Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Descomposición de volúmenes en esculturas de gran formato

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Andrés Alonso Durán

**Tutor**: Miguel Sánchez López

2014/2015

*"A todos los que me han ayudado apoyado*
*y animado en estos 6 duros años,*
*en especial a mi princesa y mi principito,*
*ahora mi tiempo es vuestro, Gracias"*

# Resumen

La fabricación de esculturas de gran formato se puede abordar mediante la composición de múltiples secciones 3D. La descomposición de un modelo original en múltiples partes es un proceso de considerable complejidad computacional. Este proyecto se centra en el desarrollo de algoritmos que puedan llevar a cabo la tarea de descomposición de una malla 3D de partida en bloques de menor tamaño con los que se pueda componer el modelo inicial, cumpliendo con una serie de restricciones del proceso, las cuales ahondaran en un compromiso entre un número de piezas no excesivo y una definición correcta de la pieza.

**Palabras clave:** 3D, CNC, escultura, algoritmos, inteligencia artificial.

# Abstract

The manufacture of large format sculpture can be addressed by the composition of multiple 3D sections. The decomposition of an original model is a process of considerable computational complexity. This project focuses on the development of algorithms that can perform the task of decomposing a 3D mesh into smaller blocks, fulfilling a range of constraints, reaching a compromise between a small number of parts and a correct definition of the piece.

**Keywords:** 3D, CNC, sculpture, algorithms, artificial intelligence.

# Tabla de contenidos

## Tabla de contenido

1. Introducción ..... 9
1.1. Descripción del problema ..... 9
1.2. Motivación ..... 10
1.2.1. Fallas y hogueras ..... 10
1.2.2. CAD/CAM ..... 12
1.3. Objetivos ..... 13
1.4. Estructura del Trabajo ..... 14
2. Contexto mecánico y tecnológico ..... 15
2.1. Fresado de fallas ..... 15
2.2. STL ..... 16
2.2.1. ASCII ..... 17
2.2.2. Binario ..... 18
2.3. Software y Tecnologías ..... 19
2.3.1. Processing ..... 19
2.3.2. Java ..... 19
2.3.3. Netfabb ..... 20
2.3.4. STL View ..... 21
2.4. Escenario inicial ..... 22
3. Algoritmos e implementación ..... 24
3.1. Anulación de los oscuros ..... 26
3.2. Reorientación de oscuros ..... 27
3.3. Girado de pieza original reorientando oscuros ..... 28
3.4. Triángulos contiguos sin oscuros ..... 29
3.4.1. Normal de referencia, primer triángulo ..... 30
3.4.2. Normal de referencia, media de normales ..... 32
4. Pruebas finales y conclusiones ..... 33
4.1. Figura simple ..... 33
4.2. Figura complejas ..... 34
4.3. Figura de entorno real ..... 35
4.4. Conclusiones ..... 36

5. Valoración personal y Ampliaciones ..... 46
5.1. Valoración personal ..... 46
5.2. Ampliaciones ..... 46
6. Referencias..... 47
Apéndices..... 49
Apéndice I: Código - Anulación de oscuros ..... 49
Apéndice II: Código - Reorientación de oscuros ..... 53
Apéndice III: Código - Girado de pieza original reorientando oscuros..... 55
Apéndice IV: Código - Triángulos contiguos sin oscuros, normal de referencia primer triángulo. ..... 58
Apéndice V: Código - Triángulos contiguos sin oscuros, normal de referencia media de normales. ..... 61

# 1. Introducción

En este capítulo se introducirá la problemática que se quiere abordar mediante una breve descripción del mismo, la motivación que ha influido para seleccionar este problema para el trabajo y los objetivos marcados durante el desarrollo del proyecto.

## 1.1. Descripción del problema

En la actualidad la toma de decisión del traspaso de una maqueta 3D a, lo que en un futuro será, una pieza de falla u hoguera la realiza de forma artesanal el artista fallero, para posteriormente pasarlo a la máquina de fresado, la cual es controlada de forma manual o mediante control numérico.

El control numérico o control decimal numérico (CN) es un sistema de automatización de máquinas herramienta, que son operadas mediante comandos programados en un medio de almacenamiento, en comparación con el mando manual mediante volantes o palancas, que es el método manual.

La automatización del proceso decisorio, respecto a cómo segmentar la maqueta para traspasar directamente a la máquina herramienta la malla totalmente preparada para su fresado, llevaría a un gran avance a la industria en tiempos de fabricación y toma de decisiones.

Dentro de este dilema global, en este proyecto se va a abordar una parte de dicho problema, el estudio de segmentación de la pieza original, para lograr el aprovechamiento de los recursos, dejando de lado el lonchado de la misma, ya que esto depende de factores como por ejemplo el grosor de la plancha de material.

## 1.2. Motivación

Una de mis grandes pasiones es el mundo de las fallas, concretamente el de las fallas que se plantan en la plaza, también llamadas catafalcos, monumentos etc…, junto con mi profesión como Informático y la oportunidad que surgió de realizar un trabajo que abarcase mis dos grandes pasiones, no dudé en capturar dicha oportunidad, e iniciar mi andadura en el mundo de las mallas 3D y su descomposición.

El proyecto ha resultado tratarse de un problema de difícil solución, ya que se trata de un *"NP-Hard"*, por lo cual originalmente imponía el abordarlo, pero gracias a la motivación de mi tutor y asumiendo que se trataba de un problema realmente complejo en fase de investigación, del cual muy probablemente no se sacarían resultados completos, finales ni óptimos, se inició el trabajo con gran ilusión y ganas.

### 1.2.1. Fallas y hogueras

Una falla u hoguera es una expresión artística creada para ser quemada, las fallas, tradicionalmente, la noche del 19 de marzo y las hogueras el 24 de junio. Estos catafalcos se separan entre infantiles y mayores, los primeros son de alrededor de 2,5 metros de altura y de diámetro, mientras que los segundos no tienen límite de medidas, pudiendo alcanzar grandes alturas. Normalmente dichos monumentos constan de una o varias piezas centrales, y unos bajos con ninots más pequeños formando escenas, que habitualmente representan un guión con un tema concreto, del cual la figura central también es partícipe.

En la actualidad, la gran mayoría de esculturas falleras u hogueras están realizadas en poliestireno expandido (corcho blanco), con máquinas herramienta, ya que todavía quedan artistas que utilizan el método tradicional de fallas de cartón-piedra, pero cada vez menos común.

Figura 2 – Detalle de la figura central de la Falla Convento Jerusalén 2015

El proceso de diseño y fabricación de una falla u hoguera comienza con una idea original, escogiendo un tema sobre el que tratar y marcando unas pequeñas directrices de volúmenes. A continuación, se realiza el diseño inicial o boceto, para posteriormente realizarse una maqueta física o digital, aunque si la maqueta es física se escanea para ser digitalizada. Una vez digitalizado el proceso, se modelan las piezas, ya sea manualmente o mediante maquinas herramienta, una vez está la figura montada en el caso de que se haya realizado por partes, se lijan las zonas en las cuales puede haber algún desperfecto, no existe la definición requerida o el corcho no es uniforme. A continuación se procede al empapelado de la pieza con papel de periódico y aquaplast, que una vez seco, se pinta con una capa base de blanca y sobre ella se da color a la figura de la forma deseada.

Figura 2 – Imagen del taller de V. Martínez, detalle del proceso de montaje y pintado

### 1.2.2. CAD/CAM

Computer-aided design (CAD) (diseño asistido por ordenador), es el uso de programas informáticos para crear representaciones gráficas de los objetos físicos en dos o tres dimensiones (2D o 3D). El software CAD se utiliza, entre otros usos, para diseñar productos físicos en una amplia gama de industrias, donde el software realiza los cálculos para determinar la forma y tamaño óptimos para una variedad de productos y aplicaciones de diseño industrial. En nuestro caso va a ser fundamental para que se nos proporcione una malla 3D con la que trabajar. (Ref 1)

Computer-aided manufacturing (CAM) (fabricación asistida por ordenador), comúnmente se refiere al uso del control numérico (NC), que son las aplicaciones de software para crear instrucciones detalladas (G-código) que lleven a las herramientas de unidad de control numérico (CNC) a la fabricación de piezas. (Ref 2)

## 1.3. Objetivos

Este trabajo pretende iniciar un estudio para la solución del problema "NP-Hard" que nos aparece a la hora de automatizar el paso de una maqueta 3D a una fresadora, para que finalice de la mejor manera la pieza o sus partes, llegando a un correcto compromiso entre el detalle fresado de la pieza, el número de trozos y la cantidad de corcho desperdiciado.

Concretamente, en este trabajo se pretende desarrollar un conjunto de algoritmos que lleguen a aportar un poco de luz en referencia al troceado de las figuras en piezas más pequeñas, para posibilitar que todos los trozos puedan ser fresados y queden las mínimas zonas donde la fresadora, que se mueve en dos ejes, no pueda acceder y a su vez valorar un compromiso entre el número de piezas generadas y el tamaño de las mismas, ya que una pieza que contenga solo un triángulo de la malla de un tamaño ínfimo no es interesante para su posterior mecanizado.

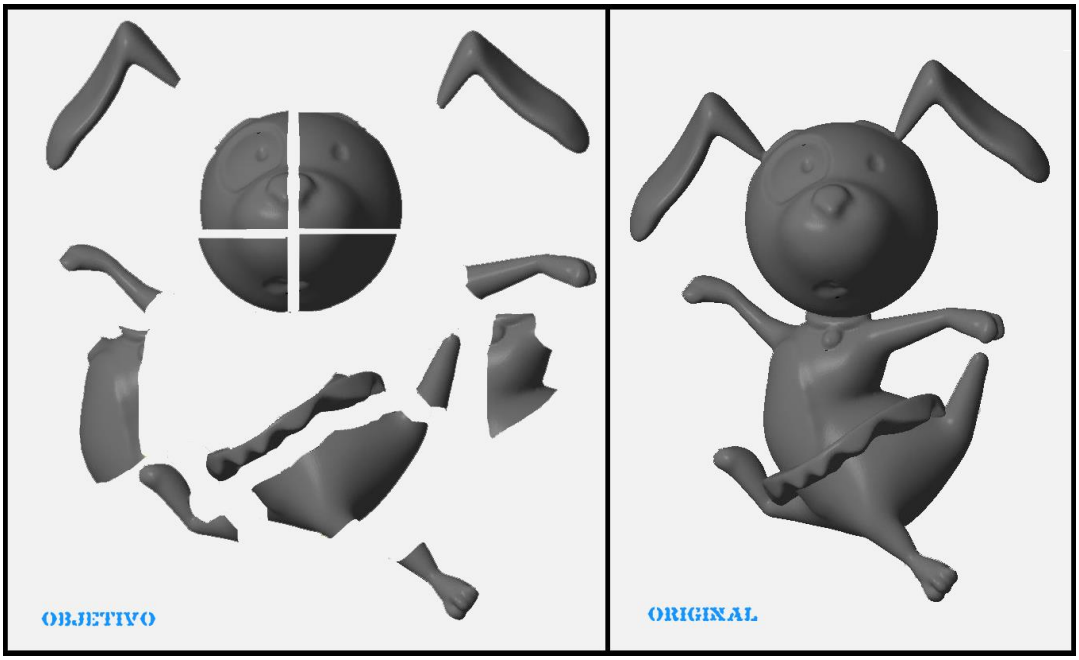



Figura 3 – Idea inicial aproximada de troceado

Teniendo siempre en cuenta la cantidad de variables que existen y pudiendo tomar diferentes caminos en nuestras decisiones en este trabajo, que tal vez desechemos por no poder ser aprovechables para el mismo, el objetivo real de este proyecto no es tanto un resultado finalizado, sino un estudio de las líneas a seguir o no, para así poder descartar algunas vías de investigación, o abrir una nueva línea de investigación que añada una parte de la lógica que conduzca a un producto definitivo.

## 1.4. Estructura del Trabajo

El presente trabajo queda organizado en 5 capítulos.

En el capítulo 1 se introduce brevemente el problema que deseamos resolver, la motivación junto unas pequeñas nociones básicas relativas al problema y los objetivos que deseamos obtener con la conclusión de este trabajo.

En el capítulo 2 se describe el contexto tecnológico y mecánico al cual nos vamos a enfrentar, tales como a que maquinaria va a estar orientado el estudio y las herramientas lógicas que se van a utilizar.

El caso de estudio se expone en el capítulo 3, donde se muestran todos los algoritmos y líneas de investigación que se han seguido en este proyecto.

La evaluación, pruebas y conclusiones respecto a todo lo desarrollado en el capítulo 3 ocupa por completo el capítulo 4.

El capítulo 5 dará unas breves directrices a las posibles ampliaciones de este trabajo y una valoración personal del mismo.

# 2. Contexto mecánico y tecnológico

En este apartado se va a exponer la realidad actual en el trabajo del fresado de fallas, así como una breve explicación de los conceptos y herramientas tecnológicas que se han utilizado para la realización de este proyecto.

## 2.1. Fresado de fallas

Una máquina herramienta fresadora para fallas es, básicamente, igual que la tradicional fresadora que se utiliza en cualquier taller, pero con unas dimensiones mayores, y que es manejada por control numérico o mediante un software destinado para ello.

Esta máquina modela el material por arranque de viruta mediante el movimiento de una herramienta rotativa de varios filos de corte, llamada fresa.

Figura 4 – Teccam 4000 (imagen de http://www.perezcamps.com/) (Ref. 13)

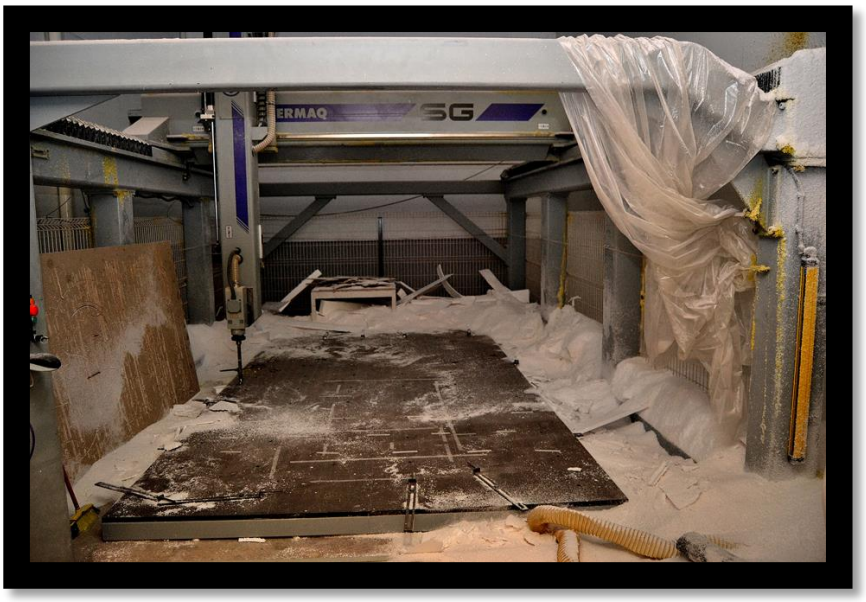



Figura 5 – Fresadora del taller fallero de V. Martínez

## 2.2. STL

STL es un formato de archivo informático de diseño asistido por computadora (CAD), que define geometría de objetos 3D, excluyendo información como color, texturas o propiedades físicas que sí incluyen otros formatos CAD. Existen los STL con color, pero no los vamos a utilizar en este proyecto. (Ref. 3)

Fue creado por la empresa 3D Systems, concebido para su uso en la industria del prototipado rápido y sistemas de fabricación asistida por ordenador. En especial desde los años 2011-2012, con la aparición en el mercado de impresoras 3D de extrusión de plástico termofusible (personal y asequible), el formato STL está siendo utilizado ampliamente por el software de control de estas máquinas. (Ref. 3)

Este formato utiliza una malla de triángulos cerrada para definir la forma de un objeto. Cuanto más pequeños son estos triángulos, mayor será la resolución del fichero final; a su vez, el tamaño de los triángulos es directamente proporcional con el peso del fichero.

No todos los triángulos de un STL han de mantener el mismo tamaño, ni ser en absoluto iguales, en zonas donde se necesita menos resolución o son muy rectas se utilizan triángulos más grandes y menos cantidad, en cambio, en zonas curvas o con necesidad de más resolución, los triángulos a utilizar serán más pequeños y por consiguiente existirá mayor población de los mismos.

Todo modelo 3D está compuesto de triángulos, y estos a su vez de vértices, líneas y caras. Estas últimas poseen un vector unitario llamado normal, que indica la dirección de la cara externa de cada uno. Naturalmente, todas las normales de un objeto deben apuntar hacia el exterior, de manera que no haya lugar a equívocos sobre si es dentro o fuera de la pieza.

Existen dos tipos de archivos STL, los ASCII y los binarios.

### 2.2.1. ASCII

Un archivo STL ASCII comienza con la línea:

```
solid nombre
```

Donde nombre es una cadena opcional. El archivo continúa con cualquier número de triángulos, cada uno representado como sigue:

```
facet normal ni nj nk
  outer loop
    vertex vx1 vy1vz1
    vertex vx2 vy2 vz2
    vertex vx3 vy3 vz3
  endloop
```

Donde cada “n”(ni, nj, nk) o “v” (vxq, vy1, vz1) es un float.

El archivo concluye con:

```
endsolid nombre
```

(Ref. 9)

### 2.2.2. Binario

Los archivos STL ASCII pueden llegar a ser muy pesados, por este motivo existe la versión binaria de STL. Un archivo STL binario tiene una cabecera de 80 caracteres. La siguiente línea después de la cabecera es un entero sin signo de 4 bytes, que indica el número de triángulos en el archivo. A continuación muestra los datos que describen cada triángulo. El archivo simplemente termina después del último triángulo.

Cada triángulo es descrito por doce números en coma flotante de 32 bits: tres para la normal y tres para las coordenadas X, Y y Z de cada vértice. Después de éstos continua un entero sin signo, que es el número de bytes del atributo; este dato debería ser cero, porque la mayoría de los softwares no interpretan otra cosa.

UINT8[80] – Cabecera

UINT32 – Número de triángulos

Para cada triangulo

REAL32[3] – Vector normal

REAL32[3] – Vértice 1 (x,y,z)

REAL32[3] – Vértice 2 (x,y,z)

REAL32[3] – Vértice 3 (x,y,z)

UINT16 – Contador de bytes, normalmente 0.

(Ref. 9)

## 2.3. Software y Tecnologías

Este capítulo es el encargado de documentar el software y las tecnologías utilizadas en el proyecto, tanto a nivel de desarrollo, como visualización de resultados.

### 2.3.1. Processing

Processing es un lenguaje de programación, entorno de desarrollo, y comunidad en línea. Desde 2001, Processing ha promovido la alfabetización de software dentro de las artes visuales y la alfabetización visual dentro de la tecnología. Inicialmente creado para servir como un cuaderno de bocetos de software y para enseñar los fundamentos de programación de computadoras dentro de un contexto visual, Processing evolucionó hasta convertirse en una herramienta de desarrollo para los profesionales. Hoy en día, hay decenas de miles de estudiantes, artistas, diseñadores, investigadores y aficionados que utilizan Processing para el aprendizaje, la creación de prototipos y producción.

El entorno de programación de Processing consiste en un simple editor de texto para escribir código, un área de mensajes, una consola de texto, pestañas para la gestión de archivos, una barra de herramientas con botones de “Play” que se usa para ejecutar e-código escrito, “Stop” cancelación de la ejecución en curso, “New” para la creación de nuevos proyectos, “Open”, “Save”, “Export”, y una serie básica de menús.

Processing dispone de diferentes modos de programación, para hacer posible el despliegue de bocetos sobre distintas plataformas y programas. De forma predeterminada y el que se ha usado en este proyecto, es el modo java, el cual como su nombre indica está basado en Java, y como su lenguaje nativo, orientado a objetos.

(Ref. 7)

### 2.3.2. Java

El lenguaje de programación Java es un lenguaje de propósito general, imperativo, concurrente, orientado a objetos y multiplataforma de fácil portabilidad, ya que se ejecuta sobre una máquina virtual Java, sin tener en cuenta la arquitectura del ordenador residente. Este lenguaje tiene un extenso catálogo de librerías disponibles de uso libre y gratuito, que pueden adaptarse a Processing de la misma manera que se hace en Java. (Ref. 11)

### 2.3.3. Netfabb

Netfabb no es solo un visualizador de la una malla 3D, sino que permite una serie de creaciones, reparaciones, selecciones, etc... En resumen, es un software avanzado para impresión 3D y fabricación aditiva. (Ref 5)

En el caso de este proyecto lo hemos utilizado en su versión basic (free), como visualizador, para crear objetos básicos para las posteriores pruebas y exportar dichos proyectos 3D con extensión .fabbproject a los STL que nos ocupan en este trabajo. Se ha utilizado este software a su vez, de motivador de ideas sobre cómo utilizar, modificar o consultar la malla en los distintos algoritmos que se han ido desarrollando a lo largo del proyecto.

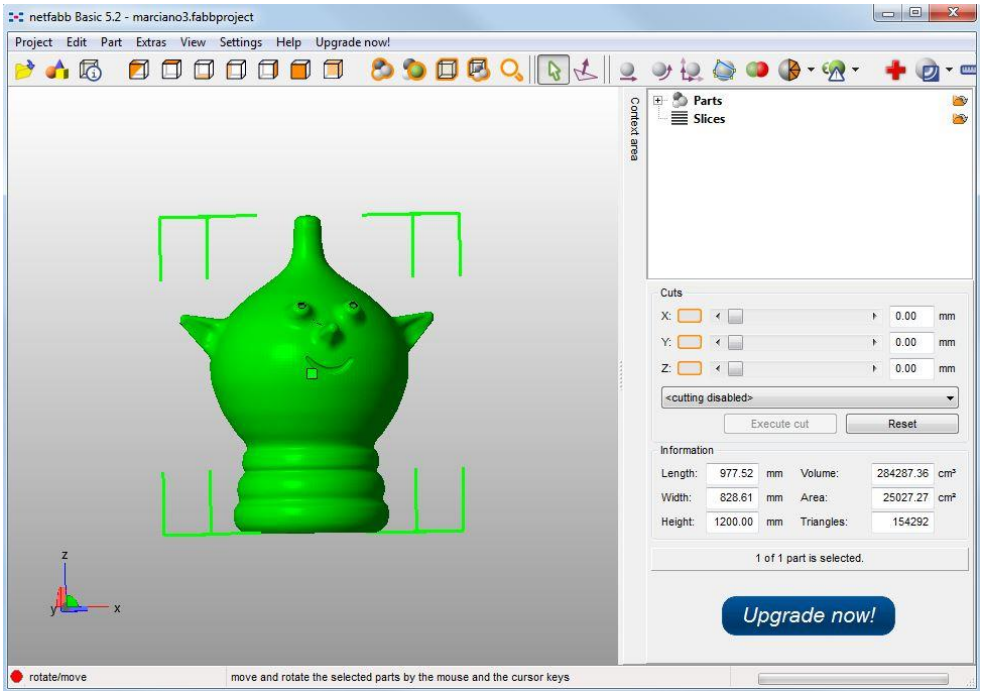



Figura 6 – Interface netfabb basic

### 2.3.4. STL View

STLView es un visor de STL básico, que se ha utilizado principalmente en este trabajo para la visualización conjunta de todos los trozos/STL resultado en un solo modelo, cada uno de un color, pudiendo ocultar o mostrar los fragmentos que han interesado. (Ref. 10)

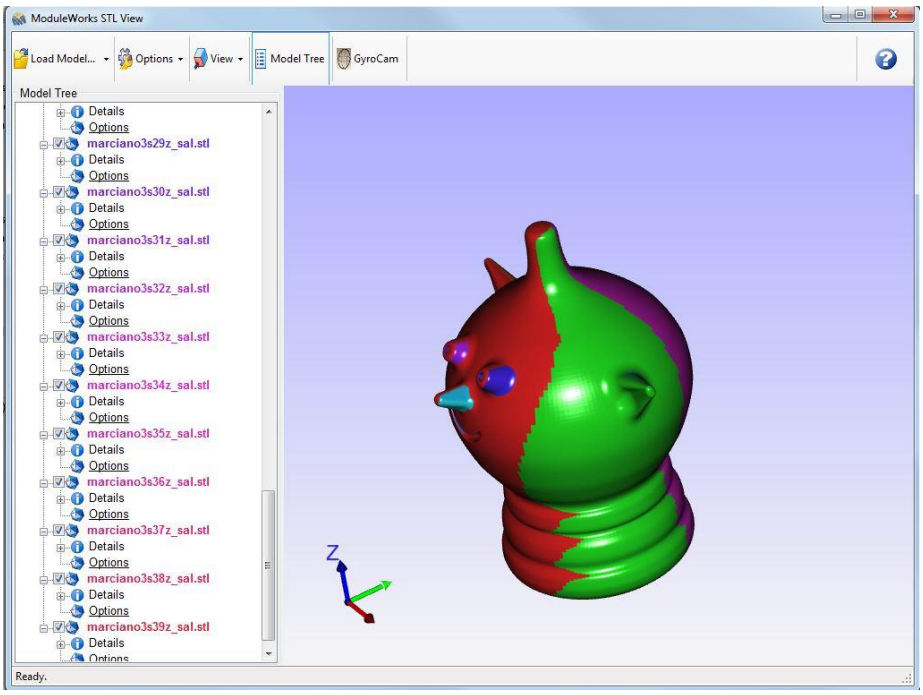



Figura 7 – Interface STLView

## 2.4. Escenario inicial

En este apartado se va a explicar tanto el cómo se aborda de forma manual el problema global que nos ocupa en los talleres de constructores de fallas, así como las primeras investigaciones y conceptos iniciales que necesitábamos para introducirnos en el problema a nivel tecnológico.

En los talleres, en la actualidad, el proceso de decidir el despiezado de la maqueta 3D para lograr un menor coste en corcho o para dejar la figura modelada lo más finamente posible, y así minimizar el proceso de lijado, se realiza de forma artesanal, decidiendo el artista como realizar las partes para que sea posible el fresado de las piezas de la forma más eficiente posible, ya que una misma figura no ocupa igual en las planchas de corcho, donde se van a fresar las piezas, si el despiece está realizado de una manera u otra. Si se secciona la maqueta en un exceso de piezas generará un alto nivel de detalle, en cambio si el seccionado es menor, el detalle será menor también, lo que conllevará un mayor proceso de lijado, aunque a su favor hay que indicar que el aprovechamiento del corcho es posible que sea mayor. Probablemente que existan muchas piezas muy pequeñas tampoco sea lo más óptimo a la hora del montaje de la figura final, por ello la mejor solución posiblemente sea lograr un compromiso ente aprovechamiento del corcho y volumen de lijado (nivel de detalle) que se desea que genere la máquina.

Como se ha comentado en apartados anteriores, la idea es desarrollar una línea de investigación para intentar lograr un despiece de forma automatizada, mediante una serie de algoritmos que realizaran dicho despiece, valorando posteriormente mediante varias pruebas si las líneas de investigación han se han encaminado en la dirección correcta.

A nivel técnico, antes de empezar el proceso algorítmico es necesario un proceso de investigación inicial, donde, tras estudiar respecto de las mallas, sus triángulos y de los vectores normales de cada uno de los triángulos, es necesario conocer qué es un oscuro. Llamamos oscuro a la zona no accesible desde la máquina herramienta por estar su normal a un ángulo mayor de 90º respecto del eje de mecanizado.

Se llama eje de mecanizado o dirección de mecanizado, el eje por el que la máquina herramienta accederá a la plancha de corcho para empezar a moverse y modelar la pieza final.

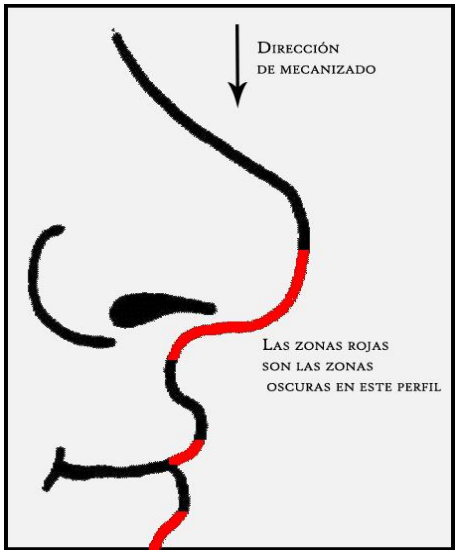



Figura 8 – Oscuros

A nivel tecnológico se disponía de un software de código abierto llamado SuperSkein, que realiza el lonchado de bocetos STL, mostrando por pantalla lo que serían los pasos en un proceso de impresión 3D del STL utilizado. Este trabajo originalmente se ha inspirado en las ideas que el código de este software nos ha facilitado, siendo utilizadas y adaptadas algunas de sus clases y sus métodos para la realización de nuestro estudio de seccionado, y en consecuencia formar parte de nuestro código.

# 3. Algoritmos e implementación

En este apartado se va a proceder a la descripción de los algoritmos que se han desarrollado durante este proceso de investigación, tanto los más básicos utilizados para la familiarización con las mallas y el entorno, como los últimos más elaborados.

En todos los distintos algoritmos que se presentan a continuación, se interpreta originalmente el archivo de entrada .stl que se decide en cada momento, independientemente de tratarse de un STL Binario o un STL ASCII. La lectura del fichero y el guardado de los datos se realiza guardando cada triángulo de la malla en un objeto Triangle, y a su vez, esos objetos se van almacenando en un ArrayList de triángulos. Todo esto se realiza en la clase mesh, que contendrá el grueso de la aplicación.

Para las pruebas y en este entorno de desarrollo en el que se enclava el proyecto, la forma de pasarle el archivo .stl original al programa, se realiza de la siguiente manera:

El primer paso es ubicar el archivo a utilizar en la misma carpeta donde se encuentran los archivos del proyecto. En este caso mesh.pde y Triangle.pde. Tras este paso, simplemente es necesario asignar a la variable global archivo, el nombre del archivo sin la extensión.

La lectura de los archivos STL originales se realiza de la siguiente forma:

En el caso de los STL ASCII se procede a la lectura en sí del archivo mediante el método LoadTextMesh, el cual omite el guardado de los datos que no nos interesan y guarda en variables float los datos que precisamos; en total 12 floats que serán a su vez las coordenadas de cada uno de los vértices del triángulo y las coordenadas del punto de la normal, ya que las normales tienen su origen en el eje 0,0,0, y solo existe en el archivo un punto para definir el vector.

Finalmente utilizamos las variables guardadas anteriormente para crear el objeto triangulo y almacenamos cada uno de estos triángulos en un ArrayList mediante la línea:

Triangles.add (new Triangle (x1, y1, z1, x2, y2, z2, x3, y3, z3, xn, yn, zn, xc, yc, zc));

Siendo las variables x1, y1, z1, las coordenadas del punto del primer vértice. Lo mismo resulta con las variables x2, y2, z2, x3, y3, z3 en el caso de los otros dos vértices del triángulo. Las variables xn, yn, zn, se corresponden con las coordenadas de la normal y las variables xc, yc, zc se tratan de los puntos centrales del triángulo, los cuales han sido calculados en este mismo método dividiendo entre tres la suma de todas las x de los tres vértices del triángulo para sacar xc y realizando el mismo proceso respectivamente para yc y zc.

En el caso del STL binario se procede a la lectura en del archivo mediante el método LoadBinaryMesh, para ello recorremos los datos del archivo omitiendo los 84 Bytes de cabecera más el número de triángulos y guardamos los siguientes cuatro bloques REAL32[3], que determinan los puntos de la normal y los vértices del triángulo, en un vector de floats, para posteriormente crear el objeto Triangle y guardarlos en el ArrayList, igual que hacíamos para los ASCII. También se realiza el mismo cálculo para los puntos centrales de cada triangulo, de la forma:

Triangles.add(newTriangle(Tri[3],Tri[4],Tri[5],Tri[6],Tri[7],Tri[8],Tri[9],Tri[10],Tri[11],Tri[0],Tri[1],Tri[2],xcb,ycb,zcb));

Evidentemente, tras guardar los datos en el vector, omitimos la lectura de los siguientes 4 Bytes que existen en el archivo.

Para un correcto funcionamiento de las lecturas mencionadas se ha creado una clase triangle, que se encarga de la creación de los objetos, conteniendo un constructor de la clase y unos métodos que retornan los valores de los mismos. Esta clase será modificada y ampliada con nuevos métodos durante las diferentes etapas del proyecto.

Posteriormente, en todos los casos se generan uno o una serie de stl finales, que se guardan en la ruta que se ha destinado al respecto. En el caso actual es una ruta local fija del ordenador utilizado. Para cambiar la ruta simplemente es necesario modificarla en la creación de cada uno de los ficheros en el método create() de la clase mesh.

El método setup es el lanzador inicial del programa que se encarga de crear el objeto mesh para iniciar la lectura del archivo, y llama al método create() que es el que crea el STL resultante. Este método también va a ir variando en cada una de las etapas del proyecto.

El método create() va  a ser el encargado de crear los archivos resultantes y de calcular la gran parte de la lógica. Es donde se va a trabajar principalmente para poder realizar las pruebas de los algoritmos.

Es importante mencionar que este proyecto va a ir siendo más complejo conforme se vaya avanzando en el trabajo, donde cada paso va a ir basándose en ideas y código de los desarrollos anteriores, que quedaran explicados en sus apartados concretos.

## 3.1. Anulación de los oscuros

Como punto de partida en el proyecto, y para realizar una primera toma de contacto con el mundo 3D de mallas, triángulos y normales, se ha procedido a desarrollar un algoritmo que manteniendo la estructura de la figura a trabajar, como resultado final generara un archivo STL ASCII donde no mostrara los oscuros, asumiendo como eje de mecanizado el eje z.

El método setup() en este primer algoritmo se encarga de la llamada para la creación del objeto mesh, y desde ahí proceder a la lectura del archivo stl, y tras informar por pantalla del número de triángulos que tiene el archivo origen realiza la llamada al método create() que será el encargado de la creación del STL resultado. Una vez finalizado dicho método se muestran por pantalla el número de triángulos añadidos a la nueva malla y los que han quedado omitidos por ser oscuros.

El método create(), que es el encargado principal del algoritmo, realiza la creación del PrintWriter para la generación de los ficheros finales, donde se decide a su vez la ruta donde se van a guardar dichos archivos. Posteriormente, recorremos todos los triángulos del mesh, y para cada triángulo de dicho mesh realizamos la comprobación de si es oscuro respecto a la orientación del mecanizado. Esto se realiza comparando si la normal está a 90º o menos del eje de mecanizado; para ello se revisa si la coordenada z de la normal es mayor o igual a cero; en caso afirmativo nos indica que no es un oscuro respecto a nuestra orientación de mecanizado definida, la cual se ha definido sobre el eje z en sentido descendente, y este triángulo se añadirá en el archivo final.  En este mismo método aprovechamos el finally para cerrar los ficheros.

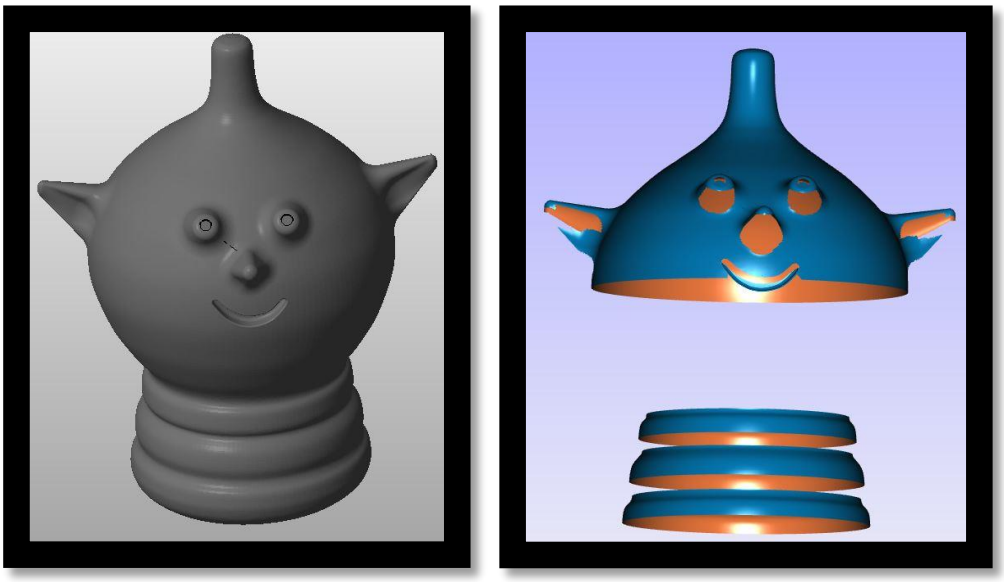


Figura 9 – Figura original y Figura sin oscuros

## 3.2. Reorientación de oscuros

Tras la toma de contacto, se ha modificado el programa de eliminación de oscuros para que como resultado final se generen seis archivos STL ASCII, asumiendo como eje de mecanizado el eje z en sentido de las coordenadas positivas hacia cero, para el primer archivo; el eje y para el segundo y el eje x para el tercero de ellos, todos en el mismo sentido de mecanizado que el archivo uno (de positivos hacia cero), el cuarto, el quinto y el sexto, se basan en los mismos ejes de mecanizado pero en sentido contrario (en dirección de coordenadas negativas hacia cero), realizando una rotación de 180º a los triángulos de los archivos cuarto, quinto y sexto, para que, tras su creación, queden orientados para ser mecanizados en el mismo sentido que su fichero inverso.

La única modificación sobre el método setup() se trata de la información que se muestra tras la creación por pantalla, donde se imprime por pantalla el número de triángulos añadidos a las diferentes nuevas mallas, ya que no se omite ninguno, y se añaden todos los triángulos a una u otra malla.

En el método create() se realiza la creación de los PrintWriter, que ya serán seis y no uno, con su correspondiente ruta donde se van a guardar, para la creación de los ficheros finales. Posteriormente, recorremos todos los triángulos del mesh, y para cada triángulo de dicho mesh realizamos la comprobación de si es oscuro respecto a la primera orientación del mecanizado, comparando si la normal está a 90º o menos del eje de mecanizado. Esto lo realizamos revisando

si la coordenada correspondiente de la normal del triángulo es mayor o igual a cero, por ejemplo, si la coordenada z de la normal es mayor o igual a cero, quiere decir que no es un oscuro en el caso que la dirección de mecanizado sea de los positivos hacia el cero del eje z; en caso afirmativo, ese triángulo se añadirá en ese archivo; en caso negativo se añadirá en un archivo que se compondrá de los oscuros de su fichero "inverso", pero reorientados para poder coincidir con la orientación de mecanizado original.

Esta reorientación del triángulo se realiza multiplicando por menos uno la coordenada que genera el oscuro en cada uno de los tres vértices del triángulo y en su normal, así se logra que mantenga el fichero final la misma orientación que tenía su fichero contrapuesto. En este mismo método aprovechamos el finally para cerrar los ficheros.

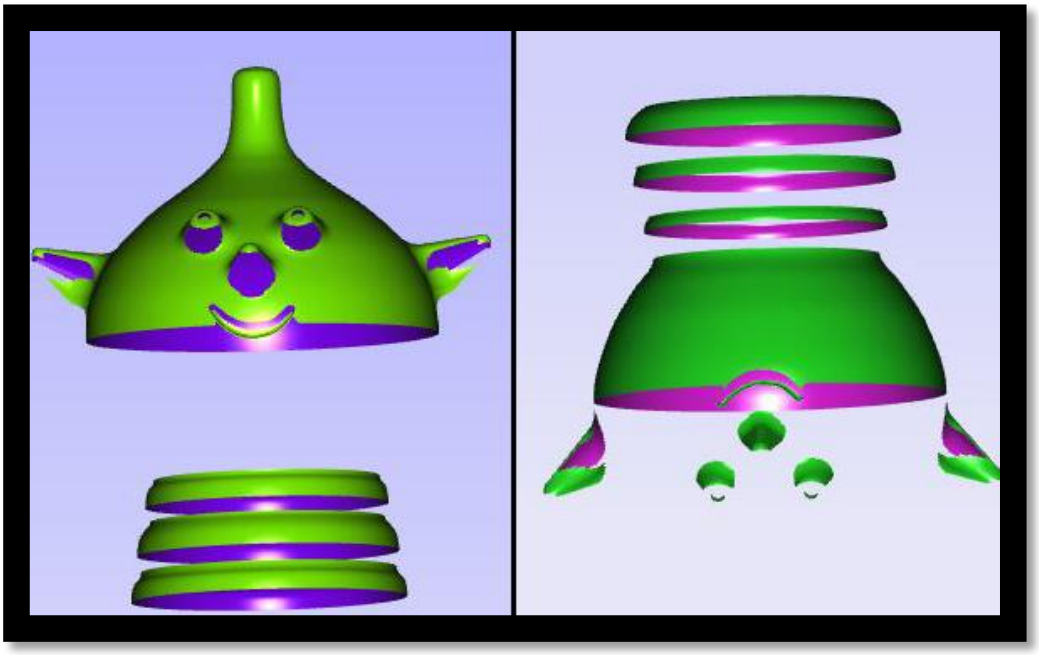


Figura 10 – Boceto sin oscuros y su inverso rotado respecto del eje z.

## 3.3. Girado de pieza original reorientando oscuros

El siguiente paso, que parece una lógica evolución, es el de intentar ejercer los mismos algoritmos que se aplican en el apartado 3.2, pero desde más orientaciones de mecanizado que los que nos ofrecen exclusivamente los ejes de coordenadas habituales, ya que las muestras generadas con los ejes de coordenadas estándar son muy pocas para determinar si es óptimo o no el troceado de la pieza.

Por ello, la idea de este algoritmo es ir rotando la pieza original una cantidad de grados en el eje x hasta realizar una vuelta completa en este eje, y a cada movimiento de la pieza original aplicarle los algoritmos del apartado anterior. Posteriormente, en el eje y se realizan los mismos procesos, y para finalizar en el eje z se repite la ejecución.

Para la realización de esta idea, y antes de explicar cómo quedaran los métodos create() y setup() respectivamente, se ha tenido que ampliar la clase Triangle, donde se han creado tres métodos iguales, uno para cada uno de los ejes, los cuales, recibiendo el ángulo de rotación y aplicando la fórmula del siguiente ejemplo que rota sobre el eje z, a todos los puntos del triángulo: "*xn = x1*cos(Angle) - y1*sin(Angle);  yn = x1*sin(Angle) + y1*cos(Angle);*" logran generar la rotación del triángulo correspondiente.

Dentro de la clase mesh se han creado tres métodos más, llamados rotatX(), rotatY() y rotatZ(), que son los encargados de rotar todos los triángulos utilizando los métodos modificadores de cada triángulo, explicados anteriormente, de la clase Triangle. Para ello, se recorre todo el mesh y se aplica el modificador del objeto a cada triángulo.

El método create() no varía respecto al ejemplo anterior. En cambio, el método setup() si tiene unas ligeras modificaciones. Para llamar al método create() una vez por cada giro, se han creado tres bucles, uno para cada eje, en cada uno de ellos definimos el ángulo de rotación y se lo pasamos en radianes al método rotatX() cuando giramos respecto al eje x, al rotatY cuando la rotación es sobre el eje y, y lo propio para rotatZ(). Posteriormente, se aplica el método create() al objeto rotado.

Los grados a girar la pieza original en cada uno de los bucles del setup() y antes de llamar a los métodos rotat..() correspondientes, es necesario pasarlos a radianes, utilizando la fórmula: PI/datoenangulos, ya que la formula aplicada en los métodos de la clase Triangle trabaja en radianes.

## 3.4. Triángulos contiguos sin oscuros

En este algoritmo, del que se han desarrollado dos vertientes, se ha decidido optar por la búsqueda de triángulos contiguos en la malla, y dibujar solo en cada STL los triángulos contiguos que no están en orientación de oscuro respecto al eje de mecanizado. La decisión de dicho eje de mecanizado es la única diferencia entre las dos modalidades de este algoritmo.

### 3.4.1. Normal de referencia, primer triángulo

En esta variante del algoritmo se ha decidido que la normal del primer triangulo que se dibuja en cada uno de los STL, sea el eje de mecanizado por el que se va a determinar qué triángulos están en oscuro o no. Y basándose en las premisas de no ser un oscuro, el triángulo respecto del eje de mecanizado y ser contiguo a alguno de los triángulos ya dibujados en ese fichero .stl, se van creando diferentes archivos que son el resultado de este algoritmo.

Para iniciar el proceso se ha tenido que fabricar una clase nueva para crear los objetos tipo Point. Cada objeto tipo point se corresponde con las coordenadas de un punto de un triángulo; dicha clase consta de su constructor, Point(), que recibe las tres coordenadas de un punto y tres consultores que retornan dichas coordenadas, getx(), gety(), getz().

Por otra parte, en la clase Triangle se le ha añadido como parámetro al constructor, un booleano que será el que indique si se ha añadido en algún archivo o no dicho triangulo. También se ha creado un método modificador que cambia esta variable de false a true en el caso de que ya haya sido pintado el triángulo, y un consultor que retorna si ha sido visitado el triángulo o no.

Ya de vuelta a la clase mesh, en el método setup() se llama al método créate() mientras queden triángulos por añadir. Esto se consigue mediante un bucle al que se entra mientras queden triángulos por añadir. Esto se ha logrado sumando uno a la variable “contvisit” cada vez que se añade un triángulo en algún STL, y mientras esta variable sea menor que el tamaño del ArrayList Triangles se repite el bucle. En dicho bucle, la variable “primero” se pone a true para lograr que en la primera entrada al create() sea éste el triángulo del cual se usa la normal como referencia de eje de mecanizado.

En esta misma clase se ha creado un método entreangulos() que recibe las tres coordenadas de dos vectores distintos, siempre recordando que el otro punto del vector normal es 0, 0, 0, y nos retorna true si el ángulo que forman es mayor a 90º, o la diferencia que se defina en dicho código, mediante la fórmula:

*“acos(((px*qx)+(py*qy)+(pz*qz))/( sqrt((px*px)+(py*py)+(pz*pz)) * sqrt((qx*qx)+(qy*qy)+(qz*qz))));”.*

Ya en el método create() y habiendo creado una variable global de tipo cola que se llenará de objetos tipo Punto, se repite un bucle mientras existan puntos a valorar en dicha cola, consultando la cola mediante Puntos.peek();, para no realizar un desencolado, o si es el primer triangulo a añadir, es decir si "primero" es igual a true. Posteriormente, si la variable "primero" ha sido false, quiere decir que se ha entrado al bucle por tener puntos encolados, entonces los desencolamos, ahora si, mediante Puntos.poll(); y guardamos sus valores en variables, para después ser comparadas. Dentro del bucle primero, se entra en otro bucle que se repetirá para todos los triángulos del mesh, si "primero" es igual a true y ese triángulo no ha sido añadido aún en ningún stl, se añade dicho triángulo y se guardan los puntos de las normales para usar ese vector como referencia, y uno de los puntos del triángulo también en variables, las mismas en las que se guardaba si "primero" era igual a false. Los otros dos puntos del triángulo se encolan en la cola de objetos punto mediante Point.add(new Point(tri.getx3(),tri.gety3(),tri.getz3()));. Si no es el primer triangulo se comprueba si ha sido añadido o no y si no es oscuro respecto al eje de mecanizado; en este caso, la normal del primer triángulo añadido, siempre que esta consulta sea afirmativa, se comprueba si algún punto del triángulo coincide con el punto que se ha desencolado, y si es así, se añade el triángulo, se suma uno a la variable "contvisit" y se encolan sus otros dos puntos. Y todo esto como he comentado antes, hasta que están todos los triángulos añadidos en algún archivo.

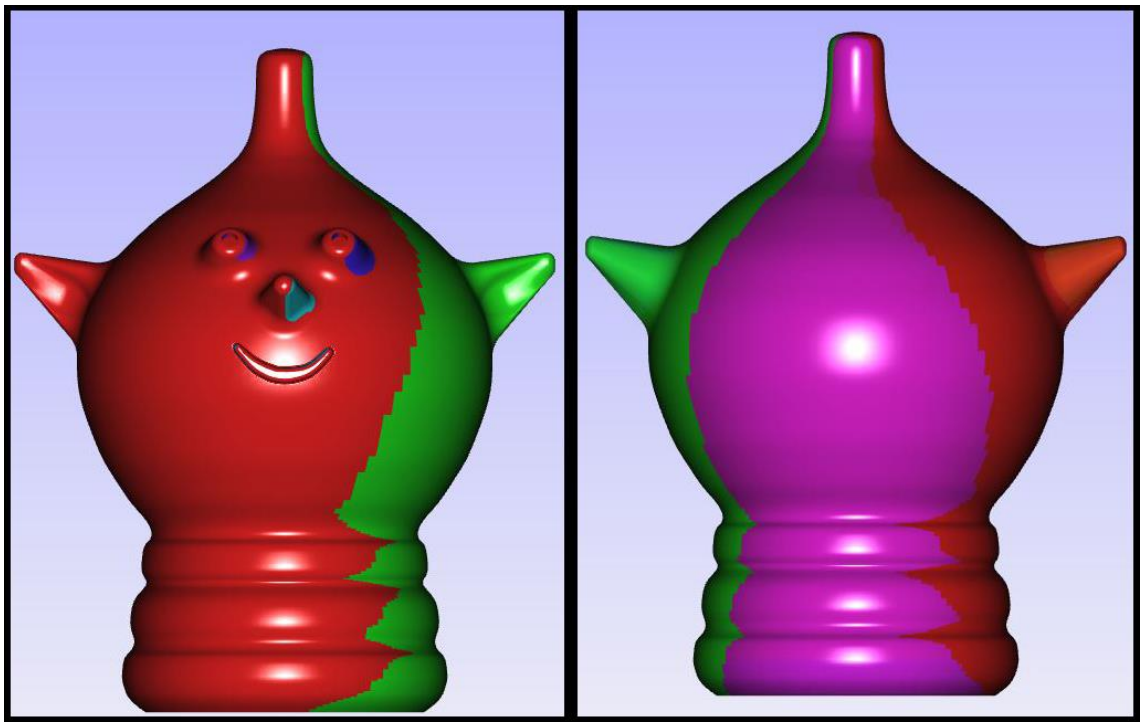


Figura 11 – Los 39 archivos resultado vistos en conjunto.

### 3.4.2. Normal de referencia, media de normales

En este algoritmo el código es básicamente el mismo que en el ejemplo anterior, con la salvedad de que el eje de mecanizado es el que sería el vector media de todas las normales de los triángulos, que aún no han sido añadidos en ningún archivo resultado.

Esto se consigue creando en la clase mesh tres métodos iguales, cada uno para cada coordenada “x”, “y” o “z”, llamados normalmediarestx(), normalmediaresty() y normalmediarestz(), que consiste en que cada uno encuentre la media de cada una de las coordenadas de cada punto. Por ejemplo, en el basado en la x, este método retornará la media de todas las x de las normales de los triángulos que aún no se han añadido, recorriendo todo el mesh en cada uno de los métodos y sumando la cantidad si no ha sido añadido, luego se divide por la cantidad de puntos sumados.

En el método create(), la única modificación es que cuando se asigna el vector de referencia en lugar de asignar a esas variables los valores del vector normal del primer triángulo, se llama a los métodos anteriormente explicados.

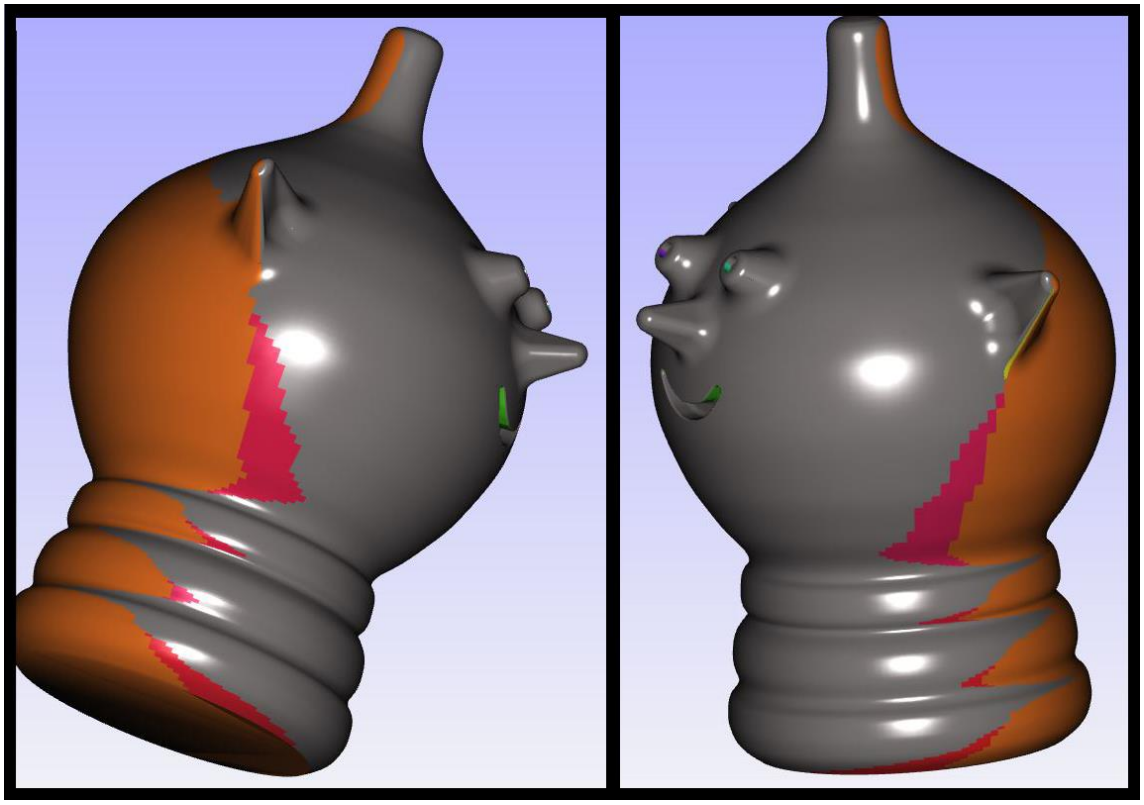


Figura 12 – Los 218 archivos resultado vistos en conjunto.

# 4. Pruebas finales y conclusiones

En este apartado se va a proceder a explicar los resultados obtenidos en todas las pruebas realizadas con los algoritmos arriba detallados, mostrando capturas de dichos resultados.

Inicialmente se explican brevemente las figuras utilizadas para las pruebas realizadas durante este proyecto, dichas figuras van aumentando de complejidad por cantidad de triángulos y detalle de las mismas piezas, al igual que la variabilidad de oscuros.

## 4.1. Figura simple

Como figura simple se ha optado usar una figura geométrica 3D con caras planas, pero con algo más de complejidad que un simple cubo, su nombre es Rhombicuboctahedron. Dicho poliedro consta de dieciocho caras cuadradas y ocho triangulares. En el caso de la malla de prueba de este poliedro, consta de 44 triángulos en su malla, dividendo las caras cuadradas en dos triángulos iguales y formando sus caras triangulares un único triángulo cada una.

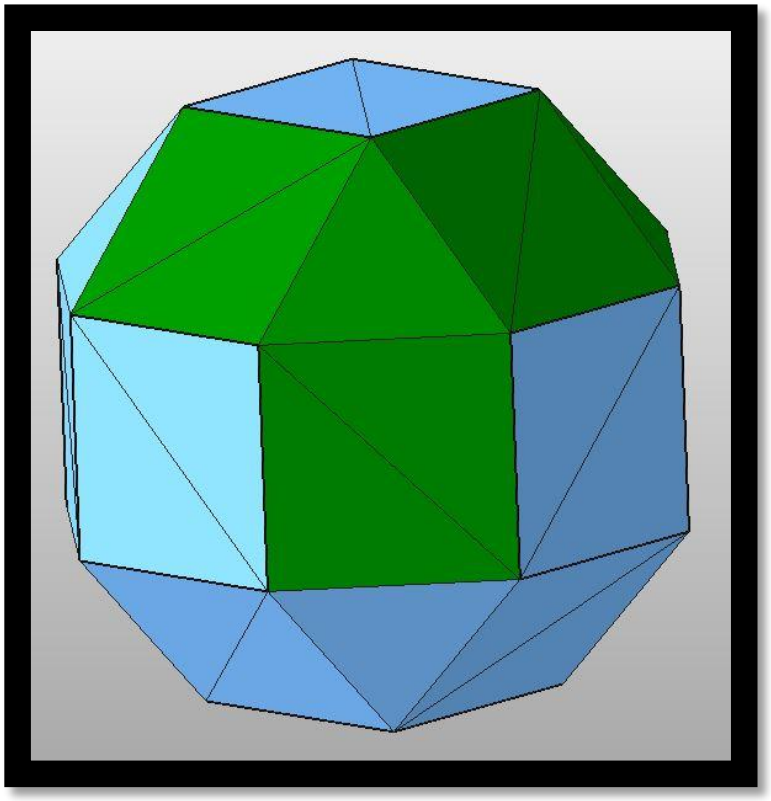


Figura 13 – Rhombicuboctahedron

## 4.2. Figura complejas

Para el caso de la figura compleja, se ha optado por una figura que prácticamente no tiene partes planas, lo que genera una cantidad y diversidad de triángulos amplia en la malla. En este caso, se trata de una figura llamada knot, o nudo. Dicha figura tiene ese nombre por su similitud a un nudo. En este caso, la malla 3D que nos ocupa consta de 76.452 triángulos.

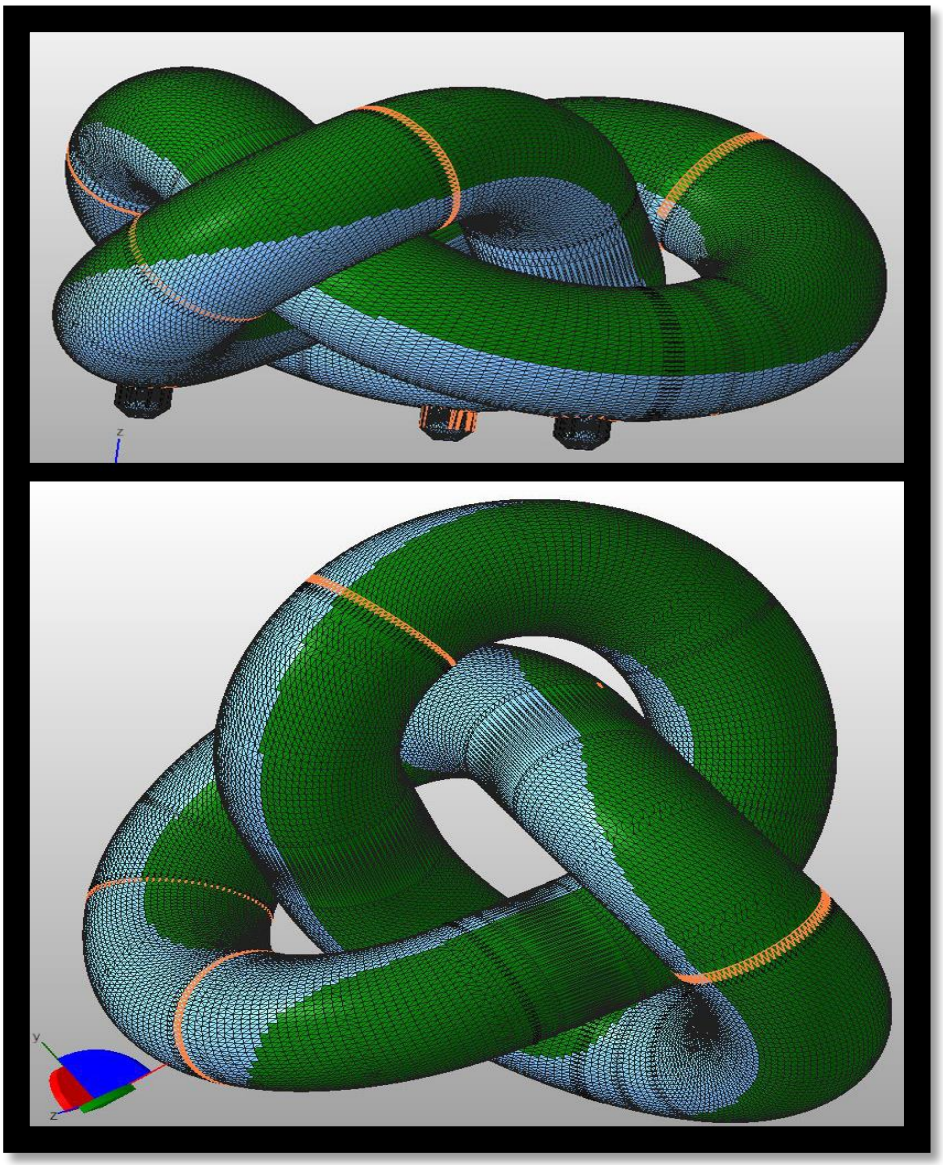


Figura 14 – knot

## 4.3. Figura de entorno real

En este caso se ha optado por realizar las pruebas con una malla cedida por Juan Ramón Vázquez García - Graphic Designer (Ref, 14) un reconocido diseñador de fallas, que trabaja con los artistas de más reconocidos. En este caso, hablamos de la figura de una perrita bailarina que fue plantada hace unos años en una falla infantil. Dicha malla está compuesta por 130.635 triángulos, debido a la gran cantidad de detalle.

La diferencia con la figura compleja, radica, aparte de una mayor cantidad de triángulos, en que consta de muchos más recovecos y detalles, o que genera muchos más oscuros.


Figura 15 –perrita

## 4.4. Conclusiones

A continuación se procede a exponer algunas de las pruebas realizadas en cada una de las etapas de estos desarrollos, basándose esta explicación en el orden cronológico de desarrollo de los algoritmos y utilizando los tres tipos de mallas comentadas con anterioridad. Empezando por la *"Anulación de oscuros"*, la que además de servir para la familiarización con las mallas 3D, sus triángulos, sus normales, etc..., nos ha aportado una visión espacial diferente y nos ha acercado a poder pensar en tres dimensiones. A continuación se muestran los resultados de este algoritmo para los tres tipos de figuras:

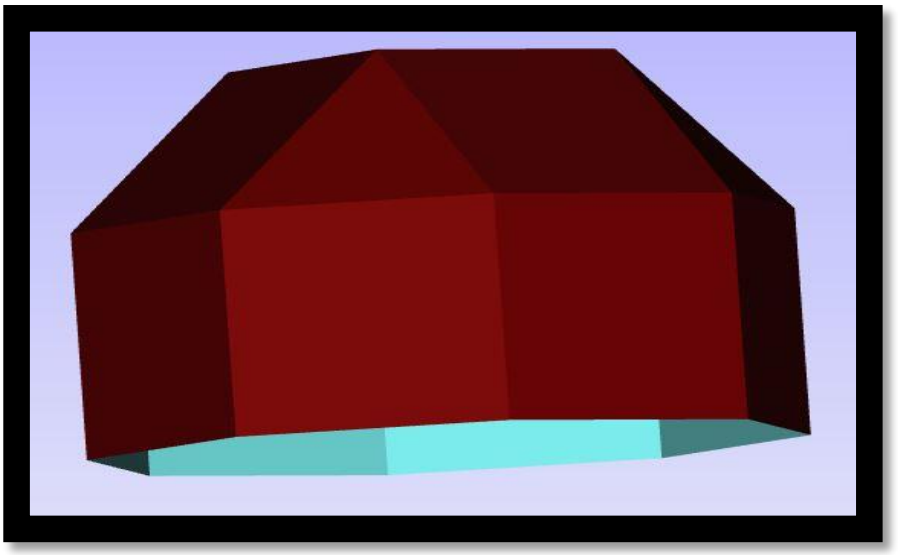


Figura 16 –Prueba sin oscuros, figura simple

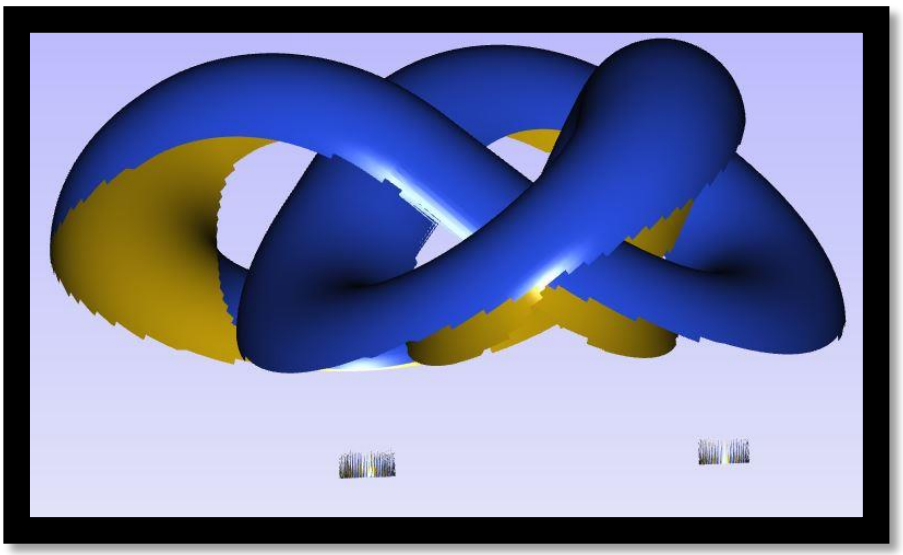


Figura 17 –Prueba sin oscuros, figura compleja

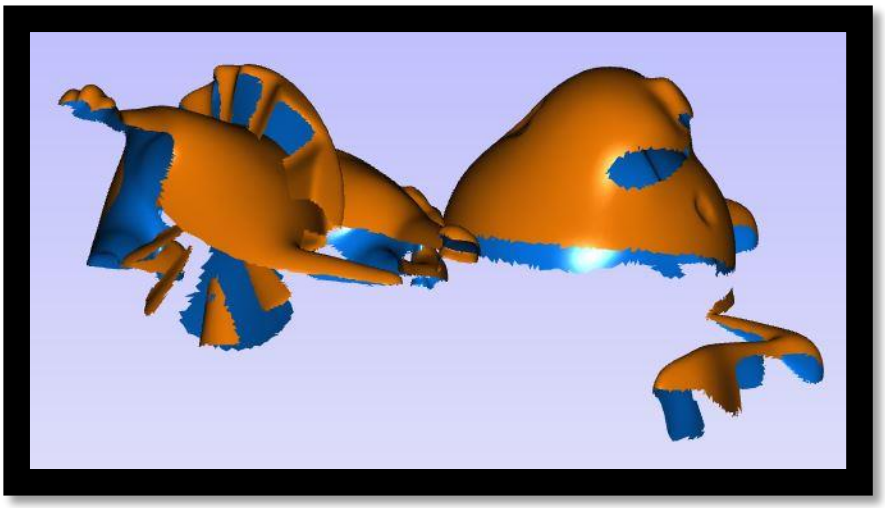


Figura 18 –Prueba sin oscuros, figura real

Basándonos en los resultados obtenidos, podemos decir que el primer código se comporta de la misma manera independientemente de la complejidad de la malla, tardando más o menos su ejecución dependiendo del número de triángulos, pero no hay más diferencia. Debido a la simplicidad del algoritmo, realmente este no sirve más que como toma de contacto con el trabajo con mallas 3D, ya que a este programa, en este estado, le faltan muchas cosas que cumplir, entre otras, que no se demuestra el total del STL original sino simplemente una parte. Ello nos ha dirigido a mostrar la parte que nos falta implementado en el código escrito en *"Reorientación de oscuros",* resultados que se muestran a continuación:

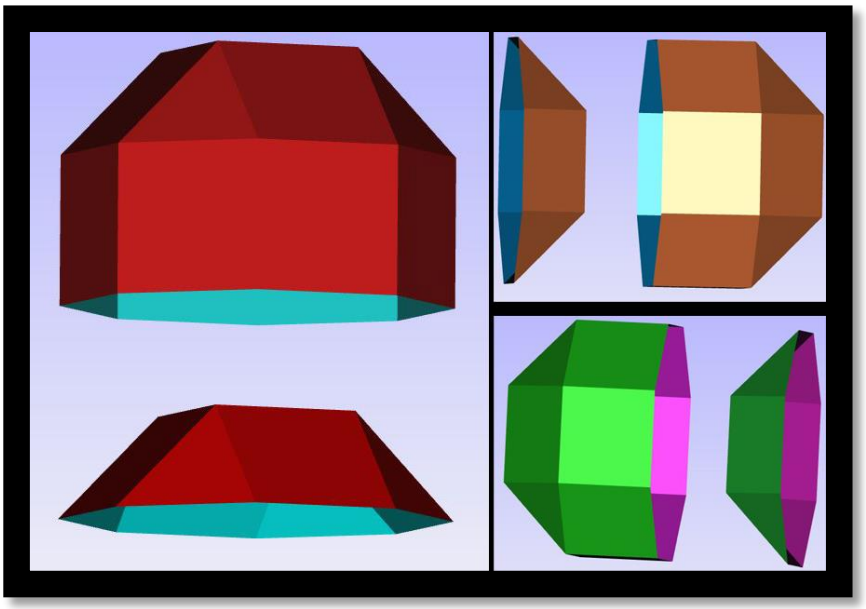


Figura 19 –Reorientación de oscuros, figura simple

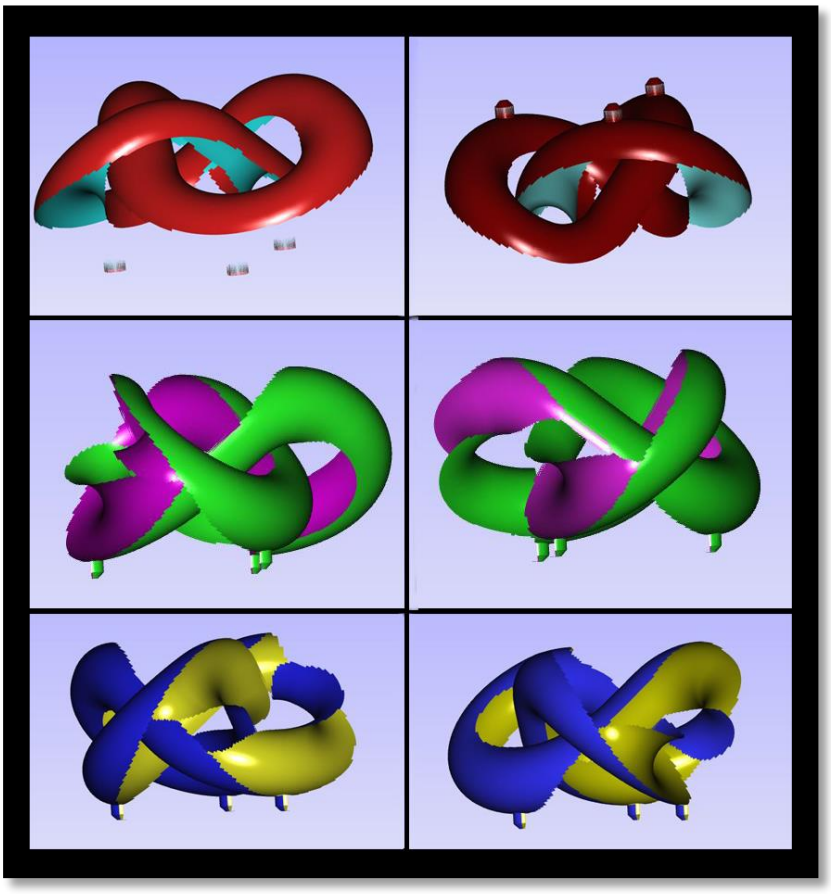


Figura 20 –Reorientación de oscuros, figura complejo

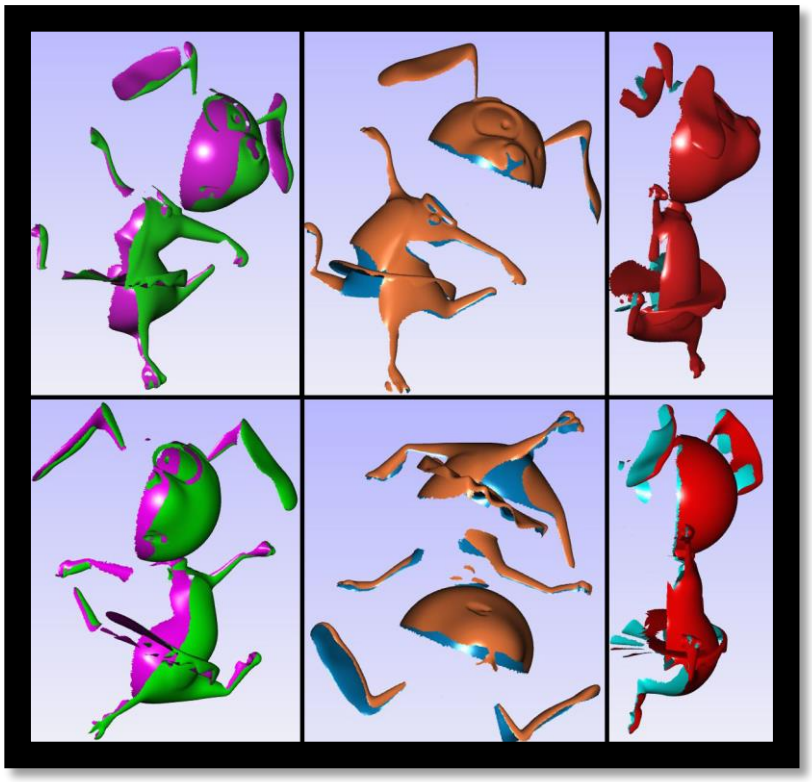


Figura 21 –Reorientación de oscuros, figura real

Como se puede observar en los resultados mostrados con anterioridad, en figuras simétricas y simples no afecta cual se usa de eje de mecanizado. En el caso de que hablemos de los tres ejes de coordenadas “x”, “y”, “z”, se comporta de la misma manera en todos sus casos. En cambio, cuando observamos las figuras complejas o las de entorno real, sí se ve claramente que no son los mismos resultados dependiendo del eje de mecanizado que estemos hablando, probablemente en la figura compleja, fuera el resultado más útil el mecanizado desde el eje z y en la figura real, el mecanizado desde el eje x, las coloreadas roja y azul de la Figura 20 y Figura 21 respectivamente.

Tras estudiar estos resultados, se llega a la conclusión que para poder valorar realmente cual es la mejor decisión del eje de mecanizado, estas seis muestras son escasas y sería más interesante si pudiéramos valorar más cantidad de resultados, lo cual llevó a la implementación de *“Girado de pieza original reorientando oscuros”,* donde como resultado genera muchísimas más muestras de las 6 escasas que nos aportaba el algoritmo anterior. En este caso, nos muestra seis diferentes archivos en cada una de las posiciones que se coloca el archivo original. En nuestra demo va girando en los tres ejes de coordenadas de diez en diez grados, lo que nos generará tres ejes por 36 posiciones en cada eje por  archivos en cada posición, un total de 648 archivos por figura, lo que a priori parece ya una cantidad más cercana a las necesidades de afinado:

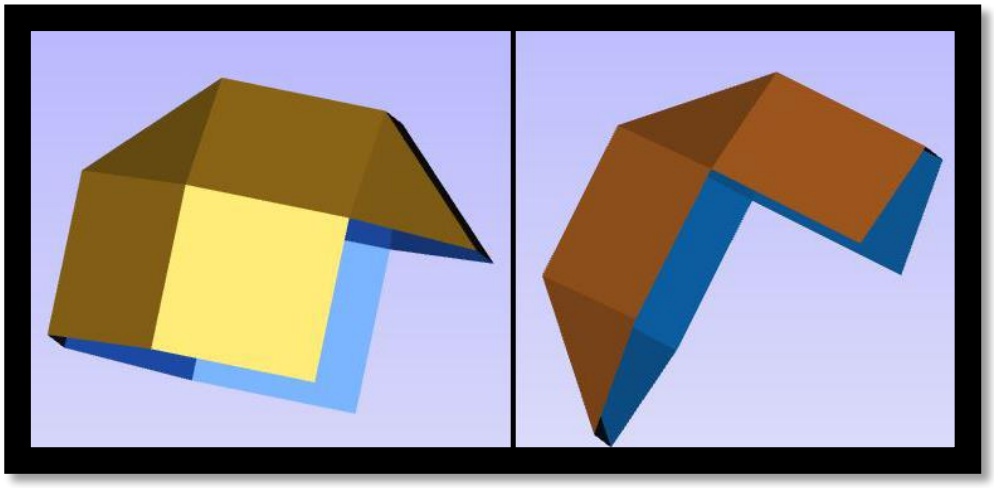


Figura 22 –Ejemplo del resultado de un giro fuera de coordenadas x, y, z con figura simple

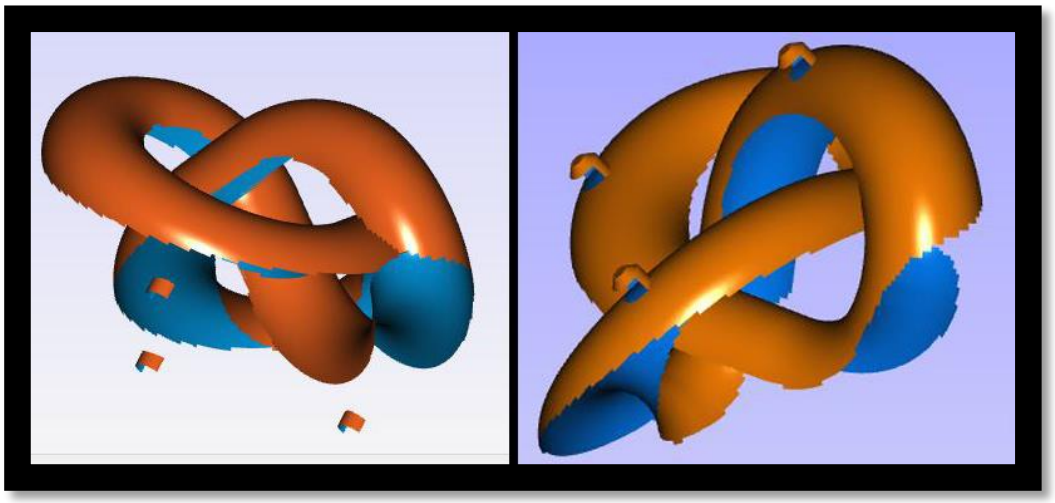


Figura 23 –Ejemplo del resultado de un giro fuera de coordenadas x, y, z con figura compleja

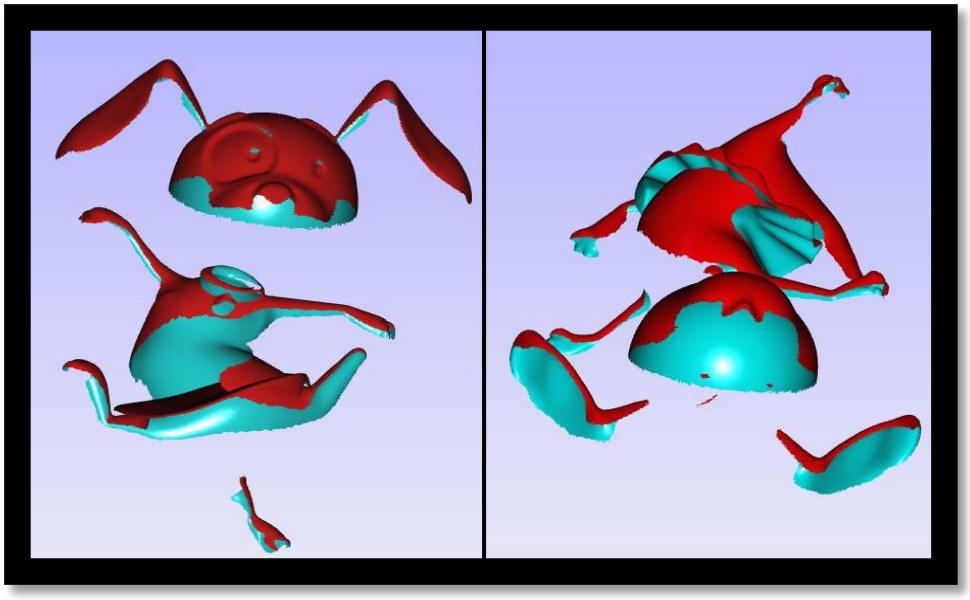


Figura 24 –Ejemplo del resultado de un giro fuera de coordenadas x, y, z con figura real

Observando los resultados de las pruebas, se ve con claridad que en el caso de las figuras tan simples como la figura de ejemplo que estamos utilizando, Rhombicuboctahedron, se acaban repitiendo las diferentes mallas resultado, lo que genera muchas operaciones para los mismos resultados. En cambio sí existe variabilidad entre las mallas resultado, según va aumentando la complejidad de las figuras originales.

Llegados a este punto se puede observar que, tal vez, la línea de investigación que se está manteniendo, no sea en la dirección apropiada, ya que no separa en piezas diferentes los triángulos que están separados físicamente, por ello se toma la decisión de intentar alguna variación. En este caso, intentar separar las mallas por oscuros como estábamos haciendo y también en diferentes mallas los triángulos que, no siendo oscuros entre ellos, tampoco comparten ningún vértice, lo que nos dice que no son contiguos. Esto da paso a *"triángulos contiguos sin oscuros"*.

Esta idea se ha desarrollado en dos vertientes distintas dependiendo cual será el eje de mecanizado de la pieza resultado. Una de las ideas es tomar como referencia la normal del primer triangulo que se pinta en la malla resultado concreta, la cual se crea en ese preciso momento; para cada malla resultado existirá por tanto un primer triangulo. En cambio la otra prueba es utilizar como eje de mecanizado la media de las normales de los triángulos que aún no han sido añadidos en ninguna malla resultado- Esto parece en principio una opción más interesante, ya que al utilizar la media y no elegir un triángulo concreto, se adaptará mejor a la idea de buscar el mecanizado óptimo en cuanto a número de piezas, y la optimización del posterior montaje. Pongamos el caso de media esfera, si seleccionamos como eje de mecanizado la normal de un triángulo que no sea el punto más alto de la misma, esa media esfera acabará seguro separada como mínimo en dos partes. En cambio si el eje de mecanizado es el punto más alto, se mecanizará de una sola pieza. Este es el motivo de intentar buscar la media de las normales o algún otro método que no sea el primer triangulo que venga escrito en el fichero STL.

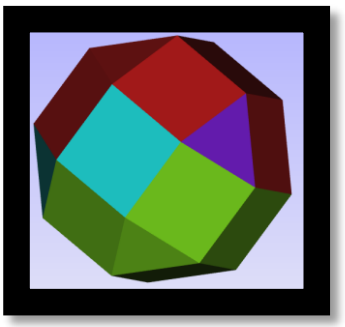


Figura 25 – Figura simple, eje de mecanizado normal del primer triangulo

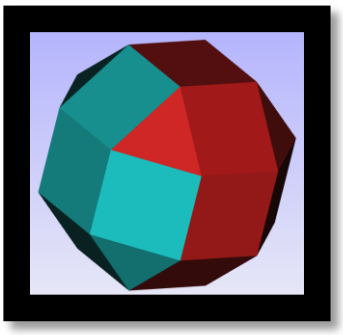


Figura 26 – Figura simple, eje de mecanizado media de las normales.

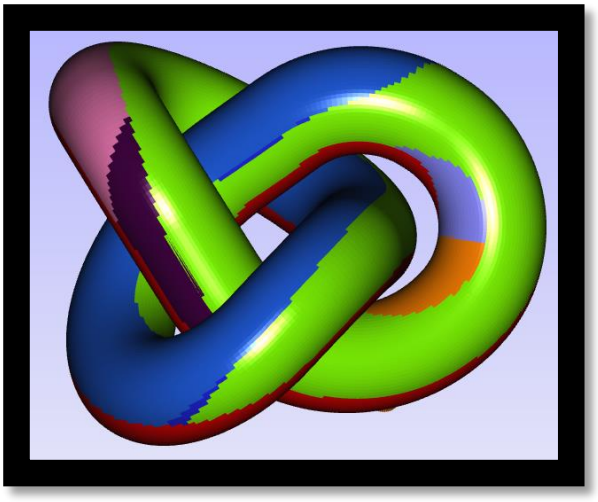


Figura 27 – Figura compleja, eje de mecanizado normal del primer triangulo.

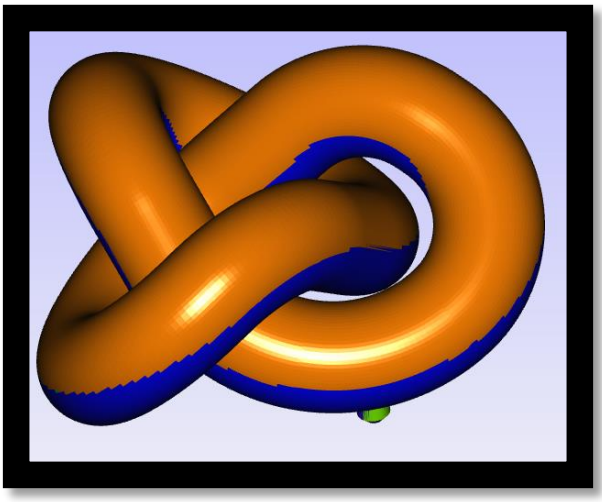


Figura 28 – Figura compleja, eje de mecanizado media de las normales.

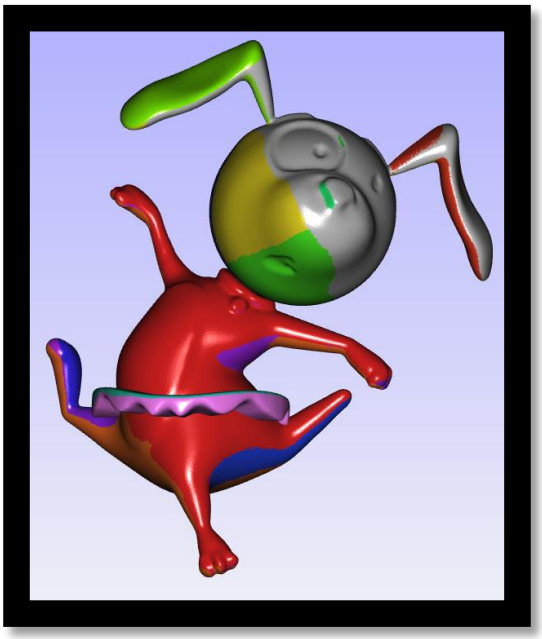


Figura 29 –Figura compleja, eje de mecanizado normal del primer triangulo.

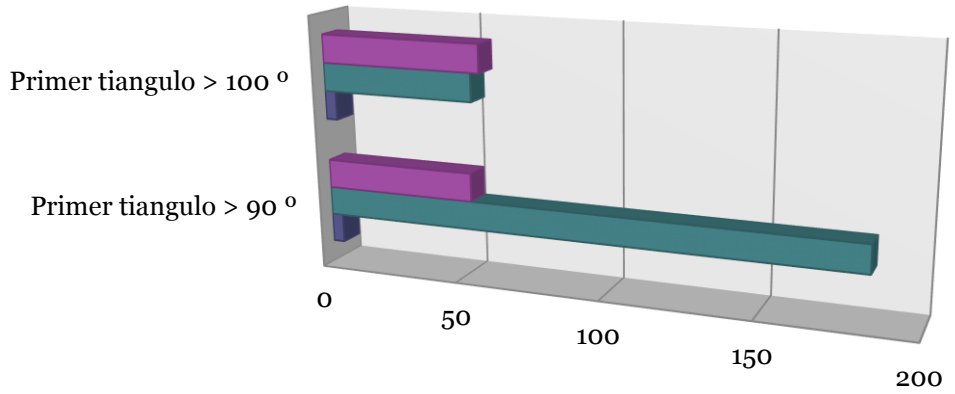
Figura 30 – Figura real, eje de mecanizado media de las normales.

En cambio, como se puede observar en los resultados gráficos que se adjuntan en las figuras anteriores, se generan muchos más archivos en entorno real, y a su vez, la relación de los archivos creados menores de cinco triángulos por archivo es muy superior cuando se aplica la media de normales como eje de mecanizado, comparándolo a cuando es la normal del primer triangulo a añadir, quien marca el eje de mecanizado. Por ejemplo: la figura de entorno real en el caso de aplicar la media de normales, genera 10.774 archivos, de los cuales 10.750 contienen menos de cinco triángulos. En cambio, con la técnica del primer triangulo, se generan 53 archivos, de los cuales 16 contienen menos de cinco triángulos.

Por este motivo, se cree que la línea de investigación y pruebas ha de continuar trabajando con los ángulos diferentes que se usan para determinar que es un oscuro o no, utilizando como eje de mecanizado la normal del primer triángulo, o de un triángulo fijo y no la media o cálculos sobre las normales de los triángulos que quedan por añadir, ya que en realidad en el mundo de las fallas, existe una cantidad muy superior de figuras de complejidad como nuestra "perrita bailarina", que figuras simples como media esfera o nuestro Rhombicuboctahedron.

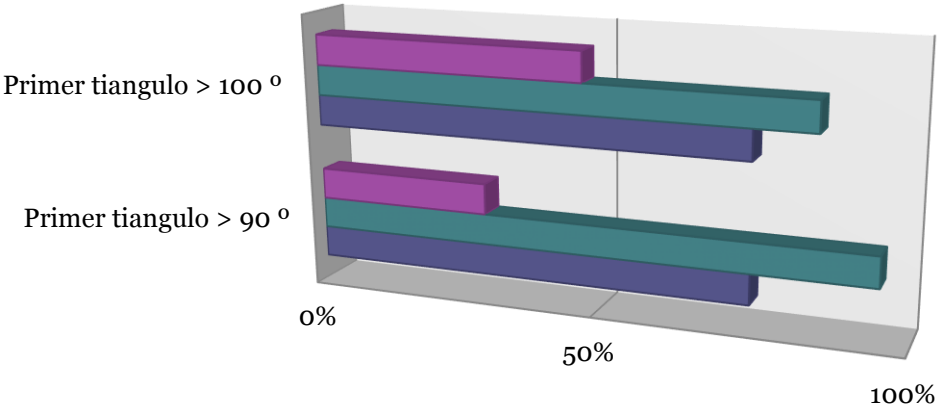
Continuando en la línea de un triángulo fijo como referencia, en este caso el primer triángulo que nos da el fichero original, se realizan dos pruebas diferentes para cada una de las figuras, una determinando como oscuro, cuando el eje de mecanizado genera con la normal del triángulo a valorar un ángulo de 90 grados, y la otra, siendo dicho ángulo de 100 grados. Es de suponer que optando por la opción de 100 grados, existirán zonas donde nuestra máquina herramienta no podrá acceder, lo que generará un trabajo artesano posterior de lijado, pero tal vez compense respecto al aprovechamiento del poliestileno. Los resultados en cuanto a cantidad de oscuros y número de archivos pequeños, se muestran a continuación:

## Numero de archivos STL resultado




| | Primer tiangulo > 90 º | Primer tiangulo > 100 º |
|---|---|---|
| Real | 53 | 57 |
| Compleja | 184 | 54 |
| Simple | 4 | 4 |

## Porcentaje archivos de menos de 5 triangulos




| | Primer tiangulo > 90 º | Primer tiangulo > 100 º |
|---|---|---|
| Real | 30% | 47% |
| Compleja | 95% | 85% |
| Simple | 75% | 75% |

Observando estas gráficas podemos determinar que la idea lógica que se preveía, es que aumentando el ángulo que determina el oscuro se reducirían el porcentaje de archivos pequeños, para así lograr aprovechar material a costa de tener que realizar más proceso de lijado posterior, aunque no en todos los casos se cumple. Pero también es valorable que un alto porcentaje de piezas de falla son más similares a las que hemos utilizado como malla de entorno real. Por ello, es comprensible que esto se cumpla para un alto porcentaje de figuras.

También es interesante añadir que tras todas las pruebas realizadas, puede ser fundamental cómo escoger y qué triángulo escoger como primero, ya que dependiendo el triángulo a escoger que decida el eje de mecanizado, los resultados pueden ser muy variables. Una idea que surge una vez ya finalizadas las pruebas de este proyecto, es continuar la investigación trabajando con los triángulos más alejados entre ellos, o por su contra los más próximos, utilizando la distancia entre los puntos centrales de los triángulos para determinar cuáles de ellos son los más próximos en línea recta entre sí o los más alejados.

# 5. Valoración personal y Ampliaciones

## 5.1. Valoración personal

La realización de este proyecto ha contribuido a la introducción al mundo del 3D, las mallas, los archivos STL, al igual que afianzar conocimientos de programación, geometría y algoritmia para poder desarrollar los algoritmos que se han ido pensando y llevando a cabo.

Además, de formar parte del proceso de investigación de un proyecto en el que se está estudiando un problema NP-Hard, lo que es realmente fascinante y a su vez divertido y gratificante.

Por último, la sensación de no saber qué iba a ocurrir en sucesivas etapas del proyecto, ya que se trata de un problema no resuelto por nadie conocido y ha sido un reto interesante a la vez que difícil, debido a esa incertidumbre.

## 5.2. Ampliaciones

Esta línea de investigación está totalmente abierta, siendo este proyecto un buen punto de partida para su posible continuación, ya sea decidiendo cual puede ser el primer triángulo, o con nuevas ideas que puedan dar un poco más de luz al problema, o tal vez continuando con las ideas expuestas al final de las conclusiones que no se han llegado a desarrollar.

A su vez, se puede ampliar este proyecto unificando los algoritmos de interés que se han desarrollado aquí para lograr resultados más afinados.

Y como no, en el caso de lograr una lógica del programa totalmente funcional, siempre existe la posibilidad de continuar el proyecto desarrollando un front-end para el usuario final.

# 6. Referencias

A continuación se detallan las referencias utilizadas, para la realización del proyecto y la memoria.


1. Siemens Global Website, CAM / Computer-Aided Manufacturing / Fabricación asistida por ordenador: http://www.plm.automation.siemens.com/es_es/plm/cam.shtml
2. Siemens Global Website, CAM / Computer-Aided Manufacturing / Diseño asistido por ordenador: http://www.plm.automation.siemens.com/es_es/plm/cad.shtml
3. R3ALD, ¿Qué es un fichero STL? http://www.r3ald.com/que-es-un-fichero-stl
4. Oliva, S., Usar Processing en Java: http://www.sebastianoliva.com/2010/03/usar-processing-en-java/trackback/
5. Netfabb Software web: http://www.netfabb.com
6. SuperSkein OpenSource Software
7. Processing Software web: https://processing.org/
8. Wikipedia en inglés: https://en.wikipedia.org
9. Wikipedia en español: https://es.wikipedia.org
10. STLView web: http://www.freestlview.com/
11. Java Plataform Standard Edition 7 Documentation: http://docs.oracle.com/javase/7/docs/index.html
12. Java Plataform Standard Edition 7 API: http://docs.oracle.com/javase/7/docs/api/
13. Pérez Camps, Maquinas de grabado y corte laser, fresadoras CNC y Pantógrafos: http://www.perezcamps.com/

14. Vazquez García, J R, Repositorio y pagína de Facebook de Juan Ramón Vazquez Garcíahttps: www.facebook.com/pages/Juan-Ram%C3%B3n-V%C3%A1zquez-Garc%C3%ADa-Graphic-Designer/670499103015701?sk=info
15. Interactive Mathematics http://www.intmath.com/

# Apéndices

En este apartado se va a documentar el código de los diferentes algoritmos iniciando siempre desde el más básico y ampliando en orden cronológico el resto de clases.

## Apéndice I: Código - Anulación de oscuros

Código de la clase mesh:

```
import java.awt.geom.*;
import java.io.*;

class Mesh {
  ArrayList Triangles;
  boolean Valid;

  Mesh(String FileName)
  {
    Valid = false;

    println ("Cargando archivo " + FileName + "...");
    Triangles = new ArrayList();

    if (LoadTextMesh (FileName))
    {
      Valid = true;

    }
    else if (LoadBinaryMesh (FileName))
    {
      Valid = true;

    }else{ System.out.println("Archivo no valido");}
  }

  private boolean LoadBinaryMesh (String FileName)
  {
    try
    {
      byte b[] = loadBytes(FileName);
      float[] Tri = new float[12];

      int offs = 84;

      while(offs<b.length){

        for(int i = 0; i<12; i++)
        {
          Tri[i]=Float.intBitsToFloat( b[offs] &0xff
          | (b[offs+1]<<8)&0xff00 | (b[offs+2]<<16)&0xff0000 |
          (b[offs+3]<<24)&0xff000000);

          offs=offs+4;
        }
        offs=offs+2;
        Triangles.add(new Triangle(Tri[3],Tri[4],Tri[5],Tri[6],Tri[7],
        Tri[8],Tri[9],Tri[10],Tri[11],Tri[0],Tri[1],Tri[2]));
      }
    }
    catch (Exception e)
    {
      return false;
    }

    return true;
  }
```

```
  private boolean LoadTextMesh (String path)
  {


    BufferedReader reader;
    String buf;

    try
    {
      reader = createReader (path);

      buf = reader.readLine();
      if (buf.indexOf ("solid") != 0)
      {

        return false;
      }
    }
    catch (RuntimeException ex)
    {

      return false;
    }
    catch (Exception e)
    {

      return false;
    }

    try
    {
      while (true)
      {
        buf = reader.readLine();
        if (buf == null || buf.indexOf ("endsolid") == 0)
        {

          break;
        }

        if (buf.indexOf ("facet normal") == -1)
         {

          return false;
        }
        String[] floats;
        int offset;
        //Lectura de las normales
        floats = buf.split("[\\s,;]+");
        offset = floats.length == 6 ? 2 : 0;
        float xn = Float.parseFloat (floats[1 + offset]);
        float yn = Float.parseFloat (floats[2 + offset]);
        float zn = Float.parseFloat (floats[3 + offset]);
        buf = reader.readLine();  // "    outer loop"
        //Lectura del primer punto del triangulo.
        floats = reader.readLine().split("[\\s,;]+");
        offset = floats.length == 5 ? 1 : 0;
        float x1 = Float.parseFloat (floats[1 + offset]);
        float y1 = Float.parseFloat (floats[2 + offset]);
        float z1 = Float.parseFloat (floats[3 + offset]);
        //Lectura del segundo punto del triangulo.
        floats = reader.readLine().split("[\\s,;]+");
        offset = floats.length == 5 ? 1 : 0;
        float x2 = Float.parseFloat (floats[1 + offset]);
        float y2 = Float.parseFloat (floats[2 + offset]);
        float z2 = Float.parseFloat (floats[3 + offset]);
        //Lectura del tercer punto del triangulo.
        floats = reader.readLine().split("[\\s,;]+");
        offset = floats.length == 5 ? 1 : 0;
        float x3 = Float.parseFloat (floats[1 + offset]);
        float y3 = Float.parseFloat (floats[2 + offset]);
        float z3 = Float.parseFloat (floats[3 + offset]);

        Triangles.add (new Triangle (x1, y1, z1, x2, y2, z2,
        x3, y3, z3, xn, yn, zn));

        reader.readLine(); // "   endloop"
        reader.readLine(); // "  endfacet"
      }
    }
    catch (Exception e)
    {
      return false;
    }
    return true;
  }
}
```

```
String archivo = "marciano3";
Mesh m=null;
int trisin =0;
int trisout = 0;

void setup() {
   m= new Mesh("../"+archivo+".stl");
   System.out.println("Triangulos="+m.Triangles.size());
   create();
   System.out.println("Se han añadido a la malla nueva:"
   + trisin + " Triangulos");
   System.out.println("Se han omitido en la malla nueva: "
   + trisout + " Triangulos");
}

void create(){
   FileWriter fichero = null;
        PrintWriter pw = null;
        try
        {

            fichero = new FileWriter(
            "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/CADCAM/Trabajo2/salidas/"
            +archivo+"_sal.stl");
            System.out.println("Se ha iniciado la creacion del fichero "
            + archivo + "_sal.stl");
            pw = new PrintWriter(fichero);
            pw.println("solid " + archivo);
            for(int i = m.Triangles.size()-1;i>=0;i--){
              Triangle tri = (Triangle) m.Triangles.get(i);
              if(tri.getnz()>=0){
                trisin ++;
                pw.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                pw.println("outer loop");
                pw.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                pw.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                pw.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                pw.println("endloop");
                pw.println("endfacet");
              }else{trisout++;}

            }
            pw.println("endsolid" + archivo);
            System.out.println("Ha finalizado la creacion del fichero " + archivo + "_sal.stl");

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
           try {
           // Aprovechamos el finally para
           // asegurarnos que se cierra el fichero.
           if (null != fichero)
              fichero.close();
           } catch (Exception e2) {
              e2.printStackTrace();
           }
        }
   }
```

Código de la clase Triangle:

```
class Triangle {

  float x1,x2,x3,y1,y2,y3,z1,z2,z3,xn,yn,zn,xnor,ynor,znor;

  Triangle( float tX1, float tY1, float tZ1,float tX2, float tY2, float tZ2,
  float tX3, float tY3, float tZ3 , float tXn, float tYn, float tZn) {
    xnor = tXn;
    ynor = tYn;
    znor = tZn;
    x1 = tX1;
    y1 = tY1;
    z1 = tZ1;
    x2 = tX2;
    y2 = tY2;
    z2 = tZ2;
    x3 = tX3;
    y3 = tY3;
    z3 = tZ3;

  }


  float getnx() {return xnor;}
  float getny() {return ynor;}
  float getnz() {return znor;}
  float getx1() {return x1;}
  float gety1() {return y1;}
  float getz1() {return z1;}
  float getx2() {return x2;}
  float gety2() {return y2;}
  float getz2() {return z2;}
  float getx3() {return x3;}
  float gety3() {return y3;}
  float getz3() {return z3;}

}
```

## Apéndice II: Código - Reorientación de oscuros

En este apéndice se muestran los métodos modificados, de la clase mesh, respecto al código de anulación de oscuros.

Método setup():

```
void setup() {
   m= new Mesh("../"+archivo+".stl");
   System.out.println("Triangulos="+m.Triangles.size());
   create();
   System.out.println("Se han añadido a la malla nueva: " + trisinz + " Triangulos");
   System.out.println("Se han omitido en la malla nueva: " + trisoutz + " Triangulos");
   System.out.println("Oscuros Arr->Aba " + trisoutz + " Triangulos");
   System.out.println("Oscuros Aba->Arr " + trisinz + " Triangulos");
   System.out.println("Oscuros Ya " + trisouty + " Triangulos");
   System.out.println("Oscuros Yb " + trisiny + " Triangulos");
   System.out.println("Oscuros Xa " + trisoutx + " Triangulos");
   System.out.println("Oscuros Xb " + trisinz + " Triangulos");

}
```

Método create():

```
void create(){
     FileWriter fichero = null;
     FileWriter fichero2 = null;
     FileWriter ficheroy = null;
     FileWriter ficherox = null;
     FileWriter ficheroy2 = null;
     FileWriter ficherox2 = null;
     /*File ficheror = null;
     File ficherorx = null;
     File ficherory = null;*/
       PrintWriter pw = null;
       PrintWriter pw2 = null;
       PrintWriter pwy = null;
       PrintWriter pwx = null;
       PrintWriter pwy2 = null;
       PrintWriter pwx2 = null;

       try
       {
           fichero = new FileWriter(
           "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+"_sal.stl");
           fichero2 = new FileWriter(
           "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+"2_sal.stl");
           ficheroy = new FileWriter(
           "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+"y_sal.stl");
           ficherox = new FileWriter(
           "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+"x_sal.stl");
           ficheroy2 = new FileWriter(
           "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+"y2_sal.stl");
           ficherox2 = new FileWriter(
           "D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+"x2_sal.stl");

           pw = new PrintWriter(fichero);
           pw.println("solid " + archivo);
           pw2 = new PrintWriter(fichero2);
           pw2.println("solid " + archivo + "2");
           pwy = new PrintWriter(ficheroy);
           pwy.println("solid " + archivo+ "y");
           pwx = new PrintWriter(ficherox);
           pwx.println("solid " + archivo + "x");
           pwy2 = new PrintWriter(ficheroy2);
           pwy2.println("solid " + archivo+ "y2");
           pwx2 = new PrintWriter(ficherox2);
           pwx2.println("solid " + archivo + "x2");
```

```
            for(int i = m.Triangles.size()-1;i>=0;i--){
              Triangle tri = (Triangle) m.Triangles.get(i);
              if(tri.getnz()>=0){
                trisinz ++;
                pw.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                pw.println("outer loop");
                pw.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                pw.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                pw.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                pw.println("endloop");
                pw.println("endfacet");

              }else{
                trisoutz ++;
                pw2.println("facet normal " + tri.getnx()+" "+tri.getny() +" "+tri.getnz()* -1);
                pw2.println("outer loop");
                pw2.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1() * -1);
                pw2.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2() * -1);
                pw2.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3() * -1);
                pw2.println("endloop");
                pw2.println("endfacet");
              }

              if(tri.getny()>=0){
                trisiny ++;
                pwy.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                pwy.println("outer loop");
                pwy.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                pwy.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                pwy.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                pwy.println("endloop");
                pwy.println("endfacet");
              }else{
                trisouty ++;
                pwy2.println("facet normal " + tri.getnx()+" "+tri.getny()* -1+" "+tri.getnz());
                pwy2.println("outer loop");
                pwy2.println("vertex " + tri.getx1()+" "+tri.gety1()* -1+" "+tri.getz1());
                pwy2.println("vertex " + tri.getx2()+" "+tri.gety2()* -1+" "+tri.getz2());
                pwy2.println("vertex " + tri.getx3()+" "+tri.gety3()* -1+" "+tri.getz3());
                pwy2.println("endloop");
                pwy2.println("endfacet");
              }
              if(tri.getnx()>=0){
                trisinx ++;
                pwx.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                pwx.println("outer loop");
                pwx.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                pwx.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                pwx.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                pwx.println("endloop");
                pwx.println("endfacet");
              }else{
                trisoutx ++;
                pwx2.println("facet normal " + tri.getnx() * -1  +" "+tri.getny()+" "+tri.getnz());
                pwx2.println("outer loop");
                pwx2.println("vertex " + tri.getx1()* -1+" "+tri.gety1()+" "+tri.getz1());
                pwx2.println("vertex " + tri.getx2()* -1+" "+tri.gety2()+" "+tri.getz2());
                pwx2.println("vertex " + tri.getx3()* -1+" "+tri.gety3()+" "+tri.getz3());
                pwx2.println("endloop");
                pwx2.println("endfacet");
              }
            }
```

```
            pw.println("endsolid" + archivo);
            pw2.println("endsolid" + archivo+ "2");
            pwy.println("endsolid" + archivo+ "y");
            pwx.println("endsolid" + archivo+ "x");
            pwy2.println("endsolid" + archivo+ "y2");
            pwx2.println("endsolid" + archivo+ "x2");

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
           try {
           // Aprovechamos el finally para
           // asegurarnos que se cierra el fichero y eliminar lo que no sirve
           if (null != fichero)
              fichero.close();
              fichero2.close();
              ficheroy.close();
              ficherox.close();
              ficheroy2.close();
              ficherox2.close();

           } catch (Exception e2) {
              e2.printStackTrace();
           }
        }
    }
```

## Apéndice III: Código - Girado de pieza original reorientando oscuros

En este apéndice se muestran los métodos modificados y añadidas, de la clase mesh, respecto al código de reorientación de oscuros.

Método setup():

```
void setup() {
    m= new Mesh("../"+archivo+".stl");
    System.out.println("Triangulos="+m.Triangles.size());
    create();
    giro = "gx";
    System.out.println("Empieza gira X");
    for(int i = 0; i < 20; i++){
        contarch++;
        rotadorx = rotadorx+10.0;
        rotatX(PI/rotadorx);
        create();
    }
     giro = "gy";
     contarch = 0;
     System.out.println("Empieza gira Y");
    for(int i = 0; i < 20; i++){
        contarch++;
        rotadory = rotadory+10.0;
        rotatY(PI/rotadory);
        create();
    }
     giro = "gz";
     contarch = 0;
     System.out.println("Empieza gira Z");
    for(int i = 0; i <= 20; i++){
        contarch++;
        rotadorz = rotadorz+10.0;
        rotatZ(PI/rotadorz);
        create();
    }
    System.out.println("Se han añadido a la malla nueva: " + trisinz + " Triangulos");
    System.out.println("Se han omitido en la malla nueva: " + trisoutz + " Triangulos");
    System.out.println("Oscuros Arr->Aba " + trisoutz + " Triangulos");
    System.out.println("Oscuros Aba->Arr " + trisinz + " Triangulos");
    System.out.println("Oscuros Ya " + trisouty + " Triangulos");
    System.out.println("Oscuros Yb " + trisiny + " Triangulos");
    System.out.println("Oscuros Xa " + trisoutx + " Triangulos");
    System.out.println("Oscuros Xb " + trisinz + " Triangulos");

}
```

Método s rotaX(), rotaY() y rotaZ():

```
void rotatX(float Angle)
  {

    for(int i = m.Triangles.size()-1;i>=0;i--)
    {
      Triangle tri = (Triangle) m.Triangles.get(i);
      tri.girax(Angle);
    }

  }

  void rotatY(float Angle)
  {
    //if(float.isNaN(Angle))return;
    for(int i = m.Triangles.size()-1;i>=0;i--)
    {
      Triangle tri = (Triangle) m.Triangles.get(i);
      tri.giray(Angle);
    }
  }

  void rotatZ(float Angle)
  {
    //if(float.isNaN(Angle))return;
    for(int i = m.Triangles.size()-1;i>=0;i--)
    {
      Triangle tri = (Triangle) m.Triangles.get(i);
      tri.giraz(Angle);
    }
  }
```

En este apartado se añaden tres métodos a la clase triangle, respecto al código de reorientación de oscuros.

Métodos giraz(), giray() y girax():

```
  void giraz(float Angle)
  {
    float xn,yn;
    xn = xnor*cos(Angle) - ynor*sin(Angle);
    yn = xnor*sin(Angle) + ynor*cos(Angle);
    xnor = xn;
    ynor = yn;
    xn = x1*cos(Angle) - y1*sin(Angle);
    yn = x1*sin(Angle) + y1*cos(Angle);
    x1 = xn;
    y1 = yn;
    xn = x2*cos(Angle) - y2*sin(Angle);
    yn = x2*sin(Angle) + y2*cos(Angle);
    x2 = xn;
    y2 = yn;
    xn = x3*cos(Angle) - y3*sin(Angle);
    yn = x3*sin(Angle) + y3*cos(Angle);
    x3 = xn;
    y3 = yn;
  }
```

```
void giray(float Angle)
{
  float xn,zn;
  xn = xnor*cos(Angle) - znor*sin(Angle);
  zn = xnor*sin(Angle) + znor*cos(Angle);
  xnor = xn;
  znor = zn;
  xn = x1*cos(Angle) - z1*sin(Angle);
  zn = x1*sin(Angle) + z1*cos(Angle);
  x1 = xn;
  z1 = zn;
  xn = x2*cos(Angle) - z2*sin(Angle);
  zn = x2*sin(Angle) + z2*cos(Angle);
  x2 = xn;
  z2 = zn;
  xn = x3*cos(Angle) - z3*sin(Angle);
  zn = x3*sin(Angle) + z3*cos(Angle);
  x3 = xn;
  z3 = zn;

}
void girax(float Angle)
{
  float yn,zn;
  yn = ynor*cos(Angle) - znor*sin(Angle);
  zn = ynor*sin(Angle) + znor*cos(Angle);
  ynor = yn;
  znor = zn;
  yn = y1*cos(Angle) - z1*sin(Angle);
  zn = y1*sin(Angle) + z1*cos(Angle);
  y1 = yn;
  z1 = zn;
  yn = y2*cos(Angle) - z2*sin(Angle);
  zn = y2*sin(Angle) + z2*cos(Angle);
  y2 = yn;
  z2 = zn;
  yn = y3*cos(Angle) - z3*sin(Angle);
  zn = y3*sin(Angle) + z3*cos(Angle);
  y3 = yn;
  z3 = zn;

}
```

## Apéndice IV: Código - Triángulos contiguos sin oscuros, normal de referencia primer triángulo.

A continuación se van a mostrar las clases desarrolladas para este algoritmo en concreto y las modificaciones realizadas en las clases existentes.

Clase Point:

```
class Point{
float x, y, z;


  Point(float px, float py, float pz){
      x = px;
      y = py;
      z = pz;
  }

  float getx(){return x;}
  float gety(){return y;}
  float getz(){return z;}
}
```

Constructor de la clase Triangle:

```
Triangle( float tX1, float tY1, float tZ1,float tX2, float tY2, float tZ2,float tX3, float tY3,
float tZ3 , float tXn, float tYn, float tZn, float tXc, float tYc, float tZc, boolean visit) {
  xnor = tXn;
  ynor = tYn;
  znor = tZn;
  x1 = tX1;
  y1 = tY1;
  z1 = tZ1;
  x2 = tX2;
  y2 = tY2;
  z2 = tZ2;
  x3 = tX3;
  y3 = tY3;
  z3 = tZ3;
  xc = tXc;
  yc = tYc;
  zc = tZc;
  visitado = visit;

}
```

Modificador y Consultor de la clase Triangle:

```
void setvisitado(boolean flagvis){
  visitado = flagvis;
}
boolean getvisitado(){return visitado;}
```

Método setup() de la clase mesh:

```
void setup() {
   m= new Mesh("../"+archivo+".stl");
   System.out.println("Triangulos="+m.Triangles.size());
   while(contvisit<m.Triangles.size()){
     primero = true;
     contarch++;
     create();
  }
  System.out.println("FINALIZADO");
}
```

Método create() de la clase mesh:

```
void create(){
     FileWriter fichero = null;

       PrintWriter pw = null;


       try
       {

           fichero = new FileWriter("
           D:/Documentos/DocumentosAndres/ITS_UPV/Año6/TFG/TFG/Primeraspruebas/Salidas/"+archivo+giro+contarch+"z_sal.stl");

           pw = new PrintWriter(fichero);
           pw.println("solid " + archivo+ "z");
            while(Puntos.peek() != null || primero){
              System.out.println("Entro al WHILE" + contvisit);

               if (primero == false){
                  Point p = (Point) Puntos.poll();
                  xp = p.getx();
                  yp = p.gety();
                  zp = p.getz();
               }
             for(int i = m.Triangles.size()-1;i>=0;i--){
              Triangle tri = (Triangle) m.Triangles.get(i);
              if(primero && tri.getvisitado() != true){
                xnp = tri.getnx();
                ynp = tri.getny();
                znp = tri.getnz();
                xp = tri.getx1();
                yp = tri.gety1();
                zp = tri.getz1();
                pw.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                pw.println("outer loop");
                pw.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                pw.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                pw.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                pw.println("endloop");
                pw.println("endfacet");
                primero = false;
                tri.setvisitado(true);
                contvisit++;
                Puntos.add(new Point(tri.getx2(),tri.gety2(),tri.getz2()));
                Puntos.add(new Point(tri.getx3(),tri.gety3(),tri.getz3()));
               // System.out.println("Montado el trianglo primero   " + tri.getvisitado());
```

```
            }else if (tri.getvisitado() != true && entreangulos(xnp, ynp, znp, tri.getnx(), tri.getny(), tri.getnz())  == false){
              if(xp == tri.getx1() && yp == tri.gety1() && zp == tri.getz1()){
                    pw.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                    pw.println("outer loop");
                    pw.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                    pw.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                    pw.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                    pw.println("endloop");
                    pw.println("endfacet");
                    tri.setvisitado(true);
                    contvisit++;
                    Puntos.add(new Point(tri.getx2(),tri.gety2(),tri.getz2()));
                    Puntos.add(new Point(tri.getx3(),tri.gety3(),tri.getz3()));
              }else if(xp == tri.getx2() && yp == tri.gety2() && zp == tri.getz2()){
                    pw.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                    pw.println("outer loop");
                    pw.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                    pw.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                    pw.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                    pw.println("endloop");
                    pw.println("endfacet");
                    tri.setvisitado(true);
                    contvisit++;
                    Puntos.add(new Point(tri.getx1(),tri.gety1(),tri.getz1()));
                    Puntos.add(new Point(tri.getx3(),tri.gety3(),tri.getz3()));
              }else if(xp == tri.getx3() && yp == tri.gety3() && zp == tri.getz3()){
                    pw.println("facet normal " + tri.getnx()+" "+tri.getny()+" "+tri.getnz());
                    pw.println("outer loop");
                    pw.println("vertex " + tri.getx1()+" "+tri.gety1()+" "+tri.getz1());
                    pw.println("vertex " + tri.getx2()+" "+tri.gety2()+" "+tri.getz2());
                    pw.println("vertex " + tri.getx3()+" "+tri.gety3()+" "+tri.getz3());
                    pw.println("endloop");
                    pw.println("endfacet");
                    tri.setvisitado(true);
                    contvisit++;
                    Puntos.add(new Point(tri.getx2(),tri.gety2(),tri.getz2()));
                    Puntos.add(new Point(tri.getx1(),tri.gety1(),tri.getz1()));
              }

             }

           }
         }
        pw.println("endsolid" + archivo+"z");

        System.out.println("Ha finalizado la creacion del fichero " + archivo + "_sal.stl");

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
       try {
       // Aprovechamos el finally para
       // asegurarnos que se cierra el fichero.
       if (null != fichero)
          fichero.close();

       } catch (Exception e2) {
          e2.printStackTrace();
       }
    }
}
```

## Apéndice V: Código - Triángulos contiguos sin oscuros, normal de referencia media de normales.

En este apartado se crean los métodos normalmediarestx(), normalmediaresty() y normalmediarestz(), que son las llamadas desde el método create():

```
float normalmediarestx(){
  float media = 0.0;
  int j = 0;
  for(int i = m.Triangles.size()-1;i>=0;i--){
       Triangle tri = (Triangle) m.Triangles.get(i);
       if(tri.getvisitado() != true){
         media=media+tri.getnx();
         j++;
       }
  }
  media = media/j;
  System.out.println("La media de las X es: " + media);
  return media;
}
float normalmediaresty(){
  float media =0.0;
  int j = 0;
  for(int i = m.Triangles.size()-1;i>=0;i--){
       Triangle tri = (Triangle) m.Triangles.get(i);
       if(tri.getvisitado() != true){
         media=media+tri.getny();
         j++;
       }
  }
   media = media/j;
  System.out.println("La media de las Y es: " + media);
  return media;
}
 float normalmediarestz(){
  float media = 0.0;
  int j = 0;
  for(int i = m.Triangles.size()-1;i>=0;i--){
       Triangle tri = (Triangle) m.Triangles.get(i);
       if(tri.getvisitado() != true){
         media=media+tri.getnz();
         j++;
       }
  }
  media = media/j;
  System.out.println("La media de las Z es: " + media);
  return media;
}
```