UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de segmentación automático de modelos 3D para la fabricación de esculturas de gran formato

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pablo Sánchez Jiménez

Tutor: Miguel Sánchez López

Curso 2015/2016

Resumen

La fabricación de esculturas de gran formato a partir de modelos 3D en la actualidad se facilita mediante el uso de software y maquinaria especializada. A pesar de esta ayuda, los artistas que se encargan de ello siguen teniendo que hacer gran parte del trabajo a mano, ya que las grandes esculturas no se pueden construir en una sola pieza. Para ello utilizan técnicas de segmentación teniendo en cuenta diferentes limitaciones. Este proyecto se centra en el desarrollo de tecnologías que permitan automatizar y facilitar el proceso de segmentación, permitiendo así ahorrar tiempo y trabajo a los artistas.

Palabras clave: 3D, escultura, segmentación, fabricación, automatización.

Abstract

Nowadays, manufacturing large scale sculptures from 3D models is a process that's been made easier with the use of specialized software and machinery. Still, artists involved have a large amount of manual labour to do in order to produce these large sculptures, since they cannot build them in a single piece. In order to solve this, they use segmentation techniques taking into account several limitations. This project approaches the problem by developing technologies that allow to automate the segmentation process, making the artists job easier and quicker.

Keywords: 3D, sculpture, segmentation, manufacturing, automation.

Tabla de contenidos

1. Motivación y objetivos del trabajo 7
1.1 Motivación 7
1.2 Objetivos 11
1.3 Estructura de la memoria 11
2. Estado del arte 12
3. Desarrollo del proyecto 14
3.1 Visión general 14
3.2 Tecnologías empleadas 16
3.3 Estado inicial 17
3.4 Conceptos de mallas 19
3.5 Fundamentos Técnicos 20
3.6 Implementación y proceso 22
4. Análisis y resultados 24
4.1 Figura Sencilla (cactus) 24
4.2 Figura Sencilla (Pieza ajedrez) 25
4.3 Figura compleja (Elefante) 26
5. Conclusiones 28
Bibliografía 30
Anexo 1: Algoritmo de segmentación 31
Anexo 2: Conversión Off 33

1. Motivación y objetivos del trabajo

1.1 Motivación

En los últimos años, las tecnologías que se abordan en este proyecto han comenzado a tener mayor relevancia a la hora de la construcción de una de las tradiciones valencianas: las Fallas.

Entre estas tecnologías para la creación de piezas, este trabajo se centra principalmente en el fresado con maquinaria de tres ejes y la impresión 3D.

Una fresadora es una máquina herramienta, compuesta por un cabezal giratorio con una fresa y una mesa, también móvil, donde se fija el material que se quiere fresar.

Figura 1.1: Fresadora CNC de tres ejes (para piezas de aprox. 40x30cm)

El proceso de fresado consiste en, partiendo de un bloque de un determinado material, ir eliminando o esculpiendo el bloque hasta conseguir la pieza deseada. El fresado en tres ejes supone algunas limitaciones a la hora de trabajar con determinadas piezas, haciendo incluso que algunas sean imposibles de mecanizar directamente, debido a la naturaleza de su movimiento.
En la figura 1.2 se ilustra un ejemplo de una pieza que no se puede mecanizar directamente con una fresadora de tres ejes, ya que tiene un agujero por debajo y la máquina no tiene manera de acceder a esa parte.

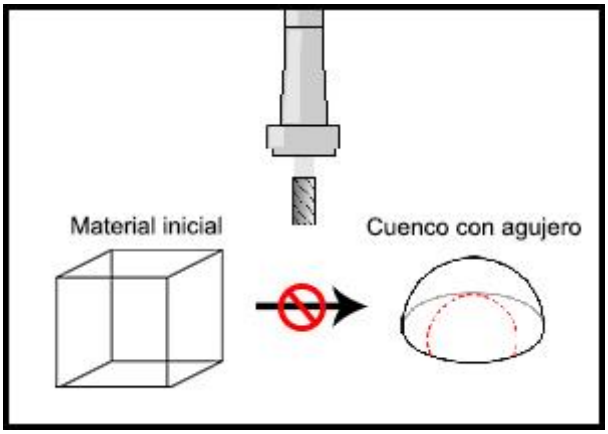


Figura 1.2: Mecanizado imposible en tres ejes

Con las fresadoras, se facilita el proceso mediante la rotación y/o segmentación de las piezas cuyo mecanizado no es posible inicialmente.

Una impresora 3D es una máquina herramienta, compuesta por una boquilla capaz de moverse en dos ejes con la ayuda de algunos motores y una mesa de impresión también móvil en un eje, proporcionando movimiento en tres ejes.

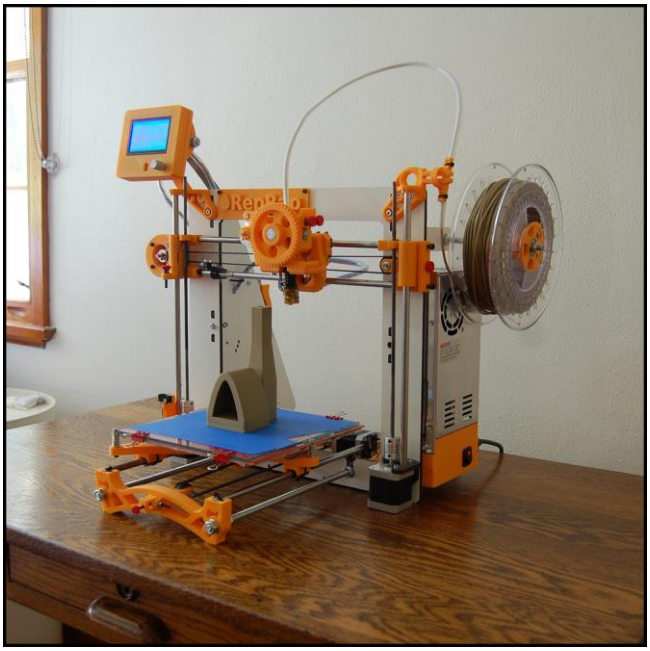


Figura 1.3: Impresora 3D (imagen de tomlauerman.com)

Algunas impresoras tienen también un sistema para calentar la mesa de impresión, mejorando la fijación de algunos materiales.
A su vez, la boquilla tiene una entrada para el material y una zona donde se calienta hasta reblandecerlo.
Con ayuda de un motor, el material se va empujando por la boquilla, que al fundirlo va dejando caer un filamento con el que se va construyendo la pieza.

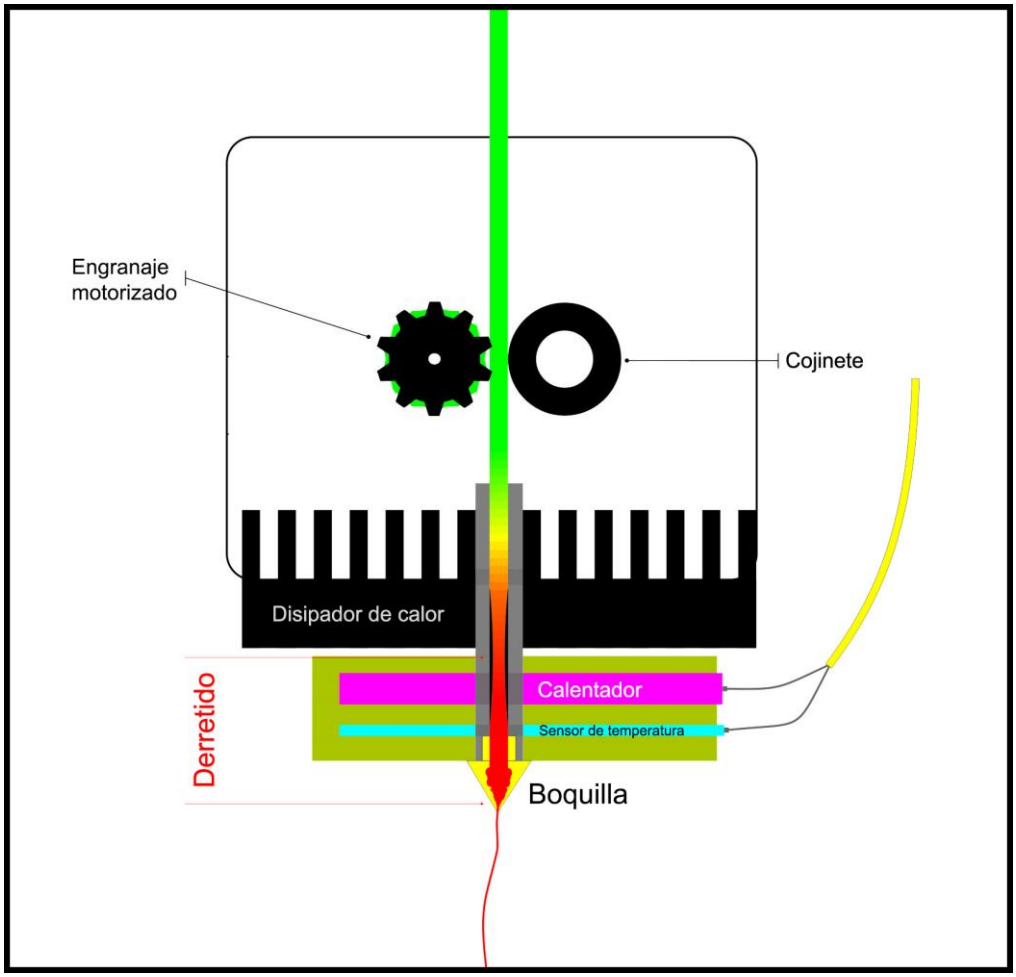


Figura 1.4: Funcionamiento de una boquilla (imagen de 3dprintingsystems.freshdesk.com)

Las impresoras 3D generan sus piezas añadiendo material, en lugar de eliminándolo como es el caso con las fresadoras. En este caso, las piezas se generan dejando finas capas de material una encima de la anterior, construyendo así la figura deseada.

El problema surge cuando determinadas figuras, dada su geometría, requieren un uso adicional de material y tiempo para construir lo que se conoce como estructuras de soporte, cuyo objetivo es el de sujetar aquellas partes de la pieza que la desestabilizarían durante la construcción de la misma.

Estas estructuras, deben ser posteriormente eliminadas a mano por un técnico, lo cual supone aún más tiempo y trabajo. En la figura 1.5 se ilustra un ejemplo de este problema.

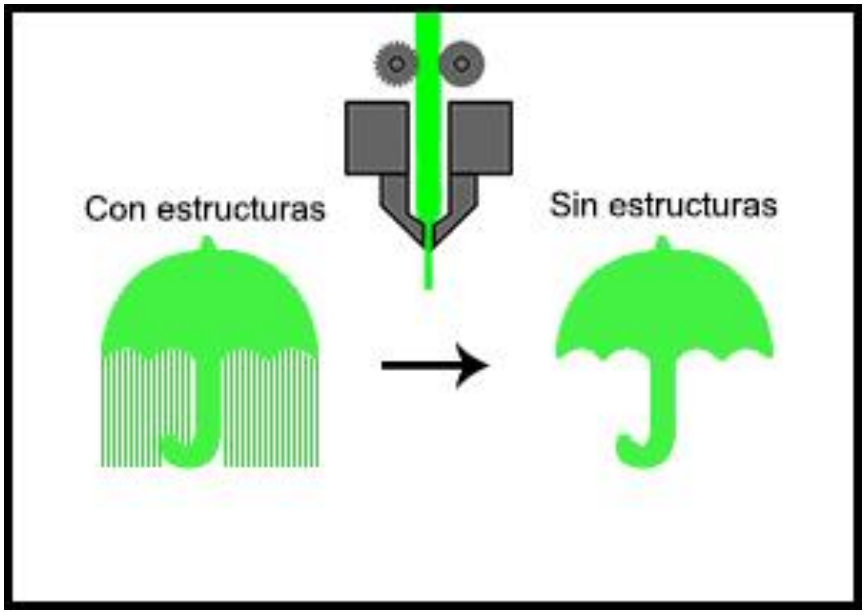


Figura 1.5: Eliminación de las estructuras de soporte

En las impresoras 3D, el proceso es similar.
Al simplificar las piezas en otras más pequeñas, ahorramos material al no tener que utilizar estructuras de soporte que serían necesarias en piezas más complejas.

Estos procesos tienen también una limitación que quizá no sea evidente a primera vista: el tamaño de la pieza a mecanizar.

Las máquinas utilizadas en la construcción de las piezas son de un tamaño limitado, lo que limita el tamaño de las piezas que se pueden mecanizar.
Por ello, es importante ser capaces de segmentar grandes piezas en bloques más pequeños para permitir el mecanizado por motivos diferentes a la geometría de la pieza.

1.2 Objetivos

El objetivo principal de este proyecto es lograr segmentaciones que puedan resultar interesantes a la hora de mecanizar un modelo no mecanizable directamente.

Este objetivo se organiza en diferentes fases, para simplificar su estructura:

- Analizar el modelo a construir y extraer los datos relevantes para realizar la segmentación.
- Utilizando los datos analizados, realizar el proceso de segmentación de la pieza.
- Obtener un modelo segmentado que permita o facilite el mecanizado.

1.3 Estructura de la memoria

En el capítulo 1 se expone la relevancia del tema tratado en el proyecto y la motivación para llevarlo a cabo. También se exponen los objetivos que queremos conseguir durante el desarrollo.

En el capítulo 2 se van a detallar métodos existentes relacionados con el problema tratado en este proyecto.

En el capítulo 3 se especifica todo aquello que se ha hecho durante el desarrollo del proyecto.
Se explican con detalle los conceptos básicos necesarios para comprender el proyecto, así como los aspectos más técnicos relacionados con la implementación del algoritmo.

En el capítulo 4 se analizan y contrastan diferentes resultados, ayudando a comprender mejor el trabajo realizado.

En el capítulo 5 se cierra el ciclo del proyecto, presentando las conclusiones tanto a nivel del proyecto, como a nivel personal.

En el capítulo 6 se enumeran las diferentes fuentes bibliográficas que se han empleado en este proyecto.

2. Estado del arte

Inicialmente, una falla se construía acoplando cabezas y manos de cera a estructuras de madera y recubriéndolas con ropajes.

Alrededor de 1956, se comenzaron a construir las figuras íntegramente con cartón.
Para ello, se utilizaban moldes de escayola, que posteriormente pasaron a ser de poliéster y más adelante de fibra de vidrio.
Los problemas que presentan estos moldes es que si se quieren construir dos figuras similares pero no iguales, se necesitan dos moldes diferentes, además de que los talleres de los artistas falleros terminan acumulando una gran cantidad de moldes, perdiendo mucho espacio útil.

Figura 2.1: Taller lleno de moldes (imagen de Rubén Tortosa)

A partir de los años 80, los métodos cambiaron con la introducción de la pintura acrílica, nuevas colas y pastas comerciales.

Sin embargo, el auténtico cambio se produjo con la introducción del poliestireno expandido, las máquinas de corte por hilo caliente y CNC (control numérico computerizado).

Las máquinas de corte por hilo caliente se utilizan para cortar poliestireno y materiales similares.
Se compone de un cable de metal que se calienta con una resistencia eléctrica hasta aproximadamente los 200ºC.

A medida que el cable pasa a través del material a cortar, el calor del cable vaporiza el material a su paso.

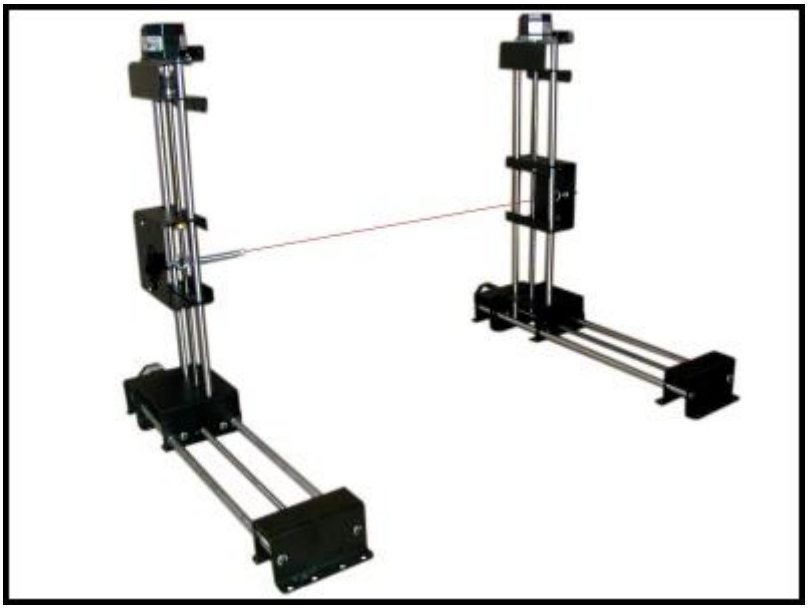


Figura 2.2: Máquina de corte por hilo caliente (imagen de foamlinx.com)

Por otra parte, el control número computerizado es un sistema de automatización de máquinas herramienta que permite operarlas mediante comandos programados en lugar de los mandos manuales.
Se usa un sistema de coordenadas que especifica el movimiento de la herramienta.
En el caso de las fresadoras e impresoras 3D, se controlan los movimientos en tres ejes.

Figura 2.3: Máquina CNC de corte por hilo caliente

Estos cambios hacen que el proceso sea mucho más eficiente y sencillo para los artistas falleros, aunque el poliestireno expandido produce una serie de elementos en la combustión que son poco respetuosos con el medio ambiente.

3. Desarrollo del proyecto

En este capítulo se describe el proceso seguido para el desarrollo del proyecto.

3.1 Visión general

En este proyecto, se trata de resolver un problema complejo para el que ya existen diversas posibles soluciones propuestas.

Como primera aproximación al problema, se han estudiado algunas de esas soluciones para familiarizarse con el problema.

La primera solución estudiada se trata de un artículo publicado por un grupo de tres universidades de Canadá, China e Israel. [1]
En este artículo, se habla sobre la descomposición piramidal de objetos.
Para que un objeto sea piramidal, debe tener una base plana y para cualquier punto P dentro del objeto, el segmento formado por el punto P y su proyección ortogonal P' en la base debe estar dentro del objeto en su totalidad.

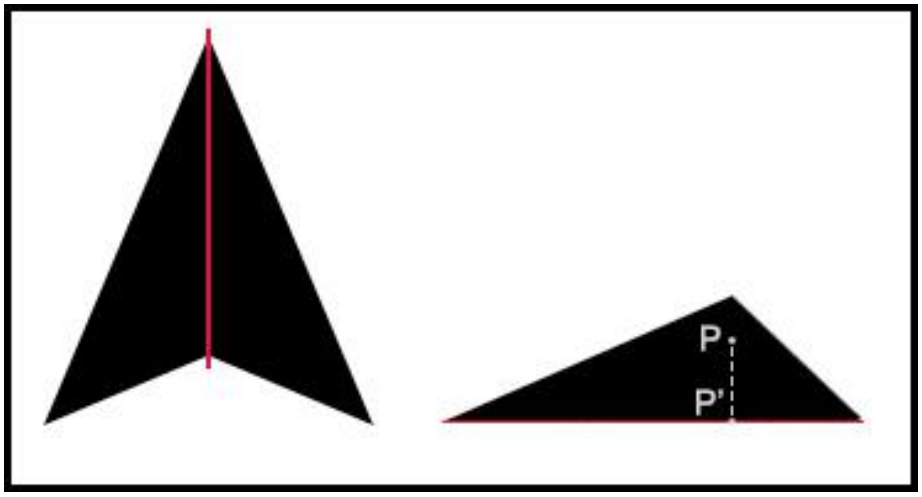


Figura 3.1: Ejemplo de descomposición piramidal

Como podemos observar en la figura 3.1, partimos de una pieza que no cumple los criterios de piramidalidad y por lo tanto, no es directamente mecanizable. Sin embargo, descomponiendo la pieza en dos mitades que sí lo cumplen, tenemos como resultado una descomposición que sería directamente mecanizable.

Esta solución resulta interesante ya que cualquier pieza piramidal es directamente mecanizable, aunque sus autores no han querido publicar su implementación, lo cual dificulta el uso de esta solución en otros trabajos.

La segunda solución estudiada es una librería de algoritmos geométricos que lleva en construcción desde el año 1995. [2]
Esta librería existente bajo una licencia de código abierto contiene implementaciones útiles para solucionar una gran variedad de problemas.
Entre las implementaciones disponibles, podemos encontrar una implementación para segmentación de mallas tridimensionales, introducida en el año 2014. [3]
Esta implementación se basa en una función conocida como Shape-Diameter Function o SDF, cuyo funcionamiento se explicará más adelante.

Esta solución, a pesar de no garantizar piezas directamente mecanizables como resultado, es más interesante que la piramidal por dos motivos.
Por una parte, las figuras con las que se suele tratar en la construcción de las fallas suelen ser más órganicas que geométricas.
Por el otro lado, esta librería proporciona implementaciones abiertas y modificables, lo cual permite su uso en proyectos paralelos a la librería.

El problema a solucionar en este proyecto es un problema NP-duro, como se comenta en el artículo de la descomposición piramidal.
Dada su complejidad, deberemos encontrar soluciones en las que se alcance un compromiso entre el detalle de la pieza final y el número de trozos.

El compromiso a alcanzar se debe a que a pesar de que una pieza se puede segmentar en todas las piezas que se quiera, éstas deben ser posteriormente ensambladas en una figura completa, además de que cada pieza fresada conlleva un cierto desperdicio de material, con lo que una mayor cantidad de piezas supone mayor trabajo y material desechado.

Por lo tanto, a la hora de segmentar un modelo, no nos interesa ser demasiado estrictos con las piezas seleccionadas ya que podríamos acabar con segmentos demasiado pequeños que no serían convenientes para cumplir nuestros objetivos.

Dada la complejidad del problema a tratar, es posible que los resultados obtenidos no cumplan con los requisitos que se buscan.

Sin embargo, cualquier resultado obtenido, ya sea positivo o negativo, nos ayudará a conservar o descartar ciertas líneas de investigación para futuros proyectos.

3.2 Tecnologías empleadas

3.2.1 CGAL

La implementación de la solución se ha realizado utilizando la librería de código abierto *CGAL (Computational Geometry Algorithms Library).*
Esta librería en C++ proporciona muchas facilidades y estructuras de datos preparadas para la implementación de algoritmos de computación geométrica.

3.2.2 OFF

Para la definición de los modelos 3D utilizamos el formato *OFF (Object File Format).*
Este formato contiene una cabecera OFF, seguido de una línea que especifica el número de vértices, el número de caras y el número de aristas.
A continuación, contiene una lista de vértices y una lista de caras.

Cada vértice se define en una línea mediante sus coordenadas en el espacio.
Las caras, también en una línea cada una, se definen como el número de vértices que las componen seguido de la enumeración de los mismos.

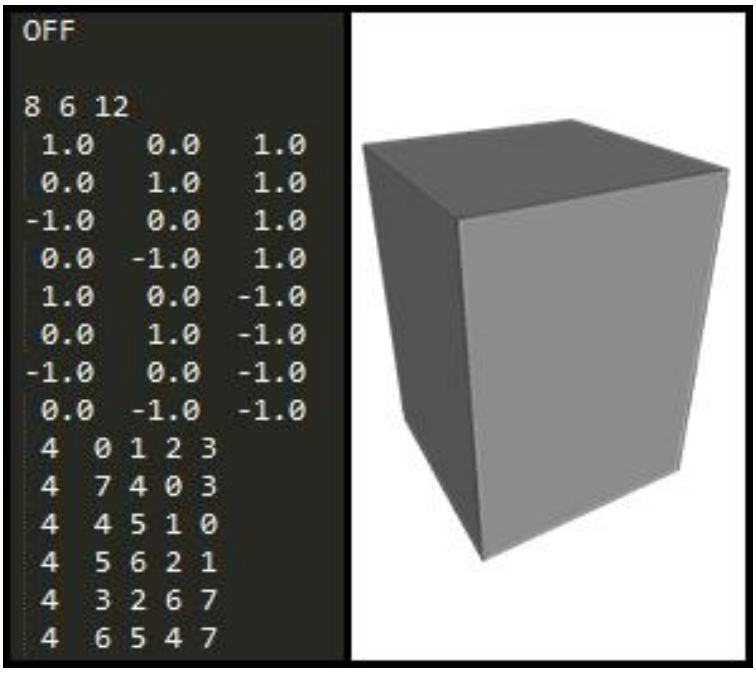


Figura 3.2: Fichero OFF y su pieza correspondiente

En la figura 3.2 encontramos una pieza formada por 8 vértices y 6 caras, cada una de las cuales está formada por 4 vértices.

3.2.3 Meshlab

Meshlab es un software de código abierto que permite el procesamiento y edición de mallas 3D.
Proporciona una serie de herramientas para la edición, limpieza, corrección, inspección, renderización y conversión de este tipo de mallas.

Para este proyecto, se han utilizado principalmente dos herramientas: la conversión de formatos y el coloreo por mallas.

Por un lado, la conversión de formatos nos ha permitido utilizar algunas mallas encontradas en Internet que inicialmente eran de otro formato, como por ejemplo STL, que es uno de los formatos más populares en el diseño asistido por computadora.

Por otra parte, el coloreo por mallas nos permite diferenciar el resultado del proceso de segmentación de la malla original.

3.2.4 C++

El lenguaje de programación C++ surgió con la intención de añadir al lenguaje C mecanismos que permitiesen la manipulación de objetos.

Se trata de un lenguaje multiparadigma, ya que combina el paradigma de orientación a objetos y el imperativo.

Se ha utilizado este lenguaje ya que la librería de algoritmos gráficos previamente mencionada, se encuentra programada en este lenguaje.

3.2.5 AWK

AWK es un lenguaje de programación para procesar textos.
Funciona mediante ficheros de órdenes, que especifican cómo procesar la entrada.

3.3 Estado inicial

En este apartado se van a detallar los conceptos básicos relacionados con el problema, que permitirán comprenderlo a la hora de idear una solución.

Actualmente, el proceso de despiezado de los modelos 3D se realiza de forma artesanal, siendo el artista el que elige como realizar las diferentes piezas para

que el proceso de fresado sea lo más económico y con el mejor acabado posible. Es importante tener en cuenta que se suele trabajar con planchas de corcho de tamaños determinados, y se debe tratar de distribuir la descomposición de las piezas de forma que se desperdicie la menor cantidad posible de material.

Un número elevado de piezas, permite que la figura tenga mucho detalle, mientras que un número pequeño hace que el detalle sea menor y haya que trabajar más en el ensamblado.

Por estos motivos, hay que encontrar un compromiso entre el material desechado y el nivel de detalle deseado.

Es importante conocer también un detalle técnico para poder idear una solución: lo que conocemos como un oscuro.

Un oscuro es una zona a la que la máquina no tiene acceso ya que su normal está a un ángulo mayor de 90º respecto al eje de mecanizado.
El eje de mecanizado es aquel por el que la máquina herramienta accede al material para modelar la pieza.

Figura 3.3: Ejemplo de oscuros

Una manera de convertir la pieza de la figura 3.3 en una pieza mecanizable es mediante su segmentación.

Figura 3.4: Segmentación mecanizable

Como se ilustra en la figura 3.4, si segmentamos la pieza por la mitad en dos piezas diferentes y las orientamos adecuadamente, pasaría a ser una pieza mecanizable.

3.4 Conceptos de mallas

Antes de indagar en el funcionamiento del algoritmo a nivel técnico, es necesario aclarar algunos conceptos sobre lo que es una malla 3D y su funcionamiento.

Una malla es una colección de vértices, caras y aristas que definen la forma de un objeto poliédrico.
Las caras normalmente se componen de triángulos, cuadriláteros u otros polígonos.
Para este proyecto, se han utilizado triángulos en todo momento.

En la figura que aparece a continuación, podemos observar un ejemplo de una malla 3D.

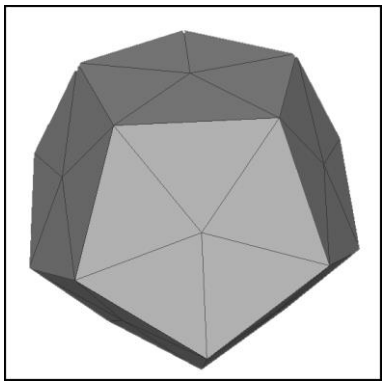


Figura 3.5: Dodecahedro

Como se puede apreciar, esta figura tiene 5 triángulos en cada una de sus 12 caras, por lo que la figura se describe con un total de 60 triángulos.

3.5 Fundamentos Técnicos

En esta sección, se van a detallar las técnicas empleadas en la implementación del algortimo de segmentación de modelos.

3.5.1 Shape-diameter function

Esta función, que abreviaremos como SDF [4], es la base del algoritmo de segmentación.

La tarea de esta función es proporcionar una estimación del diámetro del objeto local en cada triángulo de la malla (valores SDF).
Para cada triángulo, se muestrean varios rayos formando un cono utilizando su centroide (el punto donde intersectan las medianas del triángulo) como cumbre y la normal interior como eje.

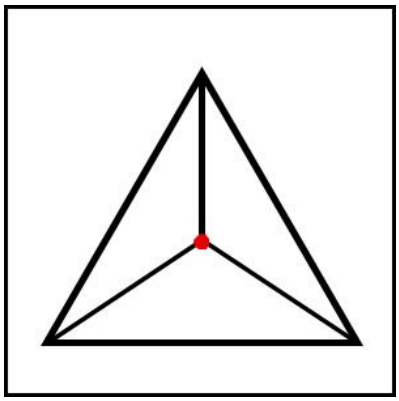


Figura 3.6: Centroide de un triángulo

Cada rayo se convierte en un segmento cuyos puntos son la cumbre del cono y la primera intersección con la malla.

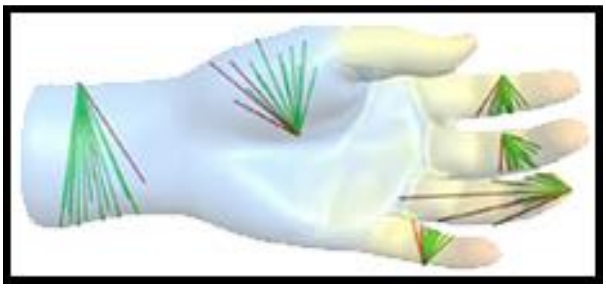


Figura 3.7: ejemplos de SDF

Una vez trazados varios segmentos, el valor SDF se calcula aplicando primero un proceso de eliminación de valores extremos y tomando la media de las longitudes.

Después de calcular el valor SDF para cada triángulo, se deben procesar antes de poder utilizarlos en el algoritmo de segmentación.

Aquellos triángulos que no tengan ningún valor SDF recibirán la media de los valores de los vecinos con los que comparta alguna arista.
Si aún así no tuviese un valor un SDF se le daría el mínimo de todos los valores SDF que se han asignado.

Tras esto, los valores SDF son linealmente normalizados entre 0 y 1.

A continuación, se realizan dos fases de agrupamiento.

En el primer agrupamiento, se define un número de grupos como parámetro. Este número es una estimación sobre cuántos grupos creemos que formarán el modelo de entrada.
Una vez definido, se tratará de ajustar los valores SDF con tantas distribuciones Gaussianas como grupos hayamos estimado.
Como resultado de este proceso, obtendremos un vector de probabilidades para cada valor SDF, indicando a que grupo es más probable que pertenezca.
Es importante saber que durante esta fase de agrupamiento sólo se tienen en cuenta los valores SDF y no las relaciones de vecindad entre triángulos.

En el segundo agrupamiento, utilizamos como entrada los vectores de probabilidad generados durante el primer agrupamiento.
Sin embargo, además de tener en cuenta los resultados previos, en esta fase se va a tener en cuenta la geometría del modelo.

Para ello se minimiza una función de energía mediante un algoritmo de corte de grafos [5, 6].

$$E(\bar{x}) = \sum_{f \in F} e_1(f, x_f) + \lambda \sum_{\{f,g\} \in N} e_2(x_f, x_g)$$

El primer término de la función aporta las probabilidades cálculadas en el primer agrupamiento.

El segundo término es un criterio geométrico cuyo valor es mayor cuando dos triángulos adyacentes comparten una arista, formando un ángulo muy cerrado.

Se incluye también en la función un valor de suavizado (λ) que hace que el criterio geométrico tenga mayor o menor peso a la hora de agrupar.

Finalmente, se asigna a cada triángulo el ID del grupo al que pertenece y se forman los diferentes segmentos.
Cada segmento es un *set* de triángulos conectados que pertenecen a un mismo grupo.

3.6 Implementación y proceso

3.6.1 Instalación de CGAL

Antes de comenzar con la implementación, ha sido necesario realizar un proceso de instalación de la librería CGAL y sus dependencias.

Este proyecto se ha desarrollado utilizando un sistema Linux, por lo que la instalación de CGAL resulta muy sencilla mediante el siguiente comando:

sudo apt-get install libcgal-dev

Sin embargo, es importante asegurarse de disponer además de algunas librerías de terceros que no son proporcionadas en el paquete de CGAL.

Esta librería requiere dos librerías esenciales, que son la Standard Template Library y las librerías Boost.
Para el uso de algunas demos proporcionadas por CGAL, también es necesario disponer de OpenGL y Qt5.

OpenGL proporciona una API para aplicaciones que utilizan gráficos en 2D y 3D, mientras que Qt5 se encarga de proporcionar un entorno multiplataforma para la gestión de interfaces de usuario.

3.6.2 Algoritmo de segmentación

El primer paso que realiza el algoritmo es leer la malla de entrada, siempre que ésta sea válida.
Este paso se realiza mediante una función proporcionada por la librería CGAL, que se encarga de convertir la malla leída a un objeto Polyhedron, que facilita la manipulación de la misma.

A continuación, se crean dos mapas de propiedades: uno para los valores SDF y otro para los identificadores de segmento.

Una vez creados, se calculan los valores SDF y se envían a la función de segmentación, que almacena los resultados en el mapa de identificadores de segmentos.

Tras este proceso, disponemos de todos los triángulos clasificados por segmento, con lo que el siguiente paso del algoritmo es iterar por todos los segmentos y seleccionar todos los triángulos que pertenezcan a cada uno.
Mediante este proceso, se generan diferentes mallas para cada segmento, creando un modelo final compuesto por todos los diferentes segmentos.

Es importante mencionar que existen dos parámetros que pueden alterar el resultado de la segmentación: el número estimado de grupos y el parámetro de suavizado.

3.6.3 Compilación

A la hora de compilar programas desarrollados con CGAL, es necesario documentarse de los pasos a seguir, ya que la lista de dependencias necesarias es muy grande.

Para este fin, CGAL nos facilita un script que se encarga de crear un archivo que contiene toda la información necesaria para poder compilar nuestro programa de manera sencilla.
Mediante su ejecución, se genera un archivo CMakeLists.txt.

Posteriormente, utilizando la herramienta *cmake*, generamos un archivo Makefile, cuyo objetivo es realizar el paso final de organización de la compilación del programa.

Finalmente, usando la herramienta *make*, compilamos nuestro programa sin ningún problema.

La compilación mediante compiladores habituales para C++ como *gcc* o *g++* resulta mucho más engorrosa y complicada si no se utiliza el script facilitado por CGAL, ya que se debe mantener un seguimiento de todas las dependencias de nuestro programa, para más adelante ser capaces de gestionar su inclusión.

4. Análisis y resultados

4.1 Figura Sencilla (cactus)

Como primer ejemplo, hemos utilizado una malla bastante sencilla con la forma de un cactus compuesto por un cilindro central con dos ramas.

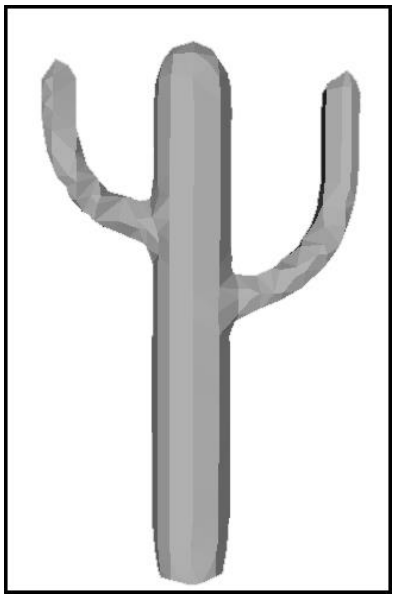


Figura 4.1: Modelo inicial

Tras la segmentación, el cactus se ha separado en tres partes: el cilindro central, la rama izquierda y la rama derecha.

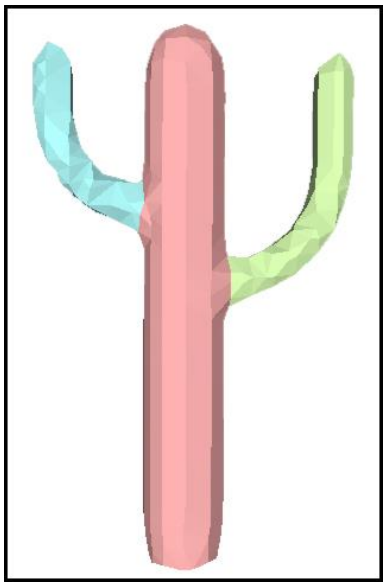


Figura 4.2: Modelo segmentado

Dada la sencillez de esta figura, no se han modificado los parámetros para obtener diferentes resultados.

4.2 Figura Sencilla (Pieza ajedrez)

A continuación, hemos utilizado otra malla sencilla que consiste en un rey de un juego de ajedrez.

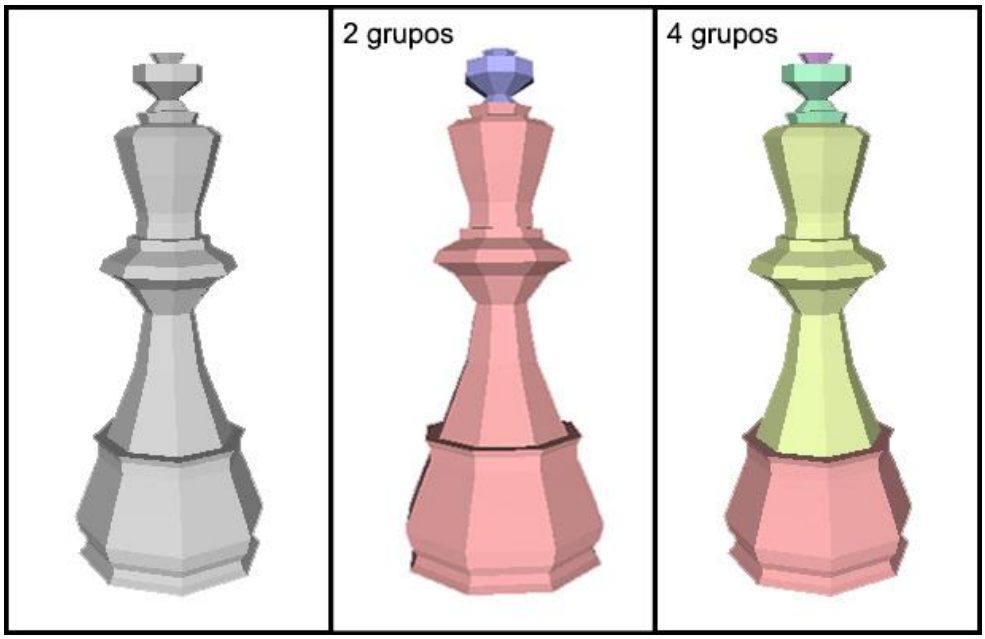


Figura 4.3: Comparativa

Se ha realizado una primera segmentación estimando 2 grupos, cómo se puede observar en la parte central de la figura 4.3.

La segunda segmentación, realizada estimando 4 grupos, se puede observar en el lado derecho de la figura 4.3.

Con esta pieza podemos observar claramente el efecto que tiene la modificación del número de grupos sobre el resultado de la segmentación.

Sin embargo, es importante mencionar que a pesar de lo que pueda parecer, no existe una relación directa entre el número de grupos y el número final de segmentos.
El número de grupos representa el número de niveles de una segmentación basada en agrupar triángulos cuyos valores SDF sean similares, sin tener en cuenta su vecindad.

4.3 Figura compleja (Elefante)

Como último ejemplo, vamos a utilizar una figura mucho más compleja: un elefante.

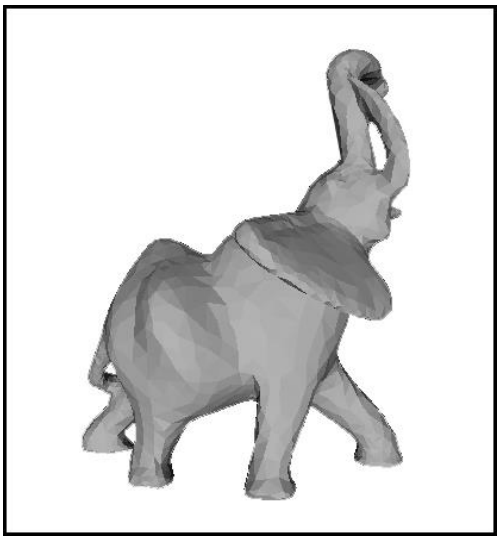


Figura 4.4: Modelo inicial

Como se puede observar, este modelo incrementa la complejidad considerablemente con respecto a los anteriores.

Se han realizado tres pruebas con diferentes valores de suavizado para obtener diferentes segmentaciones y ver cómo la variación del valor afecta al resultado.

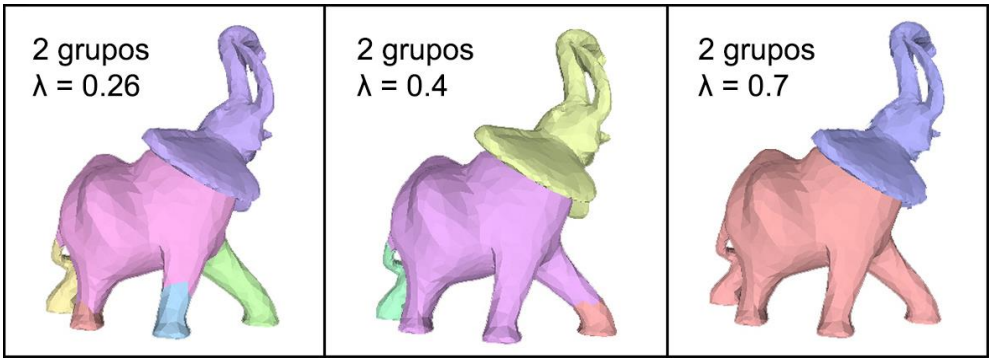


Figura 4.5: Modelo segmentado con diferentes valores de suavizado

Con esta pieza en lugar de modificar el número de grupos, hemos modificado el valor de suavizado, cuya función es dar mayor peso a la geometría del modelo a la hora de realizar la segmentación.

A continuación, se han realizado tres pruebas con diferentes números de grupos pero manteniendo un mismo valor de suavizado.

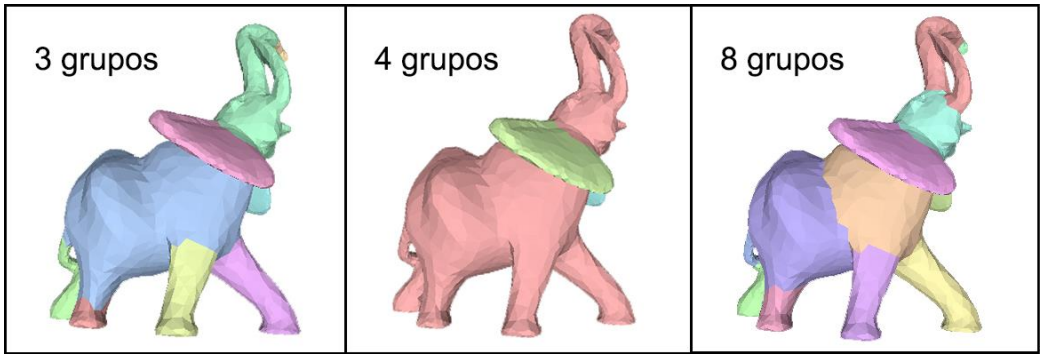


Figura 4.6: Modelo segmentado con diferente número de grupos

Como resultado podemos ver que con 3 grupos se ha segmentado en 9 piezas.
Con 4 grupos, la segmentación ha resultado en 3 piezas.
Finalmente, con 8 grupos, la segmentación contiene 12 piezas.

Por norma general, un mayor número de grupos nos dará como resultado una segmentación con más piezas.
Sin embargo, si observamos la segmentación en 4 grupos de la figura 4.6, podemos observar que tiene considerablemente menos piezas que la de 3 grupos.

Esto se debe a que en el proceso se tiene en cuenta en mayor o menor medida, dependiendo del valor de suavizado, la geometría de la pieza.

5. Conclusiones

Tras los análisis realizados podemos observar que la modificación de cualquiera de los dos parámetros altera el resultado de diferentes maneras.

Por lo tanto, para cada figura se debería encontrar el equilibrio ideal entre ambos parámetros que realizase una segmentación interesante.

Este método de segmentación puede ser útil a la hora de ayudar a los artistas a realizar una primera partición de las mallas.
Sin embargo, el resultado del algoritmo implementado no necesariamente hace que las segmentaciones realizadas sean directamente mecanizables, por lo que el artista o técnico debería realizar un proceso para permitir el mecanizado de determinadas piezas.

Con respecto a los objetivos planteados al inicio del proyecto, se han cumplido en gran parte.
Se ha podido extraer de un modelo inicial toda aquella información relevante al problema y se ha podido procesar de forma que se consiga segmentar un modelo en diferentes piezas.

Por el contrario, el objetivo de proporcionar piezas directamente mecanizables como salida del algoritmo no se ha logrado cumplir, dada la complejidad del problema.
Sin embargo, se podría extender la funcionalidad de este proyecto a futuros trabajos de la universidad, empleando los resultados de este proyecto como punto de partida.

También cabe mencionar que, a pesar de no haber resuelto completamente el problema del mecanizado, se plantea una observación que podría ser interesante para muchas de nuestras piezas.
Una vez realizada la segmentación, muchas de las piezas resultado que no sean mecanizables pueden cortarse por la mitad y, utilizando el plano de corte como base para cada una de las piezas resultantes, éstas podrían convertirse en piezas mecanizables. Aunque esta observación no ha sido verificada con suficientes ejemplos como para asegurar que sea una estrategia válida, a continuación hay un ejemplo visual de esta estrategia, utilizando un segmento de la figura 4.3.

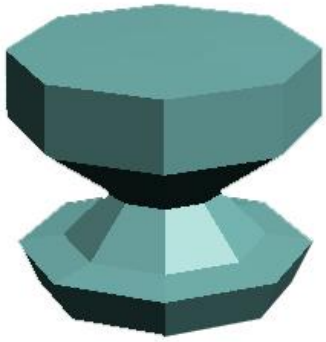


Figura 5.1: Figura completa

Figura 5.2: Media figura

Como podemos observar en las figuras 5.1 y 5.2, si tratásemos de mecanizar la pieza completa desde arriba, existen oscuros a los que la máquina no podría acceder.
Sin embargo, la media pieza es mecanizable desde arriba sin ningún problema.

A nivel personal, el proyecto ha sido muy interesante por diversos motivos.
Se han utilizado en el desarrollo tecnologías y lenguajes con los que no estaba familiarizado, lo cual me ha servido para tomar contacto con ellas y lograr una mejor comprensión de su uso.
Por otro lado, el tema tratado es muy visual, con lo que es más sencillo observar resultados y comprender lo que está ocurriendo en el proceso.
Además, es un campo que podría ver mayor protagonismo en un futuro no muy lejano y deja lugar a muchísima investigación y mejora, ya sea mediante proyectos de la Universidad Politécnica de Valencia u otros proyectos paralelos.

Bibliografía

[1] Ruizhen Hu, Honghua Li, Hao Zhang y Daniel Cohen-Or. *Approximate Pyramidal Shape Decomposition.* ACM Transactions on Graphics, Noviembre 2014

[2] The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 4.8.1 edition, 2016.

[3] Ilker O. Yaz y Sébastien Loriot. Triangulated Surface Mesh Segmentation. In *CGAL User and Reference Manual.* CGAL Editorial Board, 4.8.1 edition, 2016.

[4] Lior Shapira, Ariel Shamir y Daniel Cohen-Or. *Consistent Mesh Partitioning and Skeletonisation using the Shape Diameter Function.* The Visual Computer, Abril 2008

[5] Yuri Boykov, Olga Veksler y Ramin Zabih. *Fast Approximate Energy Minimization via Graph Cuts.* Noviembre 2001

[6] Ramin Zabih y Vladimir Kolmogorov. *Spatially Coherent Clustering Using Graph Cuts.* Julio 2004

Anexos

En este apartado se documenta el código empleado para el proyecto.

Anexo 1: Algoritmo de segmentación

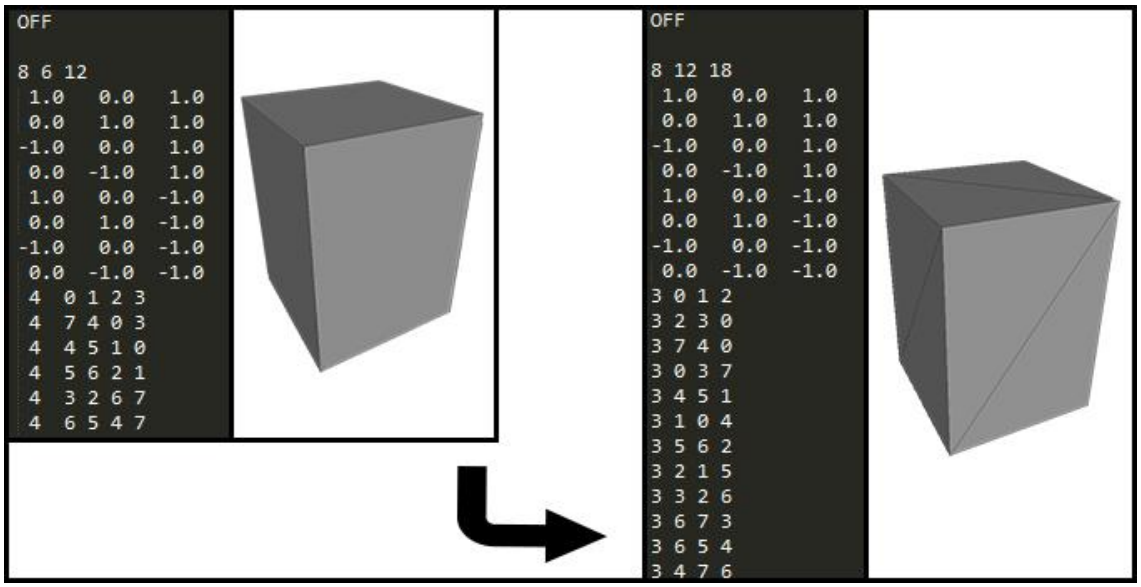
```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/boost/graph/graph_traits_Polyhedron_3.h>
#include <CGAL/Polyhedron_incremental_builder_3.h>
#include <CGAL/polyhedron_cut_plane_3.h>
#include <CGAL/IO/Polyhedron_iostream.h>
#include <CGAL/mesh_segmentation.h>
#include <CGAL/Simple_cartesian.h>
#include <CGAL/Surface_mesh.h>
#include <CGAL/property_map.h>
#include <iostream>
#include <fstream>
#include <vector>

typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;
typedef CGAL::Simple_cartesian<double> K;
typedef CGAL::Surface_mesh<K::Point_3> Mesh;
typedef K::Point_3 Point_3;
typedef Mesh::Vertex_index vertex_descriptor;
typedef Mesh::Face_index face_descriptor;
typedef Polyhedron::Halfedge_around_facet_circulator Halfedge_facet_circulator;
typedef Polyhedron::HalfedgeDS HalfedgeDS;

template<class HDS>
class Build_triangle : public CGAL::Modifier_base<HDS> {
public:
    Point_3 p1,p2,p3;
    Build_triangle() {}
    void operator()( HDS& hds) {
        // Postcondition: hds is a valid polyhedral surface.
        CGAL::Polyhedron_incremental_builder_3<HDS> B( hds, true);
        B.begin_surface( 3, 1, 6);
        typedef typename HDS::Vertex   Vertex;
        typedef typename Vertex::Point Point;
        B.add_vertex(Point(p1.x(), p1.y(), p1.z()));
        B.add_vertex(Point(p2.x(), p2.y(), p2.z()));
        B.add_vertex(Point(p3.x(), p3.y(), p3.z()));
        B.begin_facet();
        B.add_vertex_to_facet( 0);
        B.add_vertex_to_facet( 1);
        B.add_vertex_to_facet( 2);
        B.end_facet();
        B.end_surface();
    }
};

int main(int argc, char *argv[]) {
    // Crear un objeto de tipo Polyhedron en el que almacenar la malla leída de un fichero .off
    // Además se realizan comprobaciones de errores para asegurar que el número de argumentos sea
    Polyhedron mesh;
    if(argc < 2) {
        std::cerr << "No input file." << std::endl;
        return EXIT_FAILURE;
    }
    std::ifstream input(argv[1]);
    //std::cout << argv[1] << std::endl;
    if ( !input || !(input >> mesh) || mesh.empty() ) {
      std::cerr << "Not a valid off file." << std::endl;
      return EXIT_FAILURE;
    }

    // Crear un mapa de propiedades para los valores SDF
    typedef std::map<Polyhedron::Facet_const_handle, double> Facet_double_map;
    Facet_double_map internal_sdf_map;
    boost::associative_property_map<Facet_double_map> sdf_property_map(internal_sdf_map);
```

```
    // Calcular los valores SDF utilizando los parámetros por defecto
    CGAL::sdf_values(mesh, sdf_property_map);

    // Crear un mapa de propiedades para los identificadores de segmento
    typedef std::map<Polyhedron::Facet_const_handle, std::size_t> Facet_int_map;
    Facet_int_map internal_segment_map;
    boost::associative_property_map<Facet_int_map> segment_property_map(internal_segment_map);

    // Definición de los valores por defecto para el número de grupos y el valor de suavizado
    // NOTA: es posible cambiar estos valores mediante paso por parámetros
    std::size_t number_of_clusters = 4;
    if(argc >= 3) number_of_clusters = atoi(argv[2]);
    std::cout << number_of_clusters << std::endl;

    double smoothing_lambda = 0.3;
    if(argc == 4) smoothing_lambda = atof(argv[3]);
    std::cout << smoothing_lambda << std::endl;

    // Llamada a la función de segmentación a partir de los valores SDF con los parámetros establecidos
    std::size_t number_of_segments = CGAL::segmentation_from_sdf_values(
        mesh, sdf_property_map, segment_property_map, number_of_clusters, smoothing_lambda);
    std::cout << "Number of segments: " << number_of_segments << std::endl;

    // Bucle de construcción de la malla de salida
    // Se crearan tantos ficheros .off como segmentos haya creado la segmentación
    // siendo la malla final segmentada la combinación de todos ellos.
    for(int i=0; i<number_of_segments; i++) {
        Polyhedron outmesh;
        for(Polyhedron::Facet_iterator facet_it = mesh.facets_begin();
            facet_it != mesh.facets_end(); ++facet_it) {
            if(segment_property_map[facet_it] == i) {
                Halfedge_facet_circulator hc = facet_it->facet_begin();

                Build_triangle<HalfedgeDS> triangle;
                Point_3 p1(hc->vertex()->point().x(), hc->vertex()->point().y(), hc->vertex()->point().z());
                triangle.p1 = p1;
                hc++;

                Point_3 p2(hc->vertex()->point().x(), hc->vertex()->point().y(), hc->vertex()->point().z());
                triangle.p2 = p2;
                hc++;

                Point_3 p3(hc->vertex()->point().x(), hc->vertex()->point().y(), hc->vertex()->point().z());
                triangle.p3 = p3;
                hc++;

                outmesh.delegate(triangle);
            }
        }
        char nombre[30];
        sprintf(nombre, "segmento%d.off", i);
        std::ofstream output(nombre);
        output << outmesh;
        output.close();
    }

    // Impresión de identificadores de segmento
    for(Polyhedron::Facet_const_iterator facet_it = mesh.facets_begin();
        facet_it != mesh.facets_end(); ++facet_it) {
        std::cout << segment_property_map[facet_it] << " ";
    }
    std::cout << std::endl;
```

Anexo 2: Conversión Off

Para algunos archivos OFF que se han utilizado en el proyecto, ha sido necesario crear un script de conversión, ya que existen mallas que utilizan cuadriláteros para formar las caras, mientras que nuestro algoritmo utiliza triángulos.

Por ello, es necesario convertir aquellas mallas que usen cuadriláteros a mallas que utilicen triángulos.
Con este fin, se ha utilizado AWK para procesar un fichero OFF de entrada y convertir los cuadriláteros en triángulos.



Ejemplo de conversion mediante el script AWK

Como se puede observar en el ejemplo, las caras pasan a ser triangulares.
Por ello, aumenta el número de caras y aristas.