00324

5

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

UN SISTEMA DE MODELADO DE EXPRESIONES FACIALES CON SINCRONIZACION DE VOZ EN ESPAÑOL

T E S I S

QUE PARA OBTENER EL TITULO DE

M A T E M A T I C O

P R E S E N T A

JORGE CASTRO CESAR

FACULTAD DE CIENCIAS
UNAM

DIRECTOR DE TESIS:

MAT. ANA LUISA SOLIS GONZALEZ COSIO

MEXICO, D. F. 2003

PAGINACION DISCONTINUA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Autorizo a la Dirección General de Bibliotecas de la UNAM a difundir en formato electrónico e impreso el contenido de mi trabajo recepcional.

NOMBRE: Jorge Castro César

FECHA: 11 | Febrero | 2003

FIRMA:

DRA. MARÍA DE LOURDES ESTEVA PERALTA
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

Un sistema de modelado de expresiones faciales con sincronización de voz en español

realizado por Jorge Castro César

con número de cuenta 090524208 , quien cubrió los créditos de la carrera de:

Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Mat. Ana Luisa Solís González-Cosio	
Propietario	Dr. Homero Vladimir Ríos Figueroa	
Propietario	M. en C. Miguel Miranda Miranda	
Suplente	Dra. Amparo López Gaona	
Suplente	M. en C. Jorge Luis Ortega Arjona	

Consejo Departamental de Matemáticas

M. en C. José Antonio Gómez Ortega

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL DE MATEMÁTICAS

Contenido

Dedicatoria

A mi esposa:

Fis. Erica Maribel Reyes Gutiérrez

Dedico esta tesis por compartir los sufrimientos y la desesperación que pase para realizar este trabajo, por impulsarme y creer en mí. Sin su gran apoyo esta tesis no hubiera sido posible. Gracias por su cariño y comprensión. TE AMO.

A mi papá:

Sr. Dionisio Castro Huerta

Dedico este trabajo por su gran apoyo, cariño y ejemplo de lucha. El cual me enseño a levantarme una y otra vez hasta lograr mi objetivo, aun cuando todo parecía desfavorable.

A mi mamá:

Sra. María Luisa César Preciado

Dedico esta tesis por toda una vida de entrega, dedicación y apoyo. Dentro de los cuales sus consejos me ayudaron de guía para lograr todo lo que me he propuesto.

A mi Sensei:

Lic. Nobuyoshi Murata

Dedico esta tesis a mi maestro y gran amigo, él cual sin saberlo me indujo a la Ciencia. Sin sus enseñanzas y consejos de que el Karatedo Shito Ryu es considerado un sistema científico. No hubiera tomado la decisión de estudiar esta increíble carrera y mucho menos hubiera sido posible este trabajo.

Agradecimientos

A mi maestra y tutora:

Mat. Ana Luisa Solís González Cosio
Mi gran admiración y respeto porque sin su ayuda y enseñanzas no hubiera sido posible la realización de este trabajo.

A mis hermanas:

Srita. Patricia Castro César
Gracias por todas las alegrías compartidas de toda una vida, por su apoyo y cariño. Pero sobre todo por considérame su gran amigo en los momentos difíciles de la vida.

A mi Tío:

Dr. Salvador Castro Huerta
Mi gran admiración y respeto porque sin su ayuda no hubiera sido posible terminar este trabajo, así como sus valiosos consejos que me inspiraron para seguir adelante.

A mis sinodales y amigos:

M. en C. Miguel Miranda Miranda, M. en C. Jorge Luis Ortega Arjona, Dr. Homero Vladimir Ríos Figueroa y Dra. Amparo López Gaona. Gracias por su amistad, sus consejos y observaciones al presente trabajo.

Prof. Rocío Castro César
Gracias por ser una gran amiga, alumna y colaboradora, por compartir conmigo sus logros, sueños y anhelos. Pero sobre todo por su cariño y ayuda.

A mis suegros:

Sr. Pedro Reyes y Sra. María del Carmen
Gracias por ser mis amigos e impulsarme a terminar este trabajo. Sin su apoyo y gran cariño no hubiera sido posible esta tesis.

A mi maestro:

Dr. Humberto Cárdenas Trigos

Gracias por brindarme su incondicional apoyo por el lapso de dos increíbles años en el Instituto de Matemáticas, época en el cual aprendí de él su ejemplo, que la ciencia matemática no es solo la acumulación de conocimientos y números, sino, una forma de disfrutar, de ver y explicar diferentes sucesos que se tienen en la naturaleza.

A mis abuelos, tíos y primos:

Pero en especial **al Sr. Andrés Castro, al Sr. Alfredo César, al Sr. Alfonso Castro, al Sr. Eduardo Castro, al Sr. Armando Castro, al Sr. Valentín Nequíz, a la Sra. Alejandra César, a la Sra. Sara Castro y a la Sra. Alejandra Bustos.**

Gracias por permitirme compartir esta alegría y hacerme sentir una persona triunfadora.

A mis alumnos:

Lic. Raúl Arriaga, Prof. Miguel Agustín, Prof. Carlos Ayala, Sra. Tere Yañes, al joven Alejandro Chávez y al Ing. Moisés Jaramillo por brindarme su apoyo y cariño.

A mi Sensei:

Prof. German Mendoza López

Gracias por ser mi amigo, compañero y maestro, por escucharme, protegerme y apoyarme en todos mis proyectos. Pero sobre todo, por que siempre ha estado ahí para aconsejarme. Por esto, más que mi amigo, lo considero el hermano mayor que nunca tuve.

A la familia García Reyes :

Lic. Luis García y Lic. Claudia Reyes

Gracias por compartir sus alegrías y triunfos conmigo, pero sobre todo por su gran apoyo.

A la familia Lavariega Reyes:

Ing. Juan Lavariega y Lic. Carmen Reyes

Gracias por su amistad y cariño, pero sobre todo por estar siempre felices y con una sonrisa en el corazón.

A mis amigos y compañeros de la Facultad de Ciencias:

Lic. José Campero, Act. Claudia Palacios y a la Srita. Norma Ramírez, y muchos otros que me ayudaron y compartieron su increíble amistad conmigo.

Capítulo 1

Introducción

Este trabajo forma parte del proyecto NFS/Conacyt "Distributed Simulation of Gesture Recognition Interfaces and Intelligent Agents for Virtual Environments", financiado por CONACYT. Uno de los objetivos del proyecto es crear agentes inteligentes con capacidad de diálogo en ambientes virtuales.

La presente tesis está enfocada a resolver uno de los problemas de la arquitectura general del proyecto antes mencionado, que consiste en incorporar un sintetizador de voz a un modelo facial con el objeto de producir sincronización. Para esto se desarrollo:

1. Un editor de visemas y expresiones, que servirá para modelar la boca del modelo facial.
2. Un módulo que incorpora un sistema que convierte texto en sonido denominado Festival al modelo facial.
3. Un módulo de sincronización entre el modelo facial y el sonido.

4. Documentación por medio de diagramas que permiten comprender el proceso de sincronización entre el modelo facial y la generación del sonido.

La tesis esta estructurada de la siguiente manera:

En el CAPÍTULO PRIMERO, se presenta esta introducción.

En el CAPÍTULO SEGUNDO, se habla de los antecedentes de la animación y la sincronización de voz.

En el CAPÍTULO TERCERO se presenta una interfaz que permite modificar los músculos de una cara y boca para poder modelar expresiones y visemas.
En el CAPÍTULO CUARTO se describe el modelado de visemas a partir de los fonemas del español.

El CAPÍTULO CINCO explica la teoría necesaria para incorporar el sistema Festival y la biblioteca OpenAL a un modelo facial para poder producir voz a partir de un texto.

El CAPÍTULO SEIS incorpora toda la información de los capítulos anteriores para hablar de la sincronización del modelo facial y voz.

El CAPÍTULO SIETE muestra la implementación del sistema. Así como, la forma en que el programa fue construido modularmente, mostrando los diagramas descriptivos del mismo. Se utilizaron entre varios elementos, las bibliotecas de programación del lenguaje C, además del lenguaje Scheme que es una parte de festival y que permite manipular el texto para generar sonido, así como diferentes bibliotecas visuales como OpenGL, Glut para la interfaz de la animación y GTK para crear el editor de visemas y OpenAL para manipular el sonido.

Capítulo 2

Antecedentes

Esta sección es una breve sinopsis del tipo de desarrollos en el campo de la animación facial y la sincronización con la voz. Muchos trabajos en animación facial han sido publicados de una forma u otra. Los más populares han sido los procedimientos y notas de los cursos del congreso de ACM SIGGRAPH, seguidas por otras revistas de graficación por computadora relacionados y otras memorias de distintos congresos.

Históricamente, el primer modelo facial fue generado por Park como parte del curso en graficación por computadora de Ivan Sutherland en la Universidad de Utah a principios de 1970. Park comenzó con una representación poligonal de la cabeza, la cual resultó de una animación flip-pack (intercambio de imágenes) de abrir y cerrar sus ojos y la boca [Park72]. Mientras tanto en 1971 en la Universidad de Utah, Henri Gouraud estaba también completando su trabajo de disertación en su entonces nuevo algoritmo de suavizamiento de sombras para polígonos; para demostrar la efectividad de la técnica, aplicó esto a un modelo digitalizado de la cara de su esposa. Uso estas nuevas técnicas inovadoras de sombreado para producir varios segmentos de animación facial bastante realista. Hizo esto obteniendo los datos de los polígonos de las expresiones faciales de caras reales, usando técnicas de

medición fotográfica y simple interpolación entre expresiones para crear animación. Por otro lado, en 1971, Chernoff primero publicó su trabajo que usa la computadora para generar dibujos bidimensionales para representar un espacio de k dimensiones [Cher71]. Usando una representación gráfica simple de la cara, se derivó un esquema de la codificación detallado. También en 1973, Gillenson en la Universidad del estado de Ohio reportó su trabajo en un sistema interactivo para ensamblar y editar el dibujo de líneas de bidimensionales de imágenes faciales con la meta de crear un sistema de herramientas de equipo computarizado photoidenti [Gill74]. En 1974, Parke motivado por el deseo rápido de producir animación facial, completo la primer parametrización tridimensional del modelo facial [Park74] (figura 2.1).

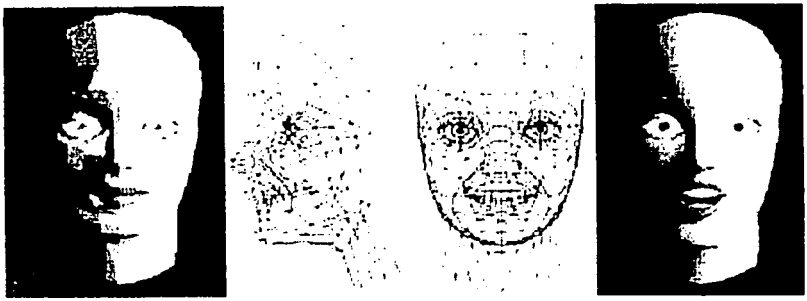


Figura2.1 Modelo de Park

De 1974 a 1976, se desarrollo por diferentes investigadores la animación facial 3D, la cual era esencialmente no interactiva. Sin embargo, durante este periodo, se desarrollaron sistemas de animación asistidos por computadora en dos-dimensiones en el Instituto Tecnológico de Nueva York, de la Universidad de Cornell, y más tarde, en Hanna Barbera. Estos sistemas soportaban animación de caricaturas en dos dimensiones incluyendo animación facial.

En 1975, Frederic I. Parke desarrolló un modelo para caras humanas que permitía la sincronización de la animación y la voz (figura 2.2).

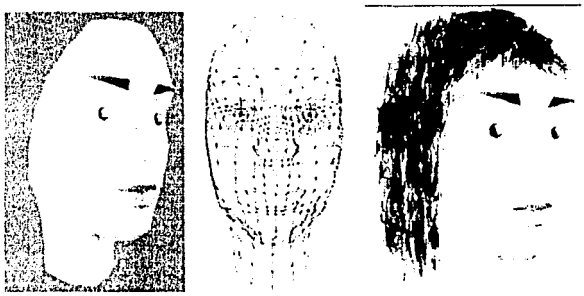


Figura 2.2 Modelo de sincronización de Park

En 1980, en la Universidad de Pensylvania, Platt publicó su tesis de maestría que consiste de un modelo de expresiones faciales controlado por músculos basado en física [Pla80]. En 1982, Brennam, en el MIT reportó su trabajo en técnicas de cómputo que producen caricaturas faciales de dos dimensiones [Bre82]. También en el MIT en 1982, Weil reportó un trabajo que usa un sistema de video basado en discos para seleccionar interactivamente y componer los rasgos faciales [Wei82]. Después, basado en este trabajo, Burson desarrolló técnicas basadas en computadora para envejecer imágenes faciales, específicamente imágenes de niños.

Como ejemplos de los desarrollos de animación facial de los ochentas, se tiene el cortometraje animado, Tony de Peltrie (ver figura 2.3), producido por Bergeron y Lachapelle en 1985, en cual fue un hito para la animación facial [BL85]. Este fue el primer cortometraje de animación por computadora donde las expresiones faciales y voz fueron una parte fundamental de la narración en la historia.

Tony White en 1986 estructuró el método tradicional de sincronizar la animación con una pista de voz. Tradicionalmente, la sincronización implicó grabar la voz en una cinta, bajo la observación de la interpretación fonética de las palabras cuadro por cuadro. Es esencial que la sincronización esté hecha correctamente mientras que los errores pueden significar tener

que hacer un rediseño de las secciones enteras de la animación. Este método de sincronización del habla es realizable, mientras uno tiene un guión y tiempo para elaborarlo.

Figura 2.3 Tony de Peltri

En 1987, Waters reportó un nuevo modelo muscular (ver figura 2.4) que se aproximaba a la animación de expresiones faciales. Este modelo permite una variedad de expresiones faciales a ser creadas por control de la musculatura que está debajo del rostro [Wat87]. En 1988, Magnenat-Thalmann y sus colegas describieron un modelo abstracto de acción muscular [MTPT88] (ver figura 2.5). En 1987 Lewis [LP87] y en 1988 Hill [PWWH86] reportaron técnicas para automatizar la voz y la animación facial.

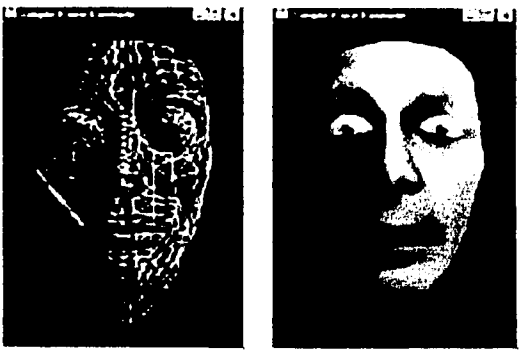


Figura 2.4 Modelo de Waters

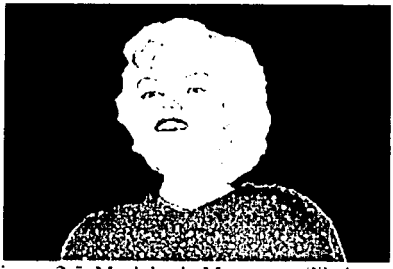


Figura 2.5 Modelo de Magnenat-Thalmann

En 1989, Wyvill y Hill presentaron, en un curso del congreso de ACM SIGGRAPH, la forma de como controlar expresiones usando síntesis de voz.

Una pequeña animación que sentó un gran precedente fue Tin Toy (ver figura 2.6), la cual recibió un premio de la academia de Hollywood en 1989. Fue producida por la compañía Pixar, y es un ejemplo de la capacidad de la animación facial por computadora. En particular un modelo muscular fue usado para articular la geometría facial de un bebé dentro de una variedad de expresiones [Park90].

Figura 2.6 *Bebe del cortometraje Tin Toy*

El desarrollo de escáner de rango óptico, tal como los escáneres tridimensionales ópticos Cyberware provee una nueva riqueza de datos para la animación facial [Cyb90]. En 1990, Williams reportó el uso de mapas de registros de texturas de imágenes faciales (ver figura 2.7 (a) y (b)) para la animación de expresiones faciales en 3D [Wil90].

(a)Escáner Cyberware

(b) Un rostro cilíndricamente escaneado

Figura 2.7

La nueva tendencia del escaneo y procesamiento de imágenes promete introducir un nuevo estilo de animación facial. En 1993, Lee, Terzopoulos, y Waters describen técnicas para trazar a los individuos en representaciones canónicas de la cara, donde sus atributos serán movidos con base a modelos basados en física [LTW93] (ver figura 2.8).

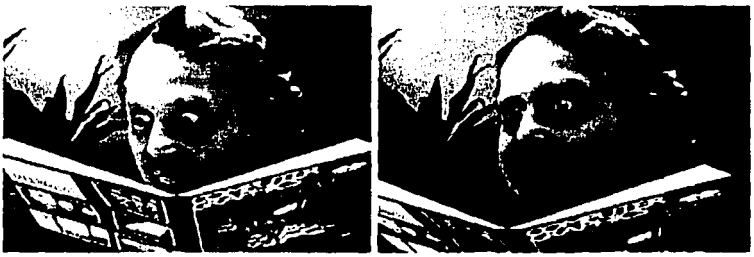


Figura 2.8 Modelo de Demetri Terzopolous

Otra gran área dentro de la animación por computadora es la medicina, donde hay un enfoque en procedimientos de planificación quirúrgicos. En 1988, Deng (y después Pieper en 1991) uso un modelo de elemento finito de tejido superficial para simular las incisiones superficiales y cierre de la herida [Den88].

Y por supuesto, un área que llama la atención es la animación de personajes creados en un ambiente de producción, dado que representa muchos desafíos para un animador: la gran variedad de proyectos presentan diversas necesidades y requisitos. Diversas empresas se han concentrado en la animación de personajes por muchos años, trabajando en como crear, desarrollar y manipular diferentes tipos de personajes en el mismo ambiente. Con cada nuevo personaje, el diseño del personaje, su estilo de animación, y el tamaño del proyecto son todos factores en desarrollo para sistemas de animación facial. Un sistema de animación facial para un personaje de dibujos animados con expresiones faciales complejas, no puede ser apropiado para un personaje que debe desplegar, expresiones humanas sutiles.

El proyecto de la película "Simpsons" involucró animar un personaje con distintas expresiones faciales, reconocibles y exageradas (ver figura 2.9). Además, con tiempo limitado, se tenía que desarrollar una solución en un horario relativamente corto. Se creo un sistema de capas basado en interpolación y comandos de deformación de alto nivel. Esto permitió al animador especificar las expresiones exageradas con relativa facilidad.

Figura 2.9 Homero Simpson

En tiempos recientes, ha surgido un gran interés en el análisis facial en la comunidad de visión por computadora. El interés tiene dos propósitos: el primero es proveer la habilidad para rastrear el rostro humano; el segundo es desarrollar la habilidad para detectar expresiones faciales y por eso derivar estados emocionales.

Capítulo 3

Modelado de Expresiones Faciales

El presente capítulo pretende ser una breve exposición sobre la forma de modelar expresiones faciales dentro de la animación por medio de la comprensión de los movimientos de la cabeza, en especial del rostro, ya que solo un gesto involucra la movilidad de la mandíbula y la lengua, la contracción de los ojos y la boca, etc.

La concientización de lo anterior permite modelar una geometría detallada del rostro seleccionando algún algoritmo y técnica disponible para proceder a construir su topología.

Esto se logra superponiendo los vectores musculares que son modelos matemáticos capaces de emular el músculo o la piel. Los cuales son el músculo esfínter (para la boca y ojos) y el músculo lineal.

La habilidad para modelar la cara humana y después darle animación permite el manejo y representación de varias expresiones faciales, lo cual establece un desafío interesante en la graficación por computadora. La animación facial ha progresado significativamente en pocos años y una variedad de algoritmos y técnicas disponibles como la Interpolación, permiten

generar los cuadros intermedios necesarios para generar una animación determinada como se explicará en el presente capítulo.

3.1 Anatomía de la cabeza

El cráneo es la parte ósea principal conformada por catorce huesos de los cuales la mandíbula es la única parte que tiene la libertad de movimiento. Los huesos del cráneo se dividen en dos partes principales: el cráneo que (ver figura 3.1) es esencialmente una cubierta protectora para el cerebro y el esqueleto facial que establece la definición personal de un rostro . El cráneo, a su vez se divide en base calvaria y base craneal, donde la base calvaria se encuentra en la base superior de la cubierta del cerebro y la base craneal en la parte inferior, ambas conformadas por varios huesos.

Figura 3.1 Diferentes vistas del cráneo

El esqueleto facial se encuentra por debajo y anterior a la base craneal integrada como se muestra en la figura 3.2. El tercio superior del esqueleto facial consiste en las órbitas y huesos nasales, el medio tercio de las cavidades nasales y maxilares, y el más bajo tercio consiste en la región de la mandibular.

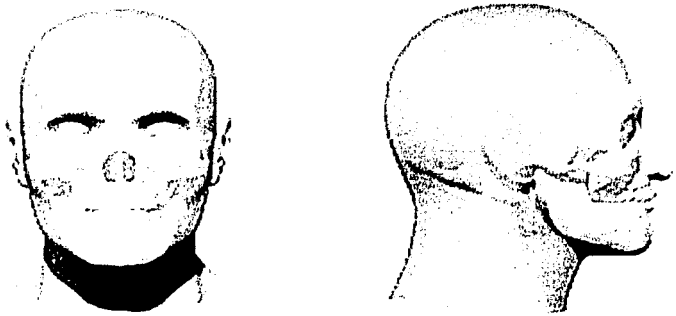


Figura 3.2 Diferentes vistas de la piel y del esqueleto facial integrados

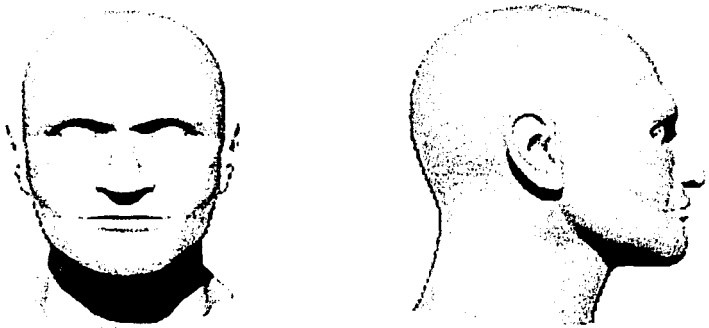


Figura 3.3 Diferentes vistas de la piel en una cabeza

El esqueleto facial es de particular interés en el modelado en tres dimensiones ya que proporciona el marco en el cual los músculos y la piel se encuentran, como se muestra en las figuras 3.3 y 3.4.

El estudio de los músculos resulta de gran importancia porque juegan un papel privilegiado en la comunicación dado que ellos son los que permiten mover a la piel y como muestra de ello se puede mencionar que los principales son: músculos de la boca, músculos de los ojos, músculos de la nariz y los músculos de la mandíbula; mismos que se muestran en la figura 3.4 (a) y se señalan en la figura 3.5.

(a) (b) (c)

Figura 3.4

La piel (figura 3.4 (b) y (c)) está compuesta por dos capas, la dermis y la epidermis, cubriendo ésta última a la primera. La epidermis es una capa de células muertas que protege a la dermis de los elementos del medio ambiente. Debajo de la piel se encuentra una capa de grasa subcutánea, y debajo de esta grasa, está la fascina, la cual es un tejido fibroso conectado al músculo y cartílago de la piel. La elasticidad de la piel se debe al colágeno (72%) y la fibra elastina (4%), lo cual hace que la piel sea flexible.

Cabe destacar que el tratamiento de los músculos de la boca tienen la interacción más compleja en el rostro, dado que existe un grupo de estos músculos que realizan el proceso de abrir los labios, llamados radiales, los cuales se dividen en músculos radiales del labio superior e inferior, y otro grupo que se encarga del proceso de cerrarlos formado por el

orbicular de los labios u obicularis oris el cual es un músculo esfínter sin hueso como liga, el depresor del ángulo de la boca y el depresor del labio inferior

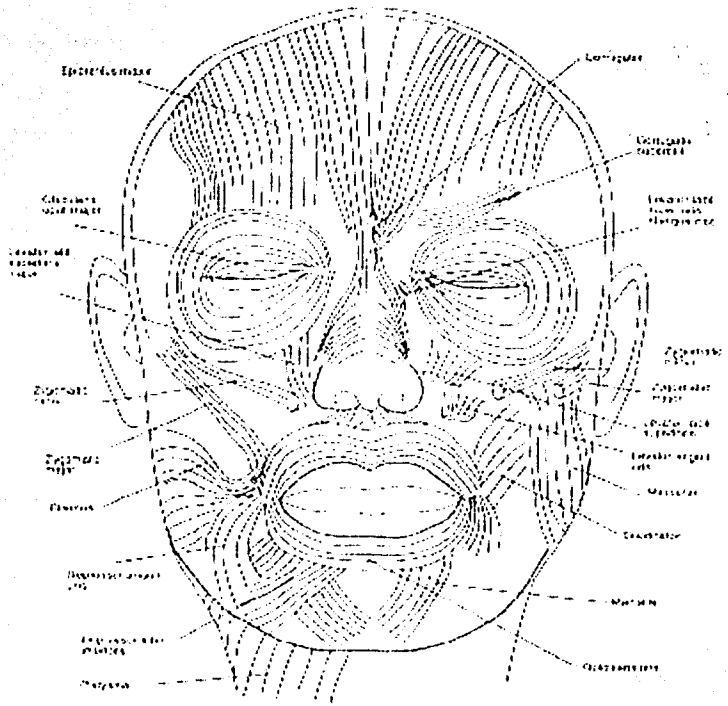


Figura 3.5 Descripción de los músculos faciales

La base principal que permite trabajar con los músculos faciales en la animación por computadora radica en determinar que una de las terminaciones de un músculo lineal de la cara tiene una liga al hueso, que permanece estática, mientras que la otra terminación se encuentra en el tejido de la piel.

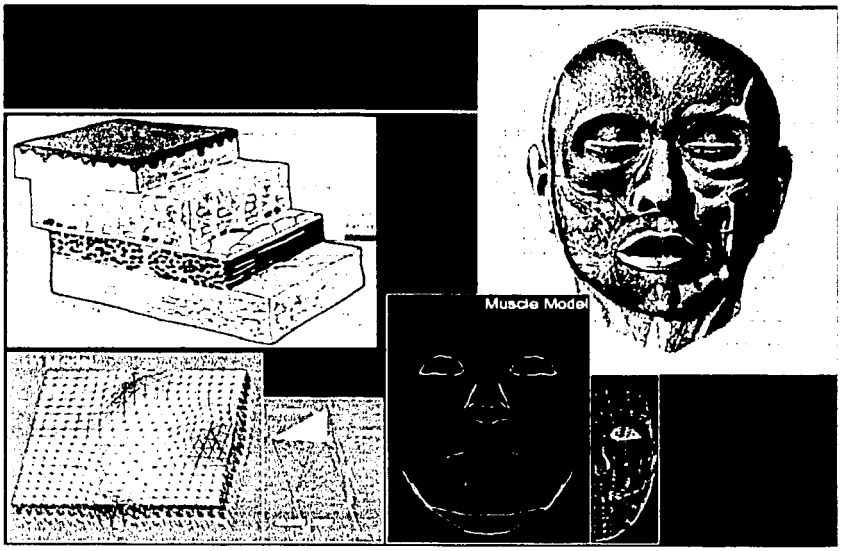


Figura 3.6 Anatomía Facial

En resumen, la anatomía de la cara y la cabeza es un complejo ensamble dinámico de huesos, cartílago, músculos, nervios, vasos sanguíneos, glándulas, tejido grasoso y piel; y para modelar el rostro humano solo se requiere del conocimiento de la anatomía del esqueleto, los músculos y la piel que se apoyan en técnicas y algoritmos para especificar y controlar el diseño y movimiento faciales, mismos que se muestran en la figura 3.6 y se explicarán más adelante.

3.2. Geometría facial

El objetivo de las diversas técnicas de animación del modelo facial, en determinado tiempo, es la de crear y manipular una imagen final a partir de una base de datos con información de una malla compuesta de polígonos y una imagen mapeada como textura sobre la malla.

Ante esta situación surge la pregunta: ¿cómo representar geométricamente una cara y una cabeza que permitan al mismo tiempo ser una animación efectiva y que tenga un renderizado eficiente?

En la actualidad existen dos tipos de modelado, para esto:

- Representación volumétrica
- Representación de superficie

3.2.1 Modelado basado en representaciones volumétricas para imágenes médicas

Este tipo de representación construye sólidos geométricos (CSG), arreglos de elementos de volumen (*voxel*) y agrega variantes jerarquizados de *voxeles*, tales como *Octrees*.

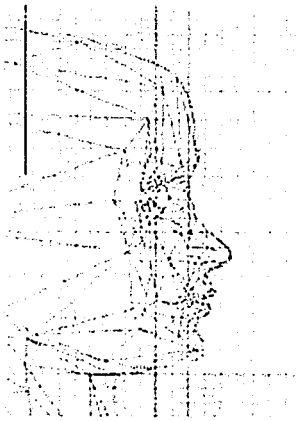
CGS es usado en la actualidad para representar estructuras anatómicas e imágenes médicas (técnicas de resonancia magnética). La ventaja de este método es que se pueden realizar cortes de datos de dos dimensiones en estructuras de tres dimensiones. Sin embargo, las animaciones de este tipo tienen un problema, las rotaciones son lentas, dada la gran cantidad de información que se tiene que mover.

3.2.2 Modelado basado en representación de superficies

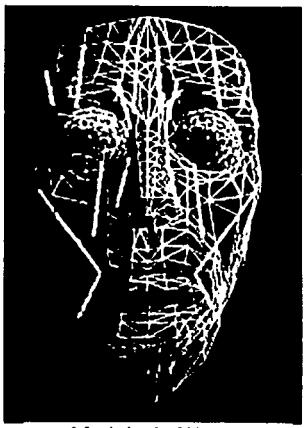
Este tipo de representaciones requiere de superficies y estructuras que utilizan geometría básica para modelos faciales. Es decir, generar una superficie de la piel como una malla donde cada nodo tiene un grado finito de movilidad. Este tipo de estructuras de superficie se

usan actualmente para realizar animación facial, porque permiten cambiar la forma de la superficie dependiendo de la necesidad de la animación. Las técnicas descritas de superficie posibles incluyen superficies implícitas, superficies paramétricas, y superficies poligonales. Las superficies paramétricas incluyen Bézier, Catmull-Rom, Beta-spline, B-spline, B-spline jerárquico y superficies de NURBS [BBB87]. Las superficies poligonales son las más usadas en el área de la animación interactiva, dado que incluyen mallas poligonales regulares y arbitrarias.

Los sistemas gráficos tienden en su mayoría a desplegar superficies poligonales y pueden actualizar los modelos faciales complejos casi en tiempo real. La mayoría de modelos faciales existente están basados en descripciones poligonales. Estas superficies poligonales pueden estar en la forma de mallas de polígonos regulares o como mallas arbitrarias de polígonos conectados como se muestra en la figura 3.7 y la figura 3.8.



Modelo de Parke
Figura 3.7



Modelo de Waters
Figura 3.8

3.3 Desarrollo de una topología poligonal

En este contexto, una topología se refiere a cómo los polígonos están conectados para formar una superficie. Por ejemplo, la topología de la malla regular organiza los vértices del polígono dentro de arreglos rectangulares. Estos vértices son entonces conectados con polígonos triangulares o cuadriláteros formando la superficie deseada.

Existen un número de pasos importantes a considerar al modelar un rostro humano (ver figura 3.9). Estos fueron considerados por Gouraud [Gou71] y son los siguientes:

- Los polígonos deben ser colocados de tal manera que permitan cierta flexibilidad y cambios naturales de la forma del rostro. Si los polígonos no son triángulos, deberían permanecer aproximadamente planos sin dobleces tal y como lo requiere un rostro real.

- Los polígonos deben aproximarse a la cara para cada expresión. La topología de los polígonos puede necesitar modificarse interactivamente hasta que permita representaciones razonables del rostro humano en todas sus posibles expresiones. Los parpados y los labios requieren una atención especial para asegurar que puedan abrir y cerrar naturalmente.

- La cantidad de información (datos 3-D) que se requiere para definir a la superficie de una cabeza debe ser distribuida de acuerdo a la curvatura de dicha superficie, es decir, las áreas de alta curvatura en la superficie como son la nariz, la boca, los alrededores de los ojos, y la orilla de la barbilla, necesitan una gran cantidad de información para una mejor definición y un control sutil de la superficie. En cambio las áreas de curvatura baja (la frente, las mejillas y el cuello) necesitan menos cantidad de información.

- Usar el mínimo número posible de polígonos para proporcionar buenos resultados con el objetivo de tener una aceptable precisión de la forma de la superficie. Esto es

posible porque un número pequeño de polígonos permite manejar menos información en la memoria de la computadora, lo cual permite generar imágenes rápidamente.

- Las orillas de los polígonos deben coincidir con los pliegues del rostro. En particular se deben modelar los polígonos que se encuentran bajo los ojos, los lados de la nariz, el borde de los labios, y las comisuras de los labios, cuidando que los polígonos no generen pliegues para permitir un mayor realismo.

- Es necesario un cuidado especial si los pliegues tienen que ser visibles en las superficies sombreadas. Para ello se requiere mantener las normales separadas de los vértices a lo largo del pliegue. Esta acción asegura que habrá una discontinuidad sombreada a lo largo del pliegue, y que el pliegue será visible.

- Dado que la cara es casi simétrica, se puede elegir para modelar solo un lado de la cara. El otro lado puede ser tomado por una reflexión en el plano de simetría. Este enfoque no es recomendado cuando los modelos requieren de precisión según el deseo del animador.

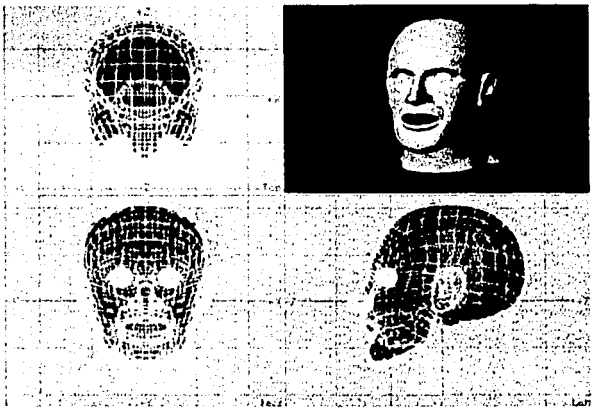


Figura 3.9 Modelando la superficie de una cabeza

3.4 Características de la cara

Un modelo facial es usualmente la suma de muchas partes y detalles. Gran parte de la investigación en animación facial está enfocada a técnicas para definir la geometría y manipular la máscara facial. Una completa aproximación es integrar a la máscara facial características faciales detalladas, y cabello en modelos de una cabeza completa.

La cara debería ser vista como una parte integrada de una cabeza completa y, a la vez, como una parte de un personaje sintético completo. Un modelo de cabeza completo incluye la cara, los oídos, el cabello y el cuello. La habilidad para especificar la forma de la cabeza, la conformación de los oídos, el cabello y el estilo del cabello es importante en la representación de caras de personas específicas.

3.5 Adquisición de datos para la superficie facial

Un objetivo básico de modelado es crear una descripción que represente fielmente la cara o las caras a partir de primitivas geométricas. Para especificar un cambio significativo dentro de una superficie tridimensional de una cara o algún objeto complejo se han creado cuatro aproximaciones generales [Frederic96] las cuales son usadas para determinar la forma de la superficie para modelar la cara:

Figura 3.10

- Técnicas de medición de superficie tridimensional: pueden realizarse usando digitalizadores, técnicas de medición de fotografías o técnicas de escáner con láser. Por ejemplo, podríamos dibujar una malla de polígonos en un modelo, y después digitalizar el modelo utilizando escáneres láseres, como se

muestra en la figura 3.10.

- Esculpido y ensamblado de componentes de superficies interactivas pueden crearse usando operaciones análogas a aquellos usados por los escultores, donde se incluyen métodos de ensamblaje de esculturas de varios componentes e interactúan modificando las formas de la superficie de la escultura (como la forma en que se crearon los personajes de la película de *chicken run* [pollitos en fuga]).

- Las directrices de la estructura de una cabeza se basa en el trabajo de Kunz[Kun89], Halton, y Hogarth es usado en el diseño de modelos de caras. Estas directrices son generalizaciones de diferentes componentes de una cabeza, dado que existen considerables variaciones de un individuo a otro. Es importante, tener ciertas pautas para modelar una estructura como la cabeza humana.

- Creación de nuevas caras a partir de caras existentes, aplicando deformaciones por medio de Interpolación entre caras individuales especificas.

3.6 Animación facial

Muchas personas conocen que la animación es elaborada por una serie de imágenes estáticas que son cambiadas rápidamente ante un observador, quien, detecta los cambios de la imagen, al leer el resultado con una secuencia de movimiento (ver figura 3.11). La animación usa una limitante de nuestro ojo, la cual es pasar varios cuadros (frames) a una velocidad de 30 frames por segundo.

Frame 1 **Frame 2** **Frame 3** **Frame 4** **Frame 5**

Figura 3.11 Secuencia de frames dentro de una animación tradicional

El objetivo de las diversas técnicas de animación es manipular la superficie de la cara para que tenga la expresión deseada en cada *frame* (cuadro) de las secuencias de la animación. Este proceso implica directamente o indirectamente la manipulación de la superficie de polígonos y vértices con el tiempo.

Figura 3.12 Interfaz realizada en GLUT para generar la Animación

Hay por lo menos cinco aproximaciones fundamentales para realizar animación facial (ver figura 3.12 y anexo C). Estas son:

- La interpolación.

- La captura de movimiento (performance-driven).
- Parametrización directa.
- Pseudo-músculos
- Músculos vectoriales.

Interpolación

La *interpolación* es tal vez la más usada de las técnicas. En su forma más simple, corresponde a una aproximación *key-framing* (cuadro-principal) que se encuentra en la animación convencional. La idea detrás del *key-frame o key-pose* (pose-principal) animado es que la expresión facial deseada se especifique para ciertos puntos en el tiempo y un algoritmo de computadora genere los *frames* que están entre los *key-frame*. La animación *key-pose* requiere especificaciones completas del modelo geométrico para cada expresión facial. Esta especificación requiere una labor muy intensa de aproximación.

Captura de movimiento

La animación por *captura de movimiento* involucra la medición de acciones humanas reales. Los datos de los dispositivos de entrada interactivos como menciona Waldos [deG89], los datos de guantes, sensores de movimiento en el cuerpo, o escaneo láser y sistemas de video basado en el rastreo movimiento se usa para controlar la animación.

Parametrización directa

En el modelo de *parametrización directa* [Par 74], el conjunto de parámetros usados para definir modelos usa regiones de interpolación local, trasformaciones geométricas, y técnicas de mapeo para manipular las características de la cara como se muestra en la figura 3.7.

Pseudo-músculos y músculos vectoriales

Con la animación facial *basada en pseudo-músculos*, Platt y Badler [PB81] usan un modelo de *masa-y-resorte (mass-and-spring)* para simular músculos faciales. Waters [Wat87] desarrolló un modelo facial que incluye dos tipos de músculos: músculos lineales que jalan y músculos esfínter que aprietan, como se muestra en la figura 3.8. Los músculos de Waters tienen propiedades de dirección (vector). En el caso de Platt y Badler, los músculos usan un modelo de *masa-y-resorte* para la piel y losmúsculos.

Terzopoulos y Waters [TW90] aplican técnicas de modelado físico. Las acciones musculares de la cara están modeladas por simulación de propiedades físicas de la piel que afectan las expresiones de la cara, incluyendo los músculos.

3.6.1 Animación basada en músculos vectoriales

Hoy día no se ha reportado en los modelos de animación facial detalles anatómicos completos. Sin embargo, algunos modelos han sido desarrollados basándose en modelos simplificados de estructuras de huesos faciales, músculos, tejido adiposo, y piel. Estos modelos proveen la habilidad para manipular las expresiones faciales basadas en la simulación de los músculos faciales y tejido facial.

Platt y Badler [PB81] inicialmente desarrollaron un modelo facial dinámico en el cual los vértices de un polígono en la superficie de la cara (piel) tuvieran elasticidad interconectada con el modelo de resorte. Estos vértices también fueron conectados a la subyacente estructura ósea del modelo usando músculos simulados. Estos "músculos" tienen propiedades de elasticidad y pueden generar fuerzas de contracción. Las expresiones faciales fueron manipuladas por aplicando fuerzas al músculo para la conexión elástica de la malla en la piel.

Waters desarrolló un modelo dinámico facial que incluye dos tipos de músculos: Músculos lineales y músculos esfínter [Wat87]. Los músculos de Waters tienen propiedades direccionales (vector) que son independientes de la subyacente estructura del hueso. Estos vectores hacen al modelo muscular independiente de una topología facial específica. Cada músculo tiene una zona de influencia. La influencia de un músculo particular es disminuida como una función de distancia radial de los puntos adjuntos al músculo.

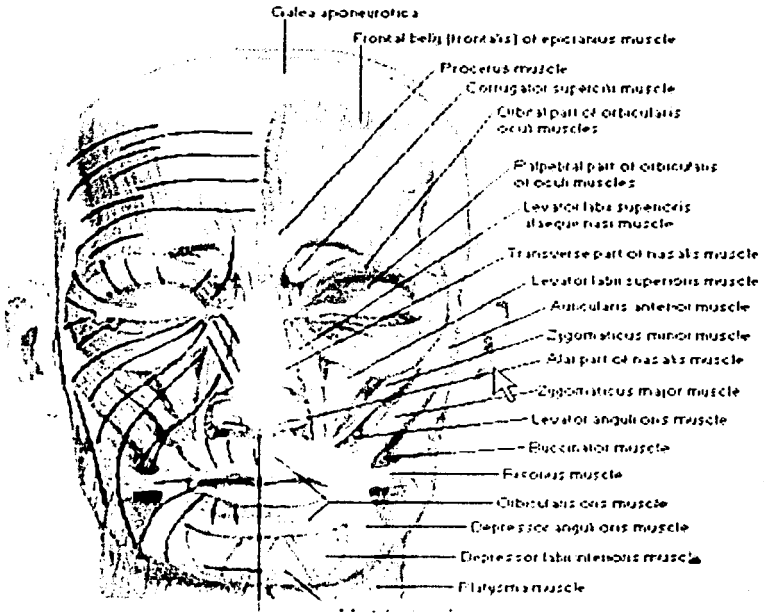


Figura 3.13 Músculos Faciales

Waters, Platt y Badler realizaron un análisis a partir de la anatomía humana (figura 3.14), denotaron que de los 200 músculos (figura 3.13) que influyen a la cara se distinguen los siguientes:

- **18 Músculos lineales/paralelos:** curveados o planos con solo dos puntos de unión.
- **5 Músculos hoja/limitados:** planos y ancho, y no contienen sólo dos puntos de unión.
- **3 Músculos esfínter:** alrededor de los orificios del cráneo.

Los principales motivadores de la expresión facial de un músculo real son:

- La inserción al hueso
- La unión a la piel
- La contracción y estiramiento de la piel hacia la unión del esqueleto.

Figura 3.14. Modelo muscular

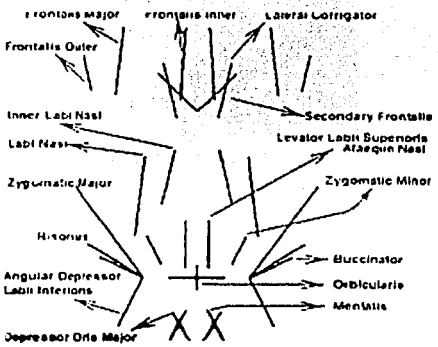
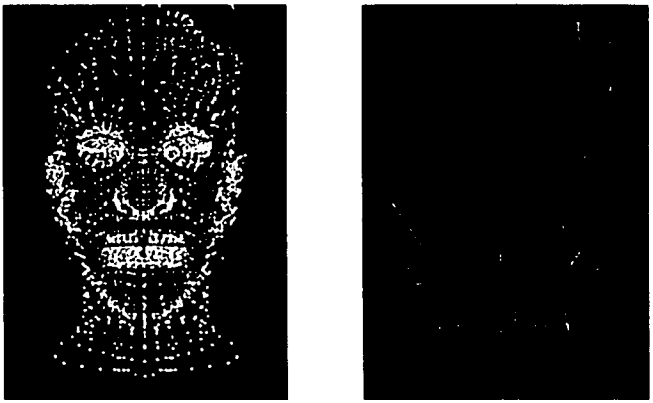


Figura 3.15. Modelo abstracto

A partir de las consideraciones anteriores, se requiere poder determinar el lugar y la posición de los músculos abstractos (ver figura 3.15) en un espacio tridimensional. Para ello, se utiliza

un editor que permite manipular con precisión los polígonos donde se mueven dichos músculos (ver figura 3.16 (a)) donde se muestran como líneas (ver figura 3.16 (b)) que representan los músculos lineales abstractos en la cara y rombos que representan músculos esfínteres tanto de la boca como de los ojos.



(A) (B)

Figura 3.16 Músculos abstractos colocados por medio de un editor

3.6.2 Modelos musculares

Los modelos musculares son consideraciones biomecánicas mímicas del músculo aplicándoles distorsión geométrica. La base principal de muchos de los músculos faciales estriba en determinar que una de las terminaciones de un músculo lineal tiene una liga al hueso, la cual permanece estática, mientras que la otra terminación se encuentra en el tejido suave de la piel.

Para definir un modelo matemático muscular se debe ignorar algunos de los atributos físicos y debe simplemente imitar las características primarias de los desplazamientos del tejido

facial mediante una función geométrica de distorsión, siendo este proceso más rápido computacionalmente.

A partir de lo anterior, encontramos que existen dos tipos primarios de músculos: el lineal y el esfínter. Dichos músculos son descritos como vectores músculo, como su nombre lo sugiere, siguiendo la dirección e inserción de fibras del músculo. Considerando que el músculo real consiste de muchas fibras individuales, el modelo matemático debe asumir una única dirección y un lugar fijo. Con esta simple suposición, un músculo puede ser descrito con dirección y magnitud en dos o tres dimensiones: la dirección es hacia un punto de atadura en el hueso y la magnitud del desplazamiento depende de la constante de resorte del músculo y la tensión creada por la contracción del músculo.

Como se puede observar, cuando un músculo se mueve, ocurre un desplazamiento en la piel. Dicho desplazamiento contiene una zona de influencia mínima y máxima alrededor del músculo, como se explica a continuación.

3.6.3 Músculo lineal

En dicha figura se muestra el músculo lineal el cual básicamente realiza una contracción y estiramiento de un punto en 3-D (de inserción en el **hueso**) hacia otro punto 3-D (de unión en la **piel**). Dicho desplazamiento contiene una zona o **ángulo** de influencia mínima y máxima alrededor del músculo y una distancia de **inicio** y otra de **fin** de la contracción, así como un valor de escalamiento **clampv** (figura 3.18). Dicha información se encuentra en la tabla 3.1.

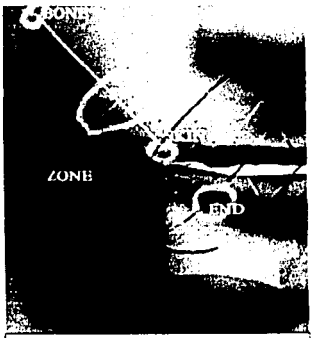


Figura 3.18 Músculo Lineal

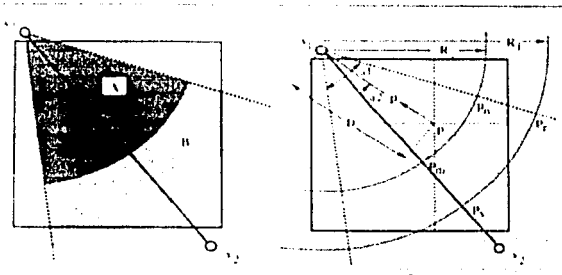
El músculo lineal básicamente realiza la contracción y estiramiento de un punto de intersección en el hueso hacia otro de unión en la piel. Para esto, es necesario calcular el desplazamiento del nodo **p** (figura 3.17 (b)), afectado por una contracción de un vector muscular. Se asume que no hay desplazamiento al punto de inserción en el hueso, y que la máxima deflexión ocurre al punto de inserción dentro de la piel. Consecuentemente, una disipación de la fuerza es pasada al tejido colindante cruzando los sectores **A** y **B** en la figura 3.17 (a).

Calcular el desplazamiento de un nodo arbitrario **p** en la figura 3.17 (b), localizada en la malla a un nuevo desplazamiento del nodo **p'** con el segmento $v_1p_1p_0$ hacia el nodo v_1 consigo el vector p, v_1, la siguiente expresión es utilizada:

$$p' = p + akr \frac{pv_1}{\| pv_1 \|} \qquad (1.1)$$

$$a1 = \cos(a2), \qquad (1.2)$$

Aquí la nueva ubicación del modo p' es una función de un parámetro de desplazamiento angular



(a) (b)

Figura 3.17 Descripción geométrica del Músculo Lineal

Donde $a2$ es el ángulo entre los vectores (v_1, v_2) y (v_1, p), D es $\|v_1\text{-}p\|$, r un parámetro de desplazamiento radial.

$$r = \begin{Bmatrix} \cos\left(\frac{1-D}{R_s}\right); & para\ \ p\ \ dentro\ \ del\ \ sector\ \ (v_1 p_n p_m v_1) \\ \cos\left(\frac{D-R_s}{R_t-R_s}\right); & para\ \ p\ \ dentro\ \ del\ \ sector\ \ (p_n p_r p_s p_m) \end{Bmatrix} \qquad (1.3)$$

Y k una constante fija que representa la elasticidad de la piel. Para variar el factor de contracción del músculo, el desplazamiento de la piel aparece moviéndose a lo largo del eje principal del músculo hacia la raíz del vector como se muestra en la figura 3.18.

Tabla 3.1 Biblioteca de datos con información de 18 músculos lineales,abstractos

	Nombre	Piel			Hueso			Angulo	inicio	final	clampv
1	Left Zygomatic Major	-1.57	-10.8775	1.4208	-3.405	-9.1575	3.8728	2.3	132.0	22.0	0.9
2	Right Zygomatic Major	1.57	-10.8775	1.4208	3.405	-9.1575	3.8728	2.3	132.0	22.0	0.9
3	Left Angular depressor	-1.57	-10.8775	1.4208	-2.059	-8.4075	-0.5291	0.3	151.0	55.0	1.33
4	Right Angular Depressor	1.57	-10.8775	1.4208	2.059	-8.4075	-0.5291	0.3	151.0	55.0	1.33
5	Left Frontalis Inner	-0.4869	-10.7575	7.0798	-0.3539	-9.9575	9.6878	0.1	100.0	35.0	0.5
6	Right Frontalis Inner	0.4869	-10.7575	7.0798	0.3539	-9.9575	9.6878	0.1	100.0	35.0	0.5
7	Left Frontalis Major	-2.381	-10.4575	7.2708	-2.244	-9.8275	9.8038	0.3	120.0	65.0	2.0
8	Right Frontales Major	2.381	-10.4575	7.2708	2.244	-9.8275	9.8038	0.3	120.0	65.0	2.0
9	Left Frontalis Outer	-2.9	-9.5251	7.1108	-3.139	-9.0835	9.0888	0.3	141.0	50.0	0.6
10	Right Frontalis Outer	-3.1598	-9.5254	7.1108	3.3211	-8.8055	9.4535	0.5	141.0	50.0	0.6
11	Left Labi Nasi	-1.776	-10.9348	2.2322	-1.56	-10.387	5.0186	0.1	133.0	35.0	0.25
12	Right Labi Nasi	1.776	-10.9348	2.2322	-1.56	-10.387	5.0186	0.1	133.0	35.0	0.25
13	Left Inner Labi Nasi	-1.002	-11.5195	4.2238	-0.6379	-10.718	5.5248	0.1	200.0	35.0	1.0
14	Right Inner Labi Nasi	1.002	-11.5195	4.2238	0.6379	-10.718	5.5248	0.1	200.0	35.0	1.0

15	Left Lateral Corrugator	-1.2932	-10.8378	8.1163	-0.1232	-11.08	6.7203	0.5	163.0	45.0	0.5
16	Right Lateral Corrugator	1.3180	-10.8378	8.1410	0.1480	-11.08	6.7450	0.5	163.0	45.0	0.5
17	Left Secondary Frontales	-0.5839	-10.8845	6.4928	-0.0909	-10.922	8.3028	0.1	130.0	45.0	1.0
18	Right secondary Frontales	0.5840	-10.8845	6.4928	0.0910	-10.922	8.3028	0.5	130.0	45.0	1.0

El músculo lineal tiene algunas variantes, dependiendo de donde se encuentre el músculo en la cara. Los músculos son: el músculo asociado a la mejilla, el músculo asociado a los labios y el músculo asociado a la frente como se explican a continuación:

(a) Músculo asociado a la mejilla

Este músculo se muestra en la figura 3.19, y se deforma a lo largo de cada una de las normales de los vértices basados en la distancia del **centro** del músculo al **final** de la contracción del mismo. Principalmente se consigue inflar las mejillas. La información que define a los dos músculos asociados a las mejillas se encuentra en la tabla 3.2.

Figura 3.19 Diferentes vistas del músculo asociado a la mejilla

	nombre	Piel			Hueso			Angulo	Inicio	final	Clamps
19	Right Cheek	1.57	-10.8775	1.4208 7	3.405	-7.1575	1.5408 6	0.3	150.0	55.0	1.1
20	Left Cheek	-1.57	-10.8775	1.4208 7	-3.405	-7.1575	1.5408 7	0.3	150.0	55.0	1.1

Tabla 3.2 Biblioteca de datos con información de 2 músculos lineales asociados a las mejillas

(b) Músculo asociado a los labios

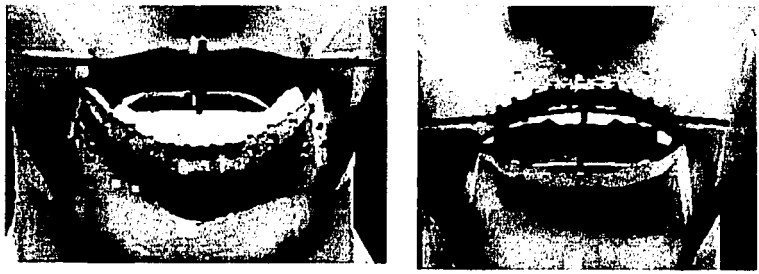
Este músculo se asocia al movimiento de los labios como se muestra en la figura 3.20 el cual sirve para mover tanto el labio inferior (*Orbes*) como el labio superior (*Mentails*). Se agrega un algoritmo transversal al músculo lineal para mover el labio superior e inferior. Esto permite mover los labios independientemente. Para esto se requiere un valor de escala (*clampv*), la longitud del desplazamiento (*pathlen*) que comienza en el punto de unión de la **piel** y un factor de escala (*xfactor*) en el eje "x" para aplicar el algoritmo trasversal y poder desplazar un labio. La información que define a los dos músculos se encuentra en la tabla 3.3.

El algoritmo trasversal lo que hace es:

- Encontrar el punto más cercano
- Agregar la dirección "xfactor" a los puntos dentro de la zona (ángulo) de influencia
- Si los vértices son los mismos, no se eliminan las direcciones a esos puntos.
- Eliminar las direcciones a los puntos usados.
- Agregar el punto para etiquetar la lista.
- Usar las nuevas direcciones como raíz de una nueva búsqueda
- Salir cuando no encuentre una dirección de salida

	nombre	piel			hueso			inicio	final	zona	clamp	Pathlen	xfactor
21	Orbes	4.07e0 3e-007	-10.857	2.9208 7	3.81423 e-007	-11.957	1.24	1.0	120	45.0	1.1	300.0	0.2
22	Mentails	2.5306 5e-007	-10.957	-0.7591	3.47064 e-007	-11.157	1.24	0.3	120	55.0	1.1	300.0	0.2

Tabla 3.3 Biblioteca de datos con información de 2 músculos lineales asociados a los labios



(a) Orbes (b) Mentails

Figura 3.20 Músculos asociados a los labios

(c) Músculo asociado a la frente

El músculo asociado a la frente se muestra en la figura 3.21 se basa en el músculo *asociado a los labios*. La principal diferencia es que el músculo asociado a los labios comienza en la trasversal del vértice más cercano al punto de inserción del **hueso,** mientras que el músculo asociado a la frente comienza en el punto unión de la **piel.** La información que define a dicho músculo se encuentra en la tabla 3.4.

Figura 3.21 Músculo asociado a la frente

	nombre	piel			hueso			inicio	final	cuspo	clampy
23	forehead	-0.1001478	-11.15	7.5906	0.0121	-10.487	9.97	0.1	132	70.0	1.1

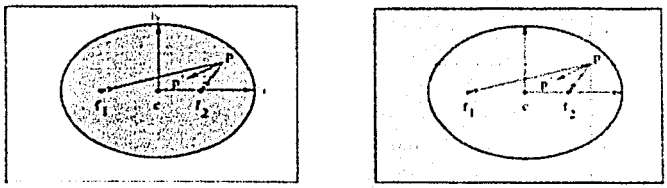
Tabla 3.4 Biblioteca de datos con información del músculo lineal de la frente

3.6.4 Músculo esfínter

Para el músculo esfínter se requiere de la contracción de un punto e que jugara como centro de la elipse. Como un resultado, la superficie alrededor de la boca se arrastra y tiende a apretarse. Esencialmente, el músculo esfínter es elíptico en apariencia, y puede ser simplificado a un elipsoide parametrizado con un eje mayor y menor, *lx* representa el eje semimayor, y *ly* el eje semimenor cerca de un epicentro imaginario con se ilustra en la figura 3.22 (a)

Para calcular el desplazamiento de un nodo de la posición **p** a la posición **p'** en la figura 3.22 (b), la siguiente ecuación es usada:

$$f = 1 - \frac{ly^2 p_x^2 + lx\ p_y^2}{l_x l_y} \qquad (1.4)$$



(A) **(B)**

Figura 3.22 Descripción geométrica de las zonas de influencia del músculo esfínter

La figura 3.23 ilustra la contracción en una zona (radx, rady, radz) de influencia de un músculo esfínter con un incremento en el factor de contracción en el eje *x* (facx) y en el eje *y* (facy).

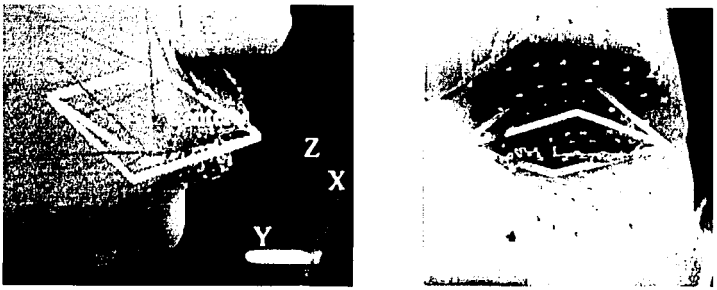


Figura 3.23 Músculo esfínter asociado a la boca y al ojo

En dicha figura se muestra el musculo lineal el cual básicamente realiza una contracción y estiramiento de un punto en 3-D (de inserción en el **hueso**) hacia otro punto 3-D (de unión en la **piel**). Dicho desplazamiento contiene una zona o ángulo de influencia mínima y máxima alrededor del músculo, una distancia de comienzo y finalización de la contracción, así como un valor de escalamiento (clampv), y la información que define a dicho músculo se encuentra en la tabla 3.5.

#	nombre	Centro del vector muscular			Zona de influencia				Factor de influencia		
					radx	rady	radz	dir	facx	facy	posliz
24	mouth	3.5762 8e-007	-10.735	1.3298	2.44	2.9	0.95	2	1.0	-0.2	0.0
25	Right eye	-1.9	-9.3608	6.432	1.5	2.01	1.76	2	0.0	-0.1	0.25
26	Left eye	1.9	-9.3608	6.432	1.5	2.01	1.76	2	0.0	-0.1	0.25

Tabla 3.5 Biblioteca de datos con información de 3 músculos esfínteres

3.7. Modelado de expresiones faciales

Una expresión es una postura particular del rostro. Las expresiones faciales permiten identificar los procesos mentales cuando estos están ocurriendo. El proceso de análisis de expresiones puede también contribuir al entendimiento para asociar una emoción con una determinada expresión facial. Asimismo, las expresiones faciales prometen exitosas

aplicaciones en la investigación médica, especialmente en los casos de análisis del estado psicológico del paciente.

Determinadas posturas del rostro humano son generadas a partir de diferentes investigaciones hechas por psicoanalistas de comunicación que han establecido una categorización de expresiones faciales primarias que han considerado ser genéricas para el rostro humano. Ekman, Friesen y Ellsworth [EFE72] proponen la felicidad, el enojo, el miedo, la sorpresa, el disgusto, y la tristeza como las seis categorías primarias como se muestra en las figuras 3.25 y 3.26. Otras expresiones más complejas, tales como el interés, la calma, la ironía, la inseguridad, el escepticismo, etc. son generadas a partir de la unión o suma de expresiones primarias.

Una expresión se obtiene combinando la deformación de la geometría del modelo facial, para lograr esto se propone un editor de expresiones faciales que permita modificar o variar los valores de los parámetros de los músculos involucrados, como se explica más adelante

3.7.1 Diseño de una interfaz para el modelado facial

Para realizar el modelado de expresiones faciales en esta tesis se diseño e implemento un editor de visemas y expresiones faciales (ver figura 3.24) que a partir de la modificación de los valores de los músculos que son los principales motivadores de la expresión facial, se modela tanto la boca, como las expresiones de la cara de forma sencilla y amigable. Para la creación del editor se buscó la portabilidad entre plataformas, que permitiera a un programador mejorar la interfaz de forma sencilla con las bibliotecas gráficas OpenGL y el lenguaje C.

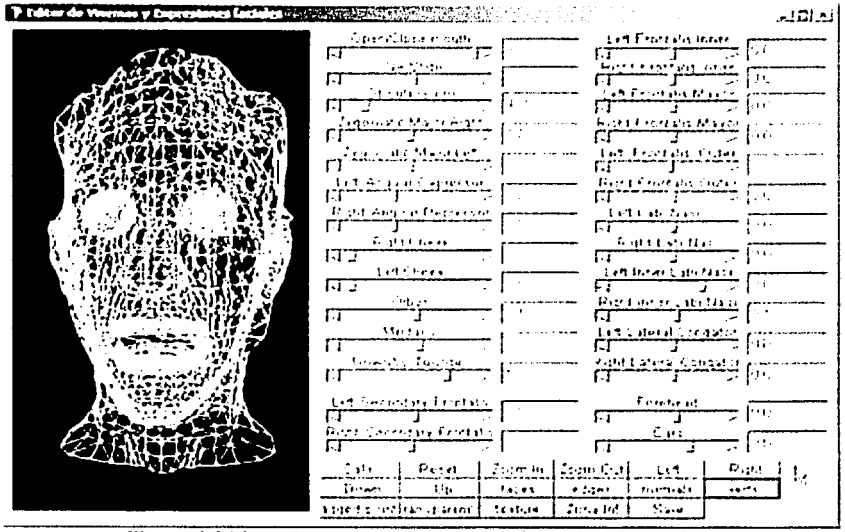
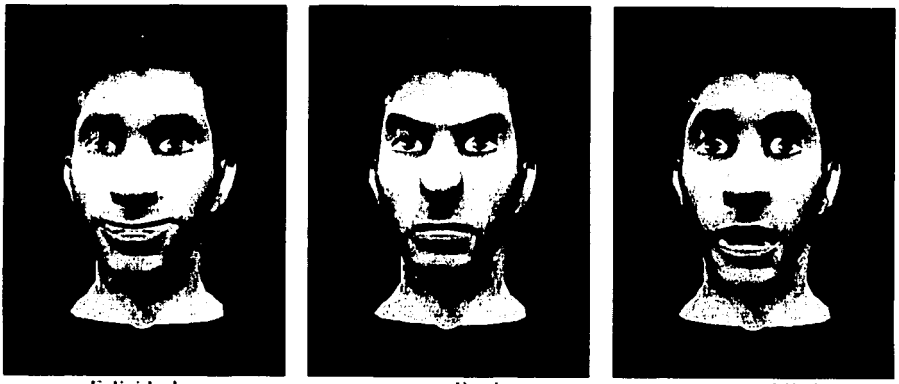


Figura 3.24 Interfaz para modelar expresiones faciales y visemas

Se utilizo la Biblioteca gtk/gdl, la cual permitió diseñar y elaborar la interface (botones, scrolls, menús, cuadros de dialogo, etc.) que se muestra en la figura 3.24, dada su portabilidad para trabajar en ambientes Windows, Linux.

Para generar la animación que muestra la sincronización entre el modelo facial y el sonido se utilizó como interfaz de programación a GLUT (ver anexo B) como se muestra en la figura 3.12, porque requiere de muy pocas rutinas o funciones para desplegar una escena grafica usando OpenGL. El API GLUT es muy útil para desplegar los *frames* (ver figura 3.25 y 3.26) generados por OpenGL.

El editor permite, generar nuevas expresiones para ser utilizadas en diferentes fines como las que se muestran en la figura 3.25 y 3.26



Felicidad Enojo Miedo

Figura 3. 25 Lista de expresiones básicas

Sorpresa Disgusto Tristeza

Figura 3. 26 Lista de expresiones básicas

3.8 Animación de expresiones por interpolación

La interpolación es un método matemático que permite la obtención de curvas a partir de un conjunto de puntos dado y es una herramienta básica en animación por computadora. En la actualidad el uso de ciertos esquemas para implementar y controlar la animación facial es el uso de poses de expresiones clave (key expression) e interpolación. Parke primero demostró el uso de esta aproximación para producir animación facial viable [Par72].

$$P(u) = (P_1 - P_0) u + P_0 \qquad \text{donde} \qquad 0.0 < u < 1.0$$

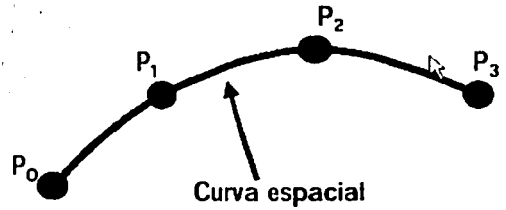
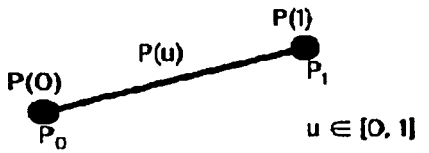


Figura 3.27

La interpolación es un camino para manipular superficies flexibles tales como modelos faciales. La noción de interpolación es muy simple. En el caso unidimensional. Dados dos valores (ver figura 3.27) requerimos determinar un valor intermedio donde el valor intermedio deseado es especificado por un coeficiente de interpolación fraccionario u.



$$P(u) = (P_1 - P_0)\, u + P_0$$
$$P(u) = (1-u)\, P_0 + u\, P_1$$
$$P(u) = a_1 u + a_0$$

$$p_x(u) = a_{x1} u + a_{x0}$$
$$p_y(u) = a_{y1} u + a_{y0}$$
$$p_z(u) = a_{z1} u + a_{z0}$$

Figura 3.28

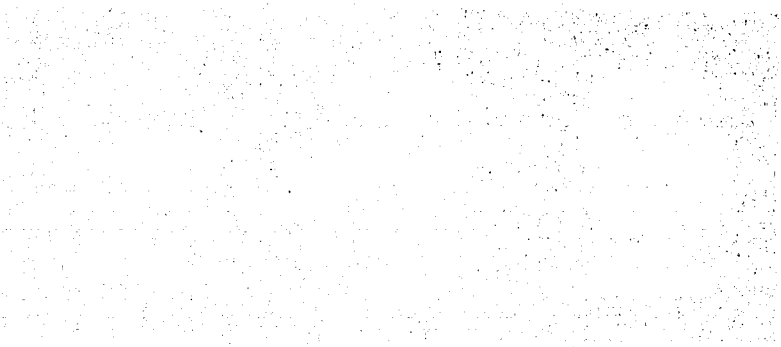
Este concepto es fácil de expandir a más de una dimensión (ver figura 3.28), aplicando este procedimiento a cada dimensión. La idea generalizada a la superficie del polígono es aplicar el esquema a cada vértice definida a la superficie. Cada vértice tiene dos posiciones de tres dimensiones asociadas al polígono. Las formas intermedias de la superficie son logradas por Interpolación a cada vértice en dos posiciones extremas.

A continuación en la figura 3.29 se muestra un ejemplo (*frame2*, pueden ser más) de Interpolación entre expresiones utilizando la interfaz creada con GLUT para la animación (ver figura 3.24).

Frame 1 Frame 2 Frame 3

Figura 3.29 Cuadros intermedios al aplicar Interpolación

La idea es definir la geometría que describen la cara en por lo menos dos expresiones diferentes. Entonces un único parámetro, el coeficiente de interpolación, es usado como una función de tiempo para cambiar la cara de una expresión dentro de otra. Una suposición básica subordinada a la interpolación de superficies faciales es que una única topología facial puede ser usada para cada superficie. Si la superficie de la topología es fija, se manipula la forma de la superficie involucrada, es decir, lo que se manipula en realidad son los vértices de la superficie de una malla.



Capítulo 4

Fonemas y modelado de Visemas en Español

En esta sección se da una introducción de algunos conceptos lingüísticos útiles, que sirven para entender la producción de los sonidos del habla y como se articulan. Esto permite obtener los visemas de cada fonema del español.

4.1 Fonética y fonología

La *fonología* y la *fonética* son disciplinas lingüísticas que estudian los sonidos, pero con diferentes fines. La *fonología* estudia la función de los sonidos en una lengua. La *fonética* estudia los sonidos desde el punto de vista físico y fisiológico.

En la actualidad la *fonología* clasifica a los sonidos en un sistema, basándose en sus características articulatorias y en la distribución de estos sonidos en la cadena sonora del habla. Los *fonemas* son las unidades de sonido establecidas por la fonología y se representan gráficamente con símbolos lingüísticos entre barras (/ /). Por ejemplo, el fonema /s/ representa el sonido producido al pronunciar las letras s y z en las palabras *casa* y *caza*, respectivamente.

La *fonética* es la ciencia que estudia la producción, transmisión y recepción de los sonidos en una lengua, en donde se distingue los sonidos por sus particularidades contextuales y sus propiedades sonoras en el habla. Las unidades de sonido establecidas por la fonética son las unidades fonéticas. A cada uno de los sonidos producidos al articular un fonema de distintas formas le corresponde una unidad fonética. Las unidades fonéticas se representan por símbolos fonéticos. Sus principales ramas son:

- fonética experimental,
- fonética articulatoria,
- fonemática o fonética acústica.

4.1.1 Fonética Experimental

Estudia los sonidos orales desde el punto de vista físico, reuniendo los datos y cuantificando los datos sobre la emisión y la producción de las ondas sonoras que configuran el sonido articulado. Utiliza instrumentos como los rayos X y el quimógrafo, que traza las curvas de intensidad. El conjunto de los datos analizados al medir los sonidos depende únicamente de la precisión del instrumental, así como de otros conocimientos conexos. También se han descubierto diferencias importantes en cada sonido oral.

4.1.2 Fonética Articulatoria

Es la que estudia los sonidos de una lengua desde el punto de vista fisiológico, es decir, describe los órganos orales que intervienen en su producción, su posición y cómo esta varía los distintos caminos que puede seguir el aire cuando sale por la boca, nariz, o garganta, para que se produzcan sonidos diferentes. No se ocupa de todas las actividades que intervienen en la producción de un sonido, sino que selecciona sólo las que tienen que ver con el lugar y la forma de articulación. Los símbolos fonéticos y sus definiciones articulatorias son las descripciones abreviadas de tales actividades o **transcripción fonética**. Los símbolos

fonéticos que se usan más frecuentemente son los adoptados por la Asociación Fonética Internacional en el alfabeto fonético internacional (A.F.I.) que se escriben entre corchetes "[...]" y que cualquier símbolo entre corchetes representa un alófono (sonido) y no una letra. Por lo que viene trascrito entre corchetes se llama trascripción fonética.

Los órganos que intervienen en la articulación del sonido son **móviles** o **fijos**.

- Son **móviles** los labios, la mandíbula, la lengua y las cuerdas vocales, que a veces reciben el nombre de **órganos articulatorios**. Con su ayuda el hablante modifica la salida del aire que procede de los pulmones.
- Son **fijos** los dientes, los alvéolos, el paladar duro y el paladar blando.

4.1.3 Fonética Acústica

Estudia la onda sonora como la salida de un resonador cualquiera, esto es, equipara el sistema de fonación humana con cualquier otro sistema de emisión y reproducción de sonidos. En la comunicación, las ondas sonoras tienen un interés mayor que la articulación o producción de los sonidos; para un determinado auditorio recibe y decodifica el sonido a pesar de que haya sido emitida por medio de una articulación oral, o por medio de un determinado aparato emisor de sonidos o incluso por medio de un ave como una cotorra.

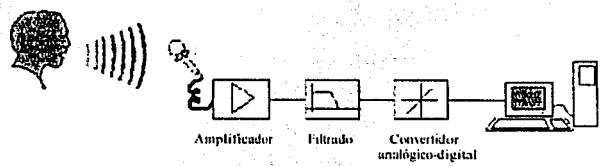


Figura 4.1 Análisis de la voz humana

Para grabar las características más significativas de las ondas sonoras (figura 4.1) y para determinar el resultado de las distintas actividades articulatorias se puede emplear el espectrógrafo que de forma experimental, permite llegar a saber cuáles son los rasgos necesarios y suficientes que identifican los sonidos de la lengua (figura 4.2).

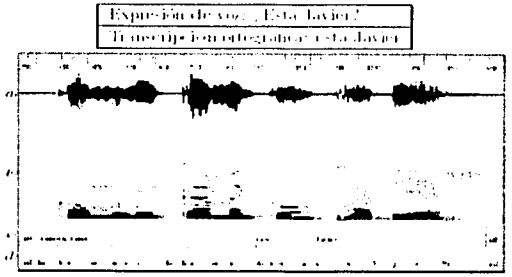


Figura 4.2

4.2 Sistema fonológico del lenguaje español

El sistema fonológico del español consta de 23 fonemas (Tabla 4.1):

- 18 consonantes
- 5 vocales

	Fonema	*Palabra*	*Secuencia de Fonemas*
1	/p/	punto	/p/ /u/ /n/ /t/ /o/
2	/b/	baños	/b/ /a/ /ñ/ /o/ /s/
3	/t/	tino	/t/ /i/ /n/ /o/
4	/d/	donde	/d/ /o/ /n/ /d/ /e/
5	/k/	casa	/k/ /a/ /s/ /a/
6	/g/	ganga	/g/ /a/ /n/ /g/ /a/
7	/ch/	chato	/ch/ /a/ /t/ /o/
8	/f/	falda	/f/ /a/ /l/ /d/ /a/
9	/s/	casa	/k/ /a/ /s/ /a/
10	/j/	jamás	/j/ /a/ /m/ /a/ /s/
11	/y/	un yugo	/u/ /n/ /y/ /u/ /g/ /o/
12	/m/	mano	/m/ /a/ /n/ /o/
13	/n/	nada	/n/ /a/ /d/ /a/
14	/ñ/	baño	/b/ /a/ /ñ/ /o/
15	/l/	lado	/l/ /a/ /d/ /o/
16	/ll/	pollo	/p/ /o/ /ll/ /o/
17	/r/	pero	/p/ /e/ /r/ /o/
18	/rr/	perro	/p/ /e/ /rr/ /o/
19	/a/	caso	/k/ /a/ /s/ /o/
20	/e/	mesa	/m/ /e/ /s/ /a/
21	/i/	piso	/p/ /i/ /s/ /o/
22	/o/	modo	/m/ /o/ /d/ /o/
23	/u/	cura	/k/ /u/ /r/ /a/

Tabla 4.1 Conjunto de fonemas del español

Existe una correspondencia entre fonemas y letras. Las letras son los signos gráficos que se utilizan para representar a los fonemas en la escritura y cuyo conjunto constituye el alfabeto de una lengua.

En el español existen 29 letras para representar a los 23 fonemas que se muestran en la tabla 4.1. Existen varias formas de relación entre los fonemas y las letras, como se muestra a continuación:

a) Fonemas que se representan por dos o más letras diferentes:

Fonemas	*Letras*	*Ejemplos*
/b/	b, v	beber, vivir
/s/	c, z, s	cenicero, zapato, sala
/k/	c (ante *a*, *o*, *u* o consonante) qu (ante *e*, *i*) k	casa, comer, cuento, cráter queso, quitar kilogramo
/i/	i, y	iglesia, ley
/j/	g (ante *e*, *i*) j	genio, gitano jabón, juez
/rr/	r al principio de palabra y después de *l*, *n* y *s* rr (entre vocales)	ruido alrededor, Enrique, Israel ferrocarril

b) Fonemas diferentes que se representan por una misma letra:

Fonemas	*Letras*	*Ejemplos*
/s/, /k/	c	cielo, casa
/g/, /j/	g	gota, corregir
/i/, /s/	y	ley, leyes
/r/, /rr/	r	pero, roer

c) Letras que corresponden a dos fonemas:

Fonemas	*Letras*	*Ejemplos*
/k/ + /s/	x	explorador
/g/ + /s/	x	examen

d) Fonemas simples representados por dos letras:

Fonemas	*Letras*	*Ejemplos*
/ll/	ll	llover, pollo
/rr/	rr (entre dos vocales)	perro, corre
/ch/	ch	muchacho
/k/	qu (ante *e*, *i*)	queso, quitar
/g/	gu (ante *e*, *i*)	merengue, águila

e) Letra que no representa ningún fonema:

Fonemas	*Letra*	*Ejemplos*
—	h	haber, cohete

f) La u que

g) h acompaña a la g y a la q cuando estas consonantes aparecen ante e, i, no se pronuncia.

4.3. Clasificación de los fonemas

La clasificación de los fonemas del español está en función de la forma y el lugar de articulación. Los sonidos están divididos en dos grupos principales:

1. *vocales*, si el aire sale libremente.
2. *consonantes*, si la corriente de aire es detenida u obstruida

4.3.1. Fonemas asociados a las vocales

Una **vocal** es un sonido que se produce al pasar el aire de los pulmones a la laringe y luego por la boca (o por la nariz y la boca) sin ninguna obstrucción audible con la excepción de las vibraciones de las cuerdas vocales.

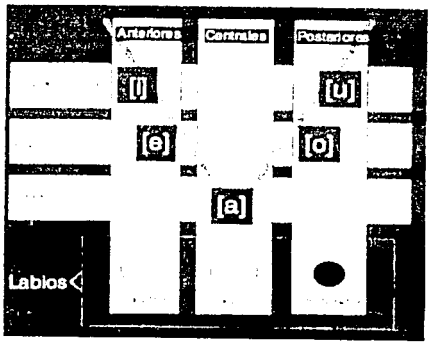


Figura 4.3 Triangulo vocálico

Se emplean tres parámetros para describir las **vocales** (figura 4.3) según varíe la posición de la lengua: tanto a partir de su **eje vertical** (alta, media y baja); como a partir de su **eje horizontal** (anterior, central y posterior) y de la **posición de los labios** (redondeados para /u/ y /o/, neutros para las demás vocales).

Las vocales que se distinguen por su posición en el eje vertical (figura 4.3) son:

- vocales **altas** las vocales de la palabra "huir", es decir, la [i] y la [u].
- vocales **medias** la [e] y la [o], es decir las vocales de la palabra "pero"
- vocal **baja** la [a] de la palabra "va".

Así, la lengua va de abajo arriba para pronunciar las dos vocales seguidas de la palabra "aire", pero desciende a una posición media para pronunciar su última vocal. Hace el camino contrario de arriba abajo para pronunciar "puerta".

Las vocales que se distinguen por su posición en el eje horizontal son:

- Son vocales **anteriores** del español la [i] y la [e], es decir las vocales seguidas de la palabra "piel";
- las vocales **posteriores** son la [o] y la [u], es decir las vocales de la palabra "puro";
- la [a] es la vocal **central**.

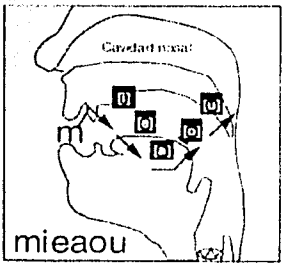


Figura 4.4
Triangulo vocálico

La lengua se mueve de atrás hacia adelante para emitir las vocales de la palabra totales, hace el camino contrario para emitir las vocales de la palabra piélago. Las posiciones que mantiene la lengua para emitir las vocales u, i y a constituyen los vértices del llamado esquema vocálico -i-e-a-o-u-i- como se muestra en la figura 4.4.

4.3.2. Fonemas asociados a las consonantes

En la producción de una consonante, la salida del aire de los pulmones es interrumpida y modificada por algún órgano bucal, generalmente la lengua, aunque también con cierta frecuencia, intervienen los labios. En la descripción (tabla 4.2) de las consonantes se emplean asimismo tres parámetros:

1. el lugar o punto de articulación (el "¿dónde?")
2. la manera o modo de articulación (el "¿cómo? se modifica el aire al salir de la boca"), y
3. presencia o ausencia de sonoridad (sordo / sonoro, i.e., con o sin vibración de las cuerdas vocales).

Lugar de Articulación

Al hablar de **lugar de articulación** (tabla 4.2), se debe centrar siempre en dónde se produce una consonante determinada. Los sonidos se producen cuando se ponen en contacto dos órganos articulatorios.

Para el idioma Español hablado en México hay nueve lugares de articulación para las consonantes:

1. *Bilabial.* [p] de poco, que exige el contacto entre los dos labios (superior e inferior).
2. *Labiodental* [f] de fuente que exige el contacto entre el labio inferior y los incisivos superiores.
3. *Dental* se produce con la lengua contra el borde o atrás de los dientes superiores frontales: la [t] de todo es dental.
4. *Alveolar* se produce cuando una parte de la lengua toca la región que está inmediatamente detrás de los dientes, los alvéolos. Ejemplo: [n] en luna.
5. *Palatal:* Posición de la lengua contra o cerca del paladar duro. Ejemplo [λ] en yeso.
6. Velar: Dorso de la lengua contra o cerca del velo. Ejemplo [k] en cocina.
7. Labio-velar: El labio inferior cerca del labio superior y al mismo tiempo el dorso de la lengua crece del velo. Ejemplo [w] en hueso
8. *Alveopalatales* se pronuncian en la región que está inmediatamente detrás de los alvéolos, es decir, se trata de un sonido parcialmente alveolar y parcialmente palatal.: la [ĉ], ortográficamente "ch" de chico es un sonido álveo-palatal.

9. *Uvular.* Por ejemplo [rr] en carro, dorso de la lengua contra la úvula. También se toma como velar.

En resumen, los nueve puntos de articulación:

Sonidos

1. *bilabiales:* [p], [b], [m]
2. *labiodentales:* [f]
3. *dental:* [t], [d], [n]
4. *alveolar* : [s], [n], [l], [r],[ρ] (=rr)
5. *labiovelar:* [w] (=u)
6. *alveopalatal:* [č](=ch)
7. *palatales:* [ñ], [y], [λ](=ll)
8. *velar:* [k], [x](=j), [w](=u), [g]
9. *Uvula:* [x]

Modo de articulación

Al hablar del **modo de articulación** (tabla 4.2), significa centrar la atención en **cómo** se produce un determinado sonido. El modo de articulación se determina por la disposición de los órganos móviles en la cavidad bucal y cómo impiden o dejan libre el paso del aire. Se han distinguido seis modos de articulación, lo que lleva a hablar de:

1. *Sonidos oclusivos.* Se dan porque el aire se detiene momentáneamente al cerrar el pasaje bucal y después es liberada con una pequeña explosión. Ejemplo: [p] en puse.
2. *Sonidos nasales.* Se dan porque el aire pasa por la nariz: cf. [m, n, ñ]).
3. *Sonidos fricativos.* Se dan por que un cierre parcial de la corriente de aire causa fricción [= vibración del aire] en la producción de sonidos [f], [s], [x],[w]).

4. *Sonidos africados (oclusivos + fricativos).* Se dan porque la corriente de aire es detenida como en una oclusiva; pero en lugar de ser liberada abruptamente, es liberada con fricción como en un fricativo. Ejemplo: [ll] en llueve.
5. *Sonidos vibrantes líquidos.* Se dan porque la [r] simple o doble que "vibran" contra los alvéolos). Ejemplo: [r] en rosa.
6. *Sonidos laterales líquidos.* Se dan porque la cavidad oral es cerrada a la mitad, pero la corriente de aire escapa por ambos lados del lugar de articulación. Ejemplo: [l] en lana.

También se dice que las consonantes oclusivas, fricativas y africadas se clasifican como **obstruyentes** por el hecho de que, al salir el aire por la cavidad bucal hay obstrucción (parcial o total). Las consonantes nasales, laterales y vibrantes se agrupan bajo el término de **sonantes.**

Los 6 modos de articulación

Sonidos:

1. oclusivos: [p], [t], [k], [b], [d], [g]
2. nasales: [n], [m], [ɲ], [ñ]
3. fricativos: [f], [s], [x]
4. africados: [č], [λ]
5. laterales: [l]
6. vibrantes: [r], [ɾ]

Presencia o ausencia de sonoridad

Son *sonoros* (tabla 4.2) todos los sonidos que se producen con vibraciones de las cuerdas vocales (laringe).

Son sonoras:

- La mayoría de las consonantes. [b], [d], [g], [ɛ], [ʝ], [l], [r], [ɾ], [w].
- Todas las vocales. [a], [e], [i], [o], [u].
- Todas las nasales. [n], [m], [ɱ], [ñ]

Son *sordos* todos los sonidos que no llevan vibración laríngea. [f], [s], [x]

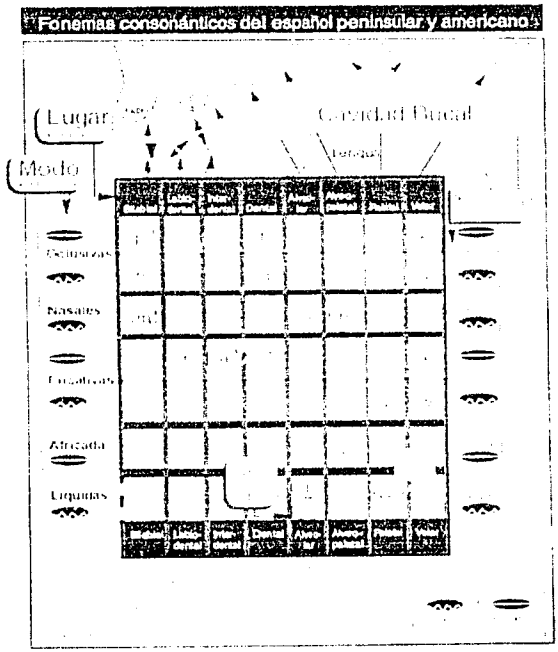


Tabla 4.2 Fonemas asociados a las consonantes (Richard Barrutia [Richard94])

4.4. Fonología

Entre la gran variedad de sonidos que puede emitir un hablante es posible reconocer los que representan el 'mismo' sonido, aunque las formas de pronunciarlo resulten distintas desde el punto de vista acústico; a la vez se pueden distinguir los sonidos que señalan una diferencia de significado. Cada vez que se emite una palabra no se realiza de la misma manera, porque cada emisión depende de los otros sonidos que la rodean. Los sonidos adquieren valores distintos según la función que ocupen en un contexto dado. Sin embargo existen unos rasgos que no varían y que permiten reconocerlos sin confusiones en cualquier posición.

Por otro lado, los sonidos que componen una palabra son las unidades mínimas que la hacen diferente de otra. Una prueba sencilla es la comparación de lo que se llama 'segmentos portadores de significado de los llamados pares mínimos': los sonidos que forman la palabra "más" pueden ser sustituidos por otros, y al hacerlo, se forman palabras diferentes: vas, mes, y mar. Por este procedimiento se pueden aislar las unidades mínimas que distinguen los significados, es decir, los fonemas.

Cada fonema se describe siguiendo criterios físicos y articulatorios, en función del punto de articulación o de su carácter de sonoro o sordo. Cada uno de los componentes que define un sonido es un rasgo distintivo /mas/ es distinto de /vas/ en función de los fonemas /m/ y /b/; se definen, /m/ como [bilabial], [sonoro], [nasal]; y /b/, como [bilabial], [sonoro], [oclusiva]; el único rasgo que los diferencia es la condición de nasalidad y oclusividad. Lo mismo podría hacerse al comparar /a/ y /e/, /s/ y /r/ y cuantas oposiciones revelen sonidos diferentes.

Por rasgos distintivos se describen todos los sonidos que constituyen una lengua. La teoría de los rasgos distintivos se formuló en primer lugar dentro de la escuela estructuralista; está incorporada a la teoría generativa que trata de construir una explicación fonológica [Aguilar97] dentro de la teoría general de la gramática.

A este análisis de los fonemas en términos de segmentos fónicos aislados se le llama fonología de los segmentos; existe otra rama que trata de los suprasegmentos y se ocupa de las unidades mayores del componente fónico, tales como la sílaba, bien estudiada por Straka, las frases y las oraciones, así como los contornos de intensidad y entonación. A este enfoque de la fonología se le llama fonología de los suprasegmentos.

4.5 Alfabetos fonéticos

Un alfabeto fonético es un conjunto de símbolos que representan fonemas y variaciones alofónicas (sonidos) entre fonemas y que es comúnmente utilizado para trasmitir datos de voz. El proceso de asociar un símbolo del alfabeto fonético a un sonido de voz se le llama **etiquetado fonético** o **trascripción fonética** de sonidos de voz. A la trascripción de una secuencia finita de sonidos de voz le corresponde una secuencia finita de símbolos fonéticos. Al conjunto de símbolos fonéticos OGIbet (Oregon Graduate Institute) para el español hablado en México esta formado a partir del alfabeto fonético de Worldbet (ver tabla 4.3).

Woldbert	OGI	Ejemplo	Descripción
p	p	Poco "little"	voiceless bilabial plosive
pc	pcl		voiceless bilabial closure
b	b	Boca "mouth"	voiced bilabial plosive
bc	bcl		voiced bilabial closure
t[	t	Tengo "I have"	voiceless dental plosive
tc	tcl		voiceless dental closure
d[	d	Dentro "inside"	voiced dental plosive
dc	dcl		voiced dental close
k	k	Coma "coma"	voiceless velar plosive
kc	kcl		voiceless velar close
g	g	Goma "glue"	voiced velar plosive
gc	gcl		voiced velar closure

tS	ch	Chica "little girl"	voiceless palatal affricate
tSc	chcl		voiceless palatal affricate closure
dZ	jh	Llama "he calls"	voiced palatal affricate
dZc	jhcl		voiced palatal affricate closure
V	bx	Sabio "wise"	voiced bilabial fricative
f	f	Favor "favor"	voiceless labiodental fricative
T	tx	Plaza "plaza"	voiceless interdental fricative
D	dx	Cada "each"	voiced dental fricative
s	s	Sol "sun"	voiceless dental sibilant
hs	hs	Estás "you are"	aspiration replacing s
z	z	Desde "since"	voiced dental sibilant
s-j	sh	Dos "two"	voiceless palatalized /s/
Z	jh	Ella "she"	voiceless palatal sibilant
x	hx	Jota "letter j"	voiceless velar fricative
G	gx	Lago "lake"	voiced velar fricative
r	rr	Perro "dog"	alveolar trill
rf	r	Pero "but"	alveolar, retroflex flap
m	m	Matar "to kill"	bilabial nasal
n	n	Nadar "to swim"	dental nasal
n	ny	Niño "child"	palatal nasal
N	ng	Cinco "five"	velar nasal
L	l	Lana "wool"	dental lateral
W	w	Hueso "bone"	Labiovelar glide
L	ly	Tortilla "tortilla"	palatal lateral approximant
j	y	Llorar "to cry"	palatal glide

Tabla 4.3 Alfabetos fonéticos Woldbet y OGI

4.6 Modelado de Visemas

Un *visema* se define como la forma de la boca que se asocia a un fonema audible. Los *visemas* básicos para pronunciar cada fonema (vocal o consonante) son construidos por la acción combinada de varios músculos a partir de la descripción fonética (modo y lugar de articulación) de cada fonema (sección 4.3).

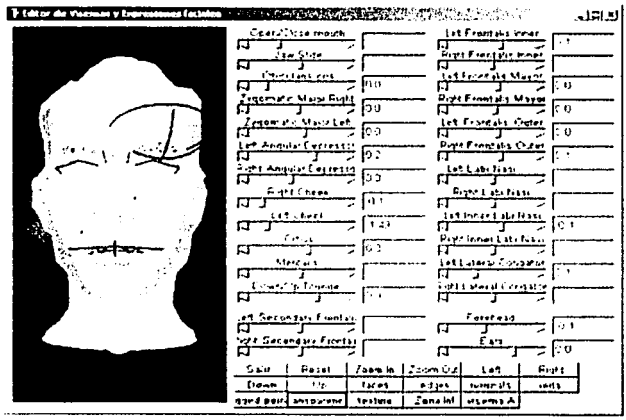


Figura 4.5 Editor de visemas

Como parte del presente trabajo se implementó un editor de *visemas* y expresiones faciales (ver figura 4.5 y la sección 3.7.1) que permite modelar la forma de los visemas de una boca en específico cuándo se articula un fonema en un momento dado.

El editor permite crear un conjunto de *visemas* o fonemas visuales, dado de formas de la boca para cada fonema, y producir la animación para la boca como una sucesión de *visemas*

del conjunto predefinido. Los problemas de decidir la sucesión de visemas del conjunto predefinido es que:

- El conjunto de formas de la boca (visemas) sea usado según convenga al proposito del animador.
- La sucesión de tiempo de los *visemas* requeridos es conocido, y guardado en la forma de un archivo de entrada ASCII (ver figura 4.6).

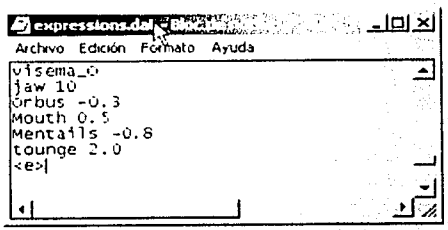


Fig. 4.6 Archivo 'expressions.dat'

Lo que resta para al utilizar el editor de *visemas* es:

- Diseñar un conjunto de visemas a ser usados para la boca.
- Hacer una animación de la boca según la sucesión de visemas dada.

Una vez que el conjunto de visemas es definido por medio de la comprensión y descripción fonética (figura 4.5 y figura 4.6), y de los alfabetos fonéticos (ver tabla 4.3) que se usan en un sistema que convierte texto en sonido, por lo que es necesario que el animador haga corresponder ambas informaciones con cada uno de los visemas (ver tabla 4.4) que se necesitan en el español mexicano.

Cuando se está intentando sincronizar la animación con el habla, se requiere entender la duración e inflexión de los fonemas para poder tomar una decisión acerca de qué visema usar y cuándo.

Alfabetos fonéticos			Visemas	Ejemplo	Descripción Fonética	
Fonología Y Fonetica Barrutia	Festival				Modo de Articulación	Lugar de Articulación
	Worldbet	OGI				
/a/	/a/	/a/	visema_A	Lata	Vocal	vocal
/e/	/e/	/e/	visema_E	Pepe	vocal	vocal
/i/	/i/	/i/	visema_I	pipa	vocal	vocal
/o/	/o/	/o/	visema_O	boca	vocal	vocal
/u/	/u/	/u/	visema_U_W	duda	vocal	vocal
/p/	/p/	/p/	visema_P_B_M_PAU	poco	Oclusiva sorda	Bilabial
/b/	/b/	/b/	visema_P_B_M_PAU	boca	Oclusiva sonora	Bilabial
/t/	/t[/	/t/	visema_T_D	tiene	Oclusiva sorda	Dental
/d/	/d[/	/d/	visema_T_D	dedo	Oclusiva sonora	Dental
/k/	/k/	/k/	visema_K_G	casa	Oclusiva sorda	Velar
/g/	/g/	/g/	visema_K_G	goza	Oclusiva sonora	Velar
/f/	/f/	/f/	visema_F	foco	Fricativa sorda	Labio-dental
/s/	/s/	/s/	visema_S	sola	Fricativa sorda	Alveolar
/x/	/x/	/x/	visema_J	juez	Fricativa sorda	Uvular
/ʧ/	/tS/	/tS/	visema_CH	chica	Africada sorda	Alveo-palatal
/ʤ/	/dZ/	/dZ/	visema_Y_LL_RR_R	llueve	Africada sorda	Palatal
/m/	/m/	/m/	visema_P_B_M_PAU	moto	Nasal	Bilabial
/n/	/n/	/n/	visema_N_L	nadar	Nasal	Alveolar
/ɲ/	/n_~/	/ny/	visema_ENIE	niño	Nasal	Palatal
/ŋ/	/N/	/N/	visema_N_L	cinco	Nasal	Dental
/l/	/l/	/l/	visema_N_L	lisa	Lateral líquida	Alveolar
/ʝ/	/j/	/j/	visema_Y	mayo	Fricativa sonora	Palatal
/w/	/w/	/w/	visema_U_W	hueso	Semivocal	Labio-velar
/ɾ/	/r(/	/r/	visema_Y_LL_RR_R	rosa	Vibrante líquida	Alveolar
/r/	/r/	/rr/	visema_Y_LL_RR_R	corral	Vibrante líquida	Alveolar

Tabla 4.4

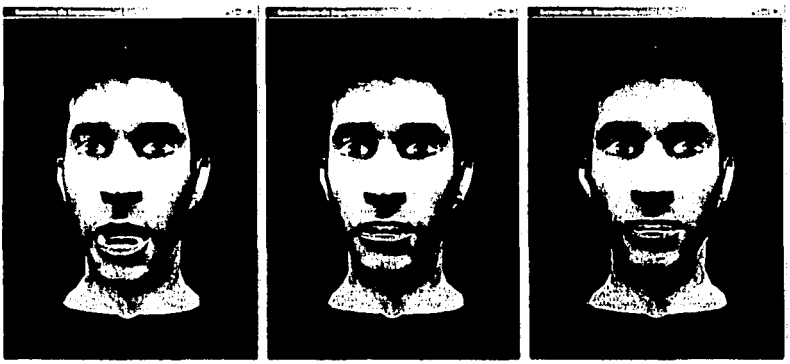
El editor de visemas y expresiones, permite diseñar bocas y expresiones para situaciones determinadas, en general las formas de las vocas y las expresiones faciales varían de una situación a otra, dependiendo del contexto donde se utilicen.

Las clasificaciones de los *visemas* se basan en el estudio de las secciones anteriores y la tabla 4.4. Por lo anterior, obsérvese la clasificación de los visemas figura 4.7 que se modelaron a partir de la observación realizada a una persona al momento de hablar y gesticular en español mexicano. Para esto se genero una tabla como se muestra en la tabla 4.5 para cada modelo con el editor. Podemos observar que se tienen menos visemas que fonemas en la lengua del español mexicano, se obtuvieron 14 visemas, así como los músculos involucrados.

Viseme	Jaw	Tounge	Orbus	Mentalis	Mouth	Left Zygomatic Major	Right Zygomatic Major	Left Angular Depressor	Right Angular Depressor
A	13.0								
E	3.0	-2.0	0.3	-0.3					
I	1.0	-1.0	0.3						
O	10.0	2.0	-0.3	-0.8	0.5				
U_W	4.0	2.0	0.2		4.0				
P_B_M	-4.0		0.2	-0.1		-0.2	-0.3	-0.2	-0.2
T_D	3.0	-9.0							
K_G		-2.0	0.2	0.2					
F			0.3	-1.8					
S	-4.0		0.5	0.2					
J	2.0		0.2	-4.0					
CH	-2.0		0.1	0.4	0.2			0.2	0.2
n_N_L	0.2	-0.8	0.2	0.4	0.2				
Y_LL_R RR ENE	1.0	-7.0	0.1	0.2	0.1				

Tabla 4.5 Resultados del editor de visemas

En seguida se muestra el resultado (ver figura 4.7) de modelar los visemas en el editor. Las imágenes también sirven como guía para modelar los *visemas*.

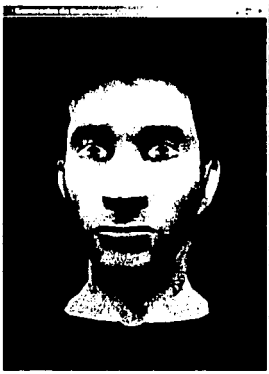


Visema_A Visema_E Visema_I

Figura 4.7 Clasificación de visemas

Visema_0

Visema_U_W



Visema_P_B_M_PAU

Visema_T_D

Visema_K_G

Visema_F

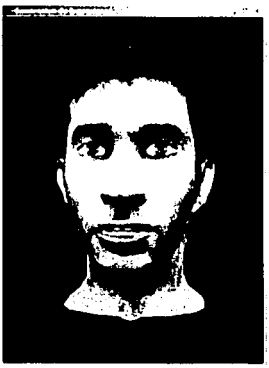
Figura 4.7 Clasificación de Visemas

Visema_S

Visema_J

Visema_CH

Visema_n_N_L



Visema_Y_LL_R_RR_ENE

Figura 4.7 Clasificación de Visemas

Capítulo 5

Conversión de Texto a Voz

La conversión de texto a voz (Text to Speech [TTS] o sistemas TTS) permite a una computadora transformar automáticamente *un texto* en su correspondiente *forma sonora.* La posibilidad de producir artificialmente un habla similar a la humana ha despertado siempre el interés de los científicos. Una muestra de los esfuerzos realizados para conseguir este objetivo se encuentra en la «máquina parlante» del barón Wolfgang Von Kempelen (figura 5.1), descrita en 1791. La 'máquina' intentaba reproducir fielmente la anatomía del aparato

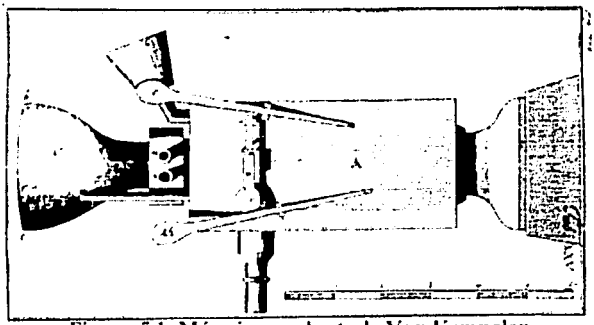


Figura 5.1 Máquina parlante de Von Kempelen (Reproducida de Dudley y Tarnoczy, 1950).

fonador humano, al igual que lo haría más tarde el sistema diseñado por Sir Charles Wheatstone (figura 5.2) en 1835.

Más adelante, los componentes mecánicos se sustituyeron por los circuitos eléctricos, como en el Voder (*Voice Demonstrator*) de Homer Dudley, presentado en dos ferias mundiales en 1939. Un paso crucial se dio en los años sesenta, cuando el control de los ya denominados *sintetizadores en habla* empezó a realizarse desde una computadora completamente automatizada, abriendo así el camino a los productos actuales, cuyas primeras versiones se comercializaron en la década de los ochenta.

Se ha pasado de los fuelles, las lengüetas y las palancas a programas fácilmente accesibles a todos, que permiten que una computadora lea en voz alta un texto cualquiera, cada vez de una forma más natural y en un creciente número de lenguas.

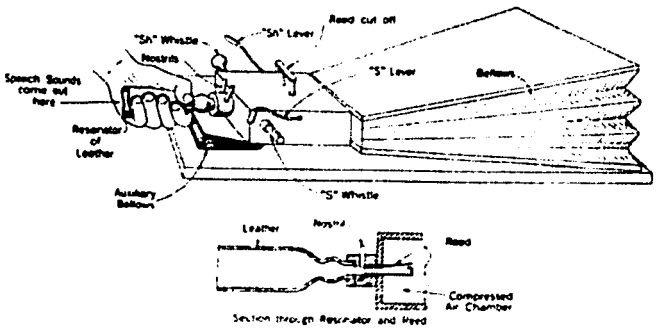


Figura 5.2 Máquina parlante de Wheatstone (Reproducida de Dudley y Tarnoczky, 1950).

A continuación se presenta, algunas características de la voz, así como también cómo funciona en general un sistema TTS y en particular cómo funciona el sistema Festival [Black99]. Al final se expone, como incorporar un archivo de sonido a un modelo facial.

5.1 Características de la voz humana en un sistema TTS

En el aspecto lingüístico, el primer problema que se encuentra en un sistema de conversión texto a voz es que debe inferir el contenido real de la representación escrita del mensaje. Para ello, como se muestra en la figura 5.3, se debería realizar un procesado lingüístico del texto a partir de un análisis fonético-morfológico para derivar la pronunciación, un análisis sintáctico para dar la estructura gramatical del texto y poder inferir rasgos prosódicos, un análisis semántico para dar una representación del significado del mensaje y un análisis pragmático para dar una relación entre frases e ideas de la conversación global. Claramente, este procesado lingüístico es muy ambicioso, y los sistemas actuales simplemente realizan un análisis fonético-morfológico y sintáctico para de este modo determinar los rasgos segmentales y prosódicos de los sonidos que componen el mensaje oral.

Un aspecto importante en la inteligibilidad y naturalidad de la señal sintetizada son las reglas prosódicas, que aunque en cierta medida pueden ser inferidas de la estructura sintáctica de la frase, la mejor forma de generar una entonación adecuada a una frase es que la máquina comprenda a nivel gramatical lo que está diciendo.

A continuación se presenta una clasificación de la señal de voz, en función de sus rasgos segmentales (sonidos elementales) y los rasgos prosódicos (características de enlace entre los aspectos lingüísticos y acústicos de la generación de voz).

Rasgos Segmentales:

- Son características propias de cada sonido elemental que permiten diferenciar los sonidos y reconocer las palabras.
- Los sonidos se clasifican según su carácter sordo/sonoro y la posición de los órganos articulatorios

- La articulación de un sonido influye en la articulación de los sonidos adyacentes que llamaremos coarticulación.

Rasgos Prosódicos:

- Características más globales que afectan a segmentos mayores que el fonema
- Rasgos principales:
  - Entonación (tono fundamental)
  - Duración (tiempo)
  - Intensidad (Energía)
- Aportan información de ritmo, acentuación, tipo de frase, emotividad, etc.
- Evolución más lenta que los rasgos segmentales.
- Especialmente significativos en los núcleos vocálicos

Su correcta realización proporciona naturalidad al habla sintetizada.

5.2 Sistemas de conversión de texto a voz

Los sistemas TTS actualmente estudian el proceso de generación de un mensaje oral desde el punto de vista acústico y lingüístico, por lo cual es necesario entender el comportamiento físico del aparato fonador del ser humano y cómo son procesados por el sistema auditivo humano para desarrollar un modelo matemático del mismo. A la vez, hay que saber cómo extraer del texto, en base a su estructura lingüística, la información necesaria para controlar el modelo matemático y, de este modo, la conversión de texto en habla transforma un texto escrito en su equivalente en voz como se muestra en la figura 5.3.

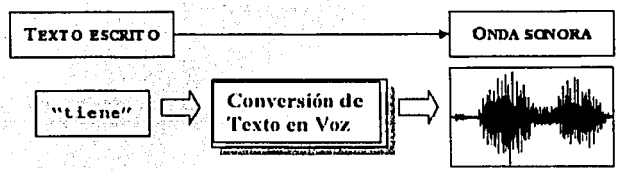


Figura 5.3 Conversión Texto a Voz

Un sistema TTS consta de varios componentes como se muestra en la figura 5.4, los cuales son indispensables para realizar conversión de texto en voz.

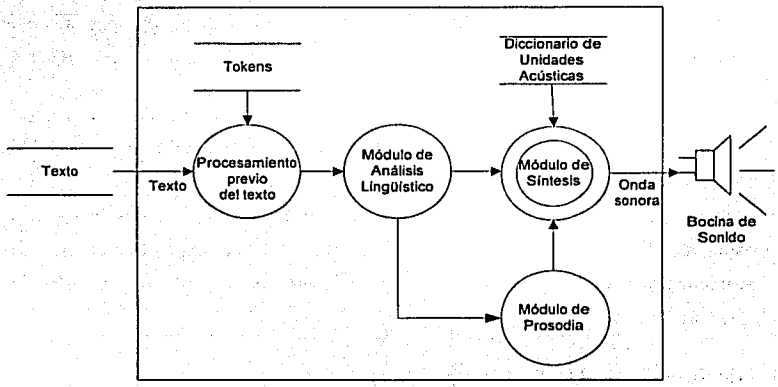


Figura 5.4 Sistema TTS

5.2.1 Procesamiento previo del texto

Un texto escrito que se utiliza como entrada de un conversor de texto a voz requiere un procesamiento previo. Por ejemplo, si se observa un texto real (como el que puede encontrarse en un periódico), se verá que aparecen en él abreviaturas (Dr.), Siglas (TTS), números, fechas, símbolos (como *$*), etc. denominado 'token'. Todos los *token* deben ser, por así decirlo, "deletreados" para que después se pueda realizar la correspondiente trascripción fonética. Esto se consigue creando una lista de reglas de traducción de *tokens* a palabras en los que se especifica cuál es la forma escrita completa de cada elemento.

5.2.2 Módulo de análisis lingüístico

El módulo de análisis lingüístico del texto lleva a cabo dos funciones fundamentales:

1. Transforma la representación ortográfica del mismo en una representación fonética (como en la tabla 4.3), es decir, determina la sucesión de difonemas y trifonemas que lo componen.
2. Extrae del texto la información prosódica del mismo. Esta información se llevará al generador de prosodia, el cual genera la plantilla de prosodia adecuada que permita al sintetizador generar voz con una buena entonación.

Lógicamente, este análisis lingüístico del texto es diferente para cada idioma, teniendo que adaptarse a las características propias de cada uno. Hay que tener en cuenta que tanto la base de fonemas como las características prosódicas son distintas para cada idioma. El texto que se desea sintetizar ha de ser analizado conforme a sus propiedades sintácticas, semánticas y contextuales para producir los parámetros adecuados

5.2.3 Diccionario de Unidades Acústicas

La forma más sencilla de generar voz consiste simplemente en grabar la voz de una persona pronunciando las frases deseadas. Este sistema sólo es viable cuando el número de frases que es necesario sintetizar es pequeño. Por ejemplo, un número concreto de mensajes que se emiten en una estación de tren. En casos como éste, la calidad del sistema depende de la calidad de la grabación de las frases.

Sin embargo, en el caso de un sistema TTS, se necesita un sistema que permita sintetizar cualquier texto que se introduzca por teclado. La solución consiste en dividir la voz en segmentos, los cuales van a constituir una base de sonidos con la que trabajará el módulo de síntesis.

Los diferentes sistemas TTS emplean para producir voz el método de Síntesis Directa o Concatenativo, que consiste precisamente en concatenar uno tras otro todos los sonidos que constituyen el texto. Estos sonidos han sido previamente almacenados y constituyen la base de sonidos.

El primer problema que se plantea para crear una buena base de sonidos consiste en decidir el tipo de unidades acústicas o segmentos fónicos adecuados para formar parte de dicha base. Existe un compromiso entre la calidad de voz conseguida y el tamaño de la base de datos que se necesita para almacenar los segmentos. Cuanto más pequeñas sean las unidades en que se descompone la voz, menor es la base de sonidos utilizada, pero la calidad de la voz también decrece. Una solución a este problema es el empleo de difonemas, los cuales están compuestos por la porción final de un fonema y la inicial del fonema que le sigue. Como el corte está hecho en el centro del fonema, las transiciones entre ellos permanecen intactas. Para el caso del castellano, el número de difonemas necesario para constituir una base es al menos, de 550. Para aumentar la calidad de la síntesis se puede utilizar un número limitado de trifonemas para representar a sonidos en los que los tres fonemas se coarticulan a elevada velocidad (pla, ple, pli, plo, plu, tra, tre, ...)

Una forma de obtener la base de difonemas y trifonemas consiste en la grabación de logótomas. Los logotamas son palabras carentes de significado, compuestas por tres sílabas, que permiten que el segmento a tratar esté aislado sin coarticular con los sonidos anterior y posterior.

5.2.4 Módulo de prosodia

La naturalidad al hablar se consigue con una buena entonación, la cual puede ser incluso necesaria en algunos casos para la inteligibilidad del mensaje. Por ejemplo, la frase "Juan dijo Pedro es un mentiroso", se puede pronunciar de forma diferente, de manera que se podría interpretar de cualquiera de los siguientes modos: "Juan dijo: Pedro es un mentiroso", o "Juan, dijo Pedro, es un mentiroso". En este caso, la entonación contribuye a que cambie el significado del mensaje.

La entonación se considera uno de los principales responsables de la calidad de un sistema de TTS.

La entonación es, ante todo, un fenómeno lingüístico relacionado con la sensación perceptiva que produce la variación a lo largo de todo un enunciado de tres parámetros físicos:

- Frecuencia fundamental (en adelante se le denominara F0)
- Duración
- Amplitud

Proporcionando esta información de distintos tipos, permite al receptor una variación. Estas variaciones se dan en unidades mayores que la palabra, normalmente sintagmas o frases, que son las unidades lingüísticas contenidas en los grupos fónicos. Un estudio de la entonación implica, por tanto, estudiar las variaciones de los parámetros entonativos a lo largo de todo el grupo.

La entonación es un fenómeno que relaciona tres niveles diferentes:

1. Plano físico (o acústico): en este sentido, es el resultado de la variación temporal de una serie de parámetros físicos. Se considera que los tres parámetros antes mencionados son los responsables de la entonación. Éstos también intervienen en otros fenómenos, como el ritmo o el acento, lo que hará que en ocasiones sea difícil atribuir la variación de un parámetro determinado a un fenómeno u otro.
2. Plano perceptivo: el oído humano actúa como un filtro que, en cierta medida, transforma la señal sonora que le llega, desechando algunas de las variaciones de esos parámetros físicos tratados en el apartado anterior.
3. Plano semántico-funcional: el oyente extrae de las variaciones de los parámetros antes mencionados diversas informaciones de tipo lingüístico, o incluso, extralingüístico.

Además, transmite los siguientes tipos de información:

- Información de tipo lingüístico, referente al mensaje, que es generalmente información sintáctica. En primer lugar, permite delimitar las diferentes frases que componen el mensaje. También aporta información sobre el tipo de oración de que se trata. Igualmente, permite distinguir entre enunciados no finales, situados al principio o en medio de una oración, y los resultados al final de las mismas.
- Información sociolingüística, acerca de la procedencia geográfica y social del hablante.
- Información expresiva, acerca del estado de ánimo del hablante.

Según un estudio de Jassem y Demenko[1] existen catorce factores diferentes que pueden determinar las variaciones de los parámetros que influyen en la entonación. Estos factores son los siguientes:

1 Condiciones dependientes de la actitud del hablante, situaciones y del discurso.
2 Situación del acento temático.
3 Realización del acento léxico o gramatical.
4 Longitud de la curva, condicionada segmental o léxicamente.
5 Variación alotónica libre.
6 Efectos de estados emocionales.
7 Rasgos fisiológicos personales del hablante.
8 Rasgos patológicos de la voz.
9 Cambios momentáneos en el tono medio (confidencias, frases parentéticas...).
10 Tiempo personal de elocución.
11 Cambios momentáneos del tiempo.
12 Efectos de los rasgos segmentales.
13 Irregularidades no patológicas en la fonación.
14 Estilo.

5.2.5 Módulo de Síntesis

Podría pensarse que para el funcionamiento de un sistema de síntesis del habla es suficiente con disponer de la información sobre los parámetros que deben utilizarse para que la acción conjunta de una **fuente** y un **filtro** (ver figura 5.5) dé como resultado cada uno de los sonidos del enunciado que se desea reproducir.

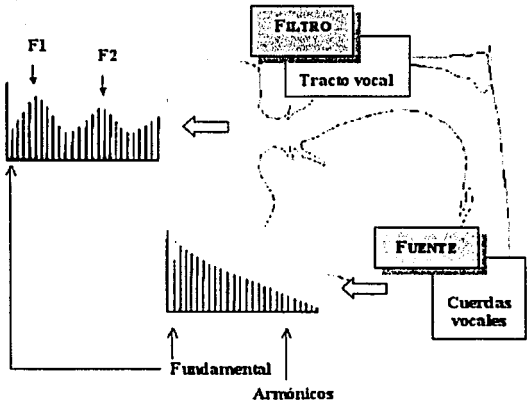


Figura 5.5 Modelo de fuente y filtro

Concretando la síntesis se comienza cuando la transcripción fonética no tiene dificultades, dentro del cual se procede a localizar las unidades fonéticas en el diccionario de unidades. Estas unidades fonéticas deben concatenarse y modificarse a fin de adaptar la duración y la intensidad de los sonidos a cada enunciado, y deben aplicarse también el patrón melódico adecuado al significado y a la forma del enunciado. Finalmente, toda esta información debe convertirse en un conjunto de parámetros acústicos que harán que la fuente y el filtro del sintetizador produzcan la onda sonora que llegará al receptor y que, idealmente, tendría que parecerse lo más posible a la lectura del texto que habría realizado un hablante humano.

A continuación se describen brevemente los elementos que se requieren para realizar la síntesis.

Trascripción fonética

Aunque pueda parecer en principio un proceso trivial, existen ciertos problemas. Por ejemplo, en español "Dr." debe asociarse a "Doctor" para el masculino y "Doctora" para el femenino y el sistema cuando realiza la trascripción fonética debe elegir la forma correcta utilizando información sobre el género del nombre que sigue.

Para ello se utilizan programas que realizan lo que se conoce como un trascripción fonética, es decir, convierte las grafías del texto en símbolos fonéticos correspondientes a los sonidos guardados en el Diccionario de Unidades Acústicas o de Síntesis.

Concatenación de Unidades Acústicas

En cualquier representación acústica del habla puede observarse que los sonidos no se producen aisladamente, sino que se encadenan unos con otros (difonemas) a partir de la información del diccionario de unidades acústicas, con el fin de conseguir habla sintetizada de calidad, concatenando sonidos aislados e intentando imitar el resultado acústico de los movimientos de tracto vocal en las transiciones entre sonidos lo que es una operación sumamente difícil.

Modelo de Fuente y Filtro

En 1960, Gunnar Fant formuló detalladamente este modelo – conocido como el modelo de la fuente con una acción análoga a la de las cuerdas vocales, y un filtro que hace las veces de tracto vocal como se muestra en la figura 5.4.

Para producir artificialmente los sonidos sordos y sonoros, basta con disponer de un mecanismo – sonido en síntesis como la fuente – capaz de generar ondas sonoras periódicas y ondas sonoras aperiódicas (ver figura 5.6), de modo que se pueda imitar las características acústicas de los sonidos sordos y sonoros.

La acción del tracto vocal sobre los armónicos de la onda sonora que se crea en la laringe puede imitarse en la síntesis, con lo que se denomina un filtro; es decir, un mecanismo que aumenta la amplitud de determinados armónicos y reduce la de otros, dando así una configuración acústica diferente a cada sonido.

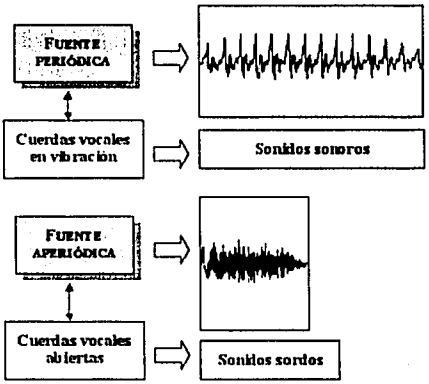


Figura 5.6

En los apartados siguientes se expone brevemente cómo un sistema TTS como Festival convierte automáticamente un texto escrito en sonido y como es posible manipular su ambiente.

5.3 Festival

Festival es un sistema de síntesis de voz multi-lenguaje desarrollado por CSTR (The Centre for Speech Technology Research) de la universidad de Edinburgh. Ofrece un completo sistema TTS con varios APIs (Application Programing Interfase), también es un ambiente para el desarrollo e investigación de técnicas de síntesis de voz. Esta escrito en C++ con un intérprete de comandos basado en Scheme para el control general.

Festival [Black99] se diseñó como un sistema de síntesis para tres niveles de usuarios.

- El primer nivel es para los usuarios que simplemente requieren de una síntesis de voz de alta calidad para la lectura de textos cualquiera con el mínimo de esfuerzo.
- El segundo nivel es para aquellos que desarrollan sistemas de lenguaje y desean incluir una salida con voz sintetizada. En este caso existe una gran cantidad de opciones como voces diferentes, redacción específica, tipos de diálogos entre otros.
- El tercer nivel de usuarios permite el desarrollo y prueba de nuevos métodos de síntesis.

Festival es un sistema TTS que provee todo un ambiente modular, por lo que se pueden modificar los módulos e incorporar los nuevos. Esto permite adaptar el sistema TTS a las necesidades particulares de cada usuario para probar nuevas técnicas de síntesis de voz. El concepto de módulos sustituibles o extensibles en los sistemas de TTS no es exclusivo de Festival. El sistema ATR's CHART [Black94] sigue también esta filosofía.

Festival se ha beneficiado de trabajos previos en su desarrollo. Por ejemplo Osprey y los proyectos políglotas [Taylor91] influenciaron varias decisiones de diseño en Festival. Por lo que también se ha influenciado por programas generales para Ingeniería de Software, especialmente de proyectos de GNU Octave y Emacs. Los scripts de Festival tienen una estructura similar a los que se utiliza en GNU.

A menudo otros sistemas de lenguaje y voz son colecciones de pequeños scripts y código no muy ordenado. Pocas son las personas que pueden describir perfectamente cómo funcionan los algoritmos. Tampoco se puede evaluar con fiabilidad esos sistemas para decir si son buenos o no. El punto importante a desarrollar en sistemas de lenguaje y voz es describir el programa con documentación para poder ser implementado por otros. Festival ofrece un armazón común donde múltiples técnicas pueden ser implementadas por los investigadores y hacer pruebas más confiables en el mismo ambiente.

5.3.1 Arquitectura General de Festival.

Festival cumple con la arquitectura de un sistema de TTS descrita en la sección anterior. Su arquitectura se divide en cuatro partes (figura 5.7): el control del sistema, estructuras de datos, módulos de TTS y conformación de datos.

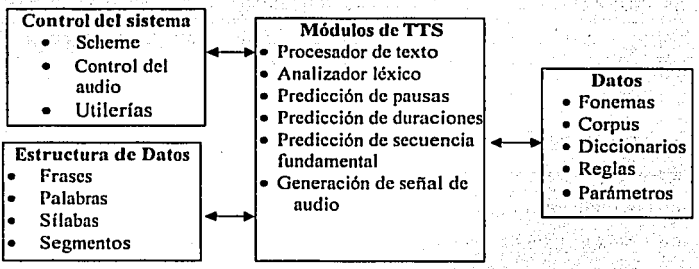


Figura 5.7 Módulos de la Arquitectura de Festival

5.3.2. Control del Sistema

El control del sistema está formado de programas en C y clases de C++ que son la base para el resto del sistema de Festival. Los programas en C/C++ se encargan de funciones de bajo nivel que requieren de ejecución rápida. También se encargan de las funciones de entrada y

salida, las cuales pueden hacerse por medio de un intérprete de comandos o utilizando el concepto de cliente-servidor.

Las funciones de alto nivel de Festival se encuentran escritas en Scheme, y por lo tanto, también cuenta con un intérprete de comandos del mismo. A lo largo del desarrollo de Festival se han podido incorporar nuevos intérpretes de scripts, se facilita la especificación de parámetros y control de flujo del sistema, independientemente de las funciones de bajo nivel. La relación entre las funciones de alto nivel de Scheme con las de bajo nivel en C/C++ aumenta la flexibilidad y la eficiencia del sistema.

5.3.3 Estructuras de datos de Festival

Para lograr la extensibilidad del sistema, se incluye en Festival un esquema eficiente de acceso a los datos. Las estructuras de datos tienen la capacidad para modelar varios aspectos que caracterizan a la voz, tales como duración, entonación y composición. Este esquema también facilita el acceso a los datos a lo largo del todo el proceso de TTS y en cada una de las etapas como se muestra en la figura 5.8.

La principal estructura de datos que se utiliza en el sistema Festival es la denominada *utterance*. Dicha estructura es dada como parámetro de entrada a cada módulo del sistema donde se procesa y modifica. Una estructura *utterance* está definida por un tipo básico de secuencias nombradas, así como también el número de dichas secuencias [Black97]. Una *secuencia* es una lista ordenada de elementos con características y valores que pueden relacionarse con otros elementos en otras secuencias. Los tipos de pronunciación que Festival maneja son *Text*, *Word* y *Segment*. Las secuencias que utiliza son *Word*, *Syllable* y *Wave*.

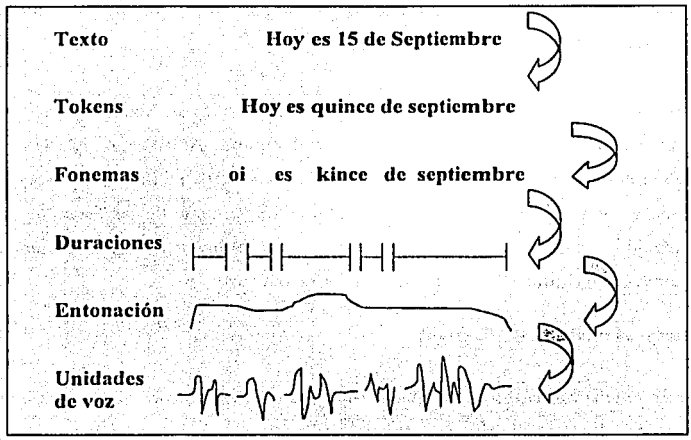


Fig. 5.8 Proceso de Síntesis de Festival

Cada elemento en la *utterance* se encuentra definido por un conjunto de características que permiten modelar varios aspectos relacionados con la voz. Tales características pueden ser por ejemplo la posición inicial y final de un elemento, inicio de una palabra dentro de una frase, inicio y final de cada fonema dentro de la palabra entre otras. Cada elemento también cuenta con un nombre en varios niveles como *palabra*, *token* o *fonema*. Esto facilita el manejo de los elementos en el módulo de *análisis de texto*.

También existen marcas de entonación que permiten definir la frecuencia fundamental para cada elemento.

5.3.4 Proceso general de conversión texto a voz en Festival

En general el proceso de conversión de texto a voz es tomar un *utterance*, el cual contiene una simple cadena de caracteres y convertirla paso a paso, rellenando la estructura del

utterance con más información hasta llegar a construir el *waveform* que dice el contenido del texto. Los procesos involucrados en la convención son, en general, son como sigue:

Tokenización:

Convertir las grafías del texto o cadena de caracteres dentro de una lista de *tokens (símbolos fonéticos)*. Típicamente esto significa separar los *tokens* con espacios en blanco de la cadena de texto original.

Identificación del Token:

Identificación de los tipos en generales de *tokens*, usualmente esto es trivial pero requiere algo de trabajo para identificar los *tokens* de los dígitos como años, días, números, etc.

Token a word (palabra):

Convertir cada token a cero o a palabras, expandiendo números, abreviaciones, etc.

La parte del habla:

Identificar la parte sintáctica del habla para las palabras.

Prosodia o léxico de la frase:

Encontrar la pronunciación de cada palabra para una letra/lexicón a un sistema de reglas de sonido incluyendo fonética y estructura silábica

Acentos de entonación:

Asignación de la duración de cada fonema en el *utterance*.

Generando contorno F0 (melodía):

Generar la melodía basado en los acentos.

Convertir a waveform:

> Convertir a *una forma de onda* a partir de los fonemas, duración y los valores de F0. Esto puede tomar varios pasos incluso la selección de la unidad (como los *difonemas* u otras unidades clasificadas según tamaño), la imposición de la prosodia deseada (duración y *F0*) y la reconstrucción *waveform*.

Cada uno de estos pasos en Festival es logrado por un modulo el cual típicamente agrega nueva información a la estructura del *utterance*.

El número de pasos que suceden pueden variar y eso depende en particular de la voz seleccionada y del tipo de *utterance*.

La estructura del *utterance* consiste de un conjunto de *items* (elementos), los cuales pueden ser parte de una o más relaciones. Los *items* representan cosas como palabras, fonemas, quizás pueden también ser usados para representar objetos concretos como las frases del nombre y nodos en los árboles métricos. Un *item* contiene un conjunto de características (nombre y valor). Las relaciones son típicamente listas simples de *items* o árboles de *items*.

Por ejemplo la relación *Word* es una simple lista de *items* cada uno de los cuales representa una palabra en el *utterance*. Las palabras también están en otras relaciones, como en la relación *SylStructure* donde una palabra está en la cima de un árbol de estructura conteniendo sus 'sílabas y segmento'.

5.3.5 Módulo de Síntesis de Unidades en Festival

Un sintetizador concatenativo forma la voz sintética pegando unidades de voz digitalizadas. Sin embargo es difícil decidir cuáles son los segmentos a ser utilizados como unidades básicas, por lo que se sugiere el uso de una unidad que va desde la mitad de un fonema a la mitad del siguiente, o las semisílabas, formadas por el primer sonido completo y la mitad del

segundo. Tal unidad se le llama difonema [Peterson, 1959]. Además de concatenar unidades, es necesario modificar la entonación y la duración de los difonemas almacenados con el fin de modificar la prosodia de la síntesis.

Festival, permite utilizar varios métodos de síntesis basados en concatenación de difonemas, entre ellos está TD_PSOLA el cual se encuentra como sintetizador nativo. En esta tesis se utilizará el sintetizador MBROLA por que aparece en el contexto de Síntesis de Alta Calidad de Texto en Habla (HQ-TTS) por concatenación como uno de los mejores sintetizadores ya que combinan la eficiencia del algoritmo original TD_PSOLA incluido en Festival con la flexibilidad del modelo híbrido H/S (Harmonic/Stochastic).

MBROLA es un sintetizador basado en concatenación de difonemas con el objetivo de construir WAVEFORMS. Es decir que toma una lista de fonemas como entrada, junto con información prosódica (la duración de fonemas, sus segmentos y las características de entonación, como la intensidad y la melodía) para generar el audio con formato Wave. MBROLA (Anexo B.2) soporta un número de 50 conjuntos de voces (difonemas para cada idioma y hablante) incluido el español.

A continuación se explicará como manejar un archivo de audio generado por MBROLA con formato WAVE y generar audio en un espacio simulado tridimensional.

5.4 Sistema de Audio OPENAL

Para incluir en un sistema 3D la voz producida por Festival y MBROLA, y sincronizada por el modelo facial, se utilizó la Biblioteca OpenAL con OpenGL.

OpenAL (Open Audio Library) es una interfaz entre el software y el hardware de audio. La interfaz consiste de un número de funciones que permiten a un programador especificar los objetos y operaciones para producir salida de audio de alta calidad, específicamente salidas

multicanal de arreglos tridimensionales de fuentes de sonido alrededor de un observador que escucha.

El API de OpenAL esta diseñado para ser multi-plataforma y de fácil uso. Se parece al API de OpenGL por el estilo de sintaxis y sus convenciones. OpenAL sirve para generar audio en un espacio simulado tridimensional. Permitiendo la atenuación relacionada con la distancia y el efecto doppler entre la fuente de sonido y un observador que escucha, produciendo efectos tales como la reflexión, obstrucción, transmisión y reverberación. OpenAL es una máquina de estados que controla un sistema de procesos multicanal para sintetizar un flujo digital, pasando una muestra de datos a través de una cadena de señales parametrizadas de audio digital de operaciones procesadas.

La principal característica que posee esta OpenAL es que es capaz de proporcionar un sonido 3D, es decir el sonido variará dependiendo de donde se encuentre el receptor virtual, de si se mueve, de las cosas que tenga delante, etc. Dicho sonido también dependerá de donde se encuentre la fuente (source) de sonido, de su velocidad, de si se aleja o se acerca, del tipo de sala en la que se encuentre y así como de los objetos que se encuentren entre el observador (listener) que escucha y la fuente de sonido, etcétera.

A continuación se definen los principales conceptos que se utilizan con más frecuencia dentro de OpenAL:

- **Sonido:** El sonido es la vibración de un medio elástico, bien sea gaseoso, líquido o sólido. Cuando nos referimos al sonido audible por el oído humano estamos hablando de la sensación detectada por nuestro oído, que producen las rápidas variaciones de presión en el aire por encima y por debajo de un valor estático. Este valor estático nos lo da la presión atmosférica (alrededor de 100.000 pascals) el cual tiene unas variaciones pequeñas y de forma muy lenta, tal y como se puede comprobar en un barómetro.

El sonido se puede producir diferentes fuentes, desde una persona hablando hasta un altavoz, que es una membrana móvil que comprime el aire generado ondas sonoras

- **Frecuencia:** La frecuencia de un sonido es el número de ciclos de una onda sonora durante un segundo. Se mide en Herzios (Hz). La frecuencia de un sonido se incrementa al aumentar el número de ciclos por segundo. Los sonidos agudos, tales como los producidos por un silbato o una flauta, son de altas frecuencias y contienen miles de ciclos por segundo. Los sonidos graves, tales como los producidos por un trueno lejano o una tuba, son de bajas frecuencias y contienen pocos ciclos por segundo. Las vibraciones que varían entre los 20 y los 20.000 Hz. son percibidas por el oído de las personas oyentes como sonidos

La figura 5.9 muestra cómo el nivel de presión sonora de un sonido de baja frecuencia se forma comparado con un sonido de alta frecuencia de la misma amplitud (intensidad). Debe fijarse en que en el mismo periodo de tiempo, el sonido de baja frecuencia describe un solo ciclo mientras que el de alta frecuencia describe ocho ciclos. Por lo tanto, si el sonido de baja frecuencia representara 1 KHz, el de alta frecuencia representaría 8 KHz.

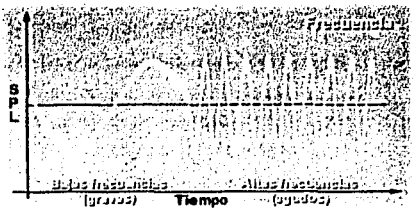


Figura 5.9

- **Intensidad:** La intensidad de un sonido se refiere a la amplitud del sonido y se mide en dB SPL (decibelios de nivel de presión sonora) o en dB IL (decibelios de nivel de intensidad). Cuanto mayor sea la amplitud, mayor serán los dB SPL, y más fuerte será el sonido (por ejemplo, el zumbido de un avión comparado con el sonido de una respiración suave) como se muestra en la figura 5.9.

Un adulto sano es más sensible a los sonidos que se encuentran en un rango de frecuencias que va desde los 500 a los 8000 Hz, lo cual se corresponde con las frecuencias de los sonidos del habla. El sonido más suave (el umbral de audición) que un oído normal puede percibir, se encuentra alrededor de los 0 dB SPL. El sonido más fuerte que puede tolerar (umbral de inconfort) se encuentra entre 120-140 dB SPL.

$$Suma\_de\_dB = 10 * \log\left(\sum_{1}^{n} 10^{\frac{dB}{10}}\right) \qquad 5.2$$

- **Nivel de intensidad:** El decibelio (dB) es la unidad en la que se mide la intensidad de los sonidos. El oído humano tiene una respuesta logarítmica a los estímulos o sonidos que le llegan por lo que la fórmula para obtener la intensidad de dichos sonidos será:

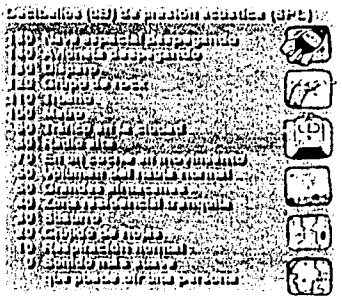


Figura 5.9

dB = 10 * log (I / Io)

con **Io** : intensidad de referencia

Debido a la respuesta logarítmica del oído respecto del sonido se tiene que el nivel de intensidad (en dB) de dos o más fuentes no va a ser igual a la suma de los niveles de intensidades de cada fuente, es decir que por ejemplo dos fuentes de 10 dB de nivel de intensidad no es igual a una fuente de 20 dB sino que es igual a una fuente de 13 dB. La suma de diversas fuentes sigue la siguiente fórmula:

- **Efecto Doppler:** Este efecto es debido principalmente a la velocidad relativa de la fuente de sonido (source) con respecto al observador (listener) y en menor medida a la velocidad del sonido en el medio donde nos encontremos. Lo que produce dicho

efecto es un cambio de intensidad que se manifiesta de la forma siguiente: cuando la fuente sonora se acerca al observador el sonido parece más agudo y cuando la fuente sonora se aleja del observador el sonido parece más grave. Un ejemplo sencillo en el que se nota dicho efecto es cuando llega un tren a la estación y cuando se aleja dicho tren de la estación.

- **Atenuación por la distancia:** Debido al incremento de la separación del observador respecto de la fuente de sonido se produce una disminución del nivel de intensidad **I** de dicha fuente. Esto es debido a que el nivel de intensidad de una **fuente** es proporcional a la potencia de dicha fuente e inversamente proporcional al cuadrado de la distancia (formula 5.3), es decir $I = W/A$ donde W es la potencia en watios y **A** es el área de la esfera que rodea a la fuente (ya que el sonido se propaga en todas las direcciones) y tiene un valor de $4*PI*radio$ al cuadrado. Por lo tanto sustituyendo en la fórmula de niveles de intensidad se puede obtener la atenuación o ganancia de decibelios debido a la distancia:

$$Atenuacion\_o\_ganancia\_de\_dB = 10 * \log\left(\frac{distancia2}{distancia1}\right)^2 \qquad 5.3$$

- **Reverberación:** Es cuando el sonido se refleja en todas las direcciones y con igual modulo y probabilidad, pudiendo llegar al observador una vez que ha terminado de emitirse dicho sonido.

Basado en lo anterior es importante definir con exactitud en OpenAL ciertas variables que contienen los atributos del observador (listener) que escucha. Como se observará más adelante este tipo de definiciones son similares a OpenGL. El *listenerPos* especifica la posición (3, 4, 0) donde se encuentra ubicado el observador con respecto al origen (0, 0, 0). El *listenerVel* se usa en OpenAL para sintetizar el efecto Doppler percibido por el observador para la fuente de sonido, basado en la velocidad (0, 0, 0) de la fuente y el

observador relativo al medio. El *listenerOri* especifica la orientación del observador con respecto a su posición.

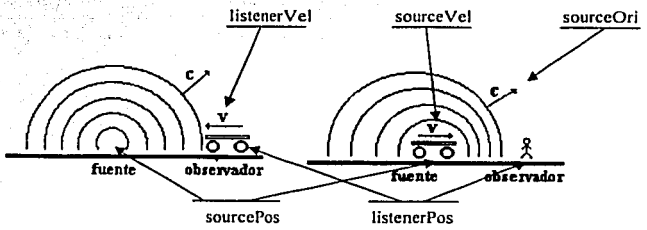


Figura 5.10

El *sourcePos* especifica la posición de la fuente de audio y *sourceVel* sirve también para sintetizar el efecto Doppler percibido por el observador para la fuente de sonido, basado en la velocidad de la fuente y el observador (tanto el observador como la fuente pueden ser relativos).

Capítulo 6

Sincronización del Modelo Facial y Voz

En el presente capítulo se utiliza la información de los capítulos anteriores para hablar de la sincronización del modelo facial y voz.

En la actualidad existe una necesidad para sincronizar varias acciones, tales como las expresiones faciales, los visemas y la voz generada en español mexicano donde el objeto de sincronizar es la de producir una animación realista.

Una animación facial en este sistema está basada en unas capas de especificación múltiple. Consecutivamente las capas más altas definen las entidades de las maneras más abstractas, comenzando con visemas y trabajando a través de las palabras, las frases y las expresiones faciales. Las capas de alto nivel permiten la manipulación abstracta de las entidades de la animación, permitiendo manejar la sincronización de expresiones y visemas con palabras fluyendo de frases.

6.1 Sincronización Facial basada en Capas Abstractas

El problema de la sincronización es descomponer dentro de 5 capas las entidades a utilizar. Cada nivel es una capa independiente con sus propias entradas y salidas de información. Las cinco capas definidas son:

a) Capa 0: Definición de la geometría.

b) Capa 1: Definición de entidades de músculos abstractos.

c) Capa 2: Definición de las expresiones y los visemas.

d) Capa 3: Definición de las palabras y las emociones.

e) Capa 4: Mecanismos de sincronización entre las emociones y la voz.

6.2 Capa 0: Definición de la geometría

Este nivel bajo de abstracción corresponde al modelo facial básico, donde el objeto es mostrar un modelo facial de una cabeza completa como se muestra en la figura 6.1. Donde para mostrar dicho modelo facial se requiere de la entrada de las siguientes bases de datos:

- 'gedalia.msh' con información de la malla
- 'gedalia.bmp' con textura de la piel y 'gedaliaeye.bmp' con textura para los ojos

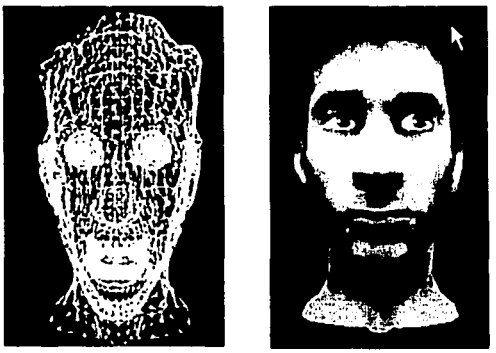


Figura 6.1 Geometría facial y textura de la piel y ojos de una cabeza

Estas fueron modeladas con un editor como se explica en el capítulo 3, con el fin de mostrar y colocar una malla de polígonos triangulares de la superficie de una cabeza en lenguaje C y OpenGL, como se explica en el Anexo C.

6.3 Capa 1: Definición de los Músculos Abstractos

Este nivel corresponde al sistema de animación (ver anexo C.1, C.2 y C.3). Este nivel básico de animación facial está basado en un conjunto de parámetros independientes que controlan la emulación específica de los músculos abstractos de una cara. Dichos músculos en realidad mueven los vértices de los polígonos de la malla. Cada músculo abstracto tiene parámetros tales como valores mínimos, valores máximos (ver sección 3.7.1) y el valor actual de contracción.

En este nivel de abstracción es importante crear una biblioteca de músculos, a partir de la información modelada (como se mostró en el capítulo 3) donde se diseñaran nuevos músculos (ver figura 6.2), que podrán ser agregados o modificados en la base de datos de músculos predefinida y su funcionamiento (como se muestra en el anexo C.5).

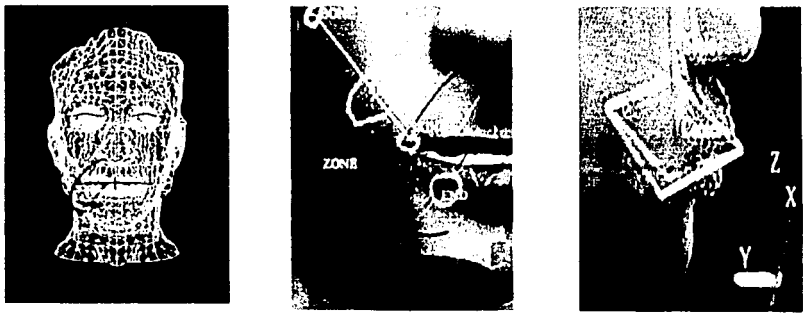


Figura 6.2 Definición de músculos abstractos

6.4 Capa 2: Definición de las expresiones y los visemas

Un *frame* puede corresponder a un visema, a una expresión, o a ambos. Un *frame* es construido a partir de la observación, el uso y de algunas teorías generadas por psicólogos de comunicación no verbal, las cuales permitirán formar las expresiones de alto nivel tales como el enojo, la sonrisa y la sorpresa.

Se pueden crear *frames* de expresiones naturales que muestren la idiosincrasia y rasgos característicos de una cultura. La intensidad de una expresión en un *frame* es directamente determinada por la conjunción de acciones musculares. Una expresión fuerte o débil puede ser creada por cambios apropiados realizados por un editor de expresiones y visemas.

En este nivel de abstracción es importante crear una biblioteca de expresiones. Donde si diseñamos expresiones nuevas, se pueda agregarlas a la base de datos de expresiones predefinida.

La edición de expresiones es por diseño, independiente de la realización a bajo nivel de los músculos y sus acciones. Con esta independencia, la implementación de bajo nivel de una expresión puede ser una simple rotación o cambio de escala de una región o como un complejo modelo de elemento finito 3D. Sería posible usar modelos implementados diferentes para cada caso específico sin efectuar control de alto nivel.

Una vez definido un *frame* se tiene la forma de alguna de las imágenes mostradas en la figura 3.23 y en la figura 4.7. Su manejo y funcionamiento se muestra en el anexo C.4, C.5, C.6 y C.7.

Varios *frames* pueden ser activados al mismo tiempo (ver figura 6.2). Esta actividad sincrónica permite especificar un visema y la sonrisa al mismo tiempo del intervalo.

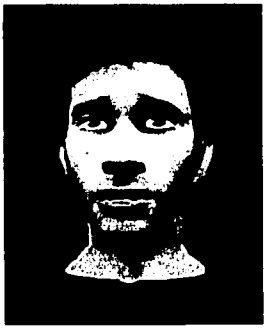


Figura 6.2
Visema + Expresión

Visemas

Un *frame* de un visema (ver sección 4.6) define la posición de la boca y los labios durante la emisión de un alófono en particular. Un visema (ver figura 6.3) esta definido como un conjunto de músculos articulados (ver tabla 4.5) a partir de un modo y un lugar de articulación de cada fonema, tal como se muestra en la figura 4.7.

Figura 6.3
Visema

Lo más importante es que los visemas juegan un papel vital en la animación de sincronización de labios. Cuando se está intentando sincronizar la animación del habla, se requiere entender la duración e inflexión de los fonemas totalmente para poder tomar una decisión informada acerca de qué visema usar y cuándo

Expresiones

Un *frame* de una expresión (ver figura 6.4) es una postura particular de la cara. Estas posturas son generadas por la conjunción de varios músculos o movimientos de la cabeza, mandíbula, ojos y lengua, tal como se muestra en la figura 3.27.

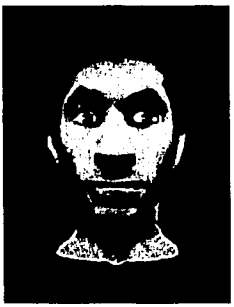


Figura 6.4
Expresión

6.5 Capa 3: Definición de las palabras y las emociones

Las palabras y emociones son definidas como secuencias de *frames*.

Palabras

Una palabra puede ser especificada como una secuencia de fonemas en una pronunciación (*utterance*) durante el habla.

Un problema es determinar la duración de cada fonema relativo a la duración promedio del fonema, basado en su contexto actual. La duración esta influenciada por los fonemas anteriores al actual y al posterior. Varios métodos heurísticos han sido propuestos por investigadores de síntesis de texto a voz (TTS) tal como se muestra en el capitulo 5. Un sistema TTS puede generar una secuencia correcta de fonemas para cada intervalo de tiempo.

Adicionalmente los sistemas TTS pueden usar comandos que controlen la intensidad, duración, y énfasis de cada palabra. Las pausas pueden ser agregadas para controlar el ritmo y entonación de la sentencia.

Emociones.

Las emociones son definidas como cambios de las expresiones faciales dependientes del tiempo en un contexto dado. Cuando una emoción genérica es introducida en un diccionario de emociones es fácil producir instancias para especificar la duración y la magnitud.

6.6 Capa 4: Mecanismos de Sincronización entre las emociones y la voz

En esta capa, el formato de entrada de texto es introducido para especificar cuándo inicia una expresión facial y cuándo termina para posteriormente iniciar la pronunciación de un texto determinado.

La sincronización de diferentes acciones en tiempo real tiene numerosos parámetros que necesitan ser especificados para definir no solo el cambio de un visema en conjunto con el habla, sino además el cambio de la expresión facial, el movimiento de la cabeza como un todo (inclinarse hacía adelante, hacía atrás, girar a la derecha, etc.), pestañeo y movimiento de los ojos, el procedimiento requerido para manejar muchos parámetros es, sin embargo, muy complejo. En esta tesis, usamos el texto (sentencias, cadena de caracteres) como entrada para expresar ambos el contenido de las palabras de la pronunciación "utterance" en el habla, y el control de la información necesaria para representar expresiones y movimientos de la cabeza para generar una emoción a nivel visual.

Formato del texto de entrada.

La figura 6.5 se muestra el formato básico de texto de entrada. Las expresiones faciales y movimientos son representados por *palabras de control* consistentes de palabras en español y que se insertan dentro de '<emocion_ .... >', mientras que la parte subrayada es el *utterance* que será convertido en sonido por el sistema. Al final del texto dentro del guión siempre hay que colocar la palabra de control '<emocion_NEUTRAL>'.

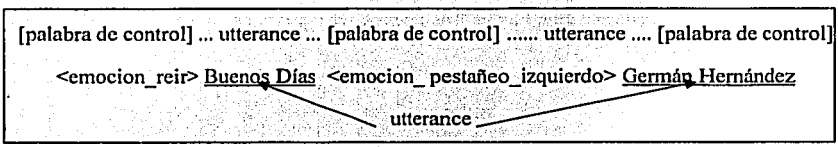


Figura 6.5 Formato de texto de entrada

Palabras de Control

Las palabras de control son en general clasificadas dentro de palabras que se requieren para generar la sincronización entre la voz y los visemas. Dichas palabras de control permiten representar las expresiones faciales y los movimientos de la cabeza. La clasificación de las palabras de control se encuentra en la Tabla 6.1 la cual contiene 7 grupos de la A la F.

Grupo	Palabras de control
A Expresiones básicas	Neutral, Reír, Enojo, Sorpresa, Tristeza, Miedo, Disgusto, Reflejo del disgusto, Seriedad, Apretar labios.
B Expresiones compuestas	Si, no, cara de escéptico diciendo que no, carcajada, mirar alrededor, bostezo, silbido, wahha, dormir, cabeza que gira, mover músculo.
C Estados de ánimo	Feliz, triste, cansado, enojado, asustado y escéptico.
D Movimiento 3-D de la cabeza	Girar a la Derecha y a la Izquierda, Inflar y Succionar las mejillas, Mover mandíbula a la derecha y a la izquierda, Abrir boca.
E Movimientos de Pupilas y Ojos	Pestañeo Izquierdo y Derecho, Cerrar normal y fuerte los ojos, Mirada de Águila, detección de

	movimiento del cursor para las pupilas.
F Visemas del Español	visema_A, visema_E, visema_I, visema_O, visema_U_W, visema_P_B_M_PAU, visema_T_D, visema_K_G, visema_F, visema_S, visema_J, visema_CH, visema_n_N_L, visema_ENIE, y visema_Y_LL_R_RR

Tabla 6.6 Palabras de Control

Este tipo de palabras de control se encuentran dentro de bases de datos (expression.dat y keyframe.dat) como se muestran en la figura 6.6, las cuales fueron construidas como se mencionó en el capítulo 3 y en el anexo C.2, C.4, C.6 y C.7.

Un problema es determinar la duración de cada palabra de control durante diferentes momentos de un diálogo. La duración de una palabra de control esta determinada por la suma de los fonemas de un *utterance* en cualquier parte del diálogo.

Análisis de las palabras de control

El sintetizar expresiones que parecen vivas a partir de un texto de entrada como se describió anteriormente, hace necesario separar individualmente las palabras de control y determinar la función de esta palabra en las series de movimientos musculares o movimientos de la cabeza que serán sintetizadas. Esto se hace en los siguientes pasos:

1. Reconocimiento de las palabras de control.

Las palabras de control individuales son primero separadas de las cadena del texto. Esta separación se comprende al reconocer las palabras de control por medio del formato '<emocion_ .... > y al "utterance" como palabras que se serán convertidas en sonido. Pero, tanto la cadena de fonemas del "utterance" como las palabras de control tienen tiempos deben ser separados. Por un lado, la duración de cada fonema (que será obtenido por un sistema TTS) y el fonema mismo debe ser guardada en un archivo y las palabras de control en conjunto con su tiempo de ejecución (suma de la duración de cada uno de los fonemas) debe ser guardado en otro.

2. Determinación de valores de los parámetros para la expresión de un keyframe.

Los parámetros de la expresión específicos son determinados por el key-frame basado en las palabras de control reconocidas. En el principio, la palabra de control permanece constante hasta que un tipo similar de palabra de control aparece después de cada "utterance". Para encontrar los valores que determinan una expresión y generar su *keyframe* determinada se realiza una comparación y búsqueda entre las expresiones (bibliotecas *expression.dat* y *keyframe.dat*) y las palabras de control (ver tabla 6.1).

Por otro lado, si se desea generar un nuevo *frame* denominado *keyframe* a partir de una expresión (palabra de control) y un visema, lo primero que hay que obtener son sus valores tanto de uno como de otro para posteriormente sumar ambos valores y generar una expresión nueva, la cual formará el nuevo *keyframe*.

En este nivel de abstracción alto es importante remarcar que al hablar de sumar expresiones en realidad estamos hablando de sumar músculos y movimientos de la cabeza, ojos, mandíbula y lengua. Pero aún más lo que estamos sumando en realidad son la colocación de vértices en un cierto lugar en el espacio 3D en el nivel más bajo.

Para producir la sincronización entre la voz y las imágenes faciales es importante tomar en cuenta cuánto tiempo debe durar una imagen facial que corresponda a la entrada de un texto. Por lo tanto, el tiempo de duración de una imagen facial con respecto a un texto es determinado a partir de la suma del tiempo de duración de todos los fonemas y etapas de silencio en términos de milisegundos.

La posición de los *keyframes* para representar una expresión dada en un tiempo dado es entonces determinada según las *palabras de control*. Si una *palabra de control* aparece en el formato "<palabra de control .A.> - <palabra de control .B.> Buenos Días <palabra de control .C.> " en parte del habla del texto, esta palabra de control es básicamente eficaz del punto al

cual el habla cambia de una *<palabra de control .A.>*. a *<palabra de control .B>*. La forma de los parámetros para los *frames* intermedios entre los *keyframes* inmediatos se encuentra por interpolación (ver sección 3.9) de los valores de los parámetros para los keyframes.

Con respecto a las formas de la boca o visemas, los valores del parámetro son igualmente determinados analizando los fonemas a 1/30 segundos intervalos. Es decir, el visema que corresponde al fonema a cada 1/30 segundo que se asigna básicamente a cada *frame*. Pero la variación de cada visema con respecto al tiempo de duración de un fonema no es el mismo, como para obtener una transición y natural de los visemas encima de los fonemas consecutivos. Para ello se tienen que aproximar y sumar la duración de cada fonema con el tiempo de despliegue de los *frames* intermedios entre un visema y otro, al igual que entre una expresión y otra.

Capítulo 7

Implementación

La implementación del Sistema de Sincronización de Expresiones, Visemas y Voz en español, se desarrollo en dos partes:

1. Principales componentes, problemas, usos del sistema y como conviven los componentes del sistema implementado a nivel muy general.
2. La arquitectura del sistema y la forma en que se realizó la implementación.

7.1. Principales componentes, problemas y usos del sistema

Para poder implementar el Sistema de Sincronización de Expresiones, Visemas y Voz en español se realizó un análisis contextual de lo que el sistema debe de hacer como primera etapa. Esto supone buscar, analizar y utilizar tecnologías adecuadas de fácil obtención (free ware) y manejo de interfaces API's con funciones de alto nivel para C/C++.

El sistema fue programado en Lenguaje C; se utilizaron las bibliotecas de OpenGL, GLUT, GTK/GDK, GLArea, Festival y OpenAL. El compilador y editor que se utilizó para codificar el programa fue Visual C++ 6.0.

El sistema tiene dos ambientes: uno interno y otro externo. El ambiente interno se creó conjuntado y utilizando las herramientas de cada uno de los sistemas independientes como las bibliotecas antes mencionadas, las cuales tienen una interacción (interfaces API's en C/C++) entre ellas. A estos sistemas independientes se les denomina componentes, que pueden tener modificaciones hechas por el programador y la unión de estos componentes es el ambiente externo del sistema.

A continuación se describe brevemente cada uno de los componentes y sus problemas (figura 7.1), los cuales son indispensables para producir la sincronización entre expresiones, visemas y voz:

- **Diálogo**

  Este es un archivo de texto que funge como guión para producir la sincronización y el texto como se explicó en la sección 6.1. El diálogo sirve para alimentar al sistema por medio de una base de datos 'cajadetexto.th'.

- **TTS: Festival-MBROLA**

  Uno de los principales problemas en la investigación de sistemas de animación y voz en tiempo real es incorporar sistemas completos de síntesis de voz (sección 5.1) a una aplicación. Mediante el análisis semántico del texto se obtienen las palabras de control para desplegar las expresiones faciales y los fonemas en archivo de sonido. Festival (Sección 5.2) permite este tipo de integración gracias a su manejo de API's con funciones de alto nivel para C/C++, lo cual ofrece una interfaz entre Scheme y el Shell, que permite incorporar a Festival al sistema.

  El uso de MBROLA (ya descrito en la sección 5.3.4), es con el fin de mejorar la calidad de audio en el archivo de sonido que se genera.

- **OpenAL**

  Otro gran problema al que se hace frente al investigar la manera de cómo sincronizar las imágenes faciales y el sonido en tiempo real es como dar la impresión de trabajo en paralelo con ambos, porque al tratar de sincronizar el audio con la animación, se encontró que se ejecutaba primero la animación y al terminar iniciaba el audio o viceversa. Al buscar una solución se encontró que OpenAL permite trabajar e interactuar casi al mismo tiempo con la animación, gracias a su manejo de API's con funciones de alto nivel para el lenguaje C.

  Este módulo recibe y procesa el archivo WAVE que es enviado por el ambiente interior del sistema produciendo una salida de audio de alta calidad (como se explica en la sección 5.4) por medio de la estructura de datos **source[0]**.

- **Modelo Facial**

  Uno de los problemas con la sincronización al momento de generar el Modelo Facial es la forma de sincronizar los *visemas* (ver figura 4.27) y las *expresiones faciales* (ver figura 3.26). La forma de solucionar este problema es utilizando los conceptos de máquina de estados, interpolación y manejo de eventos, lo cual permite juntar lo anterior en un solo *frame* y después mandar la orden para desplegarlo.

  Este módulo recibe la orden para crear o modificar un actor denominado **brandy,** es decir, crear o modificar una cabeza la cual en realidad esta organizada como estructura de datos dinámica, misma que recibe la información relativa a la geometría, textura, músculos, expresiones faciales básicas y compuestas, y *visemas* para la boca de una cabeza modelada.

- **OpenGL**

  Este módulo recibe instrucciones precisas del módulo externo **Modelo Facial,** el cual le indica el tipo de objetos y operaciones a realizar para producir modelos faciales

gráficos y especificar las imágenes (textura) a color de objetos tridimensionales de un rostro humano. En este proceso OpenGL interactúa constantemente con el hardware gráfico para realizar sus cálculos y enviar sus resultados al *FrameBuffer*.

- **GLUT**

  Este módulo recibe instrucciones precisas de OpenGL y algunos dispositivos de entrada como son el teclado y el ratón, pero en general GLUT (OpenGL Utility Toolkit) es un programa de interfaz (ver anexo C) entre ANSI C y OpenGL, el cual permite al sistema mostrar una ventana para desplegar (rendering) imágenes del *FrameBuffer* enviados por OpenGL en un monitor.

- **Dispositivos de Entrada y Salida**

  En este módulo se tienen en constante detección los dispositivos de *entrada manejados por GLUT* como son las teclas del *teclado*, los *botones* y movimiento del *ratón* para poder interactuar constantemente con la aplicación. Por otro lado, el sistema requiere de dispositivos de salida, que son manejados por *OpenAL*, como son las *bocinas* de sonido que desplegaran el audio de una voz masculina mexicana y por último, también se requerirá de otro dispositivo de salida manejado por GLUT como es el monitor, él cual desplegará las imágenes faciales generadas por el sistema de una cabeza humana articulada.

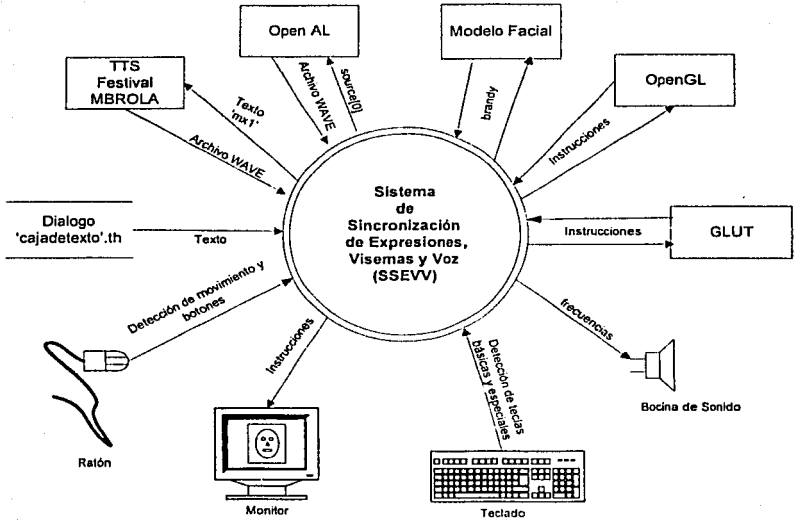


Figura 7.1 Diagrama de Contexto

7.2. Arquitectura del sistema de sincronización de expresiones faciales, visemas y voz

La arquitectura del sistema (ver la figura 7.2) tiene como objetivo el automatizar la posición de las expresiones faciales y los visemas (labios, mandíbula y lengua), los cuales deben estar relacionados con la voz (sonido). Esta animación se genera en tiempo real y se puede modificar de acuerdo a las necesidades del usuario por medio de un archivo de texto como entrada. También tiene una interacción entre el sonido y la imagen que se van generando a través de los módulos.

A continuación se explicará la forma en que se implementó el sistema, así como la manera en que interactúan cada uno de los módulos externos.

- **Diálogo**

  Se alimenta al sistema por medio del archivo denominado **'cajadetexto.th'**, el cual puede ser modificado por un editor de texto las veces que se considere necesario dependiendo de lo que se desee que realice la animación. Por ello el tipo de guión que se utilice es muy importante para que la animación sea más realista. Cabe destacar que el diálogo es el primer paso dentro de la sincronización, dado que funge como el guión de un actor.

- **TTS: Festival-MBROLA**

  Este módulo realiza un análisis léxico y prosódico (Festival) del texto produciendo dos bases de datos; el primero es **th.ph** que contiene información sobre los fonemas y sus tiempos del texto hablado en el guión y el segundo es **th.com** que contiene información sobre las palabras de control y sus duraciones, por otra parte este mismo módulo realiza la síntesis del habla por medio del sintetizador MBROLA produciendo un archivo de sonido con formato WAVE.

- **OpenAL**

  Este módulo recibe y procesa el archivo WAVE, mismo que es enviado por el ambiente interior del sistema produciendo una salida de audio de alta calidad (como se explicó en la sección 5.4) por medio de la utilización de la estructura de datos **source[0]**, la cual en realidad es la organización de la información como estructura de datos del archivo WAVE.

- **Sincronización**

  Este módulo le dice a **alSourcePlay** que inicie el despliegue de sonido y al mismo tiempo recibe los archivos de datos **th.ph** y **th.com**, los cuales son procesados con el objeto de generar un *frame* conjuntado los movimientos de la cabeza, las expresiones

faciales, el movimiento ocular y los visemas de la boca para posteriormente modificar al actor **brandy**.

- **Modelo Facial, OpenGL y GLUT**

Este módulo recibe y guarda la orden para crear o modificar el actor denominado **brandy**, es decir, una cabeza que, en realidad, es la organización de una estructura de datos dinámica que recibe la información relativa a la geometría, textura, músculos, expresiones faciales básicas y compuestas, y visemas para la boca de una cabeza modelada (como se explica en los capítulos 3 y 4); para posteriormente llamar al módulo **OpenGL** y proyectar sobre una ventana en un monitor las imágenes faciales construidas vía los módulos de **OpenGL y GLUT**. Por último, dentro de la sincronización se tiene ya implementado el movimiento ocular, que se da a partir de la detección del movimiento del cursor vía el ratón, **OpenGL** y **GLUT** durante el proceso de animación.

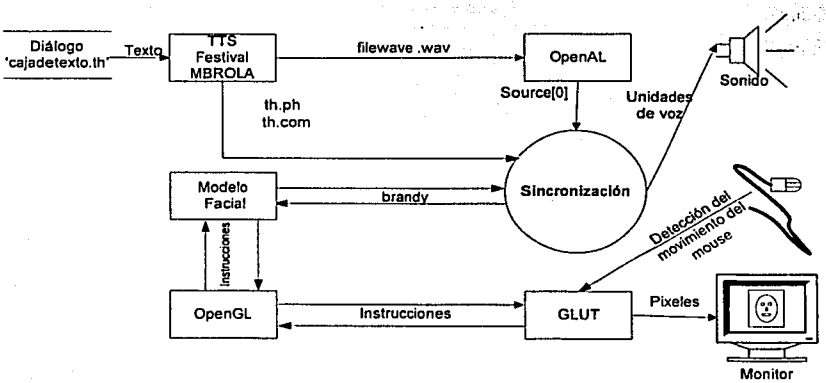


Figura 7.2 Arquitectura del sistema

7.3 Módulo TTS: Festival-MBROLA

En esta subsección explicaremos el módulo *TTS: Festival-MBROLA* que se muestra en la figura 7.2, la cual se encarga de realizar una fase de inicialización y carga del Sistema Festival, con ello se procede a cargar las funciones en Scheme **thmode.scm y mbrola.scm** (ver Anexo B) las cuales servirán para realizar TTS más adelante. El siguiente paso es cargar la voz en español con acento mexicano, que permitirá a **TTS** realizar un análisis léxico (el conjunto de fonemas y reglas de letras a sonido) y prosódico (entonación y parámetros de duración) del mismo, el cual es un método que permite generar un conjunto de valores apropiados para los parámetros que requiere todas las sub-partes del sistema FESTIVAL, incluyendo parámetros léxicos y parámetros prosódicos. Por último, se evalúa el archivo **caja_de_texto.th** y se le aplica **TTS** produciendo dos bases de datos **th.com y th.ph** con información de fonemas y otras características, y en conjunto con la base de difonemas **mx1** se genera un archivo con formato WAVE.

Básicamente el algoritmo que se requiere para realizar TTS es como se describe a continuación (figura 7.3):

- Inicializar y cargar el sistema Festival, para leer los datos en Scheme. A continuación se encuentra un fragmento del código en C, que muestra esta parte del proceso.

```
void generate_PHONEMES_WAVE(int inicia)
{
        int heap_size = 210000;
        int load_init_files = 1;
        festival_initialize(load_init_files, heap_size);
```

- Cargar las funciones **th-mode.scm y mbrola.scm**

festival_eval_command("(load \"data_file/th-mode.scm\")");

- Cargar la voz **abc** de difonos de un hombre en el idioma español

  La Voz de difonos **abc** es un conjunto de funciones en Scheme básicos que necesita el sistema para realizar un análisis léxico y prosódico del español, mismo que es un método que permite generar un conjunto de valores apropiados para los parámetros que requiere todas las sub-partes del sistema FESTIVAL, incluyendo parámetros léxicos (el conjunto de fonemas y reglas de letras a sonido) y parámetros prosódicos (entonación y parámetros de duración).

  Por lo que, si se desea utilizar una voz diferente a la actual para realizar el análisis, se requerirá seleccionar otra voz de la biblioteca (LIB) de Festival.

  festival_eval_command("(voice_abc_diphone)");

- Evalúa el archivo **caja_de_texto.th** y aplica TTS.

  festival_eval_command("(tts \"data_file/caja_de_texto.th\" 'th)");

  festival_wait_for_spooler();

Para comprender como se modificaron los archivos en Scheme fue necesario entender como Festival soporta la noción de modos de texto donde el tipo de archivo de texto puede ser identificado, permitiendo a Festival procesar el archivo de texto en una forma apropiada. Actualmente solo dos tipos de formatos dentro del sistema Festival no presenta cambios entre versiones y son considerados estables: STML y raw, pero otros tipos de formatos son también utilizados dentro de Festival los cuales tienen la desventaja que tienen que ser actualizados o adaptados, tales como email, HTML, Latex, th, etc. Los *modos* son especificados como el tercer argumento de la función ***tts.*** Entonces para sintetizar un texto para nuestro caso en particular se requiere el archivo **th-mode.th** (anexo A) mediante el uso de la siguientes dos líneas en *Scheme:*

(load "data_file/th_mode.scm")

(tts "data_file/caja_de_texto.th" 'th)

Las modificaciones hechas en Festival se basan principalmente en dos archivos escritos en Scheme **th-mode.scm** y **mbrola.scm** (ver Anexo A). Sus principales funciones, se comentarán a continuación.

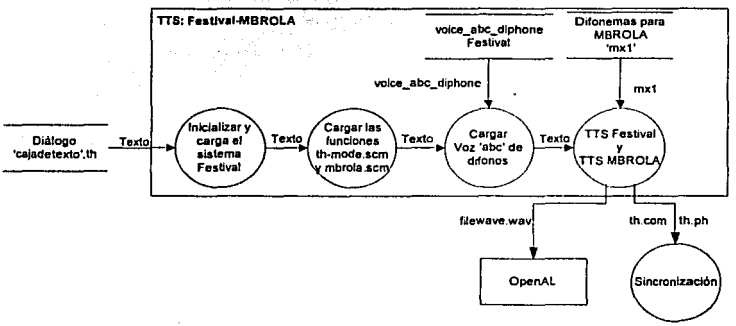


Figura 7.3 TTS: Festival-MBROLA

7.4 Módulo TTS Festival y TTS MBROLA

En esta subsección explicaremos el módulo TTS Festival y TTS MBROLA que se muestra en la figura 7.3, la cual se encarga de realizar una evaluación y aplicación de los módulos Festival y MBROLA para generar las *palabras de control*, el **"***utterance***"** (estructura de datos que contiene toda la información de la pronunciación de una voz) y el archivo WAVE.

TTS Festival- En este apartado se tiene que Festival vía el archivo Scheme **thmode.scm** realiza el análisis del **"utterance"**, por medio del módulo **análisis del diálogo**, el cual realiza un análisis previo para separar las palabras de control

y el texto hablado o frase. Al finalizar dicho análisis la frase se guarda en una estructura de datos denominada **"utterance"** y las palabras de control son enviadas al módulo **almacenar palabras de control y tiempos.** Estos tiempos son calculados con base a la duración de los fonemas y al final se genera la base de datos **th.com.** Por otro lado, el **"utterance"** es recibido por el módulo **análisis del texto hablado o evalúa y aplica TTS frase,** el cual realiza un segundo análisis sintáctico para obtener las características del **"utterance".**

Posteriormente se pasan a dos módulos; el primero tiene el objetivo exclusivamente de **almacenar fonemas y duración** a partir del **"utterance"** para generar el archivo **'th.ph'**. En el segundo módulo **almacenar en formato MBROLA** se utilizan las funciones del archivo en Scheme 'mbrola.scm' para realizar un análisis del **"utterance"** para adecuar sus características (fonemas, duración, características de los segmentos y suprasegmentos) y generar el archivo **f_phone.pho** con formato **MBROLA.**

TTS MBROLA- En este otro apartado se tiene que el sintetizador externo **MBROLA** recibe la base de datos **f_phone.pho** ya antes procesada y al mismo tiempo recibe una base de datos de difonemas **mx1** de una voz masculina mexicana con el objetivo de sintetizar un archivo de sonido con formato WAVE **file_wave.wav.**

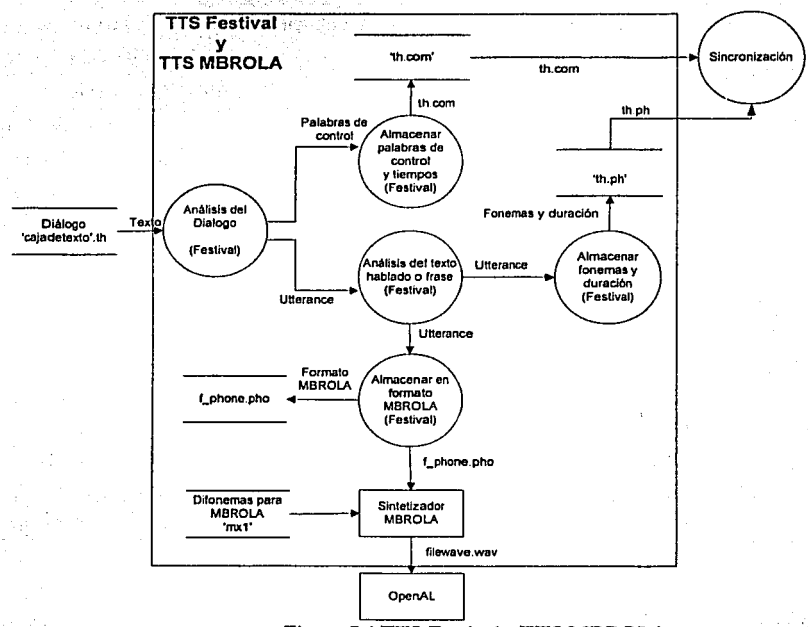


Figura 7.4 TTS Festival y TTS MBROLA

7.5. Análisis del texto hablado o frase (Festival)

En esta subsección explicaremos el módulo *Análisis del texto hablado o frase (Festival)* que se muestra en la figura 7.4. La forma en la cual trabaja el módulo Festival (sección 5.3), en particular para el español que se habla en México fue elaborado por Barbosa (OGI). La figura 7.5 muestra la entrada de una estructura **"utterance"** (utt) que al ser procesada y analizada por funciones Scheme del núcleo del sistema Festival alimenta con información a la estructura de datos del mismo **"utterance"**. A continuación se describirá paso a paso el proceso de síntesis que se realiza en el idioma español por Festival.

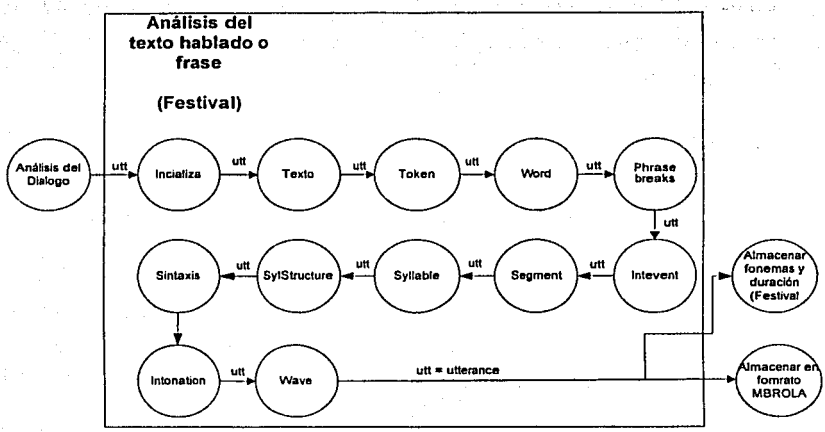


Figura 7.5 Análisis del texto hablado

- **Inicializa**

  Esta etapa normalmente carga las relaciones necesarias para el formato de entrada y borra todas las relaciones anteriores para la síntesis.

- **Texto**

  Esta etapa guarda el texto como una cadena de caracteres a ser analizados.

- **Token**

  Esta etapa esta organizada en una estructura de datos como una lista de árboles. Es formada como una lista de tokens encontrados en una cadena de texto de caracteres.

- **Word**

  Esta etapa esta organizada en una estructura de datos como una lista de palabras. Estos *items* también aparecerán como hijos (nodos hoja) de las relaciones "*Token*". Ellos

también pueden aparecer en la relación de *sintaxis* (como hoja) si se utiliza la probabilidad *parser*. Ellos también serán organizados como hojas de la relación "*Phrase breaks*".

- **Phrase breaks**

  Esta etapa esta organizada en una estructura de datos como una lista de árboles. Esto es una lista de frases raíz cuyos hijos son las *Word* dentro de esas frases.

- **Sintaxis**

  Esta etapa esta organizada en una estructura de datos como un árbol único. Esto significa que si la probabilidad es llamada, es un árbol de bifurcación binario sintáctico encima de los miembros de la relación Word.

- **SylStructure**

  Esta etapa esta organizada en una estructura de datos como una lista de árboles. Esto liga la palabra, la silaba y las relaciones de *segment*. Cada *Word* es la raíz de un árbol cuyos hijos inmediatamente son las silabas y sus hijos en turno son los *segmentos*.

- **Syllable**

  Esta etapa esta organizada en una estructura de datos como una lista de *syllables* (sílabas). Cada miembro está en la relación *SylStructure*. En esa relación su padre será la *Word* y sus hijas serán los *segmentos* que están en él. Las *Syllables* están también en la relación entonación dadas las ligas a su relación de eventos de *intonation* (entonación).

- **Segment**

  Esta etapa esta organizada en una estructura de datos como una lista de segmentos (fonemas). Cada miembro (excepto silencios) será nodo hoja en la relación de

SylStructure. Estos pueden también estar en la relación Target (blanco) ligada a los puntos *F0 target*.

- **IntEvent**
  Esta etapa esta organizada en una estructura de datos como una lista de eventos de entonación (acentos y *bounaries*). Estos están relacionados a las *syllables* por la relación de *Intonation* como hoja en que relación. Así su padre en la relación de *Intonation* es la *syllable*. Estos eventos son adjuntos.

- **Intonation**
  Esta etapa esta organizada en una estructura de datos como lista de árboles relacionando las *syllables* a eventos de *intonation*. La raíz del árbol es la *Intonation* son las *Syllables* y sus hijos son *IntEvents*.

- **Wave**
  Esta etapa genera un *item* único con una característica llamada *wave* (onda) cuyo valor es la generación *waveform*.

7.6. Módulo de OpenAL

En esta subsección explicaremos el módulo *OpenAL* que se muestra en la figura 7.2, la cual se encarga de recibir un archivo de sonido **filewave.wav** y producir una salida de tipo **source[0]**.

A continuación se describirá brevemente el proceso que se lleva a cabo para procesar un archivo con formato WAVE (ver anexo D) y prepararlo para su ejecución (figura 7.6).

El primer paso es inicializar las variables a utilizar en *OpenAL* (anexo D) y preparar un manejador de errores en el caso de que alguno de los pasos que se llevan a cabo dentro de *OpenAL* genere error. Así mismo, se generan las fuentes (**source[0]**) de sonido, que en este caso es solo una y los entornos (**environment**) necesarios para esa fuente de sonido. Como

siguiente paso es necesario definir las propiedades del observador (**listener**) y de la fuente (**source[0]**) definiendo la posición, la velocidad y la orientación de estos. Además es necesario cargar el archivo **filewave.wav** de audio con formato WAVE para producir la información sobre las características del formato (format) del archivo, datos (data) del archivo, tamaño (size), frecuencia (freq) y loop que es una variable booleana que se inicializa como falso para producir una continua ejecución del archivo. En este proceso es importante asociar a cada uno de los Buffers una fuente (source[0]) de sonido. Así como, especificar si cada una de las fuentes de audio debe tocarse una sola vez o deben ejecutarse de forma cíclica (loop) mientras este vivo el sistema. Por último, es necesario designar la fuente a utilizar y asociar las propiedades del mismo al entorno y la fuente de sonido preparando a **source[0]** para una posible ejecución.

La incorporación y la manera de operar *OpenAL* dentro del sistema se comenta a continuación.

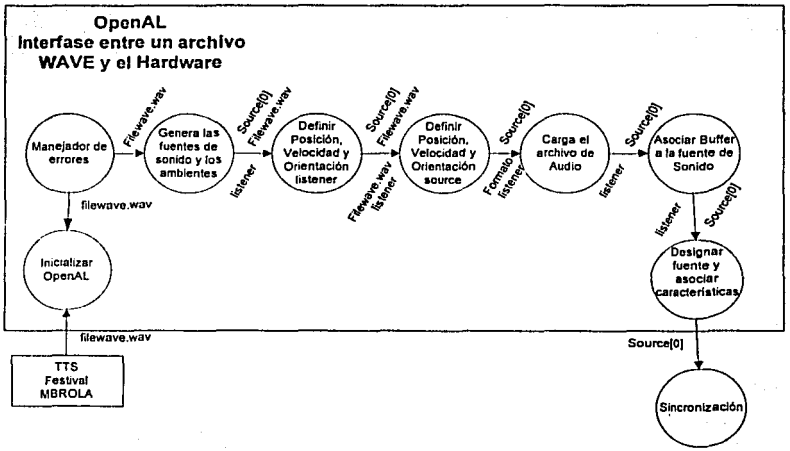


Figura 7.6 OpenAL

7.6.1 Incorporación de OpenAL al sistema de sincronización

Para poder integrar OpenAL (ver sección 5.4) a un sistema de animación es necesario conocer tanto las bibliotecas como las funciones que componen OpenAL, descritas a continuación. Los archivos que contienen la declaración de las funciones contenidas en la biblioteca de OpenAL que se agregaron al sistema de animación fueron:

```
#include <al.h>

#include<alc.h>

#include<alut.h>
```

Se definirán tres variables. Las dos primeras definirán una o varias fuentes (sources) de sonido a las cuales se les asociara con uno o varios Buffers donde se almacenará un archivo de audio. La última variable a definir es el número de ambientes (environment) usados en nuestra aplicación. Un ambiente es un lugar donde se especifican cosas como reverberación y reflexión de sonidos.

```
#define NUM_BUFFERS 1

#define NUM_SOURCES 1

#define NUM_ENVIRONMENTS 1

ALuint buffer[NUM_BUFFERS];

ALuint source[NUM_SOURCES];

ALuint  environment[NUM_ENVIRONMENTS];
```

Posteriormente se creo una función en lenguaje C:

*void carga_archivo_wav (char *file_WAV);*

La cual tendrá el objetivo de cargar un archivo WAV y prepararlo para su posterior reproducción. Los pasos que describen a dicha función son:

1. Definir e inicializar ciertas variables que contienen los atributos de nuestro observador (listener) que escucha. Como se observará más adelante este tipo de definiciones son similares a OpenGL. El *listenerPos* especifica la posición (3.0, 4.0, 0.0) donde se encuentra ubicado el observador con respecto al origen (0, 0, 0). El *listenerVel* se usa en OpenAL para sintetizar el efecto Doppler percibido por el Observador para la fuente de sonido, basado en la velocidad (0, 0, 0) de la fuente y el observador relativo al medio. El *listenerOri* especifica la orientación del observador con respecto a su posición.

   El *sourcePos* especifica la posición de la fuente de audio y *sourceVel* sirve también para sintetizar el efecto Doppler percibido por el observador para la fuente de sonido, basado en la velocidad de la fuente y el observador (tanto el observador como la fuente pueden ser relativos).

   ALfloat listenerPos[ ]={2.0,3.0,4.0};

   ALfloat listenerVel[ ]={1.0, 1.0, 1.0};

   ALfloat listenerOri[ ]={0.0, 1.0, 0.0, 0.0, 0.0, 1.0};

   ALfloat sourcePos[ ]={ 0.0, 4.0, 0.0};

   ALfloat sourceVel[ ]={ 3.0, 4.0, 0.0};

2. Inicializar OpenAL
   alutInit (NULL, 0);

3. Agregar los atributos del sujeto que escucha ya antes definidos por medio de la función

alListenerfv:

alListenerfv(AL_POSITION, listenerPos);

alListenerfv(AL_VELOCITY, listenerVel);

alListenerfv(AL_ORIENTATION, listenerOri);

4. Generar los Buffer asociados a cada una de las fuentes.

   alGenBuffers(NUM_BUFFERS, buffer);

5. Cargar un archivo (file_WAV) de audio WAVE por medio de la función *alutLoadWAVFile* y obtener información sobre el formato (format) del archivo, datos (data) del archivo, tamaño (size), frecuencia (freq) y loop que es una variable booleana que se inicializa como falso.

   alutLoadWAVFile(file_WAV, &format,&data, &size, &freq, &loop);

6. Agregar los datos del archivo al buffer, por medio de la función:

   alBufferData(buffer,format,data,size,freq);

7. Liberar la información de los datos del archivo de sonido.

   alutUnloadWAV(format,data,size,freq);

8. Generar las fuentes de sonido en el ambiente:

   alGenSources(NUM_SOURCES,source);

9. Definir e inicializar ciertas variables que contienen los atributos de las fuentes (source) de sonido:

alSourcef (source[0], AL_PITCH, 1.0f);

alSourcef (source[0], AL_GAIN, 1.0f);

alSourcefv(source[0], AL_POSITION,source0Pos);

alSourcefv(source[0], AL_VELOCITY,source0Vel);

10. Asociar a cada uno de los Buffers una fuente (source) de sonido. Además, especificar si cada una de las fuentes de audio deben tocarse una sola vez o deben ejecutarse de forma cíclica mientras este vivo el sistema.

    alSourcei (source[0], AL_BUFFER, buffer[0]);

    alSourcei (source[0], AL_LOOPING, AL_FALSE);

7.7. Módulo de Sincronización

En esta subsección explicaremos el módulo *sincronización* que se muestra en la figura 7.2, el cual es el módulo más importante y porque aquí es donde se genera el proceso que va a hacer interactuar a todos los otros módulos, es decir, se hace la completa fusión entre lo que es el ambiente interno y el externo.

Este módulo de *sincronización* (ver figura 7.7), es un programa que hace que interactúen todos los módulos descritos en las secciones anteriores. Aquí se procesa la información dada en el diálogo que a través de su paso por cada uno de los módulos se generan archivos que al final se juntan en este último.

Se inicia con un archivo de texto que al pasar por las diversas etapas va llamando a los programas que procesan la información y lo regresan en archivos y en una variable (source[0], th.com y th.ph) que puedan ser utilizados por las variables que maneja el sincronizador.

Así este sincronizador hace que la información inicial recorra un camino de abstracción que va desde el despliegue de la malla de una cabeza, utilización de la textura, la selección de las emociones y su expresión facial, la animación, y la utilización de la expresión adecuada, ya codificada por una palabra de control, a través de bibliotecas para la generación del modelo facial que interpreta nuestro texto inicial.

Esto se hace por medio de capas que ya fueron descritas en el capítulo 6. Así a través de un texto escrito inicialmente se llega a la creación de un modelo facial que interprete dicho texto por medio de expresiones faciales y sonidos.

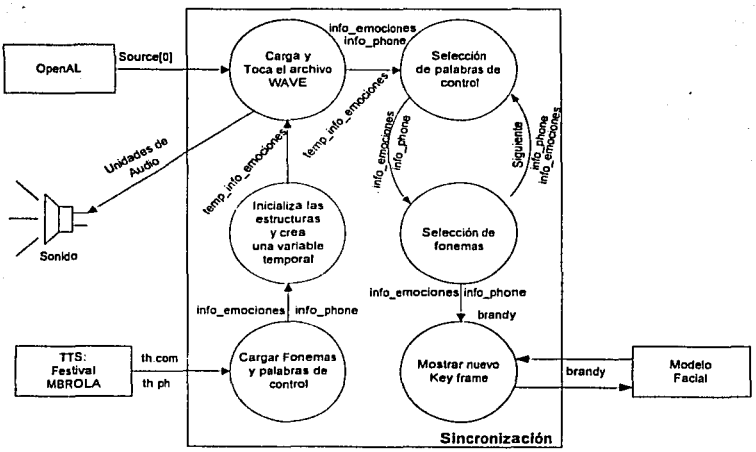


Figura 7.7 Sincronización

Este módulo recibe dos bases de datos **th.com** y **th.ph** los cuales son cargados y guardados en dos estructuras de datos para manipular la información y por otro lado se recibe la estructura de datos **source[0]** y en conjunto será todo procesado para generar la sincronización de la

cabeza produciendo cambios en la información de la estructura del actor **brandy** y generando el audio.

- **Cargar fonemas y palabras de control.**

  En esta etapa del proceso se cargan dos archivos **th.com** y **th.ph**, los cuales son leídos por las siguientes dos funciones que fueron creadas en Lenguaje C:

```
cargar_fonemas();                    // Carga archivo 'th.ph'
carga_comandos_de_control();         // Carga archivo 'th.com'
```

  El objeto de estas dos funciones es guardar la información de dichos archivos en dos listas que tienen las siguientes estructuras:

```
typedef struct DataPhone   // Estructura para guardar información en 'th-ph'
{
        char phone[3];              // Fonema
        float duration;             // Tiempo de duración del fonema
        int stress;                 // del fonema
        int F0WordPos;              // del fonema
        DataPhone *sig;             // apuntador a la siguiente entrada
} DataPhone;

DataPhone *primero, *ultimo;

typedef struct ControlExpression          // Estructura para guardar información
                                          en 'th.com'
{
        char  name[31];             // Palabra de Control
        float time;                 // tiempo de duración de la palabra de control
```

```
        ControlExpression *sig;      // apuntador a la siguiente entrada
} ControlExpression;

ControlExpression *pri_words, *ult_words;
```

- **Inicializa las estructuras y crea una variable temporal**

```
info_phone       = primero;   // Inicializa la lista de fonemas
info_emociones   = pri_words; // Inicializa la lista de Comandos de Control
temp_info_emociones = pri_words; // Genera un apuntador temporal
```

- **Carga y toca archivo WAVE**

En esta etapa, se carga el archivo WAVE al sistema por medio de **OpenAL**, obteniendo la estructura **source[0]** y la función que permite realizar lo anterior es:

```
carga_archivo_wav ("data_file/file_wave.wav"); // cargar el audio
```

Como segunda etapa del proceso, se procede a mandar las unidades de audio de **source[0]** a las bocinas para ser escuchadas, como se muestra a continuación:

```
if (prender_audio == 1 )
{
   printf ("\n..... ** Audio prendido ** .....\n"); // Audio prendido
   alSourcePlay(source[0]);
}
else
{
   alSourceRewind(source[0]);          // Volver a encender
   alSourcePlay  (source[0]);
```

```
    printf("\n..... **alSourceRewind(source[0]) **\n");
}
```

- **Selección de palabras de control y fonemas**

En esta etapa del proceso, se hace frente al problema de cómo seleccionar y sincronizar, este tipo de situación se resolvió utilizando un guión (texto) el cual selecciona fonemas y palabras de control, los cuales permiten sincronizar (ver sección 6.6) como se muestra en la figura 7.8. Dado que se busca una determinada intencionalidad en el modelo facial se necesita que ambas, la expresión visual y el sonido tengan la misma duración. Para que eso ocurra se necesita crear variables que permitan que la duración de la expresión facial sea la misma que la duración de la suma de los tiempos de los fonemas, es decir, el tiempo que dura una palabra de control a través de los visemas y los alófonos.

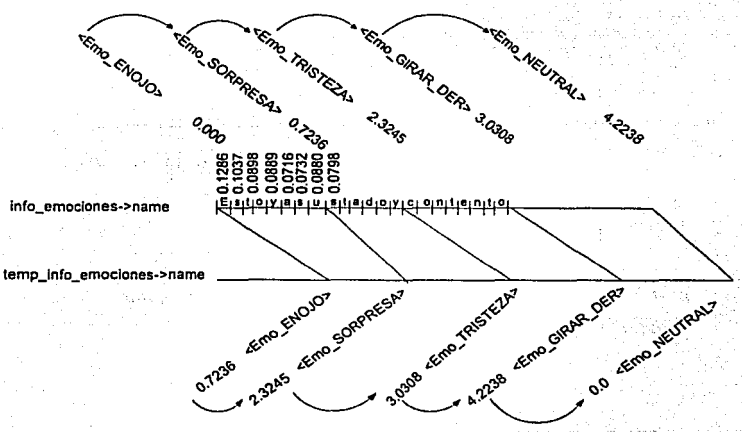


Figura 7.8

Las variables denominadas **temp_info_emociones** y **suma_dur_fon** permiten realizar una comparación entre la suma de las duraciones de los fonemas **suma_dur_fon** y el tiempo asignado a una palabra de control **temp_info_emociones->time** permite ir recorriendo tanto a los fonemas como las palabras de control en un guión.

El código fuente que se implementó para mostrar lo anterior es el siguiente:

```
while(info_phone->phone != "\0" && strncmp( temp_info_emociones->name,
"<Emo_NEUTRAL>", 17) != 0)
    {
        temp_info_emociones = temp_info_emociones->sig;   // Apuntador a la
                                                              palabra de
control

        if ( temp_info_emociones->time > 0 && strncmp( temp_info_emociones-
>name, "<Emo_FIN>", 13 ) != 0)
            while ( suma_dur_fon < temp_info_emociones->time )
            {
                sincronizacion_visema_Emociones ( info_phone, info_emociones );
                suma_dur_fon = suma_dur_fon + info_phone->duration;
                printf("*");

                info_phone = info_phone->sig;        // Obtener el siguiente fonema
            }

        // Obtener la siguiente emoción
        info_emociones = info_emociones->sig;
```

```
        if ( strncmp( temp_info_emociones->name, "<Emo_NEUTRAL>", 17) == 0
&& (strncmp( info_phone->phone, "_", 2 ) == 0))
        {
            sincronizacion_visema_Emociones ( info_phone, info_emociones );
}   // end_while
```

- **Mostrar nuevo Key frame**

Esta etapa tiene el objetivo de mostrar un nuevo *frame* con respecto a la sincronización de los visemas, las expresiones faciales, el movimiento de ojos y de la cabeza. La función creada para esto es:

```
sincronizacion_visema_Emociones ( info_phone, info_emociones );
```

La función básica es calcular la lista de acciones para cada fonema y emoción, entre *frames* que son obtenidos por Interpolación Lineal. La interpolación (ver sección 3.8) de los fonemas y emociones en realidad es aplicada a su respectivo visema y expresión facial de acuerdo con la duración del fonema, con el fin de modificar al actor **brandy** con respecto a la nueva información recibida de **key frame** (figura. 7.9), dicho módulo proviene de la figura7.7.

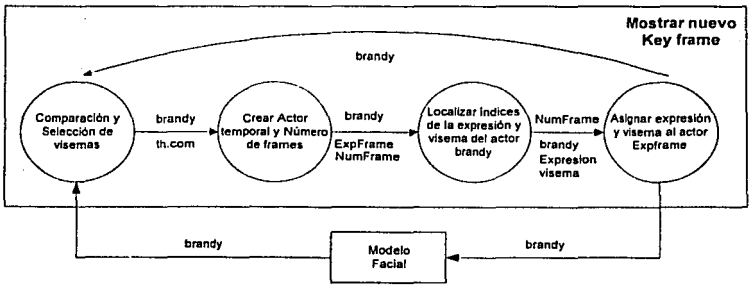


Figura 7.9 Mostrar nuevo Key frame

- **Comparación y Selección de Visemas**

  En esta etapa se compara cualquier fonema con respecto al fonema seleccionado en **info_phone->phone**, para posteriormente localizar y asignar el índice del visema correspondiente.

```
if (info_phone->phone == "tS")     //  visema_CH
{
        visema = brandy->exphead->LocateExp("visema_CH");
```

- **Crear actor temporal y número de frame**

  El objeto de esta etapa es crear un actor temporal que contenga la información de la ultima expresión facial del actor **brandy**, y pasarla a **frameExp**. De esta forma se optimiza la memoria dado que este nuevo actor solo vive dentro de la función y después desaparece.

```
Expression frameExp = *brandy->exphead->outputExp;
```

- **Localizar índices de la expresión y visema del actor brandy**

  En esta parte del proceso se procede a localizar el índice del tipo de palabra de control **info_words->name** en el actor **brandy**.

```
Emo  = brandy->exphead->LocateExp(info_words->name);

visema = brandy->exphead->LocateExp("visema_CH");
```

- **Asignar expresión y visema al actor *frameExp***

  En esta parte del proceso se realizan dos cosas muy importantes: una de ellas es asignar el índice ya localizado del tipo de la emoción y visema del actor **brandy** y encontrar la

expresión requerida. La segunda es que las expresiones se pueden sumar, dado que en realidad lo que se suman son vectores.

*frameExp = *brandy->exphead->expressions[visema] + *brandy->exphead->expressions[Emo];*

La función que permite realizar, este tipo de suma es:

```
const  Expression operator + (const Expression & v) const
{
int i;
Expression bob;
int size = m.size();
for (i =0; i < size; i++)
{
     bob.m.push_back( m[i] + v.m[i]);
  }
bob.name ="temp expression";
return bob;
  } ;
```

- **Playback e Interpolación entre *frames***

  Esta etapa realiza varias cosas, entre ellas son:

  a. Manejar eventos **expEvent**, dado que la animación puede manejarse como una máquina de estados jerárquica, es decir, no mandar a realizar nada hasta que se haya completado todo y poder pasar al siguiente evento.

  b. Controlar el número de cuadros **Frames_Inter** durante la interpolación no es fácil. Lamentablemente este dato se modificó dependiendo del tipo de máquina

o PC en las que se trabaja, para que pudiera sincronizarse el audio y la animación. Los datos que se encontraron son los que se muestran en la tabla 7.1 y la variable utilizada para ello es:

Frames_Inter= info_phone->duration + PENTIUM_III;

c. Realizar playbackThisExp, esto significa que la nueva expresión manda a modificar a **brandy** con los datos obtenidos hasta ese momento.

d. Crear una nueva expresión temporal que servirá para modificar a **brandy** y liberar memoria.

La función que realiza lo anterior es la siguiente:

g_eventque.push_back(new expEvent (Frames_Inter, expEvent::playbackThisExp, new Expression(frameExp)));

Por último la función que permite aplicar Interpolación (ver sección 3.8) entre expresiones es InterpolateKeyframes:

```
InterpolateExpression (Expression * res, const Expression * start, const Expression
*end, float percent)
{
    if  (percent >= 1.0)
    {
        *res = *end;
        return;
    }
    int i;
    float value;

    i = jaw_ID;
```

```
res->m[i] = value= percent*(end->m[i] - start->m[i]) + start->m[i];
. . . . .
for (i=0; i< nmuscles; i++)
{
    res->m[i] = value= percent*(end->m[i] - start->m[i]) + start->m[i];
}
}

InterpolateKeyframes(Expression * res, cExpression * keylist, float percent){
if (percent >= 1.0) {
    *res = *keylist->keyframes[keylist->keyframes.size()-1];
    return;
}
int nkeys = keylist->keyframes.size();
float len = keylist->length;
float millisecs = len * percent;

Expression * start= NULL;
Expression * end= NULL;
int i = (keylist->keyframes.size()-1);
for (; i > -1; i--){
    float curlen =keylist->keyframes[i]->length;
    if (curlen < millisecs) {
        start = keylist->keyframes[i];
        end = keylist->keyframes[i+1];
        break;
    }
}
if ((start ==NULL) &&(end ==NULL)) {
```

```
    float percent2 = millisecs/(keylist->keyframes[0]->length);
    InterpolateExpression(res, curExp, keylist->keyframes[0], percent2);
}
else{
    float percent2 = (millisecs-start->length)/(end->length - start->length);
    InterpolateExpression(res, start, end, percent2);
}
return;
}
```

Máquina	PC armada	PC armada	Laptop Toshiba	Laptop Toshiba
Procesador	Pentium II	Pentium III	Pentium III	Pentium III
Memoria RAM	128 MB	128 MB	128 MB	128MB
Sistema Operativo	Windows 2000 Server	Windows 2000 Server	Windows XP Professional	Windows 98
Frames para interpolación	0.2940	67.5	55.0	61.2

Tabla 7.1

Resultados y Conclusiones

Esta tesis solo cubre uno de los problemas de la arquitectura general, del proyecto "Distributed simulation of gesture recognition interfaces and intelligent agents for virtual environments", financiado por CONACYT, que resultó en la incorporación de un sintetizador de voz a un modelo facial.

Para cumplir lo anterior:

1. Se desarrollo e implemento un Editor de Visemas y Expresiones.
2. Se incorporó el sistema Festival al modelo facial para poder agregar texto y convertirlo en sonido.
3. Se creo un módulo de sincronización entre el modelo facial y el sonido.
4. Se documento por medio de Diagramas.

En cuanto a la cantidad de trabajo que se invirtió fue bastante, ya que se tuvo que dividir por etapas, además de tener que pasar por varios periodos de aprendizaje, para comprender el

funcionamiento de cada una de las herramientas que se utilizaron para concluir la implementación del sistema.

Los resultados fueron los siguientes:

- El editor de Visemas es una herramienta que permite al usuario seleccionar de manera intuitiva los músculos que entran en acción cuando se articula un fonema. Para implementar dicho editor, se tuvo que estudiar y comprender la filosofía de la biblioteca GTK/GDL, así cómo el tipo de funcionamiento del mismo con el fin de lograr la conjunción del modelo facial que esta programado en OpenGL y la interacción con GTK/GDL. Con ese objetivo tuvimos que utilizar otra biblioteca denominada GTK/GLArea que sirve como intermediario entre ambas. Con esto, un usuario interactúa con el modelo facial por medio de los *scrolls* y botones del editor que modifican los músculos del modelo facial y otras características necesarias para modelar los visemas.

- La sincronización de expresiones faciales y voz en español, permite al usuario generar una animación en tiempo real, a partir de un archivo de texto que contiene que contiene texto e instrucciones que permiten manipular las expresiones faciales y los *visemas* por medio de *palabras de control*. Dichas *palabras de control* son interpretadas por el sistema para mostrar el modelo facial en una postura determinada. El tiempo que requiere durar dicha postura se determina a partir de la duración de cada fonema obtenido por la síntesis de la voz en el sistema FESTIVAL.

- La documentación se realizó por medio de diagramas de flujo del sistema. Estos diagramas se generaron por medio de Microsoft Visio, con el objetivo de acortar la curva de aprendizaje y permitir a nuevos programadores no solo conocer el código fuente, sino también conocer la estructura y la filosofía que hay detrás de dicha programación del sistema. De ahí que los diagramas que propuse serán de gran utilidad para el programador.

Como trabajo futuro:

- Aún queda mucho por hacer, ya que falta la variación en el tono de voz y la variación o cambio de intensidad en cada una de las expresiones faciales.

- Aún falta mejorar el sistema para que la sincronización entre el modelo facial y el sonido sea la misma independientemente del tipo de procesador donde se ejecute.

- Se requiere mejorar el sistema para que permita cambiar el modelo facial por alguno de nuestra predilección.

Anexo A

A.1 Módulo "th-mode.scm"

El módulo 'th-mode.scm' es utilizado para tratar el diálogo de un guión de entrada.

init_function.- Esta función Scheme será llamada cuando se inicie la ejecución de este modo. Y dentro de esta función se encontrarán otras funciones Scheme las cuales realizarán lo siguiente:

1. *(set! th_previous_t2w_func token_to_words)*
   Una fase crucial del texto analizado es la tokenisation inicial del texto. Para ello los símbolos que comienzan con '<Emo_' y acaban con '<' son tratados palabras de control y no son pasados para la síntesis al habla. A continuación se muestran la implementación que se realizó para ello.

```
(define (th_token_to_words token name)
"(th_token_to_words TOKEN NAME) tokens a reglas de palabras."
  (cond
    ((string-matches name "\<Emo_.*\>") nil)
    (t (th_previous_t2w_func token name)) ))
```

2. *(set! th_previous_after_analysis_hooks after_analysis_hooks)*
   Listas de funciones Scheme a ser aplicadas antes del análisis y después de la síntesis. Con el objeto de hacer un análisis sintáctico y de síntesis del texto, tal como se muestra en la figura ___ .

3. *(set! after_analysis_hooks (list th_output_info))*
   Esta función es llamada después del análisis lingüístico pero antes de realizar la síntesis waveform. Es decir, que recoge datos y los guarda en 'th.ph' con información de los fonemas y su duración, y también genera un archivo 'th.com' con información de palabras de control y su duración durante cada frase del segmento. Al final, generamos un archivo de sonido 'file_wave.wav' por medio de un sintetizador externo MBROLA. Y las funciones en Scheme que permiten lo anterior son las siguientes:

   (set! th-current-file "data_file/th")

   (utt.save.phonedata utt (string-append th-current-file ".ph"))

   (utt.save.commands utt (string-append th-current-file ".com"))

   (MBROLA_Synth utt)

4. *(set! spanish_token_to_words th_token_to_words)*
   Los tokens en el idioma español son analizados más allá dentro de una lista de palabras, números, etc. Una palabra es un átomo que puede darse una pronunciación por el léxico (o letras o reglas del sonido). Un token puede dar lugar a varias palabras o a ninguna de todas.

5. *(set! token_to_words th_token_to_words)*
   Regresa una lista de palabras para el NOMBRE del TOKEN. Esto le permite al usuario personalizar varias traducciones dependientes del contexto, no-local, multi-palabra de tokens a palabras.

exit_function.- Esta función Scheme será llamada al finalizar algún proceso permitiendo restablecer las reglas de tokenisación. Y dentro de esta función encontraremos otras funciones Scheme las cuales realizaran lo siguiente:

(set! token_to_words th_previous_t2w_func)

(set! spanish_token_to_words th_previous_t2w_func)

(set! after_analysis_hooks th_previous_after_analysis_hooks)

(audio_mode 'async) // Función que reestablece el audio

(set! Parameter th_previous_Parameter)

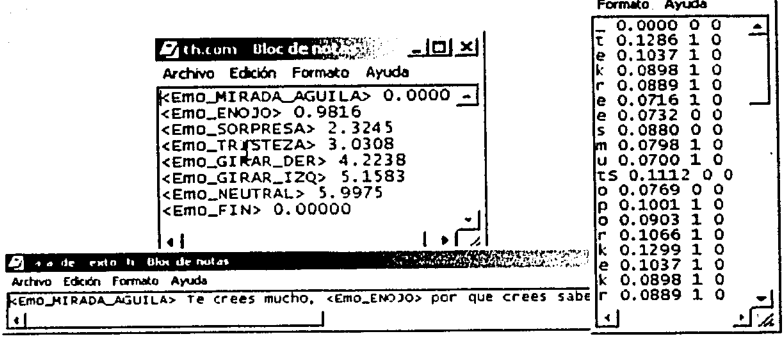


Figura 5.4 Contenido del archivo 'caja_de_texto' y de los archivos generados por 'th-mode' en Festival

A.2 Código fuente del módulo "th-mode.scm"

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;                    Centre for Speech Technology research
;;;                        University of Edinburgh, UK
;;;
```

```
;;;                          Copyright (c) 1996,1997
;;;                           All Rights Reserved.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  El 'modo texto th' de un diálogo sirve para alimentar una
;;;  animación de una cabeza que habla y que produce la salida de dos
;;;  archivos "data_file/th.ph" (información sobre fonemas) y
;;;  "data_file/th.com"
;;;  (palabras de control que se requieren para representar expresiones
;;;  faciales y movimiento de la cabeza) para cada utterance dentro
;;;  del archivo de texto.  Estos archivos son creados y el programa
;;;  externo MBROLA es llamado para realizar la sintesis waveform para
;;;  el utterance y obtener la salida de un archivo de sonido
;;;  "file_wave.wav".
;;;
;;;  Modified by:   Jorge Castro <kochukai@hotmail.com>
;;;  Modified at:   Miercoles, 19 de Julio de 2002
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;  Se necesita cargar el archivo mbrola, para poder hacer uso de sus
;;;  funciones
(load "data_file/mbrola.scm")

(define (utt.save.phonedata utt filename)
"(utt.save.mydata UTT FILE)
  Crea un archivo filename con información sobre fonemas, duración, stress,
F0
  word pos a partir del 'utt'"
  (let ((fd (fopen filename "w")))
    (mapcar
      (lambda (seg)
        (format fd "%s %2.4f %s %s"
             (item.feat seg "name")
             (item.feat seg "segment_duration")
             (item.feat seg "R:SylStructure.parent.stress")
             (item.feat seg "R:Target.daughter1.name")
        )    ;; END_FORMAT
          ;; output word name and part of speech if start of word
        ;;(if (and (not (item.relation.next seg "SylStructure"))
            ;;(not (item.next
            ;;      (item.relation.parent seg "SylStructure"))))
        (format fd " %s %s"
            (item.feat seg "R:Intonation.daughter1.name")
            (item.feat seg "R:SylStructure.parent.parent.pos")
          )                    ;; END_FORMAT
        (format fd "\n")
      )                              ;; END_LAMBDA
      (utt.relation.items utt 'Segment)
      )                        ;; END_MAPCAR
    (format fd "<e> 0\.0000")
    (print "**** Creando archivo \"th.ph\" con informacion de fonemas y
sus duraciones.")
```

```
    (fclose fd)
  utt)                                  ;; END_LET
 )                                      ;; END_DEFINE

(define (utt.save.commands utt filename)
"(utt.save.commands UTT FILE)
  Crea un archivo filename con información de las palabras de control con
el tiempo de ejecución durante el diálogo a partir de la información del
'utt'.  Las palabras de control son aquellos tokens que comienzan con
'<Emo_' y finalizan con '>'."
  (let ((fd (fopen filename "w")))
      (mapcar
      (lambda (tok_item)
        (if (string-matches (item.name tok_item) "\<Emo_.+\>")
            (format fd "%s %2.4f \n"
                 (item.name tok_item)
                   (find_com_time utt tok_item)
              )                         ;; END_FORMAT
            )                           ;; END_IF
        )                         ;; END_LAMBDA
        (utt.relation.items utt 'Token)
      )                                 ;; END_MAPCAR
  (format fd "<Emo_FIN> 0\.00000")

  (fclose fd)
  (print "**** Creando archivo \"th.com\" con palabras de control y
tiempos de \n ejecución durante el dialogo.")
   utt)                                 ;; END_LET
)                                       ;; END_DEFINE


(define (find_com_time utt tok_item)
"Regresa el tiempo de ejecución de tok_item.    Mira hacia atrás los
primeros token que están relacionados con una palabra (word) y regresa la
suma final de cada uno de los fonemas de que compone esa palabra."
  (cond
    ( (item.daughtern tok_item)
      (item.feat (item.daughtern tok_item) "word_end")
    )  ;; end_item
    ( (not (item.prev tok_item))  ;; start of stream
     0.0)  ;; end_not
    (t
      (find_com_time utt (item.prev tok_item))
     )   ;; END_T
  )  ;; END_COND
)  ;; END_DEFINE

(define (th_output_info utt)
  "(th_output_info utt)
Esta función es llamada despues del análisis linguistico pero antes de
realizar la sintesis waveform. Es decir, que recoje datos 'th.ph' sobre
los
fonemas y su duración y también algunos comandos 'th.com'. "
```

```
  (set! th-current-file "data_file/th")          ;;  subdirectorio donde
almacenan
                                                                        ;;
los datos de los fonemas y
                                                                        ;;
las palabras de control
  (utt.save.phonedata utt (string-append th-current-file ".ph"))
  (utt.save.commands utt (string-append th-current-file ".com"))

  (print "\n******** Archivo 'mbrola.scm' cargado *********\n")
  (MBROLA_Synth utt)                                    ;; Sintetizador
MBROLA
  (print "\n*** Se terminó el análisis sintáctico y la generación de
archivos ***")
utt)     ;; END_DEFINE

;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Define un nuevo modo de texto para cabezas que hablan
;;;
;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (th_init_func)
 "Función que inicializa el modo texto para recibir un diálogo

 (print "--- Preparando análisis previo del texto de Tokens a palabras,
para \n localizar las palabras de control que no serán tratadas por
Festival")
 (set! th_previous_t2w_func token_to_words)
 (print "--- Preparando análisis sintáctico del texto para obtener todos
los \n elementos de la pronunciación")
 ;; Un hook en términos de Lisp es una posición  donde una pieza de código
 ;; Puede usarse para especificar su propia tarea o encargo. En Festival
hay
 ;; Un número de lugares donde los hooks son usados. Para nuestro caso el
 ;; after_analysis_hooks es aplicado después de que la síntesis a sido
aplicada
 ;; Para permitir especificar un encargo tal como un reanálisis o
modificación del aumento de la síntesis waveform.
 (set! th_previous_after_analysis_hooks after_analysis_hooks)
 (set! after_analysis_hooks (list th_output_info))
 (print "--- Preparando análisis del texto de Tokenks a palabras en
Español")
 (set! spanish_token_to_words th_token_to_words)
 (set! token_to_words th_token_to_words)

 (print "Iniciando análisis sintáctico del texto del archivo \"caja de
texto.th\"")

)   ;; END_DEFINE

(define (th_exit_func)
 "Función de salida (th_exit_func) que restablece el modo texto"
```

```
 (set! token_to_words th_previous_t2w_func)
 (set! spanish_token_to_words th_previous_t2w_func)
 (set! after_analysis_hooks th_previous_after_analysis_hooks)
 (audio_mode 'async) ;; so we can reset the audio
 (set! Parameter th_previous_Parameter)
 print("** Termina Análisis Sintáctico del texto del archivo
'caja_de_texto.th' **");
)

(define (th_token_to_words token name)
"(th_token_to_words TOKEN NAME)
Especifica las reglas de token a word cuando se lee el texto de entrada de
un dialogo."
 (cond
     ( (string-matches name "\<Emo_.*\>")
       ;;  las palabras de control son aquellos tokens que comienzan con
'<Emo_'
       ;;  y finalizan con '>'.
      nil)   ;; END_STRING
     (t (th_previous_t2w_func token name) )
  )   ;; END_COND
 )    ;; END_DEFINE


(set! tts_text_modes
   (cons
    (list
      'th           ;; Nombre del modo
      (list         ;; ogimarkup mode params
       (list 'init_func th_init_func)
       ;;(list 'exit_func th_exit_func)
        )
      )
   tts_text_modes)
)

(provide 'th-mode)
```

A.3 Módulo "mbrola.scm"

Esta función es llamada después del análisis lingüístico pero antes de realizar la síntesis waveform. Es decir, que recoge datos 'th.ph' sobre los fonemas y su duración y también algunos comandos 'th.com'. Los nombres de estos archivos son pasados a un programa ejecutable externo el cual los procesa para poder generar una cabeza que habla."

La función (MBROLA_Synth utt) es llamada cuando los parámetros Synth_Method es MBROLA. La función simplemente genera un archivo 'file_phone.pho' con información prosódica del *utterance* por medio de las siguientes funciones en Scheme:

(save_segments_mbrola utt filename) y (save_seg_mbrola_entry name start dur targs fd).

Posteriormente llama a un programa externo 'mbrola' en conjunto con una base de datos de difonos 'mx1/mx1', y genera un archivo 'file_wave.wav' con la pronunciación. Como se muestra a continuación:

mbrola mx1/mx1 file_phone.pho file_wave.wav

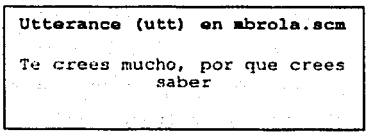
Las siguientes variables son usadas en el proceso:

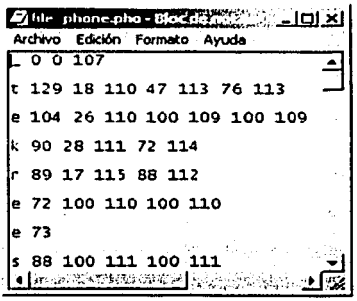
mbrola_progname. Aquí se coloca el nombre y la ruta donde se encuentra el ejecutable mbrola. Ejem: *mbrola_progname "mbrola"*

mbrola_database. Aquí se coloca el nombre y ruta de la base de datos de la voz a usar. Ejem: *mbrola_database "mx1/mx1"*

f_phone. Aquí se colocará el nombre donde se guardará el archivo de fonemas. Ejem: *f_phone "data_file/file_phone.pho"*

f_wave Aquí se colocará el nombre donde se guardará el archivo de sonido: Ejem: *f_wave "data_file/file_wave.wav"*





A.4 Código fuente del módulo "mbrola.scm"

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                Centre for Speech Technology research
;;;                    University of Edinburgh, UK
;;;                     Copyright (c) 1996,1997
;;;                      All Rights Reserved.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Modified by:   Jorge Castro <koobukai@hotmail.com>
;;;  Modified at:   Miércoles, 19 de Julio de 2002
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Soportado para MBROLA como un módulo externo.
;;;


(defvar mbrola_progname "mbrola"
  "mbrola_progname
  El nombre del programa para mbrola.")

(defvar mbrola_database "mxl/mxl"
  "Base de Datos.  El nombre de la Base de datos utilizada durante la
sintesis de MBROLA.")

(define (SaveTextMbrola string)
"(SaveTextMbrola TEXT)
A partir del texto introducido (TEXTO) se construye un archivo
(caja_de_texto.th), el cual se lee y a partir de esta información se
generan dos archivos más con información sobre fonemas, su duración y
comandos de expresiones"
  (let (
          (f_texto "data_file/caja_de_texto.th")
            (fd)
        )
        (set! fd (fopen f_texto "w") )
      (format fd "%s" TEXT)
      (fclose fd)
        (MBROLA_Synth (utt.synth (eval (list 'Utterance 'Text string))))
  )
)     ;; END_DEFINE


(define (MBROLA_Synth utt)
  "(MBROLA_Synth UTT)
  Síntesis realizada usando a MBROLA como un módulo externo.  Básicamente
descarga la información del utterance. Se llama a MBROLA y crea el archivo
de sonido 'waveform'
  a partir del 'utt'. [see MBROLA]"
```

```
   (let ((f_phone "data_file/file_phone.pho") (f_wave
"data_file/file_wave.wav"))
     (save_segments_mbrola utt f_phone)
     (system (string-append mbrola_progname " "
                    mbrola_database " "
                    f_phone " "
                    f_wave ))
     (utt.import.wave utt f_wave)        ;; Esta función
     (apply_hooks after_synth_hooks utt)
       (print "**** MBROLA mx1/mx1 data_file/file_phone.pho
data_file/file_wave.wav")
       (print "**** Se creo el archivo 'file_wave.wav' con el sintetizador
MBROLA")

     utt)
)


(define (save_segments_mbrola utt filename)
  "(save_segments_mbrola UTT FILENAME)
   Guarda la información del segmento en 'filename' con formato MBROLA. el
formato contiene fonemas, inicio del segmento, duración (ms) [% posición,
F0, target]*.
   [ver MBROLA]"
   (let ((fd (fopen filename "w")))
     (mapcar
       (lambda (segment)
         (save_seg_mbrola_entry
                  (item.feat segment 'name)
                  (item.feat segment 'segment_start)
                  (item.feat segment 'segment_duration)
                  (mapcar
                        (lambda (targ_item)
                              (list
                                    (item.feat targ_item "pos")
                                    (item.feat targ_item "f0")
                              )           ;; END_LIST
                        )                 ;; END_LAMBDA
                        (item.relation.daughters segment 'Target)
                  )                       ;; list of targets mapcar
           fd)                            ;; END_SAVE
       )                                  ;; END_LAMBDA
       (utt.relation.items utt 'Segment)
       )                                  ;; END_MAPCAR
       (print "**** Creando archivo de fonemas : 'file_phone.pho'")
     (fclose fd)
   )                                      ;; END_LET
)                                         ;; END_DEFINE

(define (save_seg_mbrola_entry name start dur targs fd)
  "(save_seg_mbrola_entry ENTRY NAME START DUR TARGS FD)
   La entrada contiene, (nombre duración num_targs comienzo 1st_targ_pos
1st_targ_val)."
```

```
  (format fd "%s %d " name (nint (* dur 1000)))
  (if targs     ;; if there are any targets
      (mapcar
       (lambda (targ) ;; targ_pos y targ_val
                 (let ((targ_pos (car targ))
                       (targ_val (car (cdr targ)))
                      )
                       (format fd "%d %d "
                             (nint (* 100 (/ (- targ_pos start)
dur))) ;; % pos of target
                             (nint (parse-number targ_val))
                       )           ;; target value
                 )
           )
      targs)
  )
  (terpri fd)
  (terpri fd)
)

(provide 'mbrola)
```

Anexo B

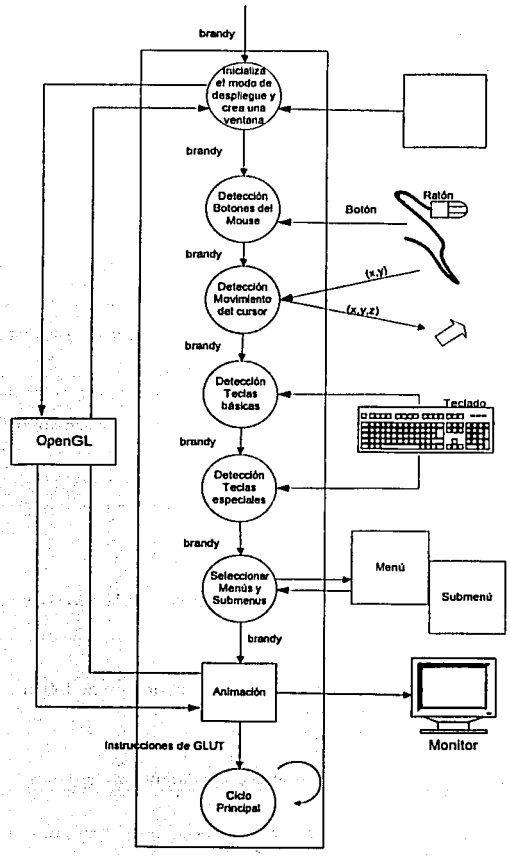
Diagrama de GLUT

En este acontecimiento se reciben instrucciones precisas de OpenGL y algunos dispositivos de entrada, pero en general GLUT (OpenGL Utility Toolkit) es un programa de interfase entre ANSI C y Fortran para escribir programas OpenGL independientes del sistema de Ventanas o plataforma. El Toolkit soporta las siguientes funcionalidades:

- Múltiples ventanas para desplegar (rendering) imágenes de OpenGL.
- Manejo de eventos Callback.
- Manejo del framebuffer.
- Sofisticados dispositivos (ratón, teclado, etc.) de entrada.
- Rutinas de tiempo e 'idle' para manejar diferentes simulaciones de la animación.
- Facilidad para manejo de menús en cascada.
- Rutinas para generar varios objetos en forma de sólidos y de malla.
- Soporte para manejar bitmaps y fonts.
- Diferentes tipos de funciones para manejar el sistema de ventanas.

GLUT simplifica la implementación de programas usando OpenGL rendering. La interfase de programación para una aplicación (API) GLUT requiere de muy pocas rutinas para

desplegar una escena grafica usando OpenGL. El API GLUT es un sistema de ventanas completo e independiente. Por esta razón, GLUT no regresa algún manejador, puntero u alguna estructura de datos del sistema de ventanas nativo.



Anexo C

Modelado Facial

El objetivo de esta tesis no es mostrar como se realiza la síntesis del *modelo facial* a nivel de implementación, sino la sincronización entre las imágenes faciales y el audio, pero se ha observado a lo largo de dicha investigación y al estar en contacto con otros especialistas que existe un problema de cómo diseñar, crear y simular una cabeza articulada a nivel de implementación como se mencionó en el capítulo III. Por ello en esta sección se mostrará de forma muy breve y concreta los requerimientos, el tipo de arquitectura y como conviven los procesos internos que tiene el módulo del *modelo facial* que se muestra en la figura 7.2 que es parte de Sistema.

C.1 Módulo del modelo facial

En este subanexo explicaremos el módulo del *modelo facial* que se muestra en la figura 7.2, la cual se encarga de crear el actor **brandy** y desplegarlo en una ventana de OpenGL vía GLUT. Cuando se ejecuta el módulo de *sincronización*, el módulo del modelo facial recibe la orden de crear o modificar (figura C.1) la imagen presentada.

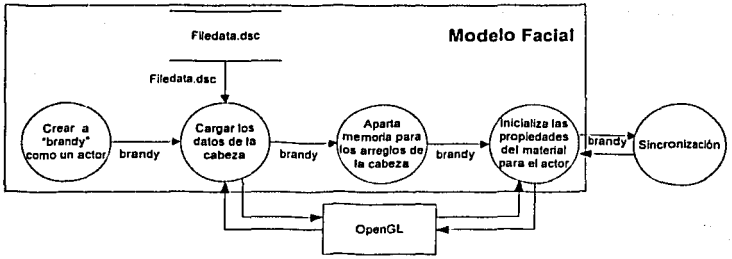


Figura C.1 Módulo del modelo facial

Como se muestra a continuación.

En este diagrama se observa claramente que es necesario alimentar al sistema por medio del archivo **filedata.dsc** que permite ubicar e inicializar una cabeza (en esta sección será denominado actor), cuyo contenido se describe a continuación:

- face-data/gedalia2/gedalia.msh // Archivo con información de la geometría de una malla.
- face-data/gedalia2/gedtags2.dat //Archivo con las parametrizaciones de la mandíbula, los ojos y la lengua de la malla 'gedalia.msh'.
- face-data/gedalia2/gedmuscle.dat //Archivo con información de los músculos de la Cara
- face-data/expressions.dat //Biblioteca de expresiones básicas y visemas
- face-data/keyframes.dat //Biblioteca de expresiones compuestas.

Por lo que, si se desea utilizar un actor diferente al actual, se requerirá crear los archivos antes enlistados y guardarlos en un subdirectorio para posteriormente llamarlos a través del archivo **filedata.dsc** y que funcione el sistema con otro actor. Para modificar este archivo se puede utilizar cualquier editor de texto.

C.2 Módulo de carga los datos de la cabeza

En esta subanexo explicaremos el módulo *carga los datos de la cabeza* que se muestra en la figura C.1, la cual se encarga de realizar de recibir la estructura de datos **brandy** sin información. Durante la transición del procesador se reciben los datos de la topología de la cabeza, los datos parametrizados de la mandíbula y los ojos, los músculos, las expresiones básicas y los visemas, y las expresiones compuestas. El flujo de información es como se muestra en la figura C.2.

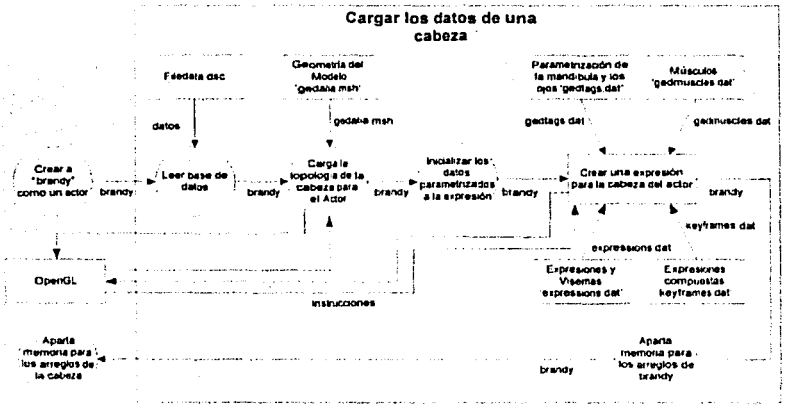


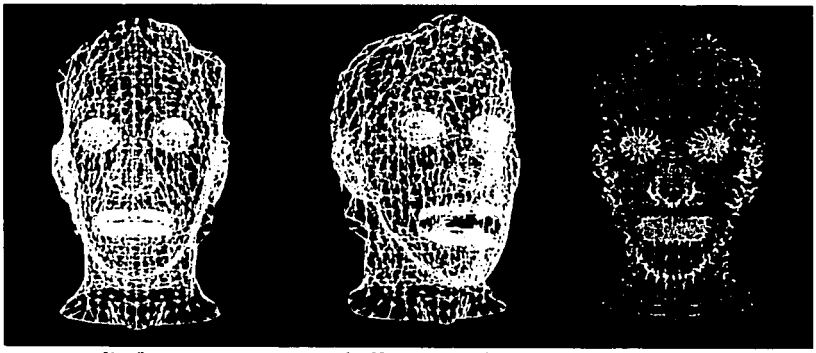
Figura C.2 Módulo para cargar los datos de una cabeza

C.3 Módulo de carga la topología de la cabeza para el Actor

En esta subanexo explicaremos el módulo *carga la topología de la cabeza para el actor* que se muestra en la figura C.2, la cual se encarga de leer o cargar la geometría del modelo facial, y con los datos obtenidos se construye la malla de la cabeza del actor **brandy**. Posteriormente se genera la topología del actor **brandy** y se leen los archivos que contienen

la información de las texturas de la piel y del ojo, las cuales son asignadas a la topología del actor brandy. El proceso se muestra en el diagrama C.6.

Para codificar un modelo-basado en movimientos de imágenes faciales, un modelo 3-D que representa la forma de la cabeza humana se crea y se usa como se explica en el capítulo III. Éste generalmente es un "wire-frame" (dibujo de líneas de una malla) que consiste en un número grande de triángulos. La Figura C.4 muestra un ejemplo de un wire-frame usado en esta tesis y en el sistema, y fue construida a partir de un modelador, como Rhino, Maya, Amapi 3D, 3D Studio MAX, etc. Esta compuesta de aproximadamente 4000 triángulos. Se modeló la cabeza, los ojos, dientes y la lengua.



(a) De frente **(b) En perspectiva** **(c) nodos o vértices de los polígonos**

Figura C.4 Modelo wire-frame de la cabeza.

El tamaño, la forma y la posición relativa de la cabeza, ojos y boca varían de una persona a otra, por lo que se debe siempre cambiar los valores de las coordenadas de cada vértice en un modelo estándar que corresponda a los puntos representativos entre el wire-frame y la imagen de la persona real.

Al saber donde se localizan exactamente los vértices de la cabeza permite especificar información acerca de la apariencia de la persona. Cuando la persona se mueve y habla, el modelo se deforma por simulación de los ojos y de la boca. El mapeo de textura es entonces usado para dar información de luminosidad, reflejo, y dar mayor realismo al movimiento facial de la animación.

El mismo método puede usarse para sintetizar imágenes faciales humanas de un texto de la entrada en lugar de una imagen de la entrada. Los pasos involucrados son por consiguiente:

1. Preparar un modelo 3-D de una persona cuyo expresión facial deseamos simular, y preparando una imagen como información de textura (luminosidad e información del chrominance) como se muestra en la figura C.5. Corresponda entre cada parte de la forma del modelo y cada región en la imagen debe ser determinada para que la textura sea aplicada apropiadamente.

(a) Textura de la cabeza

(b) Textura del Ojo

Figura C.5 Imágenes BMP

2. Manipular la forma de diferentes partes faciales desde un modelador como son las cejas, párpados y la boca para obtener un modelo "wire-frame" que tiene el movimiento deseado y la expresión deseada. Al finalizar se crea el archivo 'gedalia.msh' el cual contiene la información necesaria de la malla, y los vértices de la misma.

3. Trazar la información de la textura hacia la forma del modelo 3-D como se obtuvo en el punto 2.

El problema más difícil en este proceso es como se menciono en el punto 2, es decir, cómo deben manipularse la forma y el movimiento de cada parte facial para producir una imagen que parece viva con una expresión natural.

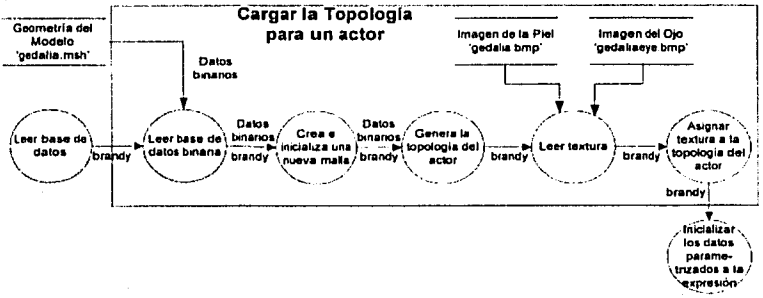


Figura C.6 Módulo que carga la topología para un actor

C.4 Módulo de crear una expresión para la cara del actor

En esta subanexo explicaremos el módulo *crear una expresión para la cara del actor* que se muestra en la figura C.2, la cual se encarga de manipular una expresión facial, para ello es necesario modelar dicha expresión como se mencionó en el capítulo 3 y guardar dicha información en el archivo 'expressions.dat' el cual debe contener la siguiente formato.

<Número de expresiones>

<Nombre de la expresión>

lista de *<canal de la expresión>* *<valor del canal>* // donde el **nombre del canal** es el nombre del músculo o un canal adicional que tiene que ser definido en el código.

"<e>" // fin de la lista de expresiones.

El proceso de dicho módulo se muestra en el diagrama C.7.

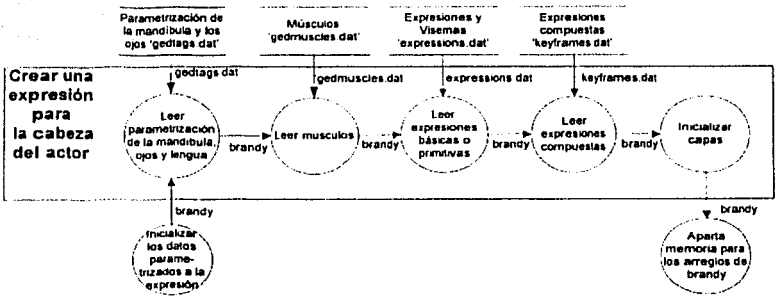


Figura C.7 Módulo que carga la topología para un actor

C.5 Módulo de leer Músculos

En esta subanexo explicaremos el módulo que se encarga de *leer músculos* como se muestra en la figura C.2, la cual se encarga de de leer una base de datos (ver figura C.8) con información sobre los músculos de una cabeza, que es construida como se explicó en sección 3.6 y sirve para crear una secuencia de animación que permita a un programa tomar la entrada de una lista de parámetros o descripciones de acción de un archivo **gedmuscles.dat** utilizando un editor de el cual debe contener el siguiente formato:

< Número de datos> Número de músculos

"*<cadena de caracteres>* " tipo de músculo

"*<nombre del músculo>* "
[<posición del hueso o centro>]

[<Posición donde el músculo hace contacto con la piel o la escala>]

"<e>" de la lista y de las expresiones.

El proceso se muestra en el diagrama C.9.

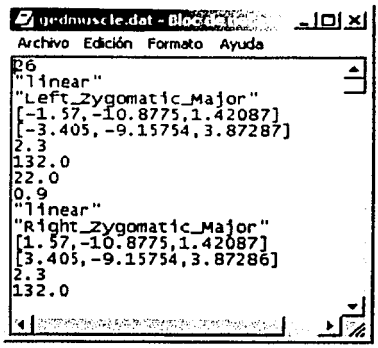


Figura C.8 archivo gedmuscle.dat

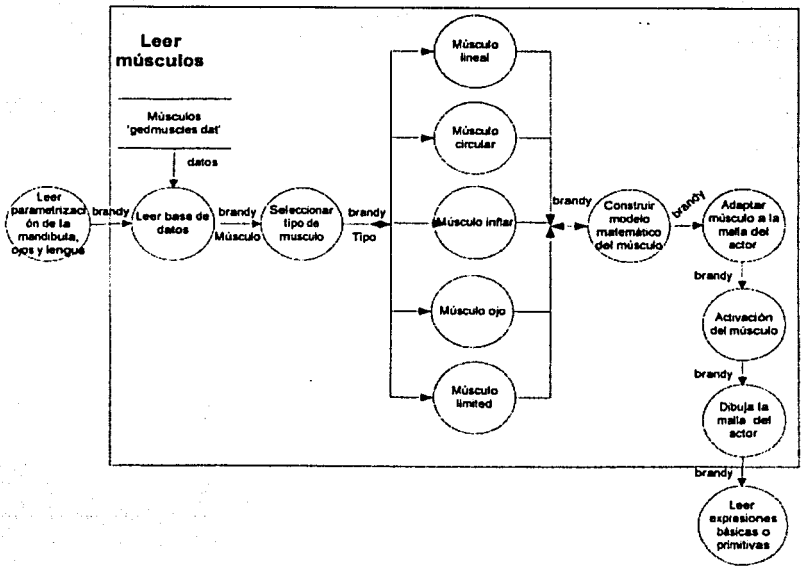


Figura C.9 Módulo leer músculo

C.6 Módulo de leer expresiones básicas o primitivas

En esta subanexo se muestra el módulo que se encarga de *leer expresiones básicas o primitivas* como se muestra en la figura C.7. El proceso se muestra en el diagrama C.10.

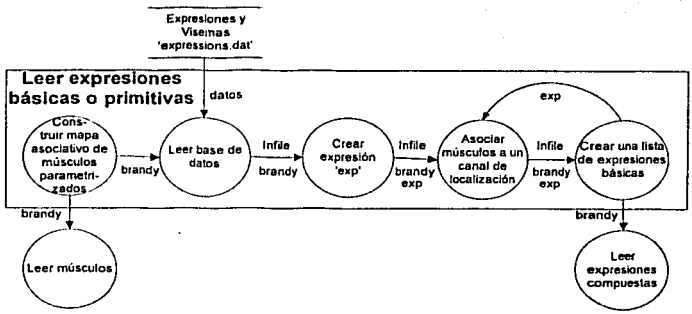


Figura C.10 Módulo Leer expresiones básicas o primitivas

C.7 Módulo de leer expresiones compuestas

En esta subanexo se muestra el módulo que se encarga de *leer expresiones compuestas* como se muestra en la figura C.7. El proceso se muestra en el diagrama C.11.

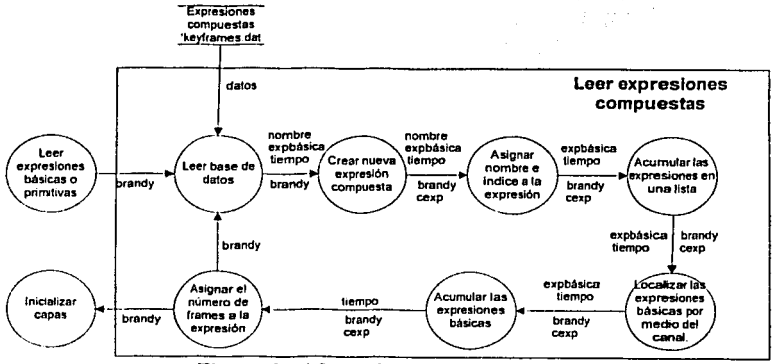


Figura C.11 Módulo leer expresiones básicas

Anexo D

D.1 Código fuente "Sincronización.cpp"

```
/*-------------------------------------------------------------------------
 * Filename:      syncronization.c
 * Version:       1.0
 * Copyright:     Laboratorio de Graficación, Facultad de Ciencias
 * Author:        Jorge Castro <koobukai@hotmail.com>
 * Description:   Biblioteca de funciones que permite realizar la sincronización entre el
 *                Modelo Facial y el audio
 * Created at:    Miércoles, 03 de Agosto del 2001
 * Modified by:   Jorge Castro <koobukai@hotmail.com>
 * Modified at:   Jueves, 08 de Agosto del 2002
 * -------------------------------------------------------------------------
 * Bibliotecas de OpenAL:          openal32.lib alu.lib alut.lib alc.lib
 * Controladores de OpenAL:        openal32.dll
 *-------------------------------------------------------------------------*/
/*------------------------------ Inclusiones ------------------------------*/
#include "syncronization.h"
#include "festival_text_speech.h"          // Biblioteca de Festival
```

```
#include "expEvent.h"
#include "exphead.h"
#include "global.h"    //class that holds the globals

#include "Expression.h"
#include "cexpression.h"

#include "actor.h"

#include <stdio.h>              // Bibliotecas estandar de 'C'.
#include <malloc.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <memory.h>

#include <al\al.h>              // Bibliotecas de OpenAL
#include <al\alu.h>
#include <al\alc.h>
#include <al\alut.h>

/*----- Declaración de variables internas y de variables globales externas -----*/
using namespace ExpressionNS;
using namespace std;

extern Global * L;
extern CActor *brandy;

extern vector <expEvent *> g_eventque;
```

```
#define NUM_BUFFERS 1
#define NUM_FUENTES 1
#define NUM_ENVIRONMENTS 1

#define PENTIUM_II  0.2940
#define PENTIUM_III 67.5
#define LABTOP      55.0

ALfloat listenerPos[]={2.0,3.0,4.0};  // Definición de variables para OpenAL
ALfloat listenerVel[]={1.0,1.0,1.0};
ALfloat listenerOri[]={0.0, 1.0, 0.0, 0.0, 0.0, 1.0};

ALfloat fuentePos[]={ 2.0, 3.0, 4.0};
ALfloat fuenteVel[]={ 3.0, 4.0, 0.0};

ALuint buffer[NUM_BUFFERS];
ALuint fuente[NUM_FUENTES];
ALuint  environment[NUM_ENVIRONMENTS];

ALsizei     size,freq;
ALenum      format;
ALvoid      *data;
ALboolean   loop;
ALint       error;
/*--Declaración de estructuras y variables globales para manejar los fonemas y las PC --*/

typedef struct DataPhone
{
```

```
        char phone[3];
        float duration;
        int stress;
        int F0WordPos;
        DataPhone *sig;                 // apuntador a la siguiente entrada
} DataPhone;

DataPhone *primero, *ultimo;        // apuntador a la última entrada

typedef struct ControlExpression    // Estructura para guardar información de 'th.com'
{
        char  name[31];
        float time;
        ControlExpression *sig;
} ControlExpression;

ControlExpression *pri_words, *ult_words;

/* ---------------------------- Función Pública ---------------------------- */

//*****************************************************************************
// void dl_insert( struct direc *i, struct direc **ppio, struct direc **final)
// Función que crea una lista simplemente enlazada
//*****************************************************************************
void dl_insert( DataPhone *nuevo,                   // nuevo elemento
                DataPhone **primero,     // primer elemento de la lista
                DataPhone **ultimo)      // último elemento de la lista
{
        if (!*ultimo)
```

```
    {
        //printf("Primer elemento\n");
        *primero = nuevo;
        *ultimo  = nuevo;
    }
    else
    {   /* el que hasta ahora era el último tiene que apuntar al nuevo */
        (*ultimo)->sig = nuevo;
        nuevo->sig = NULL;
        *ultimo = nuevo;
    }
}

// ***************************************************************************
// void dl_insert_Emoes( struct direc *i, struct direc **ppio, struct direc **final)
// Función que crea una lista simplemente enlazada
// ***************************************************************************
void dl_insert_Emoes( ControlExpression *nuevo,        // nuevo elemento
                      ControlExpression **primero,     // primer elemento de la lista
                      ControlExpression **ultimo)      // último elemento de la lista
{
    if (!*ultimo)
    {
        *primero = nuevo;
        *ultimo  = nuevo;
    }
    else
    {   /* el que hasta ahora era el último tiene que apuntar al nuevo */
        (*ultimo)->sig = nuevo;
```

```
            nuevo->sig = NULL;
            *ultimo = nuevo;
        }
}

// ***************************************************************************
// función: void cargar_fonemas ( void )
// Función que se encarga de cargar el archivo de fonemas
// ***************************************************************************
void cargar_fonemas ( void )
{
        FILE *fp;
        DataPhone *info_phone;

        printf("\n\n ..... Leyendo archivo: \"th.ph\" .....\n\n");
        fp = fopen("data_file/th.ph", "r" );

        if (!fp)
        {
                printf("\n ** ERROR: No se puede abrir el archivo **\n");
                return;
        }

        while(ultimo)           // liberar cualquier memoria previamente asignada
        {
                info_phone = ultimo->sig;
                free(info_phone);
                ultimo = info_phone;
        }
```

```
    ultimo = NULL;              // reinicializar los apuntadores

  while(!feof(fp))
      {
            info_phone = (DataPhone *) malloc(sizeof(DataPhone));
            if (!info_phone)
            {
                  printf("No hay memoria");
                  return;
            }
            fscanf(fp,  "%s%f%d%d",  &info_phone->phone,  &info_phone->duration,
&info_phone->stress, &info_phone->F0WordPos );

            info_phone->sig = NULL;     // ahora metemos el nuevo elemento en la lista.
            if ( info_phone->phone != "")  // lo situamos al final de la lista
            {
                  dl_insert(info_phone, &primero, &ultimo);
                  printf("*");
            } // end_if
      }     //end_while

  fclose(fp);
}

// ***************************************************************************
// función: carga_comandos_de_control(void)
// Función que se encarga de cargar el archivo de comandos de las Emoes
// ***************************************************************************
```

```
void carga_comandos_de_control(void)
{
        FILE *fp;
        ControlExpression *info_words;


        printf("\n\n .....  Leyendo archivo:  \"th.com\" \n");
        fp = fopen("data_file/th.com", "r" );

        if (!fp)
        {
                printf("\n ** ERROR: No se puede abrir el archivo **\n");
                return;
        }

        while(ult_words)        //liberar cualquier memoria previamente asignada
        {
                info_words = ult_words->sig;
                free(info_words);
                ult_words = info_words;
        }

        ult_words = NULL;               // reinicializar los apuntadores

 while(!feof(fp))
        {
                info_words = (ControlExpression *) malloc(sizeof(ControlExpression));
                if (!info_words)
                {
```

```
                printf("No hay memoria");
                return;
            }
            fscanf(fp, "%s%f", &info_words->name, &info_words->time);

            printf("\nEmo: %s  tiempo: %f  ", &info_words->name, info_words->time);

            info_words->sig = NULL;

           /* ahora metemos el nuevo elemento en la lista. lo situamos al final de la lista */
            if (info_words->name != "\0")
            {
                dl_insert_Emoes(info_words, &pri_words, &ult_words);
            }
        }
    fclose(fp);
}
// ************************************************************************
//   void sincronizacion_visema_Emociones (DataPhone *info_phone, ControlExpression
*info_words )
// Función que muestra la animación de los visemas
// ************************************************************************
void  sincronizacion_visema_Emociones  (DataPhone  *info_phone,  ControlExpression
*info_words )
{
        int visema, Emo ;
        float Frames_Inter;
        Expression frameExp = *brandy->exphead->outputExp;
        Emo  = brandy->exphead->LocateExp(info_words->name);
```

```
Frames_Inter= info_phone->duration + PENTIUM_III;

if (info_phone->phone == "tS")    // visema_CH
{
        visema = brandy->exphead->LocateExp("visema_CH");
        frameExp = *brandy->exphead->expressions[visema] + *brandy->exphead-
>expressions[Emo];
        g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
}
if (info_phone->phone == "ny")    // visema_ENIE
{
        visema = brandy->exphead->LocateExp("visema_ENIE");
        frameExp = *brandy->exphead->expressions[visema] + *brandy->exphead-
>expressions[Emo];
        g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
}
if (info_phone->phone == "dZ" || info_phone->phone == "rr") //
visema_Y_LL_R_RR
        {
        visema = brandy->exphead->LocateExp("visema_Y_LL_R_RR");
        frameExp = *brandy->exphead->expressions[visema] + *brandy->exphead-
>expressions[Emo];
        g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
}
if ( info_phone->phone != "dZ" || info_phone->phone != "ny"
  || info_phone->phone != "tS" || info_phone->phone != "rr")
```

```
{
    char phonechar = info_phone->phone[0];

        switch (phonechar)
        {
    // Vowels
        case 'a':    // visema_A
                     visema = brandy->exphead->LocateExp("visema_A");
                     frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                     g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                     break;
        case 'e':    // visema_E
                     visema = brandy->exphead->LocateExp("visema_E");
                     frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];

                     g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));

                     break;
        case 'i':    // visema_I
                     visema = brandy->exphead->LocateExp("visema_I");
                     frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                     g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                          break;
```

```
            case 'o':    // visema_O
                        visema = brandy->exphead->LocateExp("visema_O");
                        frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                        g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                        break;
            case 'u':    // visema_U_W
            case 'w':
                        visema = brandy->exphead->LocateExp("visema_U_W");
                        frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                        g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                        //g_eventque.push_back(new expEvent (info_phone->duration,
expEvent::playbackExp, 45));
                        break;

      // consonants
                  case 'p':    // visema_P_B_M_PAU
                  case 'b':
                  case 'm':
                  case '_':
                  visema = brandy->exphead->LocateExp("visema_P_B_M_PAU");
                        frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                        g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
```

```
                    //g_eventque.push_back(new expEvent (info_phone->duration,
expEvent::playbackExp, 46));
                    break;
                case 't':       // visema_T_D
                case 'd':
                    visema = brandy->exphead->LocateExp("visema_T_D");
                    frameExp    =    *brandy->exphead->expressions[visema]    +
*brandy->exphead->expressions[Emo];
                    frameExp    =    *brandy->exphead->expressions[visema]    +
*brandy->exphead->expressions[Emo];
                    g_eventque.push_back(new      expEvent      (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                    break;
                case 'f':     // visema_F
                    visema = brandy->exphead->LocateExp("visema_F");
                    frameExp    =    *brandy->exphead->expressions[visema]    +
*brandy->exphead->expressions[Emo];
                    g_eventque.push_back(new      expEvent      (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                    break;
                case 's':               // visema_S
                    visema = brandy->exphead->LocateExp("visema_F");
                    frameExp    =    *brandy->exphead->expressions[visema]    +
*brandy->exphead->expressions[Emo];
                    g_eventque.push_back(new      expEvent      (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                    break;
                case 'x':     // visema_J
                    visema = brandy->exphead->LocateExp("visema_J");
```

```
                    frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                    g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                    break;
                case 'n':   // visema_n_N_L
                case 'N':
                case 'l':
                    visema = brandy->exphead->LocateExp("visema_n_N_L");
                    frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                    g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                    break;

                case 'j':   // visema_Y_LL_R_RR
                case 'r':
                    visema = brandy->exphead-
>LocateExp("visema_Y_LL_R_RR");
                    frameExp = *brandy->exphead->expressions[visema] +
*brandy->exphead->expressions[Emo];
                    g_eventque.push_back(new expEvent (Frames_Inter,
expEvent::playbackThisExp, new Expression(frameExp)));
                    break;

                default:
                    break;
            } // end_switch
        } // end_if
```

```
}
// ************************************************************************
// función:  ALvoid DisplayALError(ALbyte *szText, ALint errorcode)
// función que maneja los errores en del audio en OPENAL
// ************************************************************************
ALvoid DisplayALError(ALbyte *szText, ALint errorcode)
{
        printf(szText);
        switch (errorcode)
        {
                case AL_INVALID_NAME:
                        printf("AL_NOMBRE_INVALIDO\n");
                        break;
                case AL_INVALID_ENUM:
                        printf("AL_ENUM_INVALIDO\n");
                        break;
                case AL_INVALID_VALUE:
                        printf("AL_VALOR_INVALIDO\n");
                        break;
                case AL_INVALID_OPERATION:
                        printf("AL_OPERACION_INVALIDA\n");
                        break;
                case AL_OUT_OF_MEMORY:
                        printf("AL_SIN_MEMORIA\n");
                        break;
                default:
                        printf("ERROR_INDEFINIDO\n");
                        break;
        }
```

```
        return;
}

// ***************************************************************************
// función:  ALboolean CargaWAV(const char *fname,int buffer)
// La razón de crear dos funciones ( una para cargar y otra para descargar
// archivos WAV) reside en que las funciones utilizadas para realizar dichos
// procesos son ligeramente distintas según utilicemos Windows o linux
// ***************************************************************************

ALboolean CargaWAV(char *fname)
{

#ifdef _WIN32
    /* Carca archivo y verica errores */
    alutLoadWAVFile(fname,&format,&data,&size,&freq, &loop);
    if ((error = alGetError()) != AL_NO_ERROR)
    {
        DisplayALError("alutLoadWAVFile wave1.wav : ", error);
        alDeleteBuffers(NUM_BUFFERS, buffer);       // Delete Buffers
        exit(-1);
    }
    alBufferData(buffer[0],format,data,size,freq);
    if ((error = alGetError()) != AL_NO_ERROR)
    {
        DisplayALError("alBufferData buffer 0 : ", error);
        alDeleteBuffers(NUM_BUFFERS, buffer);       // Delete buffers
        exit(-1);
    }
```

```
#endif
#ifdef _LINUX

        alutLoadWAV(fname,&data,&format,&size,&bits,&freq);
        if ((error = alGetError()) != AL_NO_ERROR)
        {
                DisplayALError("alutLoadWAVFile wave1.wav : ", error);
                // Delete Buffers
                alDeleteBuffers(NUM_BUFFERS, buffer);
                exit(-1);
        }
        alBufferData(buffer[0],format,data,size,freq);
        if ((error = alGetError()) != AL_NO_ERROR)
        {
                DisplayALError("alBufferData buffer 0 : ", error);
                // Delete buffers
                alDeleteBuffers(NUM_BUFFERS, buffer);
                exit(-1);
        }
#endif
        return AL_TRUE;
}
// ************************************************************************
// función:  ALboolean DescargaWAV(void){
// función que maneja la descarga de la información del archivo de audio
// ya sea en Windows o linux
// ************************************************************************
ALboolean DescargaWAV(void)
```

```
{
#ifdef _WIN32

        alutUnloadWAV(format,data,size,freq);
        if ((error = alGetError()) != AL_NO_ERROR)
        {
                DisplayALError("alutUnloadWAV : ", error);
                // Delete buffers
                alDeleteBuffers(NUM_BUFFERS, buffer);
                exit(-1);
        }
#endif
#ifdef _LINUX
        free (data);
#endif
        return AL_TRUE;
}

// ************************************************************************
// función:   void carga_archivo_wav ( char *file_WAV )
// Función que se encarga de cargar y manejar el audio en la animación
// ************************************************************************
void carga_voz_en_espanol ( char *file_WAV )
{
        /* inicializo openal*/
        alutInit(NULL,0);

        /* Generamos los buffer que serán almacenados en el arrays buffer[i]
          y compruebo errores */
```

```
alGenBuffers(NUM_BUFFERS,buffer);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alGenBuffers :", error);
        exit(-1);
}
/* cargo y decargo archivo*/
CargaWAV(file_WAV);

DescargaWAV();
/* Generamos las fuentes almacenándolas en el array fuentes[i]
  y compruebo errores*/
alGenSources(NUM_FUENTES,fuente);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alGenSources 2 : ", error);
        exit(-1);
}
/* Definimos las propiedades del oyente indicándole su posición, velocidad y
orientación */
alListenerfv(AL_POSITION,listenerPos);
alListenerfv(AL_VELOCITY,listenerVel);
alListenerfv(AL_ORIENTATION,listenerOri);

/* defino propiedades de la fuente de sonido*/
alSourcef(fuente[0],AL_PITCH,1.0f);   // Con esto podemos disminuir la frecuencia
alSourcef(fuente[0],AL_GAIN,1.0f);    // Con esto definimos la ganancia en amplitud
( intensidad de sonido)
alSourcefv(fuente[0],AL_POSITION,fuentePos);   // posición
```

```
        alSourcefv(fuente[0],AL_VELOCITY,fuenteVel); // velocidad
        alSourcei(fuente[0],AL_BUFFER,buffer[0]);   // Con esto pasamos el archivo wav
del buffer a la fuente
        alSourcei(fuente[0],AL_LOOPING,AL_FALSE); // Con esto le indicamos que la
fuente no se reproduzca una y otra vez
}

/* ---------------------- Función Privada ---------------------- */
// ************************************************************************
//      int  Animacion_Sincronizacion_Emociones_Visemas_Sonido(int  prender_audio,int
seleccionar)
// Función que muestra la sincronización de las Emoes, vicemas y sonido
// ************************************************************************
void Sincronizacion_Emociones_Visemas_Sonido(int prender_audio,int seleccionar)
{
        DataPhone               *info_phone;
        ControlExpression *info_emociones;
        ControlExpression *temp_info_emociones;
        float suma_dur_fon = 0.0;
        ultimo = NULL;
        generate_PHONEMES_WAVE(prender_audio);   // Genera archivo de sonido WAV
        cargar_fonemas();                        // cargar fonemas
        carga_comandos_de_control();             // cargar Emoes
       carga_voz_en_espanol ("data_file/file_wave.wav");     // cargar el audio
        info_phone       = primero;   // Inicializa la lista de fonemas
        info_emociones   = pri_words; // Inicializa la lista de Emoes
        temp_info_emociones = pri_words;  // Genera un apuntador temporal sonre las
Emoes
        printf("\n**** Iniciando los calculos para la sincronización *****");
```

```
        if (prender_audio == 1 )
        {
                printf ("\n..... ** Audio prendido ** .....\n"); // Audio prendido
                alSourcePlay(fuente[0]);
        }
        else
        {
                alSourceRewind(fuente[0]);                    // Volver a ensender
                alSourcePlay  (fuente[0]);
                printf("\n..... ** alSourceRewind(fuente[0]) **\n");
        }
    sincronizacion_visema_Emociones ( info_phone, info_emociones ); // mostrar visemas
    info_phone = info_phone->sig;                 // Obtener el siguiente fonema

    if ( seleccionar == 1 )
   while( strncmp( info_phone->phone, "#", 2 ) != 0 )
        {  // mostrar visemas
           strcpy( info_emociones->name, "<Emo_NEUTRAL>");
           sincronizacion_visema_Emociones ( info_phone, info_emociones );

           info_phone = info_phone->sig;         // obtener el siguiente fonema

           if ( strncmp( info_phone->phone, "#", 2 ) == 0)
           {
               sincronizacion_visema_Emociones ( info_phone, info_emociones );
           }

        }  // end_while
    else
    while(strncmp( temp_info_emociones->name, "<Emo_NEUTRAL>", 17) != 0 &&
(strncmp( info_phone->phone, "#", 2 ) != 0))
        {
           temp_info_emociones = temp_info_emociones->sig;  // Apuntador a la siguiente
expresion
           if ( temp_info_emociones->time > 0 && strncmp( temp_info_emociones->name,
"<Emo_FIN>", 13 ) != 0)
                while ( (suma_dur_fon < temp_info_emociones->time) &&
(strncmp( info_phone->phone, "#", 2 ) != 0))
```

```
            {
                sincronizacion_visema_Emociones ( info_phone, info_emociones );
                suma_dur_fon = suma_dur_fon + info_phone->duration;
                info_phone = info_phone->sig;              // Obtener el siguiente fonema
            }
        info_emociones = info_emociones->sig;    // obtener la siguiente emoción

        if ( strncmp( temp_info_emociones->name, "<Emo_NEUTRAL>", 17) == 0 &&
(strncmp( info_phone->phone, "3", 2 ) == 0))
        {
            sincronizacion_visema_Emociones ( info_phone, info_emociones );
        }
    } // end_while

    free(info_phone);     // Libera memoria
    free(primero);
    free(ultimo);
    free(info_emociones);
    free(pri_words);
    free(ult_words);
    printf("\n*********** Ya termine los cálculos para sincronizar
*****************\n");
}
```

D.2 Código fuente "festival_text_speech.cpp"

```
#define SYSTEM_IS_WIN32 1
/*-------------------------------------------------------------------------
 * Filename:      festival_text_speech.h
 * Version:       1.0
 * Copyright:     Laboratorio de Graficación, Facultad de Ciencias
 * Author:        Jorge Castro <koobukai@hotmail.com>
 * Description:   Programa en "Ansi C" que permite interactuar con festival
 *                a nivel background
 * Created at:    Miercoles, 01 de abril de 2002
 * Modified by:   Jorge Castro <koobukai@hotmail.com>
 * Modified at:   Miercoles, 05 de junio de 2002
 *-------------------------------------------------------------------------
 * Bibliotecas de Festival:          libFestival.lib, libestbase.lib, libeststing.lib,
 *                                   libestools.lib, wsock32.lib, winmm.lib,  tcl80.lib
 * Controladores utilizados:         tcl80.dll, tclpip80.dll, tclreg80.dll, tk80.dll
 * -------------------------------------------------------------------------
 *
 * Programas escritos en scheme para realizar la sincronización de una
 * animación facial y el sonido dentro de Festival:
 *
 * Funciones en scheme utilizadas:   mbrola.scm
 *                                   th_mode.scm
 * -------------------------------------------------------------------------*/

/*------------------------------ Inclusiones ------------------------------*/

#include <festival.h>
#include "festival_text_speech.h"

//EST_Wave wave;

/*---------------------- Función Pública -----------------------------*/

//*************************************************************************
// Función: generate_PHONEMES_WAVE();
// Carga las funciones necesarias para crear un archivo de fonemas "file_phono.pho"
// necesario para generar un archivo "file_wave.wav" y ejecutarlo en:
// mbrola mx1/mx1 data_file/file_phone.pho y data_file/file_wave.wav
//*************************************************************************

void generate_PHONEMES_WAVE(int inicia)
```

```
{
    int heap_size = 310000;
    int load_init_files = 1;

    festival_initialize(load_init_files, heap_size);

    festival_eval_command("(load \"data_file/th-mode.scm\")");
    printf("\n\n******** Archivo 'th-mode.scm' cargado *********");

    festival_eval_command("(voice_abc_diphone)");
    printf("\n\n******** Voz 'abc_diphone' cargada *********\n\n");

    festival_eval_command("(tts \"data_file/caja_de_texto.th\" 'th)");

    festival_wait_for_spooler();

}
```

Referencias

Aguilar97	Aguilar, L. Garrido, J. M. Llisterry, "*Incorporación de conocimientos fonéticos a las tecnologías del habla*", Actas del Primer Congreso de Lingüística General. Volumen III 1997, Universidad de Valencia, paginas 5-13.
Bill99	Bill Fleming, Darris Dobbs, *Animating Facial Features & Expressions*, Charles Rivera Media, 1999.
Black94	Black Alan W., Paul Taylor, *CHATR: a generic speech synthesis system*, Proceedings of COLING-94, Kyoto, Japon 1994.
Black97	Black Alan W., Paul Taylor, *Assigning Phrase Breaks from Part-of-Speech Sequences*, Eurospeech97, Rhodes, Grecia, 1997.
Black99	Black Alan W., Paul Taylor y Richard Caley, "*The Festival Speech Synthesis System*", System documentation. Edition 1.4, for Festival Versión 1.4.0, 17 de junio de 1999.
BL85	P.Bergeron and P. Lachapelle. Controlling facial expression and body movements, In *Advanced Comuter Animation*, SIGGRAPH'85 Tutorials, Volume 2, pages 61-79. ACM, New York, 1985.
Bre82	S. E. Brennan. *Caricature generator*. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1982.
BY95	M. J. Blackmnad Yacoob. *Tracking and recognizing rigid and non-rigid facial motios using local parametric models of image motion.* In IEEE International Conference on Computer Vision, pages 374-381. IEEE Computer Society Press, Los Alamitos, CA, June 1995
BBB87	R. Bartles, J. Beatty, and B. Barsky. Introduction to Splines for Use in Computer Graphics and Geometric Modeling. Morgan Kaufmann, Los Altos, CA, 1987.
Cano01	Rafael Cano, "*Apuntes de Gramática Española*", Madrid, Facultad de Derecho, 2001.
Cath95	Catherine Pelachaud, Norman I. Badler, Mark Steedman. *Facial Expression for Speech.* Department of Computer and Information

Science, University of Pennsylvania. Philadelphia, PAA 19104-6389, 1995.

Cher71 H. Chernoff. *The use of faces to represent points in n-dimensional space graphically.* Technical Report Project NR-042-993, Office of Naval Research, Washington, DC, December 1971.

Cyb90 Cyberware Laboratory Inc. *4020/RGB 3D Scanner with Color Digitizer. Monterey*, CA, 1990

deG89 B. deGraf. Notes on facial animation. In state of the Art in Facial Animation, SIGGRAPH'89 Tutorials, Volume 22, pages 10-11. ACM, New York, 1989.

Den88 X. Q. eng. A inite Element Análisis of Surgery of the Human Facial Tissue. PhD thesis, Columbia University, New Cork, 1988.

D'Introno95 D'Introno Francisco, Enrique del Teso, Rosemary Weston. "*Fonética y Fonología Actual del Español*", Ediciones Cátedra, S.A. 1995.

Frederic96 Frederic I. Parke, Keith Waters, "*Computer Facial Animation*", A. K. Peters, Wellesley Massachusetts, 1996.

Gou71 H. Gouraud. Continous shading of curved surfaces. IEEE Trans on Computers, 20(6): 623-629, June 1971.

Gill74 M. L. Gillenson y A. V. Hill, The Interactive Generation of Facial Images on a CRT Using a Heuristic Strategy. PhD thesis, Ohio State University, Computer Graphics Research Group, Columbus, OH, March 1974.

KMMTT91 P.Kalra , A. ;amgili, N. Magnenat-Thalmann, and D. Thalmann. SMILE: a multi layered facial animation system. In IFIP WG 5.10, pages 189-198. Tkyo, 1991.

Loki00 Loki Software, "OpenAL Specification and Reference", Version 1.0 Draft Edition, 2000.

Llisterri87 Llisterri Joaquim y West Martin, "*Los sistemas de conversión de texto a voz mediante síntesis por reglas: una aproximación interdisciplinar*", Actas del II Congreso de Lenguajes Naturales y Lenguajes Formales. Barcelona: Publicaciones Universitarias, 1987, paginas 183-196.

Llisterri01 Llisterri Joaquim, "*La conversión de texto en voz*", in Quark, Ciencia, Medicina, Comunicación y Cultura 20001, paginas 79-89.

LP87 J. P. Lewis and F. I. Parke. *Automatic lip-synch and speech synthesis for character animation.* In Proc. Graphics Interface '87 CHI+CG '87, pages 143-147. Canadian Information Processing Society, Calgary, 1987.

LTW93 Y. Lee, D. Terzopoulos, and K. Waters. *Constructing physics-based facial models of individuals.* In Proc. Graphics Interface '93, pages 1-8. Canadian Information Processing Spciety, May 1993.

Park72 F. I. Parke. "Computer *generated animation of faces*". Tesis de maestría, University of Utah, Salt Lake City, UT, Junio del 1972. UTEC-Csc-72-120.

Park74 F. I. Parke. *Computer generated animation of faces*. Master's thesis, University of Utah, Salt Lake City, UT, Junio 1974

Park90 F. I. Parke, editor. State of the Art in Facial Animation, SIGGRAPH'90 Course Notes 26. ACM, New York, August 1990.

Pla80 S. M. Platt. A *system for computer simulation of the human face*. Master's thesis, The Moore School, University of Pennsylvania, Philadelphia, 1980.

Pie91 S. D. Pieper. CAPS:Computer-Aided Plastic Surgery. PhD thesis, Masachusetts Institute of Technology, Media Arts and Sciences, Cambridge, MA, September 1991

PWWH86 A. Pearce, B. Wyvill, G. Wyvill, and D. Hill. Speech and expression: A computer solution to face animation. In Proc. Graphics Interface '86, pages 136-140. Canadian Infromation Processing Society, Calgary, 1986

MTPT88 N. Magnenat-Thalmann, N. E. Primeau, and D. Thalmann. Abstract muscle actions procedures for human face animation. Visual Computer, 3(5):290-297, 1988

Richard94 Richard Barrutia, Armin Schwegler, "*Fonética y Fonología Españolas*", Editorial Wiley, 1994.

Taylor91 Taylor P, Fair I., Sutherland y Jack M., "*A real time speech synthesis system*", Eurospeech91, vol. 1, paginas: 341-244, Genoa, Italia, 1991.

Tony99 Tony Ezzat and Tomaso Poggio, *Visual Speech Synthesis by Morphing Visemes*, Massachusetts Institute of Technology, 1999.

Valenzuela00 Valenzuela Joseph I., "*Tutorial using OpenAL and OpenGL to create a 3D application with 3D sound*", Developers Gallery, © Fotis Chaztinikos, 2000.

YCH89 A.L. Yuilee, D. S. Cohen, and P. W. Hallinan. *Feature extraction from faces using deformable templates*. In IEEE Computer Society Conference on Computer Society Press, Los Alamitos, CA, June 1989.

Wat87 K. Waters. *A muscle model for animating three-dimensional facial expressions*. Computer Graphics (SIGGRAPh'87), 21(4):17-24, July 1987.

Wat96 K. Waters and T. Levergood, "*An Automatic Lip- Synchronization Algorithm for Synthetic Faces*, 149-156. In ACM, San Francisco, CA, 1994.

Wei82 P.Weil. About face. Master's thesis, Massachusetts Institute of Technology, Architecture Group, Cambridge, MA, August 1982.

Wil90 Williams. 3D paint. Computer Graphics, 24(2):225-233, March 1990.