Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

ACHURADO PARA OBJETOS 3D CON ANIMACIÓN

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS,
MENCIÓN COMPUTACIÓN

EDUARDO NICOLÁS GRAELLS GARRIDO

PROFESOR GUÍA:
MARÍA CECILIA RIVARA ZÚÑIGA

MIEMBROS DE LA COMISIÓN:
NANCY HITSCHFELD
BENJAMÍN BUSTOS
DOMINGO MERY

SANTIAGO DE CHILE
NOVIEMBRE 2010

Resumen

El Rendering No Fotorrealista, o NPR (*Non Photorealistic Rendering*), es una aplicación del proceso de rendering tradicional que busca generar imágenes con algún grado de abstracción. Una técnica puntual de NPR es la de *Hatching*, o achurado. El achurado es una técnica de pintado a mano alzada que utiliza trazos cortos de líneas para graficar un objeto. La dirección de las líneas abstrae la forma de éstos, mientras que la cantidad de ellas, denominada densidad del achurado, permite interpretar la intensidad de la iluminación a través de la superficie del objeto.

La técnica de achurado que se aborda en esta tesis, propuesta por Praun et al. en SIGGRAPH 2001 en un artículo llamado "*Realtime Hatching*", solamente está definida para modelos 3D estáticos, por lo que para modelos 3D animados la deformación de la geometría no es considerada en el algoritmo. Esta tesis aborda esa problemática: se propone mejorar esa técnica de achurado, extendiéndola para ser aplicada en objetos animados.

La elección del algoritmo mencionado se justifica en la tesis a través de una revisión bibliográfica. También se detalla la implementación realizada de dicho algoritmo, se especifican las componentes del sistema y se muestran las herramientas de software libre utilizadas: OpenMesh como estructura de datos base, Eigen como biblioteca de álgebra lineal, y L-BFGS como biblioteca de optimización no lineal.

Luego de implementar el algoritmo original, se estudia su extensión para ser aplicado a un modelo 3D que presente animación. Ya que el algoritmo de achurado no está definido para objetos animados, la tesis estudia y define lo que se puede denominar como *comportamiento correcto del achurado en modelos animados*. Así, se presentan las extensiones necesarias para que el algoritmo de achurado funcione con modelos animados, junto con detalles de implementación de dichas extensiones. Además, se muestran resultados de la aplicación del algoritmo a conjuntos de datos de animación de captura de movimiento de personas. Estos resultados tienen una apariencia visual satisfactoria respecto a la definición de achurado y presentan un rendimiento adecuado para animaciones en tiempo real, al superar los 60 cuadros por segundo en su renderizado.

Finalmente, se realiza una discusión en la que se comentan los resultados del algoritmo, en particular sus cualidades y también sus limitantes. Entre sus cualidades se encuentra un planteamiento sencillo y fácil de entender que permite obtener resultados correctos. Entre sus limitantes se encuentra su aplicabilidad solamente en objetos que mantienen una topología constante durante animaciones y el hecho de funcionar solamente con animaciones preconcebidas, no con animaciones arbitrarias producto de interacción con el usuario. Sin embargo, estas limitantes son analizadas y se proponen diferentes alternativas para enfrentarlas como trabajo futuro.

La gloria, como todos saben, tiene sabor amargo.
– Yukio Mishima
"El marino que perdió la gracia del mar"

Con amor, dedicado a los Passarinhos.

Índice general

Capítulo 1

Introducción

El Rendering No Fotorrealista, o NPR por sus siglas en inglés (*Non Photorealistic Rendering*), es una aplicación del proceso de rendering tradicional que busca generar imágenes con algún grado de abstracción. Esta aplicación de rendering existe ya que en algunos escenarios no se desea una imagen totalmente realista, llena de detalles que pueden no ser relevantes, más bien se requiere una imagen que satisfaga los criterios de visualización de un observador determinado. Entre las distintas aplicaciones de NPR se encuentra la ilustración técnica, la visualización de datos de volumen, la visualización de planos arquitectónicos y mapas geográficos, aplicaciones lúdicas como películas y videojuegos y la simulación de medios tradicionales como la pintura o la acuarela.

Figura 1.1: Técnica de Achurado aplicada a un modelo tridimensional de una mano. Fuente: [WPFH02].

Una técnica puntual de NPR es la de *Hatching*, o achurado, visible en la Figura 1.1. El achurado es una técnica de pintado a mano alzada que utiliza trazos cortos de líneas para graficar un objeto. La dirección de las líneas abstrae la forma de éstos, mientras que la cantidad de ellas (la densidad del achurado) permite interpretar la intensidad de la iluminación a través del objeto.

1.1. El Problema a Resolver

La técnica de achurado que se aborda en esta tesis, propuesta por Praun et al. [PHWF, WPFH02] trabaja en espacio objeto, es decir, directamente con la geometría de los objetos presentes en la escena a graficar, y sólo está definida para modelos 3D estáticos, por lo que para modelos animados la deformación de la geometría no es considerada en el algoritmo. Esta tesis aborda esa problemática: se propone mejorar esa técnica de achurado, extendiéndola para ser aplicada en objetos animados.

El problema visual que se genera al aplicar la técnica original a objetos animados consiste en que las líneas de achurado se alargan y se ensanchan al seguir la deformación de la geometría. Esta deformación en las líneas no representa de manera visual la deformación de la geometría, por el contrario, tiene una apariencia elástica que no necesariamente la representa y que desvía la atención desde la abstracción de la superficie hasta la deformación de las líneas.

Un trabajo reciente que aborda este problema es una técnica de achurado en espacio imagen de Kim et al. [KYYL08]. Al trabajar en espacio imagen, la técnica trabaja directamente sobre los píxeles de la escena ya renderizados, por lo que se aplica indistintamente de si hay animación o no la hay. Dicho algoritmo resuelve las inquietudes planteadas, pero al llevarse a cabo en espacio imagen, no se tiene control sobre el resultado, mientras que trabajar en espacio objeto permite una mayor flexibilidad y extensibilidad en la técnica, y también un mejor desempeño.

1.2. Objetivos

El objetivo general de la tesis es implementar un algoritmo de achurado, que presente un comportamiento correcto respecto al algoritmo original en modelos animados. El algoritmo debe extender el algoritmo de Praun et al. [PHWF, WPFH02].

1.2.1. Objetivos Específicos

- Estudiar e implementar el algoritmo original de Praun et al. [PHWF, WPFH02] para modelos estáticos.
- Estudiar el comportamiento del algoritmo original al ser aplicado a modelos animados.
- Proponer extensiones al algoritmo original con el fin de obtener resultados correctos para modelos animados.
- Implementar el algoritmo propuesto y evaluar su alcance respecto a los distintos tipos de animaciones existentes.

1.3. Herramientas

Como Memoria para optar al título de Ingeniero Civil en Computación, se diseñó un framework que implementa estructuras de datos y algoritmos para rendering tradicional y rendering no fotorrealista básico [Gra08]. Entre otras cosas, el framework implementa el cálculo de las propiedades geométricas de un modelo 3D y permite graficar modelos 3D estáticos y animados con la técnica *Vertex Animation*. Incluye estructuras de datos que se pueden comunicar con la GPU, extracción de curvas de nivel, e implementaciones de técnicas elementales de Computación Gráfica. El framework utiliza OpenGL [SWND03] como biblioteca gráfica y el lenguaje C++ [S+97] como lenguaje principal de desarrollo.

El framework base es replanteado y extendido en la tesis, reemplazando las estructuras internas con una biblioteca robusta para mallas de triángulos llamada OpenMesh [BSBK02] y una biblioteca de álgebra lineal llamada Eigen [JG], mejorando el diseño y el rendimiento del framework.

1.4. Metodología y Contenido de la Tesis

La metodología de la tesis considera la siguientes etapas, cada una contenida en un capítulo:

- Estudiar el estado del arte del problema mediante una revisión bibliográfica en el Capítulo 2.
- Discutir las herramientas utilizadas y su preparación para la implementación de la tesis en el Capítulo 3.
- Implementar la técnica original de achurado Praun et. al [PHWF], utilizando la mejora de texturas 3D planteada por Webb et al. [WPFH02]. La implementación se muestra en el Capítulo 4.
- Diseñar e implementar el algoritmo de achurado extendido para superficies animadas. El resultado se muestra en el Capítulo 5.
- Discutir las ventajas y desventajas del algoritmo resultante, así como el trabajo futuro, en el Capítulo 6.

1.5. Contribuciones de la Tesis

El trabajo desarrollado en la tesis presenta dos contribuciones al estado del arte del rendering no fotorrealista:

- Se provee un conjunto de herramientas que permiten trabajar con algoritmos de renderizado no fotorrealista. Dicho trabajo fue enviado y aceptado en la conferencia SIACG 2009 (Simposio Ibero-Americano de Computación Gráfica)[1].
- Se diseña e implementa un algoritmo que permite graficar objetos animados con técnica de achurado en espacio objeto, utilizando el framework del punto anterior. Dicho trabajo será enviado a la conferencia NPAR 2011 en forma de *paper*.

1.6. Disponibilidad de la Implementación

Junto a la tesis se incluye un disco que contiene la implementación realizada. El Apéndice A detalla los contenidos del disco y las instrucciones de compilación y ejecución, junto con todos los requerimientos necesarios.

[1] El trabajo no fue presentado por motivos económicos.

Capítulo 2

Antecedentes

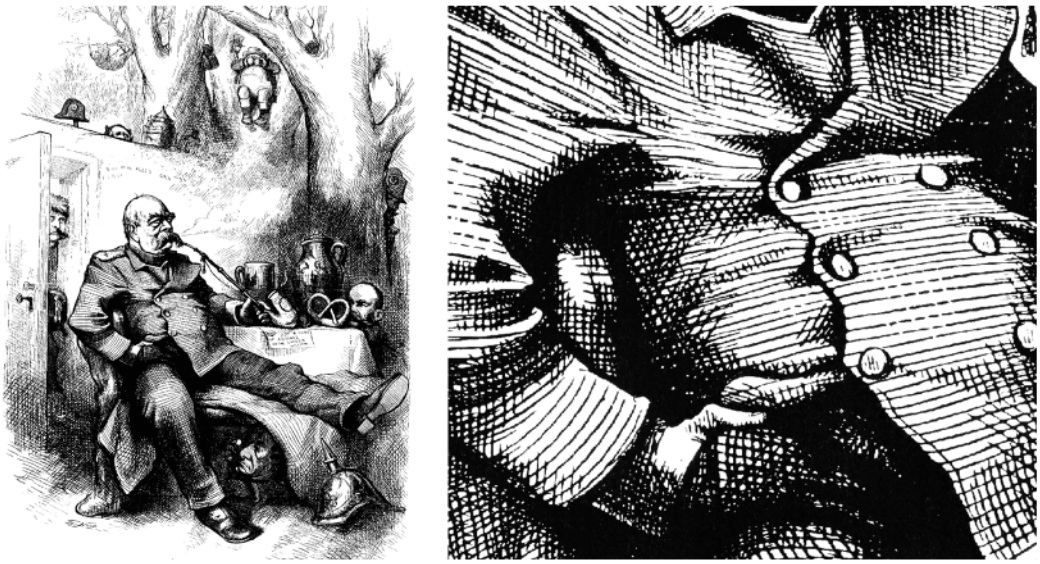


Figura 2.1: Izquierda: ilustración de Thomas Nast. Derecha: detalle del achurado. Fuente: [HZ].

El dibujo de un objeto se compone, en general, de dos partes: el contorno o silueta del objeto y el pintado interior. El achurado es una técnica de pintado interior en una ilustración, en la que el espacio dentro del contorno de los objetos es rellenado con diversas líneas o patrones. La Figura 2.1 muestra una imagen de ejemplo, creada por el ilustrador Thomas Nast, donde se aprecian líneas de achurado. A partir del detalle de la figura se pueden inferir dos propiedades básicas de dichas líneas:

- **Dirección**: localmente dentro de la superficie del objeto, las líneas siguen la misma dirección. Esta dirección se utiliza para representar la forma geométrica, en términos de la curvatura, del interior de un objeto.
- **Densidad**: la densidad de las líneas define la cantidad de luz que llega a la superficie: a menor densidad, mayor iluminación. La densidad puede aumentar con líneas que avanzan en la misma dirección, dibujándose más líneas juntas, o bien con un patrón de líneas entrecruzadas.

El principal concepto detrás del achurado es que en base a líneas que varían su cantidad, grosor, dirección y espaciamiento, se comunica tanto la geometría de un objeto como las condiciones de luz que lo afectan. Las líneas *no* son parte del objeto, más bien, son un añadido extra que, mediante el sombreado de éste, abstrae las dos características ya mencionadas.

2.1. Rendering: Gráficos Generados por Computador

Por *rendering* se entiende el proceso de generar una imagen en un computador a partir de la descripción de una escena que puede ser 2D o 3D. Esta descripción incluye uno o más objetos, cada uno con posición, orientación, color y otras propiedades. También se incluye la descripción de fuentes de luz y cámaras.

En general, cuando se habla de rendering, de manera ímplicita se está hablando del *rendering fotorrealista*. La Figura 2.2 muestra una escena de ejemplo, que contiene varios objetos. Se observan efectos de sombras, reflectividad de los materiales, materiales que absorben luz (efecto conocido como *subsurface scattering*), entre otros. Si bien el aspecto de la imagen no es perfecto, las técnicas y algoritmos que permiten obtener una calidad de imagen fotorrealista están continuamente mejorando.

Figura 2.2: El Conejo de Stanford graficado dentro de una escena con condiciones de iluminación y materiales de apariencia fotorrealista. Fuente: Henrik Wann Jensen, Stanford.

Una clasificación para las técnicas de rendering proviene del tiempo que toma ejecutar el proceso. En particular, se habla de *rendering en tiempo real* cuando la imagen que visualiza una escena se puede generar *al instante*, en tiempo de ejecución del programa que realiza el rendering; y de rendering en tiempo no real o *rendering off-line* cuando el tiempo que toma generar la imagen se extiende más allá de lo que se podría considerar instantáneo.

Como la noción de “generación instantánea de la imagen” es subjetiva, se ha llegado a la siguiente convención: para que una técnica de rendering sea considerada como de ejecución en tiempo real, es necesario que dentro de un segundo se pueda graficar en pantalla al menos 15 veces [AMH08]. La cantidad de veces que se genera una imagen por segundo es llamada *frames per second*, *fps*.

El rendering en tiempo real tiene un gran número de aplicaciones, en particular todas aquellas que requieren interactividad con el usuario. Debido a que se requiere esa interactividad, donde el usuario modifica parámetros de la visualización o incluso de los objetos que componen la escena, la imagen es constantemente generada para reflejar los cambios provocados por la interacción.

2.1.1. Rendering No Fotorrealista

Además del rendering tradicional, existe una clasificación llamada *rendering no fotorrealista*, donde el énfasis no se pone en las características de la escena que se está graficando, sino que en lo que quiere o necesita visualizar el observador. El rendering no fotorrealista (NPR, *Non Photo-Realistic Rendering*) genera una visualización abstracta de la escena, entendiéndose abstracción como el acto de realzar o destacar características de los objetos, en particular las características que son de importancia para el observador.

Figura 2.3: Imagen no fotorrealista del Conejo de Stanford, graficándolo como si fuese un dibujo hecho con líneas. Fuente: [DFRS].

La Figura 2.3 muestra al Conejo de Stanford de manera no fotorrealista, utilizando la técnica *Suggestive Contours* de De Carlo et al. [DFRS, DFR04]. Esta visualización plantea una abstracción del objeto utilizando líneas que destacan el contorno o silueta del objeto, más lo que se conoce como “contornos sugestivos”, representados por las líneas al interior del objeto. Así, la técnica permite que el observador reconozca y entienda la figura del conejo utilizando solamente líneas ubicadas en lugares característicos del objeto.

En NPR existen numerosas técnicas que abstraen diferentes cualidades y características

de los objetos. En particular, las técnicas más conocidas son *Gooch Shading* [GGSC], utilizado en ilustración técnica; *Toon Shading* [LMHB00], utilizado en animación y videojuegos; las técnicas basadas en líneas como *Suggestive Contours* [DFRS, DFR04] y *Apparent Ridges* [JDA]; las basadas en imitar estilos pictóricos, llamadas *Painterly Rendering* [Her]; las que buscan imitar medios tradicionales como acuarela llamadas *Watercolor Rendering* [BKTS06]; entre otras, incluyendo la del achurado que se describirá más adelante en este Capítulo.

Cuando una técnica se aplica directamente sobre la escena a graficar, trabajando con las mallas y texturas de la escena, se dice que es una técnica de espacio objeto. Cuando la técnica trabaja con píxeles luego de realizar el renderizado, se dice que es una técnica de espacio imagen.

2.1.2. Superficies como Mallas de Triángulos

Toda técnica de rendering, independiente de su clasificación, trabaja con descripciones de escenas en las cuales se indican los objetos a graficar. Estos objetos son superficies arbitrarias, que en computación gráfica se representan utilizando *mallas poligonales*, en particular las de *cuadriláteros* y *triángulos*. En la Figura 2.4 se muestra la malla de triángulos de la tetera de Utah.

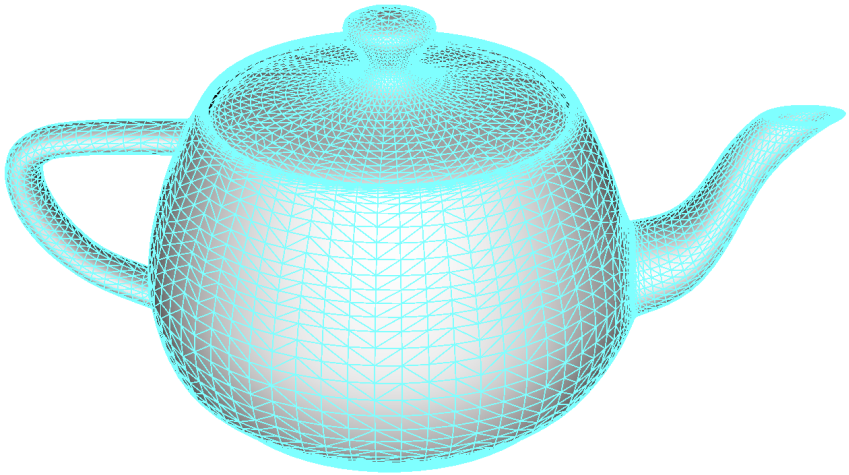


Figura 2.4: Malla de triángulos para la tetera de Utah. Imagen generada utilizando el software RTSC [DFR04].

Una malla de triángulos es una discretización de una superficie, donde se utiliza un conjunto de triángulos conectados entre sí mediante aristas y vértices. Estos elementos se pueden definir de la siguiente manera:

- **Vértice**: un punto en el espacio que es parte de la topología de la malla.
- **Arista**: cerradura convexa de dos vértices.
- **Triángulo**: cerradura convexa de tres vértices.

- **Malla de Triángulos**: conjunto de triángulos $\tau = \{T_1, \ldots, T_m\}$ que sólo se intersectan a través de aristas $\varepsilon = \{E_1, \ldots, E_l\}$ y vértices $\upsilon = \{\mathbf{p}_1, \ldots, \mathbf{p}_{n+b}\}$. En particular, los vértices se dividen en n vértices de interior $\upsilon_I = \{V_1, \ldots, V_n\}$ y b vértices de borde $\upsilon_B = \{V_{n+1}, \ldots, V_{n+b}\}$.
- **Vecindad**: dos vértices distintos, $\mathbf{p}_i, \mathbf{p}_j \in \upsilon$ son llamados *vecinos* si son los puntos extremos de una arista $E = [\mathbf{p}_i, \mathbf{p}_j]$. También se define el conjunto de índices vecinos de un vértice $\mathbf{p}_i$ como $N_i = \{j : [\mathbf{p}_i, \mathbf{p}_j] \in \varepsilon\}$.

Esta definición impone la siguiente condición: dos triángulos sólo pueden intersectarse en una arista o en un vértice.

Para el rendering en tiempo real, el hardware gráfico está preparado para graficar una gran cantidad de triángulos por segundo (el orden es de millones de triángulos). En caso de otras representaciones (como mallas de cuadriláteros o superficies implícitas), es necesario convertirlas a mallas de triángulos.

2.2. Achurado Generado por Computador

De acuerdo a las definiciones anteriores, la técnica de graficar una escena 3D con un estilo de achurado pertenece al área del NPR. Implementar una técnica de achurado implica implementar por separado las dos características del achurado, densidad y dirección de las líneas, y utilizarlas dentro del mismo contexto de manera coherente. La Figura 2.5 muestra dos técnicas de achurado, una en espacio imagen y off-line, otra en espacio objeto en tiempo real.

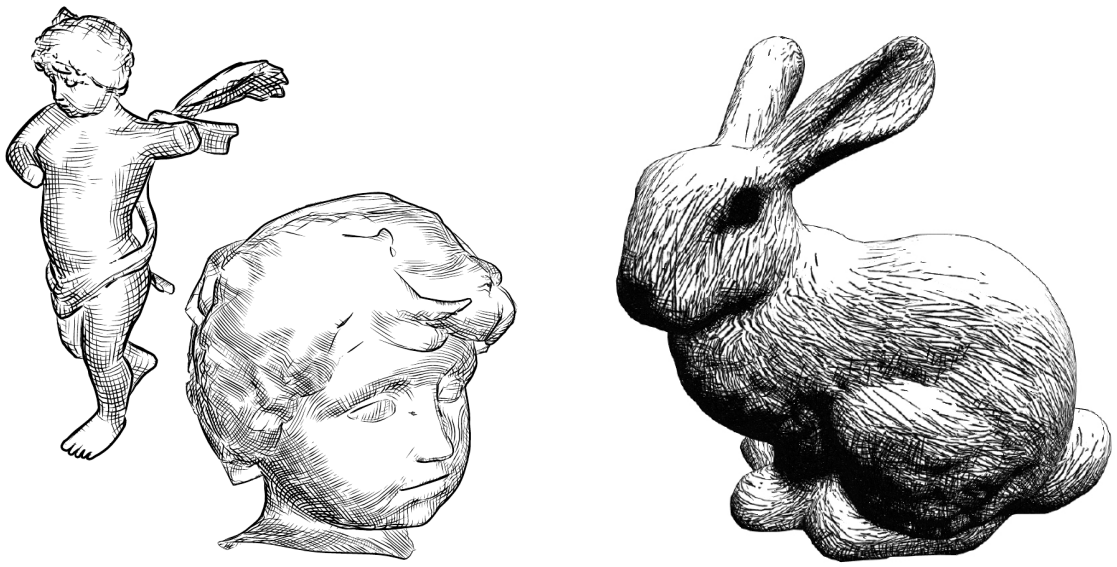


Figura 2.5: Dos técnicas de achurado. En la izquierda, una técnica en espacio imagen que se ejecuta off-line, de Hertzmann y Zorin [HZ]. En la derecha, una técnica en espacio objeto que se ejecuta en tiempo real, por Praun et al. [PHWF]

En general, la dirección de las líneas se establece de manera automática, utilizando las direcciones principales [dC76] de la superficie. Estas direcciones indican, en un punto de la superficie, en qué dirección se encuentran las curvaturas mayor y menor (en un punto cualquiera de la superficie existen infinitas curvaturas, una por cada plano de corte con normal tangencial a la superficie). El uso de las direcciones principales ha sido validado experimentalmente por diferentes técnicas de NPR (en particular, para el achurado, por Interrante [Int], Hertzmann y Zorin [HZ] y Praun et al. [PHWF, WPFH02]), por lo que en la actualidad se considera que son suficientes para abstraer la forma que tiene un objeto. En el caso de una superficie estática, las direcciones principales se calculan una única vez en una etapa de preproceso de la escena que se va a visualizar. Dicho cálculo se puede realizar utilizando el algoritmo de cálculo de curvatura para mallas de triángulos de Rusinkiewicz [Rus04].

Para la densidad de las líneas se utiliza la componente difusa del modelo de iluminación de Phong: a mayor intensidad de iluminación, menor densidad de líneas.

El principal problema en lo que respecta al achurado generado por computador no es *cómo realizar el achurado*, sino más bien *cómo hacerlo dentro de un contexto particular*. En el caso de la tesis, este contexto es el rendering en tiempo real. Un enfoque ingenuo para graficar el achurado puede ser graficar líneas sobre la superficie de manera independiente (una por una). Esta solución no funciona en tiempo real ya que la magnitud de las líneas puede ser muy grande y, para la cantidad de líneas independientes entre sí que se va a dibujar, no se aprovecha el hardware gráfico, que está optimizado para graficar triángulos.

2.2.1. Técnicas de Achurado a lo largo del tiempo

Una de las primeras técnicas de achurado es la de Saito y Takahashi [ST90]. Esta técnica funciona en espacio imagen, en la cual se mezclan distintos *G-Buffers* y distintas visualizaciones abstractas de los objetos con un achurado precalculado, como se observa en la Figura 2.6. Un G-Buffer es un "buffer geométrico", similar al *Depth Buffer*, pero que almacena información geométrica de la escena. Por ejemplo, un G-Buffer puede contener la posición de la superficie visible en un píxel, la normal de la superficie visible en un píxel, o ambas. A partir de estos G-Buffers se establecen las densidades de achurado en la imagen, mientras que la dirección de las líneas se encuentra predeterminada mediante curvas de nivel. El achurado además es complementado con detección de bordes en el Z-Buffer y en un G-Buffer de normales, para marcar los contornos de los objetos. Si bien la técnica es difícil de implementar, en particular por el cálculo de curvas de nivel para el achurado, en la actualidad el uso de G-Buffers es común, no sólo para NPR, sino también para rendering tradicional.

Una técnica posterior que funciona en espacio de imagen, pero que tiene una complejidad mucho menor, es la de Lake et al. [LMHB00], visible en la Figura 2.7. Funciona en tiempo real, pero la dirección de las líneas de achurado son independientes de los objetos que se están graficando. La técnica funciona del siguiente modo: para cada triángulo que se grafica, se establece una intensidad de iluminación promedio. Cuando el triángulo es proyectado en el plano de visión, se le superpone una textura de achurado que representa la intensidad de iluminación calculada. La superposición de la textura es siempre fija en pantalla, lo

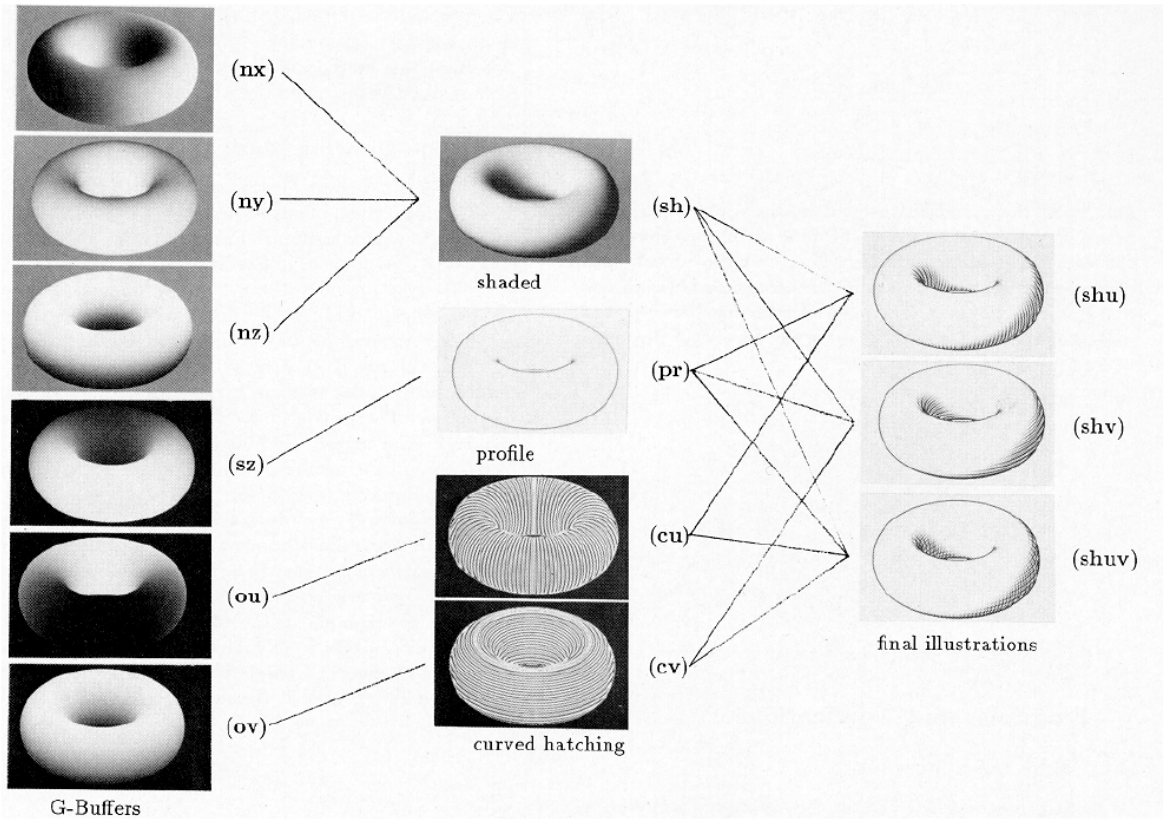


Figura 2.6: Algoritmo en espacio imagen de Saito y Takahashi [ST90].

cual genera un problema llamado efecto de corredera o *shower door effect*. Este problema se refiere a que en la imagen hay efectos visuales que se mantienen fijos en el espacio de la imagen, como la puerta de corredera de una ducha. En el caso de la técnica recién mencionada, las texturas de achurado se mantienen fijas en el plano de visión mientras las superficies se mueven debajo de ellas.

Una tercera técnica es la propuesta por Hertzmann y Zorin [HZ], que mezcla el espacio objeto y el espacio imagen, visible en la Figura 2.8. Esta técnica utiliza las direcciones principales de un objeto para establecer la dirección de las líneas de achurado, pero además realiza un post-proceso de dichas direcciones con el fin de suavizarlas y obtener un conjunto de líneas más cercano a los utilizados por artistas reales. La técnica descrita es híbrida porque, si bien las líneas de achurado se calculan en espacio objeto, su graficación se realiza en espacio imagen. Esto se debe a que los autores han propuesto su técnica para generar imágenes, no para realizar rendering de escenas 3D de manera interactiva. Otra contribución clave es que, si la superficie presenta muchos detalles, es mejor realizar el proceso de obtención de líneas de achurado en una versión suavizada de la malla, lo cual parece intuitivo pero en realidad no había sido un tema abordado hasta ese momento debido a que fue la primera técnica que trabajó en espacio objeto.

La siguiente técnica de achurado es la propuesta por Praun et al. [PHWF], que funciona en espacio objeto, en tiempo real, y que es la base de trabajo de esta tesis. Dicha técnica será analizada en la Sección 2.3.

La técnica más reciente de achurado es la propuesta por Kim et al. [KYYL08], visible

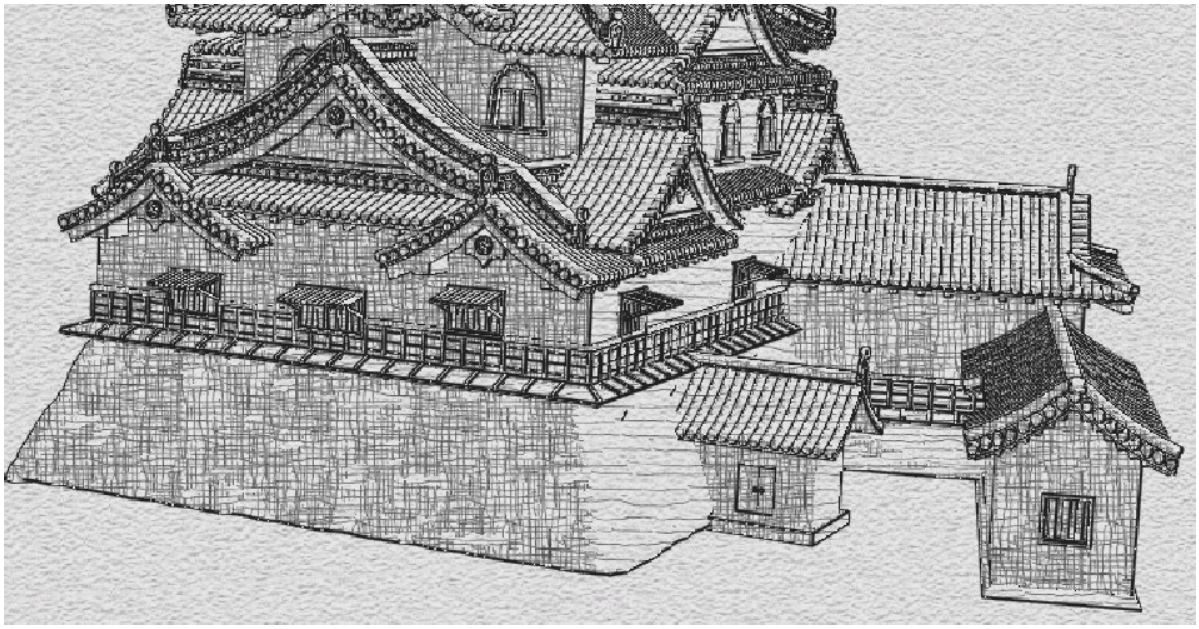


Figura 2.7: Algoritmo en espacio imagen de Lake et al. [LMHB00].

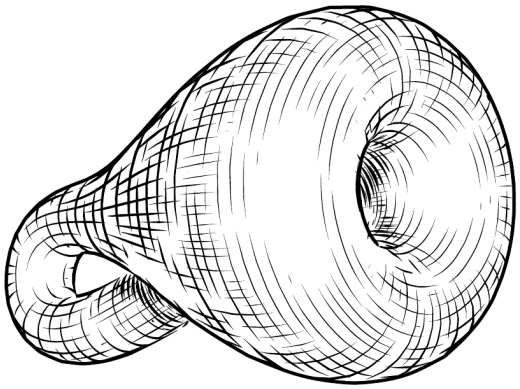


Figura 2.8: Botella de Klein graficada con la técnica de Hertzmann y Zorin [HZ].

en la Figura 2.9. Esta técnica funciona en espacio imagen, y se puede considerar como un algoritmo completo debido a que incluye achurado incluso en condiciones de sombras, reflexión y refracción de objetos. La técnica trabaja con *G-Buffers* [ST90] y Ray Tracing en la GPU. Funciona calculando la curvatura y las direcciones principales en cada píxel constantemente, independiente de si la superficie ha presentado deformaciones o no. Las texturas de achurado se posicionan en la escena de acuerdo a un algoritmo que mezcla las imágenes a partir de las direcciones principales en cada píxel.

Al ser una técnica ejecutada en espacio imagen, no es necesario considerar si la geometría es animada o presenta deformación, ya que el único input que se necesita son los G-Buffers de la escena. Esto es una ventaja cuando se trabaja con escenas complejas, puesto que el costo de aplicar el algoritmo es independiente del número de objetos graficados; por otro lado, es una desventaja pues impide tener control sobre el algoritmo, ya que el algoritmo no tiene acceso a los datos de los objetos que se están graficando.

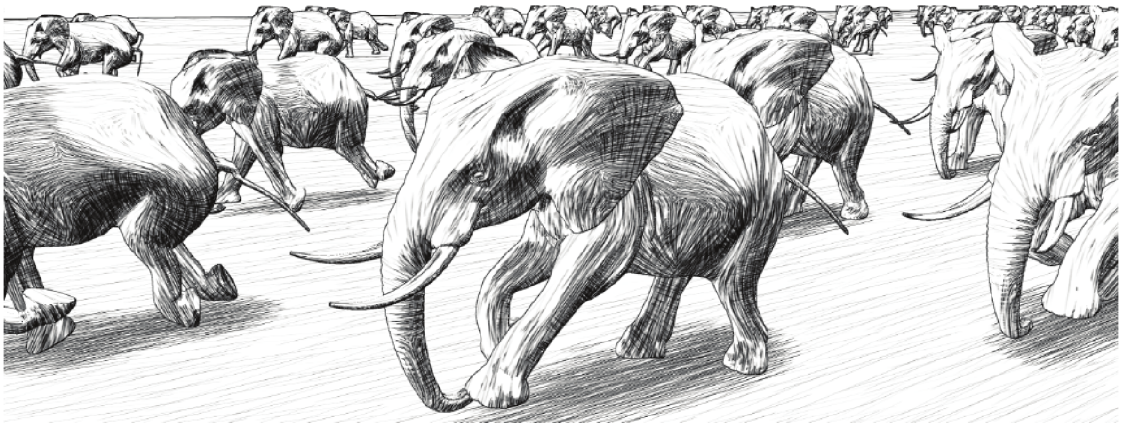


Figura 2.9: Achurado en espacio imagen para una animación. Fuente: Kim et al. [KYYL08].

Un problema generalizado a las técnicas en espacio imagen es la coherencia temporal. Usualmente no se utiliza información de los frames anteriores para generar el achurado en el frame actual, por lo que las líneas pueden cambiar súbitamente de posición u orientación.

2.2.2. Cross Fields o campos de líneas que se cruzan entre sí

En la Sección anterior se mencionó que las direcciones principales son utilizadas como campo de direcciones sobre la superficie, con el fin de guiar las líneas del achurado. Un aspecto clave de esto es que Hertzmann y Zorin [HZ] validaron el uso de estas direcciones, y además propusieron realizar un post-proceso de suavizamiento laplaciano que, debido a que la superficie se vuelve más curva, entrega un campo direccional más uniforme, similar al achurado que se hace en la vida real, donde el artista se guía por la geometría del objeto pero se toma licencias respecto a cómo plasmar esa geometría en las líneas.

De acuerdo al estudio que ellos realizaron sobre ilustraciones de achurado, en los sectores parabólicos de la superficie, aquellos que localmente tienen forma cilíndrica, las líneas deben seguir las direcciones principales de la superficie; en los otros sectores, las líneas deben seguir a las curvas geodésicas de la superficie[1]. Para garantizar este comportamiento de las líneas de achurado, los autores plantean la minimización de un funcional no-lineal, que considera el ángulo entre los vectores del campo de direcciones entre triángulos vecinos. El funcional es no lineal debido a la presencia de funciones seno y coseno.

El resultado de la optimización es un cross field, definido como un campo de vectores tangenciales sobre la superficie, que cumple con la siguiente característica: entre dos triángulos vecinos de la malla de triángulos, los ángulos entre dos vectores, uno de cada cara, son lo más pequeño posible. Para cada cara existen 4 vectores, que forman una cruz pues son ortogonales entre sí.

La Figura 2.10 muestra el comportamiento del campo de direcciones en una malla que modela un torso humano. Como se observa, al utilizar solamente las direcciones principales, el patrón obtenido efectivamente representa la geometría del objeto, pero es

[1]Las geodésicas son las trayectorias de menor distancia entre un punto y otro de una superficie.

un patrón muy complejo que difícilmente sería utilizado por un artista para achurar un torso humano. Al suavizar parcial o totalmente el campo, los resultados son mucho más agradables estéticamente.

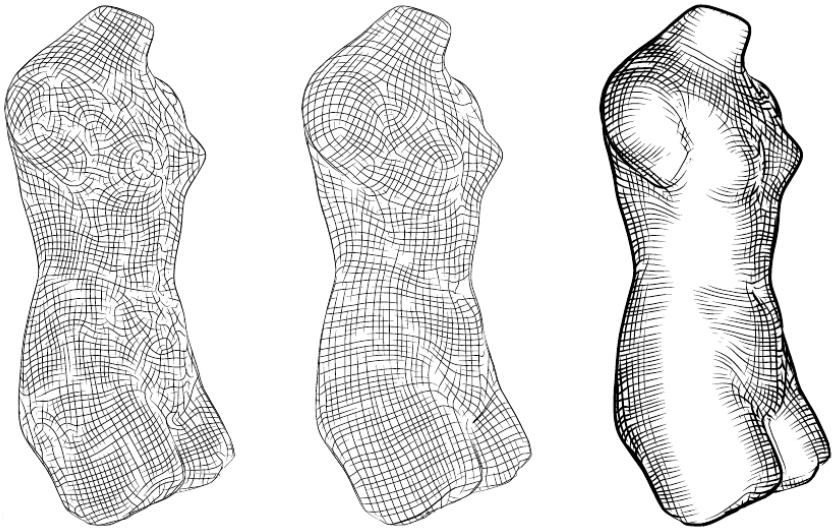


Figura 2.10: La primera imagen muestra el *cross field* definido por las direcciones principales. La segunda imagen muestra el *cross field*. La tercera muestra el achurado. Fuente: Hertzmann y Zorin [HZ].

La gran desventaja que presenta la técnica de Hertzmann y Zorin es que la generación del campo de direcciones suave depende de un proceso de optimización no-lineal, del cual no se tiene control. Es decir, solamente se puede especificar el campo de direcciones inicial. El campo suavizado puede presentar singularidades sobre las cuales no se tiene control de la posición dentro de la superficie.

Un trabajo interesante es el de Palacios y Zhang [PZ07], que establece toda una teoría para campos con simetrías rotacionales, llamados *N-RoSy* (*N Rotational Symmetries*). En particular, un *cross field* es un campo vectorial *4-Rosy*. Un punto a destacar de ese trabajo es que la ilustración de los campos 4-RoSy en una superficie son graficadas con el algoritmo de Hertzmann y Zorin.

2.3. Achurado en Tiempo Real

El achurado en tiempo real con el que se trabaja en la tesis fue planteado por Praun et. al [PHWF], con mejoras de Webb et. al [WPFH02]. La técnica planteada trabaja de la siguiente manera: codifica el achurado dentro de un mapa de texturas que se aplica a la superficie, y aplica estas texturas en la superficie utilizando técnicas de parametrización de mallas [HLS].

2.3.1. Codificación del Achurado en Texturas: Tonal Art Maps

El mapa de texturas que se aplica en la superficie tiene las siguientes características:

- Existe un set de texturas de achurado, cada una representa una intensidad de iluminación para la superficie.
- Cada textura de achurado está compuesta por líneas verticales. Si el achurado es muy denso, también tiene líneas horizontales.
- Para dos intensidades de iluminación I_1 e I_2 y dos texturas de achurado T_1 y T_2, si $I_1 < I_2$ entonces T_2 contiene a T_1, es decir, las líneas que están dentro de T_1 también están dentro de T_2.
- Todas las texturas de achurado tienen el mismo tamaño.

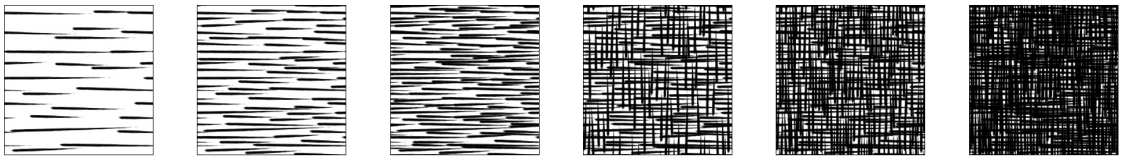


Figura 2.11: Tonal Art Map usando texturas de achurado. Fuente: [PHWF].

Este tipo de mapas es conocido como *Tonal Art Map* (TAM), y fue definido por Praun et al. [PHWF], aunque en técnicas anteriores (como la de Lake et al. [LMHB00]) se utilizó un mapa de texturas similar. En la Figura 2.11 se muestra un TAM especial para el achurado.

2.3.2. Lapped Textures

Un TAM se aplica sobre una superficie utilizando la técnica *Lapped Textures* de Praun et. al [PFH]. Dicha técnica realiza una segmentación de la malla de triángulos de una superficie, dividiéndola en *parches*. A cada parche se le crea una parametrización del espacio de texturas (ejes u y v) tal que la textura aplicada está alineada con un campo de direcciones $(\mathbf{S}, \mathbf{T})$ especificado por el usuario en cada triángulo. La Figura 2.12 muestra el resultado de la técnica en base a un campo de direcciones especificado por un usuario y a una textura de ejemplo.

La entrada del algoritmo es la siguiente:

- Una malla de triángulos sobre la cual aplicar la textura.
- Un campo de direcciones especificado sobre la malla que contiene dos vectores de dirección $(\mathbf{S}, \mathbf{T})$, ortogonales, por cada triángulo, una dirección de avance ($\mathbf{S}$) y una dirección "arriba" ($\mathbf{T}$)).
- La textura o patrón a aplicar. Los ejes u, v de la textura deben alinearse con las direcciones S y T del campo de direcciones.

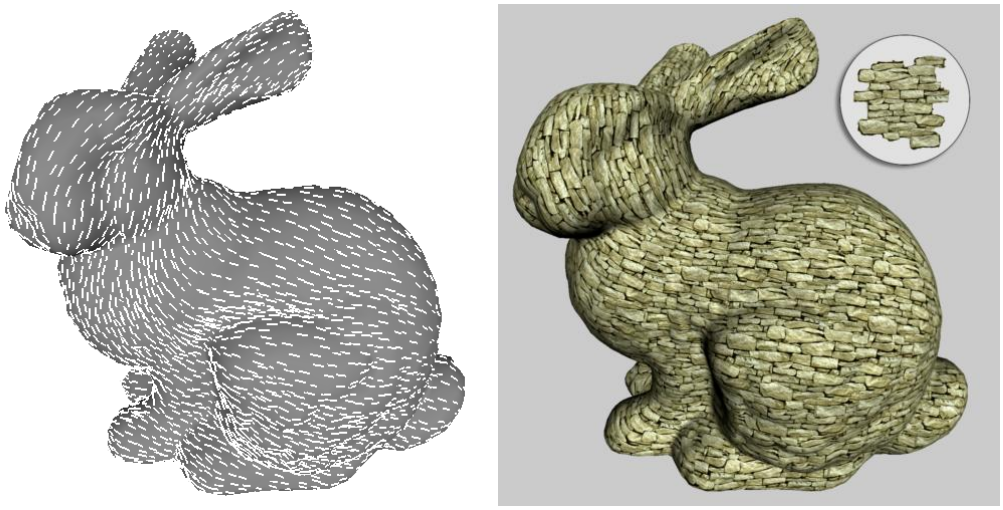


Figura 2.12: Izquierda: vector de direcciones en el conejo de Stanford (sólo se muestra la dirección S, T se calcula como $S \times N$). Derecha: resultado de Lapped Textures usando una textura de piedra en el conejo de Stanford. Fuente: Praun et al. [PFH].

La salida del algoritmo es un conjunto de parches que en su totalidad cubren toda la superficie. Cada parche tiene una parametrización cuyas coordenadas de textura alinean de manera óptima los ejes de la textura utilizada con el campo direccional en cada triángulo de la malla.

A continuación se enumera el funcionamiento de esta técnica:

- Se establece el tamaño del área de superficie que será cubierta por un parche.
- Se segmenta la superficie en parches de forma circular, cuidando que cada parche no sobrepase el tamaño especificado en el punto anterior.
- Cada parche se construye de manera incremental, primero agregando un triángulo y luego sus triángulos vecinos.
- A medida que se van agregando triángulos a un parche, se va construyendo una parametrización inicial, alineada solamente con el primer triángulo agregado al parche. La condición para poder agregar un triángulo es: 1) al menos la parametrización de 1 vértice tiene que estar dentro del círculo que engloba la parametrización, 2) la parametrización debe permanecer con topología de disco una vez agregado el triángulo.
- Cuando la superficie está cubierta completamente por parches, se alinea de manera óptima, mediante mínimos cuadrados, el campo de direcciones de cada triángulo con la textura que se está aplicando. Para esto los ejes de la parametrización se alinean con el campo direccional especificado en cada triángulo.

Optimización de la Parametrización los Parches

La optimización de la parametrización de los parches es el paso final del algoritmo, visible en la Figura 2.13.

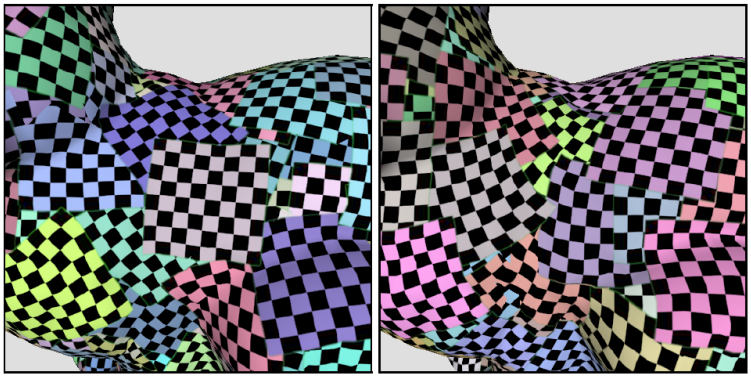


Figura 2.13: A la izquierda, los parches de acuerdo a la parametrización inicial. A la derecha, los parches luego de una optimización de la parametrización. Fuente de la imagen: Praun et al. [PFH].

Ahora bien, esto implica que se está alineando la parametrización, especificada en un espacio 2D, con los vectores de dirección especificados para cada triángulo, especificados en un espacio 3D. Para poder realizar la conversión desde un espacio a otro, se puede recurrir a las **coordenadas baricéntricas**, utilizadas para especificar las coordenadas de un punto en el plano de un triángulo con respecto a sus vértices, por lo cual son válidas en todos los sistemas de coordenadas. Se definen como:

$$P = \alpha P_1 + \beta P_2 + \gamma P_3$$

, donde los P_i son vértices del triángulo y (α, β, γ) son las coordenadas baricéntricas del punto P. Si P pertenece al triángulo, $0 \leq \alpha, \beta, \gamma \leq 1$.

Como una parametrización es un mapeo lineal por definición [HLS], si Φ es una parametrización, Φ puede aplicarse a la ecuación anterior, quedando:

$$\Phi(P) = \alpha\Phi(A) + \beta\Phi(B) + \gamma\Phi(C)$$

Un vector también puede ser expresado en coordenadas baricéntricas, por lo que para cada par de vectores $(\mathbf{S}, \mathbf{T})$ de un triángulo se puede obtener un par $(\Phi(\mathbf{S}), \Phi(\mathbf{T}))$, como lo muestra la Figura 2.14.

Para optimizar la parametrización del parche, en cada triángulo se calculan las siguientes diferencias:

$$\mathbf{d}_S = \Phi(\mathbf{S}) - \hat{\mathbf{s}}, \mathbf{d}_T = \Phi(\mathbf{T}) - \hat{\mathbf{t}}$$

donde $(\hat{\mathbf{s}}, \hat{\mathbf{t}})$ son los ejes del plano de la textura trasladados al centro de cada triángulo. Como se desea que la textura quede alineada con las direcciones que se especificaron, se establece el problema de optimización donde se busca las parametrización que minimiza el siguiente funcional:

$$\sum ||\mathbf{d}_S||^2 + ||\mathbf{d}_T||^2$$

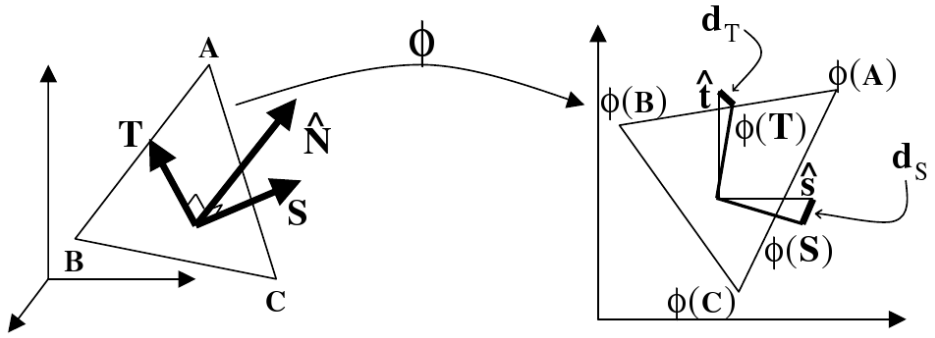


Figura 2.14: La parametrización Φ del triángulo se puede aplicar a los vectores definidos en el plano del triángulo gracias a las coordenadas baricéntricas. Fuente: Praun et al. [PFH].

Utilizando coordenadas baricéntricas, es posible expresar el problema como un sistema de ecuaciones lineales, donde cada ecuación representa un vector de dirección asociado a un triángulo, y por lo tanto tiene solamente 3 variables, las coordenadas baricéntricas de dicho vector. Entonces, el sistema tiene en total $6n$ variables, donde n es el número de triángulos del problema.

Como la matriz que expresa el problema es dispersa, y se tiene una solución inicial para el problema de optimización (la parametrización inicial del parche), los autores de Lapped Textures sugieren utilizar un método iterativo para resolver el problema: el *gradiente conjugado* [She94].

2.3.3. Uniendo TAMs y Lapped Textures: Achurado en Tiempo Real

Para el algoritmo de achurado en tiempo real, se necesitan dos consideraciones:

1. Se utiliza una textura de los TAMs como textura base para Lapped Textures, el resultado es una superficie con una parametrización tal que sobre la superficie hay líneas de achurado. Gracias a las características de los TAMs, la parametrización obtenida es válida para cualquiera de las texturas de achurado dentro del mismo TAM, por lo que es posible utilizar la parametrización con el TAM completo.

2. Se utiliza un campo de direcciones calculado directamente a partir de las direcciones principales o bien suavizado de acuerdo a lo visto en el algoritmo de Hertzmann y Zorin [HZ].

Lo anterior entrega la textura de achurado dispuesta sobre cada parche en la superficie. Respecto al desempeño en tiempo real, graficar una superficie con una textura tradicional no presenta diferencia alguna con graficarla con una textura perteneciente a un TAM. Por lo tanto, si es posible graficar una escena con texturas en tiempo real, será posible graficar una escena con achurado en tiempo real. La limitación de este planteamiento tan simple es que sólo se utiliza una textura de achurado en cada triángulo, provocando que entre cada arista se pueda producir un cambio muy brusco de textura, como se muestra en la Figura 2.15.

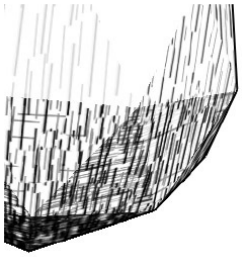


Figura 2.15: Al utilizar una textura de achurado distinta en cada triángulo, hay problemas de continuidad entre las texturas. Fuente: Praun et al. [PHWF].

La solución es utilizar más de una textura, en particular tres: las tres que correspondan a las intensidades de iluminación calculadas en cada vértice de los triángulos. Para implementar esto se pueden mezclar con un shader en la GPU las texturas 2D al momento de dibujar o bien convertir el TAM en una textura 3D de modo que el hardware haga la mezcla automáticamente. En ambos casos la mezcla es rápida y no interfiere con el desempeño en tiempo real, ya que el hardware actual soporta tanto el uso de múltiples texturas como el uso de texturas 3D.

El resultado final del algoritmo se observa en la Figura 2.16.

Figura 2.16: Técnica de Achurado en tiempo real aplicada a un modelo tridimensional de una mano. Fuente: Webb et al. [WPFH02].

2.4. Animación de Objetos

En la animación tradicional se utilizan varios cuadros de animación que se superponen rápidamente para dar la sensación de movimiento en un objeto. Con los modelos 3D se puede realizar el mismo procedimiento: se pueden tener distintas mallas para distintos instantes de tiempo y se pueden superponer con el fin de crear una animación.

Figura 2.17: Keyframes de una animación para un modelo 3D. En este caso, se trata de la animación que representa a un personaje corriendo.

La ventaja que se posee en la animación por computadora es que además de los cuadros de animación originales, llamados *keyframes*, es posible reconstruir cuadros intermedios, simplemente llamados *frames*. Estos cuadros intermedios permiten que la animación sea suave: por ejemplo, para una animación de un brazo en movimiento, se pueden utilizar dos keyframes: uno con el brazo en una posición y otro con el brazo en otra. Luego, es tarea de la computadora interpretar ambos frames y crear el movimiento del brazo.

Existen dos técnicas para expresar animaciones de superficies 3D, en particular mallas de triángulos. Estas técnicas son *Morph Target Animation* y *Skinning*.

2.4.1. Morph Target Animation

Este tipo de animación también es conocido como *Vertex Animation*, un nombre que es más explícito respecto a la implementación. Al hablar de morph target se habla de objetivos de transformación, correspondientes a los keyframes. Los keyframes que determinan la animación se consideran objetivos, y los frames entre cada keyframe corresponden a una transformación entre los dos objetivos que lo rodean. Es decir, se realiza una interpolación directa entre los keyframes. Dicha interpolación se realiza en los vértices, motivo por el que la técnica tiene el nombre Vertex Animation.

De lo anterior, se puede concluir que para animar un objeto utilizando Morph Target, se poseen tantas mallas representativas del objeto como keyframes, donde la malla M_i para el keyframe k_i tiene la misma topología que M_j para el keyframe k_j. Por ejemplo, la Figura 2.17 muestra una pequeña animación de 4 keyframes: cada keyframe es una malla distinta, pero todas comparten la misma topología, por lo que la interpolación de sus vértices adquiere sentido.

La interpolación considera las posiciones y los vectores normales de ambas mallas para una diferencia de tiempo t normalizado entre dos keyframes, tal que $t \in [0,1]$:

$$v_i = (1{,}0 - t) * v_{im_1} + t * v_{im_2}$$

para los vértices y:

$$n_i = (1{,}0 - t) * n_{im_1} + t * n_{im_2}$$

para las normales del modelo, que luego de la interpolación deben ser normalizadas. Usualmente las coordenadas de textura se mantienen intactas al igual que los otros atributos de la malla.

Pros

- Es una técnica sencilla de implementar, ya que básicamente se tiene un conjunto de mallas estáticas en las cuales se realizan interpolaciones lineales de puntos y vectores.
- Su especificación es simple, por lo que es soportada por una gran cantidad de motores y aplicaciones 3D.

Contras

- No permite flexibilidad para crear o modificar animaciones, lo que evita su uso en animaciones complejas, ya que se trabaja directamente con los vértices, cuya cantidad puede ser del orden de miles o millones.
- Es demasiado costosa en términos de memoria, ya que la malla debe replicarse una vez por cada cuadro de la animación.

2.4.2. Skinning

Skinning[Lan98, KCvO07] consiste en tratar a la malla de triángulos como la piel que envuelve a un esqueleto, de modo que si el esqueleto se mueve, la piel se deforma para adecuarse a ese movimiento.

Para representar una animación esqueletal se utiliza un sistema tradicional de cuadros, donde cada cuadro contiene una *pose* para el esqueleto y no una malla completa, lo cual facilita enormemente la manipulación, almacenamiento y creación de animaciones, ya que un esqueleto tiene un orden de decenas de huesos (en contraste con miles o millones de vértices). Lo que se hace para animar el modelo es realizar una interpolación de las poses en distintos tiempos de la animación y luego actualizar la malla de acuerdo a la pose final del esqueleto.

Skinning también es conocida como *Vertex Blending*, nombre que se deriva del uso de una fórmula de ponderación de vértices con pesos para calcular la deformación de la malla: cada vértice v_i de la malla se asocia a uno o más huesos del esqueleto, teniendo para cada hueso un peso asociado w_j. La suma de todos los pesos asociados a un vértice debe ser 1. La posición y orientación de un hueso se representa mediante una matriz de transformación de 4×4.

La malla tiene una forma inicial, asociada a una pose inicial del esqueleto, llamada *bind-pose*. Además existe la malla deformada, donde se almacena la deformación producida por la animación del esqueleto. La posición de un vértice respecto al esqueleto se define como:

$$v'_i = \sum w_{i,j} M_j v_i$$

donde v'_i es el vértice en su posición final, M_j es la matriz de transformación asociada al hueso j, $w_{i,j}$ es el peso del hueso j asociado al vértice inicial v_i. Para las normales la ecuación es similar:

$$n'_i = \sum w_{i,j} M_j^{-1^T} n_i$$

Se usa la matriz inversa transpuesta de M_j porque sólo se desea que se apliquen transformaciones rígidas a los vectores normales [Tur90].

Las ecuaciones de Skinning muestran que cada hueso tiene una matriz de transformación M independiente, compuesta por una orientación y una traslación. En la práctica, los esqueletos se suelen representar como un árbol jerárquico de huesos, donde la transformación de un hueso es afectada por la transformación de su hueso padre en caso de no ser el hueso raíz. Así, antes de realizar la animación de la malla, primero se calculan las matrices de transformación de cada hueso de acuerdo a la jerarquía. Un esqueleto típico se muestra en la Figura 2.18.

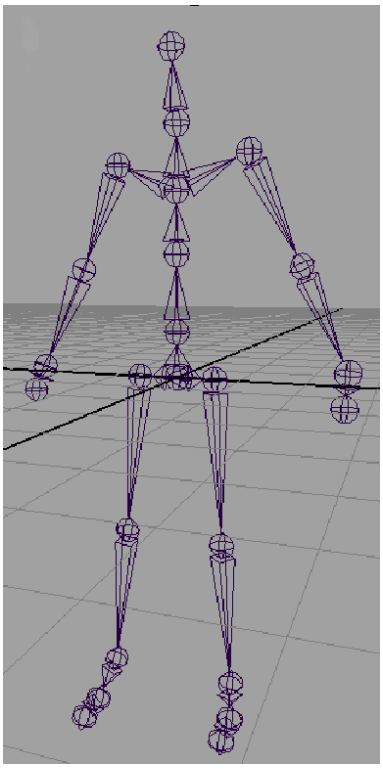


Figura 2.18: Representación gráfica de un esqueleto jerárquico. Imagen de [DR05].

Para un hueso j se define su matriz local L_j como:

$$L_j = \begin{bmatrix} R(\theta_j) & T(t_j) \\ 0 & 1 \end{bmatrix}$$

donde $R(\theta_j)$ es una matriz de 3×3 que representa la orientación del hueso j y $T(t_j)$ es el vector que representa la traslación del hueso j. Para el hueso raíz $M_{root} = L_{root}$, mientras que para un hueso no raíz, $M_j = M_{parent} * L_j$. Esta forma de animar el esqueleto es conocida como *Forward Kinematics*, debido a que se avanza desde la raíz del esqueleto hacia sus huesos hijos, anidando las transformaciones que afectan a cada hueso. Con este esquema, una animación se puede expresar como un conjunto de ángulos y traslaciones locales para cada hueso, que luego es transformado en un conjunto de matrices que representan la pose del esqueleto.

Pros

- Las animaciones se gestionan utilizando poses, facilitando la creación y modificación de animaciones.
- Sólo se necesita dos instancias de la malla, independiente del número de keyframes, por lo que el uso de memoria es óptimo.
- Una pose es aplicable a más de un esqueleto, permitiendo que las animaciones sean utilizadas en distintos objetos o en diferentes versiones del mismo objeto.

Contras

- No existe una forma estándar de interpretar los esqueletos. En general, distintas aplicaciones de modelado 3D interpretan de modos diferentes o tienen diferentes estructuras de datos para representar esqueletos.
- Debido a lo anterior, la implementación es muy compleja.

2.4.3. ¿Cuál esquema de animación utilizar?

En la actualidad, la técnica más utilizada es Skinning, debido a que supera a Morph Target en todos los sentidos, excepto en la facilidad de implementación, aunque dicho costo sólo se paga una vez. Además, es posible encontrar implementaciones de calidad en la red, con licencias de software libre.

Ahora bien, respecto al tema de la tesis, el esquema de animación utilizado es independiente de la técnica de achurado, porque se trabaja con el objeto ya animado, en una etapa posterior a la obtención del frame actual de animación.

2.5. Oportunidad: Achurado en Objetos Animados

2.5.1. El Problema Actual

El algoritmo de achurado en tiempo real funciona en base al precálculo de datos: tanto las líneas de achurado como las intensidades de iluminación representadas por las líneas se encuentran establecidas antes de realizar la visualización. Para un objeto estático esto no es problema, pues ninguno de esos datos cambia con el tiempo.

El caso de las coordenadas de textura es distinto. En las técnicas de visualización tradicionales, las coordenadas de textura no cambian durante la animación, ya que permiten aplicar imágenes sobre la superficie que representan, por ejemplo, la apariencia de ropa o de un tatuaje. Si la piel de un tatuaje se estira, el tatuaje también lo hace, si un pedazo de tela se estira, el espaciamiento entre las fibras también aumenta. Entonces, en la mayoría

de los casos es correcto que las coordenadas de textura no cambien. Sin embargo, ya que el achurado en tiempo real depende de las coordenadas de textura, es necesario analizar qué sucede ante una deformación en la geometría.

El achurado requiere dos tipos de datos: la dirección de las líneas y la densidad. La densidad se determina a partir de un modelo de iluminación simplificado que solamente requiere conocer la normal de la superficie. La normal está considerada dentro de las ecuaciones de Skinning, por lo que ya se dispone de ese dato y es posible volver a calcular la densidad de las líneas.

El problema aparece cuando se deforman los triángulos de la malla:

- Como los triángulos no tienen la misma área, ni los mismos ángulos interiores, las líneas que se desplegaban sobre ellos presentan artefactos visuales: se ensanchan y/o alargan de acuerdo a la deformación de los triángulos que las contienen. Esto implicaría tener que ejecutar el algoritmo Lapped Textures nuevamente.
- Las direcciones principales cambian, por lo que la dirección original de las líneas de achurado no representa necesariamente la geometría de la nueva forma del objeto. Esto implicaría tener que estimar las direcciones principales de la superficie nuevamente.

Por lo tanto, al deformarse la superficie, los datos precalculados son invalidados. Un recálculo total de ellos es imposible en tiempo real, por lo que se vuelve necesario un algoritmo para precalcular la mayor cantidad de datos posibles y recalcular solamente lo necesario.

2.5.2. Análisis Inicial de una Solución en Espacio Objeto

Considerando que existe una solución para espacio imagen, ¿por qué es necesaria una solución en espacio objeto? A continuación se enumeran distintas razones:

- Se aprovecha mejor el hardware gráfico, incurriendo en un menor *overhead* al aplicar la técnica.
- No se realizan cálculos redundantes, lo cual sí sucede en espacio imagen ya que las curvaturas y direcciones principales se recalculan en cada píxel a cada momento.
- La técnica se puede mezclar con otras técnicas de visualización de manera más transparente para el implementador.
- Se puede utilizar información de la superficie, con el fin de determinar una mejor dirección y densidad de las líneas de achurado.

Ahora se enumeran las dificultades que tiene una solución en espacio objeto:

- La técnica de Lapped Textures está definida para una malla estática. Si la malla presenta deformación, la creación de los parches también debiese considerar la deformación que afectará a los parches.

- Las propiedades geométricas de la superficie cambian después de la deformación. Para establecer la dirección de las líneas, es necesario recalcularlas, pero este proceso es demasiado costoso, siendo imposible aplicarlo en tiempo real (aunque existen aproximaciones como la propuesta por Kalogerakis et al. [KNS+08]). En el caso de que sí fuese posible, la parametrización necesaria (etapa de optimización de Lapped Textures) tampoco se puede llevar a cabo en tiempo real.

- Aun si todo lo necesario se pudiese calcular en tiempo real, existe una última dificultad que impediría el mejor desempeño posible: se usaría demasiado ancho de banda del hardware gráfico para re-enviar constantemente la malla deformada y su parametrización. El orden de tamaño de la parametrización es cercano a $\frac{2}{3}$ del tamaño de la malla, por lo que por cada objeto de la escena se estarían transfiriendo casi el doble de datos a la GPU.

- No se tiene certeza de que la animación de un objeto mantenga en un estado correcto la topología de la malla.

Estas dificultades se pueden resolver si se considera el supuesto de conocer la animación en su totalidad. En la actualidad, la mayoría de las animaciones que se utilizan en aplicaciones interactivas se encuentran predeterminadas. Gracias a esto se puede plantear un algoritmo que precalcula la mayor cantidad de datos posibles, permitiendo graficar ese tipo de animaciones con una estética de achurado.

Capítulo 3

Formulación de Herramientas y Datos de Trabajo

En el Capítulo de Antecedentes se habló de la información que posee una malla de triángulos: vértices, aristas y triángulos, en conjunto con algunas restricciones, en particular que una arista no puede ser compartida más que por dos triángulos. Para algunas aplicaciones almacenar directamente esa información es más que suficiente, por ejemplo, en el rendering tradicional sólo se necesita la lista de triángulos y vértices. Por otro lado, técnicas como Lapped Textures requieren más información. En particular se necesita realizar consultas sobre la malla, como:

- ¿Cuáles son los triángulos vecinos de un triángulo dado?
- ¿Cuáles son los triángulos que comparten un vértice?
- Dada una arista, ¿cuáles son los triángulos que la comparten?

En la Introducción se mencionó que, en lo que respecta a implementación, la tesis continúa un framework realizado como trabajo previo, llamado Zahir [Gra08]. Sin embargo, este framework no permitía responder de manera eficiente estas preguntas, de manera que se vuelve necesario realizar mejoras en la estructura de datos.

3.1. Estructura Halfedge para Mallas de Triángulos

Existen diferentes estructuras de datos que entregan la información requerida. Un estudio detallado realizado por Kettner [Ket98] describe dichas alternativas, encontrando que la mejor estructura de datos para responder esas preguntas es *half-edge* [WE85]. Esta estructura, que almacena "medias aristas", como se observa en la Figura 3.1, guarda la siguiente información por cada half-edge: el vértice que inicia la half-edge, el vértice que la finaliza, la cara a la que pertenece, y la half-edge que va en dirección opuesta.

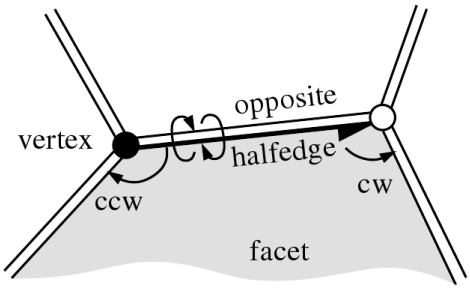


Figura 3.1: Estructura de datos half-edge. La imagen muestra la información conocida por una half-edge. Fuente: [Ket98].

La estructura half-edge es difícil de implementar. Debido a esto, se optó por utilizar una implementación existente. Las alternativas son dos: OpenMesh[BSBK02] y CGAL[FGK+00]. Ambas están disponibles para el lenguaje C++, son software libre y multiplataforma.

3.1.1. OpenMesh

Pros

- Licencia LGPL en todas sus componentes, lo que permite utilizarla en aplicaciones propietarias y de código abierto.
- Estructura basada en arreglos dinámicos, lo que permite almacenar la malla de una manera óptima en términos de acceso a los elementos de la malla.
- Mecanismo de propiedades dinámicas para la malla, lo que permite añadir propiedades a las mallas sin modificar el código fuente de la biblioteca.
- Enfocada a aplicaciones en tiempo de ejecución/tiempo real.

Contras

- Poca documentación, dificultando el uso.
- Incluye pocos algoritmos implementados (suavizamiento de mallas, nivel de detalle mediante mallas progresivas, subdivisión).

3.1.2. CGAL

Pros

- Incluye una gran cantidad de algoritmos implementados, incluyendo técnicas de parametrización.

- Tiene especial énfasis en la robustez, en especial en lo que a precisión de números de punto flotante se refiere.
- Existe documentación y ejemplos de uso.

Contras

- Licencia a la medida, sólo algunas componentes son de licencia de código abierto.
- Estructura basada en listas, lo que impide un acceso óptimo a las propiedades de la malla.
- Enfocada a aplicaciones de procesamiento geométrico, no necesariamente tiempo real.

3.1.3. Decisión

La biblioteca elegida fue OpenMesh. La justificación de la decisión es la siguiente: aunque CGAL incluye algunas técnicas de parametrización, éstas son incompatibles con Lapped Textures. Además, se prefiere una biblioteca que tenga mejor desempeño para aplicaciones en tiempo de ejecución o tiempo real, y que además tenga una licencia unificada y compatible con la libre disposición del trabajo realizado. OpenMesh supera estas dificultades, a lo cual se suma el uso de propiedades dinámicas de la malla y los algoritmos que ya incluye (que son pocos, pero útiles en el contexto de la tesis). En particular el uso de propiedades dinámicas es importante porque, en el contexto de la tesis, las superficies tienen propiedades relativas a las gráfica (como son los materiales y los tonal art maps del achurado) y a la animación (los cuadros de cada animación).

3.2. Biblioteca de Álgebra Lineal

Tanto OpenMesh como CGAL incluyen sus propias clases de vectores y matrices. Estas clases proveen operaciones como aritmética de vectores y matrices y productos vectoriales, y en la mayoría de las situaciones, con éstas basta.

Ambas bibliotecas también proveen mecanismos que permiten utilizar otras estructuras de datos como vectores y matrices, por lo que, considerando las técnicas que se requieren para la tesis (en particular el uso de coordenadas baricéntricas y la optimización lineal de sistemas de ecuaciones), se evaluaron dos bibliotecas de álgebra lineal: Eigen [JG] y Boost Linear Algebra [WK]. Estas dos bibliotecas siguen el mismo paradigma de implementación y diseño que OpenMesh y CGAL, por lo que son integrables entre sí.

3.2.1. Eigen

Pros

- Utiliza la misma licencia que OpenMesh (LGPL).
- Está enfocada en la versatilidad, elegancia y flexibilidad de implementación.
- La implementación otorga el mejor desempeño posible para operaciones aritméticas, al utilizar características avanzadas de la CPU (instrucciones SIMD, SSE2).
- Incluye un sub-paquete gráfico con implementaciones de Hiperplanos, *Quaternions*, Transformaciones homogéneas, etc.
- Es una biblioteca consciente de los problemas de precisión de los números de punto flotante.

Contra

- Es una biblioteca joven, sin gran trayectoria, por lo que su API no es final y es susceptible a cambios.

3.2.2. Boost Linear Algebra

Pros

- Es una biblioteca con gran trayectoria y soporte comunitario.
- Se adhiere completamente al estándar BLAS (Basic Linear Algebra Subprograms).

Contras

- No incluye un paquete gráfico con clases utilitarias.
- No se obtiene el mejor desempeño posible, la biblioteca no tiene ese fin.

3.2.3. Decisión

Para los propósitos de la tesis, la biblioteca Eigen incluye todo lo necesario para poner manos a la obra. Está enfocada a un uso más práctico de las técnicas de álgebra lineal, y el hecho de que ya incluya varias clases ad-hoc para computación gráfica indica que la biblioteca está pensada para aplicaciones como la planteada en la tesis.

3.3. Estimación de Curvatura

Uno de los algoritmos necesarios para la implementación del achurado es la estimación de la curvatura de la superficie aproximada por la malla de triángulos. El algoritmo de Szymon Rusinkiewicz [Rus04] es el más utilizado hasta el día de hoy, ya que es el que produce mejores resultados que otros algoritmos y además existe una implementación disponible. Esta implementación se enmarca dentro de una biblioteca ad-hoc de Rusinkiewicz, `trimesh2`[Rus]. Esta biblioteca tiene licencia GPL (que es incompatible con LGPL de manera directa) y dicha implementación, en términos de integración de código, no es utilizable junto con Eigen/OpenMesh. La Figura 3.2 muestra un modelo 3D donde los colores representan distintos valores de la curvatura, calculada utilizando trimesh2.

Figura 3.2: Representación gráfica de la estimación de curvatura en el conejo de Stanford. Imagen obtenida utilizando el software RTSC[DFR04], que estima las curvaturas mediante la biblioteca trimesh2 [Rus].

Entonces, en vez de integrar trimesh2 con el framework de la tesis, se realizó una implementación compatible con Eigen/OpenMesh basada en la de trimesh2. Para esto se hicieron los siguientes pasos:

1. Se estudiaron las clases y métodos de álgebra lineal utilizados por el algoritmo. En particular se encontraron vectores de 3 y 4 dimensiones, matrices de tamaño arbitrario y operaciones entre ellas, incluyendo inversión y transposición, y resolución de sistemas de ecuaciones utilizando descomposición LDLT, todo esto para poder realizar una minimización de mínimos cuadrados presentada en el artículo que define el algoritmo. Estas clases y métodos fueron reemplazadas por las clases propias de Eigen.

2. Se estudió el acceso a datos en la malla de triángulos de trimesh2. Las operaciones que realiza el algoritmo son las siguientes: recorrido de todos los vértices de la malla,

recorrido de todos los triángulos de la malla, consulta de los vértices de un triángulo, consulta de los triángulos que comparten un vértice. Dichos recorridos y consultas son soportados en su totalidad por OpenMesh, por lo que fue posible reemplazar la estructura de datos de trimesh2.

Realizar una conversión del algoritmo en vez de una implementación desde cero permite asegurar que el resultado de la estimación de curvatura es robusto y que la implementación es estable. Además, al ser aplicable en OpenMesh, es una contribución a la comunidad, ya que la estimación de curvatura es un problema que está presente en distintas aplicaciones. Al ser una derivación del código original, se solicitó autorización a Szymon Rusinkiewicz para distribuir esta nueva versión con licencia LGPL, autorización que fue concedida mediante e-mail.

Con esto, el framework Zahir se ha actualizado utilizando una estructura de datos robusta para mallas de triángulos y con operaciones de estimación de curvatura.

3.4. Datos de Prueba

A diferencia de los objetos 3D estáticos, que se pueden obtener de manera directa en la red (gracias a lugares como el repositorio de scans 3D de Stanford), en el caso de objetos animados no existen sets de datos, y menos aún sets estándares que permitan comparar técnicas. Sin embargo, gracias al trabajo de Vlasic et al. [VBMP], es posible contar con un set de animaciones, tanto en formato Morph Target como Skinning, de modelos 3D de humanos realizando acciones como caminar, bailar y saltar.

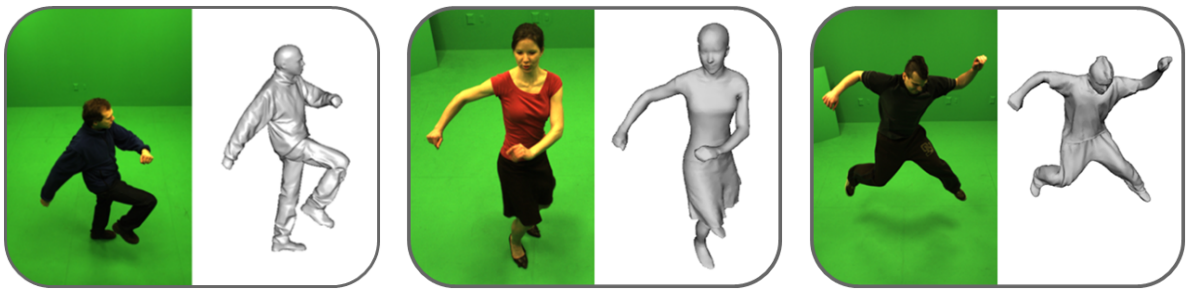


Figura 3.3: Algunas capturas de las animaciones del set de datos de Vlasic et. al [VBMP], obtenidas desde su sitio web.

Entre algunas características de este set de datos se encuentran:

- Contiene 10 animaciones, cada una con una cantidad de keyframes cercana a 150.
- Cada animación incluye tanto las mallas para Morph Target, como la animación para un esqueleto. Sin embargo, el formato de esqueleto no está documentado, por lo que es muy difícil implementar la lectura de dicho formato.
- Las mallas tienen una cantidad moderada de polígonos, cercana a 19000 en promedio, con 9900 vértices en promedio.

- Las mallas no incluyen coordenadas de textura ni material.
- Las animaciones se generaron utilizando captura de imágenes de personas realizando esos actos. Debido a eso, las mallas contienen también el movimiento de sus ropas, pero la malla es una sola tanto para la piel como para la ropa, por lo que una de las desventajas de las animaciones es que poseen una gran cantidad de "micromovimientos", que representan el movimiento que tenían sus ropas al realizar las acciones.

El set de datos es de libre descarga en `http://people.csail.mit.edu/drdaniel/mesh_animation/`. La Figura 3.3 muestra tres capturas de ejemplo, extraídas directamente desde el sitio web de los datos. La Figura 3.4 muestra cuatro frames graficados utilizando Zahir.

Figura 3.4: Rendering de una animación de Vlasic et al [VBMP], utilizando OpenMesh/Eigen y Zahir.

Ahora bien, el set de datos mencionado, al ser muy rico en datos y presentar mallas irregulares, puede ser contraproducente al no tener datos más sencillos con los cuales probar el algoritmo. Para ello, también se realizarán pruebas con datos de captura de movimiento de ropa, en particular de pantalones, gracias a una animación puesta a disposición general por White et al. [WCF07]. Este set de datos presenta figuras más sencillas que tienen un mallado regular. El set completo se observa en la Figura 3.5.

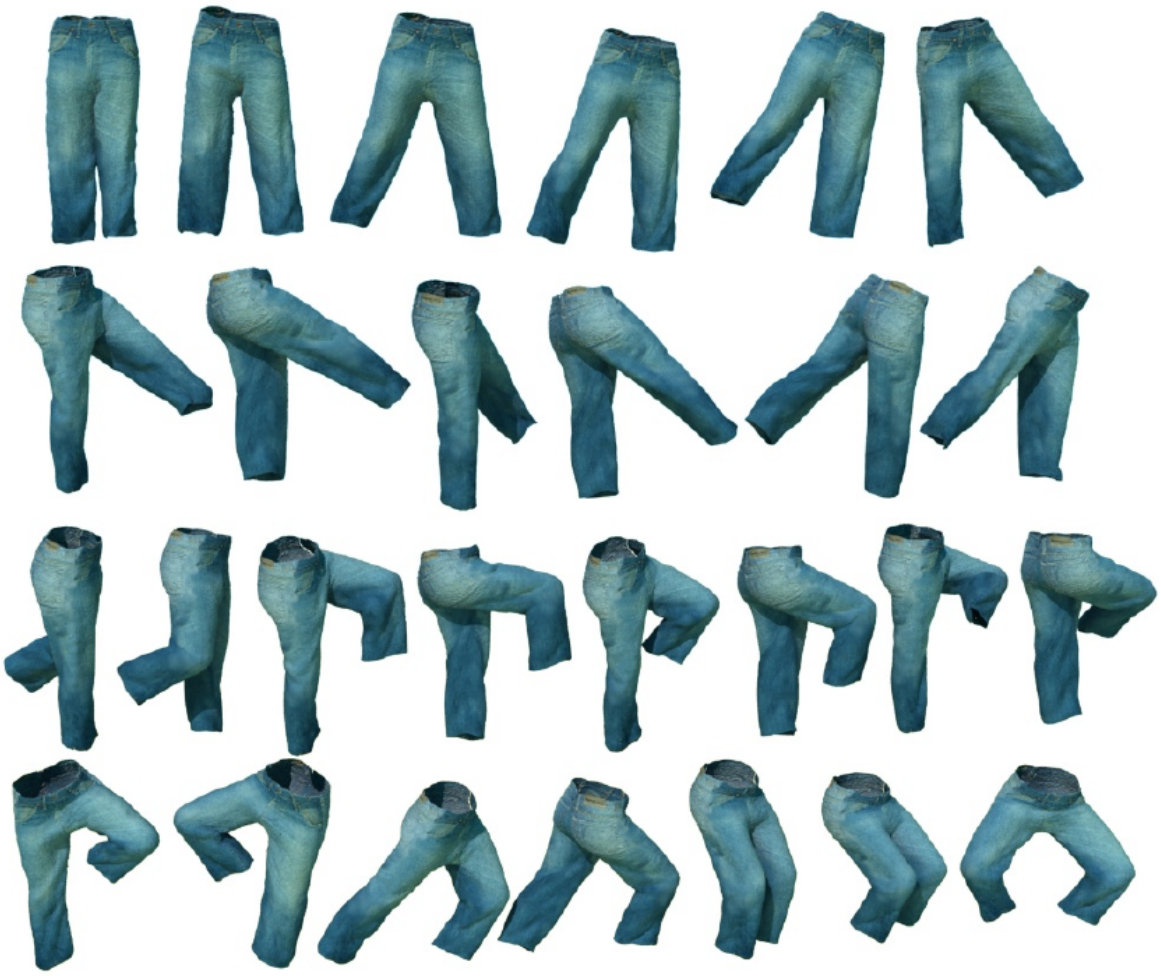


Figura 3.5: Keyframes de la animación de captura de datos de pantalones de White et al. [WCF07]. Fuente: [WCF07]

Capítulo 4

Implementación de Achurado para Objetos Estáticos

En este Capítulo se describe la implementación del algoritmo original de achurado de Praun et al. [PHWF]. En este algoritmo, el achurado de un objeto tiene dos etapas: una etapa de preprocesamiento y una etapa de rendering. La etapa de preprocesamiento genera un conjunto de coordenadas de textura para el objeto, y la etapa de rendering utiliza esas coordenadas para elegir las texturas de achurado que se sobrepondrán en la superficie del objeto.

4.1. Parametrización Local mediante Lapped Textures

La Figura 4.1 muestra las distintas componentes del sistema que intervienen en la implementación de Lapped Textures. Se distinguen tres componentes: OpenMesh (algoritmos y estructuras base), el framework Zahir y la tesis como tal. De acuerdo a lo visto en el Capítulo anterior, OpenMesh provee la estructura de datos de mallas de triángulos, junto con operaciones como suavizamiento de mallas; Zahir especializa las mallas de OpenMesh agregando estimación de curvatura; y la tesis, por su parte, contribuye con la implementación de Lapped Textures y del optimizador de campos de direcciones.

Lapped Textures recibe como input una superficie y entrega como resultado un conjunto de parches, que son representados como sub-superficies: mallas de triángulos donde los elementos tienen los mismos atributos de la malla original, además de la información que permite asociar elementos de la sub-malla con la malla original. Este esquema permite, por ejemplo, saber a cuál triángulo de la malla original pertenece un vértice de la sub-malla. Concretamente, tener la información de asociatividad entre elementos de la sub-malla y la malla original permite almacenar y procesar el campo de direcciones una sola vez, sin necesidad de replicarlo localmente en cada sub-malla. Como se conoce el *mapping* entre un triángulo de la sub-malla y el triángulo de la malla original, se puede acceder de manera semidirecta a los vectores de dirección asociados a cada triángulo de la sub-malla.

Antes de realizar la parametrización de cada submalla, se prepara el campo de direcciones. El campo se prepara sin tener conocimiento de los parches, ya que éstos no existen todavía.

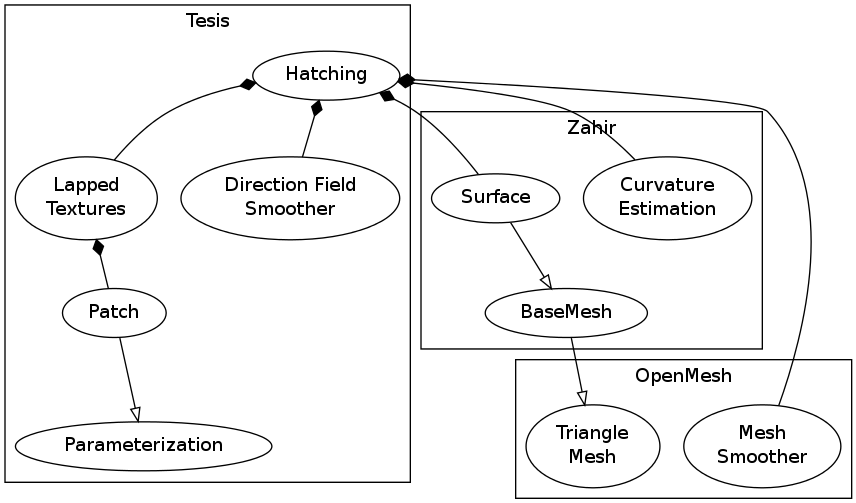


Figura 4.1: Esquema de diseño de componentes del sistema.

Luego se lleva a cabo la parametrización local sobre cada submalla. Finalmente se optimiza la parametrización de cada submalla.

4.1.1. Preparación de la Malla y del Campo de Direcciones

Cuando Lapped Textures recibe la superficie, se realizan las siguientes operaciones:

1. Opcionalmente, se suaviza la malla (utilizando el `Mesh Smoother` de OpenMesh). Las posiciones originales de los vértices son almacenadas para su posterior reposición. El suavizamiento elimina detalles pequeños o específicos en la superficie, que pueden perturbar la estimación de curvaturas en la superficie. Si bien los artículos de achurado se refieren solamente a "suavizar la superficie", usualmente lo más utilizado es *Laplacian Smoothing*, un suavizamiento local sencillo de implementar donde la posición de cada vértice es ponderada entre las posiciones de los vértices vecinos a través de las aristas que salen de él:

   $$x_i = \frac{1}{N} \sum_{j=0}^{N} x_j$$

   Al requerirse la información asociada al *1-ring* de un vértice (información que ya se dispone en las estructuras de datos típicas), la implementación del suavizamiento es directa.

2. Se estiman las normales por vértice, utilizando una ponderación de las normales que tienen las caras que comparten cada vértice [Max99].

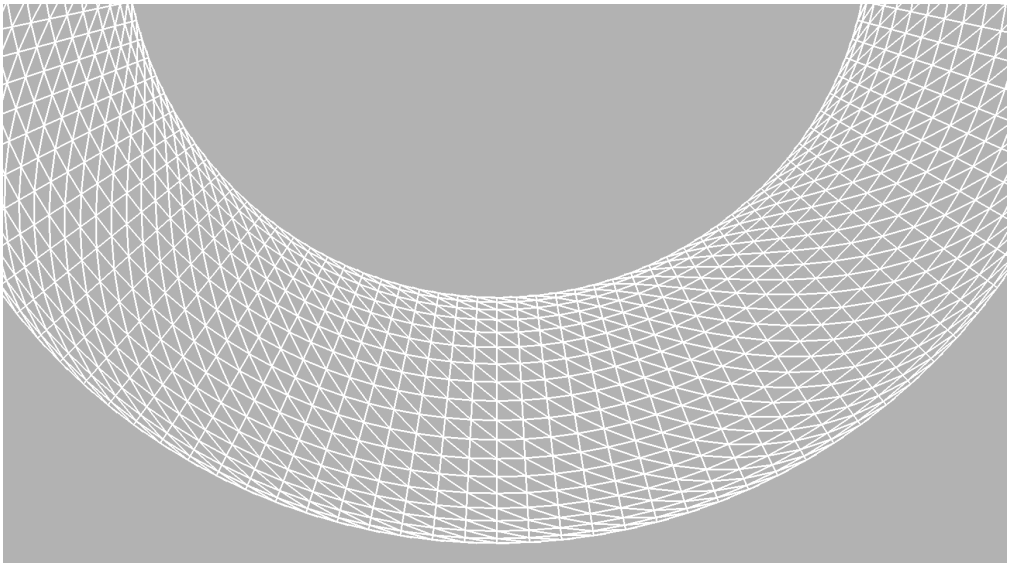
3. Se estiman las curvaturas de cada cara y cada vértice de la malla [Rus04], para obtener las *direcciones principales*. Estas direcciones generan el *campo de direcciones* inicial en la superficie.

4. Se restauran las posiciones originales de los vértices en la superficie.

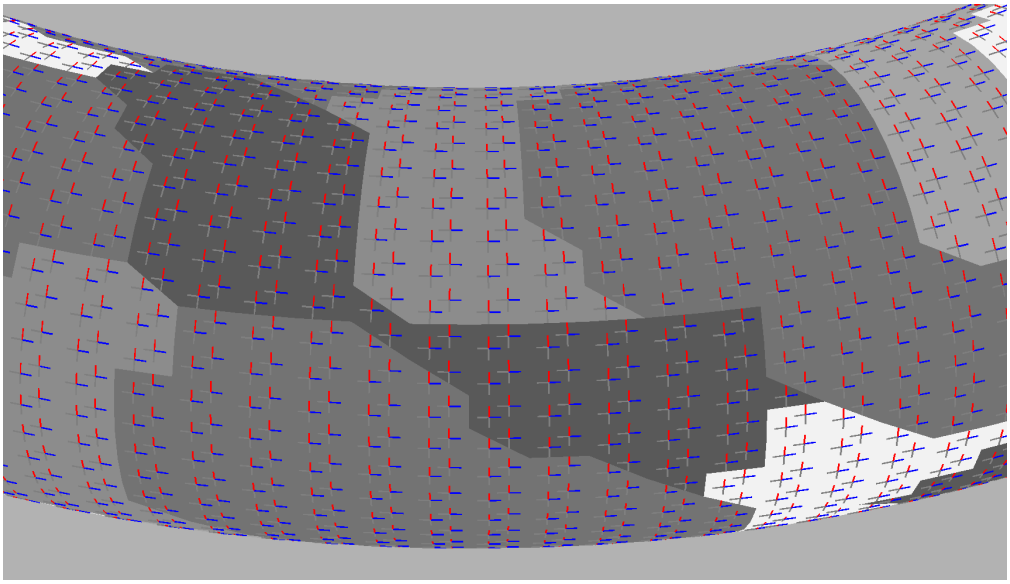
5. Se re-proyectan los vectores de dirección sobre la superficie, puesto que se requiere que el campo de direcciones sea tangencial.

6. Opcionalmente, se suaviza el campo de direcciones utilizando el algoritmo de Hertzmann y Zorin [HZ]. Este suavizamiento entrega lo que Hertzmann y Zorin denominan *cross field*.

Al finalizar este proceso se tiene la superficie original, con un campo de direcciones tangencial apto para el achurado. Ver Figura 4.2.



(a) Wireframe de un toroide.



(b) Direcciones Principales.

Figura 4.2: Detalle de un toroide con una malla de triángulos regular. La segunda imagen muestra además la segmentación de la superficie.

4.1.2. Segmentación de la Superficie

La segmentación de la superficie recibe como input la superficie misma, el campo de direcciones asociado, una forma de parche y un tamaño de parche. Para achurado, la forma del parche se asume circular, ya que esto simplifica la implementación. El algoritmo tiene los siguientes pasos:

1. Se crea una cola FIFO con los triángulos de la malla. Dicha cola se ordena de acuerdo a un criterio no determinado, dependiente de la aplicación o de la superficie que se quiere procesar. Este paso es opcional.
2. De la cola se saca el primer elemento, y a partir de él se hace crecer un parche. Cada cara que pertenece al parche es marcada como visitada.
3. Una vez construido el parche, se repite el paso anterior. Si la cara que sigue en la cola ya fue agregada a un parche, se descarta y se sigue en la cola hasta encontrar una cara que no pertenece a ningún parche.

El proceso de crecimiento de un parche se realiza de la siguiente manera:

1. Dada una cara, ésta se posiciona en el centro del parche. En particular, su centroide es ubicado en el centro. El plano definido por la cara será utilizado como plano para las coordenadas de textura del parche, como dominio de parametrización. Los ejes x e y del plano del triángulo se hacen coincidir con sus campos de direcciones mediante transformaciones de rotación aplicadas al triángulo dentro de su propio plano.
2. Se crea una lista de candidatos para ser agregados al parche. Para el triángulo inicial, los tres primeros candidatos son los vecinos de ese triángulo.
3. Se ordena la lista de candidatos de acuerdo a su distancia hasta el centro del parche. La distancia se calcula como sigue: dada una arista del candidato que ya pertenezca al parche, se utiliza la menor de tres distancias: las distancias a los vértices de la arista, y la distancia desde el centro de la parametrización hasta su proyección a la línea recta que contiene a la arista.
4. Para que un candidato sea agregado al parche debe cumplir las siguientes condiciones:
   - Su ángulo diedro con el parche no puede ser mayor a un umbral predeterminado. Esta condición permite que los parches no crucen zonas de alta curvatura.
   - Su proyección en el plano de la textura no debe intersectarse con la proyección actual del parche. Esta condición permite que el parche tenga topología de disco.
5. El candidato más cercano se agrega al parche, y sus vecinos que no sean candidatos o no estén dentro del parche son agregados como candidatos.
6. El proceso se repite hasta que se cumpla una de las siguientes condiciones:

- El parche alcanza la forma y el tamaño establecido. Para asegurar que el parche tiene una forma circular, cada vez que se agrega una cara se calcula una circunferencia que contenga a la parametrización. Se utiliza el algoritmo de Ritter [Rit90], que no entrega la mínima circunferencia, pero sí una cuyo *fit* de la parametrización es suficiente para restringir el tamaño del parche[1].
- No hay más triángulos en las vecindades del parche que no pertenezcan a otros parches.

Luego de segmentar la superficie, cada parche contiene una copia de la parte de la superficie que le corresponde, en términos de posiciones y normales de los vértices. En sus coordenadas de textura se almacenan las proyecciones de los vértices en el plano de la textura, creando efectivamente una parametrización local de la superficie. Ver Figura 4.3(a).

Un detalle de implementación interesante es que cada parche mantiene un valor interno llamado "centroide de la parametrización". Este centroide contiene el promedio de la parametrización de todos los vértices, y es actualizado cada vez que se agrega una cara al parche. Se utiliza cada vez que se prueba si el parche ha alcanzado la forma y tamaño establecido para los parches. En el caso del achurado, la forma se ha fijado de manera circular, y al actualizar constantemente el centro de la parametrización, se puede asegurar que la parametrización del parche alcanzará una forma lo más similar posible a círculo, independiente de cuál fue la primera cara que se agregó al parche. En la versión original del algoritmo, se consideraba como centro de la parametrización el centroide del primer triángulo añadido al parche.

4.1.3. Optimización de los Parches

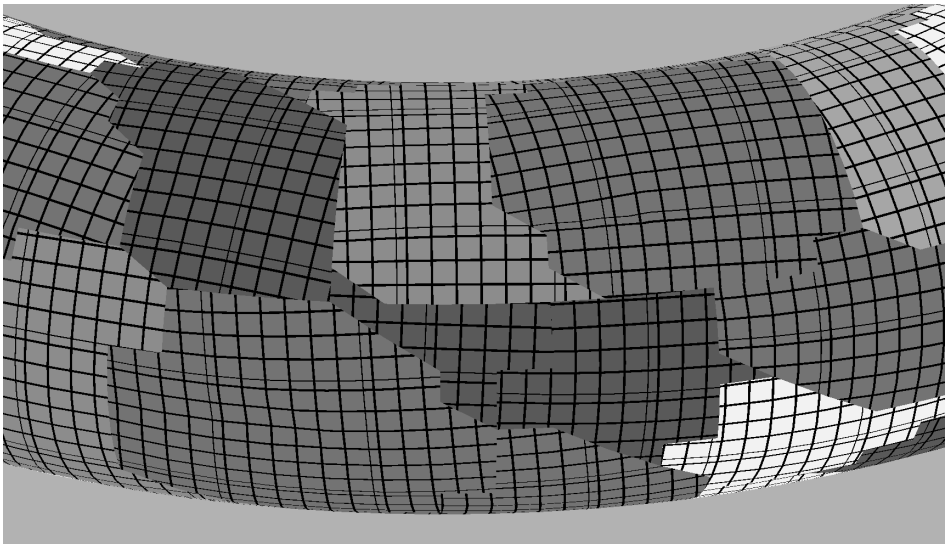
La parametrización que se obtuvo en la etapa anterior ya basta para graficar mediante achurado a la superficie. Sin embargo, la calidad visual de la parametrización de cada parche es mala debido a que las líneas sólo siguen el campo de direcciones en el triángulo inicial de cada parche, por lo que no existe ningún tipo de continuidad asegurada entre las orientaciones de parches distintos, y ningún tipo de correspondencia entre la orientación de una cara arbitraria y el campo de direcciones en la superficie.

Para resolver esta situación se plantea el problema de optimización mencionado en la Sección 2.3.2. El problema es lineal y, ya que se tiene una primera solución (la parametrización inicial), se procede a obtener la mejor solución, de acuerdo a mínimos cuadrados, utilizando un método iterativo, en particular el Gradiente Conjugado [She94].

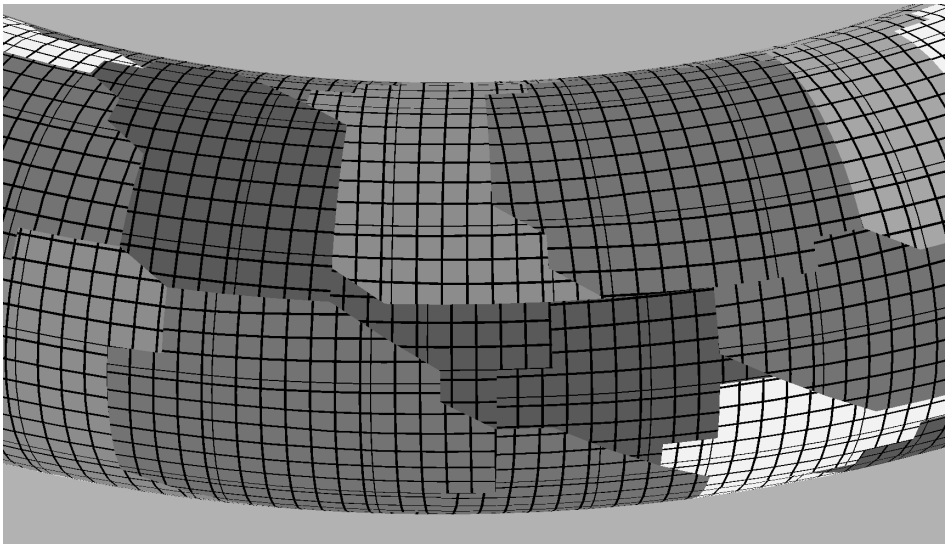
Una vez aplicada la optimización, cada parche está listo para ser graficado utilizando un Tonal Art Map (Sección 2.3.1). La Figura 4.3(b) muestra la optimización resultante de los parches.

La Figura 4.4 muestra el resultado de la optimización de un toroide. El toroide es una figura interesante de achurar debido a que las direcciones principales entregan un patrón entrecruzado perfecto (fila superior y central de la Figura 4.4). Por otro lado, la suavización

[1]No se necesita la circunferencia óptima puesto que sólo se usa la circunferencia como referencia del tamaño del parche.



(a) Parametrización inicial de los parches.



(b) Parametrización optimizada de los parches.

Figura 4.3: Detalle de parametrización de un toroide. En la primera imagen no se ha alineado la parametrización con el campo de direcciones; en la segunda, sí. La diferencia es notoria en los bordes de los parches.

genera un patrón de líneas en forma de espiral alrededor del toroide (fila inferior de la Figura 4.4).

4.1.4. Variables y Características que Afectan los Resultados

Las etapas mencionadas anteriormente generan resultados que dependen de las características de la superficie y de la malla de triángulos que se está procesando. A continuación se mencionan dichas variables y los escenarios relevantes:

- **Ángulo Diedro entre caras:** este valor es utilizado para determinar hasta dónde crece un parche, o más bien, el tipo de límite que tiene con otros parches. Es de relevancia cuando la superficie que se está procesando tiene características de poliedro, con bordes que probablemente están representados con varias aristas y que marcan crestas importantes (como las aristas de un cubo).

  Ahora bien, cuando una superficie está pensada para rendering en tiempo real, es muy probable que la cantidad de triángulos que posea sea baja, y por lo tanto, las probabilidades de obtener un ángulo diedro alto son mayores incluso en zonas que no corresponden a crestas en el diseño de la superficie. Por lo tanto, si bien se puede utilizar un valor predeterminado para el umbral de ángulo diedro, como 35 grados, el valor más adecuado para una superficie dependerá del diseño de ésta y del uso que se desea darle en el contexto del algoritmo de achurado.

- **Tamaño del Parche:** el radio del tamaño del parche se calcula a partir de la "feature size" de un objeto, calculado como una ponderación entre el promedio del primer $20\,\%$ de las aristas de la malla (ordenadas de mayor a menor largo) y el radio de la esfera que contiene a un objeto. Esta heurística se obtuvo desde la implementación de *Real Time Suggestive Contours* de De Carlo et al.[DFR04].

  Por definición, la *feature size* depende tanto de la cantidad y calidad de los triángulos de la superficie como de la forma que tenga el objeto. Por ejemplo, en una superficie que tenga un mallado regular o semiregular, el promedio de las aristas entrega un valor que es representativo de toda la superficie, mientras que en superficies cuyo mallado es irregular, con triángulos con áreas muy distintas, el promedio del tamaño de las aristas no es representativo, por lo que nuevamente el resultado final depende del diseño y del uso que se desea dar a la superficie.

4.1.5. Dificultades con el Campo de Direcciones

Actualmente, la optimización de la parametrización de cada parche puede fallar y generar una parametrización altamente distorsionada. Esto se debe a que los vectores del campo de direcciones presentes en el parche pueden presentar configuraciones que impiden la obtención de buenos resultados. A continuación se describen las configuraciones que se han encontrado y las problemáticas que pueden ocasionar:

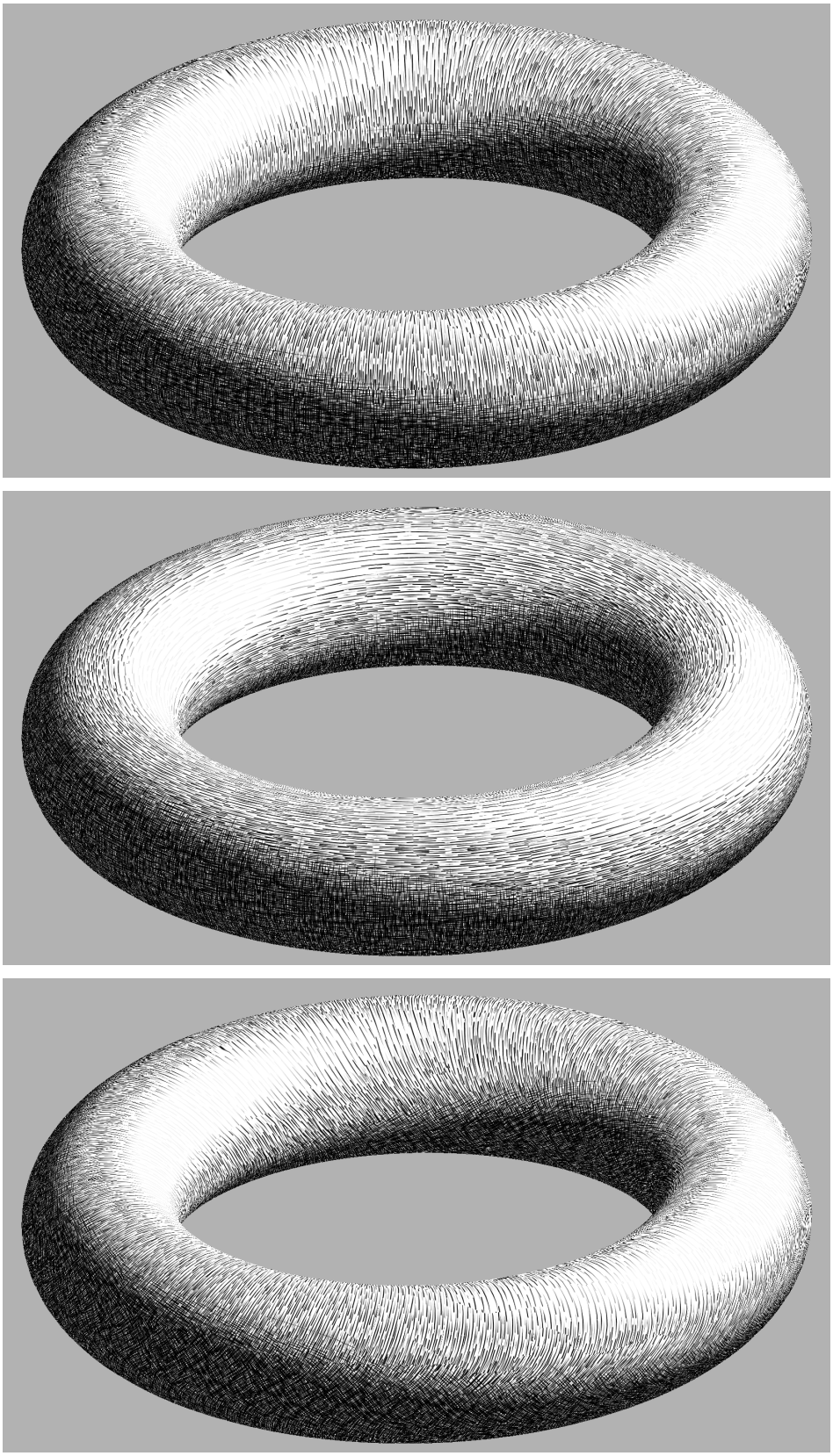


Figura 4.4: Achurado del toroide. En la primera y segunda imagen se utiliza el mismo campo de direcciones, pero en la segunda imagen se han invertido los ejes de la parametrización. En la tercera se ha suavizado el campo de direcciones utilizando la técnica de Hertzmann y Zorin [HZ].

- **Campo suavizado sin orientación**: al ser suavizado el campo de direcciones, para cada triángulo existen 4 vectores que forman una cruz. Si bien el campo en sí es suave, esto trae problemas a la hora de realizar la parametrización: el campo suavizado es 4-RoSy, pero la parametrización requiere dos campos 1-RoSy, uno para la dirección primaria y otro para la dirección secundaria. Por lo tanto, se requiere que para cada parche se decida cuáles vectores dentro de las cruces serán las direcciones primaria y secundaria que utilizará Lapped Textures. Ahora bien, como para cada parche se elige una dirección primaria, es posible que parches vecinos tengan direcciones primarias que apunten en direcciones muy distintas, lo cual le quita continuidad al achurado en los bordes de los parches.

- **Campo que contiene singularidades**: cuando un vértice es una singularidad del campo de direcciones y se encuentra dentro de un parche, la parametrización óptima no es suave y presenta una alta distorsión, como muestra la Figura 4.5. La solución para este problema tiene dos etapas: *1)* en lo posible, suavizar la superficie; *2)* evitar que un parche contenga en un interior un vértice que sea singularidad, es decir, las singularidades sólo deben ser vértices de borde en los parches.

Ahora bien, los problemas mencionados tienen solución. Es posible detectar singularidades en el campo de direcciones, tal como lo indican Tong et al. [TLHD03]. Si se implementaran los campos 4-RoSy de acuerdo a Palacios y Zhang [PZ07], en dicho trabajo también se incluye una fórmula para el cálculo de singularidades (que tiene su base en la Tong et al. [TLHD03]). Sin embargo, ambos enfoques por sí mismos tienen el tamaño y dificultad de un trabajo de tesis.

A pesar de la dificultad, es posible dotar a la herramienta de achurado de dos operaciones que permitirán resolver manualmente los problemas:

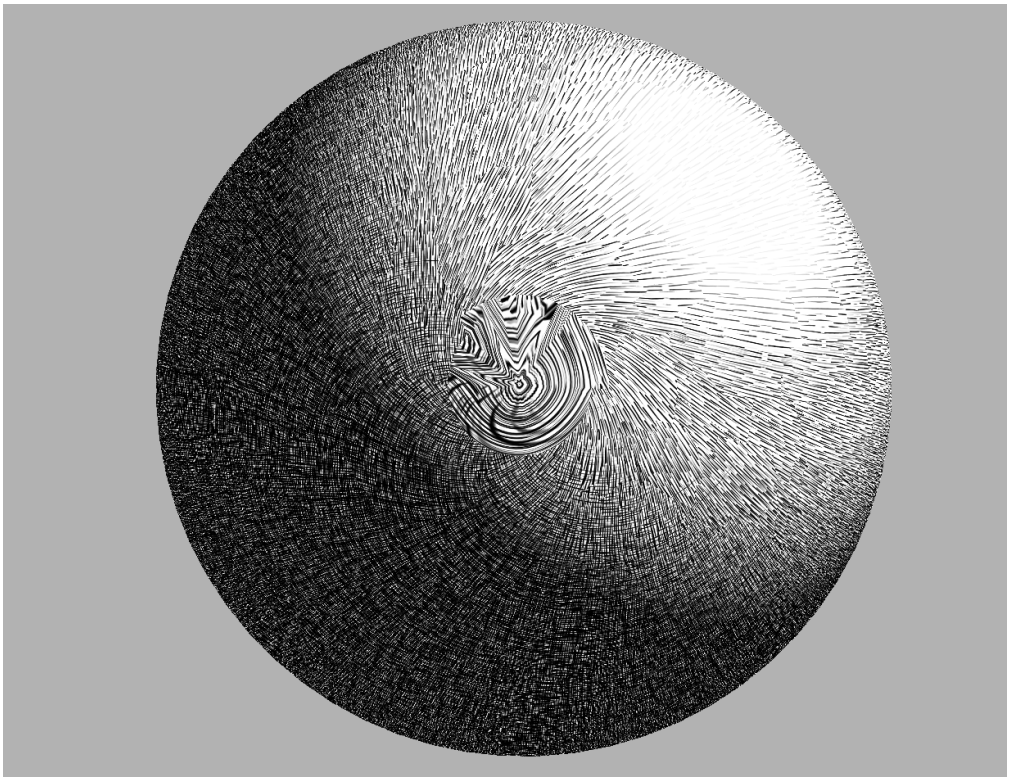
- Mediante la selección de vértices con el puntero del mouse, se puede indicar si un vértice es singularidad o no.

- Mediante la selección de parches con el puntero del mouse, se puede pedir que el parche sea reorientado en caso de que su orientación no se corresponda con la de sus vecinos.

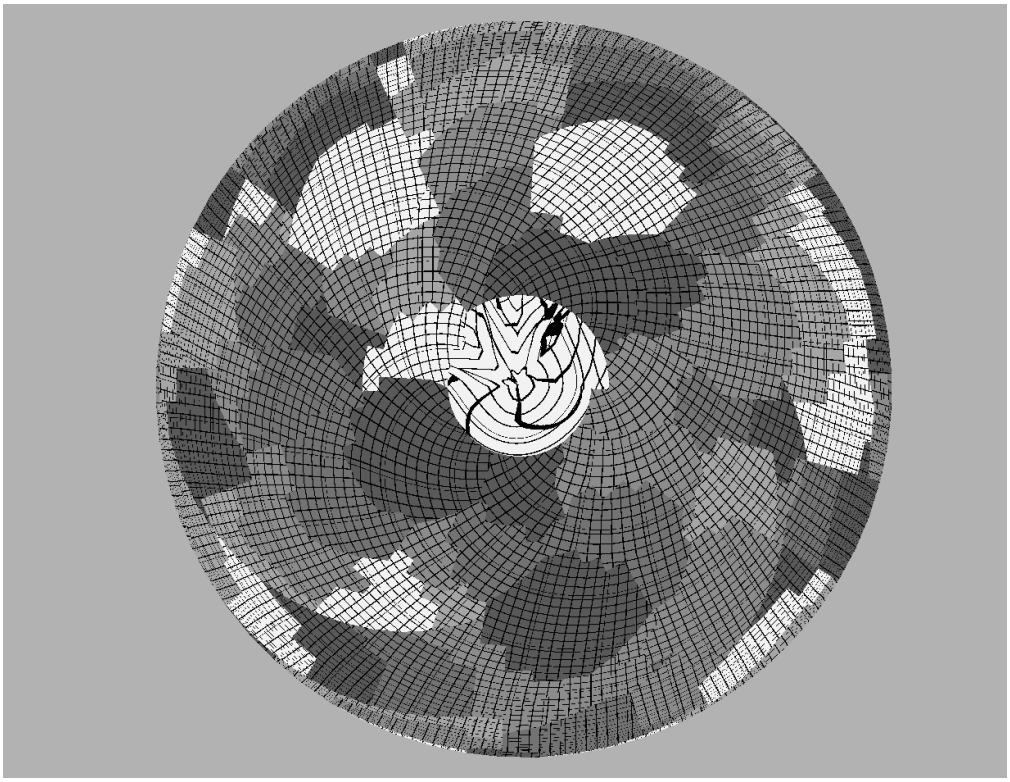
Así, traspasando la responsabilidad de determinar las singularidades y la corrección de los parches al usuario, es posible generar una parametrización aceptable para toda la superficie. La Figura 4.6 muestra una singularidad corregida.

4.2. Rendering

El rendering de la técnica de achurado tiene dos aristas: la primera tiene relación a cómo se entrega el objeto a la biblioteca gráfica; la segunda, en como se utiliza dicha información para representar el achurado.



(a) Singularidad 2 (Hatching)



(b) Singularidad 2 (Parametrización)

Figura 4.5: Singularidades de una esfera. Las dificultades en la parametrización son evidentes, puesto que en los parches que no tienen singularidades, la parametrización (y en consecuencia el achurado) no presenta problemas.

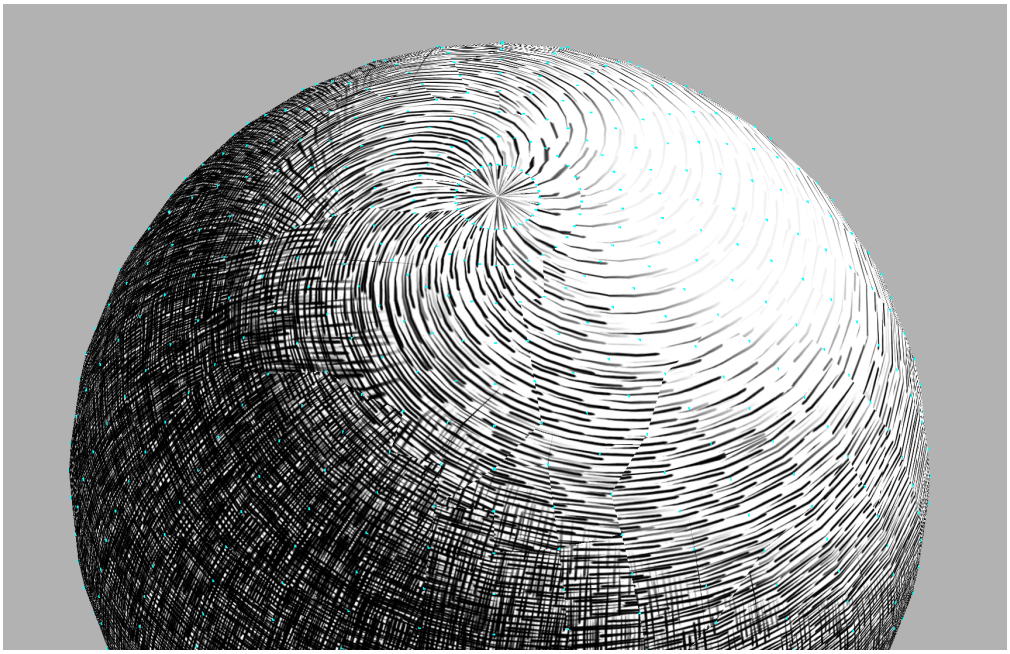


Figura 4.6: Singularidad corregida en el polo de una esfera.

Siguiendo la especificación del algoritmo original, la implementación de la tesis renderiza cada parche por separado. Cada parche se grafica del mismo modo que se grafica un objeto 3D típico: entregando sus vértices, normales y coordenadas de textura a la tarjeta de vídeo.

Las coordenadas de textura 2D son las generadas por Lapped Textures. Como el Tonal Art Map (conjunto de texturas de achurado) tiene una misma parametrización para todas sus texturas, no es necesario especificar cuál se quiere utilizar, ya que eso lo decide la segunda parte del algoritmo.

Dicha segunda parte consiste en decidir cuál textura utilizar al pintar cada píxel de los triángulos. En la primera versión del algoritmo original de Praun et al. [PHWF], se calculaba la intensidad de iluminación en cada vértice del triángulo. Dichas intensidades determinaban tres texturas, una para cada vértice, y el triángulo se pintaba interpolando las tres texturas con un método llamado *6-way blending*.

La segunda versión utiliza un enfoque distinto que aprovecha las características del hardware actual. De acuerdo a lo especificado por Webb et al. [WPFH02], el Tonal Art Map es codificado en una textura 3D, donde las texturas de achurado se apilan de abajo hacia arriba, dejando las imágenes más oscuras (con mayor densidad líneas) abajo y las más claras arriba. Dentro de esta textura 3D, se hace un corte transversal por cada triángulo, determinado por las coordenadas de textura asignadas a cada vértice. La componente de altura dentro de la textura 3D de cada coordenada es determinada por la intensidad de iluminación (calculada como $N \cdot L$). De este modo, es el mismo hardware el que se encarga de realizar la mezcla (*blending*) de las texturas del Tonal Art Map. Este enfoque es el utilizado en la implementación de la tesis, donde la coordenada de textura t para un punto p dentro de la malla se obtiene del siguiente modo:

`Vector3` t_p `= new Vector3( p.texcoord.x, p.texcoord.y, p.normal` $\cdot$ `L )`

Si se utiliza un modelo de sombreado de Gouraud, para cada triángulo se calculan tres

coordenadas de textura 3D, interpoladas dentro del triángulo. Si se utiliza un modelo de sombreado de Phong, lo que se interpola dentro del triángulo es la coordenada de textura 2D de los vértices y la normal de los vértices. En la práctica ambos modelos de sombreado tienen resultados muy similares, ya que la iluminación representada por el achurado corresponde a la "componente difusa" de la luz. Por lo tanto, es más conveniente utilizar el sombreado de Gouraud, ya que el de Phong tiene un costo computacional más alto y, al no utilizar información especular, no presenta beneficios visuales.

4.2.1. Limitaciones

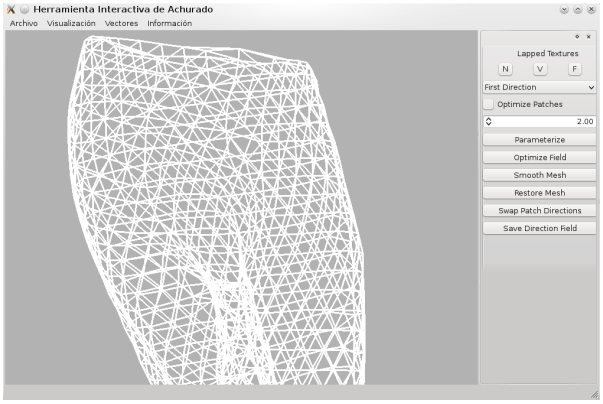
La implementación realizada de Lapped Textures no considera sobreposición u *overlapping* de parches. Se tomó esta decisión debido a que facilitaba la implementación, ya que no es necesario considerar cuánto de un parche es cubierto por otro. La desventaja de no contar con sobreposición es que los bordes entre parches son notorios. Afortunadamente, dicha limitación no impide trabajar y extender el algoritmo de achurado, es sólo una limitación visual que es posible corregir en el futuro.

4.3. Vista General de la Herramienta

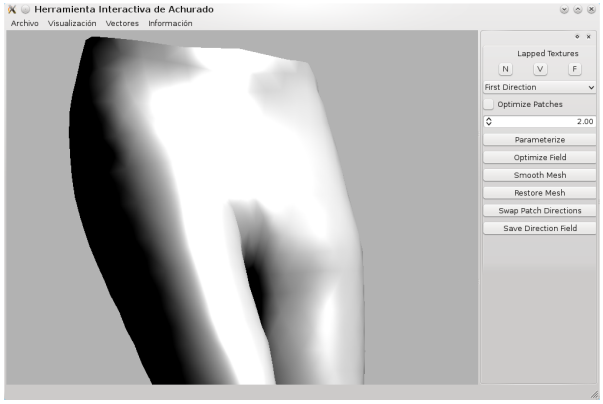
La herramienta implementada permite cargar un modelo 3D estático y realizar las siguientes operaciones:

- Aplicar Lapped Textures.
- Suavizar/Restaurar la malla.
- Optimizar el cambio de direcciones.
- Seleccionar la dirección principal de parametrización.
- Seleccionar el tamaño de los parches (relativo a la *feature size* de la malla).
- Guardar el campo de direcciones en un archivo.
- Guardar la parametrización en un archivo.
- Corregir la orientación de los parches.
- Utilizar visualizadores tradicionales (Wireframe, Phong Shading) para ver la malla, además del achurado.
- Inspeccionar la parametrización de cada parche de la segmentación.

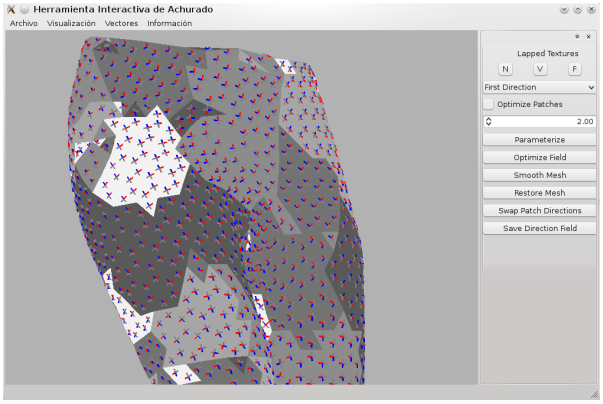
La Figura 4.7 muestra imágenes de la herramienta cargando una malla del set de datos de pantalones de White et al. [WCF07]. En particular, se muestran las distintas posibilidades de graficación de la malla que se ha cargado. Nótese que la superficie presenta agujeros, esto se debe a que la superficie vista en la figura contiene inconsistencias que impidieron que Lapped Textures la procesara por completo.



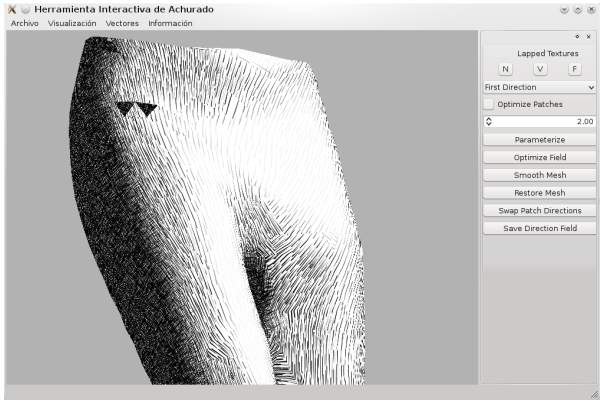
(a) Wireframe



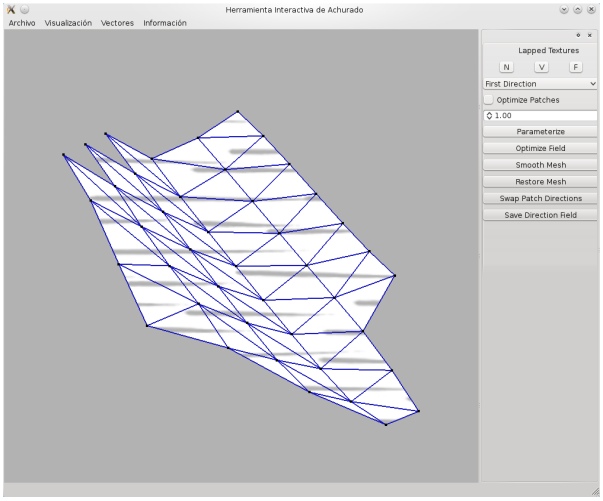
(b) Phong Shading



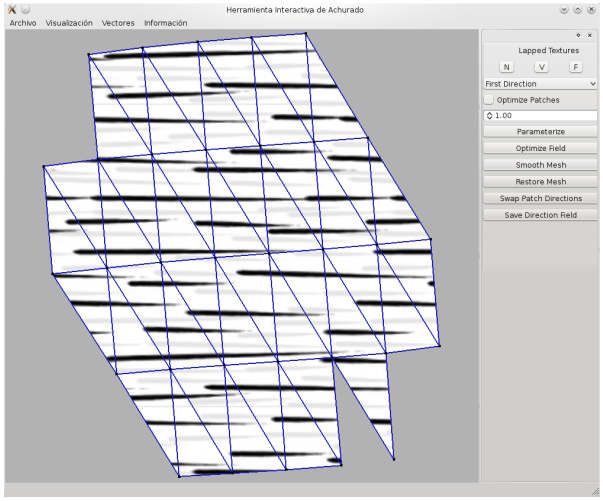
(c) Parches y Cross Field



(d) Achurado



(e) Parametrización



(f) Parametrización

Figura 4.7: Imágenes de la herramienta interactiva en uso.

4.4. Resultado

Además de las imágenes mostradas a lo largo del Capítulo, la imagen de la Figura 4.8 muestra al conejo de Stanford. En este modelo, para evitar los problemas mencionados anteriormente, se utilizó un campo de direcciones artificial, obtenido mediante suavizamiento del campo de direcciones y un alineamiento local en los triángulos del campo de direcciones con los ejes cartesianos.

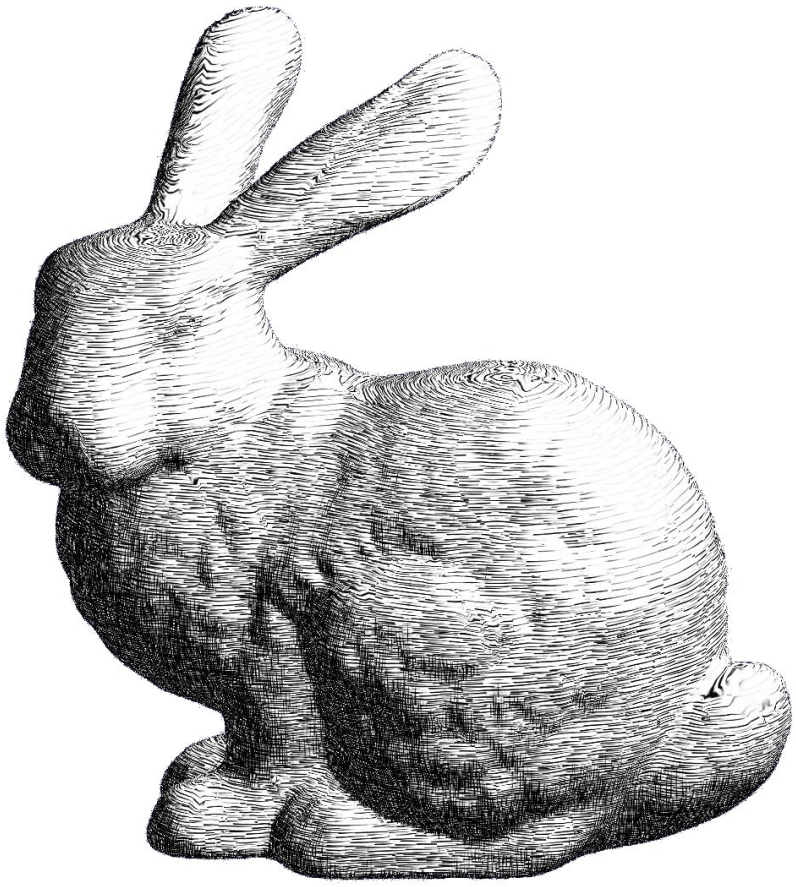


Figura 4.8: Conejo de Stanford con la implementación actual de achurado.

4.5. Desempeño

La Figura 4.9 muestra el tiempo que ha tomado la etapa de pre-proceso para diversas mallas de triángulos. La etapa de pre-proceso incluye:

- Estimación de curvatura y de direcciones principales.
- Generación del cross field a partir de las direcciones principales.
- Segmentación y parametrización de la superficie usando Lapped Textures.

El resultado es de un tiempo mínimo de $0{,}308$ segundos para un modelo de $1,984$ triángulos, y un tiempo máximo de $13{,}69$ segundos para un modelo de 93824 triángulos. Todos los modelos de menos de $20,000$ triángulos demoraron menos de 2 segundos en realizar el pre-proceso.

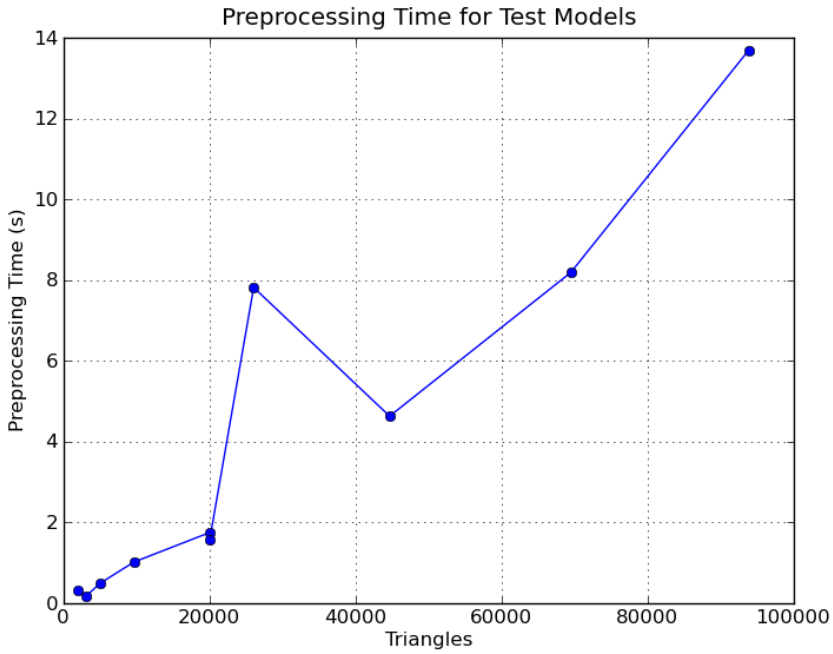


Figura 4.9: Tiempos de pre-proceso para distintas mallas de triángulos.

Respecto al desempeño del renderizado, se graficaron los modelos del set de datos de Vlasic et al. [VBMP]. Estos modelos contienen aproximadamente $20,000$ triángulos. El resultado del renderizado es de $92,47$ cuadros por segundo.

Las pruebas fueron realizadas en un computador portátil con procesador Intel Core i5 520M de $2{,}53$ GHz, con tarjeta de vídeo integrada y 2 GB de RAM.

Capítulo 5

Achurado para Objetos Animados

En el Capítulo 2 se mencionó que una animación de un modelo 3D consiste en reconstruir el objeto en un instante de tiempo a partir de diferentes keyframes, donde cada keyframe representa al mismo objeto en un tiempo predeterminado. Cuando un objeto tiene coordenadas de textura, dichas coordenadas se aplican constantemente en el objeto, sin importar el tiempo ni los keyframes que se utilizan para la animación.

En el rendering tradicional dicho comportamiento es correcto. En el caso del achurado el algoritmo genera coordenadas de textura para el objeto, pero estas coordenadas de textura no tienen el mismo significado que tienen en el rendering tradicional, puesto que en el achurado las coordenadas de textura codifican el sombreado que se aplicará al objeto, no la apariencia que se asocia a la superficie. Por lo tanto, el resultado puede no ser correcto. La Figura 5.1 ilustra el problema: si se considera un cuadro texturizado (imagen izquierda) que es deformado hasta duplicar su ancho, utilizando el enfoque tradicional se obtiene la imagen central, en la cual la textura sigue exactamente la misma deformación. Las líneas verticales del achurado han mantenido su largo pero han duplicado su grosor, mientras que las horizontales han mantenido el grosor pero duplicado su largo. Como resultado, las líneas del achurado han perdido coherencia entre sí. El comportamiento correcto, o más bien, *deseable*, se muestra en la imagen derecha, en la que se han adaptado las coordenadas de textura a la nueva dimensión del cuadro.

(a) Cuadro Original (b) Cuadro Deformado, con distorsión (c) Cuadro Deformado, sin distorsión

Figura 5.1: Muestra del problema que ocurre al mantener las coordenadas de textura al animar un cuadro. El cuadro original (izquierda), es deformado hasta duplicar su ancho. El esquema de animación tradicional entrega el resultado central. El esquema requerido por el algoritmo de achurado es el que muestra la imagen derecha.

Entonces, los problemas se pueden resumir en que las líneas pierden coherencia entre sí,

entorpeciendo el aspecto visual del sombreado, haciéndolo parte de la superficie del objeto al recibir la misma deformación. En este Capítulo se estudia y propone una solución para este problema, definiendo el comportamiento que permite obtener el resultado de la imagen derecha en la Figura 5.1.

5.1. Enfoque Ingenuo

La primera solución que se puede proponer es el enfoque ingenuo o solución de fuerza bruta. Si se desea obtener un algoritmo que adecúe el achurado a todos los keyframes de una animación, ¿por qué no aplicar el algoritmo original a todos los keyframes y luego interpolar los resultados? Existen dos importantes limitantes que impiden utilizar este enfoque:

- Si se ejecuta Lapped Textures en cada keyframe, la segmentación en un keyframe puede ser muy distinta a la segmentación en otro keyframe, por lo que no sería posible realizar una interpolación de los resultados.
- Si se calcula un campo de direcciones en cada keyframe, debido a que el cálculo de direcciones principales es muy sensible a deformaciones en la malla, y a que se utiliza una optimización no lineal sobre la que no se tiene control, es muy posible que se obtengan campos direccionales muy diferentes entre sí a lo largo de los keyframes. Esto provocaría que las líneas de achurado no mantengan una dirección "constante" relativa a la superficie.

Así, un algoritmo de achurado para objetos animados debiese ser capaz de sobreponerse a esas dos limitantes.

5.2. Definición de Animación de Textura

La solución propuesta en la tesis considera crear una animación de las coordenadas de textura, de modo que las líneas de achurado se adapten a los cambios en la superficie, de acuerdo a la Figura 5.1. Los elementos que se consideran para definir la animación de la textura de achurado son:

- Objeto representado como malla de triángulos en 3D, con animación de n keyframes.
- Conjunto de parches con parametrización local del objeto, obtenida con el algoritmo original de achurado.
- Campo de direcciones utilizado en el algoritmo original.

La animación de la textura de achurado se define como una animación 2D que afecta a los parches que se crearon con Lapped Textures de un objeto:

- **Animación 2D**: para los keyframes del objeto, se define una segmentación donde cada parche tiene la misma topología (es decir, representa a las mismos triángulos) a través de los distintos keyframes.
- **Parches por cada keyframe**: la información dependiente del tiempo en cada parche es la posición de sus vértices en 2D, es decir, sus coordenadas de textura.

De este modo, para cada reconstrucción de la animación, independientemente del método utilizado (Morph Target o Skinning), dados dos keyframes K_i y K_j, y dos colecciones de parches P_i y P_j, se define la animación del objeto en el instante t, incluyendo animación de textura de achurado, como:

- **Animación del Objeto**: se reconstruye el objeto M_t utilizando $\texttt{MorphTarget}(K_i, K_j, t)$ o $\texttt{Skinning}(K_i, K_j, t)$, según corresponda.
- **Animación de la textura**: por cada parche p de la malla que representa al objeto, se define p_i que pertecene a K_i, y p_j que pertenece a K_j. Se define P_t como el conjunto que contiene todas las interpolaciones $\texttt{MorphTarget2D}(p_i, p_j, t)$. Los parches p_i y p_j son mallas de triángulos en 2D cuyos vértices son las coordenadas de textura para K_i y K_j. Los parches p_i y p_j deben tener la misma topología.

5.3. Algoritmo de Achurado para Objetos Animados

El algoritmo de achurado para objetos animados se resume, en su etapa de preproceso del objeto y animaciones, como el cálculo del algoritmo para objetos estáticos en cada frame de la animación, donde la topología de todos los parches se obtiene a partir de la malla que mejor representa al objeto (en caso de skinning se utiliza la *bind-pose*; en caso de Vertex Animation, el primer frame de la animación) y cada frame reutiliza dicha topología. Lo anterior permite generar animaciones de la textura de cada parche que representarán el achurado correspondiente a la animación del objeto dentro del parche respectivo.

5.3.1. Vista General del Algoritmo

El algoritmo de achurado para objetos animados se define, en su etapa de pre-proceso del objeto y sus animaciones, como sigue:

1. Dado un objeto con animación, obtener la malla del objeto que mejor lo represente. Para un objeto animado mediante Skinning, esta malla corresponde a la malla del objeto en la *bind-pose*, es decir, en su posición neutral. Para Morph Target, se debe elegir manualmente el keyframe que más se acerque a una posición neutral del objeto.
2. Una vez seleccionada la pose neutral, estimar o generar un campo de direcciones D para la superficie.

3. Codificar D para que sea expresado en términos de la superficie.

4. Utilizando el campo de direcciones del punto anterior, aplicar Lapped Textures para generar un conjunto de parches P, de acuerdo al algoritmo original de achurado.

5. Para cada keyframe K de la animación, obtener la malla M_k y copiar la topología de los parches de P a P_k. Cada parche en P_k tiene, inicialmente, la parametrización obtenida del algoritmo original.

6. Generar un conjunto de direcciones D_k, que representa el campo de direcciones D decodificado para que sus direcciones sigan siendo tangentes a la superficie M_k.

7. Para cada parche p_i en P_K, generar una nueva parametrización local, volviendo a ejecutar la optimización de la parametrización. Esta vez se utiliza como campo de direcciones a D_k. La solución inicial de la optimización es la parametrización obtenida en P en caso de $k = 1$, o la parametrización obtenida en P_{k-1} si $k > 1$.

Una vez cumplidos todos los pasos, a la animación propia del objeto se suma la animación en 2D que tendrá la textura de achurado.

5.3.2. Topología de Parches Común

Del mismo modo que en Morph Target se requiere que todas las mallas de la animación tengan una topología común, en esta versión 2D de Morph Target es necesario que todas las mallas 2D sean coherentes entre sí. Como consecuencia de este requerimiento, es necesario que la parte de creación de los parches y segmentación de Lapped Textures se ejecute una única vez, porque si Lapped Textures se ejecutara una vez por cada keyframe, no habría forma de asegurar que la topología de los parches sea coherente durante la animación.

La Figura 5.2 muestra tres frames de una animación que comparten una misma topología, siguiendo el esquema mencionado.

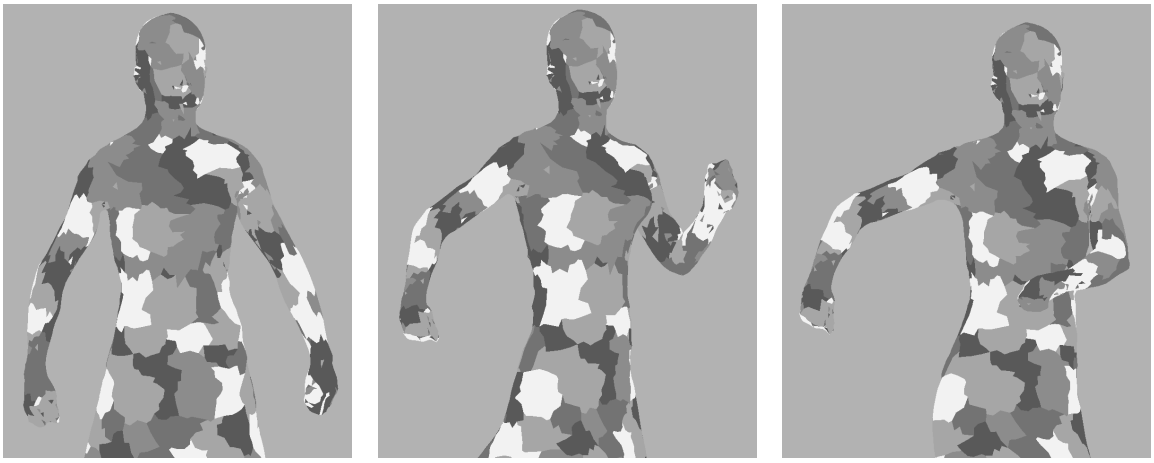


Figura 5.2: Topología de parches es constante durante la animación.

Ahora bien, el generar los parches una única vez plantea interrogantes que pueden ser difíciles de responder, en particular, ¿cómo asegurar que los parches generados son idóneos? Para objetos estáticos se puede utilizar el criterio de la curvatura, con el cual se acota la variación de la curvatura dentro de un parche, dejando así que los bordes de los parches coincidan con los lugares de curvatura más extrema, pero para objetos animados ese criterio no siempre es adecuado, ya que las zonas que presentan alta curvatura en un keyframe pueden cambiar en otro.

La solución a este problema es calcular una métrica que evalúe la deformación que presenta cada triángulo a través de los cuadros de la animación. Dichas métricas existen, en particular hay una que es relevante para este trabajo, la métrica de DeCoro y Rusinkiewicz [DR05]. Dicha métrica, aplicada en el contexto de simplificación de mallas, otorga un puntaje a una arista para determinar su grado de deformación a través de diferentes poses de una animación. Este puntaje es utilizado para decidir si la malla se simplifica en esa arista mediante un *edge collapse*. Lamentablemente ese enfoque no es directamente aplicable debido a los siguientes problemas:

- El puntaje de deformación se calcula considerando cambios en la malla, pero en el caso de la tesis, no existen dichos cambios. La métrica debiese ser modificada para entregar un puntaje similar, donde, por ejemplo, se determine si la curvatura inicial de la superficie se mantuvo dentro de un rango o si varió con mucha frecuencia y/o rapidez.
- Para estimar el puntaje, se requiere que la técnica de animación sea Skinning. En Morph Target sería necesario adivinar o predecir un sistema de coordenadas común en todos los keyframes para cada triángulo, con el fin de determinar si la deformación es tal o si solamente es un cambio de posición y/o orientación. En cambio, al utilizar Skinning, los sistemas de referencia ya están planteados, ya que los vértices de la malla están especificados en el espacio de coordenadas del esqueleto. El problema que afecta a la tesis es que los datos de prueba no utilizan Skinning.

No obstante estos problemas, es posible probar el comportamiento de la segmentación en parches sin considerar la deformación en el futuro. Esto puede entregar resultados aceptables porque los datos de prueba presentan objetos cuyas mayores deformaciones se dan en regiones específicas de la superficie (por ejemplo, un modelo sencillo de un ser humano se deforma en sus articulaciones principales, como los hombros, codos y rodillas).

En los experimentos con los datos de prueba, la falta de una métrica no ha generado problemas, aunque, por completitud del algoritmo, en el futuro será necesario plantear dicha métrica.

5.3.3. Campo de Direcciones Común

El enfoque ingenuo estipulaba que para cada keyframe se debía estimar un nuevo campo de direcciones, de modo de obtener las direcciones más representativas para la malla en ese instante. Aunque el cálculo del campo (y su posible optimización) es costoso, en la práctica

es un pre-proceso que no incide en el desempeño del algoritmo en tiempo de ejecución, por lo que era una alternativa a considerar.

Sin embargo, cuando se realizó la implementación del enfoque ingenuo, los resultados que presentaba el recálculo del campo de direcciones no eran satisfactorios, principalmente por dos motivos:

1. Sensibilidad de la estimación de curvatura a los cambios en la geometría: cambios pequeños en la geometría pueden generar grandes variaciones en el campo de direcciones.

   Si se adoptara la estimación de un campo para cada keyframe, es muy probable que existan direcciones que, de un frame a otro, presenten cambios bruscos de orientación. Esto presenta un grave problema, ya que las parametrizaciones entre keyframes serán muy diferentes, y al realizar una animación, será necesario interpolar dichas parametrizaciones. Si no se puede asegurar que las parametrizaciones serán similares, es seguro que aparecerán artefactos gráficos producto de la interpolación. Estos artefactos son similares a los explicados por Baxter et al. [BBA08]. Baxter analiza los problemas al interpolar imágenes 2D, situación similar a la interpolación de parametrizaciones. La Figura 5.3 muestra el problema: en la fila superior se ilustra la interpolación lineal entre la orientación inicial de una imagen y su orientación final. Se observa que la imagen es altamente distorsionada, perdiendo todo su significado en los puntos intermedios. La fila inferior muestra que la solución ideal considera el contexto y genera una interpolación coherente.

2. Respecto a la percepción del observador, se puede argumentar que si las líneas cambian de dirección constantemente a lo largo de una animación, se estaría distrayendo la atención hacia el movimiento de las líneas, y no hacia el movimiento del objeto. Por lo tanto, se deben minimizar los cambios en la parametrización, y una forma de ayudar dicha minimización es mantener constante, respecto a la superficie, el campo de direcciones.

Para solucionar estos problemas, se ha decidido calcular un campo de direcciones que se mantenga constante sobre la superficie. Para esto, se calcula el campo de direcciones en una pose representativa para el objeto. En el caso de que el objeto esté animado con Skinning, la mejor pose es la *bind-pose*, porque corresponde a la posición neutral del objeto desde la cual se crean las animaciones. En el caso de Morph Target es necesario que se elija manualmente la mejor pose (lo que equivale a elegir un keyframe de la animación), o bien simplemente usar el primer keyframe, que en el caso de las animaciones de los datos de prueba, era muy similar a una pose neutral.

Ahora bien, una vez que se ha decidido utilizar el mismo campo de direcciones, hay que definir *qué significa* mantener dicho campo constante, ya que el campo inicial, que es un campo vectorial tangente a la superficie, difícilmente mantendrá su condición de tangencial una vez que se ha deformado la superficie. La solución a este planteamiento es el uso de **coordenadas baricéntricas**, que permiten expresar el campo de direcciones en términos de la topología de la superficie, volviéndolo así independiente de la posible deformación que pueda afectar a la superficie. Esto no asegura que el campo siga siendo óptimo en lo

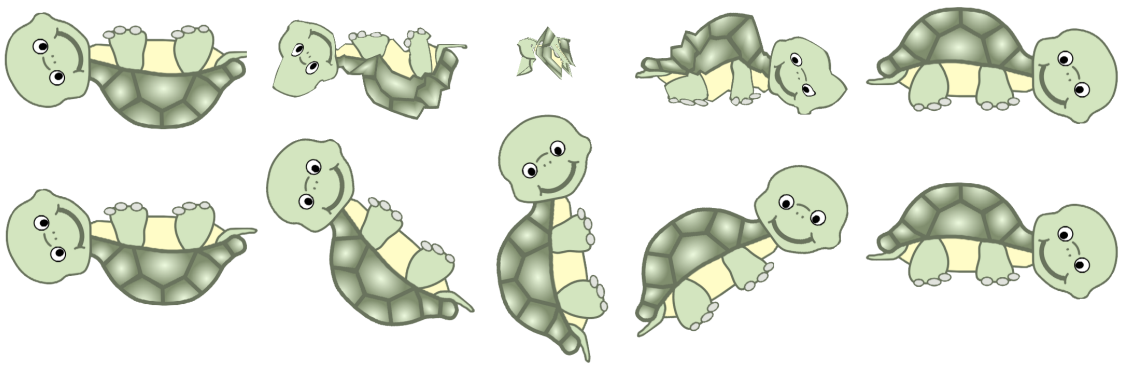


Figura 5.3: Artefactos gráficos generados por la interpolación lineal entre una imagen que está rotada (fila superior). La fila inferior muestra la interpolación que debiese llevarse a cabo. Fuente [BBA08].

que respecta a suavidad del entrecruzado, pero se puede forzar que se siga obteniendo un *cross field*: basta codificar solamente la dirección primaria del campo de direcciones, el resto de los vectores se obtiene invirtiendo dicha dirección y obteniendo el producto cruz entre ambos vectores y la normal de cada triángulo.

La Figura 5.4 muestra el comportamiento del campo durante tres frames (el inicial, uno intermedio y el final) de una de las animaciones del set de datos de prueba. El campo fue calculado para el primer frame y codificado a coordenadas baricéntricas, y para cada frame siguiente fue recalculado a partir de dichas coordenadas. Se observa que el campo se mantiene tangente a la superficie, y que si bien analíticamente no es suave, la diferencia no es perceptible y por lo tanto es apto para la parametrización que se requiere.

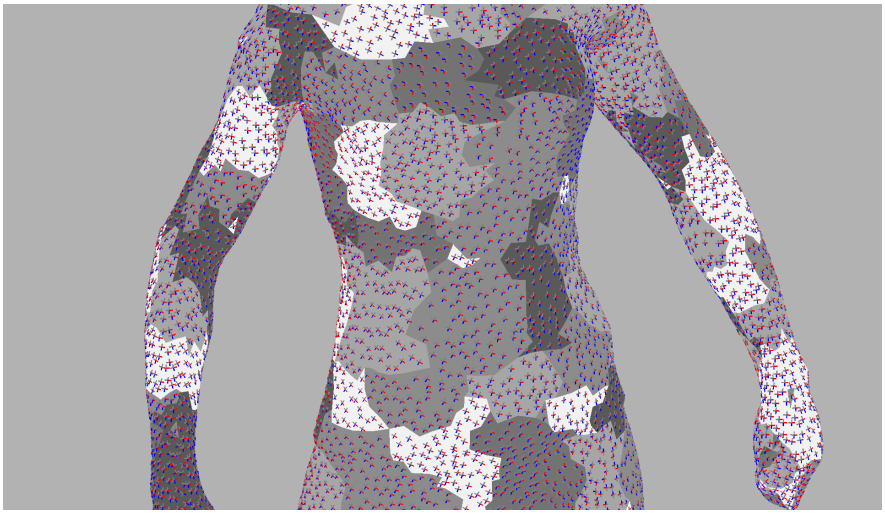
5.3.4. Cálculo de la Parametrización en Cada KeyFrame

En el algoritmo original, la optimización de la parametrización recibía como input una solución inicial y el campo de direcciones en coordenadas baricéntricas. En el algoritmo con animación, ese procedimiento se lleva a cabo para la mejor pose del objeto. Para las otras poses o keyframes se vuelve a optimizar la parametrización, considerando el campo de direcciones correspondiente a ese keyframe y como solución inicial la parametrización optimizada del keyframe anterior.

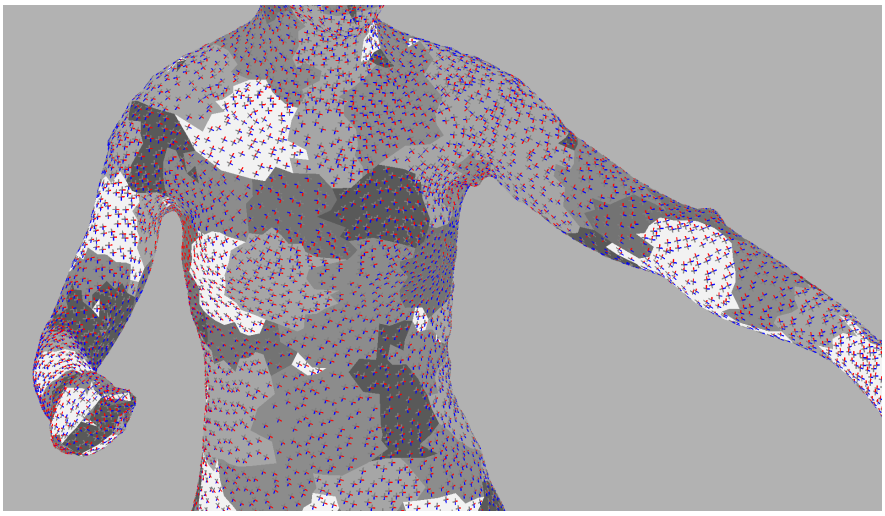
Realizar este procedimiento permite asegurar que exista coherencia en la animación de la textura de achurado. Además, como la optimización se hace mediante Conjugate Gradient [She94], la solución es iterativa, y en caso de haber poca deformación de la geometría del objeto entre dos keyframes seguidos, la parametrización en cada keyframe variará poco y por lo tanto será calculada rápidamente.

5.3.5. Pseudocódigo

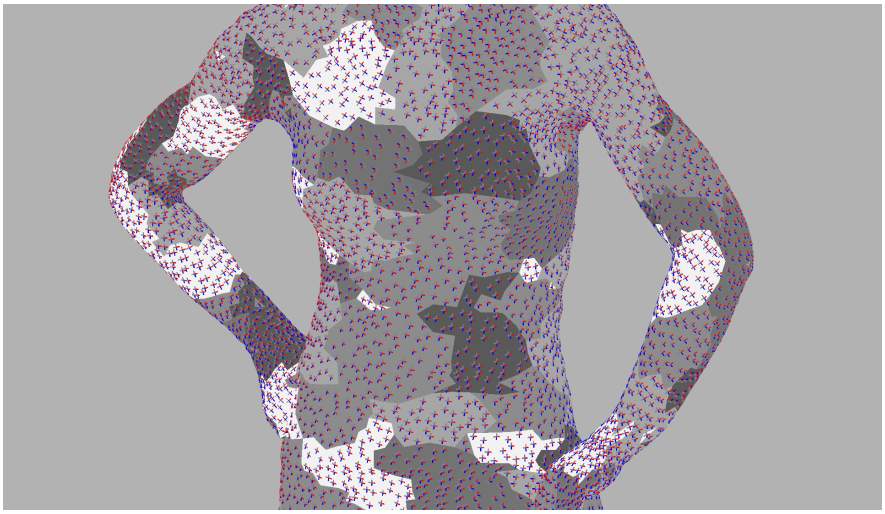
El Algoritmo 1 muestra el pseudocódigo del algoritmo.



(a) Campo Inicial



(b) Frame Intermedio



(c) Frame Final

Figura 5.4: Campo de direcciones durante tres frames distintos. El campo inicial se recalcula en los otros frames luego de haber sido codificado utilizando coordenadas baricéntricas.

Algorithm 1 Algoritmo de pre-proceso de achurado para objetos animados.

```
1: O = objeto animado
2: M = malla característica de O
3: D = campo de direcciones para achurado de M
4: M.calculateBarycentricCoords( D )
5: P = LappedTextures(M, D)
6: for all keyframe K in O do
7:    M_k = malla de K
8:    P_k = P
9:    P.updatePositions()
10:   P.updateNormals()
11:   D_k = M.reproject( D )
12:   for all parche p_ik en P_k do
13:      if k == 0 then
14:         p_ik.makeInitialParameterization()
15:      else
16:         p_ik.copyParameterizationFrom( p_(i-1)k )
17:      end if
18:      p_ik.optimize()
19:   end for
20: end for
```

5.3.6. Almacenamiento

Al utilizar la técnica Morph Target en 2D, la técnica de achurado para objetos animados hereda todas sus desventajas en lo que almacenamiento se refiere. A continuación se desglosa el almacenamiento requerido por la técnica propuesta, para una superficie M que contiene T triángulos, con una animación que contiene K keyframes.

- Si se consideran P parches en la segmentación de M, se puede aproximar como $\frac{T_P=T}{P}$ la cantidad de triángulos por parche.
- Cada parche tiene $T_P + 2$ vértices, debido a que cada parche contiene una malla cerrada y *two-manifold*. Cada vértice ocupa al menos 8 bytes (2 números de punto flotante) para la coordenada de textura.
- Por lo tanto, cada keyframe de la animación de textura utiliza $8T_P + 16$ bytes.
- En total, un modelo con K keyframes, que utilizan B_K bytes, y P parches, utiliza $B_K * P * (8T_P + 16)$ bytes.

La consecuencia de este desglose es que, para animaciones grandes, el espacio utilizado por los keyframes y su animación puede aumentar considerablemente. Si bien en los tiempos actuales la memoria RAM difícilmente se acabará, el mayor problema en ocupar tanto espacio es el ancho de banda entre la RAM y la memoria de la GPU. TODO: mostrar un gráfico o tabla que muestre cuánto crece esto a medida que se agregan keyframes.

5.3.7. Rendering

El rendering de la animación con achurado consiste en una extensión al rendering presentado en el capítulo anterior. La diferencia radica en que antes se graficaba directamente el objeto en cuestión, pero en presencia de animación se crea una malla extra, llamada *malla interpolada*. La malla interpolada contiene el resultado de la animación (sea *vertex animation* o *skinning*).

La malla interpolada también contiene un conjunto de parches. Este conjunto de parches es el que contiene la animación de la textura. Por lo tanto, el rendering de un objeto animado tiene 3 pasos:

1. Generar la malla interpolada.
2. Generar la parametrización interpolada.
3. Graficar la malla interpolada siguiendo el esquema del Capítulo anterior.

De los tres pasos del algoritmo, el punto 1 es común a todo tipo de rendering para objetos animados, y el punto 3 es común a todo algoritmo de rendering. El paso 2, de interpolar la parametrización, consiste en interpolaciones lineales en 2D, un proceso que es paralelizable debido a que cada interpolación es independiente de las otras. Se puede decir que el rendering del achurado con animación no es complejo respecto a técnicas tradicionales de rendering y que tiene potencial para ser ejecutado en tiempo real siempre que cada punto sea implementado de manera óptima.

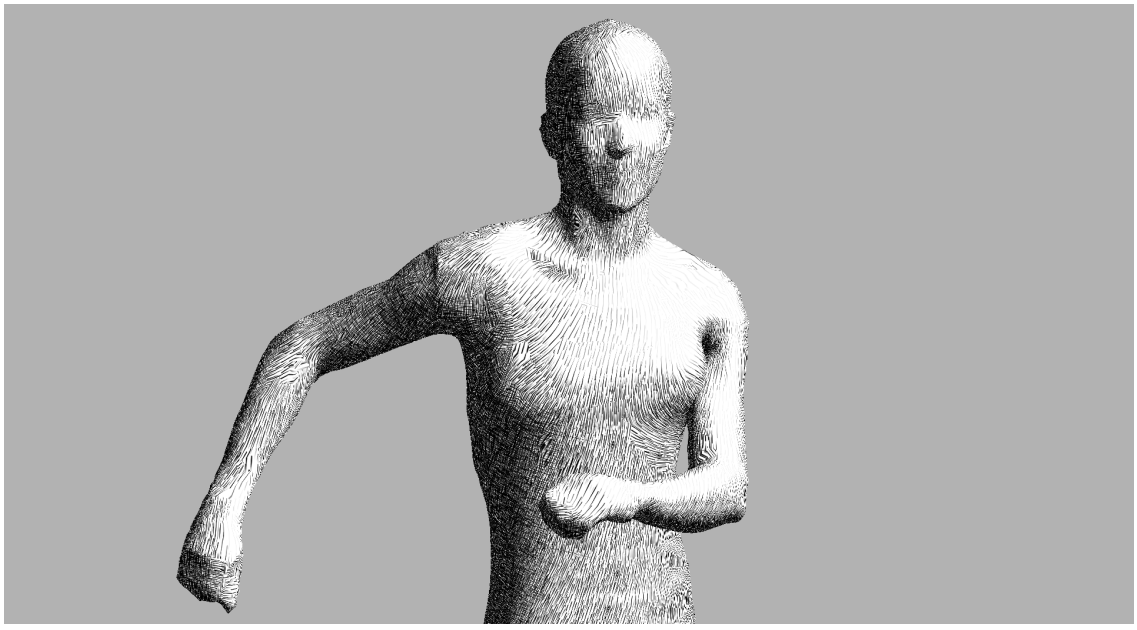
5.4. Resultados

La metodología para evaluar los resultados es la siguiente:

1. Se carga la malla representativa como si fuese un modelo estático en la herramienta presentada en el capítulo anterior.
2. Se genera una parametrización de achurado y campo direccional para la malla representativa. Tanto la parametrización como el campo direccional se almacenan en el disco duro en un archivo junto a la malla representativa.
3. Se carga la aplicación de achurado para modelos animados, teniendo como input la animación y los archivos generados en el punto anterior.

5.4.1. Datos de Captura de Movimiento de Personas

A continuación se muestran resultados de aplicar el algoritmo a dos sets de los datos de Vlasic et al. [VBMP]. La Figura 5.5 muestra el detalle de las dos animaciones.



(a) Imagen de set de datos `samba`.

(b) Imagen de set de datos `march_1`.

Figura 5.5: Detalles de las animaciones presentadas.

La Figura 5.6 muestra 10 frames de la animación `samba`. En este caso el resultado es visualmente bueno, aunque la superficie presenta diferentes zonas en las cuales la parametrización presenta distorsiones. Por otro lado, la Figura 5.7 muestra la animación `march_1`, una animación más complicada debido a que la superficie presenta muchos detalles debidos a la vestimenta de la persona a la cual se le capturó el movimiento.

A nivel general, los resultados del algoritmo, en lo que respecta a la animación de estos datos, son satisfactorios. Si bien se presentan artefactos visuales, éstos se deben a problemas en la parametrización, derivados de la dificultad de implementación del algoritmo original de achurado.

5.4.2. Datos de Captura de Movimiento de Ropa

En el Capítulo 3 se mencionó que se dispone de un set de datos de captura de movimiento de pantalones, publicado por White et al. [WCF07]. Este set de datos presenta un mallado regular y figuras sencillas; sin embargo, a pesar de ser en teoría un set más simple que los datos anteriores, las mallas no tienen una topología totalmente coherente, lo que influye en el cálculo del campo direccional que se estima para la superficie.

La Figura 5.8 muestra el detalle de un keyframe de este set de animaciones. La Figura 5.9 muestra cuatro keyframes del set de animaciones.

Visualmente hay varios artefactos que se atribuyen a la segmentación de Lapped Textures, aunque se puede apreciar que en una parte considerable de la superficie se obtuvieron buenos resultados. Este set es particularmente interesante porque la deformación a la que fueron sometidos los datos es extrema en algunos lugares, como las rodillas. El comportamiento del algoritmo en esos lugares es visualmente aceptable, aunque no es del todo correcto de acuerdo a lo propuesto debido a que la parametrización no es totalmente correcta.

5.4.3. Desempeño

El renderizado del achurado con animación para el set de datos `samba`, el más complejo de los tres, es de $74{,}05$ cuadros por segundo. La animación tiene 125 frames. Utilizando la fórmula mencionada anteriormente, se necesita un mínimo de $7{,}98$ MB de memoria para almacenar toda la animación.

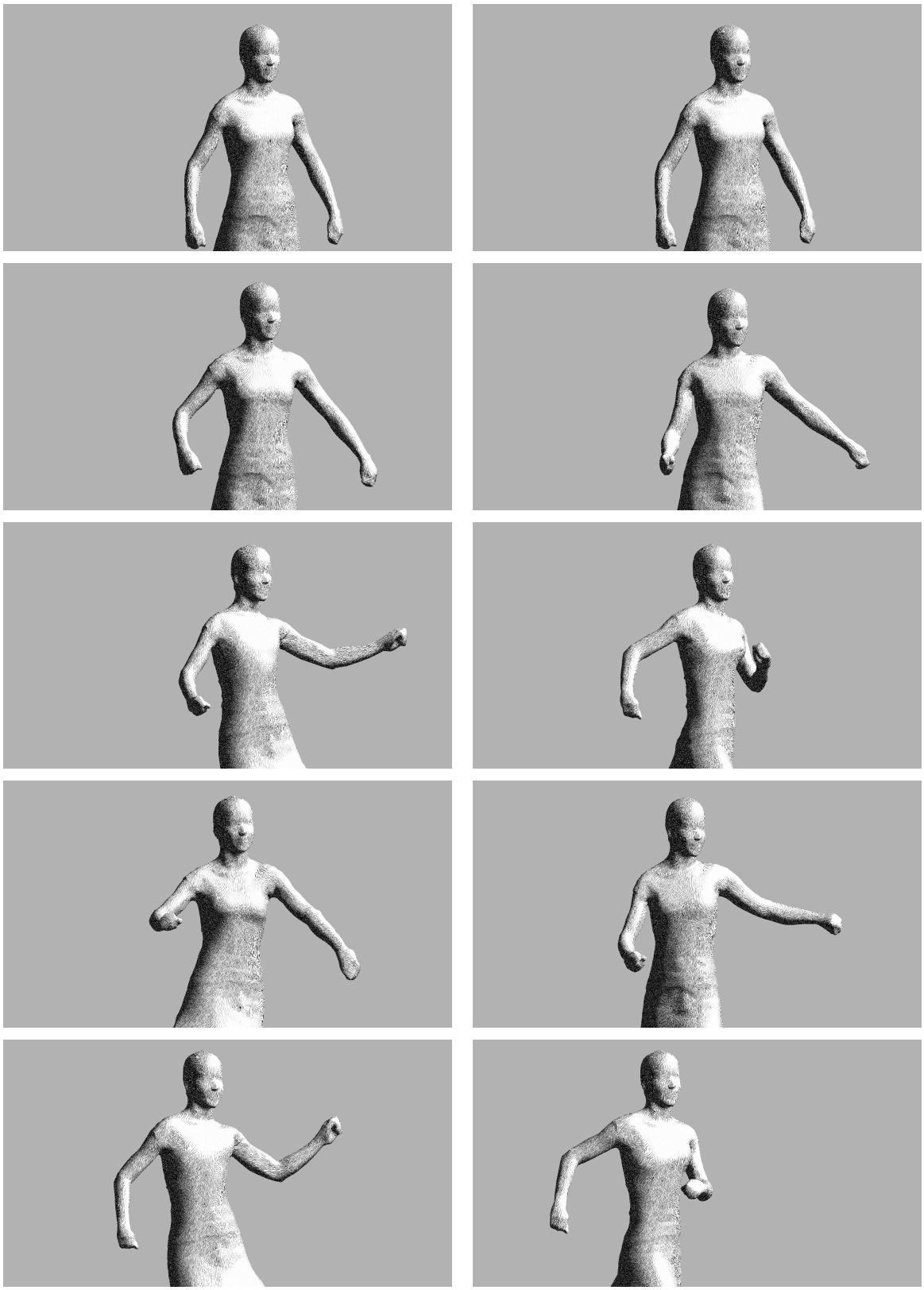


Figura 5.6: Resultados de una animación, llamada `samba`, 10 frames.

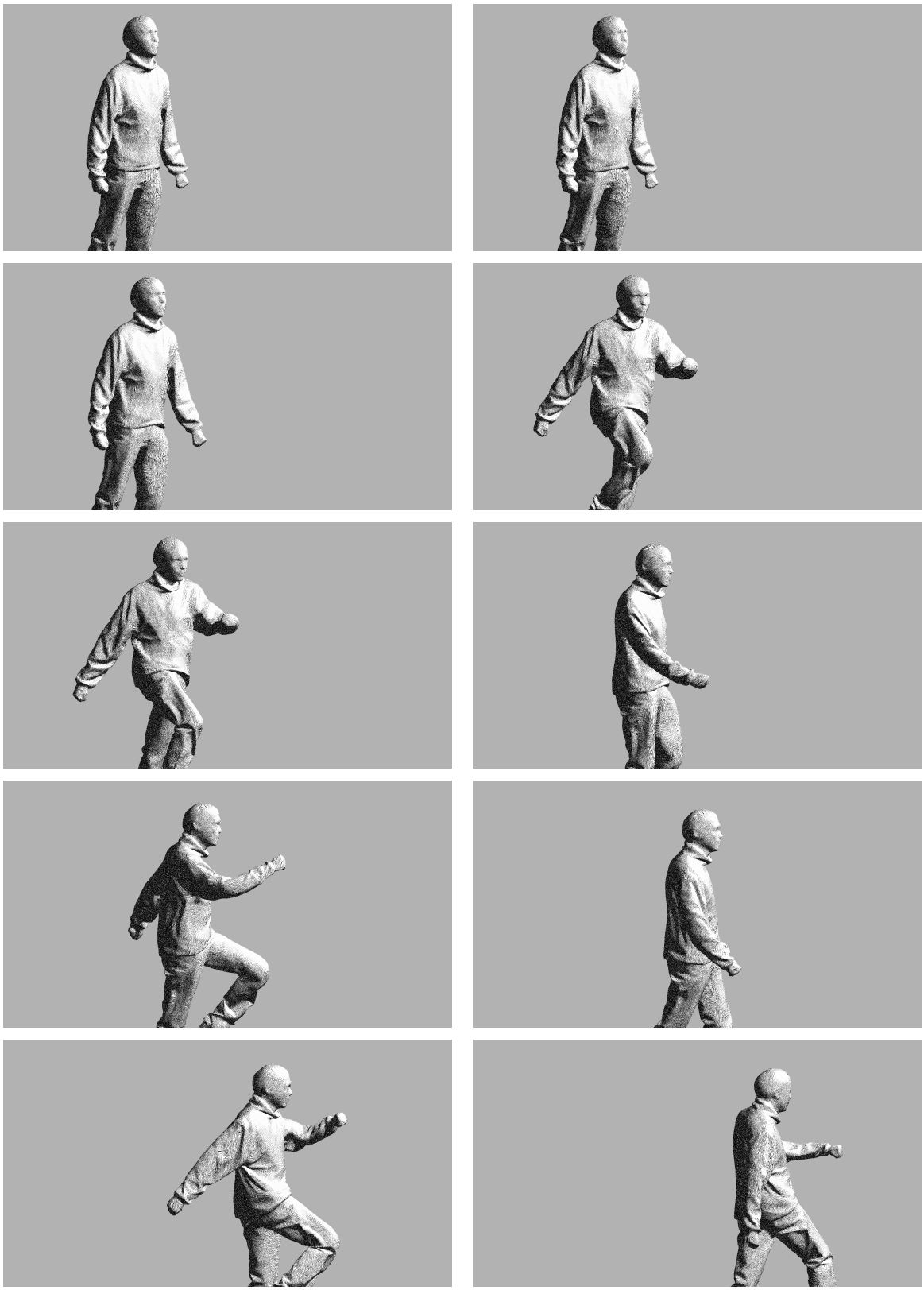


Figura 5.7: Resultados de una animación, llamada `march_1`, 10 frames.

Figura 5.8: Detalles de la animación de pantalones. En el costado exterior de la pierna izquierda se observan artefactos en la parametrización, pero en el resto de la superficie el achurado se comporta correctamente.

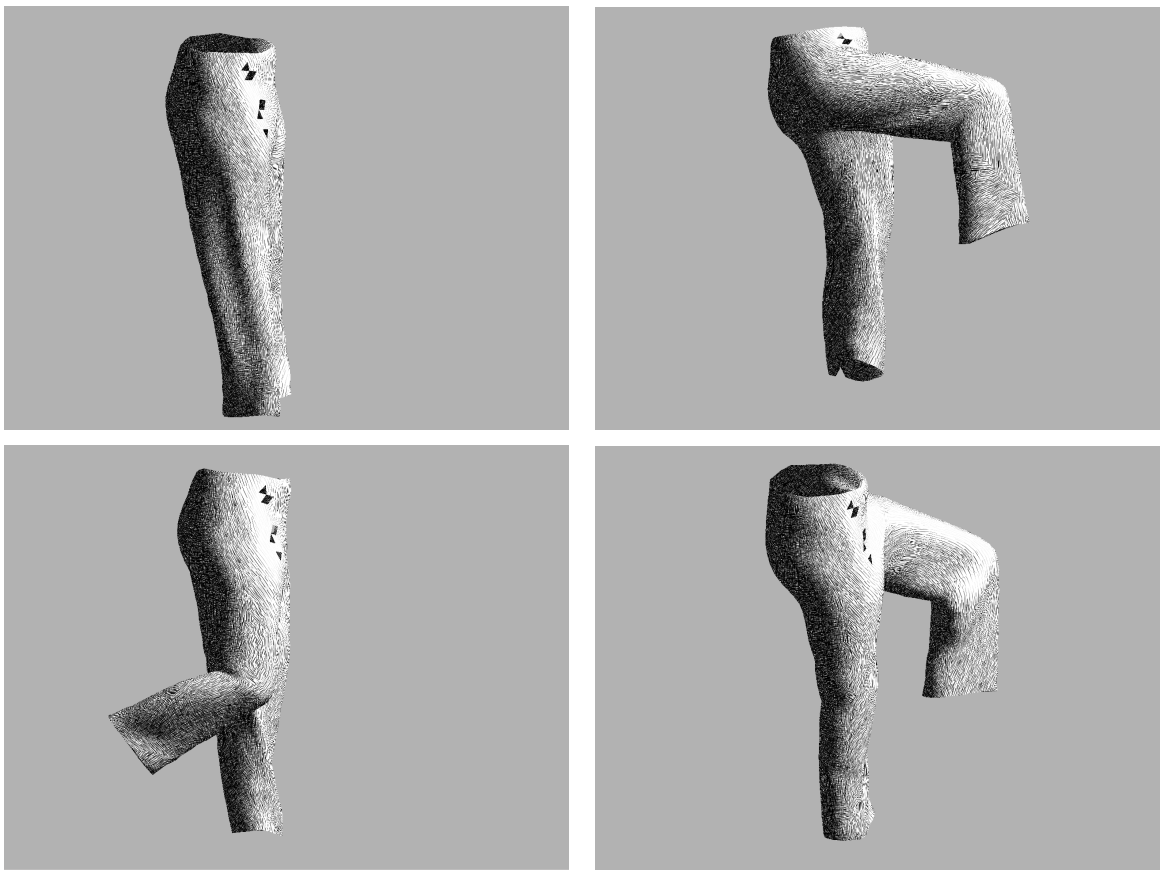


Figura 5.9: Resultados de la animación de movimiento de pantalones.

Capítulo 6

Discusión

6.1. Validación Visual de la Técnica

En los capítulos anteriores se presentaron los resultados de la implementación del algoritmo propuesto en la tesis. Ahora bien, ¿es efectivo el achurado resultante?¿Se puede definir la efectividad de una técnica basada en la percepción visual del observador, que no se conoce a priori?

Dichas preguntas son comunes en las técnicas de rendering no fotorrealista, aunque en general no existen propuestas sobre cómo responderlas. Donde sí ha habido un gran esfuerzo es en el campo de las técnicas basadas en líneas, como *Suggestive Contours* [DFRS]. Se han realizado estudios en los cuales se ha pedido a diferentes personas, tanto artistas como no artistas, que tracen las líneas que ellos creen más representativas. Tal es el estudio de Cole et al. [CGL$^+$08].

Lamentablemente no existe ninguna validación de ese tipo aplicable al achurado. Ante eso, sólo es posible asumir que los resultados visuales, que tienen buen aspecto (entendiéndose esto último como líneas coherentes con la geometría de la superficie), son buenos, puesto que visualmente son agradables.

6.2. Rendering en Tiempo Real

Si bien los resultados presentados tienen un desempeño que califica como excelente, superando los 60 cuadros por segundo al utilizar la técnica con un modelo 3D, el enfoque de la tesis no es generar un algoritmo en tiempo real, debido a que garantizar que la técnica funciona con ese nivel de desempeño es un tema aparte. En este caso, si bien la implementación no se realizó con un enfoque de desempeño máximo, se utilizaron buenas prácticas y se privilegió un diseño elegante, lo que al final desemboca en un buen desempeño.

Un punto a favor del posible desempeño en tiempo real, es que la técnica en la cual se ha basado la tesis presenta ese desempeño. Las extensiones al algoritmo que pueden mermar

el desempeño son interpolaciones lineales en 2D, que no son costosas como para pensar que el desempeño cae drásticamente. Aún así, para hablar de tiempo real es necesario probar la técnica en un ambiente real, con escenas complejas y otros cálculos que se ejecuten simultáneamente, como simulación física e inteligencia artificial.

6.3. Uso de Memoria

En las animaciones de prueba, el uso de memoria era aproximadamente de 8 MB, una cifra muy pequeña considerando las cantidades de memoria actuales, tanto de memoria RAM como de memoria de vídeo. A pesar de ser una cifra baja, el algoritmo presenta una desventaja: el uso de memoria es directamente proporcional a la cantidad de frames. Ahora bien, 125 frames es una cantidad grande para una animación. Por ejemplo, el total de las animaciones de un personaje en un videojuegos podría alcanzar tal cantidad de frames, pero difícilmente superarla, por lo que se puede concluir que el uso de memoria es adecuado.

6.4. Restricciones de la Técnica

El algoritmo planteado asume que la topología de la superficie a achurar se mantiene constante durante la animación. Sin embargo, no se ha considerado el caso en el cual esto no sucede, es decir, donde producto de la animación la topología de la malla se corrompe o simplemente cambia de manera arbitraria. En la actualidad existen escenarios en los cuales ésto sucede de manera esperada, como escenas con interacción física que presentan destrucción (como destruir una pared) y teselación en tiempo de ejecución para mejorar el detalle de los objetos al ser vistos de cerca.

En el caso en que la topología no se mantenga constante, existe la alternativa de utilizar el algoritmo en espacio imagen, propuesto por Kim et al. [KYYL08].

6.5. Comparación con Algoritmo en Espacio Imagen

Respecto al algoritmo diseñado en la tesis, la técnica en espacio imagen no requiere pre-procesamiento y es independiente de lo que se esté graficando. Se quería hacer una comparación de los resultados, y se pidió una implementación de prueba a los autores del algoritmo en espacio imagen, pero la respuesta fue negativa. Por lo tanto, sólo se puede especular respecto de las ventajas y desventajas de dicho algoritmo y del propuesto en la tesis. En algunos escenarios un algoritmo en espacio objeto es más recomendado debido a:

- **Rendimiento**: al tener una etapa de pre-proceso, muchos cálculos repetitivos sólo se hacen una vez. En el algoritmo en espacio imagen, en cada frame se calculan las direcciones principales en las superficies visibles, lo que es computacionalmente costoso y redundante.

- **Flexibilidad**: los algoritmos en espacio objeto se pueden mezclar, e incluso aplicar diferentes técnicas a diferentes objetos.
- **Predictibilidad y Configurabilidad**: en el algoritmo en espacio objeto se tiene control sobre el campo de direcciones y, por lo tanto, es posible configurar el comportamiento de las líneas de achurado.

Por otro lado, en ocasiones donde las restricciones del algoritmo propuesto son una piedra de tope, el algoritmo en espacio imagen debiese ser utilizado. En los otros escenarios, el algoritmo en espacio objeto posiblemente tendrá un mejor desempeño y una calidad visual más controlable por los usuarios.

6.6. Trabajo Futuro

6.6.1. Mejoramiento de la Herramienta

Las áreas en las cual se propone mejorar la herramienta son:

- *Campos de direcciones*: se puede mejorar la estimación de campos de direcciones utilizando otros enfoques, como el de Palacios y Zhang [PZ07]. También se podría considerar otra estrategia de generación de campos de dirección, como la propuesta por Xu et al. [XCOJ$^+$], que suaviza un campo de direcciones pero restringiendo los vectores de dirección de acuerdo a las crestas y valles de una superficie. Un trabajo actual es el de Krane et al. [CDS10], que proponen un método lineal que permite obtener campos direccionales suaves de manera interactiva. Afortunadamente, este último trabajo ha publicado código fuente para que sea estudiado por la comunidad, por lo que es el enfoque con más potencial de ser utilizado para mejorar la herramienta de la tesis.
- *Lapped Textures*: se puede diseñar una métrica para análisis de deformación en los parches generados por la técnica, con el fin de obtener los mejores parches posibles dada una superficie y una animación. Además se pueden mejorar las limitaciones de la implementación, en particular se debería mejorar la calidad visual al implementar el *overlapping* entre parches y la especificación de formas de parches distintas a círculos.
- *Interactividad*: es deseable que el usuario tenga más control del resultado final, mediante la especificación de aristas restringidas en la segmentación y de manipulación del campo de direcciones utilizando gestos con el puntero del mouse.
- *Graficación de escenas*: para aprovechar el framework gráfico en el cual se basó la tesis, una aplicación interesante es la graficación de escenas completas, tanto con achurado como con diferentes estilos visuales.

6.6.2. Almacenamiento de Animaciones

Como se mencionó en el Capítulo anterior, la textura de achurado se almacena y anima utilizando la técnica Vertex Animation en 2D. Debido a esto, el uso de memoria utilizada por los datos del achurado es directamente proporcional a la cantidad de keyframes y animaciones, por lo que podría ser necesario un método para comprimir la animación de la parametrización. Existe una técnica llamada *Skinning Arbitrary Deformations*, de Kavan et al. [KMD+07], que recibe como entrada una animación de tipo Morph Target y entrega como salida una malla con un conjunto de huesos, sin relaciones jerárquicas entre ellos (o bien, todos hijos de un mismo hueso que no tiene transformación asociada), como muestra la Figura 6.1. Para cada hueso se especifican las traslaciones y orientaciones necesarias para que en cada frame se reconstruya la superficie de manera fiel. Esta técnica permite compactar la animación, de modo que para cada frame la parametrización se representa como un conjunto de transformaciones para los huesos. En algunos casos la técnica logra reducir hasta 17 veces la memoria necesaria para almacenar una animación.

Lo que se propone como trabajo futuro es considerar la animación de la parametrización como una animación en 2D que debe ser comprimida utilizando una técnica S.A.D. en 2D. Esto permitirá expresar los cambios en la parametrización de la malla en cada frame en función de cambios en ángulos y traslaciones de huesos en el espacio de la textura.

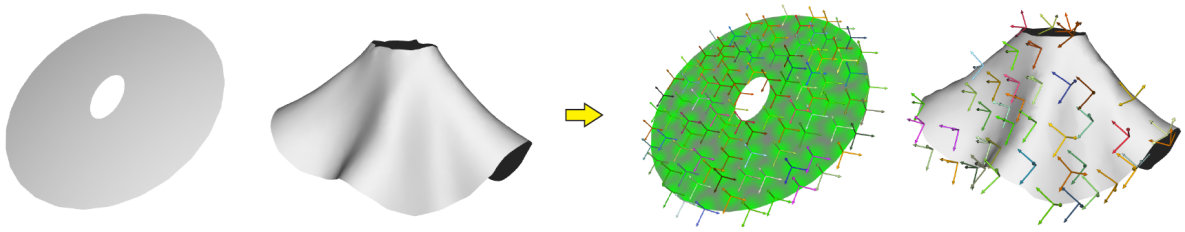


Figura 6.1: Ilustración del resultado de Skinning Arbitrary Deformations: dada una forma original y una deformación, se computa una formulación de la deformación utilizando Skinning. Fuente: [KMD+07].

Como es necesario validar que la animación comprimida reconstruya fielmente la original, es necesaria una métrica que entregue el porcentaje de error. Esta métrica está definida por James y Twigg [JT] en un trabajo previo a S.A.D. que también comprime animaciones, pero que se enfoca en el caso específico de animaciones de personajes bípedos y cuadrúpedos.

6.6.3. Deformación Arbitraria

Otro problema pendiente es el achurado para objetos con animación arbitraria, de la cual no se tiene conocimiento a priori, como puede ser la manipulación de un esqueleto en tiempo real por un usuario o mediante una simulación física. El enfoque que se ha planteado es insuficiente para estos casos porque requiere que se conozca información sobre el tiempo posterior a la animación.

Ya que no se conoce información que permita construir la información del achurado que falta, se puede adoptar un enfoque de *machine learning*, en el cual se crea un modelo

predictivo de dicha información. Kalogerakis et al. [KNS+08] realizaron un modelo predictivo, que funciona en tiempo real, para información de curvatura y direcciones principales, que si bien no es aplicable directamente en este caso, es un buen punto de partida. Este modelo recibe las orientaciones de los huesos de un objeto y, en base a ellas y a los datos de entrenamiento (que pueden ser animaciones ya conocidas del objeto) predice los valores de curvatura en la superficie. Los autores de dicho trabajo también consideran que, como trabajo futuro, su técnica puede ser aplicada al achurado en tiempo real.

6.7. Conclusiones

Se ha presentado un algoritmo de achurado que se puede aplicar a objetos con animaciones conocidas, expresadas mediante keyframes de animación o como una animación para un objeto con esqueleto. El algoritmo propuesto es una extensión al algoritmo original de achurado propuesto por Praun et al. [PHWF]. La presentación del algoritmo incluye el diseño, las herramientas utilizadas como base y la implementación.

La tesis comenzó como una aventura en la cual se deseaba desarrollar un algoritmo de achurado para todo tipo de animaciones. No se consideró que el algoritmo de achurado para objetos animados fuese diferente para distintos tipos de animaciones, lo cual alargó el planteamiento del algoritmo y provocó varios problemas a la hora de realizar el marco teórico y el diseño del algoritmo. Pero el escrito mismo de la tesis ha planteado un desafío interesante: la técnica presentada, de manera similar a muchas técnicas de Computación Gráfica, incluyendo otras de NPR, fue realizada a partir de la intuición sobre cómo un algoritmo de achurado podría funcionar, y se llegó a un resultado coherente, donde cada decisión se ha justificado citando fuentes y utilizando parámetros y técnicas de ingeniería.

Al algoritmo desarrollado le falta trabajo para ser considerado completo. Tal como se ha indicado en el trabajo futuro, todavía hay mucho por hacer. Sin embargo, se tiene la convicción de que se ha entregado una buena base sobre la cual seguir trabajando. El haber utilizado software libre para realizar la implementación, permitirá compartir con la comunidad el trabajo desarrollado, a modo de recibir feedback y, posiblemente, realizar aportes a las bibliotecas utilizadas.

Bibliografía

[AMH08] E. Akenine-Moller, T. Haines and N. Hoffman. *Real-Time Rendering*. AK Peters, Ltd. Natick, MA, USA, 2008.

[BBA08] William Baxter, Pascal Barla, and Ken-ichi Anjyo. Rigid shape interpolation using normal equations. In *NPAR '08: Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, pages 59–64. ACM Press, June 2008.

[BKTS06] Adrien Bousseau, Matthew Kaplan, Joëlle Thollot, and François Sillion. Interactive watercolor rendering with temporal coherence and abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. ACM, 2006.

[BSBK02] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh-a generic and efficient polygon mesh data structure. *OpenSG Symposium*, 2002.

[CDS10] Keenan Crane, Mathieu Desbrun, and Peter Schröder. Trivial connections on discrete surfaces. *Computer Graphics Forum (SGP)*, 29(5):1525–1533, 2010.

[CGL+08] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 27(3), August 2008.

[dC76] M.P. do Carmo. *Differential geometry of curves and surfaces*. Prentice-Hall Englewood Cliffs, NJ, 1976.

[DFR04] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 15–145, New York, NY, USA, 2004. ACM.

[DFRS] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 848–855, New York, NY, USA. ACM.

[DR05] Christopher DeCoro and Szymon Rusinkiewicz. Pose-independent simplification of articulated meshes. In *Symposium on Interactive 3D Graphics*, April 2005.

[FGK+00] A. Fabri, G.J. Giezeman, L. Kettner, S. Schirra, and S. Sch
'onherr. On the design of CGAL a computational geometry algorithms library. *Software Practice and Experience*, 30(11):1167–1202, 2000.

[GGSC] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA. ACM.

[Gra08] Eduardo Graells. Marco de trabajo de rendering no fotorrealista. Memoria para optar al título de Ingeniero Civil en Computación, Universidad de Chile, Mayo 2008.

[Her] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, New York, NY, USA. ACM.

[HLS] Kai Hormann, Bruno Lévy, and Alla Sheffer. Mesh parameterization: theory and practice. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 1, New York, NY, USA. ACM.

[HZ] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Int] Victoria Interrante. Illustrating surface shape in volume data via principal direction-driven 3d line integral convolution. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 109–116, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[JDA] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 19, New York, NY, USA. ACM.

[JG] B. Jacob and G. Guennebaud. Eigen. `http://eigen.tuxfamily.org/index.php?title=Main_Page`.

[JT] Doug L. James and Christopher D. Twigg. Skinning mesh animations. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 399–407, New York, NY, USA. ACM.

[KCvO07] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, New York, NY, USA, 2007. ACM.

[Ket98] L. Kettner. Designing a data structure for polyhedral surfaces. *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 146–154, 1998.

[KMD+07] Ladislav Kavan, Rachel McDonnell, Simon Dobbyn, Jiří Žára, and Carol O'Sullivan. Skinning arbitrary deformations. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 53–60, New York, NY, USA, 2007. ACM.

[KNS+08] Evangelos Kalogerakis, Derek Nowrouzezahrai, Patricio Simari, James McCrae, Aaron Hertzmann, and Karan Singh. Real-time line drawing for animated surfaces. Technical Report CSRG-571, Computer Science Department, University of Toronto, March 2008.

[KYYL08] Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. Line-art illustration of dynamic and specular surfaces. *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA)*, ?(?), December 2008.

[Lan98] J. Lander. Skin them bones: Game programming for the web generation. *Game Developer Magazine*, 5:11–16, 1998.

[LMHB00] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 13–20, New York, NY, USA, 2000. ACM.

[Max99] Nelson Max. Weights for computing vertex normals from facet normals. *J. Graph. Tools*, 4(2):1–6, 1999.

[PFH] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[PHWF] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581, New York, NY, USA. ACM.

[PZ07] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3):55, 2007.

[Rit90] J. Ritter. An efficient bounding sphere. In *Graphics gems*, page 303. Academic Press Professional, Inc., 1990.

[Rus] S. Rusinkiewicz. trimesh2. `http://www.cs.princeton.edu/gfx/proj/trimesh2/`.

[Rus04] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*, 2004.

[S+97] B. Stroustrup et al. *The C++ programming language*. Addison-Wesley Reading, MA, 1997.

[She94] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[ST90] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206, 1990.

[SWND03] D. Shreiner, M. Woo, J. Neider, and T. Davis. The OpenGL Programming Guide (Red Book), 2003.

[TLHD03] Yiying Tong, Santiago Lombeyda, Anil N. Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graph.*, 22(3):445–452, 2003.

[Tur90] K. Turkowski. Transformations of surface normal vectors. Technical report, Tech. Rep. 22, Apple Computer, July, 1990.

[VBMP] Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popović. Articulated mesh animation from multi-view silhouettes. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–9, New York, NY, USA. ACM.

[WCF07] Ryan White, Keenan Crane, and David Forsyth. Capturing and animating occluded cloth. In *ACM Transactions on Graphics (SIGGRAPH)*, 2007.

[WE85] K. Weiler and G. Electric. Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. *Computer Graphics and Applications, IEEE*, 5(1):21–40, 1985.

[WK] J. Walter and M. Koch. Boost basic linear algebra. `http://www.boost.org/doc/libs/1_43_0/libs/numeric/ublas/doc/index.htm`.

[WPFH02] Matthew Webb, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Fine tone control in hardware hatching. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 53–ff, New York, NY, USA, 2002. ACM.

[XCOJ+] Kai Xu, Daniel Cohne-Or, Tao Ju, Ligang Liu, Hao Zhang, Shizhe Zhou, and Yueshan Xiong. Feature-aligned shape texturing.

Apéndice A

Implementación del Algoritmo en DVD

Junto a la tesis se incluye un DVD que contiene todo el código de la implementación realizada, en conjunto con datos y aplicaciones de prueba. Para compilar la aplicación se recomienda utilizar una distribución de GNU/Linux, como puede ser Ubuntu[1].

Además de la versión en DVD, la implementación actualizada se puede encontrar en `http://egraells.org`.

También se requiere el compilador de C++ de GNU, g++, y las bibliotecas GLEW[2], Cg Toolkit[3] y Boost[4]. La compilación se administra con el sistema de compilación CMake[5].

El disco contiene:

- Carpeta `src`: código fuente.
- Carpeta `data`: datos de prueba, animaciones del Capítulo 5.
- Carpeta `doc`: la versión electrónica de la tesis, en conjunto con otros documentos relevantes.

Las aplicaciones que se pueden ejecutar son:

- `hatching` es la implementación del algoritmo original de achurado. Como parámetro en línea de comandos recibe el modelo 3D a visualizar en formato .OBJ.
- `anim` es la implementación del algoritmo de achurado para objetos animados. Como parámetros en la línea de comandos recibe el prefijo de los modelos 3D a cargar y los números de frame inicial y final. Ver ejemplo de ejecución más adelante.
- `test_performance` permite saber cuánto se demora el pre-proceso de un conjunto de mallas de triángulos.

[1] `http://kubuntu.org`
[2] `http://glew.sourceforge.net/`
[3] `http://developer.nvidia.com/object/cg_toolkit.html`
[4] `http://www.boost.org/`
[5] `http://www.cmake.org/`

Una vez que se han instalado las bibliotecas necesarias, hay que editar el archivo `CMakeLists.txt` en la carpeta del código fuente. Allí hay que configurar la carpeta en la cual se encuentra Zahir (el framework de la tesis), con el fin de que al compilar no haya problemas en incluir los archivos. En particular, las líneas que hay que editar son las siguientes:

```
include_directories (
  /home/egraells/resources/eigen # carpeta de Eigen
  ~/proyectos/cpp/new_zahir/ # raíz de Zahir
  ~/proyectos/cpp/new_zahir/Zahir/External # bibliotecas externas
)
```

Una vez editado el archivo, se debe generar un archivo Makefile para poder realizar la compilación. Los comandos son los siguientes:

```
cd zahir
cmake -G ‘‘Unix Makefiles’’
make
```

La carpeta `zahir` es la que contiene todo el fuente, copiada en el disco duro u otra ubicación en la cual se pueda escribir.

Antes de ejecutar cualquier programa que venga incluido con Zahir, es necesario explicitar la carpeta en la cual se encuentra instalado el framework. Así:

```
export ZAHIR_PATH=~/proyectos/cpp/zahir/
./hatching ~/resources/modelos3d/PrimSphere.obj
./anim_hatching ~/proyectos/magister-thesis/datos/samba/meshes/mesh_ 0 40
```