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

PLANTEAMIENTO E IMPLEMENTACIÓN DE HERRAMIENTA COMPUTACIONAL PARA EL ESTUDIO SOBRE LA BISECCIÓN ITERATIVA DE TETRAEDROS POR SU ARISTA MÁS LARGA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

STEFFAN LEÓN WICHE ELORZA

PROFESOR GUÍA:
MARIA CECILIA RIVARA ZUÑIGA

MIEMBROS DE LA COMISIÓN:
PABLO BARCELÓ BAEZA
LUIS MATEU BRULE
JORGE PEREZ ROJAS

SANTIAGO DE CHILE
2017
*

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

POR: Steffan Wiche Elorza
FECHA: JULIO 2017
PROFESOR GUÍA: María Cecilia Rivara

PLANTEAMIENTO E IMPLEMENTACIÓN DE HERRAMIENTA COMPUTACIONAL PARA EL ESTUDIO SOBRE LA BISECCIÓN ITERATIVA DE TETRAEDROS POR SU ARISTA MÁS LARGA

Dado un tetraedro, o pirámide de base triangular, es posible bisectarlo por el plano formado por el punto medio de una arista y los vértices opuestos a la misma lo que produce dos nuevos tetraedros. Si se aplica esto desde el punto medio de una de las aristas más largas se conoce como 'bisección por la arista más larga'. Existe interés en estudiar las propiedades matemáticas de los tetraedros obtenidos por la bisección iterativa tetraedros por su arista más larga.

El tema de memoria propuesto consiste en el desarrollo de una herramienta computacional para estudiar la bisección iterativa de tetraedros por su arista más larga.

Para esto se construye una estructura de datos en forma de árbol en que cada nodo hijo almacene los tetraedros semejantes y distintos que se obtienen al aplicar dicha bisección sobre el tetraedro de su nodo padre.

En esencia esta memoria consta de tres partes: (1) El desarrollo de un sistema que soporta realizar operaciones sobre tetraedros utilizando computación exacta; (2) La implementación de la bisección por la arista más larga sobre tetraedros; (3) La construcción de una estructura de datos que soporte la aplicación iterativa de esta técnica sobre los tetraedros resultantes.

El trabajo y la herramienta computacional se utilizarán para apoyar el estudio teórico sobre el comportamiento de la bisección iterativa de tetraedros, y para desarrollar nuevas técnicas de triangulación con propósitos aplicados.

Agradecimientos

El autor agradece a Robinson Castro por su entusiasmo en el trabajo de intentar rebatir las conjeturas de semejanza desarrolladas en el planteamiento de la solución.

También desea extender un agradecimiento a la profesora María Cecilia Rivara por su dedicación y paciencia en su labor como guía de este trabajo de memoria. Verdaderamente este trabajo no habría llegado a su etapa actual sin su confianza y apoyo.

Tabla de Contenido

1. Introducción.....1

1.1. Objetivos específicosÍndice.....1

1.2. Metodología.....2

1.3. Contribuciones.....3

2. Planteamiento del problema.....4

2.1. El problema en dos dimensiones.....4

2.2 Bisección de tetraedros: el problema tridimensional.....6

2.3. Propuesta.....7

3. Computación Exacta y CGAL.....9

3.1. Investigación.....9

3.1.1. Paradigma de la computación exacta.....9

3.1.2 Estudio de herramientas.....10

3.2. El proyecto CGAL.....11

3.2.1. Tipos de Número en CGAL.....11

3.2.2. Núcleos de CGAL.....12

3.2.3. Estructuras de CGAL.....12

3.2.4. Visualización.....13

3.3 Instalación de CGAL.....13

4. Planteamiento y Desarrollo de Solución.....15

4.1. Planteamiento de la Solución.....15

4.1.1 Elección del tipo de número.....15

4.1.2. Árboles de bisección de tetraedros.....16

4.1.3 Descomposición de Tetraedros.....17

4.2. Tetraedros y Bisección.....18

4.3. Volumen y Calidad.....21

4.4. Semejanza.....22

4.5. Solución Actual.....24

5. Proceso de Implementación.....25

5.1. Primeros Pasos.....25

5.1.1 Compilación y Ambiente de Trabajo.....25

5.1.2 Test Driven Development.....25

5.1.3 Modularidad.....26

5.2 Bisección de Tetraedros.....26

5.2.1. Aritmética de puntos.....27

5.2.2. Estructura y primeros cálculos.....27

5.3. Descomposición de Tetraedros.....28

5.3.1. Volumen.....28

5.3.2. Calidad.....29

5.3.3. Semejanza.....29

5.3.4. Descomposición.....30

5.4 Árboles de Bisección.....30
5.4.1. Construcción minimal.....30
5.4.2. Construcción por niveles.....30
5.4.3. Construcción del Árbol de Bisección.....31

6. Resultados.....33
6.1. Herramientas de Análisis.....33
6.2. Pruebas de Comparación.....34

7. Conclusiones.....37
7.1. TetraCut.....37
7.2. Planteamiento Teórico.....37

8. Bibliografía.....38

1. Introducción

Para la representación objetos geométricos tridimensionales se utiliza frecuentemente mallas de triángulos o cuadriláteros que adoptan la forma de la superficie del objeto a representar. Para algunas aplicaciones esto es insuficiente y se recurre a mallas tridimensionales que consideran el contenido del objeto como parte del mismo. La buena calidad de una malla es esencial para la fidelidad y estabilidad de su computación numérica. El refinamiento de mallas bidimensionales ha avanzado más rápidamente que su contraparte tridimensional, habiendo ya desarrolladas múltiples técnicas para el refinamiento de estas mallas.

Un ejemplo de esto es la bisección iterativa de triángulos por su arista más larga. Consiste en la subdivisión de los triángulos de la malla en triángulos más pequeños formados al trazar nuevas aristas que cruzan los distintos triángulos desde el punto medio de su arista más larga hasta el vértice opuesto. Ha sido demostrado que la aplicación iterativa de este proceso en los triángulos de una malla produce triángulos con calidad mínima acotada, es decir, cuyos ángulos tienen una cota inferior. En el área de la geometría computacional se ha cuestionado la extensión de este método a su contraparte tridimensional, para desarrollar buenas mallas de tetraedros.

Las mallas de tetraedros son una de las representaciones tridimensionales utilizadas más comúnmente. Entre otras también se utilizan mallas de hexaedros, de pirámides con base cuadrada, de prismas triangulares y mallas híbridas que incluyen todos estas figuras. Por su capacidad de cubrir el espacio subyacente a cualquier poliedro cerrado, la malla de tetraedros es posiblemente la más poderosa de las mallas no-híbridas. El éxito del trabajo de un trabajo de investigación respecto de métodos de refinamiento de mallas de tetraedros podría impulsar su uso en las aplicaciones donde es relevante, y ayudar a mejorar la forma en que se manejan estas estructuras.

El objetivo de esta memoria es desarrollar una herramienta computacional para estudiar el comportamiento de la partición iterativa de tetraedros utilizando el método de la bisección por la arista más larga. Se ha requerido hacer una herramienta que lleve a cabo los cálculos de forma exacta, ya que será utilizada en investigación científica. El producto final permite calcular todos los tetraedros distintos generados en este proceso iterativo.

Se trata de un software que permite a los investigadores del área estudiar los tetraedros en detalle de manera de obtener evidencias concretas e ideas acerca de cómo avanzar en los resultados teóricos.

1.1. Objetivos específicos

- Definir el tipo de número a utilizar en el trabajo computacional para manejar aritmética exacta. Con este fin se estudiaron los distintos cálculos a realizar sobre los tetraedros para decidir la forma más eficiente de mantener exactitud computacional.

- Implementar la partición por la arista más larga. Esto es una función que recibe un tetraedro y entrega una lista con los posibles pares de tetraedros obtenidos de esta manera. Se habla de *posibles pares* pues si existe más de una arista que empate por ser la arista más larga, se produce más de una opción de corte.

- Desarrollo e implementación de un algoritmo que permite establecer si dos tetraedros son semejantes. Esto permite construir el árbol minimal de bisección ya que si dos tetraedros son semejantes, los subárboles obtenidos a través de su partición serán iguales en ambos desarrollos.

- Implementación de un algoritmo que permita obtener un árbol de particiones en un formato legible y cómodo. Debe desarrollarse en su forma eficiente, evitando en lo mayor posible utilizar un algoritmo exhaustivo de semejanza.

- Establecer un sistema de análisis sobre árboles de tetraedros que permita obtener estadísticas relevantes sobre los mismos para dilucidar las propiedades de la bisección iterativa.

1.2. Metodología

Para obtener un buen resultado se adoptaron las siguientes prioridades de trabajo:

- Exactitud en el Cálculo

  El objetivo principal del sistema es dar respuestas matemáticamente correctas. La exactitud de los resultados debe prevalecer ante todo, y para esto se busca como primer objetivo encontrar una forma matemáticamente exacta de obtener cada resultado esperado.

- Modularidad

  Para minimizar bugs y maximizar la cobertura de testing se debe atomizar lo más posible el programa resultando en pequeños módulos que resuelvan los distintos problemas que se presenten a través del desarrollo programa. Cada módulo categorizado según el nivel de abstracción en el que trabaje: aritmética de puntos, geometría de tetraedros o bisección de tetraedros. Cada categoría contenida en su respectivo archivo.

- Test Driven Development:

  Es imperativo de que todas las funciones desarrolladas sean fuertemente evaluadas. Un testing preliminar reducirá los tiempos de debugging agilizando el proyecto y fomentando la modularidad, dando cimiento a una etapa de testing más rigurosa.

Además el programa buscará seguir las convenciones de programación de CGAL [9].

Se plantea la siguiente metodología de trabajo:

1. Estudiar los cálculos que deberá realizar el software. Decidir el tipo de número a utilizar en los tetraedros por conveniencia para mantener exactitud aritmética.
2. Estudiar las estructuras de datos existentes en CGAL para representar tetraedros y decidir si utilizarlas ó crear una alternativa. La idea es de que se puedan llevar a cabo todos los cálculos de manera oportuna y de que sean óptimos en consumo de recursos.
3. Una vez establecida la estructura de datos, deberá ser implementada la bisección por la arista más larga de tetraedros.
4. Para ahorrar recursos es necesario calcular si dos tetraedros son semejantes. Este es un paso riesgoso de la implementación del sistema. Se debe establecer cómo comparar rápidamente tetraedros para decidir si son semejantes. Esto es crucial para poder descartar ramas completas del árbol de tetraedros por ser réplicas de otras ramas.
5. Desplegar el árbol de particiones, ya teniendo todas las herramientas desarrolladas, debiera tratarse de una labor larga y exhaustiva, aunque sin mayores dificultades técnicas.

1.3. Contribuciones

Este trabajo de memoria finaliza con la creación de una herramienta computacional que permite el análisis de la bisección iterativa de tetraedros por su arista más larga para su estudio teórico y eventual aplicación de sus resultados en el ámbito ingenieril.

Este sistema ha sido implementado utilizando el paradigma de la computación exacta con el propósito de entregar resultados reales para su posterior estudio. Puede simular la bisección iterativa por su arista más larga de tetraedros, creando árboles de tetraedros que almacenan cada uno de los tetraedros resultantes de esta bisección de manera ordenada y minimal, sin caer en repetir los cálculos sobre tetraedros cuyos semejantes ya han sido analizados. Los resultados han sido comprobados en cada paso y contra los resultados teóricos esperados por la literatura en este tema.

Además, en el trabajo de memoria se dilucidan ciertas propiedades geométricas de la bisección iterativa de tetraedros que simplifican el estudio de esta operación. Estos resultados han sido utilizados para optimizar el sistema de manera de poder llevar el estudio a cabo en su mayor profundidad posible.

2. Planteamiento del problema

Muchas aplicaciones de las áreas de ingeniería y computación gráfica requieren del uso de triangulaciones como tecnología básica par discretizar objetos geométricos complejos. En general requieren que los triángulos sean de buena calidad (con ángulo no muy pequeño). Una de las técnicas utilizadas para refinar triangulaciones es la aplicación iterativa de la bisección por la arista más larga de los triángulos. Sin embargo algunos problemas de modelamiento deben ir un paso más allá y desarrollar *mallas de tetraedros*.

La motivación de este trabajo de memoria es apoyar el estudio teórico de la bisección iterativa por la arista más larga de tetraedros. Se trata del mismo problema en la tercera dimensión. Si la bisección iterativa por el lado más largo de triángulos produce triángulos con mínimo de calidad acotado, ¿la bisección iterativa de tetraedros produce tetraedros acotadamente similares al equilatero? Para abordar el problema se describen las propiedades de la bisección iterativa por la arista más larga de triángulos.

2.1. El problema en dos dimensiones

Definición: Llamaremos bisección por la arista más larga a la partición de un triángulo que se obtiene al trazar la mediana entre el punto medio de su arista más larga y el vértice opuesto. Al hacer esto se obtienen dos triángulos.

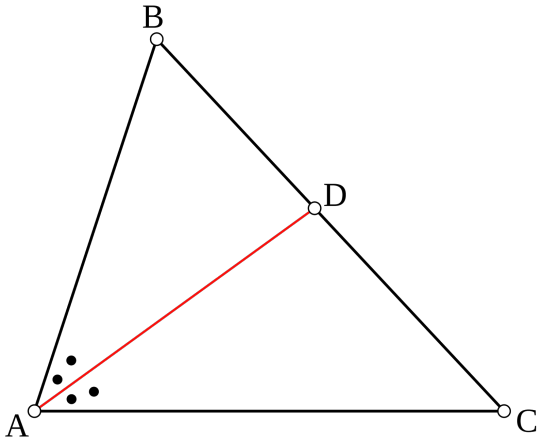


Fig. 2.1 - Para el caso del triángulo ABC, BC es la arista más larga. D es el punto medio de BC. Al trazar el segmento AD se obtienen ABD y ADC, triángulos resultantes de la bisección por la arista más larga del triángulo ABC.

Definición: Repetir la bisección por la arista más larga sobre los triángulos resultantes y los que procedan se entenderá como la *bisección iterativa por la arista más larga.*

Rosenberg y Stenger [1], y Stynes [2] han demostrado que se cumplen las siguientes propiedades para la bisección iterativa por la arista más larga:

1) El ángulo mínimo de cada triángulo resultante es mayor o igual que la mitad del ángulo mínimo del triángulo del que fueron obtenidos. La igualdad solo se cumple para el triángulo equilátero.

2) Se genera un número finito de triángulos semejantes distintos.
3) Existe un conjunto de triángulos de buena calidad del que se producen a lo más 4 triángulos semejantes distintos.

Otra manera de formular la primera propiedad es definiendo una función de calidad de triángulos. Se ha demostrado que en dos dimensiones todas las medidas de calidad son equivalentes.

Definición: Llamaremos la *calidad* de un triángulo al resultado de la función de calidad:

$$q(t) = c * área(t) / L^2$$

Esta función mide el parecido del triángulo *t* al equilatero, donde 'L' es el largo de la arista más larga y 'c' es la constante que hace que para el triángulo equilátero

$$q(t_{equilatero}) = 1$$

Esta función está acotada entre 0 y 1, y permite discernir buenos y malos triángulos por su cercanía al resultado 1. La función de calidad de triángulos muestra la razón entre las aristas (a partir del área) y la arista más larga. Esta razón es en cierta forma proporcional al ángulo mínimo y se utilizó para determinar la cota inferior de dicho ángulo.

Más recientemente Gutiérrez et. al. [6] categorizó los triángulos por la posición del vértice opuesto al lado más largo del triángulo. Los seis grupos de triángulos que se obtienen se muestran en la figura 1.2.

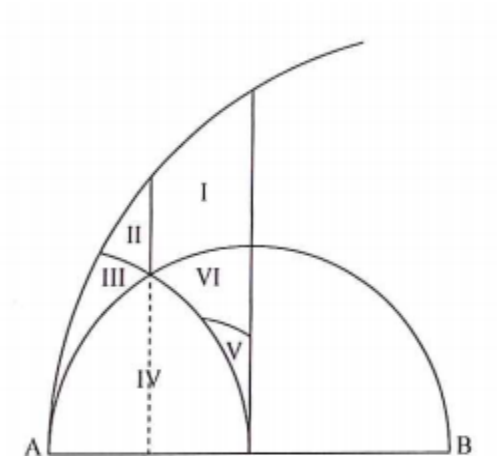


Fig. 1.2 - La figura muestra la categorización del triángulo ABC, según en cuál de las regiones delimitadas se ubica el punto C, dado que AB≥BC≥CA.

Las regiones I y VI corresponden respectivamente a los triángulos casi-equiláteros agudo y obtuso.

Mediante un estudio de casos, Gutierrez et. al.[6] demostró que los triángulos al ser bisectados producen triángulos de categorías específicas, siguiendo el flujo representado por el grafo de la figura 1.3.

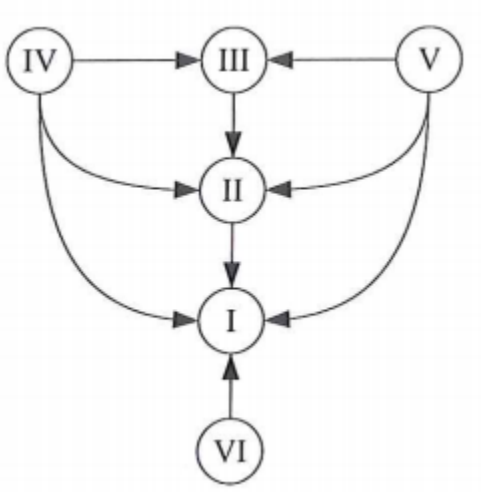


Fig. 1.3 - Cada categoría de triángulos representados por números romanos está especificada por la posición del ángulo opuesto a la arista más larga de un triángulo. El flujo muestra las categorías a las que pueden pertenecer los triángulos resultantes de la bisección al aplicarla sobre un triángulo perteneciente a alguna categoría. Por ejemplo, la categoría V produce triángulos de categorías I, II ó III.

En base a las propiedades de la partición, se han desarrollado algoritmos de refinamiento de triangulaciones que se usan principalmente en las aplicaciones de ingeniería [3][4].

2.2 Bisección de tetraedros: el problema tridimensional

Las propiedades matemáticas de la bisección en 3D no se han demostrado debido a la dificultad del problema. Existe evidencia práctica que sugiere que la bisección de tetraedros se comporta de manera análoga a las dos dimensiones[5]. En otras palabras, se conjetura que:

a) El conjunto de tetraedros diferentes que se producen al bisectar iterativamente un tetraedro por su arista más larga es finito.
b) Una medida de calidad apropiada será acotada por un valor mayor a cero.
c) Existe un conjunto de tetraedros que al ser bisectados iterativamente producen un número muy acotado de tetraedros diferentes.
d) La bisección iterativa de tetraedros de mala calidad aumenta la proporción de tetraedros de buena calidad en el proceso.

La idea es utilizar computacionalmente las nociones usadas en las demostraciones en dos dimensiones y explorar otras posibilidades para obtener evidencia empírica sobre la bisección iterativa de tetraedros por su arista más larga.

Definición: Sea la función descrita como sigue: $q(tet) = c * vol(tet) / L^3$

Diremos que ‘q(tet)’ es la *función de calidad* del tetraedro *tet* en que ‘L’ es su lado más largo y ‘c’ una constante tal que el resultado de la función sea 1. Llamaremos al resultado de esta función la *calidad* del tetraedro evaluado. La calidad de un tetraedro está acotada entre 0 y 1, y los valores más cercanos a 1 son los tetraedros que más se asemejan al equilatero.

La calidad es una medida construida de manera similar a su versión bidimensional. Si se encuentra una cota inferior mayor a cero de esta función se puede deducir de que su ángulo sólido mínimo también se encuentra acotado. Cabe destacar que dos tetraedros semejantes tendrán la misma calidad, pero que la calidad no garantiza semejanza.

2.3. Propuesta

Varias propiedades han sido demostradas en la literatura respecto del comportamiento de ciertos tetraedros. Antti Hannukainen, Sergey Korotov y Michal Křížek [13] han mostrado que la bisección iterativa sobre ciertos tetraedros produce un conjunto acotado de tetraedros.

El primero es el tetraedro de vértices (0,0,0), (1,0,0), (1,1,0) y (1,1,1). Cada iteración de la bisección sobre este tetraedro produce dos tetraedros semejantes entre sí. En la tercera iteración los tetraedros obtenidos son además semejantes al tetraedro original. Esto se muestra en la figura 2.1.

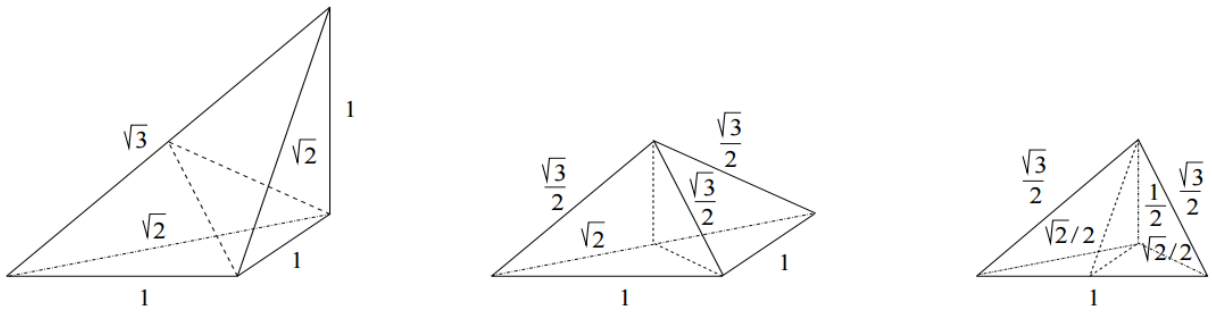


Fig 2.1.: Se muestra la aplicación de la bisección iterativa de tetraedros sobre el tetraedro de vértices (0,0,0), (1,0,0), (1,1,0) y (1,1,1). Se puede observar que las aristas del tetraedro obtenido son proporcionales a las del tetraedro original en razón 2:1.

El segundo es el 'Sommerville space-filler tetrahedron' de vértices (−1,0,0), (1,0,0), (0,−1,1), y (0,1,1). De manera análoga al tetraedro anterior, este encuentra semejanza con los tetraedros obtenidos en la cuarta iteración de la bisección por su arista más larga.

Los mismos autores explican que para el caso del tetraedro equilátero y para el tetraedro (0,0,0), (1,0,0), (0,1,0) y (0,0,1), semejante a la esquina de un cubo, el cálculo se complica pues al producirse tetraedros irregulares con más de un lado más largo se producen diversas opciones a través de las cuales cortar.

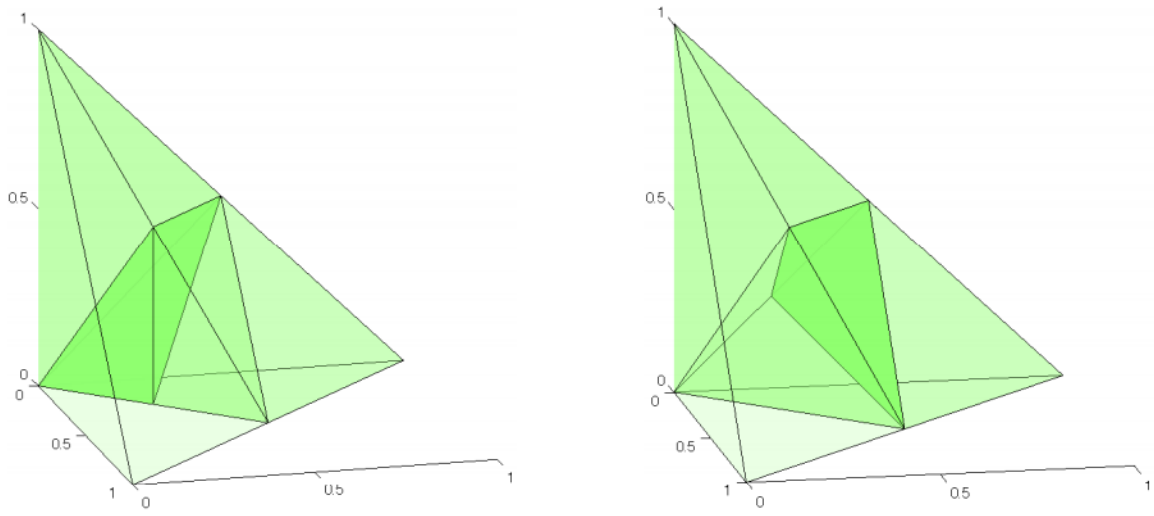


Fig. 2.2. - Se observan dos posibles caminos por los cuales es posible bisectar iterativamente el tetraedro esquina de cubo, produciendo tetraedros distintos.

Andrew Adler [14] especifica que existe una familia de tetraedros que al ser bisectados iterativamente producen un total de tetraedros no semejantes entre sí igual o inferior a 37. En su obra explica que estos tetraedros tendrían su segunda arista más larga opuesta a su arista más larga, y que la diferencia de largo entre su arista más larga y su arista más corta no sería mayor al 5%.

El software propuesto y obtenido en el trabajo de esta memoria encuentra que todos estos tetraedros se comportan de la manera en que se han descrito.

Los tetraedros que producen 3 y 4 tetraedros simulados en el software han otorgado estos resultados. Se ha generado distintos tetraedros pertenecientes a la familia de los tetraedros de Adler y en el sistema entregan menos de 37 resultados.

Además se ha encontrado que los tetraedros esquina de cubo y equilátero generan conjuntos de tetraedros no semejantes de tamaño mayor a 30.000 tetraedros. Hasta este punto no ha sido posible instanciar todos los tetraedros correspondientes a estas bisecciones iterativas.

3. Computación Exacta y CGAL

En la presente sección se describe el trabajo de investigación inicial. Aquí se describe el paradigma de la computación exacta y las herramientas que implementan las nociones de este paradigma. Además se explican las características de las herramientas seleccionadas para desarrollar el trabajo de esta memoria, y las implicancias de las mismas. Se busca dar una base al desarrollo de la solución explicitado en la sección que le procede.

3.1. Investigación

La mayor fuente de incertidumbre de este proyecto son los errores involucrados en el trabajo numérico clásico de computadores basado en redondeamiento. Para manejar este problema se han investigado distintas herramientas que permiten utilizar aritmética exacta en lugar de aproximaciones.

3.1.1. Paradigma de la computación exacta

Con el advenimiento de la computación científica se ha comenzado a dar una importancia mayor a la exactitud de las respuestas numéricas que ofrece un computador. La solución que se plantea a esta falta de robustez se describe en dos requerimientos [8]:

1. Que todos los valores numéricos sean exactamente representados.
2. Que todas las decisiones ramificantes sean libres de error.

Para lograr esto el uso de aritmética de precisión múltiple es un prerrequisito, aunque por sí sola no garantiza que un método computacional sea exacto.

La aritmética de precisión múltiple se basa en sus propios tipos de número (como los ‘double’ y los ‘int’) para representar números exactos. Esencialmente en tipo de número que representa enteros con precisión múltiple. Esta es una forma de representar cualquier entero sin importar el tamaño que estos ocupen en la memoria. Se construyen a partir de múltiples enteros de tamaño fijo de máquina y se denominan comúnmente “BigInteger” o “BigNumber” y se representan como listas enlazadas, arreglos, u otros. Traspasar esta noción a tipo de número de punto flotante consiste en almacenar dos de estos números, el que representa la parte entera y el que representa la parte decimal.

Aumentar los bits de precisión permite computar resultados con mayor exactitud, pero no garantiza cuantos dígitos del resultado sean de hecho correctos. Las distintas implementaciones de sistemas numéricos deben lidiar con esto. Algunas lo hacen desarrollando sistemas de números racionales, otros diseñando intervalos de estos números donde la respuesta existe, otras desarrollan intrincados sistemas de números reales.

3.1.2 Estudio de herramientas

Existen múltiples herramientas desarrolladas con el fin de proveer aritmética exacta a quien decida desarrollar en ellas. Se consideraron las siguientes:

Decimal

Se trata de un módulo desarrollado para Python incluido desde la versión 2.4 en la librería estándar. Se basa en la idea de almacenar los números en una estructura decimal en lugar de binaria. Desarrollan una estructura de datos con los diez dígitos utilizando precisión múltiple. Además implementan manualmente todas las operaciones basándose en la Decimal Arithmetic Specification. Utilizan además una implementación de números en forma de fracción (numerador y denominador) que permite manejar de manera apropiada el sistema de división para asegurar clausura en las cuatro operaciones básicas.

iRRAM

Se trata de una librería de C++ para manejo de números usando un tipo de dato que asemeja a los números reales, basada en el concepto Real RAM. Real RAM (random access machine) es un modelo de computador que trabaja con números reales en lugar de binarios como la mayoría de los computadores actuales [7]. Sus capacidades van desde operaciones aritméticas simples y funciones trigonométricas hasta algebra lineal e incluso ecuaciones diferenciales.

CGAL

El proyecto CGAL consiste de librerías de C++ basadas en el paradigma de la computación exacta que proveen fácil acceso a algoritmos geométricos confiables. El objetivo del proyecto es hacer accesibles al público los distintos algoritmos que permiten resolver problemas geométricos. Con este fin se usan distintos paradigmas de computación exacta y análisis geométrico.

Existen otras herramientas para resolver la necesidad de aritmética exacta, incluidas: Apfloat, float, JLinAlg, MPIR, MAPM y Maple. Además de diversas aplicaciones que lo desarrollan por su cuenta: Maple, Kcalc, Sympy.

Debido a su prestigio, modularidad, estandarización y facilidad de acceso a sus referencias, se ha decidido trabajar con el proyecto CGAL. Su amplia literatura y buenas prácticas evidencian una librería de buena calidad, y en el tiempo se ha demostrado su funcionamiento en múltiples aplicaciones.

3.2. El proyecto CGAL

The Computational Geometry Algorithms Library (CGAL)[9] es un proyecto muy completo que soporta distintos paradigmas de representación numérica. Es desarrollado de manera colaborativa por diversos desarrolladores de investigación trabajando en institutos, universidades y compañías. Para soportar el desarrollo de computación exacta, el proyecto CGAL ofrece distintos *tipos de número* que no incurren en aproximaciones y distintos *núcleos geométricos* de representación espacial. A continuación se presentan estas herramientas ofrecidas por CGAL.

3.2.1. Tipos de Número en CGAL

CGAL dispone de versiones propias de los tipos de número más utilizados en aritmética exacta[10]:

- MP_Float: Es una representación de tipo punto flotante utilizando precisión múltiple. Se vale de un par de enteros de precisión múltiple para representar números decimales de manera exacta.
- Quotient: Se trata de una representación de número racional que guarda un numerador y un denominador de algún tipo de número. Conserva exactitud si tanto numerador como denominador son exactos.
- Interval: Consiste de un par de números que representan un intervalo. De no ser posible llevar a cabo un cálculo de manera exacta, Interval es capaz de decir con certeza si un número es mayor que otro si la totalidad del intervalo se encuentra por sobre el límite superior del otro. Puede ser construido por números exactos soportados por CGAL, y es exacto mientras el tipo de número almacenado sea exacto.
- Lazy_exact_nt: Es un tipo de número que retrasa el cálculo del tipo de número que tenga almacenado hasta que es necesitado. En otras palabras, impone el paradigma de computación *lazy* sobre cualquier tipo de número. Es exacto mientras el tipo de número almacenado sea exacto.

Además de los tipos de números desarrollados por el equipo de CGAL, la librería soporta wrappers que permiten el uso de los tipos de números provistos por otras librerías. El manual de CGAL hace hincapié en que las versiones externas de sus mismos tipos de número pueden resultar más eficientes en ciertos casos. Existe un tipo de número introducido por estas librerías del cual CGAL no tiene una versión propia:

- Real: Se trata de una representación que se aproxima al número real de las matemáticas. Guarda información sobre los valores que multiplican a las distintas raíces que componen a los números reales. Esto funciona de forma anidada, y si bien es una estructura altamente eficiente, sus tiempos de computación y uso de memoria son altos.

Las librerías soportadas por CGAL vía wrapper son GMP, LEDA y Core.

- Gnu Multiple Precision (*GMP*) arithmetic library es una librería libre de aritmética de precisión arbitraria desarrollada originalmente por Torbjörn Granlund. Soporta enteros, números racionales y números de punto flotante. Sus estructuras son más eficientes que las que provee CGAL y ofrece el uso de una amplia gama de funciones para usar con sus tipos de número.
- A Library of Efficient Data Types and Algorithms (*LEDA*) es una librería para la computación exacta desarrollada por *Algorithmic Solutions Software GmbH*. Maneja el tipo de dato real de una forma altamente Real: Se trata de una representación que se aproxima al número real de las matemáticas. Guarda información sobre los valores que acompañan eficiente, lo que entrega resultados matemáticamente reales mientras se utilice de forma correcta.
- A Core Library for Robust Numeric and Geometric Computation (*Core)* es una librería en C++ que implementa diversos algoritmos basados en los principios de Exact Computation Geometry (ECG). Desarrollado en un principio por Vijay Karamcheti, Chen Li, Igor Pechtchanski, y Chee Yap, esta librería fue diseñada para ser extremadamente fácil de utilizar.

3.2.2. Núcleos de CGAL

Dentro de CGAL se encuentran distintos *núcleos geométricos* que proveen de entidades geométricas y operaciones primitivas sobre ellas. Estos núcleos representan distintos paradigmas de parametrización del espacio. Existen dos categorías de núcleo: representaciones *cartesianas* y representaciones *homogéneas*.

Las coordenadas homogéneas en contraste a las cartesianas son invariantes de escala. Utilizan una coordenada extra para representar un número que representa la escala. Esto es útil en geometría proyectiva y por tanto se descarta como opción de trabajo. Dentro de las representaciones cartesianas se encuentran, por ejemplo, las coordenadas cartesianas simples en dos y tres dimensiones, coordenadas circulares, cilíndricas, esféricas, y representaciones para mayor número de dimensiones.

3.2.3. Estructuras de CGAL

Para modelar objetos en el espacio tridimensional CGAL ofrece la estructura de datos Point_3. Esta consiste de las coordenadas de un punto en el espacio tridimensional utilizando la forma propia del núcleo del que se le llame. Las coordenadas pueden estar compuestas por cualquiera de los tipos de número listados.

Además ofrece una estructuras de datos para poliedros como por ejemplo Polyhedron_3. Esta consiste de la información de sus vértices, aristas, caras y una relación de incidencia entre ellas. Los vértices se caracterizan por su ubicación espacial. Las aristas se componen de dos semiaristas. Cada semiarista se compone de dos vértices idénticos a los de su contraparte, y una dirección opuesta. Las caras del poliedro se definen por algún número de semiaristas en un mismo plano en una dirección.

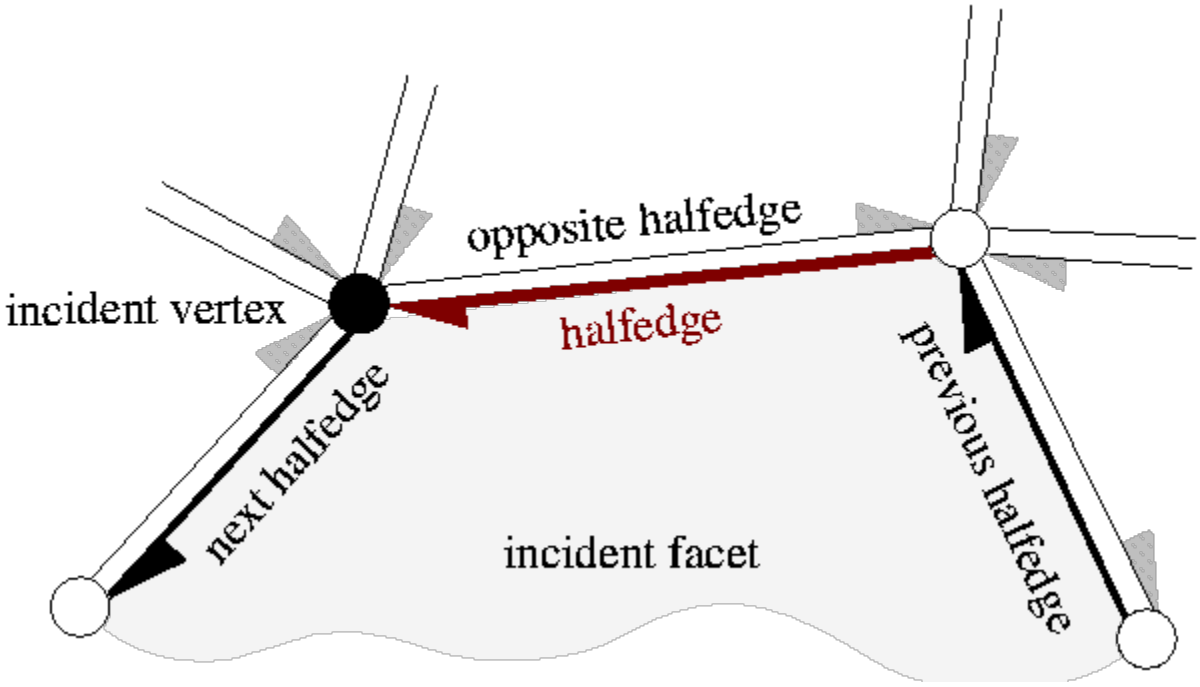


Fig. 3.1. - Se observa una semiarista direccionada. (halfedge).

Una alternativa para representar tetraedros se encuentra en Tetrahedron_3, una estructura de datos para modelar tetraedros direccionados compuesto por los cuatro vértices del tetraedro. La dirección del tetraedro depende de la posición del último vértice respecto de los anteriores tres, que definen un plano que divide el espacio en un lado positivo y uno negativo.

3.2.4. Visualización

Una de las ventajas de utilizar las estructuras de datos propias de CGAL es que nativamente pueden ser desplegadas en distintos sistemas de visualización. Las demostraciones incluidas en CGAL utilizan principalmente *Geomview* y *QGLViewer*. Se estimó conveniente utilizar uno de estos por motivos de validación de resultados.

- Geomview es un programa de visualización tridimensional de Unix. Permite al usuario ver y manipular objetos tridimensionales, tanto estáticos como cambiantes. Su uso es de implementación sencilla y no requiere instalación en sistemas basados en Unix.
- QGLViewer es una librería de C++ que implementa las típicas funciones de visualización, pero que en contraste a Geomview permite un manejo más fino de la interfaz y del despliegue de objetos para el programa.

3.3 Instalación de CGAL

La instalación de CGAL está descrita en el manual de CGAL[9]. Para su compilación se utiliza CMake. Por lo diversas que son las librerías soportadas para conexión con CGAL, se recomienda utilizar CMake-GUI para generar de forma más sencilla el listado de dependencias a compilar en cada proyecto. CMake-GUI permite al usuario ver cuáles dependencias se encuentran especificadas en cuales variables, y cuales de ellas no han podido ser automáticamente identificadas por CMake. Una vez llenos estos formularios de variables y de haber instalado las dependencias correspondientes, CMake genera los Makefiles de CGAL. Estos aún pueden encontrar dependencias faltantes, que deberán ser instaladas también. Una vez que los Makefiles funcionen correctamente, se tiene una instalación de CGAL al hacer Make install.

ĆGAL en sí mismo no es una librería muy grande, pero es insuficiente para desarrollar visualizaciones por sí mismo. Por esto usa otras librerías en sus demostraciones de lo que es modelado en el sistema. Para aprender a utilizar correctamente CGAL fue necesario hacer correr alguna de las demostraciones gráficas en tres dimensiones incluidas en la librería. Después de algunos intentos fallidos, se logró hacer correr un set de demostraciones que utilizan Geomview como sistema de visualización (ver Fig. 2.2).

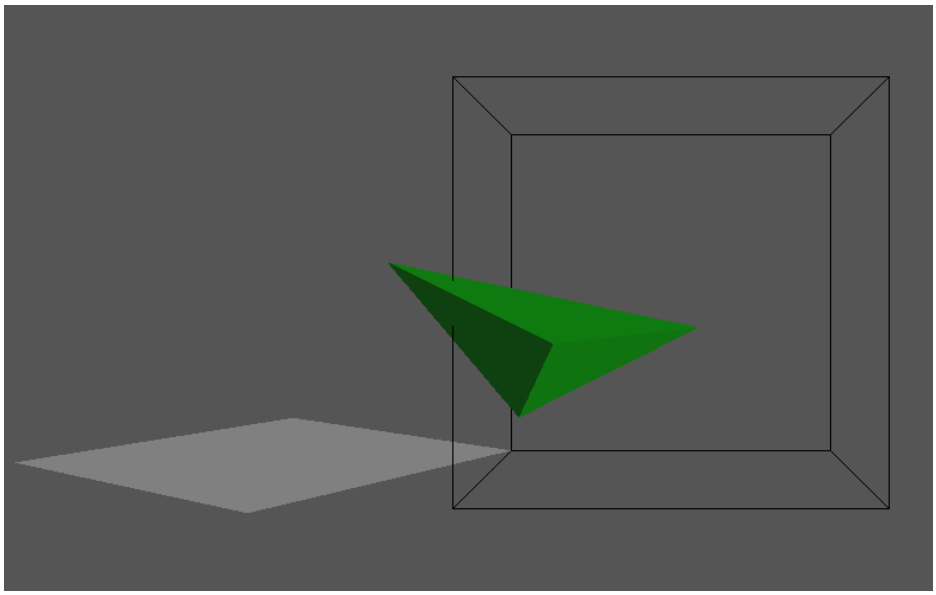


Fig. 3.2 - Se muestra un tetraedro verde, un plano gris y una caja transparente desplegados en Geomview.

4. Planteamiento y Desarrollo de Solución

El objetivo principal del sistema desarrollado es construir una estructura en forma de árboles que guarde información acerca de los resultados de la bisección iterativa de tetraedros utilizando computación exacta para su cálculo. Esto se puede descomponer en tres partes:

- Representar la bisección por la arista más larga de tetraedros utilizando computación exacta.
- Descomponer tetraedros en todos los posibles tetraedros resultantes del proceso de bisección.
- Construir un *árbol de bisección iterativa*. Es decir, realizar iterativamente este proceso de descomposición sobre los tetraedros resultantes y guardar la información obtenida en una estructura de datos con forma de árbol.

En esta sección se explicará el planteamiento inicial de la solución, y cómo este fue evolucionando a medida de que se evidenciaban propiedades de la bisección iterativa de tetraedros. Los detalles de implementación del sistema serán discutidos en la sección 5.

4.1. Planteamiento de la Solución

El programa provee resultados exactos en la aritmética involucrada en la generación de árboles de bisección de tetraedros. Para lograr esto se decidió utilizar alguno de los tipos de números ofrecidos por CGAL o de las librerías que soporta como la base del sistema. Esta librería se presenta para C++ y por tanto es este el lenguaje con el que se trabaja.

Para decidir cual de estos tipos de número conviene utilizar se han estudiado las operaciones que se realizarán a través de la ejecución del programa. La mayor parte de las operaciones realizadas por el sistema son sumas, restas y multiplicaciones. Sin embargo para determinar el punto medio entre dos puntos en el espacio es necesario utilizar división por dos, y para obtener la distancia entre dos puntos es necesario utilizar raíces cuadradas.

4.1.1 Elección del tipo de número

Para manejar raíces cuadradas es necesario utilizar el tipo de número Real otorgado por LEDA ó Core. Con esto es posible realizar las operaciones exactas de suma, resta, división, multiplicación y raíz n-ésima. Se trata de una estructura complicada que crece rápidamente en consumo de recursos, especialmente memoria. Se debe tener en cuenta que entre más memoria sea reservada por los números, menor será el alcance del análisis del programa por consumo de memoria.

Con esto en mente se decidió manejar las distancias entre dos puntos sin resolver las raíces cuadradas, directamente comparando sus cuadrados. La idea es proceder de esta forma desarrollando el software evitando el uso de raíces cuadradas. La

implementación de CGAL permite hacer del cambio de paradigma algo sencillo de lograr, ya que los números se manejan de la misma forma sin importar cómo estén diseñados, siempre y cuando soporten las operaciones utilizadas. Es por esto que si se llega a un punto en que es imposible continuar sin resolver raíces cuadradas, el trabajo no se pierde, sino que se adapta fácilmente al nuevo tipo de número.

Los puntos medios y otras operaciones que requieren la división no pueden ser evitadas de la misma manera pues la constante división de tetraedros invocará cada vez más divisiones anidadas. Por tanto, las operaciones requeridas de manejo para el tipo de número son suma, resta, multiplicación y división. Por esto se toma la decisión de utilizar una representación racional de números. Se trata de los números *quotient* de CGAL que guardan información de numerador y denominador para representar un número en forma de fracción. Ambos operandos determinados con el tipo de número de punto flotante con precisión múltiple. Esto le permite a TetraCut representar cualquier número perteneciente al conjunto de los racionales. Se utilizan números de punto flotante por natividad de CGAL. Estos pueden ser cambiados en cualquier momento por enteros de precisión múltiple y los resultados serían los mismos.

Definición: Llamaremos *quotient (quo)* al tipo de dato que representa un número racional a partir de un numerador y un denominador, ambos consistentes de un número de punto flotante con precisión múltiple.

Se toma así la decisión de trabajar con el tipo quotient como la base de operatoria del sistema. Una vez resuelta la problemática del tipo de número se presenta la de la estructura de datos.

4.1.2. Árboles de bisección de tetraedros

Para ahorrar recursos se propone en un principio la siguiente definición:

Definición: Un *árbol de bisección de tetraedros* es una estructura con forma de árbol que guarda en cada nodo la información de un tetraedro y un puntero hacia los nodos que representan a los tetraedros que son resultantes de la bisección por la arista más larga del tetraedro guardado. Un árbol de bisección de tetraedros será una estructura no-autocontenida. Esto quiere decir que si un tetraedro *o un tetraedro semejante* ya pertenece al árbol, cada nueva aparición resultará en un nodo terminal.

```
Nodo:
    Tetraedro tet
    Bool terminal?
    Nodo[12] hijos
```

El tamaño de un árbol crece exponencialmente según el número de nodos hijos tenga cada nodo. Si calculamos todos los tetraedros que produce el tetraedro con más lados largos (el equilátero) al ser bisectado por todos sus posibles lados es de 12. Esto quiere decir que un árbol de bisección de tetraedros de 50 niveles potencialmente

contendrá 12^{50} nodos. Considerando el gran tamaño que adoptarán los números que representan estos tetraedros a medida de que avanzan los niveles se espera un alto consumo de memoria y tiempo de procesamiento. Esta definición permite desarrollar el árbol reservando espacio para los cálculos que no han sido aún realizados.

El árbol de bisección buscará ser construido por niveles. El razonamiento detrás de esto es que si se construye en profundidad se encontrarán las primeras apariciones de muchos tetraedros muy profundas en el árbol en la primera rama, siendo que quizás se encontraban mucho más cerca de la raíz por la siguiente rama.

4.1.3 Descomposición de Tetraedros

Definición: Llamaremos *descomposición de un tetraedro* al conjunto de todos tetraedros resultantes de bisección por la arista más larga de dicho tetraedro por cada uno de sus lados más largos, tales de que no son semejantes entre sí.

Para construir cada nodo del árbol de tetraedros de manera no-autocontenida se utiliza esta noción de descomposición. Los nodos del árbol contienen un tetraedro y tienen por nodos hijos a los que contienen a los tetraedros resultantes de su descomposición. Para obtener una descomposición se lleva a cabo la bisección por cada uno de los lados más largos. Luego se construye un conjunto agregando los tetraedros que no son semejantes entre sí.

A través del trabajo de memoria se dió cuenta de una propiedad de la descomposición de tetraedros que acota el espacio de memoria requerido por la misma. Se demuestra a continuación:

Teorema: El conjunto de tetraedros resultado de la descomposición de un tetraedro contiene a lo más 8 tetraedros.

> *Demostración:*
>
> *Cada tetraedro produce a lo más dos tetraedros por bisección, y se hace una bisección por cada arista más larga. Se hará una demostración por casos dependiendo del número de aristas más largas del tetraedro inicial N.*
>
> *Caso N=6: Se trata del tetraedro equilátero. En cada corte del tetraedro equilátero se producirán dos tetraedros semejantes entre sí. Con esto ya se descartan suficientes casos para la demostración, pero además todos los distintos cortes producen el mismo tetraedro. La descomposición del equilátero produce un conjunto de tamaño 1.*
>
> *Casos N=1, N=2, N=3 y N=4: Su descomposición produce trivialmente a lo más 8 lados largos.*
>
> *Caso N=5: Sea el Tetraedro ABCD de 5 aristas de largo L y una de largo S donde L>S. Supongamos sin pérdida de generalidad que la arista más corta es AD. Por atravesar el eje de simetría, la bisección por BC producirá dos tetraedros semejantes entre sí. Las biseccciones AB, AC, BD y CD son reflejas respecto de*

los dos ejes de simetría produciendo los mismos pares de tetraedros. Por lo tanto su descomposición produce a lo más 5 tetraedros diferentes.

Como en todos los casos el tamaño del conjunto obtenido es de a lo más 8, se demuestra esta propiedad.

4.2. Tetraedros y Bisección

Para modelar y almacenar los datos de los tetraedros se propuso una la estructura de tetraedros compuesta por los cuatro vértices de un tetraedro utilizando la estructura Point_3 compuesta por quotients.

Desarrollar la bisección por alguna arista requiere encontrar el punto medio de la arista en cuestión. Utilizando una función que encuentra el punto medio M entre dos puntos en el espacio, es posible modelar esta bisección creando dos nuevos tetraedros que utilicen de vértice el punto medio obtenido y uno de los vértices de la arista bisectada y el resto de los vértices del tetraedro original no relacionados con la arista bisectada.

Esta simple estructura permite tanto calcular la distancia al cuadrado como la bisección de tetraedros por alguna arista. Al poder calcular la distancia al cuadrado es posible reconocer por cual o cuales aristas realizar la bisección. Por esto se adopta la estructura de datos propuesta, sujeta a cambios en el proceso de programación, y se plantea una primera versión de la función de bisección.

Tetraedro:

Vértice A, B, C, D

Bisección por AB:

Tetraedros X, Y

Vértice M = Punto Medio(A, B)

X = (M, A, C, D)

Y = (M, B, C, D)

Return (X,Y)

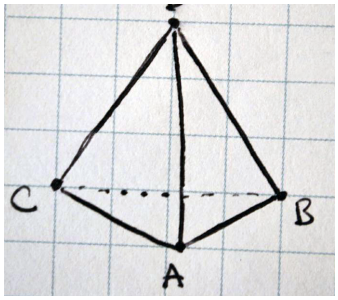
Para facilitar el análisis de los datos se incluye además una convención sobre el orden de los vértices de un tetraedro. Se trata de imponer una restricción sobre D respecto de A, B y C.

Definición: Diremos que un tetraedro ABCD sigue la *convención de construcción de tetraedros* si la cara ABC observada desde D se describe con sus vértices ordenados en contra del sentido de las manecillas del reloj.

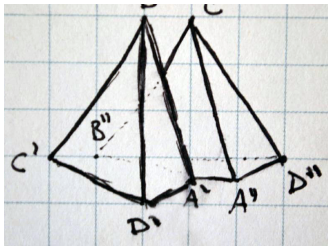
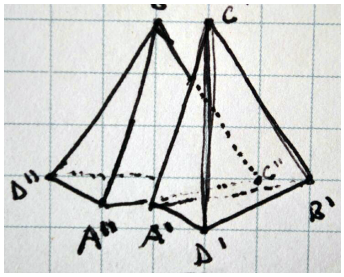
Para evitar revisar si cada tetraedro sigue la convención, se elaboró una parametrización tal de que si un tetraedro que sigue la convención es bisectado de esta

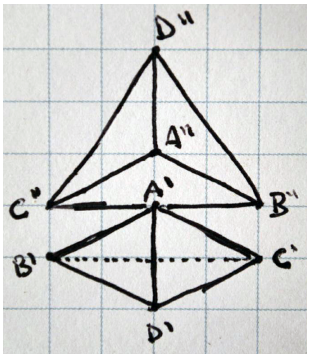
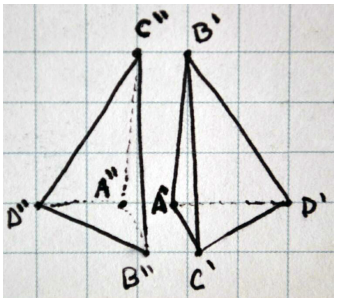
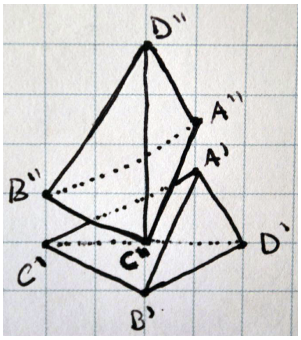
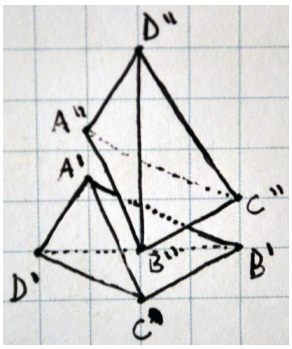
manera, los tetraedros resultantes de dicha bisección también seguirán la convención de construcción de tetraedros.

El diagrama de a continuación muestra claramente la forma en que serán bisectados los tetraedros para asegurar mantener la convención. Cabe destacar de que si el tetraedro ingresado no sigue la convención, ninguno de los tetraedros resultantes de bisección lo hará.

<table>
<tr>
<td> 
</td>
<td>Definición: Sea el tetraedro ABCD que sigue la convención de construcción de tetraedros. Sea M el punto medio obtenido para la bisección.

Llamaremos X e Y a los tetraedros de vértices A’B’C’D y A”B”C”D” respectivamente resultantes de la bisección de ABCD.

Se define la parametrización de corte como sigue:</td>
</tr>
<tr>
<td> 
</td>
<td>Bisección por AB (M = Punto Medio(A,B))
X: Y:
A’ = M A” = M
B’ = D B” = C
C’ = C C” = D
D’ = A D” = B</td>
</tr>
<tr>
<td> 
</td>
<td>Bisección por AC
X: Y:
A’ = M A” = M
B’ = B B” = D
C’ = D C” = B
D’ = A D” = B</td>
</tr>
</table>

	Bisección por AD X: Y: A′ = M A″ = M B′ = C B″ = B C′ = B C″ = C D′ = A D″ = D
	Bisección por BC X: Y: A′ = M A″ = M B′ = D B″ = D C′ = A C″ = B D′ = B D″ = C
	Bisección por BD X: Y: A′ = M A″ = M B′ = A B″ = C C′ = C C″ = A D′ = B D″ = D
	Bisección por CD X: Y: A′ = M A″ = M B′ = B B″ = A C′ = A C″ = B D′ = C D″ = D

Esta es la bisección que utiliza actualmente el sistema. Se puede observar a través de las imágenes de que en todos los casos se mantiene la convención de construcción de tetraedros.

Definición: A la bisección por punto medio de arista arbitraria que siga esta parametrización se le llamará *corte.*

4.3. Volumen y Calidad

Para analizar los arboles de bisección tetraedros se ha usado la función de calidad de sus tetraedros basada en el volumen, que extiende la función usada en triángulos. La ecuación de calidad está descrita como sigue:

$$q(tet) = c * vol(tet) / L^3$$

Existen dos problemas para la implementación de esta función de calidad. El problema menor es que se debe encontrar el valor del lado más largo a la tercera, que requiere obtener finalmente la raíz cuadrada, y que c=√72 (el valor que hace que la función evalúe a 0). Como la calidad, al igual que la medida de los lados se utiliza solo en comparaciones, y esta comparación puede lograrse comparando los cuadrados. Aún se requiere, sin embargo, obtener el volumen de un tetraedro.

La fórmula clásica de volumen de tetraedros está dada por: $\frac{Area(base) \cdot Altura}{3}$

Se puede ver como obtener cualquiera de estos valores incurre en la necesidad de obtener el valor exacto del largo de los lados del tetraedro y además deducir la altura a través de pitágoras recurriendo a múltiples raíces cuadradas. Sin embargo, existe otra forma de determinar el volumen de un tetraedro sin incurrir en estos cálculos.

Dado un tetraedro OABC donde O está descrito en el origen de coordenadas. Sharygin [12] logra obtener el volumen a partir de la siguiente fórmula:

$$V = \frac{1}{6} \left| A \bullet (B \times C) \right|$$

Que considerando productos punto y cruz solo manejan multiplicaciones es perfectamente implementable dentro del sistema definido con quotients. De no estar ubicado en el origen el tetraedro todo lo que se debe hacer es trasladarlo al origen respecto de alguno de sus puntos y resolver la fórmula para el tetraedro trasladado. La traslación consta simplemente de restar a todos los vértices el punto que describe al vértice que se llevará al origen.

4.4. Semejanza

Parte del desarrollo del árbol de bisección de tetraedros incluye encontrar el árbol minimal de semejanza. Si dos tetraedros son semejantes entre sí, producirán subárboles de bisección semejantes. En consecuencia, en el árbol solo se incluye el primer subárbol encontrado, lo que lleva a la necesidad de calcular la semejanza entre dos tetraedros.

A través del desarrollo de la memoria se probaron tres conjeturas sobre el cálculo computacional de la semejanza que permiten reducir el volumen de los cálculos.

Conjetura 1: Si se ordenan los segmentos que componen un tetraedro por tamaño y se compara esta misma lista obtenida de otro tetraedro escalado al tamaño del primero, entonces ambos serían semejantes.

A través del proceso de implementación del programa, esta noción de semejanza fue fue descartada ya que se encontraron contraejemplos.. Robinson Castro, estudiante de la FCFM en el Departamento de Ciencias de la Computación, encontró que se puede formar dos tetraedros no semejantes entre sí utilizando las aristas de largo (1,1,1,0.6,0.6,0.6) utilizando en uno los segmentos de largo 1 como base y en otro los segmentos de largo 0.6 como base.

Cabe hacer nota que la implementación considera un paso para ahorrar tiempo. Si dos tetraedros t_1, t_2 son semejantes, entonces $q(t_1) = q(t_2)$ y esto se verifica antes de comparar por semejanza directamente. Se pensó de que no era posible formar dos tetraedros utilizando los mismos segmentos y que además compartiera su calidad. Cuando los lados son los mismos, no hace mucha diferencia comparar entre calidad y volumen.

Conjetura 2: Si se ordenan los segmentos que componen un tetraedro por tamaño y se compara esta misma lista obtenida de otro tetraedro escalado al tamaño del primero, y ambos comparten volumen entonces ambos serían semejantes.

Con respecto a la conjetura 2 Robinson Castro [13] nuevamente encontró un contraejemplo en la construcción de dos tetraedros que comparten volumen y lista de segmentos ordenados (y por tanto calidad). La lista de segmentos que los producen es (1,1,1,1,$\sqrt{0.5}$,$\sqrt{0.5}$). Uno de ellos tiene los segmentos de largo $\sqrt{0.5}$ opuestos en el tetraedro y el otro los tiene adyacentes, como se puede ver en la figura.

Se decidió entonces probar la conjetura 3 basada en comparar pares de aristas opuestas ordenadas.

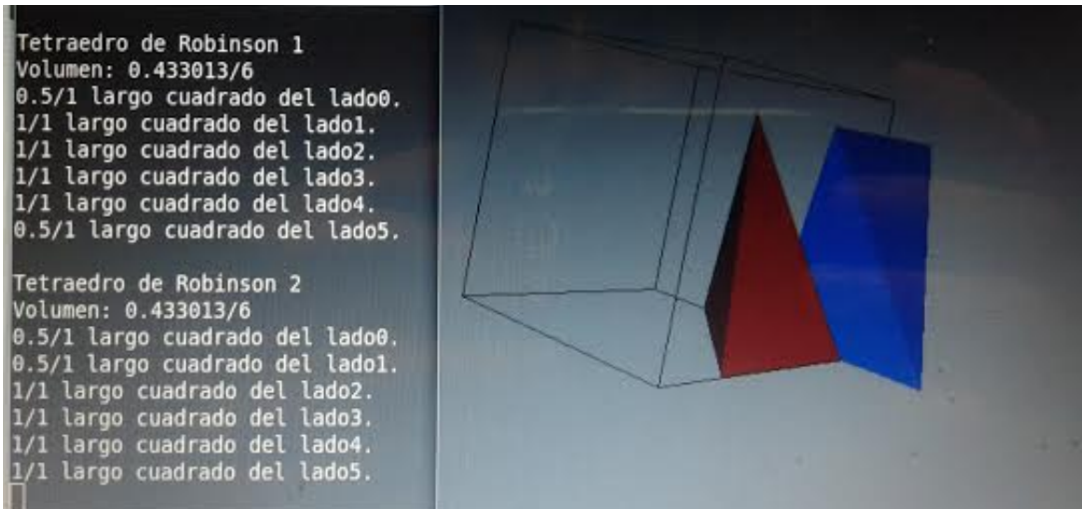


Fig. 4.1. - Se muestran dos tetraedros que fueron modelados utilizando Geomview y la estructura Tetrahedron_3 de CGAL y números de tipo quotient. Se puede observar como la forma de ambos tetraedros difiere, sin embargo tienen el mismo volumen y comparten los segmentos de recta que los componen.

Conjetura 3: Cada tetraedro se representa por tres pares adecuados de longitudes de aristas definidas como sigue: a) cada par (L_1,L_2) corresponde a las aristas opuestas; b) (L_1,L_2) se ordenan de modo que el primero sea mayor que el segundo; c) la representación ordena los pares por el largo de su arista más larga. Si dos tetraedro escalados tienen iguales representaciones, los tetraedros son semejantes. Esto no ha sido demostrado, y de existir un contraejemplo a este criterio en conjunto con discriminación por calidad, entonces deberá ser utilizada una última noción basada en comparar los triángulos que componen las caras de los tetraedros.

Lema: Si las cuatro caras triangulares de dos tetraedros son semejantes entonces los tetraedros serán semejantes.

Esta operación es lenta, sin embargo, y repercutirá negativamente en los tiempos de procesamiento del sistema en comparación a la definición actual.

4.5. Solución Actual

La solución actual del problema consiste en construir árboles de bisección de tetraedros utilizando la noción de descomposición de tetraedros. Para representar los tetraedros se utilizarán estructuras que guarden los vértices de los tetraedros. Para formular la descomposición de los tetraedros se utilizará la semejanza para descartar tetraedros similares. Además, antes de agregar los tetraedros de cada descomposición al árbol se comparará estos tetraedros con los que actualmente existen en el árbol de modo de no formar árboles autocontenidos.

5. Proceso de Implementación

En la presente sección se discutirá el proceso de implementación de la solución descrita en la sección anterior.

5.1. Primeros Pasos

Dado el contexto de CGAL y la disponibilidad de ejemplos y demostraciones gráficas de uso de la librería, se decide utilizar algunos de estos como templates para comenzar

la programación del sistema. Estos ejemplos utilizan CMake para su compilación.

5.1.1 Compilación y Ambiente de Trabajo

CMake es un programa que genera makefiles de manera automática, encargándose de que el programa compilado tenga acceso a todas las librerías que necesita. Las librerías no siempre se encuentran en su ubicación estándar en el sistema de archivos de un computador, de modo que es necesario especificar la ubicación de estas librerías. Para simplificar este proceso se utiliza la interfaz provista por CMake-gui, programa que tiene este propósito. Una vez creado el makefile del sistema todo lo que se necesita para correr un programa es compilarlo utilizando el make.

Para programar se utilizó Geany como procesador de texto, y una consola de comandos para rápidamente compilar y ejecutar los programas. Pero para realmente aplicar metodologías ágiles es necesario una plataforma de unit testing. Para esto se utilizó la librería Catch.

Catch es una librería que implementa unit testing en un solo archivo y sin requerir acceso a librerías externas. Solo requiere ser incluido el archivo Catch.hpp en el sistema para utilizar sus funciones. Catch tiene un buen sistema de modularización, y en general permite el uso las buenas prácticas del unit testing.Para asegurar la exactitud del cálculo, la correcta funcionalidad del sistema y la agilidad del desarrollo, se impone el uso de Test Driven Development.

5.1.2 Test Driven Development

Normalmente Test Driven Development consta de un ciclo de desarrollo que impone los requerimientos de funcionalidad en un test. Si el test es pasado en conjunto con todo el resto de los tests, se procede con el siguiente test. Sino, deberá ser refactorizar el código para resolver el error. Sin embargo en el caso de este sistema cuyo requerimiento es rigurosidad científica, parte del proceso de desarrollo es testear fuertemente la funcionalidad de cada módulo del programa. Es por esto que al proceso de desarrollo se le impone un ciclo de testing en 2 fases:

1) Test de implementación: Se define previo al desarrollo de la función la forma de utilizarla creando un test sencillo que la utilice y corrobore su resultado.
2) Test de overflow: Se intenta hacer overflow a la función utilizando números muy grandes. Esto se debe hacer una vez que se considere implementada la función.

El motivo de la segunda fase: Test de overflow, es que esto demuestra que el sistema está en realidad utilizando un tipo de número que no hace sobreflujo al sobrepasar cierto tamaño. Muestra que se está obteniendo correctamente un resultado que un *double* no habría logrado.

Esto no aplica a la construcción del árbol de bisección debido a la falta de precedentes al respecto de la estructura.

5.1.3 Modularidad

El sistema se divide en cuatro módulos:

1) Point Arithmetics: Es el programa que formula el tipo de número y lo llama desde CGAL, y que trae desde el núcleo a la estructura Point_3 conformada por este tipo de número. Todas las funciones que trabajen exclusivamente con puntos en el espacio se encuentran programadas aquí.
2) Tetra: Este módulo implementa la estructura de tetraedros llamada Tetra, y que utiliza a Point Arithmetics para manejar los cálculos que envuelvan manejo genérico de tetraedros.
3) TetraCut: El programa principal del sistema donde se implementan las operaciones principales requeridas por el sistema: la bisección por la arista más larga, la descomposición de tetraedros y la construcción del árbol de bisección.
4) Archivos de Testings: Tres programas dedicados a la evaluación de los otros tres módulos descritos.

El desarrollo de los distintos módulos se basa en los requerimientos para alcanzar en orden los tres hitos que concluyen el funcionamiento del sistema.

1) Llevar a cabo la bisección tetraedros por su arista más larga utilizando computación exacta.
2) Implementar la descomposición de tetraedros.
3) Construir árboles de bisección de tetraedros a partir de cualquier tetraedro.

A continuación se presenta el proceso de implementación del sistema en función de estos objetivos. Cabe destacar de que todas las funciones se encuentran debidamente precedidas por su firma y una descripción de su funcionamiento, y las más complejas tienen incluso comentarios pertinentes explicando los distintos pasos de su funcionamiento.

5.2 Bisección de Tetraedros

La bisección de tetraedros consta de encontrar la arista más larga de un tetraedro, calcular el punto medio de esa arista, y crear dos tetraedros utilizando la información de los vértices del tetraedro original y del punto medio encontrado. A continuación se describe la implementación de esta funcionalidad.

5.2.1. Aritmética de puntos

Para encontrar la arista más larga es necesario saber el largo de las aristas. Para esto se requiere una función que mida la distancia entre dos puntos: los vértices que definen una arista. Son por tanto dos funciones que se requieren en aritmética de puntos: cálculo de punto medio y de distancia comparable entre dos puntos.

Se implementan en el programa Point Arithmetics las funciones *squared_distance* y *middle_point*. La primera, *squared_distance*, utiliza el cuadrado de la fórmula de la distancia entre dos puntos para retornar el quotient con el resultado:

$$Distancia(A,B)=(B.x-A.x)^2+(B.y-A.x)^2+(B.z-A.z)^2$$

Mientras *middle_point* utiliza la fórmula de punto medio para retornar el Point_3 que produce:

$$Punto\,Medio(A,B)=((A.x+B.x)/2,(A.y+B.y)/2,(A.z+B.z)/2)$$

5.2.2. Estructura y primeros cálculos

La estructura que soporta un tetraedro en el sistema se llama *Tetra*, y se le implementa en su programa homónimo. Un *Tetra* corresponde a cuatro Point_3: A, B, C y D, que representan los vértices del tetraedro. Se crea también un constructor llamado *make_tetra* que recibe cuatro Point_3 y retorna un *Tetra* utilizando los puntos respectivamente en A, B, C y D.

Para llevar a cabo la bisección de un tetraedro por su arista más larga primero es necesario identificar cuales son las aristas más largas del tetraedro. Esta funcionalidad se divide en dos pasos: identificar la longitud al cuadrado de la arista más larga y determinar cuales aristas comparten esta medida.

El primer paso lo lleva a cabo la función *long_edge_measure*, la cual recibe un Tetra y utiliza la función *squared_distance* para calcular la distancia al cuadrado entre las distintas combinaciones de vértices. Luego las compara para determinar cuál es la mayor distancia y a continuación retorna su valor.

Para identificar las aristas que comparten esta medida se utiliza un arreglo de asignaciones de verdad. En el programa Tetra a cada arista se le asigna un índice dentro del arreglo para simplificar la lectura del programa, en forma de macro..

AB = 0; AC = 1; AD = 2;

BC = 3; BD = 4; CD = 5;

La función *get_long_edges* retorna uno de estos arreglos asignando a cada índice el valor 1 si la longitud al cuadrado de la arista correspondiente coincide con el largo al cuadrado de la arista más larga obtenida utilizando *long_edge_measure.* En caso contrario se asigna el valor 0.

Finalmente, una vez que el programa puede calcular las aristas a través de la cuales se debe llevar a cabo un corte, se implementa la función *cut*, la primera función del programa TetraCut, siguiendo la construcción de la sección 4.3. Esta función recibe un Tetra y un índice y retorna el par de tetraedros resultantes de su corte a través de la arista correspondiente al índice. En otras palabras retorna los dos tetraedros resultado de bisectar por el punto medio de la arista indicada al tetraedro ingresado. Para esto calcula el punto medio con la función *middle_point* y construye los dos nuevos tetraedros utilizando el punto medio calculado y los vértices del tetraedro original manteniendo la convención de construcción de tetraedros.

Utilizando *get_long_edges* y *cut* es posible realizar todas las bisecciones por la arista más larga de un tetraedro cualquiera.

5.3. Descomposición de Tetraedros

La estructura de datos que guarda una descomposición de tetraedro se llama *Decomposition* y cuenta con un arreglo de tetraedros de tamaño 8 y un índice que indica la cantidad de tetraedros que pueblan el arreglo. Sin embargo, para poder construir la descomposición de tetraedros es primero necesario implementar la semejanza para descartar los tetraedros semejantes a otros tetraedros incluidos en el análisis. Y como se discutió en la sección 4.4. se utiliza la función de calidad para ahorrar tiempo en el cálculo de semejanza.

5.3.1. Volumen

Calcular la calidad de un tetraedro, según se discutió consta de calcular el volumen y la longitud al cubo de la arista más larga del tetraedro.

Para calcular el volumen se utiliza la fórmula de volumen: $V = \frac{1}{6}\left|A \bullet (B \times C)\right|$

Sin embargo, esta función requiere que el vértice D se encuentre en el origen de coordenadas. Como los tetraedros que maneja TetraCut pueden ubicarse en cualquier parte del espacio tridimensional, lo que se hace es crear una función de traslación de puntos según un vector, y se utiliza el inverso aditivo de D para crear el vector y trasladar todos los puntos del tetraedro de manera de que se conserve su forma y D se encuentre en el origen. De esto se encarga la función *difference* que traslada un punto utilizando otro punto como vector calculando su diferencia. Esta se implementa en Point Arithmetics en conjunto con las otras dos funciones que requiere el volumen para su cálculo: el producto punto y el producto cruz.

Las funciones *dot_product* y *cross_product* utilizan las fórmulas clásicas:

$$ProductoPunto: A \cdot B = A_x B_x + A_y B_y + A_z B_z$$

$$ProductoCruz: A \times B = C$$

$$\leftrightarrow C_x = A_y B_z - A_z B_y$$

$$\wedge C_y = A_z B_x - A_x B_z$$

$$\wedge C_z = A_x B_y - A_y B_x$$

La función *dot_product* retorna un quotient con el valor del producto punto, y la función *cross_product* retorna un Point_3 con el resultado de su producto cruz.

La función *volume* traslada los puntos del tetraedro utilizando difference para fijar D al origen, y luego implementa la fórmula de volumen utilizando *dot_product y cross_product* para retornar un quotient con el volumen del tetraedro.

5.3.2. Calidad

Considerando que la longitud de arista que maneja el programa es al cuadrado y la fórmula lo requiere al cubo, lo que se calcula es la calidad al cuadrado de un tetraedro, de manera de no necesitar calcular la longitud de arista usando una raíz cuadrada. Para esto se utiliza la función *volume* y se eleva al cuadrado su valor, y se le divide por el cubo del resultado del *long_edge_measure*, obteniendo así la medida de calidad al cuadrado.

Como la calidad es un valor lento de calcular y que se utiliza en diversas ocasiones a través de la ejecución del programa se toma la decisión de incluir este resultado en la estructura *Tetra* en su forma de quotient. El constructor *make_tetra* se modifica para utilizar la función de calidad sobre cada *Tetra* que crea para agregar el valor a la estructura.

5.3.3. Semejanza

La actual implementación de semejanza utiliza la tercera conjetura de semejanza obtenida en la cual se comparan los sets de longitudes de aristas opuestas.

Para esto la función *similarity* recibe dos *Tetra*. Si ambos *Tetra* comparten calidad, genera la lista ordenada descrita en la sección 4.5. para ambos y utilizando *long_edge_measure* obtiene una constante de proporcionalidad entre ambos Tetras. Finalmente revisa si los valores homólogos de las listas sostienen la misma proporción y si es así retorna 1. En cualquier otro caso se retorna 0.

Una vez implementada *similarity* se tienen todas las herramientas requeridas para la descomposición de tetraedros. Sin embargo, por modularidad, se crea primero la función *is_similar_in_array*, la cual recibe un *Tetra* y un arreglo de *Tetras*. Esta función revisa si dentro del arreglo existe un *Tetra* similar al ingresado y retorna la posición del *Tetra* similar en el arreglo, o -1 en caso de no haberlo.

5.3.4. Descomposición

La función *decompose* en TetraCut recibe un *Tetra* y construye su *Decomposition* utilizando *get_long_edges* para identificar las posibles aristas de corte y poblando el arreglo *Decomposition* con los resultados de todos las posibles bisecciones por la arista más larga del *Tetra*. Antes de agregar un *Tetra* al arreglo, se utiliza *is_similar_in_array* para dar cuenta de si existe un *Tetra* similar en el arreglo. Si se encuentra que un tetraedro es similar a alguno de los *Tetra* del arreglo, se descarta. Finalmente la función retorna el *Decomposition*.

5.4 Árboles de Bisección

Para la construcción del árbol de bisección se planteó la estructura de nodo actualizado con el planteamiento.

```
Nodo:
    Tetraedro tet
    Bool terminal?
    Nodo[8] hijos
```

Sin embargo existen dos requerimientos sobre árbol de bisección que requieren una versión un poco más refinada de esta estructura: (1) construcción por niveles; (2) construcción minimal.

5.4.1. Construcción minimal

Para cumplir con el requerimiento de construcción minimal se utiliza un arreglo de *Tetra* para almacenar aquellos que ya han sido ingresados al árbol. El propósito de esto es poder revisar rápidamente utilizando *is_similar_in_array* si un *Tetra* debe ser agregado al árbol como nodo abierto o como terminal, y notarlo adecuadamente en la asignación de verdad en la estructura. Con esto se asegura de que si dos tetraedros son similares dentro del árbol, solo de uno sea calculado el subárbol.

5.4.2. Construcción por niveles

Normalmente en el desarrollo de árboles lo que se hace es construir en profundidad, ya que la recursión está dada naturalmente por la forma de los árboles y esta produce tal construcción. Sin embargo, considerando el gran tamaño que pueden tener tanto los números exactos utilizados en cada tetraedro como los árboles de bisección, se ha

estimado necesario para su análisis que sea construido por niveles. Para esto se utiliza un arreglo de punteros a los nodos que aún no han sido desarrollados, y en lugar de desarrollar cada rama naciente de un nodo recién creado, se agregan las ramas a dicho arreglo para luego ser retomado al desarrollar el siguiente nivel. Se plantea la estructura de árbol como sigue:

```
Árbol:
        Nodo raíz
        Arreglo Tetraedros_No_Similares;
        Arreglo Nodos_Pendientes;
        int niveles;
```

Se describe a continuación la construcción del árbol en detalle.

5.4.3. Construcción del Árbol de Bisección

La construcción del árbol de bisección se lleva acabo en el programa TetraCut y consta de tres funciones: (1) *init_tetra_tree*; (2) *develop_tetra_tree*; (3) *init_tetra_node*. La primera crea las condiciones iniciales de un árbol sin desarrollo, y llama a la segunda para desarrollarlo hasta un nivel arbitrario. La segunda utiliza a la tercera para inicializar cada uno de los tetraedros pendientes de desarrollo. La tercera se encarga de construir un nodo del tetraedro.

Un nodo del tetraedro se estructura de la siguiente manera:

```
Nodo:
        Tetraedro tet
        int ubicación
        Bool hoja
        Nodo[8] piezas
        int conteo_piezas
```

La función *init_tetra_node* recibe un nodo y el árbol al que pertenece. La función confía en que el nodo recibido tiene previamente inicializado su *Tetra*. Lo primero que hace es revisar si en el arreglo de tetraedros diferentes del árbol se encuentra un *Tetra* similar al del nodo utilizando *is_similar_in_array*. De ser así se marca que el nodo es terminal asignando verdadero al valor de verdad hoja, y guardando la ubicación en el arreglo en que fue encontrado el tetraedro similar. En caso contrario se guarda la última posición disponible del arreglo como ubicación, se ingresa el *Tetra* al arreglo de tetraedros diferentes del árbol, y se inicializan los *Tetra* de los nuevos nodos

utilizando *decompose* sobre el tetraedro del nodo, y se agregan los nodos formulados de esta manera a una nueva lista de nodos pendientes de desarrollo del árbol.

Por su parte, la función *develop_tetra_tree* recibe un árbol de bisección y toma cada nodo de la lista de nodos pendientes de desarrollo del árbol, cuyo *Tetra* ya se encuentra inicializado, y utiliza *init_tetra_node* en ellos para desarrollar un nivel del árbol. Además, crea una nueva lista de nodos pendientes de desarrollo para el siguiente nivel del árbol, y aumenta el conteo de niveles en 1.

Finalmente, *init_tetra_tree* recibe un árbol de bisección sin inicializar, un *Tetra* inicial y un entero indicando el máximo nivel a desarrollar. Lo que hace esta función es crear las condiciones iniciales del árbol:

- Crear el arreglo de *Tetras* diferentes vacío.
- Inicializar el *Tetra* del nodo raíz.
- Agregar el nodo raíz a la primera lista de nodos pendientes de desarrollo.
- Fijar el conteo de niveles a 0.

Finalmente entra en un ciclo de desarrollo de niveles utilizando la función *develop_tetra_tree* hasta que el árbol tenga el número de niveles explicitado.

Esto concluye la implementación del sistema requerido por este trabajo de memoria.

6. Resultados

Para el análisis de resultados fue creada una última función y modificadas ligeramente las funciones. Lo que se hace es implementar un ligero análisis acumulativo del árbol y un análisis por niveles. La idea es poder observar de manera estadística las distintas características del árbol de bisección, de manera de poder dilucidar sus propiedades. Además se comparan los resultados obtenidos con los resultados que entrega la bibliografía pertinente.

6.1. Herramientas de Análisis

El análisis acumulativo lo lleva a cabo la función *develop_tetra_tree* que inserta a la salida estándar un conteo de los tetraedros diferentes hasta el nivel desarrollado, la máxima y mínima calidad de los tetraedros contenidos y el nivel hasta el cual se ha desarrollado el árbol.

Además fue creada la función *analyse_tetra_floor* que recibe el arreglo de nodos pendientes y formula un histograma en el que se categorizan los distintos tetraedros asociados a estos nodos por su calidad. Para evitar obtener la raíz cuadrada de la calidad en el análisis lo que se hace es comparar los valores de calidad de cada uno de los tetraedros contra los límites de los intervalos analizados.

Por ejemplo, para obtener cuantos tetraedros tienen calidad entre 0.6 y 0.9 se revisa que la calidad al cuadrado de los tetraedros se encuentre dentro del intervalo [0.36, 0.81[cuyos límites corresponden a los cuadrados de 0.6 y 0.9.

Esto se lleva a cabo para los límites 0, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.6, 0.9 y 1. Por ejemplo, el histograma del nivel 50 del árbol de bisección del tetraedro equilátero se ve así:

```
TetraTree for Tetra:

(1,1,1), (0,0,1), (0,1,0), (1,0,0)
--------------------
Nivel 50.
Minimum squared quality: 0.00203548.
Maximum squared quality: 1.
Different tetra: 31559.

Analisis for floor 50:
  Analysed Tetra: 7050.
  Worst Tetra:
  (0.423571,0.551563,0.520046)
  (0.423569,0.551533,0.520039)
  (0.423574,0.551547,0.520045)
  (0.423588,0.551574,0.520034)
  Quality(squared): 0.0028535
  Best Tetra:
  (0.410784,0.779062,0.561253)
```

```
  (0.410796,0.779048,0.561249)
  (0.410801,0.779064,0.561256)
  (0.410792,0.779065,0.561241)
  Quality(squared): 0.563491

  Quality Distribution Histogram
Floor 50:
  0.00  -  0.05  : 0
  0.05  -  0.10  : 317
  0.10  -  0.15  : 1048
  0.15  -  0.20  : 1378
  0.20  -  0.30  : 2392
  0.30  -  0.40  : 1013
  0.40  -  0.60  : 860
  0.60  -  0.90  : 42
  0.90  - <1.00  : 0
  1.00  -  1.00  : 0
--------------------
```

Se puede observar en este resumen que el árbol de bisección de tetraedros que se han producido 31559 tetraedros semejantes distintos hasta este punto, y que de los 7050 que conforman el nivel por desarrollar, 2392 tienen una calidad entre 0.2 y 0.3. Además, se despliegan los vértices aproximados del tetraedro de mejor calidad obtenido cuya calidad al cuadrado es 0.563491 aproximadamente, lo que corresponde a una calidad de 0.750660. El programa maneja estos números de forma exacta, pero desplegarlos de forma exacta en la pantalla resultaría en un histograma ilegible para un ser humano por la cantidad de dígitos de estos números.

Observación: La combinatoria involucrada en el árbol de bisección para el tetraedro equilátero (de 6 aristas más largas) implica que el número de tetraedros distintos y número de niveles sea grande. Por ejemplo, para el nivel 50 se esperan 2^{50} tetraedros. Se muestra, sin embargo, que el análisis requiere revisar tan sólo los 31559 tetraedros semejantes distintos.

6.2. Pruebas de Comparación

Para demostrar el funcionamiento del programa se han utilizado distintos tetraedros cuyo análisis ha sido desarrollado a mano en distintos papers. Un ejemplo de esto es el paper de Antti Hannukainen, Sergey Korotov y Michal Křížek [13] sobre la bisección por la arista más larga de tetraedros. En él se explicitan unos cuantos ejemplos de tetraedros que cumplen ciertas propiedades al ser bisectados iterativamente.

Uno de ellos es el tetraedro definido por los vértices (0,0,0), (0,0,1), (0,1,1) y (1,1,1). Existe un error en el paper explicitando el segundo vértice como (1,0,0) y esto da origen a un cuadrado en el plano, pero al corregirlo se muestra la propiedad que deducen: este tetraedro produce un árbol de bisección con tan solo tres tetraedros en él. Congruente con este resultado, la construcción de este árbol de tetraedros produce el siguiente histograma para su tercer nivel en TetraCut:

```
TetraTree for Tetra:

(0,0,0), (0,0,1), (0,1,1), (1,1,1)

--------------------
Nivel 3.
Minimum squared quality: 0.0625.
Maximum squared quality: 0.125.
Different tetra: 3.

Analisis for floor 3:
  Analysed Tetra: 1.
  Worst Tetra:
  (1,0.5,0)
  (0.5,0.5,0.5)
  (0.5,0.5,0)
  (1,0,0)
  Quality(squared): 0.0740741
  Best Tetra:
  (0,0,0)
  (1,0,0)
  (0,1,1)
  (1,1,1)
  Quality(squared): 0.0740741

  Quality Distribution Histogram
Floor 3:
  0.00  -  0.05  : 0
  0.05  -  0.10  : 0
  0.10  -  0.15  : 0
  0.15  -  0.20  : 0
  0.20  -  0.30  : 1
  0.30  -  0.40  : 0
  0.40  -  0.60  : 0
  0.60  -  0.90  : 0
  0.90  - <1.00  : 0
  1.00  -  1.00  : 0
--------------------
```

El siguiente nivel del árbol de bisección solo vacía la lista de nodos pendientes y encuentra 0 resultados y detiene el desarrollo del árbol.

Otro ejemplo dentro del mismo paper corresponde al *Sommerville space-filler tetrahedron* con vértices (-1,0,0), (1,0,0), (0,-1,1) y (0,1,1), el cual según el paper produce un total de 4 tetraedros distintos. TetraCut también llega a este mismo resultado.

Finalmente, Andrew Adler [14] encuentra que si un tetraedro es casi-equilátero, y su segunda arista más larga es opuesta a la primera, se producirá un conjunto finito de particiones de tamaño no mayor a 37. Se especifica en el paper que casi-equilátero es una noción difícil de definir, pero que por ejemplo funciona si las aristas se encuentran dentro de un margen de longitud del 5%. Siguiendo esta lógica se armó el tetraedro

que cumple estas reglas utilizando los vértices (100,0,66), (-100,0,66), (0,105,-66) y (0,-105,-66). TetraCut, al recibir este tetraedro resuelve que se producen 13 tetraedros distintos al particionarlo, deteniendo el desarrollo del árbol en el nivel 9 al no encontrar más tetraedros distintos.

Se cree que TetraCut es capaz de encontrar un número finito de tetraedros a partir de la partición de cualquier tetraedro. Sin embargo, su consumo de recursos no permite que una máquina con 8 gb de RAM pueda almacenar una estructura de árbol de más de 54 niveles. Esto es, sin embargo, el nivel más avanzado que ha sido computado en trabajos de similar funcionamiento.

7. Conclusiones

A continuación se presentan las conclusiones respecto de este trabajo de memoria.

7.1. TetraCut

TetraCut es una herramienta que en su desarrollo considera distintas formas de ser un software eficiente en el uso de memoria, tiempo de computación y exactitud en el cálculo, pero dentro de todo es una herramienta que tiene ciertas limitaciones.

Muchos tetraedros en el estudio de esta área son formulados de acuerdo a las longitudes de sus aristas, lo que lleva al planteamiento de vértices que pueden no ser representables en forma racional. TetraCut no ofrece ninguna forma de representar dichos vértices, haciendo que el análisis de estos tetraedros esté fuera del alcance de este sistema.

Además, existen un par de estructuras dentro del sistema que no son óptimas en su consumo de recursos. Por ejemplo, los arreglos del árbol de tetraedros que almacenan los nodos pendientes y los tetraedros no-semejantes consumen memoria de forma no-óptima. Si fueran reemplazados por listas enlazadas que inicializan sus consumos de memoria dinámicamente, es posible que el análisis de árboles de bisección pudiese avanzar unos cuantos niveles más en su máximo.

Sin embargo, los resultados obtenidos por TetraCut son consistentes con la bibliografía pertinente, y prueba ser una herramienta útil en el estudio de la bisección por la arista más larga de tetraedros.

7.2. Planteamiento Teórico

Esta tesis contribuye además al desarrollo teórico de la geometría de tetraedros, cuyo estudio ha sido limitado. Se ha corroborado más de una conjetura teórica respecto de este problema, y se ha acotado la cantidad de tetraedros que se pueden producir en la descomposición de tetraedros. Sin embargo, quedan muchas preguntas por contestar como por ejemplo si es acotado el conjunto de tetraedros no-semejantes que se producen al bisectar iterativamente el tetraedro equilátero y el tetraedro esquina de cubo.

Otra problemática que queda descubierta es el comportamiento de los tetraedros de Adler. Puede que sea posible encontrar una correlación entre la cantidad de tetraedros obtenidos al bisectar iterativamente estos tetraedros y su forma. Puede que encontrar estos tetraedros al bisectar los tetraedros más complicados permita encontrar un camino de bisección en que se limite la cantidad de tetraedros obtenidos en los casos de mayor tamaño de conjunto de tetraedros no semejantes.

8. Bibliografía

[1] I.G. Rosenberg and F. Stenger, A lower bound on the angles of triangles constructed by bisecting the longest side, Math. Comput. 29, pp. 390-395, 1975.

[2] M. Stynes, On faster convergence of the bisection method for all triangles, Math. Comput. 35, pp. 1195-1201.

[3] M.-C. Rivara, Mesh refinement processes based on the generalized bisection of simplices, SIAM J. Numer. Anal, 21, pp. 604-613, 1984.

[4] M.-C. Rivara, Selective refinement/derefinement algorithms for sequences of nested triangulations. Int. J. Numer. Methods Eng. 28, pp. 2889-2906, 1989.

[5] M.-C. Rivara and C. Levin, A 3rd refinement algorithm suitable for adaptive and multi-grid techniques, Comm. Appl. Numer. Methods 8, pp. 281-290, 1992.

[6] Claudio Gutiérrez, Flavio Gutiérrez, María-Cecilia Rivara. Complexity of the bisection method. Theoretical Computer Science 382, pp. 131–138, 2007.

[7] Shamos, Michael Ian. Computational Geometry, Ph.D. dissertation, Yale University. 1978.

[8] Chee Yap, Thomas Dubé. The exact computation paradigm. Computing in Euclidean Geometry 2nd Edition, pp. 6-13, 1994

[9] The CGAL Project. CGAL User and Reference Manual. CGAL Editorial Board, 4.10 edition, 2017.

[10] Michael Hemmer and Susan Hert and Sylvain Pion and Stefan Schirra. Number Types. In CGAL User and Reference Manual. CGAL Editorial Board, 4.10 edition, 2017.

[11] Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee Yap. The CORE Library Project, 1.2 edition, 1999.

[12] I. Sharygin. Problems in Solid Geometry, Volume 1 of Science for everyone. Mir, 1986.

[13] Antti Hannukainen, Sergey Korotov, Michal Křížek. On numerical regularity of the face-to-face longest-edge bisection algorithm for tetrahedral partitions, Science of Computer Programming 90, pp. 34-41, 2014.

[14] Andrew Adler. On the Bisection Method for Triangles, Mathematics of Computation, Vol. 40, No. 162, pp. 571-574, 1983.