Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

VISUALIZACIÓN 3D DE IMÁGENES BIOMÉDICAS EN FORMATO .OBJ

Alumno: Ramos Alamos, Lander

Director: Espinosa Acereda, Jon Koldobika

Curso: 2017-2018

Fecha: Bilbao, 18, 07, 2018

Resumen trilingüe

Resumen

Este proyecto se basa en la realización de una herramienta de visualización 3D utilizando C como lenguaje de programación y las librerías openGL e ImageMagick. El proyecto permitirá, tras leer el fichero .obj, situar vértices en puntos específicos de un escenario 3D. La unión de estos puntos en bloques de 3 representará un triángulo; en bloques de cuatro, un cuadrado. La agrupación de múltiples polígonos traerá como consecuencia la representación de la silueta de la imagen deseada en 3D, a la que posteriormente se le insertarán las texturas. La representación conjunta de la silueta creada por los polígonos y las texturas pegadas en ella, tendrán como resultado la imagen 3D a color.

Laburpena

3Dko bistaratze-erraminta bat egitea da proiektu honen helburua. Horretarako, openGL eta ImageMagick liburutegiak eta C programazio lengoaia erabiliko dira. .obj fitxategia irakurri ondoren, proiektua 3Dko agertoki bateko puntu zehatzetan kokatzea baimentzen du. Puntu hauek hiru blokeetan lotzen dituenean, hirukiak sortzen dira. Lau blokeetan lotzen baldin baditu, aldiz, karratuak. Askotariko poligonoak batzean baldin badira, gura den irudiaren 3Dko silueta irudikatzea lortu ahal izango da. Geroago, silueta horiei ehundurak sartuko zaizkie. Poligonoek eta testurek elkarrekin sortzen duten azken irudia, koloreetan egongo den 3D ko irudia izango dute emaitza bezala.

Abstract

This project is based on the realization of a 3D visualization tool using C as a programming language and openGL and ImageMagick libraries. The project, after reading the .obj file, allows you to place vertices in specific points of a 3D scene. The union of these points into blocks of 3 will represent a triangle; in blocks of four, a square. The grouping of multiple polygons will bring as a consequence the representation of the silhouette of the desired image in 3D, to which later the textures will be insertedThe joint representation of the silhouette created by the polygons and the textures stuck on it, will result in the 3D color image.

Acrónimos

- 3D: tres dimensiones.
- 2D: dos dimensiones.
- JPG: Joint Photographic Experts Group.
- PNG: Portable Network Graphics.
- obj: Wavefront 3D Object File.
- OpenGL: Open Graphics Library.
- EEG: Electroencefalografía.
- MEG: Magnetoencefalografía.
- UE: Unión Europea.
- ICT: Tecnologías de la información y la comunicación.
- NURBS: Non-Uniform Rational B-spline.
- fbx: Autodesk FBX.
- max: 3DS Max.
- c4d: Cinema 4D.
- ma/mb: Maya.
- blend: Blender.
- unitypackage: Unity Game Engine.
- upk/uasset: Unreal Game Engine.
- dae: Collada.
- 3ds: 3D Studio.
- skp: SketchUp.
- lxo: Luxology Modo.
- lwo/lws: Lightwave.
- stl: StereoLithography.
- DARPA: Defense Advanced Research Project Agency.
- API: Interfaz de Programación de Aplicaciones.
- VTK: Kit de Herramientas de Visualización
- FFT: Fast Fourier Transform.
- OpenCV: Open Source Computer Vision Library
- CPU: Unidad Central de Procesamiento.
- GPU: Unidad de Procesamiento Gráfico.

Índice

Índice de tablas

Índice de figuras

1. Introducción

Este documento contiene la propuesta para un visualizador 3D de imágenes biomédicas realizado mediante lenguaje C y librerías OpenGL e ImageMagick; con el objetivo de poder representar cualquier imagen con formato .obj. Se le atribuye el nombre de visualizador 3D a aquellas herramientas que son capaces de leer archivos de imagen 3D y representar estas en un escenario de 3 dimensiones. Se denomina imágenes biomédicas como aquellas imágenes que representan el cuerpo humano o ciertas partes de éste, con el objetivo de poder usarlas en el ámbito de la medicina. Esto es una definición general, puesto que se debe mencionar que ciertas pruebas las cuales no tienen como resultado realmente una imagen, también se les denomina imágenes biomédicas, modelo de estas son la EEG o la MEG.

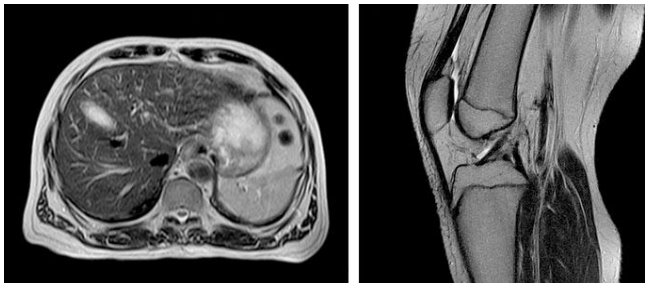


Figura 1: Imagen biomédica de hígado y rodilla

Se definirá el proyecto de manera detallada, explicando los diferentes puntos de éste; empezando por exponer el contexto por el que este proyecto se lleva a cabo, dejando claros los objetivos que serán concretados posteriormente, así como el alcance del proyecto. Se presentarán los diferentes beneficios, extraídos del contexto, que puede tener la realización de éste proyecto, en diversos ámbitos y en diversos aspectos, con el fin de justificar su implantación.

Se estudiarán las diferentes alternativas para cada parte del proyecto asociado, expuestas previamente en el apartado del estado del arte en el cual se empezarán a detallar los detalles técnicos, logrando un criterio selectivo de mayor eficiencia a la hora de realizar éste, utilizando como parámetros ciertos aspectos de cada solución.

A continuación, se presentará la propuesta, es decir la solución elegida para llevar a cabo el proyecto. Se explicará de forma puntualizada en qué consisten las librerías OpenGL e ImageMagick, y se definirá la estructura de los archivos con extensión .obj, los cuales son los utilizados por el visualizador de imágenes. El conocimiento de su estructura es imprescindible para entender el funcionamiento utilizado para la visualización. Se explicará la construcción de la imagen mediante el método elegido, así como los diferentes controles que se pueden utilizar para rotar la imagen en tres dimensiones.

Puesto que en la realización de cualquier proyecto que se precie la planificación y el presupuesto son dos de los factores más importantes de éste, se expondrán con el fin de

comprobar la rentabilidad de la solución. Finalmente, basándose en todos los puntos anteriores se expondrán las conclusiones obtenidas, explicando cada una de ellas.

2. Contexto

La simulación mediante ordenadores ha sufrido una evolución a lo largo de los años, ofreciendo información más detallada, precisa y fiable. Esto junto a la evolución sufrida por la medicina desde 1900, provocó que en la década de los 70 aparecieran las primeras simulaciones médicas gracias a la aparición de los microprocesadores en el mercado. Estas simulaciones se basaban en unas imágenes 2D y unos cálculos sencillos, puesto que en aquella época los ordenadores sufrían de una alta limitación en cuanto a potencia de cálculo se trataba. En la década de los 90, aparecieron las primeras tarjetas gráficas capaces de generar gráficas en 2D y 3D. En esta época se disponía de un microprocesador para realizar únicamente los cálculos matemáticos y el procesado de información, lo que permitió realizar modelos 3D de forma rápida y permitir la interacción con ellos, pudiendo rotar la imagen, hacerla zoom, etc. Desde esa época hasta hoy día las CPUs han aumentado las prestaciones y disminuido el espacio, y las GPUs han aumentado la potencia de cálculo. Esto ha tenido mucha repercusión en áreas tan importantes como la medicina, la docencia o la biología entre otras.

Los modelados computacionales aplicados a la medicina han aumentado en gran cantidad estos últimos años. Como se ha comentado anteriormente, antes solo se utilizaban modelos simples en 2D, es en estos últimos 15-20 años cuando estos modelados han empezado a tener importancia con motivo de la aparición de los modelados tridimensionales y la posibilidad de hacer simulaciones complejas. Con una visualización 3D se puede representar casi cualquier cosa, pudiendo realizar un estudio más detallado que de otra forma sería inviable.

Algunas nuevas tendencias a nivel internacional como *Physiome* o a nivel europeo como *Virtual Physiological Human* apostaron por la utilización de modelos ICT para el modelado del organismo humano de manera virtual. Existen simulaciones de múltiples órganos, y para cada uno de ellos con diferentes físicas. Por ejemplo, se pueden encontrar actualmente modelados de corazón con física eléctrica. También son de interés otras físicas como el estudio de la mecánica o de los fluidos de ciertos órganos. La mayoría de modelos actuales no suelen estar orientados a la globalidad de un órgano, sino que se suelen centrar en ciertos aspectos de éste. Hoy día, se habla también del modelado multiescalar el cual es capaz de modelar cada física en distintos niveles de resolución, pudiendo representar en una sola imagen 3D las células, los tejidos y el órgano al completo.

La tecnología actual de multicore y GPUs avanzadas permiten hacer simulaciones 3D muy realistas a nivel estructural, que requieren miles de millones de cálculos para generarlas, en tiempos aceptables (horas o días). Gracias las simulaciones logradas hoy día, es posible lograr que las operaciones actuales se puedan realizar con el mínimo riesgo posible para el paciente (teniendo en cuenta el alto nivel de dificultad de estas) debido a que el profesional puede ensayar la operación cuantas veces quiera mediante estas simulaciones. Se sabe que en

la realidad pueden producirse dificultades imprevistas que la simulación no contempla, pero al menos se parte de una idea más clara y completa de lo que se tenía en épocas pasadas [1].

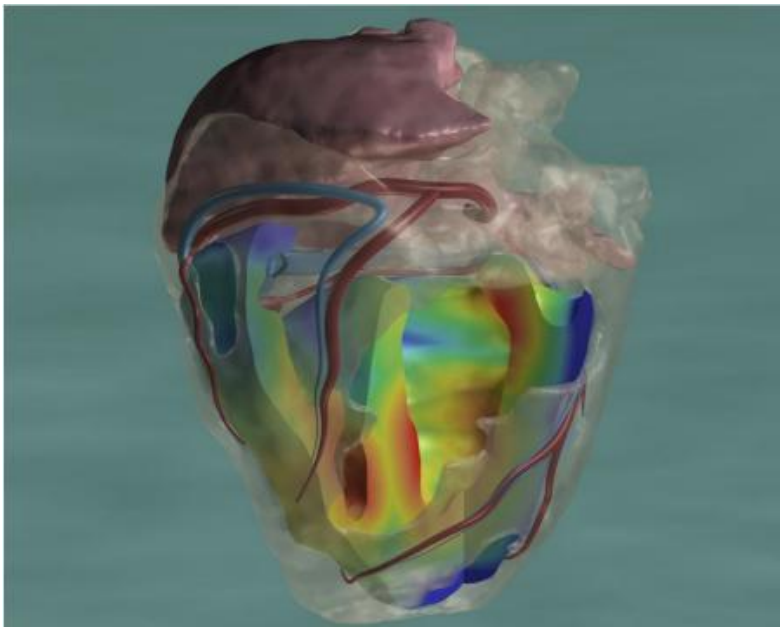


Figura 2: Modelado del corazón realizado con las tecnologías de hoy día

El ser humano tiene como objetivo en muchos ámbitos crear tecnologías que puedan representar imágenes que se asemejen lo máximo posible a la vida real; es por esto que cada vez son más comunes aquellas tecnologías que permiten mostrar imágenes en 3D. El cine o los videojuegos son un claro ejemplo de esto. En el cine, una investigación realizada por la *Universidad Goldsmitsh* y el *Laboratorio Thrill* obtuvo unos resultados significativos, los cuales demostraban que ver películas en 3D podría aumentar hasta un 23% la capacidad de procesamiento cognitivo [2]. Esta capacidad se relaciona con el procesamiento de la información, es decir, la atención, percepción, memoria, resolución de problemas, comprensión, establecimientos de analogías, etc. Volviendo al campo de la medicina, estos factores son de gran importancia a la hora de poder diagnosticar a un paciente mediante la visualización de imágenes biomédicas.

Muchas de las pruebas utilizadas en este ámbito, tienen como finalidad mostrar imágenes de las partes deseadas, pudiendo, de esta forma, descubrir en la zona analizada, enfermedades las cuales puede llegar a padecer el paciente [3]. Ejemplo de este tipo de pruebas son la resonancia magnética, los rayos X, la tomografía, el ultrasonido, la ecografía Doppler, etc. Además cada vez es más frecuente el uso de técnicas basadas en imagen durante una operación quirúrgica.

La docencia es también otro ámbito en el que la visualización de imágenes en 3D puede ser de gran ayuda. La representación de las estructuras anatómicas es demasiado

compleja como para poder ser visualizada en solo dos planos del espacio. Históricamente, la enseñanza de la anatomía se ha hecho a partir de representaciones bidimensionales, de modelos físicos tridimensionales o de cuerpos reales. Es ahora cuando ha sido factible crear modelos anatómicos digitales tridimensionales, que pueden ser explorados en línea a través de Internet [4].

Otras tecnologías en alza relacionadas con el mundo tridimensional son las impresoras 3D. Cada vez más común, desde 2003 se ha obtenido un crecimiento considerable en la venta de las impresoras que permiten la creación de piezas en 3D, en parte por la reducción del coste que éstas presentan. La utilización de este tipo de impresoras abarca una cantidad de ámbitos así como joyería, calzado, diseño industrial, arquitectura, ingeniería, educación, etc [5].

3. Alcance y Objetivos

El proyecto a realizar contempla diferentes tipos de objetivos, los cuales se expondrán con detalle en los siguientes párrafos; empezando por el objetivo principal, exponiendo posteriormente los objetivos parciales los cuales forman el primero y finalmente definiendo el objetivo secundario:

- Objetivo principal: Realizar un sistema de utilidad para la visualización de imágenes biomédicas, el cual pueda ser utilizado por distintos profesionales a los que éstas les sean de ayuda. Esta herramienta deberá leer con eficacia imágenes en formato .obj, representar los vértices y las caras triangulares o rectangulares del archivo en un escenario 3D, formando la silueta correspondiente de la imagen deseada, y añadirle las texturas convenientes logrando finalmente la imagen completa; pudiendo rotar ésta, hacia arriba, abajo, derecha o izquierda, logrando así la visualización en todos los ángulos de una imagen 3D.

Para lograr con satisfacción el objetivo principal expuesto anteriormente, se deben lograr una serie de objetivos parciales:

- Buscar de manera exitosa una librería la cual permita representar los vértices y las caras triangulares con el fin de crear la silueta. Esta búsqueda se basará en diferentes alternativas, de las cuales se utilizará una de ellas, con el fin de lograr el desarrollo de la acción especificada.

- Desarrollar de manera correcta de la silueta de la imagen en 3D con la librería seleccionada, pudiendo aprender sobre ésta, familiarizarse con los comandos que proporciona y que son de utilidad para el proyecto, y verificar la correcta elección en el objetivo anterior.

- Buscar una librería que nos permita la lectura de las texturas que se podrán encontrar en formatos .jpg, .png, etc. Esta deberá ser compatible con la primera librería, pues será ella la que tenga los comandos para pegar las texturas en la silueta ya creada.

- Desarrollar una solución que permita el pegado de las texturas en la silueta utilizando las dos librerías seleccionadas, aprendiendo sobre el uso de la segunda librería y sobre los nuevos comandos de utilidad de la primera. Se verificará la correcta elección en el objetivo anterior.

A continuación, se expondrá el objetivo secundario del proyecto:

- Diseñar un sistema de visualización no invasivo el cual mejore los actuales, logrando así una máxima captación de usuarios.

En conclusión, el alcance del proyecto se basa en el desarrollo de la silueta de la imagen deseada en 3D, a la que posteriormente se le añadirán texturas lográndose así la imagen biomédica final, con la intención de la utilización del programa en el ámbito de la medicina.

4. Beneficios del proyecto

En este apartado se presentarán los beneficios que podría acarrear el uso de la solución en diferentes ámbitos.

4.1. Beneficios Técnicos

4.1.1. Medicina

Como se viene diciendo desde un principio este proyecto está principalmente enfocado a la medicina. La visualización de imágenes en 3D en un campo tan importante como la medicina, puede ser un factor significativo para descubrir enfermedades difíciles de apreciar en imágenes en 2D. Al ser este mundo un mundo en tres dimensiones, el profesional que deba realizar una operación quirúrgica obtendrá una mayor realidad de un cuerpo en 3D, pudiendo extraer de las imágenes mayor información, un elemento de importancia para la preparación de estos trabajos. Como cada vez es más frecuente el uso de técnicas basadas en imagen durante la operación, la optimización de éstas es esencial; una de estas optimizaciones podría ser el uso de imágenes en 3D siendo así de mayor facilidad la toma de decisiones en ciertos puntos drásticos, o la mayor apreciabilidad de ciertos problemas durante la incisión.

4.1.2. Docencia

A la hora de impartir una clase o de realizar una presentación, es bien sabido que el dinamismo de las herramientas usadas es de gran ayuda para que los oyentes presten una mayor atención. El uso de herramientas 3D que proyectan imágenes las cuales se pueden rotar logrando un ángulo de visualización completo, pueden ser una herramienta clave para que el alumnado capte la información del emisor con una mayor facilidad y de una manera más entretenida que utilizando imágenes estáticas o en 2D. En carreras como la medicina o la enfermería el uso de un programa que refleje nuestro cuerpo en 3D o parte de este, será más ilustrativo y más real que una simple fotografía o prueba médica en 2D, y de menos coste que modelos físicos tridimensionales o cuerpos humanos reales. Además se tendrá gran versatilidad de archivos para visualizar en la red, lográndose así que en el momento de tener que tratar con pacientes reales, el cambio a estos sea mínimo, debido a la cantidad de imágenes visualizadas, consiguiéndose así una mayor eficiencia en este sector.

4.1.3. Impresión

Cuando se quiere imprimir una pieza en 3D, se puede acudir a internet y partir de un objeto ya diseñado o se puede crear uno propio. En ambos casos el objeto debe ser visualizado de alguna manera, básicamente para conocer lo que se va imprimir y evitar resultados no deseados, es decir, se debe tener una herramienta que visualice de manera eficiente el contenido que se desea imprimir. El uso de ésta debe permitir visualizar la imagen y poder rotarla en busca de fallos.

4.2. Beneficios Económicos

4.2.1. Medicina

El uso de herramientas en 3D en la medicina puede traer consigo un beneficio económico considerable. La mayor facilidad de visualización que proporciona esta tecnología puede evitar el tener que hacer pruebas más concretas y de un mayor coste, para llegar a un diagnostico claro. Por ejemplo, según el periódico El País, cada prueba de resonancia magnética tiene un coste de 244 €, y una de cada tres resonancias magnéticas de lumbares son innecesarias, el uso de esta solución junto a otras podría evitar este problema [8]. En una operación pueden surgir problemas que encarecen los costes de estas, por lo que una mejor preparación previa con una mayor tecnología podría reducir el gasto y lograr una mayor eficiencia.

4.3. Beneficios Sociales

4.3.1. Docencia

En la enseñanza a niños o adolescentes también podría tener un papel importante, el uso de estas tecnologías podría dar un enfoque más divertido y más dinámico a ciertas materias, alcanzándose así un mayor interés por parte de estos y evitando en parte el abandono escolar, el cual en España es de un 18,98%, uno de los más altos de la UE colocándose la segunda en el ranking después de Malta [9].

4.3.2. Redes Sociales

El uso de esta herramienta puede traer consigo el entrar a formar parte en una red social en la que se comparten diseños creados por uno mismo, o la descarga de modelos creados por otros usuarios con el fin de visualizarlos en la herramienta creada.

5. Estado del arte

Para conseguir los objetivos del proyecto se analiza el estado del arte de las tecnologías aplicables al proyecto.

5.1. Proceso de modelado

5.1.1. Modelado poligonal

Se basa en la representación de puntos en un espacio 3D, a los que se les denomina vértices. Estos puntos en unión forman distintas formas poligonales las cuales agrupadas construyen la imagen 3D deseada. Como ventaja se puede mencionar que son flexibles y que un ordenador puede renderizarlos con rapidez; como desventaja se podría decir, que al ser los polígonos empleados planos, las superficies no pueden ser curvas, solo pueden aproximarse, empleando varios polígonos para este fin [10].

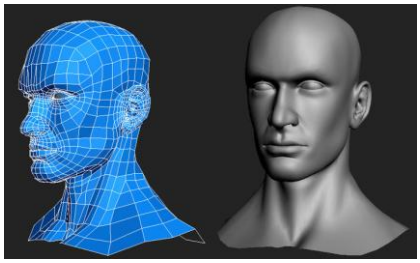


Figura 3: Modelado poligonal

5.1.2. Modelado de curvas (NURBS)

Las superficies se definen por una serie de curvas con unos vértices de control. En este modelado se utilizarán ciertos puntos para controlar las curvas de interpolación que finalmente darán como resultado la imagen en 3D. Cada punto tendrá una ponderación, aumentar el peso de uno de los puntos dará como resultado que la curva más cercana a él pase por ese lugar. Esta es la técnica de mayor precisión, es por esto que es utilizada en la arquitectura y en el ámbito industrial, donde se prima la precisión [11].

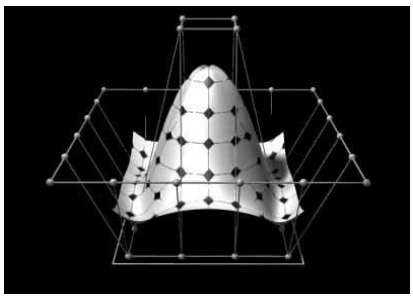


Figura 4: Modelado de curvas

5.1.3. Escultura digital

Es un método con pocos años. Se pueden diferenciar 3 tipos en la actualidad: Desplazamiento, Voxel y Teselación Dinámica. El desplazamiento usa un mapa de 32bit en el cual se almacenan las ubicaciones ajustadas de los vértices utilizados. Voxel tiene capacidades similares al desplazamiento, sin sufrir de polígonos forzados cuando no existen suficientes polígonos en cierta región con la que poder lograr una deformación. Finalmente, la Teselación dinámica es similar a Voxel, con la diferencia de que se usa la triangulación para lograr una superficie lisa y conseguir unos detalles más finos. Este modelado puede ser generado para impresión, animación o videojuegos [10].

Figura 5: Escultura Digital

5.2. Formatos de archivos 3D

Se realiza el análisis de los formatos 3D más utilizados.

5.2.1. Archivos .obj

El formato .obj es un formato común y como consecuencia puede ser exportado en una diversidad de aplicaciones de software 3D para su posterior visualización y edición. Este formato de archivo contiene las coordenadas 3D (puntos y polígonos), mapa de texturas y otra información de objeto. No contiene definiciones de colores de las caras, por lo que se deben utilizar las texturas para la representación en color del objeto.. Los archivos en formato OBJ se pueden abrir con Autodesk Maya 2013, Blender, y MeshLab en Microsoft Windows, Mac OS y Linux [12].

5.2.2. Archivos .fbx

Se define como .fbx a un dibujo 2D o 3D guardado en el formato Autodesk FBX. Se conserva la fidelidad y la funcionalidad del archivo digital; este podrá ser manipulado por una multitud de programas. Se utiliza para la creación de la interoperabilidad entre aplicaciones 3D. Utiliza modelado de curvas (NURBS) [13].

5.2.3. Archivos .max

Los archivos .max son archivos desarrollados por Autodesk, los cuales son exclusivamente utilizados por Autodesk 3DS Max. Este tipo de archivos son usados para el diseño y desarrollo de juegos, programas de televisión y películas. Se puede guardar en ellos una variedad de gráficos, textos, y otros datos, como líneas, luces, efectos, sombras etc [14].

5.2.4. Archivos .3ds

Desarrollados para el uso junto a la aplicación *3DStudio.* El archivo está formado por una gran cantidad de información referente al programa, lo que incluye datos de malla, atributos, mapas de bits, referencias, configuraciones y demás. Este formato no solo contiene información geométrica del objeto, sino que también almacena el color y la información sobre el fondo. Este tipo de archivo 3D, guarda su contenido en formato binario de manera que la aplicación utilizada pueda leer el objeto con una mayor rapidez y a la vez hacer que el archivo sea lo más pequeño posible [14].

5.3. Librería Gráfica

Se define como librería gráfica aquella capaz de generar imágenes en base a modelos matemático y ciertos patrones de iluminación, texturas, etc.

5.3.1. OpenGL

Se puede utilizar para varios lenguajes de programación (Java, Fortran, Delphi, C, C++, C#, Perl, Phyton, VB, Ada, etc.) y varias plataformas. Es usada para escribir aplicaciones que representen gráficos en 2D y 3D, así como para desarrollar programas de realidad virtual, representación científica, visualización de información y simulación de vuelo; además también se utiliza en la creación de videojuegos. Esta librería contiene más de 250 funciones que permiten representar escenas en 3D a partir de primitivas geométricas, como por ejemplo puntos, líneas o triángulos. El funcionamiento de esta librería consiste básicamente en aceptar primitivas tales como puntos, líneas y polígonos y convertirlas en píxeles [15]. Esta API está basada en procedimientos de bajo nivel, es decir requiere que un programador dicte los pasos exactos para renderizar, construir y manejar la escena, en contraste con otras APIs en las cuales el programador describe la escena y deja que la biblioteca controle los detalles para representarla. A pesar de que esto último supone una dificultad añadida, permite una total libertad para realizar los efectos y las acciones deseadas en la imagen 3D [16]. Se le puede añadir ciertas extensiones tales como GLU, GLUT o GLUI que añaden características no disponibles en la propia

librería. OpenGL se dedica exclusivamente al renderizado de las imágenes ofreciendo altas prestaciones para este fin, es decir no realiza la gestión de las ventanas, sonido, etc. Al ser una librería muy extendida, existe una gran documentación sobre esta. Se ha utilizado OpenGL en proyectos de distintas tarjetas gráficas como la NVIDIA GeForce 400 series o la Tegra K1 y Tegra X1 [15].

5.3.2. Direct3D

Es una API para la programación de gráficos 3D. Solo disponible para Windows de 32 y 64 bits. Utiliza entidades gráficas elementales, como líneas, polígonos y texturas para crear los gráficos 3D deseados y realizar transformaciones geométricas sobre estos. Suele ser usada para la creación de videojuego u otras aplicaciones en las cuales el rendimiento es fundamental. La mayor ventaja de esta librería es que se comunica de forma directa con los drivers de pantalla, consiguiendo unos mejores resultados en la representación de una imagen 3D. Su licencia es de pago y solo está permitido el uso de esta librería al comprador de la mismo [17].

5.3.3. VTK

Es un sistema de software libre para la realización de gráficos 3D, procesamiento de imágenes y visualización, programado en C++, Tcl, Perl, Python y Java. A diferencia de las dos anteriores bibliotecas, esta es de alto nivel o descriptiva [16]. Incluye una amplia variedad de algoritmos de visualización que incluyen escalares, vectores, tensores, texturas y métodos volumétricos, así como técnicas avanzadas de modelado como modelado implícito reducción de polígonos, suavizado de malla, corte, contorneado y triangulación de Delaunay. Contiene además un amplio marco de visualización de la información y un conjunto de widgets de interacción 3D. Está disponible para las plataformas Linux, Windows, Mac y Unix [18].

5.3.4. Matpack

Librería numérica y de representación gráfica en 3D, basada en lenguaje C+. Al igual que la librería anterior, es de alto nivel o descriptiva. Además de poder generar gráfico de superficies 3D, gráficos de imagen, histogramas, superficies paramétricas, manipular imágenes, etc. Puede realizar algoritmos como por ejemplo, generar números aleatorios, criptografía, FFT, etc. Como en OpenGL, a esta librería se le pueden añadir extensiones para completarla. Está disponible para Unix, Linux y Windows [16].

5.4. Librería de procesamiento de imágenes

Una librería de procesamiento de imágenes es aquella que permite la lectura, modificación, conversión, visualización y mejora de las imágenes entre otras cosas. Se espera que esta lea diferentes formatos como por ejemplo .jpg, .gif o .png. Se estudiará el uso de distintas librerías de procesamiento de imágenes, las cuales serán utilizadas para la lectura de las texturas de los archivos .obj.

5.4.1. ImageMagick

Conjunto de utilidades de código abierto. Permite la lectura, visualización, manipulación, conversión y escritura de imágenes en 200 formatos diferentes. Consiste en un conjunto de herramientas de línea de comandos para manipular imágenes. No tiene interfaz gráfica de usuario como Adobe Photoshop, pero sí una API para una multitud de lenguajes de programación: Ada, C, Ch, COM+, C++, Java, Julia, Lisp, LuaJIT, Python, Pascal, etc. ImageMagick utiliza múltiples subprocesos computacionales para aumentar el rendimiento y poder leer, procesar o escribir imágenes que pueden llegar a componerse de hasta terapixeles. Es un software gratuito y multiplataforma, pudiendo ser utilizado en Linux, Windows, Mac OS X, iOS, sistema operativo Android y otros [19].

5.4.2. OpenCV

Librería de visión artificial desarrollada por Intel y utilizada en un sinfín de aplicaciones. Esta ofrece más de 500 funciones que van desde la lectura de imágenes hasta sistemas de seguridad con detección de movimiento, reconocimiento facial, calibración de cámaras, visión aérea, visión robótica o reconocimiento de objetos. Su licencia permite que su uso sea libre para ambito académico y comercial. Tiene interfaces C++, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android. La librería OpenCV está programada en código C y C++ optimizado, lo cual proporciona un entorno de desarrollo fácil de utilizar y con una gran eficiencia. Proyectos como el vehículo no tripulado Stanley de la Universidad de Stanford (ganador en el año 2005 del Gran Desafío DARPA) han sido realizados usando esta librería [20].

5.4.3. FreeImage

FreeImage es un proyecto de biblioteca de código abierto creada para desarrolladores que desean utilizar formatos de imágenes gráficas populares como PNG, JPG u otros. Es fácil de usar, rápida y compatible tanto con Windows como con Linux y Mac OS X. Se puede utilizar en muchos lenguajes de programación, incluidos C, C ++, C#, Delphi, Java, Perl, Python, PHP, TCL o Ruby [21].

6. Análisis de alternativas

Para optimizar el funcionamiento de la herramienta 3D y el tiempo requerido para diseñar está, se han tenido en cuenta una serie de alternativas, las cuales por unas razones u otras han sido seleccionadas o descartadas, siguiéndose así un camino hasta el resultado final de la solución.

6.1. Proceso de modelado

6.1.1. Criterios de selección

Para la selección del proceso de modelado, se tiene en cuenta principalmente, la rapidez del renderizado de la imagen, la precisión de la solución, la información accesible que hay sobre la solución, el coste que suponen los equipos que pueden implementar estas soluciones y la cantidad de archivos que utilizan los diferentes tipos de modelado.

6.1.2. Selección de la solución

La escultura digital es una gran alternativa para representar imágenes, puesto que la calidad que se logra en ellas es superior a la lograda por el resto de modelados. Aun así esta requiere de equipos de valor elevado debido a los recursos que utiliza para el renderizado de la imagen, esto mezclado junto a la escasez de información encontrada sobre el método hace que se descarte como opción. Entre el modelado de curvas y el modelado poligonal, se escoge el segundo. El modelado de curvas es de mayor precisión y es por esto que se utiliza en el ámbito industrial o en la arquitectura donde la representación de una pieza de un tamaño pequeño, por ejemplo, ha de ser exacta, puesto que la manipulación de estas piezas suele ser llevada a cabo por maquinas; sin embargo, los usuarios que utilicen esta herramienta no serán maquinas, luego la pequeña mejora obtenida mediante el modelado de curvas será completamente inapreciable para el ojo humano; la rapidez que proporciona el modelado poligonal a la hora de representar una imagen, en cambio, sí que puede ser de gran ayuda en ciertos momentos críticos de una operación. Como añadido, hay que mencionar que la mayor parte de los modelos 3D hoy en día están construidos como modelos de textura poligonal [10], por lo que eligiendo este modelado se tendrá la opción de una mayor probabilidad de compatibilidad con imágenes realizadas por otros usuarios, si es necesario acudir a ellas en algún momento.

6.2. Formato del archivo 3D

6.2.1. Criterios de selección

Para la elección del formato del archivo 3D lo primero que se hace es investigar los tipos de formatos que más se utilizan en las páginas web que permiten la descarga de imágenes 3D, como pueden ser las páginas *TurboSquid* u otras,

puesto que estos servidores serán los proveedores de las imágenes utilizadas en el proyecto. A continuación, se representa la tabla 1, con la cantidad de formatos que se pueden encontrar en el portal mencionado anteriormente (*TurboSquid*), el número de imágenes que se pueden hallar en ese formato, y el por ciento calculado respecto al total de las imágenes.

Tabla 1: Formatos y porcentajes

Formato.	Imágenes en ese formato	Nº total de imágenes	Por ciento respecto al total
.obj	432.194	681.830	63,38%
.fbx	326.789	681.830	47,92%
.max	470.915	681.830	69,07%
.c4d	120.410	681.830	17,66%
.ma/mb	102.675	681.830	15,06%
.blend	52.450	681.830	7,69%
.unitypackage	4.286	681.830	0,63%
.upk/uasset	3.488	681.830	0,51%
.dae	43.535	681.830	6,38%
.3ds	280.280	681.830	41,11%
.skp	9.922	681.830	1,45%
.lxo	7.646	681.830	1,12%
.lwo/lws	77.908	681.830	11,43%
.stl	27.726	681.830	4,06%

Se debe aclarar que la suma de imágenes en diferentes formatos da un número mayor que el total de la derecha y que la suma de porcentajes es mayor que el 100%, esto ocurre puesto que si la misma imagen está en diferentes formatos se cuenta como una sola en el total, pero al filtrar por formatos se estaría contando como una imagen para cada tipo de archivo.

Se seleccionan como posible opción los 4 archivos con un porcentaje mayor en la tabla 1, descartando el resto de formatos. Además de esto se descartarán diferentes opciones utilizando como criterios, la opción de modelado elegida en el apartado anterior; la dificultad, flexibilidad y libertad que tienen los diferentes formatos a la hora de ser visualizados, la facilidad de encontrar los distintos tipos de archivos en la red y el funcionamiento de estos en diferentes sistemas operativos.

6.2.2. Selección de la solución

Entre los archivos expuestos, debido a la selección de la opción de modelado poligonal del apartado anterior se descartan los archivos .fbx. Tanto los archivos .max como .3ds son archivos muy vincularos a ciertos programas para su representación y realizar un visualizador para ese tipo de archivos sería un trabajo más costoso que el de realizar uno para archivos .obj, logrando un

resultado similar o con una ventajas que no influyen en el campo para el que está pensado la aplicación. Debido a la libertad y flexibilidad que proporcionan los archivos .obj para su lectura y representación, se elige esta opción; además como se ha mencionado anteriormente este tipo de archivos son muy comunes (el 63,38% de las imágenes de *TurboSquid* pueden ser representadas en este formato) por lo que no se tendrán problemas a la hora de encontrar la imagen de la parte del cuerpo que se quiere representar en la red. El poder usar este tipo de archivos en prácticamente todos los sistemas operativos es otro punto a favor (Microsoft Windows, Mac OS y Linux).

6.3. Librería Gráfica

6.3.1. Criterios de selección

Para la elección de la alternativa de la librería gráfica, se tienen en cuenta la posibilidad de que éstas puedan ser utilizadas en múltiples plataforma, el coste de la librería, la libertad y flexibilidad que proporcionan éstas a la hora de representar una imagen, las dificultad que tiene cada librería a la hora de ser utilizadas, la cantidad de funciones que contienen, la información que hay sobre estas en la red y la compatibilidad de cada una con extensiones que proporcionen funciones adicionales.

6.3.2. Selección de la solución

Las limitaciones que presenta Direct3D en cuanto al exclusivo uso de la librería sobre el sistema operativo Windows hacen que esta sea descartada, además OpenGL es igual de potente (son claros competidores) y no se necesita la compra de la librería para poder utilizarla, con el añadido de que esta última si puede ser usada en diferente plataformas. Tanto VTK como Matpack son librerías de alto nivel, lo cual es de ayuda; pero quita cierta libertad y flexibilidad a la hora de visualizar una imagen. Además el añadido que tienen esas librerías no es de gran ayuda para el proyecto que se desea realizar. La gran cantidad de funciones que se pueden usar en la librería OpenGL y la flexibilidad que proporciona esta, hace que sea la elegida desde un principio. El uso es más difícil que el de las librerías descriptivas, pero la gran cantidad de información que hay en la red sobre la utilización de esta, y la cantidad de ejemplos y tutoriales que se pueden encontrar sobre el manejo de esta API hará que al final la dificultad disminuya. El poder utilizar extensiones para completar esta librería es otro punto a favor, puesto que como se verá en un futuro se necesitará la extensión GLUT para poder realizar el proyecto. La cantidad de lenguajes con los que OpenGL tiene compatibilidad también es otra cualidad a destacar.

6.4. Librería de procesamiento de imágenes

6.4.1. Criterios de selección

El uso de la librería de procesamiento de imágenes en este proyecto es muy específico, es por esto que no se prima la cantidad de funciones de cada librería, pero sí la cantidad de formatos que puede abrir cada una, las limitaciones que se tienen en cuanto al tamaño de las imágenes que deben leer y la facilidad de utilización de éstas.

6.4.2. Selección de la solución

OpenCV es una librería muy potente que permite realizar una infinidad de proyectos. Esta sería una gran opción si se necesitara hacer con ella algo de más complejidad que leer una imagen de texturas. Es por esta razón que la opción es descartada, puesto que las otras dos opciones de este apartado, pese a tener menos funciones, son más idóneas para el cometido que se desea realizar. Entre FreeImage e ImageMagick se escoge la segunda. La cantidad de formatos de imagen que permite leer ImageMagick es básicamente lo que hace descartar la otra opción, puesto que con tal cantidad de formatos compatibles, 200 para ser exactos, será casi imposible encontrar una imagen en 3D la cual no tenga un archivo de texturas compatible con esta librería. Además el fácil uso de esta es un punto a favor, puesto que como se ha remarcado anteriormente, la labor que desempeña este tipo de librerías no es de gran complejidad, en el caso de este proyecto es básicamente la de leer los ficheros de imagen pixel a pixel y devolverlos en un formato que la librería gráfica escogida en el paso anterior (OpenGL) pueda utilizar. Como añadido el que ImageMagick pueda trabajar con imágenes de gran tamaño (puede leer imágenes de hasta terapixeles) es de agradecer, ya que esto supondrá que prácticamente no haya limitaciones en el tamaño de los archivos utilizados como textura.

7. Descripción de la solución

Partiendo del contexto, de los objetivos expuestos y de los resultados del análisis de alternativas, se propone lo siguiente: un visualizador 3D de imágenes biomédicas programado en lenguaje C, utilizando las librerías OpenGL e ImageMagick.

OpenGL es una librería escrita en C, desarrollada por Silicon Graphics, que nos permite dibujar escenas complejas en dos o tres dimensiones, a partir de primitivas geométricas simples, como pueden ser los puntos, las líneas, los triángulos, etc. Es decir, nos permite dibujar cualquier figura compleja utilizando otras figuras más simples. Esta librería también permite manipular las figuras, pudiendo rotarlas, por ejemplo. Esto es de gran utilidad a la hora de visualizar una imagen 3D, ya que se podrá rotar ésta, pudiendo visualizar la figura desde cualquier ángulo. En este proyecto se utiliza esta librería junto a la extensión GLUT, la cual facilita una funcionalidad para el manejo de ventanas e interacción por medio del teclado y ratón.

Para empezar se utiliza la librería expuesta con el fin de crear la silueta de la imagen deseada. Para ello, lo primero que se hace es descargar una imagen .obj de la página ya mencionada anteriormente en este documento, *TurboSquid*. Se elige como imagen a representar una cabeza humana.

Figura 6: Cabeza humana vista desde el visualizador web

La figura 6 se visualiza desde el visualizador 3D que proporciona la página web, ese deberá ser el resultado final de la solución.

7.1. Inicialización de ventana de visualización

Se comenzará el programa de visualización. Lo principal será inicializar la ventana donde se visualizará la imagen 3D descargada. Para lograr lo anterior se utilizarán los siguientes comandos en el orden expuesto:

glutInit (&argc, argv); es una llamada a GLUT, la cual inicializará OpenGL.

glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
Se define cómo se quiere representar la imagen, en este caso en particular se elige una ventana de doble buffer (GLUT_DOUBLE), una aplicación de color definida por tres valores numéricos, uno para el rojo, otro para el verde y el último para el azul (GLUT_RGB), y le se le indica que se le va a dar profundidad a la imagen (GLUT_DEPTH).

glutInitWindowSize(500,500);
Se especifican las dimensiones de la ventana, en este caso 500 pixeles de alto por 500 pixeles de ancho.

glutInitWindowPosition(0,0); Se coloca la ventana en un punto de la pantalla.

glutCreateWindow("Imagen Biomédica 3D"); Se le introduce un título a la ventana.

glEnable(GL_DEPTH_TEST); Cada vez que se va a renderizar un pixel, se comprobará que no se haya dibujado antes en esa posición un pixel que esté más cerca respecto a la cámara.

Tras haber iniciado correctamente la ventana a utilizar, se procede a dibujar la imagen seleccionada en ésta. Para ello, se hace uso del comando **glutDisplayFunc(display3);** donde el nombre display3 (se puede usar cualquier otro nombre) representa la función que se debe invocar cada vez que se quiera dibujar algo en pantalla. En esta función es donde el usuario deberá insertar la programación necesaria para la visualización de la imagen biomédica.

7.2. Estructura de los ficheros .obj

Se procede a la programación de la función display3. El primer paso que se debe hacer en esta función será el de leer el archivo objeto, para ello se debe conocer la estructura de este tipo de archivos. Acudiendo a la imagen descargada anteriormente, se abre el .obj con un fichero de texto y se visualiza su contenido.

Los archivos .obj son archivos de datos simples que representan la geometría 3D. Estos incluyen la posición de cada vértice, la posición de cada vértice de

coordenadas de texturas, los vértices normalizados y las caras que componen cada polígono.

A continuación, se procede a explicar de forma más detallada estos puntos. Si se lee un archivo .obj en formato de texto se encuentra lo siguiente:

- Las líneas que comienzan con la letra v, representan los vértices, estos vendrán definidos por 3 números. El primer número será la posición de ese vértice en el eje X, el segundo será la posición de este en el eje Y, y el tercero la posición del vértice en el eje Z; logrando así un punto en el espacio de tres dimensiones.

  v 28.426630020141602 109.14207458496094 30.866069793701172

- Las líneas que comienzan con la letra vn serán los vértices normalizados, es decir, representan lo mismo que las líneas explicadas anteriormente pero los números que definen el vértice en el espacio tendrán un valor comprendido entre 0 y 1.

  vn 0.786630020141602 0.9707458496094 0.543069793701172

- Las líneas que comienzan con la letra vt, son los vértices de texturas. Las texturas vendrá definidas en una imagen 2D (normalmente una imagen .png o .jpg) la cual hará que la imagen 3D tenga diferentes colores y no sea solo una silueta. Es por el hecho de que la imagen esté en 2D que para identificar el fragmento de textura que se desea utilizar para pegar en la cara formada por los vértices, solo se necesitarán 2 números, los cuales serán las coordenadas del fragmento.

  vt 0.567452020141602 0.432147458496094

- Las líneas que comienzan con la letra f (las caras que componen cada polígono), son algo más complicadas; por ello se utilizará un ejemplo para explicarlas. A f le seguirán 3 componentes con 3 números cada uno:

  f 1/1/4 2/2/5 3/8/6

  El primer número de cada componente representa el vértice que se quiere utilizar, estos vendrán ordenados según la posición que tengan en el archivo, es decir, la línea v que más arriba esté tendrá asignado el número 1, la siguiente el número 2 y así sucesivamente. En este caso se utilizan los vértices número 1, 2 y 3. Juntando esos 3 vértices (se recuerda que en cada línea v leída están las posiciones exactas de los vértices) se creará un triangulo. El tercer número de cada componente representa el vértice

normalizado. Utilizando los distintos vértices y las distintas líneas f, se crearán múltiples triángulos logrando la silueta de la imagen deseada. El segundo número de cada componente de las líneas f serán los vértices de textura, en este caso se utilizan las líneas uno, dos y ocho que empiezan con la letra vt. Si la línea f tuviese 4 componentes en vez de 3, como en el ejemplo explicado, se tendrían 4 vértices por unir, es decir que se necesitaría crear un polígono cuadrilátero para representar correctamente la imagen.

Como no todas las imágenes .obj contienen los valores normalizados (vn) y dado que para que OpenGL pueda representar los valores, estos deben estar comprendidos entre 0 y 1, se decide coger en todos los casos los valores de las líneas v (vértices sin normalizar) buscar el máximo valor de todas estas líneas y normalizar todos los vértices respecto a este valor máximo hallado.

7.3. Creación de la función para dibujar en pantalla

7.3.1. Creación de la silueta de la imagen

Cuando ya se conoce la estructura de los archivos que se van a utilizar, se separan los valores leídos en cada tipo de línea en matrices diferentes, con las cuales se trabajará. Una vez se tienen compuestas estas matrices, se crearán los polígonos usando los siguientes comandos:

glBegin(GL_TRIANGLE);
Especifica la primitiva o las primitivas que se crearán a partir de los vértices especificados entre este comando y el comando glEnd. Las opciones de creación de polígonos son GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP y GL_POLYGON.

glVertex3f(vertices[f[r][0]-1][0]/max,vertices[f[r][0]-1][1]/max, vertices[f[r][0]-1][2]/max);
Sitúa el vértice en la posición X, Y, Z del escenario 3D. Como se aprecia se utiliza la matriz vértices donde se guardan todos los valores de los vértices que componen la imagen. Para la elección del vértice correspondiente, dentro de la matriz vértices se introduce el valor numérico de la matriz f en ese valor de r (r irá aumentando hasta acabar toda la matriz f y formar todos los polígonos), es decir si se tiene que el vértice requerido para formar el triángulo es el cuarto de la lista, f[r][0] será igual a 4, y como se han guardado de forma ordenada los valores de cada vértice, en la matriz vértices; el valor del cuarto vértice vendrá dado por vértices[4-1][0] (el -1 se utiliza puesto que la matriz empieza en la posición 0 y los valores de f empiezan en 1) el cual te devolverá la coordenada del vértice 4 en X, vértices[4-1][1] el cual te devolverá la coordenada del vértice 4 en Y y vértices[4-1][2] el cual te devolverá la coordenada del vértice 4 en Z. El /max se utilizará para normalizar el vértice.

Como se está creando un triángulo con este vértice, se necesitarán otros dos vértices para formar la cara:

- glVertex3f(vertices[f[r+1][0]-1][0]/max,vertices[f[r+1][0]-1][1]/max, vertices[f[r+1][0]-1][2]/max);
- glVertex3f(vertices[f[r+2][0]-1][0]/max,vertices[f[r+2][0]-1][1]/max, vertices[f[r+2][0]-1][2]/max);

glEnd(); Indica el fin del polígono.

Insertando este método en un bucle y haciendo que el valor de r aumente, se logrará representar de forma ordenada todos los polígonos que forman la imagen 3D, creando la silueta de esta.

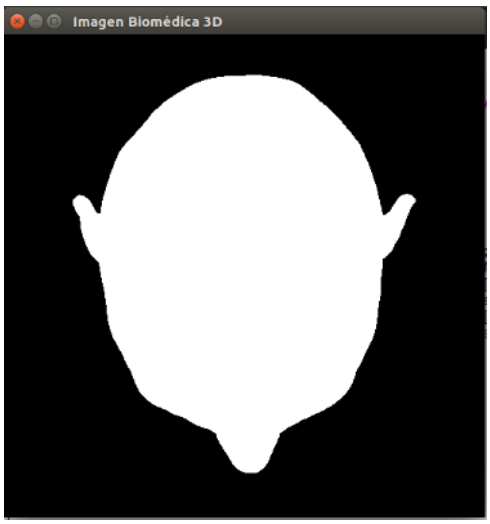


Figura 7: Silueta de la imagen 3D

En la figura 7 se puede apreciar que la silueta de la imagen se ha representado correctamente y que coincide con la forma de la cabeza vista en la figura 5. Tras esto se procede a la inserción de las texturas que crearán la imagen completa en 3D. Para realizar esta acción, se hará uso de ImageMagick, la segunda librería elegida. Esta junto a OpenGL serán las responsables de leer la imagen de texturas pixel a pixel y pegarla en la silueta de la figura 7.

7.3.2. Carga de texturas para el pegado posterior

Como se ha mencionado anteriormente la textura es una imagen en 2D la cual permite que mediante las coordenadas que proporciona el archivo .obj en las líneas de tipo vt, puedan utilizarse fragmentos de ésta y ser pegados en una cara poligonal definida por vértices.

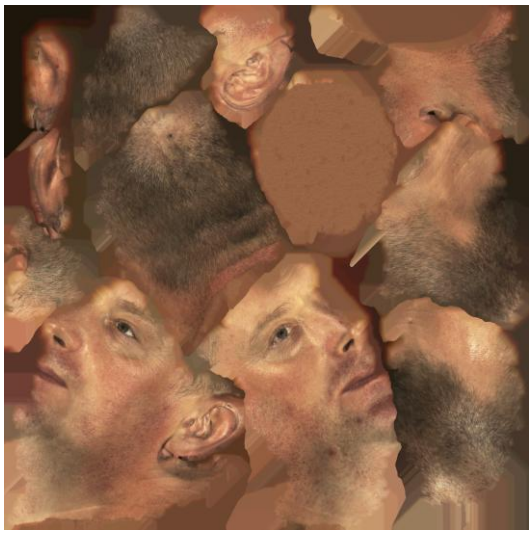


Figura 8: Imagen de texturas

Sabiendo esto, se procede a la inserción de las texturas en la silueta lograda en la figura 7.

MagickWandGenesis(); se inicializa la librería ImageMagick.

MagickWand *wand=NewMagickWand();
Devuelve un nuevo recurso en el que se guardará la imagen leída que aparece en la figura 8.

MagickReadImage(wand,"Pasha_guard_head_0.png");
Lee la imagen de texturas y la guarda en el recurso creado en el comando anterior.

***width = MagickGetImageWidth(wand);** Se obtiene la anchura de imagen.
***height = MagickGetImageHeight(wand);** Se obtiene la altura de la imagen.

MagickExportImagePixels(wand, 0,0, *width, *height, "RGB", CharPixel, data);
Se guarda la imagen almacenada en el recurso llamado wand, en una matriz de caracteres llamada data. Para ello se le debe indicar a ImageMagick la altura, la anchura y la aplicación del color a la imagen;

glTexImage2D(GL_TEXTURE_2D,0,GL_RGB, *width, *height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
Especifica la imagen de texturas bidimensional que se va a utilizar para el pegado en la silueta.

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
Estos comandos establecen los parámetros de las texturas para el pegado de estas. GL_TEXTURE_WRAP_S y GL_TECTURE_WRAP_T permiten envolver la silueta con las texturas leídas. GL_REPEAT hace que la parte entera de la coordenada s o t sea ignorada, el GL usa solo la parte fraccionaria, creando así un patrón de repetición. Inicialmente GL_TEXTURE_WRAP_S y GL_TEXTURE_WRAP_T se establecen con GL_REPEAT. Los dos últimos comandos sirven en el caso en el que la textura deba ser magnificada o minificada para el pegado de esta. Inicialmente GL_TEXTURE_MAG_FILTER y GL_TEXTURE_MIN_FILTER se establecen con GL_LINEAR.

glEnable(GL_TEXTURE_2D); Activa las texturas.

Utilizando estos comandos se consigue cargar las texturas para su pegado.

7.3.3. Pegado de texturas

Para especificar donde se desea pegar la textura se debe poner la siguiente instrucción antes de definir cualquier vértice de cada polígono, es decir antes de los comandos **glVertex3f** explicados previamente:

glTexCoord2f(vt[f[r][1]-1][0],vt[f[r][1]-1][1]);
El funcionamiento de este comando es básicamente el mismo que el de glVertex3f, con la diferencia de que al estar los vértices de texturas en una imagen 2D solo son necesarias las coordenadas en el eje X e Y. El modo de lectura de los vértices de textura es igual que el del comando glVertex3f, es decir dentro de la matriz vt se introduce el valor numérico de la matriz f en ese valor de r (r irá aumentando hasta acabar toda la matriz f y formar todos los polígonos), con la diferencia de que en este caso se coge la segunda columna de la matriz f, la correspondiente a los vértices de texturas en vez de la primera correspondiente a los vértices.

En conclusión para crear mediante este método todos los triángulos con sus texturas correspondientes, se necesitará el siguiente segmento de código.

```
glBegin(GL_TRIANGLES);
while(r<s){
        glTexCoord2f(vt[f[r][1]-1][0],vt[f[r][1]-1][1]);
        glVertex3f(vertices[f[r][0]-1][0]/max,vertices[f[r][0]-1][1]/max,vertices[f[r][0]-1][2]/max);
        glTexCoord2f(vt[f[r+1][1]-1][0],vt[f[r+1][1]-1][1]);
        glVertex3f(vertices[f[r+1][0]-1][0]/max,vertices[f[r+1][0]-1][1]/max,vertices[f[r+1][0]-1][2]/max);
        glTexCoord2f(vt[f[r+2][1]-1][0],vt[f[r+2][1]-1][1]);
        glVertex3f(vertices[f[r+2][0]-1][0]/max,vertices[f[r+2][0]-1][1]/max,vertices[f[r+2][0]-1][2]/max);
        r+=3;
}
glEnd();
```

Donde s es la variable que indica el total de filas de la matriz f. Tras la lectura y el pegado de las texturas en la silueta se consigue la imagen final de la figura 9.

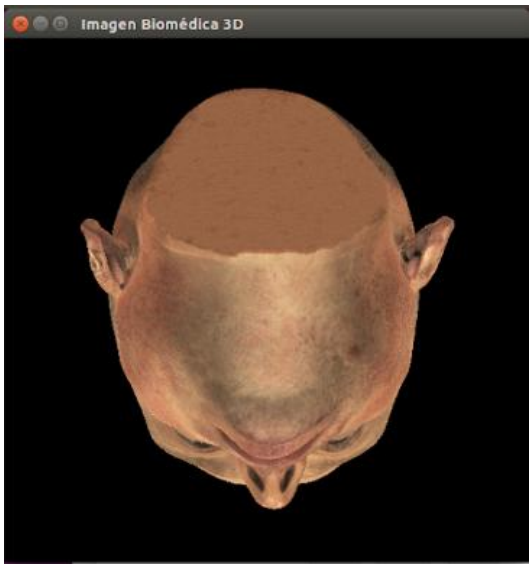


Figura 9: Imagen final

Con esto se termina la función display3 función que se debe invocar cada vez que se quiera dibujar algo en pantalla.

7.4. Creación de la función para la rotación de la imagen

Se observa que la imagen de la figura 9 se ha representado correctamente y que las texturas se han pegado en el sitio correcto. Aun así el ángulo en el que se visualiza la imagen en un comienzo, es poco representativo y sin la habilidad de poder rotar la figura de poco vale visualizar una imagen en tres dimensiones. Es por esto que se procede al diseño de una función que permita visualizar la imagen en distintos ángulos. Para ello se comienza añadiendo en el main del programa la instrucción **glutSpecialFunc(specialKeys);** donde specialKeys será la función que se llamará cada vez que se pulse una tecla. La función será la siguiente:

```
void specialKeys( int key, int x, int y ) {
  if (key == GLUT_KEY_RIGHT)
    rotate_y += 5;
  else if (key == GLUT_KEY_LEFT)
    rotate_y -= 5;
  else if (key == GLUT_KEY_UP)
    rotate_x += 5;
  else if (key == GLUT_KEY_DOWN)
    rotate_x -= 5;
  glutPostRedisplay();
}
```

Esta función no tiene apenas complejidad. Se lee la tecla pulsada y se compara con las teclas que proporciona la extensión GLUT. En este caso se opta por utilizar las flechas del teclado para la acción de rotar la imagen. Si se pulsa la tecla que indica la derecha se incrementa el valor de la variable rotate_y en 5 grados. Si por el contrario se pulsa la flecha con la dirección hacia la izquierda a rotate_y se le restarán 5 grados. Lo mismo ocurro con las teclas arriba y abajo, solo que estas modificarán la variable rotate_x. Finalmente, con la instrucción **glutPostRedisplay();** se vuelve a dibujar la ventana con las modificaciones insertadas.

Para que esta variable tenga utilidad se necesita añadir a la función display3 ciertos comandos que modifiquen el ángulo desde el que se mira la imagen en 3D, justo antes de representar los vértices en el display:

glMatrixMode(GL_MODELVIEW);
Se especifica que se quiere variar la matriz relacionada con la vista y aplica las operaciones de matriz subsiguientes a la pila de matriz de vista de modelo.

glRotatef(rotate_x,1.0,0.0,0.0);
glRotatef(rotate_y, 0.0, 1.0, 0.0);
Multiplica la matriz actual por una matriz de rotación, donde rotate_x e y representan el ángulo que se quiere rotar la imagen alrededor de los ejes X,Y y Z. Los siguientes 3 campos están referidos a la elección del eje en el que se quiere aplicar la rotación, es decir si se elige el valor 1.0, se aplica la rotación a ese eje, si se selecciona el valor 0.0, en cambio, no.

Con todo esto se consigue la imagen final completa de la figura 10, pudiendo visualizarla desde cualquier ángulo y rotarla con total libertad:

Figura 10: Imagen final rotada

Para concluir se añade **glutMainLoop();** en el main para que el programa entre en un bucle y no concluya hasta que se pulse la x en la ventana de la visualización.

8. Descripción de tareas

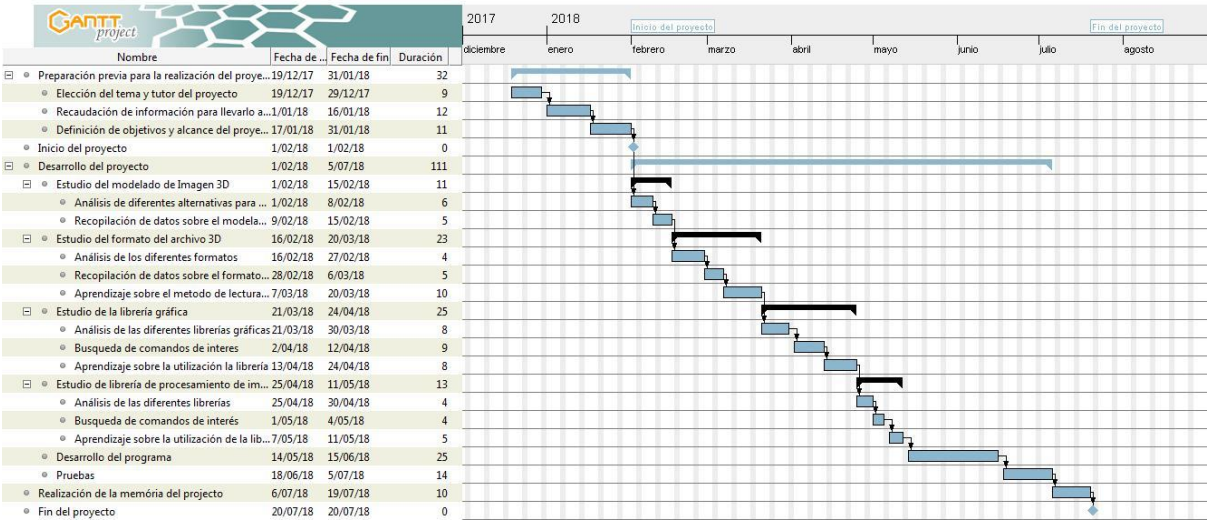
A continuación, se presentan las tareas en las que se ha dividido el proyecto:

Tabla 2: Tareas relacionadas con el proyecto

EDT	Nombre de tarea	Duración	Comienzo	Fin
PT-1	**Preparación previa para la realización del proyecto**	**32 días**	**19/12/17**	**31/01/18**
T-1.1	Elección del tema y tutor del proyecto	9 días	19/12/17	29/12/17
T-1.2	Recaudación de información para llevarlo a cabo	12 días	01/01/18	16/01/18
T-1.3	Definición de objetivos y alcance del proyecto	11 días	17/01/18	31/01/18
H-1	Inicio del proyecto	0 días	01/02/18	01/02/18
PT-2	**Desarrollo del proyecto**	**111 días**	**01/02/18**	**05/07/18**
T-2.1	**Estudio del modelado de Imagen 3D**	**11 días**	**01/02/18**	**15/02/18**
T-2.1.1	Análisis de diferentes alternativas para el modelado	6 días	01/02/18	08/02/18
T-2.1.2	Recopilación de datos sobre el modelado elegido	5 días	09/02/18	15/02/18
T-2.2	**Estudio del formato del archivo 3D**	**23 días**	**16/02/18**	**20/03/18**
T-2.2.1	Análisis de los diferentes formatos	8 días	16/02/18	27/02/18
T-2.2.2	Recopilación de datos sobre el formato elegido	5 días	28/02/18	06/03/18
T-2.2.3	Aprendizaje sobre el método de lectura del formato	10 días	07/03/18	20/03/18

T-2.3	Estudio de la librería gráfica	25 días	21/03/18	24/04/18
T-2.3.1	Análisis de las diferentes librerías gráficas	8 días	21/03/18	30/03/18
T-2.3.2	Búsqueda de comandos de interés	9 días	02/04/18	12/04/18
T-2.3.3	Aprendizaje sobre la utilización la librería	8 días	13/04/18	24/04/18
T-2.4	**Estudio de librería de procesamiento de imagen**	**13 días**	**25/04/18**	**11/05/18**
T-2.4.1	Análisis de las diferentes librerías	4 días	25/04/18	30/04/18
T-2.4.2	Búsqueda de comandos de interés	4 días	01/05/18	04/05/18
T-2.4.3	Aprendizaje sobre la utilización de la librería	5 días	07/05/18	11/05/18
T-2.5	**Desarrollo del programa**	**25 días**	**14/05/18**	**15/06/18**
T-2.6	**Pruebas**	**14 días**	**18/06/18**	**05/07/18**
PT-3	**Realización de la memoria del proyecto**	**10 días**	**06/07/18**	**19/07/18**
H-2	Fin del proyecto	0 días	20/07/18	20/07/18

El proyecto da un total de 153 días para una sola persona realizándolo, comenzando este el 19 de diciembre del 2017 y acabándolo el 20 de julio del 2018.



Nombre	Fecha de ...	Fecha de fin	Duración
Preparación previa para la realización del proye...	19/12/17	31/01/18	32
Elección del tema y tutor del proyecto	19/12/17	29/12/17	9
Recaudación de información para llevarlo a...	1/01/18	16/01/18	12
Definición de objetivos y alcance del proye...	17/01/18	31/01/18	11
Inicio del proyecto	1/02/18	1/02/18	0
Desarrollo del proyecto	1/02/18	5/07/18	111
Estudio del modelado de Imagen 3D	1/02/18	15/02/18	11
Análisis de diferentes alternativas para ...	1/02/18	8/02/18	6
Recopilación de datos sobre el modela...	9/02/18	15/02/18	5
Estudio del formato del archivo 3D	16/02/18	20/03/18	23
Análisis de los diferentes formatos	16/02/18	27/02/18	4
Recopilación de datos sobre el formato...	28/02/18	6/03/18	5
Aprendizaje sobre el metodo de lectura...	7/03/18	20/03/18	10
Estudio de la librería gráfica	21/03/18	24/04/18	25
Análisis de las diferentes librerías gráficas	21/03/18	30/03/18	8
Busqueda de comandos de interes	2/04/18	12/04/18	9
Aprendizaje sobre la utilización la librería	13/04/18	24/04/18	8
Estudio de librería de procesamiento de im...	25/04/18	11/05/18	13
Análisis de las diferentes librerías	25/04/18	30/04/18	4
Busqueda de comandos de interés	1/05/18	4/05/18	4
Aprendizaje sobre la utilización de la lib...	7/05/18	11/05/18	5
Desarrollo del programa	14/05/18	15/06/18	25
Pruebas	18/06/18	5/07/18	14
Realización de la memória del projecto	6/07/18	19/07/18	10
Fin del proyecto	20/07/18	20/07/18	0

Figura 11: Diagrama de Gantt

9. Análisis de costes

9.1. Horas internas

Dentro de recursos humanos se sitúa el apartado de horas internas, en donde se ha supuesto que el trabajo lo realiza una sola persona graduada en tecnología de telecomunicación. Se han utilizado las siguientes consideraciones:

- El sueldo por hora de un ingeniero técnico en España que trabaja como autónomo para un proyecto, es de aproximadamente 35€/h.
- El trabajo que se ha realizado se lleva a cabo en unas 250 horas aproximadamente.
- Si se calcula el total del coste de las horas internas, multiplicando las horas que ha trabajado el graduado en el proyecto por lo que cobra este por horas, se obtiene un total de 8,750€.

Tabla 3: Horas internas

HORAS INTERNAS				
	Cantidad	Coste/hora (€)	Horas	Coste total (€)
Ingeniero Técnico	1	35,00	250	8,750
Total:				**8,750€**

9.2. Amortizaciones

Puesto que en este proyecto solo se ha utilizado un ordenador y todas las librerías utilizadas son de libre uso, solo se tiene en cuenta el coste de este y la licencia de Microsoft Office.

- Se utiliza un ordenador de gama alta puesto que la realización del programa y las renderizaciones probadas necesitan de un alto requerimiento de CPU y tarjeta gráfica. Se utiliza un MSI GS65 Stealth Thin 8RE-025ES valorado en 2.119,03 euros adquirido en 2016.
- La vida útil del ordenador se considera de 2 años, que el uso de este es de unos 320 días al año durante 8 horas al día. Si se calcula la vida útil de este equipo en horas, da un total de 5.120.
- Se divide el coste inicial entre la vida útil calculada en el apartado anterior, lo que da una tasa horaria de unos 0,41€
- Dado que se ha dicho anteriormente que el ingeniero ha trabajado 250 horas en este proyecto y como todas las horas de este proyecto se han realizado utilizando el ordenador, el coste de este será de unos 103,47€

- La licencia de Microsoft Office utilizada se adquirió junto al ordenador en 2016 y su coste fue de 99,0€.
- Se estima que esta aplicación se desactualiza en 5 años, con esto y sabiendo que un ingeniero en la empresa la utiliza durante 42 días al año, escribiendo informes sobre aplicaciones, utilizando 3 horas cada día para ello, da una vida útil de 630 horas.
- Como se ha realizado con el ordenador, se divide el coste inicial del programa entre su vida útil, quedando una tasa horaria de 0,16€/h.
- Se sabe que el ingeniero a realizado el informe del proyecto en 82 horas, por lo que el coste será de unos 13,12€.
- Teniendo en cuenta los costes del ordenador y de la licencia de Microsoft Office, el coste total de las amortizaciones del proyecto asciende a 116,59€.

Tabla 4: Amortizaciones

AMORTIZACIONES						
	Cantidad	Coste (€)	Vida útil (h)	Tasa horaria (€/h)	Uso (h)	Coste total (€)
Ordenador	1	2.119,03	5120	0,41	250	103,47
Microsoft Office	1	99,00	630	0,16	82	13,12
Total:						**116,59€**

9.3. Gastos

Se tiene en cuenta el gasto material del proyecto el cual se basa, prácticamente al completo, en la impresión de las imágenes en papel para demostrar los resultados obtenidos a ciertos clientes.

- Este gasto material incluye papel, impresión de las imágenes y material para escribir… Y asciende a un total de 60€

Tabla 5: Gastos

GASTOS	
	Coste total (€)
Material de oficina	60
Total:	**60€**

9.4. Coste total

Se realiza la suma de las horas internas, las amortizaciones y los gastos del proyecto, logrando el coste total de este.

Tabla 6: Coste total

COSTE TOTAL	
Horas internas	8,750 €
Amortizaciones	116,59€
Gastos	60€
Total:	**8923,59€**

10. Conclusiones

Una vez realizado el proyecto se aprecia que se han cumplido los objetivos que se exponían al principio de este documento. Se ha logrado realizar una aplicación la cual visualiza imágenes en 3D desde cero, haciendo frente a los múltiples problemas que se han tenido para la correcta realización de ésta. Se han empleado métodos de programación estudiados en el grado mejorando y extendiendo los conocimientos que se tenían sobre ellos en un principio. Se ha aprendido mucho sobre el mundo virtual en tres dimensiones, conociendo y aprendiendo sobre diferentes formatos existentes hoy día, y sobre la relación tan estrecha que tiene este mundo virtual con un ámbito tan importante como el de la medicina. De esto se han deducido unas conclusiones:

- Una correcta herramienta de visualización de imágenes 3D en un ámbito tan importante como la medicina puede ser un factor importante para descubrir enfermedades difíciles de apreciar en imágenes 2D.

- En un futuro muchas de las pruebas de diagnóstico de pacientes, la cuales devuelven imágenes, se visualizarán en 3D debido a la comodidad que éstas ofrecen para el profesional.

- La visualización 3D no solo es útil en el diagnóstico de enfermedades, sino también a la hora de la preparación para una operación quirúrgica o en el mismo quirófano a la hora de intervenir.

Se ha aprendido a comparar diferentes alternativas, eligiendo la más idónea por múltiples razones y aprendiendo sobre ellas aunque fuesen descartadas en un final, de esta forma se han adquirido conocimientos sobre tipos de modelados 3D, tipos de formatos de archivos 3D, tipos de librerías gráficas y tipos de librerías para el procesamiento de imágenes.

Todo el proyecto se ha podido realizar con un ordenador portátil, lo que demuestra que hoy día gracias a la gran evolución de los computadores, las posibilidades para realizar proyectos interesantes son elevadas y de poco coste. Se han utilizado diferentes librerías para realizar el visualizador, aprendiendo sobre la instalación, el compilado y el uso de éstas, adquiriendo conocimientos elevados de ellas y logrando metas que al comienzo del proyecto no sabía que eran alcanzables con tan solo un ordenador.

En vistas al futuro, éste puede ser el primer paso de mi persona hacia la creación de herramientas 3D más completas, las cuales tengan en cuenta las diferentes físicas de los órganos y los diferentes movimientos de estos, logrando una imagen 3D más completa.

Referencias

[1] Víctor Pérez Sainz, «Herramienta para modelado estocástico y visualización 3D del sistema de conducción del corazón », Septiembre 2011. https://www.uv.es/rasea3/docs/VPerezDiploma2011.pdf

[2] Planeta vivo, « ¿Ver pelis en 3D potencia nuestro cerebro? » https://planetavivo.cienradios.com/ver-pelis-en-3d-potencia-nuestro-cerebro/

[3] Grupo de Bioingeniería y Telemedicina, Universidad Politécnica de Madrid «Imagen biomédica» http://www.gbt.tfo.upm.es/Imagen+biom%C3%A9dic

[4] Gonzalo Arrondo «Visualización de modelos digitales tridimensionales en la enseñanza de anatomía: principales recursos y una experiencia docente en neuroanatomía» https://www.sciencedirect.com/science/article/pii/S1575181316301048

[5] Wikipedia, «Impresión 3D», Julio 2018. https://es.wikipedia.org/wiki/Impresi%C3%B3n_3D

[6] Fundación ONCE, «Capacidades cognitivas – Accesibilidad Cognitiva urbana », 2012. http://accesibilidadcognitivaurbana.fundacionononce.es/capacidadesCognitivas.aspx

[7] García Fenoll, Ignacio, «Aportaciones a Segmentación y Caracterización de Imágenes Medicas 3D», http://bibing.us.es/proyectos/abreproy/11854/direccion/Volumen+1%252F

[8] Elena G. Sevillano, «Casi una de cada tres resonancias magnéticas lumbares son innecesarias», Abril 2013. https://elpais.com/sociedad/2013/04/10/actualidad/1365590905_078982.html

[9] El Mundo, «España, segundo país con la tasa de abandono escolar más alta de la UE, sólo por detrás de Malta », Abril 2017. http://acodigo.blogspot.com.es/2016/04/cargar-modelo-3d-formato-obj.html

[10] Wikipedia, «Modelado 3D», Junio 2018. https://es.wikipedia.org/wiki/Modelado_3D

[11] ADM, «Modelado tridimensional», Marzo 2015. https://www.web20.co/desarrollo/modelado-tridimensional/

[12] OpenTheFile, «¿Qué es un archivo obj y cómo puedo abrir un archivo obj?». https://www.openthefile.net/es/extension/obj

[13] OpenTheFile, «¿Qué es un archivo fbx y cómo puedo abrir un archivo fbx?». https://www.openthefile.net/es/extension/fbx

[14] DownloadAstro, «Archivos de Imagen 3D». https://es.downloadastro.com/Archivos%20de%20Windows/Archivos%20de%20Imagen%203D/

[15] Wikipedia, «OpenGL», Abril 2018. https://es.wikipedia.org/wiki/OpenGL

[16] Abel Morón Barroso, «Representación Tridimensional de Imágenes Termográficas». http://deeea.urv.cat/public/PROPOSTES/pub/pdf/1307pub.pdf

[17] Wikipedia, «Direct3D», Febrero 2018. https://es.wikipedia.org/wiki/Direct3D

[18] VTK, «Visualize Your Data With VTK». https://www.vtk.org/

[19] ImageMagick, «ImageMagick», 2018. https://www.imagemagick.org/script/index.php

[20] Wikipedia, «OpenCV», Febrero 2018. https://es.wikipedia.org/wiki/OpenCV

[21] FreeImage, «What is FreeImage», 2015. http://freeimage.sourceforge.net/

[22] Tutor de programación, «Cargar Modelo 3D formato OBJ », Abril 2016. http://acodigo.blogspot.com.es/2016/04/cargar-modelo-3d-formato-obj.html

[23] OpenGL Refpages, «OpenGL». https://www.khronos.org/registry/OpenGL-Refpages/gl2.1

Anexos

1. Compilación del programa

Se deben añadir las librerías utilizadas en la compilación del programa de la siguiente manera.

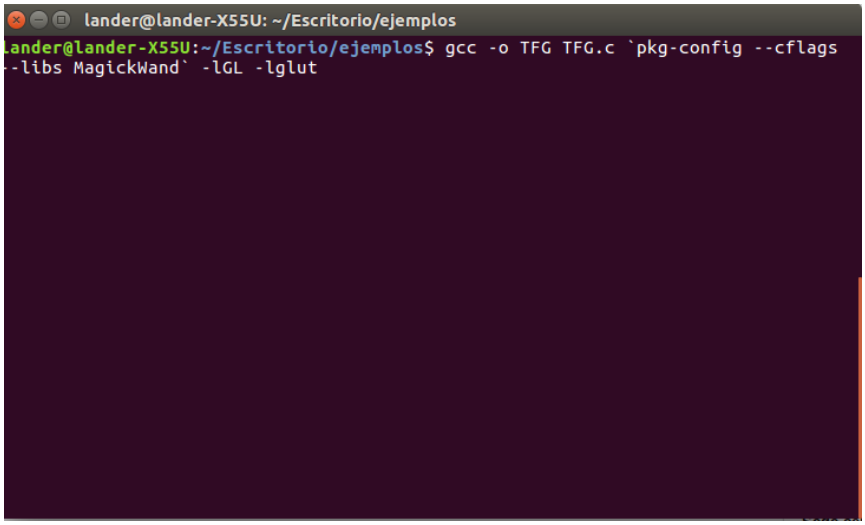


Figura 12: Compilación del programa

2. Ejecución del programa

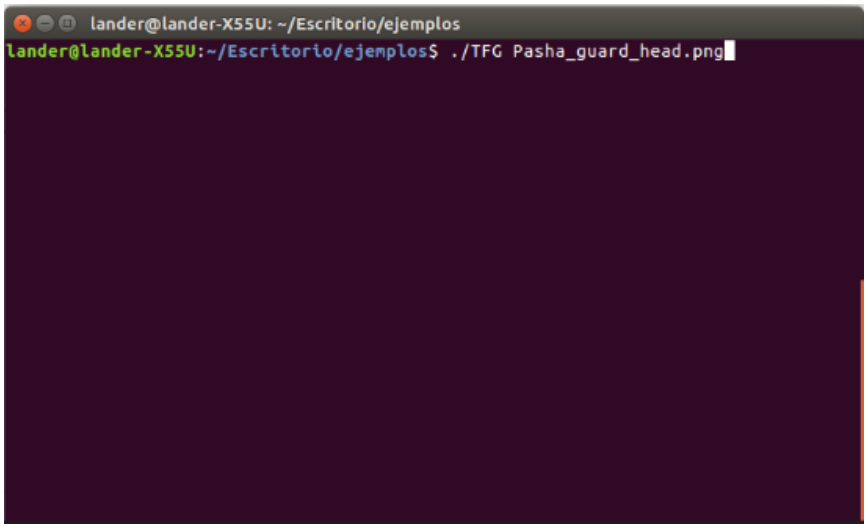


Figura 13: Ejecución del programa

Donde Pasha_guard_head.png es el nombre del archivo de texturas de la imagen que se quiere abrir. Tanto el archivo .obj como el archivo de texturas (.png o .jpg) deben tener el mismo nombre para que el programa se ejecute correctamente.

3. Prueba de funcionamiento con imagen diferente

Se prueba el correcto funcionamiento de la aplicación representando otra imagen 3D diferente de la utilizada.

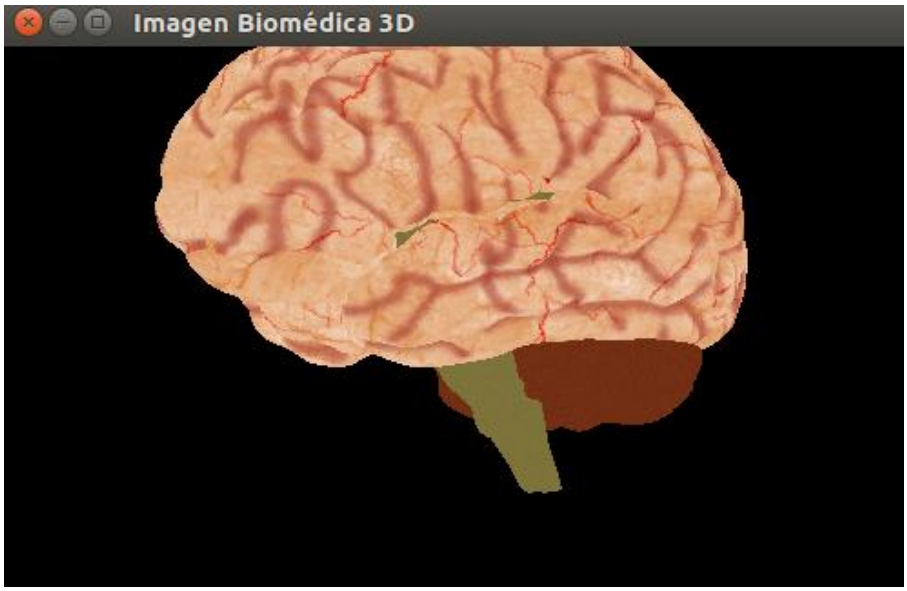


Figura 14: Cerebro humano 1

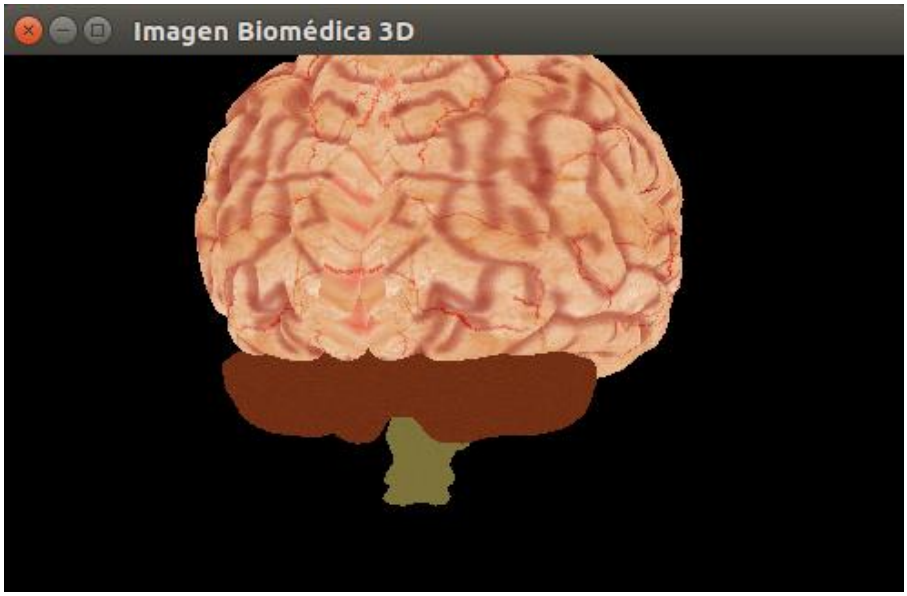


Figura 15: Cerebro humano 2

4. Programa final completo

```
#include <stdio.h>
#include <stdarg.h>
#include <math.h>
#include <string.h>
#include <GL/glew.h>
#include <GL/gl.h>
#include <stdlib.h>
#include <MagickWand/MagickWand.h>
#include <MagickCore/MagickCore.h>
#define GL_GLEXT_PROTOTYPES
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif


void display();
void display2();
void display3();
void specialKeys( int key, int x, int y );
void texturas();
unsigned char *reservarmat(int num);
int **reservarmatint(int fil,int col);
void contador(int *contadorV, int *contadorVT,int *contadorVN,int
*contadorF);
int leerarchivo();


double rotate_y=0;
double rotate_x=0;
int max=0;
int contbarra=0;
int desactivar=0;


float vertices[900000][3];
float vt[900000][2];
float vn[900000][3];
int f[900000][3];

char *data;
char nombre[20];


int main(int argc, char** argv){
      int contadorV=0,contadorVT=0,contadorVN=0,contadorF=0;
      glutInit(&argc,argv);
      strcpy(nombre,argv[1]);
      contador(&contadorV,&contadorVT,&contadorVN,&contadorF);
      glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(0,0);
      glutCreateWindow("Imagen BiomÃ©dica 3D");
      glEnable(GL_DEPTH_TEST);
```

```
	glutDisplayFunc(display3);
	glutSpecialFunc(specialKeys);
	glutMainLoop();

	return 0;
}



void display3(){
	FILE *fich;
	int s,r=0;
	unsigned long width,height;
	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_MODELVIEW);
	glRotatef( rotate_x, 1.0, 0.0, 0.0 );
	glRotatef( rotate_y, 0.0, 1.0, 0.0 );
	s=leerarchivo();
	texturas(&width,&height);
	if(contbarra==5 || contbarra==8){
	glBegin(GL_TRIANGLES);
	while(r<s){//forma triangulos
		glTexCoord2f(vt[f[r][1]-1][0],vt[f[r][1]-1][1]);
		glVertex3f(vertices[f[r][0]-1][0]/max,vertices[f[r][0]-
1][1]/max,vertices[f[r][0]-1][2]/max);
		glTexCoord2f(vt[f[r+1][1]-1][0],vt[f[r+1][1]-1][1]);
		glVertex3f(vertices[f[r+1][0]-1][0]/max,vertices[f[r+1][0]-
1][1]/max,vertices[f[r+1][0]-1][2]/max);
		glTexCoord2f(vt[f[r+2][1]-1][0],vt[f[r+2][1]-1][1]);
		glVertex3f(vertices[f[r+2][0]-1][0]/max,vertices[f[r+2][0]-
1][1]/max,vertices[f[r+2][0]-1][2]/max);
		r+=3;
	}
	glEnd();
	}else if (contbarra== 6|| contbarra==10){//forma cuadrilateros
	glBegin(GL_QUADS);
	while(r<s){
		glTexCoord2f(vt[f[r][1]-1][0],vt[f[r][1]-1][1]);
		glVertex3f(vertices[f[r][0]-1][0]/max,vertices[f[r][0]-
1][1]/max,vertices[f[r][0]-1][2]/max);
		glTexCoord2f(vt[f[r+1][1]-1][0],vt[f[r+1][1]-1][1]);
		glVertex3f(vertices[f[r+1][0]-1][0]/max,vertices[f[r+1][0]-
1][1]/max,vertices[f[r+1][0]-1][2]/max);
		glTexCoord2f(vt[f[r+2][1]-1][0],vt[f[r+2][1]-1][1]);
		glVertex3f(vertices[f[r+2][0]-1][0]/max,vertices[f[r+2][0]-
1][1]/max,vertices[f[r+2][0]-1][2]/max);
		glTexCoord2f(vt[f[r+3][1]-1][0],vt[f[r+3][1]-1][1]);
		glVertex3f(vertices[f[r+3][0]-1][0]/max,vertices[f[r+3][0]-
1][1]/max,vertices[f[r+3][0]-1][2]/max);
		r+=4;

	}
	glEnd();
	}
	free(data);
	glFlush();
```

```
	glutSwapBuffers();
}

int leerarchivo(){
	FILE *fich;
	char linea[1024];
	char letra[10], cad[20], cadpre[20];
	int n=0,l=0,m=0, s=0,i;
	strcpy(cadpre,nombre);
	cadpre[strlen(cadpre)-4]='\0';
	sprintf(cad,"%s.obj",cadpre);
	fich=fopen(cad,"r");
	if(fich!=NULL){
	while(fgets(linea,1024,fich)!=NULL){
		sscanf(linea,"%s",letra);
		if(strcmp(letra,"v")==0){
			sscanf(linea,"%s %f %f
%f",letra,&vertices[n][0],&vertices[n][1],&vertices[n][2]);
			n++;
			if(vertices[n][0]>max && vertices[n][0]>1){
				max=vertices[n][0];
			}
			if(vertices[n][1]>max && vertices[n][1]>1){
				max=vertices[n][1];
			}
			if(vertices[n][2]>max && vertices[n][2]>1){
				max=vertices[n][2];
			}
		}else if(strcmp(letra,"vt")==0){
			sscanf(linea,"%s %f %f",letra,&vt[l][0],&vt[l][1]);
			l++;

		}else if(strcmp(letra,"vn")==0){
			sscanf(linea,"%s %f %f
%f",letra,&vn[m][0],&vn[m][1],&vn[m][2]);
			m++;
		}else if(strcmp(letra,"f")==0){

			if(desactivar==0){
			for(i=0;i<1024;i++){
				if(linea[i]=='/'){
					contbarra++;

					}
				}
				desactivar=1;
			}
			if(contbarra==5){
				sscanf(linea,"%s %d/%d %d/%d
%d/%d",letra,&f[s][0],&f[s][1],&f[s+1][0],&f[s+1][1],&f[s+2][0],&f[s+2
][1]);
				s+=3;
			}else if(contbarra==8){
				sscanf(linea,"%s %d/%d/%d %d/%d/%d
%d/%d/%d",letra,&f[s][0],&f[s][1],&f[s][2],&f[s+1][0],&f[s+1][1],&f[s+
1][2],&f[s+2][0],&f[s+2][1],&f[s+2][2]);
				s+=3;
```

```
                  }else if(contbarra==6 ){
                        sscanf(linea,"%s %d/%d %d/%d %d/%d
%d/%d",letra,&f[s][0],&f[s][1],&f[s+1][0],&f[s+1][1],&f[s+2][0],&f[s+2
][1],&f[s+3][0],&f[s+3][1]);
                        s+=4;
                  }else if(contbarra==10){
                        sscanf(linea,"%s %d/%d/%d %d/%d/%d %d/%d/%d
%d/%d/%d",letra,&f[s][0],&f[s][1],&f[s][2],&f[s+1][0],&f[s+1][1],&f[s+
1][2],&f[s+2][0],&f[s+2][1],&f[s+2][2],&f[s+3][0],&f[s+3][1],&f[s+3][2
]);
                        s+=4;
                  }
            }
      }
      }
      fclose(fich);
      return s;
}

void contador(int *contadorV, int *contadorVT,int *contadorVN,int
*contadorF){
      FILE *fich;
      char linea[1024], cad[20], cadpre[20];
      char letra[3];
      strcpy(cadpre,nombre);
      cadpre[strlen(cadpre)-4]='\0';
      sprintf(cad,"%s.obj",cadpre);
      fich=fopen(cad,"r");
      if(fich!=NULL){
      while(fgets(linea,1024,fich)!=NULL){
            sscanf(linea,"%s",letra);
            if(strcmp(letra,"v")==0){
                  *contadorV=*contadorV+1;
            }else if(strcmp(letra,"vt")==0){
                  *contadorVT=*contadorVT+1;
            }else if(strcmp(letra,"vn")==0){
                  *contadorVN=*contadorVN+1;
            }else if(strcmp(letra,"f")==0){
                  *contadorF=*contadorF+1;
            }
      }
      }
      printf("Número de vertices: %d\n",*contadorV);
      printf("Número de valores de texturas: %d\n",*contadorVT);
      printf("Número de vertices normalizados: %d\n",*contadorVN);
      printf("Número de caras: %d\n",*contadorF);
      fclose(fich);
}

void texturas(unsigned long *width, unsigned long *height){
      int i;
      GLuint texture;
      char cad[20];
      MagickWandGenesis();
      MagickWand *wand = NewMagickWand();
      MagickBooleanType status = MagickReadImage(wand,nombre);
      if (status != MagickFalse){
```

```
            *width = MagickGetImageWidth(wand);
            *height = MagickGetImageHeight(wand);
            MagickFlipImage(wand);
            glGenTextures(1, &texture);
            glBindTexture(GL_TEXTURE_2D, texture);
            data=reservarmat(*width*(*height)*3);
            MagickExportImagePixels(wand, 0,0, *width, *height, "RGB",
CharPixel, data);
            glTexImage2D(GL_TEXTURE_2D,0,GL_RGB, *width, *height, 0,
GL_RGB, GL_UNSIGNED_BYTE, data);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
            glEnable(GL_TEXTURE_2D);
      }
}

unsigned char *reservarmat(int num){
      unsigned char *mat;
      mat=(unsigned char*)malloc(num*sizeof(unsigned char));
      return mat;
}


void specialKeys( int key, int x, int y ) {
  if (key == GLUT_KEY_RIGHT)
    rotate_y += 5;
  else if (key == GLUT_KEY_LEFT)
    rotate_y -= 5;
  else if (key == GLUT_KEY_UP)
    rotate_x += 5;
  else if (key == GLUT_KEY_DOWN)
    rotate_x -= 5;
  glutPostRedisplay();
}
```