Visualización Progresiva de Redes Tetraédricas en el Dominio de Wavelet

Mauricio López, Silvia Castro, Sergio Martig
Dpto. de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur
Bahía Blanca, Argentina
maur.lopez@gmail.com
{smc, srm}@cs.uns.edu.ar

Abstract

Currently, the progressive transmission of tetrahedral meshes in the wavelet domain is very efficient. It is necessary to pass on first the base mesh along with the coefficient obtained during the multi resolution analysis before transmitting the details. The details can be transmitted in packages without a specific order. In this paper we suggest a simple but strong algorithm for progressive visualization of tetrahedral meshes in the wavelet domain. We adjust the mesh during the execution time using the red/green refinement technique in order to show all the possible details. At the same time, we improve the volume renderer performance by reducing its work load in all the non visible parts with regard to the point of view.

Keywords: Wavelets, Multiresolution Representation, Scientific Visualization, Volume Rendering, Computer Graphics

Resumen

La transmisión progresiva de redes tetraédricas en el dominio wavelet es muy eficiente. Sólo es necesario transmitir primero la red base junto con los coeficientes obtenidos durante el análisis multirresolución, mientras que los detalles pueden ser transmitidos después, por paquetes y en desorden. En este trabajo proponemos un algoritmo simple y robusto para visualización progresiva de tales redes tetraédricas. Adaptamos la red en tiempo de ejecución para mostrar el máximo detalle posible mediante un esquema de refinamiento rojo/verde al tiempo que aumentamos la eficiencia del rendering reduciendo el detalle en las zonas no visibles con respecto al punto de vista.

Palabras Clave: Wavelets, Multirresolución, Visualización Científica, Rendering de Volúmenes, Computación Gráfica.

1. INTRODUCCION

Recientemente se han planteado modelos volumétricos multirresolución basados en redes tetraédricas que permiten tanto la compresión como la transmisión progresiva de los mismos; estos modelos se basan en dos extensiones de wavelets sobre dominios tetraédricos ([3], [4], [5], [21]). La transmisión de tales redes tetraédricas es muy eficiente, ya que se puede hacer por paquetes, siendo la red base la única geometría que se envía. Posteriormente solo es necesario transmitir los detalles extraídos en la fase de análisis, los cuales adicionalmente pueden estar comprimidos. Este modelo de transmisión progresiva de redes tetraédricas es nuevo y constituye el soporte fundamental del presente trabajo.

En este trabajo nos centramos en la visualización progresiva de tales redes tetraédricas. Nuestro algoritmo es apropiado para visualizar datos transmitidos progresivamente por la red, ya que no se necesita la presencia de todo el detalle para realizar la visualización. De hecho, la única geometría necesaria es la malla de menor resolución, ya que el resto se va generando conforme sea necesario, de modo que la visualización se adapta a la cantidad de detalles disponibles. La única restricción que deben cumplir los volúmenes es tener la propiedad de conectividad de subdivisión; esta propiedad debe verificarse para todo el volumen excepto para la red base.

La cantidad de tetraedros aumenta exponencialmente con cada nivel de refinamiento; es por ello que, para lograr la visualización en tiempos interactivos, nuestro algoritmo refina selectivamente la malla de acuerdo a dos heurísticas, ambas dependientes de la ubicación del observador. La primera busca refinar aquellos tetraedros cuya proyección en perspectiva no cumpla cierta tolerancia de error medida en píxeles y la segunda busca realizar un *coarsening* de aquellos tetraedros que estén fuera de la pirámide de visión.

El algoritmo desarrollado es simple y robusto; se enfoca en realizar los refinamientos y *coarsening*[1] sobre conjuntos grandes de tetraedros para aumentar su eficiencia. Debido a que adoptamos la estrategia de refinamiento rojo/verde de Bey ([2]) y Lohner *et al* ([14]), generamos mallas tetraédricas de buena calidad; esto redunda en la calidad de nuestra visualización.

2. TRABAJOS RELACIONADOS

2.1 Representaciones Multirresolución de mallas tetraédricas

Muchas representaciones multirresolución se crearon en base a un método de simplificación subyacente. En los modelos no anidados, el colapso de arista es la modificación con fines de simplificación más general, ya que en base a ella se pueden construir las demás. De Floriani, Puppo y Magillo ([8]) introducen un marco conceptual teórico denominado *Multiresolution Simplicial Model* (MSM), en el cual la multirresolución se crea partiendo de la malla a máxima resolución, aplicando sucesivas modificaciones locales de simplificación (colapsos de arista o colapsos de cara). Las relaciones de dependencia de cada modificación se guardan como un Grafo Acíclico Dirigido (DAG). Cualquier triangulación que satisfaga las restricciones de orden establecidas por el DAG, es valida. Popovic y Hoppe ([16]) extendieron las Mallas Progresivas ([10]) y propusieron una representación multirresolución mas general aplicada a *Simplicial Complexes*, cuyo método de simplificación subyacente es el colapso de arista. Otros investigadores ([6], [7], [17]) plantean algoritmos de simplificación de mallas tetraédricas basados en el colapso de arista y triangulaciones de Delaunay que dan lugar a representaciones multirresolución. Sin embrago, ningún método anteriormente mencionado, produce representaciones multirresolución que cumplen con la propiedad de conectividad de subdivisión.

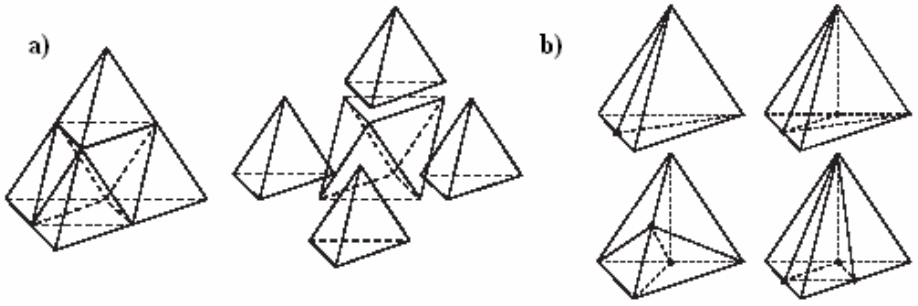

Figura 1: a) subdivisión regular en 8 subtetraedros. b) cuatro casos de subdivisión irregular: 2, 3, y 4 subtetraedros.

Para modelos anidados, Bey ([2]) extiende el refinamiento rojo/verde de superficies de Bank *et al.* ([1]) a tres dimensiones. En el esquema rojo/verde, un tetraedro rojo puede ser refinado regularmente o irregularmente. Los tetraedros provenientes de un refinamiento irregular son verdes, el resto continúan siendo rojos. La subdivisión es estable ya que sólo los tetraedros regulares pueden ser subdivididos. La subdivisión se adapta a dominios irregulares refinando únicamente ciertos tetraedros seleccionados. Los tetraedros verdes mantienen la malla conforme.

El refinamiento regular consiste en subdividir un tetraedro en ocho subtetraedros de igual volumen (Figura 1a). Existen 62 posibles patrones de refinamiento irregular, que pueden ser

[1] Dado que no hay palabra en castellano que sea el antónimo de refinar, se utiliza el verbo en inglés *to coarsen*, que significa pasar de una mayor a una menor resolución.

reducidos a 9 debido a consideraciones de simetría. Grosso *et al.* ([9]) restringen el refinamiento irregular a cuatro casos (Figura 1b), dejando los demás casos como refinamiento regular. La regla para refinar un tetraedro irregular es convertirlo primero a regular.

Zhou *et al* ([20]), presenta un método general para refinamiento de mallas regulares denominado MTF (Multiresolution Tetrahedral Framework) que bisecciona los tetraedros basado en el criterio de subdividir por la arista mas larga. Pascucci ([15]) generaliza el método MTF a *n* dimensiones y lo denomina SGS (Slow Growing Subdivision), además su nuevo método puede lidiar con volúmenes mucho más grandes. SGS está basado en la subdivisión y fusión de tres clases de diamantes. Un diamante se forma uniendo pirámides, tetraedros o, en su defecto, ambos.

2.2 Wavelets sobre dominios tetraédricos

En ([3], [4], [5], [21]) se construye una representación tetraédrica multirresolución compacta, adecuada para compresión y transmisión progresiva por la red, basándose en el análisis multirresolución de wavelets. Se definen dos bases de wavelets de Haar diferentes sobre tetraedros; la primera base está definida sobre un tetraedro y la segunda, sobre los vértices del mismo. La transformación de la malla tetraédrica al dominio wavelet se denomina *análisis* y la transformación inversa se denomina *síntesis*. En la fase de análisis se descompone (o filtra) el dominio de la malla a máxima resolución, usando las funciones base de Haar en dos conjuntos, uno de coeficientes (baja frecuencia) y otro de detalles (alta frecuencia); los coeficientes conforman una malla de menor resolución y los detalles se guardan por separado. Es necesario que la malla posea la propiedad de conectividad de subdivisión. En la fase de síntesis se aplica la transformación inversa de wavelet, tomando como parámetro a la malla de menor resolución junto con los detalles obtenidos en la fase de análisis. Mediante refinamiento regular se obtiene la malla de mayor resolución. En ambas fases, el esquema de subdivisión es el de Bey ([2]), que subdivide un tetraedro en ocho subtetraedros donde cada uno de éstos tiene igual volumen.

2.2.1 Transmisión progresiva de Redes Tetraédricas en el Dominio Wavelet

Castro ([21]) y Castro *et al.* ([5]), sugieren aprovechar su nueva representación multirresolución para mejorar la transmisión progresiva de redes tetraédricas. Primero debe transmitirse la red base, que el receptor puede mostrar inmediatamente, luego se deben transmitir los detalles y esto puede hacerse en secuencia arbitraria. Si cada detalle o conjunto de detalles especifican a qué tetraedro pertenecen en la multirresolución, la transmisión ordenada de los mismos no es importante. De hecho, la pérdida de algunos detalles durante la transmisión no impide la visualización de la malla, solo introduce algunos errores locales. Castro *et al* proponen el uso del protocolo TCP para la transmisión de la malla base ya que es mas seguro y UDP para la transmisión de los detalles por ser mas veloz.

3. ESTRUCTURAS DE DATOS

La malla tetraédrica esta conformada por un conjunto de cinco arreglos dinámicos y un conjunto de cuatro índices dinámicos. Ambos conjuntos forman nuestra base de datos. El conjunto de arreglos dinámicos lo conforman los arreglos de vértices, escalares, tetraedros, normales y aristas. El arreglo de vértices guardará todos los vértices de la malla; asimismo, el arreglo de escalares guardará todos los escalares asociados a cada vértice. Como la relación entre escalares y vértices es de uno a uno, ambos arreglos tendrán el mismo tamaño. Los arreglos de normales y aristas, guardarán las normales asociadas a cada tetraedro y las aristas respectivamente. El arreglo de tetraedros guarda todos los tetraedros de la multirresolución (Figura 2).

Cada tetraedro consta de cuatro índices al arreglo de vértices, índices que le sirven para acceder al arreglo de escalares; cada tetraedro posee cuatro índices al arreglo de normales y seis índices al arreglo de aristas (Figura 2). Si bien cada tetraedro posee 14 índices en total, esta representación es

económica, ya que todos los tetraedros comparten vértices, escalares, normales y aristas, lo cual aumenta la escalabilidad del sistema cuando crece el numero de tetraedros.

```
class   mr_tetraedro
{
        unsigned int        _ivertices[4];
        unsigned int        _inormales[4];
        unsigned int        _iaristas[6];
        unsigned int        _padre;
        unsigned int        _hermano;
        unsigned int        _hijo;
        unsigned short      _flags;
        half                _diametro;
};
```

```
class   mr_malla
{
        array< vector3<float> >     _vertices;
        array< half >               _escalares;
        array< mr_tetraedro >       _tetraedros;
        array< vector3<float> >     _normales;
        mapaEdges                   _edges;
        octree                      _indiceVerts;
        gaussMap                    _indiceNormales;
        garbageCollector            _indiceTetras;
};
```

Figura 2: estructuras de datos para un tetraedro y para la malla indexada de tetraedros.

El tipo *half* ([12]) es una clase C++ que almacena números flotantes de 16 bits. Posee la misma funcionalidad que el tipo flotante de 32 bits. El rango de números representables esta en el rango 6.0×10^{-8} – 6.5×10^{4}. El tipo *half* ahorra espacio en memoria y es convertido al formato IEEE-754 de 32 bits antes de realizar cualquier operación aritmética. Los arreglos de la multirresolución crecen dinámicamente cuando se rebasa su límite, se expanden delta elementos cada vez que sea necesario, logrando un sobre-dimensionado que favorece la performance. Delta debe ser un número pequeño si se desea favorecer el ahorro de memoria o un número relativamente grande si se desea favorecer la velocidad. En la práctica un crecimiento moderado de 64 o 128 elementos es adecuado para nuestro propósito.

Muchos algoritmos de Rendering de Volúmenes, tales como el de Proyección de Tetraedros ([18]), utilizan estructuras indexadas porque esto permite resolver eficientemente las referencias a vértices y atributos al tiempo que minimizan el uso de memoria. Para poder generar rápidamente una malla indexada, es necesario realizar búsquedas eficientes sobre los distintos arreglos. Nuestra malla posee un conjunto de índices que hacen las búsquedas muy eficientes.

El arreglo de vértices está indexado por un Octree que encierra espacialmente a la malla base y se subdivide uniformemente hasta que cada nodo terminal tenga un radio proporcional al 3% del radio del nodo raíz, esto es aproximadamente 5 niveles de subdivisión. El arreglo de normales está indexado por un mapa gaussiano, similar al presentado por Kumar *et al* ([13]), y por Zhang y Hoff ([19]) para hacer *backface culling* jerárquico. Sin embargo, en lugar de usar una esfera de radio uno, nosotros utilizamos un cubo que encierra a dicha esfera. Subdividimos las caras del cubo uniformemente con un *kd-tree* y proyectamos cada pequeña cara en la superficie de la esfera. De este modo las normales insertadas en el mapa gaussiano quedan agrupadas en los nodos de los *kd-trees*. Esto nos permite buscarlas rápidamente.

En la práctica sólo tres caras del cubo son utilizadas, ya que agrupamos las normales iguales y de sentidos opuestos sin considerar el sentido negativo; esto reduce enormemente el espacio de almacenamiento de las mismas, sin embargo, al estar indexadas, siguen siendo accedidas rápidamente. El arreglo de aristas está indexado por un mapa Hash. La función de hashing (Figura 3) combina los dos índices de vértices que forman la arista (a, b) en un número de 32 bits. Es necesario ordenar los índices a y b antes de aplicar la función de hashing, de tal modo que el índice "a" sea menor que el índice "b".

```
unsigned int   hash_func( unsigned int a, unsigned int b )
{
        return ( (a & 1023) << 10 ) + (b & 1023);
}
```

Figura 3: función de hashing que indexa la arista (a, b).

Todos los índices previamente descritos son necesarios para asegurar que los vértices, aristas y normales serán únicos, ya que deben ser compartidos por los tetraedros. Esto quiere decir que durante la creación de un nuevo tetraedro, éste deberá buscar primero sus vértices en el arreglo de vértices e insertarlos sólo en caso de no encontrarlos. El mismo proceso debe ocurrir con sus aristas y normales.

Finalmente, el arreglo de tetraedros está indexado implícitamente como veremos a continuación. Cada tetraedro guarda tres índices o enlaces a otros tetraedros: padre, hijo y hermano. La estructura jerárquica de la multirresolución es una foresta de árboles donde cada tetraedro de la malla base es la raíz un árbol. Los tetraedros hijos de un mismo padre, están enlazados unos con otros de manera circular guardando el índice de su hermano. Un tetraedro padre sólo necesita guardar el índice de un hijo y a través de él puede tener acceso a los demás. Por último, cada tetraedro hijo guarda el índice de su padre.

Los primeros elementos del arreglo de tetraedros corresponden a la malla base. Estos tetraedros son inamovibles. El resto de tetraedros se crearán conforme se refine sucesivamente la malla. Sin embargo, cuando un tetraedro es eliminado, el arreglo queda intacto, y su índice está a disposición para ser reutilizado posteriormente durante la creación de otro tetraedro. Esto sucede frecuentemente durante el refinamiento selectivo en tiempo real. La clase denominada ***garbageCollector*** es la encargada de este proceso.

La eliminación de aristas es un poco más compleja. Cada arista mantiene un contador de referencias. Cada vez que es creado un tetraedro, éste encuentra sus aristas en el arreglo de aristas (o bien inserta nuevas aristas). Cada arista referenciada aumenta su contador; mas tarde, cuando se eliminan tetraedros, los contadores de referencias de sus aristas disminuyen. Si algún contador llega a cero, la arista es marcada como disponible para ser reutilizada y se borra del índice de aristas (mapa hash).

4. ALGORITMOS DE REFINAMIENTO Y DESREFINAMIENTO

4.1 Refinamiento de la Malla

El refinamiento de la malla lo efectuamos basándonos en el algoritmo rojo/verde de Lohner *et al*. ([14]). Lo hacemos en tres etapas: Marcado, Clasificación y Refinamiento.

En la *etapa de* ***Marcado***, se marcan las aristas de todos los tetraedros que se refinarán regularmente, esto significa marcar las seis aristas de cada tetraedro. Dado que los tetraedros comparten aristas, los tetraedros adyacentes se ven modificados por esta acción; el marcado de aristas es una operación sencilla y rápida. Mantenemos un arreglo de bits de la misma dimensión que el arreglo de aristas, un bit apagado simboliza que la arista no está marcada; por tanto el marcado se reduce a prender un bit del arreglo. La Figura 4 muestra el pseudo código del algoritmo para la etapa de marcado.

```
Proc  MarcarTetraedrosRojos( listaTetras )
       Para cada tetraedro t en listaTetras
              MarcarTetraedro ( t )
       Fin Para
Fin Proc

Proc   MarcarTetraedro( Tetraedro )
       Para cada arista a en Tetraedro
              MarcarArista( a )
       Fin Para
Fin Proc
```

Figura 4: Marcado de tetraedros regulares para refinamiento.

En la *etapa de* ***Clasificación***, analizamos cada tetraedro y lo clasificamos según el número de marcas que tienen sus aristas. A continuación se describe el tipo de refinamiento que aplicaremos; éste depende únicamente del número de marcas que logra acumular cada tetraedro.

Tipo1 (Refinamiento Regular): todas las aristas del tetraedro están marcadas; el tetraedro se subdivide en ocho subtetraedros de igual volumen.

Tipo2: el tetraedro tiene marcadas tres aristas de una misma cara; el tetraedro se subdivide en cuatro subtetraedros.

Tipo3a: el tetraedro tiene marcadas dos aristas que no pertenecen a la misma cara: el tetraedro se subdivide en cuatro subtetraedros.

Tipo3b: el tetraedro tiene marcadas dos aristas de una misma cara; el tetraedro se subdivide en tres subtetraedros.

Tipo4: el tetraedro tiene marcada una arista; el tetraedro se subdivide en dos subtetraedros.

Si un tetraedro posee cuatro o cinco aristas marcadas, no puede pertenecer a ningún tipo antes mencionado, por lo tanto, la malla no quedaría conforme. Esto sucede también con los tetraedros que poseen tres aristas marcadas que no pertenecen a la misma cara. En estos casos se deberá marcar las aristas que restan para completar seis marcas y el tetraedro quedará clasificado como *Tipo1*. Adicionalmente, tenemos el caso que un tetraedro irregular posea aristas marcadas; como no es posible refinar tetraedros irregulares, es necesario eliminar el refinamiento irregular y cambiarlo por regular, para luego proceder a una reclasificación. Esto puede causar un efecto dominó, ya que al introducir nuevas marcas, nos vemos obligados a reclasificar algunos tetraedros y potencialmente introducir nuevamente otras marcas.

Claramente esta etapa es la más costosa del proceso. Al no guardar explícitamente la adyacencia de tetraedros por cada arista, que aumentaría innecesariamente el gasto de almacenamiento de memoria, optamos por un algoritmo iterativo que recorre sucesivamente los tetraedros terminales de la jerarquía multirresolución (Figura 5). Se considera que la aproximación recursiva planteada por Lohner *et al* ([14]), no es adecuada para nuestro propósito ya que podría agotar rápidamente la pila. El algoritmo de la Figura 5 optimiza el uso de memoria y es robusto al eliminar el problema del desbordamiento de pila.

```
Proc ClasificaMalla( Malla )
       Repetir
              EsConforme ← VERDADERO.
              ListaIrregulares ← Vacia.
              Para cada tetraedro terminal t en Malla
                     Si t es irregular y tiene marcas
                            ListaIrregulares ← t
                            EsConforme ← FALSO.
                     Si No
                            Si t tiene 6 marcas: t ← Tipo1
                            Si t tiene 3 marcas en la misma cara: t ← Tipo2
                            Si t tiene 2 marcas no en la misma cara: t ← Tipo3a
                            Si t tiene 2 marcas en la misma cara: t ← Tipo3b
                            Si t tiene 1 marca: t ← Tipo4
                            En otros casos:
                                   MarcarTetraedro( t )
                                   EsConforme ← FALSO
                     Fin Si
              Fin Para
              Si ListaIrregulares no esta vacia
                     Para cada tetraedro t en ListaIrregulares
                            EliminarSubTetraedros( t.padre )
                            MarcarTetraedro( t.padre )
                            RefinarRegularmente( t.padre )
                     Fin Para
              Fin Si
       Repetir Mientras Not EsConforme
Fin Proc
```

Figura 5: Clasificación de los tetraedros de la malla

Terminada la etapa de clasificación ya tenemos una malla conforme lista para ser refinada. La Figura 6 ilustra el refinamiento de cada tetraedro según su tipo:

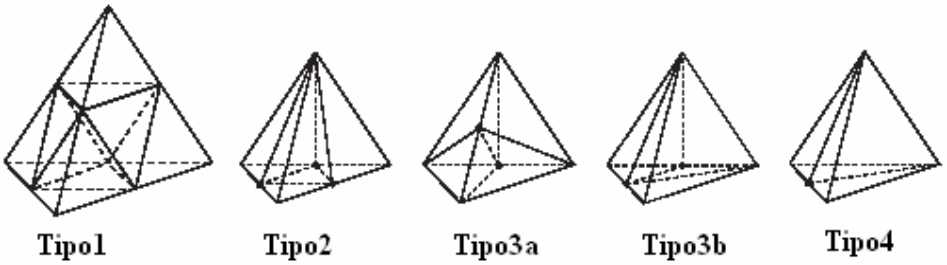

Figura 6: Cinco tipos de refinamiento. El Tipo1 corresponde a refinamiento regular, los demás a refinamiento irregular.

El refinamiento regular toma los cuatro coeficientes de los escalares del tetraedro y, usando su propio índice, toma seis detalles de la tabla de detalles provista con la malla base. Siguiendo el proceso de síntesis descrito en ([3], [4], [5]) obtenemos los 10 escalares necesarios para aumentar el nivel de detalle de la malla. De manera similar obtenemos los escalares para cada tetraedro irregular. En el caso que los detalles no estén disponibles, el refinamiento no se puede cancelar ya que la malla tiene que ser conforme. En cambio, interpolamos linealmente los cuatro escalares del tetraedro para obtener los seis escalares que hacen falta para el refinamiento. Esto introduce cierto error en el dominio, sin embargo, se mantiene bajo debido a que este caso solamente sucede cuando es necesario mantener la conformidad de la malla a través de la multirresolución.

A menudo se da el caso que se necesita forzar el refinamiento de tetraedros irregulares, debido a que se dispone de los detalles. En esta situación, antes de la etapa de clasificación, es necesario eliminar los tetraedros irregulares y refinar regularmente al padre.

4.2 *Coarsening* de la Malla

Sólo es posible realizar el *coarsening* de los tetraedros refinados regularmente, ya que los tetraedros irregulares existen para hacer conforme la malla, a través de la multirresolución. Sin embargo, junto con los tetraedros regulares, también aplicamos el *coarsening* sobre los tetraedros irregulares que son adyacentes, esto provoca que el *coarsening* de los regulares sea verdaderamente efectivo (Figura 7).

El algoritmo de *coarsening* es capaz de volver a la malla base desde cualquier resolución, ya que podemos aplicarlo a conjuntos grandes de tetraedros. De forma similar que el algoritmo de refinamiento, se divide al *coarsening* en tres etapas: Borrado, Clasificación y Refinamiento.

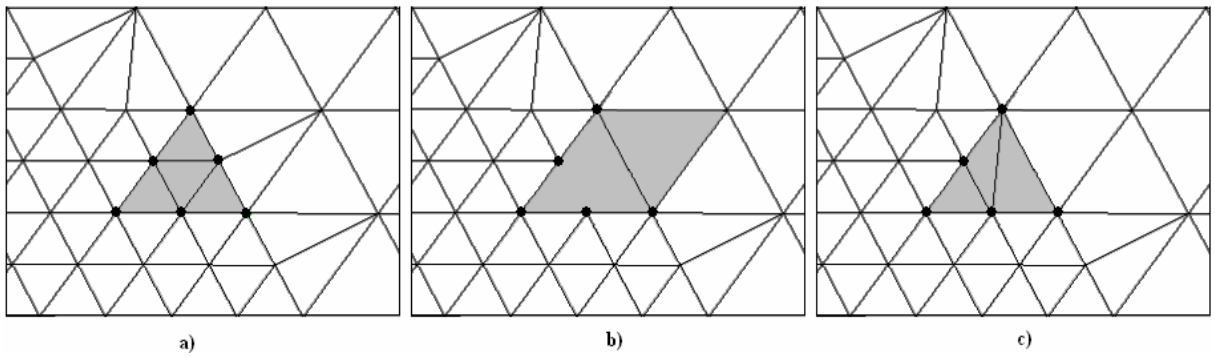

Figura 7: a) tetraedro refinado regularmente al que se aplicará *coarsening*. b) después de eliminar los subtetraedros se deberá desmarcar las aristas adyacentes a tetraedros irregulares, en consecuencia también estos se serán eliminados. c) finalmente, se reclasifica la malla según el número de marcas que tiene para dejarla conforme.

En la *etapa de* ***Borrado***, se eliminan todos los subtetraedros regulares cuyos padres pasen a una menor resolución. Adicionalmente necesitamos desmarcar las aristas del padre que son adyacentes sólo a tetraedros irregulares; así, estos podrán eliminarse posteriormente. Desafortunadamente, al no guardar explícitamente la adyacencia de arista, no podemos saber directamente qué tetraedros son

adyacentes. Nuestra solución es bastante simple. En primera instancia, desmarcamos todas las aristas de los padres y, debido a que los tetraedros comparten aristas, los adyacentes se verán modificados por esta acción. Posteriormente, sólo los adyacentes regulares restaurarán sus marcas y permanecerán intactos, mientras que los vecinos irregulares serán detectados y eliminados. El algoritmo se detalla en la Figura 8.

```
Proc   Borrado( Malla, listaRegulares )
       arbolBusqueda ← Vacio
       Para cada tetraedro t en listaRegulares
              EliminarSubTetraedros( t.padre )
              Para cada arista a en t.padre
                     DesmarcarArista( a )
                     arbolBusqueda ← a
              Fin Para
       Fin Para
       Para cada tetraedro terminal t en Malla
              Si t.padre no ha sido procesado antes
                     Si alguna arista de t.padre existe en arbolBusqueda
                            Si t es regular
                                   MarcarArista( aristas de t.padre )
                            Si No
                                   EliminarSubTetraedros( t.padre )
                            Fin si
                     Fin Si
              Fin si
       Fin Para
Fin Proc
```

Figura8: seudo código para la fase borrado de tetraedros.

Las etapas de clasificación y refinamiento son exactamente iguales a las del refinamiento convencional.

4.3 Refinamiento Selectivo

El refinamiento selectivo consiste en adaptar la malla con respecto al punto de visión en tiempo de ejecución. Teniendo en cuenta que la visualización se dará en tiempos interactivos, realizamos el refinamiento selectivo basándonos en dos heurísticas planteadas por Hoppe [11]: *View Frustum (Frustum de Visión)* y *Screen Space Geometric Error (Error Geométrico en el Espacio de la Pantalla)*. Estas heurísticas determinarán si un tetraedro debe ser refinado, *coarsed* o ignorado. El efecto de usar estos criterios se puede observar en la Figura 9.

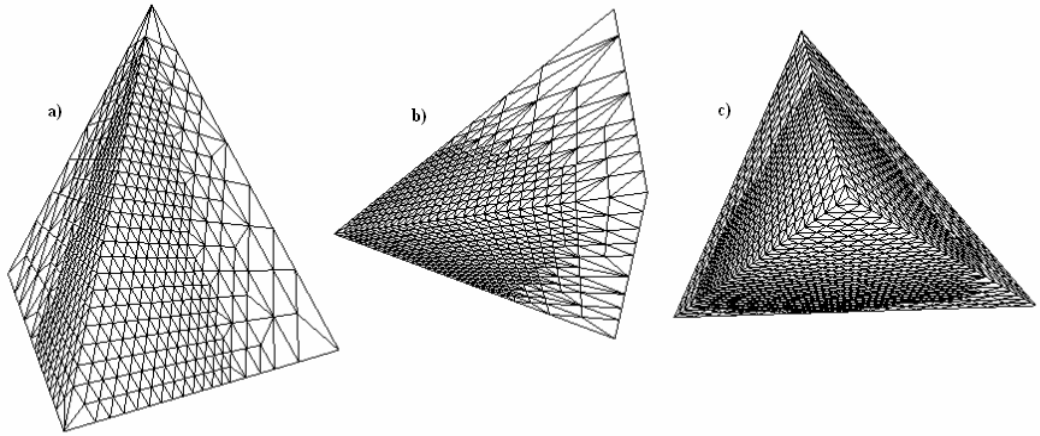

Figura 9: a) y b) Todos los tetraedros fuera de la pirámide de visión son *coarsed*. c) Refinamiento basado en error el geométrico en el espacio de la pantalla.

Frustum de Visión: el objetivo es realizar el *coarsening* de todos los tetraedros que están fuera de la pirámide de visión. Al igual que Hoppe ([11]), usamos esferas como volumen de encierro para

aproximar un tetraedro. Si la esfera está completamente fuera de la pirámide de visión, el tetraedro es coarsed. En caso contrario es ignorado.

Error Geométrico en el Espacio de la Pantalla: se propone calcular el error tomando como base la proyección en perspectiva de la esfera que encierra al tetraedro. Según esta heurística, cuando la proyección en perspectiva ocupa más píxeles verticales que cierta tolerancia, el tetraedro debe ser refinado.

En el momento de la creación de cada tetraedro, calculamos el diámetro de la esfera que lo encierra. Posteriormente usamos ese diámetro precalculado para encontrar la proyección de la esfera, así como para determinar si está dentro de la pirámide de visión. La Ecuación (1) calcula el número de píxeles verticales que proyecta la esfera sobre la pantalla, donde *diam* es el diámetro de la esfera, *yres* es la resolución vertical de la pantalla en píxeles, *dist* es la distancia euclidiana del centro de la esfera al punto de visión y *fov* es campo de visión en radianes. En suma, ésta expresión asegura que la esfera que contiene cada tetraedro tendrá una proyección vertical equivalente a *err* píxeles.

$$err = diam \times yres\left(2 \times dist \times \tan\left(fov \times 0.5\right)\right)^{-1} \quad (1)$$

Nuestro algoritmo funciona incrementalmente, es decir, dado un nuevo punto de visión, no es necesario refinar nuevamente toda la malla, sino que la podemos modificar gradualmente. Si asumimos que el punto de visión no experimenta cambios bruscos, es posible la adaptación de la malla en tiempos interactivos. Nuestro algoritmo procede en dos etapas: *Coarsening* y Refinamiento. El algoritmo se muestra en la Figura 10.

En la etapa de *Coarsening* nos encargamos de eliminar el sobre refinamiento que puede existir en la malla. Dado que una nueva posición del punto de visión introduce un nuevo error geométrico y posiblemente una nueva posición con respecto al *Frustum*, utilizamos ambos criterios para realizar el *coarsening* de la malla. Si un tetraedro esta fuera del *Frustum* o su proyección esta muy por debajo de la tolerancia, será pasado a una resolución menor.

En la etapa de *Refinamiento* utilizamos las dos heurísticas planteadas para decidir si un tetraedro debe ser refinado. Un tetraedro será refinado, si y solo si, disponemos de los detalles calculados en el análisis multirresolución, está dentro del *Frustum* y su error geométrico es mayor que cierta tolerancia dada por el usuario. La tolerancia debe estar expresada en cantidad de píxeles. Debemos tener en cuenta que la etapa de refinamiento no termina hasta que todos los tetraedros de la malla cumplen con la tolerancia de error. Es posible que en esta etapa refinemos algunos tetraedros que fueron *coarsed* en la etapa anterior.

```
Proc RefinamientoSelectivo( Malla )
        ListaDesRefinamiento ← Vacia
        Para cada tetraedro terminal regular t en Malla
                Si t no esta contenido en Frustum :  ListaDesRefinamiento ← t
                Si t esta sobre refinado :    ListaDesRefinamiento ← t
        Fin Para
        Si ListaDesRefinamiento No esta Vacia
                DesRefinaListaTetras( ListaDesRefinamiento ).
        Fin Si
        Repetir
                ListaRefinamiento ← Vacia
                Para cada tetraedro terminal t en Malla
                        Si están disponibles los detalles de t
                                Si t esta contenido en Frustum
                                        Si t.errorGeometrico > Tolerancia
                                                ListaRefinamiento ← t
                                        Fin Si
                                Fin Si
                        Fin Si
                Fin Para
                Si ListaRefinamiento No esta Vacia
                        RefinaListaTetras( ListaRefinamiento ).
                Fin Si
        Repetir Mientras ListaRefinamiento no esta Vacia
Fin Proc
```

Figura10: seudo código para el refinamiento selectivo.

5. RESULTADOS EMPIRICOS

Mostramos empíricamente el desempeño del algoritmo refinando selectivamente un solo tetraedro; nuestra simulación consiste en rotar el tetraedro un ángulo de 0.1 radianes durante 131 cuadros (aproximadamente 4π). Entonces, durante la simulación el tetraedro gira dos vueltas completas; la Figura 11 muestra la cantidad total de tetraedros que se generaron durante la simulación (27,533), así como la cantidad total de vértices (5,295) y aristas (34,642). Podemos apreciar también que la cantidad de tetraedros terminales varía en cada cuadro debido al refinamiento selectivo, y repite su ciclo cuando llega al cuadro 60 (aproximadamente 2π). Los picos de la curva corresponden a variaciones bruscas que se dan por la falta de coherencia que existe entre cuadros, es decir, si la simulación fuera más suave, con incrementos angulares menores a 0.1 radianes, la coherencia entre cuadros aumentaría reduciendo las variaciones en el numero de tetraedros. Notamos también que la cantidad total de tetraedros (es decir tamaño del arreglo de tetraedros) se estabiliza conforme transcurre la simulación. Esto favorece el rendimiento del sistema, ya que no se necesita reservar ni liberar más memoria en tiempo de ejecución; lo mismo sucede con la cantidad total de vértices y de aristas.

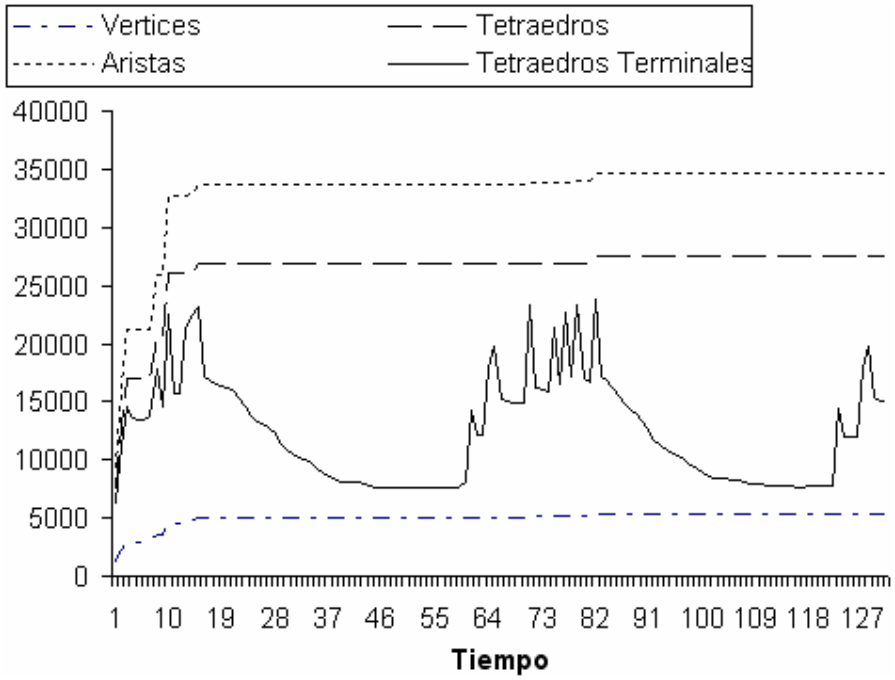

Figura 11: cantidad de vértices, tetraedros y aristas durante la simulación.

La Figura 12 muestra la utilización de memoria durante la simulación. Como se esperaba, el consumo de memoria se mantiene bajo (2.5 MB) y llega a ser estable conforme la simulación se hace mas coherente o predecible. La Figura 13 muestra el tiempo en segundos que toma realizar el refinamiento selectivo (excluido el tiempo de rendering) sobre un procesador Intel Pentium IV corriendo a 2.8 GHz y con 512 MB de RAM; incluimos el numero de tetraedros terminales normalizado para efectos de comparación. Notamos que el máximo tiempo fue 0.64 segundos y ocurrió cuando recién comenzaba la simulación; esto se debe al creciente salto inicial en el número de tetraedros (de 6,344 a 22,540), sin embargo vemos que no se repite después ya que la coherencia es mayor y las estructuras de datos se mantienen estables.

Como dato adicional, podemos decir que el número de normales llego a 158. Este bajo número se debe a que muchos tetraedros comparten la misma normal en la base de datos. Esto confirma la eficacia de nuestro índice basado en el mapa gaussiano, así como el bajo costo de memoria obtenido.

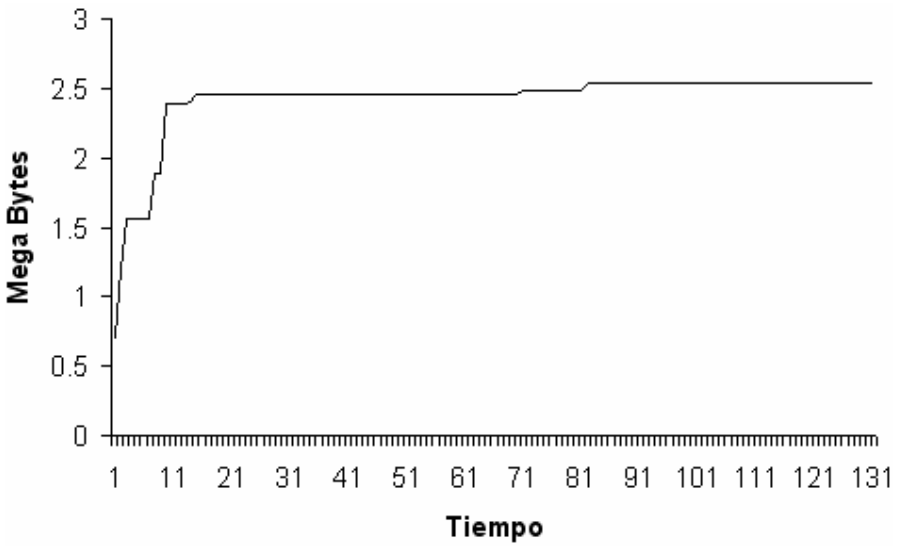

Figura 12

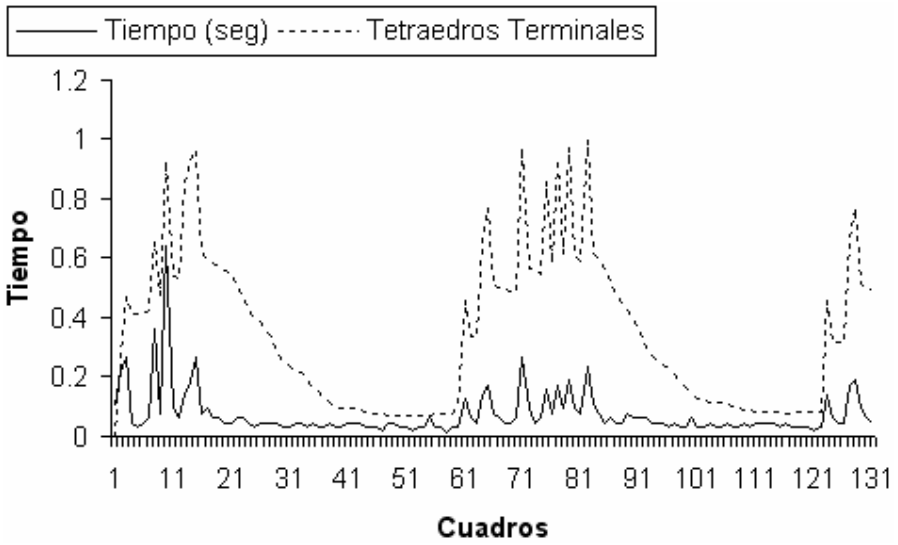

Figura 13

6. TRABAJO FUTURO

En el futuro se explorarán algoritmos de memoria externa capaces de visualizar progresivamente redes tetraédricas que exceden la capacidad de memoria local. También se investigarán algoritmos de refinamiento y *coarsening* aun más eficientes. Se buscará extender el algoritmo para visualizar datos comprimidos y que además el usuario pueda graduar la calidad de visualización que desea en relación con la velocidad de respuesta del sistema, así, el usuario podría elegir tener una "calidad interactiva", "mediana calidad" semi-interactiva o "máxima calidad" no interactiva.

7. REFERENCIAS

[1] R.E.Bank, A.H. Sherman, and A. Weiser, *"Refinement algorithms and data structures for regular local mesh refinement"*, in Scientific Computing, R. Stepleman, ed., Amsterdam IMACS/North Holland, 1983, pp 3-17.

[2] Bey, Jürgen. *"Tetrahedral Grid Refinement"*, Computing, 55(4):355—378 (1995).

[3] Boscardín, Liliana. *"Wavelets Definidas Sobre Volúmenes"*. MS thesis, Universidad Nacional del Sur, 2001.

[4] Castro, Silvia. *"Modelamiento Y Rendering de Volúmenes Mediante la Transformada Wavelet"*. MS thesis, Universidad Nacional del Sur, 1997.

[5] Castro S., Castro L., Boscardín L., De Giusti A., *"Multiresolution Wavelet Based Model for Large Irregular Volume Data Sets"*, in Proceedings of WSCG'2006, pp 101–108, 8, 2006.

[6] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, *"Simplification of tetrahedral volume with accurate error evaluation"*, Proceedings IEEE Visualization'00, IEEE Press, 2000, pp. 85–92.

[7] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno, *"Multiresolution modeling and rendering of volume data based on simplicial complexes"*, Proc.of 1994 Symp. on Volume Visualization, October 17-18 1994, pp. 19–26.

[8] L. De Floriani, E. Puppo, and P. Magillo, *"A formal approach to multiresolution modeling"*, Theory and Practice of Geometric Modeling, Springer-Velrag, 1997.

[9] R. Grosso, C. Lurig, and T. Ertl, *"The multilevel finite element method for adaptive mesh optimization and visualization of volume data"*, IEEE Visualization ´97, pp. 387–394.

[10] H. Hoppe, *"Progressive meshes"*, Proceedings of SIGGRAPH '96 (1996), 99–108.

[11] H. Hoppe, *"View-dependent refinement of progressive meshes"*, Computer Graphics (SIGGRAPH'93 Proceedings), 1997.

[12] Kainz F, Bogart R, *"Technical Introduction to OpenEXR"*, Industrial Light & Magic technical product report, www.openexr.com

[13] S. Kumar, D. Manocha, W. Garrett, and M. Lin. *"Hierarchical back-face computation"*. In Proceedings of Eurographics Workshop on Rendering techniques, pages 235–ff., 1996.

[14] Lohner R, Baum JD. *"Adaptive h-refinement on 3D unstructured grids for transient problems"*. Int J Numer Meth Fluids 1992;14:1407-19.

[15] V. Pascucci, *"Slow growing subdivision (sgs) in any dimension: towards removing the curse of dimensionality"*, Computer Graphics Forum (Proc. EUROGRAPHICS'02), no. 3, 451 – 460.

[16] J. Popovic and H. Hoppe, *"Progressive simplicial complexes"*, ACM Computer Graphics Proc., Annual Conference Series, (SIGGRAPH 97), pp. 217–224.

[17] Shewchuk Jonathan Richard, *"Delaunay Refinement Mesh Generation"*, PhD. Thesis Dissertation, Carnegie Mellon University, 1997.

[18] Shirley P., Tuchman A., *"A Polyhedral Approximation to Direct Scalar Volume Rendering"*, Computer Graphics 24 (November, 1990), 63–70.

[19] H. Zhang and I. Kenneth E. Hoff. *"Fast backface culling using normal masks"*. In Proceedings of Symposium on Interactive 3D Graphics, pages 103–106, 1997.

[20] Y. Zhou, B. Chen, and A. Kaufman, *"Multiresolution tetrahedral framework for visualizing regular volume data"*, Proc. IEEE Visualization '97, pp. 135–142.

[21] Castro Silvia M., *"Compresión de Volúmenes"*, Disertación de Tesis de Doctorado, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Argentina, 2005.