Om
SaiRam

Max: A Talking Head

6.837 Final Project Report

December 3, 1999

Theresa Burianek
Roshan Gupta
Sripriya Natarajan
TA: Damian Isla

Abstract

This paper describes the design and development of an animated face that moves in synchrony to spoken text. Based on the user mode, the animated face either repeats input text or converses with the user. We generated realistic facial models, visemes, with several modeling tools, and our implementation uses IBM ViaVoice as our text-to-speech engine and OpenGL implementation for Java to perform the actual rendering. Despite many issues with the phoneme generation of ViaVoice, we have achieved fairly good synchronization. Experiments in linearly interpolating between visemes, either evenly or with acceleration, to generate more frames, suggest that spline curve interpolation may not be necessary to create a realistic speaking face since the movements are small and the triangulation of the face is very precise. This interpolated animation suffers from speed limitations imposed by our model and rendering utility and is not well synchronized with the sound. Although our final application does not use transitioning and has a frame rate of one frame per phoneme, the visual effect is quite compelling.

---

1. Introduction

This paper discusses our effort to create an animated face that appears to speak, either by repeating a user-given line of text or by responding conversationally to a user. We will first describe prior work that has been done in this area and our motivation for creating a talking head. Section 2 will then explain our initial goals for the project. Section 3 describes our actual achievements in the project by both giving the details of the system?s design and discussing the problems encountered during design and implementation. Section 4 outlines the individual contributions to the project. We conclude in Section 5 with a discussion of the lessons we have learned through doing this project.

1.1 Background Information

Animating and rendering faces with realistic expressions has been an active area of research in both academics and industry. Studies show that facial information contributes significantly to an observer?s comprehension and interpretation of the corresponding speech. In fact, visual cues are stronger than auditory cues (that is why it appears that the actors are talking in a movie although the sound comes from loudspeakers). Thus, providing visual cues via an animated face should improve the comprehensibility of computer-generated speech. This task is as difficult as it is important, since the human face has many degrees of freedom and nonlinear couplings, and its posture cannot always be uniquely deduced by the spoken words.

There has been much work in recent years to address many relevant issues. Matthew Brand of MERL has studied the motion of the entire face, and developed a probabilistic model of which face position should be displayed given a vocal recording. Researchers from Microsoft Corporation and the University of Washington have created a system to capture from video data the three-dimensional geometry, color and shading data to create a polygonal face model for rendering. The University of Washington, The Hebrew University and Microsoft Research have studied techniques for creating three-dimensional facial models from photographs. There are also commercial "talking head" programs available; *Verbot*, for example, features a speaking alien as well as human faces. These are just a few of the many examples of recent work done in this area.

There are essentially two problems to be solved: (1) to generate a facial model, and (2) to manipulate this model, driven manually, by text or by sound waves. Given text, a model can also be driven by a series of phonemes derived from that text. Phonemes are the basic unit of sound used to form words. The IPA and ARPANET Standard lists 66 phonemes used to model speech, although other systems classify phonemes differently. Each phoneme is associated with a unique facial position, or viseme, which can be used to guide the animated face?s animations as it speaks.

1.2 Motivation

Each of the team members had some background knowledge (although of varying degrees) of speech synthesis, speech recognition or speech processing prior to starting this project, so we sought to pursue the implementation of a graphical interface that incorporated this specific interest. We wanted to explore the modeling of different visemes, the facial expressions that correspond to different phonetic sounds, and the transitions between them to produce a realistic computer "talking head." We chose to develop our system using Java, given the ease of development in Java and its potential cross-platform portability.

1.3 Relation to Computer Graphics

Realistic model generation, smooth animation and synchronization of sound with graphics to enhance the visual experience are all issues of our project that are pertinent to graphics. Model generation required us to become familiar with several graphical modeling tools. Making the face appear to talk forced us to consider the different degrees of accuracy in animation. We also had to resolve issues about whether generating images in real-time would be sufficiently efficient to smoothly synchronize the sound and motion. Finally, our commitment to developing our system in Java required that we learn OpenGL, the standard graphics backbone for two- or three-dimensional rendering.

2. Goals

Our initial goal was to create an interface where a user could type in a line of text, and then have it be spoken by a "talking head," which would be modeled in three dimensions with triangles. Potentially, there would be a large pause while the text was converted to phonemes and all the images required for the given text were calculated beforehand, but we hoped to draw efficiently enough to generate images as needed. We planned to generate a facial image for each phoneme and generate transition images between these images. We predicted that the generation of these images, based on published information relating lower face positions (lips, tongue, jaw, etc.) to the phonemes, would constitute a significant

portion of the work for the project. We wanted to explore different methods of making transitions between different visemes, beginning with a simple linear interpolation of vertices, then attempting acceleration and finally, if necessary, attempting interpolation along a spline curve describing the face contour, in order to find the simplest way to achieve high output quality.

Our original design for the system involved three basic steps:

(1) Convert text to phonemes

(2) Look up the images corresponding to the phonemes and phoneme transitions

(3) Display these images in synchronization with the output of our text-to-speech (TTS) engine to present an animated face

3. Achievements

3.1. Max: A Talking Head

Users are presented with a picture of an animated face, affectionately dubbed Max. Below Max is a marquis that prints his words as he speaks, a box in which the user can input text and buttons to clear the box and submit the text. Our application runs in three modes, Copycat, Psychotherapist and Narrator. In Copycat mode, Max speaks whatever the user types in. In Psychotherapist mode, Max replies to the user input. An Eliza engine hooked up to the system generates Max?s answers. Finally, in narrator mode, Max speaks the contents of a text file chosen by the user. There is also a menu to switch Max?s voice; the choices depend on the voice synthesizer used. (See Figure 1.)

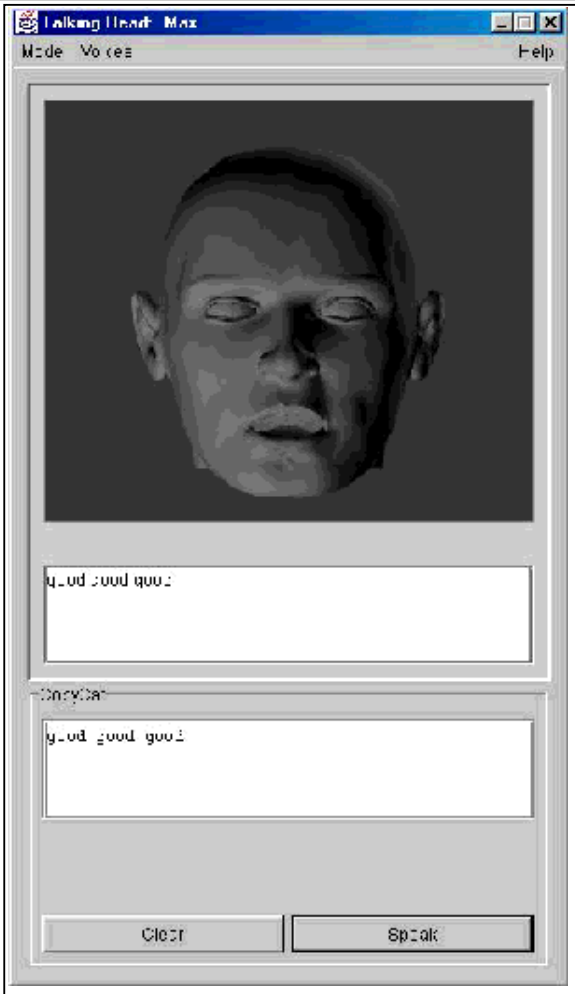

Figure 1. Screensot of Max: The Talking Head

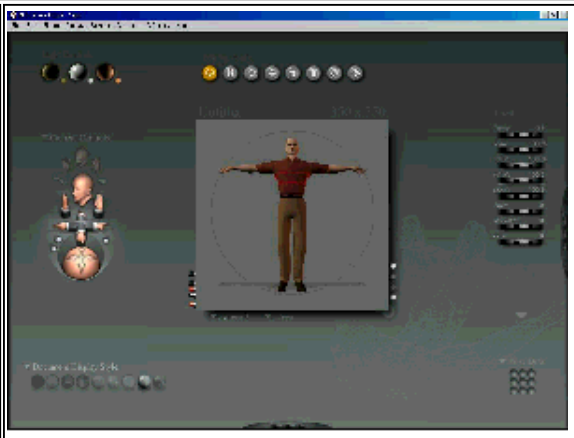

Figure 2. Screenshot of Poser

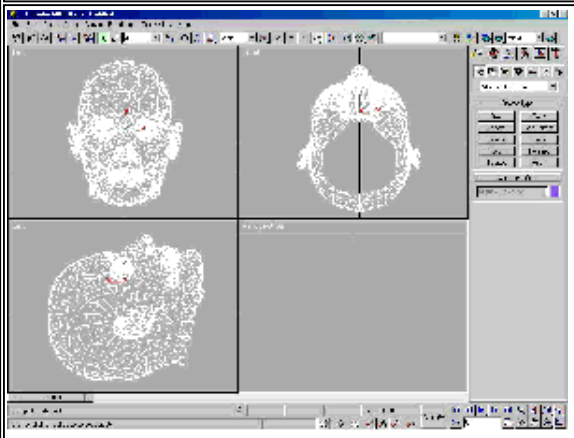

Figure 3. Screenshot of 3D Studio Max

Figure 1.

3.2. Building Max

3.2.1. Face Modeling

Creating the animated face for the final application requires multiple steps through multiple applications. First we determined our requirements for the animated face. We needed a modeled face for each one of the phonemes we wished to display. We needed the file format for the models to easily parse through our Java application. We had initially chosen VRML for the file format, on the basis that there are VRML parsers freely available. However, as we started investigating these, they presented many compilation errors within our tools, and so we chose to implement the parser through our own application.

Having chosen VRML as the file format, we needed an application to generate these files. We looked at many graphics applications, including Poser and 3D Studio Max. Both of these

applications gave us pieces of what we needed: 3D Studio Max produced the VRML files we were looking for and Poser had a full human figure built on the triangular structure we were looking for. Poser also had many toggles that allowed for easier manipulation of the facial expressions such as opening the mouth, widening the mouth and moving the tongue up and down. Because neither package individually produced what we desired, we chose to incorporate both into our modeling scheme. The face manipulation and modeling was done in Poser (Figure 2), then just the head was exported to 3D Studio Max. With 3D Studio (Figure 3), we exported the face to VRML format. We used 3D Studio Max to scale the size of the face, specify color per vertex, which was missing in the Poser export, and add normals. Adjusting all these attributes for the VRML export facilitated parsing and rendering in the final application.

3.2.2. Code Structure

We attempted to structure our code hierarchically and with abstraction barriers, although these design advantages were sometimes sacrificed for speed. The GUI serves as an interface to a main class that passes the text to the TTS engine. A set of phonemes, grouped by word, is returned for each sentence. Two threads are simultaneously dispatched, one to speak each word, and the other to render the face, using OpenGL calls, based on the phonemes in a word.

3.2.2.1. JSAPI Interface

Accessing the TTS engine via the Java Speech API (JSAPI) was a straightforward task. When the application first starts, a Synthesizer object is instantiated and is allocated resources. To retrieve the list of phonemes corresponding to a line of text, the application simply calls the synthesizer?s phoneme method. To speak a given line of text, the application calls the synthesizer?s speakPlainText method. Unfortunately, the actual TTS engine utilized, IBM ViaVoice, did not comply completely with these JSAPI conventions, which ultimately affected the way we utilized the TTS. For example, we discovered that upon calling the method phoneme, all subsequent calls to speakPlainText would not work. Also, IBM ViaVoice would sometimes fail to generate every phoneme for certain words; for example, the vowel sounds in "how", "bite", and "boy" would be missing. Section 3.3.1 details how we dealt with these issues.

3.2.2.2. Face Animation

The graphics class, AnimateFace, provides functions that manipulate the state of the canvas on which the face is drawn. When a set of phonemes are passed to AnimateFace, the class looks up the corresponding viseme information and passes this information onto the canvas, which then renders the transition from the current face position to the next face position.

When the program is started, serialized Vectors of Triangle objects are loaded for each viseme. Each Vector contains an ordered set of Triangles specifying a face position. These objects are created by a helper function that parses a VRML file under the assumption that only triangles are contained in the file. The parser collects all the

coordinate and color information for each triangle in a Triangle object. The y coordinate was set to be original z coordinate and the z coordinate set to be the negative of the original y coordinate, since **y** is the up vector in OpenGL but **z** is the up vector in Poser. Originally, we intended to have each Triangle, which contains vertex and color per vertex values, to render itself onto the canvas. This abstraction however proved costly in terms of rendering time, so each Vector of Triangles is converted into large arrays of floats, where the R, G, B colors and then the X, Y, Z coordinates are stored in order for each vertex of each triangle. These float arrays are themselves stored in an array. To shorten the start-up time, we have also modified the parser to create binary files of floats instead of serialized Triangle objects.

To draw a particular phoneme, the appropriate float array is passed to the canvas. For each frame that is rendered, the canvas display function sets the GL parameters, such as background color and view frustum. The shade mode is set for Gouraud shading so that, given the colors per vertex, a very smooth face will result. After setting the overall parameters, for every three vertices specified in the float array, the display function begins a GL_TRIANGLE, specifies color and position for each vertex, and then ends the graphic. After processing every vertex in the array, the image is flushed to the screen. The full implementation of our animated face usually displays one frame per phoneme due to speed concerns, but we have programmed different versions of the display function that generate an arbitrary number of frames that are linearly interpolated between visemes. A modification of this transition technique accelerates the interpolation with time, rather than evenly moving from one viseme to another.

3.3 Obstacles

3.3.1. Imported Code Packages

A primary frustration over displaying our face stemmed from the difficulty in finding a Java 3D rendering package. Initial attempts were made with both an "OpenGLForJava" system and a different system that read in VRML files and automatically parsed and rendered the specified image using Java3D. These graphics packages, as well as a few VRML parsers that we tried, suffered from several compilation problems, poor documentation and general bugginess. The GL4Java package from Jausoft finally alleviated these problems, but by then, it was too late to investigate the synchronization, optimization and transition issues in the depth we originally intended. We underestimated the time it would take to learn these tools and get them up and running, and so were unable to explore the significant issues and to attempt extensions like mapping skin texture onto the face.

Another major issue in our project was finding a TTS engine for the spoken words. This proved to be a major issue because of the requirements of our design. The engine needed to implement the JSAPI interface for our Java application. We eliminated our initial idea of using Microsoft?s SAPI interface through a Java conversion tool once we discovered that it was an incomplete implementation. The engine also needed to export phonemes from the text given. It was difficult to find a readily available TTS engine that conformed to these specifications; most did not extend the permissible interface with all these options.

We were able to acquire a commercial copy of IBM ViaVoice from Justine Cassell of the MIT Media Lab. This satisfied our basic requirements but as we got deeper into

implementation we found restrictions in this product. The specifications within ViaVoice suggest that the phoneme string generator does conform to the IPA (International Phonetic Alphabet) standards. However, in testing words with the phone generator we discovered some phonemes were not represented, and that Via Voice returned other non-phone characters in its response. In discussing this with IBM engineers we were told that the phoneme generator was inherently buggy since they had hacked their own code to get around the fact that their original C implementation did not provide access to the phoneme decomposition of the text. After more observation of the phonetic structure (as given by ViaVoice) we concluded that these extra characters were duration markers within Via Voice. The lack of accurate documentation and bugginess within Via Voice caused frustration and annoyance throughout our project.

Fortunately, we were able to compensate for some of the shortcomings of ViaVoice with a little bit of creativity. For example, as previously mentioned, calling the phoneme method would somehow disable the ViaVoice TTS engine. We were successfully able to isolate the problem as a failure to signal the end of synthesized line of text, and we were able to work around the error by reallocating the engine?s resources-essentially reinstantiating it- after every call to phoneme. This trick was not a time sink, as we feared it might be. The other main problem was ViaVoice?s failure to generate certain phonemes as well as its generation of phonemes outside of the IPA character set. To compensate, we simply left the face in its current position until the start of a new phoneme (in the case of a missing phoneme) or the start of an IPA phoneme (in the case of unsupported phonemes).

3.3.2. Modeling Tool Limitations

Our system?s slow frame rate also complicated our attempts to synchronize face movement and sound. If our face was a less complex representation, speed could be greatly enhanced, but neither Poser nor 3D Studio Max featured options to save a less precise model of the face. Although the face looks very realistic, it also has the complex couplings of motions of a real face; for example, opening the mouth affects the crinkle of the eyes. While brainstorming how to transition between faces, we had considered the idea of discretizing facial movements into small changes like lips open, jaws dropped, etc., but the couplings prevent such a simplification of the transition issue. We could ignore some features such as the back of the head or the ears, but Poser exports the entire face (except the eyeballs) as one object, so this optimization was not possible. In retrospect, modeling our own face might have given us more control over the complexity and vertex movement, but creating a realistic face would have been a great challenge.

3.3.3. Java Issues

The frame rate was also affected by our decision to implement the animated face in Java, initially influenced by our desire have our code be potentially platform independent, just by switching the underlying TTS engine. For the sheer number of graphics calls that need to be made, however, Java code does not run fast enough. Each face is made up of over 8,000 triangles, and the smoothness of facial transitions is greatly limited by rendering speed. Currently, displaying more than one frame per phoneme causes an audible pause between words and the face stops moving noticeably after the voice has stopped talking. Our decision to sacrifice object-oriented design for speed provided one optimization to increase frame

rate, but given the tools we used (IBM ViaVoice, Java, Jausoft?s GL4Java, and the modeling tools), the frame rate is constrained to be relatively slow and synchronization must be achieved through heuristic methods.

4. Individual Contributions

4.1. Theresa Burianek

Theresa?s was to provide the phonetic modeling of faces. Having taken courses in speech recognition, she was the group member most qualified to analyze the phonetic requirements of the program. Her contributions included generating models for each phoneme and correlating the output of IBM?s phone generator with the IPA (International Phonetic Alphabet). Theresa also assisted with some of the text-to-speech issues within the main application and the correlation of the spoken text with the graphical modeling.

4.2. Roshan Gupta

Roshan investigated all the JSAPI issues, then designed and implemented the code to interface with IBM ViaVoice?s classes. He developed the workaround for obtaining phonemes without crashing the speech engine. With Theresa, he assisted in the correlation of the IBM phoneme outputs with the IPA, and modified some features of Sripriya?s VRML parser. Roshan also integrated graphics and TTS modules, which involved some manipulation of threads, and designed the GUI. He also integrated an Eliza for Java program into the system so that our face could have a conversation with the user instead of only repeating what the user typed.

4.3. Sripriya Natarajan

Sripriya?s role was to investigate the graphics utilities available for Java and then create a parser that would read in the VRML files, create any necessary data structures and make the appropriate calls to display the specified face onto the screen. She designed the structure of the graphics classes and implemented their methods. Since she became the most familiar with the OpenGL syntax, she also wrote the experimental display methods for the even and accelerated linear interpolation animation schemes for transitioning between faces.

5. Lessons Learned

Throughout the course of this project, each of the group members learned many lessons. In addition to expanding our knowledge of graphical and speech systems and tools, we learned about the nuances of creating an application with many internal and external dependencies.

5.1. Graphics Lessons

This project gave us an opportunity to familiarize ourselves with many graphics tools. Through creating the face models, Theresa and Roshan learned the many nuances of the modeling tools Poser and 3D Studio Max. As we developed a process to port Poser faces to an ASCII-style file format, we realized the preponderance of graphics formats available. While writing a parser to

interpret the VRML file, we became very familiar with the format and its similarities to Inventor files. These modeling and format issues are very important to create a good visual effect using several different tools.

While exploring the different OpenGL tools for Java and finally writing the display code, Sripriya became very familiar with OpenGL and its capabilities. This lesson was very valuable since OpenGL is a veritable standard for writing rendering three-dimensional objects to the screen. It has been implemented in both C and Java and is a necessary tool to know when developing or enhancing graphics tools.

Finally, in our brief exploration of transitioning between visemes, we were able to explore some of the animation issues discussed in 6.837, trying even and accelerated linear interpolation. We initially expected linear interpolation to be utterly unacceptable, but even modest interpolation provided rather smooth transitions. The change of facial position from phoneme to phoneme is fairly subtle, and the triangles that make up the face are each very small, so perhaps linearity is a good assumption in these areas. Since the face shading was generated by vertex colors rather than lighting, perhaps interpolating the colors made the transition appear somewhat smoother than would be expected. Furthermore, the human eye itself does some interpolation. Even when only one face is flashed per phoneme the effect of motion is surprisingly compelling. Although time did not permit us to test the performance of moving vertices along spline curves, the project did give us an opportunity to consider these animation issues in a concrete context.

5.2 Software Engineering Lessons

This project also taught all three group members important lessons of software engineering. We all underestimated how difficult it would be to integrate outside code packages (OpenGL for Java, JSAPI) and how much time would be spent, not debugging our own code, but inventing ways to circumvent their bugs! Although these issues were obstacles in our development of an animated face system, they have taught us to watch out for these issues in future projects.

While solving these problems, we learned many coding lessons. We had to make decisions between having clean object-oriented code and increasing frame rate. Experimenting with this issue demonstrated to us the overhead of object method calls. Since we had neatly dividing up the coding tasks, we had to continually keep the ease of integration in mind while programming our own sections. Since most large systems are developed by more than one person, this experience was very valuable. In addition, we were able to enhance our Java skills through while coding our system.

5.3 Speech Lessons

Through our struggles with JSAPI and IBM ViaVoice, we gained a lot of insight into what tools are realistically available for generating speech from text. We have a much better understanding of what the latest commercial technologies are for text-to-speech conversion. This project enhanced our understanding of speech modeling issues as well. Mapping the different phoneme codes generated by IBM ViaVoice to the ARPAbet standard exposed us to other phoneme classification. Matching phonemes to visemes taught us that many of the sound differences are generated more in the throat than in upper oral cavity, so some phonemes mapped to the same viseme-in fact, we finally needed only 29 visemes. Finally, the research we did prior to starting our project exposed

us to some of the other approaches taken to this problem, such distorting photographic images instead of using a polygonized model or using sound signals directly to drive the face movement instead of phoneme information.

6. Acknowledgements

We would like to thank several people for their advice and help as we developed our graphics project. Our 6.837 TA, Damian Isla, provided valuable feedback and pointed us in the right direction for many of the graphics tools we used. Professor Seth Teller also provided feedback on our proposal so that we were aware of what issues to consider as our system developed. Professor Justine Cassell suggested several research sites to visit for ideas. We thank her and Dave Boor for helping us to obtain a copy of IBM ViaVoice, without which we would not have been able to interface to a working Java TTS engine. We are extremely indebted to Max Chen who pointed us to several useful papers for background, lent us the modeling tools that we used and readily helped us whenever we had a question about the tools. Thanks to Scott Eaton for directing us to Jausoft?s GL4Java, a welcome relief from the other buggy OpenGL for Java tools we were using!

Our system uses several external code packages. IBM ViaVoice provides our TTS engine. Jausoft?s GL4Java implements OpenGL and provides an interface for us to render our objects to the screen. The Eliza module is from Charles Hayden (http://www.monmouth.com/~chayden/eliza).

7. Bibliography

Brand, Matthew. "Voice Puppetry." SIGGRAPH, 1999

Guenter, Brian, et. al. "Making Faces." ACM, 1998

Pighin, Frederic, et. al. "Synthesizing Realistic Facial Expression from Photographs." ACM, 1998

Spoken Language Systems Group, "6.345 Course Notes - IPA Standard." M.I.T., Spring 1999, http://www.sls.lcs.mit.edu/6.345/notes/IPA/P001.html

8. Appendix

The source code files can be found at

- http://web.mit.edu/nataraja/www/6.837/Max.zip and
- http://web.mit.edu/nataraja/www/6.837/VRMLParser.zip.

The application, however, will only run on a PC which has IBM ViaVoice installed; thus it cannot be readily compiled and used. Currently, the application is also always launched from Metrowerks CodeWarrior rather than using a command line call to a make file. The application will be demonstrated on December 9, 1999, at 7:25 PM in M.I.T. Rm. 3-133.