious operation. Baldwin's mass assignment theory overcomes this problem through the mass assignment definition and combination methods.

5.4 Symbolic Methods

5.4.1 Fuzzy Set Theory

Fuzzy set theory was originally introduced in 1965 by Zadeh (1965) to address uncertainty. A further motivation behind the introduction of fuzzy sets was to provide a more natural and transparent mapping between the real world and mathematics.

Possibility theory as explained in section 5.3.1 enables us to obtain the possibility of a conjoined event solely from the possibility of the individual events; possibility is truth functional. Further, possibility theory makes no assumptions about underlying distributions and so it is non parametric. A disadvantage of possibility theory is that there is no central limit theorem as in parametric statistics although more evidence can be used to restrict a possibility distribution to have fewer values. If we work with possibility distribution then we have what is known as a fuzzy logic Zadeh (1965). We may however take any multi-valued logic and work with distributions and provided it is truth functional the advantages and disadvantages outlined above will tend to apply.

Dempster Shafer theory and the above have introduced possibility as a basis for fuzzy logic; however, other viewpoints can be taken. Probability is based on precise events and the probability of an event is based on the number of times the event occurs divided by the number of possible events. The crucial point is the set of events that forms the basis being precise. We might find it difficult to say whether a particular person is tall or not and so it then becomes difficult to assess the probability of, say, the next person to enter a room being tall as the definition of "TALL" is imprecise or fuzzy.

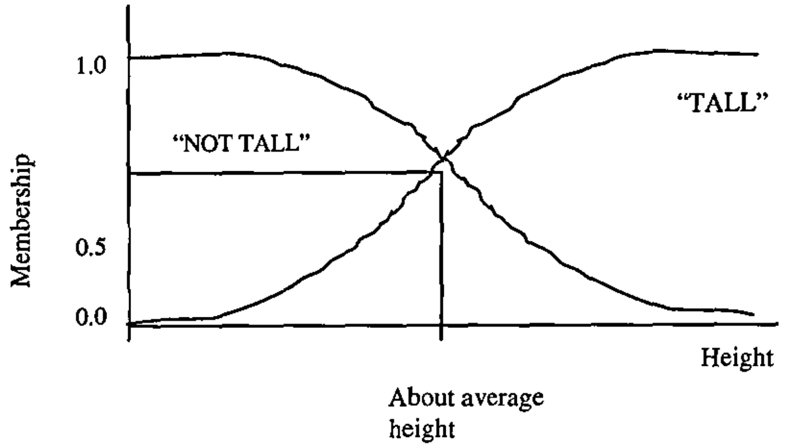


Figure 5.1. Possibility Distribution "TALL" and the complemented "NOT TALL"

A fuzzy set is characterised by a membership function which maps each element x in the universe of dicsourse Ω to membership value in the unit interval [0..1] as opposed to {0, 1} in traditional set theory. For example the fuzzy set TALL could be characterised by the membership function $\mu_{TALL}(x)$ (depicted in Figure 5.2).

Given the fuzzy membership function in Figure 5.2 as a definition of the concept "TALL" then given such a membership function we are able to read off a grade of membership given a height and also read back a height given a grade of membership.

In this case height values in the interval [6 feet and higher] have a membership value of 1 and correspond to the core of the fuzzy set. Values in the intervals [5'6",6'] have membership values in the range [0, 1]. While other values in the universe have zero membership in this definition of the concept of tall. Values having membership greater than zero in a fuzzy set correspond to the support of the fuzzy set.

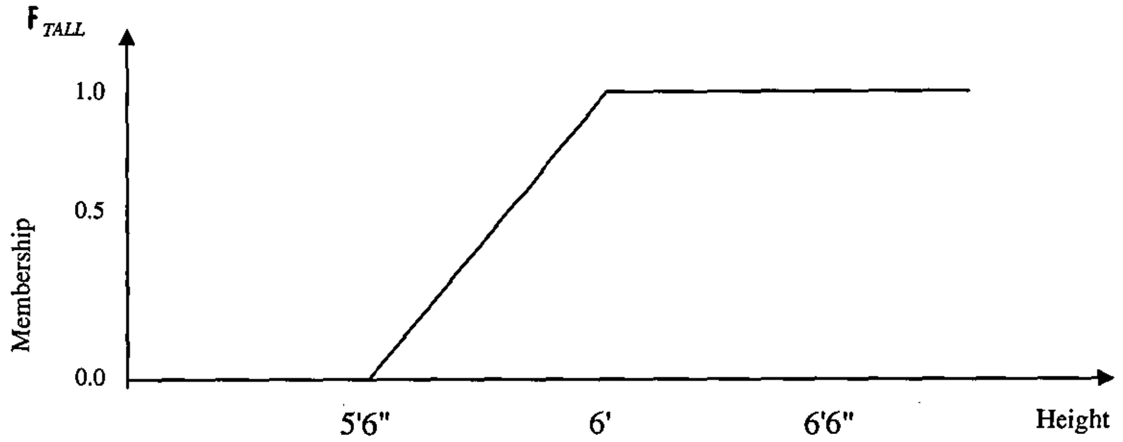


Figure 5.2. An example of a fuzzy set defined over the universe of height values expressed in centimetres.

An extensive calculus of fuzzy set operations exists including union, intersection, complement etc., which in most cases are generalisations of traditional crisp set theory (Klir and Yuan, 1995). Furthermore, a reasoning framework has been developed based upon fuzzy truth-values: fuzzy logic (Zadeh 1979). Fuzzy set theory and fuzzy logic has enjoyed considerable success in the knowledge-based systems such as motor control and found applications in numerous other fields including:

- Pattern recognition
- Decision making
- Robot planning
- Engineering design
- Systems modeling
- Process control
- Social interaction systems
- Structural semantics
- Chromosome classification

Fuzzy sets are based on the idea of continuously graded degrees of membership of sets. The characteristic function of an ordinary set

$$\mu_A(x): \mathrm{U} \rightarrow \{0,1) \quad \text{where } \mu(x) = 0 \quad x \text{ in A}$$
$$\mu(x) = 1 \quad x \text{ not in A}$$

is replaced for a fuzzy set, with a characteristic function of the form

$$\mu_A(x) \ : \mathrm{U} \rightarrow [0,1] \tag{5.21}$$

which specifies the 'degree of membership of x in A'. with this definition 'crisp' concepts can still be represented adequately, but there is no necessity to assign artificial boundaries to concepts which are inherently vague.

The standard set operations: union, intersection, complement can be defined for fuzzy sets in several different ways. The definitions which are most commonly used are:

- *union*

$$\mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)], x \in \mathrm{U} \tag{5.22}$$

- *intersection*

$$\mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)], x \in \mathrm{U} \tag{5.23}$$

- *complement*

$$\mu_A(x) = 1 - \mu_{\bar{A}}(x), x \in \mathrm{U} \tag{5.24}$$

It can be shown that these definitions of union and intersection are the only one that are consistent with the requirements that the operations should reduce to the normal set operations for degree of membership of 0 and 1, that they should be order preserving and continuous, and that the normal *associativity*, *commutativity*, *distributivity*, and *idempotence* rules should be obeyed. If the *distibutivity* and *idempotence* requirements are dropped, which may be considered desirable for reflecting natural language usuage, then Zadeh's alternative definitions can be used:

$$\mu_{A\cup B}(x) = [\mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)], x \in \mathrm{U} \qquad (5.25)$$

$$\mu_{A\cap B}(x) = [\mu_A(x) \cdot \mu_B(x)], x \in \mathrm{U} \qquad (5.26)$$

As well as the standard set operations, there is a range of operations which are specific to fuzzy set, for example concentration which reduces the degree of membership of elements which are 'only partly' in the set. Normalization which adjusts the degree of membership so that at least one element is 'totally' in the set, intensification and fuzzification.

Imprecise statements can be modelled as fuzzy sets using linguistic variables, variables whose values are natural language expressions referring to some quantity of interest. These expressions can be represented by fuzzy sets composed of the possible values that the quantity of interest can assume. For example, if the quantity of interest could assume an integer value between 1 and 10, the expression 'few' could be represented by

{0.4/1, 0.8/2, 1/3, 0.4/4}

The natural language expressions normally form a structured finite set, with syntactic rules for generating expressions and semantic rules for associating fuzzy sets with them. Primary terms are modelled by fuzzy sets, and hedges (e.g. 'very', 'fairly', 'quite' etc) are modelled by fuzzy set operations. (Schmucker, 1984).

Fuzzy set theory can be used to extend classical logic to produce fuzzy logic in which the constraint that every statement must be either absolutely true or absolutely false no longer applies. The compositional rule of inference, which states that if R is a fuzzy relation from U to V and X is a fuzzy subset of U, the fuzzy subset of V which is induced by X is given by the composition of R and X, can be used when variables range over finite sets.

If X is B then Y is C
X is A
....
Y is D

Where X and Y are variables in universe U and V respectively, A and B are fuzzy subsets of U, and C and D are fuzzy subsets of V.

The concept of fuzziness can be extended to mathematical structures, replacing the concept of the value of a variable with 'the degree of membership of a value', as a result of which values seem to play the role of functions and non-fuzzy functions become functional (Gaines, 1976), and to the domains of interest of sets: operations which map a fuzzy set and domain of interest into a new fuzzy set and new domain of interest can be used.

One problem with fuzzy set theory is that there is no proof that it models perception or judgement, and no clearly defined way of determining if a given membership function is right (Wise, 1986). The theory assumes that grades of membership of property categories may be expressed by functions, the values of which submit to the conventional arithmetic operations, and if unary operations such as the transformation of fuzzy sets with hedges are to be meaningful, a ratio scale must be used for subjective measurements. Other problems with Zadeh's fuzzy logic include extreme vagueness of results in fuzzy conditional propositions, and weaknesses in the ways in which chain reasoning, conjunctive fuzzy conditional propositions and combination of evidence are dealt with (Nafarieh, 1988).

Fuzzy Numbers & Hedges

Rules used by people use "linguistic" variables such as "much lower", "a lot", "a little", which we need to interpret more precisely. For this we need to develop the idea of a fuzzy number. An approximation to a fuzzy number is such a method. By approximating a normal distribution with the view that almost any "reasonable" interpretation will give us "reasonable" results then we could take a much simpler approximation and be just as right or wrong (Zadeh L.A., 1965). The fuzzy membership function, distribution diagram (Figure 5.1) shows such an approximation. The distribution diagram (Figure 5.1) representing a version of the concept "TALL" can be simplified substantially as shown in Figure 5.3.

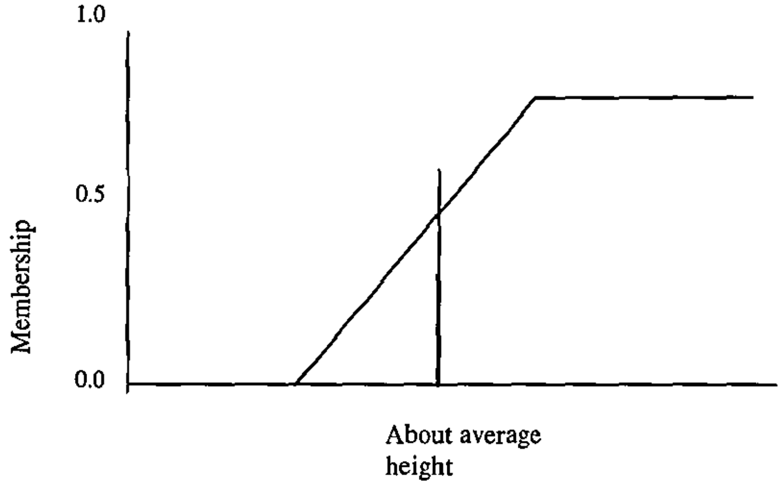


Figure 5.3 Simplified version of the fuzzy membership function "TALL"

Traingular and trapezoidal two sided distributions are usually used to represent fuzzy concepts and fuzzy numbers however it is also necessary to catch outlying points and so, especially for input sets, single tailed distributions like in Figure 5.1 will be needed.

The relationship between the sets is important and Figure 5.4 shows that a particular real number can belong to more than one set, but with different degrees of membership. This is important for interpolating between various different rule outputs when we come to transfer the fuzzy output sets into a control action. Fuzzy membership diagrams are a way of selecting which grade of membership is most suitable for any given crisp value.

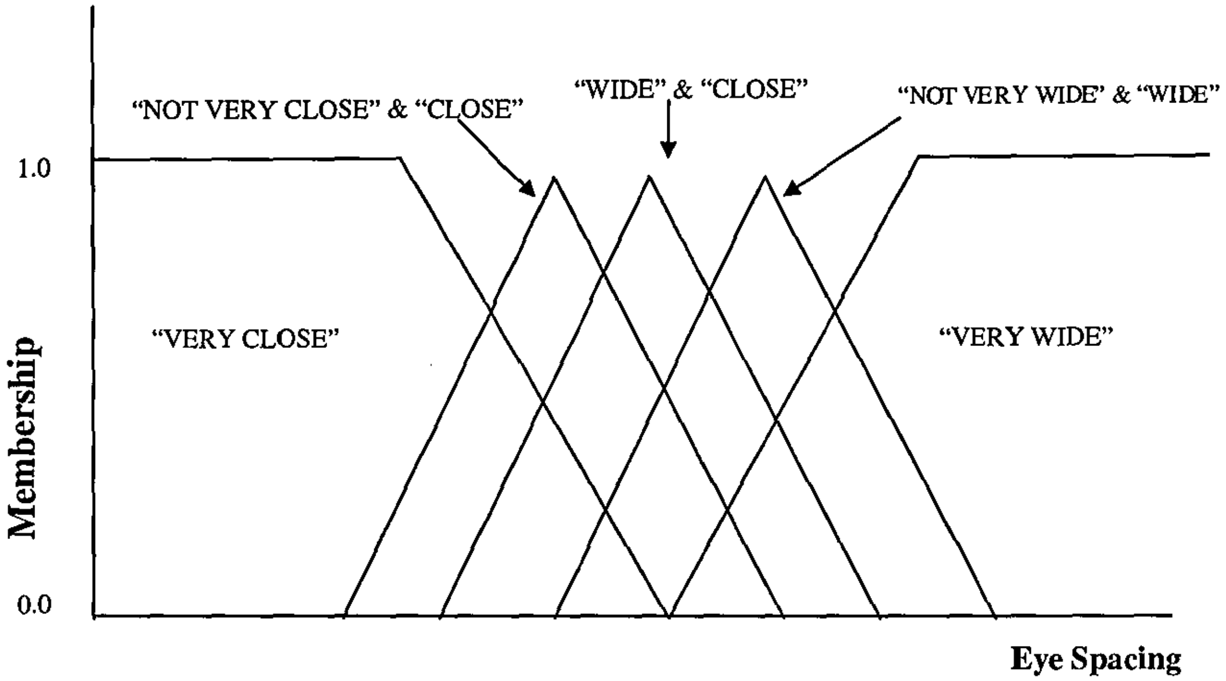


Figure 5.4 A relationship between the fuzzy sets for Eye Spacing

These sets are usually difficult to describe accurately and precisely, therefore it is standard and computationally efficient to use triangular sets. As fuzzy distributions are generally used to describe vague and approximate concepts this is a reasonable decision with respect to the operation of the fuzzy system.

The concept of "Hedges" within the topic of fuzziness is an important one and highly relevant to the project in discussion. Apart from distributions such as "large", "small", "medium", "wide", etc. There could be other distributions derived from these such as "very wide" and "fairly wide". These adjectives "very" and "fairly" are known as hedges and modify the distributions they are applied to.

The use of hedges enables finer distinctions in the sets to be derived and so allow better judgement to be made about which set something should be a member of. Hedges are unary operators and so "NOT", "FAIRLY" and "VERY" can be interpreted as follows:

If $\mu_A(x)$ is the membership of the support element X in the set A then the hedges "VERY", "FAIRLY" and "NOT" are usually denoted as:

$$\mu_{Very(A)}(X) = \mu_A(X)^2 \tag{5.27}$$

$$\mu_{Fairly(A)}(X) = \mu_A(X)^{0.5} \tag{5.28}$$

$$\mu_{Not(A)}(X) = 1 - \mu_A(X) \tag{5.29}$$

Whereas the usual membership function for "Tall" is shown in Figure 5.5

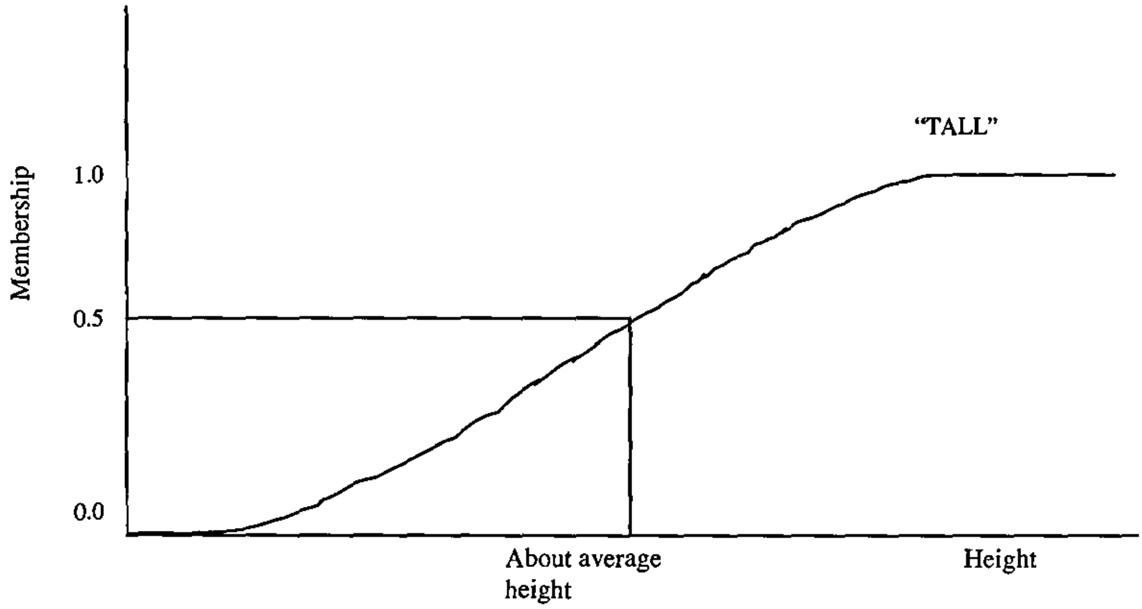


Figure 5.5 Shows a denotation of the membership function "TALL"

The membership function for "VERY TALL" might look like Figure 5.6 with "TALL" shown for comparison. Figure 5.7 shows the transformation NOT applied to "VERY TALL"

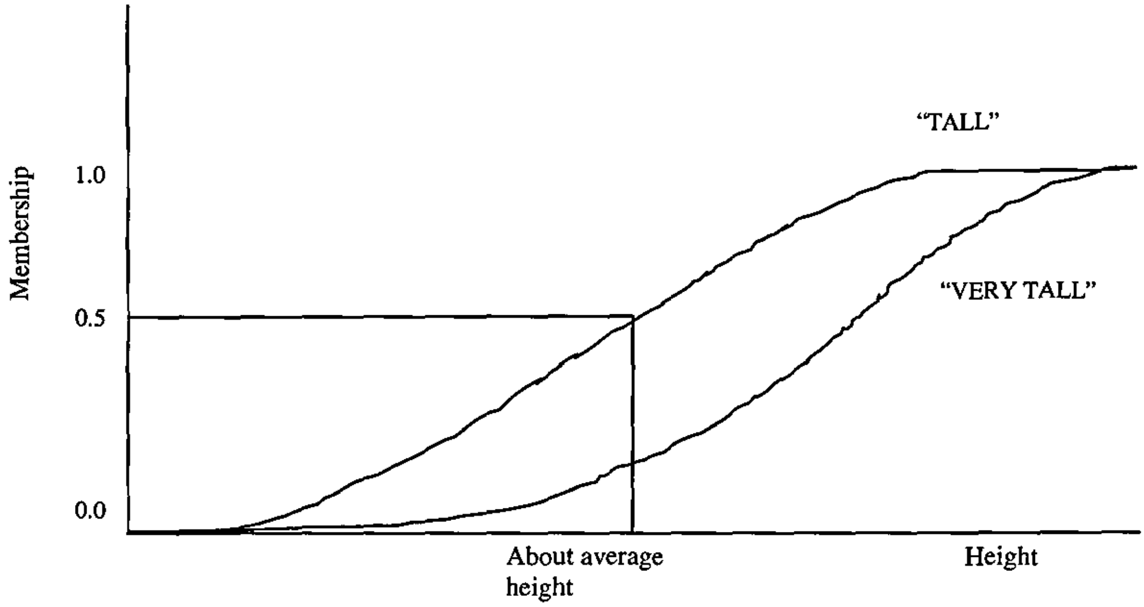


Figure 5.6 Shows membership functions for sets "TAL" and "VERY TALL"

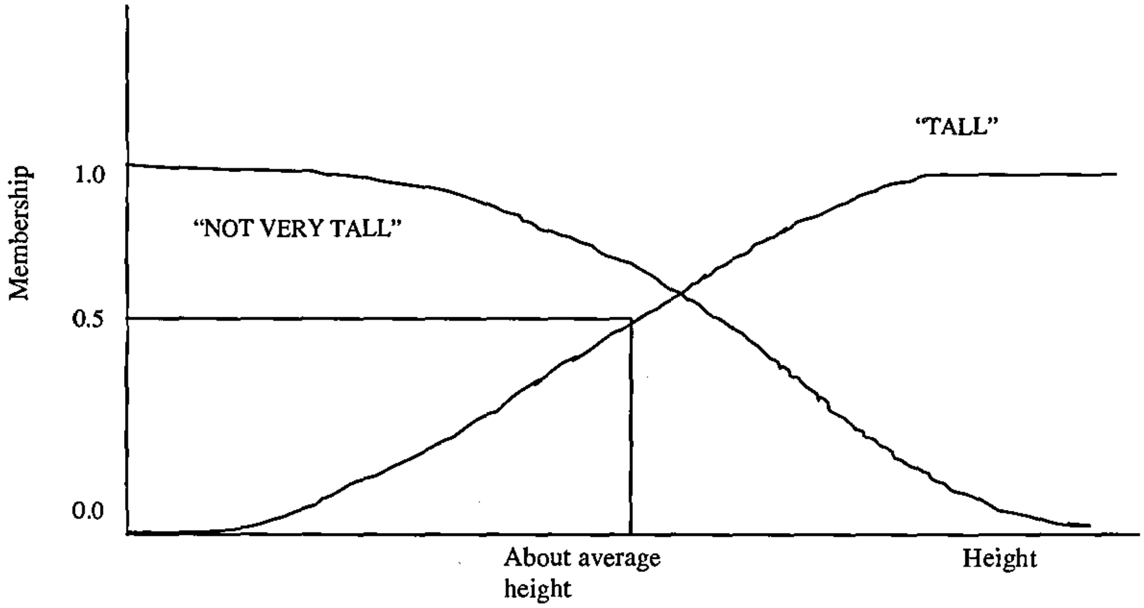


Figure 5.7 Showing the transformation NOT on "VERY TALL". The intersection would constitute the set "TALL" but "NOT VERY TALL"

Using the intersection of "TALL" and "NOT VERY TALL" gives the fuzzy set corresponding most closely to "TALL" so some one with the height within the triangle would have a description of "TALL" but "NOT VERY TALL". Similarly the set "TALL" can be weakened to "FAIRLY TALL" by applying "FAIRLY" membership hedge. Figure 5.8 shows the result of applying the hedge "FAIRLY" to the set "TALL".

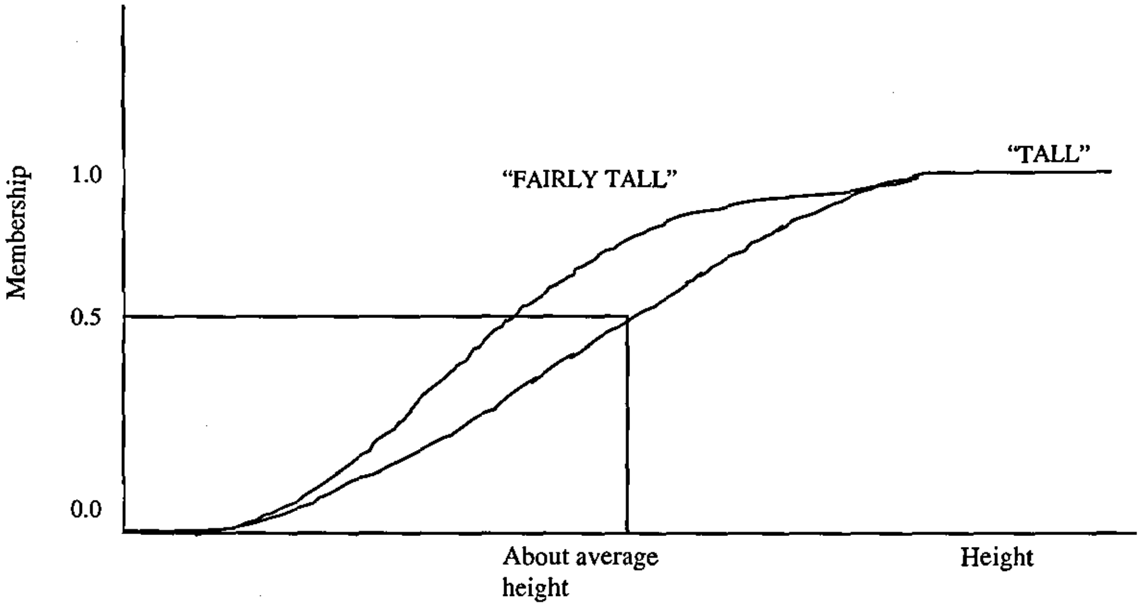


Figure 5.8 Showing the effect of the dilation operator "FAIRLY" on the set "TALL". The way "FAIRLY" and "VERY" have been defined makes them inverse.

Using hedges to intensify and dilute fuzzy sets allows other fuzzy sets to be created. Going back to figure 5.7 showing the intersection of "TALL" and "NOT VERY TALL" gives us an intersection of the two sets which is a non normal set, shown in figure 5.9.1 A non normalised set is one in which the membership does not reach the maximum value of 1.0.

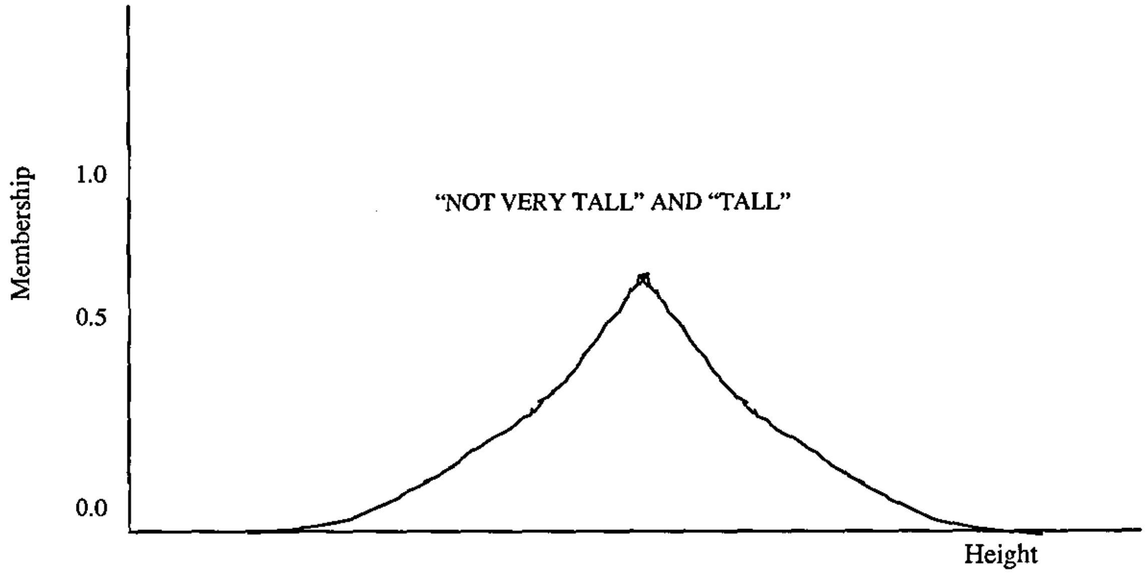


Figure 5.9.1 The result of intersecting "NOT VERY TALL" and "TALL" results in this non normalised set

Movement of the belief from uncertainty into the sets corresponding to the fuzzy set equally results in the normalised set shown in Figure 5.9.2

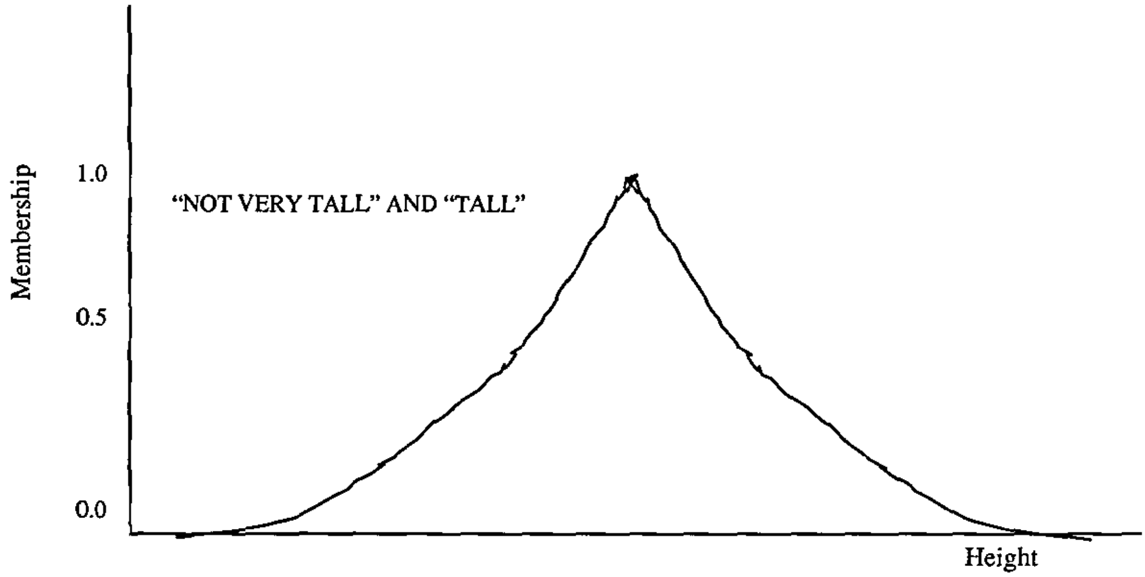


Figure 5.9.2 This is a normalised version of the set "NOT VERY TALL" and "TALL" which reaches the maximum value 1.0.

The set "NOT VERY TALL" and "TALL" when normalised gives us a set which give the membership function of a person and whether the descriptor "TALL" is appropriate. Putting this set alongside "VERY TALL" allows us to select the descriptor that best suits the person.

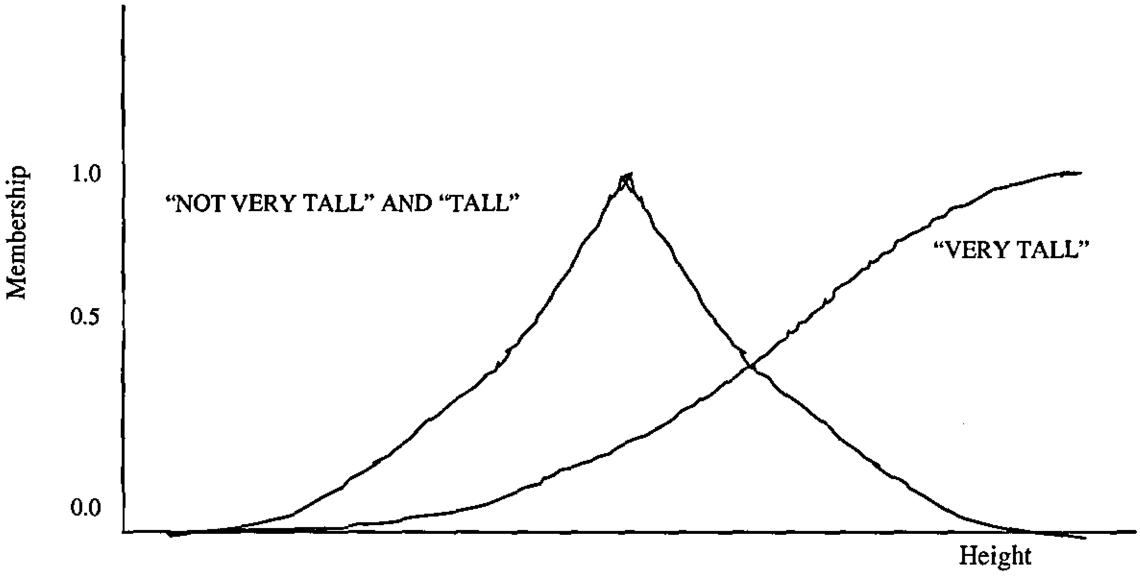


Figure 5.10 Putting "NOT VERY TALL" and "TALL" alongside "VERY TALL" enables appropriate descriptor to be selected

Given we have a measurement to make on an object, perhaps a person, then we can select the height along the abscissa and read off the value of the appropriate curve to assign the most possible descriptor. Looking at a reasonable set that could be constructed from just "TALL" we can have "TALL", "VERY TALL", "SHORT" derived from "NOT TALL" and "VERY SHORT" derived similarly. These are shown in Figure 5.11.1 and Figure 5.11.2

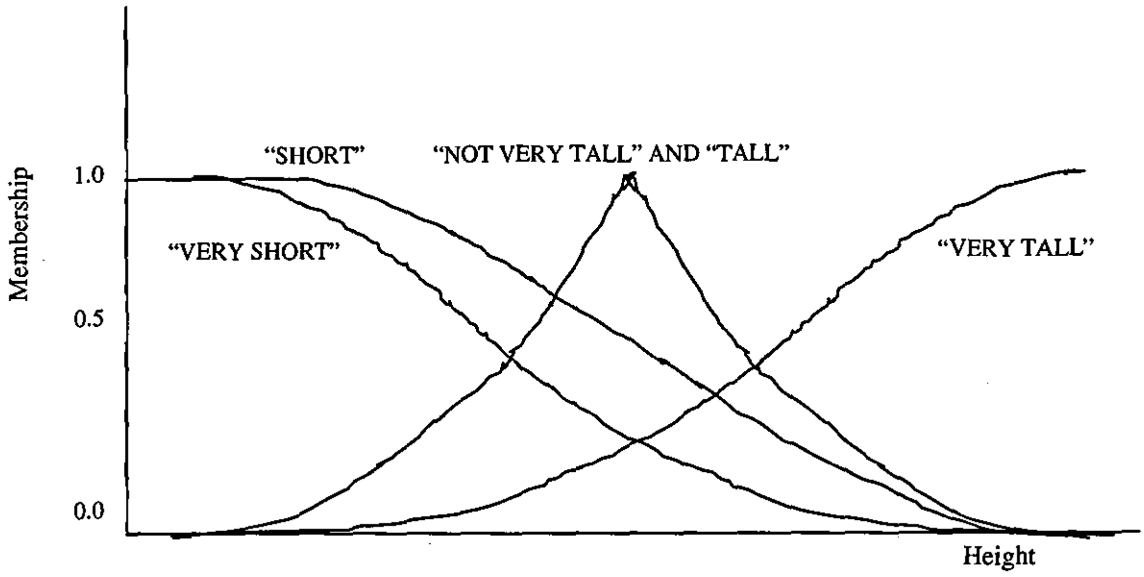


Figure 5.11.1 "SHORT" and "VERY SHORT" shown with descriptor sets derived from "TALL" and "VERY TALL"

We can extend our ability to form description sets by adding SHORT" and "NOT VERY SHORT" replacing "SHORT" and in the middle we have "NOT SHORT" and "NOT TALL" which collapses to "SHORT AND TALL". These are then normalised and we can take any value we like for height and select the most appropriate descriptor. If we need more detail then the second largest membership can be chosen.

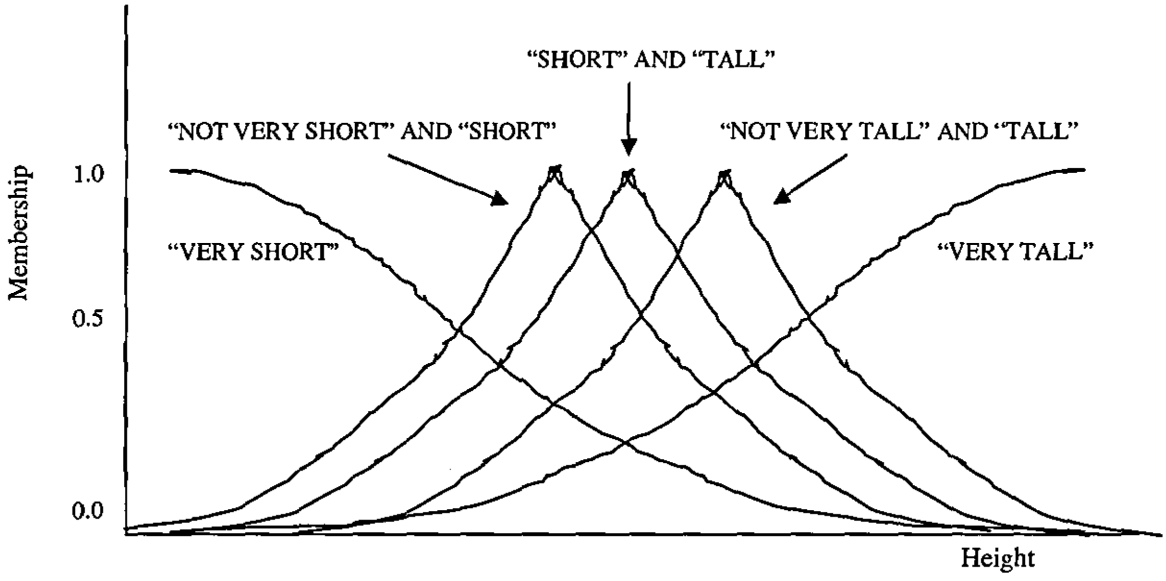


Figure 5.11.2 Some of the sets derivable from the original set "TALL" which allows us to match object with natural descriptions

These sets are very useful in the area of fuzzy control and enable input values to be mapped onto fuzzy sets. The beauty of these sets is that they are all derivable from experience. Examining the transformation implied by the two sets "TALL" and "VERY TALL" could derive the operator "VERY". The same goes for "SHORT" and "VERY SHORT". The triangular sets as shown in Figure 5.12 are useful in describing objects in a concise manner by selecting the most appropriate descriptor.

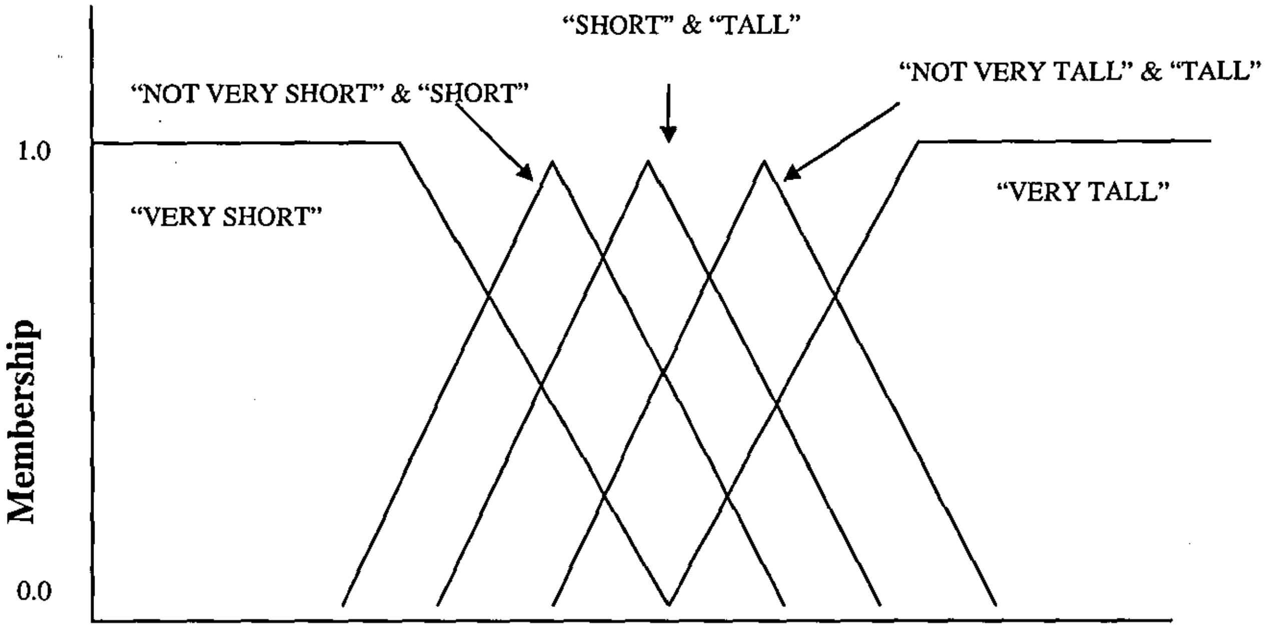


Figure 5.12 The approximations to the distributions shown in Figure 5.11made by taking the closest triangular distribution to the experientially derived distributions

5.4.2 Mass Assignement

In order to address the shortcomings of probability theory, when further incompleteness in the knowledge exists, namely that a complete probability distribution over the frame of discernment cannot be given (which corresponds to a form of ignorance), mass assignment theory (to be referred as MAT from now on) has been proposed by Baldwin (1991; 1992). In MAT, the distribution is given over the power set elements of the frame of discernment. This distribution is called a mass assignment. MAT differs from previous work in this area by Dempster and Shafer

(Dempster, 1967; Shafer, 1976), by catering for not only ignorance, but also for inconsistency (allowing mass to be assigned to the null set) and providing a different and more expressive calculus. The mass assignment is similar in representation terms to the basic probability assignment of Dempster-Shafer theory.

A mass assignment over a finite frame of discernment Ω is a function:

$$m : P(X) \rightarrow [0,1] \qquad (5.30)$$

Where *P(X)* is the power set of Ω and satisfies the condition

$$\sum_{A \in P(X)} m : (A) = 1 \qquad (5.31)$$

Every set $A \in P(X)$ for which $m(A) > 0$ is called the **focal element** of *m*.

A mass assignment can be viewed as a form of knowledge that expresses upper and lower probabilities for the individual elements of the frame of discernment. In other words, a mass assignment can be viewed as a family of probability distributions, all of which satisfy the axioms of probability theory and the upper and lower constraints delimited by the mass assignment. Consequently, although mass assignments can represent probabilities they have the added flexibility of being able to represent uncertain probabilities. For example, consider a class of undergraduate students where students can be classified as *first-class honours*, *second-class honours* or as *pass*. Consider the case where there are 100 students, where it is known that 30 are *pass* students, 40 are *second-class honours* or *pass* and the remainder unknown. This can be more succinctly written in mass assignment format as follows:

$MA_{Class} = \{pass\} : 0.3$

{pass,second-class honours}:0.4

{pass,second-class honours,first-class honours}:0.3

This mass assignment corresponds to the following family of probability distributions

$0.3 \leq \Pr(\textit{pass}) \leq 1$

$0 \leq \Pr(\textit{second-class}) \leq 0.7$ $0 \leq \Pr(\textit{first-class}) \leq 0.3$

such that

$\Pr(\textit{pass}) + \Pr(\textit{second-class}) + \Pr(\textit{first-class}) = 1.0$

A particular type of probability distribution is obtained by distributing the mass associated within the non-singleton focal elements uniformly; this distribution is termed as the **least prejudiced distribution** (LPD) (Baldwin 1992). In the case of MA_{Class} the corresponding LPD, LPD_{Class} is given as follows:

$\Pr(\textit{pass}) = 0.3 + 0.4/2 + 0.3/3 = 0.6$

$\Pr(\textit{second-class}) = 0.4/2 + 0.3/3 = 0.3$

$\Pr(\textit{first-class}) = 0.3/3 = 0.1$

The transformation of mass assignment to a least prejudiced distribution is reversible; hence given a least prejudiced distribution it is possible to find a corresponding mass assignment.

Mass Assignment Calculus

Mass assignments can be combined, corresponding to the conjunction of knowledge statements, aggregated corresponding to the combination of alternate knowledge statements and updated, corresponding to forming a posterior mass assignment from a priori mass assignment when given some specific knowledge, also expressed as a mass assignment. In Baldwin (1991; 1992) a detailed presentation of the mass assignment calculus (meet, join, restrictions, conditioning) is presented.

Mass Assignment Combination

As with probability assignments, two mass assignments can be combined. There are two basic mass assignment meets, the general *assignment* and the *multiplication* meet. These two methods are outlined below.

1. General assignment,

General assignment meet of mass assignments m_1, and m_2 assumes a unique redistribution of mass from m_1 and m_2 onto the intersection or union of focal elements in m_1 and m_2. No mass can be assigned to the empty set. The union of focal elements generates a more general assignment which can be restricted to either of the original components, but the family of probability distributions resulting are not necessarily the union of the component families of probability distributions. The intersection of focal elements on the other hand does result in a family of probability distributions which is the intersection of the component families of probability distributions.

Taking the two mass assignments,

$$m_1 = L1_i : M1_i \mid i = 1,\dots,n_1$$
$$m_2 = L2_j : M2_j \mid j = 1,\dots,n_2$$

defined over the universe of labels $\{ L1_1,\dots,L1_{n_1}, L2_1,\dots,L2_{n2} \}$.

We now define a tableau m^* of elements subject to the row and column constraints in Eqns. 5.32 and 5.33, where * represents union or intersection, and m^* ($\emptyset$) = 0.

$$\sum_{i}^{n_1} m^*(L1_i * L2_j) = m_2(L2_j), \text{ for } j = 1 \text{ to } n_2 \tag{5.32}$$

$$\sum_{i}^{n_2} m^*(L1_i * L2_j) = m_1(L1_i),\ \text{for } i = 1 \text{ to } n_1 \qquad (5.33)$$

Now we find the mass associated with each focal element in m^* from the tableau, as shown in Eqn. 5.34.

$$m^*(L_k) = \sum_{i,j,L1_i*L2_j=L_k} m^*(L1_i * L2_j) \qquad (5.34)$$

The new mass assignment m_3 is the general assignment combination of m_1 and m_2 and is written $m_3 = m_1 \oplus m_2$.

A non-unique solution is harder to calculate, and we must introduce unknowns into the resulting mass assignment expression that capture the whole family of possible mass assignments.

2. Multiplication meet, $\wedge$.

The multiplication meet is more simple than the general assignment. It is faster and simpler to calculate and generates a unique solution. On the other hand multiplication meet does allow assignment of mass to the empty set and, as a result, may generate inconsistent results.

The method of calculating multiplication meet is the same as for general assignment up to assigning masses to cells in the tableau. At this point mass for each cell is simply the product of the masses associated with the heads of the corresponding row and column.

In other words, for the cell intersecting sets L_1 and L_2 the mass associated with that cell is,

$$\mathrm{m}^*(L_1 * L_2) = m_1(L_1) \,.\, m_2(L_2) \qquad (5.35)$$

In essence the multiplication meet is identical to the Dempster-Shafer combination method, but without the reallocation of empty set mass to the non-empty sets of the mass assignment.

5.4.3 Semantic Unification

Semantic unification gives us a method of comparing one fuzzy set with another. This is crucial in semantic analysis of fuzzy concepts.

Take for example the two sentences "Fred is tall" and "Bill is short". Naturally we know that Fred is taller than Bill, but it would be much more useful to know to what degree Fred is taller. Given a new statement, "Joe is very short" we would also expect the comparison method to give a higher similarity measure between very short and short than between very short and tall. This similarity measure is provided by semantic unification.

Semantic unification of fuzzy set F with fuzzy set F′ generates a similarity measure *SU*(F, F′) in the interval [0, 11. We take this value to be equal to the conditional probability *Pr*(.F | F′). We can see from this conditional probability equivalence that semantic unification is not commutative, i.e., *Pr*(.F | F′) is not necessarily equal to *Pr*(F′ | F).

Although we have talked of semantic unification in terms of fuzzy sets, it actually operates on mass assignments. As a result fuzzy sets are translated into their mass assignment equivalents, before semantic unification.

We use two different method of semantic unification, *interval semantic unification* and *point semantic unification*

- *Interval Semantic Unification*

The interval version of semantic unification generates a measure $Pr(\mathsf{F} \mid \mathsf{F}')$ which is expressed as a support pair $[S_n, S_p]$

Take the fuzzy sets F and F'. These are converted to their respective mass assignments,

$$m_{\mathsf{F}} = \{L_i : l_i\} \tag{5.36}$$

$$m_{\mathsf{F'}} = \{M_j : m_j\} \tag{5.37}$$

Now we can calculate the semantic unification of F given F' by deriving from m_{F} and $m_{\mathsf{F'}}$, a mass assignment across the universe $\{t, f, u\}$ where t represents true, f represents false and u represents uncertain. We generate this new mass assignment from Eqn. 5.38.

$$M = \{T(L_i \mid M_j) : l_i \cdot m_j\} \tag{5.38}$$

where,

$$T(L_i \mid M_j) = \begin{Bmatrix} t : M_j \subseteq L_i \\ f : M_j \cap L_i = 0 \\ u : otherwise \end{Bmatrix} \tag{5.39}$$

Now we have one expression for $m_{(\mathsf{F} \mid \mathsf{F})}$ defined over the focal elements, as shown in Eqn. 5.40.

$$m_{(\mathsf{F} \mid \mathsf{F})} = t : \sum_{\substack{i,j \\ T(L_i|M_j)=t}} l_i \cdot m_j, f : \sum_{\substack{i,j \\ T(L_i|M_j)=f}} l_i \cdot m_j, u : \sum_{\substack{i,j \\ T(L_i|M_j)=u}} l_i \cdot m_j \tag{5.40}$$

Finally we derive the support pair $[S_n, S_p]$ for $Pr(\mathsf{F} \mid \mathsf{F}')$ as in Eqns. 5.41 and 5.42.

$$S_n = m_{(\mathsf{F} \mid \mathsf{F})}(t) \tag{5.41}$$

$$S_p = 1 - m_{(\mathsf{F} \mid \mathsf{F})}(f) \tag{5.42}$$

- *Point Semantic Unification*

A simplification of the interval semantic unification algorithm gives us the point semantic unification. This algorithm returns a point probability value for $Pr(\mathsf{F} \mid \mathsf{F}')$ rather than an interval.

Given m_{F} and $m_{\mathsf{F}'}$ defined in Eqns. 5.36 and 5.37 respectively, we generate a new mass assignment M given by Eqn. 5.43.

$$M = \{m_{ij}\} = \left\{\frac{| L_i \cap M_j |}{|M_j|}\right\} l_i \cdot m_j, \forall i, j \tag{5.43}$$

Finally the point probability $Pr(\mathsf{F} \mid \mathsf{F}')$ is given by Eqn. 5.44.

$$\mathrm{Pr}(\mathrm{F} \mid \mathrm{F}') = \sum_{i,j} m_{ij} \tag{5.44}$$

5.5 Truth Maintenance

5.5.1 Origins of Truth Maintenance

Truth maintenance systems (TMS) were developed to support the use of non-monotonic reasoning in problem solving. This type of reasoning may be appropriate when knowledge of a problem in incompatible and default assumptions must be made to enable a solution to be found, when the universe of discourse is changing or when temporary assumptions are used to test a possible solution (Frost, 1986). The truth maintenance concept is based on the use of belief values which, unlike truth values, are subject to alteration and revision in the light of new evidence. TMS are designed to be used by deductive systems to maintain logical relations among beliefs, to modify the belief structure when premises are changed and to use the logical relations to trace the source of contradictions or failures, leading to more efficient backtracking (McAllester, 1978).

The development of TMS stemmed from Stallman and Sussman's work, (Stallman & Sussman, 1977) which aimed at improving the behaviour of chronological backtracking in combinatorial search problems such as electronic circuit analysis by recording dependencies as the search progressed – dependency directed backtracking (DDB), (Shanahan & Southwick, 1989).

There are two types of TMS. The earlier type, justification based systems (JTMS) such as those produced by Doyle (Doyle, 1979) and McAllester (McAllester, 1978), store as fundamental data the immediate justifications for inferences, maintaining a single consistent hypothesis and using DDB to restore consistency by rejecting an assumption when contradictions are discovered. These systems have several limitations:

- Only one solution can be considered at a time, alternative solutions cannot be compared
- The current choice set can only be changed by introducing a contradiction, which cannot be removed later so switching states is difficult
- Their machinery is cumbersome
- If some but not all of the inferences based on an assumption set have been derived when a contradiction is found, the work may have to be repeated later if the complete set of inferences is required (de Kleer, 1984).

The later assumption-based systems (ATMS), which were developed by de Kleer in an attempt to solve these problems, record the fundamental assumptions on which inferences rest, maintaining multiple self contained but mutually inconsistent sets of hypotheses or contexts (Shanahan & Southwick, 1989). However, they too have limitations:

- If only one solution is required they are hopelessly inefficient
- They may search regions of the solution space, which DDB would avoid
- Debugging is difficult; intermediate states represent pieces of many solutions, and it can be hard to tell which is causing problems (de Kleer & Williams, 1986).

The development of a combined system which was intended to have the advantages of both types and the disadvantages of neither, using DDB to provide the search strategy with a coarse focus and to handle control assumptions, and an ATMS to provide an additional level of discrimination and to handle non-control assumptions, is described in (de Kleer & Williams, 1986). ATMS has been implemented with some form of rating system to ensure that the most promising solutions are investigated first (Hinde et al., 1989; Provan, 1990).

5.5.2 Justification-Based Truth Maintenance Systems

The JTMS developed by Doyle is generally considered to be the first true TMS. It operates by keeping track of which statements, assumptions and hypotheses are currently believed 'IN' and which are not currently believed 'OUT' (Doyle, 1979).

Doyle's JTMS employs two data structures: nodes, which represent beliefs and justifications, which represent reasons for beliefs. Each node has one or more justifications associated with it. A node is IN if and only if at least one of its justifications is valid. There are two different types of justifications, support-list justification and conditional-proof justification. Support list justifications have two part: an in-list containing nodes used in the derivation of the belief, all of which must be IN for the justification to be valid, and an out-list in which all the nodes must be OUT for validity. The out-list is used to allow assumptions to be retracted. If the out-list of an assumption A contains the node notA, the assumption will be retracted automatically if it leads to a contradiction (Norman, 1987). Conditional-proof

justifications are used when the status of the node depends on the validity of a hypothetical argument; they have three parts, a consequent, an in-list and an out-list, and are valid if the consequence is IN whenever each node in the in-list is IN and each node in the out-list is OUT.
The JTMS maintains a single consistent context (the current set of IN models) by using DDB to restore consistency when a contradiction arises. The nodes which contribute to the contradiction are found by tracing through the dependency structure, one of them is chosen as the culprit and rejected, and all justifications which depend on this node are checked for validity (Shanahan & Southwick, 1989).

McAllester developed a simplified JTMS. His system allows propositions to have one of three truth values, true, false or unknown, and represents all logical relations between propositions as disjunctive clauses; this representation makes no distinction between antecedents and consequents, which simplifies the backtracking process (McAllester, 1978).

5.5.3 Assumption-Based Truth Maintenance Systems

The ATMS described in (de Kleer, 1986a, 1986b, 1986c) was designed to allow a problem database to contain unresolved inconsistencies, so that the problem solver could follow more than one search path through the solution space at once and compare alternative solutions with one another. It was also intended to increase the ease with which results obtained in one region of the space could be carried over into other regions, by recording derivations in the most general way possible.

ATMS nodes have a label, supplied by the ATMS, which determines the environments or contexts in which the datum holds by specifying the minimal sets of assumptions from which it can be derived. A premise has an empty label, the label of an assumption specifies a single assumption set which contains only the assumption itself. Nodes also have justifications supplied by the problem solver giving the parent nodes from which they were derived.

A special node is used to represent falsity. The assumption sets specified for this node are 'no good' sets – sets from which inconsistencies have been derived. These sets are used to partition the space into self-consistent environments, and thus to ensure that inconsistencies are not propagated. When computing a node label, the system checks the assumption sets and removes any which contain 'no good' sets.

Architecture of the ATMS in the Loughborough System

The TMS works around the concept of a blackboard containing entries (concept of the blackboard system and its architecture is described in chapter 6). Entries given to the blackboard by users as specifications or requirements are in the form of assumptions, with associated ratings which specify how feasible or desirable the assumptions are felt to be. When a calculating engine such as the English engine takes a number of entries and produces a result from it, then this result is called a consequence of those entries, and the list of assumptions that led to the consequence is called the assumption base. Assumptions are initial defeasible entries, whereas initial indefeasible entries are facts. In the Loughborough system an engine can derive a consequence in two key ways; necessarily and possibly. A necessary assumption is one where the assumption base could only lead to that result through processing by the expert, and a possible assumption is one where more than one outcome is possible even if there is only one outcome delivered by the expert. The engine must also specify how feasible any outcome of a possible result is, to give it a ranking compared to other possible consequences of that assumption base. All the truth maintained agents in our system are Assumption Based in that each entry can stand without reference to its derivation path, only the assumptions which underpin its validity are needed (Hinde & Bray, 1992).

In order to understand how natural language can be interpreted using ATMS, let us describe the internal representation used in the ATMS. This should give a feel for the amount of data that may need to be stored when many assumptions are used to solve a large problem. The format of the entries is:

(*tag*, *entry*, *assumption bases*)

tag is a unique tag to distinguish entries from one another and to provide a reference for building assumption bases.
entry is the actual entry.
assumption bases are the lists of assumptions which underpin the entry, or justify it.

The statement:

{[], user, possible, [a= 1]} would result in the following entry being made:

(1, a = 1, [[1]]) This is a self justifying assumption. The reading of this is that "a=1" is true if entry 1 is true, i.e. if "a=l" is true. It stands on its own but may be contradicted.

(2, b = 0, [[2]]) This is also self justifying.

(3, c = -4, [[3]])

(4, a*x2 + b*x + c = 0, [[4]])

These may be presented to an algebraic equation solver which could deliver, as possible answers, the two entries "x=2" & "x=-2".

(5, x = -2, [[1, 2, 3, 4, 5]]) This is a partially self justifying assumption, i.e. a possible derivation of entries 1-4.

(6, x = 2, [[I, 2, 3, 4, 6]]) As is this.

(7, x > 0, [[7]]) This eliminates the entry "x = -2" from any environment containing assumption 7.

(0, false, ([[0],[1,2,3,4,5,6],[1,2,3,4,5,7]]) This is the entry that declares 5 and 6 are inconsistent in the context of 1,2,3 & 4 etc. If we were able to state that 5 & 6 are inconsistent in all possible worlds then the assumption base of our false entry would be [[0],[5,6],[1,2,3,4,5,7]]. This results in shorter assumption bases.

5.6 Conclusion

In this chapter we have examined some important uncertainty handling theories, ranging from probability theory to mass assignment to fuzzy set theory. It is important to remember that these theories do not stand independently from each other, rather they are all linked by the fundamental mathematics underneath.

It is important to see that fuzzy logic brings a linguistic perspective to human computer interaction methodologies, and natural language plays an important part in our managing uncertainty. Finally a note to mention the importance of fuzzy numbers in particular the use of fuzzy hedges as an important component of this thesis for processing natural language descriptions of faces. We have also looked at TMS especially ATMS which forms an integral part of the Natural Language Interface in interpreting natural language description of faces.

Chapter 6

Interpreting Natural Language and Translating Linguistic Data.

Abstract

This chapter looks at Natural Language Processing (NLP) within the domain of AI. We start with a brief description of natural languages, the various areas of study connected with natural language processing. We move on to inspect the anatomy of language, its orthographic structure, grammar and components of grammar. We have discussed computational tools such as Parsing, Prolog, Echo and the Truth Maintained Blackboard systems to interpret natural language descriptions of faces. We finally describe how the interpreted linguistic data is translated to numeric parameters.

Keywords: Natural Language Processing, Universal Grammar, Generative Grammar, Syntax, Semantic, Morphology, Phonology, Parsing, PROLOG, Echo, Blackboard and TMS.

6.1 Introduction

Writing a letter, reading a newspaper, having a conversation - the every-day written and spoken language of such activities is called natural language to distinguish it from artificial, made-up languages like programming languages. For over 30 years, researchers have studied how computers can be programmed to understand and generate written text and spoken utterances. The study area has been called natural language processing (NLP) or computational linguistics, though these terms tend to be associated with text processing rather than speech processing.

These days, NLP research is conducted at many universities and in the research laboratories of large companies, and there is a growing number of commercial NLP products (Obermeier, 1989) such as machine translation systems (see Hovy, 1993) and natural language interfaces (Sijtsma & Zweekhorst, 1993).

The study of natural language is frequently decomposed into a number of smaller, partially overlapping study areas: phonology, morphology, syntax, semantics and pragmatics. The scope of each area is described below, together with problems that each area presents for NLP. The descriptions of areas are adapted from (Crystal 1992). Language is a medium: its auditory form is spoken language, its visual form is written language. This view of language is briefly described below.

- Phonology
- Morphology
- Syntax
- Semantics
- Pragmatics
- Language as a Medium

6.1.1 Phonology

Phonology is the study of the sound structure of language. Sounds are organized into a system of contrasts, and analyzed in terms of phonemes, distinctive features, or other such phonological units according to the theory used. A phoneme is the minimal unit of the sound system of a language. Some languages have as few as 15; others have as many as 80. No two languages have the same system of phonemes. Distinctive features are used either to define phonemes or as an alternative to the notion of phoneme. Example pairs include +nasal and -nasal, and +voice (voiced) and -voice (voiceless). Nasal sounds are produced when there is complete closure in the mouth and all the air thus escapes through the nose, as in the `n-' sound of `nasal'. Voiced sounds are produced while the vocal cords are vibrating, e.g., the `b-' sound in `bin'; voiceless or unvoiced sounds are produced when there is no such vibration, as in the `p-' sound of `pin'.

Problems include the ratio of noise to data, the varying speech rates within and across individuals, and co-articulation. Co-articulation takes place when the articulation for two or more sounds takes place in the vocal tract, e.g., the `sh-' in `shoe' is normally pronounced with lip-rounding in anticipation of the `-oo' sound.

6.1.2 Morphology

Morphology is the study of the structure of words, especially through use of morphemes. Morphemes are commonly divided into free forms (morphemes which can occur as separate words) and bound forms (morphemes which cannot occur in this way, e.g., `unselfish' consists of three morphemes, `self' which is a free form, and `un-' and `-ish' which are bound forms.

A major morphological problem is ambiguity: the suffix `s', for example, can indicate the plural of a noun or the present tense of a verb. Another problem is exceptions, for example, the plural of the noun `foot' is `feet' (not `foots').

6.1.3 Syntax

Syntax is the study of how words are combined to form sentences in a language. Syntactic structures (or constructions) are analyzed into sequences of syntactic categories (or classes). The sequences are established on the basis of syntactic relationships that linguistic items have with each other in a construction, e.g., “tall people” is generally analyzed into a noun phrase consisting of an adjective “tall” and a noun “people”. Linguists have designed grammars for many languages. A grammar is a system of syntax and inflections for a language. Inflection is the change words undergo when used, for example, in the plural (“mouse” and “mice”) or in the past tense (“fly” and “flew”). Parsing refers to the assignment of syntactic categories and structures in single sentences. Parsers often but not always use grammars. The following are some major problems for syntactic processing.

Structural ambiguity occurs when a sentence construction can be assigned several possible structures or combinations of elements, e.g. in "Jane saw the man in the park with the telescope" the prepositional phrase "with the telescope" could be attached to either "Jane saw" or "the man in the park."

Unbounded or long distance dependency is a relationship between two syntactic components of a sentence in which the related constituents are not required to be within some bounded distance of each other. The dependency, which may extend over one or more clause boundaries, usually involves an empty noun phrase constituent called a "trace" which is co-indexed with another noun phrase appearing earlier, as in "Show me the report that Nick wanted Dan to write" where, although "report" is the object of the verb "write", there is no explicit object following the verb.

6.1.4 Semantics

Semantics is the study of meaning in language. It contains a number of branches including philosophical semantics and linguistic semantics, which have both been studied in NLP. Philosophical semantics studies relations between linguistic expressions (like sentences) and the entities in the world to which they refer, and the conditions under which such expressions can be said to be true or false. Analysis is performed with logical systems. Linguistic semantics studies the semantic properties of natural languages using a variety of linguistic constructs. Among the phenomena studied within semantics are the following.

Lexical ambiguity refers to a semantic property of words that they can have multiple senses or meanings, e.g., the word "crook" has different senses: it can mean a thief, a bend, or a shepherd's stick. Resolution of lexical ambiguity is required for understanding sentences that contain ambiguous words like "crook", e.g., in "The crook stole a diamond ring," the thief sense is meant.

Similarity or paraphrase refers to a property of sentences that different ones can have the same (or very similar) meanings, e.g.,

"Give me the Western region financial performance for July,"
"Give me the July financial performance for the Western region,"
"Give me the financial performance for July for the Western region" and
"Give me the July Western region financial performance" (McFetridge, 1991).

The problem is recognizing when two sentence are paraphrases.

Reference is a relationship of identity between linguistic units, e.g., between a pronoun and a noun or noun phrase. Pronouns are of various kinds, including definite pronouns like 'it' and 'them', personal pronouns such as 'I' and 'you', reflexive pronouns like 'myself' and 'yourself', and relative pronouns such as 'who', 'whom' and 'that'. The problem is resolving reference, i.e., connecting a pronoun with the noun or noun phrase to which it refers.

Reference can occur across sentence boundaries, and can be backwards or forwards. Anaphora (or back-reference) is reference to an earlier part of a discourse. Cataphora (or forward reference) is reference to a later part of the discourse. The difference can be seen in a two different two-sentence discourses where the first sentence each time is "John is at home." There is an anaphoric reference to John when the second sentence is "If he is not drunk, Peter will be surprised" versus a cataphoric reference to Peter when the second sentence is "If he is not drunk, Peter will take me there" (Strzalkowski & Cercone, 1986). Traditional syntactic solutions have been able to treat only simple classes of anaphora and only occasional inter-sentential references.

6.1.5 Pragmatics

Pragmatics is the study of the communicative use of language, particularly the structure of conversations and dialogue: how participants take turns in conversations, how speakers use knowledge of communication (e.g., about the context in which language is used), and the effects their use of language has on other participants. Pragmatic problems include the following.

Presupposition is the information assumed by a person when using language and which is as the centre of a person's communicative interest, e.g., "There is unrest in Macedonia" presupposes the existence of (a country called) Macedonia.

Conversational repair refers to the attempt made by participants in a conversation to make good a real or imagined deficiency in the interaction (for example, a mishearing or misunderstanding)" (Crystal, 1992). A major problem here is working out which participant is wrong or mistaken and hence should have their conversation (and understanding) repaired.

Indirect meaning refers to the communicative purpose of a piece of language which does not directly reflect its surface form. The true communicative purpose is understood from examining the context in which the piece of language was used, for example, "It's hot in here" looks like an assertion, but in the right context - spoken to someone standing by a window - might be a request to open the window. Likewise, "Can you pass the salt?" looks like a question, but can also be a request to pass the salt if said when sitting at a table and spoken to someone closer to the salt than you are!.

6.1.6 Language as a Medium

The NLP community has responded to the growing interest in multimedia systems by investigating how to integrate natural language (in typed, handwritten and spoken forms) with other kinds of multimedia input such as the use of graphics, input devices like menus and data gloves. Similarly, there have been studies of generating coordinated multimedia output in which natural language is mixed with diagrams and so forth.

6.2 Anatomy of Language

"*A standard assumption is that a language consists of two components: a lexicon and a computational system. The lexicon specifies the items that enter into the computational system, with their idiosyncratic properties. The computational system uses these elements to generate derivations and Structural Descriptions, SD's. The derivation of a particular linguistic expression, then, involves a choice of items from the lexicon and a computation that constructs the pair of interface representations.*" (Chomsky, 1992)

A language can be seen as an infinitely large set of sentences. Each sentence is characterised as a *well-formed string* over a finite vocabulary of *symbols*. For sake of simplicity, these symbols can be regarded as words, though this view is not quite correct. *Well-formed* means that the form of the string – i.e. the way the symbols are put together – does not violate certain criteria specified in *rules of formation* which are contained in a grammar.

The notion of formal grammar can be described here but in a form which is very rigidly defined. A formal grammar *G* is a quadruple <V_N, V_T, P, S>, where:

V_T is a finite set of *terminal symbols*. If the grammar generates human language these symbols coincide more or less with the words of the language.

V_N is a finite set of *non-terminal* symbols. Sometimes they are also referred to as *variables* (Hopcroft and Ullman, 1969). In linguistic applications they correspond to categories. It is their presence in the rules that allows a grammar to express general wellformedness conditions.

P is a finite set of rules called *productions*. They are of the form '$\alpha \rightarrow \beta$' (α rewrites as β) where both α and β stand for strings of elements of V_N and V_T.

S is the *starting symbol* or *root.* S is an element of V_N and has to occur at least once on the left hand side of the rewrite arrow in the productions (in the place of α)

A grammar *G* generates a language *L(G).* There exist several different types of grammar, depending on the form of the strings α and β in the rules (i.e. exactly what elements of V_N and V_T occur in α and β). The type of *G* determines the type of *L(G)*: grammars of a certain type generate languages of a corresponding type.

The term grammar is systematically ambiguous between two idea, first is an internalised grammar and the other is linguist's grammar.

An internalised grammar is the internalised knowledge of a native speaker of English that enables him or her to make judgements about language data such as make grammaticality judgements i.e. differentiate a grammatical sentence (the man depends on his car) from an ungrammatical sentence (the man depend his car). Recognise ambiguous utterances and identify the degree of ambiguity and recognise sentences that are synonymous or partial paraphrases. *John and Bill are identical* is synonymous with *John is identical to Bill.* If one sentence is true, the other must be true; and if either sentence is false, the other must be false. The sentence *John knows all the irregular past-tense forms of all French verbs* is a partial paraphrase of *John knows all the forms of all French verbs.* If the latter is true, the former must be true. But if the former is true, the latter may or may not be true.

The linguist's grammar, called a generative grammar, is the logical or computational model constructed by a linguist using computers, programs, logical notations, and other descriptive tools.

Grammar can be segmented into parts, called levels. There are 5 basic levels of linguistic structure (Dougherty R.C., 1994).

- Discourse Level: A *discourse* is sentences or utterances exchanged between two persons (e.g. question/answer pairs).
- Paragraph Level: A *paragraph* is sentences joined in a sequence with sentence separators (period, question, or exclamation marks) between them. Adverbs (*therefore, hence, thus, nevertheless...*) can occur to show the logical connectedness among the sentences.
- Sentence Level: A simple *sentence* is a full proposition consisting of a subject and a predicate. A *complex sentence* consists of two or more simple sentences joined by a coordinating conjunction (*and, but....*) or a subordinating conjunction (*although, after, that...*).
- Phrase Level: A *phrase* consists of a lexical item (*noun, verb, adjective...*) and its associated modifiers, e.g. *the, a...* precede nouns; *very, too...* precede adjectives; *will, can...* precede verbs. A phrase is always defined by the type of lexical head: noun, verb, adverb...
- Word (lexical and grammatical formative) Level: A *word* is anything in a sentence that has white spaces on either side. An *orthographic string* is a written series of words. The grammar contains each possible word in its lexicon.

Chomsky's *universal grammar* (1986a; 1986b) defines the structure of levels, the number of levels, and interrelations among levels. Factoring, a part of *universal grammar*, plays a role in defining the technical terminology used to represent information at each level and in relating the technical terminology used at one level to that used at another. Parsing, a derivational mechanism of a grammar, relates to the processes by which a particular sequence of orthographic symbols (or words) in English is assigned a specific structure at each level. A question in universal grammar (factoring) is: What is a noun phrase? The answer would be to define *noun phrase* in terms of the types of words that compose it: *determiners* (the, a), *nouns* (girl, thought, *adjectives* (tall, red). A question in English grammar (parsing) is: what is *the tall grass*? The parser might answer that *the tall grass* is a noun phrase with the structure *determiner* + *adjective* + *noun.*

Figures 6.1 to 6.4 demonstrate how the levels of grammar define the structure of language for the sentences "*The cat eats the mouse*", "*did the cat eat the mouse*", "*the mouse was eaten by the cat*", and "*was the mouse eaten by the cat*".

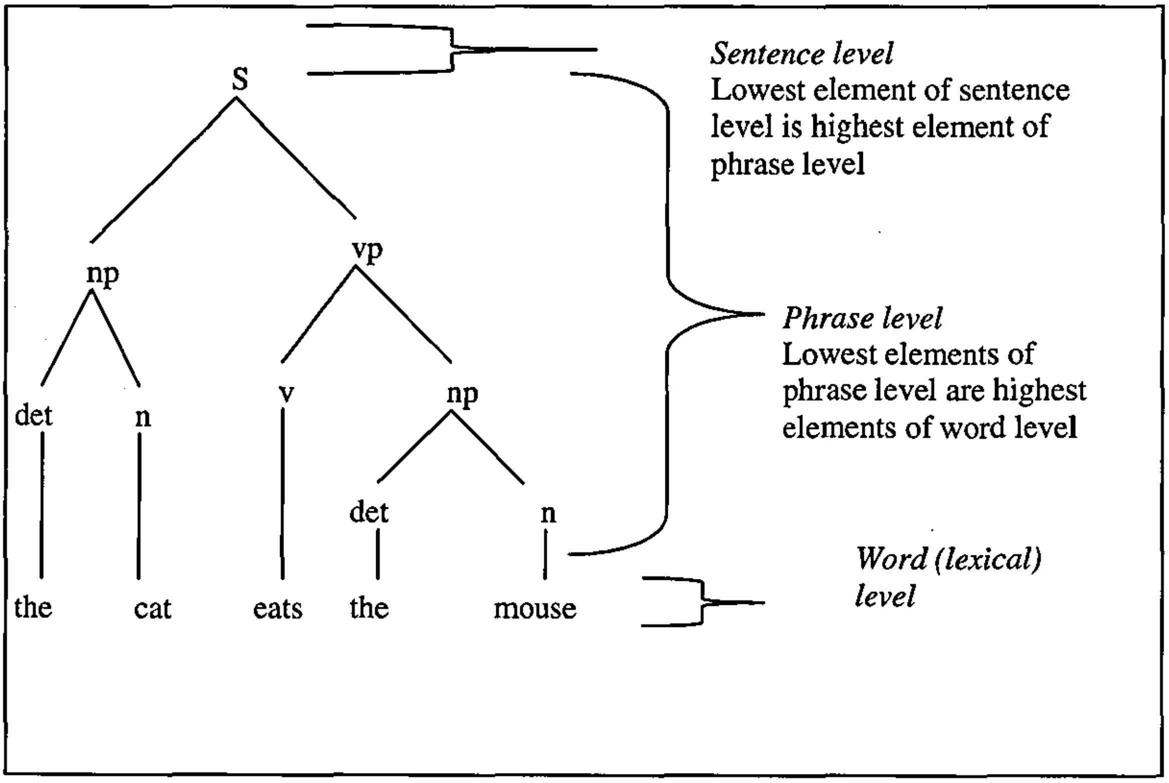

S
np
vp
det
n
v
np
det
n
the
cat
eats
the
mouse
Sentence level
Lowest element of sentence level is highest element of phrase level
Phrase level
Lowest elements of phrase level are highest elements of word level
Word (lexical) level

Figure 6.1: Phrase marker for sentence "*the cat eats the mouse*"

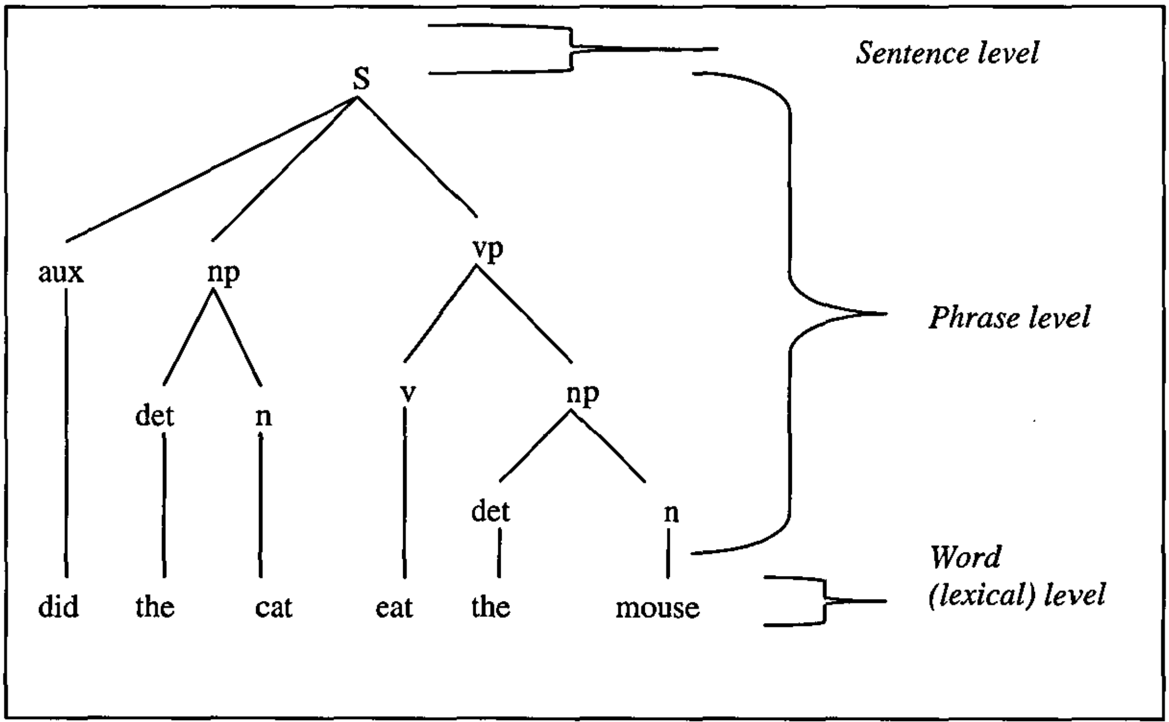


Figure 6.2: Phrase marker for sentence "did *the cat eat the mouse*"

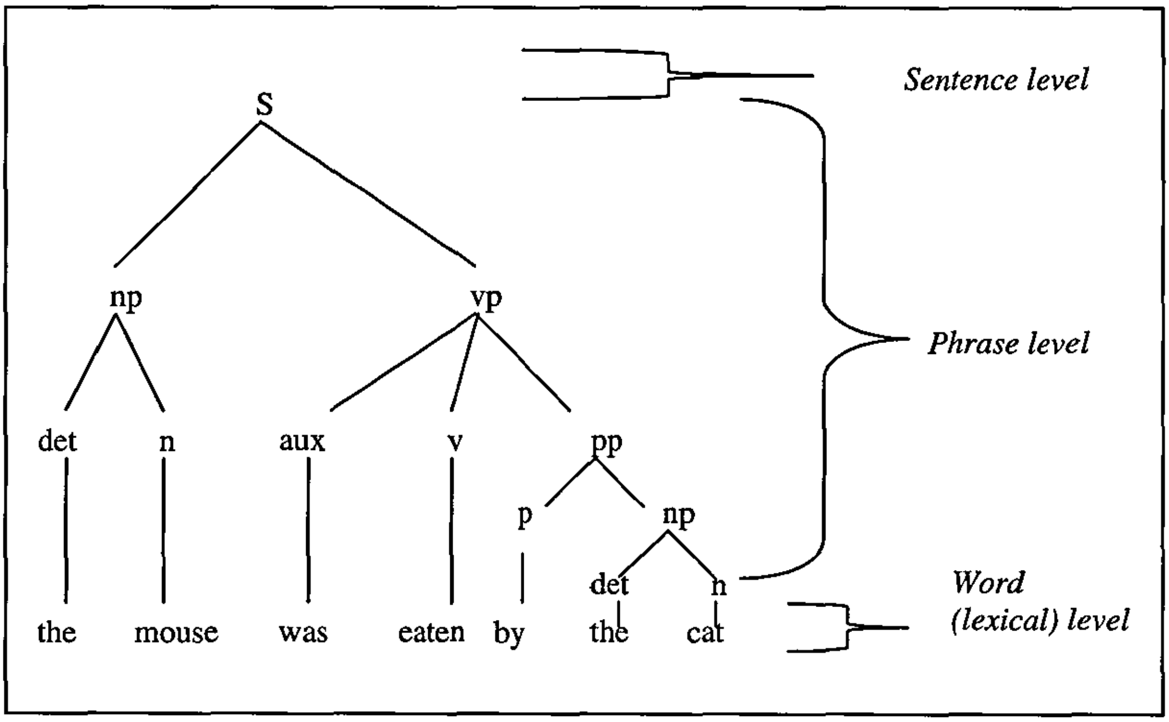


Figure 6.3: Phrase marker for sentence "*the mouse was eaten by the cat*"

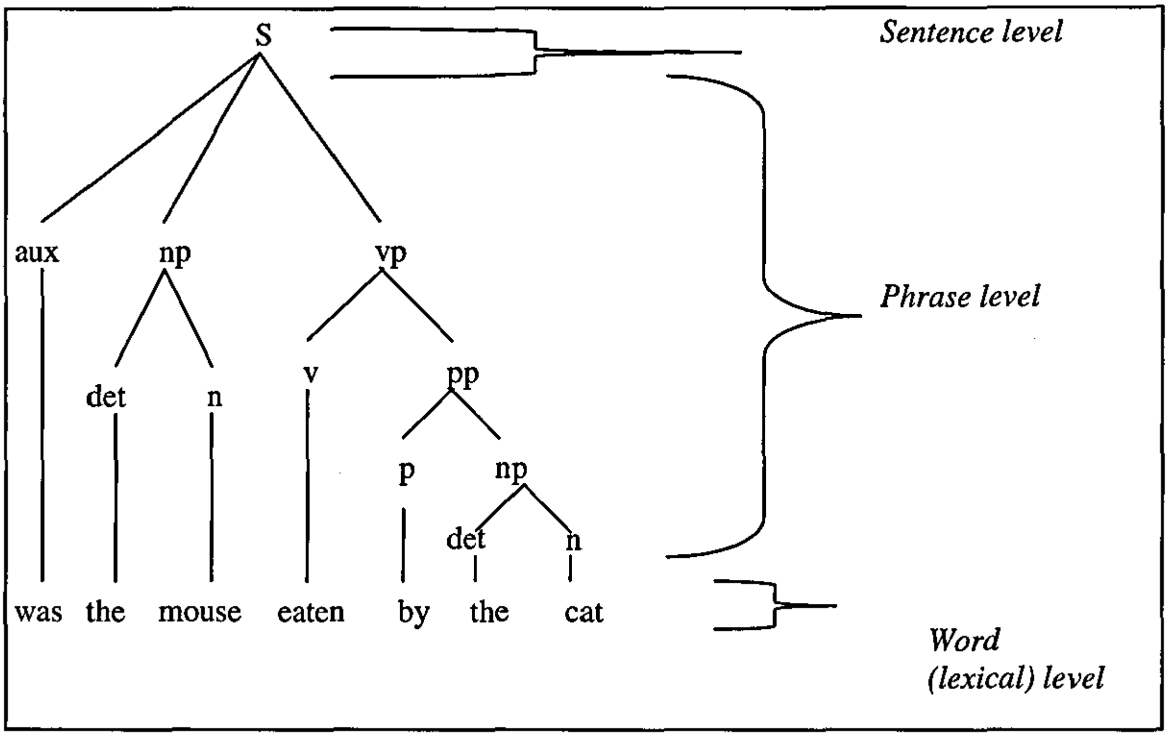


Figure 6.4: Phrase marker for sentence "was *the mouse eaten by the cat*"

6.3 Parsing

Consider the sentence "*the cat eats the mouse*", and its underlying structure:

$(_S(_{NP}(_{Det}The)(_N cat))(_{VP}(_V eats)(_{NP}(_{Det}the)(_N mouse))))$

also represented as a phrase marker graph in Figure 6.1. Although generative grammar may generates both a sentences and its underlying structure as shown in Figures 6.1 to 6.4, the grammar alone however does not offer any indication on how the link between the sentence and the structure gets established. In order to decide which structure ought to underlie a given sentence such as "*the cat eats the mouse*" a *procedure* is needed that will not just recognise the sentence but also discover how it is built. The execution of this procedure is called *parsing* and the thing that executes it is called a *parser* (King M., 1983).

So, parsers essentially do two things. On the one hand, when presented with a string, they have to recognise it as a sentence of the language they can parse. In this respect, parsers have built in recognisers. On the other have to assign to that sentence a structure which they have to output. This implies that parsers must reply on linguistic information as contained in a grammar with at least strong generative capacity, whereas recognisers, because they do not output structure, can be built referring to grammars with weak generative capacity.

A parser usually proceeds by taking a string of symbols (the input sentence) and applying a rule to it, which mostly comes down to rewriting a bit of the string. For example, the string 'ADC' by applying the rule 'B → D' (rewrite 'B' as 'D') and 'ADC' into 'AdC' according to a rule 'D → d'. The strings 'ABC', 'ADC' and 'AdC' are called *derivations*. The string 'ADC' is *directly derived* from 'ABC' since it is the result of the Application of a single rule to 'ABC'. 'AdC' is *indirectly derived* from 'ABC' as more than one rule has to be executed to link up both strings. At each step the parser can output some structure. A sentence has been parsed when we know all the structures that can be assigned to it according to the set of rules available.

6.3.1 Parsing Strategies

Let us assume that a parser works by referring to rules which reflect linguistic knowledge. Dissociated from a parser that uses them, such a set of rules can potentially be executed in many different *orders* when assigning a structure to a sentence. Each different order corresponds to a different *parsing strategy* and parsers are classified according to the strategy to which they adhere.

Two criteria for looking at parsing strategies are considered standard and occur frequently in the literature (Roeck A, 1983; Dougherty R, 1994). The first one focuses on the linguistic structure the parser outputs for the string it parses and takes into consideration whether that structure gets built starting from the input string (data) – in this case the parser works *bottom-up* – or from the starting symbol (the symbol corresponding to the *axiom* of the grammar and which always *has* to be present as the *root* of the tree in any linguistic structure denoted by the symbol 'S') – in which case the parser works *top-down.*

The other criterion for classifying parsing strategies can be better explained by means of an example. Consider a set of rewrite instructions:

1a. S → AB

b. S → CD

c. A → a

d. B → b

e. C → c

f. d → D

For a given set of rules, it is possible to construct a scheme of all possible derivations those rules can yield. Considering that the beginning symbol, 'S', is present two of the rules listed under 1 can be executed: rule 1a resulting in the derivation 'AB' and rule 1b yielding 'CD'. If rule 1a gets executed, a similar situation arises. To the string 'AB' rules 1c and 1d apply, respectively returning the strings 'aB' and 'Ab'; etc. Following this reasoning, all possible sequence of derivations that a given set of rewrite instructions allows can be discovered. The result is usually represented in the form of a tree as in Figure 6.5.

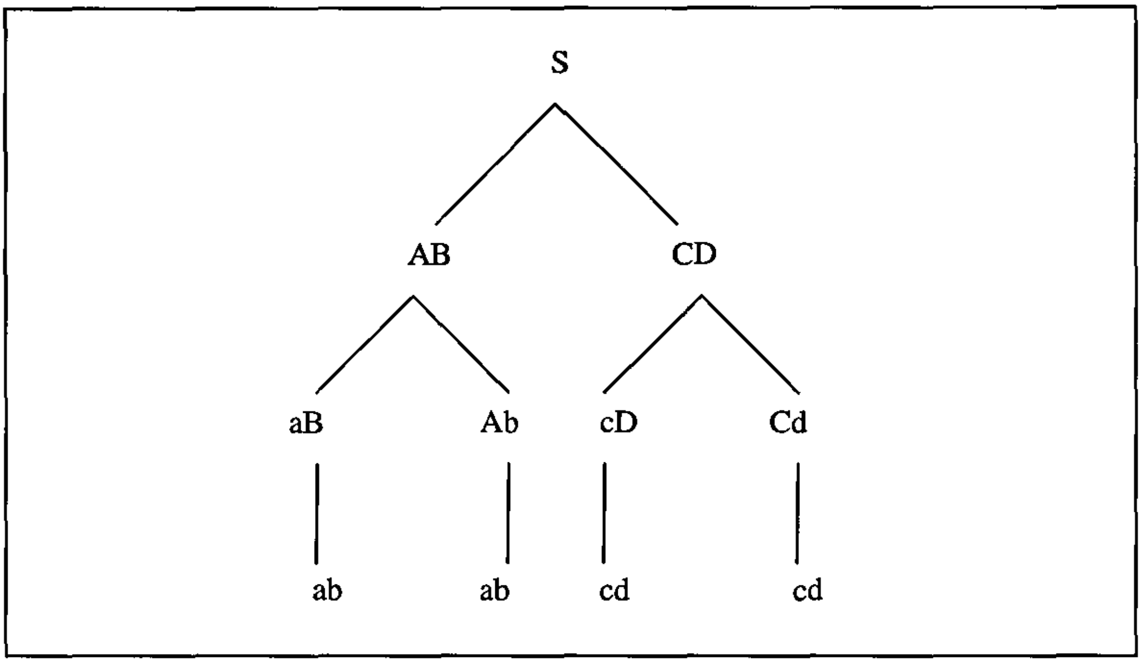


Figure 6.5 Graph showing the sequence of derivations possible from rule in 1

The tree in Figure 6.5 shows all possible sequences of derivations that can result from the rules in 1. each node in the tree represents a point in the procedure where a choice presents itself in terms of different rules potentially to be executed on the same sentential form. The *leaves* of the tree represent the sentential forms to which no further rules apply. In this case their content corresponds to those strings which can be parsed according to the rewrite instructions in 1 ('ab' and 'cd').

The tree shown if Figure 6.5 is not the same that linguists use to represent linguistic structure, and which expresses how the parts of a sentence fit together such as those in Figure 6.1 to Figure 6.4. The tree in Figure 6.5 gives all possible sequences of derivations by which a linguistic structure can be constructed. The classification of parsing strategies on the axis 'depth-first' versus 'breadth-first' is based on this kind of tree. The sections 6.3.1.a and 6.3.1.b give further details about these two basic criteria for characterising parsers (top-down versus bottom-up and depth-first versus breadth-first).

6.3.1.a Top-down versus Bottom-up Parsing

The following examples will explain how a very simple and frugal top-down and bottom-up parser assigns a structure to sentence 2

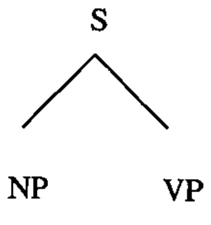
2. The cat eats the mouse

provided both have access to the same set of rewrite instructions listed in 3:

3a. S • NP VP
b. NP • Art N
c. VP • V NP
d. VP • V
e. V • eats
f. N • cat
g. N • mouse
h. Det • the

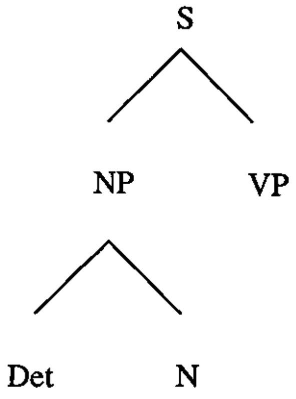
Top-down parsing. Top-down parsers always start with the starting symbol ('S'), find rules that apply to it and expand it. In this example the only rule available to do so is 13a. The result of the execution of 3a is the structure in 4:

4.



Two new nodes appeared. The parser first looks whether any of these two nodes is a *terminal* - i.e. whether they contains symbols that would belong to V_T in the corresponding grammar. If so, those symbols will be checked against the string that is being parsed (sentence 2). If not, as is the case here, the parser further expands the first non-terminal node - in the example the 'NP' node. Rule 3b applies and is executed, yielding the structure 5

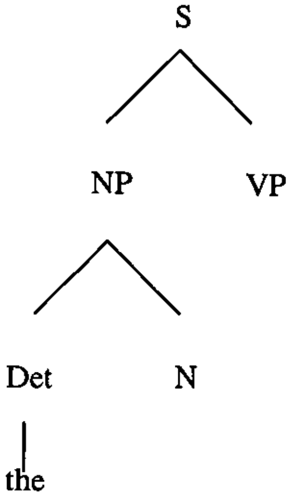
5.



Again, none of the newly constructed nodes is a terminal, and again the left-most non-terminal gets expanded. This way of proceeding is repeated and after the application of rules 3h, 3f, 3c, 3e, 3b, 3h and 3g the string to be parsed is actually met. No further rules apply and the parse, outputting the structure shown in Figure 6.1, succeeds.

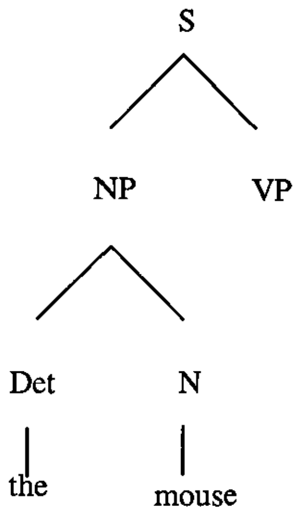
But things do not always turn out to be as straightforward as that. Take some steps back and imagine the parser has applied, to begin from the starting symbol, rules 3a, 3b and 3h yielding the structure

6.



The next non-terminal to be expanded is the node labelled 'N'. Before it was happily assumed rule 3f applies next, but there is no reason why rule 3g should not be executed instead. The parser then builds 7.

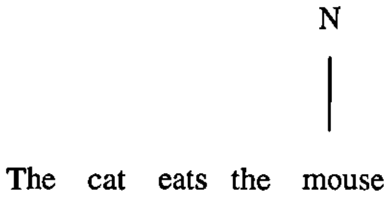
7



In that case, the parser will find out when checking the newly found terminal against the data that 'mouse' does not correspond to the symbol it finds in the appropriate position in the sentence. It discovers its mistake and now has to do two things. First, it has to remember that 3g was not the right rule to apply in the previous state (illustrated in 6); then it has to re-establish the situation occurring before the application of 3g and try and find another rule to rewrite 'N'. The jargon refers to this move backwards as backtracking or back-up. The necessity for backtracking follows from the fact that, during the execution of the rewrite rules, a situation arose in which more than one option was available as to what to do next. This is the simplest case of non-determinism in a procedure. The set of rewrite rules in 3 is non-deterministic because whenever either of the non-terminals 'VP' or 'N' are encountered in a derivation more than one rule presents itself as a candidate for execution (for 'VP' 3c and 3d, for 'N' 3f and 3g), each resulting in a different structure. Top-down parsers are sometimes called 'hypothesis driven' because they explore a particular derivation in the belief that it is the right one until they meet failure or success.

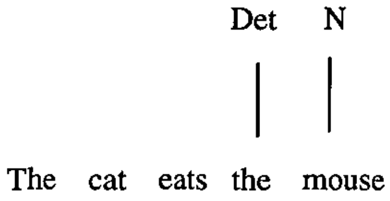
Bottom-up parsing. As opposed to top-down parsers, a bottom- up parser starts to work on the input string itself and reduces it to the root 'S'. It takes a sentence, replaces the words (terminal symbols) by their categories, and strings of categories by other categories. In order to do so it must took at the symbols on the right hand side of the rewrite rules and reduce them to the category written on the left hand side. Again, sentence 2 will get a structure assigned to it according to the grammar expressed in 3a-h, for instance by first applying 3g, yielding

8.



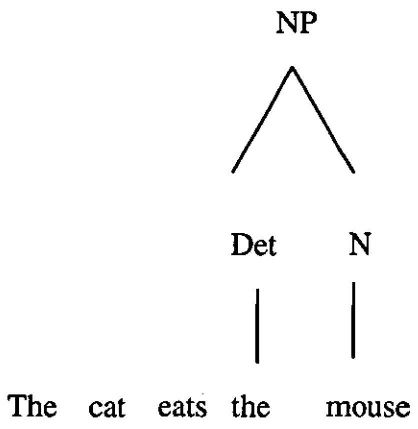
No rule applies to an 'N' node, either alone or combined with a string of terminals, so the parser looks at the next terminal, 'the', and reduces it according to rule 3h resulting in

9.



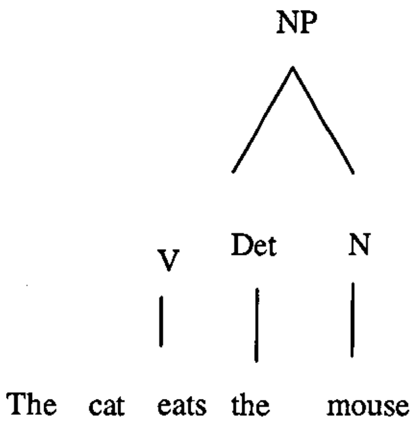
At this point there is a rule available that combines the categories 'Art' and 'N' reducing them to an 'NP' (rule 3b), as illustrated in 10.

10.



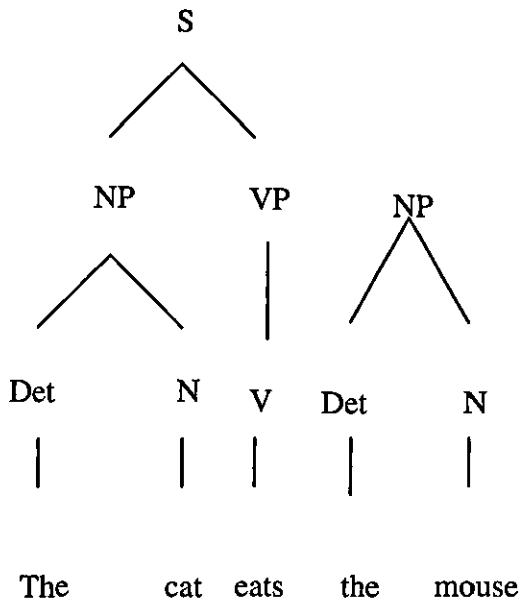
Then the terminal 'eats' is used by rule 3e, after which 3c, 3f, 3h, 3b and 3a are executed. With this strategy also, there is a need for backtracking. Imagine the intermediate structure after the execution of rule 3e in the above rule sequence, as pictured in 11:

11.



'V' can be reduced to 'VP' by rule 3d, thus leaving out the 'NP' and resulting, after the application of 3f, 3h, 3b and 3a in

12.



Structure 12 is illegal because, in spite of the fact that the root of the tree has been reached, there is a part of the structure that hangs loose (the rightmost 'NP'). In this case too the parser has to backtrack and remake a choice at an earlier stage.

It may seem odd that the parser just described parses a sentence starting from the right and working its way to the front of the string. Clearly language does not work like that, and this bottom-up parser can be argued to be psychologically not accurate on those grounds. Still, from the point of view of parsing and in terms of results obtained, a parser that starts from the left is equivalent to one that starts from the right if both refer to the same grammar, even if they follow different sequences of derivations. It may be useful to know that this right-to-left opposition has nothing to do with what is known in the literature as a right or left parse. A right parse is always the result of a bottom-up parser, which reduces sentential forms by referring to symbols found on the right hand side of rules. A top-down parser executes a left parse, deriving sentential forms by expanding the symbol found on the left hand side of rules.

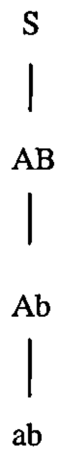
Similarly, top-down and bottom-up parsers which refer to the same grammar are also equivalent because they assign the same structures to the same sentences according to the same linguistic information - as shown by the examples. Their differences, besides the fact that they follow different sequences of derivations, have to be expressed in terms of memory needed and computing time involved.

6.3.1.b Depth-first versus Breadth-first Parsing

If we look at the rewrite instructions in 1 and the corresponding derivation tree in figure 6.5. A similar tree can be drawn for all sets of rewrite instructions, picturing all possible sentential forms they allow to be constructed. Such a tree can be approached in two different ways: one concentrating on its vertical and the other on its horizontal aspect. These two distinct viewpoints result in a criterion for classifying parsing strategies.

Depth-first parsing. Let us consider the vertical aspect of the derivation tree in Figure 6.5 and pick out one single vertical path linking the root with the sentence to be parsed. E.g. for sentence 'ab', contained in a leaf node, one could conceivably pick the path as in 13.

13.



This path, like any other vertical path in the tree, gives a sequence of sentential forms. The word sequence has some importance here. It indicates that each sentential form is the result of the application of one single rule to the result of the execution of another single rule or to the root. If to a particular derivation several rules could potentially apply, only one rewriting possibility is retained. The other options are expressed in other paths of the tree and can not be traced along a single vertical path. Any parser that follows a sequence of sentential forms as can be represented on a single vertical path in a derivation tree is called a depth-first parser. Both the top-down and the bottom- up parser described in 6.3.1.a belong to this type, the first starting the derivation at the top of the derivation tree, the other at the bottom.

Figure 6.5 shows clearly that more than one path may link a same sentence with the starting symbol 'S' (for each sentence - 'ab' and 'cd' – there are two). Since a depth first parser explores only one path at the time it is possible that the path chosen from the beginning is not the right one. In those cases it becomes necessary that the parser be able to recover from its error by undoing the mistake (back-up: the parsers described in 6.3.1.a illustrated this). For this reason depth- first parsers are usually implemented with backtracking facilities.

Breadth-first parsing. But one can also look at a derivation tree while stressing its horizontal dimension and taking into consideration all nodes at the same level in the tree. For instance, the root of the derivation tree consists of a node bearing the sentential form 'S'. Two daughter nodes hang off this node, containing, respectively, the sentential forms 'AB' and 'CD', each being the result of the application of alternative rules to 'S'. A parser which, in such a case, indeed does build both alternative derivations simultaneously and, in the next step, again applies all possible rules to both results, is called a breadth-first parser.

Breadth first parsers apply all applicable rules to all sentential forms constructed; they explore the horizontal dimension of the derivation tree, exhausting all the choices which arise at the same time and taking them to their conclusion of either failure or success. This way of proceeding makes backtracking in case of failure superfluous: even if a derivation sequence resulting from a bad choice dies out, all successful alternatives being developed simultaneously will survive.

6.4 Interpreting Natural Language Sentences

There are numerous different ways of constructing a natural language interpretation system. However, two particular techniques for analysing sentences have proved to be popular. One of these involves the use of Augmented Transition Networks (AT'Ns), while the other is based on an algorithm known as an Active Chart Parser (ACP).

ATNs were introduced by Woods (1970). ATNs are based on a grammatical description of the target language in the form of a series of networks. Traversal of the appropriate networks is the central process involved in parsing sentences using an ATN. The results of the parse are stored in a series of special- purpose registers.

ACPs derive from the work of Kay (1967), Earley (1970) and Kaplan (1973). They are so called because they make use of a graph-like data structure known as a chart to build up the analysis of a sentence. They operate in association with a grammar in the form of a context-free production system, and offer a particularly efficient means of natural language parsing. the various components of an active chart parser are as follows:

- An initialisation of the chart
- A "fundamental rule" that combines an active edge with a passive edge.
- A control strategy (either top-down or bottom-up).
- A search strategy (either breadth-first or depth-first)

As this description indicates, active chart parsers can be of different types e.g. bottom-up and depth-first, top-down and breadth first etc (see section 6.3).

The technique used in this thesis to interpret natural language description of faces involved extending the Echo project. The Echo Project initiated in 1987 (Hinde, Lawson & Connolly, 1989; Hinde & Bray, 1992) was developed by Dr Chris Hinde using PROLOG to translate natural language (English) phrases into Structured Query Language (SQL). The system was originally designed to interface with a database such as Ingres.

However modifications were made to the program to cater for the demands of this thesis. The natural language processing engine of Echo does not take a simple parsing approach as many other natural language interfaces (such as 'The Intellect system' on IBM and DEC systems, 'SPOCK' and 'NATURAL LANGUAGE' for oracle databases) for three important reasons. Firstly parsing based systems are computationally quite heavy, owing to the combinatorial explosion problem. Secondly they have difficulties with poorly formed or ungrammatical phrases - which are of course exceedingly common in normal speech. Thirdly from a review of the development of such systems in the past, and their capabilities, it was obvious that no single method will suffice to quickly and easily interpret sentences that no human would have any difficulty with, ie. different techniques must be combined and used as appropriate.

For example consider the following sentences:
1. "Ann took the cat to the vet because she had injured her tail".
2. "The robber fell off the bank".
3. "The dusting of the new cleaner was not very thorough".

Any human can interpret all three without difficulty because of semantic knowledge. A computer system may find all three ambiguous.

Echo is modeled entirely in PROLOG which is a logic programming language that is unique among programming languages in that it has, built into the language:

1. A powerful pattern-matching algorithm, called unification,

2. A powerful backtracking search mechanism, and

3. Recursion.

These features are ideally suited for the type of domain-specific tool we are talking about. PROLOG patterns, called terms, are built from simple components, but can be arbitrarily complex. It is these that can be used to model the knowledge representation of the domain.

6.4.1 Echo Architecture

The ECHO system is based around a ranked-bid truth maintained blackboard system which is unique in its combination of features so as to enable us to incorporate various techniques (i.e. information sources). There is a precedent for using blackboard systems in natural language understanding with the Hearsay systems developed at Stanford University; however the use of a truth maintenance system is novel in this application and the ranked-bid system is a further development of blackboard architectures.

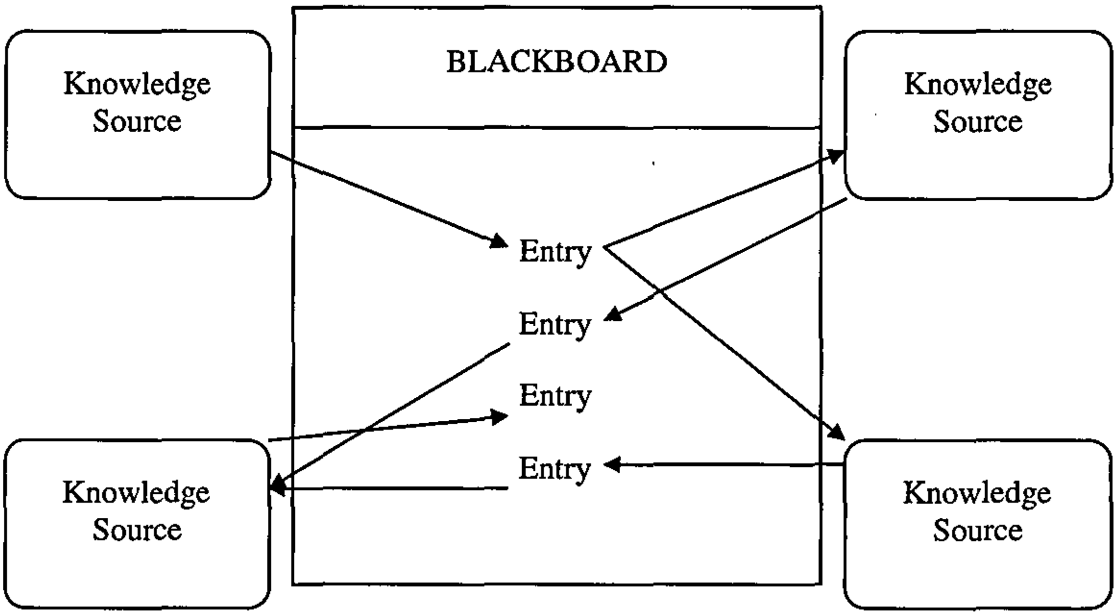


Figure 6.6 Structure of a Blackboard System

Blackboard systems use knowledge sources of various types to solve problems in a collaborative manner. Each knowledge source examines the blackboard for interesting entries that it can do something with. There is also a set of "facts" which if true could allow a rule or knowledge source to "fire". It is also possible to maintain several interpretations concurrently - for example of ambiguous sentences. A considerable amount of the work in developing the ECHO system has gone into designing the Blackboard management system and implementing the system's matching algorithm in a way to maintain efficiency (Hinde, Lawson & Connolly, 1989).

The normal cycle of activity of the blackboard system is as follows: the system controller determines which of the knowledge sources are capable of utilising the information currently held on the blackboard and determines which of the possible operations should be executed first, and then the selected operation is carried out. This produces changes in the information on the blackboard, so the controller is reactivated to assess the new situation and the cycle is repeated. The pre-conditions for each knowledge source, i.e. the information which they require, must obviously be specified so that the controller can determine which operations could be performed; some means must also be provided for assigning an order of priority to the operations, for example assessing the usefulness and reliability of the potential output (Jones and Millington, 1986).

The knowledge sources may offer many different types of knowledge or information. Many previous natural language understanding systems have been based on a syntactical analysis of the sentence followed by a thorough semantic analysis and finally the required action is formulated via consideration of the pragmatics. It is often the case though that humans can make a syntactically invalid statement which is clearly understood by the receiver; the statement "has blue eyes" has no verb and so is syntactically invalid but is interpretable by almost anyone. In any particular situation the blue colour of the eyes of an assumed object, in this case an earlier description of a face would be drawn from a previous context. Syntactic and semantic knowledge is important in determining the required action, but in the context of a facial description system, nothing is relevant unless it helps to resolve the required action. We have therefore adopted an approach primarily based on the content and structure of the sentence to guide the interpretation; and although the system incorporates an ACP, syntax is used only as and where necessary to reduce ambiguity.

6.4.2 Knowledge Sources

There are three major knowledge sources currently being used by the NLI under the Echo system:

- Description of a face by user (this includes any edit or amplification entry)
- Lexical Analysis
- Formation of descriptor list

A description of each follows.

User Descriptions Knowledge Source

The Blackboard is empty on start-up and so the users describe subsystem is the only one capable of being activated as it does not require any other entries to be present. The result is a new window called 'Describe' activated, from the Echo Menu Interface, where the user enters the description or loads a pre-defined description which is subsequently processed. The Blackboard is split into many sections corresponding to the various knowledge classes. The user description is entered as entry of type "phrase" with an appropriate assumption number and corresponding consequence number. The entry is a PROLOG term of the form:

phrase, echo (language, entry, target database, assumption bases, rating/mass)

Such as:

1. phrase, echo(english, [the, man, has, a, large, nose, and, squinted, eyes], heads),
[[1]],100 with rating enabled OR

1. phrase, echo(english, [the, man, has, a, large, nose, and, squinted, eyes], heads),

[[1]],((0.0,0.0,1.0)) with mass assignment enabled

Lexical Analysis Knowledge Source

The lexical analysis stage associates English words with their conventional lexical classes. A more complete lexical analysis would allow greater latitude in assuming domain type for unknown words, although this would be at the expense of some fault tolerance. It is this knowledge source which contains knowledge about the structure and meaning of the sentence.

This stage involves inferring the lexicon and grammar of the source language, in this case English, to firstly identify words in the lexicon and then identify the correct structure of each phrase provided in the grammar. The result of this phase is a set of entries which correspond to the interpretations placed on the words in the user description. From the entry in 1 we obtain entries of the form:

(Inference of Lexical)
language, language(lexicon clause(type of word), [type of word, lexical variable],[original phrase], lexical(lexicon clause, [type of word], target database), assumption bases, rating/mass.
OR
(Inference of Lexical and Grammar)
language, language(lexicon clause(type of word), [type of word, lexical variable],[original phrase], grammar(grammar clause, [grammar structure], [lexical(lexicon, [type of word])]), target database), assumption bases, rating/mass.

Such as:
2. english, english(definite_article([the]), [the, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(definite_article, [the]), heads),[[1, 2]],100

3. english, english(noun([man]), [lex_var, man, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(noun, [man]), heads),[[1, 3]],100

4. english, english(transitive_verb([has]), [lex_var, lex_var, has, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(transitive_verb, [has]), heads),[[1, 4]],100

5. english, english(indefinite_article([a]), [lex_var, lex_var, lex_var, a, lex_var, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(indefinite_article, [a]), heads),[[1, 5]],100

6. english, english(adjective([large]), [lex_var, lex_var, lex_var, lex_var, large, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(adjective, [large]), heads),[[1, 6]],100
7. english, english(noun([nose]), [lex_var, lex_var, lex_var, lex_var, lex_var, nose, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(noun, [nose]), heads),[[1, 7]],100

8. english, english(conjunction([and]), [lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, and, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(conjunction, [and]), heads),[[1, 8]],100

9. english, english(adjective([squinted]), [lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, squinted, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], lexical(adjective, [squinted]), heads),[[1, 9]],100
10. english, english(noun([eyes]), [lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, eyes], [the, man, has, a, large, nose, and, squinted, eyes], lexical(noun, [eyes]), heads),[[1, 10]],100

11. english, english(adjective_phrase([squinted]), [lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, squinted, lex_var], [the, man, has, a, large, nose, and, squinted,

eyes], grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), heads),[[1, 9, 11]],100

12. english, english(adjective_phrase([large]), [lex_var, lex_var, lex_var, lex_var, large, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), heads),[[1, 6, 12]],100

13. english, english(noun_phrase([eyes]), [lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [noun], [lexical(noun, [eyes])]), heads),[[1, 10, 13]],100

14. english, english(noun_phrase([squinted, eyes]), [lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])]), heads),[[1, 9, 10, 11, 13, 14]],100

15. english, english(noun_phrase([nose]), [lex_var, lex_var, lex_var, lex_var, lex_var, nose, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [noun], [lexical(noun, [nose])]), heads),[[1, 7, 15]],100

16. english, english(noun_phrase([large, nose]), [lex_var, lex_var, lex_var, lex_var, large, nose, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), heads),[[1, 6, 7, 12, 15, 16]],100

17. english, english(noun_phrase([a, large, nose]), [lex_var, lex_var, lex_var, a, large, nose, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes],

18. grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), heads),[[1, 5, 6, 7, 12, 15, 16, 17]],100

19. english, english(verb_phrase([has, a, large, nose]), [lex_var, lex_var, has, a, large, nose, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])])]), heads),[[1, 4, 5, 6, 7, 12, 15, 16, 17, 18]],100

20. english, english(noun_phrase([man]), [lex_var, man, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [noun], [lexical(noun, [man])]), heads),[[1, 3, 19]],100

21. english, english(sentence([man, has, a, large, nose]), [lex_var, man, has, a, large, nose, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [man])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])])])]), heads),[[1, 3, 4, 5, 6, 7, 12, 15, 16, 17, 18, 19, 20]],100 ....

36, english, english(sentence([the, man, has, a, large, nose, and, squinted, eyes]), [the, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase,

```
[noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase],
[lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction,
noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase],
[lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase],
[grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]),
grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun],
[lexical(noun, [eyes])])])])])]), heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 19, 21, 22, 23, 36]],100
```

The above tells us that the system has interpreted the words in the phrase "the man has a large nose and squinted eyes" and resolved each word to an associated lexical. The entries 1 to 10 relate to lexical inference only and identify which lexical variable the word belongs to. Entries 11 – 36 (see Appendix C for unlisted entries 22-35) relate to grammar and lexical inference and mean that the two knowledge sources were used to interpret and parse the user described phrase. If the English is fairly tightly written with no extraneous or unrecognised words then the system is immediate in interpreting the data. However if extra words are inserted which have little bearing on the actual description then the system tries to make sense of them, fails and then looks for other interpretations which ignore the unexplained words

Formation of Descriptor List

This stage groups the interpreted words into a list of object and qualifier lists. These are PROLOG lists with data items separated by comma within square brackets [ ]. The result of this phase picks each word from the user defined description and by comparing the word with the final interpreted result from the lexical analysis it assigns a tag to the word. This tag identifies the word as either an 'object' or a 'qualifier'. Objects can be considered as nouns such as 'man', 'nose', etc. Qualifiers are adjectives, describing the noun, such as 'african', 'european', 'asian', 'large', 'wide', etc. and adverbs also known as hedges such as 'very', 'fairly', 'slightly', etc. From the entry in 36 we obtain entries of the form:

target database, descriptor([original phrase], [[object([noun]), [qualifiers ([adjective],[adverb])]]), [[assumption bases]], rating/mass

Such as:

37, heads, descriptor([the, man, has, a, large, nose, and, squinted, eyes], [[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([squinted], [])]], [object([nose]), [qualifiers([], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 21, 29, 30, 31, 32, 34, 37]],100

38, heads, description([[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([squinted], [])]], [object([nose]), [qualifiers([], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 21, 29, 30, 31, 32, 34, 37]],100

39, heads, descriptor([the, man, has, a, large, nose, and, squinted, eyes], [[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([squinted], [])]], [object([nose]), [qualifiers([large], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 21, 25, 26, 27, 35, 39]],100

40, heads, descriptor([the, man, has, a, large, nose, and, squinted, eyes], [[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([squinted], [])]], [object([nose]), [qualifiers([large], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 21, 22, 23, 36, 40], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 21, 25, 26, 27, 35, 40]],100

The result from this stage is passed to the heads engine which translates this linguistic data to numeric parameters for the 3D head generator script.

6.5 Translating the Linguistic Data

Technically this phase of the system is conceptually different to the natural language processing stage. However since the result from the natural language interpretation directly leads in to the heads engine for translation to numeric parameters it is closely related to the natural language interface module.

We observed the last set of results from the natural language interpreter being a list of descriptor of the form:

descriptor([],[[object([man]),[qualifiers([],[])]],[object([eyes]),[qualifiers([squinted],[])]], [object([nose]),[qualifiers([large],[])]]])

The list is broken down by the heads engine into smaller lists of object and qualifiers. These lists get processed through a series of routines that identify each object and its associative qualifier and perform various actions ranging from loading the correct template file to calculating modifiers for each object taking into account hedges that affect the amount of modification applied to the original template parameters.

In the sentence “the man has a large nose and squinted eyes”. The engine will initially identify man as the parent object and load the correct template file. Currently the database of templates contains two entries; male template and female template representing geometry data of the two baseline heads constructed and described in chapter 4. These files hold the default parameters for the baseline head model. Since the object ‘man’ does not have an accompanying qualifier the process of loading the appropriate modifier parameters does not occur. Example of some acceptable qualifiers for ‘man’ could be ‘Asian’, ‘European’, ‘African’, ‘Fat’, etc. Modifiers are sets of parameters that can affect the head geometry when applied to the baseline head (see chapter 4, section 4.8).

The next stage involves processing the next object - qualifier list. The only difference is that this time the object – qualifier list contains a qualifier and the object is a child of the parent object ‘man’. This stage does not require loading of object parameters since the object is a feature of the parent and its parameters were loaded in the previous stage. In order to modify the eyes so that they appear 'squinted' the engine will first load the modifier file from the database of modifiers. The modifier file holds Prolog clauses with sets of parameters that can transform every aspect of a feature to produce a near realistic modification. The engine picks the feature specified in the object-qualifier list and recursively loops through all the parameters of the feature applying modifications to the feature list of the template head. The process is repeated until an empty object-qualifier list is reached.

Hedges are processed by the heads engine using a power function to concentrate or dilute the modifier parameters. The power function returns a high value for smaller power values and low values for high power numbers. So if the qualifiers list contained the items 'wide' and 'fairly' for object nose.

[object([nose]),[qualifiers([wide],[fairly])]] (6.a)

then the engine will identify 'fairly' as a hedge and calculate the new modifier value by diluting the original modifier value.

CONCENTRATE FUNCTION

extremely → original modifier * power(0.5) (6.b)

very → original modifier * power(0.6) (6.c)

DILUTE FUNCTION

fairly → original modifier * power(1.1) (6.d)

slightly → original modifier * power(1.2) (6.e)

There is however one small problem that must be addressed before any modifier value is applied to the original head parameters and that is ensuring all modifications lie within the upper and lower limits defined in table 4.1 for each head parameter. Let us assume the default parameter for nose width is 1.0., the upper limit for this parameter is 2.0 and bottom limit is 0.0. Now if a user description modifies the nose width by adding 0.5 increasing the parametric value of nose width to 1.5 and the user unsatisfied with this width amplifies the description to further increase the width and does so using the adverb 'extremely' then the new modifier calculated would be:

[object([nose]),[qualifiers([wide],[extremely])]] (6.f)

New modifier = modifier(nose width) * power(0.5) (from equation 6.b)

New modifier = 0.5 * power(0.5) = 0.7

New nose width = 1.5 + 0.71 = 2.2

Addition of the new modifier value to the current nose width parameter will result in a parametric value that exceeds the legitimate limit for nose width. This is valid the other way round where the new parameter value calculated may lie below the lower limit. To avoid such errors a Gaussian normal distribution function (equation 6.g) has been implemented that maintains the parameters within the specified range. The normal distribution is characterized by two parameters: the mean μ and the standard deviation . The mean is a measure of location or centre and the standard deviation is a measure of scale or spread. The mean can be any value between ± infinity and the standard deviation must be positive. Each possible value of μ and define a specific normal distribution and collectively all possible normal distributions define the *normal family*

$$f(x)=\frac{1}{(2\pi\sigma)^{1/2}}e^{\frac{(x-\mu)^2}{2\sigma^2}},\quad -\infty\langle x<\infty \qquad (6.g)$$

Figure 6.7 shows a diagram of a normal distribution implemented to determine the appropriate modifier value for a given parameter value. The diagram represents diminishing modifier values for nose width parameters above and below the default or mean value. Let us consider the normal distribution diagram in Figure 6.7, if the value for nose width is recorded being 1.5 then the modifier value for increasing nose width would be returned as approximately 0.3. In case of no hedges declared the new modifier value when added to the nose width parameter will result in a value well within the maximum limit for nose width.

New nose width = 1.5 + 0.3 = 1.8

Also if hedges are declared as in 6.f then applying hedges to the modifier will give:

New modifier = modifier(nose width) * power(0.5) (from equation 6.b)

New modifier = 0.3 * power(0.5) = 0.5

New nose width = 1.5 + 0.5 = 2.0

The new modifier influenced by application of hedges still results in a new nose width within the nose width parameter range even though it is tight.

Once the head engine has successfully translated all available linguistic data, it writes the full list of new head parameters to an ASCII text file for the Head Generator script to consult and construct the 3D head model. Code for the heads engine and the NLI can be found in Appendix C.

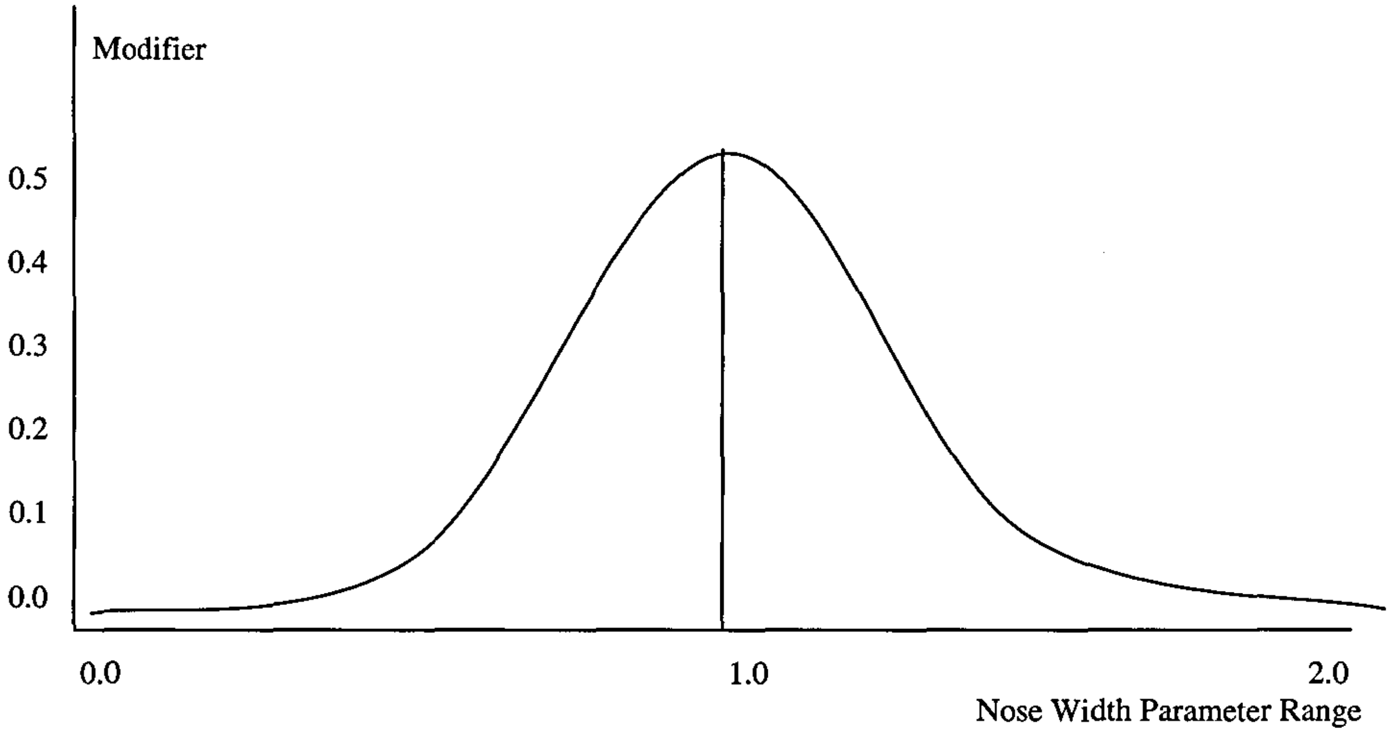


Figure 6.7 Normal Distribution function used to determine modifier value so that increment or decrement to parameter remains within range.

6.6 Conclusion

This chapter has explored natural language processing. It started with a brief description of natural languages, the various areas of study connected with natural language processing. It then moved on to inspect the anatomy of language, its orthographic structure, grammar and components of grammar. Computational tools such as Parsing, PROLOG, Echo and the Truth Maintained Blackboard system. Finally it looked at how the black board system with an assumption based truth maintenance system interprets natural language descriptions of faces and how the interpreted linguistic data gets translated by the heads engine into parameters for the facial image generation module.

Chapter 7

Does it Work? – Description of overall System Architecture, Test Data and Results

Abstract

This chapter provides a detailed overview of the final system architecture. It shows how the different modules and processes described in the earlier chapters work together to construct facial imagery from natural language descriptions. It also provides results of an exhaustive test containing a spectrum of facial descriptions in sentences both simple and complex.

Keywords: Architecture, Testing, Results

7.1 Introduction

Chapters 2 to 6 have primarily concentrated on solutions to specific processes necessary to achieve the thesis aims. This chapter shows how all the different modules and processes link together to form the overall system. In chapter 1 a broad overview of the thesis aim was discussed. Building on that broad definition we will now discuss how the three main modules i.e. the Natural Language Interface (NLI), TMS + Fuzzy Logic process, and Facial Image Generation module operate in tandem to generate 3D facial images from natural language descriptions of a human face.

7.2 System Architecture

The underlying technology for our research system is Mac PROLOG working on an Apple Power Mac G3 running Mac OS 9. The interface for describing faces is an extension of Echo and allows users to enter sentences of facial description. These sentences can be edited to refine the facial image using the *amplify* function. The *describe* and *amplify* process will be described in greater detail further on in the chapter.

The facial image generation module operates inside 3D Studio Max on a Windows OS and runs on the host machine via Virtual PC (Connectix, 2001). Apple script connects the NLI to the facial image generation module by launching 3D Studio Max (if not running) and passing the file of heads parameter generated by the heads engine to the Head Generator Script (HGS). Figure 7.1 gives a diagram of the system architecture.

The diagram in Figure 7.1 includes a collection of sub environments within a global environment in which processes and data flow to successfully accomplish the task of generating facial images from natural language descriptions. The global environment refers to the Mac OS platform under which the whole system works. The sub environments include application environments such as MacPROLOG and Virtual PC. Within these sub environments we have Echo and 3D Studio Max running.

The process starts with the user describing a facial image by entering textual data in the Describe box of the NLI. The data entered, usually in sentence form, passes through the TMS and blackboard system inferring the grammar rules and lexical database to make sense of the linguistic data. The list of descriptors produced is processed by the head engine which divides the list into smaller lists of object and qualifiers. These lists contain a single object (or feature such as eyes, nose, ears) and one or more qualifiers (also called descriptors like round, large, long, small, etc.) and hedges (or adverbs like very, fairly, slightly). These descriptors or qualifiers are first loaded from a database of modifiers then each aspect of a feature is calculated making careful adjustments for hedges and the value of an aspect with respect to its distribution table.

The distribution table enables decision to be made on the modifier value by which the parameter of an aspect should either be concentrated or diluted. This translation process generates a list of parameters for each aspect of a feature of the head. The complete list is then written to an external file and saved to a network disk or shared folder from where the facial image generation module can easily access the parameters files and generates a 3D head. Since the 3D Modelling application and HGS run outside the Echo environment, a linking process has been developed that can be invoked from within MacPROLOG, or any other Macintosh application for that matter.

This linking process utilises Apple Script Technology to launch the VPC environment and initiate 3D Studio Max. Once activated the Head Generator Script locates the *Heads Parameter* file and uses the data to generate a 3D head model. The type of baseline head model to load is included in the heads parameter file. The correct baseline head model and textures are loaded from a library of head models and textures stored on a local drive. The final task involves instructing the rendering engine to produce a rendered image of the 3D head for the user to visualise.

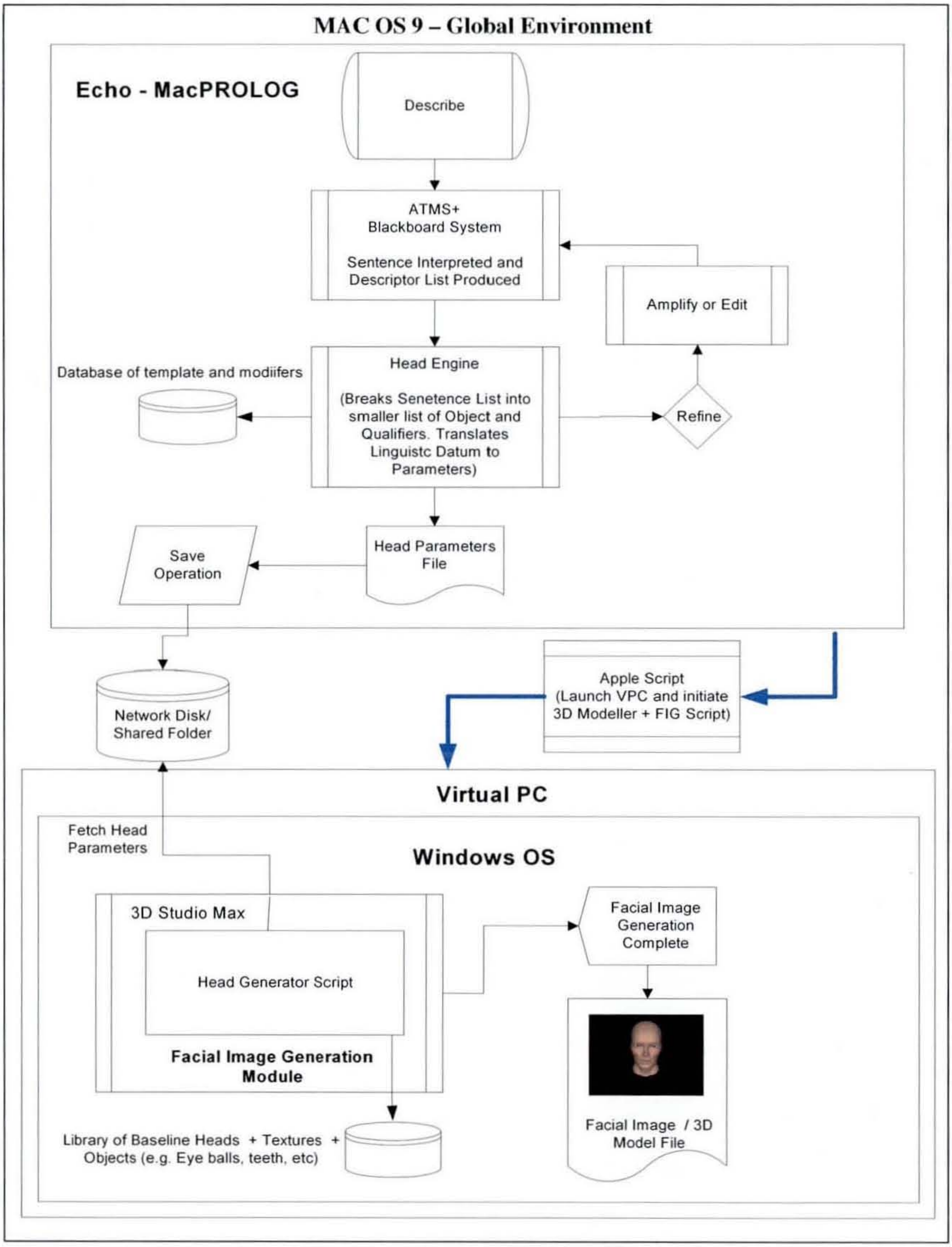
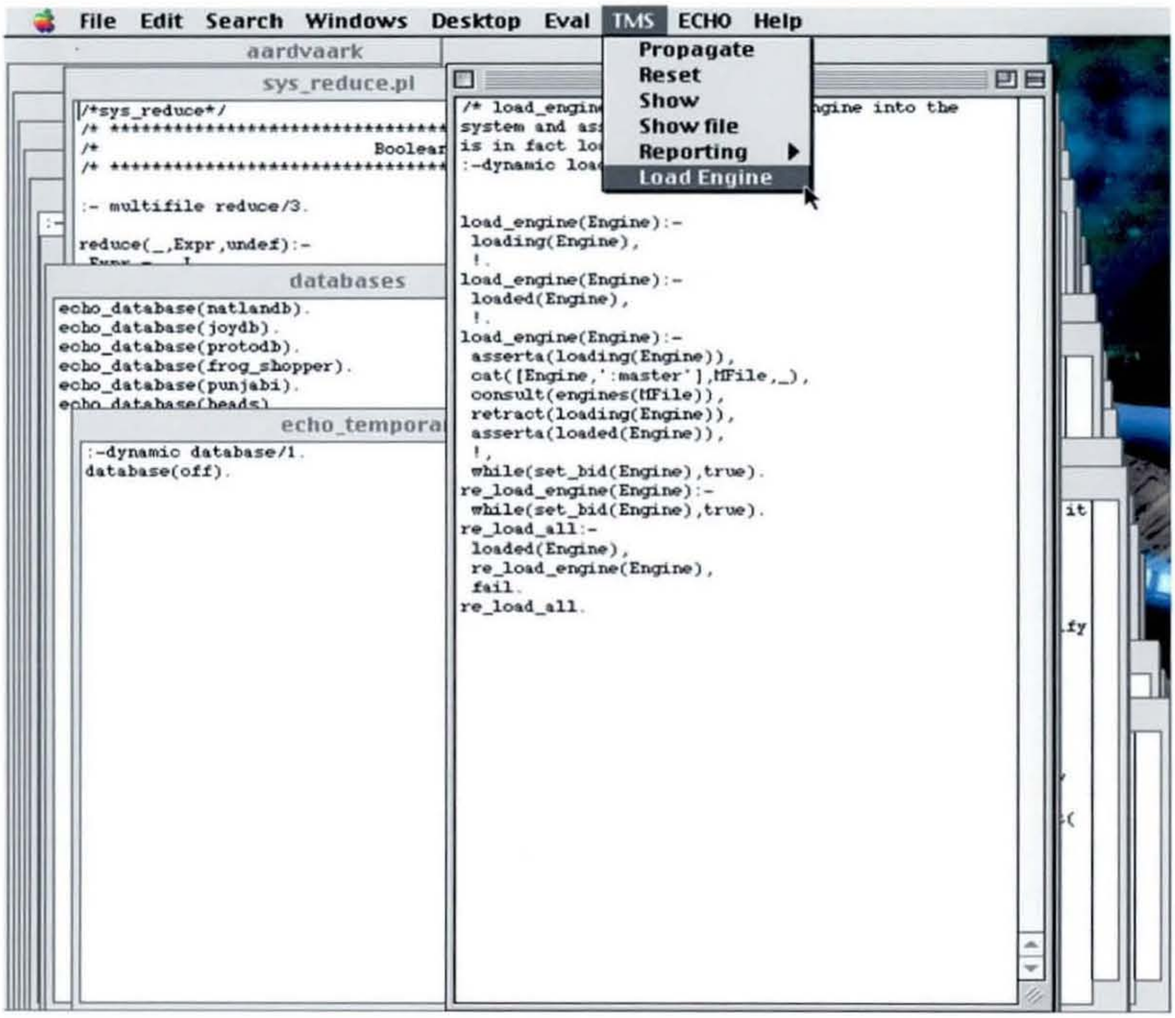


Figure 7.1 System Architecture Diagram – Shows Flow of Process and Data

The interface design for the *describe* procedure is shown in Figures 7.2 to 7.5. It consists of two simple menu functions, (a) load the head engine from the TMS menu and (b) bring up the describe window from the Echo menu where the user enters the natural language descriptions.



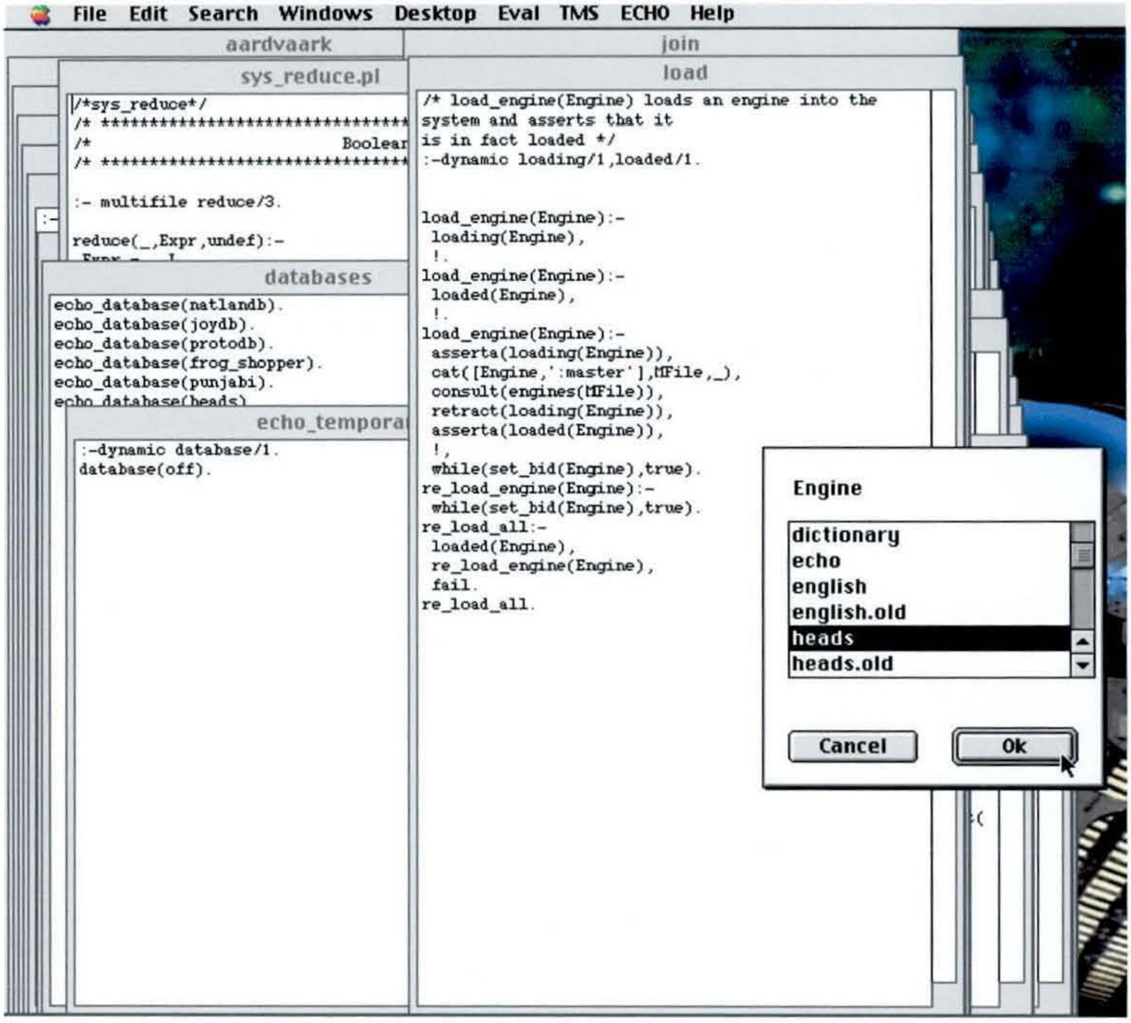
```
/*sys_reduce*/
/* ********************************
/*                          Boolear
/* ********************************

:- multifile reduce/3.

reduce(_,Expr,undef):-
```

```
echo_database(natlandb).
echo_database(joydb).
echo_database(protodb).
echo_database(frog_shopper).
echo_database(punjabi).
echo_database(heads)
```

```
:-dynamic database/1.
database(off).
```

```
/* load_engin                      gine into the
system and as
is in fact lo
:-dynamic loa


load_engine(Engine):-
 loading(Engine),
 !.
load_engine(Engine):-
 loaded(Engine),
 !.
load_engine(Engine):-
 asserta(loading(Engine)),
 cat([Engine,':master'],MFile,_),
 consult(engines(MFile)),
 retract(loading(Engine)),
 asserta(loaded(Engine)),
 !,
 while(set_bid(Engine),true).
re_load_engine(Engine):-
 while(set_bid(Engine),true).
re_load_all:-
 loaded(Engine),
 re_load_engine(Engine),
 fail.
re_load_all.
```

Figure 7.2 Shows TMS menu with 'Load Engine' highlighted for selection



```
/*sys_reduce*/
/* *********************************
/*                           Boolear
/* *********************************

:- multifile reduce/3.

reduce(_,Expr,undef):-
```

```
echo_database(natlandb).
echo_database(joydb).
echo_database(protodb).
echo_database(frog_shopper).
echo_database(punjabi).
echo_database(heads)
```

```
:-dynamic database/1.
database(off).
```

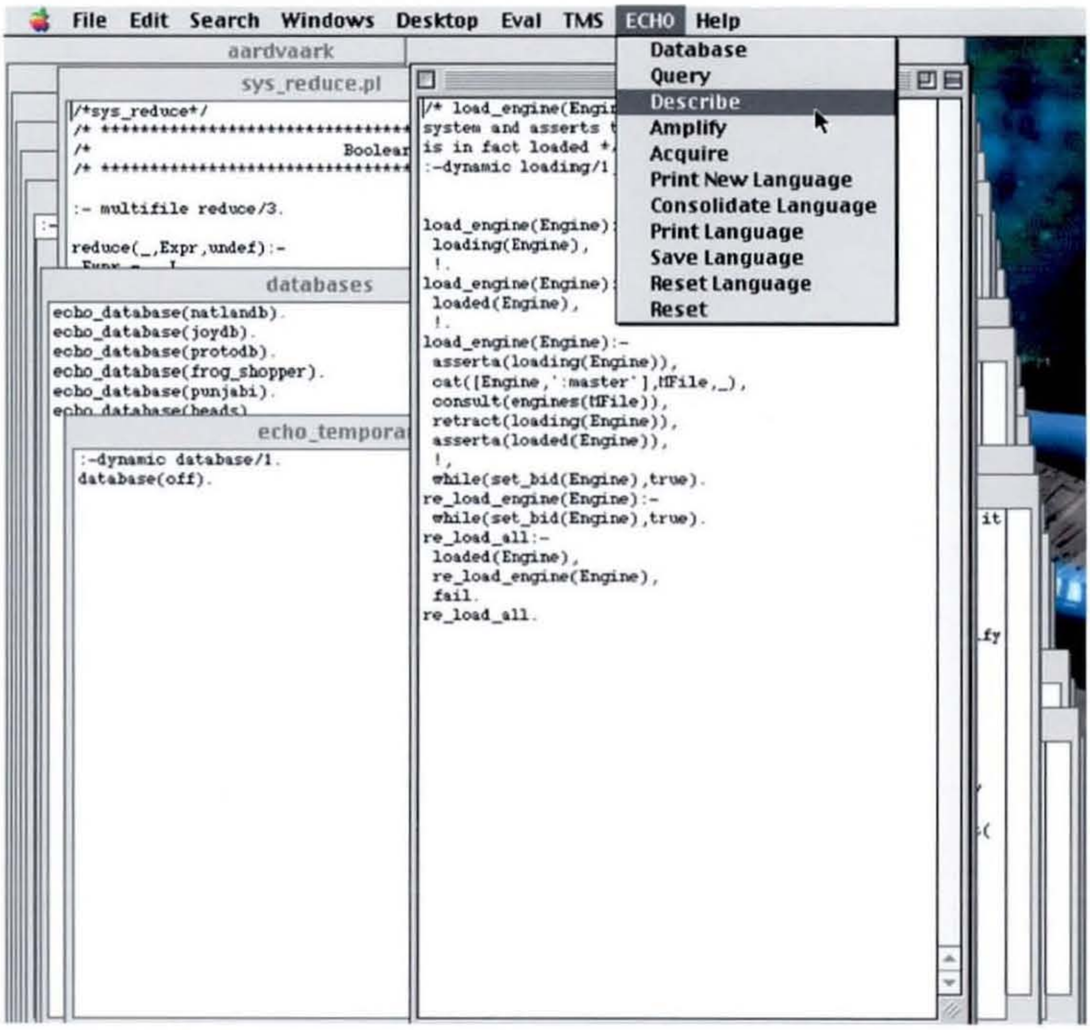
```
/* load_engine(Engine) loads an engine into the
system and asserts that it
is in fact loaded */
:-dynamic loading/1,loaded/1.


load_engine(Engine):-
 loading(Engine),
 !.
load_engine(Engine):-
 loaded(Engine),
 !.
load_engine(Engine):-
 asserta(loading(Engine)),
 cat([Engine,':master'],MFile,_),
 consult(engines(MFile)),
 retract(loading(Engine)),
 asserta(loaded(Engine)),
 !,
 while(set_bid(Engine),true).
re_load_engine(Engine):-
 while(set_bid(Engine),true).
re_load_all:-
 loaded(Engine),
 re_load_engine(Engine),
 fail.
re_load_all.
```

```
dictionary
echo
english
english.old
heads
heads.old
```

Figure 7.3 Shows the Engine Selection window with the 'heads' engine highlighted for selection

The *amplify* procedure brings up the same describe window (Figure 7.5) as the *describe* procedure but it does so without resetting the language and heads data. New entries or modifications are merged with the existing sentence structure and reprocessed appending new calculated parameters to the existing parameters list and writing a new heads parameter file at the end of the process. The code for *describe*, *amplify*, TMS and Echo menu is given in Appendix C along with a full listing of the heads engine and other related predicates.

The call for launching VirtualPC and 3D Studio Max can be triggered either by MacPROLOG or manually by the user executing the Apple script link. The HG script runs automatically on start up of 3D Studio Max. The script also provides a customised interface with controls to edit the 3D human head model. Such a feature would normally not be revealed to the end user since modifications should be handled by the NLI using the amplify process and not the modelling environment. However if finer control is need to edit the head models then the means to do so is available. Figure 7.6 shows the control window offered by the HG script to create and edit the 3D head model.



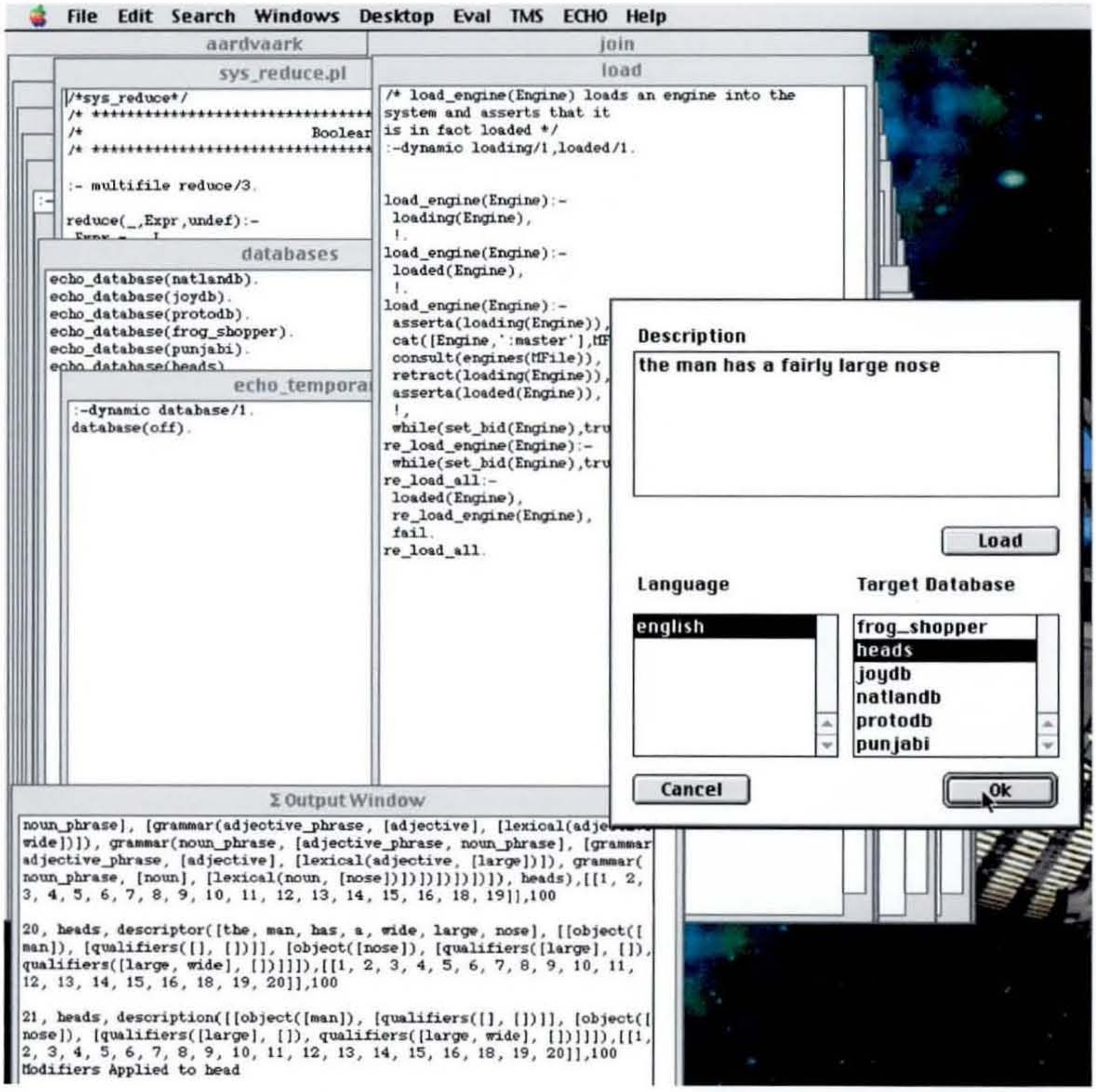
```
/*sys_reduce*/
/* ********************************
/*                         Boolear
/* ********************************

:- multifile reduce/3.

reduce(_,Expr,undef):-
```

```
echo_database(natlandb).
echo_database(joydb).
echo_database(protodb).
echo_database(frog_shopper).
echo_database(punjabi).
```

```
:-dynamic database/1.
database(off).
```

```
/* load_engine(Engi
system and asserts t
is in fact loaded *
:-dynamic loading/1


load_engine(Engine)
 loading(Engine),
 !.
load_engine(Engine)
 loaded(Engine),
 !.
load_engine(Engine):-
 asserta(loading(Engine)),
 cat([Engine,':master'],tFile,_),
 consult(engines(tFile)),
 retract(loading(Engine)),
 asserta(loaded(Engine)),
 !,
 while(set_bid(Engine),true).
re_load_engine(Engine):-
 while(set_bid(Engine),true).
re_load_all:-
 loaded(Engine),
 re_load_engine(Engine),
 fail.
re_load_all.
```

Figure 7.4 Shows Echo Menu with 'Describe' highlighted for selection, note the entry 'Amplify' in the same menu under 'Describe'



```
/*sys_reduce*/
/* *********************************
/*                          Boolear
/* *********************************

:- multifile reduce/3.

reduce(_,Expr,undef):-
```

```
echo_database(natlandb).
echo_database(joydb).
echo_database(protodb).
echo_database(frog_shopper).
echo_database(punjabi).
echo_database(heads).
```

```
:-dynamic database/1.
database(off).
```

```
/* load_engine(Engine) loads an engine into the
system and asserts that it
is in fact loaded */
:-dynamic loading/1,loaded/1.


load_engine(Engine):-
 loading(Engine),
 !.
load_engine(Engine):-
 loaded(Engine),
 !.
load_engine(Engine):-
 asserta(loading(Engine)),
 cat([Engine,':master'],MF
 consult(engines(MFile)),
 retract(loading(Engine)),
 asserta(loaded(Engine)),
 !,
 while(set_bid(Engine),tru
re_load_engine(Engine):-
 while(set_bid(Engine),tru
re_load_all:-
 loaded(Engine),
 re_load_engine(Engine),
 fail.
re_load_all.
```

Description

the man has a fairly large nose

Language

english

Target Database

frog_shopper
heads
joydb
natlandb
protodb
punjabi

```
noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adje
wide])]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar
adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(
noun_phrase, [noun], [lexical(noun, [nose])])])])])])]), heads),[[1, 2,
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19]],100

20, heads, descriptor([the, man, has, a, wide, large, nose], [[object([
man]), [qualifiers([], [])]], [object([nose]), [qualifiers([large], []),
qualifiers([large, wide], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 18, 19, 20]],100

21, heads, description([[object([man]), [qualifiers([], [])]], [object([
nose]), [qualifiers([large], []), qualifiers([large, wide], [])]]]),[[1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20]],100
Modifiers Applied to head
```

Figure 7.5 Shows the Describe window, the text box under 'Description' is where the user enters text. The selection box under 'Language' allows a different language to be selected for input, so far only English has been implemented but work on other languages is being researched such as Punjabi and French. Load button opens a new window with predefined sentences that the user can pick and edit to make data entry convenient.

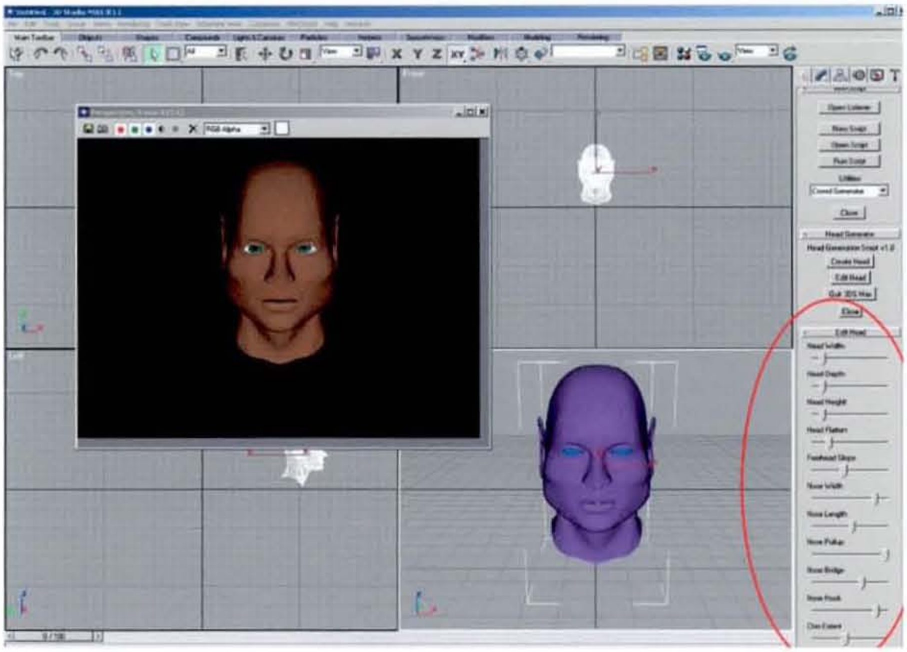
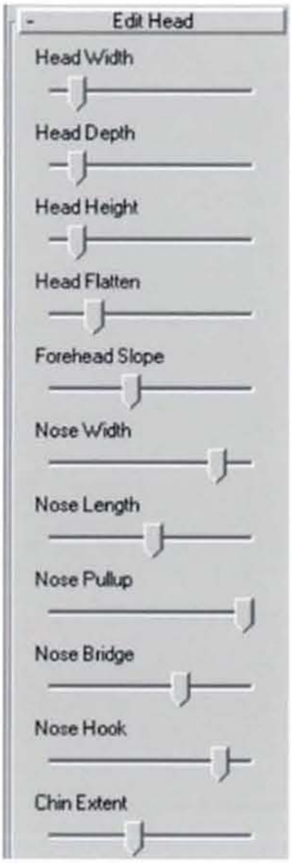


Figure 7.6 3D Studio Max interface with the HG script 'Edit Head' rollout circled in red
The figure below shows a magnified shot of a portion of the Edit Head menu.



7.3 Test Data and Experimentation

The test data consists of a number of sentences conjured to investigate operational efficiency of the overall system and natural language processing ability of the NLI. The aim of this experiment was to discover if the NLI could understand sentences of complex content and structure and interprets the data effectively so that the head engine would produce correct head parameters. The test data was fabricated so that each sentence was not always the same and predictable. Some sentences included more than a single adjective describing a feature and others were an amalgamation of several conjunctions. The test data used for the experiment is provided in table 7.1. Results from the experiment include a brief listing of the output reported. This includes the original user description, input into the natural language interpreter as entry 'phrase', the English interpretation produced by the NLI and the list of descriptors produced by the interpreter for the heads engine. The list of head parameters listed for test data 2 and 7 only show the complete list of head parameters generated by the heads engine after translating the descriptors to numeric parameters. Each test result includes a rendered image of the head generated by the Head Generator Script.

Test Data	
1	Describe – "The European man has a very wide nose"
2	Describe – "the man has a large nose and squinted eyes"
3	Describe – "the man has large eyes and a thin nose and a small mouth" Amplify – "the man has large eyes and a fairly wide nose and a small mouth"
4	Describe – "the man has puffed cheeks, and a hooked nose and a broad jaw"
5	Describe – "the african woman has a thin nose and aquamarine eyes"
6	Describe – "the man has a large wide nose and vampire eyes"
7	Describe – "the man has a very wide mouth and vampire eyes"
8	Describe – "the man has a round pugged nose and a broad jaw and an oval chin" Amplify – "The man has a round pugged nose and a fairly broad jaw and a slightly oval chin"
9	Describe – "the woman has small ears and a long nose and aquamarine eyes"

Table 7.1 – Sentences used as test data for experimentation

Test Data 1

Describe "the European man has a very wide nose" phrase, echo(english, [the, european, man, has, a, very, wide, nose], heads),[[1]], ((0.0,0.0,0.1))

english, english(sentence([european, man, has, a, very, wide, nose]), [lex_var, european, man, has, a, very, wide, nose], [the, european, man, has, a, very, wide, nose], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [european])]), grammar(noun_phrase, [noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adverb, adjective], [lexical(adverb, [very]), lexical(adjective, [wide])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])])])]), heads),[[1, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18, 21, 22, 23, 26]], ((0.0,0.0,1.0))

heads, descriptor([the, european, man, has, a, very, wide, nose], [[object([man]), [qualifiers([european], [])]], [object([nose]), [qualifiers([wide], [very])]]]),[[1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18, 19, 21, 22, 23, 24, 27]], ((0.0,0.0,1.0))

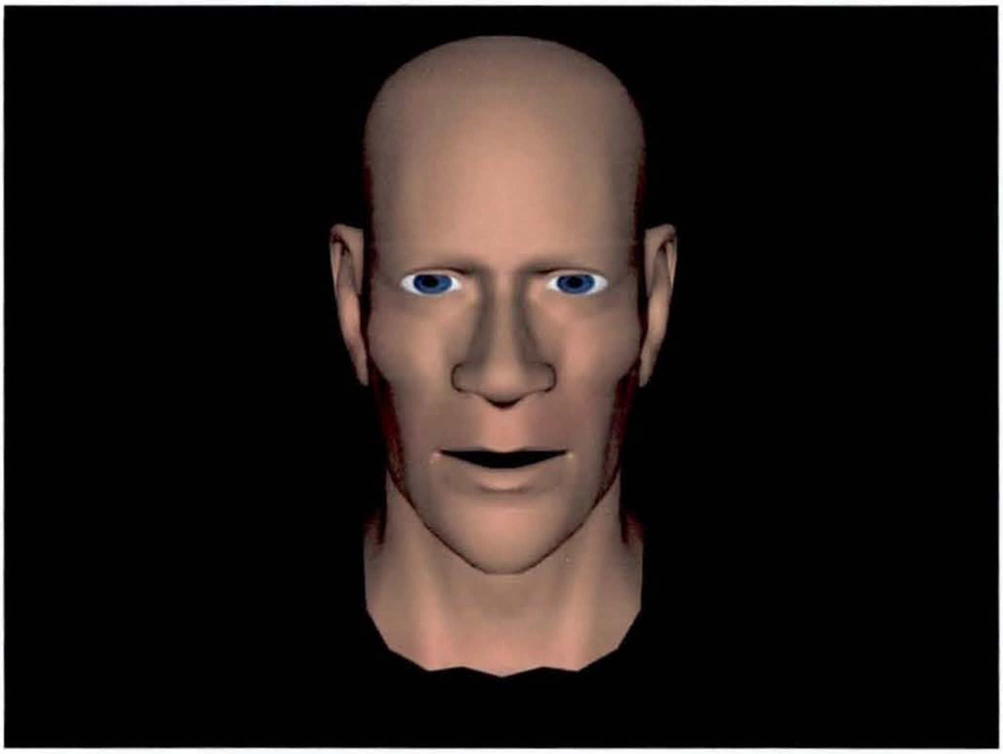


Figure 7.7 Rendered image of head generated from description "the european man has a very wide nose"

Test Data 2

Describe – "the man has a large nose and squinted eyes"

phrase, echo(english, [the, man, has, a, large, nose, and, squinted, eyes], heads),[[1]], ((0.0,0.0,1.0))

english, english(sentence([the, man, has, a, large, nose, and, squinted, eyes]), [the, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])]), heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 21, 22, 23, 36]], ((0.0,0.0,1.0))

heads, descriptor([the, man, has, a, large, nose, and, squinted, eyes], [[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([squinted], [])]], [object([nose]), [qualifiers([large], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 21, 22, 23, 36, 40], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 21, 25, 26, 27, 35, 40]], ((0.0,0.0,1.0))

Modifiers Applied to head

Man
head texture = 1
head type = 2
head strength = 0
head x_pull = 100
head y_pull = 100
head z_pull = 100
head y_offset = 0
head z_offset = 0
head width = 1
head widthskew1 = 0
head widthskew2 = 0
head depth = 1
head depthskew = 0
head height = 1

nose hook = 0
nose hook_influence = 0
chin extent = 1
chin tilt = 1
chin tilt_influence = 0
chin accent = 0
jaw width = 0
jaw influence = 0.5
jaw uniformity = 0.5
cheek extrude = 0
cheek zpos = 0
cheek curvatures = 0
cheek curvature_zpos = 0
cheek curvature_ypos = 0
cheek curvature_zfalloff = 0.5

```
head heightskew = 0
head face_squash = 1
head flatten = 1
head slope = 0
nose width = 1.5
nose width_zweight = 0.5
nose length = 1.5
nose length_zweight = 0.65
nose pullup = 1
nose bridge = 0
ears depth = 0
ears rotation = 0
mouth width = 0
mouth protrude = 1
eye1translate x = 1.8
eye1translate y = -23.85
eye1translate z = 3.9
eye2translate x = -1.8
cheek curvature_yfalloff = 0.5
eyes colour = 14
eyes separation = 1
eyes inset = 0
eyes toproundness = -0.5
eyes bottomroundness = -0.5
eyes rotation = 0
eyes brow_bulge = 0
ears height = 0
ears lobe = 0
eye2translate y = -23.8
eye2translate z = 3.8
eye1rotate x = -1
eye1rotate y = 0
eye1rotate z = -1
eye2rotate x = -1
eye2rotate y = 0
eye2rotate z = 1
```

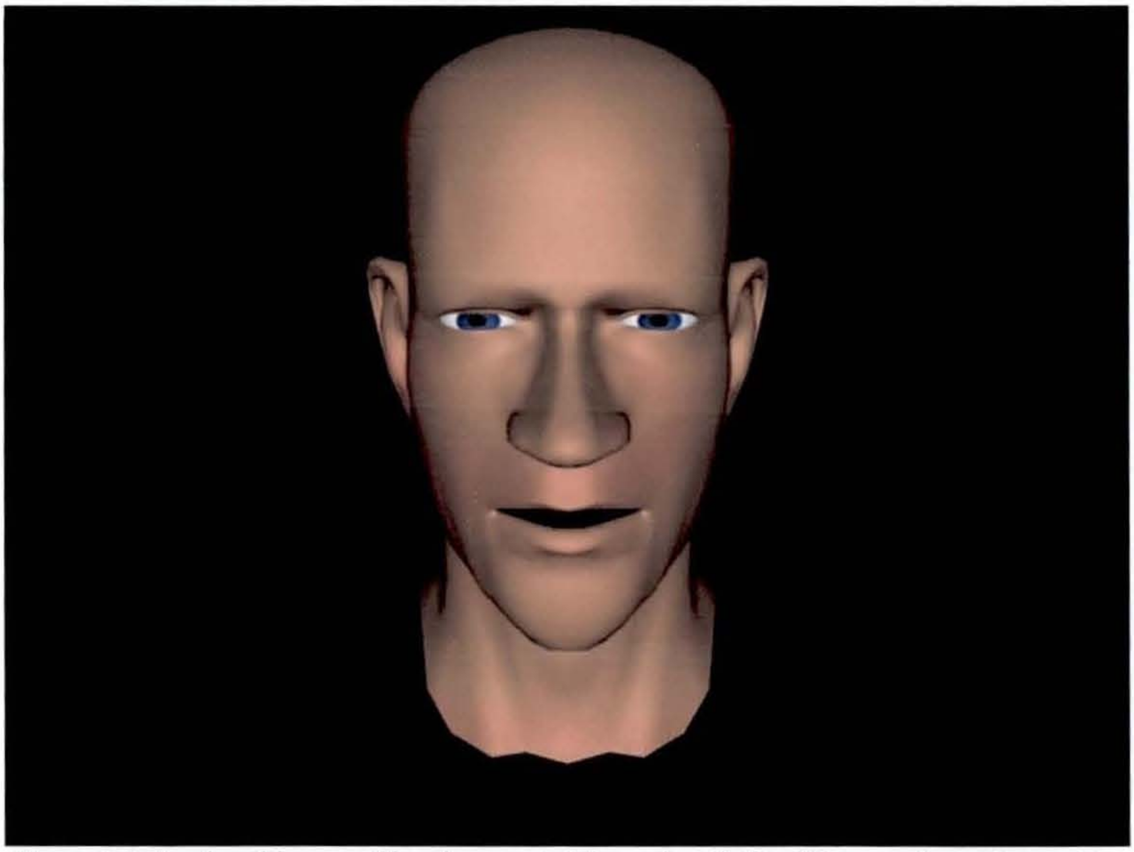


Figure 7.8 Rendered image of head generated from description "the man has a large nose and squinted eyes"

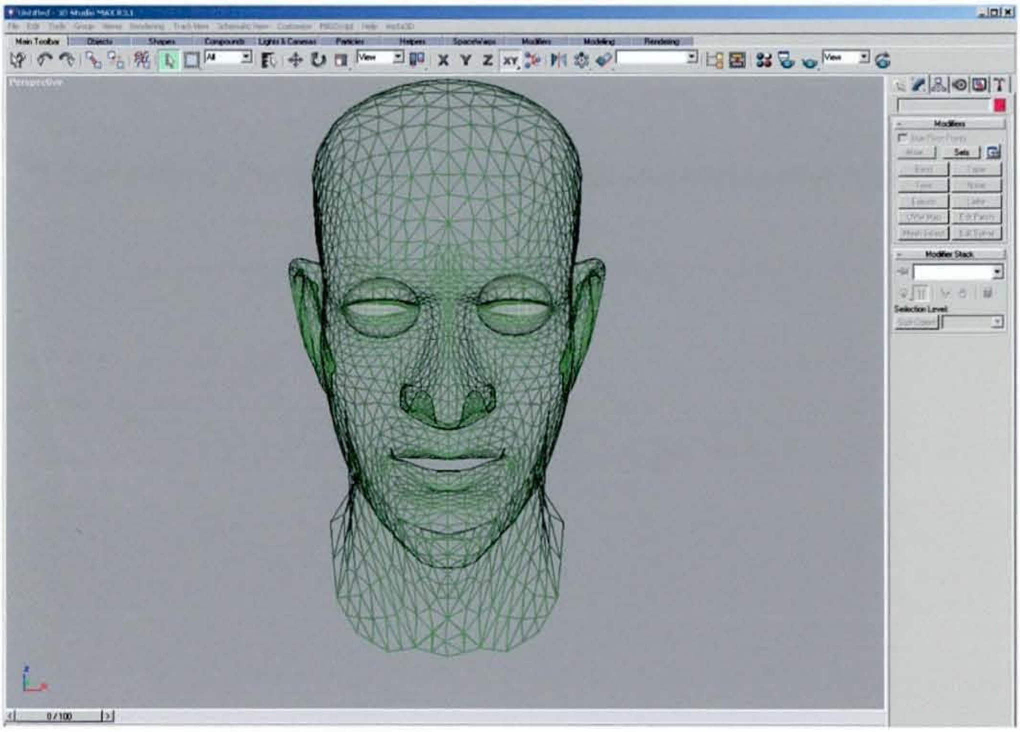


Figure 7.9 Front view of wire frame model of head generated by test data 2

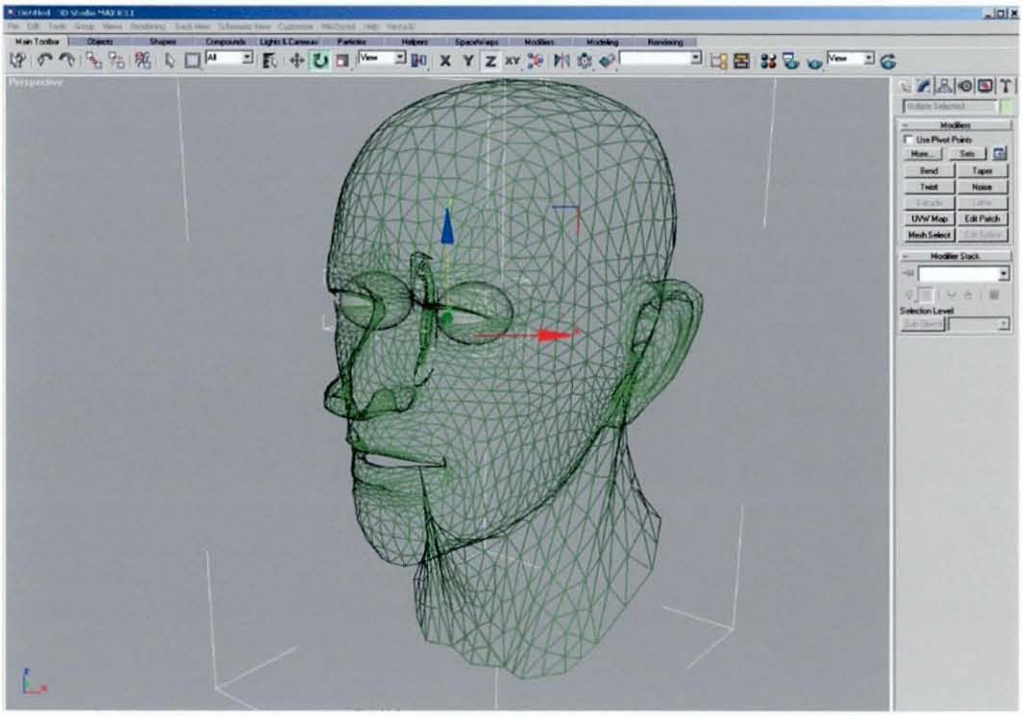


Figure 7.10 Rotated view of wire frame model of head generated by test data 2

Test Data 3

Describe – "the man has large eyes and a thin nose and a small mouth"

```
phrase, echo(english, [the, man, has, large, eyes, and, a, thin, nose, and, a, small, mouth],
heads),[[1]], ((0.0,0.0,1.0))
```

```
english, english(sentence([man, has, large, eyes, and, a, thin, nose, and, a, small, mouth]),
[lex_var, man, has, large, eyes, and, a, thin, nose, and, a, small, mouth], [the, man, has,
large, eyes, and, a, thin, nose, and, a, small, mouth], grammar(sentence, [noun_phrase,
verb_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [man])]),
grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun_phrase,
conjunction, noun_phrase], [grammar(noun_phrase, [noun_phrase, conjunction,
noun_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [eyes])]),
lexical(conjunction, [and]), grammar(noun_phrase, [indefinite_article, noun_phrase],
[lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase],
[grammar(adjective_phrase, [adjective], [lexical(adjective, [thin])]),
grammar(noun_phrase, [noun], [lexical(noun, [nose])])])])])]), lexical(conjunction, [and]),
grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [small])]), grammar(noun_phrase, [noun], [lexical(noun,
[mouth])])])])])])])]), heads),[[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 27, 60, 86, 87, 88, 90]], ((0.0,0.0,1.0))
```

```
heads, descriptor([the, man, has, large, eyes, and, a, thin, nose, and, a, small, mouth],
[[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([large], [])]],
[object([mouth]), [qualifiers([small], [])]], [object([nose]), [qualifiers([thin], [])]]]),[[1, 2,
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 29, 45,
76, 78, 79, 80], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 27, 29, 30, 42, 43, 69, 76], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 24, 25, 27, 29, 31, 32, 39, 40, 70, 76]], ((0.0,0.0,1.0))
```

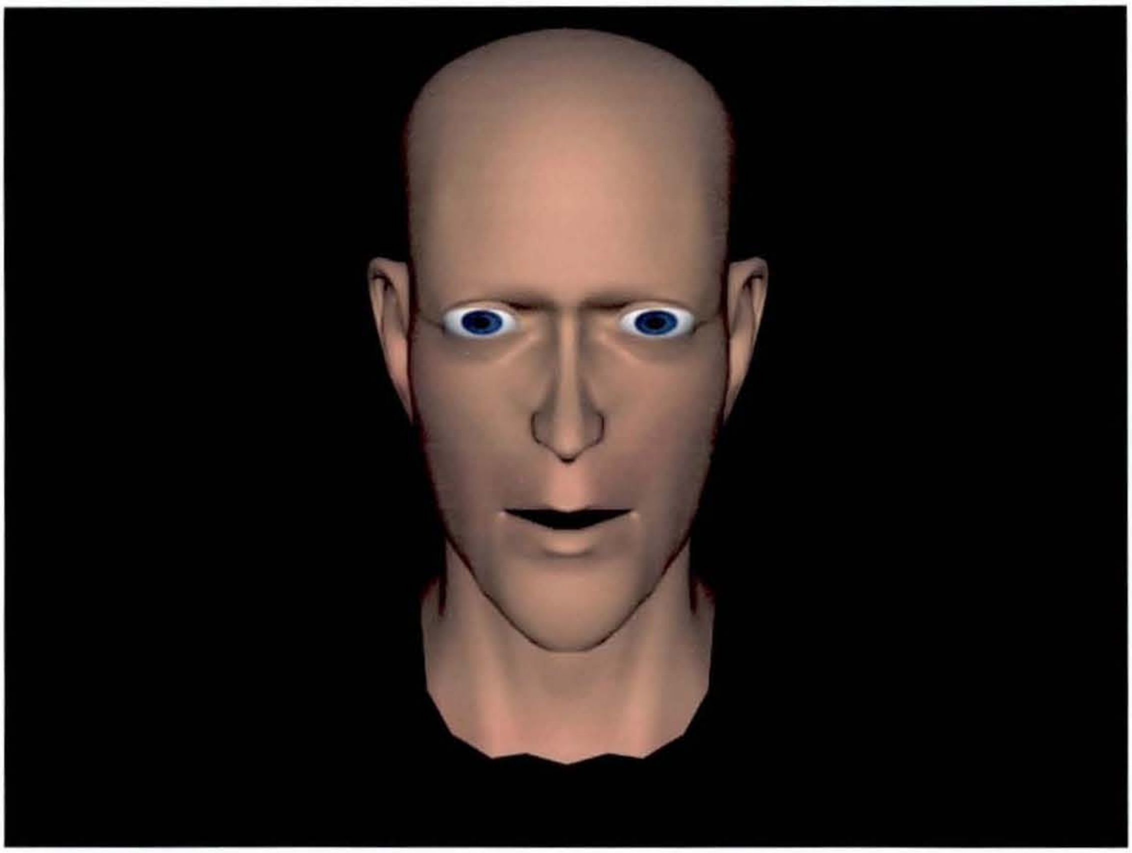


Figure 7.11 Rendered image of head generated from description "the man has large eyes and a thin nose and a small mouth"

Amplify – "the man has large eyes and a fairly wide nose and a small mouth"

```
phrase, echo(english, [the, man, has, large, eyes, and, a, fairly, wide, nose, and, a, small,
mouth], heads),[[1]], ((0.0,0.0,1.0))
```

```
english, english(sentence([man, has, large, eyes, and, a, fairly, wide, nose, and, a, small,
mouth]), [lex_var, man, has, large, eyes, and, a, fairly, wide, nose, and, a, small, mouth],
[the, man, has, large, eyes, and, a, fairly, wide, nose, and, a, small, mouth],
grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [noun],
[lexical(noun, [man])]), grammar(verb_phrase, [transitive_verb, noun_phrase],
[lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction,
noun_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase],
[grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]),
grammar(noun_phrase, [noun], [lexical(noun, [eyes])])]), lexical(conjunction, [and]),
grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]),
grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adverb, adjective], [lexical(adverb, [fairly]), lexical(adjective, [wide])]),
grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), lexical(conjunction, [and]),
grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [small])]), grammar(noun_phrase, [noun], [lexical(noun,
[mouth])])])])])])])])]), heads),[[91, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
105, 106, 108, 109, 110, 111, 112, 114, 115, 117, 118, 121, 173, 174, 180, 181, 183]],
((0.0,0.0,1.0))
```

```
heads, descriptor([the, man, has, large, eyes, and, a, fairly, wide, nose, and, a, small,
mouth], [[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([large], [])]],
[object([mouth]), [qualifiers([small], [])]], [object([nose]), [qualifiers([wide],
[fairly])]]]),[[91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 108,
109, 110, 111, 112, 114, 115, 117, 118, 120, 121, 122, 163, 169, 170, 171, 184], [91, 92,
93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 108, 109, 110, 111, 112,
114, 115, 117, 118, 120, 121, 122, 127, 150, 151, 152, 184]], ((0.0,0.0,1.0))
```

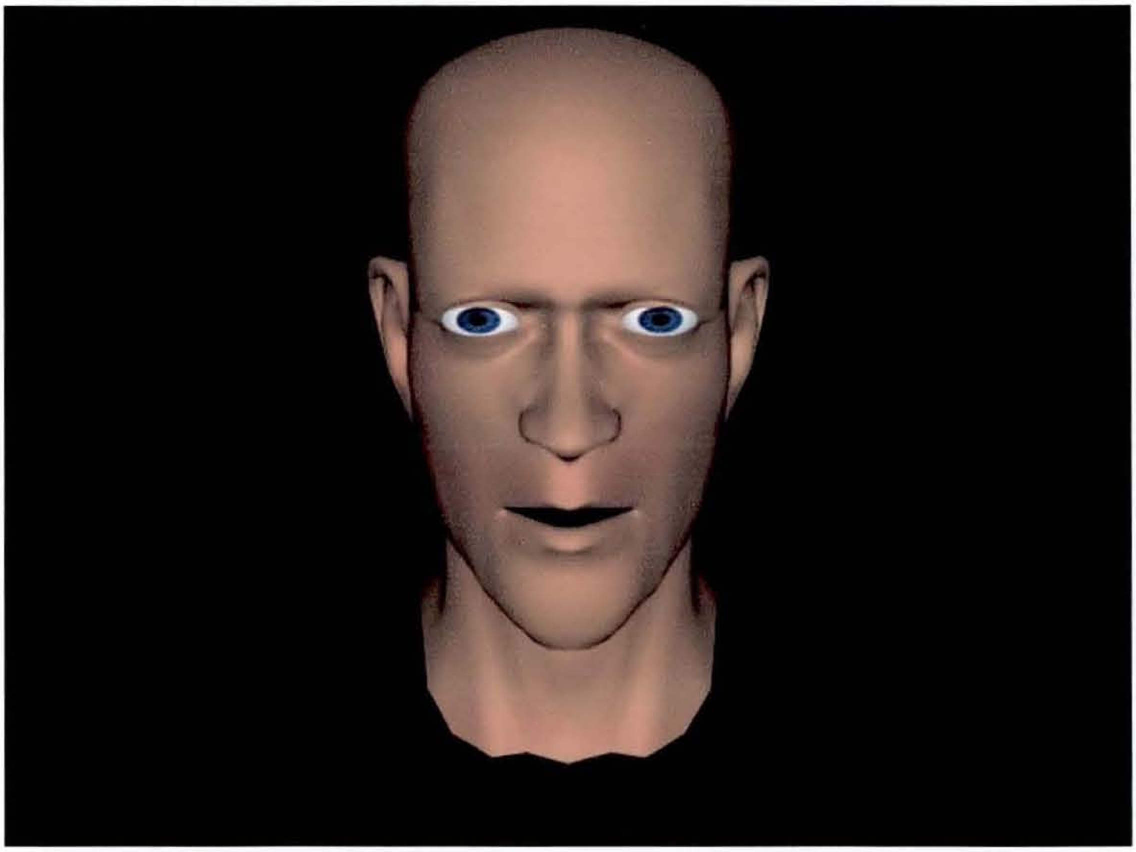


Figure 7.12 Rendered image of head generated from amplify instruction "the man has large eyes and a fairly wide nose and a small mouth"

Test Data 4

Describe – "the man has puffed cheeks, and a hooked nose and a broad jaw"

```
phrase, echo(english, [the, man, has, a, puffed, cheek, and, a, hooked, nose, and, a, broad,
jaw], heads),[[1]], ((0.0,0.0,1.0))
```

```
english, english(sentence([the, man, has, a, puffed, cheek, and, a, hooked, nose, and, a,
broad, jaw]), [the, man, has, a, puffed, cheek, and, a, hooked, nose, and, a, broad, jaw],
[the, man, has, a, puffed, cheek, and, a, hooked, nose, and, a, broad, jaw],
grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase,
[definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase,
[noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase],
[lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase],
[lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase],
[grammar(adjective_phrase, [adjective], [lexical(adjective, [puffed])]),
grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [noun], [lexical(noun, [cheek])]), lexical(conjunction, [and]),
grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [hooked])]), grammar(noun_phrase, [noun], [lexical(noun,
[nose])])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [indefinite_article,
noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase,
noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [broad])]),
grammar(noun_phrase, [noun], [lexical(noun, [jaw])])])])])])])])]), heads),[[1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 29, 31, 81, 126,
127, 128, 129, 130]], ((0.0,0.0,1.0))
```

```
heads, descriptor([the, man, has, a, puffed, cheek, and, a, hooked, nose, and, a, broad,
jaw], [[object([man]), [qualifiers([], [])]], [object([cheek]), [qualifiers([puffed], [])]],
[object([jaw]), [qualifiers([broad], [])]], [object([nose]), [qualifiers([hooked], [])]]]),[[1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 29, 31,
33, 34, 54, 55, 56, 92, 102], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 29, 31, 62, 102, 112, 113, 114, 115], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 29, 31, 62, 63, 102, 108, 109,
110], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
26, 29, 31, 32, 58, 59, 60, 91, 102], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 25, 26, 27, 29, 31, 33, 34, 41, 42, 96, 102], [1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 31, 32, 44, 45, 95,
102], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
26, 27, 29, 31, 47, 102, 104, 105, 106]], ((0.0,0.0,1.0))
```

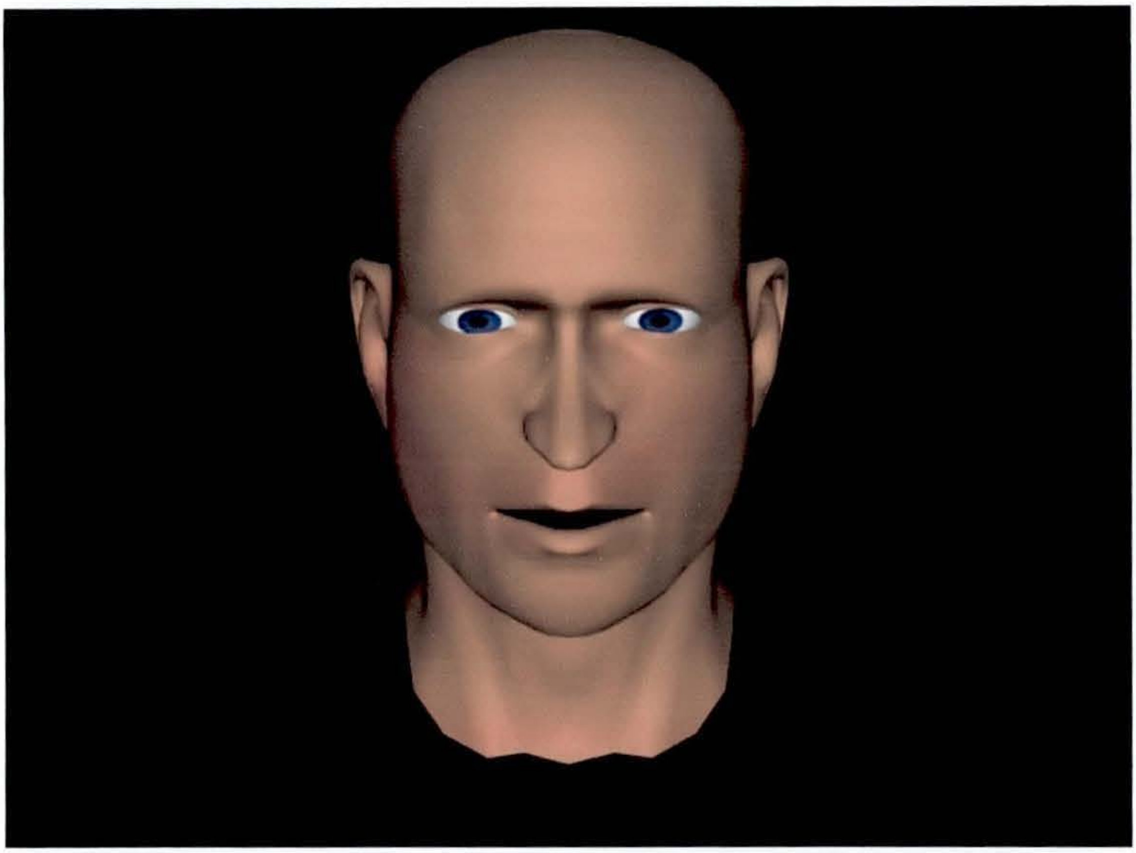


Figure 7.13 Rendered image of head generated from description "the man has puffed cheeks, and a hooked nose and a broad jaw"

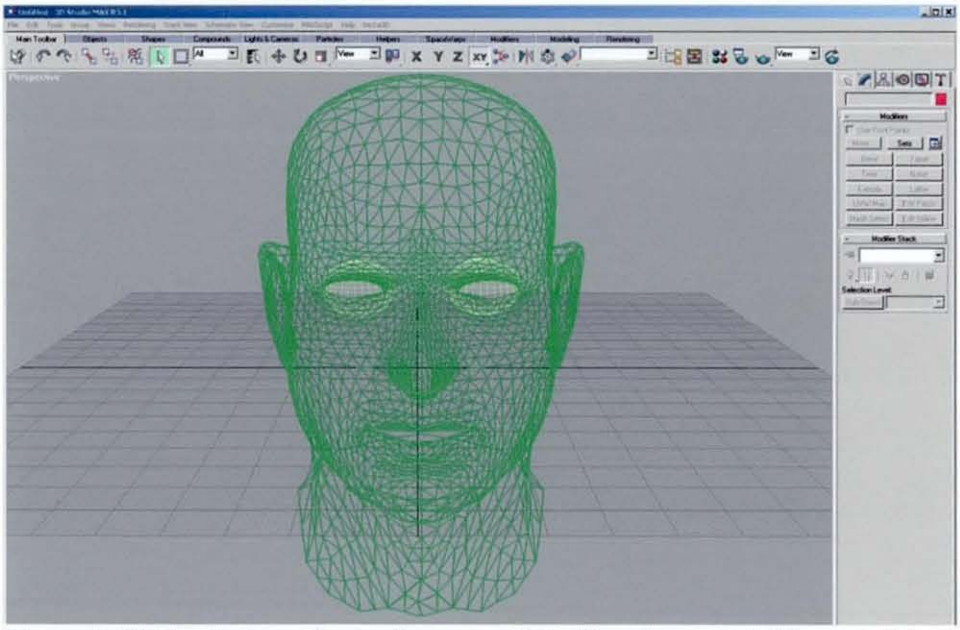


Figure 7.14 Front view of wire frame model of head generated by test data 4

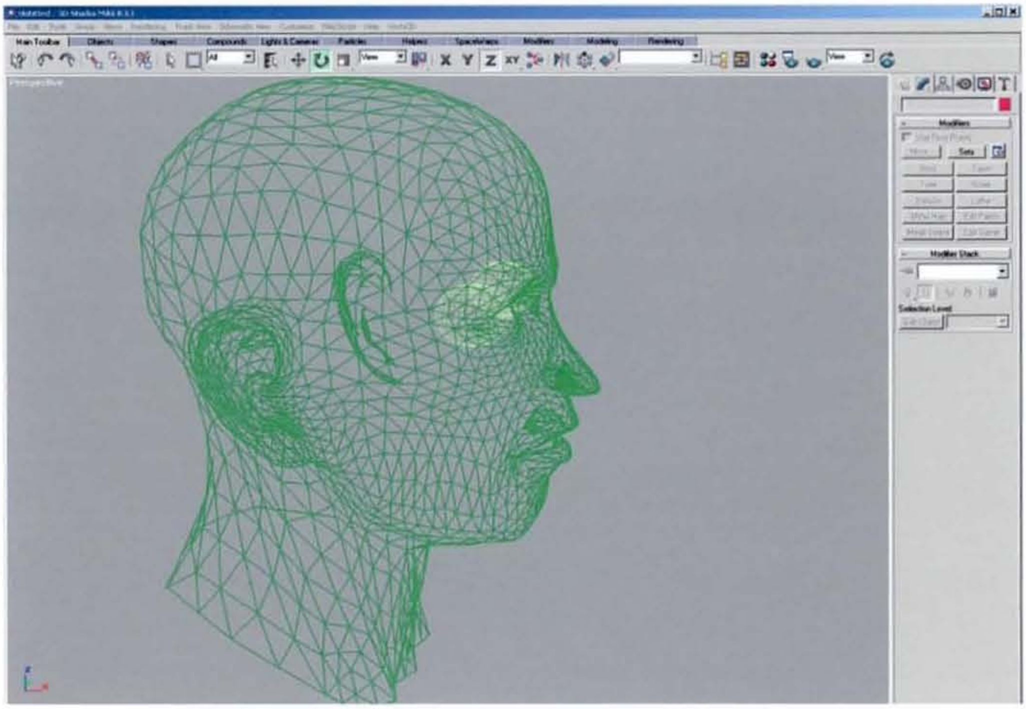


Figure 7.15 Side view of wire frame model of head generated by test data 4

Test Data 5

Describe – "the african woman has a thin nose and aquamarine eyes"

phrase, echo(english, [the, african, woman, has, a, thin, nose, and, aquamarine, eyes], heads),[[1]], ((0.0,0.0,1.0))

english, english(sentence([the, african, woman, has, a, thin, nose, and, aquamarine, eyes]), [the, african, woman, has, a, thin, nose, and, aquamarine, eyes], [the, african, woman, has, a, thin, nose, and, aquamarine, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [african])]), grammar(noun_phrase, [noun], [lexical(noun, [woman])])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [thin])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [aquamarine])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])]), heads),[[1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 27, 29, 30, 46]], ((0.0,0.0,1.0))

50, heads, descriptor([the, african, woman, has, a, thin, nose, and, aquamarine, eyes], [[object([woman]), [qualifiers([african], [])]], [object([eyes]), [qualifiers([aquamarine], [])]], [object([nose]), [qualifiers([thin], [])]]]),[[1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 23, 27, 33, 34, 35, 45, 50], [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 27, 29, 30, 46, 50]], ((0.0,0.0,1.0))

Modifiers Applied to head

```
woman
head texture = 6
head type = 2
head strength = 0.051
head x_pull = 77
head y_pull = 88
head z_pull = 74
head y_offset = 0
head z_offset = 0.334
head width = 1
head widthskew1 = -0.168
head widthskew2 = 0.194
head depth = 1
head depthskew = -0.284
head height = 1
head heightskew = 0.064
head face_squash = 0.76
head flatten = 0.7
head slope = 0
nose width = 1.628
nose width_zweight = 0.261
nose length = 0.28
nose length_zweight = 1
nose pullup = 1
nose bridge = 0.32
nose hook = 0.52
nose hook_influence = 0
chin extent = 0.562
chin tilt = 1.166
chin tilt_influence = 0
chin accent = 0.261
jaw width = 0.277
jaw influence = 0.48
jaw uniformity = 0.119
cheek extrude = 0.48
cheek zpos = 0.281
cheek curvatures = 0.145
cheek curvature_zpos = 0.258
cheek curvature_ypos = -0.613
cheek curvature_zfalloff = 0.214
cheek curvature_yfalloff = 0.715
eyes colour = 13
eyes separation = 1.542
eyes inset = 0.568
eyes toproundness = 0.274
eyes bottomroundness = 0.135
eyes rotation = 0
eyes brow_bulge = 0.137
ears height = 0.29
ears lobe = 0.209
ears depth = 0.313
ears rotation = -0.128
mouth protrude = 0.5
mouth width = 0.22
eye1translate x = 3.25
eye1translate y = -17.5
eye1translate z = 3.1
eye2translate x = -3.25
eye2translate y = -16.4
eye2translate z = 3
eye1rotate x = -1
eye1rotate y = 0
eye1rotate z = 0
eye2rotate x = -1
eye2rotate y = 0
eye2rotate z = 0
```

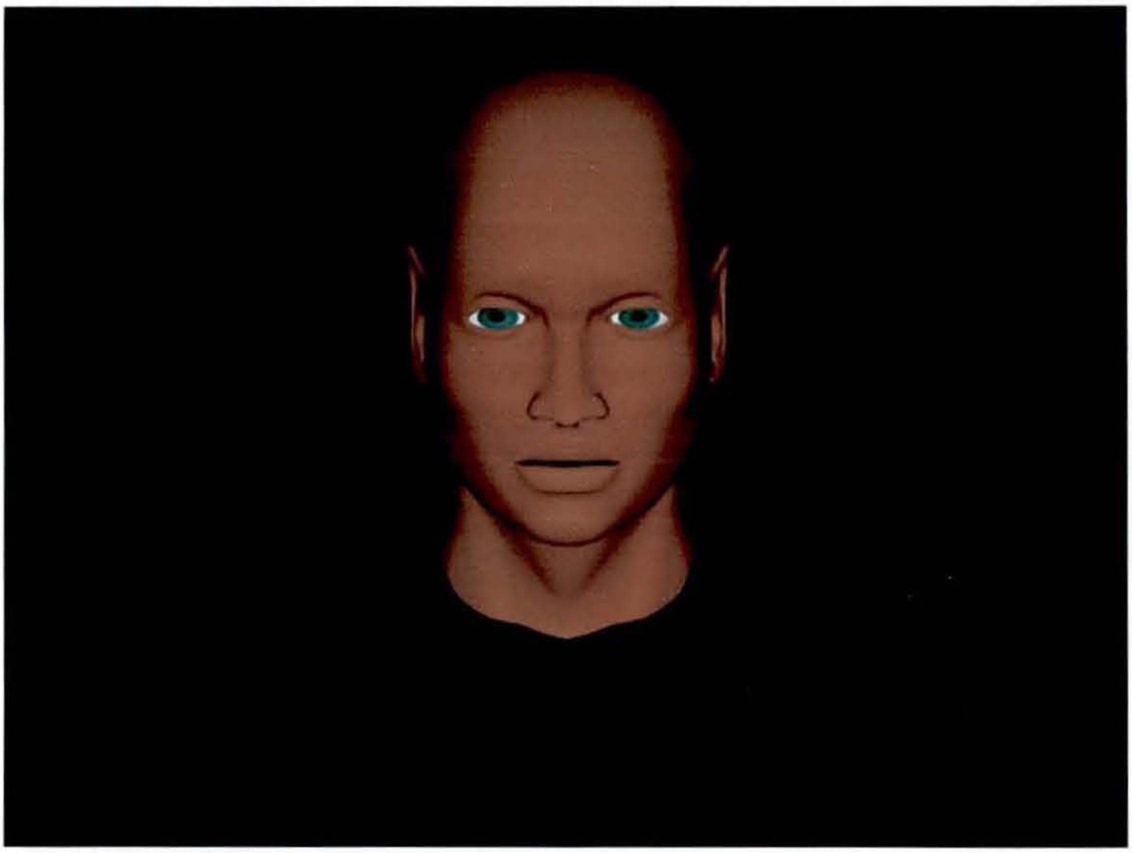


Figure 7.16 Rendered image of head generated from description "the african woman has a thin nose and aquamarine eyes"

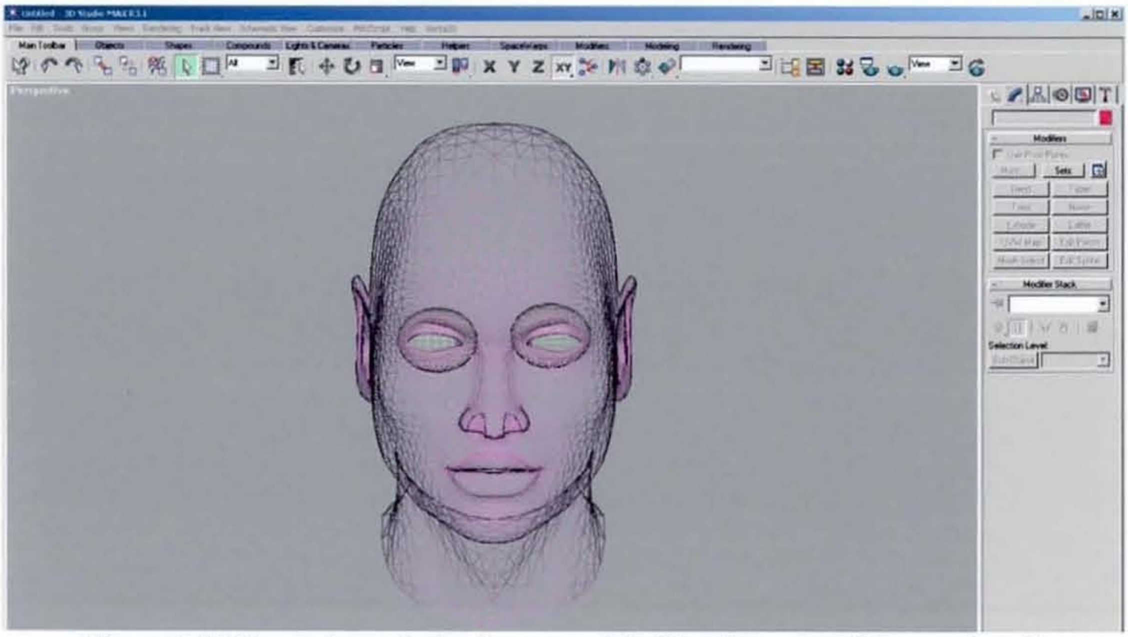


Figure 7.17 Front view of wire frame model of head generated by test data 5

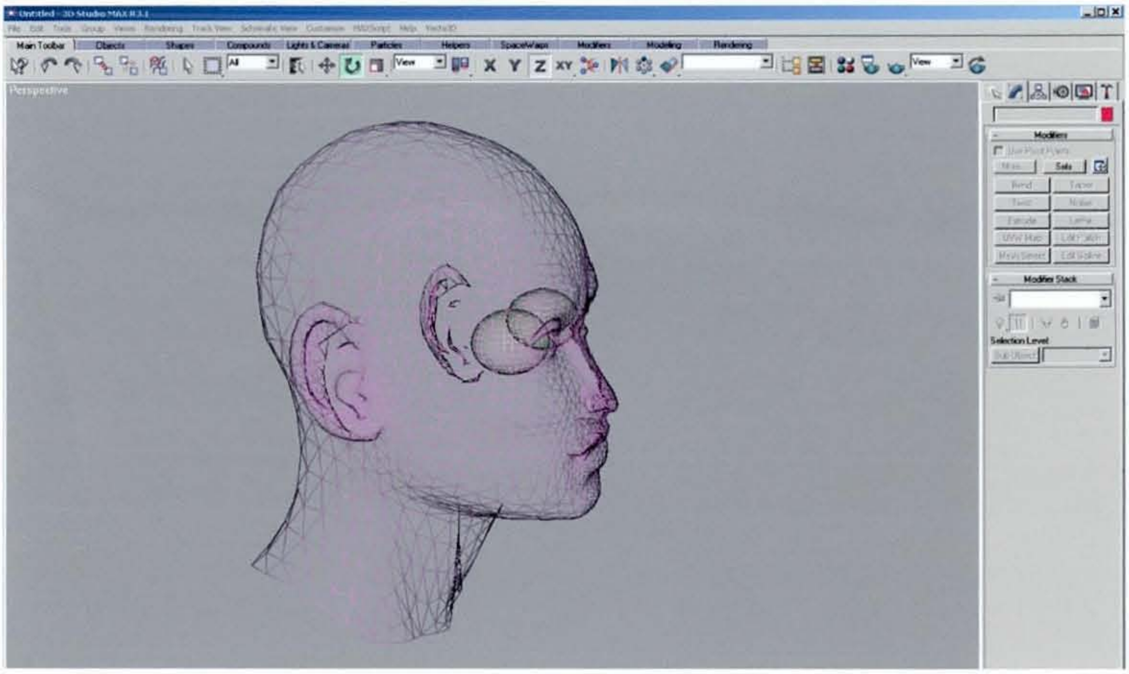


Figure 7.18 Side view of wire frame model of head generated by test data 5

Test Data 6

Describe - "the man has a large wide nose and vampire eyes"

phrase, echo(english, [the, man, has, a, large, wide, nose, and, vampire, eyes], heads),[[1]], ((0.0,0.0,1.0))

english, english(sentence([the, man, has, a, large, wide, nose, and, vampire, eyes]), [the, man, has, a, large, wide, nose, and, vampire, eyes], [the, man, has, a, large, wide, nose, and, vampire, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [noun], [lexical(noun, [man])])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [wide])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [vampire])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])]), heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 24, 25, 26, 46]],100 heads, descriptor([the, man, has, a, large, wide, nose, and, vampire, eyes], [[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([vampire], [])]], [object([nose]), [qualifiers([large, wide], [])]]],[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 24, 25, 26, 46, 51], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 24, 28, 29, 30, 45, 51]], ((0.0,0.0,1.0))

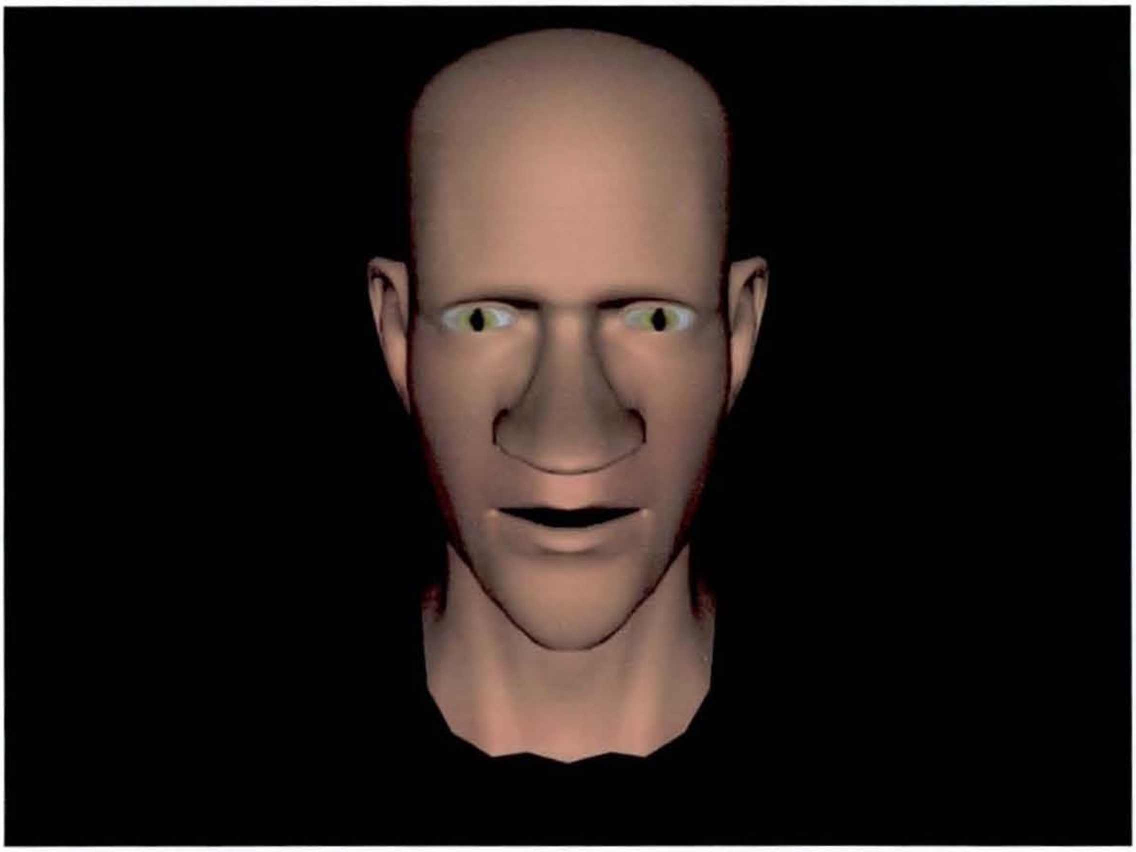


Figure 7.19 Rendered image of head generated from description "man has a large wide nose and vampire eyes"

Test Data 7

Describe – "the man has a very wide mouth and vampire eyes"

phrase, echo(english, [the, man, has, a, very, wide, mouth, and, vampire, eyes], heads),[[1]], ((0.0,0.0,1.0))

english, english(sentence([the, man, has, a, very, wide, mouth, and, vampire, eyes]), [the, man, has, a, very, wide, mouth, and, vampire, eyes], [the, man, has, a, very, wide, mouth, and, vampire, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]),

```
grammar(noun_phrase, [noun], [lexical(noun, [man])])]), grammar(verb_phrase,
[transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase,
[indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase,
[noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [adjective_phrase,
noun_phrase], [grammar(adjective_phrase, [adverb, adjective], [lexical(adverb, [very]),
lexical(adjective, [wide])])]), grammar(noun_phrase, [noun], [lexical(noun, [mouth])])]),
lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase],
[grammar(adjective_phrase, [adjective], [lexical(adjective, [vampire])]),
grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])])]), heads),[[41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 54, 55, 56, 58, 59, 60, 61, 79, 80, 81, 82]], ((0.0,0.0,1.0))
```

```
heads, descriptor([the, man, has, a, very, wide, mouth, and, vampire, eyes],
[[object([man]), [qualifiers([], [])]], [object([eyes]), [qualifiers([vampire], [])]],
[object([mouth]), [qualifiers([wide], [very])]]]),[[41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
52, 54, 55, 56, 58, 59, 60, 61, 79, 80, 81, 82, 84], [41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 54, 55, 56, 58, 59, 60, 61, 62, 75, 76, 77, 84]], ((0.0,0.0,1.0))
```

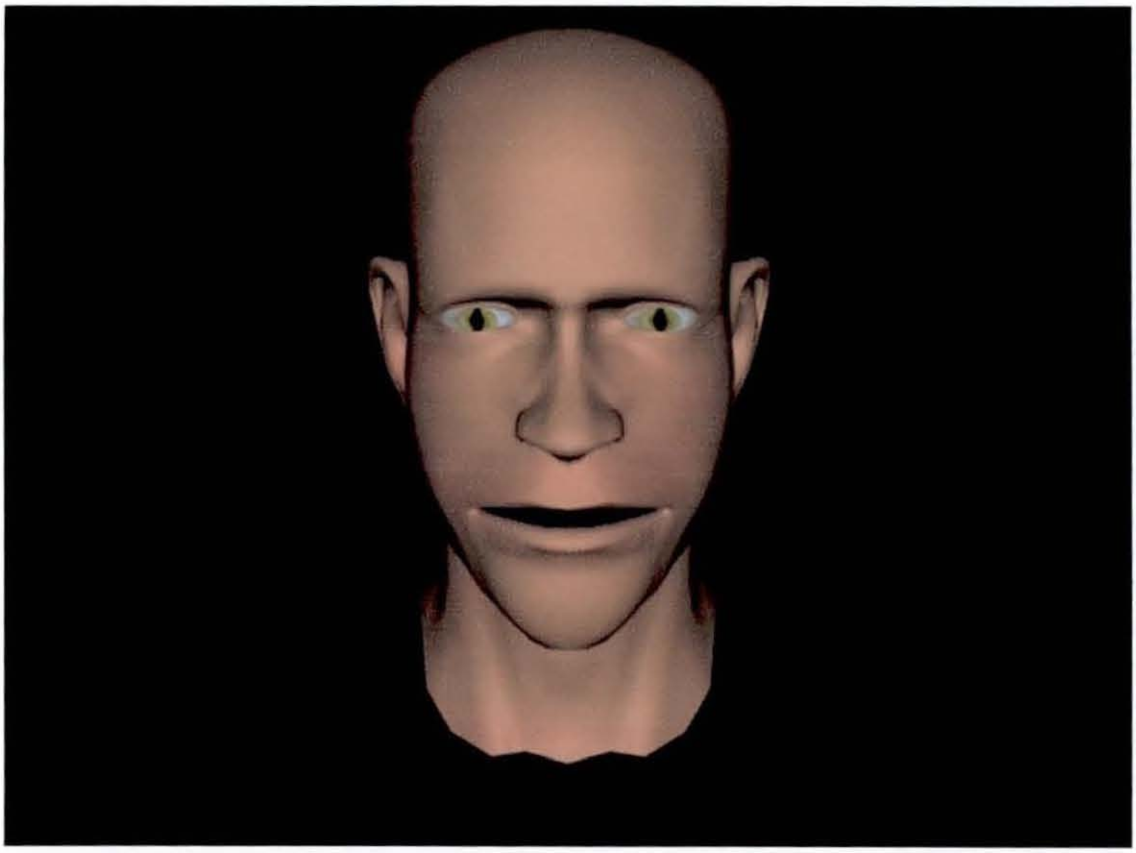


Figure 7.20 Rendered image of head generated from description "the man has a very wide mouth and vampire eyes"

Test Data 8

Describe – "the man has a bulbous nose and sunken cheek and slanting up eyes"

```
phrase, echo(english, [the, man, has, a, bulbous, nose, and, sunken, cheek, and,
slantingup, eyes], heads),[[1]], ((0.0,0.0,1.0))
```

```
english, english(sentence([the, man, has, a, bulbous, nose, and, sunken, cheek, and,
slantingup, eyes]), [the, man, has, a, bulbous, nose, and, sunken, cheek, and, slantingup,
eyes], [the, man, has, a, bulbous, nose, and, sunken, cheek, and, slantingup, eyes],
grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase,
[definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase,
[noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase],
[lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase],
[lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase],
[grammar(adjective_phrase, [adjective], [lexical(adjective, [bulbous])]),
grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [noun], [lexical(noun, [nose])]), lexical(conjunction, [and]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [sunken])]), grammar(noun_phrase, [noun], [lexical(noun,
[cheek])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase,
noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [slantingup])]),
grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])])])]), heads),[[1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 25, 27, 62, 104, 105, 106, 107, 108]],
((0.0,0.0,1.0))
```

```
heads, descriptor([the, man, has, a, bulbous, nose, and, sunken, cheek, and, slantingup,
eyes], [[object([man]), [qualifiers([], [])]], [object([cheek]), [qualifiers([sunken], [])]],
[object([eyes]), [qualifiers([slantingup], [])]], [object([nose]), [qualifiers([bulbous],
[])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25, 27, 48,
49, 80, 86, 87, 88], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 25, 27, 37, 80, 82, 83, 84], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 25, 27, 28, 44, 45, 46, 71, 80], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 28, 34, 35, 74, 80]], ((0.0,0.0,1.0))
```

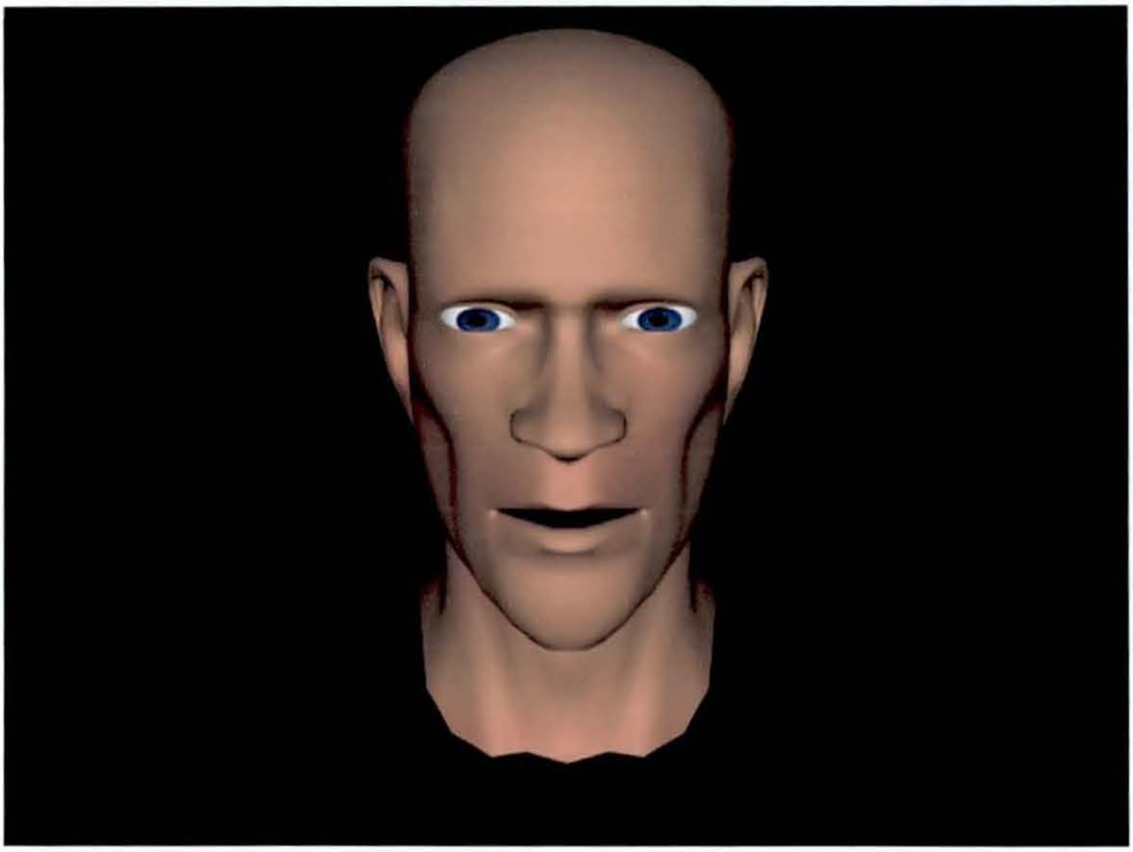


Figure 7.21 Rendered image of head generated from description "the man has a bulbous nose and sunken cheek and slanting up eyes"

Amplify - "the man has a bulbous nose and slightly puffed cheek and slanting down eyes"

phrase, echo(english, [the, man, has, a, bulbous, nose, and, slightly, puffed, cheek, and, slantingdown, eyes], heads),[[1]], ((0.0,0.0,1.0))

english, english(sentence([the, man, has, a, bulbous, nose, and, slightly, puffed, cheek, and, slantingdown, eyes]), [the, man, has, a, bulbous, nose, and, slightly, puffed, cheek, and, slantingdown, eyes], [the, man, has, a, bulbous, nose, and, slightly, puffed, cheek, and, slantingdown, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [bulbous])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],

```
[grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adverb, adjective], [lexical(adverb, [slightly]), lexical(adjective, [puffed])]),
grammar(noun_phrase, [noun], [lexical(noun, [cheek])])]), lexical(conjunction, [and]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [slantingdown])]), grammar(noun_phrase, [noun],
[lexical(noun, [eyes])])])])])])]), heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17,
18, 19, 20, 22, 23, 24, 26, 27, 29, 30, 97, 109, 110, 111]], ((0.0,0.0,1.0))
```

```
heads, descriptor([the, man, has, a, bulbous, nose, and, slightly, puffed, cheek, and,
slantingdown, eyes], [[object([man]), [qualifiers([], [])]], [object([cheek]),
[qualifiers([puffed], [slightly])]], [object([eyes]), [qualifiers([slantingdown], [])]],
[object([nose]), [qualifiers([bulbous], [])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 17, 18, 19, 20, 22, 23, 24, 26, 27, 29, 30, 40, 69, 70, 71, 113]], ((0.0,0.0,1.0))
```

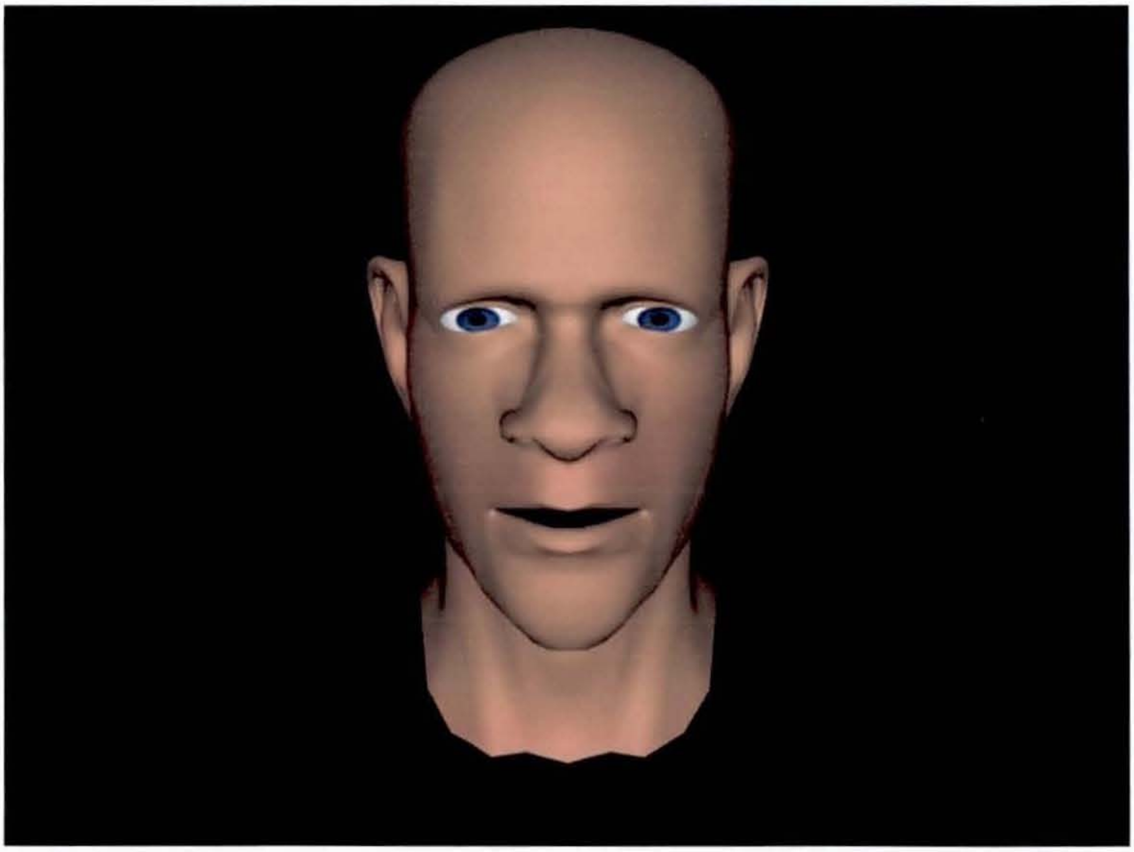


Figure 7.22 Rendered image of head generated from amplify instruction"the man has a bulbous nose and slightly puffed cheek and slanting down eyes"

Test Data 9

Describe – "the woman has small ears and a long nose and aquamarine eyes"

```
phrase, echo(english, the, woman, has, small, ears, and, a, long, nose, and, aquamarine,
eyes], heads),[[1]], ((0.0,0.0,1.0))

english, english(sentence([the, woman, has, small, ears, and, a, long, nose, and,
aquamarine, eyes]), [the, woman, has, small, ears, and, a, long, nose, and, aquamarine,
eyes], [the, woman, has, small, ears, and, a, long, nose, and, aquamarine, eyes],
grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase,
[definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase,
[noun], [lexical(noun, [woman])])]), grammar(verb_phrase, [transitive_verb,
noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [adjective_phrase,
noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [small])]),
grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase],
[grammar(noun_phrase, [noun], [lexical(noun, [ears])]), lexical(conjunction, [and]),
grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]),
grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase,
[adjective], [lexical(adjective, [long])]), grammar(noun_phrase, [noun], [lexical(noun,
[nose])])])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase,
noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective,
[aquamarine])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])])]),
heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25, 27,
58, 84, 85, 86, 87]], ((0.0,0.0,1.0))

heads, descriptor([the, woman, has, small, ears, and, a, long, nose, and, aquamarine,
eyes], [[object([woman]), [qualifiers([], [])]], [object([ears]), [qualifiers([small], [])]],
[object([eyes]), [qualifiers([aquamarine], [])]], [object([nose]), [qualifiers([long],
[])]]]),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27,
43, 74, 76, 77, 78], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 25, 27, 28, 40, 41, 67, 74], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 22, 23, 25, 27, 29, 30, 37, 38, 68, 74]], ((0.0,0.0,1.0))
```

The interpretation of the statements is based on baseline head models and modifiers taken from measurements of human heads and so their library of head geometry and modifiers provides context within which the statements are given form. By changing the set of head models and associated descriptions, the context may be changed. The interpretation of the natural language is thus based on the experience of the machine; and may arguably be termed "artistic"

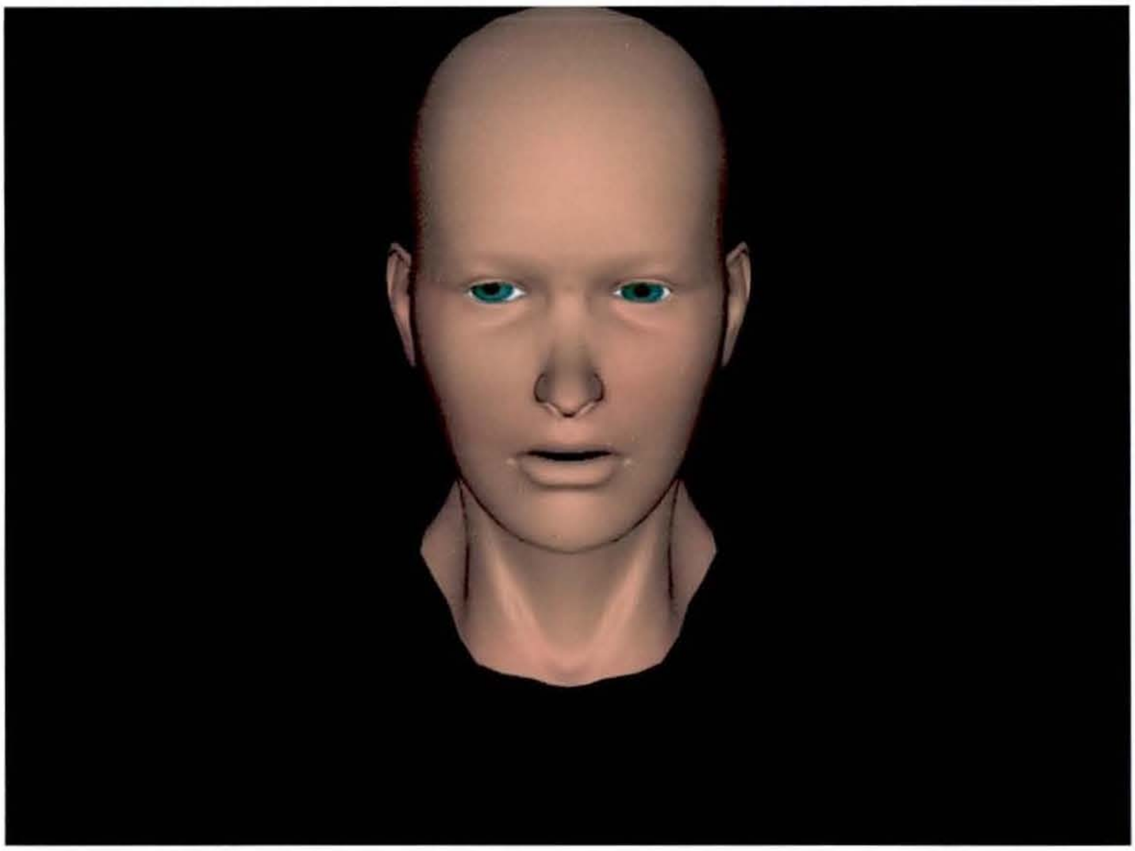


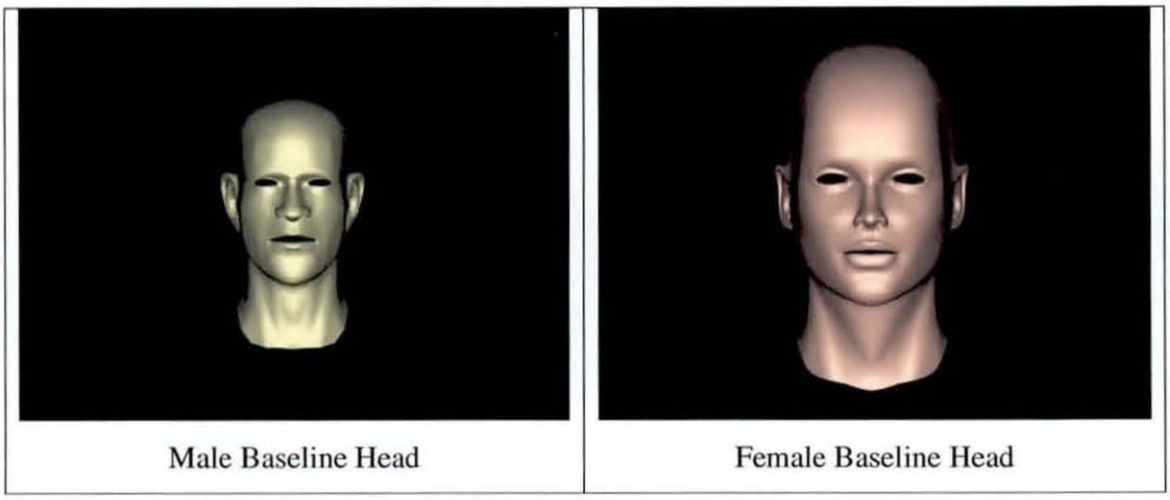
Figure 7.23 Rendered image of head generated from description "the woman has small ears and a long nose and aquamarine eyes"

7.4 Evaluation of Test Results

It is believed that the system operates reasonably well on the whole. Looking at test results from section 7.3 we can clearly see that the natural language interface/interpreter successfully processed every description, including the amplification or edit description. In all test cases user descriptions were successfully parsed by correctly identifying the grammar rule and lexical for each word (highlighted in yellow). Similarly the descriptor lists produced by the interpreter were apparent and understandable for each test case. The head parameters listed under test data 2 and 5 demonstrates the head engine's ability to translate the descriptors to parameters. Finally the rendered image shows the facial image constructed by the FIG module. The majority of images produced by test data 1 to 9 offer an acceptable representation of the description. It is true that the quality of image produced and the level of realism and recognition offered by the rendered images is questionable. However this point was stressed in chapter 4, section 4.7. Our aim was to demonstrate that geometric models of human faces can be constructed and controlled by natural language descriptions and that is what we hope the test results in section 7.3 confirm.

Even though the system appears to work satisfactorily there are moments when the individual modules can perform below expectation. For example if we look at the result from test data 7 it is reasonable to assess that the natural language interpreter successfully completed its operation and generated a correct list of descriptors. However observation of the rendered image of the head in Figure 7.20 reveals that the head engine is incapable to calculate mouth width in relation to the dimensions or proportion of the face. The result is a mouth that spreads beyond realistic limits. A similar evaluation can be deduced for nose width of the man in Figure 7.19, result of test data 6. This is an important result since it requires further enhancing the heads engine to cater for such measures. The features of any given face description are likely to be correlated, and so a statement about jaw size may influence mouth size. The current implementation has not catered for conflicting influences, however the inbuilt uncertainty handling mechanism was included precisely for those reasons, and provides a suitable platform for further work.

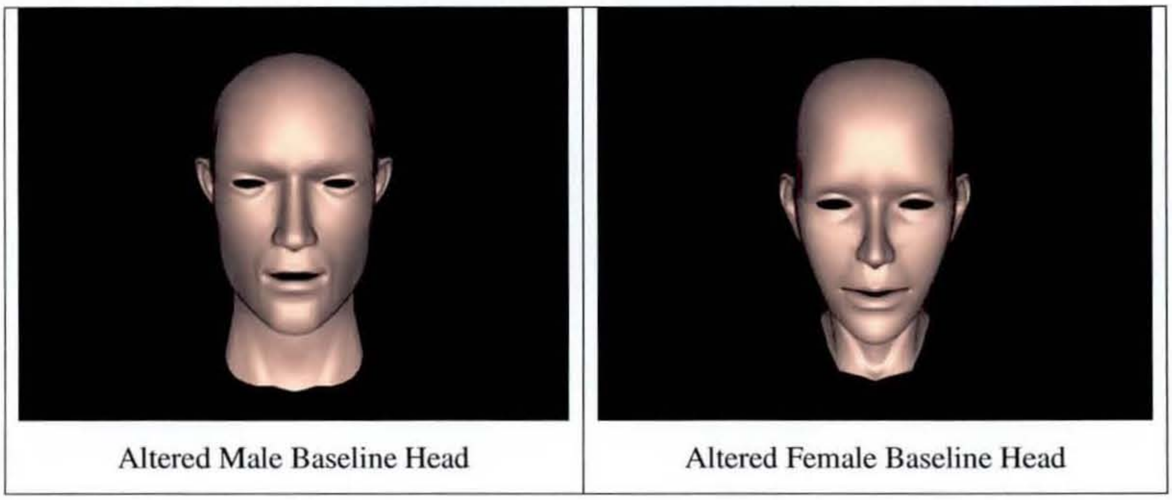
The interpretation of the "describe" and "amplify" statement is based on baseline head models and modifiers and so their library of head geometry and modifiers provides context within which the statements are given form. By changing the set of head models and associated descriptions, the context may be changed. The interpretation of the natural language is thus based on the experience of the machine; and may arguably be termed "artistic". This point can be demonstrated by changing the basic head template so that instead of loading the template male or female head (Figure 7.24) the FIG module loads completely different head geometry (Figure 7.25) and applies head parameters generated by the NLI to it.



Male Baseline Head

Female Baseline Head

Figure 7.24 Male and female head templates used by the FIG module



Altered Male Baseline Head

Altered Female Baseline Head

Figure 7.25 Male and female head templates modified to different geometry configuration

If we run Test Data 8 and 9 again with the new head templates implemented we get the following facial images shown in Figure 7.26 to 7.28.

Test Data 8

Describe – "the man has a bulbous nose and sunken cheek and slanting up eyes"

Result :-

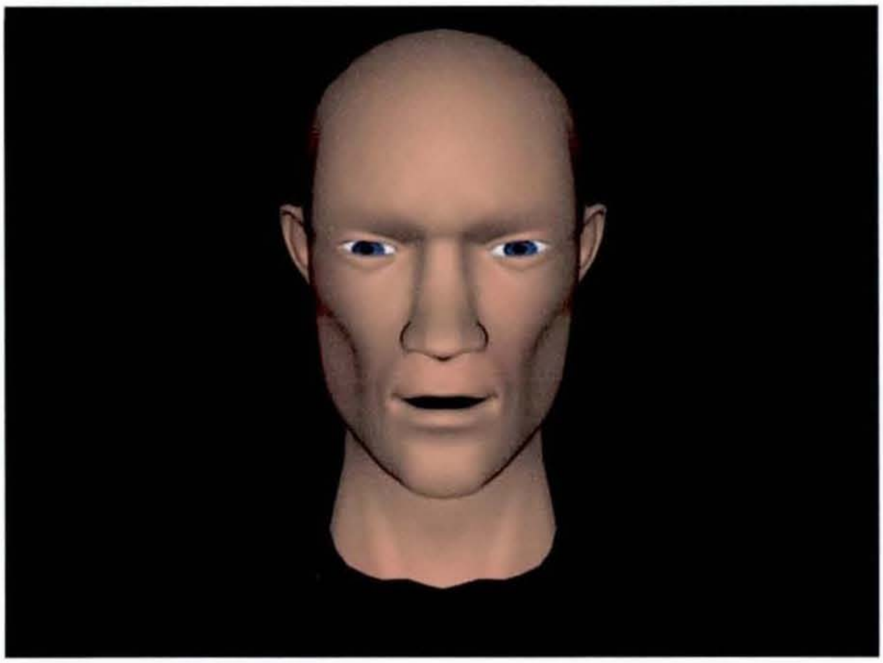
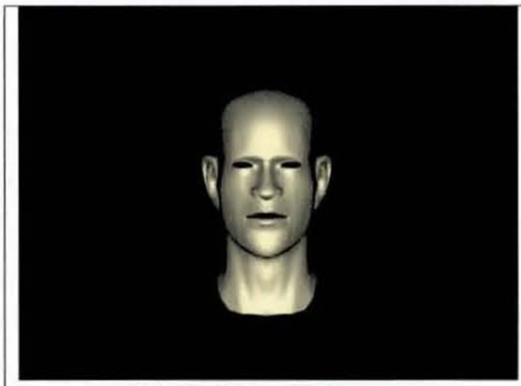
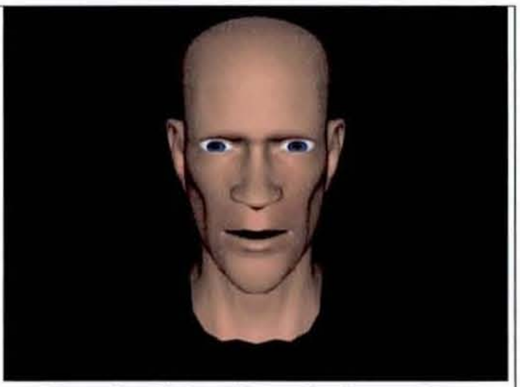


Figure 7.26 Rendered image of head generated from description "the man has a bulbous nose and sunken cheek and slanting up eyes" using new male baseline head geometry

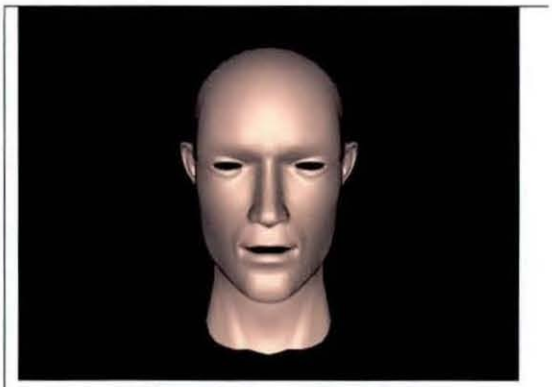


Original Head Template

▯

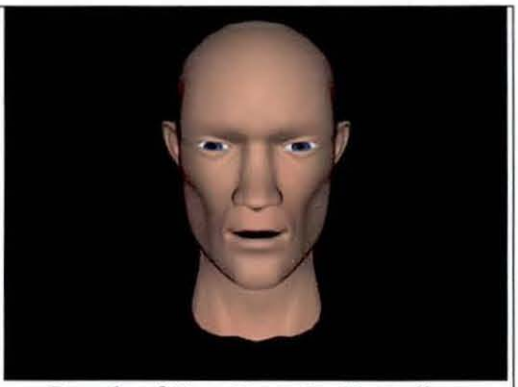


Result of Test Data 8 - Describe



Altered Head Template

→



Result of Test Data 8 - Describe

Amplify - "the man has a bulbous nose and slightly puffed cheek and slanting down eyes"

Result:-

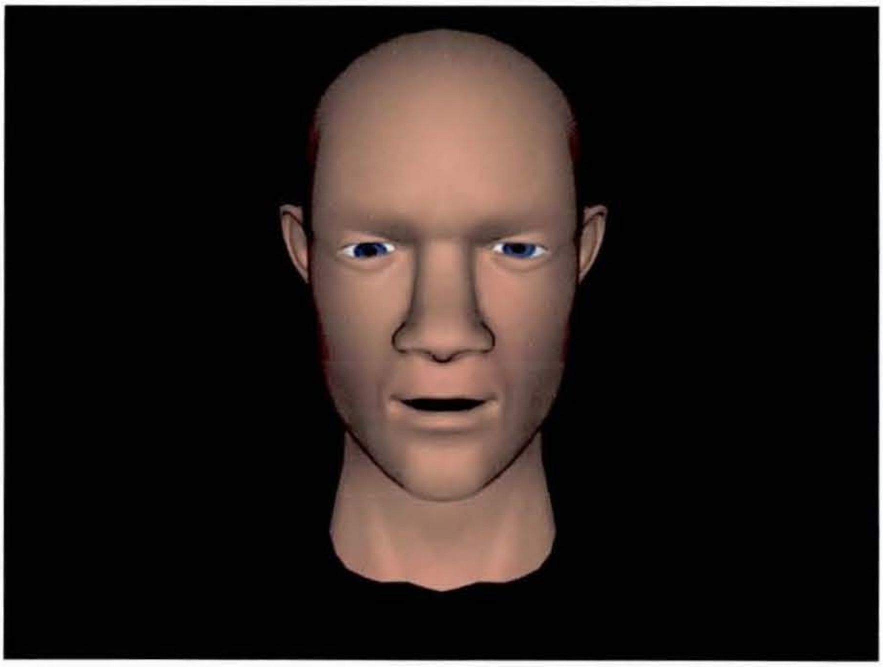
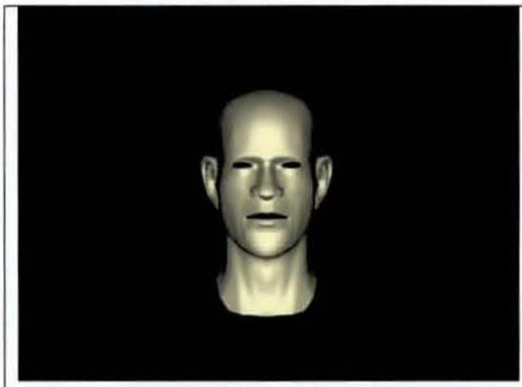
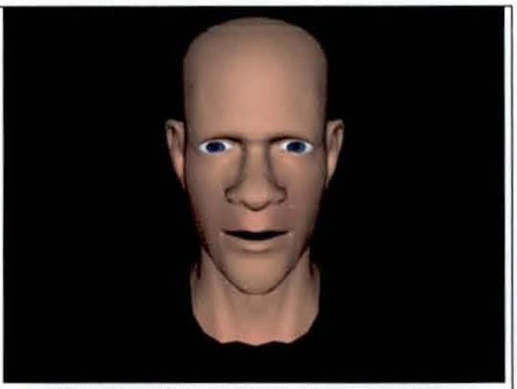
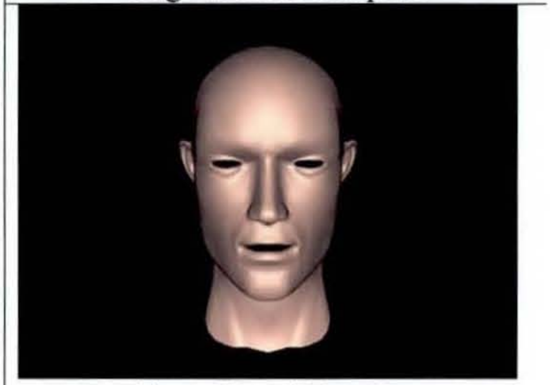
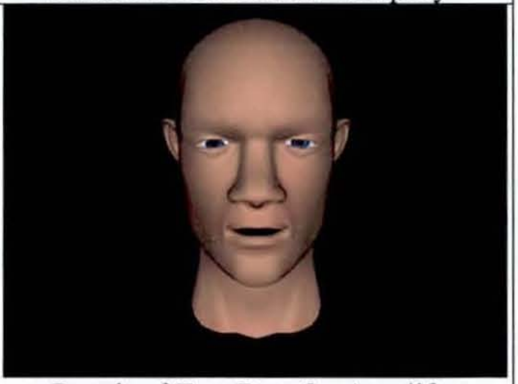


Figure 7.27 Rendered image of head generated from amplify instruction"the man has a bulbous nose and slightly puffed cheek and slanting down eyes" using new male baseline head geometry.

	▯	
Original Head Template		Result of Test Data 8 - Amplify
	→	
Altered Head Template		Result of Test Data 8 - Amplify

Test Data 9

Describe – "the woman has small ears and a long nose and aquamarine eyes"

Result:-

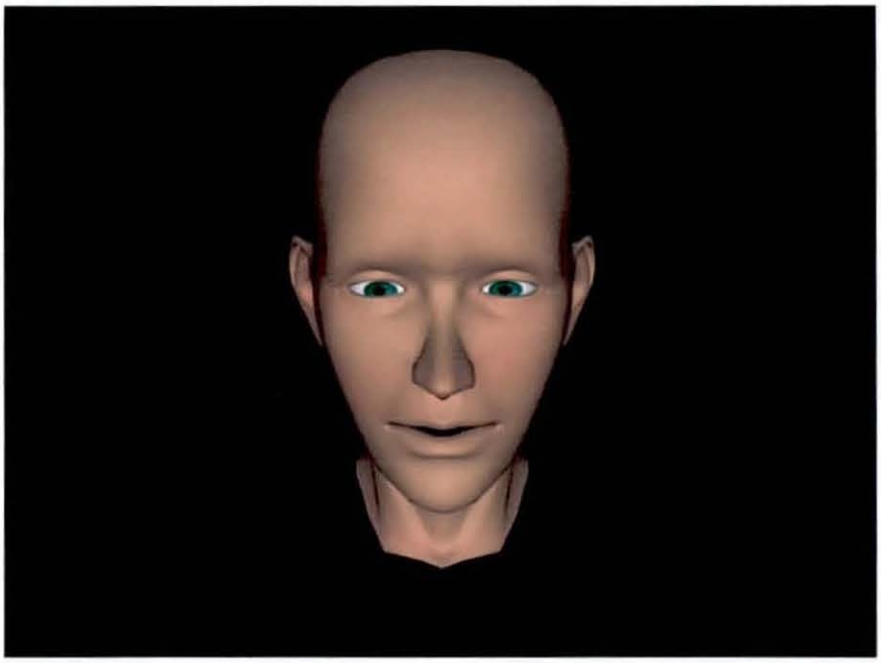
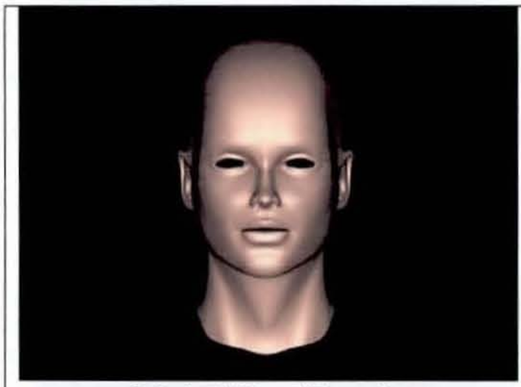
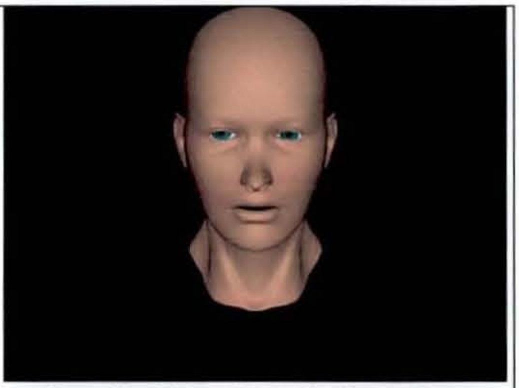
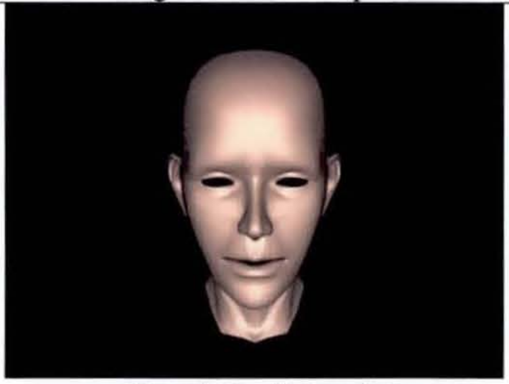
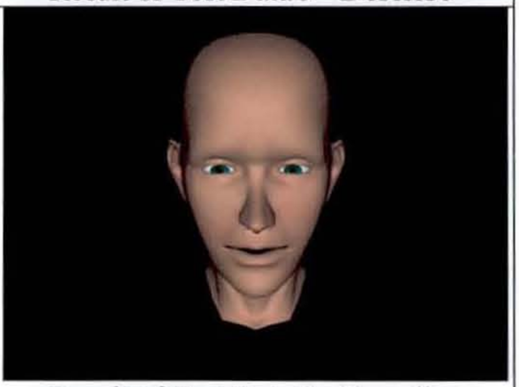


Figure 7.28 Rendered image of head generated from description "the woman has small ears and a long nose and aquamarine eyes" using new female baseline head geometry

 Original Head Template	▯	 Result of Test Data 9 - Describe
 Altered Head Template	→	 Result of Test Data 9 - Describe

The change of head geometry dramatically altered the resulting facial image even though the same describe statements are used. In this framework it can be argued that the head geometry and modifiers provides context within which the 'describe' and 'amplify' statements are given form. By changing the set of head models, the context may be changed.

7.5 Conclusion

This chapter has explored the design and architecture of the final research system proposed to generate 3D facial images via a natural language interface. It has explained how the various modules identified in chapter 1 and elaborated through the course of this thesis integrate and work together. Further more a structured test sequence was described, demonstrating the operational capacity of the system to successfully interpret and translate linguistic data to numeric parameters. The final result of each test was a rendered image of the 3D head constructed using the aforementioned parameters. It is a safe conjecture that the images represent the description fairly accurately. The chapter finishes off with an evaluation of the test results concluding that the system works in sense that it can successfully process natural language descriptions of faces, generate approximately accurate head parameters and construct 3D image of a face that reflects the original description. Some shortfalls of the system have also been highlighted concerning specific modules such as the head engine and its ability to control head geometry with finer precision in respect to head and feature proportions.

Chapter 8

Thesis Summary, Conclusions and Future Work

8.1 Thesis Summary

The focus and motivation behind this thesis was to develop a system by which 3D images of human faces could be constructed using a natural language interface. The driving force behind the project was the need to create a system whereby a machine could be made to perform an artistic task without requiring a complex control system that only skilled professionals with artistic talent can operate. The interface for such a system needed a simple and natural input mechanism doing away with a complex control structure of menus and panels of brushes and 2D /3D art tools. Hence the idea of a natural language interface since words are the most common and basic means of learning, teaching and communicating. The research was never meant to create a facial composite system like Identi-kit or E-fit. Instead the research was specifically geared towards discovering how 3 Dimensional facial images can be constructed and edited using a natural language interface.

We have presented two main methods for achieving this aim,

1. Use of a fuzzy truth maintained blackboard system to interpret and translate linguistic data
2. Use of free form deformation modifiers to parameterise and control geometry of pre-constructed 3D head models in a commercially available 3D modelling system which has pre prepared scripts to access and control templates and modifiers obtained from measurements of 3D human heads.

Both methods are diverse looking at two separate disciplines of research however in context of this thesis they are strongly connected and intertwined to solve the thesis aim.

Chapter 1 presented a description of the research aim and outlined the methodology, approach, and processes proposed to tackle the research problem.

Chapter 2 examined the human facial structure; it partly looked at medical definitions of facial structure such as the bone and muscle that give faces structure and allow facial expressions. In greater detail it identified the physical parts of a face that make faces recognisable. The work of Fredric Parke (Parke, 1982) was acknowledged for his pioneering work 'on facial animation and defining techniques to parameterise faces for artificial composition and animation. Recent work by artists like Faigin (1990) was also acknowledged.

It also looked at the work of Ellis (Ellis *et al.*, 1975) and Shepherd (Shepherd *et al.*, 1977) on facial recognition and verbal descriptions. This provided beneficial insight into what areas of a face people usually remember and recall most frequently. This information helped in planning and executing surveys necessary to acquire important data on the language ordinary people use to describe faces. Finally we examined the survey results and compiled a list of most commonly used descriptors for the lexical database in the natural language processing engine.

Chapter 3 provided an exhaustive examination of the tools and techniques available for modelling 3D objects. The chapter investigated three main areas.

1. Facial modelling - existing research and applications.
2. Representation techniques available for 3D modelling and technology available for acquiring facial data
3. Tools and technology available for constructing 3D human head geometry.

This chapter also looked at some other technologies in the domain of 3D modelling such as FFD that proved critical in the development of the 3D facial image generation module.

Chapter 4 described the facial image generation module. It described construction of a 3D head model using NURBS, Bezier patches and Polygon meshes. The Head geometry constructed using polygon was evaluated to be the best out of all three modelling procedures. The baseline heads constructed using polygon mesh formed the foundation for the facial image generation module. The baseline heads were parameterised using FFD modifiers attached to the head geometry. Each modifier was catalogued and assigned variables acting as parameters that could be edited using Maxscript.

The chapter concluded assessing efficiency and capability of the FIG module in its capacity to generate heads of different shapes, sizes and features. The heads generated by the FIG module, although not photorealistic, were qualitatively acceptable to test with the natural language interface module.

Chapter 5 presented some important uncertainty handling theories, ranging from probability theory to fuzzy set theory, mass assignment, semantic unification and truth maintenance systems. Examination of fuzzy logic offered a linguistic perspective to human computer interaction methodologies, and how natural language can play an important part in our managing uncertainty. Finally the importance of fuzzy numbers was mentioned, particularly in the use of fuzzy hedges as an important component of this thesis for processing natural language descriptions of faces. We also looked at TMS especially ATMS which forms an integral part of the Natural Language Interface in interpreting natural language description of faces.

Chapter 6 investigated natural language processing, the various areas of study connected with it. It inspected the anatomy of language, orthographic structure, grammar and components of grammar. It discussed computational tools such as Parsing, PROLOG, Echo and the Truth Maintained Blackboard system. It presented a new system for interpreting natural language sentences using a black board system with an assumption based truth maintenance system. It also presented details on the heads engine and how it can translate linguistic data into parameters for the facial image generation module.

Chapter 7 presented the design and architecture of the final system to generate 3D facial images via a natural language interface. It explained how the various modules identified in chapter 1 and elaborated through the course of this thesis integrate and work together.

Further more a structured test sequence was described, demonstrating the operational capacity of the system to successfully interpret and translate linguistic data to numeric parameters. The final result of each test was a rendered image of the 3D head constructed using the aforementioned parameters. The chapter concluded with the assessment that the system worked successfully in processing natural language descriptions of faces and generating 3D facial images that reflected the original description. It also pointed to some interesting results derived from certain test data indicating shortfalls in the head engine's ability to control head geometry with finer precision in respect to head and feature proportions. However the inbuilt uncertainty handling mechanism was included precisely for this reason, and provides a suitable platform for further work

If we briefly revisit the thesis aim as laid out in chapter 1, then our objective was to:

1. investigate whether 3D human face models can be constructed and modified using a rudimentary natural language interface and
2. if the facial images constructed can pass as recognizable human faces

We believe the main objective of this thesis has been attained and the system developed and reported in this dissertation offers strong evidence of success in achieving the main thesis aim. The second thesis aim concerned with examining if the facial images constructed can pass as recognizable human faces is difficult to conclude. The imagery produced by the system can easily be regarded as human like but how realistic in terms of, accuracy of representation and level of recognition is an open question. The 3D heads constructed by the FIG module lack accessories like hair, eyebrows, teeth and facial hair. These are important factors in determining level of realism and recognition in human faces.

8.2 Future Work

The existing setup uses a combination of TMS and fuzzy mass assignment to handle uncertainty in natural language descriptions of faces. Future work should involve a more thorough implementation of mass assignment and semantic unification to existing natural language interpreter to allow better handle of uncertainty in processing more varied and diverse descriptions of faces. It is envisaged that combination of TMS and mass assignment will improve both interpretation and translation of linguistic data.

The application of the modifiers to baseline template using a normal distribution function is a commutative operation. Modifiers either increment or decrement the baseline template parameters depending on the descriptor. This provides scope for further work in implementing fuzzy blending between sets of facial features to correlate features of any given face description such that related features may be able to influence each other..

Further improvements can be made to the facial image generation module, particularly by adding accessories such as hair, eyebrows, hats and glasses to the existing object library. Within the duration of this research such accessories could not be developed due to the author's inability to construct objects of reasonable quality and usefulness. It is reasonable to assess that implementing such accessories will enhance the quality of facial imagery generated by the system. This will result in higher level of realism and recognition, perhaps to the extent that the system could be used as a natural language based facial image composite system for identification purposes.

8.3 Conclusion

This thesis presented a novel approach to constructing 3D human faces. It is the first to look at constructing and modifying facial image artwork using a natural language interface.

Specialised modules were developed to control geometry of 3D polygonal head models in a commercial modeller from natural language descriptions. These modules were produced from research on human physiognomy, 3D modelling techniques and tools, facial modelling and natural language processing.

This work used two main methods sequentially for synthesising 3D facial images from natural language descriptions:

3. Use of a fuzzy truth maintained blackboard system to interpret and translate linguistic data which produces parameters for free form deformation modifiers to parameterise and control pre-constructed 3D head models.
4. A commercially available 3D modelling system which has pre prepared scripts to access and control head templates and modifiers obtained from measurements of 3D human heads.

A novel method of abstracting standard face images, modifiers and hedges was described where base head templates were obtained by distilling out the modifiers and modifiers obtained by differencing the modified object from a base template.

The interpretation of the natural language descriptions was based on baseline head models and modifiers taken from measurements of human heads and so the library of head geometry and modifiers provided context within which the statements were given form. By changing the set of head models and associated descriptions, the context could be changed as demonstrated in chapter 7. The interpretation of the natural language is thus based on the experience of the machine; and may arguably be termed “artistic”. The resultant facial images were consistent with the descriptions although it proved difficult to obtain detailed descriptions of faces that resulted in a recognisable match. The work has shown that it is possible to derive images that match the descriptions but that the descriptions used are insufficient to completely describe a given face. The derived templates and modifiers influence the set of faces produced from any given set of descriptions, and form the basis by which the system interprets the natural language statements.

The existing system has implemented a partial fuzzy logic solution. Rather than using a complete set of fuzzy rules and fuzzy membership functions, the system relied on specific concepts of fuzzy logic, in particular fuzzy hedges. The translation of natural language descriptions to parameters was handled so efficiently by the template, modifier tabular schema that the simplicity and robustness of the solution was accepted and adopted. There is however scope for further work using fuzzy logic, mass assignment and the inbuilt TMS based uncertainty handling mechanism to correlate features of any given face description such that related features may be able to influence each other.

This Page has been left blank intentionally!

Appendix A

Explanation of Terminology used in Chapter 2 and Survey Forms

Youngs Modulus:

The stress – strain ratio measured along the longitudinal axis of a material. Stress is applied to the longitudinal axis. Strain is measured as extension along this axis.

$$E = \frac{\partial P}{\partial L} . L$$

Poissons Ratio:

The Ratio of transverse strain to longitudinal strain of a material under stress.

Front page of survey web site and a sample questionnaire page.

A.I. FIGS Research Survey

Welcome to A.I. FIGS Research Survey Page. The Artificial Intelligence based Facial Image Generation System is a Gradients research project aimed at teaching a computer to build human faces in 3D through natural language descriptions.

In order for a human face to be generated, data about the face is needed and that is where you can help us. We need to examine how people describe faces. To be more specific we want to analyze the phrases and words you use to describe a human face.

Instructions

To begin click on an image thumbnail at the bottom of the page to go to a questionnaire form. Observe the facial image then fill in the survey form, once you have completed the form hit the submit button at the bottom of the form to send the data to us.

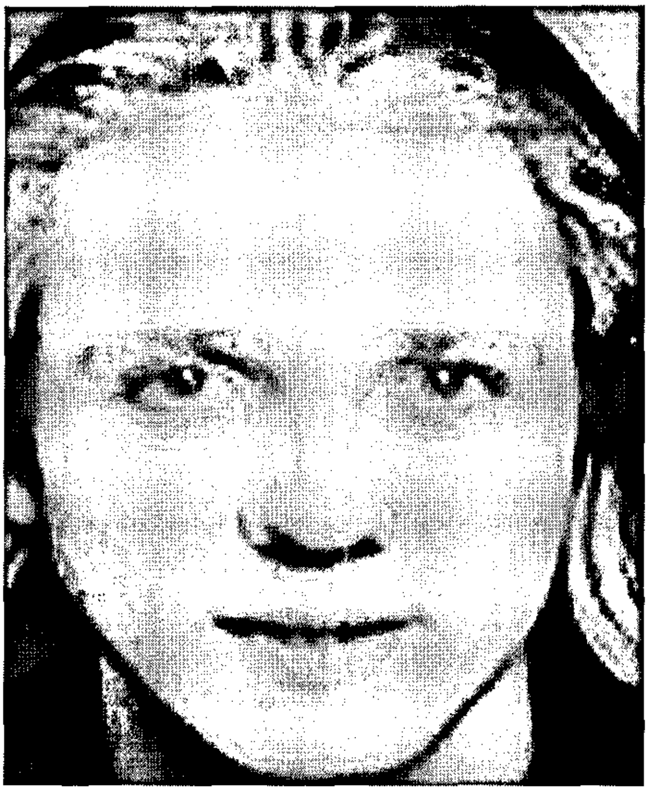
The survey consists of a number of different facial image samples. We request that you submit a minimum of three samples for data coherency reasons. You do not have to submit all three samples at the same time. The survey has been designed so that you can return to it when ever you have some time to spare. You can browse through the sample images and take your pick. Any samples that you have submitted earlier, simply ignore and choose a different one.

Thank you for your co-operation. Your help is most appreciated.

Regards

Salman Ahmad
Dr. Chris Hinde
Gradients Research Group
Computer Science Department
Loughborough University

Sample of Questionnaire Form:



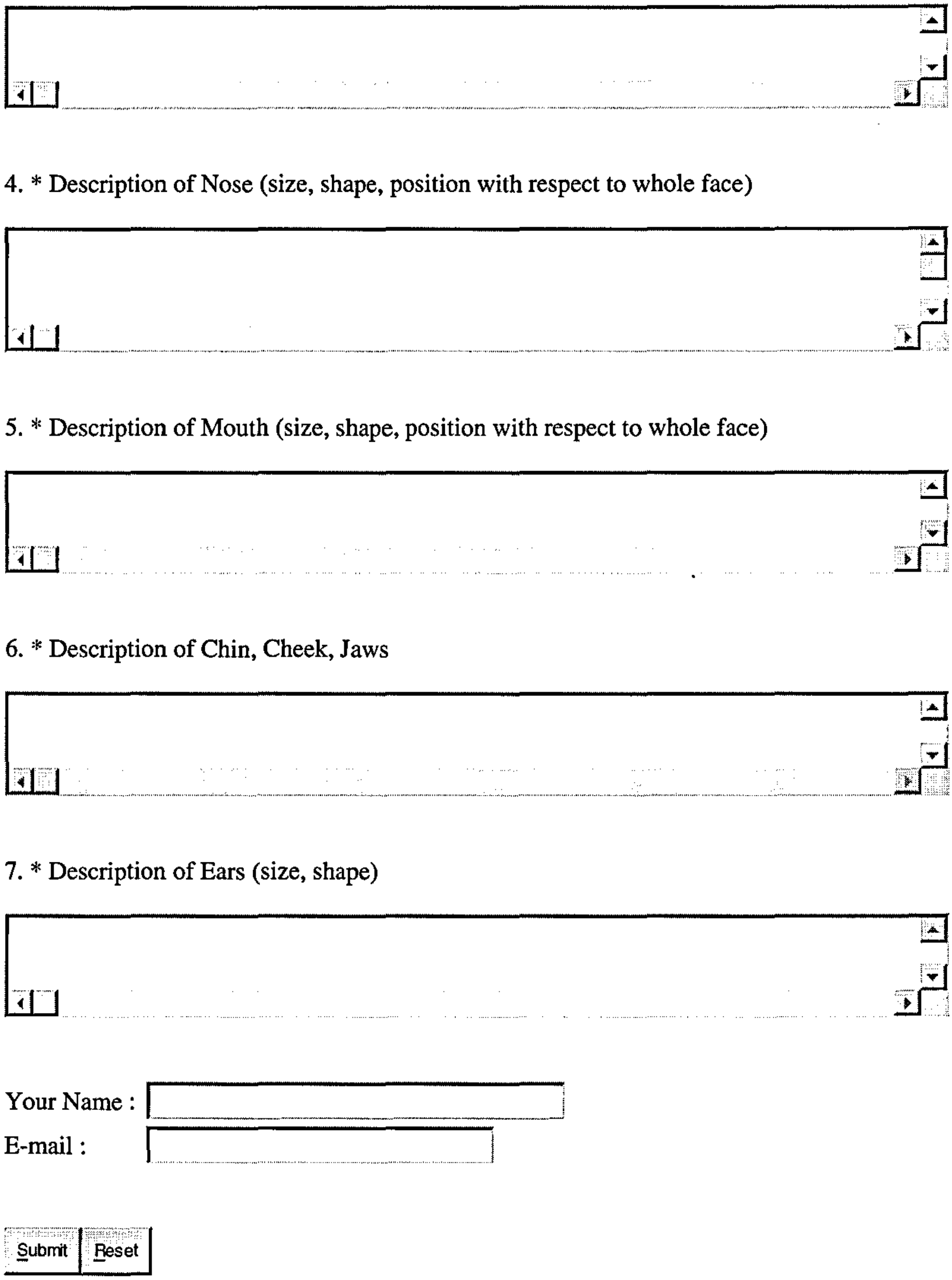
Please describe the above sample face, you may include information about size, shape and positioning of features with respect to the face such as eye spacing, forehead size, mouth width, eye width, etc. Size of head, skin texture, hair.

[Questions marked with "*" are required fields.]

1. * Description of general size, shape and skin texture of head

2. * Description of Hair (Hair Style i.e. curly, straight, etc)

3. * Description of Eyes, Eyebrows (size, shape, spacing, position with respect to whole face)



4. * Description of Nose (size, shape, position with respect to whole face)

5. * Description of Mouth (size, shape, position with respect to whole face)

6. * Description of Chin, Cheek, Jaws

7. * Description of Ears (size, shape)

Your Name :

E-mail :

Sample of Survey Results Received:

Subject: Survey sample06-Man4

Head = head large skin spotty
Hair = wavy
Eyes = eyes small round eyebrow well marked, straight
Nose = nose large blunt
Mouth = mouth quite large, well shaped
Cheek_chin_jaw = chin pointed
Ears = ears medium
Name =
email =

Subject: Survey sample06-Man4

Head = Large square-shaped head on a large neck; pale white skin with a number of blemishes.
Hair = Light coloured straight hair, quite long.
Eyes = Quite small eyes, widely spaced; dark patches under the eyes, eyebrows very close to eye sockets and indistinct. Normal eye shape.
Nose = Large and wide nose, bulbous at the base.
Mouth = Broad-lipped mouth, quite narrow compared to size of face.
cheek_chin_jaw = Cheeks full, but jaw bone very angular and an angular, protruding chin.
Ears = Ears not visible.
Name = Stephen McCoy
email = s.a.mccoy@lboro.ac.uk

Subject: Survey sample01-AfricanMan

Head = Large round overweight shaped face , dark smooth skinned
Hair = Short, dair cury hair receeding from forhead
Eyes = Heavy looking eyes thickness unde lower lid, bright laerge, dark eyes, well spaced with thick curved dark eybrows. Thicker at nose end narrowing out towards ears
Nose = Large flat nose with large nostrics central to face
Mouth = Wide mouth with thick lips normal type for Africans
Cheek_chin_jaw = Noticible cheeks round heavy jowl, slight stuble on chin
Ears = Fairly small close to head
Name = Jo McOuat
email = J.Mcouat2@lboro

Papers submitted for review and publication to journals and conference from the work in this thesis:

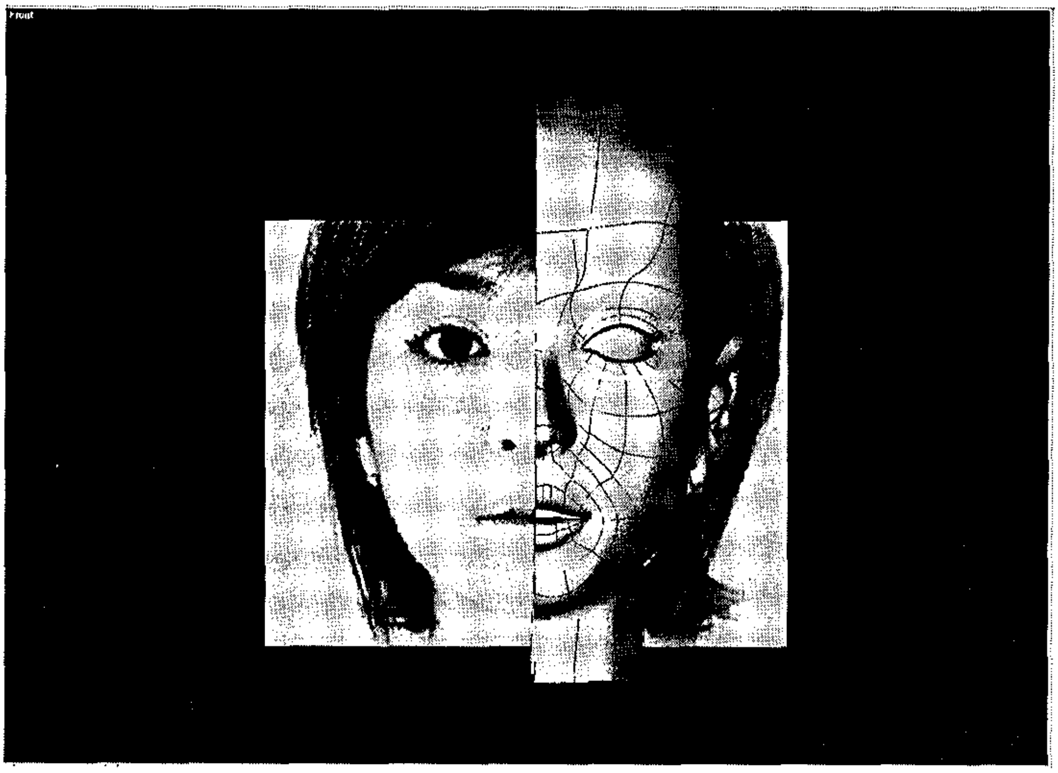
- Ahmad, S. and Hinde, C. J. (2001). Painting with Words. Submitted to Human Computer Interaction Journal. Oct, 2001.

- Ahmad, S. and Hinde, C. J. (2001). Constructing and Parameterising a Human Head using FFD inside 3D Studio Max. Submitted to Computer Graphics Journal. Nov, 2001.

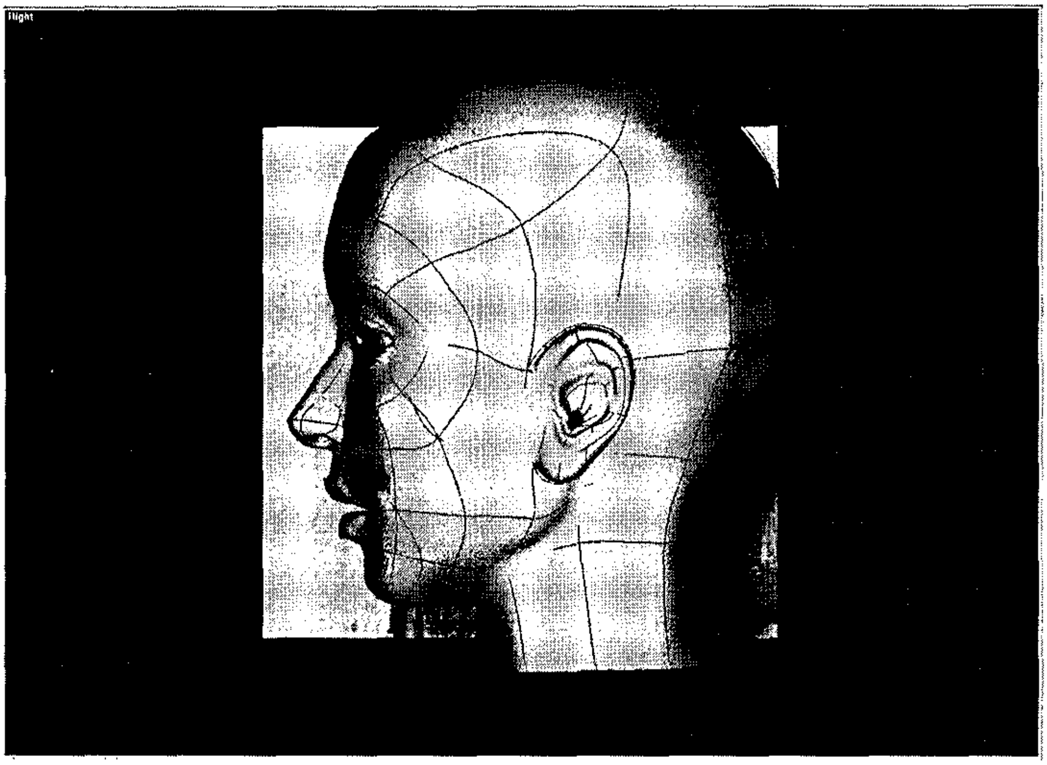
Appendix B

Facial Image Generation Module – Reference Material and HG Script

Reference Images used for construction of Spline layout in the three modelling procedures:



Front View



Side/Profile View

Head Generator Script ver 1.0 automatically loaded on start-up of 3D Studio Max

Code Listing

```
/*********************************************************************/

include "hdspprt.mse"

-- Use support script file, necessary for modifying correct FFD head variable called by
code statements in this script.

hdparam_array1 = #()                                -- Head Parameters array1
hdparam_array2 = #()                                -- Head Parameters array2

string_size = #()                                   -- String Array

-- Parameter for string truncate operation used to extract data from head parameters
file

string_size[1] = 15
```

```
string_size[2] = 12
string_size[3] = 16
string_size[4] = 14
string_size[5] = 14
string_size[6] = 14
string_size[7] = 16
string_size[8] = 16
string_size[9] = 13
string_size[10] = 18
string_size[11] = 18
string_size[12] = 13
string_size[13] = 17
string_size[14] = 14
string_size[15] = 18
string_size[16] = 19
string_size[17] = 15
string_size[18] = 13
string_size[19] = 13
string_size[20] = 21
string_size[21] = 14
string_size[22] = 22
string_size[23] = 14
string_size[24] = 14
string_size[25] = 12
string_size[26] = 22
string_size[27] = 14
string_size[28] = 12
string_size[29] = 22
string_size[30] = 14
string_size[31] = 12
string_size[32] = 16
string_size[33] = 17
string_size[34] = 16
string_size[35] = 13
string_size[36] = 19
string_size[37] = 23
string_size[38] = 23
string_size[39] = 27
string_size[40] = 27
string_size[41] = 14
string_size[42] = 18
string_size[43] = 13
string_size[44] = 19
string_size[45] = 23
string_size[46] = 16
string_size[47] = 18
string_size[48] = 13
string_size[49] = 11
string_size[50] = 12
```

```
string_size[51] = 15
string_size[52] = 17
string_size[53] = 14
string_size[54] = 18
string_size[55] = 18
string_size[56] = 18
string_size[57] = 18
string_size[58] = 18
string_size[59] = 18
string_size[60] = 14
string_size[61] = 14
string_size[62] = 14
string_size[63] = 14
string_size[64] = 14
string_size[65] = 14


-- Get Name of Head Index 2
name_head = getHeadName(2)
type = getHeadType name_head
-- Test if Head Exists in Database
test = headTypeValid type
if (test == true) then
(
  utility Head_Generator "Head Generator"   -- Generate Head Generator Utility
     (

          label params "Head Generation Script v1.0"
          button create "Create Head"
          button quit "Quit 3DS Max"

          on create pressed do
          (

               resetMaxFile() #noPrompt
               progressStart "Generating Head"
               progressUpdate (10)
               -- Assign parametric head to variable ph
               ph = param_heads head_type: type
               -- Move parameteric head [x,y,z]
               move ph [0.350917,-104.469,0]
               move ph [0,0,35.4826]
               scale ph [5.55,5.55,5.55]
               progressUpdate (15)

               -- Load Material and Textures Library
               mat_name_load = loadMaterialLibrary
               "C:\3dsmax3_1\Matlibs\Head_Textures.mat"
               -- If Materials Library existis in Database then
```

```
if mat_name_load == true then                (
        mat_name = getMatLibFileName()
        skin_type = "Material #1"
        meditMaterials[1]= currentMaterialLibrary[skin_type]
        -- Assign material skin type to slot 1 of the editor
        ph.material = meditMaterials[1]
        -- Assign material from editor to head object ph

        progressUpdate (20)
        ph.mapCoords = on
        mergeMAXFile
        "C:\3dsmax3_1\Scenes\HeadDesign\Eyes2.max"
        -- Open and merge eyes 3d model
        select $Eye01
        -- Tranformation to position Eyes appropriately
        move $Eye01 [0,0,-57.8311]
        move $Eye01 [4.4934,0,0]
        select $Eye02
        move $Eye02 [0,0,-59.0721]
        move $Eye02 [-3.25661,0,0]
        move $Eye02 [0,0,0.400439]
        move $Eye02 [0,-0.0673475,0]
        move $Eye02 [-0.410314,0,0]

        progressUpdate (25)

        select $Eye01
        move $Eye01 [0,-1.86045,0]
        move $Eye01 [0.429525,0,0]
        move $Eye01 [0,0,5.764]
        move $Eye01 [-4.23381,0,0]
        move $Eye01 [0,-21.9471,0]
        move $Eye01 [0,0,3.46355]
        move $Eye01 [0.905571,0,0]
        move $Eye01 [0,-84.0637,0]
        move $Eye01 [0,0,15.2293]
        move $Eye01 [5.74716,0,0]
        move $Eye01 [0,0,1.02152]
        move $Eye01 [0.0252424,0,0]
        move $Eye01 [-0.064502,16.2655,0]
        move $Eye01 [0,0,-3.28828]
        move $Eye01 [-1.13887,0,0]
        move $Eye02 [0,0,5.81182]
        move $Eye02 [4.29086,0,0]
        move $Eye02 [0,-92.2886,0]
        move $Eye02 [0,0,16.1499]
        move $Eye02 [-5.57312,0,0]
        move $Eye02 [0,0,1.14661]
        move $Eye02 [-0.890665,0,0]
move $Eye02 [0,-0.214613,0]
meditMaterials[2]= currentMaterialLibrary["Material
#19"]

$Eye01.material = meditMaterials[2]

progressUpdate (35)


select $Eye02
move $Eye02 [0.280777,0,0]
move $Eye02 [0,0,-0.0385544]
rotate $Eye02 (angleaxis 4 [1,0,0])
meditMaterials[3]= currentMaterialLibrary["Material
#19"]

$Eye02.material = meditMaterials[3]

select $Eye01
$Eye01.scale = [1.01942,0.970874,0.795798]
$Eye01.scale = [0.886452,0.844238,0.691998]

select $Eye02
$Eye02.scale = [1.01942,0.970874,0.795798]
$Eye02.scale = [0.886452,0.844238,0.691998]
$Eye02.scale = [0.881644,0.839659,0.688245]

progressUpdate (45)

f = openFile "C:/Head Designer/headsparam11.txt"
-- Open Parameters File
if (f != undefined) then        -- check if file exists
(
        instring = readLine f  -- Read parameters file
        modstring = replace instring 1 15 "Material #"
        -- extract value for head material code
        hdparam_array1[1] = modstring
        count = 2

        do
        (
                instring = readLine f  -- read file data
                if (count == 41) then
                (
                        modstring = replace instring 1
                string_size[count] "Material #"
                -- extract value for eye material code
                        hdparam_array1[2] = modstring
                )

        else
        (
                modstring = replace instring 1
                string_size[count] ""
                -- extract values for head parameters
                hdparam_array1[count+1] = modstring
        )
        count = count+1
) while not eof f

array_size = hdparam_array1.count

j = 42
do
(
        hdparam_array1[j] = hdparam_array1[j+1]
        j = j+1
        )while (j != array_size+1)

        print array_size
        print hdparam_array1

        count2 = 1
        for i = 3 to (array_size - 1) do
        (
                hdparam_array2[count2] =
                hdparam_array1[i] as float
                -- format parmaeters to float type
                count2 = count2 + 1
        )
        array_size2 = hdparam_array2.count
        print array_size2
        print hdparam_array2
        progressUpdate (60)
        -- Select head object ph
        select ph
        -- Edit Parametric Head Attributes assigning
           values from array
        ph.head_type = hdparam_array2[1]
        ph.Master_Strength = hdparam_array2[2]
        ph.X_Pull = hdparam_array2[3]
        ph.Y_Pull = hdparam_array2[4]
        ph.Z_Pull = hdparam_array2[5]
        ph.Y_Offset = hdparam_array2[6]
        ph.Z_Offset = hdparam_array2[7]
        ph.Head_Width = hdparam_array2[8]
        ph.Head_Width_Skew_1 = hdparam_array2[9]
        ph.Head_Width_Skew_2 = hdparam_array2[10]
        ph.Head_Depth = hdparam_array2[11]
        ph.Head_Depth_Skew = hdparam_array2[12]
ph.Head_Height = hdparam_array2[13]
ph.Head_Height_Skew = hdparam_array2[14]
ph.Face_Squash = hdparam_array2[15]
ph.Head_Flatten = hdparam_array2[16]
ph.Forehead_Slope = hdparam_array2[17]
-- doit_prog.value = 65
progressUpdate (65)
ph.Nose_Width = hdparam_array2[18]
ph.Nose_Width_Z_Weight =
hdparam_array2[19]
ph.Nose_Length = hdparam_array2[20]
ph.Nose_Length__Z_Weight =
hdparam_array2[21]
ph.Nose_Pullup = hdparam_array2[22]
ph.Nose_Bridge = hdparam_array2[23]
ph.Nose_Hook = hdparam_array2[24]
ph.Nose_Hook_Influence = hdparam_array2[25]
ph.Chin_Extent = hdparam_array2[26]
ph.Chin_Tilt_Amount = hdparam_array2[27]
ph.Chin_Tilt_Influence = hdparam_array2[28]
ph.Chin_Accent = hdparam_array2[29]

progressUpdate (70)

ph.Jaw_Width = hdparam_array2[30]
ph.Jaw_Influence = hdparam_array2[31]
ph.Jaw_Width_Uniformity =
hdparam_array2[32]
ph.Cheekbones_Extrude = hdparam_array2[33]
ph.Cheekbones_Z_Pos = hdparam_array2[34]
ph.Cheek_Curvature = hdparam_array2[35]
ph.Cheek_Curvature_Z_Pos =
hdparam_array2[36]
ph.Cheek_Cuvature_Y_Pos =
hdparam_array2[37]
ph.Cheek_Curvature_Z_Falloff =
hdparam_array2[38]
ph.Cheek_Curvature_Y_Falloff =
hdparam_array2[39]

progressUpdate (75)

 ph.Eye_Separation = hdparam_array2[40]
ph.Eye_Inset = hdparam_array2[41]
ph.Eye_Top_Roundness = hdparam_array2[42]
ph.Eye_Bottom_Roundness =
hdparam_array2[43]
ph.Eye_Rotation = hdparam_array2[44]
ph.Eye_Brow_Bulge = hdparam_array2[45]
```

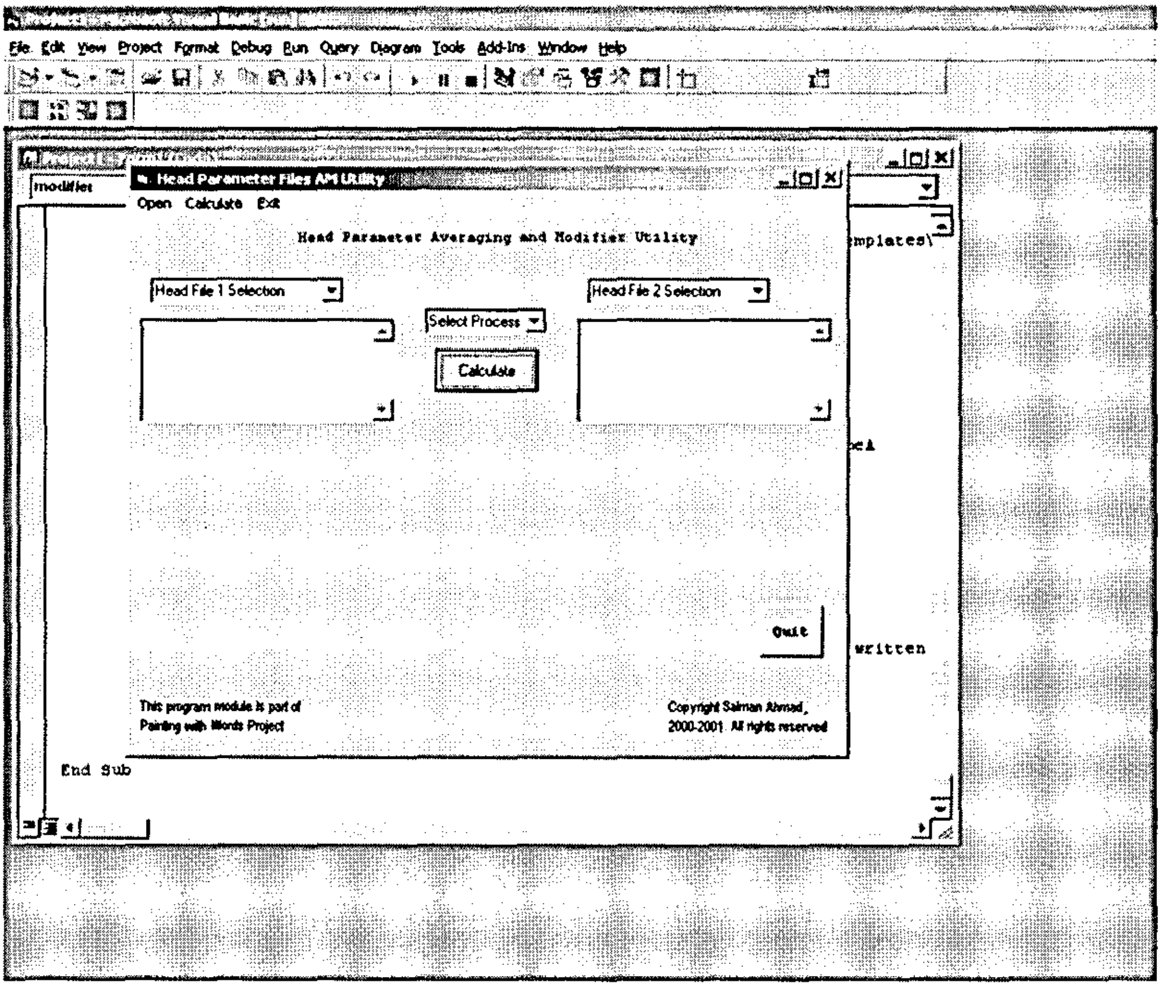
```
                    ph.Ear_Height = hdparam_array2[46]
                    ph.Ear_Lobe_Length = hdparam_array2[47]
                    ph.Ear_Depth = hdparam_array2[48]
                    ph.Ear_Rotation = hdparam_array2[49]
                    ph.Mouth_Protrude = hdparam_array2[50]
                    ph.Mouth_Width = hdparam_array2[51]

                    progressUpdate (85)

                    meditMaterials[1]=
                currentMaterialLibrary[hdparam_array1[1]]
                    ph.material = meditMaterials[1]
                -- Assign material/texture to head object

                    select $Eye01
                    -- max move
                    move $Eye01
[hdparam_array2[52],hdparam_array2[53],hdparam_array2[54]]
                    -- move $Eye01 [0,0,1.14486]
                    rotate $Eye01 (angleaxis 4
[hdparam_array2[58],hdparam_array2[59],hdparam_array2[60]])
                    meditMaterials[2]=
currentMaterialLibrary[hdparam_array1[2]]
                    $Eye01.material = meditMaterials[2]

                    progressUpdate (95)

                    select $Eye02
                    move $Eye02
[hdparam_array2[55],hdparam_array2[56],hdparam_array2[57]]
                    -- move $Eye02 [0,0,1.13579]
                    rotate $Eye02 (angleaxis 4
[hdparam_array2[61],hdparam_array2[62],hdparam_array2[63]])
                    meditMaterials[3]=
currentMaterialLibrary[hdparam_array1[2]]
                    $Eye02.material = meditMaterials[3]


                render camera outputwidth:640 outputheight:480
                -- Render and display image of 3D head model
                    progressUpdate (100)
                    progressEnd()
                    messageBox "Facial Image Generation
                    Complete"

                )
                else
                messageBox "Head Parameters File Not Found!!"
```

```
        )
        else
        print "No materials found!!"

)
on edit pressed do
(
rollout edithead "Edit Head"    -- Create rollout called Edit Head
        (
                        -- Create slider object on rollout to control
                        parameter

                        slider HeadWidth "Head Width"
        orient:#horizontal ticks:0 range:[0,10,hdparam_array2[8]]
                        slider HeadDepth "Head Depth"
        orient:#horizontal ticks:0 range:[0,10,hdparam_array2[11]]
                        slider HeadHeight "Head Height"
        orient:#horizontal ticks:0 range:[0,10,hdparam_array2[13]]
                        slider HeadFlat "Head Flatten"
        orient:#horizontal ticks:0 range:[0,2,hdparam_array2[16]]
                        slider HeadSlope "Forehead Slope"
        orient:#horizontal ticks:0 range:[-1,1,hdparam_array2[17]]
                        slider NoseWidth "Nose Width"
        orient:#horizontal ticks:0 range:[0,2,hdparam_array2[18]]
                        slider NoseLength "Nose Length"
        orient:#horizontal ticks:0 range:[0,3,hdparam_array2[20]]
                        slider NosePullup "Nose Pullup"
        orient:#horizontal ticks:0 range:[0,2,hdparam_array2[22]]
                        slider NoseBridge "Nose Bridge"
        orient:#horizontal ticks:0 range:[-1,1,hdparam_array2[23]]
                        slider NoseHook "Nose Hook"
        orient:#horizontal ticks:0 range:[-1,1,hdparam_array2[24]]
                        slider ChinExtent "Chin Extent"
        orient:#horizontal ticks:0 range:[0,2,hdparam_array2[26]]
                        slider ChinTilt "Chin Tilt" orient:#horizontal
        ticks:0 range:[0,2,hdparam_array2[27]]
                        slider JawWidth "Jaw Width" orient:#horizontal
        ticks:0 range:[0,1,hdparam_array2[30]]
                        slider CheekBones "Cheekbone Extrude"
        orient:#horizontal ticks:0 range:[-1,1,hdparam_array2[33]]
                        slider CheekCurv "Cheek Curvature"
        orient:#horizontal ticks:0 range:[-1,1,hdparam_array2[35]]
                        slider EyeSep "Eye Seperation"
        orient:#horizontal ticks:0 range:[0,2,hdparam_array2[40]]
                        slider EyeRotate "Eye Rotate" orient:#horizontal
        ticks:0 range:[-1,1,hdparam_array2[44]]
                        slider EarHeight "Ear Height" orient:#horizontal
        ticks:0 range:[0,2,hdparam_array2[46]]
                        slider LobeLength "Lobe Length"
```

```
orient:#horizontal ticks:0 range:[0,1,hdparam_array2[47]]
                slider MouthWidth "Mouth Width"
orient:#horizontal ticks:0 range:[-1,1,hdparam_array2[51]]

        -- if slider value changes then assign value to head
        parameter
        on HeadWidth changed val do
                ph.Head_Width = val
        on HeadDepth changed val do
                ph.Head_Depth = val
        on HeadHeight changed val do
                ph.Head_Height = val
        on HeadFlat changed val do
                ph.Head_Flatten = val
        on HeadSlope changed val do
                ph.Forehead_Slope = val
        on NoseWidth changed val do
                ph.Nose_Width = val
        on NoseLength changed val do
                ph.Nose_Length = val
        on NosePullup changed val do
                ph.Nose_Pullup = val
        on NoseBridge changed val do
                ph.Nose_Bridge = val
        on NoseHook changed val do
                ph.Nose_Hook = val
        on ChinExtent changed val do
                ph.Chin_Extent = val
        on ChinTilt changed val do
                ph.Chin_Tilt_Amount = val
        on JawWidth changed val do
                ph.Jaw_Width = val
        on CheekBones changed val do
                ph.Cheekbones_Extrude = val
        on CheekCurv changed val do
                ph.Cheek_Curvature = val
        on EyeSep changed val do
                ph.Eye_Seperation = val
        on EyeRotate changed val do
                ph.Eye_Rotation = val
        on EarHeight changed val do
                ph.Ear_Height = val
        on LobeLength changed val do
                ph.Ear_Lobe_Length = val
        on MouthWidth changed val do
                ph.Mouth_Width = val


)
```

```
        eh=newRolloutFloater "Modifiers" 300 220  -- Position rollout floater
        addRollout edithead eh                    -- Add rollout to interface
            )
            on quit pressed do
            (
            quitMAX() #noPrompt
            )
    )
)

/*********************************************************************/
```

Head Comparator


File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
modifier
Head Parameter Files API Utility
Open Calculate Exit
Head Parameter Averaging and Modifier Utility
Head File 1 Selection
Head File 2 Selection
Select Process
Calculate
Quit
mplates\
written
End Sub

This program module is part of Painting with Words Project

Copyright Salman Ahmad, 2000-2001 All rights reserved

Head Comparator – Averaging and Modifier Utility Interface

Code Listing

```
/*********************************************************************/
```

```
Dim head_param1(65) As String
Dim head_param2(65) As String
Dim averaged_param(65) As String
Dim file_nameA, file_nameB As String

Dim string_size(65) As String

Private Sub average_Click()
Dim fso, txtfile, StrLine$
Dim param_stringA, param_stringB, param_descript As Variant
Dim strlength, length As Integer
Dim new_headparam(65) As String
Dim paramA, paramB, average As Double


ProgressBar1.Min = 0
ProgressBar1.Max = 100
ProgressBar1.Visible = True

    If Text1.Text = "" Or Text2.Text = "" Or file_nameA = file_nameB Then
      ProgressBar1.Visible = False
      response = MsgBox("Sorry can not calculate average for null or similar files,
please select different head files for comparative analysis.", vbExclamation, "Head
File Selection Error")
    Else

      For j = 1 To 65
        strlength = Len(head_param1(j)) 'calculates length of string
        length = strlength - string_size(j)
        param_stringA = Right(head_param1(j), length) 'returns characters of
amount length from right
        param_descript = Left(head_param1(j), string_size(j)) 'returns char from
left
        paramA = CDbl(param_stringA) 'converts string to double

        strlength = Len(head_param2(j))
        length = strlength - string_size(j)
        param_stringB = Right(head_param2(j), length)
        paramB = CDbl(param_stringB)
        average = FormatNumber(((paramA + paramB) / 2), 3)
        If (j <= 3) Then
          average = Int(average)
        End If
        new_headparam(j) = param_descript & CStr(average)
        ProgressBar1.Value = Int(j / 2)
      Next j

      If (Combo1.Text = "Head File 1 Selection") Then
       file_nameA = file_nameA
```

```
        Else
         file_nameA = Combo1.Text
        End If

        If (Combo2.Text = "Head File 2 Selection") Then
         file_nameB = file_nameB
        Else
         file_nameB = Combo2.Text
        End If

        file_name = file_nameA & "-" & file_nameB & " averaged.txt"
        Set fso = CreateObject("Scripting.FileSystemObject")
        Set txtfile = fso.CreateTextFile("C:\My Documents\Head
Designer\Templates\Averaged\" & file_name, True)

        For k = 1 To 65
           txtfile.WriteLine new_headparam(k)
           StrLine = StrLine & vbCrLf
           StrLine = StrLine & new_headparam(k)
           ProgressBar1.Value = Int((65 / 2) + k)
        Next k
        txtfile.Close

        Label1.Caption = "Average of " & file_nameA & " and " & file_nameB
        Label1.Visible = True
        Text3.Text = ""
        Text3.Visible = True

        Text3.SelStart = Len(Text3)
        Text3.SelLength = 0
        Text3.SelText = StrLine

        ProgressBar1.Value = 100
        message = "Average of " & file_nameA & " and " & file_nameB & " written
to file:C:\My Documents\Head Designer\Templates\Averaged\" & file_name
        response = MsgBox(message, vbInformation, "Output to File")
        ProgressBar1.Visible = False
        ProgressBar1.Value = ProgressBar1.Min

     End If
End Sub

Private Sub Combo1_Click()
Dim filename As String
Dim Str$, StrLine$


   listvalue = Combo1.ListIndex
```

```
    Select Case listvalue
    Case 0: filename = "generic man.txt"
    Case 1: filename = "generic woman.txt"
    Case 2: filename = "african man.txt"
    Case 3: filename = "african woman.txt"
    Case 4: filename = "european man.txt"
    Case 5: filename = "european woman.txt"
    Case 6: filename = "oriental man.txt"
    End Select

    Label5.Caption = ""
    Text1.Text = ""
    Open "C:\My Documents\Head Designer\Templates\" & filename For Input As #1
    ' Read the contents of the file.
    step = 1
    While Not EOF(1)
       Line Input #1, StrLine$
       head_param1(step) = StrLine$
       If Str <> "" Then Str = Str & vbCrLf
       Str = Str & StrLine
       step = step + 1
    Wend
    Close #1

    Text1.SelStart = Len(Text1)
    Text1.SelLength = 0
    Text1.SelText = Str

End Sub

Private Sub Combo2_Click()
Dim filename As String
Dim Str$, StrLine$

    listvalue = Combo2.ListIndex

    Select Case listvalue
    Case 0: filename = "generic man.txt"
    Case 1: filename = "generic woman.txt"
    Case 2: filename = "african man.txt"
    Case 3: filename = "african woman.txt"
    Case 4: filename = "european man.txt"
    Case 5: filename = "european woman.txt"
    Case 6: filename = "oriental man.txt"
    End Select

    Label6.Caption = ""
    Text2.Text = ""
    Open "C:\My Documents\Head Designer\Templates\" & filename For Input As #2
    ' Read the contents of the file.
    step = 1
    While Not EOF(2)
        Line Input #2, StrLine$
        head_param2(step) = StrLine$
        If Str <> "" Then Str = Str & vbCrLf
        Str = Str & StrLine
        step = step + 1
    Wend
    Close #2

    Text2.SelStart = Len(Text2)
    Text2.SelLength = 0
    Text2.SelText = Str


End Sub


Private Sub Command1_Click()

Dim fso, txtfile, StrLine$
Dim param_stringA, param_stringB, param_descript As Variant
Dim strlength, length As Integer
Dim new_headparam(65) As String
Dim paramA, paramB, average, modifier As Double

ProgressBar1.Min = 0
ProgressBar1.Max = 100
ProgressBar1.Visible = True

listvalue = Combo3.Text

    Select Case listvalue
    Case "Select Process":
        ProgressBar1.Visible = False
        response = MsgBox("Please select process.", vbExclamation, "Process selection
Error")
    Case "Average":
        If Text1.Text = "" Or Text2.Text = "" Or Combo1.Text = Combo2.Text Or
file_nameA = file_nameB Then
            ProgressBar1.Visible = False
            response = MsgBox("Sorry can not calculate average for null or similar files,
please select different head files for comparative analysis.", vbExclamation, "Head
File Selection Error")
        Else

            For j = 1 To 65
                strlength = Len(head_param1(j)) 'calculates length of string
```

```
      length = strlength - string_size(j)
      param_stringA = Right(head_param1(j), length) 'returns characters of
amount length from right
      param_descript = Left(head_param1(j), string_size(j)) 'returns char from
left
      paramA = CDbl(param_stringA) 'converts string to double

      strlength = Len(head_param2(j))
      length = strlength - string_size(j)
      param_stringB = Right(head_param2(j), length)
      paramB = CDbl(param_stringB)
      average = FormatNumber(((paramA + paramB) / 2), 3)
      If (j <= 3) Then
        average = Int(average)
      End If
      new_headparam(j) = param_descript & CStr(average)
      ProgressBar1.Value = Int(j / 2)
    Next j

    If (Combo1.Text = "Head File 1 Selection") Then
     file_nameA = file_nameA
    Else
     file_nameA = Combo1.Text
    End If

    If (Combo2.Text = "Head File 2 Selection") Then
     file_nameB = file_nameB
    Else
     file_nameB = Combo2.Text
    End If

    file_name = file_nameA & "-" & file_nameB & " averaged.txt"
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set txtfile = fso.CreateTextFile("C:\My Documents\Head
Designer\Templates\Averaged\" & file_name, True)

    For k = 1 To 65
      txtfile.WriteLine new_headparam(k)
      StrLine = StrLine & vbCrLf
      StrLine = StrLine & new_headparam(k)
      ProgressBar1.Value = Int((65 / 2) + k)
    Next k
    txtfile.Close

    Label1.Caption = "Average of " & file_nameA & " and " & file_nameB
    Label1.Visible = True
    Text3.Text = ""
    Text3.Visible = True
```

```
      Text3.SelStart = Len(Text3)
      Text3.SelLength = 0
      Text3.SelText = StrLine

      ProgressBar1.Value = 100
      message = "Average of " & file_nameA & " and " & file_nameB & " written
to file:C:\My Documents\Head Designer\Templates\Averaged\" & file_name
      response = MsgBox(message, vbInformation, "Output to File")
      ProgressBar1.Visible = False
      ProgressBar1.Value = ProgressBar1.Min

    End If
  Case "Modifier":
    If Combo1.Text = "" Or Combo2.Text = "" Or Combo1.Text = Combo2.Text
Then
      ProgressBar1.Visible = False
      response = MsgBox("Sorry can not calculate modifier for null or similar files,
please select different head files for comparative analysis.", vbExclamation, "Head
File Selection Error")
    Else

    For j = 1 To 65
      strlength = Len(head_param1(j)) 'calculates length of string
      length = strlength - string_size(j)
      param_stringA = Right(head_param1(j), length) 'returns characters of amount
length from right
      param_descript = Left(head_param1(j), string_size(j)) 'returns char from left
      paramA = CDbl(param_stringA) 'converts string to double

      strlength = Len(head_param2(j))
      length = strlength - string_size(j)
      param_stringB = Right(head_param2(j), length)
      paramB = CDbl(param_stringB)
      modifier = FormatNumber((paramB - paramA), 3)
      If (j <= 3) Then
          modifier = Int(modifier)
      End If
      new_headparam(j) = param_descript & CStr(modifier)
      ProgressBar1.Value = Int(j / 2)
    Next j

    If (Combo1.Text = "Head File 1 Selection") Then
      file_nameA = file_nameA
    Else
      file_nameA = Combo1.Text
    End If

    If (Combo2.Text = "Head File 2 Selection") Then
      file_nameB = file_nameB
    Else
      file_nameB = Combo2.Text
    End If

    file_name = file_nameA & "-" & file_nameB & " modifier.txt"
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set txtfile = fso.CreateTextFile("C:\My Documents\Head
Designer\Templates\Modifiers\" & file_name, True)

    For k = 1 To 65
      txtfile.WriteLine new_headparam(k)
      StrLine = StrLine & vbCrLf
      StrLine = StrLine & new_headparam(k)
      ProgressBar1.Value = Int((65 / 2) + k)
    Next k
    txtfile.Close

    Label1.Caption = "Modifier of " & file_nameB & " from " & file_nameA
    Label1.Visible = True
    Text3.Text = ""
    Text3.Visible = True

    Text3.SelStart = Len(Text3)
    Text3.SelLength = 0
    Text3.SelText = StrLine

    ProgressBar1.Value = 100
    message = "Modifier of " & file_nameB & " from " & file_nameA & " written to
file:C:\My Documents\Head Designer\Templates\Modifiers\" & file_name
    response = MsgBox(message, vbInformation, "Output to File")
    ProgressBar1.Visible = False
    ProgressBar1.Value = ProgressBar1.Min

    End If

  End Select

End Sub

Private Sub Command2_Click()
Form1.Hide

End Sub

Private Sub exit_Click()
Form1.Hide

End Sub
```

```
Private Sub Form_Load()
  Combo1.AddItem "Generic Male"
  Combo1.AddItem "Generic Female"
  Combo1.AddItem "African Male"
  Combo1.AddItem "African Female"
  Combo1.AddItem "European Male"
  Combo1.AddItem "European Female"
  Combo1.AddItem "Oriental Male"

  Combo2.AddItem "Generic Male"
  Combo2.AddItem "Generic Female"
  Combo2.AddItem "African Male"
  Combo2.AddItem "African Female"
  Combo2.AddItem "European Male"
  Combo2.AddItem "European Female"
  Combo2.AddItem "Oriental Male"

  Combo3.AddItem "Average"
  Combo3.AddItem "Modifier"

string_size(1) = 16
string_size(2) = 16
string_size(3) = 12
string_size(4) = 16
string_size(5) = 14
string_size(6) = 14
string_size(7) = 14
string_size(8) = 16
string_size(9) = 16
string_size(10) = 13
string_size(11) = 18
string_size(12) = 18
string_size(13) = 13
string_size(14) = 17
string_size(15) = 14
string_size(16) = 18
string_size(17) = 19
string_size(18) = 15
string_size(19) = 13
string_size(20) = 13
string_size(21) = 21
string_size(22) = 14
string_size(23) = 22
string_size(24) = 14
string_size(25) = 14
string_size(26) = 12
string_size(27) = 22
string_size(28) = 14
string_size(29) = 12
string_size(30) = 22
string_size(31) = 14
string_size(32) = 12
string_size(33) = 16
string_size(34) = 17
string_size(35) = 16
string_size(36) = 13
string_size(37) = 19
string_size(38) = 23
string_size(39) = 23
string_size(40) = 27
string_size(41) = 27
string_size(42) = 18
string_size(43) = 13
string_size(44) = 19
string_size(45) = 23
string_size(46) = 16
string_size(47) = 18
string_size(48) = 14
string_size(49) = 12
string_size(50) = 13
string_size(51) = 16
string_size(52) = 17
string_size(53) = 14
string_size(54) = 18
string_size(55) = 18
string_size(56) = 18
string_size(57) = 18
string_size(58) = 18
string_size(59) = 18
string_size(60) = 14
string_size(61) = 14
string_size(62) = 14
string_size(63) = 14
string_size(64) = 14
string_size(65) = 14

ProgressBar1.Visible = False
Label1.Visible = False
file_nameA = "a"
file_nameB = "b"
Label5.Caption = ""
Label6.Caption = ""

End Sub

Private Sub HeadFile1_Click()

Dim Str$, StrLine$
```

```
  CommonDialog1.Flags = cdlOFNHideReadOnly
  ' Set filters
  CommonDialog1.Filter = "Text Files(*.txt)|*.txt"
  ' Specify default filter
  CommonDialog1.FilterIndex = 2
  ' Display the Open dialog box
  CommonDialog1.ShowOpen
  ' Display name of selected file
'  MsgBox CommonDialog1.filename
 'Exit Sub
   file_nameA = CommonDialog1.filename
   strlength = Len(file_nameA) 'calculates length of string
   length = strlength - 40
   file_nameA = Right(file_nameA, length)
   strlength = Len(file_nameA)
   length = strlength - 4
   file_nameA = Left(file_nameA, length)

   Open (CommonDialog1.filename) For Input As #1
    ' Read the contents of the file.
    step = 1
    Text1.Text = ""
    Combo1.Text = "Head File 1 Selection"
    Label5.Caption = file_nameA & ".txt Loaded"
    While Not EOF(1)
       Line Input #1, StrLine$
       head_param1(step) = StrLine$
       If Str <> "" Then Str = Str & vbCrLf
       Str = Str & StrLine
       step = step + 1
    Wend
    Close #1

    Text1.SelStart = Len(Text2)
    Text1.SelLength = 0
    Text1.SelText = Str

End Sub

Private Sub headfile2_Click()
Dim Str$, StrLine$


  CommonDialog1.Flags = cdlOFNHideReadOnly
  ' Set filters
  CommonDialog1.Filter = "Text Files(*.txt)|*.txt"
  ' Specify default filter
  CommonDialog1.FilterIndex = 2
  ' Display the Open dialog box
  CommonDialog1.ShowOpen
  ' Display name of selected file
  'MsgBox CommonDialog1.filename
 ' Exit Sub
  file_nameB = CommonDialog1.filename
  strlength = Len(file_nameB) 'calculates length of string
  length = strlength - 40
  file_nameB = Right(file_nameB, length)
  strlength = Len(file_nameB)
  length = strlength - 4
  file_nameB = Left(file_nameB, length)
  Print file_nameB
  Open (CommonDialog1.filename) For Input As #2
   ' Read the contents of the file.
   step = 1
   Text2.Text = ""
   Combo2.Text = "Head File 2 Selection"
   Label6.Caption = file_nameB & ".txt Loaded"
   While Not EOF(2)
      Line Input #2, StrLine$
      head_param2(step) = StrLine$
      If Str <> "" Then Str = Str & vbCrLf
      Str = Str & StrLine
      step = step + 1
   Wend
   Close #2

   Text2.SelStart = Len(Text2)
   Text2.SelLength = 0
   Text2.SelText = Str

End Sub


Private Sub modifier_Click()
Dim fso, txtfile, StrLine$
Dim param_stringA, param_stringB, param_descript As Variant
Dim strlength, length As Integer
Dim new_headparam(65) As String
Dim paramA, paramB, modifier As Double


ProgressBar1.Min = 0
ProgressBar1.Max = 100
ProgressBar1.Visible = True

      If Text1.Text = "" Or Text2.Text = "" Or Combo1.Text = Combo2.Text Or
file_nameA = file_nameB Then
         ProgressBar1.Visible = False
```

```
        response = MsgBox("Sorry can not calculate average for null or similar files,
please select different head files for comparative analysis.", vbExclamation, "Head
File Selection Error")
      Else

        For j = 1 To 65
           strlength = Len(head_param1(j)) 'calculates length of string
           length = strlength - string_size(j)
           param_stringA = Right(head_param1(j), length) 'returns characters of
amount length from right
           param_descript = Left(head_param1(j), string_size(j)) 'returns char from
left
           paramA = CDbl(param_stringA) 'converts string to double

           strlength = Len(head_param2(j))
           length = strlength - string_size(j)
           param_stringB = Right(head_param2(j), length)
           paramB = CDbl(param_stringB)
           modifier = FormatNumber((paramB - paramA), 3)
           If (j <= 3) Then
              modifier = Int(modifier)
           End If
           new_headparam(j) = param_descript & CStr(modifier)
           ProgressBar1.Value = Int(j / 2)
        Next j

        If (Combo1.Text = "Head File 1 Selection") Then
         file_nameA = file_nameA
        Else
         file_nameA = Combo1.Text
        End If

        If (Combo2.Text = "Head File 2 Selection") Then
         file_nameB = file_nameB
        Else
         file_nameB = Combo2.Text
        End If

        file_name = file_nameA & "-" & file_nameB & " modifier.txt"
        Set fso = CreateObject("Scripting.FileSystemObject")
        Set txtfile = fso.CreateTextFile("C:\My Documents\Head
Designer\Templates\Modifiers\" & file_name, True)

        For k = 1 To 65
           txtfile.WriteLine new_headparam(k)
           StrLine = StrLine & vbCrLf
           StrLine = StrLine & new_headparam(k)
           ProgressBar1.Value = Int((65 / 2) + k)
        Next k
        txtfile.Close

        Label1.Caption = "Modifier of " & file_nameB & " from " & file_nameA
        Label1.Visible = True
        Text3.Text = ""
        Text3.Visible = True

        Text3.SelStart = Len(Text3)
        Text3.SelLength = 0
        Text3.SelText = StrLine

        ProgressBar1.Value = 100
        message = "Modifier of " & file_nameB & " from " & file_nameA & " written
to file:C:\My Documents\Head Designer\Templates\Modifiers\" & file_name
        response = MsgBox(message, vbInformation, "Output to File")
        ProgressBar1.Visible = False
        ProgressBar1.Value = ProgressBar1.Min

    End If
End Sub

/*****************************************************************/
```

Appendix C

Head Engine and TMS + NLI Code Listing

```
/********************** HEADS ENGINE *********************/

/* false detection */

:-multifile expand/7,set_bid/1,inconsistent/3,subsumes/3.

subsumes(heads,descriptor(P1,_),descriptor(P2,_)):-
     append(LP1,P2RP1,P1),
     append(P2,RP1,P2RP1),
     (LP1 = [_|_];
     RP1 = [_|_]).
subsumes(heads,descriptor(P,D1s),descriptor(P,D2s)):-
     forall(
          member(D2,D2s),
          (forall(
                    member([qualifiers(A2s,H2s)],D2),
                    (member(D1,D1s),
                         member([qualifiers(A1s,H1s)],D1),
                         (append(LA1,A2RA1,A1s),
                              append(A2s,RA1,A2RA1);
                              append(LH1,H2RH1,H1s),
                              append(H2s,RH1,H2RH1)
                         )
                    )
               )
          )
     ).


/*  find a heads phrase */

set_bid(heads):-
     make_bid(heads,0,
          [[phrase,echo(Language,Phrase,heads),true]],
          100,
          [[heads,nil]]).
```

```
set_bid(heads):-
     make_bid(heads,100,
          [[heads,descriptor(P,D),true]],
          [[heads,description(_),true]],
          100,
          [[heads,description(D)]]).

set_bid(heads):-
     make_bid(heads,101,
          [[heads,description(D1),true],
               [heads,descriptor(P2,D2),true]],
          100,
          [[heads,description(D)]]).

set_bid(heads):-
     make_bid(heads,10,
          [[heads,description(D),true]],
          2,
          [[heads,nil]]).

expand(heads,100,
          [[heads,descriptor(P,D)]],
          necessary,
          [[heads,description(D)]],
          _,n).

expand(heads,101,
          [[heads,description(D1)],
               [heads,descriptor(P1,D2)]],
          necessary,
          [[heads,description(D)]],
          _,n):-
     append(D1,D2,DT),
     refine_qualifier_hedges(DT,D).


expand(heads,10,
          [[heads,description(D)]],
          necessary,
          [[heads,
          nil]],
          _,n):-
     dump_head(D).
dump_head(D):-
     delete_element(Obj,D,OD),
     [object([X]),Q] = Obj,
     load_head_object(X),
     apply_template_to_head,
     /* check if qualifier exists for object X and apply
qualifier by loading appropriate modifier parameters */
 modify_head_object(Q),
     do_head_object(OD),
     tell(user),
     write(X),
     nl,
     print_head_object,
     new(File,'Heads Parameter File','headsparam.txt'),
      open(File,write),
      stype(File,'TEXT',ttxt),
      telling(Current),
      tell(File),
     print_head_object,
     told,
     tell(Current),
     !.

expand(heads,0,
          [[phrase,echo(Language,Phrase,heads)]],
          necessary,
          [[Language,
          nil]],
          _,n):-
          (
          LPhrase =..
[Language,sentence(_),Phrase,Phrase,Parsed,heads],
          make_bid(heads,11,
               [[phrase,echo(Language,Phrase,heads),true],
               [Language,LPhrase,true]],
               100,
               [[heads,nil]]);

          LPhrase =..
[Language,comparison_phrase(_),Phrase,Phrase,Parsed,heads],
          make_bid(heads,12,
               [[phrase,echo(Language,Phrase,heads),true],
               [Language,LPhrase,true]],
               100,
               [[heads,nil]])
          ).

expand(heads,11,
          [[phrase,echo(Language,Phrase,heads)],
[Language,LPhrase]],
          necessary,
          [[heads,
          nil]],
          _,n):-

          LPhrase =..
[Language,sentence(P),Phrase,Phrase,_,heads],
          NPhrase =..
[Language,noun_phrase(_),_,Phrase,_,heads],
          VPhrase =..
[Language,verb_phrase(_),_,Phrase,_,heads],

     ancestor([Language,LPhrase],[Language,NPhrase,true],_)
,

     ancestor([Language,LPhrase],[Language,VPhrase,true],_)
,

     \+ancestor([Language,VPhrase],[Language,NPhrase,true],
_),

          make_bid(heads,21,
               [[Language,LPhrase,true],
               [Language,NPhrase,true],
               [Language,VPhrase,true]],
               100,
               [[heads,F]]).

expand(heads,12,
          [[phrase,echo(Language,Phrase,heads)],
[Language,LPhrase]],
          necessary,
          [[heads,
          nil]],
          _,n):-

          LPhrase =..
[Language,comparison_phrase(P),Phrase,Phrase,_,heads],
          NPhrase =..
[Language,noun_phrase(_),_,Phrase,_,heads],

     ancestor([Language,LPhrase],[Language,NPhrase,true],_)
,
          APhrase =..
[Language,adjective_phrase(_),_,Phrase,_,heads],
     ancestor([Language,LPhrase],[Language,APhrase,true],_)
,

     \+ancestor([Language,NPhrase],[Language,APhrase,true],
_),

     \+ancestor([Language,APhrase],[Language,NPhrase,true],
_),
          make_bid(heads,22,
               [[Language,LPhrase,true],
               [Language,NPhrase,true],
               [Language,APhrase,true]],
               100,
               [[heads,F]]).

expand(heads,21,
[[Language,SPhrase],
[Language,NPhrase],
[Language,VPhrase]],
          possible((0.0,0.1,0.9)),
          [[heads,
          descriptor(P,[QualifiersM|QualifiersSSR])]],
          _,n):-

     SPhrase =.. [Language,sentence(P),_,Phrase,_,heads],
/* extract the elements of the subject noun phrase */

     OPhrase1 =..
[Language,noun(MainObject),_,Phrase,_,heads],
     ancestor([Language,NPhrase],[Language,OPhrase1,true],_
),
     setof([qualifiers(Adjectives,Hedges)],
          (setall(Adjective,(
               APhrase =..
[Language,adjective([Adjective]),_,Phrase,_,heads],

     (ancestor([Language,NPhrase],[Language,APhrase,true],_
);
               NPhrase = APhrase)),
               Adjectives),
          findall(Hedge,(
               HPhrase =..
[Language,adverb([Hedge]),_,Phrase,_,heads],
    ancestor([Language,NPhrase],[Language,HPhrase,true],_)
),
            Hedges)
        ),
        Qualifiers1),
    QualifiersM = [object(MainObject)|Qualifiers1],

/* extract the elements of the verb phrase */
    setall(QualifiersS,
    (NVPhrase =..
[Language,noun_phrase(_),_,Phrase,_,heads],
    ancestor([Language,VPhrase],[Language,NVPhrase,true],_
),
    OPhraseV =..
[Language,noun(SubObject),_,Phrase,_,heads],
    OPhraseV2 =..
[Language,noun(SubObject2),_,Phrase,_,heads],
    ancestor([Language,NVPhrase],[Language,OPhraseV,true],
_),
    \+((ancestor([Language,NVPhrase],[Language,OPhraseV2,t
rue],_),SubObject \= SubObject2)),

    setof([qualifiers(Adjectives,Hedges)],
        (setall(Adjective,(
            APhrase =..
[Language,adjective([Adjective]),_,Phrase,_,heads],

    (ancestor([Language,NVPhrase],[Language,APhrase,true],
_);
            NVPhrase = APhrase)),
            Adjectives),
        findall(Hedge,(
            HPhrase =..
[Language,adverb([Hedge]),_,Phrase,_,heads],

    ancestor([Language,NVPhrase],[Language,HPhrase,true],_
)),
            Hedges)
        ),
        Qualifiers2),
    QualifiersS = [object(SubObject)|Qualifiers2]),
    QualifiersSS),
    refine_qualifier_hedges(QualifiersSS,QualifiersSSR).
```

```
expand(heads,22,
[[Language,SPhrase],
[Language,NPhrase],
[Language,APhrase]],
          possible((0.0,0.1,0.9)),
          [[heads,
          descriptor(P,Qualifiers)]],
          _,n):-

     SPhrase =..
[Language,comparison_phrase(P),_,Phrase,_,heads],

/* extract the elements of the subject noun phrase */

     OPhrase1 =..
[Language,noun(Subject),_,Phrase,_,heads],
     ancestor([Language,SPhrase],[Language,OPhrase1,true],_
),
     setof([qualifiers(Adjectives,Hedges)],
          (setall(Adjective,(
               NAPhrase =..
[Language,adjective([Adjective]),_,Phrase,_,heads],

     (ancestor([Language,NPhrase],[Language,NAPhrase,true],
_);
               NPhrase = NAPhrase)),
               Adjectives),
          findall(Hedge,(
               HPhrase =..
[Language,adverb([Hedge]),_,Phrase,_,heads],

     ancestor([Language,NPhrase],[Language,HPhrase,true],_)
),
               Hedges)
          ),
          Qualifiers1),

/* extract the elements of the adjectival phrase */

          setof([qualifiers(Adjectives,Hedges)],
          (setall(Adjective,(
               AAPhrase =..
[Language,adjective([Adjective]),_,Phrase,_,heads],

     (ancestor([Language,APhrase],[Language,AAPhrase,true],
_);
```

```
               APhrase = AAPhrase)),
               Adjectives),
          findall(Hedge,(
               HPhrase =..
[Language,adverb([Hedge]),_,Phrase,_,heads],

     ancestor([Language,APhrase],[Language,HPhrase,true],_)
),
               Hedges)
          ),
          Qualifiers2),
          append(Qualifiers1,Qualifiers2,Qualifiers12),
          Qualifiers = [[object(Subject),Qualifiers12]].

refine_qualifier_hedges([],[]).
refine_qualifier_hedges([[object(O),Q1s]|Os],NewOs):-
     delete_element([object(O),Q2s],Os,IOs),
     merge_qualifier_hedges(Q1s,Q2s,Qs),
     !,
     refine_qualifier_hedges([[object(O),Qs]|IOs],NewOs).
refine_qualifier_hedges([[object(O),Q1s]|Os],NewOs):-
     delete_element([object(O),Q2s],Os,IOs),
     append(Q1s,Q2s,Q12s),
     refine_qualifier_hedges([[object(O),Q12s]|IOs],NewOs).
refine_qualifier_hedges([[object(O),Q1s]|Os],[[object(O),Q1
s]|NewOs]):-
     \+member([object(O),Q2s],Os),
     !,
     refine_qualifier_hedges(Os,NewOs).

merge_qualifier_hedges(Qs,Qs,Qs).
merge_qualifier_hedges(Q1s,Q2s,Qs):-
     append(Q1s,Q2s,UZQs),
     delete_all(qualifiers([],[]),UZQs,UQs),
     sort(UQs,Qs).
merge_hedges([],Qs,Qs).
merge_hedges([qualifiers([Q],H1s)|Q1s],Q2s,Qs):-
     delete_element(qualifiers([Q],H2s),Q2s,Q2Ds),
     append(H1s,H2s,UHs),
     sort(UHs,Hs),
     merge_hedges([qualifiers([Q],Hs)|Q1s],Q2Ds,Qs).
merge_hedges([qualifiers([Q],H1s)|Q1s],Q2s,[qualifiers([Q],
H1s)|Qs]):-
     \+member(qualifiers([Q],H2s),Q2s),
     merge_hedges(Q1s,Q2s,Qs).
```

```
load_head_object(X):-

     files(engines('heads:templates'),Files),
     member(X,Files),
     seeing(Old),
 cat(['heads:templates:',X],TX,_),
 see(engines(TX)),
 load_head_object1,
 seen,
 see(Old).
 /* ;nl,
     cat(['Heads template file ',X,' not
found'],Message,_),
     write(Message),nl). */

load_head_object1:-
     read(Term),
     deal_with_head_term(Term).

deal_with_head_term(end_of_file):-
     !.
deal_with_head_term(end):-
     !.
deal_with_head_term(template(X)):-
     X=..[F|_],
     XX=..[F|_],
     retractall(template(XX)),
     assert(template(X)),
     load_head_object1.

apply_template_to_head:-
     retractall(head(_)),
     forall(template(H),(H =..  [F,X],hoover(X,XX),HH =..
     [F,XX],assert(head(HH)))).

hoover([],[]).
hoover([(F,V,M)|X],[(F,V)|XX]):-
     hoover(X,XX).

print_head_object:-
     do_value(head),
     do_value(nose),
     do_value(chin),
     do_value(jaw),
     do_value(cheek),
     do_value(eyes),
     do_value(ears),
     do_value(mouth),
     do_value(eye1translate),
     do_value(eye2translate),
     do_value(eye1rotate),
     do_value(eye2rotate).

do_value(X):-
     F=..[X,ValueList],
     head(F),
     do_values(X,ValueList).

do_values(_,[]):-
     !.
do_values(X,[(N,V)|R]):-
     write(X),
     write(' '),
     write(N),
     write(' = '),
     write(V),
     nl,
     do_values(X,R).

/*apply_wholehead_modifier([]):-
     !
apply_wholehead_modifier([object(O),Q|RL]):- */

do_head_object([]):-
     !.
do_head_object([[object(ObjectFeature),Q]|RD]):-
      load_modifiers(ObjectFeature,Q),
     /* apply_modifier(O),
     apply_qualifiers(O,Q), */
     do_head_object(RD).

modify_head_object([]):-
     !.
modify_head_object([qualifiers([],_)|Qs]):-
     modify_head_object(Qs).
modify_head_object([qualifiers([Q],H)|Qs]):-
     load_modifiers1([head],qualifiers(Q,H)),

modify_head([nose,chin,jaw,cheek,eyes,ears,mouth,eye1transl
ate,eye2translate,eye1rotate,eye2rotate],qualifiers(Q,H)),
     modify_head_object(Qs).
```

```
modify_head([],qualifiers(Q,H)):-
     !.
modify_head([ObjectFeature|FeatureRem],qualifiers(Q,H)):-
     apply_modifier([ObjectFeature],qualifiers(Q,H)),
     modify_head(FeatureRem,qualifiers(Q,H)).

load_modifiers(ObjectFeature, []):-
     !.
load_modifiers(ObjectFeature, [qualifiers([],_)|Qs]):-
     load_modifiers(ObjectFeature,Qs).
load_modifiers(ObjectFeature, [qualifiers([Q],H)|Qs]):-
     load_modifiers1(ObjectFeature,qualifiers(Q,H)),
     load_modifiers(ObjectFeature,Qs).

load_modifiers1(ObjectFeature,qualifiers(Q,H)):-
     files(engines('heads:modifiers'),Files),
     /* member(qualifiers([X]),Q), */
     (member(Q,Files)
      -> seeing(Older),
         cat(['heads:modifiers:',Q],TX,_),
             see(engines(TX)),
         load_modifier_object,
         seen,
         see(Older),
            /* synonym(ObjectFeature,Os), */
        apply_modifier(ObjectFeature,qualifiers(Q,H)),
        write('Modifiers Applied to head'),
        nl
     ;apply_qualifiers(ObjectFeature,qualifiers(Q,H))).


/* load_modifiers(O,[]):-
     !.
load_modifiers(O,[[qualifiers([],_)]|Mods]):-
     !,
     load_modifiers(O,Mods).
load_modifiers(O,[[qualifiers([Q|Qs],H)]|Mods]):-
     load_modifiers1(O,[qualifiers(Q,H)]),
     load_modifiers(O,[[qualifiers(Qs,H)]|Mods]).*/

load_modifier_object:-
     read(Term),
     deal_with_modifier_term(Term).

deal_with_modifier_term(end_of_file):-
     !.
```

```
deal_with_modifier_term(end):-
     !.
deal_with_modifier_term(modifier(X)):-
     X=..[F|_],
     XX=..[F|_],
     retractall(modifier(XX)),
     assert(modifier(X)),
     load_modifier_object.


apply_modifier(ObjectFeature,qualifiers(Q,H)):-
     synonym(ObjectFeature,Os),
     HeadF =..[Os,ValueListH],
     head(HeadF),
     ModifierF =..[Os,ValueListM],
     modifier(ModifierF),
     LimitsF =..[Os,ValueListL],
     limits(LimitsF),
 add_modifier(head(Os,ValueListH), modifier(Os,ValueListM),
     head(Os,ValueList),limits(Os,ValueListL),qualifiers(Q,
H)),
     reverse(ValueList,ReversedHeadList),
     NewHeadF=..[Os,ReversedHeadList],
     retractall(head(HeadF)),
     assert(head(NewHeadF)).

add_whole_modifier([],_,_):-
     !.
add_whole_modifier([Obj|Flist],ValueListH,ValueListM):-
     add_modifier(Obj,ValueListH,ValueListM),
     add_whole_modifier(Flist,ValueListH,ValueListM).

add_modifier(head(Feature,ValueListH),
modifier(Feature,[]), head(Feature, ValueListH),
limits(Feature, ValueListL),qualifiers(_,_)):-
     !.
add_modifier(head(Feature,ValueListH),
modifier(Feature,[(Aspect,Increment,Sign)|ValueListM]),
 head(Feature,NewH),
limits(Feature,[(Aspect,Default,Lower,Upper)|ValueListL]
),qualifiers(Q,H)):-
 /* find and delete aspect from head value list */
    delete_element((Aspect,HValue),ValueListH,IValueListH),

evaluate_modifier(HValue,Default,Lower,Upper,Increment,NewI
ncrement),
 /* calculate new value of aspect */
    ( Sign == add
    ->apply_hedges(NewIncrement,H,NewModifier),
    NewHValue is HValue + NewModifier
    ;NewHValue is Increment),
 /* pass on new values and apply remainder of modifier */
add_modifier(head(Feature,[(Aspect,NewHValue)|IValueListH])
,modifier(Feature,ValueListM), head(Feature,NewH),
limits(Feature,ValueListL), qualifiers(Q,H)).

evaluate_modifier(HValue,Default,Lower,Upper,Increment,NewI
ncrement):-
     HValue == Default,
 NewIncrement is Increment.

evaluate_modifier(HValue,Default,Lower,Upper,Increment,NewI
ncrement):-
     HValue > Default,
     (HValue < Upper
     ->InterValueA is 1/(0.8*sqrt(2*3.14)),
       InterValueB is (HValue - Default)^2,
       InterValueC is 2*(0.8^2),
   InterValueD is aln(InterValueB/InterValueC),
   NewIncrement is InterValueA*InterValueD
     ;NewIncrement is 0).

evaluate_modifier(HValue,Default,Lower,Upper,Increment,NewI
ncrement):-
     HValue < Default,
     (HValue > Lower
     -> InterValueA is 1/(0.8*sqrt(2*3.14)),
        InterValueB is (HValue - Default)^2,
        InterValueC is 2*(0.8^2),
    InterValueD is aln(InterValueB/InterValueC),
    NewIncrement is (InterValueA*InterValueD),
            (sign(Increment) =:= -1
                    ->NewIncrement is -(NewIncrement)
    ;NewIncrement is NewIncrement)
     ;NewIncrement is 0).

apply_qualifiers(O,[]):-
     !.
apply_qualifiers(O,[qualifiers(Q,H)|Qs]):-
     head_semantic_map(Q,F,PlusMinus),
     synonym(O,OS),
     Feature =.. [OS,ValueList],
     template(Feature),
     HeadFeature =.. [OS,HeadValueList],
     head(HeadFeature),
     member((F,_,Modifier),ValueList),
     delete_element((F,Value),HeadValueList,InterHeadValueL
ist),
     Modifier is Modifier * PlusMinus,
     apply_hedges(Modifier,H,NewModifier),
     NewValue is Value + NewModifier,
     NewHeadFeature =..
[OS,[(F,NewValue)|InterHeadValueList]],
     retract(head(HeadFeature)),
     assert(head(NewHeadFeature)),
     apply_qualifiers(O,Qs).

apply_hedges(Modifier,[],Modifier):-
     !.
apply_hedges(Modifier,[H|Hs],NewModifier):-
     apply_hedge(Modifier,H,InterModifier),
     apply_hedges(InterModifier,Hs,NewModifier).

apply_hedge(Modifier,very,NewModifier):-
     NewModifier is Modifier^2.
apply_hedge(Modifier,fairly,NewModifier):-
     NewModifier is Modifier^0.5.
apply_hedge(Modifier,slightly,NewModifier):-
     NewModifier is Modifier^0.5.

:-dynamic head_semantic_map/3.
head_semantic_map(wide,width,1).
head_semantic_map(long,length,1).
head_semantic_map(small,height,-1).

:-dynamic synonym/2.
synonym([X],X).
synonym([ears],ear).

/* Fuzzy Variables */

fuzzy_variable(forehead_slope):-
     [0,1];
     receeding, \, linear, [0,0.5];
     vertical, /\, linear, [x,y,z];
     bulging, /, linear, [yz].
fuzzy_variable(eye_width):-
     [-1,1];
     small, \, linear, [-1, -0.1];
     medium, /\, linear, [0, 0.2, 0.4];
     large, /, linear, [0.5, 1].
fuzzy_variable(eye_open):-
     [0,1];
     narrow, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     wide, /, linear, [yz].
fuzzy_variable(eye_seperation):-
     [0,1];
     close, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     wide, /, linear, [yz].
fuzzy_variable(nose_length):-
     [0,1];
     short, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     long, /, linear, [yz].
fuzzy_variable(nose_width):-
     [0,1];
     small, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     large, /, linear, [yz].
fuzzy_variable(nose_tip):-
     [0,1];
     upward, \, linear, [0,0.5];
     horizontal, /\, linear, [x,y,z];
     downward, /, linear, [yz].
fuzzy_variable(nose_profile):-
     [0,1];
     concaved, \, linear, [0,0.5];
     straight, /\, linear, [x,y,z];
     hooked, /, linear, [yz].
fuzzy_variable(mouth_width):-
     [0,1];
     small, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     wide, /, linear, [yz].
fuzzy_variable(mouth_protrusion):-
     [0,1];
     slight, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     large, /, linear, [yz].
fuzzy_variable(ear_length):-
     [0,1];
     short, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     long, /, linear, [yz].
fuzzy_variable(ear_protrusion):-
     [0,1];
     slight, \, linear, [0,0.5];
     medium, /\, linear, [x,y,z];
     large, /, linear, [yz].


:-dynamic modifier/1.
modifier(head([(texture,1.0,assign),(type,2.0,assign),(stre
ngth,0.0,add),(x_pull,100.0,add),(y_pull,100.0,add),(z_pull
,100.0,add),(y_offset,0.0,add),(z_offset,0.0,add),(width,10
0.0,add),(widthskew1,0.0,add),(widthskew2,0.0,add),(depth,1
00.0,add),(depthskew,0.0,add),(height,100.0,add),(heightske
w,0.0,add),(face_squash,1.0,add),(flatten,1.0,add),(slope,0
.0,add)])).
modifier(nose([(width,1.0,add),(width_zweight,0.0,add),(len
gth,1.0,add),(length_zweight,0.0,add),(pullup,1.0,add),(bri
dge,1.0,add),(hook,1.0,add),(hook_influence,0.0,add)])).
modifier(chin([(extent,1.0,add),(tilt,1.0,add),(tilt_influe
nce,1.0,add),(accent,0.0,add)])).
modifier(jaw([(width,0.0,add),(influence,0.0,add),(uniformi
ty,0.0,add)])).
modifier(cheek([(extrude,0.0,add),(zpos,0.0,add),(curvature
s,0.0,add),(curvature_zpos,0.0,add),(curvature_ypos,0.0,add
),(curvature_zfalloff,0.5,add),(curvature_yfalloff,0.5,add)
])).
modifier(eyes([(colour,6.0,assign),(separation,1.0,add),(in
set,0.0,add),(toproundness,0.0,add),(bottomroundness,0.0,ad
d),(rotation,0.0,add),(brow_bulge,0.0,add)])).
modifier(ears([(height,0.0,add),(lobe,0.0,add),(depth,0.0,a
dd),(rotation,0.0,add)])).
modifier(mouth([(protrude,0.0,add),(width,0.0,add)])).
modifier(eye1translate([(x,0.0,add),(y,0.0,add),(z,0.0,add)
])).
modifier(eye2translate([(x,0.0,add),(y,0.0,add),(z,0.0,add)
])).
modifier(eye1rotate([(x,1.0,add),(y,0.0,add),(z,-
1.0,add)])).
modifier(eye2rotate([(x,1.0,add),(y,0.0,add),(z,-
1.0,add)])).

:-dynamic head/1.
head(head([(texture,1.0,0),(type,2.0,0.0),(strength,0.0,1.0
),(x_pull,100.0,1.0),(y_pull,100.0,1.0),(z_pull,100.0,1.0),
(y_offset,0.0,0.1),(z_offset,0.0,0.1),(width,100.0,10.0),(w
idthskew1,0.0,0.1),(widthskew2,0.0,0.1),(depth,100.0,10.0),
(depthskew,0.0,0.1),(height,100.0,10.0),(heightskew,0.0,0.1
),(face_squash,1.0,1.0),(flatten,1.0,0.1),(slope,0.0,0.1)])
).
head(nose([(width,1.0,0.1),(width_zweight,0.0,0.1),(length,
1.0,0.2),(length_zweight,0.0,0.1),(pullup,1.0,0.1),(bridge,
1.0,0.1),(hook,1.0,0.1),(hook_influence,0.0,0.1)])).
head(chin([(extent,1.0,0.1),(tilt,1.0,0.1),(tilt_influence,
1.0,0.1),(accent,0.0,0.1)])).
head(jaw([(width,0.0,0.1),(influence,0.0,0.1),(uniformity,0
.0,0.1)])).
head(cheek([(extrude,0.0,0.1),(zpos,0.0,0.1),(curvatures,0.
0,0.1),(curvature_zpos,0.0,0.1),(curvature_ypos,0.0,0.1),(c
urvature_zfalloff,0.5,0.1),(curvature_yfalloff,0.5,0.1)])).
head(eyes([(colour,6.0,0),(separation,1.0,0.1),(inset,0.0,0
.1),(toproundness,0.0,0.1),(bottomroundness,0.0,0.1),(rotat
ion,0.0,0.1),(brow_bulge,0.0,0.1)])).
head(ears([(height,0.0,0.2),(lobe,0.0,0.2),(depth,0.0,0.1),
(rotation,0.0,0.1)])).
head(mouth([(protrude,0.0,0.2),(width,0.0,0.1)])).
head(eye1translate([(x,0.0,0),(y,0.0,0),(z,0.0,0)])).
head(eye2translate([(x,0.0,0),(y,0.0,0),(z,0.0,0)])).
head(eye1rotate([(x,1.0,0),(y,0.0,0),(z,-1.0,0)])).
head(eye2rotate([(x,1.0,0),(y,0.0,0),(z,-1.0,0)])).

:-dynamic limits/1.
limits(head([(texture,0,0,0),(type,0,0,0),(strength,0,10,0)
,(x_pull,100,1,100),(y_pull,100,1,100),(z_pull,100,1,100),(
y_offset,0,-1,1),(z_offset,0,-
1,1),(width,1,0.1,100),(widthskew1,0,-1,1),(widthskew2,0,-
1,1),(depth,1,0.1,100),(depthskew,0,-
1,1),(height,1,0.1,100),(heightskew,0,-
1,1),(face_squash,1,0,20),(flatten,1,0,2),(slope,0,-
1,1)])).
limits(nose([(width,1,0,2),(width_zweight,0,-
1,1),(length,1,0,3),(length_zweight,0,-
1,1),(pullup,1,0,2),(bridge,0,-1,1),(hook,0,-
1,1),(hook_influence,0,-1,1)])).
limits(chin([(extent,1,0,2),(tilt,1,0,2),(tilt_influence,0,
-1,1),(accent,0,-1,1)])).
limits(jaw([(width,0,0,1),(influence,0,0,1),(uniformity,0,-
1,1)])).
limits(cheek([(extrude,0,-1,1),(zpos,0,-
1,1),(curvatures,0,-1,1),(curvature_zpos,0,-
1,1),(curvature_ypos,0,-
1,1),(curvature_zfalloff,0.5,0,1),(curvature_yfalloff,0.5,0
,1)])).
limits(eyes([(colour,0,0,0),(separation,1,0,2),(inset,0,-
1,1),(toproundness,0,-1,1),(bottomroundness,0,-
1,1),(rotation,0,-1,1),(brow_bulge,0,0,1)])).
limits(ears([(height,0,0,2),(lobe,0,0,1),(depth,0,-
1,2),(rotation,0,-1,1)])).
limits(mouth([(protrude,0,-1,1),(width,0,-1,1)])).
limits(eye1translate([(x,0.0,0),(y,0.0,0),(z,0.0,0)])).
limits(eye2translate([(x,0.0,0),(y,0.0,0),(z,0.0,0)])).
limits(eye1rotate([(x,1.0,0),(y,0.0,0),(z,-1.0,0)])).
limits(eye2rotate([(x,1.0,0),(y,0.0,0),(z,-1.0,0)])).


:-dynamic template/1.
template(head([(texture,1.0,0),(type,2.0,0.0),(strength,0.0
,1.0),(x_pull,100.0,1.0),(y_pull,100.0,1.0),(z_pull,100.0,1
.0),(y_offset,0.0,0.1),(z_offset,0.0,0.1),(width,100.0,10.0
),(widthskew1,0.0,0.1),(widthskew2,0.0,0.1),(depth,100.0,10
.0),(depthskew,0.0,0.1),(height,100.0,10.0),(heightskew,0.0
,0.1),(face_squash,1.0,1.0),(flatten,1.0,0.1),(slope,0.0,0.
1)])).
template(nose([(width,1.0,0.1),(width_zweight,0.0,0.1),(len
gth,1.0,0.2),(length_zweight,0.0,0.1),(pullup,2.0,0.1),(bri
dge,1.0,0.1),(hook,1.0,0.1),(hook_influence,0.0,0.1)])).
template(chin([(extent,1.0,0.1),(tilt,1.0,0.1),(tilt_influe
nce,1.0,0.1),(accent,0.0,0.1)])).
template(jaw([(width,0.0,0.1),(influence,0.0,0.1),(uniformi
ty,0.0,0.1)])).
template(cheek([(extrude,0.0,0.1),(zpos,0.0,0.1),(curvature
s,0.0,0.1),(curvature_zpos,0.0,0.1),(curvature_ypos,0.0,0.1
),(curvature_zfalloff,0.5,0.1),(curvature_yfalloff,0.5,0.1)
])).
template(eyes([(colour,6.0,0),(separation,1.0,0.1),(inset,0
.0,0.1),(toproundness,0.0,0.1),(bottomroundness,0.0,0.1),(r
otation,0.0,0.1),(brow_bulge,0.0,0.1)])).
template(ears([(height,0.0,0.2),(lobe,0.0,0.2),(depth,0.0,0
.1),(rotation,0.0,0.1)])).
template(mouth([(protrude,0.0,0.2),(width,0.0,0.1)])).
template(eye1translate([(x,0.0,0),(y,0.0,0),(z,0.0,0)])).
template(eye2translate([(x,0.0,0),(y,0.0,0),(z,0.0,0)])).
template(eye1rotate([(x,1.0,0),(y,0.0,0),(z,-1.0,0)])).
template(eye2rotate([(x,1.0,0),(y,0.0,0),(z,-1.0,0)])).


/************************* END *************************/
```

```
/********************* ENGLISH ENGINE *******************/

/* false detection */

/* :-multifile expand/7,set_bid/1,tms_inconsistent/3. */

tms_inconsistent(english,L1,L2):-
     consistent_meaning(english,L1,L2,ConVal),
     !,
     ConVal = false.

expand(english,0,
        [[english,S1],
        [english,S2]],
        necessary,
        [[_,ConVal]],_,c):-
        consistent_meaning(english,S1,S2,ConVal).

/*  lexical analysis */

set_bid(english):-
     make_bid(english,1,
          [[phrase,_,true]],
          100,
          [[english,english(_,_,_,_,_)]]).


/*  syntax analysis */

set_bid(english):-
     grammar(english,_,[BareAntecedent|BareAntecedents],_),
     construct_antecedents(english,[BareAntecedent],Anteced
ent),
     make_bid(english,2,
          Antecedent,
          100,
          [[english,english(_,_,_,_,_)]]).


/***************************** END ***********************/


/********************** ENGLISH GRAMMAR *****************/

/* english grammar */
```

```
:-dynamic grammar/4.
:-multifile grammar/4.

grammar(english,sentence,
          [noun_phrase, verb_phrase],100).
grammar(english,sentence,
          [imperative_verb,noun_phrase],100).
grammar(english,sentence,
          [sentence,conjunction,sentence],100).
grammar(english,noun_phrase,
          [noun],100).
grammar(english,noun_phrase,
          [noun_phrase,possessor,noun_phrase],100).
grammar(english,noun_phrase,
          [noun_phrase,conjunction,noun_phrase],100).
grammar(english,noun_phrase,
          [adjective_phrase,noun_phrase],100).
grammar(english,adjective_phrase,
          [adjective],100).
grammar(english,adjective_phrase,
          [adverb,adjective],100).
grammar(english,noun_phrase,
          [noun_phrase,comparison_phrase],100).
grammar(english,noun_phrase,
          [indefinite_article,noun_phrase],100).
grammar(english,noun_phrase,
          [definite_article,noun_phrase],100).
grammar(english,verb_phrase,
          [intransitive_verb],100).
grammar(english,verb_phrase,
          [verb_phrase,prepositional_phrase],100).
grammar(english,verb_phrase,
          [transitive_verb,noun_phrase],100).
grammar(english,prepositional_phrase,
          [preposition,noun_phrase],100).
grammar(english,prepositional_phrase,
          [prepositional_phrase,
               conjunction,
               noun_phrase],100).
grammar(english,comparison_phrase,
          [noun_phrase,comparator,noun_phrase],100).
grammar(english,comparison_phrase,
          [noun_phrase,comparator,adjective_phrase],100).


/*************************** END *************************/
```

```
/*********************** ENGLISH LEX *********************/

/* english lexemes */

:-multifile lexical/4.
:-dynamic lexical/4.

lexical(english,noun,[fruit],100).
lexical(english,noun,[time],100).
lexical(english,noun,[nose],100).
lexical(english,noun,[ears],100).
lexical(english,noun,[eyes],100).
lexical(english,noun,[mouth],100).
lexical(english,noun,[head],100).
lexical(english,noun,[chin],100).
lexical(english,noun,[cheek],100).
lexical(english,noun,[jaw],100).
lexical(english,noun,[man],100).
lexical(english,noun,[men],100).
lexical(english,noun,[woman],100).
lexical(english,noun,[train],100).
lexical(english,noun,[boxer],100).
lexical(english,noun,[male],100).
lexical(english,noun,[female],100).
lexical(english,noun,[african],100).
lexical(english,noun,[european],100).
lexical(english,noun,[oriental],100).
lexical(english,noun,[caucasian],100).
lexical(english,adjective,[large],100).
lexical(english,adjective,[small],100).
lexical(english,adjective,[big],100).
lexical(english,adjective,[flat],100).
lexical(english,adjective,[fat],100).
lexical(english,adjective,[slim],100).
lexical(english,adjective,[wide],100).
lexical(english,adjective,[narrow],100).
lexical(english,adjective,[beautiful],100).
lexical(english,adjective,[african],100).
lexical(english,adjective,[european],100).
lexical(english,adjective,[oriental],100).
lexical(english,adjective,[caucasian],100).
lexical(english,adjective,[blue],100).
lexical(english,adjective,[hazel],100).
lexical(english,adjective,[grey],100).
lexical(english,adjective,[aquamarine],100).
lexical(english,adjective,[green],100).
lexical(english,adjective,[brown],100).
lexical(english,adjective,[vampire],100).
lexical(english,adjective,[bony],100).
lexical(english,adjective,[broad],100).
lexical(english,adjective,[bulbous],100).
lexical(english,adjective,[bulging],100).
lexical(english,adjective,[closeset],100).
lexical(english,adjective,[full],100).
lexical(english,adjective,[hooked],100).
lexical(english,adjective,[jutting],100).
lexical(english,adjective,[long],100).
lexical(english,adjective,[oval],100).
lexical(english,adjective,[protruding],100).
lexical(english,adjective,[puffed],100).
lexical(english,adjective,[pugged],100).
lexical(english,adjective,[receding],100).
lexical(english,adjective,[round],100).
lexical(english,adjective,[short],100).
lexical(english,adjective,[slantingdown],100).
lexical(english,adjective,[slantingup],100).
lexical(english,adjective,[squared],100).
lexical(english,adjective,[squinted],100).
lexical(english,adjective,[sunken],100).
lexical(english,adjective,[thin],100).
lexical(english,adjective,[wideapart],100).
lexical(english,adverb,[very],100).
lexical(english,adverb,[fairly],100).
lexical(english,adverb,[quite],100).
Lexical(english,adverb,[slightly],100).
lexical(english,noun,[he],100).
lexical(english,noun,[jake],100).
lexical(english,transitive_verb,[has],100).
lexical(english,noun,[flies],100).
lexical(english,adjective,[fruit],100).
lexical(english,imperative_verb,[eat],100).
lexical(english,imperative_verb,[sell],100).
lexical(english,intransitive_verb,[flies],100).
lexical(english,transitive_verb,[like],100).
lexical(english,possessor,[of],100).
lexical(english,preposition,[like],100).
lexical(english,preposition,[with],100).
lexical(english,indefinite_article,[a],100).
lexical(english,indefinite_article,[an],100).
lexical(english,definite_article,[the],100).
lexical(english,definite_article,[draw],100).
lexical(english,noun,[banana],100).
lexical(english,noun,[arrow],100).
lexical(english,noun,[shop],100).
lexical(english,conjunction,[and],100).
lexical(english,disjunction,[or],100).
lexical(english,transitive_verb,[sells],100).
lexical(english,transitive_verb,[list],100).
lexical(english,noun,[shops],100).
lexical(english,adjective,[shops],100).
lexical(english,noun,[product],100).
lexical(english,noun,[bread],100).
lexical(english,transitive_verb,[is],100).
lexical(english,comparator,[is],100).


/*************************** END *************************/


/************************** PHRASE ***********************/

:- dynamic active/0.

describe:-
     reset,
     amplify.
amplify:-
     setall(Language,(grammar(Language,_,_,_);lexical(Langu
age,_,_,_)),Languages),
     setall(Da,echo_database(Da),Ds),
     (describe_boxes(OldSource,OldPhrase,OldDataBase);
     Es = [
     'the man has a large nose',
     'the woman has small ears',
     'the man has a very wide nose',
     'the very fat woman has large eyes',
     'the woman is very beautiful',
     'the very slim woman is fairly beautiful'
     ],
     Es = [OldE|_],
     TD = 230,
     TW = 310,
     centred(TT,TL,TD,TW),

     mdialog(TT,TL,TD,TW,
     [button(187,220,26,80,'Ok'),
          button(190,10,20,80,'Cancel'),
          text(10,10,20,190,'English'),
          menu(30,10,150,290,Es,OldE,OldPhrase)
     ],
     _),
     OldSource = english,
     OldDataBase = heads),
     (member(TOldDataBase,Ds),
     OldDataBase = TOldDataBase,
     !;
     [OldDataBase|_] = Ds),
     (member(TTopLanguage,Languages),
     TopLanguage = TTopLanguage,
     !;
     [TopLanguage|_] = Languages),
     D = 350,
     W = 310,

     centred(T,L,D,W),
     mdialog(T,L,D,W,
     [button(317,220,26,80,'Ok'),
          button(320,10,20,80,'Cancel'),
          button(150,220,20,80,'Load'),
          text(10,10,20,290,'Description'),
          edit(30,10,100,290,OldPhrase,Phrase),
          text(180,10,20,140,'Language'),

     menu(210,10,100,140,Languages,TopLanguage,QLanguage),
          text(180,160,20,140,'Target Database'),
          menu(210,160,100,140,Ds,OldDataBase,DataBase)
     ],
     Button),
     (Button = 1,
          atom_string(Phrase,SPhrase),
          cat([SPhrase,` .`],SPhrase_Dot,_),
          read_in(SWords_Dot) <~ SPhrase_Dot,
          append(SWords,['.'],SWords_Dot),
          statistics(runtime,Time),
          assert(start_time(Time)),
          retractall(describe_boxes(_,_,_)),

     assert(describe_boxes(QLanguage,Phrase,DataBase)),
     assume(phrase,echo(QLanguage,SWords,DataBase),100);
     Button = 3,
     retractall(describe_boxes(_,_,_)),
     amplify).

/********************** END PHRASE **********************/
```

```
/*********************** OPEN ECHO ***********************/

file_search_path(echo,'Macintosh HD:Echo:').

'<LOAD>'(_):-
     abolish('<LOAD>'/1),
     source_load(echo(open_tms)),
     source_load(echo(echo)),
     load_engine(echo),
     /*load_engine(sql),*/
     load_engine(english),
     /*load_engine(punjabi),*/
     load_engine(mapper),
     install_menu('ECHO',['Database','Query','Describe','Am
plify','Acquire','Print New Language','Consolidate
Language','Print Language','Save Language','Reset
Language','Reset']).

'ECHO'('Query'):-
     query.
'ECHO'('Database'):-
     database.
'ECHO'('Describe'):-
     describe.
'ECHO'('Amplify'):-
     amplify.
/*'ECHO'('Acquire'):-
     acquire.*/
'ECHO'('Print New Language'):-
     print_new_language.
'ECHO'('Consolidate Language'):-
     consolidate_language.
'ECHO'('Print Language'):-
     print_language.
'ECHO'('Save Language'):-
     save_language.
'ECHO'('Reset Language'):-
     reset_language.
'ECHO'('Reset'):-
     reset,
     retractall(unisql_collection(_)),
     retractall(old_unisql_collection(_)).

/*********************************************************/
```

```
/*********************** OPEN TMS ************************/

file_search_path(tms,'Macintosh HD:Echo:').
file_search_path(engines,'Macintosh HD:Echo:ENGINES:').
logic_style(0.95).
:- multifile tms_inconsistent/3.
:- multifile tms_equivalent/3.
:- multifile trivial/3.
:- multifile subsumes/3.

'<LOAD>'(_):-
     abolish('<LOAD>'/1),
     source_load(tms(aardvaark)),
     source_load(tms(tms)),
     install_menu('TMS',['Propagate','Reset','Show','Show
file','Reporting','Load Engine']),
     install_menu('Reporting',['On','Rating','Derivation'],
'TMS'('Reporting')),
     mark_item('Reporting','On'),
     mark_item('Reporting','Rating'),
     mark_item('Reporting','Derivation'),
     init,
     load_engine(human),
     load_engine(result).

'TMS'('Propagate'):-
     propagate.
'TMS'('Reset'):-
     reset.
'TMS'('Show'):-
 show.
'TMS'('Show file'):-
     retract(current_window(Name,Type,Comment)),
     assert(current_window(Name,blackboard,Comment)),
 show,
     retract(current_window(Name,_,Comment)),
     assert(current_window(Name,Type,Comment)).
'Reporting'('On'):-
 reporting(Reps),
     append(L,[on|R],Reps),
     append(L,R,NewReps),
 retract(reporting(_)),
     assert(reporting(NewReps)),
     unmark_item('Reporting','On').
'Reporting'('On'):-
 retract(reporting(Reps)),
     \+member(on,Reps),
     assert(reporting([on|Reps])),
     mark_item('Reporting','On').
'Reporting'('Rating'):-
 reporting(Reps),
     append(L,[rating|R],Reps),
     append(L,R,NewReps),
 retract(reporting(Reps)),
     assert(reporting(NewReps)),
     unmark_item('Reporting','Rating').
'Reporting'('Rating'):-
 reporting(Reps),
     \+member(rating,Reps),
 retract(reporting(_)),
     assert(reporting([rating|Reps])),
     mark_item('Reporting','Rating').
'Reporting'('Derivation'):-
 reporting(Reps),
     append(L,[derivation|R],Reps),
     append(L,R,NewReps),
 retract(reporting(_)),
     assert(reporting(NewReps)),
     unmark_item('Reporting','Derivation').
'Reporting'('Derivation'):-
 reporting(Reps),
     \+member(derivation,Reps),
 retractall(reporting(_)),
     assert(reporting([derivation|Reps])),
     mark_item('Reporting','Derivation').
'Reporting'(X):-
     \+reporting(_),
     assert(reporting([on])).
'TMS'('Load Engine'):-
     folders(engines,UEs),
     sort(UEs,Es),
     D = 200,
     W = 200,
 centred(T,L,D,W),
     mdialog(T,L,D,W,
     [button(167,110,26,80,'Ok'),
          button(170,10,20,80,'Cancel'),
          text(10,10,20,190,'Engine'),
     menu(40,10,100,190,Es,[],LEngine)],
     Button),
     LEngine = [Engine],
     load_engine(Engine).
```

```
/**********************************************************/


/* ********************************************************/
/*            Fuzzy Mass Assignment Reduction             */
/**********************************************************/

:- multifile reduce/3.


fuzzy_number(R):-
     number(R),
     !.

fuzzy_number((_,_,_)).

fuzzy_lt(V1,V2) :-
     number(V1),
     number(V2),
     !,
     V1 < V2.
fuzzy_lt((F1,FT1,T1),(F2,FT2,T2)) :-
     T1 + FT1 < T2 + FT2,
     !.
fuzzy_lt((F1,FT1,T1),(F2,FT2,T2)) :-
     T1 + FT1 = T2 + FT2,
     !,
     T1 < T2.

fuzzy_lt(V1,(F2,FT2,T2)) :-
     number(V1),
     T1 is V1/100.0,
     T1 < T2 + FT2,
     !.
fuzzy_lt((F2,FT2,T2),V1) :-
     number(V1),
     T1 is V1/100.0,
     T1 > T2 + FT2,
     !.

reduce(fuzzy,Expression,Value):-
     logic_style(PM),
     reduce(fuzzy(PM),Expression,Value).

reduce(fuzzy(_),bottom,(1.0,0.0,0.0)):-
     !.
reduce(fuzzy(_),false,(1.0,0.0,0.0)):-
     !.
reduce(fuzzy(_),neutral,(0.0,1.0,0.0)):-
```

```
     !.
reduce(fuzzy(_),equal_false,(0.3,0.5,0.2)):-
     !.
reduce(fuzzy(_),equal,(0.33,0.33,0.33)):-
     !.
reduce(fuzzy(_),equal_true,(0.2,0.5,0.3)):-
     !.
reduce(fuzzy(_),low,(0.0,0.9,0.1)):-
     !.
reduce(fuzzy(_),high,(0.0,0.1,0.9)):-
     !.
reduce(fuzzy(_),top,(0.0,0.0,1.0)):-
     !.
reduce(fuzzy(_),true,(0.0,0.0,1.0)):-
     !.
reduce(fuzzy(_),Value,(0.0,FT,T)):-
     number(Value),
     !,
     T is Value/100.0,
     FT is 1.0 - T.
reduce(fuzzy(_),(FuF,FuT),(F,FT,T)):-
     number(FuF),
     number(FuT),
     !,
     (FuF > FuT,
     F is FuF-FuT,
     T is 0.0,
     FT is 1.0 - F;
     FuF =< FuT,
     T is FuT - FuF,
     F is 0.0,
     FT is 1.0 - T).

mass_to_fuzzy((F,FT,T),(FF,TT)):-
     FF is FT + F,
     TT is T + FT.
fuzzy_to_mass((FF,TT),(0.0,FF,T)):-
     TT >= FF,
     T is TT - FF.
fuzzy_to_mass((FF,TT),(F,FF,0.0)):-
     TT < FF,
     F is FF - TT.

truth_to_fuzzy_mass(Support,Masses,Set):-
     sort_sup(Support,Masses,SSupport,SMasses),
     to_set(SSupport,SMasses,Set).

sort_sup([],[],[],[]).
sort_sup([Element|Tail],[MassElement|MassTail],Sorted,MassSo
rted):-
```

```
     sort_sup(Tail,MassTail,SortedTail,SortedMassTail),
     sup_insert(Element,MassElement,SortedTail,SortedMassTai
l,Sorted,MassSorted).

sup_insert(Element,MassElement, [TopElement|Sorted], [TopMassE
lement|MassSorted], [TopElement|Sorted1], [TopMassElement|Mass
Sorted1]):-
     fuzzy_lt(MassElement,TopMassElement),
     !,
     sup_insert(Element,MassElement,Sorted,MassSorted,Sorted
1,MassSorted1).
sup_insert(Element,MassElement,Sorted,MassSorted, [Element|So
rted], [MassElement|MassSorted]).

to_set(SSupport,SMasses,Set):-
     to_set1(SSupport,SMasses,0.0,[],Set).

to_set1([S],[(F,FT,T)],Slack,RSupport,[[S|RSupport]:M]):-
     M is T + FT + Slack.
to_set1([Support|SSupport],[(F1,FT1,T1),(F2,FT2,T2)|SMasses]
,Slack,RSupport,[[Support|RSupport]:M|Set]):-
     M is (T1 + FT1) - (T2 + FT2) + Slack,
     to_set1(SSupport,[(F2,FT2,T2)|SMasses],0.0,[Support|RSu
pport],Set).

/* mass_to_probability(Set,PSet) */

mass_to_probability(Set,PSet):-
     mass_to_probability1(Set,[],PSet).
mass_to_probability1([],PSet,PSet).
mass_to_probability1([Set:Mass|Sets],PSet,NewPSet):-
     length(Set,LSet),
     DeltaMass is Mass/LSet,
     assign_mass(Set,DeltaMass,PSet,InterPSet),
     mass_to_probability1(Sets,InterPSet,NewPSet).

assign_mass([],DeltaMass,PSet,PSet).
assign_mass([Element|Set],DeltaMass,PSet,NewPSet):-
     (delete_element(PElement/Prob,PSet,MPSet),
          (number(Element),
               abs(Element-PElement) < 0.00001
          ;
               Element = (E1,E2),
               PElement = (PE1,PE2),
               abs(E1-PE1) < 0.00001,
               abs(E2-PE2) < 0.00001
          ),
          !,
          NewProb is Prob + DeltaMass,
     assign_mass(Set,DeltaMass,[Element/NewProb|MPSet],NewPS
et)
     ;
     assign_mass(Set,DeltaMass,[Element/DeltaMass|PSet],NewP
Set)
     ).

reduce(fuzzy([PM/P]),Expression,(FP,FTP,TP)):-
     reduce(fuzzy(PM),Expression,(F,FT,T)),
     FP is F * P,
     FTP is FT * P,
     TP is T * P.

reduce(fuzzy([PM/P|PMs]),Expression,(F,FT,T)):-
     reduce(fuzzy(PM),Expression,(F1,FT1,T1)),
     reduce(fuzzy(PMs),Expression,(FR,FTR,TR)),
     F is F1 * P + FR,
     FT is FT1 * P + FTR,
     T is T1 * P + TR.

reduce(fuzzy((AC,PM)),Op(CValue1,CValue2),(F,FT,T)):-
     atomic(AC),
     simplify(fuzzy((AC,PM)),CValue1,(F1,FT1,T1)),
     simplify(fuzzy((AC,PM)),CValue2,(F2,FT2,T2)),
     AF is (F1 + F2)/2,
     AFT is (FT1 + FT2)/2,
     AT is (T1 + T2)/2,
     reduce(fuzzy(PM),Op((F1,FT1,T1),(F2,FT2,T2)),(PF,PFT,PT
)),
     F is AC*AF + (1-AC)*PF,
     FT is AC*AFT + (1-AC)*PFT,
     T is AC*AT + (1-AC)*PT.

reduce(fuzzy((AC,PM)),Op(CValue),(F,FT,T)):-
     reduce(fuzzy((AC,PM)),CValue,(CF,CFT,CT)),
     atomic(AC),
     reduce(fuzzy(PM),Op((CF,CFT,CT)),(F,FT,T)).


reduce(fuzzy(PM),CValue1 and CValue2,(F,FT,T)):-
     atomic(PM),
     simplify(fuzzy(PM),CValue1,(F1,FT1,T1)),
     simplify(fuzzy(PM),CValue2,(F2,FT2,T2)),
     ZT is T1 * T2,
     ZFT is T1 * FT2 + FT1 * T2 + FT1 * FT2,
     ZF is F1*T2 + F2*T1 + FT1*F2 + FT2*F1 + F1 * F2,
     (PM < 0,
          MPM is -PM,
          FaF is min(1.0,F1+F2),
          FaFT is min(1.0 - FaF,min(T1 + FT1,T2 + FT2)),
          FaT is 1.0 - (FaF + FaFT),
          F3 is FaF*MPM + ZF*(1 - MPM),
          FT3 is FaFT*MPM + ZFT*(1 - MPM),
          T3 is FaT*MPM + ZT*(1 - MPM)
     ;
          PM > 0.0,
          TrT is min(T1,T2),
          TrFT is min(min(T1 + FT1,T2 + FT2) - TrT,FT1+FT2),
          TrF is 1.0 - (TrFT + TrT),
          F3 is TrF*PM + ZF*(1-PM),
          FT3 is TrFT*PM + ZFT*(1-PM),
          T3 is TrT*PM + ZT*(1-PM)
     ;
          F3 is ZF,
          FT3 is ZFT,
          T3 is ZT
     ),
     normalise_fuzzy_truth((F3,FT3,T3),(F,FT,T)).

reduce(fuzzy(PM),CValue1 or CValue2,(F,FT,T)):-
     atomic(PM),
     simplify(fuzzy(PM),CValue1,(F1,FT1,T1)),
     simplify(fuzzy(PM),CValue2,(F2,FT2,T2)),
     ZF is F1 * F2,
     ZFT is F1 * FT2 + FT1 * F2 + FT1 * FT2,
     ZT is F1 * T2 + F2 * T1 + FT1 * T2 + FT2 * T1 + T1 *
T2,
     (PM < 0,
          MPM is -PM,
          FaT is min(1.0,T1+T2),
          FaFT is min(1.0 - FaT,min(F1 + FT1,F2 + FT2)),
          FaF is 1.0 - (FaT + FaFT),
          F3 is FaF*MPM + ZF*(1 - MPM),
          FT3 is FaFT*MPM + ZFT*(1 - MPM),
          T3 is FaT*MPM + ZT*(1 - MPM)
     ;
          PM > 0.0,
          TrF is min(F1,F2),
          TrFT is min(min(F1 + FT1,F2 + FT2) - TrF,FT1 +
FT2),
          TrT is 1.0 - (TrFT + TrF),
          F3 is TrF*PM + ZF*(1-PM),
          FT3 is TrFT*PM + ZFT*(1-PM),
          T3 is TrT*PM + ZT*(1-PM)
     ;
          F3 is ZF,
          FT3 is ZFT,
          T3 is ZT
     ),
     normalise_fuzzy_truth((F3,FT3,T3),(F,FT,T)).

reduce(fuzzy(PM),CValue1 impl CValue2,(F,FT,T)):-
     atomic(PM),
     simplify(fuzzy(PM),CValue1,(F1,FT1,T1)),
     simplify(fuzzy(PM),CValue2,(F2,FT2,T2)),
     ZF is T1 * F2,
     ZFT is FT1 * F2 + FT1 * FT2 + T1*FT2,
     ZT is F1 * F2 + F1 * FT2 + F1 * T2 + FT1 * T2 + T1 *
T2,
     (PM < 0.0,
          MPM is -PM,
          TrF is min(T1,F2),
          TrFT is min(min(T1 + FT1,F2 + FT2) - TrF,FT1 +
FT2),
          TrT is 1.0 - (TrFT + TrF),
          F3 is TrF*MPM + ZF*(1 - MPM),
          FT3 is TrFT*MPM + ZFT*(1 - MPM),
          T3 is TrT*MPM + ZT*(1 - MPM)
     ;
          PM > 0.0,
          FaT is min(1.0,F1 + T2),
          FaFT is min(1.0-FaT,min(T1 + FT1,F2 + FT2)),
          FaF is 1.0 - (FaT + FaFT),
          F3 is FaF*PM + ZF*(1-PM),
          FT3 is FaFT*PM + ZFT*(1-PM),
          T3 is FaT*PM + ZT*(1-PM)
     ;
          F3 is ZF,
          FT3 is ZFT,
          T3 is ZT
     ),
     normalise_fuzzy_truth((F3,FT3,T3),(F,FT,T)).

reduce(fuzzy(PM),CValue1 equiv CValue2,(F,FT,T)):-
     atomic(PM),
     simplify(fuzzy(PM),CValue1,(F1,FT1,T1)),
     simplify(fuzzy(PM),CValue2,(F2,FT2,T2)),
     ZF is T1 * F2 + F1 * T2,
     ZFT is F1 * FT2 + T1 * FT2 + FT1 * F2 + FT1 * T2,
     ZT is F1 * F2 + T1 * T2 + FT1 * FT2,
     (PM < 0.0,
          MPM is -PM,
          TrF is min(T1,F2) + min(T2,F1),
          TrFT is min(1.0 - TrF,FT1 + FT2),
          TrT is 1.0 - (TrFT + TrF),
          F3 is TrF*MPM + ZF*(1 - MPM),
          FT3 is TrFT*MPM + ZFT*(1 - MPM),
          T3 is TrT*MPM + ZT*(1 - MPM)
     ;
          PM > 0.0,
          FaT is min(T1,T2) + min(F2,F1) + min(FT1,FT2),
          FaFT is min(F2 - min(F2,F1) + T2 - min(T2,T1),FT1
- min(FT1,FT2)) + min(F1 - min(F2,F1) + T1 - min(T2,T1),FT2
- min(FT1,FT2)),
          FaF is 1.0 - (FaT + FaFT),
          F3 is FaF*PM + ZF*(1-PM),
          FT3 is FaFT*PM + ZFT*(1-PM),
          T3 is FaT*PM + ZT*(1-PM)
     ;
          F3 is ZF,
          FT3 is ZFT,
          T3 is ZT
     ),
     normalise_fuzzy_truth((F3,FT3,T3),(F,FT,T)).

reduce(fuzzy(PM),Hedge(CValue),HValue):-
     atomic(PM),
     simplify(fuzzy(PM),CValue,Value),
     fuzzy_ma_hedge(Hedge,Value,HValue1),
     normalise_fuzzy_truth(HValue1,HValue).

fuzzy_ma_hedge(not,(F,FT,T),(T,FT,F)):-
     !.
fuzzy_ma_hedge(fairly,(F,FT,T),(VF,VFT,VT)):-
     Ft is sqrt(F),
     FTt is sqrt(FT),
     Tt is sqrt(T),
     Tot is Ft + FTt + Tt,
     VF is Ft/Tot,
     VFT is FTt/Tot,
     VT is Tt/Tot.
fuzzy_ma_hedge(very,(F,FT,T),(VF,VFT,VT)):-
     Ft is F*F,
     FTt is FT*FT,
     Tt is T*T,
     Tot is Ft + FTt + Tt,
     VF is Ft/Tot,
     VFT is FTt/Tot,
     VT is Tt/Tot.
fuzzy_ma_hedge(absolutely,(F,FT,T),(1.0,0.0,0.0)):-
     F > FT,
     F > T.
fuzzy_ma_hedge(absolutely,(F,FT,T),(0.0,0.0,1.0)):-
     T > FT,
     T > F.
fuzzy_ma_hedge(absolutely,(F,FT,T),(0.0,1.0,0.0)):-
     FT > F,
     FT > T.
```

```
normalise_fuzzy_truth((F1,FT1,T1),(F,FT,T)):-
     max(F1,0.0,F),
     max(FT1,0.0,FT),
     max(T1,0.0,T).

normalise_fuzzy(MassAssignment,NormalMassAssignment):-
     normalise_fuzzy1(MassAssignment,0.0,NormalMassAssignmen
t).

normalise_fuzzy1([Assignment:Mass],Cum,[Assignment:NormalMas
s]):-
     NormalMass is 1.0 - Cum.

normalise_fuzzy1([Assignment:Mass|MassAssignments],Cum,[Assi
gnment:Mass|NormalMassAssignments]):-
     Cum1 is Mass + Cum,
     normalise_fuzzy1(MassAssignments,Cum1,NormalMassAssignm
ents).

/*****************************************************************/

/* ********************** UTILITIES **********************/
/*      various useful general purpose utilities         */
/*******************************************************/

max(X,Y,X):-
     X >= Y.
max(Y,X,X):-
     X >= Y.
min(X,Y,X):-
     X =< Y.
min(Y,X,X):-
     X =< Y.

/*  substitute(NewTerm,OldTerm,OldExpression,NewExpression)
replaces all */
/*  occurrences of OldTerm in OldExpression with NewTerm,
producing      */
/*  NewExpression.
*/

substitute(New,Old1,Old2,New):-
     Old1 == Old2,
     !.
substitute(_,Old1,Old2,Old2):-
     var(Old2),
     \+ Old1 == Old2,
     !.
substitute(_,_,Val,Val):-
     \+(var(Val)),
     atomic(Val),
     !.
substitute(New,Old,Val,Newval):-
     Val=..[Fn|Args],
     subst_args(New,Old,Args,Newargs),
     Newval=..[Fn|Newargs].
subst_args(_,_,[],[]):-
     !.
subst_args(New,Old,[Arg|Args],[Newarg|Newargs]):-
     substitute(New,Old,Arg,Newarg),
     subst_args(New,Old,Args,Newargs).

delete_element(Element,List,NewList):-
     append(L,[Element|R],List),
     append(L,R,NewList).

delete_all(Element,List,NewList):-
     append(L,[Element|R],List),
     delete_all(Element,R,IList),
     append(L,IList,NewList).
delete_all(Element,List,List):-
     \+member(Element,List).

/*********************************************************/

/*********************** BIDDER - SET BIDS ***************/

/* bidding package, this is used to set up, rank and bid
for resources.
It comprises the pattern matcher to find things to bid for
and the record
keeper to avoid bidding for things that have already been
done.
The bids are in the form of the bidder identifier followed
by a set(list)
of patterns which are required to fire and a pattern that
can be delivered.
The delivery pattern is not used yet as there is no goal
seeking behaviour yet.
The actual propagation mechanism finds the top bid and
forms a call to the
```

```
relevant "expand" predicate with the first term as the
bidder's name and the
following terms the Types and Expressions relevant.
     The main predicate provided for the engine implementor
is
     make_bid(EngineType,PatternList,Rating,Goal,Flags).
     the set of flags Flags takes sets of values which
control the TMS.
     the mf flag takes values m and f. The values mean
     f, the system will not allow two patterns to match the
same object.
     m, the system will allow two patterns to match the
same object.
     the ap flag takes values a and p. The values mean
     a, the system ancestry checks
     p, the system doesnt ancestry check
     external means that the tms will tell other tms's that
it has the
     capability to deliver the particular goal inj the
make_bid
     The main predicate for the insertion of blackboard
entries is
     pattern_keeper(Type,Expression,ConNo,OldNew).
     The main predicate for choosing bids is
     choose_bid(BidNo,Enginetype,ConsequenceList).
     The main predicate for evaluating bids is
     execute_bid(BidNo,Enginetype,ConsequenceList,Modality,
Goal) */


/* MAIN PREDICATE */
make_bid(EngineType,Section,PatternList,NegPatternList,Rati
ng,Goals):-
     number(Rating),
     make_bid(EngineType,Section,PatternList,NegPatternList
,
               Rating,Goals,[f,a]).
make_bid(EngineType,Section,PatternList,Rating,Goals,Flags)
:-
     number(Rating),
     make_bid(EngineType,Section,PatternList,[],Rating,Goal
s,Flags).
make_bid(EngineType,Section,PatternList,Rating,Goals):-
     number(Rating),
     make_bid(EngineType,Section,PatternList,[],Rating,Goal
s,[f,a]).
```

```
make_bid(EngineType,Section,PatternList,NegPatternList,Rati
ng,Goals,Flags):-
     code_pattern_list(PatternList,CodedPatternList,SPatter
nList),
     code_pattern_list(NegPatternList,CodedNegPatternList,_
),
     code_goal_pattern_list(Goals,CodedGoalList),
     Bid = bid(BidNo,EngineType,Section,

     CodedPatternList,SPatternList,CodedNegPatternList,
               Rating,CodedGoalList,Free,Flags),
     (Bid,
     !,
     (Free = free,
     !;
     retract(Bid),
     assertz(bid(BidNo,EngineType,Section,

CodedPatternList,SPatternList,CodedNegPatternList,
               Rating,CodedGoalList,free,Flags)),
     broadcast(Goals),
     retract(bid_list(Bid_List)),
     bid_sort(Bid_List,Free_Bid_List),
     assertz(bid_list(Free_Bid_List)));
     get_num(bid,BidNo),
     Free=free,
     assertz(Bid),
     assertz(needs_pattern(BidNo,CodedPatternList)),
     set_pattern_bid_list(BidNo,CodedPatternList),
     find_bids(BidNo)),
     !.

code_pattern_list([],[],[]):-
     !.
code_pattern_list([[Type,Expression,Conds]|Rest],[Code|Code
dRest],
          [[Type,Expression]|SRest]):-
     set_pattern(Type,Expression,Conds,Code),
     code_pattern_list(Rest,CodedRest,SRest).

code_goal_pattern_list([],[]):-
     !.
code_goal_pattern_list([[Type,Expression]|Rest],[Code|Coded
Rest]):-
     set_pattern(Type,Expression,true,Code),
     code_goal_pattern_list(Rest,CodedRest).
```

```
set_functionality(_,[]):-
     !.
set_functionality(Source,[[Type,Expression]|Preconditions])
:-
     asserta(functionality(Source,[Type,Expression])),
     setall(_,

     (freshcopy((Type,Expression),(Type1,Expression1)),

     vared_pattern_type(Type1,Expression1,PatternType,_),

     clause(pattern_type(Type2,Expression2,PatternType),Con
ds2),
          cdm_name(_My_Name,My_Socket),

     send_socket(Source,message(My_Socket,result([[Type2,Ex
pression2,Conds2]])))),
          _),
     set_functionality(Source,Preconditions).

set_pattern(Type,Expression,Conds,PatternType):-
     cdm_name(My_Name,My_Socket),
     cdm_type(My_Type),
     setall(_,(functionality(Source,[Type,Expression]),
          \+(Source = null),
          \+(Source = My_Type),
          \+(Source = My_Name),
          \+(Source = My_Socket),
          send_socket(Source,message(My_Socket,
               result([[Type,Expression,Conds]])
               )))
          ,_),
     set_pattern0(Type,Expression,Conds,PatternType).
set_pattern0(Type,Expression,Conds,PatternType):-
     freshcopy((Type,Expression,Conds),
            (Type1,Expression1,Conds1)),
     numbervars((Type1,Expression1,Conds1),1,_),
     set_pattern1(Type,Expression,Conds,Type1,Expression1,C
onds1,
          PatternType).

/*
set_pattern1(Type,Expression,Conds,Type1,Expression1,Conds1
,PatternType) */
```

```
set_pattern1(_,_,_,Type1,Expression1,Conds1,PatternType):-
     vared_pattern_type(Type1,Expression1,PatternType,Conds
1),
     !.
set_pattern1(Type,Expression,Conds,Type1,Expression1,Conds1
,PatternType):-
     gensym(pattern,PatternType),
     assert(vared_pattern_type(Type1,Expression1,PatternTyp
e,Conds1)),
     assert((pattern_type(Type,Expression,PatternType):-
Conds)),
     !,
     find_patterns(PatternType).

set_pattern_bid_list(_,[]):-
     !.
set_pattern_bid_list(BidNo,[PatternType|RestCodedPatternLis
t]):-
     retract(pattern_type_bid_list(PatternType,BidList)),
     !,
     sys_unite([BidNo],BidList,NewBidList),
     assert(pattern_type_bid_list(PatternType,NewBidList)),
     !,
     set_pattern_bid_list(BidNo,RestCodedPatternList).
set_pattern_bid_list(BidNo,[PatternType|RestCodedPatternLis
t]):-
     assert_set(pattern_type_bid_list(PatternType,[BidNo]))
,
     !,
     set_pattern_bid_list(BidNo,RestCodedPatternList).

find_patterns(PatternType):-
     (setall([ConNo,Rating],
          (clause(pattern_type(Type,_,PatternType),_),
          consequence(ConNo,Type,Expression,_,Rating),
          ConNo > 0,
          pattern_type(Type,Expression,PatternType)),
          List);
     List = []),
     pattern_sort(List,SList),
     !,
     assertz(pattern_list(PatternType,SList)).

/* MAIN PREDICATE */
pattern_keeper(_,_,_,leave):-
     !.
```

```
pattern_keeper(Type,Expression,ConNo,OldNew):-
     nonvar(ConNo),
     ConNo > 0,
     pattern_type(Type,Expression,PatternType),
     insert_pattern_list(PatternType,ConNo,OldNew),
     fail.
pattern_keeper(_,_,_,_).

insert_pattern_list(PatternType,ConNo,OldNew):-
     retract(pattern_list(PatternType,List)),
     (    consequence(ConNo,Type,Expression,_,Rating),
          insert_entry(ConNo,Rating,List,NewList),
          assert(pattern_list(PatternType,NewList)),

     update_bid_list(PatternType,ConNo,Type,Expression,Rati
ng,OldNew);
          assert(pattern_list(PatternType,List))),
     !.

insert_entry(ConNo,Rating,[],[[ConNo,Rating]]):-
     !.
insert_entry(ConNo,Rating,List,NewList):-
     append(LeftList,[[ConNo,_]|RightList],List),
     append(LeftList,[[ConNo,Rating]|RightList],NewList),
     !.

insert_entry(ConNo,Rating,List,NewList):-
     pattern_sort([[ConNo,Rating]|List],NewList),
     !.

/* MAIN PREDICATE */
/* this wierd one at the front to allow crashes to occur
gracefully */
execute_bid(0,[],_,_,_):-
     !.
execute_bid(BidNo,ConsequenceList,Modality,GoalList,Rating)
:-
     bid(BidNo,EngineType,Section,_,_,_,_,_,free,_),
     form_argument_list(ConsequenceList,Args1),
/*   form_variable_argument_list(CodedGoalList,TGoalList),
*/
     append([Args1,Modality,TGoalList],[Rating,CommNonComm]
,Args),
     Bid =.. [expand,EngineType,Section|Args],
     !,
     (monitor(BidNo,Bid),
     GoalList = TGoalList;
     GoalList = [bugger,nil],
     CommNonComm = n),
     assert_bid(BidNo,ConsequenceList,CommNonComm),
     (CommNonComm=p,
     !;
     true).

form_argument_list([],[]):-
     !.
form_argument_list([ConNo|RestCons],[[Type,Expr]|RestArgs])
:-
     consequence(ConNo,Type,Expr,_,_),
     form_argument_list(RestCons,RestArgs).

form_variable_argument_list([],[]):-
     !.
form_variable_argument_list([Pattern|RestPatterns],[[Type,E
xpr]|RestGoals]):-
     clause(pattern_type(Type,Expr,Pattern),_),
     form_variable_argument_list(RestPatterns,RestGoals).


/* MAIN PREDICATE */
choose_bid(BidNo,EngineType,ConList,DerivationSet,Rating):-
     bid_list(Bids),
     choose_bid1(Bids,BidNo,EngineType,ConList,DerivationSe
t,Rating,NewBids),
     (Bids = NewBids;
     rerecord_bid_list(NewBids)),
     !.
choose_bid(BidNo,EngineType,ConList,DerivationSet,Rating):-
     nl,
     write('Choose_bid failed. bid_list reconstructed'),
     nl,
     retractall(bid_list(_)),
     retractall(bid(_,_,_,_,_,_,_,_,_,_)),
     assert(bid_list([])),
     rejigall,
     bid_list(Bids),
     choose_bid1(Bids,BidNo,EngineType,ConList,DerivationSe
t,Rating,NewBids),
     (Bids = NewBids;
     rerecord_bid_list(NewBids)),
     !.
choose_bid(0,nil,[],[],0):-
```

```
     nl,
     write('Quel Dick Head, you haven''t loaded the "human"
engine'),
     nl,
     assert(stop).

rejigall:-
     loaded(Engine),
     while(set_bid(Engine),true),
     fail.
rejigall.

choose_bid1([[TBidNo,[]]|Bids],0,nil,[],[],0,[[TBidNo,[]]|B
ids]):-
     !.
choose_bid1([[TBidNo,[[TConList,TRating]|RestCons]]|Bids],B
idNo,EngineType,ConList,DerivationSet,Rating,NewBids):-
     (bid(TBidNo,EngineType,_,_,_,CodedNegPatternList,_,_,_
,Flags),
     get_neg_list(CodedNegPatternList,NegList),
     check(TConList,NegList,DerivationSet,Flags),
     \+((justifies(TConList,EngineType,_,_,_),
          TConList=[_|_])),
     BidNo = TBidNo,
     ConList = TConList,
     Rating = TRating,
     NewBids =
[[TBidNo,[[TConList,TRating]|RestCons]]|Bids],
     !;
     bid_resort([[TBidNo,RestCons]|Bids],TempBids),
     choose_bid1(TempBids,BidNo,EngineType,ConList,Derivati
onSet,Rating,NewBids)).

get_neg_list([],[]):-
     !.
get_neg_list([CodedNegPattern|CodedNegPatternList],[NegCons
|NegList]):-
     pattern_list(CodedNegPattern,NegCons),
     !,
     get_neg_list(CodedNegPatternList,NegList).
get_neg_list([_|CodedNegPatternList],NegList):-
     get_neg_list(CodedNegPatternList,NegList).

assert_set(P):-
     P,
     !.
```

```
assert_set(P):-
     assert(P).

conjoin_ratings(R1,R2,CR):-
     R1 > R2,
     CR = R2,
     !;
     CR = R1.
disjoin_ratings(R1,R2,CR):-
     R1 < R2,
     CR = R2,
     !;
     CR = R1.
/* assert_bid(BidNo,ConList,CommNonComm)
     asserts that a bid has been completed with the
consequence list
     ConList.
     If CommNonComm = c then all permutations of the
consequence list
     are deemed to have been done.
     If CommNonComm = n then only the actual consequence
     list is asserted
     If CommNonComm = cs then this is equivalent to a
partial cut (s for
     suspended) then c
     If CommNonComm = ns then this is equivalent to a
partial cut (s for
     suspended) then n
     If CommNonComm = a then this is equivalent to a cut.
*/

assert_bid(BidNo,_,a):-
     !,
     abolish_bid(BidNo),
     !.
assert_bid(BidNo,ConList,cs):-
     !,
     retract(bid(BidNo,EngineType,Section,

CodedPatternList,PatternList,Rating,CodedGoalList,_,Flags))
,
     assert(bid(BidNo,EngineType,Section,

CodedPatternList,PatternList,Rating,CodedGoalList,suspended
,Flags)),
     assert_bid(BidNo,ConList,c),
     !.
assert_bid(BidNo,ConList,ns):-
     !,
     retract(bid(BidNo,EngineType,Section,

CodedPatternList,PatternList,CodedNegPatternList,
               Rating,CodedGoalList,_,Flags)),
     assert(bid(BidNo,EngineType,Section,

CodedPatternList,PatternList,CodedNegPatternList,
               Rating,CodedGoalList,suspended,Flags)),
     assert_bid(BidNo,ConList,n),
     !.
assert_bid(_,_,p):-
     !.
assert_bid(BidNo,ConList,NC):-
     bid_list(Bids),
     (append(First,[[BidNo,Bid_List]|Second],Bids),
          bid_delete([ConList,_],Bid_List,NewBid_List,NC),
          (NewBid_List = [],
               append(First,Second,NewBids)
          ;

     append(First,[[BidNo,NewBid_List]|Second],TNewBids),
          bid_sort(TNewBids,NewBids)),
          rerecord_bid_list(NewBids)
     ;
     true),
     !.

is_a_permute([],[]).
is_a_permute([Element|List],PList):-
     append(Left,[Element|Right],PList),
     append(Left,Right,PList1),
     !,
     is_a_permute(List,PList1).

bid_delete(_,_,[],a):-
     !.
bid_delete(_,[],[],_):-
     !.
bid_delete([Bid,_],[[Bid,_]|Bids],Bids,n):-
     !.
bid_delete([Bid,R],[[PBid,_]|Bids],NewBids,c):-
     is_a_permute(Bid,PBid),
     bid_delete([Bid,R],Bids,NewBids,c),
     !.
bid_delete(Bid, [NonBid|Bids], [NonBid|NewBids],NC):-
     bid_delete(Bid,Bids,NewBids,NC),
     !.

abolish_bid(EngineType,Section):-
     bid(BidNo,EngineType,Section,_,_,_,_,_,_,_),
     abolish_bid(BidNo).
abolish_bid(_,_).

abolish_bid(BidNo):-
     retract(bid(BidNo,_,_,
                CodedPatternList,_,_,_,CodedGoalList,_,_)),
     (retract(needs_pattern(BidNo,_));true),
     append(CodedPatternList,CodedGoalList,PatternList),
     remove_pattern_type_bid_list(PatternList,BidNo),
     bid_list(Bids),
     (append(First, [[BidNo,_]|Second],Bids),
     append(First,Second,NewBids),
     rerecord_bid_list(NewBids);
     true),
     !.
abolish_bid(_).

remove_pattern_type_bid_list([],_):-
     !.
remove_pattern_type_bid_list([PatternType|List],BidNo):-
     pattern_type_bid_list(PatternType,BidList),
        !,
     retract(pattern_type_bid_list(PatternType,BidList)),
        sys_difference(BidList,[BidNo],NewBidList),

assert(pattern_type_bid_list(PatternType,NewBidList)),
     remove_pattern_type_bid_list(List,BidNo).
remove_pattern_type_bid_list([_|List],BidNo):-
     remove_pattern_type_bid_list(List,BidNo).


/*
update_bid_list(Pattern_Type,ConNo,Type,Expression,Rating)
     This updates the whole set of bids in "bid_list" and
ranks them.
     In a highly specific system this should be a small
task but it is
     potentially large.
     The Pattern_Type will be associated with a collection
of bids which are
     set up ab initio.
     The ConNo is the actual consequence or entry that has
just been entered
     and is to give rise to some new bids.
     rating is the credibility  of the new entry. */

update_bid_list(Pattern_Type,ConNo,Type,Expression,Rating,O
ldNew):-
     (pattern_type_bid_list(Pattern_Type,Bids),
     update_bids(Bids,Pattern_Type,ConNo,Type,Expression,Ra
ting,OldNew);
     true),
     !.

update_bids([],_,_,_,_,_,_):-
     !.
update_bids([Bid|Rest],Pattern_Type,ConNo,Type,Expression,R
ating,OldNew):-
     update_bid(Bid,Pattern_Type,ConNo,Type,Expression,Rati
ng,OldNew),
     update_bids(Rest,Pattern_Type,ConNo,Type,Expression,Ra
ting,OldNew).

update_bid(Bid,Pattern_Type,ConNo,Type,Expression,Rating,ne
w):-
     update_needs_pattern(Bid,Pattern_Type),
     !,
     bid(Bid,_,_,Coded_Pattern_List,Pattern_List,_,_,_,_,Fl
ags),
     setall([Con_List,CRating],

     (find_pattern_types(Pattern_Type,[Type,Expression],
               Coded_Pattern_List,Pattern_List,
               Left_Coded,Left_Pattern,
               Right_Coded,RightPattern),
          consbid(Left_Coded,Left_Pattern,ConNo,Rating,

     Right_Coded,RightPattern,Con_List,CRating,Flags)),
          Con_Lists),
     merge_bid_list(Bid,Con_Lists).

update_bid(Bid,_,ConNo,_,_,Rating,old):-
     needs_pattern(Bid,[]),
     !,
     update_merge_bid_list(Bid,ConNo,Rating).

update_bid(_,_,_,_,_,_,_).


/*
find_pattern_types(Pattern_Type,Pattern,Coded_Pattern_List,
Pattern_List,
     Left_Coded,Left_Pattern,Right_Coded,RightPattern).
finds pattern types in lists of them,
a bit like append and member but goes down the lists in
unison */

find_pattern_types(Pattern_Type,Pattern,[Pattern_Type],[Pat
tern],[],[],[],[]).

find_pattern_types(Pattern_Type,Pattern,[Pattern_Type|Coded
_Pattern_List],
     [Pattern|Pattern_List],
     [],[],Coded_Pattern_List,Pattern_List).

find_pattern_types(Pattern_Type,Pattern,[Pattern_Type1|Code
d_Pattern_List],
          [Pattern1|Pattern_List],

     [Pattern_Type1|Left_Coded],[Pattern1|Left_Pattern],
          Right_Coded,Right_Pattern):-
     find_pattern_types(Pattern_Type,Pattern,Coded_Pattern_
List,
               Pattern_List,
               Left_Coded,Left_Pattern,
               Right_Coded,Right_Pattern).

update_needs_pattern(Bid,_):-
     needs_pattern(Bid,[]),
     !.

update_needs_pattern(Bid,Pattern_Type):-
     needs_pattern(Bid,Pattern_List),
     append(Left,[Pattern_Type|Right],Pattern_List),
     append(Left,Right,New_Pattern_List),
     retract(needs_pattern(Bid,Pattern_List)),
     assertz(needs_pattern(Bid,New_Pattern_List)),
     !,
     needs_pattern(Bid,[]).
```

```
consbid(Left_Coded,Left_Pattern,ConNo,Rating,Right_Coded,Ri
ght_Pattern,Con_List,CRating,Flags):-
     conslrbid(Left_Coded,Left_Pattern,Left_ConList, [ConNo]
,LRating,Flags),
     conslrbid(Right_Coded,Right_Pattern,Right_ConList, [Con
No|Left_ConList],RRating,Flags),
     conjoin_ratings(LRating,RRating,LRRating),
     append(Left_ConList, [ConNo|Right_ConList],Con_List),
     conjoin_ratings(LRRating,Rating,CRating).

conslrbid([],[],[],_,1000,_):-
     !.
conslrbid([Pattern_Type|RestPattern_Types],[[Type,Expressio
n]|RestPatterns],
          [ConNo|RestCons],BarredCons,Rating,Flags):-
     pattern_list(Pattern_Type,PList),
     !,
     member([ConNo,CRating],PList),
     \+((\+member(m,Flags),
          member(ConNo,BarredCons))),
     consequence(ConNo,Type,Expression,DerivSet,_),
     conslrbid1(DerivSet,RestPattern_Types,RestPatterns,Res
tCons,[ConNo|BarredCons],RRating,Flags),
     conjoin_ratings(CRating,RRating,Rating).

conslrbid1(_,[],[],[],_,1000,_):-
     !.
conslrbid1(DerivSet1,[Pattern_Type|RestPattern_Types],
          [[Type,Expression]|RestPatterns],
          [ConNo|RestCons],BarredCons,Rating,Flags):-
     \+member(m,Flags),
     pattern_list(Pattern_Type,PList),
     !,
     member([ConNo,CRating],PList),
     \+member(ConNo,BarredCons),
     consequence(ConNo,Type,Expression,DerivSet,_),
     form_derivation_set(DerivSet1,DerivSet,NewDerivSet),
     check_nogoods(NewDerivSet,_,GoodSet),
     \+((GoodSet = [])),
     conslrbid1(GoodSet,RestPattern_Types,RestPatterns,Rest
Cons,[ConNo|BarredCons],RRating,Flags),
     conjoin_ratings(CRating,RRating,Rating).
conslrbid1(DerivSet1,[Pattern_Type|RestPattern_Types],
          [[Type,Expression]|RestPatterns],
          [ConNo|RestCons],BarredCons,Rating,Flags):-
     member(m,Flags),
     pattern_list(Pattern_Type,PList),
     !,
     member([ConNo,CRating],PList),
     (member(ConNo,BarredCons),
     conslrbid1(DerivSet1,RestPattern_Types,RestPatterns,
          RestCons,BarredCons,Rating,Flags);
     \+member(ConNo,BarredCons),
     consequence(ConNo,Type,Expression,DerivSet,_),
     form_derivation_set(DerivSet1,DerivSet,NewDerivSet),
     check_nogoods(NewDerivSet,_,GoodSet),
     \+((GoodSet = [])),
     conslrbid1(GoodSet,RestPattern_Types,RestPatterns,Rest
Cons,[ConNo|BarredCons],RRating,Flags),
     conjoin_ratings(CRating,RRating,Rating)).

consallbid([],[],[],Level,_):-
     !,
     cut_off_level(Level).
consallbid(CPatterns,Patterns,Con_List,Rating,Flags):-
     conslrbid(CPatterns,Patterns,Con_List,[],Rating,Flags)
.

update_merge_bid_list(BidNo,ConNo,Rating):-
     bid_list(Bids),
     (append(First,[[BidNo,OldList]|Second],Bids),
     update_merge_bid_list1(ConNo,Rating,OldList,TNewList),
     pattern_sort(TNewList,NewList),
     append(First,[[BidNo,NewList]|Second],TNewBids),
     bid_sort(TNewBids,NewBids),
     rerecord_bid_list(NewBids);
     true).

update_merge_bid_list1(_,_,[],[]):-
     !.
update_merge_bid_list1(ConNo,Rating,[[ConList,R1]|Con_List1
],[[ConList,R2]|Con_List2]):-
     member(ConNo,ConList),
     conjoin_ratings(Rating,R1,R2),
     !,
     update_merge_bid_list1(ConNo,Rating,Con_List1,Con_List
2).
update_merge_bid_list1(ConNo,Rating,[[ConList,R1]|Con_List1
],[[ConList,R1]|Con_List2]):-
     !,
     update_merge_bid_list1(ConNo,Rating,Con_List1,Con_List
2).
```

```
merge_bid_list(BidNo,Con_Lists):-
     bid_list(Bids),
     (append(First,[[BidNo,OldList]|Second],Bids),
     merge_bid_list1(Con_Lists,OldList,TNewList),
     pattern_sort(TNewList,NewList),
     append(First,[[BidNo,NewList]|Second],TNewBids);
     pattern_sort(Con_Lists,NewList),
     TNewBids=[[BidNo,NewList]|Bids]),
     bid_sort(TNewBids,NewBids),
     rerecord_bid_list(NewBids).

merge_bid_list1(Con_List,[],Con_List):-
     !.
merge_bid_list1([],Con_List,Con_List):-
     !.
merge_bid_list1([[Con1,R1]|Con_List1],Con_List2,
          [[Con1,R1]|Con_List]):-
     append(LCon_List2,[[Con1,_]|RCon_List2],Con_List2),
     !,
     append(LCon_List2,RCon_List2,NCon_List2),
     merge_bid_list1(Con_List1,NCon_List2,Con_List).
merge_bid_list1([[Con1,R1]|Con_List1],[[Con2,R2]|Con_List2]
,
          [[Con2,R2]|Con_List]):-
     R2 > R1,
     !,
     merge_bid_list1([[Con1,R1]|Con_List1],Con_List2,Con_Li
st).
merge_bid_list1([[Con1,R1]|Con_List1],[[Con2,R2]|Con_List2]
,
          [[Con1,R1]|Con_List]):-
     merge_bid_list1(Con_List1,[[Con2,R2]|Con_List2],Con_Li
st).

find_bids(BidNo):-
     bid(BidNo,_,_,Coded_Pattern_List,PatternList,_,_,_,_,F
lags),
     (check_needs(BidNo),
     setall([Con_List,CRating],
          (consallbid(Coded_Pattern_List,PatternList,
               Con_List,CRating,Flags)),
          Con_Lists),
     pattern_sort(Con_Lists,SCon_Lists);
     SCon_Lists=[]),
      (bid_list(Bids),
      !;
      nl,
      write('Crashed-bid_list reconstructed'),
      nl,
      assert(bid_list([])),
      Bids = []),
      (SCon_Lists = [],
      !
      ;
            (append(First,[[BidNo,_]|Second],Bids),

      append(First,[[BidNo,SCon_Lists]|Second],TNewBids)
            ;
            append(Bids,[[BidNo,SCon_Lists]],TNewBids)),
            bid_sort(TNewBids,NewBids),
            (Bids=NewBids;
            rerecord_bid_list(NewBids)
            )
      ),
      !.

check_needs(BidNo):-
      needs_pattern(BidNo,[]),
      !.
check_needs(BidNo):-
      needs_pattern(BidNo,List),
      !,
      check_needs1(List,NewList),
      (List = NewList;
      retract(needs_pattern(BidNo,List)),
      assertz(needs_pattern(BidNo,NewList))),
      !,
      NewList = [].

check_needs1([],[]):-
      !.
check_needs1([Pattern1|RestPatterns],[Pattern1|NewRestPatte
rns]):-
      pattern_list(Pattern1,[]),
      !,
      check_needs1(RestPatterns,NewRestPatterns).
check_needs1([Pattern1|RestPatterns],NewRestPatterns):-
      pattern_list(Pattern1,[_|_]),
      !,
      check_needs1(RestPatterns,NewRestPatterns).
```

```
/**********************************************************/


/************************** PROPAGATE *********************/
/*       Propagation of new assumption or consequence     */
/**********************************************************/

/*  propagate(N) 'unifies' consequence N with each of the
preceding      */
/*  consequences.tms_assume(eds,eds,Entry,Base,100sequences
resulting from these unifications*/
/*  are asserted into the consequence database and
similarly propagated. */

propagate:-
     retractall(stop),
     bidsys(Modality,GoalList,Engine,ConList,DerivationSet)
,
     process_results(Modality,GoalList,Engine,ConList,Deriv
ationSet),
     stop;
     propagate.

process_results(Modality,GoalList,Engine,ConList,Derivation
Set):-
     ((Modality=necessary,
     !,
     process_consequences(no_redo,GoalList,Engine,Modality,
ConList,DerivationSet));
     (Modality=mgu,
     !,
     process_consequences(no_redo,GoalList,Engine,Modality,
ConList,DerivationSet));
     (Modality=assume(Rating),
     !,
     process_possibilities(GoalList,Engine,Modality,ConList
,DerivationSet));
     (Modality=possible(Rating),
     !,
     process_possibilities(GoalList,Engine,Modality,ConList
,DerivationSet))),
     !.

process_consequences(_,[],_,_,_,_):-
     !.
```

```
process_consequences(Redo,[[Type,Goal]|RestGoals],Engine,Mo
dality,ConList,DerivationSet):-
     process_consequence(Redo,[Type,Goal],Engine,Modality,C
onList,DerivationSet),
     process_consequences(Redo,RestGoals,Engine,Modality,Co
nList,DerivationSet),
     !.

process_consequence(Redo,[Type,Goal],Engine,Modality,ConLis
t,DerivationSet):-
     Goal = nil,
     !;
     insert_consequence(ConNo,Type,Goal,ConList,Engine,Moda
lity,DerivationSet,OldNew),
     redo_rating(Redo,OldNew,OldNew1),
     pattern_keeper(Type,Goal,ConNo,OldNew1),
     !.

redo_rating(no_redo,OldNew,OldNew):-
     !.
redo_rating(redo,leave,old):-
     !.
redo_rating(redo,OldNew,OldNew):-
     !.
process_possibilities([[Type,Goal]],Engine,Modality,ConList
,DerivationSet):-
     Goal = nil,
     !;
     insert_consequence(ConNo,Type,Goal,[ConNo|ConList],Eng
ine,Modality,DerivationSet,OldNew),
     pattern_keeper(Type,Goal,ConNo,OldNew),
     !.
process_possibilities(GoalList,Engine,Modality,ConList,Deri
vationSet):-
     check_link(GoalList,Link),
     write('found a duplicate set'),
     nl,
     !,
     insert_consequence(ConNo,link,Link,[ConNo|ConList],Eng
ine,Modality,DerivationSet,_),
     !.
process_possibilities(GoalList,Engine,Modality,ConList,Deri
vationSet):-
     gensym(possible_link,Link),
     insert_consequence(ConNo,link,Link,[ConNo|ConList],Eng
ine,Modality,DerivationSet,_),
     consequence(ConNo,link,Link,Poss_Derivation,_),
     process_consequences(no_redo,GoalList,Engine,mgu,[ConN
o],Poss_Derivation),
     !.

bidsys(Modality,Goal,EngineType,ConList,DerivationSet):-
     choose_bid(BidNo,EngineType,ConList,DerivationSet,Rati
ng),
     execute_bid(BidNo,ConList,Modality,Goal,Rating).

/*********************************************************/


/************************ RATING SYS *********************/
/*                  Rating of Consequences               */
/*********************************************************/

/* set_rating(ConNo,FDerivationSet,Modality) */

set_rating(_,_,necessary):-
     !.
set_rating(_,_,mgu):-
     !.
set_rating(ConNo,_,possible(RatingValue)):-
     (retract(rating(ConNo,OldValue)),
     simplify(fuzzy,OldValue or RatingValue,NewValue);
     NewValue is RatingValue),
     assert(rating(ConNo,NewValue)),
     !.

get_rating([DerivationSetBits|Derivations],Rating):-
     bits_to_list(DerivationSetBits,DerivationSet),
     work_out_rating(DerivationSet,Rating1),
     get_rating1(Derivations,Rating1,Rating).

get_rating1([],Rating,Rating):-
     !.
get_rating1([DerivationSetBits|Derivations],RatingSoFar,Rat
ing):-
     bits_to_list(DerivationSetBits,DerivationSet),
     work_out_rating(DerivationSet,Rating1),
     simplify(fuzzy,Rating1 or RatingSoFar,NewRating),
     get_rating1(Derivations,NewRating,Rating).


work_out_rating(DerivationSet,SetRating):-
```

```
     work_out_rating1(DerivationSet,RatingSoFar),
     !,
     work_out_rating2(DerivationSet,RatingSoFar,SetRating),
     !.

work_out_rating1([Element|Rest],Rating):-
     (rating(Element,Rating1),
     !,
     work_out_rating11(Rest,Rating1,Rating);
     work_out_rating1(Rest,Rating)).

work_out_rating11([],Rating,Rating):-
     !.
work_out_rating11([Element|Rest],RatingSoFar,Rating):-
     rating(Element,Rating1),
     simplify(fuzzy,Rating1 and RatingSoFar,NewRating),
     work_out_rating11(Rest,NewRating,Rating).

work_out_rating2(DerivationSet,RatingSoFar,SetRating):-
     setall(Rating,(rating_set(Element,Rating),bit_sys_subs
et(Element,DerivationSet)),Ratings),
     update_rating_thingy(Ratings,RatingSoFar,SetRating).

update_rating_thingy([],Rating,Rating).
update_rating_thingy([Rating|Ratings],OldRating,NewRating):
-
     simplify(fuzzy,Rating and OldRating,RatingSoFar),
     update_rating_thingy(Ratings,RatingSoFar,NewRating).

/* re_rate_entry(Delay,ConNo,Modality,Rate) re calculates
the rating for an entry if it needs to be updated, only
really necessary for necessary(R), possible(R), and
critical(R),
Delays rewriting the consequence if Delay is set to delay,
otherwise reasserts */

re_rate_entry(_,ConNo,necessary,Rate):-
     !,
     consequence(ConNo,_,_,_,Rate).
re_rate_entry(_,ConNo,mgu,Rate):-
     !,
     consequence(ConNo,_,_,_,Rate).
re_rate_entry(Delay,ConNo,possible(Rating),NewRating):-
     !,
     consequence(ConNo,Type,Consequence,AssBase,OldRating),
     set_rating(ConNo,AssBase,possible(Rating)),
      get_rating(AssBase,NewRating),
      (Delay=delay;
      retract(consequence(ConNo,Type,Consequence,AssBase,Old
Rating)),
      assert(consequence(ConNo,Type,Consequence,AssBase,NewR
ating))),
      !.

/*********************************************************/
```

Lexical Analysis Entries (22 to 35) continued from Chapter 6, Section 6.4, Subsection 6.4.2

22. english, english(noun_phrase([the, man]), [the, man, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var, lex_var], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [noun], [lexical(noun, [man])])]), heads),[[1, 2, 3, 19, 21]],100

23. english, english(noun_phrase([a, large, nose, and, squinted, eyes]), [lex_var, lex_var, lex_var, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])]), heads),[[1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 22]],100

24. english, english(verb_phrase([has, a, large, nose, and, squinted, eyes]), [lex_var, lex_var, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase],

[lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])]), heads),[[1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 22, 23]],100

25. english, english(sentence([man, has, a, large, nose, and, squinted, eyes]), [lex_var, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [man])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])]), heads),[[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 22, 23, 24]],100

26. english, english(noun_phrase([large, nose, and, squinted, eyes]), [lex_var, lex_var, lex_var, lex_var, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])]), heads),[[1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 25]],100

27. english, english(noun_phrase([a, large, nose, and, squinted, eyes]), [lex_var, lex_var, lex_var, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])]), heads),[[1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 25, 26]],100

28. english, english(verb_phrase([has, a, large, nose, and, squinted, eyes]), [lex_var, lex_var, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])]), heads),[[1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 25, 26, 27]],100

29. english, english(sentence([man, has, a, large, nose, and, squinted, eyes]), [lex_var, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [man])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase],

[lexical(indefinite_article, [a]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])])]), heads),[[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 25, 26, 27, 28]],100

30. english, english(noun_phrase([nose, and, squinted, eyes]), [lex_var, lex_var, lex_var, lex_var, lex_var, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [nose])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])]), heads),[[1, 7, 8, 9, 10, 11, 13, 14, 15, 29]],100

31. english, english(noun_phrase([large, nose, and, squinted, eyes]), [lex_var, lex_var, lex_var, lex_var, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [nose])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])]), heads),[[1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 29, 30]],100

32,english, english(verb_phrase([has, a, large, nose, and, squinted, eyes]), [lex_var, lex_var, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [nose])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])])]), heads),[[1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 29, 30, 31, 32]],100

33, english, english(sentence([man, has, a, large, nose, and, squinted, eyes]), [lex_var, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [man])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [nose])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])])])]), heads),[[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 29, 30, 31, 32, 33]],100

34, english, english(sentence([the, man, has, a, large, nose, and, squinted, eyes]), [the, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [noun], [lexical(noun, [nose])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])]), heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 21, 29, 30, 31, 32, 34]],100

35, english, english(sentence([the, man, has, a, large, nose, and, squinted, eyes]), [the, man, has, a, large, nose, and, squinted, eyes], [the, man, has, a, large, nose, and, squinted, eyes], grammar(sentence, [noun_phrase, verb_phrase], [grammar(noun_phrase, [definite_article, noun_phrase], [lexical(definite_article, [the]), grammar(noun_phrase, [noun], [lexical(noun, [man])])]), grammar(verb_phrase, [transitive_verb, noun_phrase], [lexical(transitive_verb, [has]), grammar(noun_phrase, [indefinite_article, noun_phrase], [lexical(indefinite_article, [a]), grammar(noun_phrase, [noun_phrase, conjunction, noun_phrase], [grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [large])]), grammar(noun_phrase, [noun], [lexical(noun, [nose])])]), lexical(conjunction, [and]), grammar(noun_phrase, [adjective_phrase, noun_phrase], [grammar(adjective_phrase, [adjective], [lexical(adjective, [squinted])]), grammar(noun_phrase, [noun], [lexical(noun, [eyes])])])])])])]), heads),[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 21, 25, 26, 27, 35]],100

This Page has been left blank Intentionally

Bibliography

3D Studio Max Reference Volume I and II, 1999. Discreet, AutoDesk Systems.

Ahmad, S. and Hinde, C. J. (2001). Natural Language based Facial Image Generation System. *Proceedings of Nordic Interactive Conference in association with ACM SIGGRAPH.* Copenhagen, Oct 31st – Nov 1st.

Anjyo, K., Usami, Y. and Kurihara, T. (1992). A simple method for extracting the natural beauty of hair. *Computer Graphic*s, 26(2):111-120, July.

Aspley Limited (1993). *E-fit*, Hartfield, UK: Aspley Limited.

Baker, E. and Seltzer, M. (1997). The Mug-Shot Search Problem. *Harvard University Center for Research in Computing Technology, Technical Report-20-97.*

Baldwin, J. F. (1991) A theory of Mass Assignments for Artificial Intelligence. IJCAI '91 Workshops on Fuzzy Logic and Fuzzy Control, Sydney, Australia, Lecture Notes in Artificial Intelligence, pp. 22-34.

Baldwin, J. F. (1992) Fuzzy and Probabilistic Uncertainties. Encyclopedia of AI, 2nd ed., Shapiro, pp 528-537.

Barr, A. H. (1984). Global and Local Deformations of Solid Prim-itives, *Computer Graphics* 17,3, July, pp 21-30.

Batten, G.W and Rhodes, B.T. (1978). UHMFS: The University of Houston mug file system. Proceedings of the 1978 Carnahan Conference on Crime Countermeasures. Lexington, Kentucky.

Bell, J.A. (1998). “3D Studio Max R2.5 f/x and design”, The Coriolis Group, pp 382–397.

Bezier, P. (1974). Mathematical and practical possibilities ofUNISURF, in *Computer Aided Geometric Design, R. E.* Barnhill and R. F. Riesenfeld, eds., Academic Press, New York, pp 127-152.

Bindiganavale, R., Schuler, W., Allbeck, J. M., Badler, N. I., Joshi, A. K. & Palmer, M. (2000). Dynamically Altering Agent Behaviors Using Natural Language Instructions, *Proceedings of the fourth international conference on Autonomous agents* , June 3 - 7, Barcelona Spain.

Bisson, C.L. (1965a). Measurements by computer of the distances on and about the eyes (Report No. PRI-18). Panoramic Research, Inc., Palo Alto, California.

Bisson, C.L. (1965b). Location of some facial features by computer. (Report No. PRI-20). Panoramic Research, Inc., Palo Alto, CaliforniaBledsoe, W. W. (1966). *The model method in facial recognition.* Panoramic Research Inc., Palo Alto, CA, PRI:15, Aug.

Boardman, T. and Hubbell, J. (1998). “Inside 3D Studio Max 2”, New Riders Publishing, pp 9–22.

Bledsoe, W.W. (1964). The model method in facial recognition. (Report No. PRI-15). Panoramic Research, Inc., Palo Alto, California

Bratko I. 1986: Prolog Programming for Artificial Intelligence. Addison-Wesley

Brewer, J. A. and Anderson, D. C. (1977). Visual interaction with Overhauser curves and surfaces. *Computer Graphics* 11, 2, July, pp. 132-137.

Bromley, L.K. (1977). Computer-aided processing techniques for usage in real-time image evaluation. Master's thesis, University of Houston

Capin, T. K., Pandzic , I., Thalmann , N. M. , Thalmann, D. (1998). Integration of Avatars and Autonomous Virtual Humans in Networked Virtual Environments.
Computer Graphics Laboratory, Swiss Federal Institute of Technology. MIRALAB-CU. University of Geneva

Cannon, S. R., Jones, G. W., Campbell, R., & Morgan, N. W. (1986). A computer vision system for identification of individuals. *Proceedings of IECON* (pp. 347-351).

Carey, S. & Diamond, R. (1977). From piecemeal to configurational representation of faces. *Science, 195,* 312-313.

Chernoff, H. (1971). *The use of faces to represent points in N-dimensional space graphically.* Office of Naval Research, December, Project NR-042-993.

Chernoff, H. (1973). The use of faces to represent points in K-dimensional space graphically, *Journal of American Statistical Association,* 361.

Cobb, E. S. (1984). *Design of Sculptured Surfaces using the B-spline Representation.* Ph.D. Dissertation, Department of Computer Science, University of Utah, June.

Connectix, (2001). http://www.connectix.com/products/vpc4m.html

Corlet R.A. & Todd S. J. 1986: A Monte Carlo Approach to Uncertain Inference. In: Artificial Intelligence and its Applications. Ed. Cohn & Thomas. John Wiley & Sons. Pp. 127 -137.

Craw, I., Ellis, H., & Lishman, J. R. (1987). Automatic extraction of face features. *Pattern Recognition Letters, 5,* 183-187.

Chomsky, Noam (1986a). *Barriers. Linguistic Inquiry Monograph 13.* Cambridge, MA: MIT Press.

Chomsky, Noam (1986b). *Knowledge of Language: its nature, origin and use.* Newyork: Praeger.

Chomsky, Noam (1992). *A minimalist program for linguistic theory.* Cambridge, MA: MIT Working Papers in Linguistic, pp. 3.

Christie, D. F. M., & Ellis, H. D. (1981). Photofit construction versus verbal descriptions of faces. *Journal of Applied Psychology,* 66, pp 358-363.

Crystal, David (1992). *An Encyclopedic Dictionary of Language and Languages.* Oxford, England: Blackwell.

Cutler, B. L., Stocklein, C. J., & Penrod, S. D. (1988). Emperical Examination of a Computerized Facial Composite Production System, Forensic Reports, 1:207-218

Cyberware Laboratory Inc (1990). *4020/RGB 3D scanner with color digitizer.* Monterey, CA.

DataFace (1990), Facial Animation Work Shop, Report to NIC.

Davies, G. M. (1981). Face recall systems. In G. M., H. D. Ellis, & J. W. Shepherd (Eds.), *Perceiving and remembering faces*. London: Academic Press

de Kleer J. 1985: Choices without Backtracking. Proc. Of Conference of the American Association for Artificial Intelligence, Austin, Texas. August 1986. pp. 79-85.

de Kleer J. 1986(a): An Assumption-Based TMS. Artificial Intelligence Vol. 28, pp. 127 – 162.

de Kleer J. 1986(b): Extending the ATMS. Artificial Intelligence Vol. 28, pp. 163 – 196.

de Kleer J. 1986(c): Problem Solving with the ATMS. Artificial Intelligence Vol. 28, pp. 197 – 224.

de Kleer J. & Williams B. C. 1986: Back to Backtracking: Controlling the ATMS. Proc. Of 5th National Conference on A.I., AAAI-86, August 1986.

Delinguette, H., Subsol, G., Cotin, S., and Pignon, J. (1994). A craniofacial surgery simulation testbed. Technical Report 2199, INRIA, France, February.

Dodwell, P.C. (1971). "Perceptual Processing: Stimulus Equivalence and Pattern Recognition". Appleton-Century-Crofts, New York

Dougherty, R. C. (1994). *Natural Language Computing*. New York University, Lawrence Erlbaum Associates

Doyle J. 1979(a): A Glimpse of Truth Maintenance. IJCAI-79, Vol. 1, pp. 232-237.

Doyle J. 1979(b): A Truth Maintenance System. Artificial Intelligence Vol. 12, pp. 231-272.

Earley, J. (1970). *An Efficient Context-Free Parsing algorithm.* Communications of the Association of Computing Machinery.

Ekman, P. & Oster, H. (1979). Facial expressions of emotion. *Annual Review of Psychology, 20,* 527-554.

Ekman, P. & Friesen, W. V. (1978). *Facial action coding system: A technique for the measurement of facial movement.* Palo Alto, Calif.: Consulting Psychologists Press.

Ekman, P. & Friesen, W. V. (1986). A new pan cultural facial expression of emotion. *Motivation and Emotion, 10(2),* 1986.

Elson, M. (1990). “Displacement" facial animation techniques. In *Vol 26: State of the Art in Facial Animation*, pages 21-42. ACM Siggraph'90 Course Notes, Dallas Convention Center, August 6th-10th.

Ellis, H. D., Deregowski, J. B., & Shepherd, J. W. (1975). Descriptions of white and black faces by white and black subjects. *International Journal of Psychology,* 10, pp. 119-123.

Ellis, H. D., Deregowski, J. B., & Shepherd, J. W. (1980). The deterioration of verbal descriptions of faces over different delay intervals. *Journal of Police Science and Administration,* 8, pp. 101-106.

Facial Animation, (1997). http://www.cis.ohio-state.edu/~sking/FacialAnimation.html

Faigin, G. (1990), "The Artist's Complete Guide to Facial Expression".

Foley, Van Dam, Feiner, Hughes (1996), Computer Graphics Principles and Practice. 2nd Edition, Addison-Wesley Publication.

Forsey, D.R. and Bartels, R.H. (1990). Hierarchical bspline refinement. *Computer Graphics*, pp 205-212, May.

Fox, J. (1986). Knowledge, Decision Making, and Uncertainty. *Artificial Intelligence and Statistics* (Ed. Gale). Addison-Wesley, pp. 57-76.

Fox, J. (1987). Dealing With Uncertainty. *Intelligent Knowledge Based Systems – An Introduction*, (Ed. O'Shea, Self & Thomas). Harper & Row, pp. 52-67

Franksen O. I. 1978: On Fuzzy Sets, Sunjective Measurements and Utilty. Workshop on Fuzzy Reasoning: Theory and Applications. Queen Mary College, Univ. of London. 15th September, 1978.

Fried, L. A. (1976). *Anatomy of the head, neck, face, and jaws.* Philadelphia: Lea and Febiger.

Friedman J.H. & Stuetzle, W. (1981). Projection pursuit regressio. *Journal of the American Statistics Association, 76(376)*, 817-823.

Friedman, S. M. (1970). *Visual anatomy: Volume one, head and neck.* New York: Harper and Row.

Frost, R.A. (1986). Introduction to Knowledge Based Sysstems. Collins

Gaines, B. R. 1976: Foundations of Fuzzy Reasoning. Int. Journal of Man-Machine Studies, No. 8, pp. 623-668.

Garnham, A. 1987: Artificial Intelligence – An Introduction. Routledge & Kegan Paul

Genesereth, M.R. and Nilsson, N.J. (1987) Logical Foundations of Artificial Intelligence, Morgan Kaufmann Publishers Inc, Palo Alto USA.

Gillenson, M.L. (1974). *The interactive generation of facial images on a CRT using a heuristic strategy.* Ohio State University, Computer Graphics Research Group, The Ohio State University, Research Center, 1314 Kinnear Road, Columbus, Ohio 434210

Goldstein, A. J., Harmon, L. D., & Lesk, A. B. (1971). Identification of human faces, *Proceedings of IEEE, 59,* 748.

Golomb, B.A., Lawrence, D.T., & Sejnowski, T.J. (1991). SEXNET: A neural network identifies sex from human faces. In D.S. Touretzky & R. Lippman (Eds.), *Advances in Neural Information Processing Systems, 3,* San Mateo, CA: Morgan Kaufmann.

Hill, D.R., Pearce, A., & Wyvill, B. (1988). Animating speech: An automated aproach using speech synthesis by rules. *The Visual Computer, 3,* 277-289.

Hinde, C. J. 1985: Artificial Intelligence and Expert Systems. Further Developments in Operational Research. Ed. Rand & Eglese. Pergamon Press.

Hinde, C. J. 1986: Fuzzy Prolog. Int. Journal of Man-Machine Studies Vol. 24, pp. 569-595.

Hinde, C.J., Bray A.D., Herbert P.J., Launders V.A. & Round D. (1989). A Truth Maintenance Approach to Process Planning. *Artificial Intelligence* Vol. 29, No. 2. pp. 217-222.

Hinde, C.J., Lawson, R.J. and Connolly, J.H. (1989). Natural Language: The Ultimate User-Firendly Interface. San Francisco, Interex H.P. users conference.

Hinde, C.J. and Bray, A.D. (1992). Concurrent Engineering using Collaborating Truth Maintenance Systems. In Ed Max Bramer, Research and Development in Expert systems, C.U.P.

Hoffmann, C. and Hoperoft, J. (1985). Automatic surface generation in Computer Aided Design. TR 85-6617 Dept. of Computer Science, Cornell University, January.

Hogarth, W. (1953). "The Analysis of Beauty" (J. Burke, Ed.). Oxford University Press, Oxford

Hopcroft, J.E and Ullman, J.D. (1969) *Formal Languages and their Relation to Automata'* Addison-Wesley, Reading, Mass.

Hovy, Eduard H. (1993). *How MT Works.* Byte, January 1993, pp. 167-176

Jones, B. (1977). Beauty: 6. *Sunday Times*, December 11, pp. 74-75.

Jones, J. and Millington, M. (1986). *An Edinburgh Prolog Blackboard Shell.* University of Edinburg, Dept. of Artificial Intelligence Research Paper No. 281

Komatsu, K. (1988). Human skin capable of natural shape variation. *The Visual Computer, 3,* 265-271.

Kalra, P., Gobbetti, E., Magnenat-Thalmann, N. and Thalmann, D. (1993). A multimedia testbed for facial animation control. In T.S. Chua and T.L. Kunii, editors, *International Conference of Multi-Media Modeling, MMM'93*, pp. 59-72, Singapore, Nov 9-12.
1988. Kleiser Walczak Construction Comp.

Kanade, T. (1973). *Picture processing system by computer complex and recognition of human faces.* Dept. of Information Science, Kyoto University, Nov.

Kanade, T. (1977). *Computer recognition of human faces.* Basel and Stuttgart: Birkhauser Verlag.

Kanade, T. (1981). Recovery of the 3D shape of an object from a single view. *Artificial Intelligence 17,* 409-460.

Kaplan, R (1973). *A General syntactic processor.* In Rustin, R. (ed.), Natural Language Processing. New York: Algortithmics Press. 193-241.

Kay, M. (1967). Experiments *with a Powerful Parser.* In Proceedings of the Second International COLING Conference.

Kaya, Y. & Kobayashi, K. (1972). A basic study of human face recognition. In A. Watanabe (Ed.), *Frontier of Pattern Recognition* (pp. 265).

King, M (1983). *Parsing Natural Language.* Academic Press

Kleiser, J. (1989). A fast, efficient, accurate way to represnt the human face. State of the Art in Facial Animation. *ACM, SIGGRAPH '89 Tutorials, 22,*37-40.

Klir, G. J. and Yuan, B. (1995) "*Fuzzy Sets and fuzzy Logic, Theory and Applications*" , Prentice Hall, New Jersey.

Kodratoff, Y., Manago M., and Blythe, J. (1988). Generaliszation and Noise. *Knowledge Accquisition for Knowledge Based Systems* (Ed. Gaines & Boose). Academic Press, pp. 301-324.

Larrabee, W. (1986). A finite element model of skin deformation. *Laryngoscope, 96,* 399-419.

Laughery, K. R., & Fowler, R. F. (1980). Sketch Artists and Identi-kit procedures for recalling faces. *Journal of Applied Phsycology*, 65, pp 307-316.

Lee, Y., Terzopoulos, D. and Waters, K. (1993). Constructing physics-based facial models of individuals. In *Graphics Interface* '93, pp. 1-8, Toronto, ON, May.

Lee, Y., Terzopoulos, D. and Waters, K. (1995). Realistic Modeling for Facial Animation In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 55–62.

Lewis, J. P. & Parke, F. I. (1987). Automatic lip-synch and speech synthesis for character animation. *CHI+CG '87* , Toronto, 143-147

Magnenat-Thalmann, N. and Thalmann, D. (1987). The direction of synthetic actors in the film: Rendez-vous à Montrèal. *IEEE Computer Graphics and Applications*, pp. 9-19,December.

Magnenat-Thalmann, N., Primeau, N.E., & Thalmann, D. (1988). Abstract muscle actions procedures for human face animation. *Visual Computer, 3(5)*, 290-297.

Magnenat-Thalmann, N. and Thalmann, D. (1989). Synthetic actors. In *Vol 22: State of the Art in Facial Animation*, pp. 145-152. ACM Siggraph'89 Course Notes.

Magnenat-Thalmann, N. and Thalmann, D. (1994). "*Towards virtual humans in medicine: a prospective view*", Computerized Medical Imaging and Graphics, v. 18, n. 2, pp. 97-106.

Magnenat-Thalmann N, Thalmann D (1995). Digital Actors for Interactive Television, Proc. IEEE, Special Issue on Digital Television, Part 2, July, pp. 1022-1031.

Mamdani E. H. & Gaines B. R. 1981: Fuzzy Reasoning and its Applications. Academic Press.

Marigny, T., Adjoudani, A. and Benoit, C., (1994). A 3-D Model of the Lips for Visual Speech Synthesis. In *Proc. of the 2nd ESCA/IEEE workshop on Speech Synthesis*, pages 49-52, New Paltz, NY.

McAllestrer D. A. 1978: A Three-Valued Truth Maintenance System. M.I.T. AI Memo 473

McFetridge, Paul, and Nick J. Cercone (1990*). The Evolution of a Natural Language Interface: Replacing a Parser.* In Proceedings of Computational Intelligence '90, Milan, Italy. Appeared as Installing an HPSG Parser in a Modular Natural Language Interface. In N. J. Cercone, F. Gardin and G. Valle (Eds.) Computational Intelligence III (Proceedings of the International Symposium, Milan, Italy, September 24-28 1990). Amsterdam, Netherlands: Elsevier Science Publishers, pp. 169-178.

McFetridge, Paul (1991). *Processing English Database Queries with Head Driven Phrase Structure Grammar.* In Proceedings of the 2nd Japan-Australia Joint Symposium on Natural Language Processing (JAJSNLP '91*)*, Iizuka City, Kyushu, Japan.

Middleditch, A. E. (1985). and Sears, K. H. Blend surfaces for set theoretic volume modelling systems. *Computer Graphics* 19, 3, July , pp. 161-170.

Muzekari, L. and Knudsen, H. (1986). Effect of context on perception of emotion among psychiatric patients. *Perceptual and Motor Skills*, 62(1):79-84.

Nafarieh A. 1988: A New Approach to Inference in Approximation Reasoning and its Application to Computer Vision

Nahas, M., Huitric, H., & Sanintourens, M. (1988). Animation of a B-spline figure. *The Visual Computer, 3,* 272-276.

Norman, M. (1987). A Prolog set-theoretic equation solver. Loughborough Unversity M.Sc. Thesis.

Obermeier, Klaus K. (1989). *Natural Language Processing Technologies in Artificial Intelligence: The Science and Industry Perspective.* Chichester, West Sussex, England: Ellis Horwood.

Ohya J, Kitamura Y, Kishino F, Terashima N (1995) Virtual Space Teleconferencing: Real-Time Reproduction of 3D Human Images, Journal of Visual Communication and Image Representation, Vol. 6, No. 1, pp. 1-25.

Ohmura, K., Tomono, A. and Kobayashi, Y. (1988). Method of detecting face direction using image processing for human interface. *SPIE, Visual Communications and ImageProcessing* '88, 1001:625-632.

Oka, M. Tsutsui, K., Ohba, A., Kurauchi, Y., & Tago, T. (1987). Real-time manipulation of texture-mapped surfaces. *Computer Graphics, 21(4),* 181-188.

Parent, R. E. (1977). A System for Sculpting 3-D Data. *ComputerGraphics* 11, 2, July, pp. 138-147.

Parke, F. I. (1972a). Computer Generated Animation of Faces. University of Utah, Salt Lake City, June, UTEC-CSc-72-120.

Parke, F. I. (1972b). Computer generated animation of faces. *ACM Nat'l Conference, 1,* 451-457.

Parke, F. I. (1974). A parameteric model for human faces. *University of Utah,* UTEC-CSc-75-047, Salt Lake City, Utah, December.

Parke, F. I. (1975). A model of the face that allows speech synchronized speech. *Journal of Computers and Graphics,* 1, 1-4.

Parke, F. I. (1982). Parameterized models for facial animation. *IEEE Computer Graphics and Applications, 2(9),* 61-68.

Parke F. I. (1984). “A Parametric Model for Human Faces” Technical Report University of Utah.

Parry, S. R. (1986). Free-form deformations in a constructive solid geometry modeling system, Ph.D. Dissertation, Department of Civil Engineering, Brigham Young University, April.

Patel, M. & Willis, P. J. (1991). FACES: Facial animation, construction and editing system. In F. H. Post and W. Barth (Eds.), *EUROGRAPHICS '91* (pp. 33-45). Amsterdam: North Holland.

Pearce, A., Wyvill, B., Wyvill, G., & Hill, D. (1986). Speech and expression: A computer solution to face animation. In M. Wein and E. M. Kidd (Eds.), *Graphics Interface '86* (pp. 136-140). Ontario: Canadian Man-Computer Communications Society

Pearl, J. (1988). Probabilistic reasoning in intelligent systems: Networks of plausible inference. Morgan Kaufmann.

Pelachaud, C. (1991). *Communication and coarticulation in facial animation.* University of Pennsylvania, Department of Computer and Information Science, October.

Pelachaud, C., Badler, N. I., Viaud, M. (1994). Final Report to NSF of the Standards for Facial Animation Workshop, October. University of Pennsylvania.

Penry, J. (1971). "Looking at Faces and Remembering Them", pp. 28-29, 32-38 Elek Books, London

Pentland, A., Etcoff, N., Starner, T. (1992). *Expression recognition using eigenfeatures.* M.I.T. Media Laboratory Vision and Modeling Group Techical Report No. 194, August.

Peterson, M. T. (1997). "3D Studio Max 2 Fundamental", New Riders Publishing, pp. 67-91.

Pieper, S. D. (1989). *More than skin deep: Physical modeling of facial tissue.* Massachusetts Institute of Technology, 1989, Media Arts and Sciences, MIT.

Pieper, S. D. (1991). *CAPS: Computer-aided plastic surgery.*Massachusetts Institute of Technology, Media Arts and Sciences, MIT, September.

Platt, S. M. (1980). *A System for Computer Simulation of the Human Face.*,The Moore School, 1980, Pennsylvania.

Platt, S. M. (1985). *A structural model of the human face.* The Moore School, Pennsylvania.

Platt S.M. and Badler N. I. (1981). "Animating Facial Expressions," *Computer Graphics*, Vol. 15, No. 3, August, pp. 245-252.

Rockwood, A. P. and Owen, J. (1986). Blending surfaces in solid modeling, in *Geometric Modeling,* G. Farin, editor, SIAM.

Roeck, A. De (1983). *An Underview of Parsing in Parsing Natural Language.* Academic Press

Sabin, M. A. (1970). Interrogation techniques for parametric surfaces, Proceedings~ Computer Graphics '70, Brunel University, April.

Sakai, T., Nagao, M. and Kanade, T. (1972). Computer analysis and classification of photographs of human faces. Proceedings of the First USA – Japan Computer Conference, pp. 55-62.

Schmucker K. J. 1984: Fuzzy Sets, Natural Language Computations and Risk Analysis. Computer Science Press

Sederberg, T. W. and Parry, S. R. (1986). Free-form deformation of polygonal data, *Proceedings, International Electronic Image Week,* Nice, France (April), pp. 633-639.

Shafer G. (1976). *A Mathematical Theory of Evidence,* Princeton University Press.

Shaherazam (1986). The Mac-a-Mug Pro Manual, Milwaukee, Wisconsin: Shaherazam

Shanahan M. & Sothwick R. 1989: Search, Inference and Dependencies in Artificial Intelligence. Ellis Horwood.

Shepherd, J. W., Ellis, H. D., & Davies, G. M. (1977). Perceiving and remembering faces. Technical report to the Home Office under contract POL/73/1675/24/1.

Shepherd, J., Davies, G., & Ellis, H. (1981). . In G. M., H. D. Ellis, & J. W. Shepherd (Eds.), *Perceiving and remembering faces.* London: Academic Press
Baker E. & Seltzer M. (1998). The Mug-Shot Search Problem, Vision Interface Proceedings, Vancouver, Canada.

Sijtsma, Wietske, and Olga Zweekhorst (1993). *Comparison and Review of Commercial Natural Language Interfaces. In Franciska M.G de Jong and Anton Nijholt (Eds.)* Natural Language Interfaces: From Laboratory to Commercial and User Environment, Proceedings of the Fifth Twente Workshop on Language Technology (TWLT5*).* Universiteit Twente, Enschede, Netherlands, pp. 43-58.

Stallman R. & Sussman G. 1977: Forward Reasonong and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis. Artificial Intelligence Vol. 9. pp. 135.

Strzalkowski, Tomek, and Nick J. Cercone (1986). *A Framework for Computing Extra-Sentential References.* Computational Intelligence, 2, (4), pp. 159-180

Terzopoulos, D. & Waters, K. (1990a). Analysis of facial images using physical and anatomical models. *Proceedings of the International Conference on Computer Vision,* 1990, 727-732.

Terzopoulos, D. & Waters, K. (1990b). Physically-based facial modeling, analysis, and animation. *Journal of Visualization and Computer Animation, 1(4),* 73-80.

Todd J.T., Mark Leonard S., Shaw Robert E., and Pittenger John b. (1980). "The Perception of human Growth," *Scientific American,* Vol. 242, February, pp. 132-134.

Townes, J.R. (1976). A computer algorithm for mug shot identification. EAI Symposium on Automatic Imagery Pattern Recognition, College Park, Maryland.

Turk, M. A. & Pentland, A. P. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience, 3(1),* 71-86.

Vanezis, P. (1999) "Identity Crisis" Article in New Scientist, 27 February, pp 40-46.

Vannier, M. W., Pilgram, T., Bhatia, G., & Brunsden, B. (1991). Facial surface scanner. *IEEE Computer Graphics and Applications, 11(6),*72-80.

Veenman, P. R. (1982). The design of sculptured surfaces using recursive subdivision techniques, in: *Proc. Conf. On CAD/CAM Technology in Mechanical Engineering,* MIT, Cambridge, March.

Waite, C. T. (1989). *The Facial Action Control Editor, Face: A Parametric Facial Expression Editor for Computer Generated Animation.* Massachusetts Institute of Technology, Media Arts and Sciences, Cambridge, Febuary.

Walczak, K. (1988). Sextone for President. *ACM SIGGRAPH Video Revie*w, vol. 38/39. Kleiser Walczak Construction Comp.

Wang, S.-G. & George, N. (1991). Facial recognition using image and transform representations. In *Electronic Imaging Final Briefing Report,* U.S. Army Research Office, P-24749-PH, P-24626-PH-UIR, The Institute of Optics, University of Rochester, New York.

Watanabe, Y. & Suenaga, Y. (1992). A trigonal prism-based method for hair image generation. *IEEE Computer Graphics and Applications,* January, 47-53.

Waters, K. (1986). Expressive three-dimensional facial animation. *Computer Animation (CG86)* , October, 49-56.

Waters, K. (1987). A muscle model for animating three-dimensional facial expressions. *Computer Graphics (SIGGRAPH'87), 21(4),* July, 17-24.

Waters, K. (1988). *The computer synthesis of expressive three-dimensional facial character animation.* Middlesex Polytechnic, Faculty of Art and Design, Cat Hill Barnet Herts, EN4 8HT, June.

Waters, K. & Terzopoulos, D. (1990). A physical model of facial tissue and muscle articulation. *Proceedings of the First Conference on Visualization in Biomedical Computing,* May, 77-82.

Waters, K. & Terzopoulos, D. (1991). Modeling and animating faces using scanned data. *Journal of Visualization and Animation, 2(4),* 123-128.

Waters, K. & Terzopoulos, D. (1992). The computer synthesis of expressive faces. *Phil. Trans. R. Soc. Lond., 355(1273),* 87-93.

Williams, L. (1990). Performace driven facial animation. *Computer Graphics, 24(4),* 235-242.

Wise, B.P. (1986). An Experimental Comparrison of Uncertain Inference Systems. Carnegie-Mellon University Ph.D. Thesis, pp. 1-61.

Woods, W.A. (1970) Transition network grammars for natural language analysis. *Communications of the ACM* 13, pp. 591-606

Wu, J. K., Ang, Y. H., Lam, P., Loh, H. H., and Desai, A. (1994). Inference and Retrieval of Facial Images, *Multimedia Systems,* 2:1-14.

Wu, J. K. (1998) - "Fuzzy content-based retrieval in image databases", Information processing & management, Vol.34, No.5, pp.513-534

Wyvill, B. (1989). Expression control using synthetic speech. *State of the Art in Facial Animation, SIGGRAPH '89 Tutorials, ACM, 22,* 163-175.

Yacoob, Y. and Davis, L. (1994). *Computer Vision and Pattern Recognition Conference*, chapter Computing spatio-temporal representations of human faces, pp. 70-75. IEEE Computer Society.

Yamana, T. & Suenaga, Y. (1987). *A method of hair representation using anisotropic reflection.* IECEJ Technical Report PRU87-3, May, 15-20, (in Japanese).

Yuille, A. L. (1991). Deformable templates for face recognition. *Journal of Cognitive Neuroscience, 3(1),* 59-70.

Zadeh L.A. (1965) "Fuzzy Sets". Journal of Information and Control, 8 pp.338-353

Zadeh L.A. (1979) "A Theory of Approximate Reasoning", in Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh, R. Yager et. al., Editors 1979, John Wiley & Sons. pp. 149-194.

Zadeh L. A. 1986: Is Probability Theory Sufficient for dealing with Uncertainty in AI: A Negative View. Uncertainty in Artificial Intelligence. Ed. Kanal & Lemmer. Elsevier. Pp. 103 – 116

Zadeh L.A. (1996) "*Fuzzy Logic = Computing with Words*", IEEE Transactions on Fuzzy systems, Vol.4 No.2, pp. 103-111.