



UNIVERSIDAD DE CHILE  
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

**MEJORAMIENTO Y EVALUACIÓN DE TÉCNICAS DE INTERPOLACION  
PARA LA ANIMACIÓN DE MALLAS FACIALES**

**MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN  
COMPUTACIÓN**

**VANESSA CAROLINA PEÑA ARAYA**

**PROFESOR GUÍA:  
NANCY HITSCHFELD KAHLER**

**MIEMBROS DE LA COMISIÓN:  
PATRICIO INOSTROZA FAJARDÍN  
ALEJANDRO HEVIA ANGULO**

**SANTIAGO DE CHILE  
ABRIL 2012**

# Resumen

En la actualidad, la animación de mallas faciales es un área de investigación muy amplia, con una gran cantidad de trabajos publicados con distintas técnicas desarrolladas. Como trabajo de memoria de ingeniería, un alumno del DCC desarrolló una aplicación en la que implementó un algoritmo de interpolación propio para animar una malla facial.

Los principales objetivos de esta memoria son saber qué tan competitivo es este algoritmo con respecto a aquellos usados en la actualidad además de definir un buen criterio para esta comparación. Asimismo, se busca identificar otras variables en los algoritmos que pudieran generar mejorías en la calidad de la malla facial. Todo esto con el fin último de mejorar la calidad de la malla.

Se propone una métrica de calidad de la malla facial que entrega información de manera rápida y sin necesidad de sujetos de prueba. Esta métrica muestra una alta relación con la evaluación perceptiva realizada a una animación específica en donde se pronuncian los sonidos que componen la base de todos los sonidos posibles. Se implementaron tres algoritmos seleccionados de la bibliografía estudiada: uno de interpolación geométrica, uno de deformación de forma libre orientada a superficies y Planar Bones. Además se propuso e implementó un algoritmo de interpolación nuevo. El diseño de la implementación permite variaciones de parámetros de los algoritmos. La comparación se hizo con una selección de 18 combinaciones entre todos los algoritmos implementados y sus variantes.

Al aplicar la métrica se concluye que el algoritmo de la aplicación legada, sin ser el mejor, presenta buenos resultados en comparación a otros algoritmos similares. Además, las variantes consideradas sí muestran mejorías en la superficie de la malla. Sin embargo, se observó que los parámetros de estas variantes deben ser bien escogidos puesto que sino se obtiene el efecto contrario.

# Agradecimientos

Agradezco principalmente a mi familia, en especial a mis dos hermanitas por todo el apoyo que me han dado y por apagar la música fuerte cuando tenía que escribir. A mis padres y a mis tatas, por enseñarme y aconsejarme.

A Jorge, mi pololo, y a mis amigos por aguantarme siempre, en especial cuando tenía mucho trabajo y los ignoraba por un rato.

Muchas gracias a mi profesora guía Nancy Hitschfeld y Mauricio Cerdá por toda la ayuda y motivación, por todas las correcciones y paciencia a través de todo el periodo de trabajo.

Vanessa Peña Araya

Marzo 2012

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Antecedentes generales . . . . .	1
1.2. Motivación . . . . .	2
1.3. Objetivos . . . . .	7
1.4. Organización de la memoria . . . . .	8
<b>2. Antecedentes</b>	<b>9</b>
2.1. Animación Facial . . . . .	9
2.2. Estándar MPEG-4 . . . . .	10
2.3. Proyecto CANDIDE . . . . .	12
2.4. Algoritmos de Interpolación Relacionados . . . . .	13
2.4.1. Interpolación geométrica . . . . .	14
2.4.2. Algoritmo Surface-Oriented Free Form Deformation . . . . .	15
2.4.3. Algoritmo Planar Bones . . . . .	17
2.4.4. Soluciones estudiadas y no implementadas . . . . .	21

2.5. Aplicación Legada . . . . .	25
2.5.1. Funcionalidades Específicas . . . . .	25
2.5.2. Algoritmos de Interpolación Iniciales . . . . .	26
<b>3. Diseño e implementación . . . . .</b>	<b>28</b>
3.1. Incorporación de estructuras de Datos . . . . .	28
3.1.1. Estructuras de datos del software de animación iniciales . . . . .	28
3.1.2. Octree . . . . .	30
3.1.3. Vecindad de triángulos . . . . .	32
3.2. Métricas de calidad de animación . . . . .	34
3.2.1. Ángulo diedro y rugosidad . . . . .	34
3.2.2. Aplicación de las métricas dentro de la animación . . . . .	36
3.3. Implementación de algoritmos de interpolación ya conocidos . . . . .	39
3.4. Incorporación de Patrones de Diseño y Calidad de Software . . . . .	42
<b>4. Resultados . . . . .</b>	<b>46</b>
4.1. Estructuras de Datos . . . . .	46
4.2. Evaluación de la métrica . . . . .	48
4.3. Evaluación de antiguos y nuevos algoritmos . . . . .	52
4.3.1. Comparación entre los algoritmos por grupos . . . . .	55
4.3.2. Sistemas de Pesos . . . . .	68
<b>5. Conclusiones y trabajo futuro . . . . .</b>	<b>70</b>

<b>Bibliografía</b>	<b>76</b>
<b>Apéndices</b>	<b>77</b>
<b>A. Diagrama UML de la estructura Octree</b>	<b>77</b>
<b>B. Diagrama UML de clases de la aplicación</b>	<b>79</b>
<b>C. Evaluación perceptiva de la animación con bajo ruido por sílaba</b>	<b>81</b>
<b>D. Rugosidad promedio de la animación de sílabas con el algoritmo Renato2</b>	<b>83</b>
<b>E. Promedio y varianza de rugosidad por algoritmo</b>	<b>84</b>

# Índice de tablas

4.1.	Evaluación de tiempo para la estructura Octree . . . . .	47
4.2.	Evaluación de tiempo para la estructura de arcos . . . . .	48
4.3.	Valores de rugosidad en prueba por triángulo . . . . .	50
4.4.	Valores de rugosidad de malla de prueba . . . . .	50
4.5.	Algoritmos o variaciones de algoritmos seleccionados para evaluar . . . . .	52
C.1.	Evaluación perceptiva de la animación con bajo ruido por sílaba . . . . .	81
C.2.	Evaluación perceptiva de la animación con bajo ruido por sílaba . . . . .	82
D.1.	Rugosidad promedio de la animación de sílabas con el algoritmo <code>Renato2</code> . . . . .	83
E.1.	Promedio y varianza de rugosidad por algoritmo . . . . .	84

# Índice de figuras

1.1.	Modelo de malla 3D Alice . . . . .	4
1.2.	Modelo CANDIDE ajustado al rostro de la malla de la cabeza 3D . . . . .	5
1.3.	Vértices que no son directamente proyectados en el modelo CANDIDE . . . . .	5
2.1.	Objeto deformado por la manipulación de una caja envolvente . . . . .	10
2.2.	<i>Facial Definition Points</i> de MPEG-4 . . . . .	11
2.3.	Vista frontal y de perfil del modelo CANDIDE . . . . .	13
2.4.	Calculando los pesos para una animación . . . . .	15
2.5.	Clasificación de áreas de Planar Bones . . . . .	17
2.6.	Enmascaramiento de descontinuidades por cavidades . . . . .	21
2.7.	Aplicación legada . . . . .	25
3.1.	Representación de un Octree . . . . .	30
3.2.	Estructura de arcos . . . . .	33
3.3.	Ángulo diedro y rugosidad . . . . .	35
3.4.	Ejemplo 2D para los valores de la rugosidad . . . . .	36

3.5. Área en las mallas donde la métrica es aplicada . . . . .	38
3.6. Diagrama UML simplificado de las clases de la aplicación legada. . . . .	45
3.7. Diagrama UML simplificado de las clases de la aplicación final. . . . .	45
4.1. Malla pequeña de prueba para la métrica de calidad . . . . .	49
4.2. Gráfico rugosidad versus evaluación perceptiva . . . . .	51
4.3. Promedio de rugosidad de las sílabas por cada algoritmo . . . . .	54
4.4. Varianza de rugosidad de las sílabas por cada algoritmo . . . . .	54
4.5. Algoritmo geométrico para la sílaba “ya” . . . . .	56
4.6. Algoritmo geométrico para la sílaba “yu” . . . . .	57
4.7. Algoritmo Point Distance para la sílaba “yu” . . . . .	59
4.8. Algoritmo Point Distance para la sílaba “fa” . . . . .	60
4.9. Planar Bones para la sílaba “ya” . . . . .	62
4.10. Planar Bones para la sílaba “yu” . . . . .	63
4.11. Malla ejecutada con el algoritmo Renato1 . . . . .	65
4.12. Renato 1 y 2 para la sílaba “ya” . . . . .	66
4.13. Renato 1 y 2 para la sílaba “yu” . . . . .	67
4.14. Comparación de sistema de pesos . . . . .	69
A.1. Clases del octree . . . . .	77
A.2. Clases cliente del octree creadas para la aplicación . . . . .	78
B.1. Template para los algoritmos de interpolación . . . . .	79

B.2. Clases para el sistema de pesos . . . . .	80
--	----

# Capítulo 1

## Introducción

### 1.1. Antecedentes generales

La animación y modelamiento de mallas faciales es un área ampliamente abarcada en la computación gráfica, con distintas técnicas adaptadas a la necesidad de la aplicación. Considerando la variedad que ellas presentan, todas coinciden en obtener movimientos que parezcan naturales y reales, balanceando calidad con rendimiento, apuntando en generar aplicaciones que sean posibles de ejecutar en tiempo real. Además, otra característica importante es la correcta adaptación a las distintas anatomías que el rostro a animar pueda presentar [7].

Una de las técnicas más intuitivas y usadas es la interpolación de movimientos de un reducido conjunto de puntos, llamados puntos de control, que actúan como iniciadores o guías del movimiento para el resto de vértices de la malla [7]. Así, para cada punto de la malla a animar, con excepción de los vértices de la nuca de la malla, se le asignará un desplazamiento de acuerdo al elemento de control que le influya. Esto último considerando que un punto de la malla a animar es influenciado solo por uno elemento de control, sin embargo, muchas técnicas aumenta el número de influenciadores para cada punto. Para esto se calcula la influencia de los puntos de control sobre cada uno de los vértices como un “peso” y cuya suma ponderada con los desplazamientos determinará cuánto se moverá finalmente el vértice. La manera de calcular este peso puede ser tanto determinada por el

animador como generada automáticamente por algoritmos.

Así como hay diferentes técnicas para animar, hay diferentes algoritmos de interpolación, cada uno con distintos resultados y diferentes rendimientos, lo que influye en qué aplicaciones son más adecuados para implementarse. Para medir estos resultados, y poder compararlos, es necesario establecer métricas o convenciones.

La presente memoria hace un estudio entre algunos de los algoritmos más utilizados en este campo de animación, para tener una idea estimativa de la competitividad del trabajo hecho anteriormente por Cerdá et al. [6] además de establecer parámetros, tanto para esta comparación como para determinar en qué tipo de movimientos la animación presenta mayores problemas.

Las principales contribuciones son:

- Implementación de tres algoritmos de animación de mallas faciales seleccionados de la bibliografía estudiada, adaptados a la aplicación.
- Propuesta e implementación de un nuevo algoritmo de interpolación.
- Implementación y utilización de una métrica de calidad de mallas 3D.
- Estudio y comparación de los algoritmos, incluyendo un análisis de las características y variables de los algoritmos que influyen en su desempeño.

## 1.2. Motivación

En el contexto del proyecto SticamSud *BAVI : Bio-inspired Audio/Visual information Integration*, se implementó un software para animar mallas faciales. El proyecto completo contempla dos equipos: el equipo chileno francés encargado de la animación; y el argentino, encargado de la extracción de movimientos a partir de información auditiva, cuyo trabajo es parte de las investigaciones de Lucas Terissi, de la Universidad Nacional de Rosario Terissi and Gómez [17], cuyos datos y parámetros son usados por el software de animación para producir el movimiento final en la malla.

El software de animación utiliza CANDIDE-3, una malla de control compuesta de una cantidad de

vértices del orden de  $10^2$  y que forman un rostro parametrizado movido por una serie de movimientos locales y globales [2], y una malla 3D de una cabeza real mucho más refinada, con aproximadamente 6000 vértices, cantidad del orden de  $10^4$ , como la que se observa en la Figura 1.1. El proyecto CANDIDE será explicado con más detalles en la Sección 2.3.

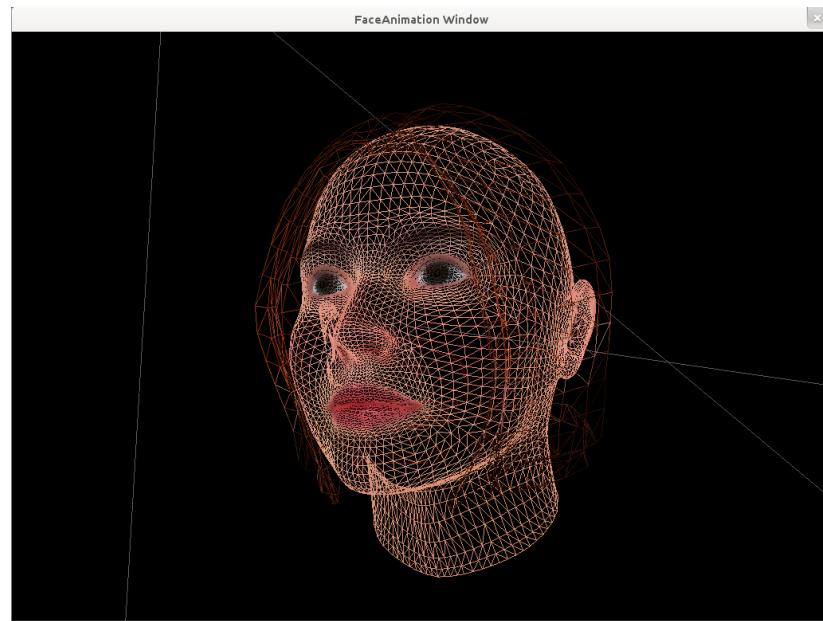
La aplicación está desarrollado en C++ y utiliza OGRE [1], uno de los motores de renderización gráfica de código abierto más populares en la actualidad para el manejo de las mallas. Los pasos seguidos para llevar acabo la animación son los siguientes [6, 19]:

1. **Ajuste manual:** se hace un ajuste manual de la malla CANDIDE para que coincida con la fisionomía del rostro de la malla de la cabeza 3D que se desea animar. Esta correspondencia se puede ver en la Figura 1.2
2. **Proyección:** luego de este ajuste se hace una proyección para mapear cada punto de la malla de CANDIDE a un, y sólo un vértice, del rostro de la malla de la cabeza 3D, obteniendo el set de vértices de control que posteriormente deformarán la malla.
3. **Interpolación con puntos de control:** en este punto ya se sabe cómo se moverán el set de puntos de control de la cabeza 3D, por lo que sólo falta saber cómo lo hará el resto de los puntos. El método propuesto por Cerda et al. [6] asocia cada vértice a un conjunto de puntos de control. Esta técnica utiliza los triángulos de la malla Candide-3 como elementos de control. Así, los puntos de control de un vértice  $P$  son los 3 vértices que definen el triángulo en la malla Candide-3 asociado a este y su movimiento por interpolación queda definida como:

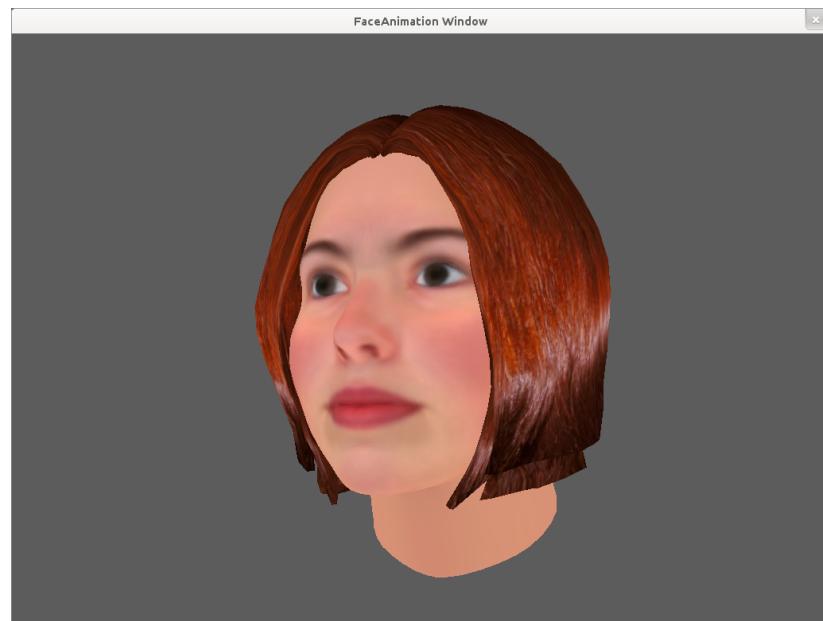
$$\vec{v}_k = \mu_0 \vec{C}_{i-1} + \mu_1 \vec{C}_i + \mu_2 \vec{C}_{i+1} \quad (1.1)$$

En donde  $\vec{v}_k$  es la posición inicial para cada vértice de la cabeza,  $\vec{C}$  es un punto de control y  $\mu = (\mu_0, \mu_1, \mu_2)$  son los tres pesos calculados para la interpolación.

Como se ve en la Figura 1.2 la malla CANDIDE solo cubre la parte del rostro de la cabeza de la malla a animar, por lo que los vértices de las zonas como el cuello u otras cavidades, como la boca y los bordes de los labios, no quedan dentro de vértices de la malla CANDIDE. Los



(a)



(b)

Figura 1.1: Modelo de malla 3D Alice, en dos vistas diferentes: (a) triangulación que la define y (b) triángulos con textura.



Figura 1.2: Modelo CANDIDE ajustado al rostro de la malla de la cabeza 3D

vértices que no quedan cubiertos por la malla CANDIDE se ven en la Figura 1.3, marcados con azul.

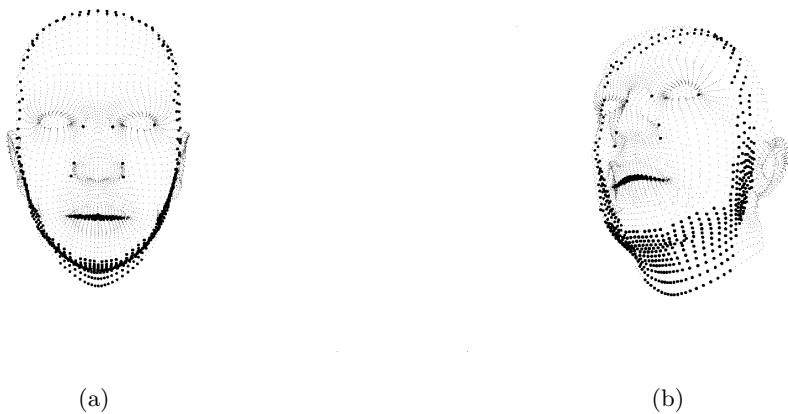


Figura 1.3: Vértices de la malla de la cabeza 3D que sí son parte de la animación pero no son directamente proyectados en el modelo CANDIDE

Por lo tanto esta solución tiene inconvenientes para interpolar puntos que están fuera de todos los triangulos de la malla CANDIDE-3. Para manejar estos casos se utiliza la interpolación:

$$\vec{v}_k = \vec{C}_{i-1} e^{-\frac{|\vec{v}_k - \vec{C}_{i-1}|}{\alpha^2}} + \vec{C}_i e^{-\frac{|\vec{v}_k - \vec{C}_i|}{\alpha^2}} + \vec{C}_{i+1} e^{-\frac{|\vec{v}_k - \vec{C}_{i+1}|}{\alpha^2}} \quad (1.2)$$

Donde  $\alpha$  es manualmente seleccionado, tomando toma valores entre 0 y 1.0, y es fijado para todos los vértices de la malla. El parámetro  $\alpha$  define cuánto serán éstos afectados por los desplazamientos de la malla CANDIDE.  $\vec{C}_{i-1}, \vec{C}_i, \vec{C}_{i+1}$  son los vértices del triángulo más cercano a  $\vec{v}_k$ , para el cual existe un camino, sin discontinuidades, en la malla para llegar a él desde  $\vec{v}_k$ . Esta restricción permite tener un control sobre las discontinuidades, no permitiendo que, por ejemplo, un triángulo de los labios superiores de la malla CANDIDE afecte a un vértices de los labios inferiores en la malla de la cabeza.

Un ejemplo claro en donde se ven las consecuencias de la interpolación de puntos que están fuera de la malla, es la animación de labios que es una situación recurrente y en donde la separación de estos generan una discontinuidad.

Aunque el método de interpolación presenta muy buenos resultados, puesto que la animación es en general correcta, se pueden observar irregularidades a nivel de labios en la malla al momento en que se anima. Esto se ve tanto en movimientos amplios, como por ejemplo cuando se abre la boca de manera exagerada, en los cuales el labio inferior genera una forma en “v” en vez de mostrar un labio redondo; como en movimientos más pequeños, en donde los triángulos que componen los labios se juntan de manera poco homogénea generando una superficie muy poco regular. Esto principalmente se debe a que la malla en ese sector es una cavidad, un hoyo, que se traduce en una discontinuidad por lo que el método de interpolación actual tiene problemas en la identificación adecuada del triángulo de la malla CANDIDE en donde un punto queda proyectado.

## Medición de la calidad de la animación

Los usos de este software son muy variados. Entre ellos está el apoyo visual para conferencias en las que solo es posible tener el audio y para personas con problemas de habla. Para este último caso existen estudios que verifican que las personas al verse ellos mismos hablando, les es mucho mas fácil reeducar problemas del habla.

La evaluación de la calidad por ahora ha sido solamente en experimentos en laboratorio en donde se les pregunta a un conjunto de personas y ellas evalúan la animación. Sin embargo este proceso es por un lado costoso y por el otro muy demoroso, ya que luego se debe recoger la información y analizarla. Esto significa que es necesario una manera de automatizar este proceso para hacer más eficiente esta extracción de información.

Dentro del trabajo por la parte auditiva se generó un set de bases de datos con información de la calidad de una animación en donde se pronuncia un conjunto de sílabas, dada cierta cantidad de ruido en la generación de la animación Terissi et al. [18]. Esto es, dado un conjunto de sílabas que en la animación presentan menor o mayor rugosidad en la superficie de los labios, se tienen evaluaciones de calidad para cada sílaba, dado el ruido en la señal de audio con la que fue generada la animación. Esta base de datos se encuentra en el Apéndice C. En esta base de datos se identifican las sílabas que presentan más problemas. Esto, aunque no se puede ocupar como una prueba 100 % feasible de lo que es calidad de animación, sirve mucho para tener una primera aproximación para probar técnicas de evaluación a futuro, además de analizar qué tan exacta es la métrica para así determinar la necesidad de poder prescindir de evaluadores humanos y automatizar por completo el proceso.

Dado el amplio estudio de animación de rostros, es importante tener una idea estimativa del rendimiento, tanto en calidad como rapidez a nivel de ejecución, en comparación a otros métodos ya utilizados, concluyendo la competitividad del método de interpolación actualmente implementado.

### 1.3. Objetivos

El objetivo principal de la memoria es la mejora del software de animación actual, enfocándose principalmente en la calidad de la animación, deseando obtener movimientos más realistas, en particular a nivel de labios que es donde se encuentran los principales problemas.

Los objetivos específicos son:

- Identificar movimientos que generen casos problemáticos. En este sentido se desea una formalización simple de lo que se entiende como “movimientos amplios”, por ejemplo.

- Optimizar el manejo de la malla de la cabeza 3D, en particular en las etapas de proyección e interpolación.
- Cuantificar la calidad de la malla en determinadas zonas o movimientos.
- Implementar y analizar comparativamente otros métodos de interpolación [8, 10].
- Analizar la localidad de la influencia de los puntos de control.
- Mejorar la calidad y diseño del software.

#### **1.4. Organización de la memoria**

La presente memoria está organizada de la siguiente manera: todos los antecedentes necesarios para entender lo implementado en la aplicación están detallado a continuación (Capítulo 2). En este capítulo se hace una breve introducción sobre lo que existe con respecto a animación facial; de los proyectos y librerías utilizadas en la aplicación; una revisión bibliográfica de todos los algoritmos implementados y de aquellos que se estudiaron pero no llegaron a ser incluidos en el software. Finalmente se explica la aplicación legada, explicando los algoritmos inicialmente implementados.

En el capítulo siguiente, Diseño e Implementación, se explican las estructuras de datos iniciales y las nuevas que serán de ayuda a éstas. Posteriormente, se describe la métrica a utilizar y luego los algoritmos implementados. Finalmente se describe la estructura de clases del software.

En el Capítulo 4 se exponen los cambios al implementar las estructuras, los valores de la métrica obtenidos por cada algoritmo, comparando las distintas versiones de cada uno y luego una comparación global, intentando buscar el mejor algoritmo. Por último se hace una pequeña revisión de la complejidad de cada algoritmo.

El capítulo final presenta las conclusiones y el trabajo futuro de interés potencial.

# Capítulo 2

## Antecedentes

### 2.1. Animación Facial

Como se presentó en la introducción, la animación de rostros es un área ampliamente estudiada desde hace muchos años: el primer rostro humano modelado computacionalmente fue el trabajo de Parke [14] en 1972. Aunque la evolución desde este trabajo ha sido muy significativa, logrando animación de caras con resultados muy realistas y con buen rendimiento computacional, la animación de rostros sigue siendo una tarea tediosa y que requiere intervención humana.

A modo general, existen dos categorías principales de técnicas de animación facial: aquellas basadas en manipulaciones geométricas y aquellas basadas en manipulaciones de imágenes, cada una con varias subcategorías. Algunas técnicas basadas en manipulaciones geométricas son las interpolaciones entre frames de animación específicos o interpolaciones geométricas; parametrizaciones y deformaciones de forma libre (*free-form deformations*, por sus siglas en inglés). Dentro de las manipulaciones de imágenes se incluyen lo que en inglés se denomina *morphing*, que es una transición con desvanecimiento entre dos imágenes fotográficas y mezcla entre imágenes (*blending* en inglés), entre otras [7].

Aquellas con mayor relación con el trabajo de esta memoria, y que por lo tanto serán explicadas, son la animación por **interpolación** y **free-form deformation**.

La interpolación usualmente presenta una transición suave entre dos estados o frames de una

animación en estado extremos. Sin embargo, también existe la interpolación geométrica, que actualiza directamente las posiciones de los vértices, ya sean 2D o 3D, y la interpolación paramétrica, que controla funciones que mueven los vértices indirectamente [7].

La principal desventaja de esta técnica es que se pueden producir una cantidad no muy grande de movimientos; puede ser muy restrictiva.

Free Form Deformation (FFD) deforma un objeto volumétrico manipulando un conjunto de puntos de control que lo encierran [16]. Así, un objeto 3D es encerrado en una caja imaginaria de los vértices de control en forma de grilla, como se ve en la Figura 2.1. Cualquier deformación de estos puntos de control generarán que el objeto encerrado también sea deformado. FFDs puede变形 distintos tipos de primitivas geométricas, incluyendo polígonos y modelos sólidos. La mayor ventaja de FFD y sus variantes es que la deformación no depende de la superficie a mover sino que puede abstrarse por la caja envolvente [7].

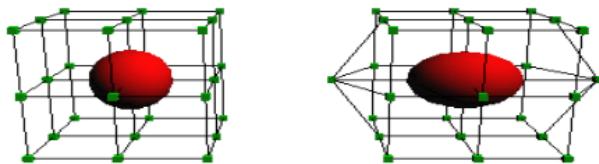


Figura 2.1: Objeto deformado por la manipulación de una caja envolvente con vértices de control

## 2.2. Estándar MPEG-4

MPEG-4 es un estándar ISO/IEC<sup>1</sup> desarrollado por MPEG (Moving Picture Experts Group), el comité que también desarrolló los premiados estándares MPEG-1 y MPEG-2. Estos estándares

---

<sup>1</sup> ISO por *International Organization for Standardization* (Organización Internacional para la Estandarización), e IEC por *International Electrotechnical Commission* (Comisión Electrónica Internacional)

hicieron posible videos interactivos en CD-ROM, DVD y televisión digital <sup>2</sup>. MPEG-4 está formado por una serie de estándares, precisamente 21, llamadas “partes”, como por ejemplo las números 2 y 3, que describen el conjunto de codecs de compresión de video y audio respectivamente.

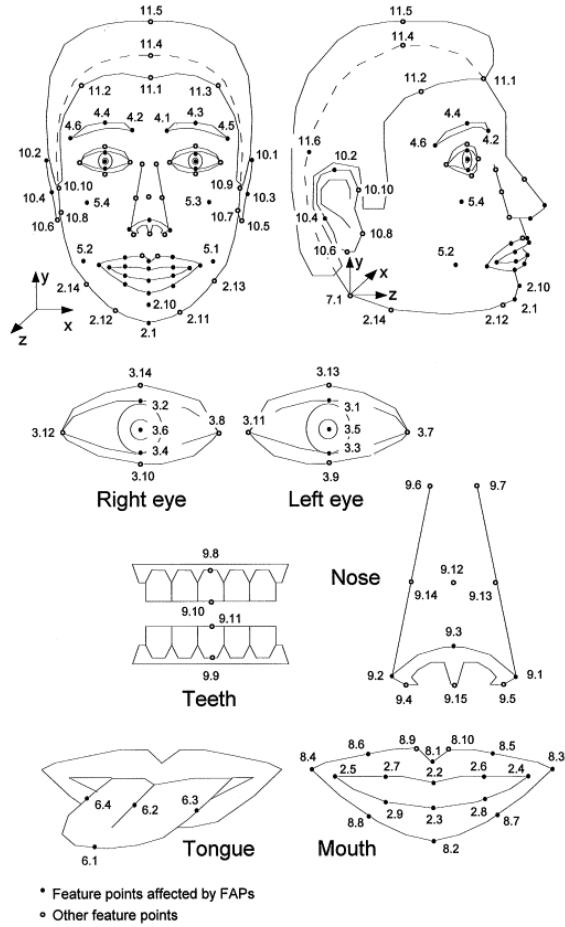


Figura 2.2: *Facial Definition Points* de MPEG-4

Con respecto a lo que es animación facial, MPEG-4 utiliza Parámetros de Definición Facial (FDPs) y/o Parámetros de Animación Facial (FAPs) para controlar la forma, texturas y expresiones de un rostro<sup>3</sup>. Los FDPs son un conjunto de puntos establecidos sobre el rostro humano con la finalidad de estandarizar los movimientos de este. Estos puntos se ven en el Figura 2.2. Por otra FAPs son un conjunto de 68 articulaciones básicas detalladas, las cuales se pueden combinar y generar movimientos más complejos y realistas. Algunos de los FAPs existentes son:

<sup>2</sup><http://mpeg.chiariglione.org/standards/mpeg-4/mpeg-4.htm>

<sup>3</sup>*Facial Definition Parameters* y *Facial Animation Parameters* sus nombres originales en inglés

- stretch\_l\_cornerlip: Movimiento horizontal de la zona inferior izquierda del labio.
- stretch\_r\_cornerlip: Movimiento horizontal de la zona inferior derecha del labio.
- thrust\_jaw: Movimiento de la mandíbula en dirección perpendicular al rostro.
- stretch\_l\_nose: Movimiento horizontal del lado izquierdo de la nariz.

La finalidad de esta sección es ofrecer contexto para la siguiente, en donde se explica el proyecto CANDIDE que es el principal componente de la aplicación de animación facial.

### **2.3. Proyecto CANDIDE**

La malla Candide es una máscara facial desarrollada para representar modelos de rostros humanos, usando un número reducido de puntos de control y polígonos (aproximadamente 100) permitiendo una rápida reconstrucción usando moderado consumo computacional.

Los movimientos de Candide son controlados por Unidades de Acción (AUs por sus siglas en inglés). Las AUs parametrizan los movimientos de los vértices de la malla, generando los movimientos que son posibles realizar por un rostro. Las Unidades de Acción globales corresponden a rotaciones alrededor de los tres ejes, mientras que las locales controlan los movimientos de la cara para que se puedan obtener diferentes expresiones.

El proyecto inicial fue creado en 1987, siendo la malla conformada por 75 vértices y 100 triángulos, la cual no es usada muy comúnmente. Posteriormente fueron creadas Candide-1 y Candide-2, las cuales poseían variaciones en la cantidad de vértices, polígonos y AUs.

La tercera versión de CANDIDE fue creada en el 2001, derivada de la original y es la que utiliza la aplicación de la presente memoria. Su principal objetivo era la simplificación de la animación de MPEG-4, con muchos de los puntos de control correspondientes a los este modelo. El modelo Candide-3 se observa en la Figura 2.3

El sitio web del proyecto [2], ofrece de manera gratuita la descarga y utilización de la malla, que viene en formato wireframe (.wfm), además de los movimientos de vértices tanto para los Action

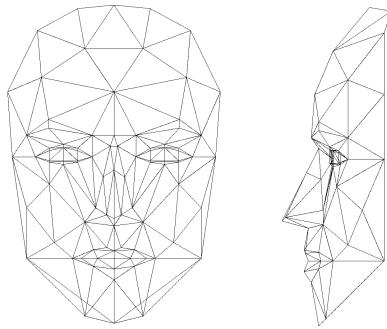


Figura 2.3: Vista frontal y de perfil del modelo CANDIDE

Units como para los FPs. De esta manera se tiene una representación que permite el movimiento de puntos de control cuyos desplazamientos influirán en los vértices de la malla de la cabeza 3D a través de técnicas de interpolación. La aplicación legada utiliza esta representación para generar los movimientos que finalmente moverán a la malla objetivo. La principal diferencia entre las unidades de animación de las mallas Candide en comparación al modelo MPEG-4, es que una AU de Candide mueve más puntos de animación que una de MPEG-4, lo que al final se traduce como que con Candide se puede controlar una animación con menos parámetros que con MPEG-4.

Por último, están las *Shape Units*, que definen las medidas de los componentes del rostro, como son la posición vertical de los ojos, el ancho y alto de los ojos, la extensión y posición vertical de la nariz y el ancho de la boca. Ajustando estos parámetros, que generan desplazamientos de los vértices correspondientes, se pueden obtener distintas fisonomías. Esta característica de generalidad del modelo permite que sea adaptado a cualquier rostro. Las *Shape Units* son utilizadas en la aplicación de la presente memoria para ajustar el modelo Candide-3 con la malla de la cabeza 3D.

## 2.4. Algoritmos de Interpolación Relacionados

Para la presente memoria se hizo un estudio de algunos de los algoritmos más usados para la animación de mallas. Los algoritmos presentados en las secciones 2.4.1, 2.4.2 y 2.4.3 fueron seleccionados para su implementación y comparación con el algoritmo explicado en Sección 1.2. Los algoritmos que fueron estudiados pero no implementados están resumidos en la Sección 2.4.4, además

de algunas soluciones que no dependían del algoritmo en sí, sino que de otras variantes que podrían influir en la calidad del movimiento de los labios.

#### 2.4.1. Interpolación geométrica

Este algoritmo de interpolación es el trabajo de Kshirsagar et al. [11] y está dividido en dos pasos: *Inicialización* y *Deformación*.

En el paso de inicialización lo primero que se hace es recorrer la malla y guardar las relaciones de vecindad entre los puntos de control. Este recorrido se hace partiendo desde cada vértice de control y avanzando de un arco a la vez, creando zonas de influencia de cada vértice. Dos puntos de control son vecinos si existe un arco que conecta las áreas a las que afecta cada uno. Luego de esto se hacen dos cosas: (1) se calcula los puntos de control que afectan a cada vértice de la malla y (2) se calculan los pesos asociados a cada uno, es decir cuánto son influenciados por estos puntos de control.

Los pesos de cada vértice se calculan dados los puntos de control que tiene como influencia. Supóngase un vértice  $P$  abordado por un punto de control  $FP_1$ , que a su vez tiene a  $FP_2$  y  $FP_3$  como vecinos dadas las relaciones de vecindad calculadas al inicio, como se muestra en la Figura 2.4.  $FP_2$  y  $FP_3$  son elegidos de este conjunto de vecinos de manera que sus ángulos entre ellos y  $FP_1$ ,  $\theta_2$  y  $\theta_3$ , medidos en radianes, sean los menores de entre todos los puntos de control que están en la vecindad de  $FP_1$ , y además sean menores que  $\frac{\pi}{2}$ .

El peso de un vértice  $P$  dado un punto de control  $i$  es calculado como sigue:

$$W_{i,p} = \sin\left(\frac{\pi}{2}\left(1 - \frac{d_{i,p}}{d}\right)\right) \quad (2.1)$$

Donde  $d_{i,p}$  es la distancia entre el vértice  $P$  y el punto de control  $i$ , y  $d$  es la suma ponderada de las distancias en la superficie de todos los vecinos del vértices de control que influencia al vértice  $P$ .

Por ejemplo en la Figura 2.4, el peso de influencia de  $FP_1$  sobre  $P$  es:

$$W_{1,P} = \sin\left(\frac{\pi}{2}\left(1 - \frac{d_{1,P}}{d}\right)\right) \quad (2.2)$$

Y dado que FP2 y FP3 cumplen la condición descrita anteriormente, la suma ponderada de las distancias entre ellos y FP1 esta dada por:

$$d = \frac{d_{12} \cos \theta_2 + d_{13} \cos \theta_3}{\cos \theta_2 + \cos \theta_3} \quad (2.3)$$

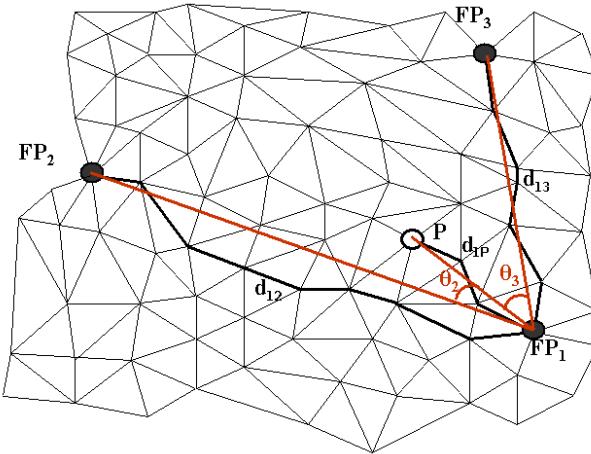


Figura 2.4: Calculando los pesos para una animación

El desplazamiento final  $D_p$ , aplicado en el paso de *Deformación*, sobre un vértice  $P$  por los  $n$  puntos de control está dado por la siguiente ecuación:

$$D_p = \frac{\sum_{i=1}^n \frac{W_{i,p} D_i}{d_{i,p}^2}}{\sum_{i=1}^n \frac{W_{i,p}}{d_{i,p}^2}} \quad (2.4)$$

#### 2.4.2. Algoritmo Surface-Oriented Free Form Deformation

Karan [9] aplica un algoritmo de interpolación orientada a superficies que se divide en tres fases: fase de vinculación, fase de registro y fase de deformación.

Durante la fase de vinculación se hace el mapeo entre la malla de control y la del objeto. La fase de registro calcula cuánto afectan los elementos de control, los triángulos de la malla que genera el movimiento, a cada punto  $P$  de la malla a deformar, calculando los pesos con la ecuación

Ecuación 2.5.

$$w_k^P = \frac{1}{1 + (d_k^P)^{local}} \quad (2.5)$$

Donde  $d_k^P$  es la distancia entre  $P$  y el triángulo de control  $k$  y *local* controla la razón por la cual decae la función  $w_k^P$  con el aumento de la distancia. En otras palabras, entre mayor es la distancia entre el triángulo de control  $k$ , más baja es la influencia de éste sobre  $P$ . Las influencias sobre el punto  $P$  son normalizados para preservar la propiedad de convexidad. Por esto  $u_k^P$  es el peso normalizado del elemento de control  $k$  para el punto  $P$  se define como:

$$u_k^P = \frac{w_k^P}{\sum_{i=1}^n w_k^P} \quad (2.6)$$

En la práctica  $u_k^P$  se hace cero para la mayoría de los elementos de control, haciendo que la solución sea lo suficientemente eficiente para animación en tiempo real.

Además de los pesos de influencia, la fase de registro calcula la representación, sin deformar, es decir, en posición neutral, de todos los vértices de la malla de la cabeza en el sistema de coordenadas de todos los triángulos de control. Dado un triángulo con vértices  $A$ ,  $B$  y  $C$ , los ejes que definen este sistema de coordenadas están dados por:

$$\vec{e}_1 = \vec{B} - \vec{A}, \vec{e}_2 = \vec{C} - \vec{A}, \vec{e}_3 = \frac{\vec{e}_1 \times \vec{e}_2}{\|\vec{e}_1 \times \vec{e}_2\|}$$

Con estos ejes, se puede calcular la matriz de transformación del elemento de control  $k$  en la malla de control en posición neutral, con notación  $R$ , como  $Q_k^R = \begin{pmatrix} \vec{e}_1 \\ \vec{e}_2 \\ \vec{e}_3 \end{pmatrix}$ .  $P_k^R = P(Q_k^R)^{-1}$ , donde  $P_k^R$  es la representación del punto  $P$  en el sistema de coordenadas local del triángulo de control  $ABC$  en la malla del control en posición neutral.

En la fase de deformación todos efectos de los pesos son agregados para calcular el punto resultante en la malla a deformar:

$$P_{def} = \sum_{k=1}^n u_k^P P_k^{def} \quad (2.7)$$

Donde  $P_k^{def} = P_k^R(Q_k^D)$ , donde  $(Q_k^D)$  es la matriz de transformación de la malla de control con el desplazamiento de la animación, u otro iniciador de movimiento.

El algoritmo de Karan muestra buenos resultados y el parámetro *local* provee un buen control sobre la localidad de la deformación.

#### 2.4.3. Algoritmo Planar Bones

Planar Bones [12] es una reformulación de Surface-oriented Free Form Deformation, que divide el desplazamiento asignado dada la influencia de un triángulo de vértices  $ABC$  sobre un punto  $P$  dependiendo de la zona en donde se encuentra el punto con respecto a este. Las clasificación de las zonas en las que puede encontrarse un vértice se muestran en Figura 2.5.

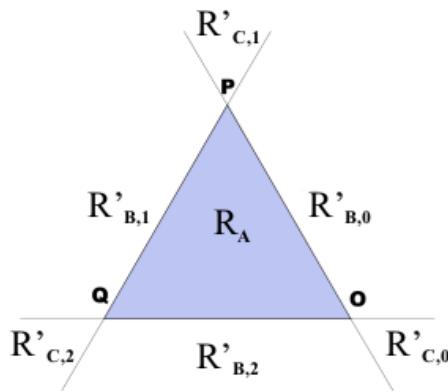


Figura 2.5: Clasificación de áreas de Planar Bones

Dado un triángulo de vértices  $ABC$  que se desplaza hasta quedar en las posiciones definidas con vértices  $abc$ , la nueva posición  $p$  para el caso base en el que el punto está dentro del elemento de

control (caso  $R_A$ ), es calculada según la siguiente ecuación:

$$p = a + (P - A) \begin{pmatrix} B - A \\ C - A \\ \hat{n} \end{pmatrix}^{-1} \begin{pmatrix} b - a \\ c - a \\ \hat{n}' \end{pmatrix} \quad (2.8)$$

En donde  $\hat{n}$  y  $\hat{n}'$  son las normales del plano formado por el triángulo antes y después del desplazamiento, respectivamente. Esta fórmula puede ser vista como un caso especial de coordenadas baricéntricas en 3D que por asuntos de espacio no se explicará en el presente informe.

Las zonas  $R_{B,i}$  y  $R_{C,i}$  definen los desplazamientos definitivos de cada punto calculándose según una fórmula asignada a cada una. Para saber a cuál pertenece un vértice, se hace una segunda clasificación:

$$\forall S \in R'_{B,i}$$

$$\begin{cases} c_i(S) < 0 & \Rightarrow S \in R_{C,i} \\ c_i(S) \in [0, \| e_i \|] & \Rightarrow S \in R_{B,i} \\ c_i(S) \geq 0 & \Rightarrow S \in R_{C,i+1} \end{cases}$$

$$\forall S \in R'_{C,i}$$

$$\begin{cases} c_{i-1}(S) \leq \| e_{i-1} \| & \Rightarrow S \in R_{B,i-1} \\ (c_{i-1}(S) \leq \| e_{i-1} \wedge c_i(S) < 0) & \Rightarrow S \in R_{C,i} \\ c_i(S) > \| e_i \| & \Rightarrow S \in R_{B,i} \end{cases}$$

Donde para cada triángulo  $ABC$ :

$$\begin{aligned} e_j &= ABC_{j+1} - ABC_j \\ c_j &= \frac{(S - ABC_j) * e_j}{\| e_j \|} \end{aligned}$$

Donde todos los índices deben tomar módulo 3. Esta notación quiere decir simplemente que se les ponen índices a los vértices del triángulo. Así, si el vértice  $A$  tiene índice 1, el vértice  $B$  tiene índice 2 y el vértice  $C$  tiene índice 3;  $ABC_1 = A$  y  $e_1 = B - A$ .

Para los puntos pertenecientes a las regiones  $R_{B,i}$ :

$$B_i^{ABC} = \begin{pmatrix} e_i \\ \frac{(e_i) \times \hat{n}'}{\| (e_i) \times \hat{n}' \|} \\ \hat{n}' \end{pmatrix} \quad (2.9)$$

$$P' = a + (p - A) (B^{ABC})^{-1} (B^{abc}) \quad (2.10)$$

Para los puntos pertenecientes a las regiones  $R_{C,i}$ :

$$C_i^{ABC} = \begin{pmatrix} \frac{(e_i) \times \hat{n}'}{\| (e_i) \times \hat{n}' \|} \\ \frac{(e_{i-1}) \times \hat{n}'}{\| (e_{i-1}) \times \hat{n}' \|} \\ \hat{n}' \end{pmatrix} \quad (2.11)$$

$$P' = a + (p - A) (C^{ABC})^{-1} \text{diag} \begin{pmatrix} \frac{Di(P)}{di(P)} \\ \frac{Di(P)}{di(P)} \\ 1 \end{pmatrix} (C^{abc}) \quad (2.12)$$

Para un vector  $v$  de dimensión  $N$ , en este caso  $N = 3$ ,  $\text{Diag}(v)$  es definida como una matriz diagonal de  $N \times N$ . Los coeficientes  $Di(P)$  y  $di(P)$  están definidos como:

$$Di(P) = \| \Pi_{ABC}(P) - A \| \quad (2.13)$$

$$di(P) = \| (\Pi_{ABC}(P) - A) (C^{ABC})^{-1} C^{abc} \| \quad (2.14)$$

Para combinar la influencia de más de un triángulo de control, a cada punto en la malla a animar se le asocia un peso, para que el desplazamiento final de un punto  $P_i$  esté dado por la siguiente ecuación:

$$P_i = \sum_{j=1}^n w_{ij} P_i^j \quad (2.15)$$

Donde  $P_i^j$  es el desplazamiento de  $P_i$  asociado al j-ésimo triángulo de control que lo influencia y  $w_{ij}$  el peso asociado a esta influencia. Estos pesos son calculados en función a la distancia entre el vértice a animar y el triángulo de control, denominada  $d_{ij}$  para el ejemplo presentado. Para calcular la distancia entre un vértice y un triángulo, se debe elegir una función que sea igual a uno cuando la distancia entre ellos sea cero y decaiga a medida que la distancia entre ellos aumenta. Esto es puesto que se quiere definir que la influencia de un triángulo de control sobre un vértices disminuye a medida que la distancia entre ellos aumenta. En la reformulación de Planar Bones de Lorenzo and Maddock [12], la función utilizada por está descrita por la Ecuación 2.16.

$$w_{ij}^n = \begin{cases} \left( \frac{1}{2} \left( 1 + \cos \left( \frac{d_{ij}}{r_j} \pi \right) \right) \right)^n & \text{si } r_j > d_{ij} \\ 0 & \text{si no} \end{cases} \quad (2.16)$$

En donde  $r_j$  es el radio definido para que el triángulo j-ésimo afecte al vértice  $P_i$ , es decir,  $r_j$  define la distancia máxima para que un triángulo afecte a un vértice. Para la implementación se seleccionaron para  $r_j$  valores con respecto al largo de las aristas del triángulo, bajo la hipótesis que un triángulo más grande influenciará a una mayor cantidad de vértices.

Para manejar discontinuidades en la malla a animar, como las que se producen en la boca y al abrir los ojos, Planar Bones [12] enmascaran alrededor de estas áreas con mapas de texturas como se observa en la figura Figura 2.6. Con enmascaramiento se refieren a tener una “textura”, indicando qué vértices pertenecen a las mismas áreas. En el caso de la Figura 2.6, dos áreas distintas serían las de la mandibula de arriba y la de abajo.

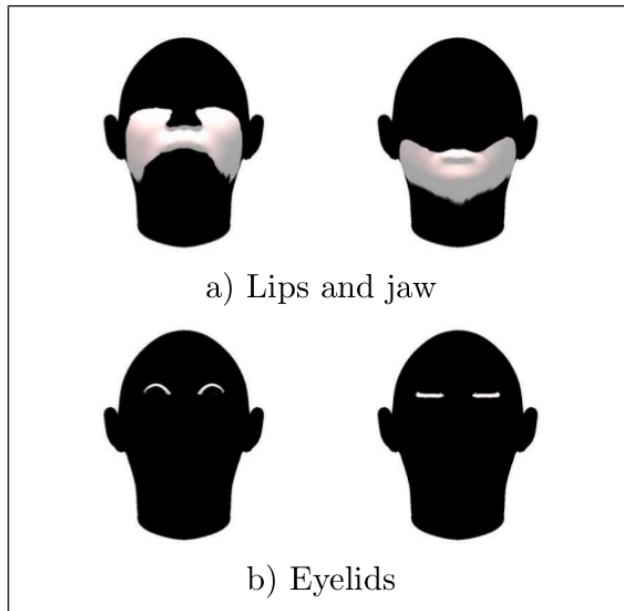


Figura 2.6: Enmascaramiento de descontinuidades por cavidades

#### 2.4.4. Soluciones estudiadas y no implementadas

Las siguientes secciones resumen otros algoritmos y soluciones estudiados que finalmente no fueron seleccionados para implementar.

##### Interpolación basada en funciones radiales

Las funciones de base radial son definidas como funciones reales cuyo valor depende solamente de la distancia desde el origen hasta el punto que evalúa y pueden ser usadas por métodos de interpolación para calcular el desplazamientos de los vértices, haciendo la interpolación desde un reducido conjunto de vértices y usando un radio de soporte que define la localidad de la deformación. El algoritmo puede ser resumido como sigue:

La función de interpolación,  $s$ , describiendo la interpolación de todo el dominio, puede ser aproximada por la suma de las funciones (basales):

$$s(X) = \sum_{i=1}^N \alpha_i \phi(\|X - X_{k_i}\|) + p(x) \quad (2.17)$$

La función basal real  $\phi$  puede ser seleccionada para tener mejoras tanto en la calidad como en el rendimiento de la animación. El valor  $X$  es la posición de un vértice normal y  $\{X_{k_i}\}_{i=1}^N$  es la posición del  $i$ -ésimo vértice de control. Los coeficientes  $\alpha = (\alpha_1, \dots, \alpha_N)^T$  y el polinomio  $p(x)$  son elegidos de tal manera que  $s$  interpole a  $\phi$  exactamente en los puntos de control:

$$s(x_{k_i}) = \phi(x_{k_i}), 1 \leq i \leq N, \quad (2.18)$$

y  $\alpha$  debe ser tal que

$$0 = \sum_{i=1}^N \alpha_i q(X_{k_i}) \quad (2.19)$$

para todos los polinomios de primer grados  $q$ . Esto lleva al sistema de ecuaciones algebraicas (SLAE por sus siglas en inglés) para que los coeficientes del polinomio y  $\beta$ , tomen la forma:

$$\begin{pmatrix} M & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix} \quad (2.20)$$

Donde  $M$  es la *matriz de interpolación*

$$M_{ij} = \phi(\|X_{k_i} - X_{k_j}\|), 1 \leq i, j \leq N \quad (2.21)$$

y  $P$  es la matriz definida por las restricciones para interpolar todos los polinomios de primer grado exactamente. Resolviendo esta ecuación para cada frame de la secuencia de animación se calculan los desplazamientos en cada dirección de las coordenadas.

Las funciones de base radial proveen una manera de interpolación muy general y flexible, contando con muy buenas propiedades de aproximación. Por esto, su uso es amplio y en muchas y variadas áreas.

Un hecho notable de este método de interpolación es que de las funciones de base radial pueden ser elegidas de acuerdo a lo que se necesita en la aplicación. Jakobsson and Amoignon [8] hace un conjunto de test muy completo usando y comparando 5 distintos tipos de funciones: cuadrática inversa, multicuadrática, multicuadrática inversa, Gaussiana y la Wenland función de soporte compacto. La calidad de la malla es medida contrastando la densidad de los puntos de control y los vértices normales, considerando la proporción entre vértices de control y normales. En este experimento la calidad de la malla es analizada con tres métricas, comparando el valor máximo y el promedio de cada una.

Esto puede ser muy útil ya que significa que, con una misma implementación, se pueden obtener distintos resultados solo cambiando la función de base radial usada en el algoritmo. Esto permite la opción de privilegiar y equilibrar entre calidad o rendimiento y, seleccionando la función correcta, este método puede ser utilizado para animación en tiempo real. Sin embargo, y siguiendo el mismo argumento, una mala elección en la función puede llevar a un método de interpolación muy malo e imposible de usar.

La principal desventaja es la manera de resolver la SLAE, puesto que puede ser muy lento, además que a veces es necesario imponer restricciones para su resolución.

### Mallas densas para interpolar

Este es el trabajo de Berger *et al.* [5] cuyo foco principal es la animación realista de malla animadas, siguiendo la misma idea de mapear el movimiento de un objeto a otro, con un enfoque diferente ya que la técnica es usada para que una malla densa coincida con una más dispersa de una misma persona, evitando el ajuste manual.

Este método requiere la adquisición de un conjunto de mallas densas en 3D de un interlocutor. Para su estudio, las mallas fueron obtenidas con el Inspect mega capturer<sup>4</sup> para quince sonidos prolongados.

Cuando se calculan las coordenadas 3D de cada marcador en el rostro del interlocutor, es muy

---

<sup>4</sup>[www.inspekt.com](http://www.inspeckt.com)

probable que un conjunto de ellos no sea observable para el sistema de estereovisión o sus coordenadas no serán lo suficientemente confiables. Esto sucede, por ejemplo, cuando los labios están cerrados o en casos en que la cabeza se mueve escondiendo partes del rostro. Los autores estiman que un 77 % de los marcadores son siempre observables y por lo tanto reconstruidos. 17 % tienen un porcentaje de reconstrucción en el intervalo entre [70 %, 95 %] y 6 % de los marcadores tienen un porcentaje de reconstrucción menor del 50 %.

Por esto, ellos usan un análisis de componentes principales para lograr dos objetivos: (i) reconstruir la malla con los marcadores que son siempre visibles y luego, usando técnicas de esperanza-maximización, completar las entradas los componentes principales con los valores de los marcadores que no sean observables en todo momento; y (ii) calcular cada movimiento de la malla menos densa, para usarlos para aproximar cada movimiento de las mallas densas como una combinación entre ellas, es decir, transfiriendo la animación de las mallas dispersas a las mallas más densas.

La mayor ventaja de este método es el uso directo de las mallas densas que contienen mucha más información, obteniendo un movimiento mucho más realista. Sin embargo, este enfoque no es 100 % igual a los ya explicados puesto que su implementación significaría un completo rediseño de la solución actual.

## Otras soluciones

La implementación de nuevos algoritmos no fueron las únicas soluciones que se contemplaron para el movimiento irregular que se observa en los labios, puesto que no es la única razón por la que esto se genera. Otras fuentes son las técnicas para obtener la animación o la inexactitud de la identificación de los marcadores puestos en el rostro para registrar los movimientos de una persona. Una de las opciones discutidas fue el agregar más marcadores y puntos de control en las áreas más problemáticas, como los labios. Con esto, sería posible tener mucho más control sobre esos lugares y mayor detalle, obteniendo un control mucho más refinado de la animación. Además, se consideró la obtención de nuevas animaciones con diferentes técnicas a las que se utilizan actualmente, esto para descartar que los problemas observados vienen por parte de la animación.

## 2.5. Aplicación Legada

Como se dijo anteriormente el software legado y sobre el cual se basó la realización de esta memoria está desarrollado en C++ y utiliza OGRE como motor de renderización gráfica.



Figura 2.7: Aplicación legada

De manera general, el software permite la carga de mallas muy refinadas, del orden de  $10^4$  , en formato Ogre y su animación usando la malla de Candide.

### 2.5.1. Funcionalidades Específicas

- Cargar mallas en formato ogre (wrl).
- Como la animación requiere un ajuste entre ambas mallas, la aplicación permite que este ajuste sea hecho de manera manual, además de permitir guardar y cargar ajustes ya realizados. Estas opciones están en el lado izquierdo del menú con las etiquetas de “Load” y “Save”, para cargar y guardar respectivamente. La edición de este ajuste se hace marcando el checkbox que se encuentra en la parte de arriba de la pantalla.
- Realizar la proyección entre ambas mallas, para la posterior animación.

- Cargar un archivo de animación para luego reproducirla. Este se hace con el botón “Read AUV”, para cargarla, y luego marcando el checkbox de “play” para reproducirla. El software mostrará en la parte posterior una barra de reproducción indicando el número de frame que está mostrándose.

El software cuenta con un archivo setup que contiene distintas posibles configuraciones a cargar, en las que se permite indicar la malla a mostrar y el algoritmo a utilizar. Un ejemplo de una configuración se muestra a continuación:

```
name=Juan Carlos
meshpath=JuanCarlos/FaceComplete.mesh
teethpath=
position=0 0 0
rotation=0 110.0 0
meshindex=5
alpha=.4
registration=conf_JuanCarlos.txt
animation=video_data.fau
interpolation=renato3
default=1
```

Cuyos valores más importantes son **meshpath**, que indica la malla a cargar; **registration**, que carga el ajuste manual indicado por el archivo guardado de algún ajuste hecho anteriormente; **animation**, que indica el archivo de la animación a cargar; **interpolation** indica el algoritmo a utilizar dentro de los ya implementados, que se describirán a continuación; y finalmente **default** que indica con un 1 si es la configuración a cargar, puesto que puede haber más de una en el archivo.

### 2.5.2. Algoritmos de Interpolación Iniciales

Inicialmente el software tiene implementados dos algoritmos:

- Interpolación de Cerdá et al. [6]: Cuenta dos algoritmos `Renato1Algorithm.cpp` y `Renato2Algorithm.cpp`, que en las siguientes secciones serán referenciados como Renato 1 y Renato 2. La implementación para ambos algoritmos para aquellos puntos que están dentro de un triángulo está dado también por Ecuación 1.1. Para los puntos que están fuera de todos los triángulos, la ecuación para `Renato1Algorithm.cpp` es la misma, con la diferencia que no se utiliza el vector calculado  $\mu = (\mu_0, \mu_1, \mu_2)$ , sino su equivalente con valores absolutos, es decir  $\mu = (|\mu_0|, |\mu_1|, |\mu_2|)$ . Para `Renato2Algorithm.cpp` la ecuación para los puntos de la malla 3D fuera de la malla CANDIDE es la Ecuación 1.2.

A modo de resumen, `Renato2Algorithm.cpp` implementa el algoritmo explicado en la motivación (Sección 1.2) y `Renato1Algorithm.cpp` implementa el mismo algoritmo, con la excepción que para los puntos fuera de la malla CANDIDE la ecuación utiliza el valor absoluto de los  $\vec{\mu}$  calculados.

- Interpolación con coordenadas baricéntricas: existen dos versiones para este algoritmo, una que tiene en consideración los puntos que no se encuentran bajo la malla CANDIDE (`Barycentrics1Algorithm.cpp`) y otra que no (`Barycentrics2Algorithm`). La ecuación base para ambas es la siguiente:

$$\vec{v}_k = \lambda_1 \vec{C}_{i-1} + \lambda_2 \vec{C}_i + \lambda_3 \vec{C}_{i+1} \quad (2.22)$$

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = T^{-1} * (\vec{v}_k - n^{\vec{C}_{i-1}, \vec{C}_i, \vec{C}_{i+1}}) \quad (2.23)$$

Esta ecuación resultante por las ecuaciones Ecuación 2.22 y Ecuación 2.23 es equivalente a la Ecuación 1.1, que por asuntos de espacio su demostración no será explicada acá. `Barycentrics2Algorithm` utiliza esta fórmula para la interpolación de todos los puntos de la malla de la cabeza, mientras que para `Barycentrics1Algorithm` solo se utiliza para los vértices que se encuentran dentro de un triángulo. Para los vértices que se encuentran fuera de cualquier triángulo, `Barycentrics1Algorithm` utiliza la ecuación Ecuación 1.2 con los tres puntos más cercanos.

# Capítulo 3

## Diseño e implementación

En esta sección se explican, las mejoras implementadas en el manejo de las estructuras de datos (Sección 3.1), la métrica elegida para medir la calidad de la animación y cómo se integró con la aplicación para obtener la rugosidad de la malla (Sección 3.2), los algoritmos seleccionados para su implementación y las decisiones que fueron tomadas en comparación a las publicaciones de referencia (Sección 3.3), y finalmente asuntos de diseño y calidad de software (Sección 3.4).

### 3.1. Incorporación de estructuras de Datos

#### 3.1.1. Estructuras de datos del software de animación iniciales

Para el manejo de las mallas tanto de alta densidad como CANDIDE se utilizan dos tipos de estructuras básicas:

- Un arreglo en donde se almacenan los vértices como vectores de 3 dimensiones pertenecientes a la librería Ogre.
- Una estructura de datos de diccionario, map parte de la C++ Standar Library, en donde se asocian para cada vértice los triángulos que componen. Es decir, al acceder a un vértice se obtienen todos los triángulos que están conformados por este.

Esto significa que para obtener información un poco más compleja como las relaciones de vecindad entre triángulos o información acerca de la cercanía entre un punto y otro, dentro de una malla o entre las dos mallas, se debe hacer un recorrido completo entre todos los vértices. Así, existían partes del software implementadas con algoritmos muy simples, pero muy ineficientes computacionalmente.

El método más ejecutado en la aplicación y en la que se recorre innecesariamente la malla de la cabeza por completo es `Projection::computePoseProjection_NombreAlgoritmo`, donde `NombreAlgoritmo` representa el nombre del algoritmo implementado. Este método asigna el desplazamiento de cada punto  $P$  de la malla a uno o más elementos de control.

`Projection::computePoseProjection_NombreAlgoritmo` se ejecuta por cada pose de la malla CANDIDE y asociados a cada pose existe un set de triángulos, de la misma malla, que se ven modificados por esta. En este método, por cada triángulo modificado, se hace un recorrido completo de todos los puntos de la malla de la cabeza 3D buscando cuales son influenciados por este triángulo, es decir, su proyección esté dentro de ese triángulo o sea el triángulo más cercano a ese vértice. Las poses tienen asociados del orden de entre 20 y 80 triángulos de la malla CANDIDE y para el caso de los AUV<sup>1</sup>, hay 11 poses, por lo que el número de veces que se hace el recorrido a todos los vértices de la malla ( $O(10^4)$  vértices) del orden de entre 200 y 800 veces. Esto significa que si se tuviera una estructura que guarde la relación entre los triángulos y los vértices que afecta se ahorraría mucho tiempo y mejoraría bastante el rendimiento. Actualmente el software guarda una estructura propia de C++ que maneja la relación inversa: dado un vértice de la malla 3D se puede saber a qué triángulo de la malla CANDIDE está asociado y el tiempo de búsqueda es de  $O(1)$ . Esto se calcula en el método `Projection::computeCloserTriangle`.

Existen otros métodos que hacen recorridos completos de la malla de la cabeza. El caso más importante es `Projection::computeRegistration` es donde se hace el mapeo entre la malla CANDIDE y la malla de la cabeza 3D, y que se hace recorriendo todos los vértices de la malla de la cabeza 3D por cada vértice de la malla CANDIDE. Los métodos `getCloserVertex` y `getClosestVertexCandide` también hacen un recorrido completo a la malla de la cabeza cada vez que son llamados, en donde dado un punto en el espacio se busca el vértice de la malla más cercano.

---

<sup>1</sup>Action Unit Vector: unidad que utiliza el estándar CANDIDE para la implementación los Action Units, que son cualquier movimiento que se realiza con la activación de un músculo del rostro

Para mejorar el rendimiento de estos métodos y otros, se incorporaron las siguientes estructuras de datos:

### 3.1.2. Octree

Un Octree es un árbol lógico que subdivide un espacio tridimensional y en el que cada nodo posee ocho hijos. Lo más importante de un octree es la relación entre los nodos padres y los nodos hijos: si en una búsqueda, por ejemplo, el padre no es importante dado un determinado criterio se pueden descartar todos los nodos hijos. La estructura de un Octree se puede ver en la Figura 3.1.

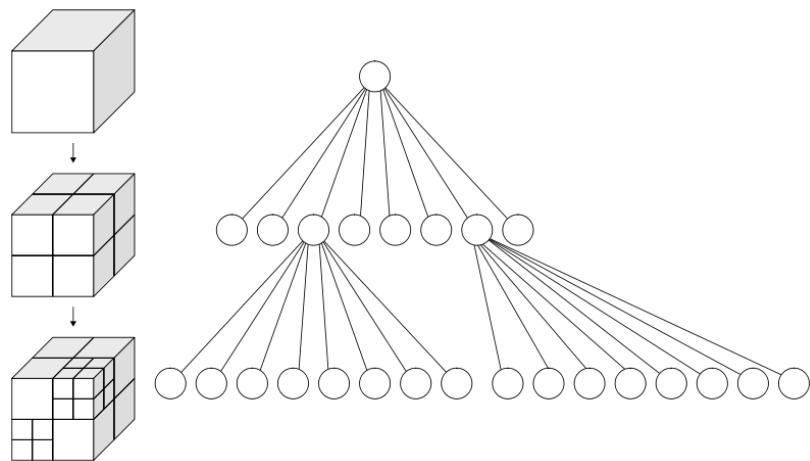


Figura 3.1: Representación de un Octree<sup>2</sup>

Utilizando un octree como estructura parte de la malla del rostro de la cabeza implica que al momento de recorrer los nodos de esta malla se espera ahorrar mucho tiempo, descartando muchos vértices de forma rápida, reduciendo tiempos de  $O(n^2)$  a  $O(n \log(n))$ .

Dada la cantidad de implementaciones que se encuentran en internet, se decidió incluir al software una ya existente. Buscando que fuera lo más general y aplicable, se seleccionó el framework de octree de Harrison Ainsworth [4]. Esta implementación está compuesta principalmente de cuatro partes: (i) una clase template `Octree.cpp`, (ii) los “items”, objetos que estarían almacenados en las hojas de este, que deben tener una composición especial que será explicada a continuación; (iii) un agente, a quien se le hacen consultas con el item por si se superpone con las celdas del octree; y (iv) visitadores, que

hacen recorridos por el octree y deciden como reaccionará.

Para los objetos que se almacenan en el octree, se creó la clase `MeshVertex.cpp` cuyas instancias definen un vértice 3D en Ogre y un índice. Es importante recordar que las estructuras anteriormente implementadas que manejan las mallas, CANDIDE o de la cabeza 3D, en la aplicación relacionan un índice a cada vértice de una malla, por lo que esta relación debía ser mantenida y almacenada para hacer consultas.

El agente fue implementado en la clase `OctreeAgentMeshVertex.cpp` y hace override del método `isOverlappingCell`, entregando como respuesta si un objeto del tipo `MeshVertex` se superpone con una celda del octree. Esta clase define el vínculo entre la clase `MeshVertex.cpp`, que almacena la información de una malla y por lo tanto con Ogre, y el octree, por lo que es obligatoria de implementar al usar el framework.

La principal etapa en la que el octree es necesaria es en la de proyección entre la malla de la cabeza 3D y la malla CANDIDE, en donde se seleccionan los vértices de la malla de la cabeza que representarán a los puntos de control y que generarán el movimiento. Para reducir el número de recorridos que se hace a la primera malla, se instancia un objeto de la clase Octree con todos los puntos de la malla densa de la cabeza 3D. La implementación encontrada tiene un constructor de la siguiente forma:

```
octree = new Octree<MeshVertex>
(vector_inicio, tamano, max_elementos, max_profundidad, min_tamano_celda);
```

En donde el `vector_inicio` indica el punto desde donde comienza el octree hasta el valor que determina el tamaño. Por ejemplo, si el `vector_inicio` es  $(0, 0, 0)$  y el tamaño del octree es 4, entonces la celda de mayor tamaño está definida desde el vector  $(0, 0, 0)$  con una diagonal hasta el vector  $(4, 4, 4)$ . El valor `max_elementos` es la cantidad máxima de elementos por hoja, por ejemplo 6; `max_profundidad` la máxima profundidad de los hijos de cada celda; y `min_tamano_celda` tal como su nombre lo indica es el tamaño mínimo de una celda, en donde `0.001f` equivale a 1mm.

El octree se crea para almacenar los vértices de la malla de la cabeza. Cuando se crea el octree, en el método `createOctree()` en la clase `Projection`, se hace un recorrido por todos estos vértices, creando

los objetos `MeshVertex` y almacenando las dimensiones de la malla completa, para tener los valores que creen el Octree del tamaño necesario para encerrarla por completo.

Para hacer consultas al octree, se creó la clase `OctreeVisitorCloserMeshVertex.cpp`, al que se le entrega un vértice 3D de Ogre y devuelve como resultado la lista de todos los vértices encontrados durante un recorrido en profundidad en que se superponían los nodos padres del árbol con el vértice.

Así para la proyección, se crea la instancia del Octree, se hace un recorrido de los puntos de la malla CANDIDE, y por cada uno se instancia un objeto de la clase `OctreeVisitorCloserMeshVertex.cpp` quien luego de visitarlo devuelve la lista de los vértices más cercanos a éste en la malla de la cabeza. Con esta lista, se hace una última búsqueda, seleccionando el más cercano al vértice CANDIDE y almacenándolo como su proyección. Usualmente la lista que entrega el visitador posee una cantidad de entre 10 y un máximo de 1000 vértices, reduciendo hasta un tercio la cantidad de vértices a visitar linealmente.

Esto implica dos cosas: (i) el número de vértices descartados es bastante grande, reduciendo finalmente el número a revisar, (ii) por la misma ventaja, se descartan puntos que posiblemente servirían para otros propósitos, cuya principal desventaja encontrada es que la cantidad de vértices cercanos no es manejable. Por ejemplo, cuando se requieran los X vértices más cercanos, no necesariamente esta cantidad es alcanzable.

La estructura de clases del Octree, incluyendo las clases agregadas para este proyecto está en el Apéndice A.

### 3.1.3. Vecindad de triángulos

Además del Octree, se usó otra estructura que permita entregar de manera rápida las relaciones de vecindad entre los triángulos, ya sea para la malla 3D de la cabeza o la CANDIDE. Esta estructura está definida en la clase `Edges.cpp` y está compuesta de 3 partes:

- Un arco, con el nombre `Edge`, que almacena dos cosas: (i) la arista de un triángulo con los dos vértices de los extremos que la definen y (ii) los dos triángulos a los que está asociada;

- una lista de instancias de la clase `Edge` que define a toda la malla a guardar;
- una estructura de diccionario en el que dado un vértice, puedo obtener todos los índices de la lista anterior, en las que el vértice está contenido en alguna de las subestructuras de aristas.

Así, para buscar por ejemplo los triángulos vecinos de un triángulo  $t$ , se extrae cada uno de sus vértices, se busca en la tabla la estructura de asociación, y en tiempo  $O(1)$  se obtiene la información. La Figura 3.2 resume más claramente la estructura.

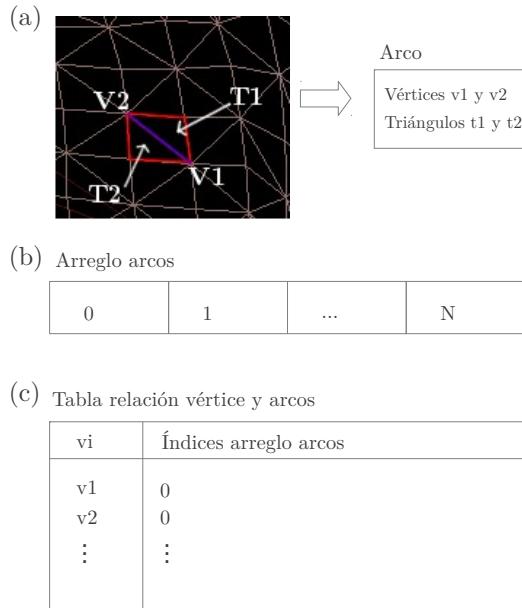


Figura 3.2: (a) A la izquierda un arco de la malla destacado en azul, con  $v_1$  y  $v_2$  los índices de los vértices que lo componen, y los dos índices a los triángulos comparten en rojo ( $t_1$  y  $t_2$ ), y a la derecha su transformación en una instancia de la estructura `Edge`; (b) arreglo en donde se almacenan todos los arcos; y (c) tabla de acceso, en donde se puede acceder al arco a través de los vértices que lo componen. En este ejemplo, el arco de la imagen superior queda almacenado en la primera posición (índice 0 del arreglo de arcos). Así, si se accede a la tabla con los índices  $v_1$  y  $v_2$ , se obtiene una lista que posee el índice 0.

La estructura fue utilizada para `Projection::computePoseProjection_NombreAlgoritmo` en donde, dado un triángulo de control, se calculan los desplazamientos de todos los vértices influenciados por este. Inicialmente el método hace un recorrido de  $O(n)$  por cada triángulo  $t$  descartando aquellos que no son influenciados por el triángulo, con  $n$  igual a la cantidad los puntos de la malla de la cabeza exceptuando los de la nuca. La idea de incluir la estructura es reducir este recorrido, aprovechando las relaciones de vecindad para encontrar a priori cuáles son los  $n'$  vértices influenciados por  $t$ , con  $n' \leq n$ . Con esto, la idea es que el recorrido sea sobre un conjunto reducido de vértices. El problema se genera en la búsqueda, que toma tiempo  $O(n' * n')$ . Este aumento se genera porque al buscar los vértices influenciados por un triángulo deben almacenar todos los vértices ya recorridos, para así no volverlos a tocar y entrar en loops, agregando cálculos extra, que pueden ser optimizados a futuro.

## 3.2. Métricas de calidad de animación

### 3.2.1. Ángulo diedro y rugosidad

Para la evaluación de la calidad de una malla existen diversas métricas, una de ellas es el ángulo diedro, que es el ángulo formado entre dos planos. Para dos planos denotados  $A$  y  $B$ , es el ángulo entre sus dos normales unitarias  $n_A$  y  $n_B$  se calcula como:

$$\cos(\phi_{AB}) = n_A \cdot n_B \quad (3.1)$$

Esta métrica puede ser usada por si sola para ver la relación y posición de los triángulos y como se comportan en la malla cuando está animada. Sin embargo se puede ir más allá, Wu et al. [20] usan el ángulo diedro para calcular la rugosidad de una superficie, denotada por  $\rho_d$ .

$$\rho_d = 1 - n_A \cdot n_B \quad (3.2)$$

Donde  $d$  se agrega para denotar la rugosidad del ángulo diedro.

Dado un triángulo  $T$  con vértices  $v_1$ ,  $v_2$  y  $v_3$ , su rugosidad es calculada como sigue:

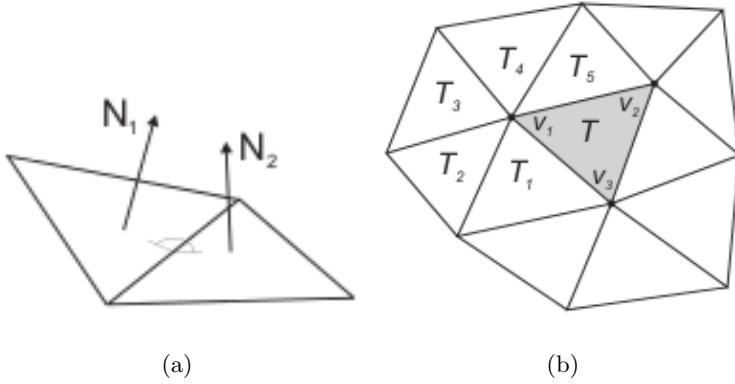


Figura 3.3: (a) Ángulo diedro. (b) La vecindad de  $T$  para calcular  $G(T)$

$$\rho(T) = \frac{G(v_1)V(v_1) + G(v_2)V(v_2) + G(v_3)V(v_3)}{V(v_1) + V(v_2) + V(v_3)} \quad (3.3)$$

Donde  $G(v_i)$  es la rugosidad promedio de los ángulos diedros entre todos los triángulos asociados al vértice  $v_i$ , y  $V(v_i)$  la varianza de estos ángulos. En el caso de la Figura 3.3, para el vértice  $v_1$ ,  $G(v_1)$  sería el promedio los ángulos diedros entre  $(T, T_1)$ ,  $(T_1, T_2)$ ,  $(T_2, T_3)$ ,  $(T_3, T_4)$ ,  $(T_4, T_5)$  y  $(T_5, T)$ ; y  $V(v_1)$  su varianza.

Una superficie rugosa puede ser considerada aquella con una alta concentración de bultos o baches de distintos tamaños. La rugosidad expresada en la Ecuación 3.3 puede medir las “protuberancias” de las superficies a nivel de caras, es decir de los triángulos, mostrando valores más altos para aquellas que presentan más.

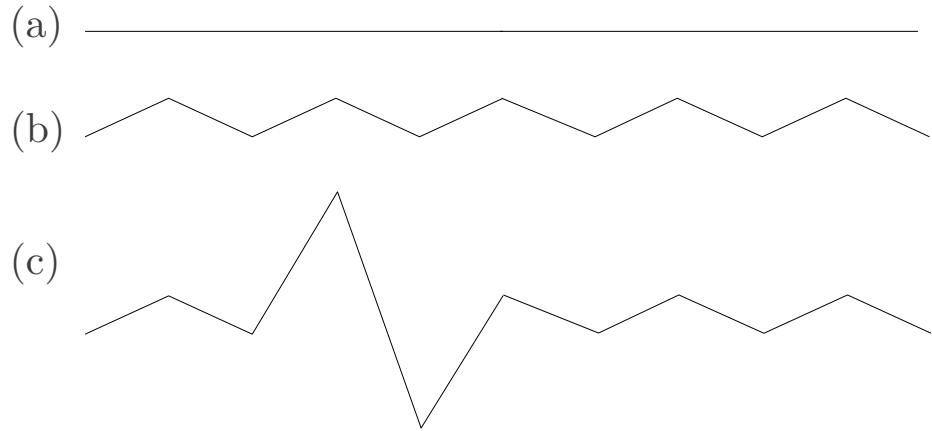


Figura 3.4: Ejemplo 2D para los valores de la rugosidad. (a) Superficie lisa, la rugosidad toma valor 0. Más abajo, superficie con (b) protuberancias distribuidas de manera homogénea y (c) con protuberancias que sobresalen, ambas con igual ángulo diedro, pero la última con mayor varianza, reflejando mayor rugosidad local

Para evaluar la calidad de la malla en la aplicación trabajada en esta memoria, el concepto de rugosidad de Wu et al. [20] fue seleccionado e implementado, como se describe en las secciones que siguen.

### 3.2.2. Aplicación de las métricas dentro de la animación

La idea de implementar una métrica de la calidad de la animación es la identificación de movimientos en los que la boca presenta más problemas, es decir, mayor cantidad de irregularidades en su superficie. Para esto se cuenta con la animación generada por Terissi et al. [18] en la que se pronuncian las siguientes sílabas:

pa pi pu ta ti tu ka ki ku ba bi bu ma mi mu fa fi fu sa si su ya yi yu da di du

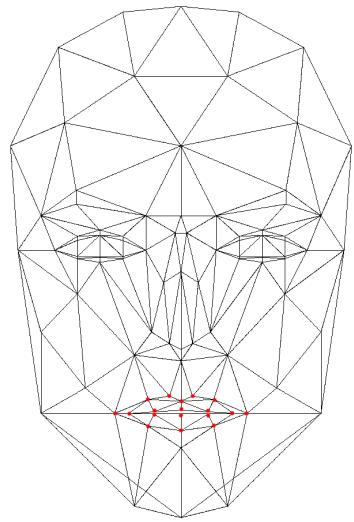
La animación es lo suficientemente general para utilizarla de referencia en este estudio pues las sílabas presentadas componen la base de todos los sonidos posibles [18].

Para esta secuencia de sonidos se tiene un archivo con la información obtenida por Lucas Terissi de una evaluación perceptiva de cada sílaba para esta secuencia para el video real, es decir, una persona pronunciando estas sílabas; y para el software de animación moviendo la malla 3D con el algoritmo propuesto por Cerdá et al. [6], explicado en la Sección 1.2. Un grupo de 20 personas fue inscrito para en los experimentos de percepción realizados para obtener la evaluación, cuyas edades estaban entre los 22 y 35 años. El grupo estaba formado por 7 mujeres y 13 hombres, informando habilidades de audición y visión normales [18]. En el experimento, cada persona evaluó que tan entendida era la pronunciación de la sílaba, asignándole un porcentaje de calidad: entre mayor es el porcentaje, más clara la sílaba fue entendida.

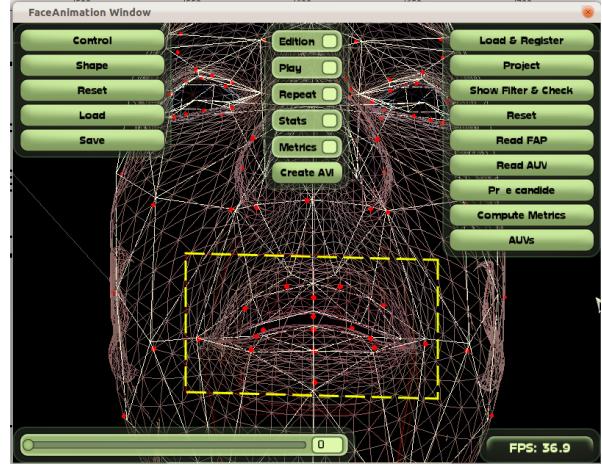
Es importante señalar que la animación es generada dado cierta cantidad de interferencia en el archivo de audio. La base de datos con baja interferencia en el archivo de audio se encuentra Apéndice C para el video real y para la animación. Con esta base de datos se puede tener una idea de cuáles sílabas poseen más problemas en la animación y luego compararla con los resultados obtenidos de la aplicación de la métrica anteriormente explicada.

Para la aplicación del concepto de rugosidad sobre la animación de la malla de la cabeza se decidió calcular la rugosidad solo alrededor de un conjunto de vértices Candide proyectados en esta. Por asuntos tanto de rapidez como de eficiencia y espacio solo fueron seleccionados los 18 vértices que definen los labios, puesto que el enfoque principal de la presente memoria es en la animación de la boca de la cabeza 3D. Estos vértices se ven resaltados en la Figura 3.5, en el lado izquierdo mostrando solo los vértices de la malla Candide y al lado derecho mostrando el área en la malla de la cabeza 3D.

Así, se creó un nuevo botón en la interfaz llamado “compute metrics”. Este botón, dada una animación cargada anteriormente, crea un archivo llamado “dataMetrics.txt” en el que se escribe por cada frame de la animación 19 filas, 18 correspondiente a los vértices de los labios y una última con el resumen total de los vértices. La columnas del archivo son las siguientes:



(a)



(b)

Figura 3.5: Área en la malla donde la métrica es aplicada. (a) En la malla Candide-3 y (b) en la malla Candide sobre la malla de la cabeza 3D, encerrada en amarillo.

- N°frame
- Tiempo del frame en segundos
- Indice vértice candide a inspeccionar
- Rugosidad mínima
- Rugosidad máxima
- Rugosidad promedio

Por ejemplo, para el frame 0 de la animación se puede obtener algo como lo siguiente:

```
0 0 6 0.00207669 0.0313111 0.0083685
0 0 30 0.00135763 0.190726 0.040728
.
.
```

```

0 0 78 0.01455 0.0237646 0.0205737
0 0 80 0.00243321 0.0745392 0.0201142
0 0 -1 0.00135763 0.190726 0.026452

```

En donde la última fila indica que no está asociada a ningún vértice Candide sino que es un resumen de todo el frame.

La razón por la que se decidió por este formato es que luego pueda ser procesado por Octave<sup>3</sup>; el archivo obtenido se procesó en este software con el que se obtiene un nuevo archivo llamado `resultados.txt` que contiene los resultados finales por cada sílaba. Cada columna del archivo tiene la sílaba, la rugosidad promedio y la cantidad de frames que formaban la sílaba. Debe recordarse que entre mayor es la rugosidad peor es la calidad de la superficie y por lo tanto peor es la calidad de la sílaba.

### 3.3. Implementación de algoritmos de interpolación ya conocidos

Para hacer el análisis de calidad y rendimiento, los algoritmos implementados son la interpolación geométrica (Sección 2.4.1), el algoritmo Surface-Oriented FFD (Sección 2.4.2) y Planar Bones (Sección 2.4.3). Los algoritmos de interpolación geométrica y Planar Bones permiten dos versiones cada uno, (i) una en la que los elementos de control fueran solo uno, el más cercano, y (ii) con más de un elemento de control, con asignación de pesos. Toda asignación de pesos, utiliza como medida para calcular la distancia entre un vértice y un triángulo de control, la del vértice del triángulo más cercano al vértice inspeccionado.

Además se implementaron dos algoritmos más en los cuales los elementos de control son los vértices CANDIDE y en los que se calcula el desplazamiento de un vértice  $P$  como la suma ponderada de los desplazamientos de los elementos de control que lo influyen, esto es, sin ningún procesamiento por alguna ecuación. Estos algoritmos llevan el nombre de `PointDistance` y serán explicados en más detalle en las secciones siguientes.

---

<sup>3</sup><http://www.gnu.org/software/octave/>

Utilizando el archivo de setup, el software permite hacer combinaciones entre los algoritmos implementados y el sistema de pesos. Este sistema es solo aplicable a aquellos algoritmos que utilizan únicamente los triángulos como elementos de control. Estos son:

- Baricéntricas 2
- Interpolación geométrica
- Planar Bones
- Renato 3

Este último algoritmo, Renato 3, fue implementado especialmente para ver la influencia de más de un elemento de control, puesto que sus algoritmos similares, Renato 1 y 2, para los vértices que están fuera de la malla CANDIDE utilizan los tres puntos más cercanos, obteniendo siempre los mismos puntos, sin generar ningún cambio.

A continuación se hace una descripción de cada uno de los algoritmos, incluidas las decisiones tomadas cuando el paper de referencia no era lo suficientemente claro.

## Interpolación Geométrica

La principal diferencia de implementación con el algoritmo descrito por Kshirsagar et al. [11] es la manera en que se selecciona el vértice de control. El algoritmo original hace un recorrido en profundidad, avanzando de un arco desde un vértice de control, hasta formar regiones de influencia por cada vértice de control. La implementación de esta memoria en cambio selecciona como el vértice más cercano aquel vértice del triángulo cuya distancia con él sea menor.

Para la versión con pesos, lo que hace es seleccionar, además del vértice más cercano del triángulo asignado como más cercano, agregar como influyentes todos aquellos triángulos que tengan una distancia menor a 10 (distancia en términos de la malla) y que estén directamente conectados, es decir, si existe una cavidad entre ellos, no se tomará en cuenta.

La clase que implementa este algoritmo es `Geometric_tAlgorithm.cpp`.

## Surface Oriented

Dado que la transformación que genera este algoritmo es equivalente a la combinación entre el algoritmo `Barycentric2Algorithm` con el sistema de pesos de la ecuación Ecuación 2.5, que por asuntos de espacio no será demostrada, la implementación de este algoritmo fue, más que el desarrollo de una nueva clase, la ejecución de la aplicación con esta combinación de parámetros.

## Planar Bones

Planar Bones está implementado en la clase `PlanarBones.cpp` y aunque no se obtubieron tan buenos resultados como los esperados, no hay decisiones diferentes de la descripción hecha por Karan [9]. La única decisión tomada fue la asignación de radio de influencia de un elemento de control. Dado que el paper indica que el radio es variable, lo que se hizo en la implementación fue, para cada triángulo, tener como área de influencia todos aquellos vértices que estuvieran a un porcentaje de la distancia de la arista más larga del triángulo. Esto bajo la hipótesis que entre más grande es un triángulo, más grande debe ser radio de influencia, es decir, debe influir a más vértices a su alrededor.

Así, por ejemplo, para un triángulo cuya arista más grande tiene una longitud de 30, su influencia estará dada por el  $x\%$ , donde  $x$  es el valor entregado en el archivo `setup.txt`.

## Point Distance

Dado que la mayoría de las referencias analizadas para esta memoria utilizaban el concepto de pesos, se propuso un nuevo algoritmo que utiliza los *vértices* de la malla CANDIDE, en vez de los *triángulos* de esta, como elementos de control. El desplazamiento de cada vértices está dado por la siguiente ecuación:

$$P' = \sum_{k=1}^n w_{ij} P'_k \quad (3.4)$$

En donde  $P'$  es la nueva ubicación de un vértice  $P$ , y  $P'_k$  es la ubicación de k-ésimo punto de la

malla CANDIDE luego del desplazamiento dada la animación. La función de pesos está dada por la ecuación Ecuación 2.5 en Sección 2.4.1.

De esta manera se deseaba probar una simplificación de todas las técnicas de interpolación en que sólo la distancia fuera la variante importante de la influencia sobre un vértice. Esta idea esta basada en que si la mayoría de los algoritmos se apoyan en agregar más elementos de control para aumentar la suavidad de una malla, esta idea probaría su efectividad, llevándola al extremo cuando se utilizan muchos o todos los vértices de control.

Con esta fórmula se implementaron dos algoritmos: uno selecciona los vértices influyentes como los tres vértices CANDIDE más cercano y el segundo toma todos los vértices CANDIDE que estén directamente conectados.

### 3.4. Incorporación de Patrones de Diseño y Calidad de Software

Los algoritmos inicialmente implementados en la aplicación eran métodos, los cuales eran llamados según el valor del parámetro *interpolation* en el archivo setup, seleccionando el correcto a través de sentencias *if*. Figura 3.6 muestra las clases principales que componían la aplicación inicialmente. Considerando que se implementarían más, se decidió hacer una refactorización para que todo nuevo algoritmo siguiera un patrón template con `InterpolationAlgorithm.cpp` como clase padre. El método principal sobreescrito es `computePoseProjection` en donde se aplica finalmente la fórmula de interpolación por cada pose de la animación. Se separan dos casos principales para todos los algoritmos: el caso base en que el punto está el área definida por la malla y los casos bordes, como los labios o el mentón. Así, para un nuevo algoritmo se debe extender la clase padre y definir dos funciones principales:

- `getInsideTriangleDesp`, para obtener el desplazamiento del vértice en caso de que el punto esté dentro de algún triángulo de la malla;
- `getOutsideTriangleDesp`, para el caso en que no esté dentro de ningún triángulo.

Además de una tercera función, `projectionMessage()`, que sirve solo para escribir en consola el algoritmo que está siendo ejecutado.

Por otra parte, se deseaba que los algoritmos a los que era posible agregarle el sistema de pesos, pudieran seguir el mismo template. Para lograr esto, el algoritmo tiene una variable de instancia booleana llamada *weighted* que indica si el algoritmo fue instanciado para usar más de un triángulo de control con el sistema de pesos. De acuerdo a esta variable se siguen dos pasos, para cada vértice se calcula el desplazamientos de su único elemento de control y se almacena como desplazamiento final; o en el caso de más de uno, se calculan todos los desplazamiento y pesos por todos los triángulos que lo influencian, y finalmente se agrega la suma ponderada entre ellos como desplazamiento final de cada vértice. La función esqueleto de clase template principal, `computePoseProjection` que se muestra a continuación en pseudocódigo.

```

begin
    comment: Cálculos de datos de la malla.
    comment: Cálculo de triángulos que son afectados por cada pose.
    for t := 1 to (triangulos_movidos_por_la_pose) step 1 do
        for v := 1 to (cant_vertices_malla_densa) step 1 do
            Ogre::Vector3 desp;
            do if v.inside(t) then
                desp = getInsideTriangleDesp(...);
            else
                desp = getOutsideTriangleDesp(...) fi od;
            comment: weighted indica si el algoritmo tiene sistema de pesos
            do if weighted then
                Ogre::Real w = getWeight(...);
                do if w! = 0 then
                    w se almacena;
                    desp se almacena;
                fi od;
            fi od;
    fi od;
fi od;

```

```

else
    Se agrega desp como el desplazamiento final de v fi od;
fi
comment: Fin for v y t
do if weighted then
    Por cada vertice v en la malla densa, se asigna un desplazamiento
    como combinacion de las influencias de los triangulos  $t_1, t_2, \dots, t_n$ 
    que lo afectan, resultando una combinacion entre los pesos
     $w_1, w_2, \dots, w_n$  y los desplazamientos desp1, desp2, ... despn
    que fueron guardados enteriormente de acuerdo al tipo de peso.
fi od;

```

Esto permitió que mucho código no fuera repetido, además de hacer mucho más fácil de agregar nuevos algoritmos puesto que solo es necesario implementar las ecuaciones para estos dos casos. Si para un algoritmo no es posible que se utilice más de un punto de control, se debe setear desde su inicialización la variable *weighted* como falsa.

Sin embargo, esto funciona solo para aquellos algoritmos cuyo elemento de control son los triángulos de la malla CANDIDE. Para aquellos en que son los vértices de esta malla los elementos de control, se debe reescribir la función `computePoseProjection` por completa de acuerdo a lo que se necesite.

Por otra parte, la extensión del software se hizo de manera en que las nuevas clases estuvieran clasificadas en carpetas cuyas funcionalidades quedaran claras. También se realizaron pequeños tests, agregados al código fuente, por cada funcionalidad o estructura lo suficientemente grande antes de integrarlo con el resto del código.

Un diagrama simplificado de todas las clases se muestra en la Figura 3.7. El diagrama UML detallado para las clases del template con `InterpolationAlgorithm` está en el Apéndice B.

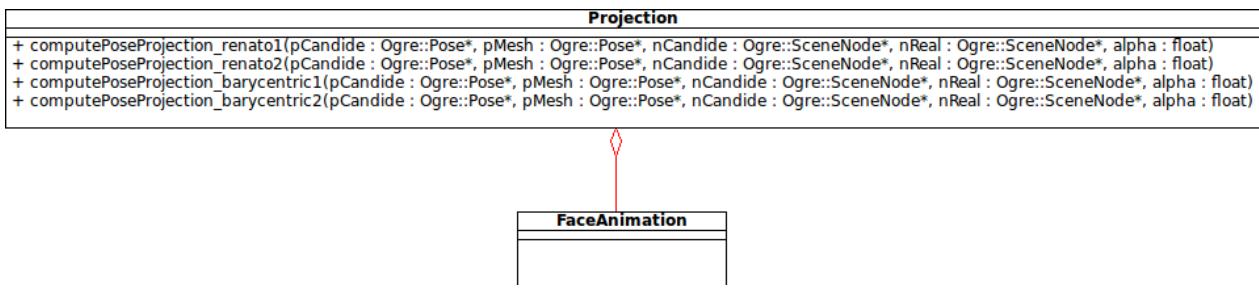


Figura 3.6: Diagrama UML simplificado de las clases de la aplicación legada.

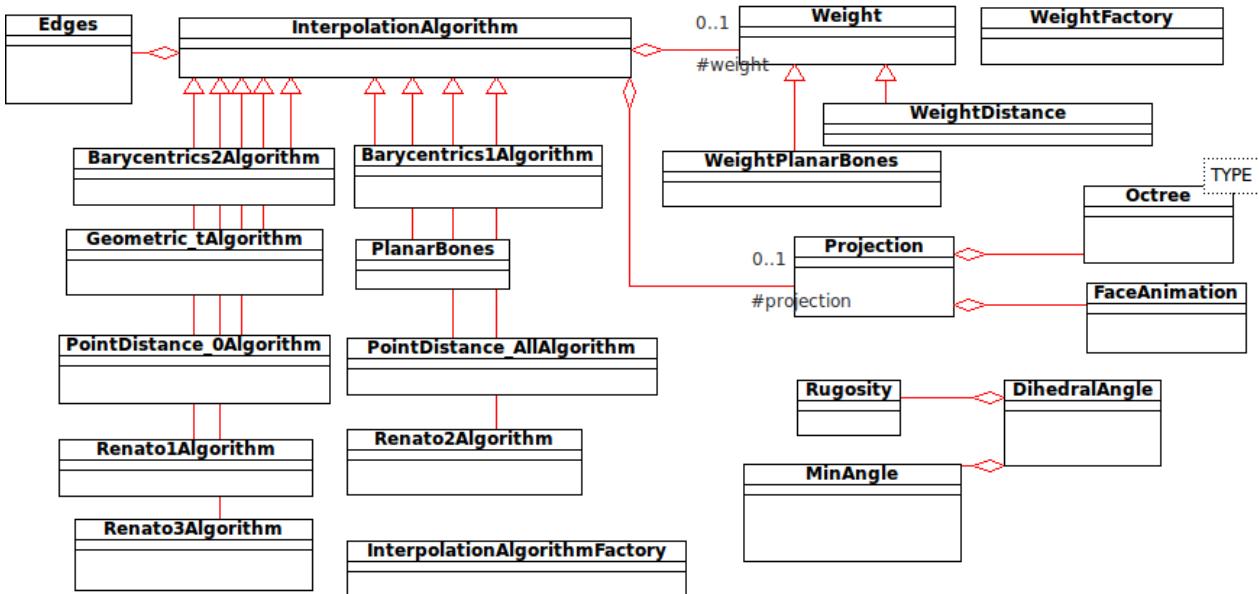


Figura 3.7: Diagrama UML simplificado de las clases de la aplicación final

# **Capítulo 4**

## **Resultados**

En esta sección se mostrarán los resultados obtenidos, en primer lugar con respecto a las estructuras de datos utilizadas, y luego con la métrica seleccionada sobre los algoritmos implementados. Con respecto a la métrica, primero se mostrará cómo se llegó a la conclusión que la métrica era representativa con la evaluación de percepción con sujetos de prueba de la calidad de la malla; segundo, con estos datos se determinaron cuáles eran los movimientos con mayores problemas; y tercero se seleccionaron los más representativos y se utilizaron para comparar los algoritmos entre sí.

### **4.1. Estructuras de Datos**

Para evaluar el impacto de las estructuras sobre la aplicación, se tomaron los tiempos de ejecución de los sectores considerados como críticos. Esto se hizo sobre de una malla que está compuesta de alrededor de 5700 vértices. Por el momento, no se cuentan con mallas faciales que se compongan de un número de vértices lo mucho mayor para hacer más pruebas. Los experimentos se realizaron con un computador Hp Envy, core i5 2.30GHz, con 4GB de memoria RAM.

Para el Octree, el sector crítico era el registro entre la dos mallas en donde se buscaba el representante en la malla densa de cada uno de los vértices CANDIDE. Los tiempos, en milisegundos, obtenidos se muestran en la tabla a continuación.

Tabla 4.1: Evaluación de tiempo para la estructura Octree

Nº Evaluación	Tiempo sin Octree (ms)	Tiempo con Octree (ms)
1	33.589	8.228
2	33.601	8.29
3	36.048	8.298
4	34.07	8.423
5	31.43	9.686
6	31.613	8.347
7	31.522	8.279
8	31.516	8.325
9	31.555	8.335
10	31.645	8.25
<b>Promedio</b>	<b>32.86</b>	<b>8.45</b>

Para hacer una evaluación de la estructura de arcos, lo que se hizo fue ejecutar el programa pero solo al esqueleto de la clase `InterpolationAlgorithm`, evitando cualquier cálculo de desplazamientos por algún algoritmo con la finalidad de obtener tiempos más limpios y que no dependieran del algoritmo de interpolación seleccionado. El método a evaluar es el que, dado un punto de control, busca todos los vértices movidos por éste. Así, se desea evaluar si el recorrido lineal es más rápido que la ejecución con este método. Los tiempos resultantes se observan en la siguiente tabla.

Tabla 4.2: Evaluación de tiempo para la estructura de arcos

Nº Evaluación	Tiempo sin Estructura de Arcos (ms)	Tiempo con Estructura de Arcos (ms)
1	883.942	11333.6
2	882.169	11027.8
3	881.32	11328
4	891.31	11057.1
5	882.373	10984.6
6	920.23	11360.9
7	888	11269.4
8	876.719	11148.5
9	881.067	11109.8
10	887.378	11049.2
<b>Promedio</b>	<b>887.45</b>	<b>11166.89</b>

Como se explicó en la Sección 3.1.3, aunque la estructura entrega mayor información de las relaciones de la malla, el método evaluado realiza muchos cálculos extra y por lo tanto hace que el programa tome más tiempo que antes. Este método puede ser optimizado en el futuro.

## 4.2. Evaluación de la métrica

Lo primero que se hizo para probar el comportamiento de la métrica, fue construir una pequeña malla constituida de triángulos a la que se sometía a pequeños o grandes cambios, obteniendo distintos valores para la rugosidad. Así, se partía con todos los triángulos a la misma altura, es decir una malla plana. Después, un triángulo era modificado para transformarse en la cima de una protuberancia, primero con una pequeña altura y luego con una altura mucho más grande. Esta malla de prueba se muestra en la Figura 4.1

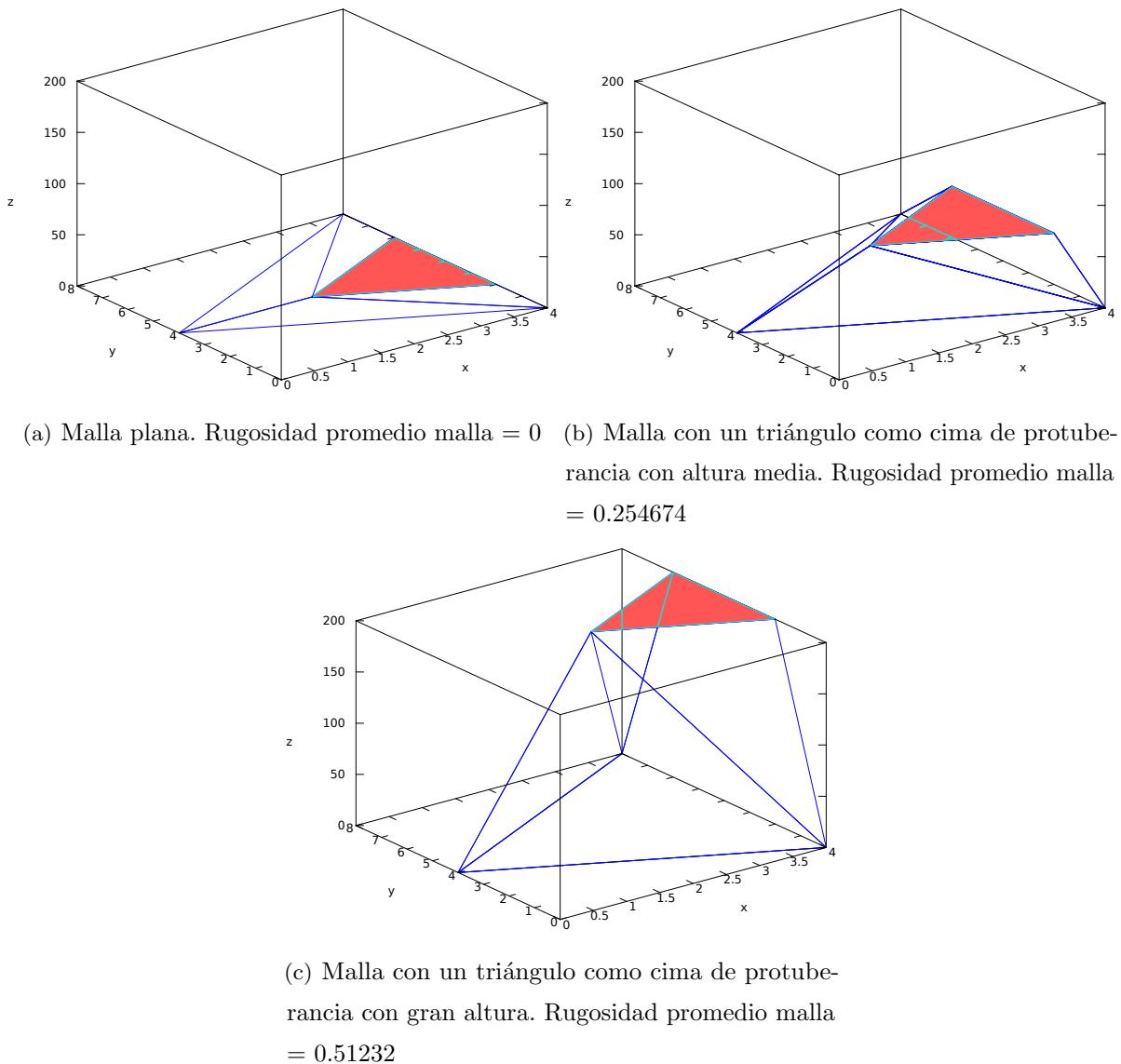


Figura 4.1: Malla pequeña de prueba para la métrica de calidad

Primero se hizo una comparación viendo los cambios del triángulo afectado. Los resultados se muestran a continuación:

Tabla 4.3: Valores de rugosidad en prueba por triángulo

Tipo de malla	Valor de la métrica por triángulo
Malla plana	0
Malla con pequeña protuberancia	0.197
Malla con gran protuberancia	0.487

Para la misma prueba, los promedios de la métrica por triángulos en la malla :

Tabla 4.4: Valores de rugosidad de malla de prueba

Tipo de malla	Promedio de la métrica en la malla
Malla plana	0
Malla con pequeña protuberancia	0.255
Malla con gran protuberancia	0.512

Al ver que la métrica se comportaba de manera esperable, se decidió hacer una prueba sobre la malla de la cabeza 3D. Tomando como base la evaluación perceptiva de la animación de las sílabas para video y para animación, se comparó con la métrica obtenida para la misma animación y algoritmo ([Renato2](#)). Una vez obtenidos todos los valores de la rugosidad promedio por frame se clasificaron por sílabas, obteniendo los resultados que se muestran en la tabla Apéndice E y que se resumen en el gráfico de la Figura 4.2.

Esta comparación está bajo la hipótesis de que si el error ya está en el video real, el sonido pronunciado es difícil y la animación no podrá mejorarlo. Una sílaba está definida como “difícil” como aquella cuya clasificación es inferior al 70 %.

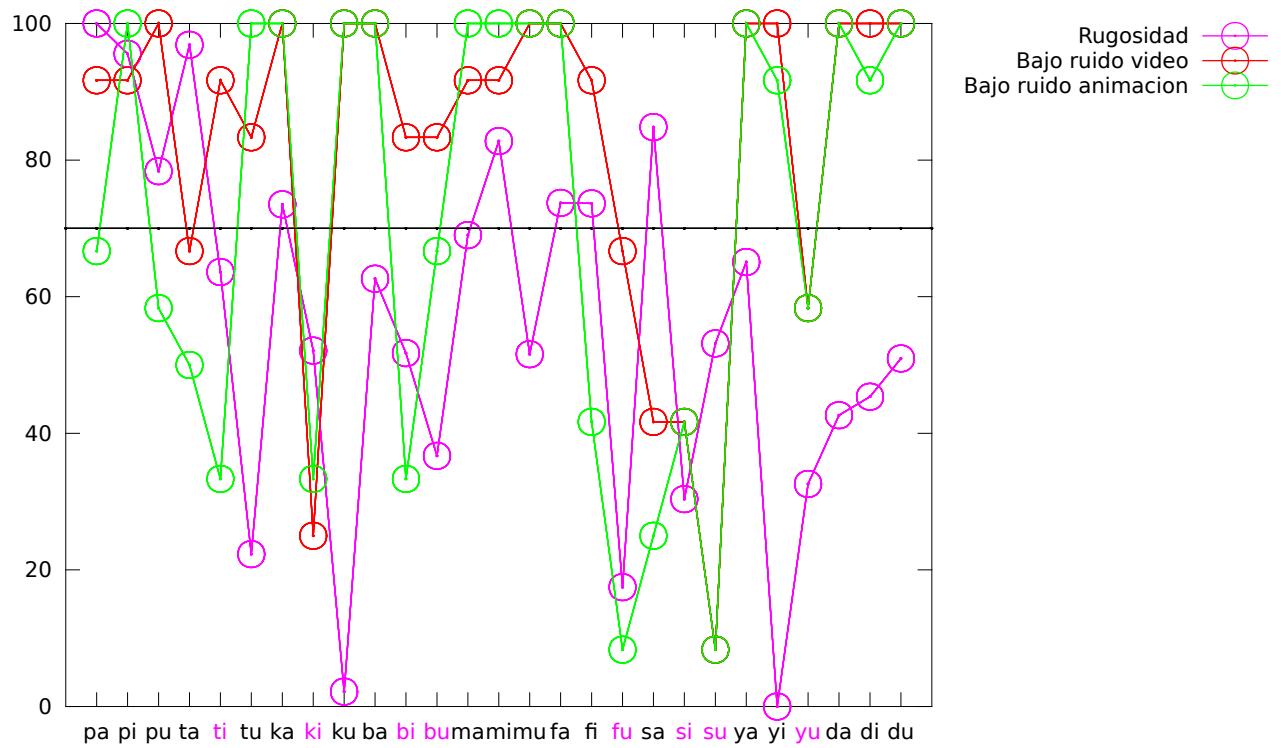


Figura 4.2: Cada punto en el eje  $x$  es una sílaba. En morado, la rugosidad promedio por sílaba; en verde, la evaluación con bajo ruido para la animación; y en rojo, la evaluación con bajo ruido para el video real. La línea en negro muestra el 70 % de la evaluación. En el eje x se muestran en color morado aquellas sílabas de la animación que fueron predichas por la métrica.

Con el gráfico y este criterio se puede ver que, como era de esperarse, el número de sílabas difíciles es mayor en la animación que en el video real. Así, hay sílabas que “aparecen” como difíciles: “pa”, “pu”, “ti”, “bi”, “bu” y “fi”, de las cuales la métrica predice: “ti”, “bi”, “bu” y “fi”, lo que equivale a un 67 %.

Con respecto a los totales, del total de sílabas con problemas en la animación, 13, la rugosidad predice 8, lo que equivale a cerca de un 61 %. Del total de sílabas con problemas en el video real, 7, la rugosidad predice 5, cerca del 71 %.

Con esto, la métrica muestra un alto nivel de predicción de sonidos difíciles por lo que cumple su objetivo y por lo tanto se define como válida para la comparación de los algoritmos.

### 4.3. Evaluación de antiguos y nuevos algoritmos

Una vez aceptada la métrica se hará un análisis del desempeño de los algoritmos respecto a la calidad de la malla de la cabeza cuando se ejecutan. Se hizo una selección de 18 algoritmos o variaciones de algoritmos que se muestran en la tabla mostrada a continuación. Cada algoritmo fue etiquetado con una letra desde la “A” a la “Q”, además de asignarle un número de grupo al que pertenece. Los algoritmos están agrupados según el algoritmo básico del que se deriva la variación. Por ejemplo, todas las versiones del algoritmo “Renato” ( Renato1, Renato2 y Renato3) pertenecen al mismo grupo.

Tabla 4.5: Algoritmos o variaciones de algoritmos seleccionados para evaluar

Grupo n°	Letra asignada	Algoritmo o variación de algoritmo
1	A	Baricéntricas 1
1	B	Baricéntricas 2
1	C	Baricéntricas 2 con peso y radio = 10 (skinning)
1	D	Baricéntricas 2 con peso y radio = 30 (skinning)
2	E	Featured based
2	F	Featured based con peso y radio = 10
3	G	Point Distance con 3 puntos más cercanos
3	H	Point Distance con todos los puntos + continuidad
4	I	Planar Bones solo, y con peso Planar Bones y radio = 10 %
<i>Continúa en la siguiente página</i>		

Continúa de la página anterior		
Grupo n°	Letra asignada	Algoritmo o variación de algoritmo
4	J	Planar Bones con peso Planar Bones y radio = 30 %, 50 % y 70 % de la arista más larga
4	K	Planar Bones con peso Planar Bones y radio = 100 % de la arista más larga
4	L	Planar Bones con peso Distance y radio = 10
4	M	Planar Bones con peso Distance y radio = 30
4	N	Renato 1
5	O	Renato 2
5	P	Renato 3
5	Q	Renato 3 con peso y radio = 10
5	R	Renato 3 con peso y radio = 30

El gráfico de Figura 4.3 representa el promedio de las rugosidades de cada sílaba de la animación para cada algoritmo. El gráfico de Figura 4.4 es el equivalente con los valores de la varianza. Los datos de estos gráficos se encuentran en Apéndice E. En estos gráficos cada barra representa un algoritmo, etiquetados según la tabla anterior. La línea en negro muestra el promedio.

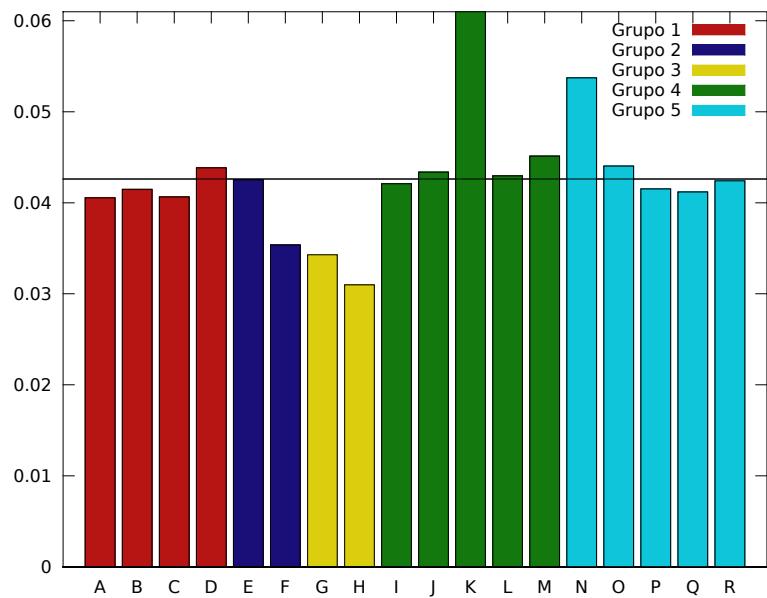


Figura 4.3: Promedio de rugosidad de las sílabas por cada algoritmo. La linea en negro muestra el promedio.

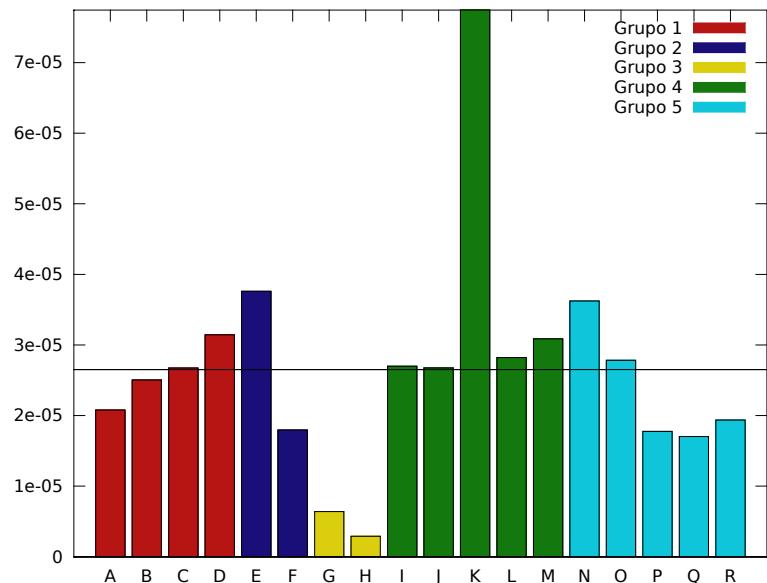


Figura 4.4: Varianza de rugosidad de las sílabas por cada algoritmo. La linea en negro muestra el promedio.

#### 4.3.1. Comparación entre los algoritmos por grupos

En las siguientes secciones se hará una evaluación del comportamiento de los algoritmos por grupo, describiendo las variaciones de estos y cómo afecta el tener más de un elemento de control como derivador de movimiento. Además, se seleccionarán las sílabas “ya” y “yu” como representativas puesto que en ambas sílabas se observan dos comportamientos importantes: (i) la apertura de la boca en amplitud y (ii) la protrusión de los labios. Esto se ve reflejado en el gráfico de Figura 4.2, en donde la mayoría de las que posee una mala evaluación dado el choque entre los triángulos de los labios. Así, para estas sílabas se agregará una imagen para aquellos algoritmos en que los cambios son importantes. Lo más importante de esta sección es evaluar qué tanto afecta a cada algoritmo los conceptos de localidad y, por lo tanto, la cantidad de elementos de control de cada uno.

##### Grupo n°1 (A-D): Algoritmo Baricéntricas

Para este grupo se observa primero que nada que para `Barycentrics2` el agregar más elementos de control significa una reducción en la rugosidad promedio, ya que comparando las barras para B y C esta última presenta un menor valor. Además, un aumento en el radio de acción de los elementos de control no aumenta la calidad de la malla, por el contrario disminuye según la métrica. Sin embargo, para ambos casos utilizar una nueva ecuación con los tres puntos más cercanos tiene una mejor calidad.

Un comportamiento similar se puede ver para la variancia. Quien presenta menor es `barycentris1`, cambios menos violentos que los otros dos. Pero en cambio para `barycentris2`, se observa que la varianza aumenta cuando se agregan más elementos de control.

##### Grupo n°2 (E-F): Interpolación Geométrica

Este algoritmo es un buen ejemplo de cómo la malla se suaviza cuando se le agregan más elementos de control. En ambos gráficos, de varianza y promedio, se pueden observar como la rugosidad disminuye considerablemente. Relacionando los números con la animación en sí, en la Figura 4.5 se observa como en el labio inferior el la línea que forma se suaviza, siendo esta menos cuadrada. Lo

mismo se ve con Figura 4.6, puesto que los triángulos en el labio inferior no parecen “chocar” entre ellos cuando se aplican pesos, obteniendo una composición mucho más homogénea.

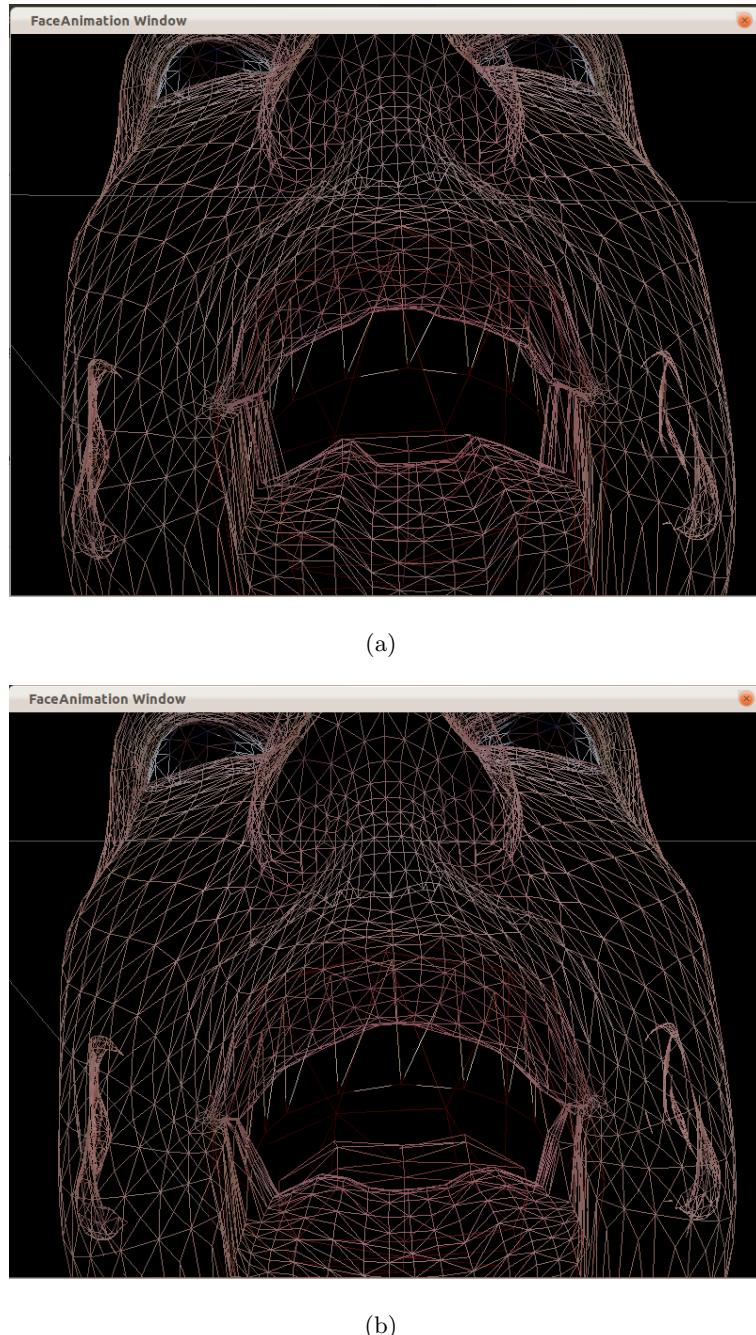
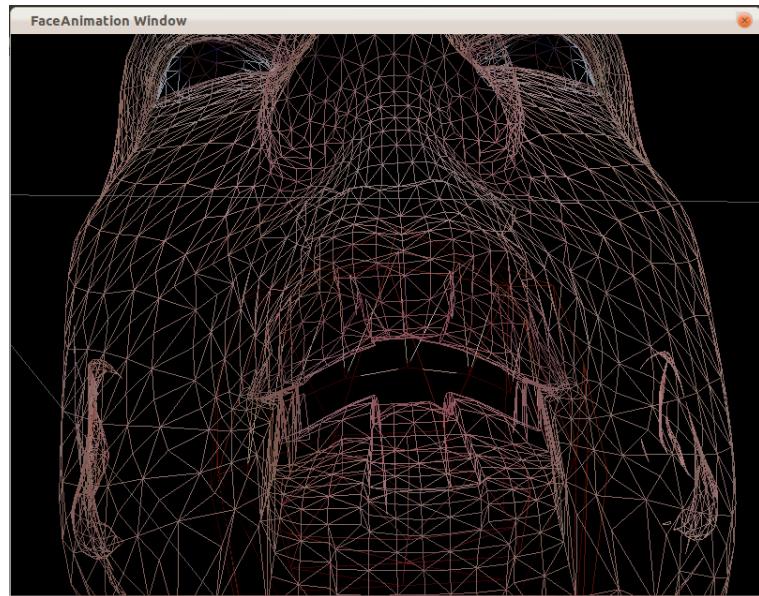
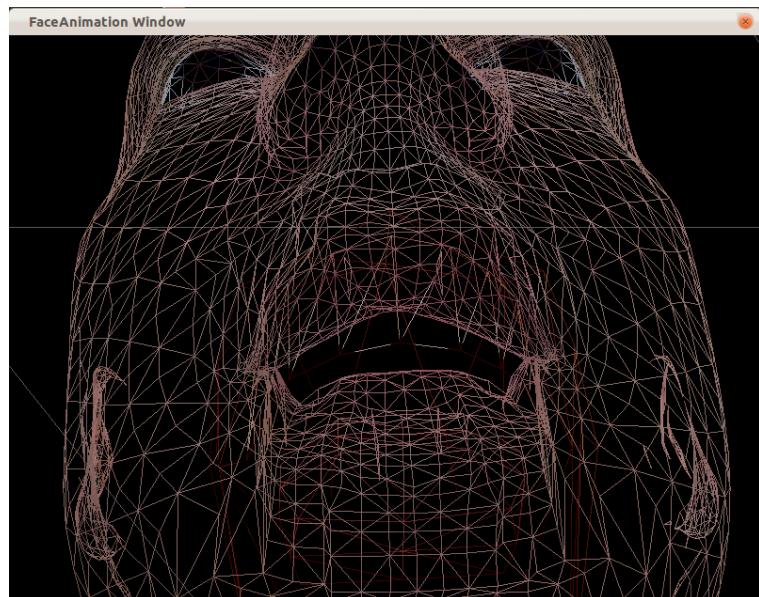


Figura 4.5: Malla de la cabeza pronunciando la sílaba “ya” utilizando algoritmo geométrico (a) sin aplicar pesos y (b) aplicando peso.



(a)



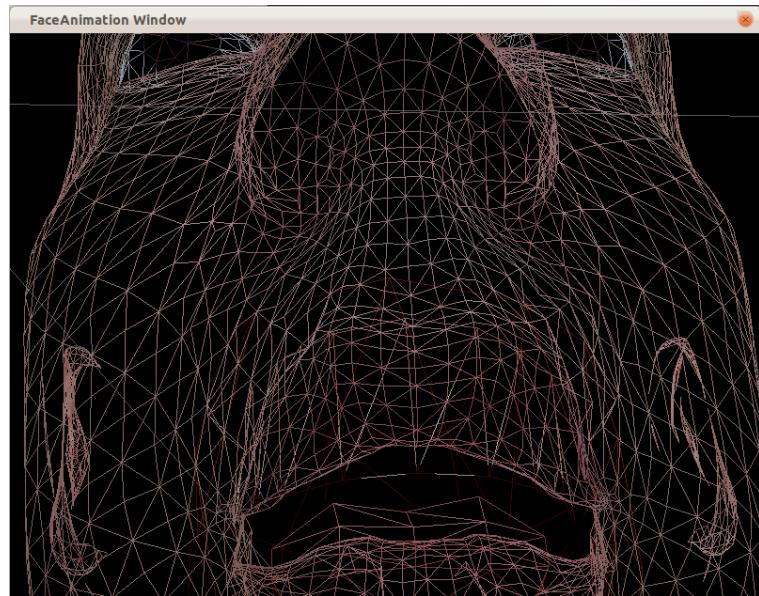
(b)

Figura 4.6: Malla de la cabeza pronunciando la sílaba “yu” utilizando algoritmo geométrico (a) sin aplicar pesos y (b) aplicando peso.

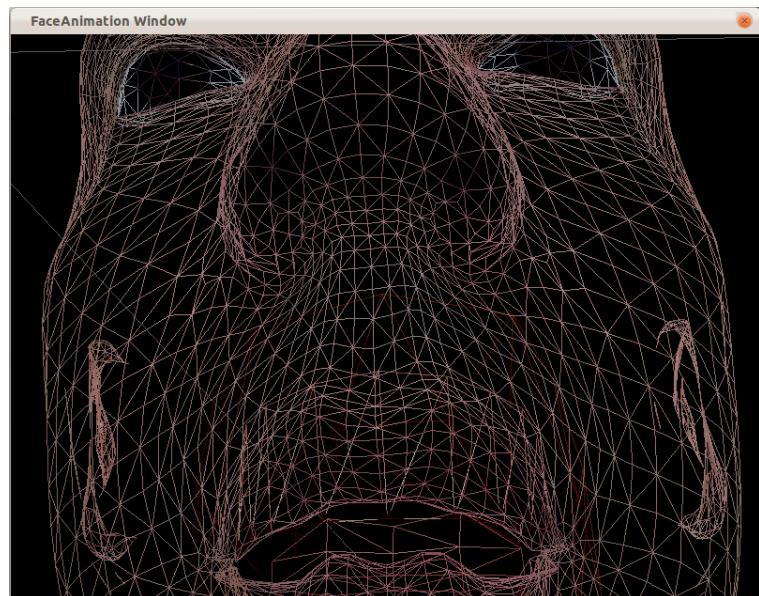
### **Grupo n°3 (G-H): Point Distance**

Este algoritmo presenta el menor promedio de rugosidad y la menor varianza dentro de todos los algoritmos. Esto presenta una mejoría en áreas como el labio superior, cuando por ejemplo pronuncia “yu”, en donde se ve una superficie mucho más suave y regular. Esto se ve mucho más pronunciado cuando se utilizan todos los vértices de control como influenciadores de cada vértices, como se observa en Figura 4.7.

Cuando este algoritmo utiliza todos los vértices de control tiene, no obstante, dos problemas grandes. Uno de ellos es el labio inferior en donde se ven dos “bultos” que, aunque tienen superficies “redondas”, forman una superficie anormal en los labios (ver 4.7(b)). Éstas son parte de la malla misma y el algoritmo lo que hace es resaltarlas innecesariamente. El segundo problema es que al tener más elementos de control influenciando a cada vértice, las aperturas de la boca se ven reducidas, disminuyendo no solo irregularidades de la superficie de los labios, sino además “aplastando” el movimiento propio de la boca. Esto se ve en la Figura 4.8, en donde es importante destacar que ambas imágenes fueron sacadas en el mismo frame de la animación, por lo que se muestran aperturas en los mismos instantes. Esto además se ve reflejado en el gráfico de varianza, en donde es muy baja en comparación con los otros algoritmos.

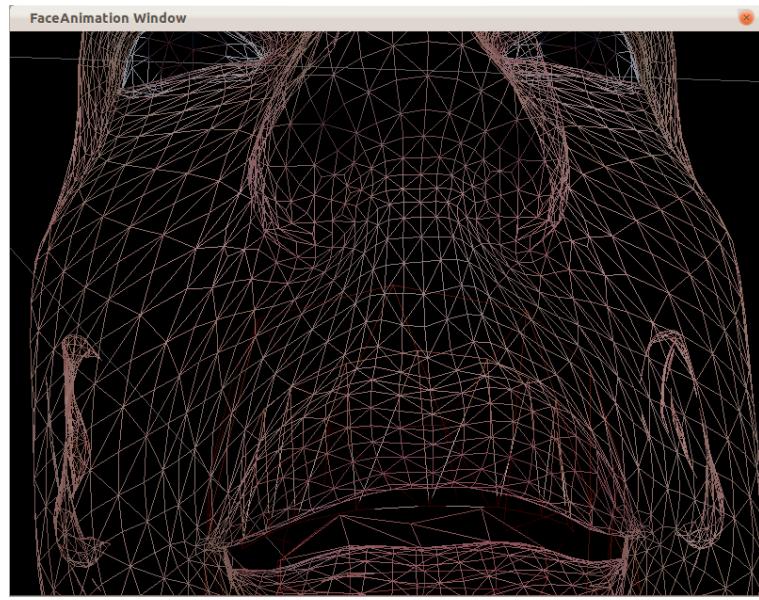


(a)

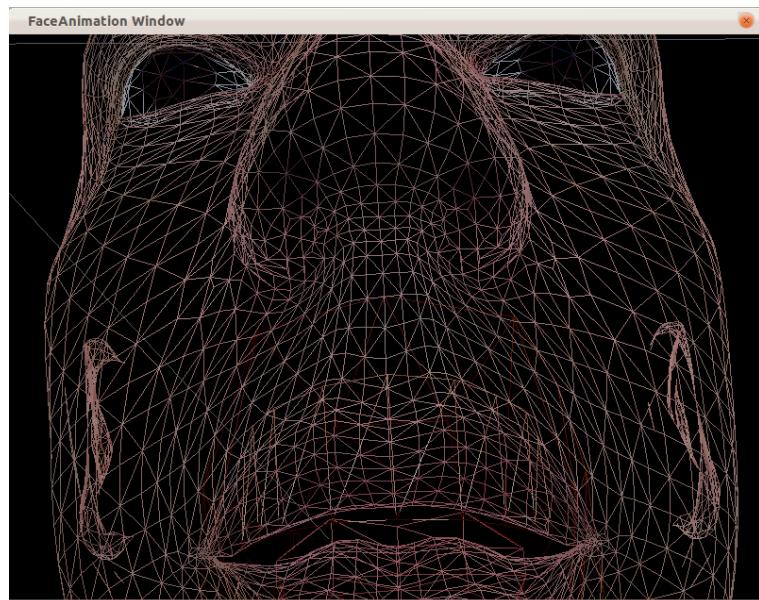


(b)

Figura 4.7: Malla de la cabeza pronunciando la sílaba “yu” utilizando point distance (a) sin aplicar pesos y (b) aplicando peso.



(a)



(b)

Figura 4.8: Apertura de la boca de la malla de la cabeza del pronunciando la sílaba “fa” utilizando point distance (a) con los 3 puntos más cercanos y (b) todos los vértices de control.

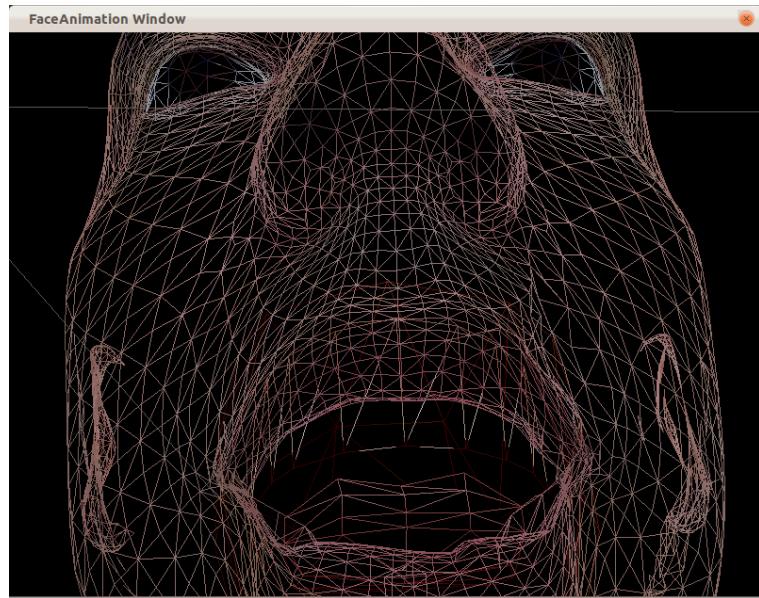
## **Grupo nº4 (I-M): Planar Bones**

La implementación de este algoritmo es el que presenta mayores dudas puesto que dado que Planar Bones está planteado como una mejora de Surface Oriented FFD se esperaba, al menos, un mejor rendimiento que Surface Oriented FFD. Sin embargo, en los gráficos de la Figura 4.3 y Figura 4.4 es el que tiene los peores valores de la métrica de rugosidad.

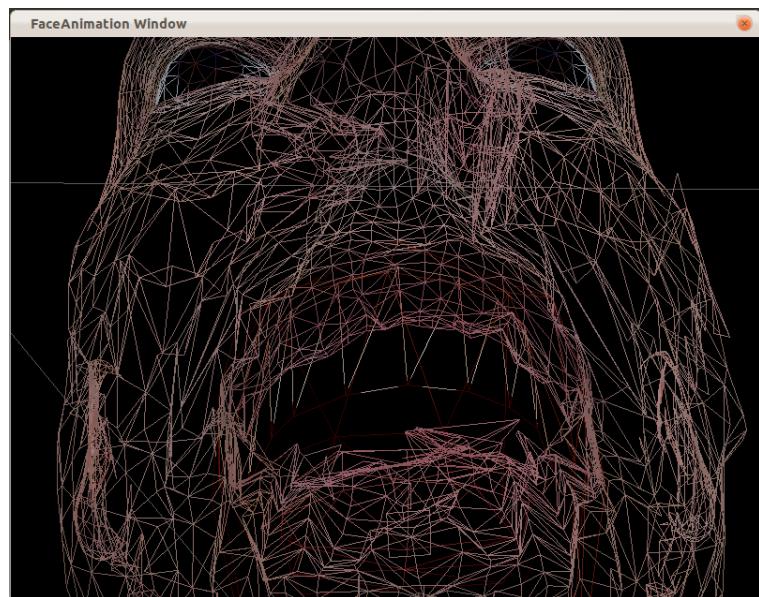
El algoritmo, al ser ejecutado presenta alrededor de la boca muchas irregularidades, como se muestra en (a) en Figura 4.9 y Figura 4.10. Con la hipótesis de que el algoritmo necesitaba más de un elemento de control para obtener esos resultados se ejecutó con el sistema de pesos que muestra el mismo paper.

Este sistema de pesos, primero que nada no disminuyó la métrica, puesto que por el contrario la rugosidad promedio aumentó; y segundo presentó problemas cuando el radio de acción era demasiado grande. Como resultado, con cualquier algoritmo al que se le aplicara el sistema de pesos de la Ecuación 2.16 resultaba con una interpolación con el resultado como en (b) en Figura 4.9 y Figura 4.10. Esto además se ve claramente con el promedio de la rugosidad, puesto que esta combinación presenta los mayores valores de rugosidad, disparándose en comparación a los otros algoritmos.

Con la idea que era el sistema de pesos era el principal responsable de esto, también se probó con el sistema de pesos de la Ecuación 2.5, llamada “Distance”. Sin embargo, los mismos resultados fueron obtenidos y nuevamente en vez de disminuir la métrica esta aumentó. Visualmente tampoco se muestran superficies más regulares y homogéneas, ya que se obtuvo todo lo contrario.

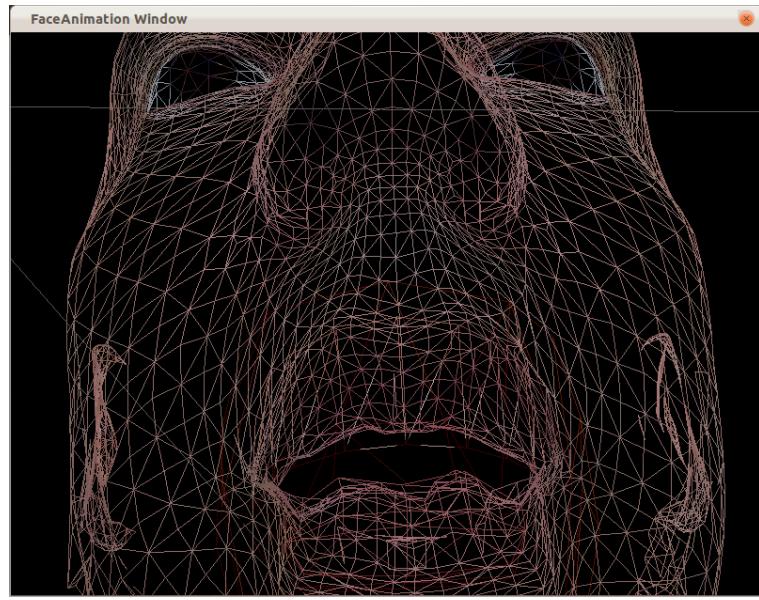


(a)

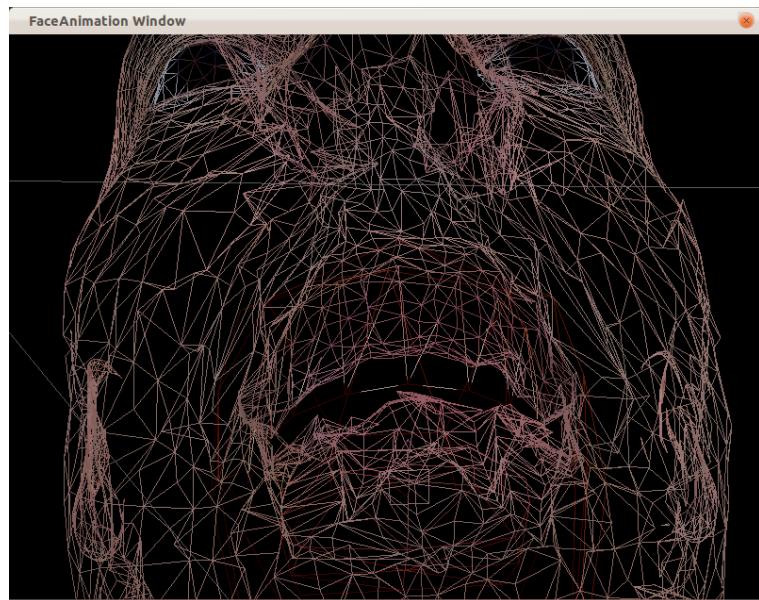


(b)

Figura 4.9: Malla de la cabeza pronunciando la sílaba “ya” utilizando Planar Bones (a) sin aplicar pesos y (b) aplicando peso.



(a)



(b)

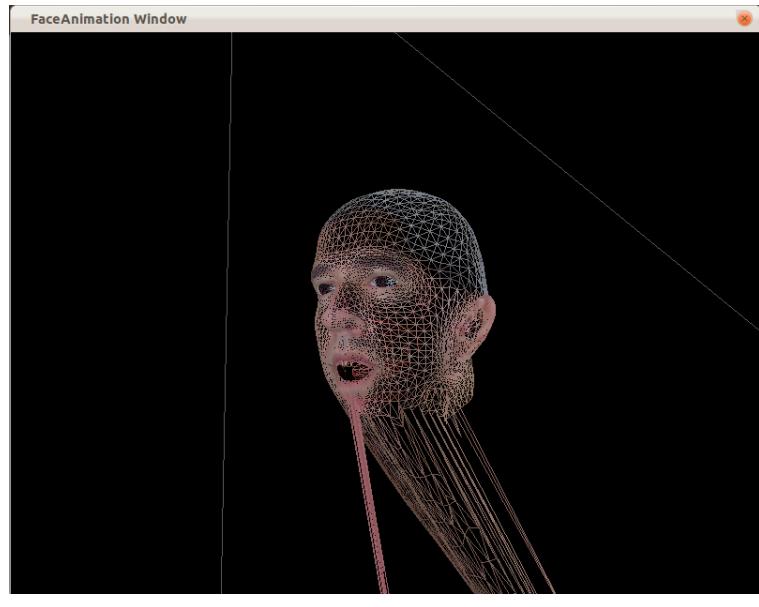
Figura 4.10: Malla de la cabeza pronunciando la sílaba “yu” utilizando Planar Bones (a) sin aplicar pesos y (b) aplicando peso.

## **Grupo n°5 (N-R): Renato**

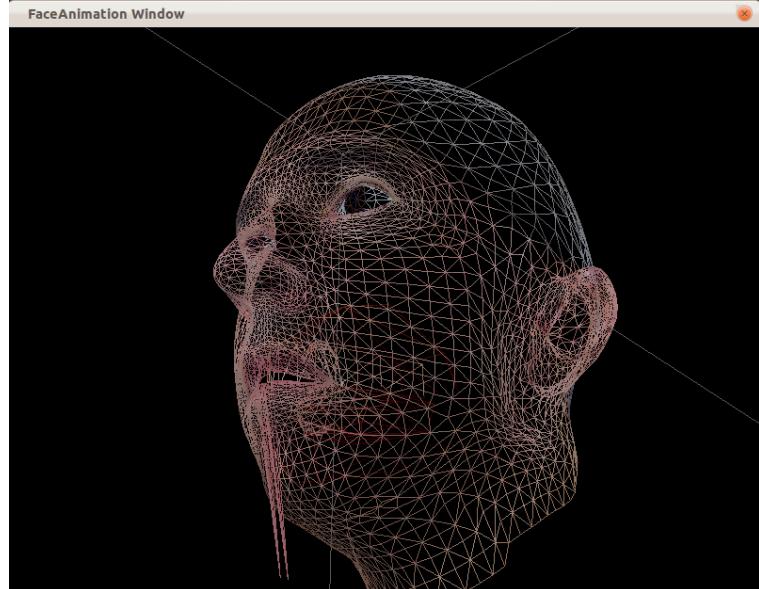
Para el primer algoritmo de este grupo, `Renato1`, se observa que esta versión del algoritmo planteado en la Sección 1.2 produce muy malos resultados. En este algoritmo se observan los peores valores de rugosidad del grupo, y es el segundo peor de los 18 algoritmos seleccionados. Aunque esto se ve con mayor claridad en el área del mentón, es importante recordar que la métrica es calculada solo para el conjunto de vértices alrededor de los labios. Visualmente esto se ve en la Figura 4.11, en donde hay muchos vértices cuyos valores de disparan en comparación a aquellos en los alrededores. En el acercamiento de la 4.11(b), se ve que incluso con una pequeña apertura de la boca de la malla facial hay vértices que presentan comportamientos que escapan excesivamente de los normales dentro de los labios.

Para el algoritmo que se deseaba analizar como parte de los objetivos, `Renato2` etiquetado como “O” en la Figura 4.3, se observa que en general presenta buen comportamiento con respecto al resto, presentando un valor de la métrica de rugosidad cercana al promedio. Sin embargo, si comparamos la calidad de la malla en la animación para los algoritmos `Renato2` y `Renato3`, este último tiene una rugosidad promedio menor, por lo que en este caso agregar una nueva ecuación para los vértices con discontinuidades no muestra mejoría en la animación, por el contrario, la empeora. En efecto, la varianza para `Renato2` es mucho mayor que para cualquiera de las 3 versiones de `Renato3`.

Con respecto al número de elementos de control, al agregar más elementos de control a `Renato3` la rugosidad disminuye un poco, aunque no es una gran mejoría en la calidad de la malla en términos de la métrica. Esto muestra nuevamente que el radio de acción debe estar controlado puesto que cuando aumenta demasiado comienza a resultar en mayores valores en la rugosidad.

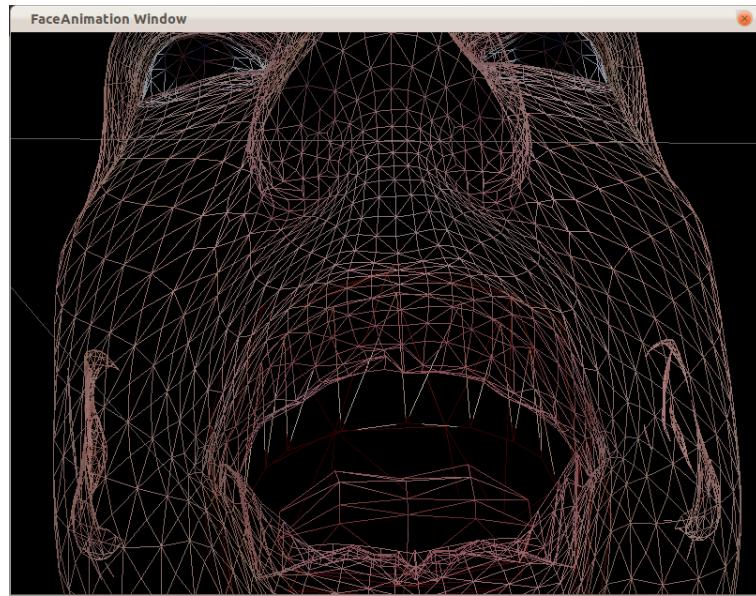


(a)

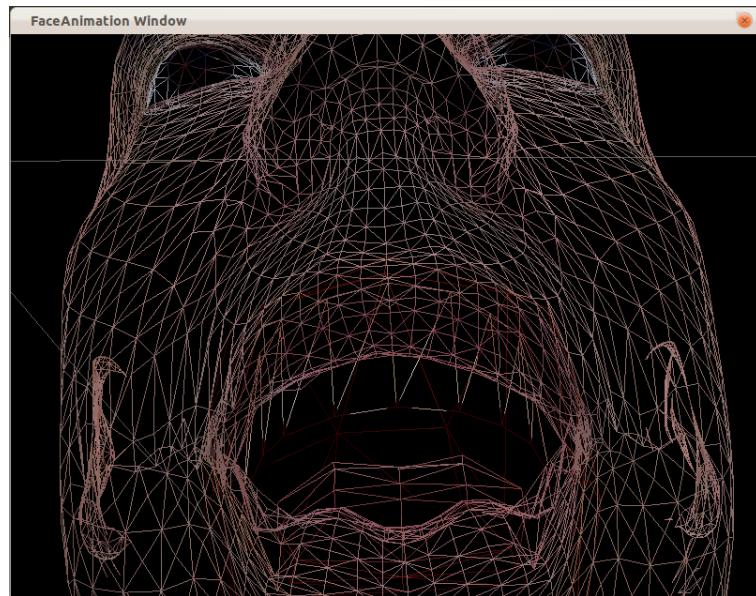


(b)

Figura 4.11: Malla ejecutada con el algoritmo Renato1 (a) para la cabeza completa, en donde se observa que en el mentón y parte de los labios existen vértices cuyos desplazamientos se disparan. (b) Acercamiento del área de los labios con una apertura pequeña de la boca, en donde también se observan comportamientos extraños.

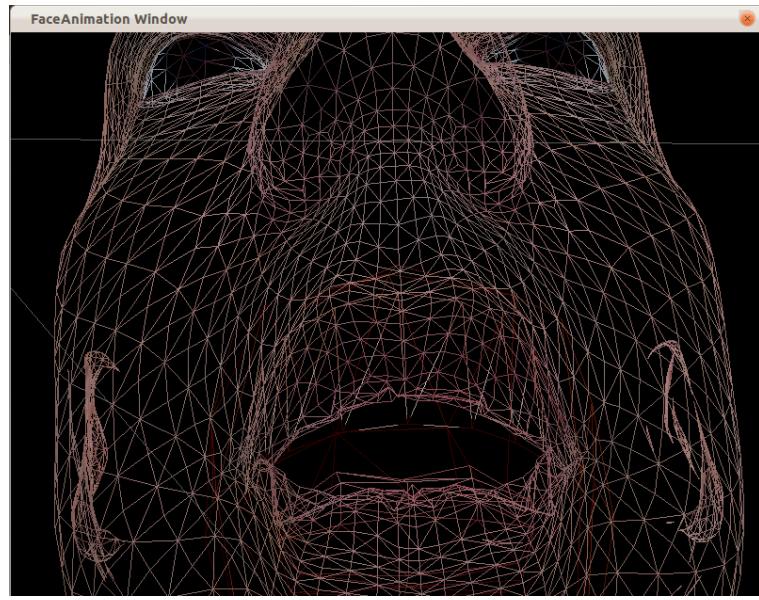


(a)

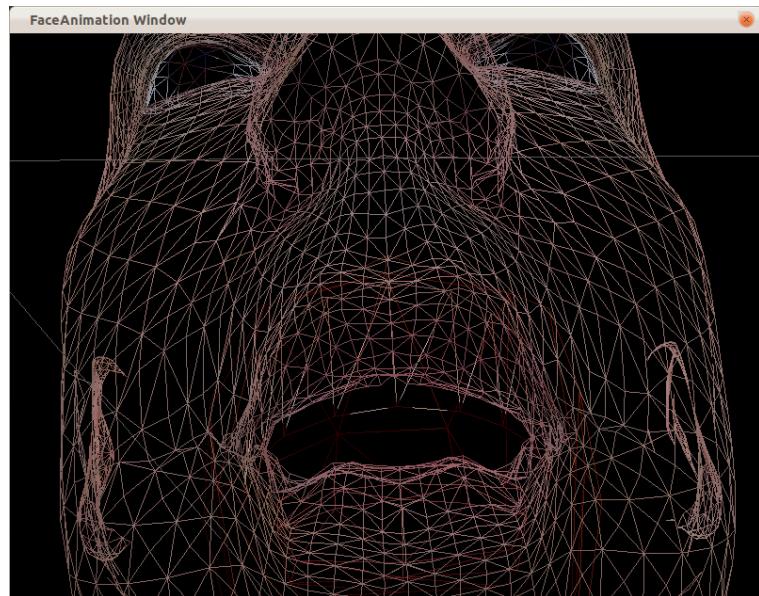


(b)

Figura 4.12: Malla de la cabeza pronunciando la sílaba “ya” (a) utilizando fórmula con tres puntos más cercanos (`Renato2`) y (b) utilizando la misma fórmula con los vértices de un triángulo más pesos con radio de 10 (`Renato3` con pesos).



(a)



(b)

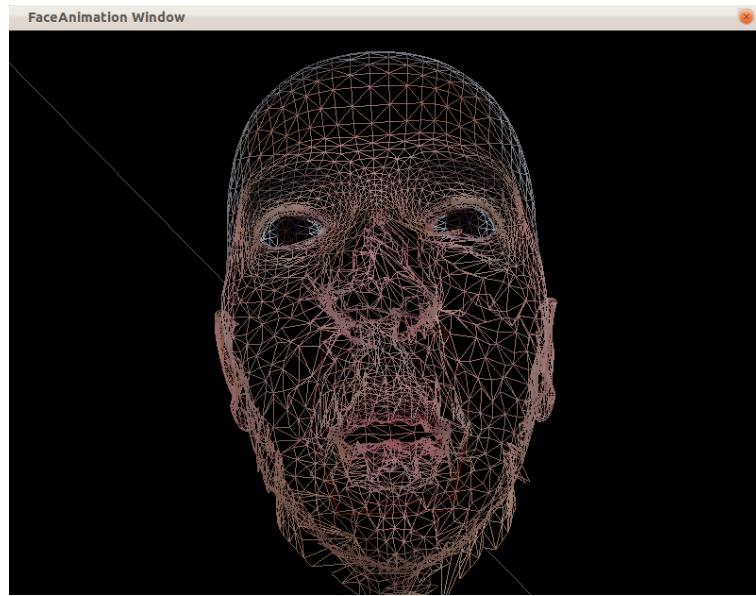
Figura 4.13: Malla de la cabeza pronunciando la sílaba “yu” (a) utilizando fórmula con tres puntos más cercanos (`Renato2`) y (b) utilizando la misma fórmula con los vértices de un triángulo más pesos con radio de 10 (`Renato3` con pesos).

#### 4.3.2. Sistemas de Pesos

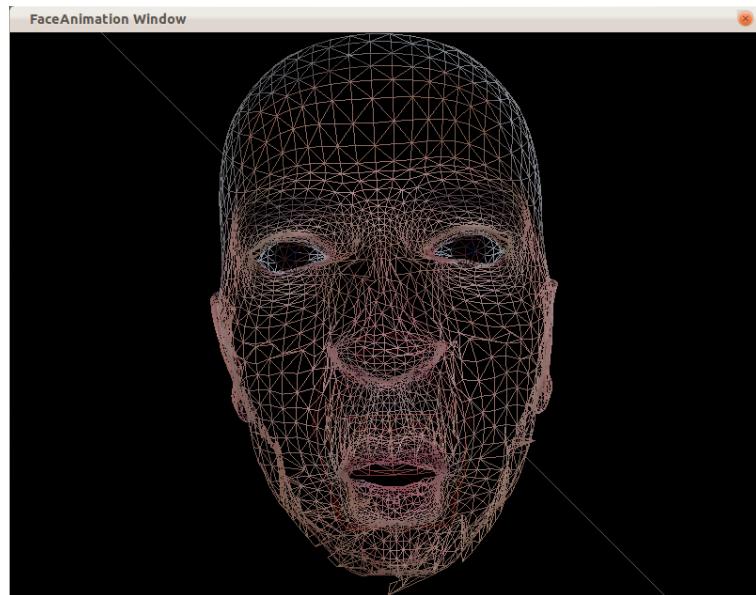
Como se ve en las secciones anteriores, además de los gráficos Figura 4.3 y Figura 4.4, se puede observar que cuando se decide utilizar más de un elemento de control la elección del radio de acción es muy importante puesto que puede afectar vértices que no corresponden. Por ejemplo, aunque el movimiento de los labios sí afecta a sectores como la nariz, estos sectores son acotados y no deberían mover la nariz completamente, como se ve la Figura 4.14.

Otro factor muy importante, y que es transversal a todo el trabajo realizado, es la fuente de la animación. Puesto que la animación de sílabas con las que se trabaja durante todo el desarrollo de los resultados solo tiene movimientos que afectan a la boca, cuando se mueven vértices con solo el desplazamientos de vértices de control de esta área, es normal que aumente la rugosidad, puesto que se mueven vértices que no deberían. Así, cuando el radio es pequeño y se tiene un área de influencia cerca de la boca, se genera un “alisamiento” de la superficie y por lo tanto el agregar más elementos de control es efectivo.

De los dos sistemas de pesos, Distance parece tener mejor comportamiento que Planar Bones, puesto que incluso cuando se aumenta el radio de acción, con el primero nunca se obtuvieron superficies tan irregulares que con Planar Bones. Esto se puede apreciar en Figura 4.14 donde se ven ambos sistemas con altos valores de radio de acción. Además, en los gráficos de Figura 4.3 y Figura 4.4 las combinaciones con el sistema de pesos Distance son las únicas que muestran alguna mejoría del algoritmo original, en cambio todas aquellas variaciones con Planar bones siempre aumentan el valor de la rugosidad de la malla.



(a) Cabeza 3D completa mientras pronuncia “yu” con peso de Planar Bones con radio de acción del largo de la arista más larga de cada elemento de control



(b) Cabeza 3D completa mientras pronuncia “yu” con peso de Distance con radio = 40

Figura 4.14: Comparación de sistemas de pesos con radio de alto valor.

## Capítulo 5

# Conclusiones y trabajo futuro

El trabajo desarrollado y presentado en esta memoria, entrega una continuación del trabajo presentado por la memoria de Valenzuela [19]. El principal objetivo la presente memoria era hacer una evaluación y mejora en la calidad y realismo de la animación de rostros desarrollada anteriormente. Para esto, las tres tareas más importantes realizadas fueron: (i) la implementación de una métrica que permite tener rápidamente una evaluación de la calidad de la malla al momento de animar, (ii) la implementación de otros algoritmos de interpolación para animación facial, con el objetivo de poder evaluar el de la aplicación legada y (iii) la inclusión de un sistema que permitiera variaciones a algunos de los algoritmos implementados. Para la comparación entre algoritmos la métrica se hace fundamental en el ahorro de tiempo, ya que anteriormente toda evaluación de la malla se hacía en experimentos de laboratorio con personas que evaluaran perceptivamente la animación.

Como parte de los resultados obtenidos, se observa que la métrica de calidad presenta muy buen comportamiento, mostrando valores mayores cuando la superficie de la malla a la que es aplicada presenta mayores desigualdades, disminuyendo cuando estas se hacen menores y teniendo valor cero cuando la malla es plana. Además, la métrica presenta un alto porcentaje de relación con la evaluación perceptiva que se hizo a la animación que contiene la base de todos los sonidos posibles presentada por Terissi et al. [18]. Así, la métrica cumple con el objetivo para el cual fue seleccionada, permitiendo a futuro ser utilizada como evaluación preliminar de la animación o incluso como evaluación definitiva.

Con la animación presentada, se pudo encontrar que los movimientos con mayores dificultades eran aquellos en los que la boca se abre mucho y aquellos que significaban la protrusión de los labios. Esto, sin embargo, necesita mayor análisis puesto que puede ser consecuencia del método particular que genera la animación. A futuro se desean hacer las mismas pruebas con diferentes fuentes de animación, para verificar que sea el método de interpolación y no el de captura la fuente de dificultades.

Con respecto a los algoritmos de interpolación se puede decir que el que tiene mejor rendimiento según la métrica de rugosidad es Point Distance que considera todos los puntos de control, puesto que obtiene el promedio más bajo. Esto tiene el costo de “aplastar” demasiado el movimiento de la boca, lo que a primera vista y que en el futuro puede ser estudiado más a fondo, pareciera reducir la expresividad de la animación. Por lo tanto, los algoritmos geométrico (Featured based) con pesos y Point Distance con los 3 puntos más cercanos como derivadores de movimiento, fueron seleccionados como los que presentan mejor desempeño según la métrica propuesta. Los peores rendimientos se obtuvieron con los algoritmos que utilizan el sistema de pesos presentado por Lorenzo and Maddock [12] cuando el radio de acción es la arista más larga de cada elemento de control. No obstante, los datos obtenidos para estos algoritmos deben ser mirados con cuidado, puesto que no parecieran reflejar los resultados logrados por los autores. Por estas razones, la implementación del algoritmo base, Planar Bones, y su sistema de pesos asociado, se ponen en duda y deberán ser sometidos a análisis o mejoras.

El algoritmo a evaluar, aquel presentado por Cerdá et al. [6], muestra buen comportamiento en comparación a los otros implementados; sin ser el mejor, los valores de su rugosidad no se disparan y están cercanos al promedio. Sin embargo, la inclusión de más elementos de control muestra mejores resultados que la nueva ecuación propuesta por los autores para las áreas con discontinuidades.

El sistema de pesos presenta un muy buen comportamiento, produciendo en general superficies más lisas y disminuyendo irregularidades. Es importante, no obstante, tener un buen control del radio de acción puesto que cuando este es demasiado grande, afecta a vértices que no deberían ser tomados en cuenta, generando animaciones extrañas, como se muestra en la 4.14(a). Este punto es muy importante puesto que la creación de nuevas técnicas de interpolación especiales para áreas que tienen discontinuidad o presentan irregularidades no es la única solución, pudiendo seguir con

la misma técnica y simplemente agregar más elementos de control que influencien a cada vértice. De ambos sistemas implementados, Distance y el asociado al algoritmo Planar Bones, el primero muestra mejores resultados, siendo el único que aporta un aumento en la calidad de malla cuando es incluido, cuando se tiene en consideración el radio de acción que se le asigne.

Por respecto a las estructuras implementadas, también se obtuvieron buenos resultados en general. El Octree muestra una disminución importante del tiempo que requiere el proceso de registro entre las dos mallas. No obstante, la búsqueda que implementa la estructura de arcos muestra un aumento en el tiempo de ejecución y del orden matemático. Aquí, es importante considerar que el tamaño de la malla puede no ser lo suficientemente grande para que su intervención sea positiva, en especial considerando los recursos computacionales disponibles hoy en día. Es fundamental comentar que el origen del aumento puede no estar en la estructura en sí misma sino en la búsqueda en la que es usada, por lo que ésta tiene muchas posibilidades de optimización.

El diseño y calidad de software de la aplicación fue mejorado, permitiendo que agregar nuevos algoritmos de interpolación sea más fácil y limpio en términos de reducción de código. Antes de la refactorización y aplicación de los patrones de diseño, agregar un nuevo algoritmo significaba crear un nuevo método que lo implementara y luego buscar todas las referencias en el programa principal para añadir una sentencia que lo ejecutara. Actualmente, agregar un nuevo algoritmo de interpolación significa crear una nueva clase que herede de `InterpolationAlgorithm` y agregar una sola sentencia que lo refiera en la clase de factory que crea la instancia del algoritmo a utilizar. Además, si el algoritmo pudiese permitir la influencia de más de un derivador de movimiento, el template del que hereda ya incluye esta posibilidad, por lo que no es necesario hacer nada más que indicarlo al momento de instanciarlo.

En términos de animación facial, existen muchas mejoras que se pueden agregar a la aplicación con el objetivo de aumentar el realismo de esta. Entre estas mejoras, se encuentra: la animación de la lengua, el parpadeo de los ojos o incluso, incluir factores como cambios en el color de la piel y arrugas [7]. Considerando que el principal objetivo de la aplicación es la correcta o realista pronunciación de palabras, incluir la animación de la lengua permitiría que sonidos ambiguos y que solo la lengua aporta información para su identificación, como son las sílabas \di\y \da\, pudieran ser reconocidos con mayor facilidad.

Entre las líneas de trabajo futuro, se plantea la posiblidad de aplicar los algoritmos de animación implementandos. Entre las posibles aplicaciones se encuentra colaborar en la educacion de personas con dificultades en el habla. Además, el uso de la animacion para mejorar la inteligibilidad de un dialogo en ambientes con ruido.

# Bibliografía

- [1] Ogre 3d graphics rendering engines. URL <http://www.ogre3d.org/>.
- [2] Jörgen Ahlberg. Proyecto candide. URL <http://www.bk.isy.liu.se/candide/main.html>.
- [3] Jörgen Ahlberg. Candide-3 - an updated parameterised face. Technical report, 2001.
- [4] Harrison Ainsworth. Framework octree. URL [http://www.hxa.name/articles/content/octree-general-cpp\\_hxa7241\\_2005.html](http://www.hxa.name/articles/content/octree-general-cpp_hxa7241_2005.html).
- [5] Marie-Odile Berger, Jonathan Ponroy, and Brigitte Wrobel-Dautcourt. Realistic face animation for audiovisual speech applications: A densification approach driven by sparse stereo meshes. In André Gagalowicz, editor, *Computer Vision/Computer Graphics Collaboration Techniques 4th International Conference, MIRAGE 2009*, volume 5496, pages 297–307. INRIA Rocquencourt, Springer Berlin / Heidelberg, 2009. URL <http://hal.inria.fr/inria-00429338/en/>.
- [6] Mauricio Cerda, Renato Valenzuela, Nancy Hitschfeld-Kahler, Lucas D. Terissi, and Juan C. Gómez. Generic face animation. In *Proceedings of the XXIX International conference of the Chilean Computer Society*, pages 252–257. Publisher IEEE Computer Society, 2010.
- [7] Zhigang Deng and Junyoung Noh. Computer facial animation: A survey. In Zhigang Deng and Ulrich Neumann, editors, *Data-Driven 3D Facial Animation*, pages 1–28. Springer London, 2007. ISBN 978-1-84628-907-1. URL [http://dx.doi.org/10.1007/978-1-84628-907-1\\_1](http://dx.doi.org/10.1007/978-1-84628-907-1_1).
- [8] Stefan Jakobsson and Olivier Amoignon. Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization. *Computers & Fluids*, 36, 2005.

- [9] Karan Singh Karan. Skinning characters using surface-oriented free-form deformations. In *In Graphics Interface 2000*, pages 35–42, 2000.
- [10] Nikita Kojekine, Vladimir Savchenko, Mikhail Senin, and Ichiro Hagiwara. Real-time 3d deformations by means of compactly supported radial basis functions. In *In Short papers proceedings of Eurographics*, pages 35–43, 2002.
- [11] Sumedha Kshirsagar, Stephane Garchery, and Nadia Magnenat-Thalmann. Feature point based mesh deformation applied to mpeg-4 facial animation. In *Proceedings of the IFIP TC5/WG5.10 DEFORM'2000 Workshop and AVATARS'2000 Workshop on Deformable Avatars, DEFORM '00/AVATARS '00*, pages 24–34, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V. ISBN 0-7923-7446-0. URL <http://portal.acm.org/citation.cfm?id=646749.704422>.
- [12] Manuel A. Sánchez Lorenzo and Steve C. Maddock. Planar bones for mpeg-4 facial animation. In *Proceedings of the Theory and Practice of Computer Graphics 2003, TPCG '03*, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1942-3. URL <http://portal.acm.org/citation.cfm?id=832264.833414>.
- [13] W. Martin and J. Aggarwal. Motion understanding robot and human vision. Kluwer Academic Publishers, 1998.
- [14] Frederick I. Parke. Computer generated animation of faces. In *Proceedings of the ACM annual conference - Volume 1*, ACM '72, pages 451–457, New York, NY, USA, 1972. ACM. doi: <http://doi.acm.org/10.1145/800193.569955>. URL <http://doi.acm.org/10.1145/800193.569955>.
- [15] S. King Sanchez Lorenzo M., J.D. Edge and S. Maddock. Use and re-use of facial motion capture data. *Uppsala UniversityUppsala University, Division of Scientific Computing, Numerical Analysis*, 36:135–142, 2003.
- [16] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH'86*, pages 151–160, 1986.
- [17] Lucas Terissi and Juan C. Gómez. Facial motion tracking and animation: an ica-based approach. *15th European Signal Processing Conference (EUSIPCO 2007)*, pages 292–296, 2007.

- [18] Lucas D. Terissi, Mauricio Cerdá, Juan Carlos Gómez, Nancy Hitschfeld-Kahler, Bernard Girau, and Renato Valenzuela. Animation of generic 3d head models driven by speech. In *2011 IEEE International Conference on Multimedia and Expo (ICME 2011)*, Barcelona, Spain, July 11-15, 2011.
- [19] Renato Valenzuela. Memoria de Ingeniero Civil en Computación, Creación de una Herramienta para la Visualización de Animaciones de Rostros. *Universidad de Chile, Departamento de Ciencias de la Computación*, 2009.
- [20] Jian-Hua Wu, Shi-Min Hu, Jia-Guang Sun, and Chiew-Lan Tai. An effective feature-preserving mesh simplification scheme based on face constriction. *Computer Graphics and Applications, Pacific Conference on*, 0:0012, 2001. doi: <http://doi.ieeecomputersociety.org/10.1109/PCCGA.2001.962853>.

# Apéndice A

Diagrama UML de la estructura Octree

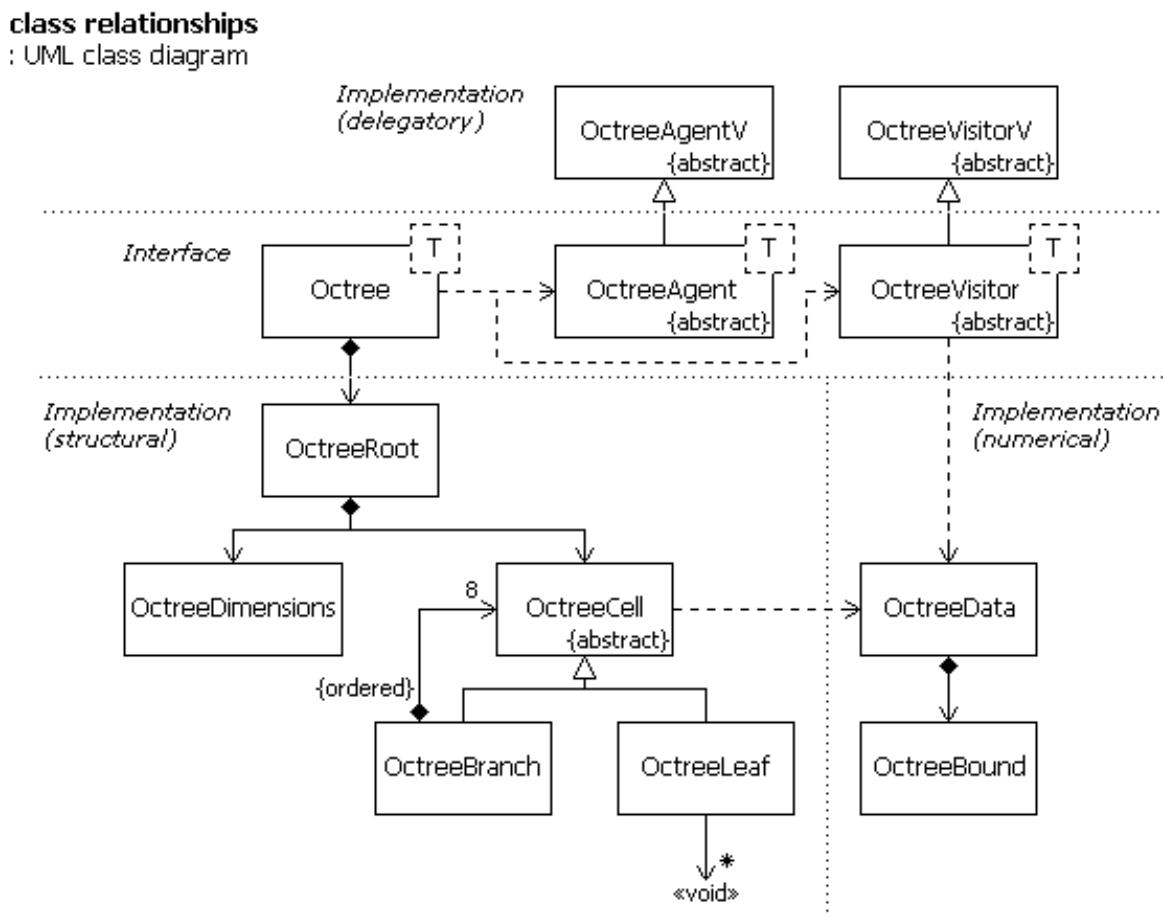


Figura A.1: Clases del octree

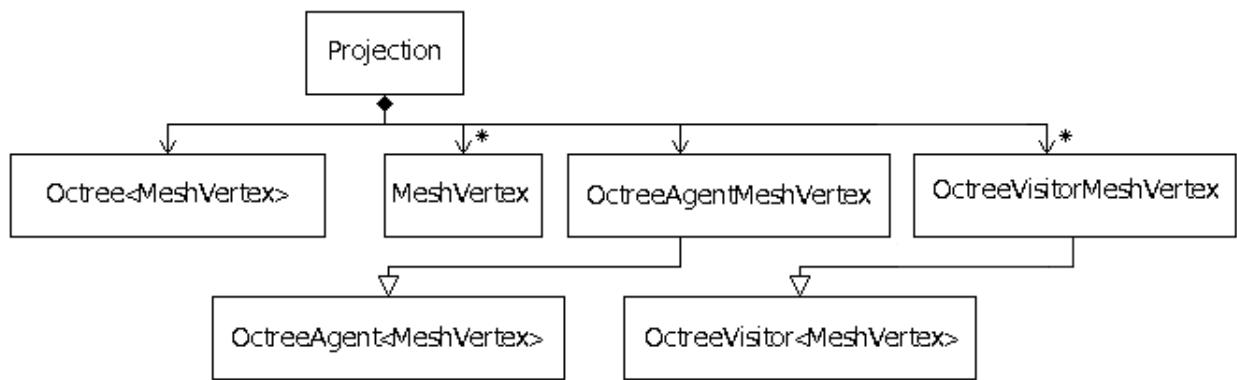


Figura A.2: Clases cliente del octree creadas para la aplicación

# Apéndice B

## Diagrama UML de clases de la aplicación

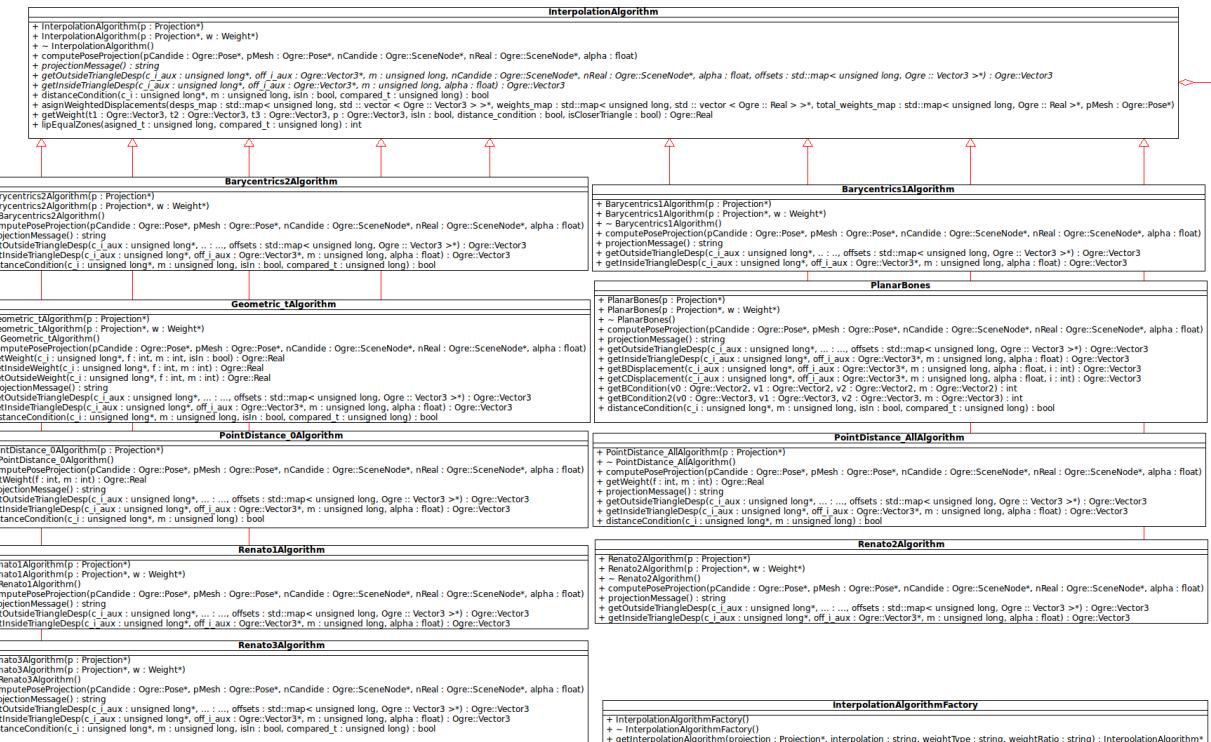


Figura B.1: Template para los algoritmos de interpolación

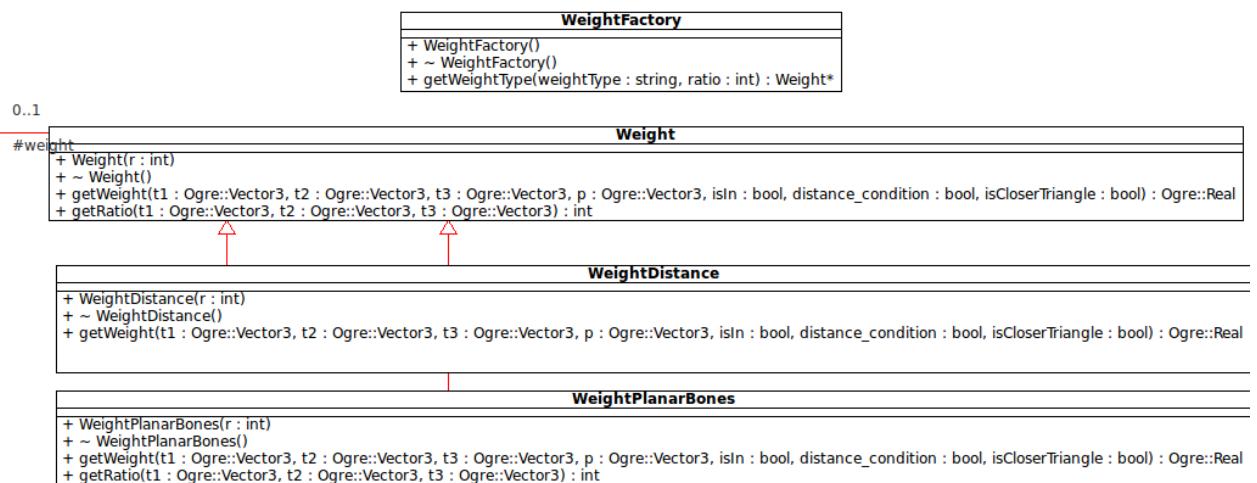


Figura B.2: Clases para el sistema de pesos

# Apéndice C

## Evaluación perceptiva de la animación con bajo ruido por sílaba

La evaluación se mide en porcentaje, entre más alto mayor es, mejor es el reconocimiento de la sílaba.

Evaluación de la animación:

Tabla C.1: Evaluación perceptiva de la animación con bajo ruido por sílaba

Sílaba	Promedio rugosidad	Sílaba	Promedio rugosidad	Sílaba	Promedio rugosidad
pa:	66.67	pi:	100.00	pu:	58.33
ta:	50.00	ti:	33.33	tu:	100.00
ka:	100.00	ki:	33.33	ku:	100.00
ba:	100.00	bi:	33.33	bu:	66.67
ma:	100.00	mi:	100.00	mu:	100.00
da:	100.00	di:	41.67	du:	8.33
sa:	25.00	si:	41.67	su:	8.33
ya:	100.00	yi:	91.67	yu:	58.33
fa:	100.00	fi:	91.67	fu:	100.00

Evaluación del video real:

Tabla C.2: Evaluación perceptiva de la animación con bajo ruido por sílaba

Sílaba	Promedio rugosidad	Sílaba	Promedio rugosidad	Sílaba	Promedio rugosidad
pa:	91.67	pi:	91.67	pu:	100.00
ta:	66.67	ti:	91.67	tu:	83.33
ka:	100.00	ki:	25.00	ku:	100.00
ba:	100.00	bi:	83.33	bu:	83.33
ma:	91.67	mi:	91.67	mu:	100.00
da:	100.00	di:	91.67	du:	66.67
sa:	41.67	si:	41.67	su:	8.334
ya:	100.00	yi:	100.00	yu:	58.33
fa:	100.00	fi:	100.00	fu:	100.00

## Apéndice D

### Rugosidad promedio de la animación de sílabas con el algoritmo Renato2

Tabla D.1: Rugosidad promedio de la animación de sílabas con el algoritmo Renato2

Sílaba	Promedio rugosidad	Sílaba	Promedio rugosidad	Sílaba	Promedio rugosidad
pa:	0.0332011	pi:	0.0326898	pu:	0.0343906
ta:	0.0361002	ti:	0.0373299	tu:	0.0401033
ka:	0.0399941	ki:	0.0415848	ku:	0.0449932
ba:	0.0394852	bi:	0.0409016	bu:	0.0423566
ma:	0.0392999	mi:	0.0388521	mu:	0.0403529
da:	0.0374423	di:	0.0392326	du:	0.044683
sa:	0.0387497	si:	0.0458779	su:	0.0376724
ya:	0.0409736	yi:	0.0477724	yu:	0.0451969
fa:	0.0435993	fi:	0.0426147	fu:	0.0367793

# Apéndice E

## Promedio y varianza de rugosidad por algoritmo

Tabla E.1: Promedio y varianza de rugosidad por algoritmo

Grupo & Letra	Algoritmo o variación de algoritmo	Promedio	Varianza
1-A	Baricéntricas 1	0.0405447	2.07743e-05
1-B	Baricéntricas 2	0.0414719	2.50292e-05
1-C	Baricéntricas 2 con peso y radio = 10 (skinning)	0.0406344	2.67238e-05
1-D	Baricéntricas 2 con peso y radio = 30 (skinning)	0.0438382	3.14295e-05
2-E	Featured based	0.0425133	3.7585e-05
2-F	Featured based con peso y radio = 10	0.0353564	1.79305e-05
3-G	Point Distance con 3 puntos más cercanos	0.0342825	6.39211e-06
3-H	Point Distance con todos los puntos + continuidad	0.0309809	2.88137e-06
4-I	Planar Bones solo, y con peso Planar Bones y radio = 10 %	0.0420838	2.69785e-05
<i>Continúa en la siguiente página</i>			

*Continúa de la página anterior*

Grupo & Letra	Algoritmo o variación de algoritmo	Promedio	Varianza
4-J	Planar Bones con peso Planar Bones y radio = 30 %, 50 %, 70 %, 80 % y 90 % de la arista más larga	0.0433552	2.67277e-05
4-K	Planar Bones con peso Planar Bones y radio = 100 % de la arista más larga	0.0609843	7.7449e-05
4-L	Planar Bones con peso Distance y radio = 10	0.0429525	2.82191e-05
4-M	Planar Bones con peso Distance y radio = 30	0.0451169	3.08538e-05
5-N	Renato 1	0.0537223	3.62147e-05
5-O	Renato 2	0.0440269	2.78166e-05
5-P	Renato 3	0.0415167	1.77456e-05
5-Q	Renato 3 con peso y radio = 10	0.04118	1.70287e-05
5-R	Renato 3 con peso y radio = 30	0.0424113	1.93542e-05