UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

(Universidad del Perú, DECANA DE AMÉRICA)

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

Escuela Académico Profesional de Ingeniería de Software

TEMA: "Proyecto Final del Curso"

PROFESOR: PAÚCAR CURASMA, Herminio

CURSO: Computación Gráfica y Visual

INTEGRANTES:

[3]Azorsa Salazar, Stephanie	17200317
[3]Mincia Retamozo, Alessandra	17200289
[3]Marroquín Gavelan, Juan Patricio	17200329
[3]Osco Pupe, Jean Williams	17200082
[3]Rojas Camargo, Melinna	17200106
[3]Velazco Huere, Leydi Mabel	17200307

LIMA-PERÚ

2020

"Año de la universalización de la salud"

Dedicado a todos nuestros padres y hermanos, por su apoyo incondicional durante esta larga etapa de investigación y desarrollo.

Índice

1. Introducción

1.1. Motivación

Este proyecto cuenta como trasfondo y motivación el gran crecimiento que hoy en día está teniendo el uso de la computación gráfica.

Al estar presente en grandes producciones como películas y videoclips, al estar presente en videojuegos ambiciosos y en la realidad virtual, al estar presente incluso en proyectos que lo combinan con IA o ML; definitivamente estamos hablando de un campo de las ciencias de la computación que ha crecido exponencialmente y que nos lleva a nuevos retos.

1.1.1. Necesidad de los Visores y Simuladores 3D

Hoy en día es posible la simulación mediante cálculos basados en la proyección de entornos tridimensionales sobre pantallas bidimensionales, como monitores o televisores. Estos cálculos requieren de una gran carga de proceso por lo que algunas computadoras y videoconsolas disponen de cierto grado de aceleración gráfica 3D gracias a dispositivos desarrollados para tal fin. Las computadoras disponen de las llamadas tarjetas gráficas con aceleración 3D.

Estos dispositivos están formados por uno o varios procesadores (unidad de procesamiento gráfico) diseñados especialmente para acelerar los cálculos que suponen reproducir imágenes tridimensionales sobre una pantalla bidimensional y de esta forma liberar de carga de proceso a la unidad central de procesamiento de la computadora.

1.1.2. Crecimiento de la Computación Gráfica

Todos los días se habla de la potencia y la importancia que tiene el mercado de la computación gráfica en la actualidad y cómo influye en prácticamente todas las esferas de nuestra vida. Ahora recientes estudios demuestran, que efectivamente es un mercado está creciendo y que lo seguirá haciendo.

Según un estudio realizado por Jon Peddie Research, las ventas relacionadas con todo lo que tenga que ver con el software de computación gráfica y su correspondiente hardware crecerá en los próximos años lo que revela un cambio en un mercado que debería estar cayendo debido a la situación de la economía global y de la caída que han tenido todas las ventas relacionadas con el hardware.

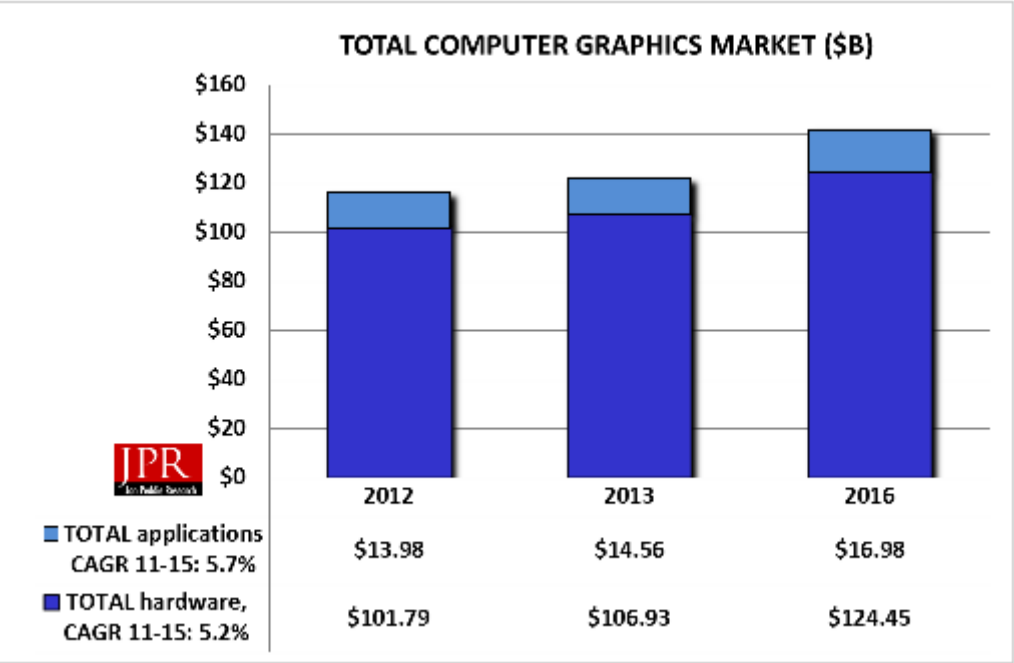
La creciente demanda de dispositivos móviles y un renovado interés en los videojuegos y diferentes tipos de aplicaciones atractivas han sido los factores que están haciendo que los productos de computación gráfica vean crecer su mercado. Este tipo de tecnología está mejorando y cada vez hay más demanda de aplicaciones especiales para películas, fotografía. Además ha aumentado el número de empresas dedicadas a la ingeniería que requieren productos de simulación gráfica, para poder «ver primero» en sus ordenadores como quedarán sus construcciones.

Todo esto apunta a que cada vez serán más las empresas que precisarán contratar diseñadores gráficos, científicos, artistas y programadores en los próximos años.

"*Estamos viendo nuevas oportunidades de crecimiento que se deben al aumento en el uso de aplicaciones para web y para consumidores. La web está creciendo como medio de distribución para los contenidos gráficos, que a su vez estimulan a que las personas utilicen nuevas herramientas, creen contenido por placer e incluso apuesten por encontrar trabajo en este campo"* (Research, 2019)

El mercado para hardware y software de computación gráfica crecerá este año cerca de 121.500 millones de dólares y se calcula que para el 2016 llegue a los 142.000 millones de dólares. En estos números no se incluyen los servicios de mantenimiento relacionados con este tipo de productos gráficos.

De este mercado, el que corresponde al hardware es mucho más grande comparado con el del software. Las ventas que corresponden al hardware para computación gráfica, entre las que se incluyen videoconsolas, componentes gráficos para dispositivos móviles… suman un total de 106.930 millones de dólares solo este año. Las ventas del software incluyen programas de video, simulación, CAD/CAM, animación y modelación. Todos estos programas sumaron un total de 14.560 millones de dólares este año.



1.2. Problema

El problema planteado para este proyecto fue el realizar un visualizador de objetos 3D programado utilizando C++ y librerías como OpenGL, GLSL y Qt para el manejo de su interfaz gráfica.

Para agregar complejidad a este proyecto, se plantea el uso de elementos gráficos en su interfaz para poder girar, rotar, ampliar, trasladar y transformar los objetos que se muestren según la necesidad del usuario.

2. Marco Teórico

2.1. Lenguaje de Programación

Para el siguiente proyecto, se utilizó C++ como lenguaje de programación.

2.1.1. C++

C++ es un lenguaje de programación que proviene de la extensión del lenguaje C para que pudiese manipular objetos. A pesar de ser un lenguaje con muchos años, su gran potencia lo convierte en uno de los lenguajes de programación más demandados en nuestra actualidad.

Si se busca programar en alto nivel, con la opción de poder bajar incluso a ensamblador, C++ sigue siendo una gran opción, ya que estamos ante un lenguaje de programación muy potente y que se ha mantenido actualizado.

La programación de videojuegos sigue siendo una profesión en auge y en ella el conocimiento de Unity y C++ es muy valorado.

Fue diseñado a mediados de los años 80 por el danés **Bjarne Stroustrup**. Si intención fue la de extender el lenguaje de programación C (*con mucho éxito en ese momento*) para que tuviese los mecanismos necesarios para manipular objetos. Por lo tanto, C++ contiene los paradigmas de la programación estructurada y orientada a objetos, por lo que se le conoce como un lenguaje de programación multiparadigma.

A C++ primero se le conoció como “C con clases”. Luego se cambió a C++ que significa “incremento de C”, dando a entender que se trata de una extensión del lenguaje de programación C.

Las principales ventajas de programar en C++ son:

- Alto rendimiento: Es una de sus principales características, el alto rendimiento que ofrece. Esto es debido a que puede hacer llamadas directas al sistema operativo, es un lenguaje compilado para cada plataforma, posee gran variedad de parámetros de optimización y se integra de forma directa con el lenguaje ensamblador.
- Lenguaje actualizado: A pesar de que ya tiene muchos años, el lenguaje se ha ido actualizando, permitiendo crear, relacionar y operar con datos complejos y ha implementado múltiples patrones de diseño.
- Multiplataforma
- Extendido: C y C++ están muy extendidos. Casi cualquier programa o sistema están escritos o tienen alguna parte escrita en estos lenguajes (desde un navegador web hasta el propio sistema operativo).

Las principales desventajas de C++ es que se trata de un lenguaje muy amplio (con muchos años y muchas líneas de código), se debe tener una compilación por plataforma y su depuración se complica debido a los errores que surgen. Además, el manejo de librerías es más complicado que otros lenguajes como Java o .Net y su curva de aprendizaje muy alta.

2.2. IDE

Para este proyecto, se utilizó Eclipse como IDE para el desarrollo.

2.2.1. Eclipse IDE

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL).

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente con estos lenguajes, ya que soporta otros lenguajes de programación.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".

2.3. Librerías

2.3.1. OpenGL

Fundamentalmente OpenGL es una especificación, es decir, un documento que describe un conjunto de funciones y el comportamiento exacto que deben tener. Partiendo de ella, los fabricantes de hardware crean implementaciones, que son bibliotecas de funciones que se ajustan a los requisitos de la especificación, utilizando aceleración hardware cuando es posible. Dichas implementaciones deben superar unos tests de conformidad para que sus fabricantes puedan calificar su implementación como conforme a OpenGL y para poder usar el logotipo oficial de OpenGL.

Hay implementaciones eficientes de OpenGL para Mac OS, Microsoft Windows, GNU/Linux, varias plataformas Unix y PlayStation 4. Existen también varias implementaciones en software que permiten ejecutar aplicaciones que dependen de OpenGL sin soporte de aceleración hardware. Es destacable la biblioteca de software libre / código abierto Mesa 3D, una API de gráficos sin aceleración hardware y completamente compatible con OpenGL. Sin embargo, para evitar los costes de la licencia requerida para ser denominada formalmente como una implementación de OpenGL, afirma ser simplemente una API muy similar.

La especificación OpenGL era revisada por el OpenGL Architecture Review Board (ARB), fundado en 1992. El ARB estaba formado por un conjunto de empresas interesadas en la creación de una API consistente y ampliamente disponible. Microsoft, uno de los miembros fundadores, abandonó el proyecto en 2003.

El 21 de septiembre de 2006 se anunció que el control de OpenGL pasaría del ARB al Grupo Khronos. Con ello se intentaba mejorar el marketing de OpenGL y eliminar las barreras entre el desarrollo de OpenGL y OpenGL ES. ARB se convirtió dentro de Khronos en el OpenGL ARB Working Group. El subgrupo de Khronos que gestiona la especificación de OpenGL se denomina OpenGL ARB Working Group. Para una relación de los miembros que componen el OpenGL ARB Working Group, véase el apartado Miembros del Grupo Khronos. El gran número de empresas con variados intereses que han pasado tanto por el antiguo ARB como por el grupo actual han hecho de OpenGL una API de propósito general con un amplio rango de posibilidades.

Mark Segal y Kurt Akeley fueron los autores de la especificación original de OpenGL. Chris Frazier fue el editor de la versión 1.1. Jon Leech ha editado las versiones desde 1.2 hasta la presente 3.0.

OpenGL tiene dos propósitos esenciales:

- Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas, presentando al programador una API única y uniforme.
- Ocultar las diferentes capacidades de las diversas plataformas hardware, requiriendo que todas las implementaciones soporten la funcionalidad completa de OpenGL (utilizando emulación software si fuese necesario).

El funcionamiento básico de OpenGL consiste en aceptar primitivas tales como puntos, líneas y polígonos, y convertirlas en píxeles. Este proceso es realizado por una pipeline gráfica conocida como Máquina de estados de OpenGL. La mayor parte de los comandos de OpenGL bien emiten primitivas a la pipeline gráfica o bien configuran cómo la pipeline procesa dichas primitivas. Hasta la aparición de la versión 2.0 cada etapa de la pipeline ejecutaba una función prefijada, resultando poco configurable. A partir de la versión 2.0 algunas etapas son programables usando un lenguaje de programación llamado GLSL.

2.3.2. GLM

Matemáticas OpenGL o Opengl Maths es una librería multiplatforma ligera de sólo cabeceras con plantillas C++ (header-only) desarrollada por G-Truc Creation para realizar cálculos complejos de manera sencilla, intuitiva y centralizada. Trabaja con elementos como vectores, matrices, ángulos, cuaternión, etc. Y se encarga de casi todos aquellos cálculos y transformaciones sobre dichos tipos de datos. Principalmente, se usa en conjunto con las nuevas versiones de OpenGL para realizar los cálculos matriciales necesarios para dicha librería.

A finales del 2004, apareció la versión 2.0 de OpenGL, en el cual el núcleo 1.x se despreció (deprecated), una de las principales características de OpenGL: la tubería de funcionalidad fija, o fixed-function pipeline. A partir de este punto, se empezaron a despreciar características principales de la librería, como la manipulación interna de matrices y las estructuras de datos que usaban.

No fue hasta la estandarización y la nueva despreciación en el año 2008 cuando el núcleo de OpenGL 2.x entre disputas internas y aprovechando la desventaja de Windows Vista y Direct3D 10.0, que aparece el nuevo núcleo y estándar de programación de OpenGL, la versión 3.0, es aquí donde finalmente se deja atrás todo rastro de la 'fixed-function pipeline', siendo que antiguamente hasta la publicación de los documentos estándar de OpenGL 2.0 y posteriormente en 2006 de OpenGL 2.1, se venía pidiendo por escrito que se dejara de utilizar y mezclar el

estándar de llamadas de 'fixed-functions' como glMatrixMode como mayormente destacado, entre otras.

La característica que reemplazó a todos estos aspectos fueron los Shaders, que permitieron programar de manera cómoda de las acciones de renderizado. Para complementar los cálculos de las matrices principales que utilizaba OpenGL 1.x, GLM proporcionó funciones sencillas y rápidas, como el cálculo de una matriz inversa o una multiplicación de matrices entre la gran cantidad de contenido que ésta proporciona.

2.3.3. Qt

Qt es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.

Qt es desarrollada como un software libre y de código abierto a través de Qt Project, donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas.4 Anteriormente, era desarrollado por la división de software de Qt de Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega Trolltech, el productor original de Qt, el 17 de junio de 2008.5 Qt es distribuida bajo los términos de GNU Lesser General Public License y otras. Por otro lado, Digia está a cargo de las licencias comerciales de Qt desde marzo de 2011.

Qt es utilizada en KDE, entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros.

2.3.4. GLSL

OpenGL Shading Language (abreviado GLSL o GLslang) es un lenguaje de alto nivel de sombreado con una sintaxis basada en el lenguaje de programación C. Fue creado por la junta de revisión de la arquitectura OpenGL (OpenGL ARB) para ofrecer a los desarrolladores más control sobre la tubería de renderizado sin tener que usar ARB assembly language o lenguajes específicos para cada hardware.

2.4. Paradigma de Programación

Un lenguaje de programación es un lenguaje formal (o artificial, es decir, un lenguaje con reglas gramaticales bien definidas) que le proporciona a una persona, en este caso el programador, la capacidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de una computadora, de manera que se puedan obtener diversas clases de datos o ejecutar determinadas tareas. A todo este conjunto de órdenes escritas mediante un lenguaje de programación se le denomina programa.

2.4.1. Programación Orientada a Objetos

La Programación Orientada a Objetos (POO, en español; OOP, según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

Muchos de los objetos prediseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

Está basada en varias técnicas del sexenio: herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos.

Los objetos son entidades que tienen un determinado "estado", "comportamiento (método)" e "identidad":

La identidad es una propiedad de un objeto que lo diferencia del resto; dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. Hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una "programación estructurada camuflada" en un lenguaje de POO.

La programación orientada a objetos difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada solo se escriben funciones que procesan datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

2.5. Controlador de Versiones

Para controlar las versiones y cambios en nuestro proyecto, se utilizó git y subió nuestro repositorio en github.

2.5.1. Git

Git (pronunciado "guit") es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos

de autor Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

Cuando los desarrolladores hacen un nuevo proyecto, siempre continúan haciéndole modificaciones al código. Incluso después de la puesta en marcha de los proyectos, todavía necesitan actualizar las versiones, corregir errores, agregar nuevas funciones, etc.

El sistema de control de versiones ayuda a registrar los cambios realizados al código. Aún más, registra quién realizó los cambios y puede restaurar el código borrado o modificado.

No hay códigos sobrescritos ya que Git guarda varias copias en el repositorio.

2.5.2. GitHub

Si Git es el corazón de GitHub, entonces Hub es su alma. El hub de GitHub es lo que convierte una línea de comandos, como Git, en la red social más grande para desarrolladores.

Además de contribuir a un determinado proyecto, GitHub le permite a los usuarios socializar con personas de ideas afines. Puedes seguir a las personas y ver qué hacen o con quién se conectan.

GitHub es un sistema de gestión de proyectos y control de versiones de código, así como una plataforma de red social diseñada para desarrolladores. ¿Pero para qué se usa GitHub? Bueno, en general, permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo.

GitHub es también uno de los repositorios online más grandes de trabajo colaborativo en todo el mundo.

3. Metodología de Desarrollo

Al iniciar este nuevo proyecto el grupo dividió las primeras tareas 2 en fases importantes:

- **Investigación de las herramientas GUI que se puedan usar.**
  Tomando en cuenta las recomendaciones del docente, el equipo de dividió en 2 para indagar e investigar cada una de las herramientas sugeridas (GTK, QT).

- **Elección de la herramienta GUI.**
  Luego de la investigación y con la información obtenida el equipo decidió optar por el uso de QT para el manejo de la GUI de nuestro proyecto, esto principalmente por 3 razones:

  - ✓ Fácil integración con OpenGl
  - ✓ Cuenta con una documentación basta y ordenada
  - ✓ Gran cantidad de recursos multimedia en la red, que nos ayudaran como base de la aplicación.

Una vez seleccionada la herramienta para la GUI, se prosiguió a familiarizarse con ella par poder tener mas conocimiento y poder así lograr el objetivo del proyecto,

Pasos para el desarrollo del proyecto

1. **Diseño del GUI**
   Apoyándonos en cualquier herramienta de diseño se crearon los mockups para nuestra aplicación, que posteriormente se tiene que plasmar en QT.

2. **Configuración de entorno de trabajo**
   QT nos brinda algunas librerías de forma nativa, tales como son GLUI (que nos proporciona elementos de control como botones, cajas, etc.), GLFW (encargada del manejo de ventana, ratón, teclado).
   Se tuvo que importar de forma manual la librería GLM, la cual nos proporcionara múltiples funciones matemáticas

3. **Creación del GUI**
   Gracias al entorno de QT la creación de la UI, resulta relativamente fácil, solo se debería respetar el mockup mapeado para nuestra aplicación.

4. **Asociar parte lógica (controlador) al GUI**
   Cada parte de la GUI nos permitirá ejecutar funciones diferentes en tiempo de ejecución, por eso es importante queda una de estas este asociado a un controlador que permita leer los valores que esta pueda expresar, todo esto sucederá dentro de la clase WIDGET.

5. **Creación de clases principales (SCENE, WIDGET)**
   - ✓ Widget: Es la clase que controla directamente la parte funcional de la GUI (eventos al dar click en un botón, checkbuttion, etc) y asu vez esta se relaciona con la clase Scene.

   - ✓ Scene: Es la clase donde ocurre la magia, es la clase donde se instancias los objetos, se hace uso de los sheaders, se manipulan los parámetros (cámara, escala, segmento). Esta clase cuenta con 2 métodos importantes los cuales son:
     - Initialize: Permite inicializar la instancia de cualquier objeto
     - Draw: Función que dibuja o grafica la figura seleccionada

6. **Crear imagen de prueba para verificar funcionalidades**
   Con el fin de verificar la correcta funcionalidad del GUI y del widget, se creará una figura de prueba (Triangle) para corroborar si todas las funciones se ejecutan de forma correcta.

7. **Crear figuras**
   Se diseñarán las figuras a mostrar siguiendo la siguiente estructura:
   - ✓ <figura>.h : Donde se creara la clase y la estructura de esta (atributos y métodos)
   - ✓ <figura>.cpp : Archivo que usara el header para detallar y especificar cada funcionalidad.

   Además la creación de cada figura se hará en base a funciones matemáticas (matriz de rotación, sen, cos, ángulos, etc)

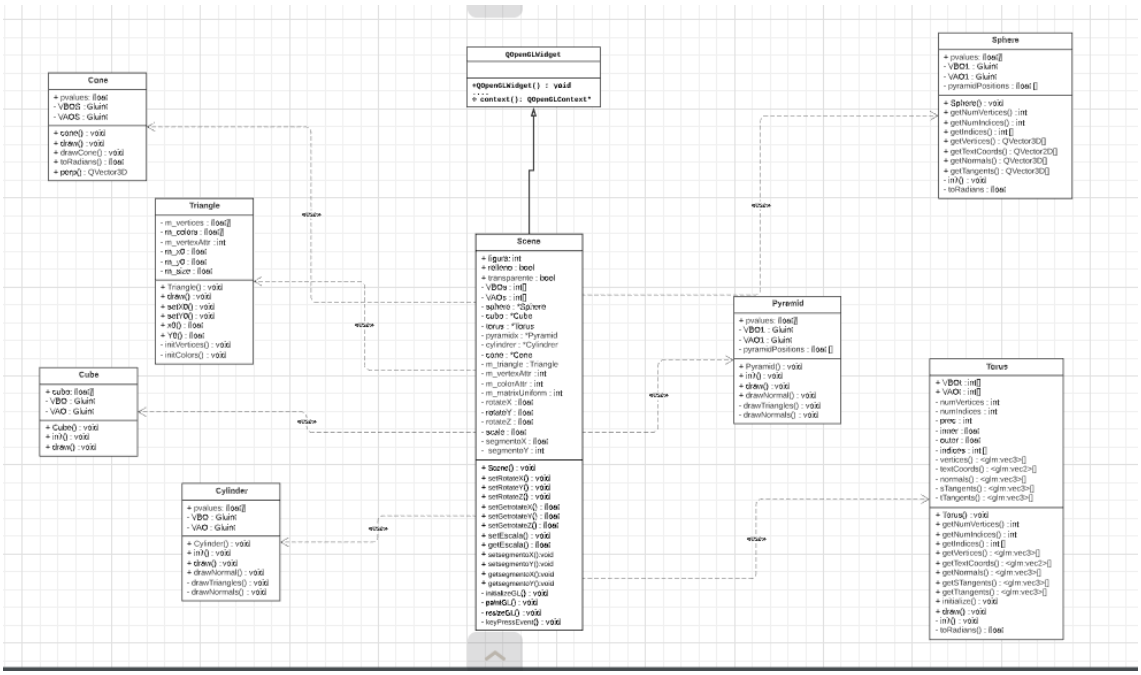
8. **Verificar la funcionalidad de las figuras**
   Para verificar el correcto funcionamiento de cada figura, se hará una prueba de cada figura de forma independiente dentro del IDE eclipse.

9. **Trasladar las figuras al entorno QT**
   Una vez la figura se halla trazado con éxito dentro de eclipse se procederá a montarla al entorno de QT respetando la estructura la mencionada.

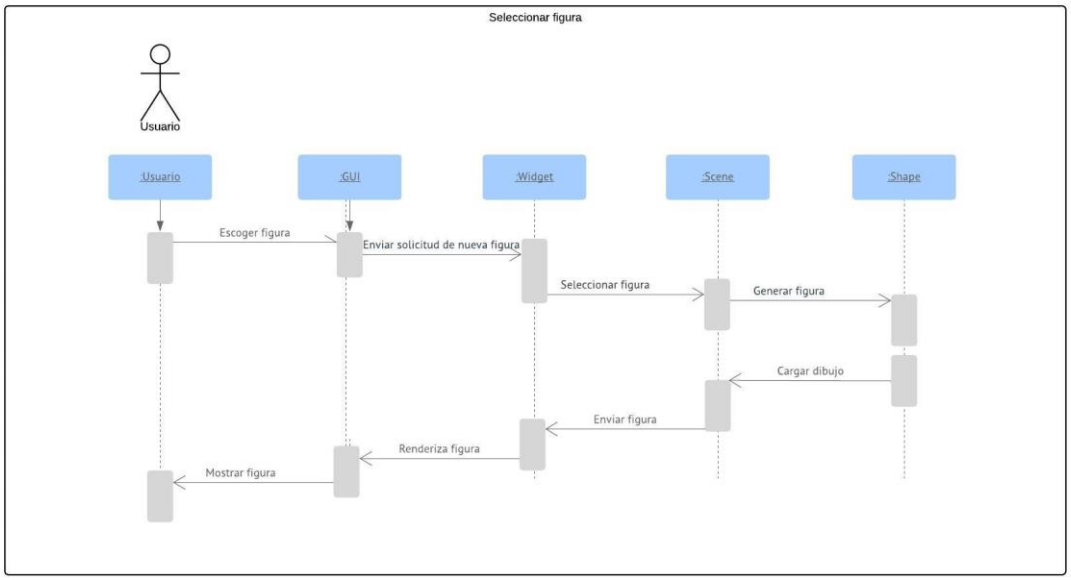
10. **Instanciar los objetos en la clase Scene**
    Con las figuras listas en sus clases, se procederá a instanciar cada una de estas dentro de la clase Scene para que puedan dibujarse, de acuerdo a lo que la clase Widget reciba de GUI (se podrá cambiar de imagen en tiempo de ejecución gracias a los switch que nos provee la GUI)

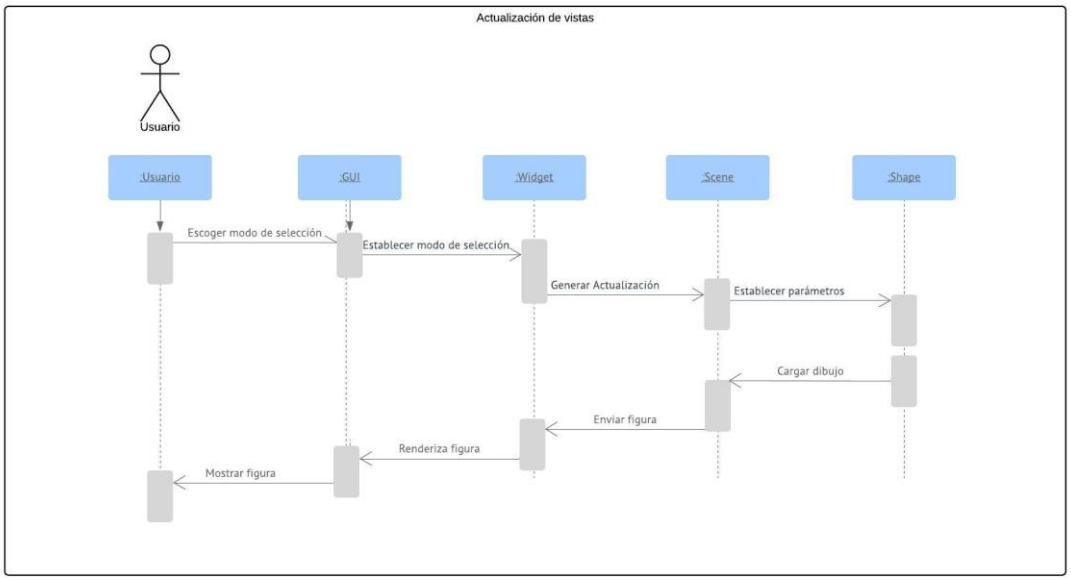
11. **Probar la aplicación**
    Hacer uso de todos los botones presentes de la GUI para corroborar y verificar que todo funcione de forma correcta.

3.1. Diagrama de Clases

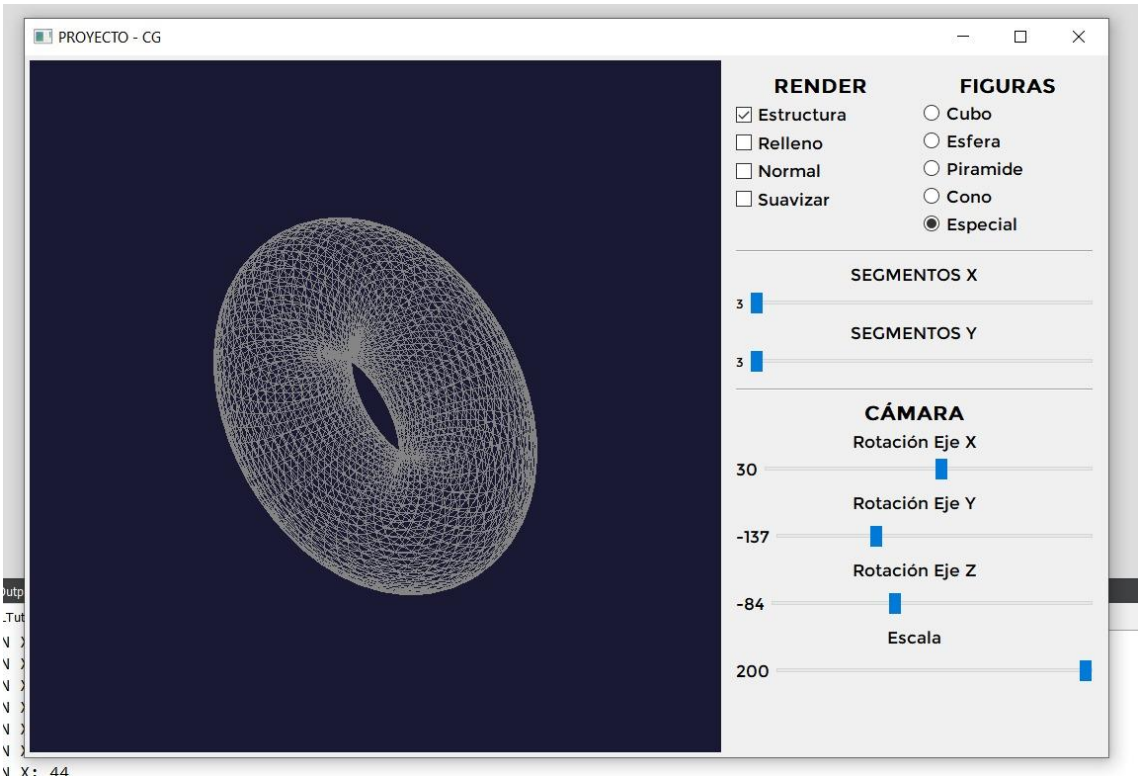


3.2. Diagrama de Secuencia





3.3. Mockups de la Aplicación



4. Aporte del Equipo

El equipo trato de seguir las indicaciones brindadas por el docente para el desarrollo del proyecto, promoviendo la comunicación y participación de todos los integrantes. Para la creación de las figuras pudo optarse por usar archivos mesh que nos brindaría los puntos de control y creación para cada figura.

Con el fin de brindar un agregado al proyecto el equipo deicidio optar por la creación de funciones matemáticas que permitan la creación de forma automática de cada figura, haciendo más eficiente la creación de los cuerpos geométricos, además de optar por modificar los segmentos que lo conforman en tiempo de ejecución.

5. Conclusiones

Contamos como conclusiones:

- Con el desarrollo de esta aplicación comprendimos que la importancia de la computación graficas va más allá del campo informático, se puede plasmar cualquier idea de forma tridimensional o bidimensional de acuerdo a lo que se requiera, esto orientado a diferentes áreas como: medicina, biología, química, arte, entretenimiento, ingeniera civil, aeronáuticas, historia, etc.
- OpenGL es un estándar para la creación de aplicaciones 2D y 3D, si bien ha evolucionado y mejorado con los años, esta no realiza todo el trabajo de forma independiente, se debe contar con un conjunto de herramientas adicionales para proveer de algunas funcionalidades extras (funciones matemáticas, Interfaz de Usuario, Manejo de ventanas, etc.)

- La computación grafica esta ligada definitivamente a un entorno matemático, pero esto no define de forma completa su usabilidad, en este sentido se necesita lograr una sinergia entre las habilidades lógicas y los conocimientos matemáticos para poder sacar el máximo provecho a las herramientas que esta nos provee.

6. Recomendaciones

Después de este desarrollo del proyecto, se recomienda a futuros trabajos:

- Al realizar este proyecto, tuvimos como primer obstáculo, la decisión de que framework o librería usaríamos para el desarrollo de la interfaz gráfica, entre las distintas opciones que contemplamos teníamos: QT y GTK, como equipo recomendamos usar QT, para el desarrollo de la GUI, por la fácil integración de esta y el OpenGL moderno.
- Estructurar y jerarquizar tanto la estructura de carpetas como el código mismo, esto con el fin de desarrollar una aplicación modular fácil de mantener y que pueda escalar en un futuro.
- Revisar la documentación oficial de las herramientas que se usaran para el desarrollo de la aplicación, con el fin de conocer su compatibilidad y sobre todo la sintaxis y los métodos que nos ayudaran a cumplir con los requisitos.
- Al ser este un proyecto de forma grupal, optar por el uso de una metodología ágil para llevar el proceso de desarrollo de una forma mas eficiente y con las medidas de contingencia necesarias para poder lograr el objetivo del mismo.

7. Referencias APA

Research, J. P. (2019). *El mercado de la computación gráfica crece y genera oportunidades de empleo.* US.