Universidad de las Ciencias Informáticas

Facultad 4

Módulo para la generación de imágenes mediante el método *Raytracing*

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor:

Daniel Alejandro Romero Geigel

Tutores:

Dr.C Augusto César Rodríguez Medina

Ing. Ernesto Alejandro Santana Hidalgo

Declaración de autoría

Declaro ser único autor del presente trabajo de diploma y autorizo a la Universidad de las Ciencias Informáticas a ser uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2016.

_____

Daniel Alejandro Romero Geigel

Firma del Autor

_____ _____

Dr.C Augusto Cesar Rodríguez Medina

Firma del Tutor

Ing. Ernesto Alejandro Santana Hidalgo

Firma del Tutor

"Los programas deben ser escritos para que los lean las personas, y sólo incidentalmente, para que lo ejecuten las máquinas"

Abelson and Sussman

Agradecimientos

A mi tutor Augusto Cesar Rodríguez Medina y a Gustavo García Gonzáles.

Dedicatoria

A mi abuela, a mis padres, a mi esposa, a mis familiares y amigos.

Resumen

El presente trabajo tiene como objetivo desarrollar un módulo para la creación de imágenes mediante el empleo del algoritmo de trazado de rayos que permita la visualización de los resultados obtenidos en los módulos realizados por el grupo de investigación Soluciones Informáticas para la Ingeniería y la Industria. El visor con el que cuenta la aplicación no permite la visualización de las texturas o colores aplicados sobre los diferentes objetos; esto dificulta el trabajo del diseñador y en ocasiones conlleva a la pérdida de tiempo en la etapa de diseño o incluso de materiales. La solución presentada fue desarrollada con *Qt, Open CASCADE* y *POVRay*, siguiendo la metodología AUP en su variante UCI. Las funcionalidades desarrolladas permiten aumentar la calidad de las presentaciones sobre resultados obtenidos con los módulos desarrollados y constituye la primera versión del generador de imágenes para los desarrollos del grupo de investigación Soluciones Informáticas para la Ingeniería y la Industria.

Palabras clave: diseño asistido por computadora, trazado de rayos, visualización

Introducción

Para la ingeniería en general, las tecnologías de Diseño Asistido por Computadoras, término que es más conocido por la denominación en inglés *Computer Aided Design* o "CAD" por sus siglas, (a lo largo de este trabajo se hará referencia a este termino con el acrónimo DAC); proporciona herramientas que automatizan los procesos de modelado, visualización y elaboración de la documentación técnica de los objetos de diseño, lo que permite incrementar la calidad, la eficiencia, reducir costos y acortar los tiempos de diseño y de producción. Los sistemas DAC se consideran tecnologías de avanzada con un peso importante en la automatización de los procesos productivos. La producción de estos en las últimas décadas se ha realizado por compañías que han estado compitiendo entre sí por el mercado con productos como *AutoCAD*, *Inventor* de *AutoDesk Incorporated*, *SolidWorks* y *CATIA* de *Dassault Systèmes* (12-13) y *SolidEdge* de *Siemens PLM Software*, entre otros; estas herramientas implementan algoritmos para el modelado, la visualización, la edición y el manejo de los datos con niveles de eficiencia que han ido creciendo con el tiempo.

Los programas informáticos mencionados anteriormente son comerciales y Cuba no puede acceder a ellos por las siguientes razones:

1. El bloqueo económico impuesto por el gobierno de los Estado Unidos de América contra Cuba contiene cláusulas que impiden el acceso a sistemas considerados de alta tecnología como es el caso de las herramientas DAC (14).

2. Son sistemas costosos y su adquisición, en caso de no existir el bloqueo, solamente estaría justificada si la comercialización de los productos resultantes asegura la recuperación de la inversión (15).

3. Se distribuyen, principalmente, bajo licencias de uso, lo que significa que el sistema informático no se adquiere como propiedad, sino que se autoriza su uso por un tiempo determinado.

4. A los resultados de diseño en ingeniería, como obras del intelecto humano, le son aplicables las regulaciones sobre la propiedad intelectual, por lo que si fueron obtenidas con herramientas sin licencia no es posible su comercialización (16).

Aunque todos los países no tienen las mismas limitaciones que Cuba, se puede apreciar que incluso algunos de fuerte desarrollo, cuentan con compañías que han implementado proyectos con tecnologías de código abierto, como alternativa para enfrentar las restricciones que impone la adquisición de los sistemas comerciales; algunos de estos son los proyectos *ELMER*, *Code-Aster*, *Code-Saturne* y *Salome-Meca*, entre otros (18-19). Lo expuesto anteriormente condujo a la idea de profundizar en el análisis de sistemas que, en *software* libre, tuviesen la mayor cantidad de nexos con las tecnologías DAC. Los estudios preliminares permitieron identificar que en esa situación se encontraban las aplicaciones *FreeCAD* y *Salome-Meca*, ambas tienen funcionalidades para el modelado y el diseño de piezas y objetos. Las herramientas mencionadas anteriormente son distribuidas bajo licencias LGPL[1] 1, lo que significa que se puede tener acceso a sus códigos, modificarlos y

[1] Licencia Pública General Reducida de GNU, o más conocida por su nombre en inglés GNU Lesser General Public License.

estudiarlos; también se evaluó que como elemento a favor está la disponibilidad de la tecnología *Open CASCADE Community Edition* como software libre.

Un aspecto importante con el que cuentan todas las aplicaciones DAC analizadas, tanto comerciales como de software libre, es la presencia de funcionalidades para la construcción de imágenes mediante el algoritmo de trazado de rayos. Mediante el empleo de este algoritmo se pueden aplicar diferentes propiedades a los objetos que se estén modelando, ejemplo de estas son la textura, el color, el relieve, la reflexión, entre otras, aportando la posibilidad de simular el aspecto de materiales reales utilizados en la industria; estas funcionalidades brindan a los diseñadores la oportunidad de visualizar el producto en un estado similar al que tendría una vez acabado, lo que permite hacer ajustes y cambios si no se está conforme con el resultado.

Tomando en cuenta la infraestructura y dominio alcanzado en el país en lo que se refiere al desarrollo de la informática, se evidencia que es viable la creación de soluciones propias para contribuir al progreso de las tecnologías DAC para la industria nacional, lo que se verifica en la práctica por resultados parciales de proyectos que se han obtenido; entre estos se encuentran la aplicación ASIXMEC, con funcionalidades para el modelado paramétrico y visualización de piezas en 3D (17), y los diferentes módulos obtenidos en trabajos previos realizados por el Grupo de Investigación “Soluciones Informáticas para la Ingeniería y la Industria” (SIPII). Entre los resultados obtenidos por parte del grupo SIPII se encuentran: “Componente para la modelación de engranajes cilíndricos de dientes rectos” (47), “Componente para acelerar el diseño de resortes helicoidales” (48), “Componente para el modelado de vigas para el Sistema CAD 2D” (49), entre otros.

Actualmente el proyecto del Grupo de Investigación SIPII se encuentra enfrascado en el desarrollo de varios módulos, como parte de una solución DAC, para brindarle una alternativa a la industria del país en esta rama; entre los módulos anteriormente desarrollados este grupo cuenta con un núcleo capaz de importar y exportar los modelos en varios de los formatos más comunes entre este tipo de aplicaciones (brep, step, iges), varios módulos para acelerar el diseño de diferentes piezas y un visor implementado bajo los estándares de la biblioteca COIN3D[2] para visualización de los modelos virtuales. Este visor no permite la visualización de las texturas o colores aplicados sobre los diferentes objetos; esto dificulta el trabajo del diseñador y en ocasiones conlleva a la pérdida de tiempo en la etapa de diseño o incluso de materiales, ya que tendría que construirse el objeto para poder visualizar el resultado. Por lo anteriormente mencionado se hace necesario el desarrollo de un módulo que brinde dichas funcionalidades, planteándose el siguiente **problema de investigación**: ¿Cómo integrar las funcionalidades para construir imágenes mediante el método de trazado de rayos a los módulos desarrollados?

Para dar solución al anterior problema se determinó, en el grupo de investigación, seguir a lo largo de todo el proceso de desarrollo los criterios metodológicos siguientes: transitar desde los problemas más simples a los más complejos y reutilizar concepciones y funcionalidades existentes en tecnologías de código abierto disponibles.

[2] Es una implementación en código abierto (biblioteca) de la interfaz de programación de aplicaciones (API) de Open Inventor.

El **objeto de estudio** se centra en los diferentes algoritmos de trazado de rayos y se define **como campo de acción** la creación de imágenes en aplicaciones DAC.

Para el desarrollo del trabajo se plantea como objetivo general: Crear imágenes mediante el empleo de los algoritmos de trazado de rayos compatibles con los módulos desarrollados en el grupo SIPII; teniendo como posibles resultados un módulo para la creación de imágenes y un informe técnico de la investigación contenido en el documento final del trabajo de diploma. Para dar cumplimiento al objetivo general planteado, se definieron los siguientes objetivos específicos:

- Establecer lo referente sobre los algoritmos de trazado de rayos y la generación de imágenes en aplicaciones DAC.
- Implementar las funcionalidades para la creación de imágenes mediante el método de trazado de rayos.
- Integrar las funcionalidades implementadas a los diferentes módulos desarrollados por el grupo SIPII.
- Realizar un análisis sobre las necesidades para incluir nuevas tecnologías de visualización en la aplicación.

Las tareas planteadas para darle cumplimiento a los objetivos específicos son las siguientes:

- Asimilación de los conceptos y tecnologías requeridos para el desarrollo del componente.
- Diseño de la investigación y defensa del perfil de tesis.
- Búsqueda y revisión bibliográfica sobre el objeto de estudio.
- Estudio de las funcionalidades de los módulos para la creación de imágenes existentes en aplicaciones comerciales para el diseño asistido por computadoras (sistemas DAC).
- Análisis de los códigos fuente de las aplicaciones *FreeCAD* y *Salome-Meca* con la finalidad de identificar y evaluar la reutilización de sus funcionalidades, así mismo, identificar las funcionalidades de la tecnología *Open CASCADE* que serán utilizadas en el proceso de desarrollo.
- Estudio de los diferentes motores de trazado de rayos existentes y la selección del más adecuado a las necesidades.
- Fundamentación de las necesidades y requerimientos del proyecto que avalan la conformación del componente.
- Proceso de síntesis (obtención de conclusiones, resultados, definición de las formas de hacer y conformación de la estructura del componente).
- Estudio de los aspectos de la metodología de desarrollo de software (AUP-UCI) que se aplicarán en la investigación.
- Obtención de requisitos a partir de los módulos de *raytracing* existentes en sistemas propietarios y de código abierto, así como de las consideraciones de los potenciales usuarios.
- Definición de la arquitectura del componente, lo que implica además definir los patrones de diseño con los que cumplirá.
- Estudio de los aspectos de la metodología de software empleada en el proyecto.
- Determinación de la estructura de clases del componente.
- Modelado del flujo de datos del componente.

- Diseño de la interfaz gráfica del componente.
- Implementación del componente destinado a la creación de imágenes mediante el método de *raytracing*.
- Pruebas al componente (de las diferentes funcionalidades y de integración).

En el capítulo 1 de este documento se aborda un estudio de sistemas DAC con funcionalidades similares a los módulos desarrollados por el grupo SIPII (aplicaciones homólogas), las funcionalidades presentes en cada uno de ellos encargadas de implementar algoritmos de *raytracing* y sus principales características; los diferentes motores de *render* existentes, comerciales y de *software* libre, y las diferentes tecnologías a emplear durante su desarrollo. En el segundo capítulo se muestra un mapa conceptual con los principales términos utilizados, los requisitos, la descripción del componente, la arquitectura y patrones de diseño empleados, entre otros; y en el tercero se muestran los resultados de las diferentes pruebas realizadas al componente desarrollado.

1 Fundamentación del trabajo

En este capítulo se abordan temas generales sobre los sistemas homólogos, aspectos técnicos sobre los diferentes motores de trazado de rayos y las características que presentan los más utilizados por este tipo de aplicaciones. Se realiza un estudio de las diferentes técnicas para la realización de imágenes mediante los algoritmos de trazado de rayos; además, se describen las diferentes tecnologías de desarrollo a utilizar durante la etapa de implementación, así como la metodología a emplear.

1.1 Propiedades de los objetos en los módulos de renderizado

En los módulos de trazado de rayos, los objetos presentan un grupo de características, las cuales definen el aspecto final del resultado (imágenes o aspecto del objeto, en el caso de los que ejecutan algoritmos en tiempo real). Entre las propiedades más utilizadas se pueden encontrar las descritas a continuación:

Texturas

En un sentido general, la palabra textura se refiere a las características de la superficie y la apariencia de un objeto dada por el tamaño, forma, densidad, disposición de los patrones visuales presentes en este (28); generalmente se describen como lisa o rugosa, suave o dura, mate o brillante, etc. En la Figura 1 se muestran diferentes ejemplos de texturas aplicadas sobre un cubo, de izquierda a derecha son: *Sandalwood, Dark_Wood* y *New_Bras.*

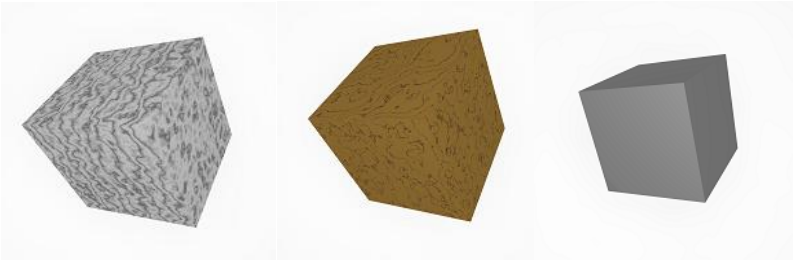

Figura 1: Diferentes texturas aplicadas a un cubo

Escala

La escala es un complemento que se utiliza junto a las texturas en el proceso de trazado de rayos para controlar la frecuencia con la que se repiten los diferentes patrones y el impacto que estos causen en el objeto final (34). En la Figura 2 se muestra un cubo con diferentes ajustes del parámetro escala.

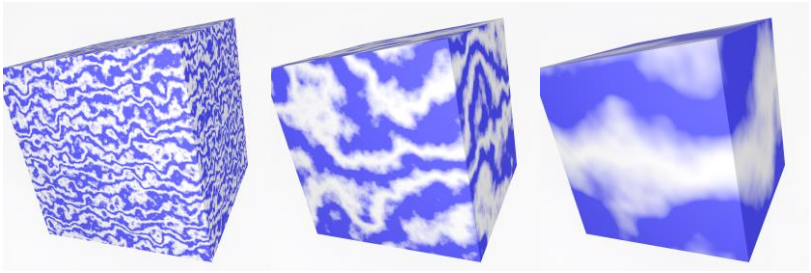

Figura 2: Diferentes valores de la escala (0, 5, 10)

Turbulencia

La turbulencia es un parámetro utilizado para simular defectos en las texturas, alterar los patrones de una forma aleatoria para provocar variaciones y lograr diversos resultados al utilizar diferentes valores de esta propiedad sobre la misma superficie (34).

Color

El color, aunque parezca el aspecto más básico y sencillo de usar en cualquier modelo obtenido por el algoritmo de trazado de rayos, tiene un elevado impacto en el aspecto final e influye en la apariencia del objeto; existen diferentes formas de representarlos, entre las más usadas están el RVA (rojo, verde y azul RVA por sus siglas) y el RVAA (rojo, verde, azul y canal alfa RVAA por sus siglas) (34). En la Figura 3 se muestra un ejemplo de los colores básicos (rojo, azul y verde).

Figura 3: Ejemplo de colores

“Bumps”

Propiedad que afecta a la textura y permite la simular protuberancias en los objetos virtuales (34). En la Figura 4 se muestra un cubo con diferentes ajustes de este parámetro.

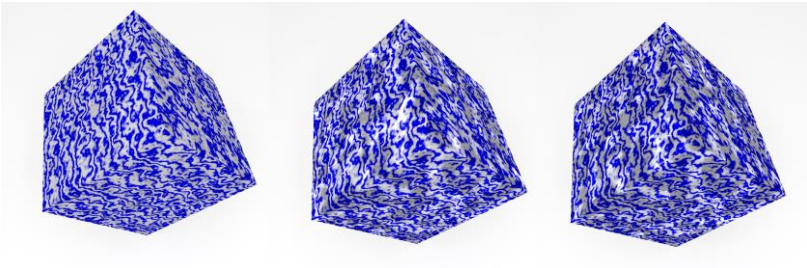

Figura 4: Diferentes valores del parámetro Bumps (0, 5, 10)

1.2 Modelos de iluminación

Una de las funcionalidades de una aplicación para la generación de imágenes, es la simulación de la luz; existen modelos avanzados que implementan algoritmos con una elevada precisión, estos se conocen como los algoritmos de *raytracing* o trazador de rayos. Los algoritmos de trazado de rayos se utilizan para generar una imagen trazando la ruta de la luz a través de los pixeles en una imagen, simulando diferentes efectos ópticos, como la reflexión, la refracción, la dispersión y la aberración cromática, en su encuentro con los diferentes objetos de la escena (29). Los modelos mencionados no son prácticos para aplicaciones de tiempo real y por eso se deben utilizar aproximaciones, que, aunque

carezcan de precisión, produzcan un resultado plausible; a continuación, se describen algunos de estos modelos.

1.2.1 Modelo de iluminación global

El modelo de iluminación global o iluminación indirecta es un nombre general para un grupo de algoritmos usados en gráficos por computadora tridimensionales que tienen como objetivo añadir realismo al modelado de la luz en escenas 3D. Dichos algoritmos tienen en cuenta, además de la luz proveniente directamente de una fuente de iluminación, los rayos provenientes de la misma fuente pero que han sido trasmitidos a través de reflexiones sobre superficies de la escena (reflexivas o no) (35).

1.2.2 Modelo de Iluminación Phong

Es uno de los modelos más simples, trabaja bajo el principio de que todos los objetos tienen un material con tres propiedades, la componente ambiental, la difusa y la reflectancia especular; se puede observar el efecto de cada una en la Figura 5. A cada componente se le asigna un valor de color, con colores más brillantes representando mayor reflectancia; cada fuente de luz también tiene estas propiedades y el resultado final de color es la suma de las interacciones de estas tres propiedades entre las luces y los materiales (29).

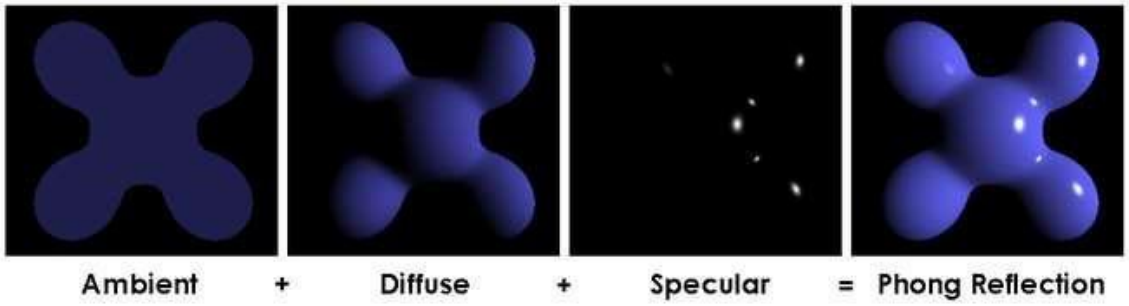

Figura 5: Iluminación de phong (30)

La luz ambiental no proviene de ninguna dirección en particular, tiene una fuente originada en algún punto, pero los rayos de luz han rebotado por el espacio y terminaron sin dirección. Los objetos sobre los que incide la componente ambiental están iluminados de forma uniforme; se puede denominar a este tipo de iluminación, un abrillantado global. La principal función de esta componente es aproximar la luz dispersa en el ambiente generada por una fuente de iluminación.

La luz difusa es la componente direccional de una fuente de iluminación. La componente especular también es direccional, pero interactúa más pronunciadamente con la superficie y en una dirección particular; causando un punto brillante en la superficie sobre la que brilla, denominado resaltado especular (*highlight* especular). Por ser altamente direccional, es posible que desde ciertos ángulos la componente especular no sea visible. El sol y un reflector son ejemplos de luces que producen fuertes *highlights* especulares, pero solo al brillar sobre un objeto que es brillante (29). El *highlight* especular se puede observar en forma de punto blanco en la esfera de la Figura 6.

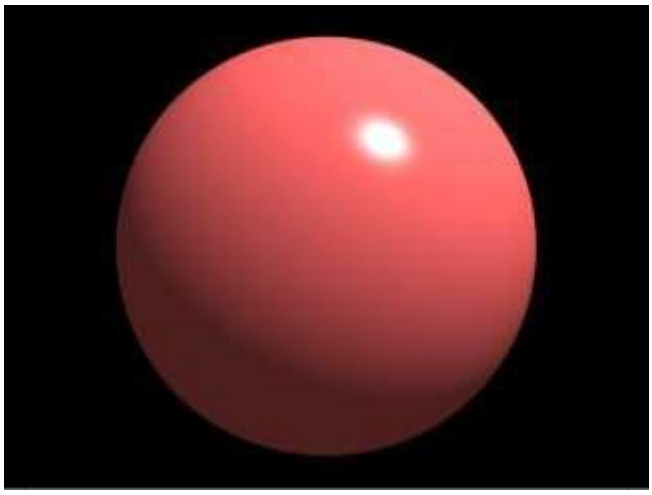

Figura 6: Modelo virtual renderizado con el modelo de iluminación de phong

1.2.3 Oclusión ambiental

La Oclusión ambiental es una técnica para simular una componente de iluminación global; esta componente es el efecto observado de la luz reflejada de objeto en objeto en una escena, de forma tal que las superficies son iluminadas indirectamente por la luz reflejada en las superficies vecinas. La componente ambiental es una aproximación a esta luz dispersa y es una cantidad pequeña y fija agregada a los cálculos de iluminación, sin embargo, en pliegues profundos o espacios entre objetos, menos emisión llegará debido a las superficies cercanas ocluyendo las fuentes de luminosidad, de aquí el término oclusión ambiental.

La componente ambiental podría ser considerada como la cantidad de luz que llega a un punto en una superficie, si esta estuviese rodeada de un número arbitrariamente grande de fuentes de iluminación. En una superficie perfectamente plana, cualquier punto es visible por cada una de las luces sobre la superficie, sin embargo, en una superficie con obstáculos, no todas las luces serán visibles desde todos los puntos. Los obstáculos en la superficie ocluyen la luz que llega a los puntos en los valles, y por lo tanto éstos recibirán menores cantidades de luminosidad, oscureciendo esa parte del modelo. En la Figura 7 se observa cómo se puede trazar una línea a ciertas fuentes de iluminación desde un punto, pero no a todas ya que las cuestas del modelo rayado obstaculizan el paso de la luz; por eso, tiene sentido que la zona del modelo en ese punto se vea más oscura que, por ejemplo, un punto en la cima de una cuesta, de donde se puede trazar una línea a todas las fuentes de iluminación (29).

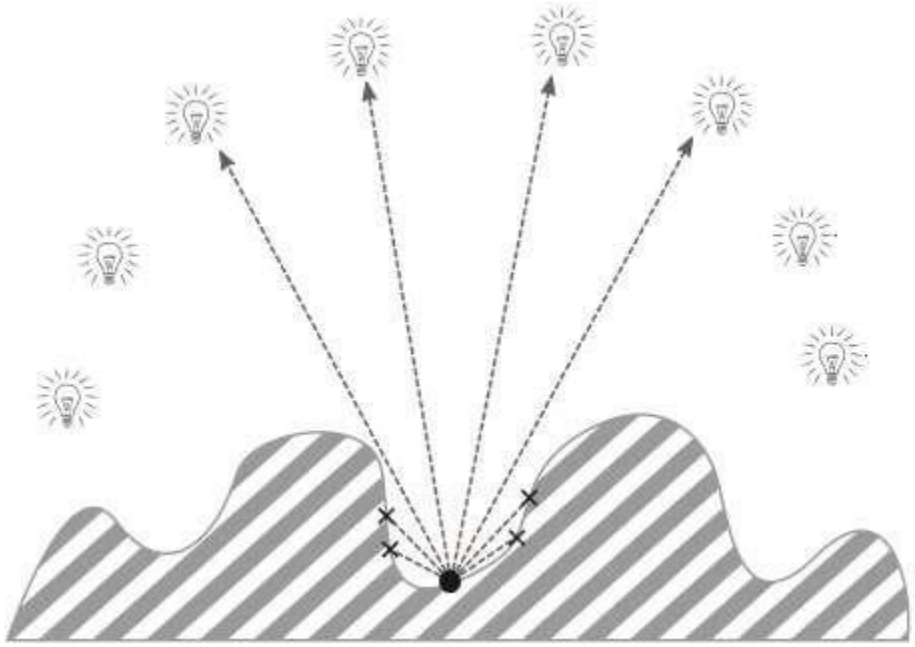

Figura 7: Alcance de la líneas de luz (29)

1.3 Aplicaciones homólogas

La mayoría de las herramientas DAC que existen en el mercado cuentan con funcionalidades para la implementación de algoritmos de trazado de rayos, a continuación, se presentan las herramientas más relevantes en esta área.

1.3.1 FreeCad

FreeCad cuenta con un módulo llamado “raytracing”, el cual permite la creación de una imagen del modelo virtual representado en la aplicación o algunos de los componentes de este, brinda la opción de escoger el color y transparencia del objeto de forma individual por cada parte existente en el proyecto, así como el motor para la generación de imágenes que se desea utilizar (POVRay o LuxRender). Como resultado final se obtiene una imagen de calidad media y sin muchos detalles, pues el sistema carece de opciones para el ajuste de texturas, acabado e iluminación (36).

1.3.2 AutoCad

Esta herramienta tiene el módulo “Render”, el cual permite generar una imagen de la escena y ajustar ciertos parámetros como el sombreado (NS, parcial, completo) y las propiedades de la luz (brillo, contraste y tonos medios); también cuenta con una biblioteca de materiales, la cual se encuentra organizada por categorías (*Ceramic, Concrete, Fabric, Metal,* entre otras) (37,50), en la Figura 8 se muestra un ejemplo de una imagen generada en esta aplicación.

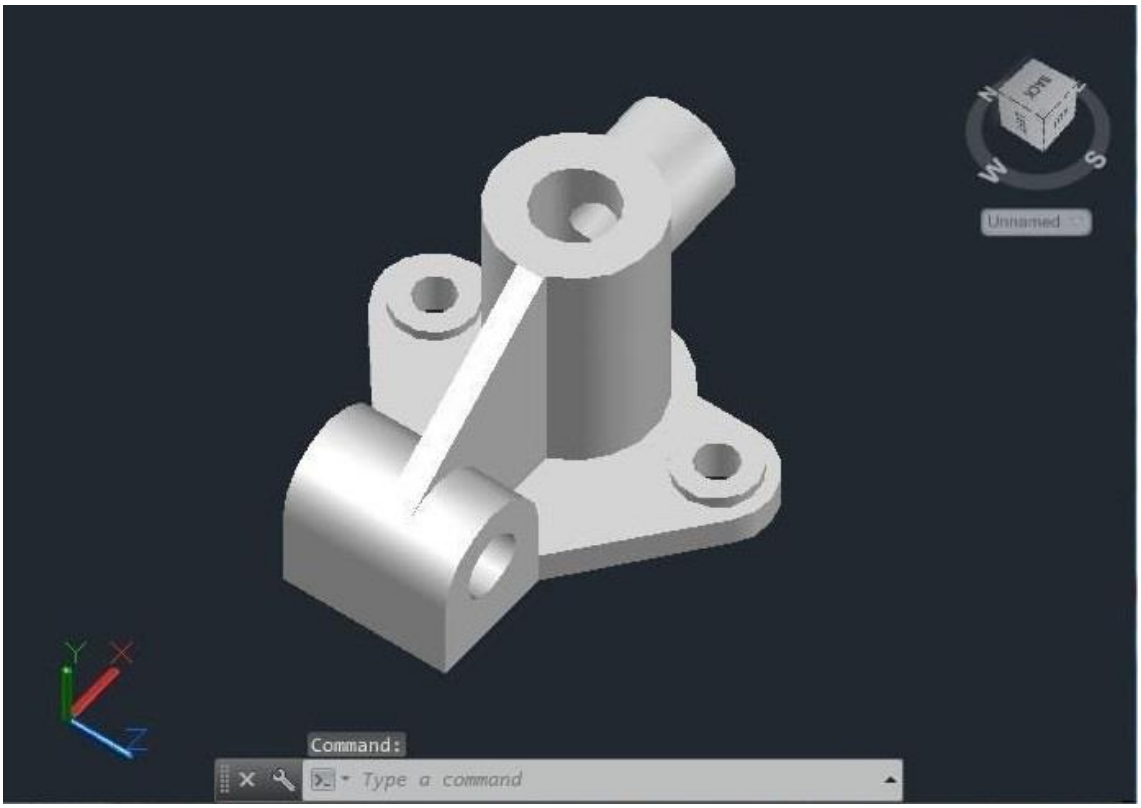

Figura 8: Pistón diseñado con AutoCad (31)

1.3.3 Autodesk Inventor

Autodesk Inventor cuenta con un módulo que permite el *raytracing* en tiempo real con diferentes opciones para el acabado de la pieza (textura, sombra y reflexión). Aunque este es poco intuitivo y engorroso a la hora de trabajar en él, tiene una calidad elevada; permite la selección entre diferentes tipos de acabados (*low*, *Draft*, *High*) y aunque se consigue una buena calidad de realismo el tiempo de procesamiento es elevado y se tiene que recalcular para cada movimiento de la vista o cambio realizado a las propiedades de la pieza, ralentizando el tiempo de respuesta de la aplicación y haciendo poco efectivo su uso durante toda la fase de desarrollo de un proyecto en esta herramienta (38).

1.3.4 Solid Works

Solid Works cuenta con dos niveles diferentes de funcionalidades para la creación del render. En el nivel básico se encuentran las opciones para la selección del color y la imagen o el mapeo de la textura permitiendo la selección del desplazamiento y la dirección de la textura. En el nivel avanzado se cuenta con las opciones que brinda el nivel básico además de las que permiten el acabado de la superficie y los ajustes de la iluminación. El acabado de la superficie permite escoger entre un grupo de valores predefinidos, mientras que la iluminación da la opción de ajustar de forma manual las variables iluminación ambiente, especular y difusa, la reflexión, la transparencia, la intensidad de la iluminación y el color; además, cuenta con una funcionalidad para guardar una imagen de la vista actual del proyecto (*screenshot*) (39).

1.3.5 Solid Edge

Esta herramienta cuenta con la funcionalidad ERA (*Explode-Render-Animate*) la cual abre una vista en la que se puede escoger entre una gran variedad de opciones para ajustar las cualidades de las texturas que se desean aplicar a los objetos en diseño, la amplia gama de parámetros que se encuentran disponibles se pueden agrupar en color, reflexión y transparencia. En las opciones del color se encuentra todo lo relacionado con las texturas y las diferentes configuraciones que se pueden realizar para modificarlas; las opciones para la reflexión permiten seleccionar la intensidad, el tipo de reflexión y otras propiedades para aumentar el realismo de la imagen final. El resultado obtenido es una imagen de muy buena calidad, pero esta se genera cada vez que se cambia algún parámetro o se mueve la figura, lo cual consume un elevado tiempo computacional (40).

1.4 Motores de *render*

Los motores de *render* son herramientas que interpretan diferentes descripciones de escenas escritas en archivos con extensiones xml, html, xsl, css, entre otros; calculan las reflexiones de los rayos de luz con los objetos 3D presentes en la escena, algunos de los más potentes y usados hoy en día se muestran a continuación.

1.4.1 LuxRender

LuxRender es un motor de renderizado imparcial y basado en la física, simula el flujo de luz de acuerdo con ecuaciones físicas, produciendo así imágenes realistas de calidad fotográfica; esto le permite capturar con precisión una amplia gama de fenómenos, lo cual significa que es totalmente compatible con la representación de rango dinámico alto (HDR). *LuxRender* presenta una variedad de tipos de materiales, además de los materiales genéricos, como mate y brillante, están presentes representaciones físicamente precisas de metal, vidrio y pintura para automóviles. Se encuentran disponibles propiedades complejas tales como absorción, refracción dispersiva y recubrimiento de película delgada (20). El programa mencionado es usado en diferentes aplicaciones entre las que se encuentran: Softimage, 3D Max, Cinema 4D y FreeCad.

1.4.2 Maxwell Render

Es un motor de *render* de los calificados "*unbiased"*, basado en algoritmos matemáticos que simulan el comportamiento de la luz real sobre los objetos, materiales reales y cámaras "*réflex"*; es un motor de *render* sencillo de utilizar ya que no hace falta ajustar parámetros para obtener una imagen de calidad fotográfica, y es predecible ya que todas las soluciones son siempre calculadas de la misma forma. Maxwell Render funciona como una aplicación autónoma y dispone de "*plugins"* para las plataformas de modelado 3D más importantes del mercado, entre las que se encuentran: Solid Works, Softimage, Archicad, 3D Max, Cinema 4D y Revit.

Entre sus características más notables se encuentran:

- Multilight
  La característica Multilight elimina la necesidad de generar una imagen cuando se requieren cambios de luz; permite cambiar las intensidades de las luces y emisores de la escena durante y después del proceso de representación, así que se puede ajustar la iluminación una y otra vez, almacenando tantas imágenes de la misma escena como quiera, y todo desde un único *render*.

- Materiales fáciles de usar
  El sistema de materiales de Maxwell es intuitivo y fácil de usar, consta de componentes que se pueden apilar igual que las capas en un programa de edición de imágenes; se basan en las propiedades ópticas reales, creando imágenes con un grado de realismo elevado y dispone de más de 3.500 materiales para usar de forma gratuita mediante descarga *online*.

- Trazado de rayos avanzado
  La tecnología de trazado de rayos avanzada, que forma la base del motor de renderización Maxwell, es capaz de simular la luz exactamente como está en el mundo real, sin trucos, que es la referencia en calidad de render para Maxwell.

- Renderizado en red multiplataforma
  Maxwell ofrece la posibilidad de generar imágenes en red multiplataforma, lo que permite una mezcla de Windows, OSX y Linux en la misma red. Puede establecer Maxwell Render para generar una imagen con todos los ordenadores, o configurar cada ordenador para procesar un fotograma de la animación.

Para poder usar las prestaciones de este motor de render se debe pagar una suma de 595 USD, en el momento del estudio, por una licencia de un año (21).

1.4.3 Mental Ray

Mental Ray, el software de generación de imágenes de *NVIDIA*[3]*,* produce resultados de alta calidad basados en técnicas avanzadas de trazado de rayos., combina una simulación de la luz basada en la física; este es utilizado en diferentes aplicaciones como *3D Max, Autocad, Cinema 4D y Revit*. Los profesionales del diseño gráfico llevan más de 25 años utilizando *Mental Ray*, que se ha convertido en el estándar de realismo digital en las industrias del diseño, el cine y los efectos especiales.

Principales características:

- Lenguaje de definición de materiales de NVIDIA
- Renderizado de materiales basados en las propiedades de la física con el formato MDL[4] de NVIDIA
- Introducción de diseños de materiales MDL de aspecto real ya empaquetados
- Resultados de alta calidad mediante el uso de efectos de *Mental Ray* (ej. desenfoque por movimiento)
- Admite todas las propiedades de los materiales de MDL, incluidas las texturas por procedimientos (CPU[5], y GPU[6])
- Posibilidad de compartir los materiales MDL con cualquier otra aplicación compatible con este lenguaje

[3] es una empresa multinacional especializada en el desarrollo de unidades de procesamiento gráfico y tecnologías de circuitos integrados
[4] formato para almacenar objetos en texto plano, donde cada objeto contiene un conjunto de propiedades y valores que están delimitadas por los saltos de línea
[5] unidad de procesamiento central
[6] unidad de procesamiento gráfico

- Limitaciones actuales: no hay medición de los materiales, restricciones en las conexiones de los parámetros de MDL
- Renderiza en la mitad de tiempo en las GPUs modernas con el nuevo algoritmo *Wavefront*
- Mapas de entorno HDRI (imagen de alto rango dinámico), captación automática de sombras proyectadas en el suelo
- Modelo de materiales en capas con arreglo a la física
- Difusión de la luz bajo la superficie, efectos atmosféricos y volumétricos
- Modelo de cámara física, efectos de profundidad de campo del objetivo y mapas de tonos

Precio de la licencia por un año, para 1 PC 295USD y 995USD para 5 PC (22).

1.4.4 POVRay

POVRay (*Persistence of Vision Raytracer*), en español ("Localizador de Rayos Persistencia de la Visión"), es un software gratuito de *raytracing*. Usa su propio lenguaje de descripción de escena, con características como macros, bucles y declaraciones condicionales. Es completamente gratuito aunque no fue lanzado bajo GPL. Es un programa de modelado tridimensional basado en algoritmos de radiosidad, una técnica que aumenta el realismo final mediante cálculos de luminosidad de la imagen.

El algoritmo de trazado de rayos es la técnica que utiliza *POVRay* para representar escenas imaginarias definidas en el ordenador mediante el uso de unos modelos, en los que se establecen los objetos y sus formas, así como puntos de luz y una cámara. La forma en que el ordenador intenta obtener una imagen lo más realista posible es siempre el resultado de obtener la mejor imagen con el menor coste de cálculo posible.

A diferencia de otros programas, *POVRay* no utiliza un entorno gráfico para la creación de modelos y escenarios, sino que interpreta instrucciones en un lenguaje estructurado similar a C a través del cual se describen los objetos, texturas, fuentes de luz y otros parámetros; las nuevas versiones incluyen un entorno de desarrollo muy práctico, con coloreado de sintaxis, menús para insertar pequeños trozos de código y visualización del renderizado en tiempo real.

Algunas de las características más destacadas:

- *Generación* parcial de una imagen, posibilidad de continuar la creación de una imagen interrumpida.
- Opciones para generar imágenes con distintas resoluciones, y con distintos grados de perfección a fin de reducir el tiempo de procesamiento.
- Posibilidad de visualización de imagen durante su creación.
- Posibilidad de un rápido vistazo en mosaico.
- Análisis sintáctico previo con buen tratamiento de errores.
- Abundante biblioteca de texturas, con posibilidades enormes para crear prácticamente cualquier clase de texturas.
- Efectos de superficie: rugosidad, brillos, reflejos, transparencia, etc.
- Figuras geométricas simples (esferas, conos, planos, cubos, prismas, etc.).
- Figuras geométricas basadas en complejas ecuaciones matemáticas.
- Figuras formadas por redes de triángulos.

- Suavizado de aristas, por ejemplo, en figuras formadas por triángulos.
- Figuras complejas formadas por agregación de otras.
- Delimitación de objetos complejos mediante figuras sencillas para optimizar el rendimiento del trazador.
- Pigmentación de una figura a partir de una imagen plana.
- Figuras generadas por sobrealzado proporcional al color de un *pixel* a partir de una imagen (muy útil para generar terrenos).
- Figuras generadas por revolución (modelado "en torno").
- Textos tridimensionales a partir de fuentes TIFF[7].
- Efectos luminosos con varios tipos de luces, focos, etc.
- Distintas opciones de Anti seudónimos (*Anti-Aliasing*). Estos efectos permiten que una línea diagonal parezca más lineal camuflando el conocido efecto escalera mediante un degradado suave. Existen distintos métodos disponibles para ello en *POVRay*.
- Completo lenguaje de descripción de escenas.
- Posibilidad de generar animaciones con distintos métodos.

Ventajas

- La calidad del trazador de rayos *POVRay* es elevada. Los productos comerciales de diseño basados en trazado de rayos suelen venir integrados con una serie de herramientas para diseñar cómodamente toda clase de objetos.
- Por añadidura las fuentes son públicas lo que permite investigar las técnicas utilizadas. El *copyright* de *POVRay* se trata al igual que *Linux* de una licencia de tipo *freeware*[8].
- El manual de *POVRay* está disponible en su versión de texto (ideal para búsquedas rápidas), en formato HTML.
- Permite hacer una incursión en el mundo de la infografía debido a sus increíbles capacidades técnicas. El dominio de la técnica permite la obtención de imágenes con un grado de realismo increíble (7).

1.4.5 V-Ray

V-Ray es un motor de renderizado usado como extensión para algunas aplicaciones de gráficos computacionales, como *3Ds Max, Maya, Modo, SketchUp, Nuke*, entre otros, es desarrollado por Chaos Software Production Studio; se distribuye con una licencia comercial por un año de 250 euros y una licencia comercial de tiempo indefinido por 500 euros.

V-Ray es una aplicación para la generación de imágenes que usa técnicas avanzadas, como por ejemplo algoritmos de Iluminación Global (GI) tales como *Path Tracing*, Mapeo de Fotones, Mapas de Irradiación y Fuerza Bruta, siendo esta última la opción primaria establecida en sus versiones recientes por su precisión y una mejor reducción de tiempos en la representación de la imagen (*render*); el uso de estas técnicas a menudo lo hacen preferible a los motores convencionales que son proporcionados por defecto por las aplicaciones 3D (41).

---

[7] es un formato de archivo informático para almacenar imágenes de mapa de bits

[8] está definido como un *software* que se distribuye en forma gratuita

1.4.6 YafaRay

YafaRay es un motor de render que utiliza un lenguaje de descripción de escenas en XML[9]; entre sus principales características se pueden apreciar las siguientes:

- Iluminación global (GI)

  Además del trazado de rayos de luz directa, *YafaRay* usa la iluminación global para producir representaciones realistas de escenas en 3D. Los algoritmos GI admitidos son Trazado de rutas, mapeo de fotones y trazado de rutas bidireccional.

- Iluminación de fondo

  Este sistema de iluminación se basa principalmente en la luz proveniente de un cielo que emite, teniendo en cuenta los cálculos de sombras suaves también involucrados.

- Efectos cáusticos

  YafaRay utiliza técnicas de mapeo de fotones para producir efectos cáusticos rápidos pero precisos; esta opción está disponible en varios métodos de iluminación además del mapeo de fotones, también se pueden calcular con rutas de trazado de ruta.

- Volúmenes

  Las características volumétricas de *YafaRay* proporcionan una simulación de la luz que interactúa con partículas suspendidas en una región del espacio. *YafaRay* usa un modelo realista basado en las propiedades físicas para representar la volumetría, y proporciona una base para crear no solo haces de luz creíbles, sino también humo, nubes, niebla y otros efectos volumétricos.

Este motor a pesar de ser distribuido bajo la licencia LGPL solo se puede integrar con *Blender* y *SketchUp* (42).

Luego del estudio realizado y atendiendo a las disponibilidad y prestaciones de las herramientas analizadas se decidió el empleo de *POVRay,* por tener funcionalidades para la representación de texturas y objetos compatibles con las tecnologías utilizadas para el desarrollo de los módulos realizados en el grupo SIPII y ser un programa gratuito.

1.5 Análisis de los requerimientos para incluir tecnologías de visualización

Las aplicaciones comerciales modernas implementan algoritmos para la generación de imágenes mediante el método de trazado de rayos, así como funcionalidades para realizar el render en tiempo real. Para alcanzar los estándares con que cuentan aplicaciones comerciales como AutoCad, Autodesk Inventor, SolidWorks, SolidEdge, entre otras de su tipo; se hace necesario contar con funcionalidades que permitan un render interactivo en tiempo real, con implementación en GPU con el objetivo de mejorar su rendimiento. Actualmente no existe ninguna herramienta de código abierto que contenga las funcionalidades mencionadas anteriormente, por lo que no serían útiles para el estudio.

[9] Lenguaje de Marcado Extensible

La tecnología utilizada actualmente para la visualización en el grupo de investigación SIPII está basada en los estándares de open inventor, el cual permite la modificación de texturas, colores y otras características de los objetos en tiempo real; pero el visor implementado no cuenta con estas funcionalidades, por lo que sería necesario agregar nuevas clases y métodos para incorporarlas. Algunas de las modificaciones necesarias son la creación de nuevas interfaces que permitan la selección de textura, color, transparencia, entre otras; así como la reutilización del código de la aplicación FreeCAD para implementar estos cambios en tiempo real. Existen otras tecnologías en *software* libre que permiten el *render* en tiempo real, entre estas se encuentran Open CASCADE, VTK, entre otras.

Una vez analizado la estructura y funcionalidades del *kernel* con que cuenta la aplicación que desarrolla el grupo de investigación SIPII se detectaron una serie de necesidades y modificaciones que deben realizarse para el render en tiempo real, las cuales se enuncian a continuación:

- Implementar el *ribbon* con el patrón *Singleton* para lograr la independencia de las clases de los módulos de la vista principal de la aplicación (*MainWindow*), lo que permite el trabajo en nuevas funcionalidades sin modificar clases externas al módulo
- Cambiar la arquitectura del proyecto a una arquitectura por *plugin* para poder incluir funcionalidades para la creación de imágenes con otros motores de render sin necesidad de recompilar la aplicación
- Crear un archivo .pro por cada módulo desarrollado para lograr independencia del proyecto principal

También se detectaron las siguientes modificaciones en el *kernel* para incluir nuevas tecnologías que permiten visualizar los objetos con un aspecto realista en tiempo real.

- Separar la selección del visor de Open Inventor y crear las interfaces que permitan la implementación de las clases necesarias para lograr la selección en la tecnología que se desee incorporar
- Implementar una nueva jerarquía de *ViewProviders* para lograr la integración de estos con todas las tecnologías incorporadas y permitir incorporar nuevos *ViewProvider* para nuevas tecnologías
- Crear una jerarquía para las tecnologías de visualización que se incorporen que permita la herencia y el trabajo a partir de clases abstractas en los niveles superiores
- Reimplementar el *MDIView* para que este pueda soportar todas las tecnologías de visualización incorporadas y no solo COIN3D como lo hace actualmente
- Agregar al *CADDocumentObject* los *Property* necesarios para gestionar los datos del render
- Implementar una biblioteca de texturas común para todas las tecnologías de visualización a fin de lograr uniformidad en el proceso y no tener que modificar las vistas para cada una de las tecnologías incorporadas
- Implementar los controladores de evento para las tecnologías que se incorporen y crear las interfaces para lograr que todos hereden de estas

Tomando en cuenta la complejidad de las modificaciones necesarias en el *kernel*, se decidió por parte del proyecto no incluirlas en la propuesta de solución de la presente investigación, sólo incorporar las funcionalidades para la creación de imágenes que brinde un mayor realismo al aspecto de los objetos modelados.

1.6 Tecnologías para el desarrollo

La evolución de las tecnologías informáticas ha llevado consigo un aumento exponencial de las herramientas y lenguajes que se emplean para el desarrollo de nuevos sistemas. Seguidamente se describen las características de las tecnologías empleadas en la presente investigación.

1.6.1 Lenguaje C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup, como extensión del lenguaje de programación C. Abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos por lo que se considera un lenguaje híbrido multiparadigma. El lenguaje C++ es usado por millones de programadores en cada dominio de aplicación, es un lenguaje de programación maduro y de gran velocidad de compilación y presenta las siguientes características:

- Lenguaje versátil, potente y general.
- Lenguaje rico en operadores y expresiones.
- Flexible, conciso y eficiente.
- Presenta programación modular.
- Introduce nuevas palabras claves y operadores para manejo de clases.

Este lenguaje no fue diseñado con la computación numérica en mente, sin embargo, muchos programas de ingeniería, numéricos y científicos son realizados en C++ (43); este puede coexistir con código escrito en otro lenguaje. Muchos sistemas, herramientas y bibliotecas están implementadas en C++, tales como Boost, Qt, Open CASCADE, Coind3D, OpenCV, entre otras.

1.6.2 Framework QT

Qt es un framework de desarrollo de aplicaciones multiplataforma para escritorio, sistemas embebidos y dispositivos móviles, compatible con Linux, OS X, Windows, VxWorks, QNX, Android, iOS, Blackberry, Sailfish OS y otros; contiene módulos para el desarrollo en áreas como redes, bases de datos, OpenGL, tecnologías web, sensores, protocolos de comunicación, procesamiento de XML y JSON, impresión, generación de PDF, entre otros (44).

Algunas características que posee el framework Qt son:

- Posee una biblioteca para la creación de interfaces gráficas.
- Se distribuye bajo una licencia libre GPL, además de la LGPL, que permite su utilización gratuita con fines comerciales.
- Compatibilidad multiplataforma con un solo código fuente.
- Fácil de internacionalizar.
- Arquitectura lista para *plugins*.

Se decidió escoger este marco de trabajo por especificaciones del proyecto y sus grandes prestaciones y funcionalidades.

1.6.3 Open CASCADE Technology

La Tecnología *Open CASCADE* (OCCT, por sus siglas en inglés), es una plataforma de desarrollo de *software* que proporciona servicios para superficies 3D y modelado de sólidos, el intercambio de datos DAC, y la visualización. La mayor parte de la funcionalidad OCCT se encuentra disponible en forma de bibliotecas de C++, puede ser aplicado en el desarrollo de *software* cuyo objetivo sea el modelado 3D (CAD), fabricación / medición (CAM[10]) o simulación numérica (CAE[11]). Es una tecnología de *software* libre; se puede distribuir o modificar bajo los términos de la Licencia Pública General de GNU (LGPL) versión 2.1, con excepción adicional.

Alternativamente, Open CASCADE puede ser utilizada bajo los términos de licencia comercial o acuerdo contractual, está diseñada para ser altamente portátil y es conocida por trabajar en una amplia gama de plataformas (UNIX, Linux, Windows, Mac OS X, Android). La versión (7.0.0. beta) de Open CASCADE está certificada oficialmente en las plataformas Windows (IA-32 y x86-64), Linux (x86-64), MAC OS X (x86-64) y Android (4.0.4 ARMv7). *Open CASCADE Community Edition* (OCE, en español Edición de la Comunidad de Open CASCADE) es una versión de OCCT en la que la comunidad del software libre aporta sus experiencias y recomendaciones de optimización a las versiones liberadas, mediante foros o la página oficial de desarrollo de esta tecnología (45).

Se decide el empleo de OCE por ser una tecnología compatible con C++ y QT y además tener funcionalidades para el trabajo con *OpenGL* de forma nativa y brindar funcionalidades para la descomposición de objetos en sus componentes primarios (puntos, vértices y aristas).

1.6.4 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas; está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar; UML incluye conceptos semánticos, notación, y principios generales, tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código, así como generadores de informes. La especificación de UML no define un proceso estándar, pero está pensado para ser útil en un proceso de desarrollo iterativo, pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos (24).

El Lenguaje Unificado de Modelado capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar una función que finalmente beneficia a un usuario externo. La estructura estática define los tipos de objetos importantes para un sistema y para su implementación, así como las relaciones entre los objetos. El

[10] *computer-aided manufacturing* o fabricación asistida por ordenador
[11] *computer-aided engineering* o ingeniería asistida por ordenador

comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos. El modelar un sistema desde varios puntos de vista, separados pero relacionados, permite entenderlo para diferentes propósitos (24).

El Lenguaje Unificado de Modelado contiene construcciones organizativas para agrupar los modelos en paquetes, lo que permite a los equipos de software dividir grandes sistemas en piezas de trabajo para entender y controlar las dependencias entre paquetes, y para gestionar las versiones de las unidades del modelo, en un entorno de desarrollo complejo; contiene construcciones para representar decisiones de implementación y para elementos de tiempo de ejecución en componentes.

1.6.5 Visual Paradigm

Visual Paradigm es una herramienta CASE[12] que brinda un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Esta herramienta emplea el lenguaje de modelado UML (46). Se caracteriza por:

- Disponibilidad en múltiples plataformas (Windows, Linux).
- Diseño centrado en casos de uso y enfocado al negocio que genera un *software*
- de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la
- comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, con diferentes especificaciones.
- Licencia: gratuita y comercial.
- Soporta aplicaciones Web.
- Varios idiomas.
- Ingeniería inversa Java, C++, Esquemas XML, XML, NET exe/dll, CORBA IDL.

1.6.6 GitLab

Un sistema de control de versiones es una combinación de tecnologías y prácticas para seguir y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación y en las páginas web. La razón por la cual el control de versiones es universal es porque ayuda virtualmente en todos los aspectos al dirigir un proyecto: comunicación entre los desarrolladores, manejo de los lanzamientos, administración de fallos, estabilidad entre el código y los esfuerzos de desarrollo experimental y atribución y autorización en los cambios de los desarrolladores. El sistema de control de versiones permite a una fuerza coordinadora central abarcar todas estas áreas. El control de versiones combina procedimientos y herramientas para gestionar diferentes versiones de objetos de configuración que se crean durante el proceso del software (8). GIT Un sistema de control de versiones distribuido diseñado por Linus Torvalds en el año 2005, surge como alternativa a *BitKeeper*, un control de versiones privativo que usaba en ese entonces para el *kernel*; es liberado bajo una licencia GNU GPL2 y su última versión estable fue publicada a inicios de abril de 2017. Por sus disímiles ventajas, GIT se ha convertido en uno de los más usados alrededor del mundo, puesto que no depende de

[12] *Computer-aided software engineering o ingeniería de software asistida por computadora*

acceso a la red o un repositorio central; además, está enfocado en la velocidad, uso práctico y manejo de proyectos grandes (9).

1.7 Metodologías para el desarrollo

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo (1).

Una definición estándar de metodología puede ser el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. Determina los pasos a seguir y cómo realizarlos para finalizar una tarea. Si esto se aplica a la ingeniería del *software*, se puede destacar que una metodología:

- Optimiza el proceso y el *software*.
- Proporciona métodos que guían en la planificación y en el desarrollo del *software*.
- Define qué hacer, cómo y cuándo durante todo el desarrollo y mantenimiento de un proyecto (1).

Una metodología define una estrategia global para enfrentarse con el proyecto. Entre los elementos que forman parte de una metodología se pueden destacar:

- Fases: tareas a realizar en cada fase.
- Productos: entradas y salidas de cada fase, documentos.
- Procedimientos y herramientas: apoyo a la realización de cada tarea.
- Criterios de evaluación del proceso y del producto: permiten determinar si se han logrado los objetivos (1).

1.7.1 AUP

El Proceso Unificado Ágil (AUP, por sus siglas en inglés) es un enfoque de modelado híbrido creado por Scott Ambler cuando combinó RUP con los métodos ágiles. Mediante la combinación de RUP con AM, Ambler creó un marco sólido de procesos que se puede aplicar a todo tipo de proyectos de *software*, grandes o pequeños (6).

Amber creó AUP bajo los siguientes principios:

- La mayoría de las personas no va a leer documentación detallada. Sin embargo, se necesitará orientación y formación de vez en cuando.
- La descripción del proyecto debe ser en unas pocas páginas.
- Se ajusta a los valores y principios descritos en la Alianza Ágil.
- El proyecto debe centrarse en ofrecer valor esencial en lugar de características innecesarias.
- Los desarrolladores deben estar libres de utilizar las herramientas más adecuadas para la tarea en cuestión, en lugar de cumplir con un decreto.
- AUP se adapta fácilmente a través de herramientas de edición de HTML comunes.

A partir de esta metodología en la Universidad de las Ciencias Informáticas se define una versión que responda al proceso de desarrollo llevado a cabo en la institución. En esta versión se definen como fases de desarrollo: Inicio, Ejecución y Cierre (Figura 20). Además, se proponen 11 roles en lugar de los 9 definidos por AUP (Figura 21).

Esta versión de AUP define cuatro escenarios en los que se puede ubicar el desarrollo de una aplicación de acuerdo a sus características, los cuales son:

- Escenario No 1: Proyectos que modelen el negocio con Casos de Uso del Negocio solo pueden modelar el sistema con Casos de Uso del Sistema (6).
- Escenario No 2: Proyectos que modelen el negocio con Modelo Conceptual solo pueden modelar el sistema con Casos de Uso del Sistema (6).
- Escenario No 3: Proyectos que modelen el negocio con Diagrama de Procesos del Negocio solo pueden modelar el sistema con Diagrama de Requisitos por Proceso (6).
- Escenario No 4: Proyectos que no modelen negocio solo pueden modelar el sistema con Historias de Usuario (6).

A partir de los escenarios antes expuestos, cada proyecto debe insertarse en uno de ellos en concordancia con las particularidades que caracterizan a cada uno de ellos:

- Escenario No 1: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema. Es necesario que se tenga claro por el proyecto que los casos de uso del negocio muestran como los procesos son llevados a cabo por personas y los activos de la organización (6).
- Escenario No 2: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda este escenario para proyectos donde el objetivo primario es la gestión y presentación de información (6).
- Escenario No 3: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad (6).
- Escenario No 4: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido donde el cliente estará siempre acompañando al equipo de desarrollo. Para elaborar HU el proyecto debe establecer conversaciones acerca de las necesidades de los clientes. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información (6).

Siguiendo la política de desarrollo de *software* de la institución, se define como metodología a emplear la AUP-UCI en el escenario número 4, debido a la necesidad de una metodología que responda con facilidad a los cambios continuos, por estar el cliente siempre acompañando el desarrollo y por estar bien definido el negocio.

1.8 Conclusiones del Capítulo

El estudio de las herramientas similares que brindan estas funcionalidades permitió identificar que la mayoría de las herramientas privadas cuentan con un amplio conjunto de funcionalidades para la creación de estas imágenes, no siendo así en las de software libre como FreeCAD. Atendiendo a los requisitos del proyecto y al análisis realizado se decidió emplear en la propuesta de solución las siguientes tecnologías: **C++11, QT 5.5.1, QTCreator 4.0.0, Open CASCADE Community Edition (OCE) 0.17** como lenguaje de

programación, *framework* para el desarrollo, IDE y bibliotecas para el modelado en 3D respectivamente; y la herramienta CASE **Visual Paradigm para UML 8.0** para el modelado de los artefactos de ingeniería de *software,* empleando la versión 2.1 de **UML**. Se identificó el empleo de **POVRay** como tecnología para la generación de imágenes, por las funcionalidades que este brinda, la compatibilidad con aplicaciones DAC y ser de *software* libre. Como metodología de desarrollo se decidió emplear, como una norma del proyecto para lograr la homogeneidad en los trabajos realizados, **AUP-UCI** en su escenario número 4.

2 Propuesta de Solución

En el presente capítulo se presenta la propuesta de solución, la cual presenta un mapa conceptual que explica los principales términos usados, los requisitos funcionales y no funcionales, las historias de usuario, la descripción del componente desarrollado; así como los diferentes patrones de diseño utilizados.

2.1 Mapa conceptual

Un mapa conceptual es un esquema de ideas que sirve de herramienta para organizar de manera gráfica y simplificada conceptos y enunciados a fin de reforzar un conocimiento. En un mapa conceptual se relacionan por medio de conectores gráficos conceptos e ideas para complementar una idea generalizada de lo que es un principal, el objetivo de un mapa conceptual es conseguir el significado por medio de enlaces que se analizan fácilmente (33).

Las principales características que estos presentan son:

- **Jerarquización**: Los conceptos se encuentran ordenados en forma jerárquica. De esta forma, la idea general se ubica en la parte superior del esquema y a partir de ella se desarrollen los demás conceptos (34).
- **Responden una pregunta de enfoque:** aunque los mapas conceptuales involucren contenidos generales y específicos, su elaboración y estudio debe permitir al individuo resolver una pregunta de enfoque a través de la cual se desarrollará el contenido del gráfico (34).
- **Simplicidad**: reflejan la información más importante de forma breve y concisa.
- **Uso de proposiciones**: se forman a partir de la unión varios conceptos mediante palabras de enlace y líneas conectoras (34).
- **Uso de enlaces cruzados**: se emplean para relacionar conceptos de diferentes partes del mapa conceptual y dar lugar a una nueva idea o conclusión (34).
- **Agradable a la vista:** cuando se elabora de forma armoniosa crea un impacto visual que facilita la comprensión del contenido planteado (34).

En la Figura 9 se muestra el mapa conceptual correspondiente a la solución desarrollada.

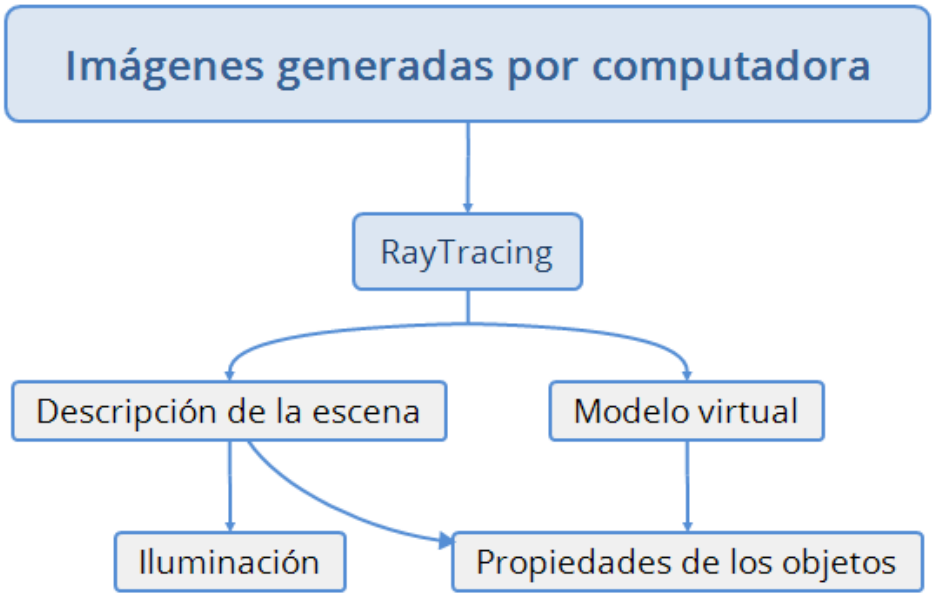

Figura 9: Mapa conceptual

Imágenes generadas por computadora: Son imágenes que se generan a partir de datos y cálculos matemáticos para la simulación de efectos de luz y otras propiedades presentes en la imagen.

Raytracing: uno de los métodos empleados para la obtención de imágenes a partir de modelos virtuales.

Modelo virtual: una representación en tres dimensiones de un objeto o un conjunto de estos y las características que serán representadas en la imagen a generar.

Descripción de la escena: texto en determinado formato que es interpretado por una herramienta para la generación de imágenes y que contiene la descripción de los objetos presentes en la escena y otras características de la escena.

Propiedades de los objetos: conjuntos de propiedades que poseen los objetos representados en la escena (textura, color, reflexión, entre otras).

Iluminación: ajustes de la iluminación de la escena atendiendo al modelo de iluminación utilizado.

2.2 Requisitos

"Los requisitos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información…" (10). Atendiendo a lo planteado anteriormente se identificaron los siguientes requisitos funcionales y no funcionales.

2.2.1 Requisitos Funcionales

RF 1. Generar una imagen del modelo representado en la aplicación.

RF 2. Seleccionar la textura deseada.

RF 3. Seleccionar una textura diferente para cada objeto.

RF 4. Seleccionar diferentes propiedades para modificar la textura seleccionada.

RF 5. Seleccionar el color de un objeto.

RF 6. Seleccionar las propiedades del terminado de la escena.

RF 7. Seleccionar la ubicación y el color de la fuente de luz.

RF 8. Adicionar otras fuentes de iluminación.

RF 9. Ajustar la calidad de la imagen a generar.

RF 10. Guardar la imagen obtenida en una ubicación escogida por el usuario.

2.2.2 Requisitos no Funcionales

Mantenibilidad

Modularidad: Tiene que permitir realizar cambios sobre el componente sin tener un gran impacto sobre el resto de la aplicación.

Portabilidad

Adaptabilidad: permitir que el sistema se ejecute en varios sistemas GNU-Linux y varias configuraciones de hardware.

2.3 Historias de Usuario

Las historias de usuario son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento se pueden modificar, reemplazar por otras más específicas o generales o añadirse nuevas. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (11). En la Tabla 1 se muestra un ejemplo de una historia de usuario, el resto puede ser consultadas en los Anexos.

Tabla 1: Historia de usuario "Generar una imagen del modelo representado en la aplicación

Número: 1	**Nombre del requisito:** Generar una imagen del modelo representado.
Programador: Daniel Romero Geigel	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado: 100h**
Riesgo en Desarrollo: Alto	**Tiempo Real: 120h**

Descripción:

1- Objetivo:

Permitir al usuario generar una imagen de los objetos modelados.

2- Acciones para lograr el objetivo (precondiciones y datos):

Debe existir un modelo en la aplicación.

3- Comportamientos válidos y no válidos (flujo central y alternos):

Si existe un modelo virtual se genera una imagen y se muestra en la aplicación.

4- Flujo de la acción a realizar:

El usuario da un clic en el botón “Render”

Flujo normal:

Se crea una imagen del modelo virtual activo en la aplicación.

Se muestra la imagen generada en la aplicación.

Observaciones:

Prototipo de interfaz:

2.4 Descripción del Componente

El módulo desarrollado sigue los criterios del estilo arquitectónico denominado llamada y retorno, específicamente el modelo por capas. Contiene tres interfaces desarrolladas en QT, la primera permite seleccionar los parámetros textura (Figura 10), escala (Figura 11), color (Figura 12), turbulencia (Figura 13), “Bumps” (Figura 14), y los relacionados con el acabado del objeto como luz ambiental, difusa, especular, el valor de la reflexión y el propio tipo de acabado (Figura 15). La segunda interfaz permite definir la intensidad de la fuente de luz y su color (Figura 16), la calidad de la imagen y sus dimensiones, así como la opción del *anti-alising*. La tercera permite seleccionar la ubicación y el nombre con el que se desea guardar la imagen resultante (Figura 17).

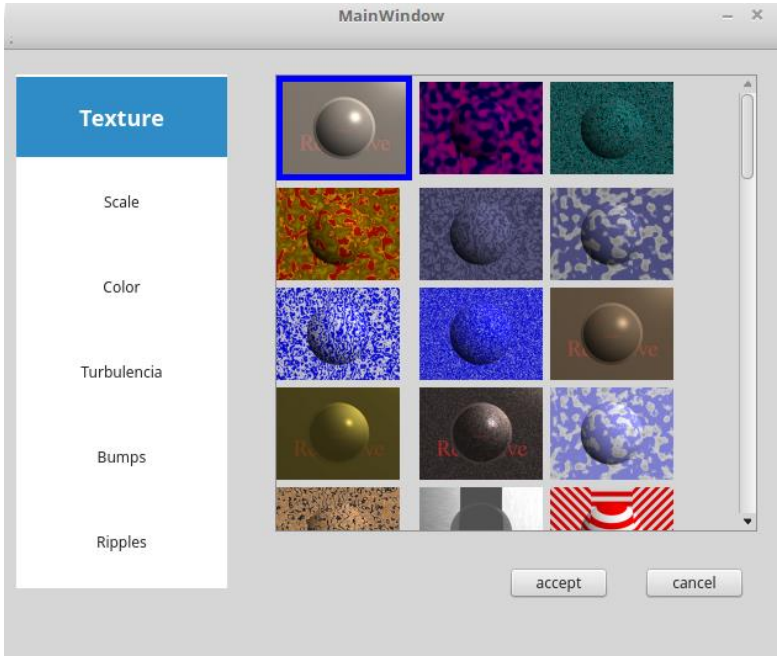

Figura 10: Vista para la selección de las propiedades del objeto

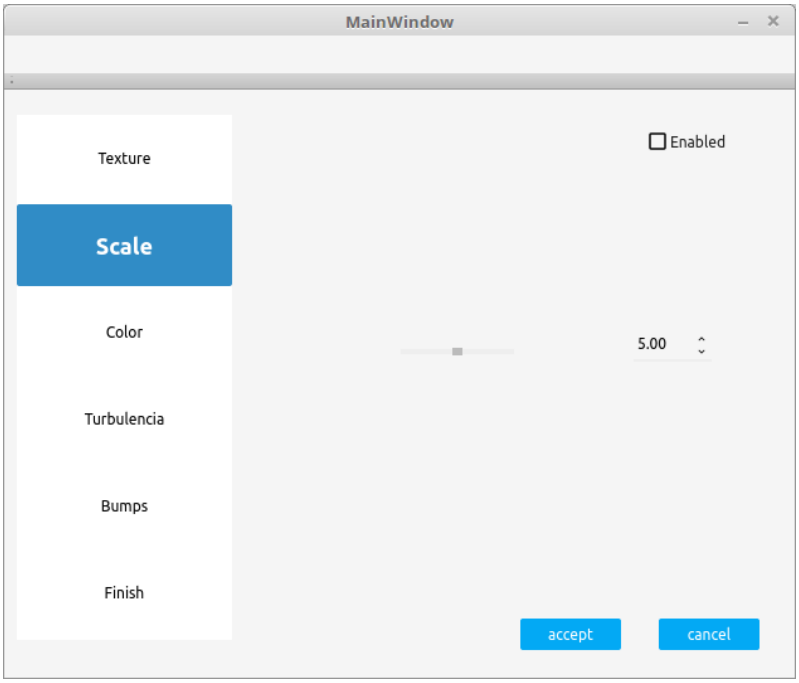

Figura 11: Vista para la selección de la escala

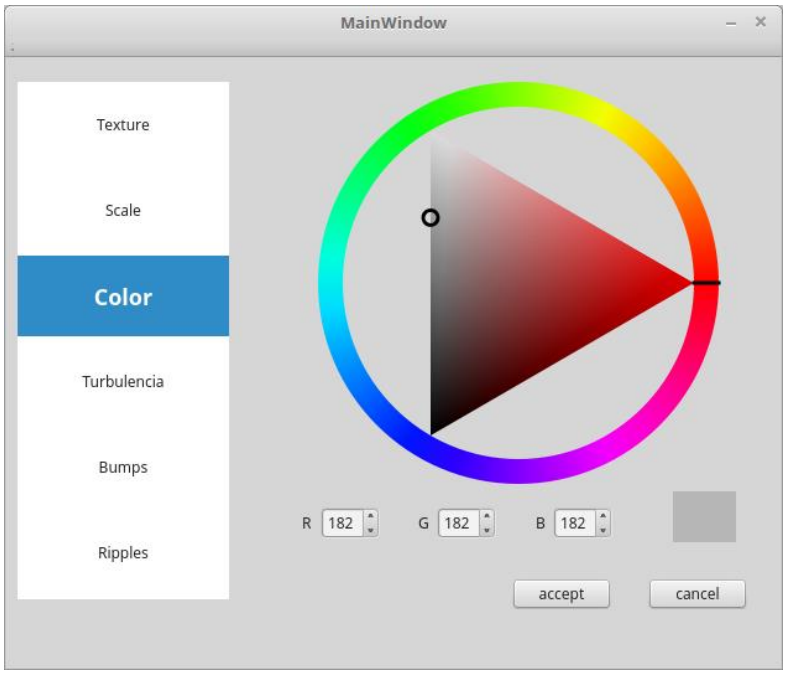

Figura 12: Vista para la selección del color

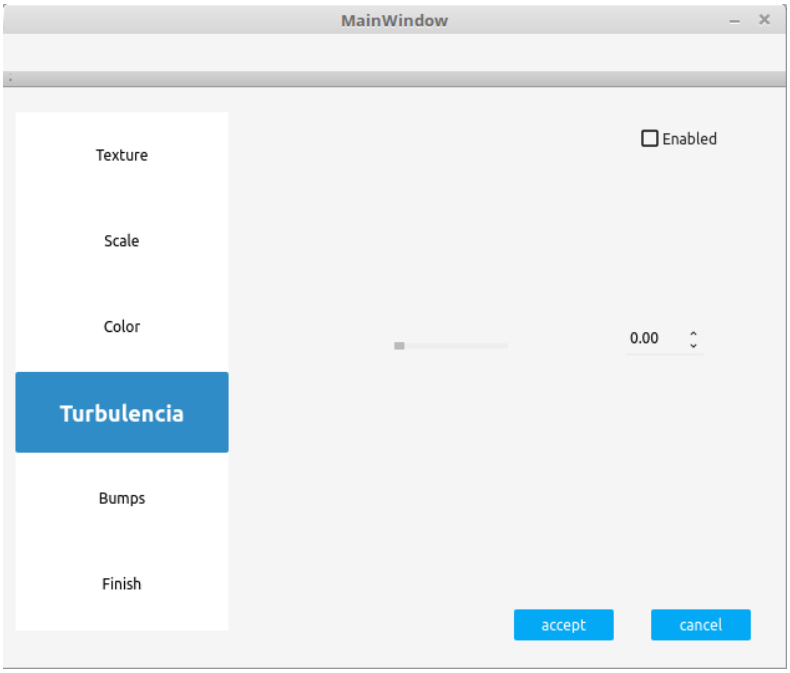

Figura 13: Vista para la selección de la turbulencia

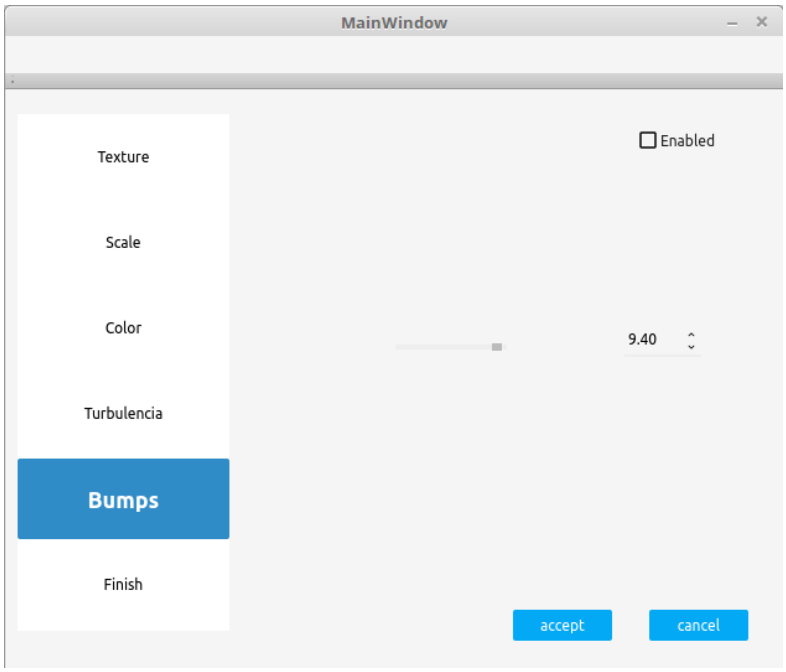

Figura 14: Vista para la selección del parámetro "Bumps"

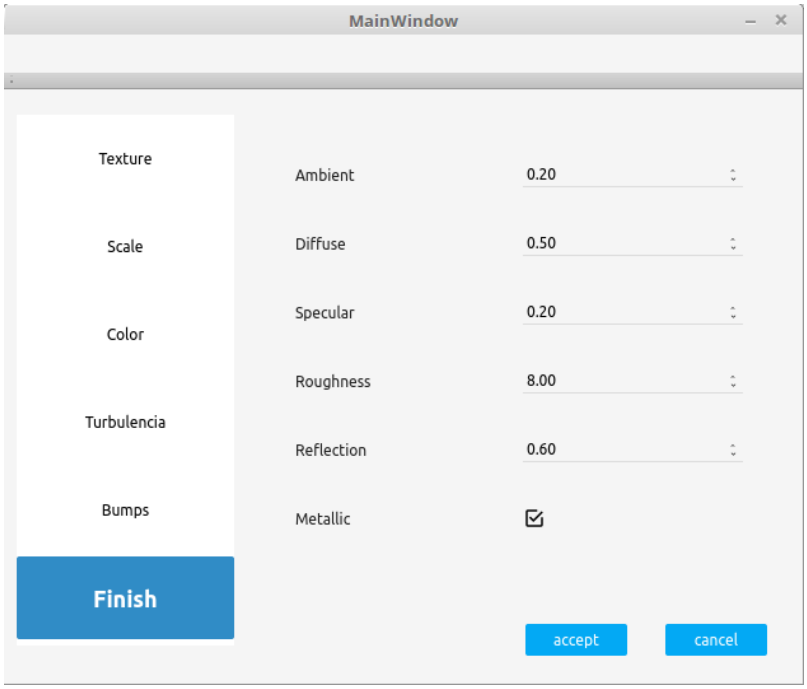

Figura 15: Vista para la selección de las opciones del acabado

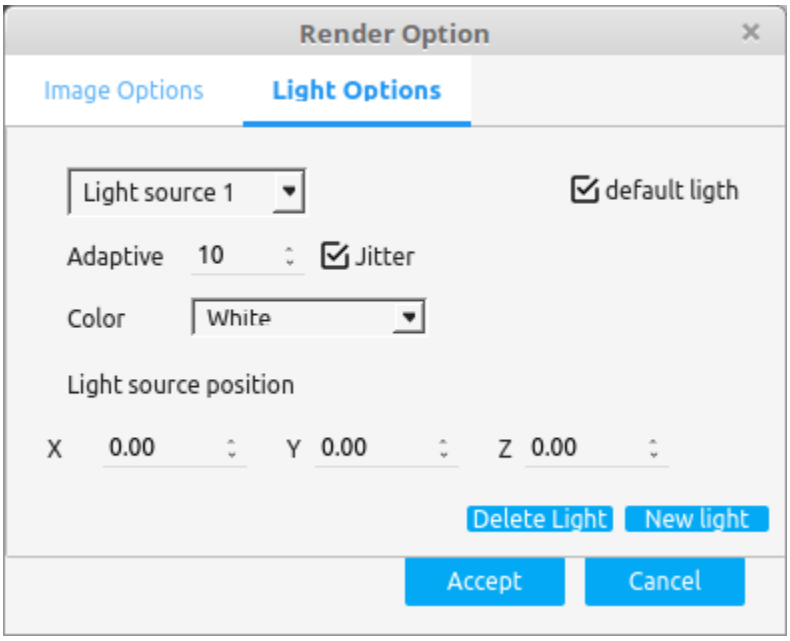

Figura 16: Vista para las opciones de las fuentes de luz

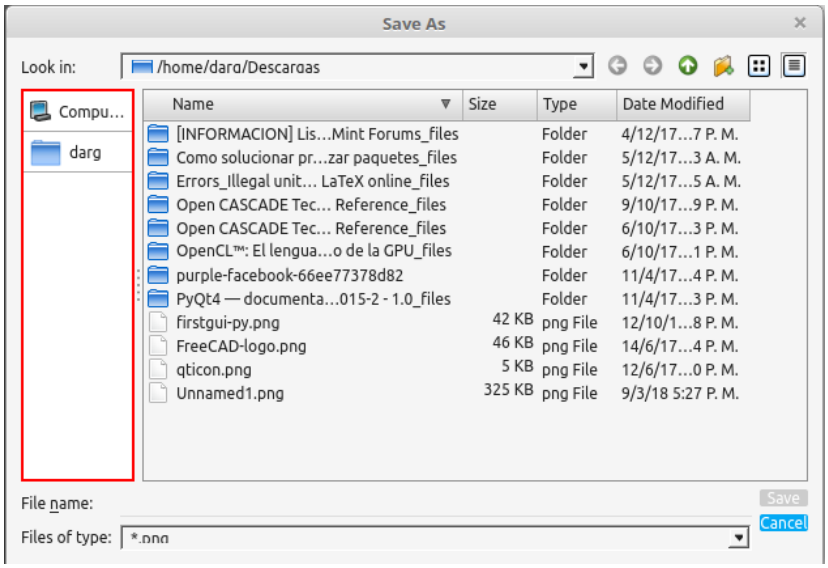

Figura 17: Vista para salvar las imágenes generadas.

Los datos recogidos por las diferentes vistas y la información del modelo virtual son recogidos por la clase *Render*, la cual con el uso de las funcionalidades que brinda la clase *POVTools*, hace la representación del modelo virtual y lo ejecuta mediante *POVRay* para obtener la imagen final, la cual se muestra en la aplicación y puede guardarse con un nombre y en una ubicación seleccionada por el usuario.

2.5 Diseño

"La esencia del diseño del *software* es la toma de decisiones sobre la organización lógica del *software*. Algunas veces, se representa esta organización lógica como un modelo en un lenguaje definido de modelado tal como UML y otras veces simplemente se utilizan

notaciones informales y esbozos para representar el diseño" (10). El proceso de diseño tiene asociado la decisión del tipo arquitectura y los patrones de diseño que empleará el sistema, así como la confección de distintos diagramas que favorezcan el trabajo en la fase de implementación.

2.5.1 Estilo y patrón arquitectónico del software.

"Un estilo arquitectónico es una transformación impuesta al diseño de todo un sistema. El objetivo es establecer una estructura para todos los componentes del sistema. En caso de que una arquitectura existente se vaya a someter a reingeniería, la imposición de un estilo arquitectónico desembocará en cambios fundamentales en la estructura del *software*, incluida una reasignación de la funcionalidad de los componentes" (7).

Se escoge como estilo arquitectónico el de Llamada y Retorno, pues "permite que un diseñador de *software* obtenga una estructura de programa que resulta relativamente fácil de modificar" (7).

Este estilo posee como patrones arquitectónicos a las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

"Un patrón arquitectónico define un enfoque específico para el manejo de alguna característica de comportamiento del sistema" (7).

"Los patrones se usan junto con un estilo arquitectónico para determinar la forma de la estructura general de un sistema" (7).

2.5.2 Arquitectura en Capas

El modelo en capas organiza el sistema en capas, cada una de las cuales proporciona un conjunto de servicios a la capa superior. Este modelo soporta el desarrollo incremental del sistema. También, soporta bien los cambios y es portable. Además, cuando las interfaces de las capas cambian o se añaden nuevas funcionalidades a una capa, solamente se ven afectadas las capas adyacentes. La principal desventaja de este modelo es que la estructuración de los sistemas puede resultar difícil y el rendimiento se puede ver afectado, pues se puede incurrir en que una funcionalidad debe pasar por muchas capas para alcanzar la capa superior que solicitó su servicio. (37)

Se decide el empleo de este modelo en la confección del componente porque soporta bien los cambios y el desarrollo incremental.

2.5.3 Patrones de diseño orientados a objeto

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes, en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. Los empleados en el módulo desarrollado fueron:

- Experto: mediante su uso, se asignan responsabilidades a la clase que cuenta con la información necesaria. Se evidenciará en la clase "Render", pues esta poseerá

un objeto de la clase “PovTools” para poder acceder a las funcionalidades que esta brinda.

- Creador: permite crear objetos de una clase determinada. Se utilizará en algunas de las clases de interfaz para crear instancias de formularios y entidades.
- Alta cohesión: este patrón caracteriza a las clases que posean responsabilidades estrechamente relacionadas, es decir, que no realicen un trabajo enorme. Con el objetivo de que la clase “TextureView” no realizara un trabajo enorme y poder reutilizar código se utilizó la clase “ColorPicker” encargada de generar el widget para la selección del color.

La propuesta de solución contendrá los siguientes patrones GOF:

- Singleton: Permite asegurar que de una clase concreta existe una única instancia y proporciona un método único que la devuelve. Se utilizará en las clases “Render” y “RenderOption”.
- Observador: Construye una dependencia entre un sujeto y sus observadores de modo que cada modificación del sujeto sea notificada a los observadores para que puedan actualizar su estado. Presente en las clases “TextureView” y “ColorPicker”.

2.5.4 Diagrama de clases

Un diagrama o modelo de clases en UML es un tipo de diagrama de estructura estática que describe la estructura de un sistema, muestra las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos (herencia, agregación, asociación, entre otras) (23).

En la Figura 18 se muestra el diagrama de clases diseñado para la solución:

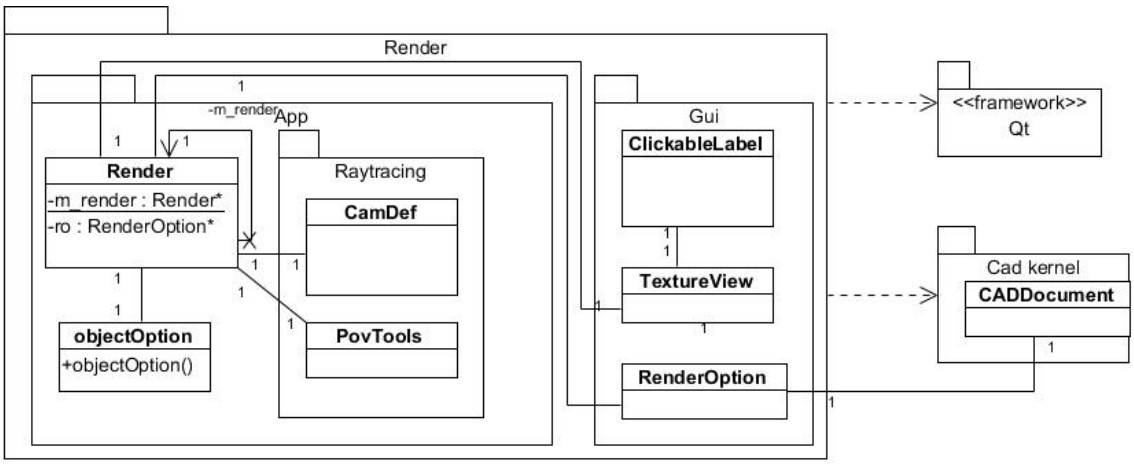

Figura 18: Diagrama de clases

Render: Esta es la clase encargada del manejo de los datos recibidos desde las vistas y de las funcionalidades que brindan el resto de las clases para la conformación de la imagen que es generada al final del proceso.

PovTools: Tiene funcionalidades que permiten la transformación de los objetos representados en *Open CASCADE* en sus topologías inferiores y su representación en el lenguaje interpretado por *POVRay*.

ObjectOption: Esta es la clase encargada de almacenar las propiedades de cada uno de los objetos presentes en la vista.

CamDef: Contiene la descripción de la cámara y los parámetros que esta contiene.

ClickableLabel: Clase que brinda funcionalidades para agregar el evento *onClick* a los *label* donde son ubicadas las imágenes que contienen las texturas en la vista.

RenderOption: Clase encargada de manejar la vista que contiene los parámetros de calidad y ajustes de la luz.

TextureView: Clase que controla la vista en la que se seleccionan las diferentes propiedades de los objetos contenidos en la escena.

2.6 Conclusiones parciales del capítulo

Se escogió la arquitectura por capa, pues esta permite el desarrollo incremental y asimila bien los cambios, y los patrones de diseños GRASP (Experto, Creador, Alta cohesión) y GOF (Singleton, Observador). Con la confección de las historias de usuario se lograron describir los requisitos funcionales que se deben desarrollar durante la presente investigación. La creación del diagrama de clases permitió definir las clases con sus funcionalidades y las relaciones entre ellas, así como la representación de cada uno de los patrones seleccionados.

3 Implementación y pruebas

En el presente capítulo se abordan cada uno los componentes que integran la etapa de implementación y prueba de la aplicación a desarrollar; se expone el estándar de codificación, el diagrama de componentes, los diseños de los casos de pruebas, así como los resultados obtenidos del proceso de verificación de la calidad.

3.1 Implementación

Para el desarrollo en la etapa de implementación se utilizan los artefactos generados en la fase anterior como son: diagramas de clases, historias de usuarios, patrones arquitectónicos y arquitectura, entre otros. En la implementación se define un estándar de codificación a utilizar para lograr mayor limpieza y unidad en el código, se implementan las historias de usuarios y otras actividades.

3.1.1 Estándar de codificación

Los estándares de codificación son pautas a cumplir por el código realizado en determinado sistema, con el objetivo de garantizar que todos los participantes lo puedan entender en menor tiempo y que el código en consecuencia se le pueda dar mantenimiento, o sea, que sea fácil de modificar para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. En la Tabla 2 se muestra el estándar utilizado en la presente investigación.

Tabla 2: Descripción del estándar de codificación

Descripción	Ejemplo
Definición de Objetos, Clases, funciones y atributos	

Todos los nombres de las clases implementadas comenzarán con letra mayúscula. En caso de poseer un nombre compuesto se escribirán de acuerdo a la normativa CamelCase-UpperCamelCase.	class Foo{
	cuerpo de la clase
	}
	class FooFirst{
	cuerpo de la clase
	}

Siempre se declara para todas las clases implementadas su respectivo destructor de clase.	virtual ~Foo()
La declaración de funciones o métodos siempre comenzarán en letra inicial minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<Tipo dato retorno> funcion() <Tipo dato retorno> funcionCompuesta() <Tipo dato retorno> funcionDobleCompuesta()

Los atributos siempre estarán escritos con letra minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<Tipo dato> atributo; <Tipo dato> atributoNombreCompuesto;
Definición de parámetros dentro de las funciones y constructores de clases	
Los nombres de los identificadores de los parámetros en las funciones deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.	<Tipo dato retorno> funcion(<tipo><id1>, <tipo><id2>, <tipo><idN>)
Los identificadores de los parámetros dentro de los constructores de las clases deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.	Clase(<tipo><id1>, <tipo><id2>, <tipo><idN>)
Definición de expresiones	

Para una mejor comprensión en la lectura y legibilidad del código los operadores binarios exceptuando los punteros, función de llamado a miembros, escritura de un arreglo y paréntesis de una función se escribirán con un espacio entre ellos.	x + y;
	x == y;
	idFuncion.miembro();
	idFuncion->miembro();
	array[];

Definición de estructuras de control y bucles	

Las estructuras de control y los bucles estarán definidos de igual manera en ambos casos siguiendo el estándar determinado por el *framework* de Qt.	Para las estructuras if, else, if else :
	<estructura control>(condición){
	tarea a ejecutar
	}
	Para los bucles while, for, do while y otros:
	<bucle>(condiciones){
	tarea a ejecutar

	}
Comentarios en el código según el estándar de C++	
Comentarios pequeños.	/* comentario sencillo */
Otros comentarios.	/* *Comentario */
Comentario de versión, descripción de clase y otras características de la clase o paquete.	/* ********** *Comentario amplio * ********** */

3.1.2 Diagrama de componentes

Un diagrama de componentes muestra dependencias entre los componentes, que no es más que una unidad física de implementación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema. Puede mostrar un sistema configurado, con la selección de componentes usados para construirlo o un conjunto de componentes disponibles (una biblioteca de componentes) con sus dependencias (24).

En la Figura 19 se muestra el diagrama de componentes de la solución, este cuenta con los paquetes App, encargado de contener los componentes que integran las clases encargadas del manejo de los datos y la lógica, Gui, es el que abarca los componentes relacionados con las interfaces de usuario, y el paquete Qt que contiene todas las librerías que nos brinda el *framework* Qt.

Entre los componentes se pueden encontrar “PreCompiled.h” y “ClickableLabel.h” los cuales brindan funcionalidades auxiliares para los demás componentes; el primero contiene una lista de las clases y paquetes que se deben incluir para el funcionamiento del módulo y el segundo aporta el evento *onclick* para los *label* que contienen las texturas, “OpenCascade”, que contienen las diferentes clases y funcionalidades que nos brinda la biblioteca *Open CASCADE Comunity Edition, “TextureView.h” y “RenderOption.h” son* los componentes encargados de las interfaces de usuario. El encargado del control de los datos es “Render.h” y para el manejo de las funcionalidades que interactúan directamente con los objetos se encuentra el componente “PovTools.h”.

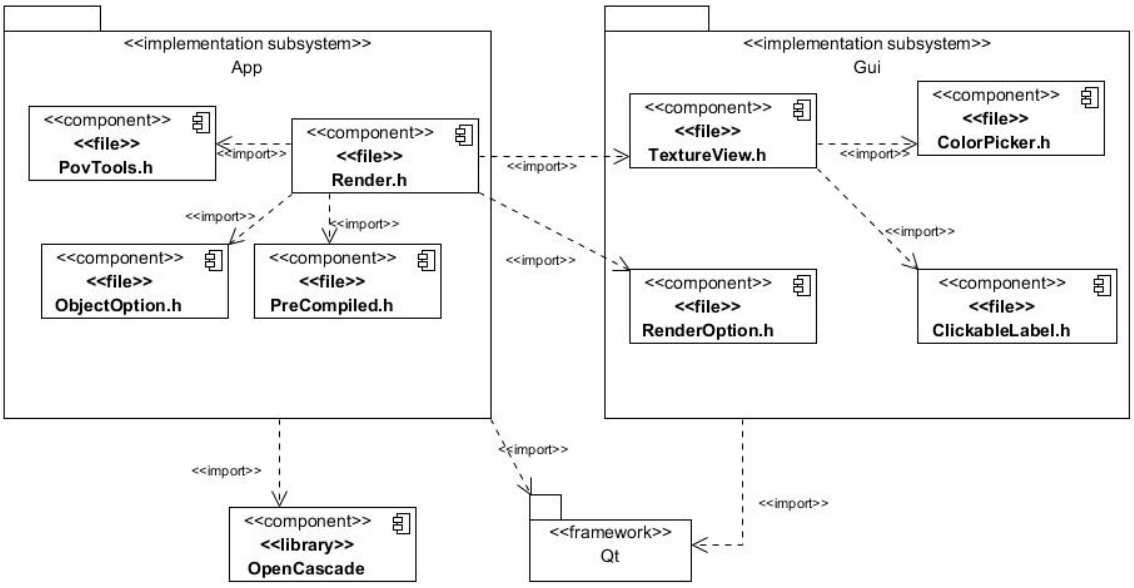

Figura 19: Diagrama de componentes

3.2 Pruebas

El proceso de pruebas de software tiene dos objetivos distintivos (10):

1 Demostrar al desarrollador y al cliente que el *software* satisface sus requisitos. Esto significa que debería haber al menos una prueba para cada requerimiento o característica que se incorporará a la entrega del producto.

2 Descubrir defectos en el *software* en el que el comportamiento de este es incorrecto, no deseable o no cumple su especificación. La prueba de defectos está relacionada con la eliminación de todos los tipos de comportamientos del sistema no deseables, tales como caídas del sistema, interacciones no permitidas con otros sistemas, cálculos incorrectos y corrupción de datos.

3.2.1 Niveles de prueba

Las diferentes pruebas que se le aplican a un *software* se dividen en varios niveles atendiendo a las diferentes etapas del proceso de desarrollo.

- Prueba de unidad

La prueba de unidad enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de *software*: el componente o módulo de *software*. Al usar la descripción del diseño de componente como guía, las rutas de control importantes se prueban para descubrir errores dentro de la frontera del módulo. La relativa complejidad de las pruebas y los errores que se descubren están limitados por el ámbito restringido que se establece para la prueba de unidad. Las pruebas de unidad se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un componente. Este tipo de pruebas puede realizarse en paralelo para múltiples componentes (25).

- Pruebas de integración

Estas son ejecutadas para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso. Permiten

descubrir errores en las especificaciones de las interfaces de los paquetes mediante la prueba de un paquete o un conjunto de estos. Los desarrolladores son los responsables de este tipo de pruebas en conjunto con los independientes (25).

- Pruebas de Sistema

Están constituidas por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas. Algunas de estas son: pruebas de recuperación, seguridad, resistencia, entre otras (25).

- Pruebas de Aceptación

Es la realización de una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Este proceso es realizado por los usuarios finales o por un desarrollador alfa y pueden durar días, incluso semanas (25-26).

Para la etapa de prueba de este componente se usarán las pruebas unitarias, de sistema y las de aceptación.

3.2.2 Métodos de prueba

- Pruebas de caja blanca

Es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que: 1) garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez, 2) revisen todas las decisiones lógicas en sus lados verdadero y falso, 3) ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas y 4) revisen estructuras de datos internas para garantizar su validez (25).

- Pruebas de caja negra

Se enfocan en los requerimientos funcionales del *software*; es decir, las técnicas de prueba de caja negra le permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa. Estas intentan encontrar errores en las categorías siguientes: 1) funciones incorrectas o faltantes, 2) errores de interfaz, 3) errores en las estructuras de datos o en el acceso a bases de datos externas, 4) errores de comportamiento o rendimiento y 5) errores de inicialización y terminación (25).

3.2.3 Diseño de Casos de Prueba

Los casos de pruebas utilizados en este trabajo fueron desarrollados en base a las historias de usuario y tienen como objetivo encontrar la mayor cantidad de errores posibles y probar todos los caminos en el *software*.

En la tabla 3 se muestra el caso de prueba para la historia de usuario Seleccionar la textura deseada, para un mayor estudio de estos ver el anexo B.

Tabla 3: Caso de Prueba de la Historia de Usuario "Selección de la textura deseada"

Seleccionar la textura deseada			
SC 2 Seleccionar una textura			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 2.1 Opción para seleccionar una textura para un objeto representado	En el "Entity tree" hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Texture y selecciona la textura deseada	Brinda la posibilidad de seleccionar una textura para el objeto seleccionado, además permite cancelar la operación en todo momento	Entity tree/Render Option/Texture
EC 2.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Entity tree/Render Option/Cancel

La realización de los casos de prueba permitió detectar un conjunto de no conformidades, las cuales se muestran en la tabla 4.

No. NC	Requisito Funcional	Descripción	Complejidad	Estado
1	1	No se generaba una imagen si no se seleccionaban las opciones generales para el render	Baja	Resuelta
2	2	No mostraba el *tooltips* con el nombre de la textura	Baja	Resuelto
3	5	Cuando se seleccionaba un color la textura no se mostraba	Media	Resuelto
4	7	No permitía la ubicación de las fuentes de iluminación en posiciones con valores negativos	Media	Resuelto

3.2.4 Pruebas Unitarias

Para la realización de estas pruebas se utilizó Qt Test, el cual es un *framework* para ejecutar este tipo de pruebas a aplicaciones y bibliotecas basadas en Qt. Este cuenta con todas las funcionalidades que normalmente se encuentran en estos tipos de *frameworks*, además de extensiones para ejecutar pruebas a interfaces gráficas de usuarios (27). En la imagen Figura 21 se muestran los resultados obtenidos al aplicar esta prueba. Las diferentes pruebas ejecutadas cumplen los objetivos de comprobar que las diferentes opciones seleccionadas en las interfaces de usuario sean las guardadas como parte de la descripción de la escena y utilizadas finalmente en la construcción de la imagen.

La prueba “testLightOptions” permite comprobar que los parámetros de iluminación y las diferentes fuentes de luz ubicadas en la escena estén en correspondencia con las especificaciones del usuario, para la verificación de que la cantidad de objetos presentes en el escenario y las opciones generales se utilizaron las pruebas “testCantidadShapes” y “testGeneralOptions” respectivamente; “testObjectOptions” y “testFinishOptions” se encargan de comprobar los parámetros que intervienen directamente con los objetos.

```
********* Start testing of TestUnitTest *********
Config: Using QtTest library 5.5.1, Qt 5.5.1 (x86_64-
PASS   : TestUnitTest::initTestCase()
PASS   : TestUnitTest::testLightOptions()
PASS   : TestUnitTest::testObjectOptions()
PASS   : TestUnitTest::testGeneralOptions()
PASS   : TestUnitTest::testFinishOptions()
PASS   : TestUnitTest::testCantidadShapes()
PASS   : TestUnitTest::cleanupTestCase()
Totals: 7 passed, 0 failed, 0 skipped, 0 blacklisted
********* Finished testing of TestUnitTest *********
```

Figura 20: Resultados de las pruebas unitarias realizadas con qttest

3.2.5 Pruebas de integración

Las pruebas de integración aplicadas cumplen con el objetivo de verificar que se ejecute de forma correcta la integración entre *POVRay* y las funcionalidades existentes en los módulos existentes. A continuación, se muestran algunas imágenes resultado de estas pruebas, en las que se probaron diferentes funcionalidades de la aplicación y se comparan los resultados obtenidos con otras aplicaciones (*FreeCAD*). En la Figura 22 se muestra una imagen de un engranaje cónico de dientes rectos, obtenido en el módulo “BevelGear”, en el visor COIN3D, la Figura 23 muestra la imagen generada por *FreeCAD* del modelo de la Figura 22 con un material de bronce; la Figura 25 muestra la imagen resultado de aplicar el proceso de renderizado al modelo de la Figura 22 en el módulo desarrollado, luego de exportar e importar el objeto, verificando así la integración de estas funcionalidades y la

Figura 24 muestra otra pieza generada con diferentes ajustes de textura, iluminación y con una luz de color rojo.

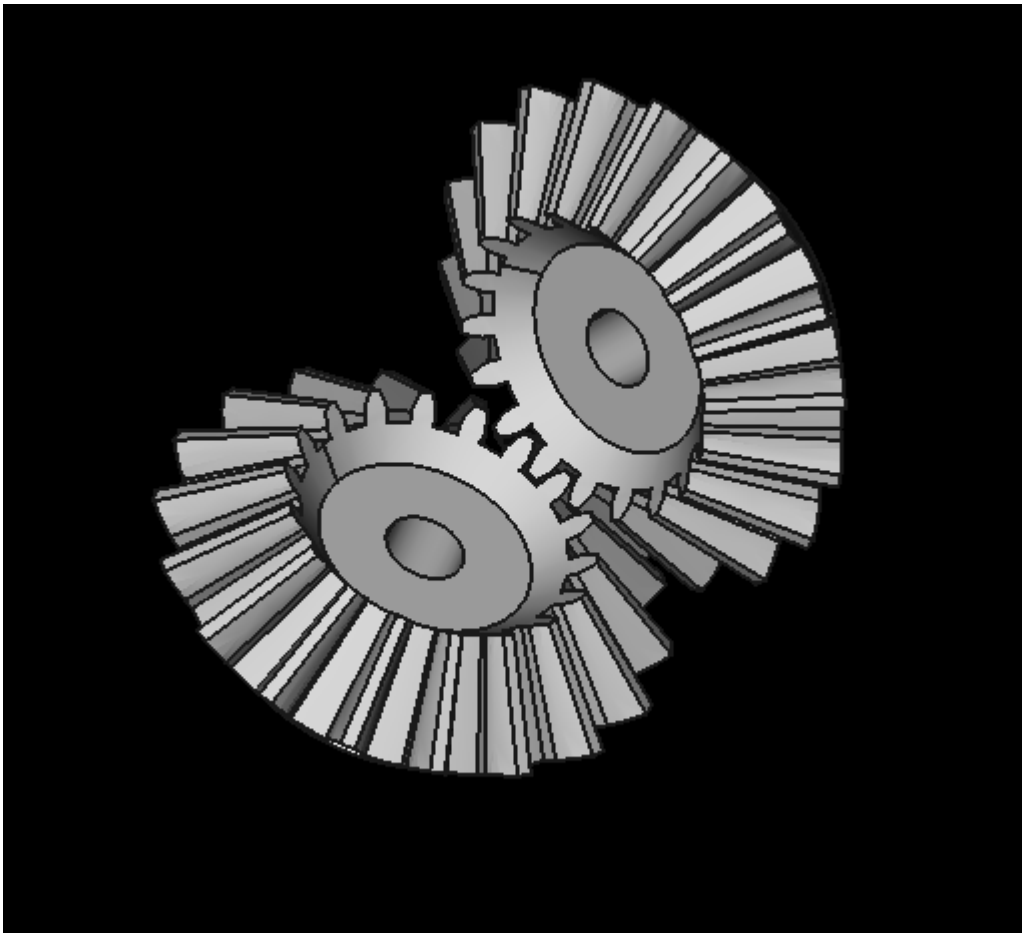

Figura 21:Imagen de piñones de dientes rectos obtenido en el módulo "BevelGear"

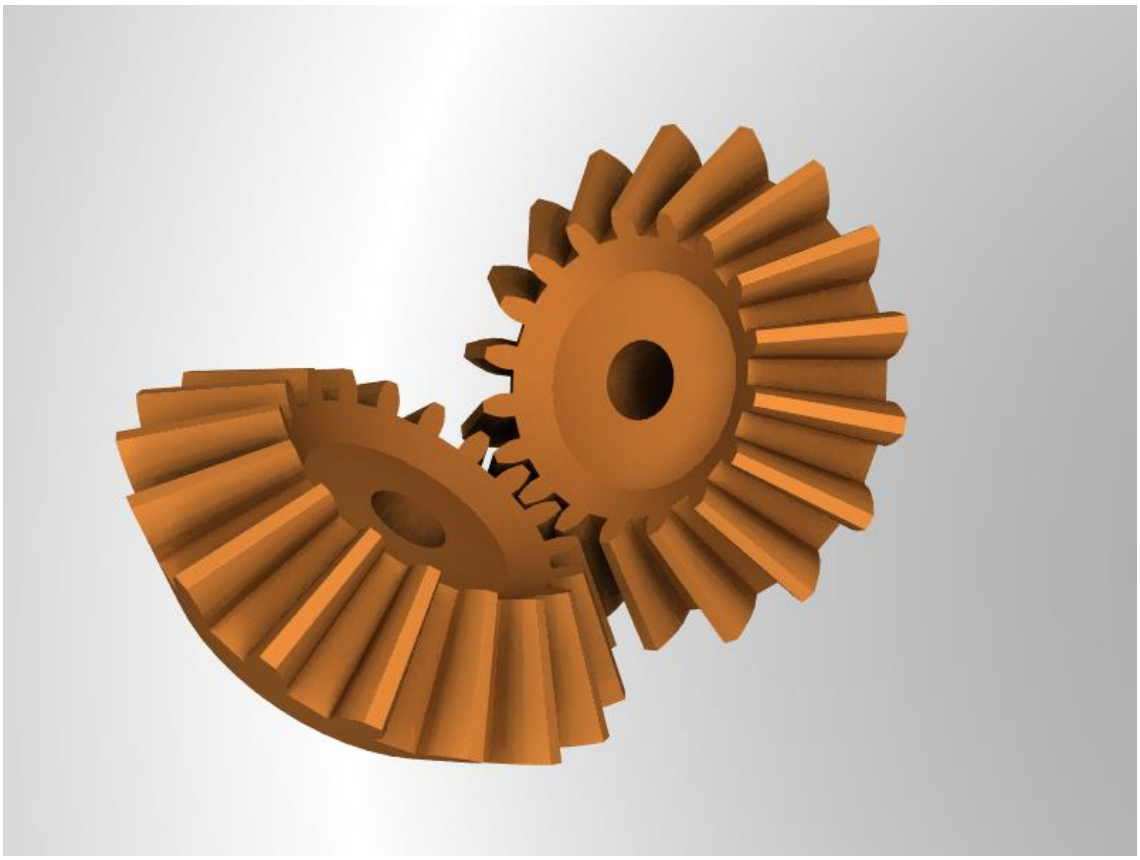

*Figura 22: R*ender obtenido en *FreeCAD* de una pieza generada en el módulo*" BevelGear"*

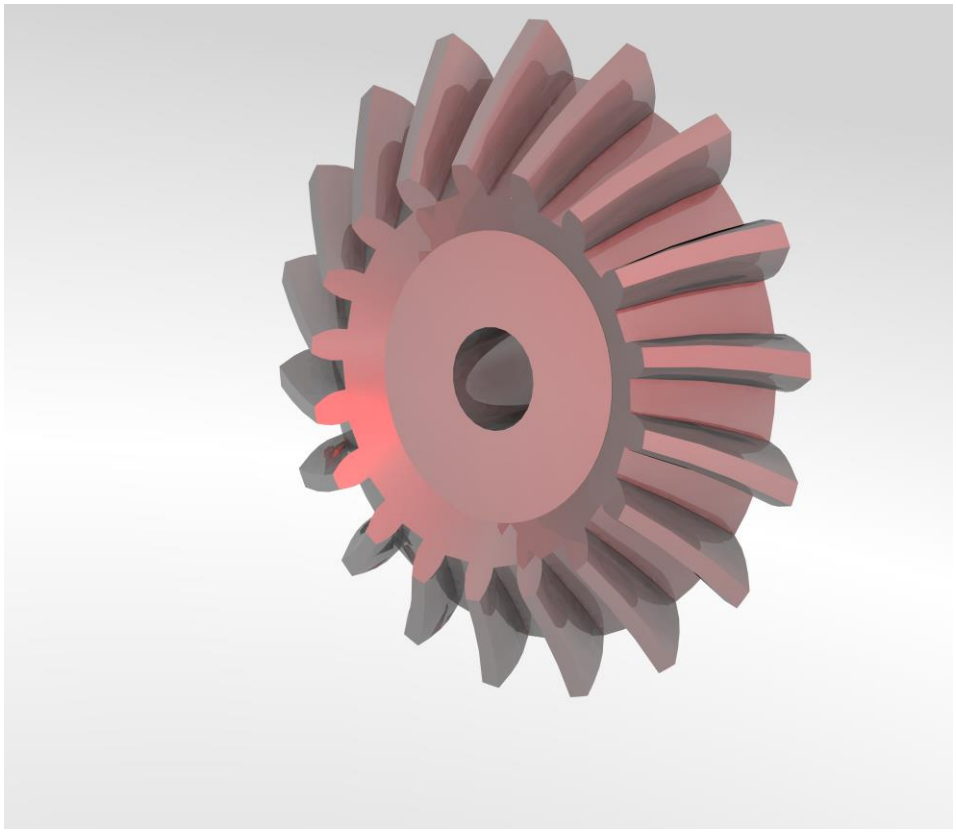

Figura 23:Render obtenido de una pieza creadaen el módulo “BevelGear”

Figura 24:Render obtenido de una pieza importada

Conclusiones

El estudio de los sistemas homólogos y los motores de render existentes permitió establecer las bases para la investigación y seleccionar las herramientas adecuadas para el proceso de desarrollo.

La solución desarrollada permite mayor flexibilidad en la generación de imágenes, mediante la implementación del método de trazado de rayos con el motor “POVRay”, respecto a la aplicación *FreeCAD,* pues incorpora 15 nuevas funcionalidades (textura, relieve, iluminación, etc.) y constituye la primera versión de generador de imágenes para los desarrollos del grupo de investigación SIPII.

Las funcionalidades del módulo permiten aumentar la calidad de las presentaciones sobre resultados obtenidos con los módulos desarrollados.

La obtención de imágenes en tiempo real con aspecto de mayor realismo, requiere la implementación en el *kernel* de la aplicación, de las interfaces para separar el visor, la selección, los proveedores de vistas (ViewProviders) y las clases necesarias para incluir la jerarquía de visores correspondientes a otras tecnologías, que se incluyan a la aplicación.

Recomendaciones

Implementar las funcionalidades detectadas para lograr el *render* en tiempo real.

Mantener actualizada la biblioteca de texturas disponibles.

Referencias Bibliográficas

1. LABORATORIO NACIONAL DE CALIDAD DEL SOFTWARE. Ingeniería de software: metodologías y ciclos de vida. 2009.
2. JAMES RUMBAUGH, IVAR JACOBSON and GRADY BOOCH. El Proceso Unificado de Desarrollo de Software. 2000.
3. SOLIS A, CAMILO J y FIGUEROA D, ROBERTH G. Metodologías de Desarrollo de SW. 2011.
4. LETELIER, Patricio and PENADÉS, M a Carmen. Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea]. 15 de abril del 2006. [Consultado el: 2 de diciembre del 2017]. Disponible en: .
5. ROBERTH G. FIGUEROA, CAMILO J. SOLÍS and ARMANDO A. CABRERA. Metodologías tradicionales vs. Metodologías agiles. [Sin fecha]
6. TAMARA RODRÍGUEZ SÁNCHEZ. Metodología de desarrollo para la actividad productiva de la UCI. 2015.
7. Roger S Pressman. Ingeniería de software: Un enfoque práctico. McGraw-Hill, Nueva York, EUA, 6ta edición, 2007.
8. Gabriel Saldaña. Introducción a Git: Un Sistema de control de versiones...bien hecho.
9. SOMMERVILLE, I., 2005. Ingeniería del software. S.l.: Pearson Educación. ISBN 978-84-7829-074-1.
10. IAN SOMMERVILLE. Ingeniería de Software. 8va Edición. 2006.
11. JEFFRIES, R., ANDERSON, A. y HENDRICKSON, C., 2001. Extreme Programming Installed. S.l.: Addison-Wesley Professional. ISBN 978-0-201-70842-4.
12. CATIA, 2017. CATIA. [en línea]. [Consulta: 14 enero 2018]. Disponible en: .
13. DASSAULT SYSTÈMES, 2017. SolidWorks. [en línea]. [Consulta: 14 enero 2018]. Disponible en: .
14. MONTANO, J.L.M. de O., 2015. La migración hacia software libre en Cuba: complejo conjunto de factores sociales y tecnológicos en el camino de la soberanía nacional. Revista Universidad y Sociedad, vol. 7, pp. 119-125.
15. MARTÍNEZ, R.R., 2004. Criterios para Seleccionar Sistemas de Diseño y Manufactura Asistidos por Computadora (CAD/CAM). Información tecnológica, vol. 15, no. 2, pp. 91-94. ISSN 0718-0764. DOI 10.4067/S0718-07642004000200016.
16. WIPO, 2017. Preguntas frecuentes: los diseños industriales (también denominados dibujos y modelos industriales). World Intellectual Property Organization [en línea]. [Consulta: 16 enero 2018]. Disponible en: /designs/es/faq_industrialdesigns.html.
17. PEÑA PEÑATE, A., 2015. AsiXMec. [en línea]. [Consulta: 18 enero 2018]. Disponible en: .

18. CODE-ASTER, 2017. Code_Aster - Code_Aster. [en línea]. [Consulta: 18 enero 2018]. Disponible en: http://www.code-aster.org/spip.php?article272.

19. CODE-SATURNE, 2017. Code_Saturne. [en línea]. [Consulta: 18 enero 2018]. Disponible en: http://code-saturne.org/cms/.

20. Lux Render, 2018. [en línea, consulta:20 enero 2018]. Disponible en: http://www.luxrender.net/en_GB/index

21. Maxwell Render, 2018. [en línea, consulta:20 enero 2018]. Disponible en: http://www.nextlimit.com/maxwell/

22. Mental Ray, 2018. [en línea, consulta:20 enero 2018]. Disponible en: http://la.nvidia.com/object/nvidia-mental-ray-la.html

23. LARMAN, Craig. UML y Patrones. Madrid: Pearson Educación, 2003.

24. RUMBAUGH, James; BOOCH, Grady; JACOBSON, Ivar. El lenguaje unificado de modelado: manual de referencia. Addison Wesley, 2000.

25. Roger S Pressman. Ingeniería de software: Un enfoque práctico. McGraw-Hill, Nueva York, EUA, 7ta edición, 2010.

26. IAN SOMMERVILLE. Ingeniería de Software. 9va Edición. 2011.

27. Qt Test Overview | Qt Test 5.11. [En línea]. [Sin fecha]. [Consultado el: 20 de mayo del 2018]. Disponible en: http://doc.qt.io/qt-5/qtest-overview.html

28. L. Urdang,editor. *Random House Dictionary of The English Language*. New York: Random House, 1968.

29. Nicolás Belazaras Teardrop: Rendering 3D en tiempo real considerando el contexto lumínico.

30. Brad Smith. Illustration of the Phong Reflection Model.

31. Pistón AutoCAD. Disponible en: https://samyakinfotech.com/training/cad-cam-training-courses-in-jaipur/autocad-for-mechanical-training/

32. Mapa conceptual. [en línea, consulta 10 de mayo de 2018]. Disponible en: http://conceptodefinicion.de/mapa-conceptual/

33. Mapa conceptual. [en línea, consulta 10 de mayo de 2018]. Disponible en: http://tugimnasiacerebral.com/mapas-conceptuales-y-mentales/que-es-un-mapa-conceptual

34. POVRay. Documentación [en línea, consulta 12 de febrero de 2018]. Disponible en: http://www.povray.org/documentation/view/3.6.1/1/

35. Alonso Rodríguez, David; Campillo Tomico, Jorge; Pérez Crespo, Jaime. Radiosidad

36. FreeCAD. FreeCAD Documentation [en línea, consulta 14 de febrero de 2018]. Disponible en: https://www.freecadweb.org/wiki/Feature_list

37. Autodesk. AutoCad [en línea, consulta 14 de febrero de 2018]. Disponible en: https://www.autodesk.com/education/free-software/autocad

38. Autodesk. Inventor [en línea, consulta 14 de febrero de 2018]. Disponible en: https://www.autodesk.com/products/inventor/overview

39. SolidWorks. SolidWorks 2018 [en línea, consulta 14 de febrero de 2018]. Disponible en: http://www.solidworks.es/

40. Siemens PLM. Solid Edge [en línea, consulta 14 de febrero de 2018]. Disponible en: https://www.plm.automation.siemens.com/es/products/solid-edge/

41. Íscar. V-Ray para SketchUp [en línea, consulta 20 de enero de 2018]. Disponible en: https://www.iscarnet.com/vray-para-sketchup/

42. RenderStreet. BLENDER and YAFARAY [en línea, consulta 20 de enero de 2018]. Disponible en: https://render.st/engine-yafaray/

43. STROUSTRUP, Bjarne. *The C++ programming language*. Pearson Education India, 2000.

44. QT Company. About Qt [en línea, consultado 24 de febrero de 2018. Disponible en: https://wiki.qt.io/About_Qt/es

45. Open CASCADE Tecnology. Overview [en línea, consulta 18 de febrero de 2018]. Disponible en: https://dev.opencascade.org/doc/overview/

46. Visual Paradigm. Features [en línea, consulta 19 de febrero de 2018]. Disponible en: https://www.visual-paradigm.com/features/

47. SANTOS GARCIA, Sandy. *Componente para la modelación de engranajes cilíndricos de dientes rectos*. Tesis de grado. Universidad de las Ciencias Informáticas. Cuba, 2015.

48. GONZÁLEZ GARCIA, Gustavo. *Componente para acelerar el diseño de resortes helicoidales*. Tesis de grado. Universidad de las Ciencias Informáticas. Cuba, 2016.

49. GONZÁLEZ PEREZ, Julio Cesar. *Componente para el modelado de vigas para el Sistema CAD2D*. Tesis de grado. Universidad de las Ciencias Informáticas. Cuba, 2016.

50. Tutorial45. Render en AutoCad [en línea, consulta 14 de febrero de 2018]. Disponible en: http://tutorial45.com/autocad-3d-modeling-tips-and-tricks/

Anexos A

Fases AUP	Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración Construcción Transición	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Figura 25: Fases de AUP-UCI (6)

Roles AUP	Roles Variación AUP-UCI
Administrador de proyecto	Jefe de proyecto
	Planificador
Ingeniero de procesos	Analista
Modelador ágil	Arquitecto de información (Opcional)
Desarrollador	Desarrollador
Administrador de la configuración	Administrador de la configuración
Stakeholder	Stakeholder (Cliente/Proveedor de requisitos)
Administrador de pruebas	Administrador de calidad

Figura 26: Roles AUP

Tabla 4: Historia de usuario "Generar una imagen del modelo representado en la aplicación"

Número: 1	**Nombre del requisito:** Generar una imagen del modelo representado.
Programador: Daniel Romero Geigel	**Iteración Asignada:** 1

Prioridad: Alta.	**Tiempo Estimado: 100h**
Riesgo en Desarrollo: alto	**Tiempo Real: 120h**
Descripción: **1- Objetivo:** Permitir al usuario generar una imagen fotorrealista de los objetos modelados. **2- Acciones para lograr el objetivo (precondiciones y datos):** Debe existir un modelo en la aplicación. **3- Comportamientos válidos y no válidos (flujo central y alternos):** Si existe un modelo virtual se crea la imagen y se muestra en la aplicación. **4- Flujo de la acción a realizar:** El usuario hace un clic en el botón “renderizar” Flujo normal: Se crea una imagen del modelo virtual activo en la aplicación. Se muestra la imagen en una vista en la aplicación. Flujo alternativo:	
Observaciones:	
Prototipo de interfaz:	

Tabla 5: Historia de usuario "Selección de la textura deseada"

Número: 2	**Nombre del requisito:** Selección de la textura deseada.

Programador: Daniel Romero Geigel	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado: 90h**
Riesgo en Desarrollo: medio	**Tiempo Real: 84h**

Descripción:

1- Objetivo:

Permitir al usuario seleccionar la textura deseada para cada objeto en la escena.

2- Acciones para lograr el objetivo (precondiciones y datos):

Debe existir un modelo en la aplicación.

3- Comportamientos válidos y no válidos (flujo central y alternos):

4- Flujo de la acción a realizar:

El usuario hace un clic derecho en un modelo en “Entity tree” y le aparece la vista para la selección de la textura.

Flujo normal:

El usuario selecciona una textura para el objeto.

Hace clic en el botón aceptar y se guarda su selección.

Flujo alternativo:

El usuario selecciona la opción cancelar y regresa a la vista anterior sin guardar los cambios.

Flujo alternativo:

El usuario cambia para alguna de las otras opciones que están presentes en la vista.

Observaciones:

Prototipo de interfaz:

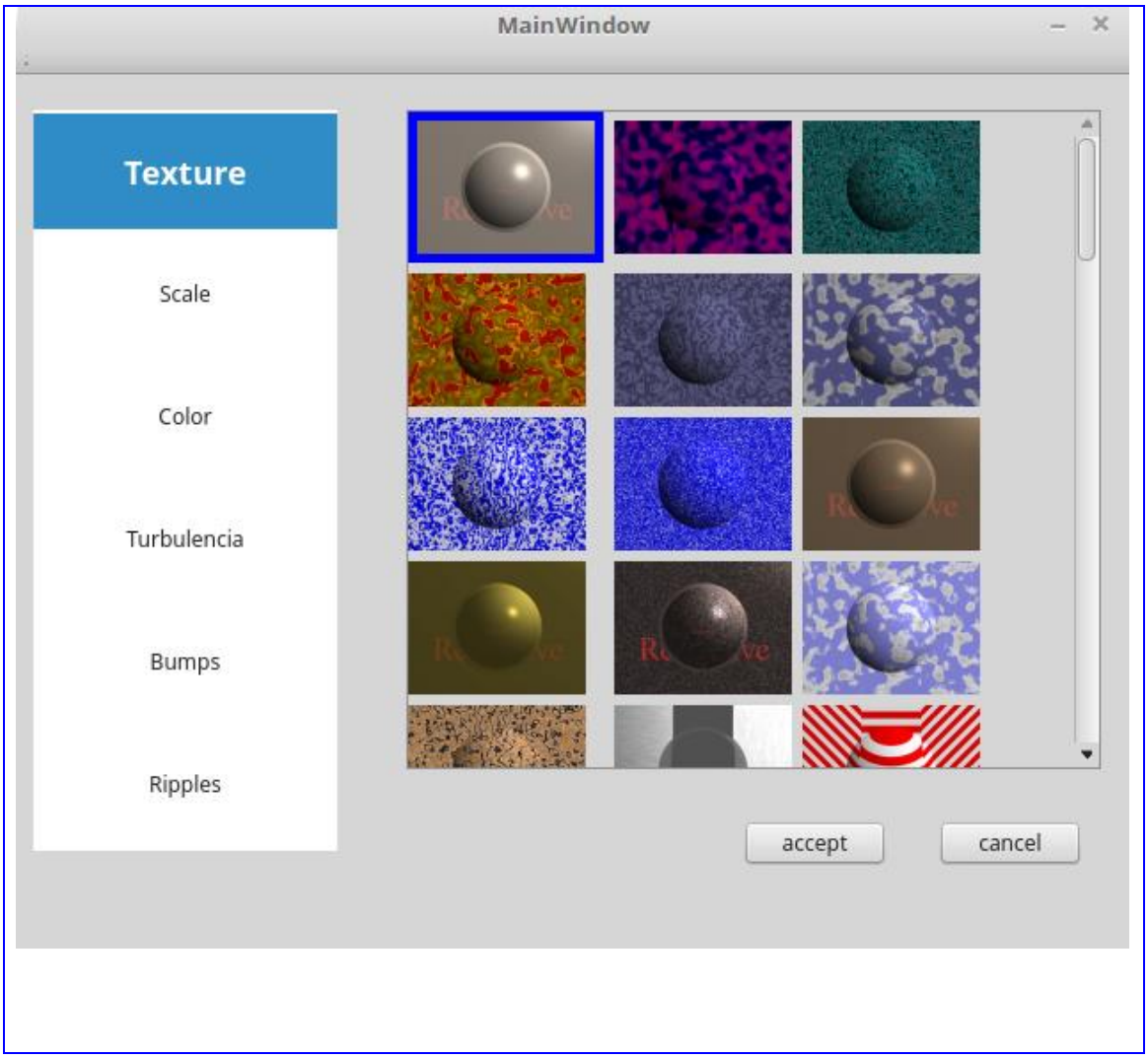

Tabla 6: Historia de usuario "Selección de una textura diferente para cada objeto"

Número: 3	**Nombre del requisito:** Selección de una textura diferente para cada objeto.
Programador: Daniel Romero Geigel	**Iteración Asignada:** 1

Prioridad: Alta.	**Tiempo Estimado: 80h**
Riesgo en Desarrollo: alto	**Tiempo Real: 106h**

Descripción:

1- Objetivo:

Permitir al usuario seleccionar una textura diferente para cada objeto en la escena

2- Acciones para lograr el objetivo (precondiciones y datos):

Debe existir un modelo en la aplicación.

3- Comportamientos válidos y no válidos (flujo central y alternos):

Si existen varios modelos en la escena el usuario puede seleccionar una textura diferente para cada uno de ellos.

4- Flujo de la acción a realizar:

El usuario hace clic derecho sobre un objeto en el “Entity tree” y selecciona la opción “renderOption”.

Se abre una vista y el usuario selecciona las propiedades que desee para ese objeto.

Hace clic en el botón aceptar y se guardan las opciones seleccionadas.

Puede repetir el procedimiento para cada objeto de forma independiente.

Flujo alternativo:

Hace clic en el botón cancelar y se cierra la vista sin guardar los cambios.

Puede repetir el procedimiento para cada objeto de forma independiente.

Observaciones:

Prototipo de interfaz:

Tabla 7: Historia de usuario "Selección de diferentes propiedades para modificar la textura seleccionada"

Número: 4	**Nombre del requisito:** Selección de diferentes propiedades para modificar la textura seleccionada.
Programador: Daniel Romero Geigel	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado: 72h**
Riesgo en Desarrollo: alto	**Tiempo Real: 68h**

Descripción:

1- Objetivo:

Permitir al usuario la selección de diferentes propiedades para modificar las texturas.

2- Acciones para lograr el objetivo (precondiciones y datos):

Debe existir un modelo en la aplicación.

3- Comportamientos válidos y no válidos (flujo central y alternos):

4- Flujo de la acción a realizar:

El usuario selecciona la pestaña de la propiedad que desea modificar y la habilita dándole un clic al *checkBox* “enable”.

El usuario asigna el valor deseado a la propiedad seleccionada.

Flujo alterno:

El usuario inhabilita la propiedad.

Flujo alterno:

El usuario hace clic en el botón cancel y no guarda los cambios

Observaciones:

Prototipo de interfaz:

Tabla 8: Historia de usuario "Selección del color de un objeto"

Número: 5	**Nombre del requisito:** Selección del color de un objeto.
Programador: Daniel Romero Geigel	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado: 100h**
Riesgo en Desarrollo: medio	**Tiempo Real: 96h**
Descripción: **1- Objetivo:** Permitir al usuario seleccionar un color para el objeto. **2- Acciones para lograr el objetivo (precondiciones y datos):** Debe existir un modelo en la aplicación. **3- Comportamientos válidos y no válidos (flujo central y alternos):** Si se cambia el color por defecto se anula cualquier textura que tenga el objeto. **4- Flujo de la acción a realizar:** El usuario selecciona la pestaña del color. El usuario selecciona el color deseado. Flujo alternativo:	

El usuario hace clic en el botón cancel.
Observaciones:
Prototipo de interfaz:

Tabla 9: Historia de usuario "Seleccionar las propiedades del terminado de la escena"

Número: 6	**Nombre del requisito:** Seleccionar las propiedades del terminado de la escena.
Programador: Daniel Romero Geigel	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado: 72h**
Riesgo en Desarrollo: bajo	**Tiempo Real: 60h**

Descripción:

1- Objetivo:

Permitir al usuario seleccionar las propiedades para el terminado.

2- Acciones para lograr el objetivo (precondiciones y datos):

Debe existir un modelo en la aplicación.

3- Comportamientos válidos y no válidos (flujo central y alternos):

El usuario va al “Entity tree” y hace clic derecho sobre el objeto deseado y selecciona la opción “Render Option” y se abre una nueva vista, en esta selecciona el tab “Finish”.

El usuario puede establecer los valores para los diferentes parámetros que aparecen (ambient,

diffuse, specular, roughness, reflection y metallic). **Flujo Alternativo:** El usuario hace clic en cancelar y se cierra la vista.
Observaciones:
Prototipo de interfaz:

Anexos B

Tabla 10:Caso de Prueba de la Historia de Usuario "Generar una imagen del modelo representado en la aplicación"

Generar una imagen del modelo representado en la aplicación			
SC 1 Generar una imagen de un modelo			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Opción para generar una imagen del modelo representado	El usuario selecciona la pestaña correspondiente al módulo de Render y selecciona la opción para generar la imagen, esta se encuentra en el botón con el nombre "Render"	Se genera una imagen con las diferentes características seleccionadas por el usuario y se muestra la imagen resultado en el visor de la aplicación	Render/Render

Tabla 11:Caso de Prueba de la Historia de Usuario "Seleccionar la textura deseada"

Seleccionar la textura deseada			
SC 2 Seleccionar una textura			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 2.1 Opción para seleccionar una textura para un objeto representado	En el "Entity tree" el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Texture y selecciona la textura deseada	Brinda la posibilidad de seleccionar una textura para el objeto seleccionado, además permite cancelar la operación en todo momento	Entity tree/Render Option/Texture
EC 2.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Entity tree/Render Option/Cancel

Tabla 12: Caso de Prueba de la Historia de Usuario "Seleccionar una textura diferente para cada objeto"

Seleccionar una textura diferente para cada objeto			
SC 3 Seleccionar una textura diferente para cada objeto			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 3.1 Opción para seleccionar una textura para cada objeto	En el "Entity tree" el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña	Brinda la posibilidad de seleccionar una textura para el objeto seleccionado y repetir la operación para el resto de los	Entity tree/Render Option/Texture

representado en la aplicación	Texture y selecciona la textura deseada. Al efectuar clic sobre el botón accept se guardan los cambios y puede repetir el procedimiento para los otros objetos	objetos, además permite cancelar la operación en todo momento	
EC 3.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados en el objeto actual sin eliminar los de los otros objetos y cierra la ventana	Entity tree/Render Option/Cancel

Tabla 13: Caso de Prueba de la Historia de Usuario "Seleccionar diferentes propiedades para modificar la textura seleccionada"

Seleccionar diferentes propiedades para modificar la textura seleccionada			
SC 4 Seleccionar diferentes propiedades para modificar la textura			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 4.1 Opción para seleccionar diferentes propiedades para modificar la textura	En el "Entity tree" el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Scale y selecciona los valores deseados para este parámetro	Brinda la posibilidad de establecer diferentes valores para los parámetros deseados, además permite cancelar los cambios en todo momento sin perder los valores o cancelar la operación	Entity tree/Render Option/Scale
EC 4.2 Opción para seleccionar diferentes propiedades para modificar la textura	En el "Entity tree" el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Turbulencia y selecciona los valores deseados para este parámetro	Brinda la posibilidad de establecer diferentes valores para los parámetros deseados, además permite cancelar los cambios en todo momento sin perder los valores o cancelar la operación	Entity tree/Render Option/ Turbulencia

EC 4.3 Opción para seleccionar diferentes propiedades para modificar la textura	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Bumps y selecciona los valores deseados para este parámetro	Brinda la posibilidad de establecer diferentes valores para los parámetros deseados, además permite cancelar los cambios en todo momento sin perder los valores o cancelar la operación	Entity tree/Render Option/Bumps
EC 4.4 Opción de cancelar los cambios en una propiedad	El usuario selecciona la opción Enabled y puede habilitar e inhabilitar el parámetro deseado	Brinda la opción de habilitar e inhabilitar el parámetro “Scale”	Entity tree/Render Option/Scale/Enabled
EC 4.5 Opción de cancelar los cambios en una propiedad	El usuario selecciona la opción Enabled y puede habilitar e inhabilitar el parámetro deseado	Brinda la opción de habilitar e inhabilitar el parámetro “Turbulencia”	Entity tree/Render Option/ Turbulencia/Enabled
EC 4.6 Opción de cancelar los cambios en una propiedad	El usuario selecciona la opción Enabled y puede habilitar e inhabilitar el parámetro deseado	Brinda la opción de habilitar e inhabilitar el parámetro “Bumps”	Entity tree/Render Option/Bumps/Enabled
EC 4.7 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Entity tree/Render Option/Cancel

Tabla 14: Caso de Prueba de la Historia de Usuario “Seleccionar el color de un objeto”

Seleccionar el color de un objeto			
SC 5 Seleccionar un color para el objeto seleccionado			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**

EC 5.1 Opción para seleccionar un color para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Color y selecciona el color deseado	Brinda la posibilidad de seleccionar un color para el objeto seleccionado, además permite cancelar la operación en todo momento	Entity tree/Render Option/Color
EC 5.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Entity tree/Render Option/Cancel

Tabla 15: Caso de Prueba de la Historia de Usuario “Seleccionar las propiedades del terminado de la escena”

Seleccionar las propiedades del terminado de la escena			
SC 6 Seleccionar los ajustes de iluminación y reflexión de un objeto			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 6.1 Opción para seleccionar el terminado para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Finish y selecciona el terminado deseado	Brinda la posibilidad de seleccionar los valores de la luz ambiente, además permite cancelar la operación en todo momento	Entity tree/Render Option/Finish/Ambient

EC 6.1 Opción para seleccionar el terminado para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Finish y selecciona el terminado deseado	Brinda la posibilidad de seleccionar los valores de la luz difusa, además permite cancelar la operación en todo momento	Entity tree/Render Option/Finish/Diffuse
EC 6.1 Opción para seleccionar el terminado para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Finish y selecciona el terminado deseado	Brinda la posibilidad de seleccionar los valores de la luz especular, además permite cancelar la operación en todo momento	Entity tree/Render Option/Finish/Specular
EC 6.1 Opción para seleccionar el terminado para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Finish y selecciona el terminado deseado	Brinda la posibilidad de seleccionar los valores de la rugosidad, además permite cancelar la operación en todo momento	Entity tree/Render Option/Finish/Roughness
EC 6.1 Opción para seleccionar el terminado para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Finish y selecciona el terminado deseado	Brinda la posibilidad de seleccionar la reflexión, además permite cancelar la operación en todo momento	Entity tree/Render Option/Finish/Reflection
EC 6.1 Opción para seleccionar el terminado para un objeto representado	En el “Entity tree” el usuario hace clic derecho sobre el objeto y selecciona la opción Render Option y se muestra una ventana, en esta selecciona la pestaña Finish y selecciona el terminado deseado	Brinda la posibilidad de seleccionar si es brillo metálico o no, además permite cancelar la operación en todo momento	Entity tree/Render Option/Finish/Metallic
EC 7.3 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Entity tree/Render Option/Cancel

Tabla 16: Caso de Prueba de la Historia de Usuario "Seleccionar la ubicación y el color de la fuente de luz"

Seleccionar la ubicación y el color de la fuente de luz			
SC 7 Ajustar las propiedades de la fuente de luz			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 7.1 Opción para ajustar las propiedades de la fuente de luz	El usuario selecciona la opción General Option y aparece una vista, en la pestaña Light Options puede ajustar las propiedades de la fuente de luz (posición, intensidad y color)	Brinda la posibilidad de ajustar las opciones de la fuente de luz, además permite cancelar la operación en todo momento	Render/General Options/ Light Options/
EC 7.2 Opción para poner por defecto la posición de la fuente de luz	El usuario selecciona la opción default light y permite habilitar e inhabilitar la posición por defecto de la fuente de luz	Habilita o inhabilita la posición de la fuente de luz	Render/General Options/ Light Options/default light
EC 7.3 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Render/General Options /Cancel

Tabla 17: Caso de Prueba de la Historia de Usuario "Adicionar otras fuentes de iluminación"

Adicionar otras fuentes de iluminación			
SC 8 Adicionar fuentes de luz			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**

EC 8.1 Opción para adicionar otras fuentes de luz	El usuario selecciona la opción General Option y aparece una vista, en la pestaña Light Options, el botón New light permite adicionar una nueva fuente de luz	Brinda la posibilidad de adicionar fuentes de luz	Render/General Options / Light Options/New light
EC 8.1 Opción para adicionar otras fuentes de luz	El usuario selecciona la opción General Option y aparece una vista, en la pestaña Light Options, el botón Delete light permite eliminar una fuente de luz	Brinda la posibilidad de eliminar fuentes de luz	Render/General Options / Light Options/Delete light
EC 8.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Render/General Options /Cancel

Tabla 18: Caso de Prueba de la Historia de Usuario "Ajustar la calidad de la imagen a generar"

Ajustar la calidad de la imagen a generar			
SC 9 Ajustar los parámetros que controlan la calidad de la imagen a generar			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 9.1 Opción para ajustar los parámetros	El usuario selecciona la opción General Option y aparece una vista, en la pestaña Image Options permite ajustar la calidad de	Brinda la posibilidad de ajustar la calidad, dimensiones y anti-alising de la imagen a generar	Render/General Options / Image Options/

de calidad de la imagen	la imagen, las dimensiones y el anti-alising		
EC 9.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Elimina los cambios realizados y cierra la ventana	Render/General Options /Cancel

Tabla 19: Caso de Prueba de la Historia de Usuario "Guardar la imagen obtenida en una ubicación escogida por el usuario"

Guardar la imagen obtenida en una ubicación escogida por el usuario			
SC 10 guardar la imagen generada			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 10.1 Opción para guardar la imagen generada	El usuario selecciona la opción *Save* y selecciona la ubicación y el nombre de la imagen generada	Brinda la posibilidad de guardar la imagen generada por la aplicación en una ubicación seleccionada por el usuario	Render/ Save
EC 10.2 Opción de cancelar	El usuario selecciona la opción de cancelar	Cierra la ventana sin guardar la imagen	Render/Save /Cancel