**Revista Venezolana de Computación**
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 8, No. 1, pp. 25-36, Junio 2021
Fecha recepción: 30/01/2021, Fecha aceptación: 25/07/2021

# Extracción de Esqueleto por Contracción de Mallado con Asignación Automática de Pesos

Moisés E. Simón A. Berenguer Gómez, Francisco José Moreno Alvarez
moises.berenguer@ciens.ucv.ve, francisco.j.moreno@ucv.ve

Centro de Computación Gráfica, Universidad Central de Venezuela, Caracas, Venezuela

**Resumen:** En este trabajo se propone un método para realizar la extracción del esqueleto de un objeto tridimensional definido por una malla geométrica. La propuesta se basa en una investigación previa que utiliza el *suavizado laplaciano* para contraer un modelo hasta formar su esqueleto curvo. En esta, se evalúan un conjunto de modificaciones para incrementar su eficiencia. Los cambios planteados, se basan en pruebas aplicadas que determinaron la combinación óptima entre: la selección del método de resolución del sistema de ecuaciones que determina la contracción del mallado y, el algoritmo de permutaciones que se utiliza en conjunción a él. Debido a la naturaleza paralela de las fases que componen el algoritmo, la solución es desarrollada tanto en CPU como en GPU, utilizando la arquitectura propietaria de NVIDIA, CUDA. Además, se complementa el algoritmo original con un conjunto de etapas posteriores que adaptan el esqueleto resultante para su uso directo en aplicaciones de diseño 3D, permitiendo deformar y realizar animaciones al modelo original a partir del esqueleto generado. Finalmente, se compara la eficiencia del algoritmo propuesto contra el original, procesando objetos tridimensionales con distinto conteo de polígonos y características topológicas. Como resultado, la implementación desarrollada en CPU es 1.7 veces más rápida que el método original y la desarrollada en GPU es 5.4 veces más rápida, donde esta última exhibe un crecimiento lineal ante el incremento de los polígonos de la malla original, a diferencia del comportamiento exponencial de las soluciones en CPU.

**Palabras Clave:** Extracción de Esqueleto; Suavizado Laplaciano; Asignación de Pesos.

**Abstract:** In the current study a set of modifications to a previous mesh-based skeleton extraction technique, which compressed a mesh into its curved skeleton by applying a laplacian smoothing, are presented. All the suggested changes are supported by efficiency tests that determined the optimal combination between several linear equation solvers and their matrix permutation algorithms. The proposed method was developed and tested in CPU and GPU using the proprietary architecture of NVIDIA, CUDA. Also, a set of post-processing steps were included to enable its use as an animation tool in any 3D design suite. To prove the robustness of the algorithm, all tests were applied over a group of 3D models with an increasingly polygon count and topological complexity. As a result, the proposed method proves to be 1.7 times faster than the original in its CPU version and exhibits an exponential time growth as the number of processed polygons increases. Likewise, the GPU adaptation is 5.4 times faster and exhibits a linear-time growth.

**Keywords:** Skeleton Extraction; Laplacian Smoothing; Weight Assigments.

## I. Introducción

Un esqueleto curvo es una representación simplificada de la geometría y topología de un objeto, su estructura es utilizada para navegar escenas al generar un camino libre de colisiones sobre la misma [1], propiedad que explota el área médica, para navegar virtualmente a través de los órganos del cuerpo humano, abarcando procedimientos tradicionalmente invasivos como: colonoscopia [2], broncoscopia [3], angioscopia [4], entre otros. También resulta de utilidad para segmentar información médica proveniente de resonancias magnéticas [5][6], detectar calcificaciones y aneurisma [7], alinear múltiples vistas de un objeto (utilizando el esqueleto como punto de referencia) [8][9], metamorfosis (transformar un objeto en otro) [10], animaciones [11][12], entre otros.

Dado el amplio abanico de casos de uso de los esqueletos curvos en distintas áreas de investigación, en este trabajo se plantea una alternativa para extraer el esqueleto curvo a partir de mallados geométricos. Se utiliza como base el método propuesto por Au et al. [13], al cual se le modifican algunas de sus etapas para incrementar su eficiencia. A partir de las modificaciones propuestas, se experimentó con múltiples técnicas en cada una de las etapas seleccionadas, lo que permitió evaluar el rendimiento de cada técnica bajo las mismas condiciones y seleccionar la combinación que demuestra el mejor rendimiento. Finalmente, se agregan las

fases necesarias al método optimizado, para que su resultado se pueda utilizar de forma transparente en cualquier software de diseño tridimensional, para animar y manipular (mediante el esqueleto) el mallado original.

El resto del trabajo se presenta en 5 secciones. La Sección II describe brevemente algunos trabajos relacionados a la propuesta que se presenta en la Sección III. La Sección IV plantea las pruebas realizadas al sistema para obtener la combinación de técnicas óptimas para generar el esqueleto curvo. Finalmente, la Sección V expone las conclusiones y trabajos futuros.

## II. Trabajos Relacionados

Una forma u objeto normalmente es descrita por los límites que conforman su silueta. Blum [14], describe un objeto mediante un conjunto de discos ubicados en su interior, cada disco está alineado al eje central del objeto y abarca su volumen, la estructura que definen corresponde con el esqueleto o estructura central de un objeto.

A partir de la definición anterior, surgen múltiples técnicas para generar el esqueleto de un objeto de manera automática, en las que el esqueleto (de volumen aproximado a cero) de un modelo tridimensional, se puede calcular mediante un modelo matemático diseñado para este propósito [15]. El objeto tridimensional puede ser representado mediante: un mallado [1], una nube de puntos, una representación topográfica o volumétrica, entre otras. El proceso de extraer el esqueleto de un modelo tridimensional es denominado "esqueletonización".

Las diversas técnicas de esqueletonización difieren en aspectos teóricos como la definición del esqueleto y, en aspectos prácticos como: la discretización del espacio (vóxeles[2] vs mallas), los algoritmos de extracción de esqueleto y los parámetros que establecen. Cada uno de estos aspectos y la sensibilidad inherente de los esqueletos al ruido en la representación del objeto, hace que cada método produzca esqueletos diferentes para el mismo modelo base, lo que causa variaciones en el desempeño al utilizar el esqueleto generado y plantea un reto de investigación que se mantiene en la actualidad.

### A. Métodos Volumétricos

El proceso de esqueletonización basado en volumen (*Voxel Based Methods*), extrae el esqueleto de un modelo tridimensional haciendo uso de voxelización. Voxelizar, involucra englobar todo el modelo en una caja contenedora o vóxel, que luego se subdivide iterativamente hasta alcanzar una representación volumétrica aproximada del objeto original. Posteriormente, se aplica un proceso (que varía entre cada autor) para conseguir la representación correspondiente al único camino de vóxeles que expone todas las características topológicas del modelo inicial. Según Bertrand y Aktouf [16], el camino resultante se denomina esqueleto curvo y solo describe la silueta del objeto al que pertenece, sin nodos o información adicional. Por ende, es necesario aplicar una última etapa en la que se agregan las características necesarias que adaptan el esqueleto a su caso de uso particular.

Los métodos de adelgazamiento y propagación de límites o *thinning*, se basan en el concepto de punto simple [17] y generan el esqueleto curvo removiendo capa a capa los vóxeles que conforman el objeto, hasta conseguir la representación más delgada posible.

Los métodos de adelgazamiento direccional solo eliminan los vóxeles de una dirección particular (norte, sur, arriba, abajo, este y oeste) en cada iteración, variando la cantidad de direcciones y condiciones para identificar los puntos finales [18]. Estos métodos son sensibles al orden en que se procesan las direcciones y, los esqueletos resultantes pueden no estar centrados dentro del modelo.

Los métodos de adelgazamiento secuenciales dividen el espacio discreto en varios subconjuntos denominados subcampos. En cada iteración, se selecciona uno de ellos y se eliminan los vóxeles pertenecientes a él. Se puede usar un número diferente de subcampos en tres dimensiones: 2, 4 y 8 [19]. Según Borgefors et al. [20], estos algoritmos consideran para su eliminación, a todos los puntos límite en una iteración de adelgazamiento, pero para mantener la topología, el vecindario debe ser inspeccionado para decidir si un vóxel debe eliminarse.

### B. Métodos Geométricos

La esqueletonización basada en geometría (*Mesh Based Methods*) procesa directamente la malla que representa el objeto tridimensional. No requiere de una representación intermedia (vóxeles), a diferencia de la basada en volumen.

Au et al. [13] proponen un método que contrae la geometría aplicando iterativamente un suavizado laplaciano que elimina el ruido sobre la superficie y sus detalles desplazando los vértices en dirección a las normales. Si dicho proceso se aplica sin ningún tipo de restricción, los vértices del mallado se suavizarían progresivamente hasta converger en un solo punto, por lo cual se utilizan puntos de anclaje para restringir el desplazamiento de los vértices. Como resultado, se obtiene un mallado degenerado de volumen aproximado a cero, que es la contracción máxima que se puede realizar sobre el objeto sin perder sus características principales. Ya que el resultado final del algoritmo es el esqueleto curvo del mallado, debe ser complementado con etapas que agreguen las características necesarias para el caso de uso particular en que el que será empleado. Por ejemplo, si el esqueleto se utilizará en una animación, es necesario segmentarlo en articulaciones y extremidades.

En esta investigación, se modifica el método propuesto por Au et al. [13], alterando algunas de sus fases con la intención de mejorar su eficiencia. Principalmente, se analiza la forma de los sistemas de ecuación que se involucran en la etapa de

[1] Colección de triángulos y vértices que aproximan una superficie en tres dimensiones.

[2] Vóxel (*Volume Pixel*): Representación mínima de un elemento en una matriz en tres dimensiones. Equivalente 3D de un píxel en 2D.

contracción geométrica y se proponen alternativas para obtener las soluciones.

## III. Propuesta

El algoritmo que se propone a continuación, construye el esqueleto[3] de un objeto a partir de la malla que representa su geometría. Además, el esqueleto se complementa con la información necesaria para ser utilizado en el contexto de animaciones por computadora. Para cumplir este objetivo, se descompone el procedimiento en las siguientes etapas:

1) Contracción de Geometría: Implementación y análisis a fondo del algoritmo planteado por Au et al. [13], con la finalidad de generar el esqueleto curvo partiendo de la geometría inicial.
2) Simplificación: Filtrado de las aristas del esqueleto generado en la etapa previa.
3) Refinamiento: Reubicación de cada nodo del esqueleto curvo al centro local correspondiente en el mallado.
4) Selección: Identificación de los huesos relevantes del modelo que puedan ser empleados como herramienta de animación del mallado original.
5) Asignación de Pesos: Cálculo del grado de influencia de los huesos sobre los vértices del mallado utilizando [22].
6) Exportado: Empaquetado del resultado de las etapas previas en un archivo formato FBX. Este formato define el estándar de representación geométrica soportado por cualquier programa de diseño tridimensional como: Blender, Maya o 3DS.

En la Figura 1 se pueden apreciar todos los pasos aplicados sobre uno de los modelos de prueba *Raptor*.

### A. Contracción de Geometría

Inicialmente se contrae el mallado hasta obtener su esqueleto curvo aplicando iterativamente el suavizado laplaciano con restricciones descrito por Au et al. [13], el cual sugiere el uso del operador laplaciano definido en la ecuación (1).

$$L_{ij} = \begin{cases} \omega_{ij} = cot(\alpha_{ij}) + cot(\beta_{ij}) & (i,j) \in E \\ \sum_{(i,k)\in E}^{k} -w_{ik} & i = j \\ 0 & otro \end{cases} \quad (1)$$

Donde según Desbrun et al. [23] $\alpha_{ij}$ y $\beta_{ij}$ son los ángulos opuestos correspondientes a la arista $(i,j)$, siendo $E$ el conjunto que representa las aristas de la malla.

Aplicar la ecuación de Laplace en repetidas ocasiones causará la pérdida progresiva de características del modelo inicial, para evitar este comportamiento se introducen dos variables adicionales a la ecuación: $W_l$ y $W_h$ que acentúan y relajan el grado de contracción respectivamente. $W_l$ con dirección al operador laplaciano y $W_h$ en dirección a la malla original. Con este planteamiento se define el sistema de ecuaciones (2) como modelo matemático de optimización, que permite contraer la geometría y preservar las características del modelo.

$$\begin{bmatrix} W_l L \\ W_h \end{bmatrix} V' = \begin{bmatrix} 0 \\ W_h V \end{bmatrix} \quad (2)$$

Las matrices $W$ (o matrices de peso) tienen la característica de ser diagonales y contienen los valores de fuerza de contracción y atracción locales a cada uno de los vértices. Es decir que las entradas $(i,i)$ de la matriz, contienen información sobre las fuerzas, basándose únicamente en su posición y en la de sus vecinos directos sin involucrar al resto de vértices.

Al ser un proceso iterativo, requiere de varias ejecuciones para obtener el esqueleto curvo de volumen cero aproximado. Por tanto, $W_l$ y $W_h$ deben actualizarse antes de la siguiente iteración. Au et al. [13] sugieren los siguientes pasos a realizar:

1) Solucionar (2) para $V^{t+1}$.
2) Actualizar $W_l$ tal que: $W_l^{t+1} = s_l W_l^t$.
3) Actualizar $W_{h,i}$ tal que: $W_{h,i}^{t+1} = W_{h,i}^0 \sqrt{A_i^0 / A_i^t}$.
4) Calcular el nuevo operador laplaciano $L^{t+1}$ con las posiciones de vértices actuales $V^{t+1}$ utilizando (1).

Analizando la matriz $L$, se observa que contiene una gran cantidad de entradas nulas. Esto se debe a que por definición (1), sus valores se concentran en la diagonal y los índices $(i,j)$ que conforman una arista en el mallado. En otras palabras la forma de la matriz $L$, es la misma que la matriz de adyacencia del modelo, junto con valores en la diagonal.

En la matriz laplaciana de la Figura 2 se puede observar que los elementos no nulos se encuentran cercanos a la diagonal y que gran parte de la matriz son valores nulos. Es por esto que la estructura de datos utilizada en implementación, debe ser una matriz esparcida. Además, tomando en cuenta que la matriz es del tamaño de la cantidad de vértices en el modelo, utilizar otro tipo de representación no sería aconsejable.

Continuando con el análisis del sistema de ecuaciones (2), definamos (3) y (4):

$$A = \begin{bmatrix} W_L L \\ W_H \end{bmatrix} \quad (3)$$

$$B = \begin{bmatrix} 0 \\ W_H V \end{bmatrix} \quad (4)$$

Estas definiciones permiten construir el sistema de ecuaciones sobredeterminado (tiene más ecuaciones que incógnitas) $Ax = B$. Donde $A$ es una matriz rectangular, cuya forma es descrita por la Figura 3 y $B$ son 3 vectores de lado derecho (*rhs, right hand side*), por lo que se plantea obtener su solución por medio de mínimos cuadrados (5), cuya forma es descrita en la segunda matriz de la Figura 2.

$$A^t A x = A^t B \quad (5)$$

[3]Según Thalmann y Thalmann [21], un esqueleto es una estructura compuesta de nodos y huesos que se asemeja a un esqueleto humano, donde los nodos representan las articulaciones, los huesos las extremidades y a su vez las articulaciones son la intersección de dos extremidades.

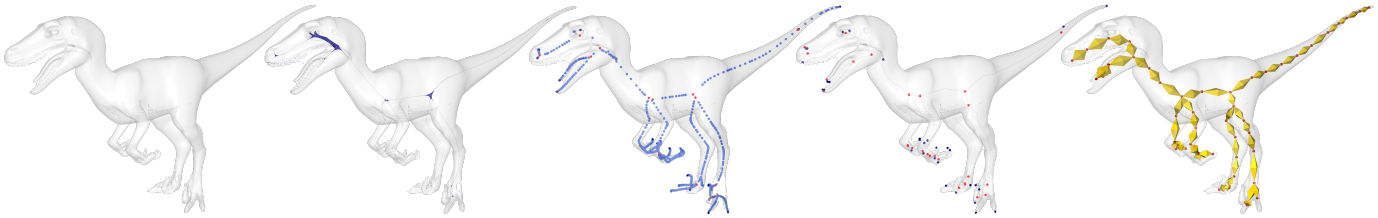

**Figura 1:** De Izquierda a Derecha cada Etapa para la Generación del Esqueleto: Modelo Inicial, Contracción de Geometría, Simplificación, Refinamiento, Selección

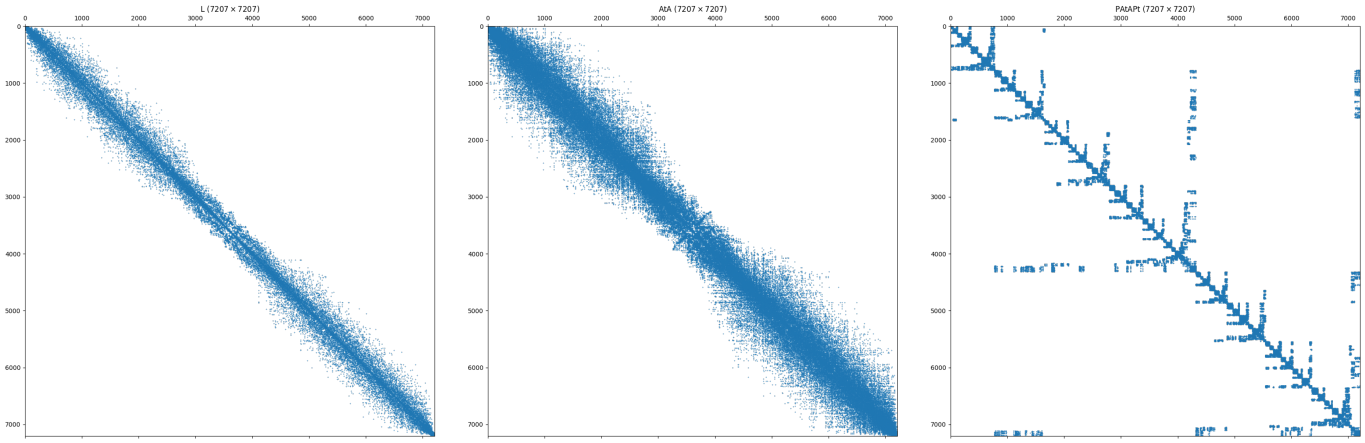


**Figura 2:** Matrices Involucradas en la Contracción Geométrica. De Izquierda a Derecha las Matrices: Laplaciana, $A^tA$ y $PA^tAP^t$ Usando la Permutación METIS. Todas Generadas para el Modelo *Cat*

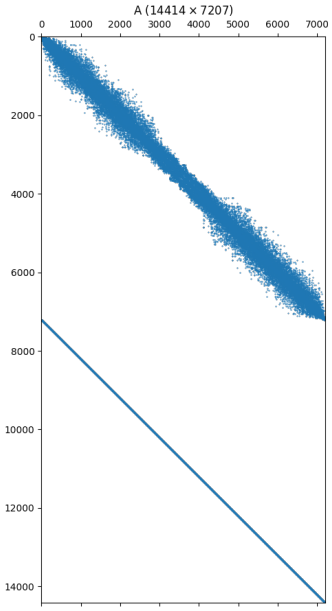


**Figura 3:** Forma de la Matriz $A$

Al utilizar el planteamiento de mínimos cuadrados, las condiciones del sistema *(matriz simétrica-positiva-definida)* resultante, son óptimas para aplicar diversos algoritmos de resolución de ecuaciones como: factorización Cholesky, LU o incluso QR. También se pueden utilizar métodos iterativos, en especial los basados en multigrilla [24][25], como lo expone Au et al. [13].

En la propuesta actual se evaluó el rendimiento de diversos métodos para resolver el sistema ecuaciones. Además, los métodos para solucionar (5) pueden ser paralelizados, por lo cual se desarrollaron alternativas que se ejecutan en la tarjeta gráfica, haciendo uso de la librería CUDA de NVIDIA.

Ya que el sistema de ecuaciones a resolver está representado en una matriz esparcida, su acceso es un cuello de botella en el rendimiento del algoritmo. El formato utilizado es CSC (*Compressed Sparse Column*) [26], el cual comprime la matriz ordenando los elementos no nulos de forma contigua en memoria por columnas y, de esta manera, minimiza la cantidad de memoria necesaria para almacenar grandes cantidades de datos.

Igualmente, la forma de acceder a las filas y columnas de la matriz difiere entre los distintos métodos para solucionar el sistema de ecuaciones, por lo cual dichos métodos son empleados en conjunción a una matriz de permutaciones $P$ que permite reordenar la matriz inicial simétricamente ($PA^tAP^t$). Dicho reordenamiento, permite a los métodos de resolución, trabajar sin necesidad de permutar la matriz original [4], es por esto, que el tipo de ordenamiento juega un factor importante en la eficiencia del algoritmo final y es considerado en las pruebas que se realizan al sistema. Finalmente, la misma permutación aplicada a $A$ debe aplicarse al vector de resultados del sistema. Luego de resolver el sistema, se aplica la permutación inversa para conservar el orden original del vector de resultados.

Existen diversos algoritmos de permutación para la resolución

[4]La permutación en una factorización de matrices, es una operación elemental que permite organizar los elementos de la mejor forma para ejecutar los pasos involucrados en ella.

de (5), se utilizaron los siguientes:

- **AMD** (*Approximate Minimun Degree ordering*) u ordenamiento de Grado Mínimo Aproximado: Pivota los valores concentrándolos bajo o sobre la diagonal [27].
- **RCM** (*Reverse Cuthill-McKee permutation*) o permutación Reversa de Cuthill-McKee: Concentra los elementos no nulos cerca de la diagonal [28].
- **MDQ** (*Minimum Degree by Quotient graph permutation*) o permutación de Grado Mínimo por grafo de Cociente: Variación del método AMD [25], con la misma finalidad [29].
- **METIS**: Permutación de matrices esparcidas por partición de grafos [30].

A nivel de implementación de todo el proceso de contracción, existen detalles que son necesarios aclarar:

1) Los objetos procesados están compuestos por triángulos y la suma de los ángulos internos de un triángulo es de 180 grados. Cuando los triángulos se comprimen para alcanzar un volumen cercano a cero, sus ángulos internos varían radicalmente. Ya que los cálculos involucran el uso de la función cotangente, la cual tiende al infinito negativo y positivo para ángulos cercanos a 0 y 180 grados respectivamente, es necesario limitar los ángulos internos de los triángulos durante el proceso de compresión para mantener la estabilidad del algoritmo.
2) Los parámetros iniciales para las matrices de peso son: $W_h^0 = 1$, $W_l^0 = 10^{-3}\sqrt{A}$ (donde $A$ representa el área inicial promedio de las caras del modelo) y $sl = 2$, todos manteniendo los mismos valores indicados por Au et al. [13]. Cabe destacar que estos parámetros pueden ser modificados, pero estos valores por defecto generan resultados homogéneos independientemente del modelo o de su escala.
3) Las matrices diagonales $W_h$ y $W_l$ incrementan todos sus valores en cada una de las iteraciones aplicadas en la contracción. Por ello, se limita su crecimiento para prevenir un efecto de "sobrecontracción" que pueda afectar la solución de las ecuaciones y eliminar las características principales del objeto. Se proponen valores límites de: $W_l <= 2048$ y $W_h <= 10000$ y, al igual que el caso anterior, estos valores pueden ser modificados, pero los indicados por defecto logran adaptarse bien a todos los modelos de forma homogénea.

### *B. Simplificación*

El proceso previo desplaza los vértices que componen la malla del objeto hasta alcanzar un volumen cercano a cero, pero no modifica bajo ninguna circunstancia su estructura, por ende, se necesita un procedimiento adicional que disminuya el número de vértices y aristas de la geometría resultante hasta conseguir los nodos claves (centrales) del esqueleto. Este proceso se conoce como "Simplificación" y consta de dos partes fundamentales: primero colapsa la mayor cantidad de vértices posibles que mantengan la estructura conexa y luego selecciona vértices y aristas, que bajo condiciones avanzadas preservan los nodos claves. De forma más detallada, los pasos del algoritmo son los siguientes:

1) Realizar una simplificación natural, que puede ser resumida en remover las aristas que sean tan cortas en el mallado, que su ausencia no represente ninguna perdida de características. En líneas generales, se define una distancia mínima constante que una arista debe sobrepasar, para que su eliminación represente un cambio visual notorio.
2) Seguidamente se aplica el algoritmo de simplificación de mallado QEM. Como indica Au et al. [13], QEM es modificado para que opere correctamente sobre vértices y aristas muy cercanos.

En esta propuesta, se realiza un paso previo al utilizado en [13], donde se agrupan todas las aristas del mallado en una cola de prioridad, tomando como valor referencial su longitud y, en cada iteración, se eliminan únicamente aquellas con longitud menor o igual a un $\delta$ especificado. Este proceso se repite descartando las aristas que superen ese tamaño hasta que eventualmente la cola quede vacía. El resultado de esta etapa es un mallado menos denso que conserva las características del mallado original. El proceso descrito fue agregado con la finalidad de reducir la entrada que se suministran a la siguiente etapa del algoritmo y mejorar su eficiencia al procesar modelos de gran tamaño.

Con el resultado previo, se hace uso del algoritmo QEM [31] para optimizar la selección de aristas a eliminar hasta conseguir el número esperado de vértices (nodos fundamentales). El proceso, califica cada arista con un peso que depende de la suma de las constantes de "forma" y "muestreo". Cada constante se calcula a partir de las posiciones actuales de los vértices, la longitud de las aristas y el número aristas que comparten un vértice en específico.

*1) Costo o Constante de Forma:* QEM calcula el costo de forma como la estimación de la distorsión causada por el colapso de una arista, mediante una métrica de error en cada vértice del mallado. La métrica mide las distancias al cuadrado entre un vértice y sus caras asociadas. En el caso específico de contracción de mallado, las caras que lo componen tienen área cero, por ende una métrica basada en caras no es válida para el caso de uso particular y se ajusta, resultando en la siguiente ecuación:

$$K_{ij} = \begin{bmatrix} 0 & -a_z & a_y & -b_x \\ a_z & 0 & -a_x & -b_y \\ -a_y & a_x & 0 & -b_z \end{bmatrix} \tag{6}$$

Donde $a$ es el vector normalizado de la arista $(i, j)$ y $b = a \times v_i$. La métrica de error inicial del vértice $i$ es la suma de todas las distancias cuadradas a sus bordes adyacentes, es decir (7):

$$F_i(p) = p^T \sum_{(i,j)\in E} (K_{ij}^T K_{ij})p = p^T Q_i p \tag{7}$$

$Q_i p$ representa la matriz de error actual para el punto $p$. Para mantener la forma de la malla intacta durante la simplificación, se seleccionó el colapso de aristas $i \to j$ basado en el costo de forma (8):

$$F_a(i,j) = F_i(v_i) + F_j(v_j) \tag{8}$$

Es decir, el colapso $(i \rightarrow j)$, es aquel con la suma mínima de distancias al cuadrado desde la posición contraída $v_j$ a las aristas adyacentes de los vértices $i$ y $j$, así como a todos los adyacentes de los vértices que se contrajeron previamente al vértice $i$ o $j$. Después de un colapso de arista, se actualiza la matriz de error del vértice $j$ como: $Q_j \leftarrow Q_i + Q_j$, de manera que las aristas que estaban asociados al vértice $i$ ahora se asocian al vértice $j$. Se almacena la matriz de error de cada vértice como una matriz $4 \times 4$ y, como en QEM, cada actualización de costos implica solo una adición de matriz.

*2) Costo o Constante de Muestreo:* El costo de forma conserva la apariencia de la malla original. Sin embargo, lleva a una simplificación excesiva en regiones rectas, produciendo bordes largos en el esqueleto y la pérdida de correspondencia fina entre el esqueleto y la superficie. Por lo tanto, se agrega un término de costo de muestreo que penaliza los colapsos que generan bordes largos. El costo de muestreo mide la distancia total que recorren las aristas adyacentes al vértice de origen, durante el colapso de la arista $(i \rightarrow j)$ (9):

$$F_b(i,j) = |v_i - v_j| \sum_{(i,k)\in E} |v_i - v_k| \tag{9}$$

Donde $E$ es la conectividad perimetral simplificada actual.

*3) Costo Total:* La función de costo total es una suma ponderada del costo de forma y el costo de muestreo (10):

$$F(i,j) = w_a F_a(i,j) + w_b F_b(i,j) \tag{10}$$

Se utilizan como valores iniciales $w_a = 1.0$ y $w_b = 0.1$, tal como lo hacen Au et al. [13], pero ambas variables pueden ser modificadas directamente en la aplicación. Ya que las matrices de error de forma son acumuladas a medida que el algoritmo itera, el costo de forma tendrá más peso en la selección de aristas a colapsar, lo que prioriza la conservación de la forma en la malla contraída.

El algoritmo es implementado mediante el uso de una cola de prioridad al igual que la etapa previa. La cola, almacena los costos totales para cada arista y se actualiza en cada iteración. En cada paso se remueven los vértices productos de un colapso y se repite el proceso hasta obtener el número de vértices deseado que representan los nodos principales del esqueleto.

### C. Refinamiento

De la etapa previa se obtienen un conjunto de nodos que componen el esqueleto curvo de la geometría original, estos nodos al ser producto de la contracción del mallado, se encuentran centrados dentro de la geometría original; sin embargo, debido a las características particulares que pueda tener un objeto, el esqueleto resultante puede no encontrarse correctamente centrado dentro de la geometría.

La etapa de refinamiento, corrige los errores de centrado que pueda tener el esqueleto curvo generado. El proceso de centrado consiste en evaluar la posición original de cada vértice que fue contraído para generar un nodo del esqueleto y reubicar dicho nodo en el centroide de los vértices seleccionados. Este proceso asegura que el esqueleto se encuentre correctamente centrado en las distintas partes que componen el objeto original.

Definiendo $\Pi_k$ como la región local compuesta por los vértices contraídos para formar el nodo $k$. Se calcula el desplazamiento necesario para mover el nodo $k$ al centro de la región, como el promedio de las posiciones de los vértices involucrados en la creación del nodo $k$ (11). Donde $d_k$ representa la nueva posición (centroide) para el nodo $k$ del esqueleto.

$$d_k = \frac{\sum_{i\in\Pi_k}(v_i - v_k)}{|\Pi_k|} \tag{11}$$

Es importante destacar que el algoritmo de refinamiento propuesto por Au et al. [13] tiene un problema en algunos modelos donde ciertas regiones locales tienen su centroide fuera del objeto, ocasionando que el esqueleto se ubique fuera de la malla. En la propuesta actual previo a desplazar el esqueleto a su nueva posición, se verifica que su ubicación se encuentre dentro de la malla, lo que corrige el problema anteriormente mencionado.

### D. Selección

El esqueleto curvo generado por las etapas previas tiene un alto número de nodos y aristas, ya que su finalidad es adaptarse lo mejor posible al objeto que lo define. Sin embargo, se plantea un proceso de selección, para reducir el número de nodos a solo los más representativos. Para el caso de uso particular de animaciones, se considera que un esqueleto compuesto de 20 a 40 nodos representa una buena base general.

El proceso es similar a la simplificación QEM antes expuesta, la diferencia radica en que las funciones de costo miden el tamaño de cada arista (ver ecuación 12), lo que permite eliminar las menos representativas (las de menor tamaño) hasta alcanzar el número de nodos deseado.

$$F(i,j) = |v_i - v_j| \tag{12}$$

### E. Asignación de Pesos

Con el esqueleto simplificado, se procede a agregar la información necesaria que permita a los huesos que lo conformen, alterar la posición del mallado original y por ende animarlo. Específicamente, el "peso", se refiere al grado de influencia que las transformaciones espaciales aplicadas a un hueso, tendrán sobre los vértices que componen el objeto original. La posición final de un vértice viene dada por la suma de las transformaciones espaciales de cada hueso que lo afecten, ponderada por los pesos o influencias de cada hueso sobre el vértice, por ende, la suma de los pesos de todos los huesos que afecten a un vértice nunca excederá uno.

Para el proceso de pesado se utiliza el planteamiento de Baran y Popoviç [22]. Donde se plantea simular un equilibrio de "calor" dentro de cada hueso, de manera que este influya sobre los vértices cercanos. El equilibrio sobre la superficie del hueso $i$ viene dado por (13), que se puede escribir como (14):

$$\frac{\partial w^2}{\partial t} = \Delta w^i + H(p^i - w^i) = 0 \tag{13}$$

$$-\Delta w^i + H w^i = H p^i \tag{14}$$

Donde $\Delta$ es la superficie discreta laplaciana, calculada con la fórmula cotangente de Meyer et al. [32], $p^i$ es un vector, donde $p^i_j = 1$ si el hueso más cercano al vértice *j* es *i* y $p^i_j = 0$ de lo contrario.

$H$ es una matriz diagonal, donde $H_{ij}$ es el peso de contribución de calor de el hueso más cercano al vértice $j$. Sea $d(j)$ la distancia desde el vértice $j$ al hueso más cercano, $H_{jj} = c/d(j)^2$, si el segmento de línea más corto desde el vértice hasta el hueso está contenido en el volumen del modelo y, $H_{jj} = 0$ en caso contrario. Para $c \approx 0.22$, el método genera pesos con transiciones similares a las calculadas al encontrar el equilibrio sobre el volumen. Pinocho[5] usa $c = 1$ (correspondiente al calor anisotrópico de difusión) dado que los resultados lucen más naturales.

Cuando $k$ huesos son equidistantes del vértice $j$, $p_j = 1/k$ para todos ellos y, $H_{jj} = kc/d(j)^2$.

La ecuación (14) es un sistema de ecuaciones lineal esparcido, donde su lado izquierdo (matriz $-\Delta + H$) no depende de $i$, siendo $i$ cada uno de los huesos en el esqueleto. Esto permite factorizar el sistema una sola vez y con el resultado calcular el peso de cada hueso. Botsch [33], muestra cómo usar la factorización Cholesky para este tipo de problema. Es importante aclarar que los pesos $\mathrm{W}^i$ siempre deben sumar 1 por cada vértice: si sumamos (14) sobre $i$, se obtiene $(-\Delta + H)\sum_i w^i = H \cdot 1$ que equivale a $\sum_i w^i = 1$.

### *F. Exportar*

Para poder empaquetar las estructuras generadas en un archivo soportado por cualquier programa de animación, es necesario conocer los fundamentos del formato animado. La idea principal es definir cada hueso del esqueleto no como dos vértices y una arista, sino por el contrario de manera analítica, como un vector con origen y dirección [34] (no es necesario incluir la longitud). Además hay que definir una jerarquía de huesos, donde cada uno tiene exactamente un padre, a excepción del hueso raíz. Cabe destacar que el hueso raíz no necesariamente se encuentra siempre dentro del modelo y eso no influye en el formato, recordemos que los huesos son estructuras que no se despliegan en la animación final, sino que simplemente sirven para el análisis de como se debe comportar cada vértice del modelo.

[5]Librería introducida por Baran y Popoviç [22].

Otra característica importante a resaltar, es la relación que existe entre cada hueso. El esqueleto es una estructura jerárquica donde la transformación espacial aplicada a un hueso padre, es heredada por su hijo. De manera que se si se mueve un hueso padre, todos sus hijos se moverán con él.

Las transformaciones de cada hueso son almacenadas en una matriz de 4x4 que contiene las transformaciones afines a aplicar al hueso canónico[6]. Existen 3 tipos importantes de matrices para expresar este modelo, y son los siguientes:

1) Matriz de Modelo: Define una transformación del origen del modelo sobre la posición absoluta del hueso. Se puede determinar a partir de la posición de origen de las aristas ($v_i$) y su dirección ($|v_j - v_i|$).
2) Matriz de Modelo (inversa): Inversa de la matriz anterior. Se utiliza para definir como se mueve el hueso al origen del modelo.
3) Matriz Local: Describe la posición y transformación de cada hueso, basada en el información de su hueso padre.

La matriz local es la única que se almacena en el formato de salida. Esta se puede calcular en base a la matriz de modelo de cada hueso, partiendo de la ecuación (15):

$$M_{Wk} = M_{Lk-n} * M_{Lk-(n-1)} * ...M_{Lk-1} \tag{15}$$

Donde $M_W$ y $M_L$ son las matrices de modelo y locales respectivamente, esta ecuación puede ser reescrita de la forma (16), para obtener la matriz local.

$$M_{Lk} = M^{-1}_{Wk-n} * M^{-1}_{Wk-(n-1)} * ...M^{-1}_{Wk-1} \tag{16}$$

Los huesos a exportar deben encontrarse, en lo que se que se conoce como la "pose en descanso", lo cual corresponde con la transformación inicial de los huesos que no alteren la posición de los vértices. Con esta información se puede crear un archivo de salida en formato Filmbox (FBX), el cual puede ser usado por diferentes aplicaciones de creación de contenido 3D, concluyendo el proceso de esqueletonización.

## IV. Pruebas

Durante las pruebas realizadas al sistema, se experimentó con 8 objetos tridimensionales con distintas características topológicas y de conteo poligonal. Dichas variaciones verifican:

- El crecimiento en el tiempo de computo ante la variación en la cantidad de polígonos del objeto procesado.
- La efectividad del algoritmo ante variaciones en la topología del objeto procesado.
- La robustez del algoritmo para procesar objetos con gran cantidad de vértices y caras.

Por cada etapa del método propuesto, se realizaron pruebas individuales que verifican su correcto funcionamiento. Los objetos tridimensionales utilizados pueden observarse en la Figura 4 y sus conteos de polígonos se indican a continuación:

[6]El hueso canónico es el que se encuentra situado con origen en (0, 0, 0) y tiene una longitud $\iota$ de tal forma que el vector que lo define es ($\iota$, 0, 0).

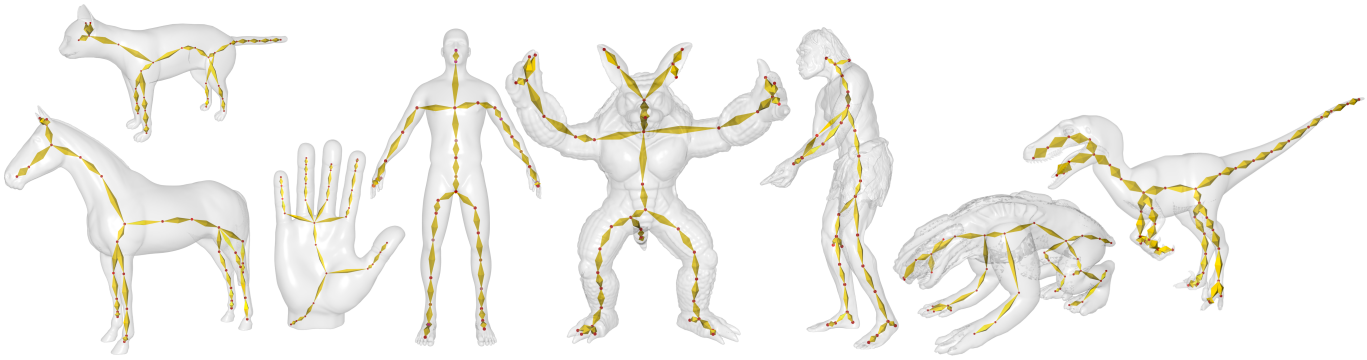

**Figura 4:** Modelos Utilizados en las Pruebas

- *Cat*: 7207 vértices y 14410 caras.
- *Man*: 24461 vértices y 48918 caras.
- *Hand*: 66848 vértices y 133692 caras.
- *Horse*: 120660 vértices y 241316 caras.
- *Armadillo*: 172974 vértices y 345944 caras.
- *Alien*: 373036 vértices y 745408 caras.
- *Raptor*: 595462 vértices y 1190656 caras.
- *Homoheidelbergensis* 1088639 vértices y 2176968 caras (escultura capturada como nube de puntos).

El ambiente en el que se realizaron las pruebas es el siguiente:

1) Procesador: AMD de 8 núcleos.
2) Memoria RAM: 16GB.
3) Tarjeta Gráfica: NVIDIA GTX 1050ti de 4GB VRAM con 768 CUDA cores.
4) Sistema Operativo: Windows 10.

En cada prueba, se realizaron todas las posibles permutaciones entre el método utilizado para resolver el sistema de ecuaciones y la permutación aplicada a la matriz que lo define. Estas pruebas tienen como intención determinar la mejor combinación entre ambos factores. De igual manera, se evaluó el desempeño del algoritmo tanto en GPU como en CPU.

Las factorizaciones de matrices para los métodos de resolución de ecuaciones directos planteadas, son:

- Cholesky.
- LU.
- QR.

Y las permutaciones ejecutadas sobre cada una de ellas son:

- AMD: Implementada con la librería Eigen.
- SymAMD: Implementada con el API de CUDA.
- MDQ: Implementada con el API de CUDA.
- RCM: Implementada con el API de CUDA.
- METIS: Implementada con el API de CUDA.

Es importante aclarar, que pese a que las pruebas se realizaron tanto en GPU como en CPU, los algoritmos para generar las matrices de permutaciones residen en CPU (inclusive en la implementación en CUDA). Igualmente, el procesamiento de la matriz de permutaciones ($P$) se realiza una única vez, ya que a pesar de que la contracción del mallado necesite varias iteraciones, la matriz $P$ siempre será la misma, dado a que esta depende únicamente de la estructura de la matriz (su forma) y no de los valores per sé.

**Tabla I:** Tiempos: Solucionador-Permutaciones

| | AMD | S.AMD | MDQ | RCM | METIS |
|---|---|---|---|---|---|
| Cholesky | 0.5371s | 0.5065s | 0.839s | 0.504s | 0.4963s |
| LU | 2.29s | 7.331s | 1.984s | 7.238s | 2.407s |
| QR | 11.46s | 19.24s | 2.55s | 19.13s | 9.77s |

En el Tabla I se pueden observar los resultados promediados en CPU y GPU para el modelo *Cat*, en donde se evidencia que la mejor combinación es la permutación METIS con la factorización Cholesky. Adicionalmente, se puede apreciar en la Figura 5 la solución obtenida. A partir de este punto, el resto de las pruebas con los métodos de solución de ecuaciones directos serán expresados bajo la combinación METIS-Cholesky (ver última columna, Figura 2).

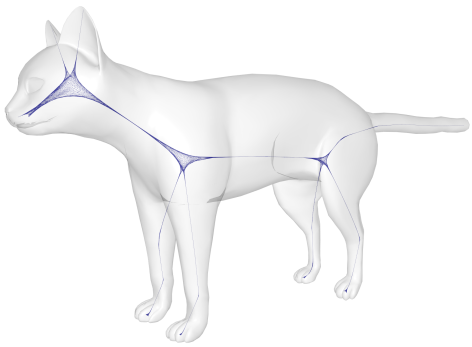

**Figura 5:** Contracción de Geometría Aplicada Sobre el Modelo *Cat*

La Tabla II compara los tiempos obtenidos por los solucionadores directos y los iterativos basados en multigrilla (AMG [24][25] implementado en CPU en este caso). En estos resultados se observa claramente como a medida que se incrementa el número de polígonos a procesar, el tiempo de procesamiento de los algoritmos incrementa. Además se puede concluir que los métodos directos son más rápidos que los basados en multigrillas, siendo la solución en CPU del método

**Tabla II:** Métodos Directos (CPU, GPU) vs. AMG

| | CPU | GPU | AMG |
|---|---|---|---|
| Cat | 0.272s | 0.713s | 0.323s |
| Man | 1.903s | 2.914s | 2.043s |
| Hand | 9.272s | 10.395s | 12.209s |
| Horse | 43.745s | 25.948s | 70.576s |
| Armadillo | 94.158s | 60.654s | 157.747s |
| Alien | 96.080s | 83.394s | 169.855s |
| Raptor | 91.341s | 108.729s | 148.99s |
| Homoheidel | 749.701s | 238.427s | 1294.5s |

directo 1.7 veces más rápida que la iterativa para el modelo más pesado y la solución en GPU 5.4 veces más rápida.

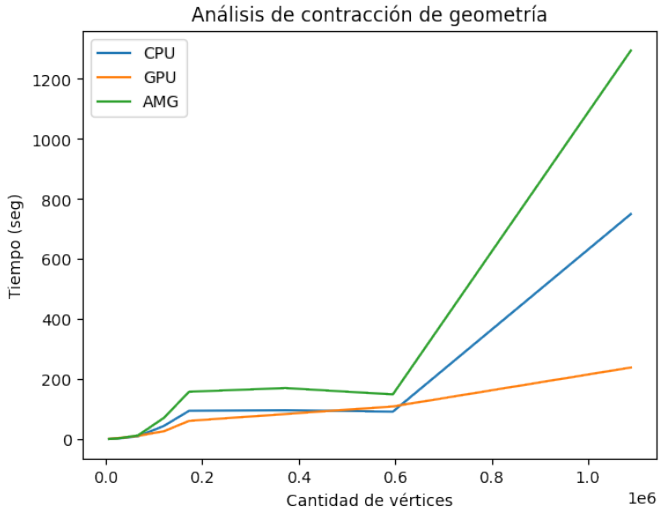


**Figura 6:** Comparación CPU, GPU, AMG

De igual manera, se evidencia que cuando se procesan modelos de bajo número de polígonos, el costo asociado a la creación del contexto para ejecutar el algoritmo en GPU supera el tiempo de procesamiento del mallado, por lo que las variantes en CPU resultan más eficientes. Sin embargo, al crecer el conteo de polígonos, el tiempo de creación de contexto se vuelve irrelevante y la implementación en GPU exhibe un crecimiento lineal a diferencia del exponencial observado en las vertientes en CPU (ver Figura 6).

Es importante mencionar que la contracción de mallado depende del laplaciano como operador principal, por ende los tiempos se comportan diferentes si la geometría y la topología del modelos son diferentes; por ejemplo: resolver la contracción para una esfera es mucho más rápido que resolverla para un gato o un humano, incluso si la cantidad de vértices es la misma. Otro punto importante es que mientras se tomen en cuenta detalles de implementación como la cotangente en infinito negativo y positivo, la solución de la contracción de geometría es insensible a la escala de la malla.

Finalmente, cabe destacar que el método utilizado no tiene limitantes en cuanto a tamaño del problema, funciona correctamente incluso para casos de prueba con un alto conteo de polígonos involucrados. La única limitante va directamente relacionada con la capacidad del hardware utilizado.

### *A. Operador Laplaciano*

Se realizaron pruebas visuales sobre el operador laplaciano que demuestran como evoluciona la curvatura en cada una de las iteraciones del algoritmo hasta que se acerca a los valores máximos y mínimos de la cotangente. Esto fue implementado con la misma metodología de visualización que posee MATLAB en su función *jet*, donde se codifica la información en colores. En la Figura 7 el color azul representa los valores mínimos (menos curvatura) de la cotangente mientras que el rojo el máximo (curvatura pronunciada). A medida que los triángulos del modelo se comprimen convergiendo al esqueleto curvo, la representación de colores se torna azul oscuro, indicando que no hay curvatura y que efectivamente los vértices y las aristas se encuentran tan cerca que el volumen tiende a cero.

### *B. Simplificación*

Del proceso de simplificación se pueden extraer resultados parciales que permiten evaluar el funcionamiento del algoritmo. Se puede visualizar el mapeo inducido *esqueleto-malla* representado en la Figura 8 mediante secciones alternando entre el color blanco y cyan. Este mapeo se refiere a los vértices absorbidos por cada nodo durante el proceso de simplificación y su función es confirmar que la etapa se completó de forma correcta ya que cada nodo incluye únicamente a los vértices cercanos a él, generando el efecto que se aprecia en dicha Figura 8.

### *C. Refinamiento*

Por otro lado, el refinamiento también genera resultados intermedios tales como la segmentación del mallado, la cual es de utilidad en diversos campos de la computación gráfica. La forma de representarla es identificando con diferentes colores cada uno de los segmentos que se desprenden del modelo inicial, tal como se puede apreciar en la Figura 9.

### *D. Selección*

Si bien el proceso de selección (ver Figura 10) no genera resultados parciales, modifica la estructura del esqueleto y por ende los resultados parciales de las etapas previas, disminuyendo el número de secciones en el mapeo inducido *esqueleto-malla* debido a que son menos nodos. El caso análogo sucede con los segmentos generados en el refinamiento, ahora abarcan más espacio y hay menos segmentos. En la Figura 11 se puede apreciar la evolución de los resultados obtenidos luego de aplicar la etapa de selección, a la izquierda se puede observar el aumento del tamaño de las secciones del mapeo inducido sobre el modelo *Man* (comparar con Figura 8), a la derecha se observa el incremento de la dimensión de los segmentos sobre el modelo *Armadillo*, a su vez se pude notar como la cola y las extremidades del modelo tienen menos secciones y como las mismas abarcan más área (comparar con Figura 9).

### *E. Asignación de Pesos*

Se verificó que cada vértice tuviese un peso asignado acorde al hueso más cercano. En la Figura 12, se selecciona un hueso

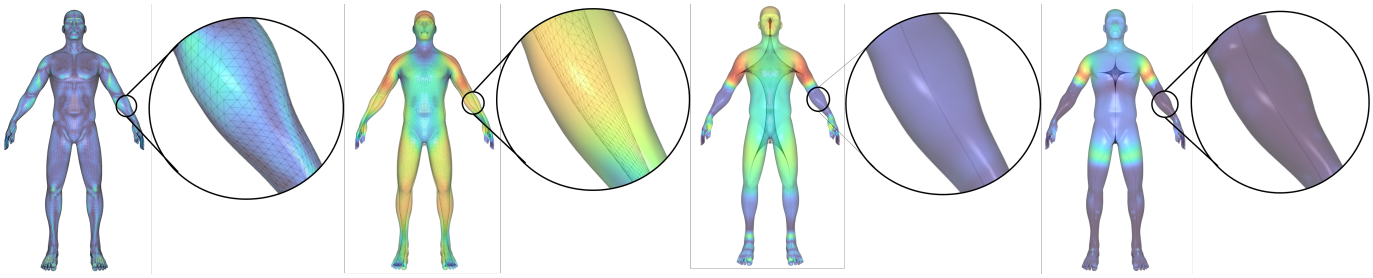

**Figura 7:** Evolución del Operador Laplaciano sobre el Modelo *Man* en cada Iteración de la Contracción

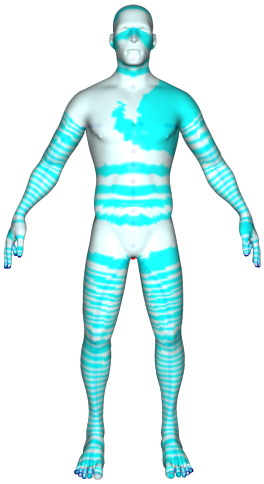

**Figura 8:** Mapeo Inducido *Esqueleto-Malla*, Representado por las Secciones Alternadas de Colores Blanco y Cyan, Asociadas a cada Nodo que las Contiene. Sobre el Modelo *Man*

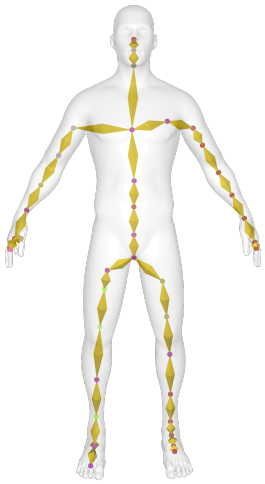

**Figura 10:** Proceso de Selección. Cada Hueso se Representa con Dos Rombos en Color Amarillo y cada Nodo como Punto Rojo. Sobre el Modelo *Man*

**Figura 9:** Segmentación sobre el Modelo *Armadillo*

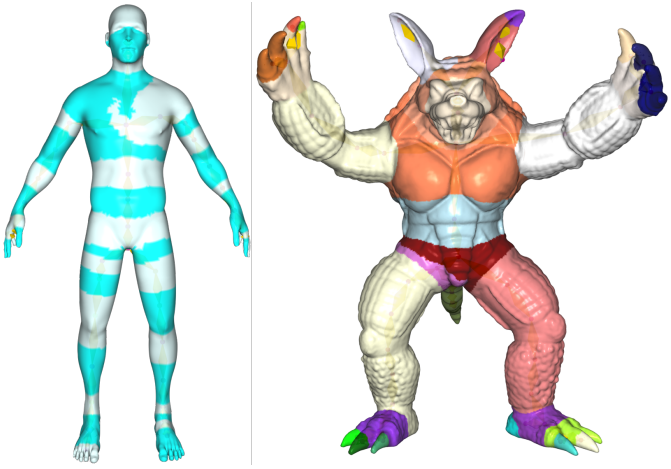

**Figura 11:** Resultados Posterior a Aplicar la Etapa de Selección (Comparar con Figuras 8 y 9)

y el color azul indica que los vértices no son influenciados por este, mientras que el color rojo expresa un alto grado de influencia.

### *F. Tiempos Totales*

Finalmente, se presenta los tiempos individuales por etapa del algoritmo propuesto. En el Tabla III se puede observar que la etapa de contracción involucra aproximadamente el 77% del total.

## V. Conclusión y Trabajos Futuros

En el presente trabajo, se desarrolló un método de esqueletonización basado en geometría, tomando como referente el propuesto por Au et al. [13]. Mediante las pruebas realizadas, se puede concluir que el método propuesto es 5.4 veces más rápido que la referencia.

Adicionalmente, las pruebas indican que los métodos de solución directa de sistemas de ecuaciones son más eficientes que los iterativos y, que la permutación METIS con la factorización

**Tabla III:** Tiempos por Etapa

| Modelo | Contracción | Simplificación | Refinamiento | Selección | Peso | Exportación | Total |
|---|---|---|---|---|---|---|---|
| Cat | 0.71379s | 0.10455s | 0.0017008s | 0.0028183s | 0.10477s | 0.085525s | 1.0132s |
| Man | 2.9148s | 0.4708s | 0.010105s | 0.016621s | 0.22983s | 0.22872s | 3.8709s |
| Hand | 10.396s | 1.4438s | 0.022026s | 0.043721s | 0.78632s | 0.59756s | 13.289s |
| Horse | 25.948s | 2.7979s | 0.038827s | 0.07831s | 2.023s | 1.075s | 31.961s |
| Armadillo | 60.654s | 4.847s | 0.0507s | 0.12393s | 2.7777s | 1.628s | 70.082s |
| Alien | 83.395s | 11.914s | 0.10427s | 0.47783s | 5.3554s | 2.5349s | 103.78s |
| Raptor | 108.73s | 20.05s | 0.1892s | 0.61818s | 7.7117s | 4.3389s | 141.64s |
| Homoheidel | 238.43s | 34.006s | 0.39226s | 0.9673s | 24.722 | 12.042s | 310.56s |

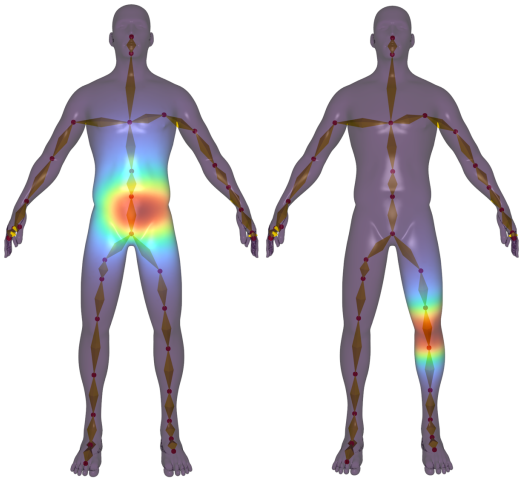

**Figura 12:** Pesos Asignados

Cholesky es la óptima para el caso particular de contracción de geometría.

Finalmente, el esqueleto curvo generado se adaptó para ser utilizado en animaciones por computador, permitiendo incluso exportar el resultado del algoritmo en un formato estándar soportado por todas las herramientas de animación tridimensional.

Como trabajo futuro, se propone expandir los casos de uso a los que puede ser adaptado el esqueleto generado por el algoritmo, a fin de abarcar más allá de la industria del entretenimiento.

## Referencias

[1] T. He, L. Hong, D. Chen, and Z. Liang, *Reliable Path for Virtual Endoscopy: Ensuring Complete Examination of Human Organs*, IEEE Transactions on Visualization and Computer Graphics, vol. 7, pp. 333–342, 2001.

[2] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He, *Virtual Voyage: Interactive Navigation in the Human Colon*, in Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 27–34, Los Angeles, CA, USA, 1997.

[3] D. Perchet, C. Fetita, and F. Preteux, *Advanced Navigation Tools for Virtual Bronchoscopy*, Image Processing: Algorithms and Systems III, vol. 5298, pp. 147–158, 2004.

[4] D. Bartz, W. Straßer, M. Skalej, and D. Welte, *Interactive Exploration of Extra- and Intracranial Blood Vessels (Case Study)*, in Proceedings of the Conference on Visualization 1999: Celebrating Ten Years, pp. 389–392, San Francisco, CA, USA, 1999.

[5] S. Pizer, D. Fritsch, P. Yushkevich, V. Johnson, and E. Chaney, *Segmentation, Registration and Measurement of Shape Variation Via Image Object Shape*, IEEE Transactions on Medical Imaging, vol. 18, pp. 851–865, 1999.

[6] S. Pizer, G. Gerig, S. Joshi, and S. Aylward, *Multiscale Medial Shape-Based Analysis of Image Objects*, Proceedings of the IEEE, vol. 91, pp. 1670–1679, 2003.

[7] M. Straka, M. Cervenansky, A. La cruz, A. Kochl, M. Sramek, E. Gröller, and D. Fleischmann, *The VesselGlyph: Focus & Context Visualization in CT-Angiography*, Visualization, pp. 385–392, 2004.

[8] C. Menier, E. Boyer, and B. Raffin, *3D Skeleton-Based Body Pose Recovery*, Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), pp. 389–396, Chapel Hill, NC, USA, 2006.

[9] S. Aylward, J. Jomier, S. Weeks, and E. Bullitt, *Registration and Analysis of Vascular Images*, International Journal of Computer Vision, vol. 55, pp. 123–138, 2003.

[10] R. Blanding, G. Turkiyyah, D. Storti, and M. Ganter, *Skeleton-Based Three-Dimensional Geometric Morphing*, Computational Geometry, vol. 15, pp. 129–148, 2000.

[11] J. Bloomenthal, *Medial-Based Vertex Deformation*, in Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 147–151, San Antonio, TX, USA, 2002.

[12] A. Vasilakis, *Skeleton-Based Rigid Skinning for Character Animation*, International Conference on Computer Graphics Theory and Applications, Funchal, MD, Portugal, 2008.

[13] O. Au, C. Tai, H. Chu, D. Cohen-Or, and T. Lee, *Skeleton Extraction by Mesh Contraction*, ACM Transactions on Graphics, vol. 27, 2008.

[14] H. Blum, *Biological Shape and Visual Science (Part I)*, Journal of Theoretical Biology, vol. 38, 1973.

[15] N. Cornea, D. Silver, and P. Min, *Curve-Skeleton Properties, Applications, and Algorithms*, IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 3, pp. 530–548, 2007.

[16] G. Bertrand and Z. Aktouf, *Three-Dimensional Thinning Algorithm using Subfields*, in Proceedings on Vision Geometry III, vol. 2356, Boston, MA, USA, 1995.

[17] D. Morgenthaler, *Three-Dimensional Simple Points: Serial Erosion, Parallel Thinning, and Skeletonization*, Technical Report: Computer Science Center, University of Maryland, 1981.

[18] T. Lee, R. Kashyap, and C. Chu, *Building Skeleton Models via 3-D Medial Surface/Axis Thinning Algorithms*, CVGIP: Graphical Models and Image Processing, vol. 56, no. 6, pp. 462–478, 1994.

[19] K. Palágyi, *A 3D Fully Parallel Surface-Thinning Algorithm*, Theoretical Computer Science, vol. 406, no. 1, pp. 119–135, 2008.

[20] G. Borgefors, I Nyström, and G. Sanniti Di Baja, *Computing Skeletons in Three Dimensions*, Pattern Recognition, vol. 32, no. 7, pp. 1225–1236, 1999.

[21] N. Thalmann and D. Thalmann, *Computer Animation: Theory and Practice*, Computer Science Workbench, 1985.

[22] I. Baran and J. Popović, *Automatic Rigging and Animation of 3D Characters*, ACM Transactions on Graphics, vol. 26, no. 3, pp. 72–80, 2007.

[23] M. Desbrun, M. Meyer, P. Schröder, and A. Barr, *Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow*, in Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 317–324, New York, NY, USA, 1999.

[24] B. Metsch, *Algebraic Multigrid (AMG) for Saddle Point Systems*, 2013.

[25] D. Demidov, *AMGCL—A C++ Library for Efficient Solution of Large Sparse Linear Systems*, Software Impacts, vol. 6, 2020.

[26] Y. Saad, *SPARSKIT: a Basic Tool Kit for Sparse Matrix Computations - Version 2*, 1994.

[27] P. Amestoy, T. Davis, and I. Duff, *An Approximate Minimum Degree Ordering Algorithm*, SIAM Journal on Matrix Analysis and Applications, vol. 17, no. 4, pp. 886–905, 1996.

[28] E. Cuthill and J. McKee, *Reducing the Bandwidth of Sparse Symmetric Matrices*, in Proceedings of the 1969 24th National Conference, pp. 157–172, New York, NY, USA, 1969.

[29] A. George and J. Liu, *A Fast Implementation of the Minimum Degree Algorithm using Quotient Graphs*, ACM Transactions on Mathematical Software (TOMS), vol. 6, no. 3, pp. 337–358, 1980.

[30] G. Karypis and V. Kumar, *METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, 1995.

[31] M. Garland and P. Heckbert, *Surface Simplification Using Quadric Error Metrics*, ACM Siggraph, 1997.

[32] M. Meyer, M. Desbrun, P. Schröder, and A. Barr, *Discrete Differential-Geometry Operators for Triangulated 2-Manifolds*, Visualization and Mathematics III, pp. 35–57, 2003.

[33] M. Botsch, D. Bommes, and L. Kobbelt, *Efficient Linear System Solvers for Mesh Processing*, Mathematics of Surfaces XI, pp. 62–83, 2005.

[34] S. Marschner and P. Shirley, *Fundamentals of Computer Graphics*, 2018.