Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVII, págs. 3081-3091 (artículo completo)
Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.)
San Luis, Argentina, 10-13 Noviembre 2008

TEXTURAS TRANSPARENTES PARA VISUALIZACION DE MALLAS Y DATOS

Walter Sotil[a] y Nestor Calvo[a,b]

[a]*Cátedra de Computación Gráfica - Ingeniería en Informática - FICH - UNL*

[b]*Centro Internacional de Métodos Computacionales en Ingeniería - INTEC - CONICET*

Palabras clave: Visualización de mallas, Visualización de datos, Texturas, Transparentes.

Resumen. En este trabajo se presenta el desarrollo e implementación de una interfaz gráfica para facilitar la visualización de mallas y datos tridimensionales.
En tres dimensiones se dificulta mucho la visualización de la malla (durante los procesos de generación, adaptación o suavizado) o del conjunto completo de resultados, por lo que se requiere de herramientas que faciliten la labor del investigador, y es por ello que se recurre a algoritmos que los muestren en forma entendible.
El programa que aquí se presenta realiza sucesivos cortes planos de una malla tridimensional que pueden visualizarse como una o varias tajadas de la malla, combinando para ello algunas características especiales de texturas y transparencias, y haciendo uso de la API gráfica OpenGL.
La principal misión del programa es servir de ayuda para el debugging gráfico, permitiendo encontrar errores o malformaciones en forma muy rápida. También sirve para identificar muy velozmente las zonas de mayor interés, para luego realizar visualizaciones más elaboradas.
Mediante una sencilla e intuitiva interfaz, basada en los movimientos del mouse se pueden cambiar libremente la posición, orientación, tamaño relativo y frecuencia de las tajadas.
El énfasis se ha puesto en la simplicidad del manejo como así también en la velocidad de la generación y aplicación de la textura.
Finalmente, el programa se integró exitosamente al generador y visualizador de mallas Meshsuite desarrollado en el CIMEC y quedó incorporado como un utilitario adicional que se ofrece al usuario a través del menú.

1 INTRODUCCION

La aproximación de sistemas de ecuaciones tridimensionales y continuas mediante dominios discretos está presente en muchas aplicaciones de la ingeniería, la física e incluso en las ciencias médicas. Se utilizan para resolver problemas de elasticidad, flujos de fluidos, interacción fluido-estructura, problemas térmicos, climatología o análisis de imágenes médicas, entre muchos otros.

Una de las primeras tareas para el modelado consiste en la discretización del dominio, es decir, la generación de la malla de cálculo (Calvo, 2005) o visualización. En la imaginería médica se suelen utilizar los dominios discretos y regulares provistos directamente por los equipos de captura de datos (tomografía y resonancia magnética). Por otra parte, para modelar numéricamente los problemas de las ingenierías, se utilizan los métodos de elementos finitos y de puntos.

El proceso completo de modelado suele dividirse en tres partes:

- Preprocesamiento: consiste en la generación del CAD y la malla de acuerdo a las necesidades del cálculo. Es un proceso que aun requiere mucha intervención del usuario, principalmente en la adaptación del CAD a las necesidades del proceso de mallado.
- Procesamiento: es el proceso de cálculo numérico y es totalmente automático
- Postprocesamiento: Consiste en la visualización y análisis de los resultados y posiblemente una estimación de errores para retroalimentación del proceso completo.

El análisis visual de la malla y los campos resultantes es un proceso con una gran cuota de subjetividad, que requiere necesariamente la intervención humana. Dadas las prestaciones de las computadoras actuales, los procesos manuales son los que están recibiendo la mayor atención, para desarrollar herramientas que permitan acelerar la labor.

La visualización del conjunto completo de resultados en tres dimensiones es impráctica; en su lugar se recurre a algoritmos que extraen automáticamente los datos interesantes y los muestren en forma entendible.

2 OBJETIVOS Y METODOS

En este trabajo se desarrolló e implementó una interfaz gráfica o GUI (Graphical User Interface) para facilitar la visualización de mallas y campos tridimensionales.

El objetivo es servir de ayuda para el debugging gráfico, permitiendo encontrar errores o malformaciones en forma muy rápida. También sirve para identificar muy velozmente las zonas de mayor interés, para luego realizar visualizaciones más elaboradas.

El programa realiza sucesivos cortes planos de una malla tridimensional compuesta por cualquier tipo de elementos, pudiéndose visualizar como una o varias tajadas de la malla. Se combinan para ello algunas características especiales de texturas y transparencias.

La normal de los planos de corte posee dos grados de libertad; hay otro para la ubicación, otro para el ancho y uno más para la frecuencia de los cortes o tajadas. En lugar de tener que definirlos numéricamente, el usuario puede alterarlos con una sencilla e intuitiva interfaz, basada en los movimientos del mouse.

El aspecto más importante del programa es la interfaz de usuario interactiva: debe hacer los cortes, mediante la aplicación de la textura, y la visualización en tiempo real. Para grandes volúmenes de datos debe, entonces, auto-limitarse; permitiendo manipular rápidamente el sistema y luego realizar los cálculos masivos en la ubicación elegida.

El programa se adosó exitosamente al generador de mallas Meshsuite desarrollado en el CIMEC y quedó incorporado como un utilitario adicional que se ofrece al usuario, a través del

menú de opciones de información.

Para el desarrollo de la GUI se hace uso opcional de las bibliotecas gráficas Qt (Trolltech) o FLTK (fltk.org), que son bibliotecas multiplataforma en C++, ampliamente utilizadas para la programación de aplicaciones y que proveen las funcionalidades necesarias para construir interfaces gráficas de usuario. La visualización de la malla y los datos se realiza mediante OpenGL, dentro de la ventana de dibujo provista por la GUI.

Los métodos aquí presentados dependen de OpenGL en los métodos de aplicación de las texturas, pero bien podrían adaptarse a otra API gráfica con prestaciones similares. Son, en cambio, totalmente independientes de la GUI elegida, es decir que pueden ser implementados con cualquier otra interfaz de usuario.

Una parte indisoluble de los objetivos es la portabilidad del sistema, la posibilidad de utilizar cualquier placa gráfica de PC estándar. Los desarrollos se hicieron utilizando la versión 1.1 de OpenGL (Davis, 1997) incorporada en el sistema operativo Windows y, por razones de mercado, es entonces soportada por casi cualquier placa gráfica. Existen otros métodos, con texturas procedurales 3D o programando el trabajo de la placa gráfica con el Shading Language GLSL que ya es estándar; se podría, por ejemplo, recortar la malla con círculos concéntricos o con las superficies de nivel de los campos calculados. Pero tales métodos condicionarían la portabilidad. Para este trabajo se eligió la simpleza a costas de la versatilidad.

3 CONCEPTOS PRELIMINARES

En esta sección se brindan los conceptos teóricos fundamentales que serán utilizados luego durante el desarrollo del trabajo.

3.1 API´s Gráficas

Una API (Application Programming Interface) es una biblioteca de rutinas para interactuar con el hardware y una interfaz mediante la cual se le permite al programador acceder a las funcionalidades que alberga.

Entre las API´s gráficas de mayor popularidad podemos mencionar a OpenGL, Direct3D, X11 y Postscript. OpenGL (Open Graphic Library) es la que prevalece en la industria al momento de desarrollar aplicaciones gráficas 3D multiplataforma, y la que se eligió para implementar el presente desarrollo.

OpenGL (opengl.org) es soportado por casi todos los sistemas operativos (Unix, Linux, Mac, Windows). Está implementado en hardware en casi todas las placas gráficas (la GPU se encarga de hacer las operaciones) y aun por software es eficiente.

Permite crear aplicaciones interactivas gráficas, bidimensionales y tridimensionales. Esta biblioteca incluye funciones para especificar las características esenciales de una escena, tales como el tipo de proyección, las características de la cámara, primitivas geométricas, luces, texturas, transparencias, double buffering para agilizar el renderizado o visualización cambiante, etc.

Entre sus principales ventajas podemos nombrar que es estable (perdura y se puede reutilizar), portable (a casi cualquier S.O.), eficiente y gratuito (tiene copyright de Silicon Graphics, pero hay implementaciones realmente libres como Mesa).

La GUI (Graphical User Interface) es la encargada de manejar los eventos y proveer la ventana gráfica para OpenGL, es la que entiende al teclado y al mouse y la que provee los menús y otras metáforas gráficas o widgets para que el usuario interactúe con el programa. Hay varias elecciones posibles, en Meshsuite se utilizan dos: Qt (Trolltech) o FLTK (fltk.org)

en forma optativa. Se eligieron (hace ya algunos años) por la funcionalidad provista, el acceso libre, la documentación y la portabilidad. Hay interfaces nuevas o distintas que bien podrían utilizarse para albergar el presente desarrollo, sin ningún cambio en su formulación.

3.2 Lineamientos generales de un programa gráfico

Los lineamientos generales seguidos al desarrollar una aplicación gráfica se pueden dividir en inicialización de OpenGL (fijación de múltiples variables de la cámara, sistema de proyección, depth-buffer, iluminación, materiales, etc.), inicialización de la GUI (definición menús y callbacks, suministrando la dirección de cada función que deberá llamarse cuando se produzca determinado evento), creación de ventanas (posición, dimensiones, nombre), y loop de ejecución que reaccionará a los distintos eventos para luego refrescar determinadas variables y finalmente redibujar la zona de interés.

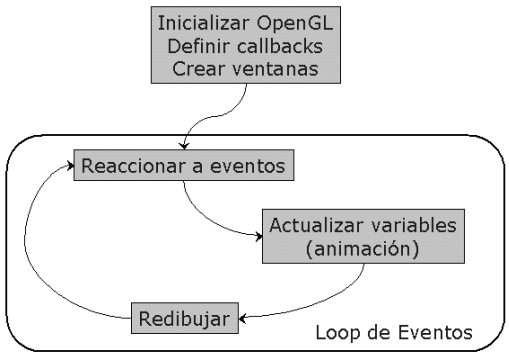

Figura 1: Lineamientos de un programa gráfico.

3.3 Texturas

Las texturas permiten simular superficies complejas con baja cantidad de polígonos, suministrando realismo a la escena e incrementando drásticamente el nivel de detalle. Matemáticamente podemos decir que “pegar” una textura sobre un objeto consiste en definir una función unívoca (función de mapeo o mapping) que a cada punto del espacio ocupado por la superficie (o línea o sólido) le hace corresponder un punto en el espacio donde está definida la textura como una imagen 1, 2 o 3 dimensional.

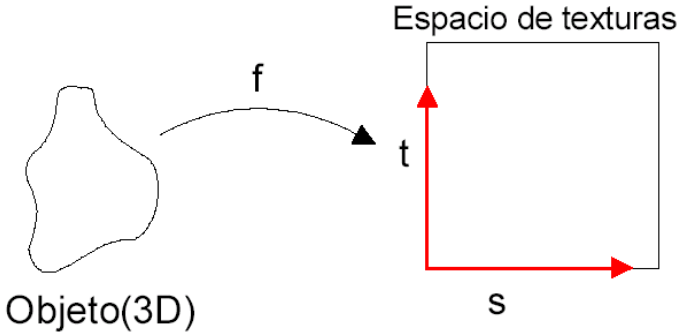

Figura 2: Función de mapeo.

Una textura, desde el punto de vista de su almacenamiento en memoria es un array de datos. A cada uno de los valores de este array lo llamamos “téxel”. Este array de datos representa una imagen, que utilizaremos para mapearla sobre un polígono. Cada téxel se compone de tres valores de color RGB y puede incluir un valor A o alpha que utilizaremos

para definir transparencia.

Según la dimensión del espacio de textura, podemos distinguir tres tipos de textura:

- Texturas en 1D: Es una imagen con anchura pero sin altura, o viceversa; tienen un único téxel de ancho o de alto. Permite una rápida generación de la textura.
- Texturas en 2D: Se podría pensar que se le está pegando una foto o imagen plana a la superficie de un cuerpo en 3D. Es el tipo de textura más utilizado.
- Texturas en 3D: Consiste en un volumen con información de color, el cual es asignado al objeto en 3D. Solo se puede visualizar en las superficies expuestas del sólido: la exterior y las que surgen al cortarlo virtualmente, el resultado visual es equivalente al que aparecería si "esculpiéramos" el objeto.

Cuando aplicamos una textura debemos especificar como queremos que se mezcle con el color del objeto. OpenGL permite elegir entre cuatro métodos, que tienen como objetivo determinar el valor RGBA final de cada pixel a visualizar a partir del color de la superficie del objeto a texturizar y del color de la textura:

- Replace (Sustitución): En este modo el color utilizado para el objeto es el de la textura.
- Decal (Sustitución con transparencia): En este modo se diferencian dos casos. Si trabajamos con formato de textura RGB el color utilizado para el objeto es el de la textura. En cambio, con formato de textura RGBA, el color a utilizar es una mezcla del color de la superficie y el de la textura, predominando más uno u otro dependiendo del valor alpha.
- Modulate (Modulación): Permite ir escalando el color final entre el color original de la superficie y el negro dependiendo del grado de luminosidad de la textura (1 para color de la superficie; 0 para negro).
- Blend (Mezcla): Se utiliza para mezclar los valores del color y de transparencia de la superficie con los de la textura. De forma añadida podemos mezclar todo a su vez con un tercer color que determinemos.

En OpenGL, si queremos aplicar textura, tenemos que seguir los siguientes pasos:

- Activar el mapeo de texturas.
- Especificar que imagen (buffer) va a ser utilizada como textura.
- Mapear la textura.
- Indicar como la textura va a ser aplicada a cada píxel.

Se pueden aplicar algunos efectos especiales como las líneas de perfil, en donde se distribuye la textura en función de la distancia al plano. El vector normal al plano indica la orientación de la textura. OpenGL genera las coordenadas de textura automáticamente.

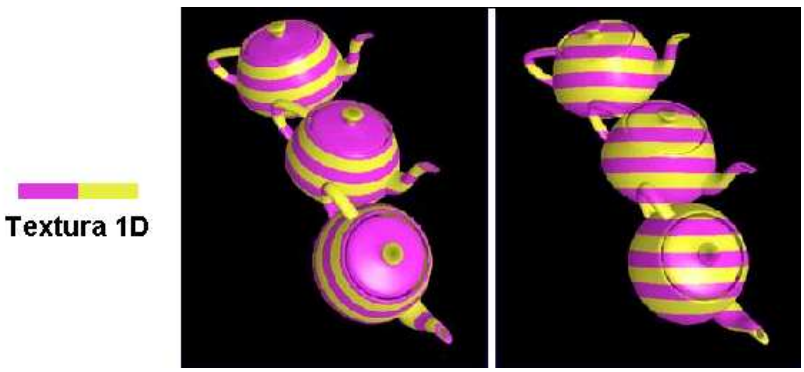

Figura 3: Mapeo con wrapping (repetición de textura) activo.

3.4 Alpha Blending

Alpha Blending es una técnica que permite crear objetos transparentes. Normalmente, un píxel que aparece en pantalla tiene valores de rojo, verde y azul. OpenGL permite, además utilizar un canal alpha para mezcla y así dar el efecto de transparencia. Un objeto puede tener diferentes niveles de transparencia: por ejemplo, una ventana de cristal limpia tendría un nivel muy alto de transparencia (un valor alpha muy bajo), mientras que un cubo de gelatina podría tener un valor alpha medio. El Alpha Blending es el proceso de combinar dos objetos en pantalla teniendo en cuenta los valores alpha. Así sería posible tener un objeto medio oculto tras un cubo de gelatina de frutilla que estaría pintado de rojo y difuminado.

3.5 Plano de corte

El plano de corte queda definido por un punto $\mathbf{P_0}(x_0,y_0,z_0)$ y un vector $\mathbf{n}$ = (a,b,c) ortogonal al plano; el punto y la normal determinan las distintas ecuaciones del plano (Lehmann, 1984).

Sea $\mathbf{P}(x,y,z)$ un punto cualquiera del plano; entonces $\mathbf{P} - \mathbf{P_0}$ es ortogonal a $\mathbf{n}$; es decir:

$$(\mathbf{P} - \mathbf{P_0}) \cdot \mathbf{n} = 0. \quad (1)$$

Siendo ésta última la ecuación vectorial.

Luego:

$$(x- x_o,y- y_o,z- z_o) \cdot (a,b,c) = 0. \quad (2)$$

$$a(x - x_o)+b(y - y_o) + c(z - z_o) = 0. \quad (3)$$

Finalmente, la ecuación implícita del plano es:

$$ax + by + cz = d. \quad (4)$$

En donde d es la distancia del plano al origen multiplicada por el módulo de $\mathbf{n}$:

$$d = \mathbf{n} \cdot \mathbf{P_0}. \quad (5)$$

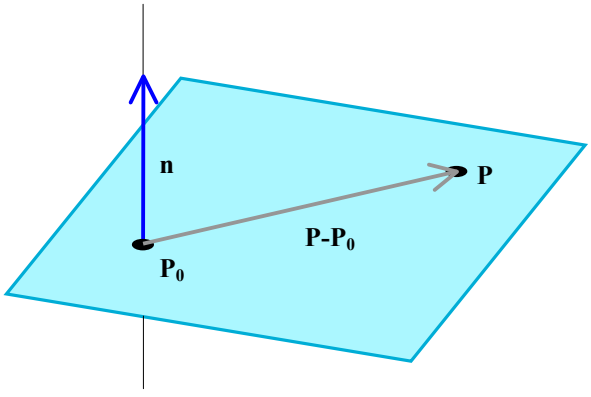

Figura 4: Plano determinado por el punto $\mathbf{P_0}$ y un vector normal $\mathbf{n}$.

4 PASOS DEL PROCEDIMIENTO

A continuación se describen los procedimientos individuales requeridos para la generación y manipulación de la textura, dando como resultado los sucesivos cortes planos de la malla tridimensional, que pueden visualizarse como una o varias tajadas de la malla.

4.1 Generación y aplicación de la textura

El objetivo de esta etapa es generar la textura con determinadas características, que va a

aplicarse a la malla.

Se aplicó una textura 1D, es decir que posee un único téxel de ancho. La textura no fue leída de un archivo de imagen, sino que se generó en forma automática, lo que se denomina textura procedural.

Se utilizó un formato RGBA ya que es necesario utilizar el canal alpha para transparencias. Esta cuarta componente, cuando tenemos el efecto blending activo, se combina con el valor del color almacenado en el frame buffer (el buffer que se va a visualizar) para generar el nuevo color. Para verlo de una forma más clara podemos pensar en la componente alpha como en el grado de "opacidad" y las demás, RGB, el color en sí.

La función utilizada para determinar el valor RGBA final de cada píxel en la superficie del objeto a texturizar fue Modulate, cuyo significado se rige por las siguientes ecuaciones:

$$C = C_f * C_t. \tag{6}$$

$$A = A_f * A_t. \tag{7}$$

A es el valor alpha y C representa cualquiera de los valores R, G o B. Los subíndices son f para el fragmento existente (en este caso, el color y alpha originales del objeto) y t para la textura.

Para la textura se asignaron los valores R=G=B=1 (valor máximo o blanco), que reemplazados en la ecuación anterior queda:

$$C = C_f. \tag{8}$$

Es decir que luego de aplicar la textura, los valores RGB de la malla no han sido alterados.

La textura unidimensional aplicada es de 32 x 1 téxels. Para el canal alpha se asignaron valores nulos a algunos téxels y valores máximos (1) al resto. Es decir que reemplazando en la ecuación queda:

$$A = A_f * A_t. \tag{9}$$

$$A_1 = A_f * 0. \tag{10}$$

$$A_2 = A_f * 1. \tag{11}$$

Por lo tanto:

$$A_1 = 0. \tag{12}$$

$$A_2 = A_f. \tag{13}$$

Es decir que luego de aplicar la textura, los valores alpha de la malla son nulos en una parte, mientras que los restantes no se modificaron, son los definidos para la malla, si hubiera un efecto de semi-transparencia, éste se conserva.

Luego, al activar el alpha-testing, se efectúa la verificación de valores alpha. Básicamente lo que se hace es dibujar o no, dependiendo de la función de comparación y el valor de alpha resultante. El alpha testing se configuró de modo que sólo se acepte el fragmento si el valor alpha origen es distinto (o mayor) que cero. Entonces, luego de esta operación sólo se logra visualizar, y sin cambios, la porción de la malla que es afectada por los téxels con alpha =1.

Con ello se logró el efecto deseado: luego de aplicar la textura, una parte de la malla queda invisible y la otra inalterada, visualizándose la malla en forma de tajadas.

Las coordenadas de la textura se asignan a los vértices del objeto según diversos procedimientos, de tal forma que se identifique cada uno de los píxeles de dicho objeto durante el renderizado con uno de los téxels de la textura para sustituir, o alterar, alguna de las

características del píxel original de superficie.

Se puede realizar la asignación las coordenadas de textura en forma manual, esto es asignando a cada vértice del polígono una coordenada en el espacio de texturas. También se puede realizar la asignación en forma automática.

En este trabajo se utilizó la asignación automática, y en particular las "líneas de perfil", en donde se define un plano y se distribuye la textura en función de la distancia al plano. La orientación y punto de aplicación de la textura están determinados por la definición de los parámetros característicos de un plano: $\mathbf{P_0}$ y $\mathbf{n}$.

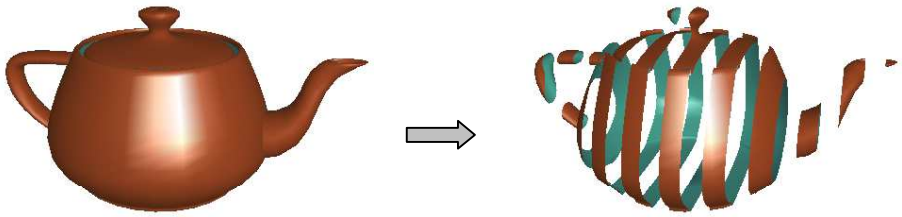

Figura 5: Resultado obtenido al aplicar la textura.

4.2 Manipulación de la textura

El objetivo de esta etapa es posibilitar la manipulación de la textura mediante controles interactivos simples en tiempo real. Se incorporó en Meshsuite la capacidad de desplazar el plano que define la textura en forma paralela, es decir trasladando el punto de definición en la dirección de la normal, y la posibilidad de rotación, manteniendo fijo el punto de definición y rotando el vector normal al plano. Se incorporó además la capacidad de modificar la proporción visible/invisible y la frecuencia de repetición de la textura.

Los cambios se producen mediante incrementos distintos, dependiendo si se presiona simultáneamente la tecla Ctrl cuando se realizan, lo que le permite realizar un cambio rápido primero y luego, presionando Ctrl, realizar un ajuste más preciso.

4.2.1 Desplazamiento de la textura

Hay dos maneras de desplazar la textura a posiciones elegidas en la malla: alterando la matriz de textura provista por OpenGL o desplazando el punto $\mathbf{P_0}$ perteneciente al plano. Se optó por la segunda, aunque cabría realizar un análisis para determinar cual supone un mayor costo computacional.

Entonces, para permitir el desplazamiento sólo se cambia la posición de $\mathbf{P_0}$, pero en la dirección del vector $\mathbf{n}$ ortogonal al plano. Dado que este cambio se realiza con movimiento de la rueda del mouse, la nueva pocisión se calcula mediante:

$$\mathbf{P_0} = \mathbf{P_0} + \mathbf{n} * \text{velocidad\_definida\_por\_Ctrl} * \text{sentido\_de\_giro\_rueda}. \quad (14)$$

4.2.2 Rotación de la textura

Hay dos maneras de realizar la rotación la textura: rotando la matriz de textura provista por OpenGL o alterando la dirección del vector normal perteneciente al plano. Se optó por esta última, pero se aplican las mismas observaciones que antes.

Se utiliza un algoritmo que simula la rotación una bola con la normal fija. Supóngase una esfera virtual centrada con la ventana; cuando el usuario "pica" en un punto se supone dicho punto sobre la esfera en lugar de estar sobre el plano del monitor. Al arrastrar el punto se produce un giro de la esfera virtual, junto con el vector normal fijo a la bola. Ese giro se

calcula con la geometría de la ventana, de la esfera virtual y los movimientos del cursor.

Más en detalle, a la bola se le asigna un diámetro igual a la diagonal de la ventana, siendo ésta de ancho w y alto h, el radio es:

$$r = \sqrt{w^2+h^2} \,/\, 2. \tag{15}$$

Cuando el usuario pica sobre un punto del viewport (ventana de visualización) de coordenadas (x,y), se proyecta el mismo sobre la semiesfera delantera, asignándole a la coordenada z el siguiente valor:

$$z = \sqrt{r^2-(x^2+y^2)}\ . \tag{16}$$

Se generan entonces dos puntos: el punto $\mathbf{M_0}$ donde se picó inicialmente y el punto $\mathbf{M_1}$ donde se soltó el botón del mouse que, proyectados sobre la semiesfera, dan $\mathbf{B_0}$ y $\mathbf{B_1}$ respectivamente. El giro está definido por los vectores que van del centro de la esfera (y de la ventana) a cada uno de dichos puntos. El eje **E** del giro es la recta de soporte del producto vectorial de dichos vectores y el ángulo de giro se calcula a partir del módulo del producto:

$$\mathbf{E} = \mathbf{B_0} \times \mathbf{B_1}. \tag{17}$$

$$\|\mathbf{E}\| = \|\mathbf{B_0}\|\,\|\mathbf{B_1}\| \operatorname{sen}(\alpha) \Rightarrow \alpha = \operatorname{arcsen}\left(\frac{\|\mathbf{E}\|}{\|\mathbf{B_0}\|\,\|\mathbf{B_1}\|}\right). \tag{18}$$

Dado que los datos del plano (**n** y $\mathbf{P_0}$) están dados en el espacio del modelo, multiplicamos **E** por la matriz de rotación del modelo provista por OpenGL y es con este eje con el que giramos el vector normal al plano.

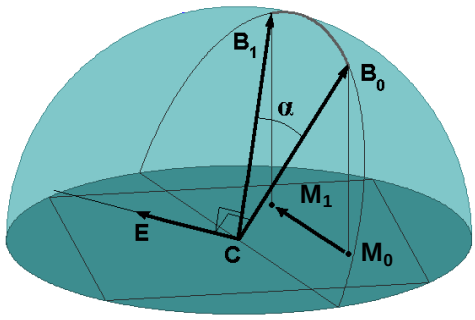

Figura 6: Movimiento del mouse proyectado sobre la semiesfera.

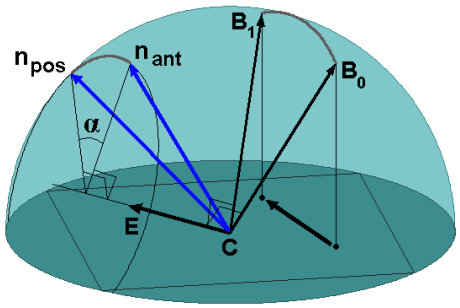

Figura 7: Giro del vector normal al plano alrededor del eje **E**.

4.2.3 Tamaño relativo de las tajadas

Para permitir el cambio de la proporción entre las zonas visible e invisible se altera el patrón de ceros y unos en el alpha de téxeles generados. Con ello, se adicionó la capacidad de aplicar texturas con saltos continuos en el ancho de las tajadas. Dado que el ancho de la textura debe ser una potencia de 2, por simplicidad se eligió 32, de modo que la porción visible puede variar entre 0 y 1 en 32 pasos. Este cambio se habilita en el menú y se realiza con la rueda del mouse.

4.2.4 Repetición o no de la textura

Con la finalidad de posibilitar una mayor flexibilidad en la herramienta, se incorporó al menú la opción de repetición o no de la textura. Si la textura no se repite, permite visualizar una única tajada. Para lograrlo es necesario que el primero y el último téxel sean "invisibles", es decir que tengan alpha nulo, en este caso se reducen a 30 los pasos que definen el tamaño de las tajadas visibles.

4.2.5 Frecuencia de repetición de la textura

La longitud del vector normal es la unidad de medida en la aplicación de la textura. Con un movimiento programado del ratón se modifica, en el intervalo (0,1], dicha longitud.

5 RESULTADOS

En esta sección presentamos los resultados obtenidos luego de la aplicación de las texturas y transparencias sobre la malla tridimensional.

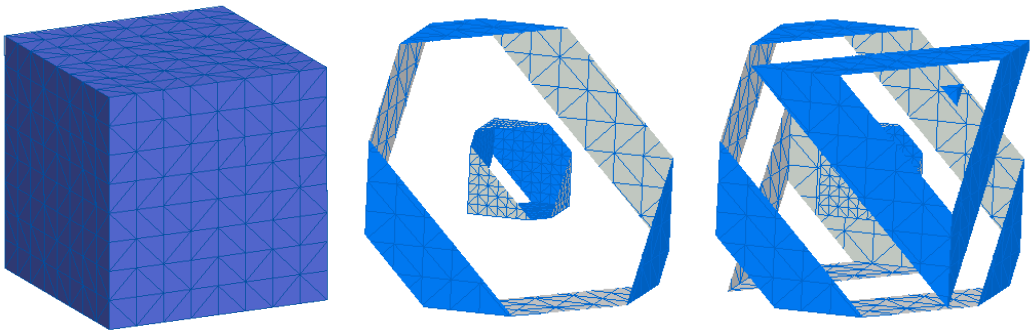

Figura 8: Aplicación de la textura en la frontera de un cubo hueco. A la derecha con repetición de textura.

Figura 9: Distintos anchos de tajadas, visualizando la frontera y el interior.

Figura 10: Aplicación de la textura en la frontera e interior de un toroide, con distintos anchos de tajadas.

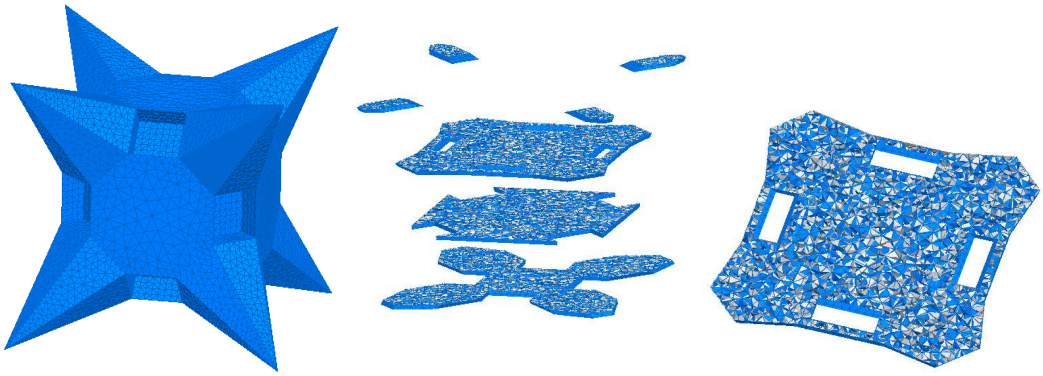

Figura 11: Aplicación de la textura en la frontera e interior de un poliedro cóncavo estrellado, y ampliación.

6 CONCLUSIONES

En el presente trabajo se ha desarrollado una herramienta que permite la realización de sucesivos cortes planos de una malla tridimensional, que pueden visualizarse como una o varias tajadas de la malla y la manipulación del plano mediante controles interactivos simples en tiempo real.

Su aplicación en forma simultánea con otras herramientas desarrolladas con anterioridad, como son la malla de corte plana y la isosuperficie, resulta de mucha utilidad al momento de analizar mallas tridimensionales.

Esto se logró gracias a la implementación robusta y eficiente de las estructuras de datos.

REFERENCIAS

Calvo, N., *Generación de mallas tridimensionales por métodos duales*. Tesis doctoral, Facultad de Ingeniería y Ciencias Hídricas, Universidad Nacional del Litoral, Santa Fe, Argentina, 2005.

Davis, T. Neider, J., *OpenGl Programming Guide: The Official Guide to Learning OpenGL*, Release 1.1, 2 edition, Addison-Wesley Publishing Company, 1997.

fltk.org: http://www.fltk.org/.

Lehmann, C. H., *Analytic Geometry*. John Wiley and Sons, New York, 8 edition, 1984.

opengl.org: http://www.opengl.org.

Trolltech: http://trolltech.com/products/qt.