PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

GENERACIÓN DE ENTORNOS 3D A PARTIR DE MAPAS DIGITALES

PABLO IGNACIO GALLARDO VARAS

MARIO ANDRES PINO ENFERRIZ

**INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA.**

NOVIEMBRE 2010

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

GENERACIÓN DE ENTORNOS 3D A PARTIR DE MAPAS DIGITALES

PABLO IGNACIO GALLARDO VARAS

MARIO ANDRES PINO ENFERRIZ

Profesor Guía: **Claudio Cubillos Figueroa**

Profesor Co-referente: **Silvana Roncagliolo de la Horra**

Carrera: **Ingeniería de Ejecución en Informática**

NOVIEMBRE 2010

Dedicatoria

“Agradecemos profundamente a nuestros familiares en especial a nuestros padres, por su apoyo y comprensión, así como a nuestros compañeros y compañeras, amigos y amigas quienes nos acompañaron a lo largo de este trayecto”

Índice

Resumen

El presente informe tiene como objetivo explicar el contexto de este proyecto, el cual consistió en el desarrollo un sistema capaz de generar un modelo 3D, el cual representa el terreno de una zona geográfica obtenida de mapas digitales. Dicho modelo puede utilizado en aplicaciones de simulación 3D.

Para esto se utilizaron las bibliotecas GDAL y OGR para obtener los datos de los mapas digitales de la misma forma que en los Sistema de Información Geográfica. Para 3D se utilizó la librería *Irrlicht* la cual cuenta con las herramientas necesarias para generar un modelo 3D y para posteriormente exportarlo a un archivo. Para validar esta solución se utilizó *Blender 3D* para importar y verificar si el modelo 3D era generado de forma correcta.

Se identificaron distintas aproximaciones para resolver la problemática del proyecto en cuanto al análisis de mapas digitales. Se encontró la forma de generar un modelo 3D a partir de un estos mapas. Finalmente, se demostraron algunos de los usos para los modelos 3D generados con el sistema.

Palabras-claves: Sistemas de Información Geográfica, Modelado 3D, Generación de terrenos.

Abstract

This report aims to explain the context of this project, which consists in the development of a software system capable of generating a 3D model that represents the surface of a geographical area obtained from digital maps. The given model can be used in 3D simulation software.

This was accomplished by using GDAL and OGR libraries to obtain the map data in the same way as in Geographic Information Systems. Irrlicht Engine was used for the creation and exporting of the 3D model. To validate this solution Blender 3D was used to check if the 3D model was generated correctly.

Different approaches were identified to solve the problems of this project on the analysis of digital maps. A way was found to generate a 3D model from these maps. Finally, some uses for the generated 3D models were shown.

Key-words: Geographic Information Systems, 3D Modeling, Terrain generation.

Lista de Figuras

Lista de Tablas

1 Descripción del Proyecto

1.1 Introducción

En la actualidad los avances realizados sobre los Sistemas de información geográfica han permitido a los usuarios crear consultas interactivas, analizar información espacial, editar datos, mapas y presentar los resultados obtenidos, de forma mucho más automática y cómoda. El objetivo principal de los SIG es ofrecer ayuda en la toma de decisiones gracias al análisis e interpretación de los datos que estos proveen. El desarrollo de estos sistemas ha dado paso a la creación de distintas extensiones, las cuales ofrecen distintas representaciones de los datos para adaptarse a las necesidades de los usuarios.

Hoy en día la tecnología de gráficos 3D, es una de las más utilizadas en la industria cinematográfica, televisión, videojuegos y hasta en aplicaciones científicas. Ésta tecnología se ha masificado de tal manera, que puede ser encontrada en entornos Web, y aplicaciones para dispositivos *handheld*, entre otras más. Por esto no es extraño encontrar implementaciones de esta tecnología aplicadas al campo de los SIG, en el caso de estos la tecnología 3D es utilizada más que nada, en la generación de gráficas con relieve, las cuales representan valores geográficos o estadísticos, estas gráficas suelen ser presentadas sobre un planeta 3D.

Por esto es natural que el siguiente paso, sea utilizar la información provista por los SIG, en más que simples representaciones de datos estadísticos, matemáticos o geográficos. Por ejemplo, el software *Google Earth* que en base a algoritmos de procesamiento de imágenes e información geográfica, genera una representación 3D del planeta, y en algunos casos, representaciones tridimensionales de ciudades completas.

Lo mencionado anteriormente es la base para el desarrollo del problema que se presentara en este informe. Éste proyecto está enfocado a estudiar el funcionamiento y tipos de datos utilizados en los SIG, con el propósito de utilizarlos en la creación, de representaciones tridimensionales del terreno descrito en estos. Dichas representaciones podrán serán utilizadas en aplicaciones que soporten tecnología 3D, especialmente para simulación.

A continuación se presenta la información obtenida durante la investigación, el diseño del sistema y detalles acerca de la planificación necesaria para el correcto desarrollo y cumplimiento de los objetivos de este proyecto.

1.2 Definición de Objetivos

En esta sección se listan y definen los objetivos del proyecto.

1.2.1 Objetivo General

El desarrollo de una aplicación capaz de generar un terreno 3D utilizando la información obtenida de distintos mapas digitales.

1.2.2 Objetivos Específicos

- Estudiar algunos de los SIG más conocidos, para así identificar los origines de datos que serán útiles para el sistema a desarrollar.

- Buscar el *framework* 3D con las funcionalidades y características más adecuadas para el desarrollo de la aplicación.

- Generar un modelo 3D a partir de los datos provistos por los orígenes de datos escogidos para luego exportarlo a un formato de archivo para modelos 3D.

- Usar una herramienta de modelado para validar la salida del sistema.

- Diseñar y realizar pruebas para el sistema desarrollado.

1.3 Metodología

Para este proyecto se analizaron varias metodologías, entre las cuales resaltaban la metodología RUP y AUP. Se prosiguió a hacer un análisis más a fondo de estas 2 metodologías, el cual está presentado a continuación.

La metodología AUP (*Agile Unified Process*) o en español Proceso Unificado Ágil, es una propuesta de Scott Ambler [Ambler, 11]. Esta describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP [Jacobson et al., 99].

Las disciplinas de AUP, a diferencia de RUP, están desarrolladas para simplificar la documentación, análisis y toma de requerimientos necesarias para llevar a cabo un proyecto. Para esto se optó por unir en AUP, otras tres disciplinas de RUP que implican bastante documentación, conocidas como Modelado del Negocio, Toma de requerimientos, Análisis y Diseño, en una sola conocida como Modelo.

Como esta metodología beneficia proyectos basados en pequeños grupos de desarrollo, otorgando facilidades en la comunicación entre el equipo, implicando un conocimiento más amplio del proyecto. Razón por la cual no se necesita un proceso detallado en la documentación para comunicar los requerimientos, con su análisis y diseño.

AUP optimiza el uso del tiempo, al describir concisamente usando las páginas necesarias, y no miles de ellas. Enfocándose más a los prototipos y a la retroalimentación del usuario en las

iteraciones, logrando una satisfacción del usuario al tener un proyecto palpable y la identificación más temprana de los posibles problemas del software.

Un punto interesante de esta metodología es que se puede usar cualquier conjunto de herramientas de desarrollo y modelado. Abaratando costos y dando más opciones en la forma de hacer el modelado.

Como el equipo de desarrollo de este proyecto está compuesto por dos personas y este proyecto es de carácter investigativo por lo que no hay un diseño establecido; con el objetivo de abaratar costos y al disponer de tiempo limitado para la finalización de este proyecto. Se optó por AUP como la metodología más adecuada.

1.4 Plan de trabajo

Una de las grandes responsabilidades dentro del desarrollo del proyecto es el plan de trabajo, dado que aquí se ordenan las tareas y metas propuestas para hacer seguimiento de éstas, y así asegurar el cumplimiento de los plazos establecidos, pues de esto depende el éxito del proyecto.

Id	Nombre de tarea	Duración
1	**Inicio**	**28 días**
2	Investigar tema	24 días
3	Establecer el ambito del proyecto y sus limites	12 días
4	Investigar tecnologias involucradas	14 días
5	Identificar riesgos del proyecto	5 días
6	Redactar informe proyecto	18 días
7	Validacion de informe	3 días
8	Entrega de Informe de Avance	0 días
9	**Elaboración**	**50 días**
10	Acotar las tecnologias involucradas	15 días
11	Comenzar desarrollo del primer prototipo	14 días
12	Diseñar plan de pruebas	5 días
13	Entregar primer prototipo	0 días
14	Redactar Informe final	25 días
15	Validacion Informe final	3 días
16	Entrega de Informe	0 días
17	**Construcción**	**46 días**
18	Elaboración del segundo prototipo	9 días
19	Desarrollo sistema final	38 días
20	Validacion del informe	3 días
21	Entrega informe de avance	0 días
22	**Transición**	**21 días**
23	Pruebas de validacion del sistema final	15 días
24	Entrega sistema final	0 días
25	Validación informe final	6 días
26	Entrega de informe final	0 días

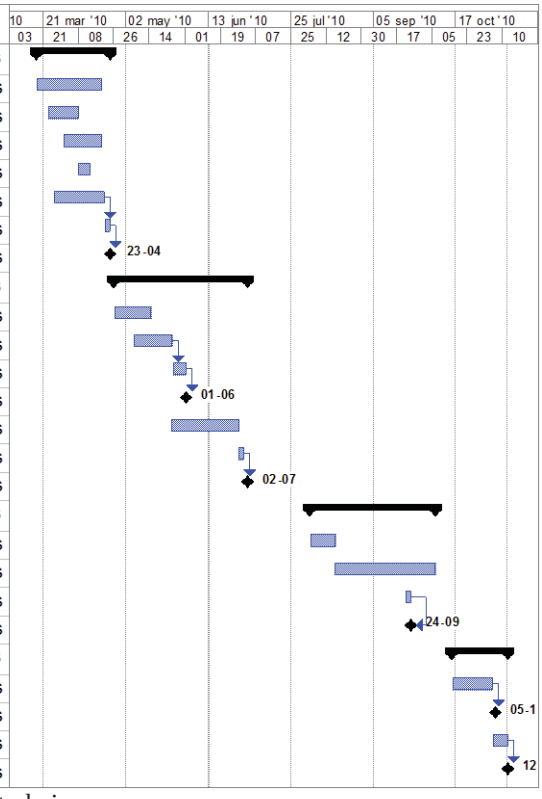


Figura 1.1: Plan de trabajo.

1.5 Estudio de Factibilidad

La importancia del estudio de factibilidad en los proyectos informáticos está dada por la cantidad limitada de recursos, lo que nos lleva a buscar las decisiones más apropiadas para hacer un uso correcto de estos. Razón de por qué los análisis de estudio de factibilidad técnica,

económica, legal y operacional, son fundamentales para saber si es viable poner en marcha el desarrollo del proyecto. Se presenta en las siguientes secciones el resultado de los estudio de factibilidad para este proyecto:

1.5.1 Factibilidad Técnica

Para la realización de todo proyecto de software se necesita analizar la disponibilidad de recursos humanos y tecnológicos (hardware y software), además de los riesgos que pudieran encontrarse durante su desarrollo. Dado que en la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso se cuenta con laboratorios, que se pueden acceder 6 días a la semana, donde se puede trabajar en la investigación de las tecnologías requeridas para el proyecto, se suple en gran medida los requerimientos tecnológicos y de hardware del proyecto.

Los *frameworks* estudiados en su mayoría están programados en lenguaje C/C++, por este motivo se optara por la IDE Visual Studio 2008 Express debido a que ofrece un gran número de herramientas que facilitan la programación y que además provee un entorno en el cual la navegación entre clases resulta sencilla.

En cuanto a las herramientas para el modelado del software y la redacción de este informe se optó por Microsoft Visio 2003 y Microsoft Office 2003, ya que la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso cuenta con las licencias para la utilización de estas.

1.5.2 Factibilidad Económica

A causa del carácter enfocado a la investigación de este proyecto, y no al uso empresarial, la tasa de retorno del capital no se considerará como un punto para tomar una decisión. Sin embargo existen otros puntos que se tuvieron consideración, como lo es el costo del hardware y software necesario para desarrollar e implementar el sistema.

Entre los Sistemas de Información Geográfica que permiten modificar su código, por costo las 3 opciones son válidas para el proyecto respecto al precio, demostrado en la Tabla 1.1.

Tabla 1.1: Sistema de Información Geográfica.

Herramientas	Costo
GRASS	$ 0
gvSIG	$ 0
Quantum GIS	$ 0

Entre los *frameworks* 3D, están las que se ven en la Tabla 1.2:

Tabla 1.2: Frameworks 3D.

Herramientas	Costo
XNA	$ 0
DarkGDK	$ 36.837
OGRE3D	$ 0
IRRLICHT	$ 0
JMonkeyEngine	$ 0
Panda3D	$ 0

Entre otras tecnologías usadas, se tienen en la Tabla 1.3:

Tabla 1.3: Tecnologías Misceláneas.

Herramientas	Costo
Visual Studio Express 2008	$ 0
StarUML	$ 0

Entre los mapas cotizados de Chile en el Instituto Geográfico Militar, se tienen en la Tabla 1.4:

Tabla 1.4: Mapas cotizados

Mapa	Costo
Cartas Vectorizadas 3D Datum Sirgas con escala 1:50000	$ 93.500
Cartas en formato Tiff georreferenciado con escala 1:50000	$ 29.500
Cartas en formato Jpg con escala 1:25000	$ 18.000

Considerando que para la correcta ejecución de la herramienta de validación del software, es necesario un computador con una tarjeta aceleradora de video, con soporte mínimo para DirectX 9.0, se presentan en la Tabla 1.5 los siguientes costos en cuanto al hardware.

Tabla 1.5: Hardware requerido.

Partes	Costo
CPU: AMD Athlon II X2 240	$ 45.106
Placa Madre: Asus AMD M4N68T-M LE (AM3)	$ 33.936
Tarjeta de Video: XFX Video PCIE HD4350 1024mb DDR2	$ 34.255
RAM: Corsair DDR2 2GB 667Mhz VS2GB667D2 box	$ 33.830
Disco Duro: Western Digital Disco Duro Sata2 160Gb 7200 rpm	$ 30.000
Monitor: Samsung SyncMaster 933SN+Plus	$ 81.904
Mouse, Teclado y Gabinete con fuente de poder:	$ 30.000
Total	$ 289.031

La suma mínima de los costos de software es $ 0, lo cual valida la factibilidad económica del proyecto, sin embargo, entre el costo de un mapa Vectorizado 3D, en conjunto de un mapa

de textura georeferenciado y el hardware necesario, el valor asciende a $ 412.031 lo cual es una suma nada despreciable si se toma el proyecto como particular. Pero como el sistema desarrollado es para la escuela, los gastos en hardware, podrían desaparecer utilizando los equipos proporcionados por los laboratorios de la universidad y asimismo el Instituto Geográfico Militar envió un mapa de muestra sin costo el que fue utilizado en el desarrollo del proyecto.

1.5.3 Factibilidad Legal

El software empleado para el desarrollo del sistema Visual Studio 2008 Express es de licencia gratuita. Para la documentación del proyecto se trabaja en los equipos de la Universidad, los cuales cuentan con todas las licencias necesarias para la utilización de las herramientas de diseño y ofimática. De igual manera se aplica para el uso de los sistemas operativos. En conclusión, siguiendo lo establecido anteriormente, el proyecto no tendrá problemas del tipo legal, además cabe señalar que las tecnologías a escoger se evaluaran con el criterio de mantenerse dentro de las normas y estándares requeridos para su completa construcción.

Según el Decreto con Fuerza de Ley Nº 83 (Diario Oficial de 27 de marzo de 1979), sobre el Estatuto Orgánico de la Dirección Nacional de Fronteras y Limites del Estado, el cual señala:

"ARTICULO 2° Corresponderá a la Dirección Nacional de Fronteras y Límites del Estado:
g) Autorizar la internación de mapas, cartas geográficas y publicaciones referentes o relacionadas con los límites internacionales y fronteras del territorio nacional. Asimismo, le Corresponderá autorizar la edición y circulación de tales instrumentos, previa su revisión."

El equipo de proyecto debe abstenerse de poner en circulación o de modificar la información de los mapas con los que se trabajará.

1.5.4 Factibilidad Operativa

Con respecto a este punto, el equipo de desarrollo cuenta con los conocimientos básicos en cuanto al uso de *frameworks* 3D. De esta manera podemos decir que si la planificación se cumple como ha sido estipulada, se contara con el tiempo suficiente para estudiar, entender y aplicar sin dificultades las tecnologías antes mencionadas.

1.6 Análisis de Riesgos

Todo proyecto informático está expuesto a sufrir contratiempos durante su desarrollo y el mal tratamiento de estos puede llevar a grandes pérdidas o incluso el fracaso del mismo. Es por esta razón que identificar los riesgos en etapas tempranas del desarrollo resulta fundamental. Estos riesgos deben clasificarse y por cada uno crear un plan de contingencia y mitigación para poder enfrentarlos y así evitar efectos indeseables sobre el proyecto.

1.6.1 Identificación de Riesgos

La identificación del riesgo es un intento sistemático para especificar las amenazas al plan del proyecto (estimaciones, planificación temporal, carga de recursos, etc.). Identificando los

riesgos conocidos y predecibles, se puede dar un paso adelante para evitarlos cuando sea posible y controlarlos cuando sea necesario. En la Tabla 1.6 se encuentran los riesgos identificados para este proyecto. Los riesgos pueden clasificarse de la siguiente manera:

- **Riesgos del proyecto:** amenazan la planificación del proyecto, afectan la calendarización o los recursos, por ello conllevan un retraso en los tiempos y un aumento en los costos.
- **Riesgos de producto:** afectan la calidad o desempeño del software que se está desarrollando, si los riesgos de este tipo se llegan a hacer realidad, la implementación puede volverse difícil o imposible.
- **Riesgos del negocio:** afectan a la organización que desarrolla o procura el software.

Tabla 1.6: Riesgos identificados.

Descripción del Riesgo	Tipo de Riesgo	Probabilidad	Impacto	Identificador
Incumplimiento de los plazos establecidos.	Proyecto, Producto	60%	Catastrófico	1
Se retira o aparta algún integrante del equipo de trabajo.	Negocio	5%	Catastrófico	2
Los miembros del equipo no conocen o no dominan las tecnologías a utilizar para el diseño e implementación del proyecto.	Producto, Negocio	40%	Catastrófico	3
El dimensionamiento del proyecto no se mantiene dentro de los márgenes establecidos.	Proyecto	30%	Serio	4
El tiempo dedicado al proyecto se ve disminuido por otras actividades.	Proyecto	40%	Serio	5
Los hitos de revisión de avance no son cumplidos.	Proyecto, Producto	70%	Serio	6

1.6.2 Gestión de Riesgos

Cada riesgo identificado debe tener asociado un plan de mitigación, con el fin de disminuir los efectos en el caso de que estos lleguen a ocurrir. Las medidas a tomar para cada uno de los riesgos identificados anteriormente son dados a conocer en la Tabla 1.7:

Tabla 1.7: Planes de mitigación.

Identificador de Riesgo	Medidas para minimizar o mitigar el riesgo	Plan de contingencia asociado
1	Evaluación y seguimiento de las actividades a desarrollar en el proyecto.	1
2	Reuniones periódicas para evaluar y motivar a los integrantes del equipo.	2
3	Debates periódicos en los cuales se evaluaran el nivel de conocimientos de los integrantes y su nivel de aprendizaje.	3
4	Establecer los límites del sistema en forma temprana y clara.	4
5	Revisiones semanales de los grados de avance en las actividades a cumplir.	5
6	Planificar y dividir equitativamente los tiempos para la responder en cada una de las actividades.	6

Los planes de contingencia son el último recurso, en caso que los planes de mitigación no hayan sido suficientes como para evitar que los riesgos ocurrieran. Dado que los planes de contingencia tienen un costo asociado, estos se utilizan únicamente sobre riesgos de carácter serio o catastrófico. A continuación en la Tabla 1.8 se muestran los planes de contingencia asociados a los riesgos mayores:

Tabla 1.8: Planes de contingencia.

Plan de Contingencia	Descripción del plan de contingencia.
1	Ajustar nuevamente los plazos, para cumplir con las fechas fijadas.
2	Adaptar los objetivos originales del proyecto para disminuir la carga sobre el miembro del equipo restante.
3	Comenzar con horarios establecidos en los cuales se realizarán cursos y charlas sobre las tecnologías a utilizar.
4	Volver a fijar los límites del sistema y dejar en forma escrita una constancia que los establezca.
5	Fijar horarios de trabajo fijos, y penalizar el incumplimiento de estos horarios.
6	Establecer horas fijas a cada actividad y velar por el cumplimiento de estas.

1.7 Estructura del documento

El resto del documento está organizado de la siguiente manera:

- **El segundo capítulo:** Este capítulo trata sobre el estado del arte, que está dividido en:
  - **Sistemas de Información Geográfica**: Se habla de los distintos Sistemas de Información Geográfica existentes en el mercado y sus métodos de almacenamiento.
  - **Mapas Digitales:** Se habla de los mapas digitales con sus respectivos modelos de datos para archivos *raster* y vectorial, interpolación de mapas de curvas de nivel mediante Redes Irregulares de Triángulos (TIN), sistemas de coordenadas proyectadas y sistemas de referencia geodésicos.
  - **Gráficos Computacionales 3D:** Se habla de las API 3D, así como los distintos framework hechas en base a estas API disponibles en el mercado.
  - **Modelos 3D:** Se habla de las mallas poligonales y las distintas formas de representación de estas, así como el mapeo de sus texturas. Se habla también de cómo generar un terreno 3d mediante un mapa de altura, los diferentes formatos de archivo para almacenar modelos 3D y herramientas de modelado 3D.
  - **Unreal Engine 3:** Se habla de UDK/ Unreal Engine 3 y de las distintas características que este motor gráfico tiene para ofrecer.
- **El tercer capítulo:** Se presenta el caso de estudio, el diseño del sistema con los distintos diagramas que explican cómo está compuesto el sistema y como se relaciona con las distintas librerías usadas, así como los diagramas de casos de uso, casos de uso extendidos, clases, secuencia y actividad. Se habla también del plan de pruebas a seguir, sus resultados y de cómo generar mapas para Unreal Engine 3 usando el sistema desarrollado.
- **El cuarto capítulo:** Presenta las conclusiones que se obtuvieron en la realización del trabajo.

2 Estado del arte

En la realización de este proyecto se hizo uso de diversas tecnologías con el fin de integrar la información precisa de los mapas digitales que manejan los Sistemas de Información Geográfica y producir un modelo 3D en base a dicha información. Por esto primero se debe conocer a fondo los distintos componentes de las tecnologías a utilizar. Durante este capítulo se hablará del estado del arte de las tecnologías involucradas en la realización de este proyecto.

2.1 Sistemas de Información Geográfica

Los Sistemas de Información Geográfica (SIG) son sistemas que permiten a los usuarios emplear las herramientas que los cartógrafos y geógrafos usan. Para entender más a fondo los SIG, hay que entender que hacen los cartógrafos y geógrafos, lo cual está definido a continuación:

"Geografía analiza y explica el comportamiento humano y del entorno y los procesos que tienen lugar en la superficie de la tierra, mejorando nuestra comprensión del mundo.
Cartografía desarrolla las teorías, conceptos, y habilidades para describir y visualizar los objetos y eventos o patrones y procesos desde la geografía, y comunicar lo comprendido." [Harvey, 08].

Estas dos son representaciones que utilizan nuestras habilidades cognitivas, están influenciadas culturalmente y socialmente por el conocimiento del mundo. Lo que la geografía analiza y explica, la cartografía lo comunica visualmente. Con estas bases los SIG permiten capturar, almacenar, analizar, administrar y presentar los datos de una zona específica en el mapa, gracias a una base de datos, para ser interpretados por el usuario.

El objetivo principal de los SIG es ofrecer ayuda en la toma de decisiones, esto gracias a la interpretación y análisis de los datos que proveen. Dándoles distintos usos, como por ejemplo obtener la mejor localización para ubicar una compañía de bomberos, en base a las zonas que sufren de mayor cantidad de incendios. Con este propósito, los sistemas SIG operan sobre una base de datos para almacenar y obtener su información. Conocidas como bases de datos espaciales, las cuales consisten en bases de datos optimizadas para guardar y consultar datos relacionados a objetos en el espacio, incluyendo puntos, líneas y polígonos. Mientras las bases de datos típicas soportan tipos de datos numéricos y de carácter, se necesitan agregar funcionalidades adicionales a las bases de datos para procesar datos de tipo espacial. Estos son llamados geometría o característica.

Para guardar los mapas se usa el concepto de escala, que determina cuantos detalles puede contener el mapa y el de categoría que representa el tipo de objeto al que nos referimos, como por ejemplo un bosque, una carretera o una iglesia. En base a la escala en el modelo conceptual geográfico la información se representa en base a 4 formas básicas:

- **Puntos:** Sin dimensión, o sea no tienen ancho, ni alto. Usados para representar características geográficas en los mapas. Por ejemplo una ciudad puede ser representada por un punto a cierta escala.

- **Líneas:** A diferencia del concepto en la vida real de línea, en SIG las líneas tienen una dimensión, que es el largo de la línea. Se usan para representar por ejemplo carreteras, líneas de tren, cercas, limitaciones entre países.

- **Polígonos:** Más conocidos como áreas. Con sus dos dimensiones, ancho y alto, sirven para medir perímetro, área, figura. Se usan típicamente en regiones políticas como estados y países, tierras cubiertas por ciertas sustancias, campos.

- **Superficies:** Tienen tres dimensione, ancho, alto, y una tercera dimensión determinada por las características de la superficie. Superficies físicas permiten calcular volúmenes, predecir tendencias, y determinar la dirección del flujo de líquidos.

2.1.1 Aplicaciones SIG

Existe una amplia gama de aplicaciones de Sistemas de Información Geográfica que usan una combinación de mapas digitales e información georeferenciada para cumplir distintos objetivos. En esta sección se presentan las herramientas más relevantes para el desarrollo del proyecto.

- **GRASS GIS**

Originalmente desarrollado por el cuerpo de Ingenieros del Laboratorio de Investigación de Ingeniería de la Construcción del Ejercito de los Estados Unidos, GRASS (*Geographic Resourse Analysis Support System*) es el más poderoso y posiblemente el más común de los SIG *Open Source* del mercado. Este está compuesto de varios módulos que soportan modelos de datos *Raster* y de vectoriales, para más información ver [GRASS, 11] y [James, 04]. Las características más relevantes son:

- Soporte para análisis de datos *Raster*.
- Manejo de volúmenes 3D.
- Modelos vectoriales en 2D y 3D con soporte para Sistemas de Administración de Bases de datos en SQL
- Es de Código Abierto.
- Gran complejidad en su uso e implementación, con una curva de aprendizaje demasiado alta.

Un módulo de GRASS llamado NVIZ permite a los usuarios dibujar múltiples superficies en un espacio 3D, opcionalmente usando colores temáticos, agregando archivos vectoriales de GRASS sobre las superficies. El modulo se presenta en la Figura 2.1.

- **ARCGIS**

Creado por el ESRI (*Environmental System Research Institute*), Arcgis es el conjunto de varias aplicaciones pagadas, de código cerrado, para la captura, edición, análisis, tratamiento, diseño, publicación e impresión de información geográfica, soportando todas estas los modelos de datos *Raster* y Vectoriales, para más información revisar [ESRI, 11].

Una de las extensiones más relevantes es el "ARCGIS 3D *Analyst*", emplea una tecnología para visualizar gran cantidad de datos geográficos en 3D, desde una perspectiva

local, hasta una global. Llamada ArcGlove permitiendo al usuario interactuar continuamente con cualquier dato geográfico en un Globo en 3D de manera similar a *Google Earth*, solo que con el poder de análisis de un SIG, para más información ver [Pilouk, 07] y [ESRI, 10]. El uso de la extensión se ve en la Figura 2.2:

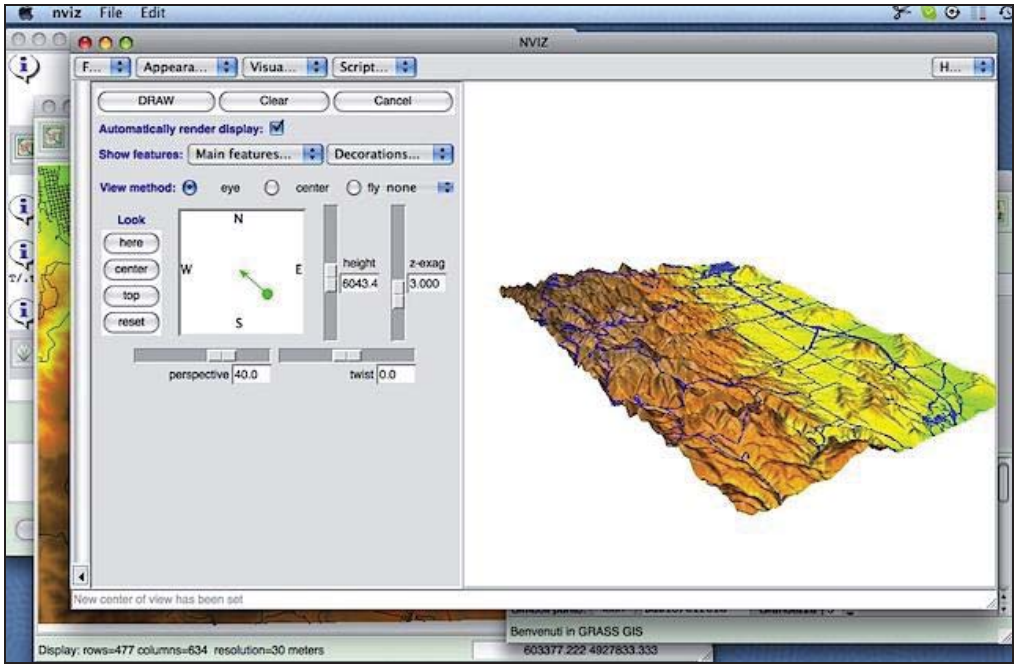


Figura 2.1: módulo NVIZ de GRASS. [Elaboración Propia]

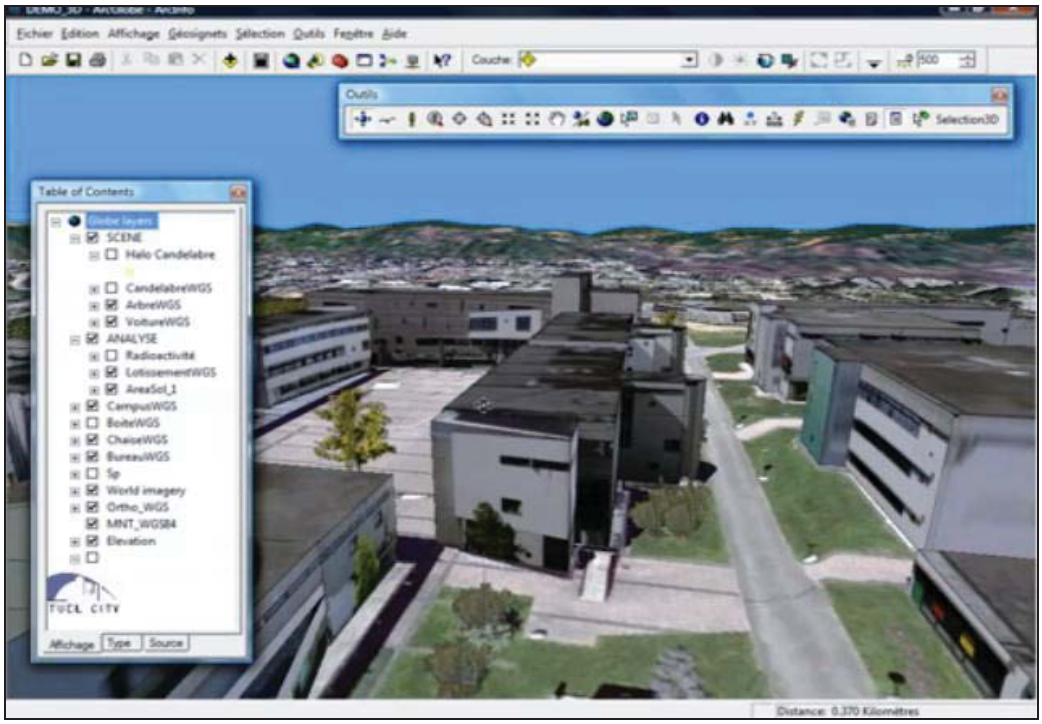


Figura 2.2: ARCGIS usando la tecnología ARCGlove [ESRI, 10].

- **GvSIG**

Es una herramienta *Open Source* desarrollada en JAVA, bajo la licencia GPL, orientada al manejo de información geográfica. Se caracteriza por una interfaz amigable, siendo capaz de acceder a los formatos más usuales de forma ágil tanto *Raster* como vectoriales, para más información ver [gvSIG, 11]. GvSIG tiene una extensión llamada gvSIG 3D la cual permite la creación de vistas en tres dimensiones, tanto planas como esféricas, para más información ver [gvSIG, 10]. En ellas pueden cargarse capas locales o remotas, y utilizar la mayoría de las funcionalidades disponibles para 2D. La extensión gvSIG 3D se puede apreciar en la Figura 2.3:

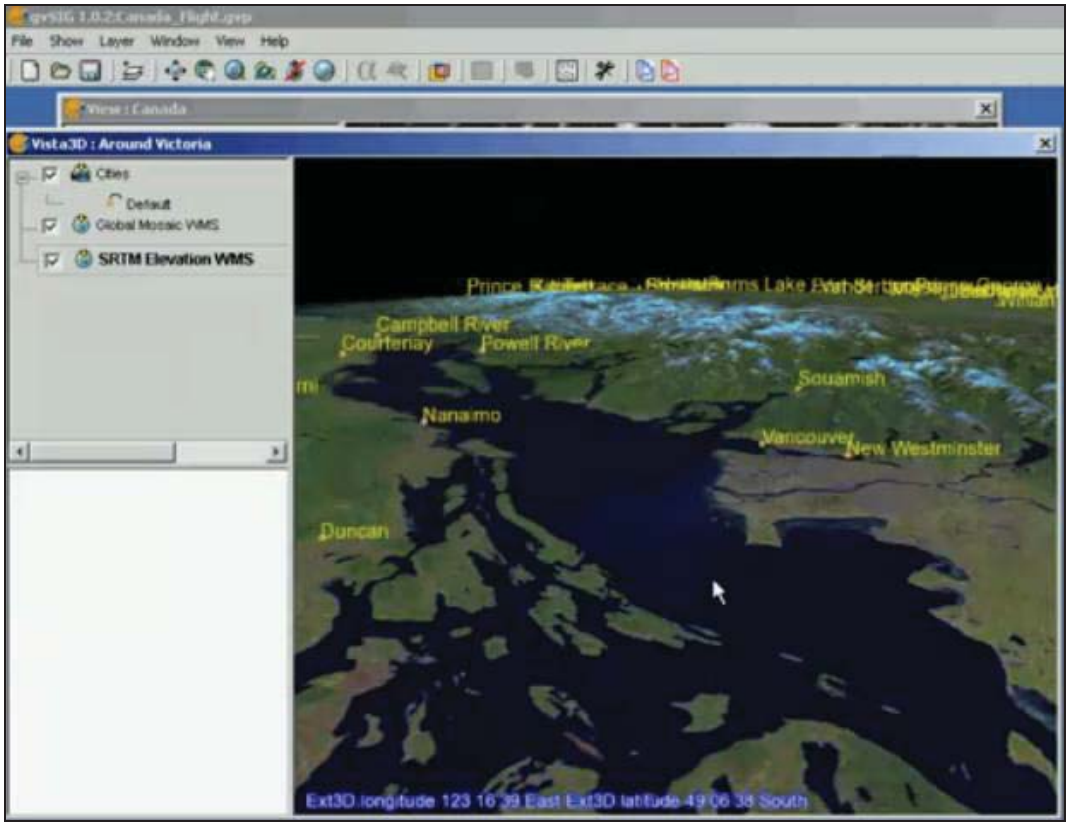


Figura 2.3: gvSIG con extensión gvSIG 3D. [Elaboración Propia]

- **Quantum GIS**

SIG *Open Source* para plataformas Linux, Unix, Mac OS y Microsoft Windows desarrollado en C++ usando la biblioteca Qt para la interfaz gráfica de usuario. Permite manejar formatos *Raster* y vectoriales, así como bases de datos, para más información ver [OSGeo, 10]. Algunas de sus características son:

- Soporte para la base de datos espacial PostGIS.
- Manejo de archivos Vectoriales y *Raster*.
- Un entorno simple y funcional.
- Soporte para la integración con GRASS.
- Proyección de mapas automatizada.
- Soporte para *Plugins*

Una de las mayores ventajas de Quantum GIS es la posibilidad de usarlo como una librería en otras aplicaciones que ofrece todas las funcionalidades que se pueden apreciar en la aplicación Quantum GIS. Los componentes que ofrece Quantum GIS se pueden observar en la Figura 2.4:

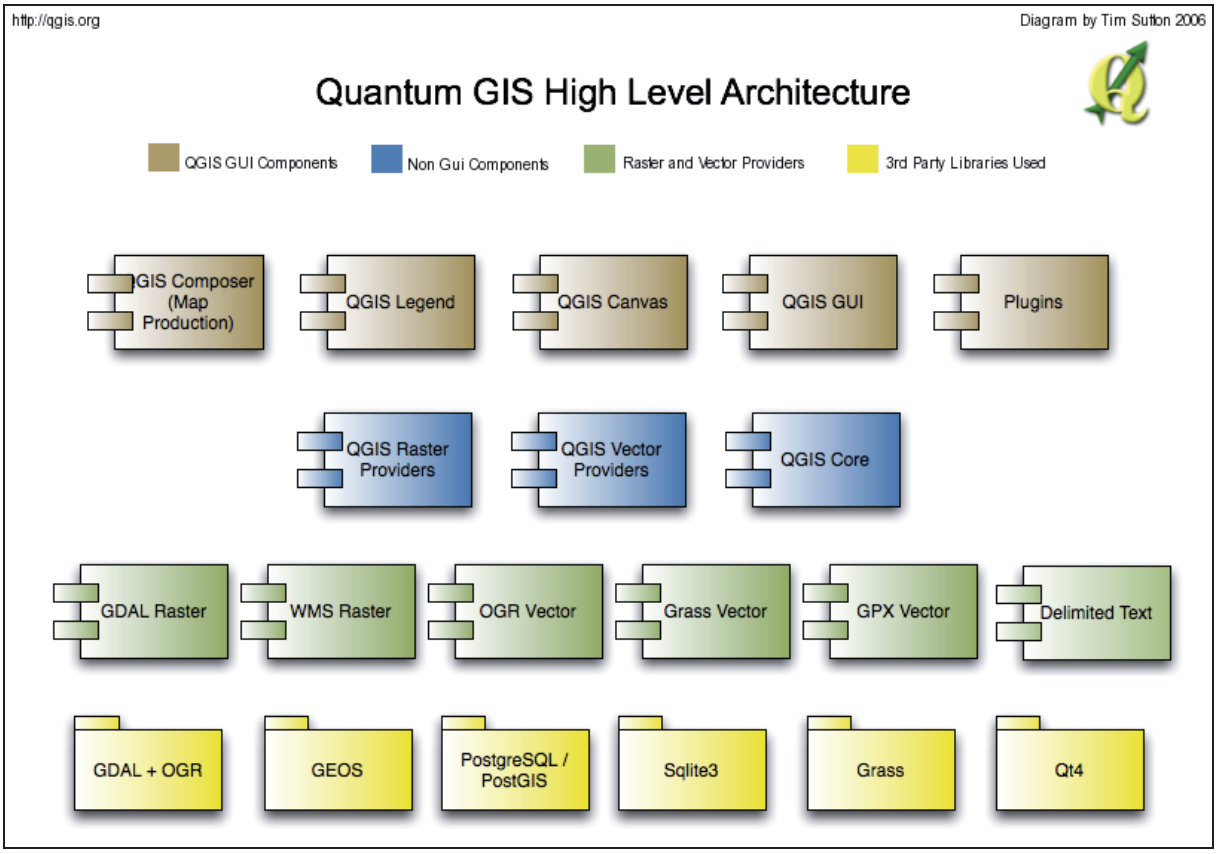


Figura 2.4: Diagrama de componentes de QGIS. [OSGeo, 10]

Las principales librerías que son útiles para el proyecto y que la librería de Quantum GIS utiliza son:

- **GDAL**: Soporte para la carga de archivos de mapas *Raster*.
- **OGR**: Soporte para la carga de archivos de mapas Vectoriales.
- **QT**: Librería orientada a interfaces de usuario, siendo de gran utilidad el manejo del *Canvas*, para dibujar los distintos mapas.
- Otras librerías que utiliza, que no son tan relevantes para el proyecto son:
- **PostGis/PostgreSQL y Sqlite3**: Soporte para bases de datos y consultas.
- **GEOS**: Manipulación de geometría.

A continuación se ve en la Figura 2.5 el Quantum GIS funcionando:

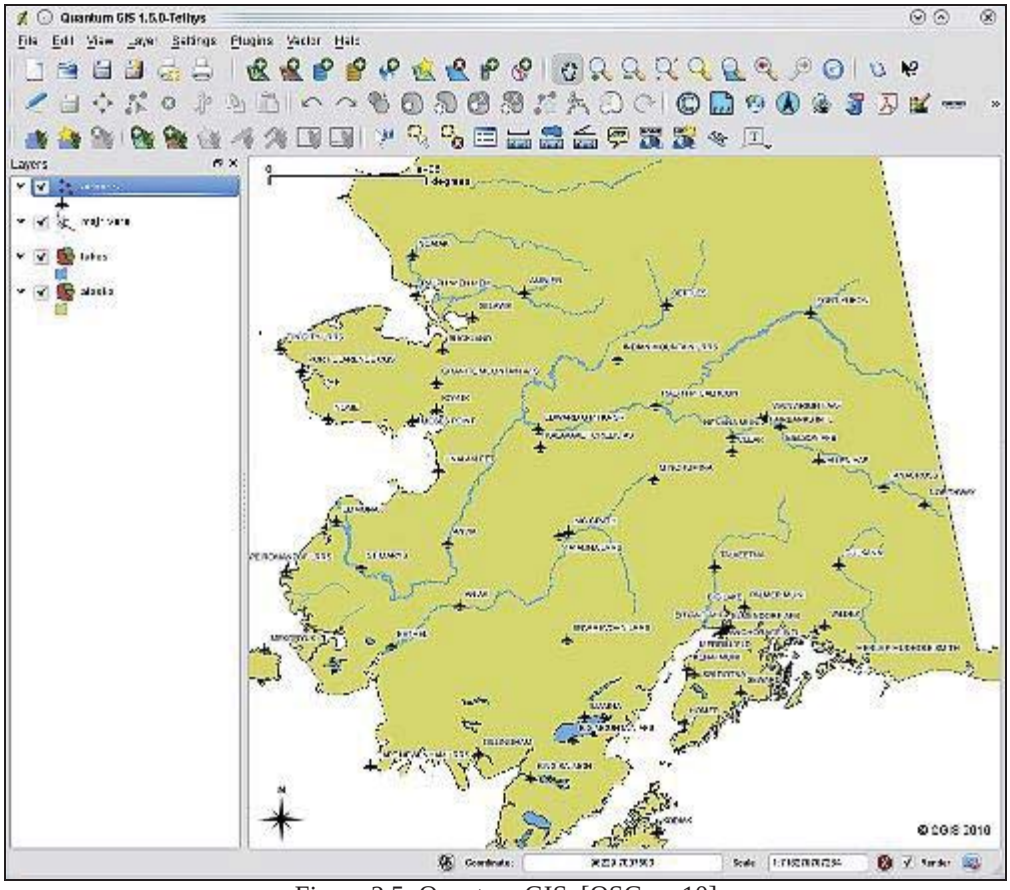


Figura 2.5: Quantum GIS. [OSGeo, 10]

- **Imagine VirtualGIS**

Es un módulo de la aplicación comercial ERDAS Imagine creada por ERDAS Inc, la cual es una poderosa herramienta de análisis, que ofrece funciones de los SIG en un entorno 3D. Lejos de los renderizados 3D simples, VirtualGIS permite crear interpretaciones 3D de los proyectos para presentaciones interactivas. Un punto relevante de esta herramienta es que permite la simulación de entornos de vuelo para los pilotos en tiempo real, para más información ver [ERDAS, 11] y [Pilouk, 07]. Esto es apreciado en la Figura 2.6:

- **GeoMedia**

La suite de GeoMedia es un conjunto de aplicaciones comerciales creadas por Integraph que proveen un gran rango de habilidades de procesamiento geoespaciales necesitadas por las industrias, como gobiernos y agencias de transporte para mapas de producción, manejo de infraestructura, y administración de las tierras.

La aplicación Geomedia Terrain de la suite resalta por ser la encargada del análisis de terreno y generación de modelos de terrenos tridimensionales, permitiendo volar dinámicamente a través de estos modelos, para más información ver [Intergraph, 11] y [Pilouk, 07].

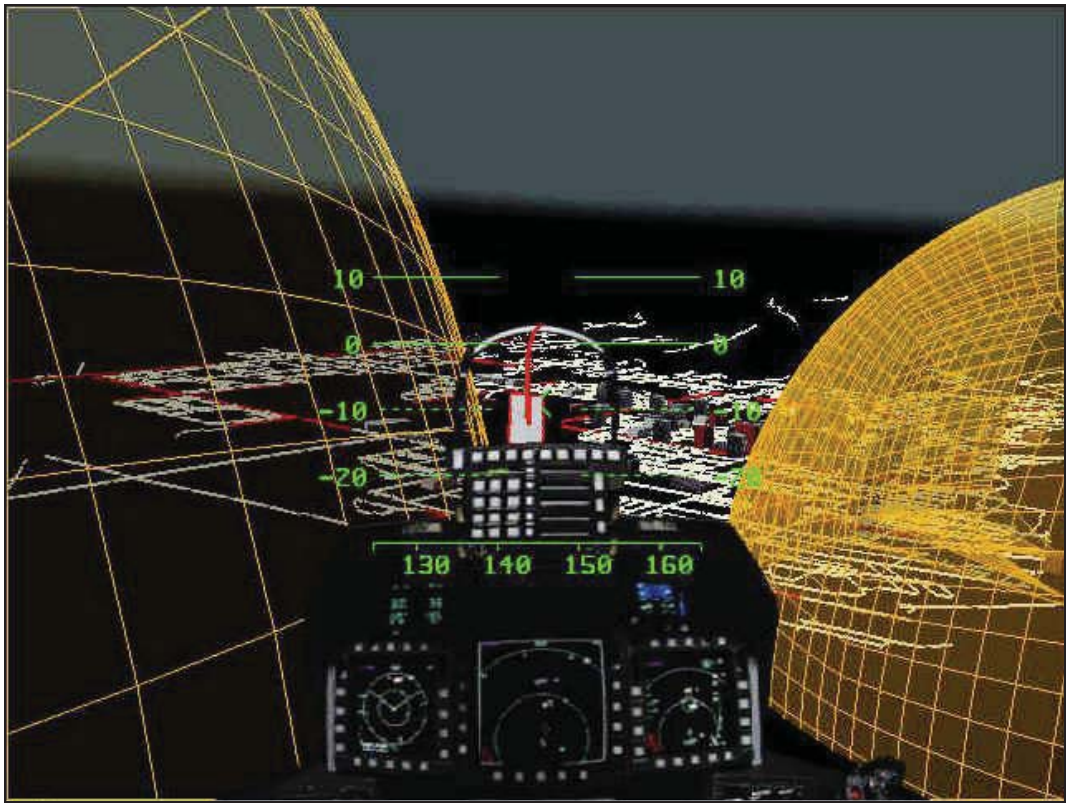


Figura 2.6: Simulador de vuelo de Imagine VirtualGIS. [ERDAS, 11]

- **Open 3D GIS**

Proyecto *Open Source* creado por OpenGEO que consiste en habilitar una base de datos geográfica para ser vista en una visualización 3D en la Web, usando un navegador. Lamentablemente el proyecto se ha dejado de actualizar hace bastantes años, así que solo seria de utilidad su código fuente como referencia, para más información ver [Helton et al., 06].

2.1.2 Almacenamiento de Mapas

En esta sección se describen las distintas formas de almacenar y obtener un mapa digital.

- **Bases de Datos Espaciales**

Una base de datos espacial está optimizada para almacenar y realizar consultas sobre datos y objetos en el espacio incluyendo puntos, líneas y polígonos. Las bases de datos pueden manejar una gran variedad de datos alfanuméricos, pero se requiere de funcionalidades adicionales para que estas puedan manejar tipos de datos espaciales, un tipo de dato espacial básico que es generalmente implementado es *geometry* el cual almacena datos geométricos del mapa como puntos, líneas o polígonos, mientras *geometrycollection* es un tipo de data que almacena colecciones de valores del tipo *geometry*.

Los sistemas de bases de datos tradicionales utilizan índices para buscar rápidamente los valores requeridos, sin embargo este sistema de datos indexados no es el más adecuado para la realización de consultas espaciales. Por esta razón, las bases de datos espaciales utilizan un índice espacial para acelerar las operaciones sobre la base de datos.

La mayoría de los SIG traen una base de datos espacial integrada, aun así existen extensiones para darle a las bases de datos tradicionales algunas funcionalidades para el manejo y operación de datos espaciales. Algunas de estas bases de datos espaciales más conocidas son las que se muestran a continuación:

- **PostGIS:** que es un módulo *Open Source* que agrega soporte para objetos geográficos a PostGreSQL, soporta datos de mapas Vectoriales y tiene un soporte muy básico para datos de mapas *Raster*, para más información ver [PostGIS, 11].

- **Oracle Spatial:** solución pagada de Oracle para una base de datos de SIG, la cual soporta un amplio rango de aplicaciones, para más información ver [Oracle, 11].

- **Microsoft SQL Server:** base de datos pagada de Microsoft que en su versión 2008 soporta bases de datos espaciales, para más información ver [Microsoft, 11].

- **Archivos de Mapas**

El almacenamiento de mapas digitales se puede hacer por medio de archivos, los cuales pueden ser leídos y procesados por distintos programas. Estos archivos almacenan dos tipos de mapas digitales: Mapas *Raster* y Mapas Vectoriales. Existen varias librerías para la carga y edición de mapas *Raster* y Vectoriales entre las cuales tenemos:

- **GDAL:** Librería que tiene el objetivo de cargar y editar mapas *Raster* liberada bajo la licencia X/MIT por la Fundación del Código Libre Geoespacial (*Open Source Geospatial Foundation)*. Respecto a las características de la librería, presenta un solo modelo abstracto de datos para cargar todos los *drivers* de los formatos soportados. Los *drivers* mencionados anteriormente son los distintos códigos para dar soporte a los distintos archivos de mapas *Raster*, para más información ver [GDAL, 11]. Además GDAL viene con otra librería que da soporte a mapas Vectoriales llamada OGR.

La forma básica de cargar un mapa *Raster* y obtener sus datos gráficos con GDAL en pseudocódigo es:

```
01     GDALRegistrarTodosLosDrivers();
02     GDALDataset dataset = GDALAbrir("C:\mapa.tif");
03     for(NumeroBanda   =0;   NumeroBanda   <   dataset->CantidadDeBandas;
NumeroBanda++)
04     {
05            GDALBand banda = dataset->ObtenerBandaNumero(NumeroBanda);
06     VectorFlotante almacenadorDeArea = banda->ObtenerAreaPixeles(x = 0, y =
0,alto = 128, ancho = 128);
07     //el almacenadorDeArea tiene los pixeles de esa área de la banda para poder
dibujarlos en pantalla
08     }
```

```
09      GDALCerrarDataset(dataset);
```

- En el código anterior se ve la facilidad con que GDAL carga los distintos formatos de mapas *Raster*. A continuación se listan algunos formatos soportados por GDAL:

    - **GeoTIFF**: es un estándar de metadato de dominio público que permite que la información georreferenciada sea incrustada en un archivo de imagen en formato TIFF (.tif).

    - **SRTM HGT Format**: formato usado por la SRTM (*Shuttle Radar Topography Mission*), que es una misión de la nasa para obtener mapas DEM de todo el planeta. El formato es SRTMHGT (.hgt).

    - **ADRG/ARC Digitilized Raster Graphics**: formato soportado por la NIMA (*National Imagery and Mapping Agency*) para mostrar información subyacente de los mapas. El formato es ADRG (.gen/.thf).

    - **Arc/Info ASCII Grid**: formato de archivo creado por la ESRI (*Environmental Systems Research Institute*) el cual es usado en los programas creados por la misma.

  La especificación de los formatos de archivo soportados por GDAL es extremadamente extensa, por esto para acceder a la lista completa de formatos se puede ingresar al sitio de GDAL [GDAL, 11] indicado en la sección de Referencias.

- **OGR:** es una librería que viene con GDAL hecha de la misma forma, solo que soporta archivos de mapas Vectoriales, presenta el mismo modelo abstracto de datos para cargar los *drivers* de los formatos de mapas soportados, pero a la hora de cargar y leer los mapas es diferente, ya que la estructura de datos Vectoriales es diferente y bastante más complicada que la de los archivos *Raster*, para más información ver [GDAL, 11].

La forma básica de cargar un mapa Vectorial y obtener sus datos gráficos con OGR en pseudocódigo es:

```
01      OGRRegistrarTodosLosDrivers();
02      OGRDataSource dataSource = OGRAbrir("C:\mapas\mapa.shp");
03      for(NumeroLayer=0;NumeroLayer<dataSource->CantidadDeLayers;
NumeroLayer++)
04      {
05              OGRLayer capa = dataSource->ObtenerLayer(NumeroLayer);
06              for(NumeroFeature=0;NumeroFeature<capa->CantidadDeFeatures;
                NumeroFeature++)
07              {
```

```
08                    OGRFeature caracteristica =
                              capa->ObtenerFeature(NumeroFeature);
09                    OGRFeatureDef definicionDeCaracteristicas =
                      capa->ObtenerFeatureDef();
10      // definicionDeCaracteristicas tiene todos los datos informativos de la
caracteristica
11                    OGRGeometry geometria =
                      caracteristica->ObtenerGeometria();
12      //Son los datos geométricos de la capa (puntos, líneas, polígonos, etc)
13             }
14      }
15      OGRCerrarDataSource(dataSource);
```

Como se ve en el código anterior es bastante más complicada y lenta la forma de leer un mapa de formato Vectorial, ya que contiene varios Arreglos de datos y listas con sublistas. OGR soporta los siguientes formatos:

- **ESRI Shapefile**: formato de archivo creado por la ESRI (*Environmental Systems Research Institute*) el cual es usado en los programas creados por la misma y se ha convertido en uno de los formatos vectoriales más utilizado. El formato es ESRI Shapefile (.shp).

- **Arc/Info .E00 (ASCII) Coverage**: formato de archivo usado en los programas creado por la ESRI. El formato es AVCE00 (.e00).

- **GML**: GML o *Geography Markup Language* es un tipo de archivo XML definido por la OGC (*Open Geospatial Consortium*) para almacenar información georeferenciada. El formato es GML (.gml).

- **U.S. Census TIGER/Line**: TIGER (*Topologically Integrated Geographic Encoding and Referencing)* es un formato usado por la Oficina de Censo de los Estados Unidos para describir atributos del terreno como caminos, construcciones, ríos y lagos.

La especificación de los formatos de archivo soportados por OGR es extremadamente extensa, por esto para acceder a la lista completa de formatos se puede ingresar al sitio de OGR [GDAL, 11] indicado en la sección de Referencias.

2.2 Mapas digitales

Los mapas digitales pueden estar basados en dos modelos diferentes de datos, cada uno de los cuales ofrece ventajas para el almacenamiento de datos específicos. Pero, por más datos que puedan ofrecer estos mapas, de nada sirven si no se puede saber la ubicación de estos sobre el globo terrestre, por esto al trabajar con estos mapas se deben especificar la referencia

geodésica y la proyección geográfica adecuadas para realizar la correcta relación entre las coordenadas de los datos y su localización en la superficie terrestre.

2.2.1 Modelo de Datos Raster

El área completa del mapa es subdividida en una rejilla de pequeñas celdas. Un valor es guardado en cada una de estas celdas, representando la naturaleza de cualquier objeto presente en la ubicación correspondiente sobre el suelo. Los Datos *Raster* pueden ser considerados como una matriz de valores.

Gráficamente los datos *Raster* guardan el color de la celda que representa esa parte de la superficie del mapa. Los valores almacenados en las celdas de la matriz son blancos, azules o rojos. Para reproducir la imagen el computador lee cada uno de los valores de la matriz uno por uno y los convierte a píxeles en pantalla, así como se aprecia en la Figura 2.7.

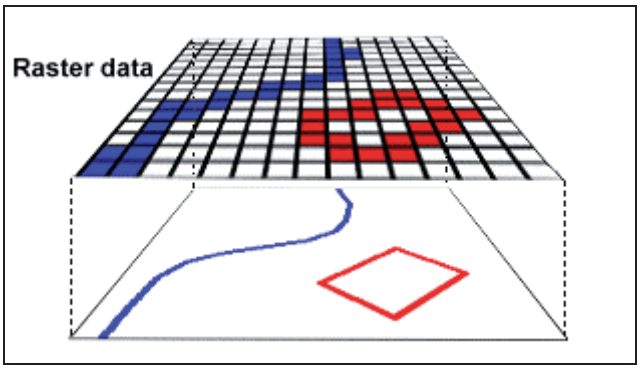


Figura 2.7: Representación de un mapa en datos Raster. [Hyam, 02]

El uso principal de los datos *Raster* es almacenar la información de los mapas como imágenes digitales, en el cual los valores de cada celda están relacionados con los colores de los píxeles de la imagen. En estos modelos, para representar puntos, líneas, aéreas y superficies, estas deben ser transformadas, para que la matriz que es usada en este modelo pueda representar estos datos. Estos se conceptualizan de la siguiente forma para cada una de las formas básicas:

- **Puntos:** En los modelos conceptuales de geográficos, los puntos no tienen dimensiones. En los SIG *Raster* un punto es representado como una celda de la matriz, con un color representativo para una categoría. Por ejemplo cada celda con el valor 3 puede representar una iglesia. Esto se puede apreciar más fácilmente en la Figura 2.8:

- **Líneas:** en los SIG *Raster*, una serie de celdas de la matriz representan un objeto lineal, como por ejemplo un camino. Al igual que con los puntos, se usan colores o patrones para representar cada categoría de la línea. Se ve gráficamente en la Figura 2.9:

- **Áreas (polígonos):** Se representan en los SIG *Raster* como un grupo de celdas, ya sea continuas o discontinuas, que represente lo más fielmente el área del modelo conceptual geográfico. Apreciado visualmente en la Figura 2.10:

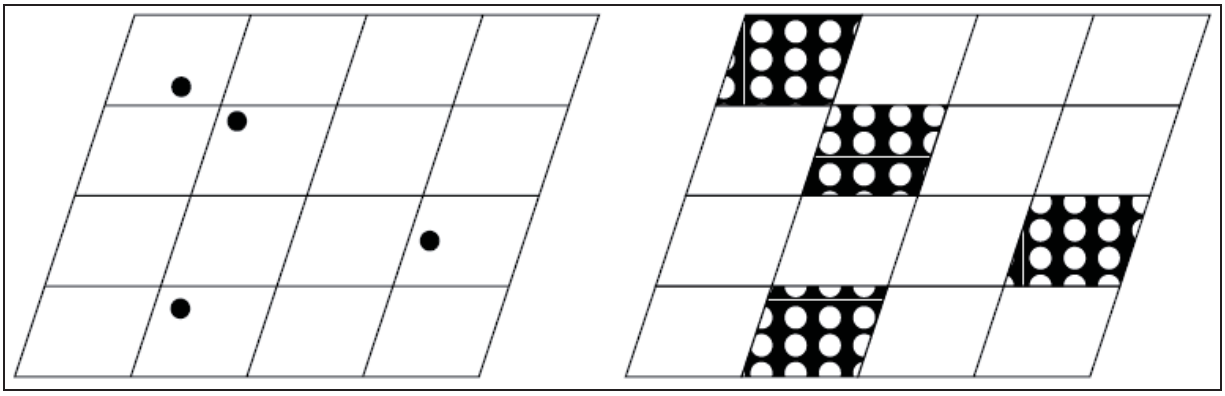


Figura 2.8: Comparación para puntos entre el modelo conceptual geográfico y el modelo Raster. [DeMers, 09]

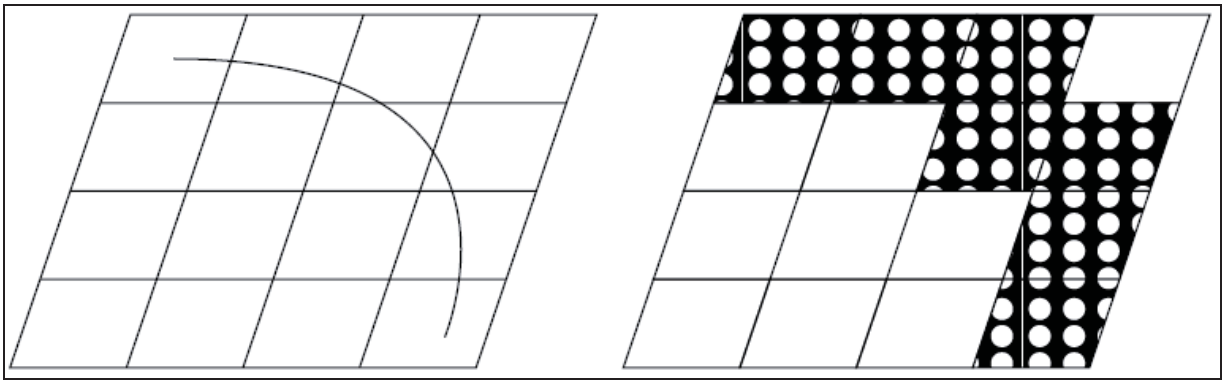


Figura 2.9: Comparación para líneas entre el modelo conceptual geográfico y el modelo Raster. [DeMers, 09]

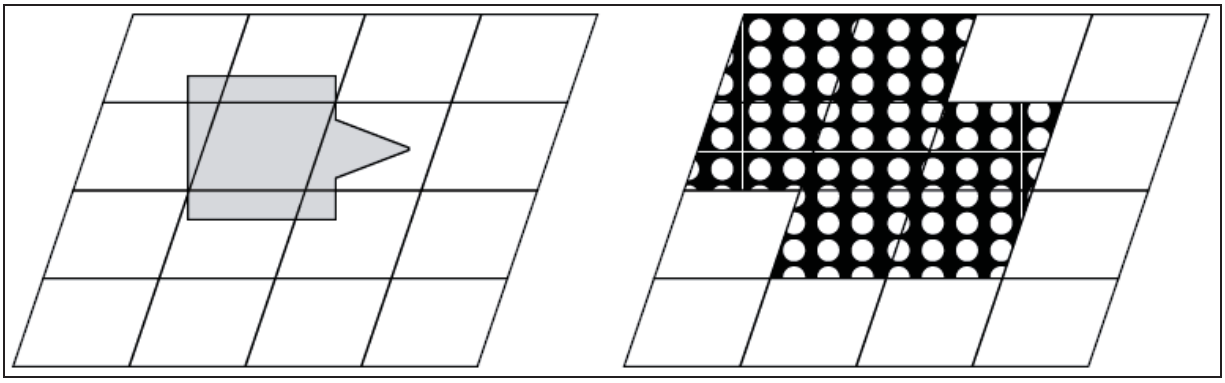


Figura 2.10: Comparación para áreas entre el modelo conceptual geográfico y el modelo Raster. [DeMers, 09]

- **Superficies (volúmenes):** En los SIG *Raster* cuando se representan superficies o sus volúmenes, cada celda tiene, además del largo y ancho, un número asociado con la altura o profundidad del espacio. Este número puede representar elevación sobre el nivel del mar o profundidad bajo el nivel del mar, con algún color representativo. En algunos SIG este número representa superficies no físicas, como densidad de población. Lo que aquí se expone se puede ver en forma gráfica en la Figura 2.11:

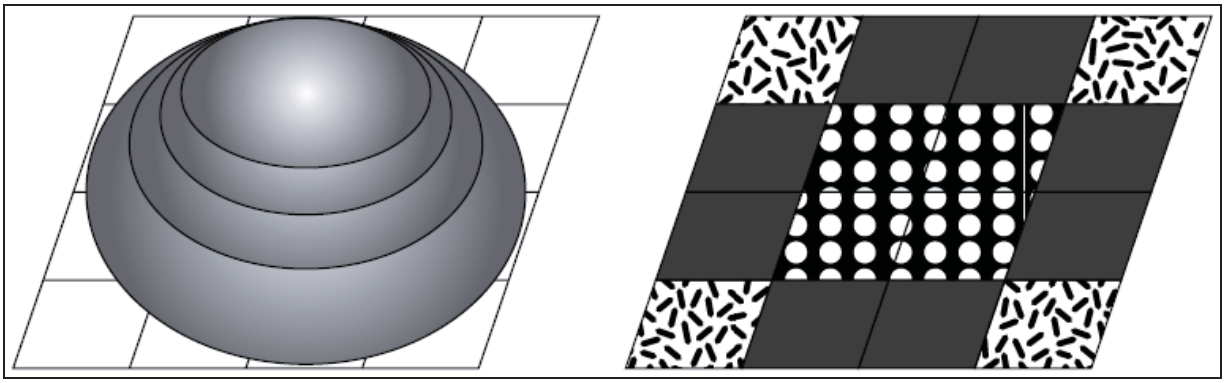


Figura 2.11: Comparación para superficies entre el modelo conceptual geográfico el modelo Raster. [DeMers, 09]

Esta forma de representar la altura en base a un valor extra en una matriz, recuerda mucho a los mapas de altura (*height maps*) de las aplicaciones 3D, basados en una imagen en escala de grises, donde el negro representa la parte más baja y el blanco la parte más alta. Estos tipos de mapas *Raster* son conocidos como DEM (*Digital Elevation Model*) y se puede apreciar la porción de uno en la Figura 2.12.

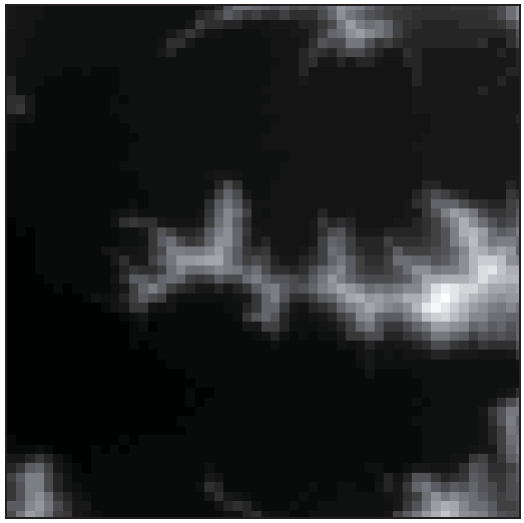


Figura 2.12: Extracto de Mapa de altura Raster. [Elaboración Propia]

Otro punto es que los Mapas *Raster* se dividen en Bandas que representan distintos tipos de información, la cual generalmente se le atribuye al color. Por ejemplo un mapa en escala de grises tiene una sola Banda, un mapa en colores tiene tres Bandas y puede haber mapas con más bandas para otros objetivos como representación visual de la temperatura en un mapa. El ejemplo de un mapa de colores gráficamente se ve en la Figura 2.13 que es una porción de un Mapa *Raster* de colores con sus tres Bandas que representan cada color primario (rojo, verde y azul), las cuales están dibujadas individualmente, para más información ver [DeMers, 09]. Al juntar las tres Bandas en una sola se forma un mapa con colores como se puede apreciar en la Figura 2.14.

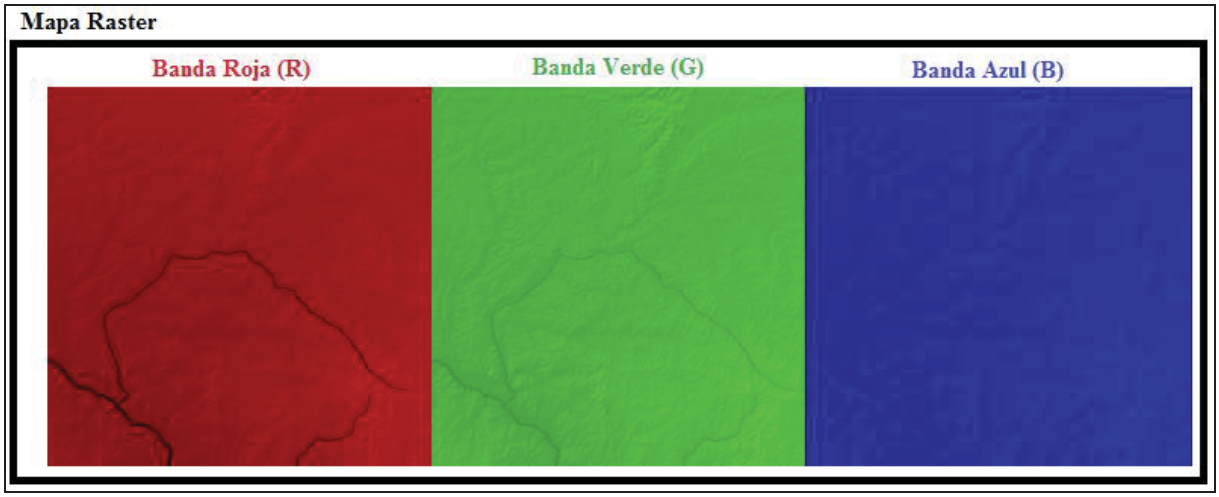


Figura 2.13: Mapa Raster RGB con las Bandas separadas. [Elaboración Propia]

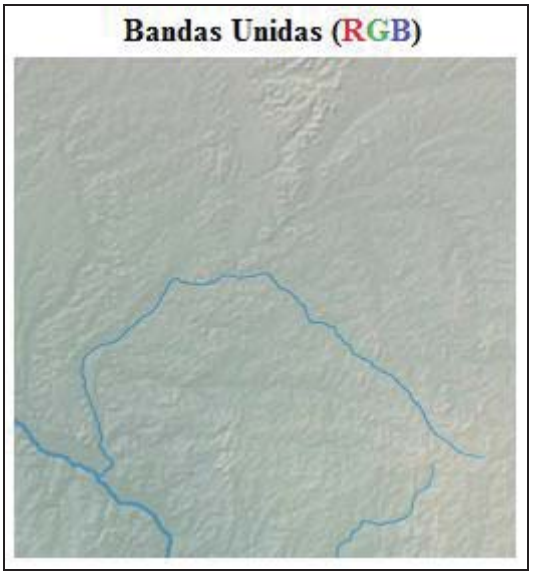


Figura 2.14: Mapa Raster RGB con las Bandas juntas. [Elaboración Propia]

2.2.2 Modelo de Datos Vectorial

Esta estructura de datos representa cada punto por un par de coordenadas X e Y, cada línea por un conjunto de 2 o más pares de coordenadas X e Y, y cada área es representada por al menos 3 líneas que crean una figura cerrada. Donde el conjunto de estas forman la representación abstracta del mapa.

En la Figura 2.15 el mapa representa una construcción, como un rectángulo rojo. En datos vectoriales la posición y la forma de la construcción es capturada como una serie de 4 pares de coordenadas numéricas. Para reproducir la construcción en un SIG el computador lee estos valores y dibuja una línea uniendo las posiciones de las coordenadas.

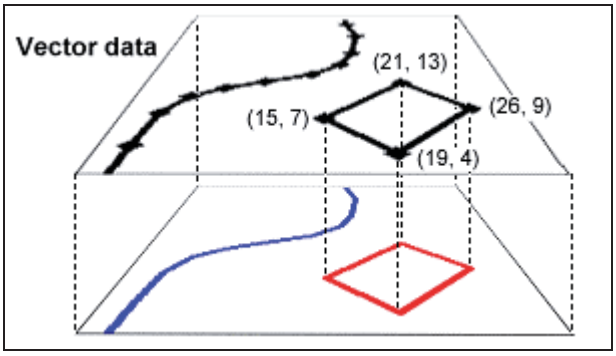


Figura 2.15: Representación de un mapa en datos vectoriales. [Hyam, 02]

En los modelos vectoriales, representar puntos, líneas, y áreas es bastante sencillo. La estructura de datos de los vectores representa cada punto por un solo par de coordenadas X e Y, cada línea por un conjunto de dos o más pares de coordenadas X e Y, y cada área por al menos tres líneas que crean una figura cerrada por su coordenada X e Y inicial, que es a la vez su coordenada final. Se puede apreciar el extracto de un mapa usando este modelo en la Figura 2.16.

Representar superficies o volúmenes usando datos vectoriales es un poco más difícil que simbolizar puntos, líneas, y áreas, pero la industria del SIG ha establecido una estructura de datos usada comúnmente. Esta estructura de datos, conocida como TIN (*Triangulated Irregular Network*), representa superficies y volúmenes como un conjunto de triángulos sin sobreponer, que es muy similar a la forma en que los juegos computacionales modelan los objetos en 3D. Los vértices (puntos) de cada triángulo tienen valores X, Y y Z (altura) únicos. Pero cada grupo de 3 vértices crean un plano triangular donde su superficie tiene un ángulo y dirección únicos.

La ventaja del modelo TIN es que puede ser utilizado para predecir (interpolar) valores faltantes, dividir superficies y volúmenes, dibujar el contorno de las líneas y lo más importante crear visualizaciones 3D.

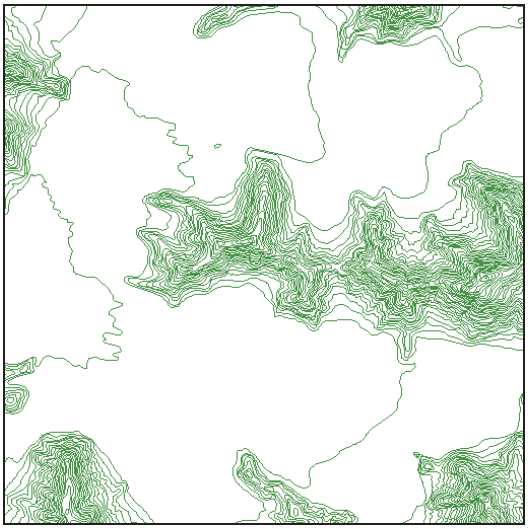


Figura 2.16: Extracto de Mapa de altura Vectorial. [Elaboración Propia]

Por otra parte los Mapas Vectoriales se dividen en varias Capas, las cuales tienen varias Características y estas a su vez tienen Definiciones de estas Características y datos Geométricos como puntos, líneas, polígonos, etc. Esto se aprecia más gráficamente en la Figura 2.17:

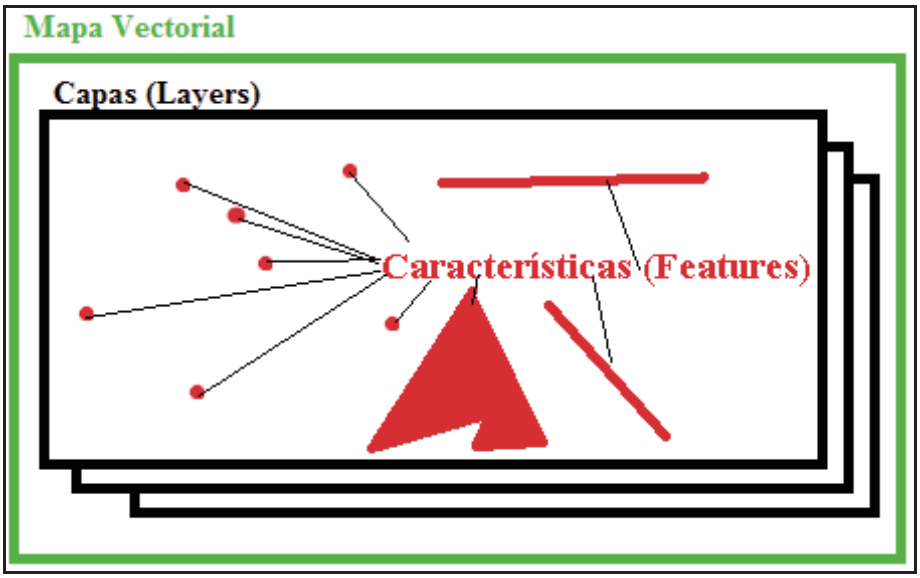


Figura 2.17: Contenido de un Mapa Vectorial. [Elaboración Propia]

Por ejemplo se tiene Mapa Vectorial de densidad de población representando en color verde, que tiene varias capas representadas en color negro y estas tienen varias Características representadas en color rojo, cada Característica tiene Definiciones de Característica que son datos como por ejemplo la densidad de población, índice de crecimiento, índice de mortalidad, nombre del área de donde se obtuvieron los datos y datos Geométricos que pueden ser puntos, líneas compuestas de varios puntos, Polígonos compuestos de varias líneas y puntos o algún otro, para más información ver [DeMers, 09]. En la Tabla 2.1 se pueden apreciar los datos ofrecidos por las Características relatadas en este ejemplo.

Tabla 2.1: Ejemplo de Definiciones de Características y Geometría de Características

Definiciones de Característica Numero 6			
# Definición	Nombre	Valor	Tipo
0	Densidad de Población	152	int
1	Tasa de crecimiento	6,42	float
2	Tasa Mortalidad	12,5	float
3	Nombre del área	Las Palmas	String
Geometría de Característica Numero 6			
Tipo	Datos		
Punto	(1,3)		

Definiciones de Característica Numero 9			
# Definición	Nombre	Valor	Tipo
0	Densidad de Población	1282	int
1	Tasa de crecimiento	8,74	float
2	Tasa Mortalidad	5,7	float
3	Nombre del área	La Dehensa	String
Geometría de Característica Numero 9			
Tipo	Datos		
Línea	((5,8),(7,9)		
Definiciones de Característica Numero 14			
# Definición	Nombre	Valor	Tipo
0	Densidad de Población	243958	int
1	Tasa de crecimiento	16,42	float
2	Tasa Mortalidad	2,3	float
3	Nombre del área	El Area P.	String
Geometría de Característica Numero 14			
Tipo	Datos		
Polígono	((10,6),(13,8),(7,8))		

2.2.3 Interpolación Local Mediante TIN

Como se explicaba brevemente en la sección anterior, las Redes Irregulares de Triángulos (TIN) se generan a partir de valores puntuales tratando de conseguir triángulos que maximicen la relación área/perímetro, el conjunto de todos los triángulos forma un objeto geométrico denominado conjunto convexo. Suelen utilizarse como método para representar modelos de elevaciones produciendo resultados visualmente buenos, sin embargo a la hora de integrarlos con el resto de la información Raster es necesario interpolar una capa Raster a partir de los triángulos como se ve en la Figura 2.18: Interpolación Mediante TIN. [Elaboración Propia].

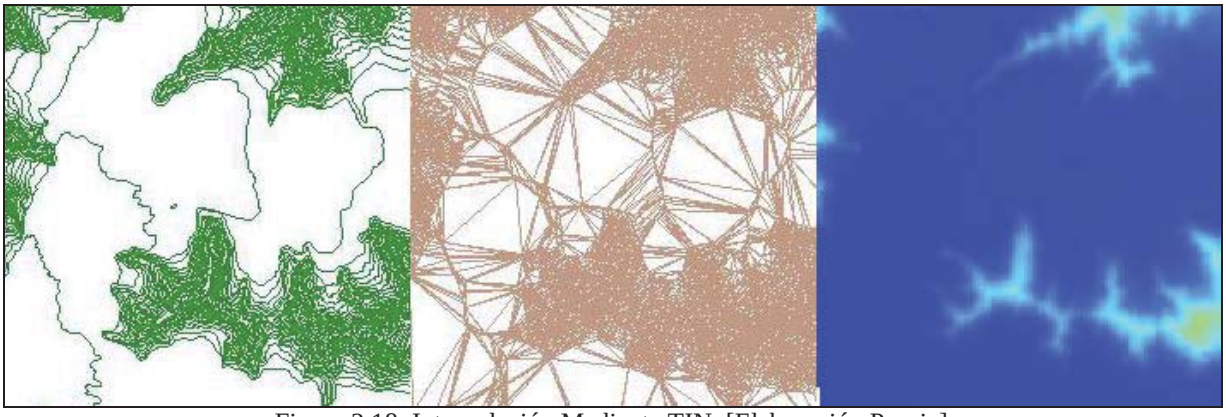


Figura 2.18: Interpolación Mediante TIN. [Elaboración Propia]

Esta interpolación se basa en que cada uno de los tres vértices de los triángulos tienen unos valores X, Y y Z a partir de los cuales puede obtenerse un modelo de regresión Z = AX + BY + C que permite interpolar la variable Z en cualquier punto del rectángulo. En definitiva el resultado siempre va estar acotado por los valores máximo y mínimo de Z en los vértices del triángulo y será más parecido al del vértice más cercano. En el resultado final de una interpolación TIN no aparecen artefactos circulares, como en otros métodos de interpolación como el de inverso de la distancia puro, pero si aparecen artefactos triangulares.

El conjunto convexo así generado tiene otra utilidad, define el área en la que es razonable interpolar dada la muestra de puntos disponible.

2.2.4 Sistema de Coordenadas Proyectadas

Según [Goizueta, 05], mediante una proyección cartográfica se hace corresponder cada punto del plano con un punto del elipsoide. De ésta forma cada punto de la tierra puede tener representación sobre el plano. Las proyecciones se pueden definir de forma matemática como funciones que convierten las coordenadas geográficas (Latitud, Longitud) en coordenadas cartesianas (X, Y) sobre un plano y viceversa.

La limitación de las proyecciones respecto a representar conjuntamente todas las cualidades de la realidad, es proporcional al área representada, puesto que las deformaciones serán mínimas si la superficie abarcada en la proyección es suficientemente pequeña como para excluir la incidencia de la curvatura terrestre. Por el contrario, en áreas extensas, y más aún en la representación de la totalidad del elipsoide, las deformaciones serán mayores productos de la dificultad de traspasar al plano una superficie curva.

La representación plana del elipsoide es una necesidad desde el punto de vista cartográfico y una comodidad bajo el prisma geodésico. La proyección más usada en cartografías es la UTM (*Universal Transversa Mercator)*, estandarización de la TM en que se minimiza el error en áreas limitadas por meridianos (husos) de 6 grados de anchura. Debido a la arbitrariedad de estas divisiones es usual que el área de interés sea cortada por alguna división de huso, tal como se puede apreciar en la Figura 2.19.

La representación UTM divide la Tierra en 60 husos de 6º de longitud, la zona de proyección de la UTM se define entre los paralelos 80º S y 84º N. Cada huso se numera con un número entre el 1 y el 60, estando el primer huso limitado entre las longitudes 180° y 174° W y centrado en el meridiano 177º W. Cada huso tiene asignado un meridiano central, que es donde se sitúa el origen de coordenadas, junto con el ecuador. Los husos se numeran en orden ascendiente hacia el este.

UTM además divide la Tierra en 20 zonas de 8º Grados de Latitud, que se denominan con letras desde la C hasta la X excluyendo las letras “I” y “O”, por su parecido con los números uno y cero, respectivamente. La zona C coincide con el intervalo de latitudes que va desde 80º S (o -80º latitud) hasta 72º S (o -72º latitud). Las zonas polares no están consideradas en este sistema de referencia. Para definir un punto en cualquiera de los polos, se usa el sistema de coordenadas UPS. Si una zona tiene una letra igual o mayor que la N, la zona está en el hemisferio norte, mientras que está en el sur si su letra es menor que la “N”. Por ejemplo, Chile se encuentra entre

los husos 18 y 19 y su zona es menor que “N”, por lo tanto se encuentra entre las zonas 18S y 19S. Las divisiones explicadas anteriormente pueden verse en la Figura 2.20.

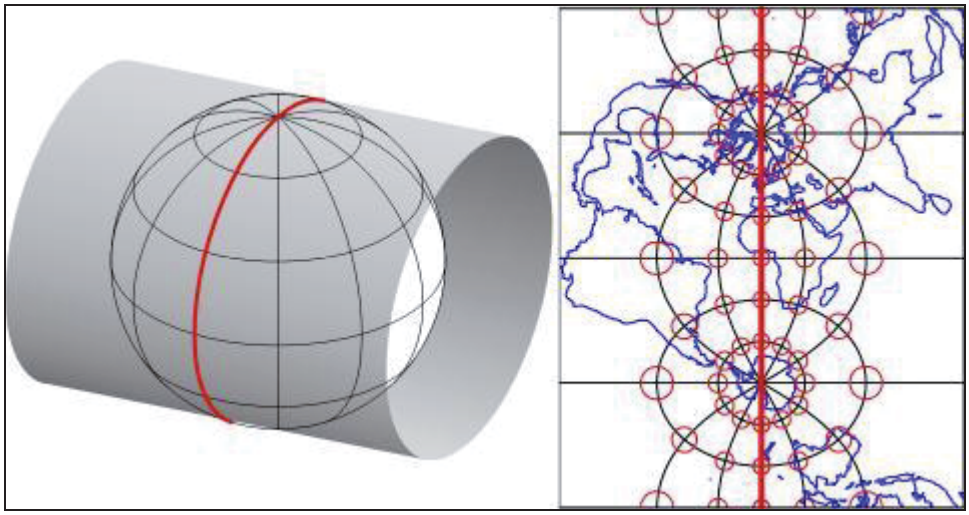


Figura 2.19: Proyección UTM. [PENNSTATE, 10]

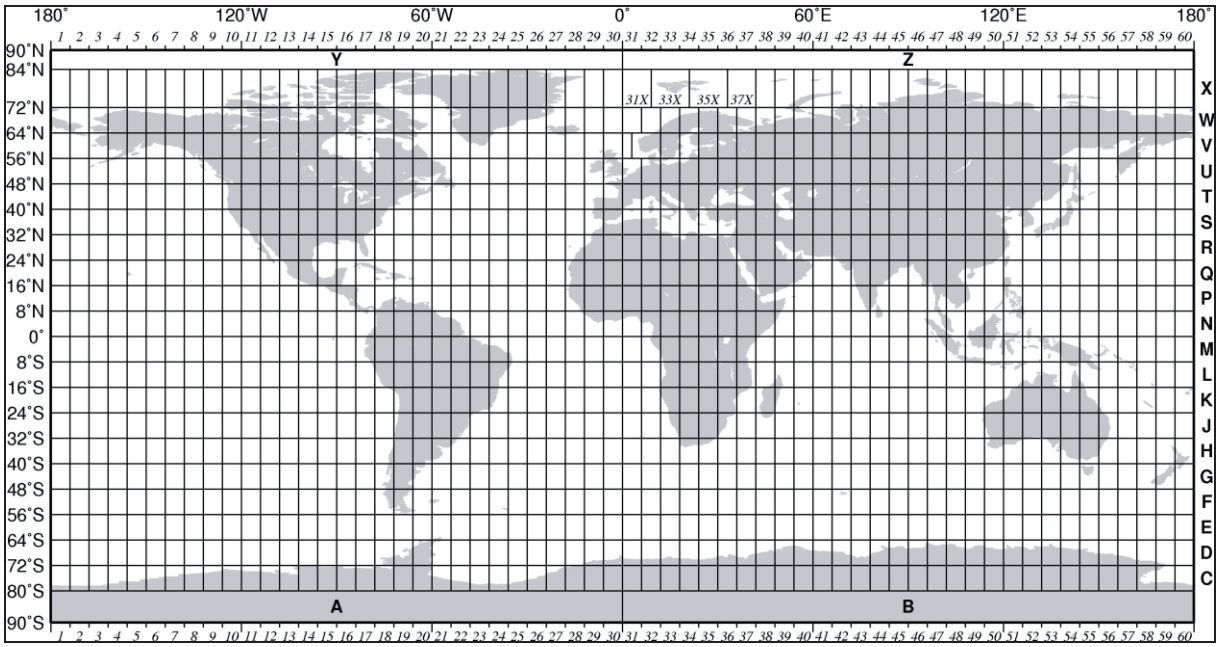


Figura 2.20: Zonas UTM. [SOEST, 10]

2.2.5 Sistemas de Referencia Geodésicos

Los sistemas de referencia geodésicos definen la forma y dimensión de la Tierra, así como el origen y orientación de los sistemas de coordenadas. Estos sistemas pueden ser descritos en base a dos modelos matemáticos: el esférico y el elíptico, los cuales son obtenidos en base parámetros físicos medidos sobre la superficie terrestre, tales como la aceleración de gravedad.

De ser la Tierra perfectamente homogénea, la forma del geoide sería la de un elipsoide de revolución o esferoide, superficie obtenida al hacer girar una elipse alrededor de uno de sus ejes (en este caso el menor). Debido a la heterogeneidad de la Tierra la forma del geoide es irregular, pero no se aparta más de 100 m. de un elipsoide bien escogido para aproximarlo. Los elipsoides más conocidos son:

- **GRS80:** Siglas de *Geodetic Reference System 1980*, es el reemplazo para *Geodetic Reference System 1967* el cual no ofrecía información lo suficientemente precisa sobre el tamaño, forma y campo gravitacional de la tierra. Este es un elipsoide de referencia que provee de constantes específicas como el radio ecuatorial y aplanamiento reciproco entre otros.

- **WGS 84:** Siglas de *World Geodetic System 1984*, este sistema es un estándar en geodesia cartografía y navegación, establecido en 1984 con su última revisión realizada en el año 2004 y es vigente hasta este año.

- **Internacional 1924:** También conocido como el elipsoide de Hayford, adoptado por la *International Union of Geodesy and Geophysics* (IUGG) en 1924, fue recomendado para ser usado en los sistemas de referencia mundiales sin mucho éxito.

Se puede buscar una buena aproximación al geoide en una zona restringida de la tierra o bien de manera global, un *datum* en el que el elipsoide se encuentra tangente al geoide. Por tanto, en éste punto las coordenadas coinciden con las referidas al elipsoide. De ésta forma el elipsoide aproxima al geoide tanto mejor cuanto más cerca del *datum* se encuentre. Es por tanto éste, un método local, válido con precisión únicamente para una zona restringida de la tierra. Actualmente, se están utilizando como referencia elipsoides centrados en el centro de masas de la tierra, consiguiendo una aproximación similar en toda la superficie terrestre.

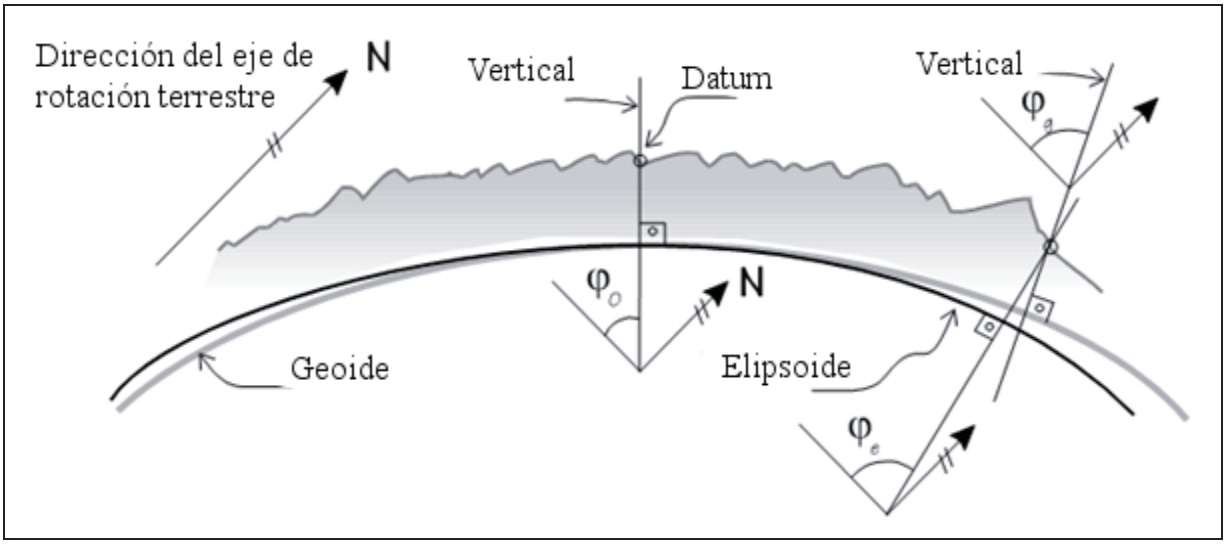


Figura 2.21: Elipsoide de Referencia. [Goizueta, 05]

La utilización de los Sistemas de Referencia Geodésicos adecuados en los mapas digitales es de suma importancia, ya que sin la referencia adecuada los datos de estos mapas pueden

quedar demasiado desplazados de su origen. Entre los Sistemas de Referencia Geodésicos más utilizados en Sudamérica se encuentran los siguientes:

- **WGS84:** Con el mismo nombre que el elipsoide que lo define, se utiliza en conjunto al sistema de coordenadas UTM.

- **PSAD67:** Siglas de *Provisional South American Datum1956*, sistema para la referencia de Chile, Perú, Bolivia, Ecuador y Venezuela, utiliza el elipsoide Internacional 1024 con punto de origen en La Canoa, ciudad de Venezuela.

- **SAD69:** Siglas de *South American Datum1969*, sistema para la referencia, utiliza el elipsoide GRS67 con punto de origen en Chua, ciudad de Brasil.

- **SIRGAS:** Siglas de Sistema de Referencia Geocéntrico para las Américas, utiliza como referencia el elipsoide GRS80, se utiliza conjuntamente con UTM para hacer corresponder la zona geográfica.

2.3 Gráficos Computacionales 3D

Son gráficos que usan una representación en tres dimensiones de datos geométricos que son guardados en el computador, con el propósito de hacer cálculos y crear una imagen en dos dimensiones para ser mostrada a través de un proceso llamado Renderizado, o ser usadas en aplicaciones no gráficas como simulaciones y cálculos. Para esto usan modelos 3D que son las representaciones matemáticas de cualquier objeto en 3D.

2.3.1 APIs 3D

Aquí se presentan las dos APIs para gráficos 3D más importantes, las definiciones siguientes fueron extraídas de [Sanchez et al, 00] y [Wright et al, 04].

- **Direct3D**: Es un componente de la API DirectX de Microsoft para Windows, encargado de mostrar gráficos en tres dimensiones en aplicaciones donde el desempeño es importante, como los juegos. Usando aceleración por hardware, si la tarjeta gráfica lo soporta. El objetivo de Direct3D es lograr una abstracción en la comunicación entre una aplicación gráfica y los *drivers* del hardware gráfico. Direct3D soporta distintas tecnologías del hardware como:

  - Z-buffering.
  - Anti-aliasing.
  - Alpha blending.
  - Mipmapping.
  - Efectos atmosféricos.
  - Perspective-correct texture mapping.

- **OpenGL**: Es una librería gráfica escrita originalmente en C que permita la manipulación de gráficos 3D. Esta librería se concibió para programar en computadores nativos de Silicon Graphics bajo el nombre de GL (*Graphics Library*). Posteriormente se consideró la

posibilidad de extenderla a cualquier tipo de plataforma para asegurar así la portabilidad y extensibilidad de uso, por lo que se llegó al termino OpenGL (*Open Graphics Library*). Gracias a esta portabilidad existen implementaciones eficientes de OpenGL para Mac OS, Microsoft Windows, Linux, Unix y la mayoría de las consolas de videojuegos que no son propiedad de Microsoft.

La librería se ejecuta independientemente de la capacidad gráfica del computador utilizado. Esto significa que la ejecución se realizará por software, a no ser que se cuente con hardware gráfico especializado, como tarjetas aceleradoras de video, aceleradoras 3D o pipelines gráficos integrados entre otros, lo que produce una ejecución en tiempo real más rápida.

2.3.2 Frameworks con Soporte para Gráficos 3D

La mayoría de las *frameworks* facilita el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel, como lo es la programación de gráficos 3D directamente utilizando direct3D u OpenGL.

Para realizar una correcta validación del producto final de este proyecto se debió desarrollar un módulo para visualizar los escenarios 3D generados, para esto se consideró el uso de los siguientes *frameworks*, para programar dicha herramienta:

- **XNA:** Es un conjunto de herramientas provisto por Microsoft que facilita la administración y el desarrollo de juegos computacionales para Microsoft Windows y Microsoft XBOX 360. XNA está basado en DirectX, por lo tanto utiliza Direct3D para el manejo de gráficos 3D, el *framework* escrita en C# requiere de la IDE Visual Studio de Microsoft para ser utilizada. El conjunto de herramientas de XNA son:

  XNA Game Studio: es un conjunto de *plugins* para Visual Studio, hechas para el desarrollo de juegos.
  XNA Framework: librerías escritas en C# que son una abstracción de más alto nivel de DirectX, que son usadas por XNA Game Studio.
  XNA es gratis para la creación de juegos en Windows, en cambio para la creación de juegos de XBOX 360 es necesario comprar una licencia comercial, más información disponible en [Reed, 08].

- **DarkGDK:** Es un *framework* creada por "*The Game Creators*" escrita en C++, que usa DirectX, la cual maneja los gráficos, sonido, entrada y salida en un alto nivel para el desarrollo de juegos para Microsoft Windows. DarkGDK soporta la programación en C++ y en un lenguaje creado por "*The Game Creators*" llamado DarkBASIC y requiere el uso de Visual Studio, para mayor información consultar [DarkMod, 10].

- **OGRE3D:** OGRE (*Object-Oriented Graphics Rendering Engine*) es un motor de gráficos 3D *Open Source* escrito en C++ diseñado para facilitar y hacer más intuitivo para los desarrolladores la producción de aplicaciones utilizando gráficos 3D con aceleración por hardware. La librería abstrae todo los detalles del uso de las librerías subyacentes del

sistema como Direct3D y OpenGL, provee una interfaz sobre objetos del mundo y otras clases intuitivas, para mayor información consultar [OGRE, 11].

- **IRRLICHT Engine:** Es un motor gráfico 3D en tiempo real de alto rendimiento, escrito y usable con C++ y lenguajes .NET. Es totalmente multiplataforma, utiliza Direct3D, OpenGL y tiene su propio *renderer* por software. Cuenta también tiene con la gran mayoría de las características encontradas en motores 3D comerciales, más información disponible en [Irrlicht, 11].

- **JMonkeyEngine: (***Java Monkey Engine* o JME) es un *framework* desarrollado en java, basado en múltiples API gráficas. JME utiliza una capa de abstracción, lo que permite añadirle cualquier sistema de renderizado. Actualmente utiliza LWJGL y JOGL los cuales son *bindings* de java para OpenGL. Es totalmente *Open Source* bajo licencia BSD, por lo que puede ser utilizado tanto en aplicaciones gratuitas como comerciales, más información disponible en [jMonkeyEngine, 11].

- **Panda3D:** Es un motor de juegos y un *framework* para desarrollo de motores gráficos 3D, juegos y aplicaciones escritos en Pitón y C++. Es *Open Source* y gratis para cualquier propósito, incluyendo incursiones comerciales, todo esto gracias a su licencia BSD, más información disponible en [Panda3D, 08].

2.4 Modelos 3D

Un modelo 3D es una representación computacional, de algún aspecto en particular del mundo real en términos de abstracción. Los modelos 3D pueden ser creados a partir de herramientas software de modelado y esculpido 3D, o mediante algoritmos especializados. Estos modelos además están compuestos por mallas poligonales, las cuales les dan su forma.

2.4.1 Mallas Poligonales

Una malla poligonal o *mesh* en inglés, es una colección de vértices, bordes y caras que definen la forma de un objeto poliedro. Las caras habitualmente consisten en triángulos, cuadriláteros, o polígonos convexos simples, lo cual simplifica el proceso de sendereado. Pero también pueden estar compuestas de polígonos cóncavos, o polígonos agujereados. Los objetos creados con mallas poligonales deben guardar distintos tipos de elementos, los elementos básicos según [Funkhouser, 02] se definen a continuación:

- **Vertex:** Se basa en el mismo concepto de un vértice, el cual es el punto de intersección de las aristas definido en coordenadas (x,y,z), además guarda información de color, su vector normal y coordenada de textura.

- **Arista:** Conexión entre dos *vertex*.

- **Cara:** es un conjunto cerrado de aristas, tres aristas son un triángulo, mientras cuatro aristas forman un cuadrángulo.

2.4.2 Estructuras de Datos Poligonales

Las mallas poligonales pueden ser organizadas en distintas estructuras de datos según las necesidades de acceso a estas. Algunas de estas distintas estructuras y sus asociaciones son:

- **Vertex a Vertex:** Representa al objeto como un conjunto de vértices conectados a otros vértices. Esta es la representación más simple de una malla poligonal, pero no la más usada, debido a que para obtener la información de aristas y caras, implica tener que procesar los datos de esta estructura, por lo tanto las operaciones sobre aristas y caras tienen una complejidad añadida.

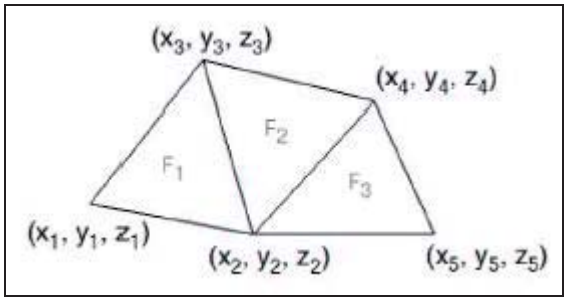


Figura 2.22: Malla simple. [Funkhouser, 02]

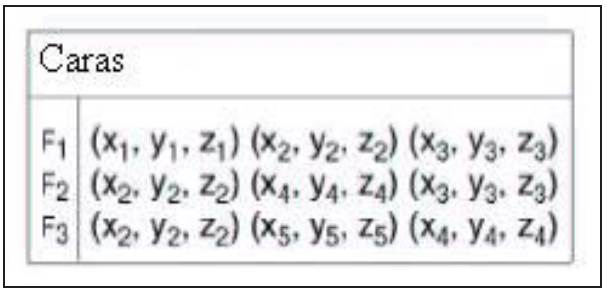


Figura 2.23: Lista de caras. [Funkhouser, 02]

- **Cara a Vertex:** Este modelo es una mejora de la estructura anterior, en el cual se agrega la capacidad de buscar de forma explícita, las caras que rodean un vértice. También se facilita la tarea de buscar los vértices pertenecientes a una cara, así mismo se facilita la búsqueda de caras adyacentes. El problema de este modelo es que persiste con la definición implícita de las aristas. En la Figura 2.24 se puede ver la organización en este modelo:

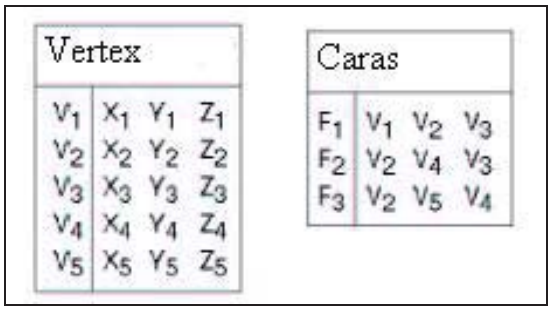


Figura 2.24: Vertex a Caras. [Funkhouser, 02]

- **Arista Alada:** En este modelo las aristas, caras y *vertex* son representadas explícitamente, esta estructura es ampliamente usada en software que requiere cambiar dinámicamente la geometría en la malla. El objetivo de esta representación, es abordar el problema de moverse entre arista y arista. Para esto se guardan hasta 4 de las aristas, conectadas a cada extremo de una arista, en sentido de las manecillas del reloj, si hay más de 4 aristas estas pueden ser obtenidas desde alguna de las aristas adyacentes

.

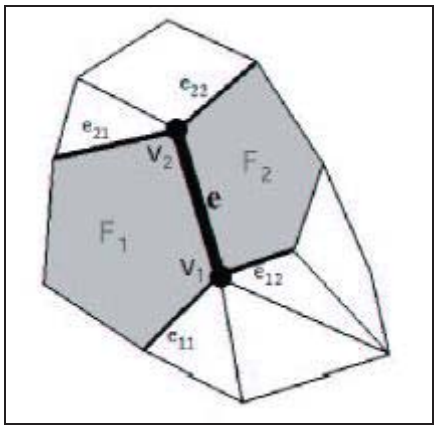


Figura 2.25: Adyacencia en Arista Alada. [Funkhouser, 02]

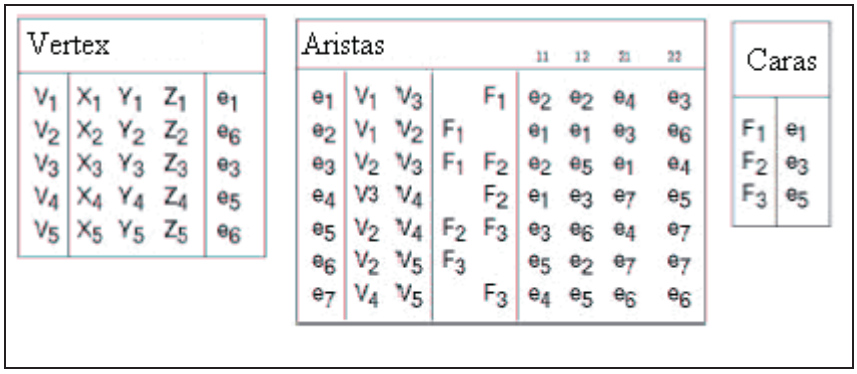


Figura 2.26: Tablas de adyacencia. [Funkhouser, 02]

2.4.3 Mapeo UV de Texturas

Este proceso de mapeo transforma una textura para ser colocada sobre un objeto 3D. En contraste a las coordenadas cartesianas "X", "Y" y "Z", es necesario agregar otro espacio de coordenadas bidimensional para describir la superficie de la malla poligonal. Para este propósito se utilizan las letras "U" y "V", dado que "X" e "Y" ya están utilizadas para el espacio del objeto 3D.

Este mapeo permite a los polígonos que componente un objeto 3D tener asignados una sección de una textura. La imagen se llama textura UV pero se trata de una imagen común y corriente. En la siguiente figura, se recorta y extiende/desenvuelve, de manera que puede ser representada en coordenadas bidimensionales, ósea "U" y "V". Esta malla extendida se proyecta sobre la textura, luego a cada vértice en cada cara de la malla se le asigna la coordenada de la porción de la textura sobre la que está extendida.

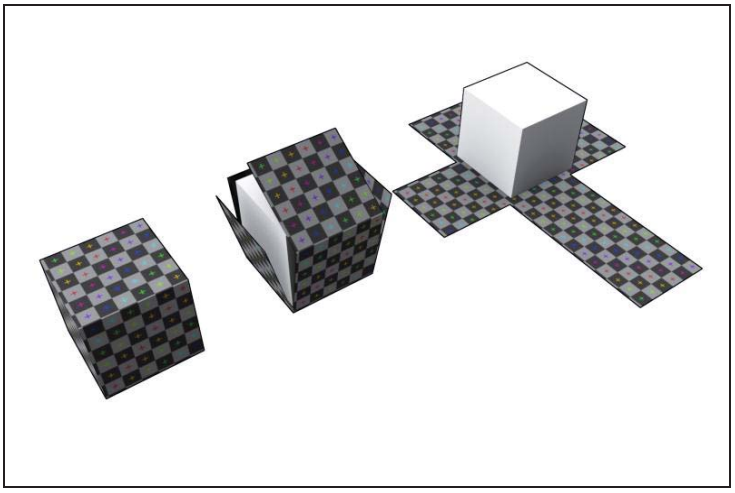


Figura 2.27: Mapeo UV. [DarkMod, 10]

2.4.4 Generación de Terrenos

La generación de un terreno 3D es realizada mediante la utilización de una imagen 2D llamada Mapa de Altura (*heightmap*). Un Mapa de Altura contiene un solo canal de color, el cual se puede visualizar como una imagen en escala de grises, cuando se aplica en la generación del terreno la tonalidad se traduce en que: mientras más oscuro sea el gris más bajo se encuentra el punto, siendo negro el punto más profundo; en cambio mientras más claro sea el gris más alto se encuentra el punto, siendo blanco el punto más elevado. Estos Mapas de Altura pueden estar guardados en archivos de imágenes conocidos como JPG, PNG y BMP, entre otros. El proceso utilizado para generar el terreno es un simple algoritmo que:

A) Genera una Malla Poligonal con el mismo ancho y alto en vértices que tiene el Mapa de Altura en píxeles. Lo más común es que las dimensiones del Mapa de altura sean de 257x257 o 513x513, en general cualquier dimensión en la forma $(2^n + 1)x(2^n + 1)$ que no exceda de 4097x4097, ya que el tiempo de procesamiento se hace demasiado extenso y la salida demasiado pesada.

B) Se ajusta la elevación del vértice correspondiente a cada uno de los píxeles del Mapa de Altura, en la proporción correspondiente a la tonalidad del píxel.

En el siguiente trozo de código se ejemplifica el procedimiento mencionado en los pasos anteriores:

```
01      //Params: Contenido del Mapa de Altura
02      //        Separacion entre cada punto del Mapa de Altura
03      Terreno(MAData *data, int separacion)
04      //Se crea un arreglo de vertex
05      //data->size contiene el tamaño alto*ancho del Mapa de Altura
06              m_vertices = new vertex[data->size];
07      //A cada vertex se le asigna una coordenada en base a los
08      //parametros de entrada
09              for(int i=0; i<data->size; i++){
```

```
10                  m_vertices[i].x = (float)(i % (separacion+1));
11      //En este caso se le asigna al eje Y la altura del MA
12                  m_vertices[i].y = data->alturas[i];
13                  m_vertices[i].z = (float)(i / (separacion+1));
14              }
15      }
```

2.4.5 Formatos de Archivo para Modelos 3D

En la realización de software que integre tecnología de gráficos 3D la mayoría de las veces resulta necesaria la utilización de modelos 3D, creados con anterioridad mediante el uso de software de modelado 3D o mediante algoritmos automáticos. Para esto los modelos 3D deben estar guardados en archivos especialmente estructurados para el almacenamiento de estos. A continuación se presentan algunos de los formatos evaluados para almacenar la salida del sistema desarrollado:

- **Wavefront OBJ**: Desarrollado por *Wavefront Technologies* para su paquete de visualización de animaciones *The Advanced visualizar*. OBJ es un formato de archivo simple que representa la geometría, la posición de cada vértice, la posición UV de cada uno de los vértices texturizados, normales, y caras que definen a los polígonos en una lista compuesta de vértices con textura. Los vértices se almacenan por defecto, ordenados en contra de las manecillas del reloj, haciendo así que la declaración explícita de las normales sea innecesaria. A continuación se presenta la estructura del archivo según lo especificado en [Reddy, 11]:

```
01      #Los comentarios van precedidos del carácter '#'
02      #Referencia a los materiales del modelo contenidos en un archivo externo
03      mtllib [archivo de texturas con extensión .mtl]
04      #Lista de vértices en coordenadas (X, Y ,Z)
05      v       1.000  2.000  3.000
06      v       2.340  3.450  4.560
07      v       …
08      #Coordenadas (U, V) de textura
09      vt      0.100  -1.000
10      vt      0.500  -1.500
11      vt      …
12      #Normales
13      vn      1.000  0.500  0.900
14      vn      0.300  0.450  0.300
15      vn      …
16      #Cada cara puede tener más de un elemento
17      #Los vertex se enumeran en orden de creación a partir de 1
```

```
18      f       v1      v2      v3      …
19      #También pueden se les puede agregar información de las normales y textura
20      f       v1/vn1/vt1      v2/vn2/vt2      …
21      #O bien solo utilizar solo normal o solo textura
22      f       v1/vn1          v2/vn2 …
23      f       v1/ /vt1                v2/ /vt2        …
```

- **Collada DAE:** COLLADA define un esquema XML de estándar abierto para el intercambio de activos digitales entre varias aplicaciones de software gráficos, las cuales en otro caso tendrían que almacenar sus activos en formatos de archivos posiblemente incompatibles entre sí. Los documentos de COLLADA que describen los activos digitales son archivos XML, generalmente identificadas con la extensión .dae (*digital asset exchange*).

  La especificación de este formato de archivo es extremadamente extensa, por esto en caso de necesitar información adicional de este formato se puede acceder al documento en PDF de su especificación en [Khronos, 10], específicamente de la versión 1.4.1 la cual es la última versión estable a la fecha de revisión. Un ejemplo de archivo COLLADA en blanco es como se muestra a continuación:

```
01      <?xml version=”1.0” encoding=”utf-8”?>
02      <!--Esto es un comentario-->
03      <!—Lo que sigue es el cuerpo del archivo-->
04      <COLLADA          xmlns=”http://www.collada.org/2005/11/COLLADASchema”
version=”1.4.1”>
05      <!—Los asset o activos entregan información acerca de su elemento padre-->
06      <!—En este caso se define a COLLADA y por ende al resto del documento-->
07              <asset>
08                      <created/>
09                      <modified/>
10              </asset>
11      <!—información sobre geometría y escenas-->
12              <library_geometries/>
13              <library_visual_scenes/>
14              <scene />
15      </COLLADA>
```

2.4.6 Herramientas de Modelado 3D

Son aplicaciones con interfaz gráfica y herramientas específicas para el modelado de objetos 3D, estas herramientas también sirven para agregar mayor detalle a modelos creados por

algoritmos automáticos para darle mayor calidad al producto final. Entre las diferentes alternativas existentes se destacan las presentadas a continuación:

- **3dStudioMax:** Es un programa de creación de gráficos y animación 3D desarrollado por Autodesk Media & Entertainment. Fue desarrollado como sucesor para sistemas operativos Win32 del 3D Studio creado para DOS. Kinetix fue más tarde fusionada con la última adquisición de Autodesk, Discreet Logic. 3d Studio Max es uno de los programas de animación 3D más utilizados. Dispone de una sólida capacidad de edición, una omnipresente arquitectura de *plugins* y una larga tradición en plataformas Microsoft Windows. 3ds Max es utilizado en mayor medida por los desarrolladores de videojuegos, aunque también en el desarrollo de proyectos de animación como películas o anuncios de televisión, efectos especiales y en arquitectura, para mayor información referirse a [Autodesk, 11].

- **Sketchup:** Es un programa de modelado y diseño en 3D, orientado al desarrollo de videojuegos, películas, ingeniería civil o simple entretenimiento personal. Los edificios creados con el programa pueden ser georeferenciados y colocados sobre las imágenes de Google Earth. Los modelos pueden ser subidos a la red mediante el propio programa Google SketchUp y almacenarse directamente en la base de datos 3D Warehouse para ser compartidos, más información disponible en [Google, 11].

- **Blender:** Es un programa multiplataforma, dedicado especialmente al modelado, animación y creación de gráficos 3D. El programa fue inicialmente distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta, aunque posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX, más información disponible en [Blender, 11].

2.5 Unreal Engine 3

Unreal Engine 3, también conocido como UDK, es un motor de videojuegos que consiste en rutinas de programación que permiten el diseño, la creación y la representación de un videojuego como tal, se encarga de proveer las funcionalidades básicas de renderizado para los gráficos 2D y 3D, manejo de físicas o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes y administración de memoria, para más información ver [Epic Games, 11], [Unreal Engine, 11] y [Jason et al., 04].

Alguna de las características más importantes del Unreal Engine 3 son:

- **Gráficos:** Unreal Engine 3 hace uso de efectos visuales avanzados, como HDR, *bloom*, *depth of field*, y todas las técnicas de iluminación por píxel y renderizado. Ejemplos de estas últimas son el *normal mapping* y el *parallax mapping* los cuales simulan relieve modificando la textura. Otra característica importante es el manejo de las sombras estáticas las cuales son proyectadas de forma permanente sobre las texturas y sombras dinámicas las cuales se proyectan correctamente sobre cualquier parte de la escena, sin importar que la fuente de luz sea móvil. Además, gracias a la función *soft shadows*, se suavizan los bordes

para que luzcan mucho más realistas, y efectos como la atenuación pueden aplicarse según el grado de iluminación presente en la escena, otros detalles importantes del motor son escenarios deformables o efectos ambientales como la niebla volumétrica.

- **Físicas y animaciones:** Cualquier cuerpo rígido dentro de un escenario de Unreal Engine 3 tiene sus correspondientes propiedades físicas, incluyendo la fricción. La herramienta incluida para la física en el motor (UnrealPhAT) soporta la creación de colisiones optimizadas para los modelos, e incluye un simulador interactivo para corroborar su comportamiento. Las animaciones vienen de la mano de la física, ya que es ésta la que afecta el movimiento de todo objeto móvil dentro de la escena. Se usa el sistema de *ragdolls* o cuerpos con esqueleto y articulaciones, avalado por una animación esqueletal que soporta hasta cuatro influencias sobre un "hueso" por vértice. Las respuestas físicas del esqueleto a los impulsos pueden ser movimientos específicos por ejemplo, el seguimiento con los ojos, caídas, vuelos o desplazamientos, entre otros. Esto se aplica a cada personaje, vehículo y objeto que forme parte del nivel.

- **Inteligencia artificial:** El recorrido trazado por los NPCS o personajes controlados por la inteligencia artificial, sorteará obstáculos tales como puertas, ascensores o mecanismos de activación, para llegar a cada uno de los rincones del escenario. La decisión sobre los movimientos efectuados puede realizarse a nivel tanto individual como grupal; en este último caso, la rutina de trabajo en equipo permite movimientos básicos como sistemas de cobertura, rodeado de objetivo y ataques coordinados, hasta el cumplimiento de objetivos a gran escala. La IA se maneja por eventos (scripts), para los cuales se usa el editor del Unreal Engine 3.0. Acciones complejas pueden constituirse sobre la base de una cadena de sucesos simples. Este esquema abre un panorama de posibilidades y jerarquías de comportamiento que enriquece el desempeño de la inteligencia artificial.

- **Sonido:** Unreal Engine 3 integra sonido posicional con soporte para seis canales (5.1) certificado por Dolby Digital y el uso del efecto *doppler* que consiste en regular la dirección y la intensidad del audio que produce una fuente sonora en movimiento, o sea producir un sonido más agudo cuando se acerca y uno más grave a medida que se aleja.

- **Red:** Unreal Engine 3 usa un sistema de red cuya arquitectura se adapta a cualquier clase de juego, usando comunicación por medio de cliente-servidor que son varios usuarios conectados a un solo servidor central o *peer to peer* que son varios computadores conectados que se comportan como iguales entre sí.

La comunicación en alto nivel que mantienen los clientes con el servidor a través del protocolo UDP está optimizada para mantenerse estable aun cuando el ancho de banda escasea y con una latencia alta. En este sistema pueden formarse partidas de hasta 64 jugadores. En cambio, en el modo *peer to peer* el número se reduce a 16. Esta característica está presente tanto para las consolas como en computadoras personales, y los jugadores de las distintas plataformas pueden convivir en una misma partida. El servidor central se encarga de proveer todo lo necesario para que cada jugador sepa dónde le conviene entrar.

Figura 2.28: Escena usando Unreal Engine 3.

Para administrar todos estos elementos nombrados anteriormente, Unreal Engine 3 provee un editor conocido como UnrealEd, el cual contiene el *Content Manager* que se encarga de administrar y crear todo el contenido del juego, como modelos, texturas, luces, animaciones, *scripts* creados con Kismet que es un sistema visual de programación, entre otros. Además da facilidad para crear geometría en el motor de forma aditiva y substractiva en base de figuras básicas como cuadrados, círculos, triángulos, así como desde modelos complejos, esta geometría creada dentro del motor se conoce como *brushes*.

3 Desarrollo

Debido a que el proyecto es de carácter exploratorio, se debió investigar, probar y evaluar distintas tecnologías con el fin de seleccionar las más acordes para su realización. La etapa de investigación sirvió para hacer un primer acotamiento en base a la factibilidad y las distintas funcionalidades ofrecidas, como por ejemplo el lenguaje de programación usado, quedando un conjunto relativamente pequeño de estas para elegir. Esta elección se hizo mediante el desarrollo de un prototipo, en el cual se fueron probando las distintas tecnologías hasta encontrar las que mejor se acomodaron para el desarrollo del proyecto.

En esta sección del informe se presenta cómo se abordó el desarrollo del proyecto, su factibilidad, cuales son las distintas tecnologías elegidas, los objetivos de estas y cómo se relacionan estas con el sistema.

3.1 Caso de Estudio

Para la realización de este proyecto se hizo uso de un conjunto de mapas digitales compuesto de un mapa vectorial que será creado a mano y por mapas vectoriales de muestra, provistos por el Instituto Geográfico Militar los cuales se encuentran referenciados con el datum Hito XVIII 1963 en la zona UTM 19S. El área representada en está delimitada por el cuadrado cuya diagonal está entre las coordenadas:

- $Long_1$ = -71.00, Lat_1 = -32.75.
- $Long_2$ = -70.75, Lat_2 = -33.00.

Figura 3.1: Aproximación al área de estudio. [Google Maps]

El área que se ve en la Figura 3.1 corresponde a la localidad de Llay-Llay y sus alrededores en la Quinta Región de Valparaíso, Chile. La imagen obtenida de Google Maps

corresponde (con un pequeño grado de error) al área que representa al conjunto de mapas a utilizar. Los mapas vectoriales que serán utilizados estarán superponiéndose entre sí (Figura 3.2). Cada uno de estos mapas contiene la siguiente información:

- **Mapa 3D**: Esta capa contiene curvas de nivel, con las cuales se puede obtener la elevación del terreno en forma de terrazas.

- **Mapa de Caminos**: Esta capa contiene líneas que representan caminos, carreteras, huellas, senderos y autopistas, entre otras. El terreno debe adaptarse a los caminos.

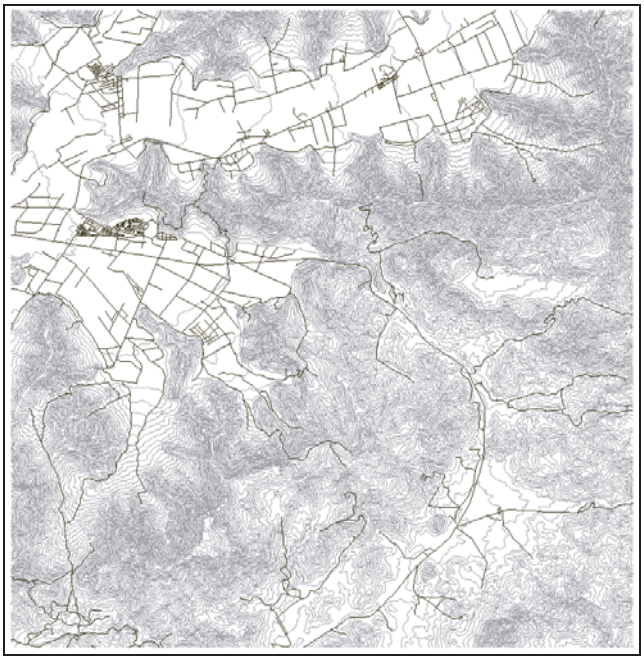


Figura 3.2: Curvas de nivel y caminos superpuestos. [Elaboración propia]

En cuanto al mapa que será creado a mano será creado y editado por medio del SIG Quantum GIS, será como dice a continuación:

- **Mapa de árboles**: Esta capa contiene información acerca de la posición en Long/Lat de los árboles en la zona, edad, y especie.

3.2 Diseño del Sistema

En esta sección se presentan los distintos diagramas del sistema, desde simples diagramas explicativos para lograr entender cómo se relaciona el sistema en general, hasta los diagramas de UML que modelan todo el sistema.

3.3 Diseño Conceptual del Sistema

El diagrama presentado en la Figura 3.3 y Figura 3.4 tiene como función mostrar el objetivo y aproximación general del sistema.

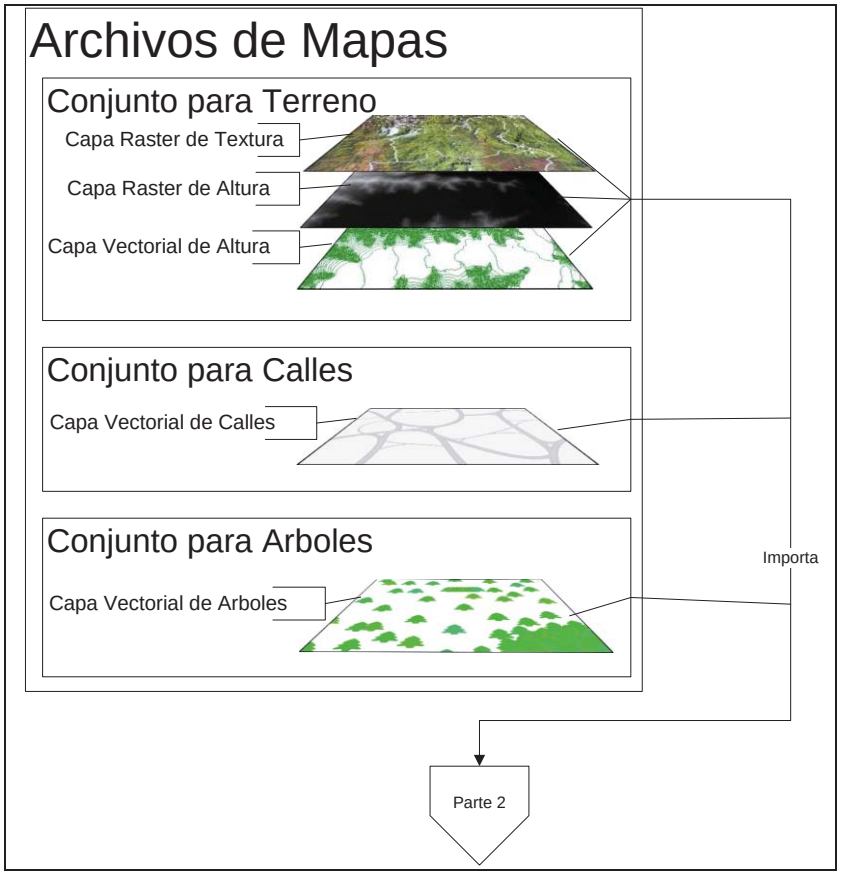


Figura 3.3: Modelo Conceptual del sistema. Parte 1.

En la Figura 3.3 y Figura 3.4 se presenta la funcionalidad principal del sistema que consiste en generar un modelo 3D a partir de la información obtenida de distintos mapas digitales, para poder ser utilizado en distintas aplicaciones 3D.

En la Figura 3.3 se presenta la visión abstracta del manejo de las capas, las que en este caso son mapas digitales. Esto consiste en un conjunto de capas que sirven para un objetivo específico para la generación del terreno 3D. Para dejar más claro lo anterior se procederá a explicar el objetivo de cada conjunto y para qué se usa su respectiva capa, lo cual es mostrado en conjunto en la Figura 3.4.

El conjunto para terreno está formado por:

- **Capa Raster de Textura:** entrega los datos de la textura que se va a usar para el terreno 3D.
- **Capa Raster de Altura:** entrega los datos para hacer un mapa de Altura.
- **Capa Vectorial de Altura:** entrega los datos para hacer un mapa de Altura.

La aplicación toma este conjunto y con el genera la parte que corresponde al terreno, con su respectiva elevación y textura, y en caso de ser necesario se genera un modelo de guía, que consiste en un modelo con las características del mapa dibujadas en la textura, para usarse de guía para posicionar los distintos objetos en los editores de modelos.

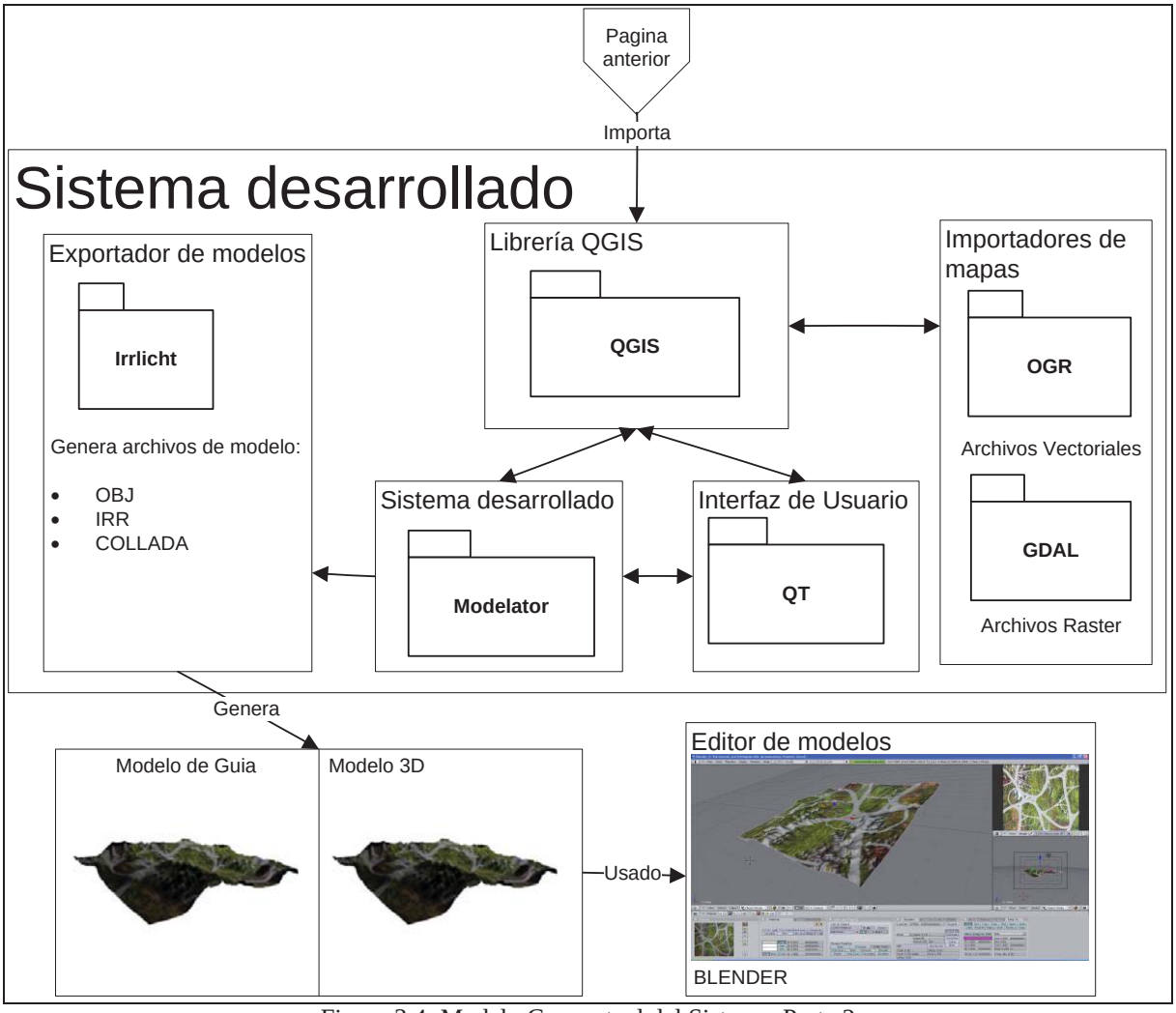


Figura 3.4: Modelo Conceptual del Sistema. Parte 2.

El conjunto para calles está formado por:

- **Capa Vectorial de Calles:** entrega los datos de las calles en el mapa.

Con estos datos se genera una textura para las calles, junto con su respectiva modificación de la malla para que no queden posicionadas fuera de lugar.

El conjunto para árboles está formado por:

- **Capa Vectorial de Árboles:** entrega los datos de la posición de los árboles en el mapa.

A partir de estos datos se da la oportunidad al usuario de posicionar modelos que representan árboles directamente en el terreno, o generar un modelo de guía con la posición de los arboles dibujada en la textura.

- **Interacción Conceptual con Librerías**

En la siguiente figura se muestra la aplicación junto a las librerías usadas, el manejo y muestra de los mapas está hecha con QGIS, el cual está compuesto por tres librerías esenciales

para el sistema, que son GDAL para la lectura de mapas de tipo *Raster* y OGR para mapas de tipo Vectorial y QT que se usa para el manejo de la Interfaz de Usuario. Para la exportación, generación y edición del modelo la aplicación usa la librería Irrlicht. Lo concerniente a estas librerías y su relación con el programa esta explicado más a fondo en el siguiente punto.

Figura 3.5: Diagrama general de librerías

En la Figura 3.5 se puede apreciar de una manera más clara y grafica las distintas librerías que ocupa el sistema. Estas son:

- **QGIS**: Librería hecha para la creación de SIG, es una librería desarrollada para el programa Quantum GIS que ofrece las mismas funcionalidades que esta aplicación. QGIS gracias al uso de varias librerías como GDAL, OGR y QT facilita la administración de mapas digitales.

- **QT**: Librería orientada a interfaces de usuario similar a swing de java, esta es la librería grafica usada por Quantum GIS. En el sistema se usa para toda la interfaz gráfica y manejo de eventos.

- **GDAL**: Librería que tiene como objetivo importar distintos tipos de mapas en formato *Raster*, usada en Quantum GIS. En el sistema se usa para cargar los mapas de tipo *Raster*.

- **OGR**: Librería que es parte de GDAL usada para importar distintos tipos de mapas en formato Vectorial, utilizado en Quantum GIS. En el sistema se usa para cargar los mapas de tipo Vectorial.

- **Irrlicht**: Motor gráfico 3D que cuenta con funcionalidades extra, entre estas está el soporte para exportar modelos en distintos formatos. En el sistema se usa para exportar, generar y editar el modelo.

Como se puede apreciar anteriormente se usa para la aplicación la librería desarrollada para Quantum GIS, esto es porque se estudió las librerías que ocupaba para realizar sus funcionalidades, como GDAL, OGR y QT. Posteriormente estas librerías se incorporaron al sistema por medio de la librería de QuantumGIS, que ofrece una robusta implementación de GDAL, OGR y facilidades para la administración de los mapas. En cuanto a la relación de estas librerías con el sistema, este carga varios mapas gracias a GDAL y OGR, los cuales componen un conjunto. Este conjunto es procesado por la aplicación, se genera un modelo y se exporta con Irrlicht. Todo esto es hecho en interfaces gráficas hechas en QT y con la ayuda de la librería de QGIS que facilita toda la administración y muestra de los mapas. Se procederá a hablar más de la relación entre los distintos componentes del sistema en los siguientes diagramas UML.

3.4 Casos de Uso

En esta sección se encuentran los diagramas y descripciones de la interacción del usuario con el sistema y una vista general de sus funcionalidades.

- **Caso de Uso del Sistema**

En Figura 3.6 se aprecia las dos principales funcionalidades del sistema.

El sistema permite al usuario en primer lugar administrar las distintas capas, ya sea cargando, eliminando o eligiendo otras capas. En segundo lugar permite Generar el modelo en base a los mapas cargados en administrar capas, con su respectiva configuración. Se explayara más a fondo las funcionalidades de Administrar capas y Generar modelo en la siguiente sección.

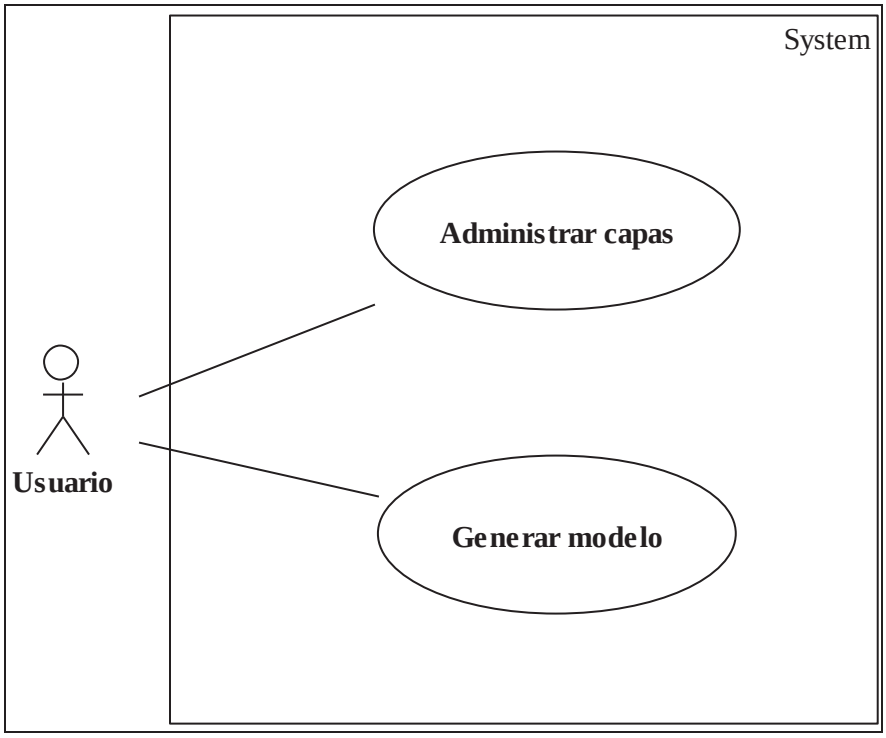


Figura 3.6: Caso de Uso del Sistema Completo

- **Caso de Uso de Administrar Capas**

  En la Figura 3.7 se aprecia las funcionalidades al momento de administrar capas.

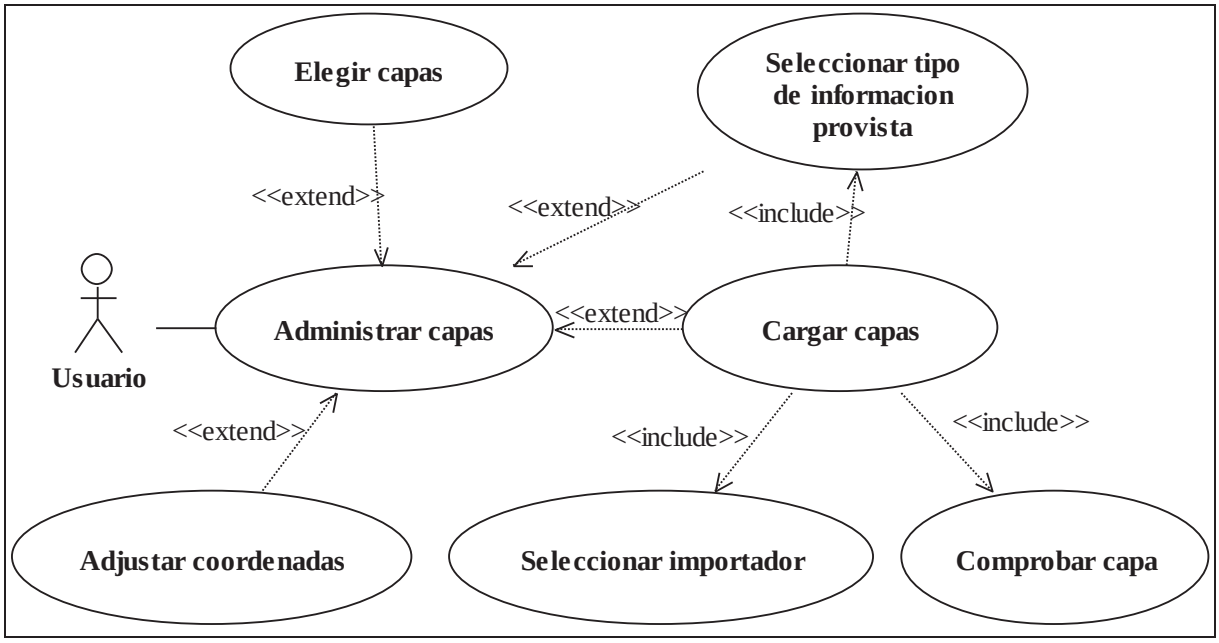


Figura 3.7: Caso de uso de Administrar capas

En el apartado de la administración de capas el sistema permite al usuario elegir qué conjunto de capas se quiere agregar, cargar capas a este conjunto de capas y para esto se debe seleccionar el importador correcto para el archivo, como por ejemplo para un ".shp" que es un

formato vectorial se selecciona OGR, se comprueba que la capa haya sido cargada correctamente y sea válida. Posteriormente se selecciona el conjunto al que pertenece la capa, como por ejemplo Terreno. Por otra parte el sistema permite ajustar las coordenadas del área seleccionada a exportar, como también el sistema de coordenadas.

- **Caso de Uso de Generar Modelo**

El diagrama presentado en la Figura 3.8 muestra todas las funcionalidades en el momento que el sistema genera un modelo.

La generación del modelo tiene la funcionalidad de exportar modelo 3D para ser leído en otras aplicaciones, para este proceso se deben procesar las capas y validarlas para generar el modelo. En lo que concierne al formato de exportación el usuario selecciona el formato a exportar y el sistema obtiene el formato y la configuración adecuada para el tipo de archivo que tiene que generar. Otra funcionalidad es la configuración del modelo, que son configuraciones generales para todos los modelos, sin importar su formato.

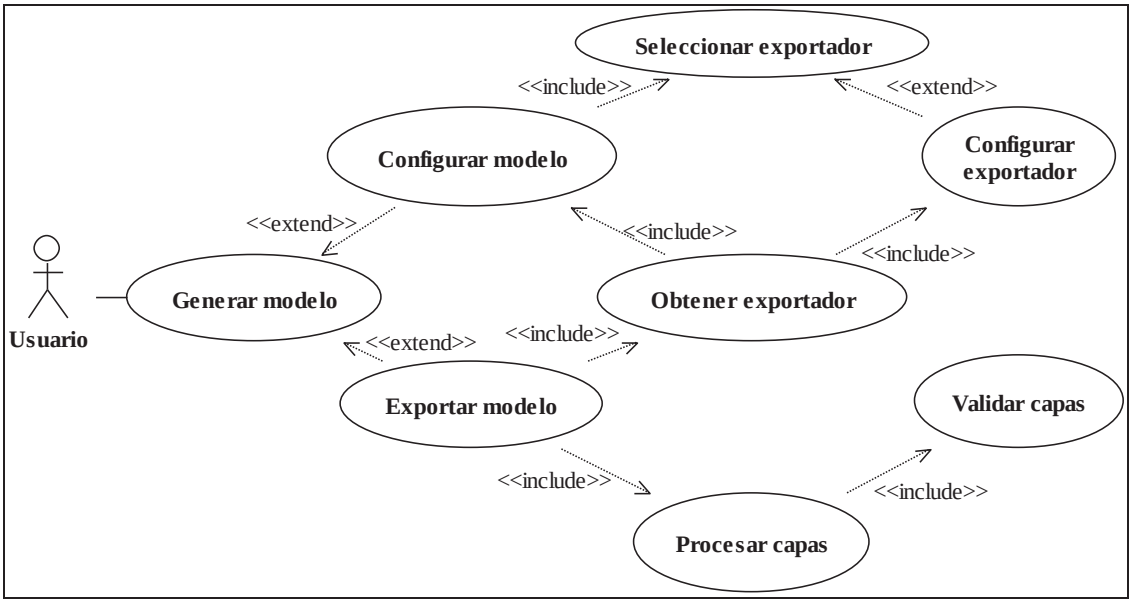


Figura 3.8: Caso de Uso de Generar Modelo

3.5 Casos de Uso Extendidos

En las siguientes tablas se explican algunos de los casos de uso más relevantes dentro del sistema.

Tabla 3.1: Caso de Uso Administrar Capas.

Nombre del Caso de Uso:	Administrar Capas.
Actor(es):	Usuario
Objetivo(s):	Agregar y eliminar conjuntos de capas o capas individuales.
Tipo de Caso de Uso:	Principal Esencial.
Descripción:	Se ofrece un conjunto de opciones para que el usuario pueda gestionar las capas que serán utilizadas en la generación del modelo 3D.

Referencias cruzadas:	Ninguna		
Curso normal de eventos			
Actor		Sistema	
1	El usuario accede al menú de agregar conjuntos de capas	2	El sistema ingresa el conjunto de capas indicado por el usuario
3	El usuario accede al menú de agregar una capa	4	El sistema carga la capa seleccionada y la presenta en su posición correspondiente en pantalla
5	El usuario accede al menú de eliminar un conjunto de capas agregado anteriormente	6	El sistema elimina el conjunto de capas, incluyendo las capas cargadas

Tabla 3.2: de Uso Generar Modelo.

Nombre del Caso de Uso:	Generar Modelo.		
Actor(es):	Usuario		
Objetivo(s):	Generar y exportar un modelo 3D del terreno.		
Tipo de Caso de Uso:	Primario Esencial.		
Descripción:	Se ofrecen las opciones de generar y exportar un modelo 3D en base a los conjuntos de capas que el usuario haya decidido utilizar, además se proporciona un menú de configuración para elegir formato y opciones para el modelo.		
Referencias cruzadas:	Ninguna.		
Curso normal de eventos			
Actor		Sistema	
1	El usuario accede al menú de generación del modelo.	2	El sistema despliega las opciones disponibles.
3	El usuario indica la opción: a) Exportar Modelo. b) Configurar Modelo.	4.a	El sistema utiliza las capas disponibles para generar un modelo 3D y luego exportarlo con las opciones definidas por defecto o por el usuario.
		4.b	El sistema despliega un dialogo que dispone del conjunto de opciones con los formatos de exportación disponibles y propiedades para la generación del modelo.
5	El usuario manipula las opciones y configura el modelo de acuerdo a sus necesidades.	6	El sistema guarda las opciones modificadas y cambia el sistema de exportación en caso que sea requerido un formato distinto al ofrecido por defecto. Luego se cierra el dialogo.
7	El usuario retorna al paso 3		

3.6 Diagrama de Clases

En el diagrama de la Figura 3.9 se muestra las distintas clases del sistema y cuál es el objetivo de cada una de ellas.

La primera clase utilizada es Modelator que sólo se encarga de crear una instancia de ModelatorUI. Este último tiene como objetivo ser la interfaz de usuario principal y clase principal del sistema, la cual le ofrece al usuario la posibilidad de agregar un conjunto de capas, eliminar un conjunto de capas, actualizar las coordenadas elegidas o generar el modelo. Otro aspecto de esta clase es que mantiene una Tabla *Hash* con los distintos conjuntos de capas existentes en el sistema.

ILayerCollection es una clase abstracta instanciada por ModelatorUI cuando se requiere agregar un nuevo conjunto de capas. El objetivo de esta clase es almacenar y cargar las capas, estas son guardadas en dos Tablas *Hash*, una para mapas *Raster* y otra para mapas Vectoriales. Esta clase es heredada por varias subclases y el objetivo de estas subclases es inicializar las Tablas *Hash* con varios tipos de mapas requerido para el tipo de conjunto. Ejemplo de esto es TerrainLayerCollection que inicializa las Tablas *Hash* para sus tres capas de la siguiente forma:

```
01      hashDeMapasRaster->agregar("Altura",NULL);
02      hashDeMapasRaster->agregar("Textura",NULL);
03      hashDeMapasVectoriales->agregar("Altura",NULL);
```

Donde se pasan dos atributos: un nombre representativo del tipo de capa, y un importador de capas, ya sea *Raster* o Vectorial o NULL si se está inicializando como en el caso de ejemplo. Posteriormente se lee esta configuración y su padre (ILayerCollection) genera los QWidget correspondientes, que consiste en unos botones. La funcionalidad del botones es la elección del mapa, creando un RasterImporter o un VectorImporter dependiendo si es un mapa *Raster* o Vectorial y cargándolo con QGIS, ya sea un mapa basado en GDAL o OGR, dependiendo de su extensión como por ejemplo RasterImporter para mapas *Raster* con extensión ".TIF" y además del borrado y zoom del mapa.

Acerca de ModelGenUI es un QWidget que se encarga de mantener la interfaz para la configuración del modelo, así como un botón de exportación. La funcionalidad de este botón es crear una instancia de ModelConfig que tiene como objetivo guardar la configuración seleccionada por el usuario para la generación del modelo 3D y se instancia un IModelExporter pasándole la configuración creada anteriormente.

La clase abstracta IModelExporter tiene por objetivo generar y exportar el modelo con ayuda de Irrlicht, que esta encapsulado en el QIrrlichtWidget. Esta clase usa una enumeración (ModelFormat) para saber el formato al cual debe exportar el modelo. Por ejemplo M_OBJ ejecuta el salvado de un modelo 3D en formato ".OBJ".

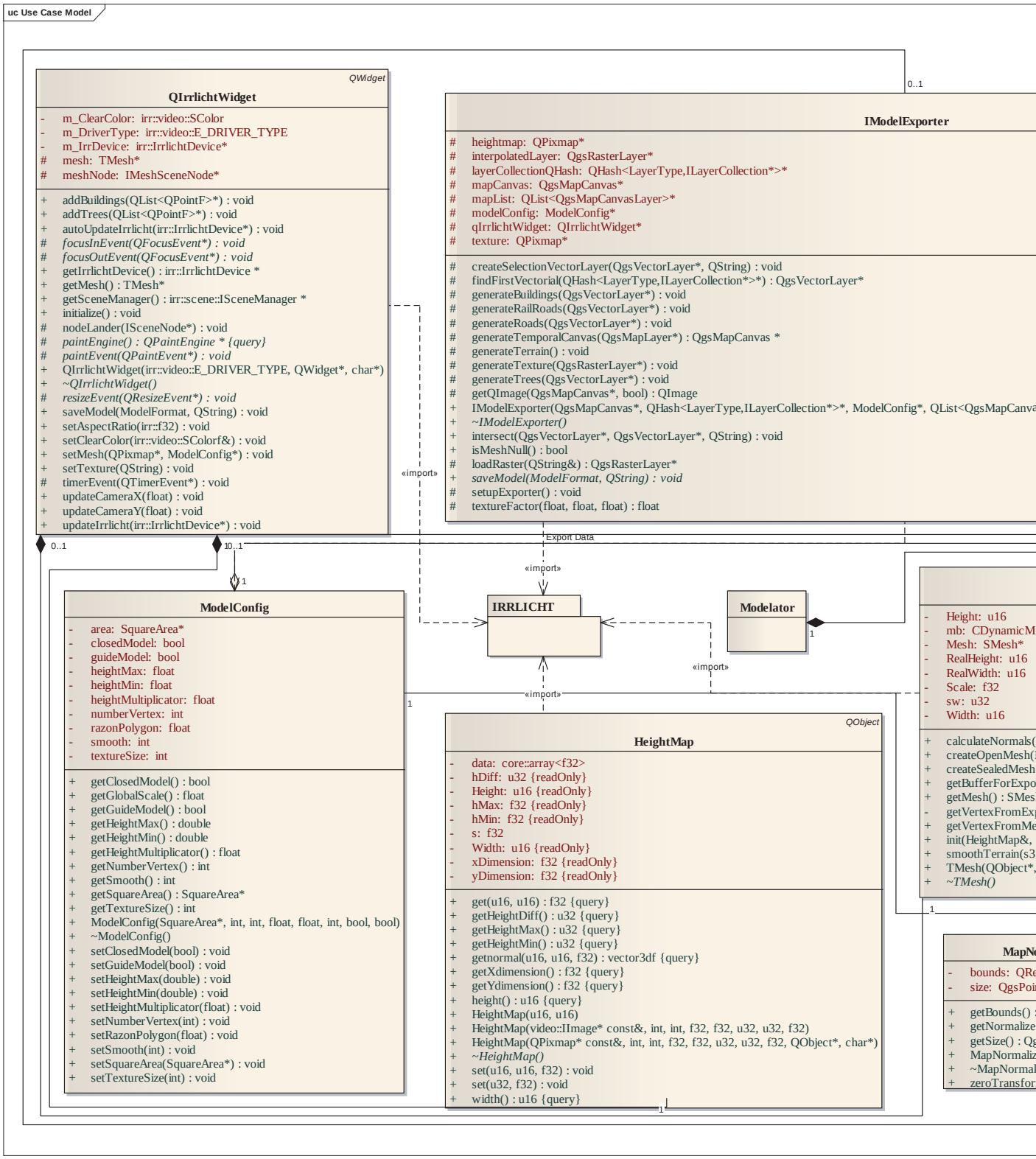
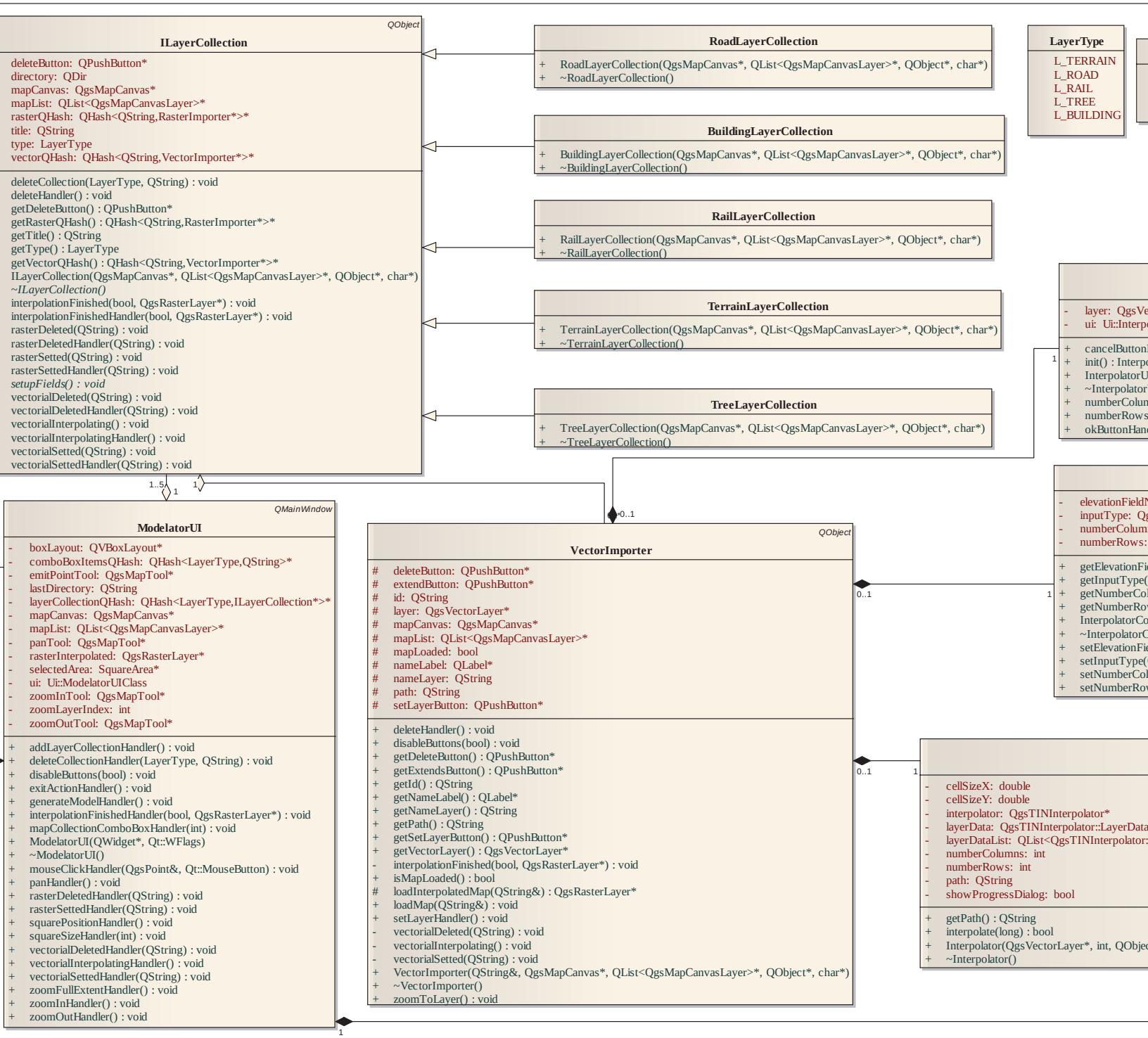


Figura 3.9: Diagrama de Clases.


QObject
asLayer>*, QgsRasterLayer*, QIrrlichtWidget*, QObject*, char*)
1
SquareArea
- deviceUnitsPosition: QgsPoint
- mapCanvas: QgsMapCanvas*
- mapUnitsPosition: QgsPoint
- positionSetted: bool
- rubberBand: QgsRubberBand*
- selectedRect: QgsRectangle
- setted: bool
- sideMeasurement: double
- size: int
+ clearSquare() : void
+ getAreaInMapUnits() : QgsRectangle
+ getDeviceUnitsPosition() : QgsPoint
+ getMapUnitsPosition() : QgsPoint
+ getRubberBand() : QgsRubberBand*
+ getSetted() : bool
+ getSideMeasurement() : double
+ getSize() : int
+ isVisible() : bool
+ setMapUnitsPosition(QgsPoint, bool) : void
+ setSize(int, bool) : void
+ setVisible(bool) : void
+ SquareArea(QgsMapCanvas*)
+ ~SquareArea()
+ update() : void
1
QObject
RasterImporter
deleteButton: QPushButton*
extendButton: QPushButton*
fileExtension: QString
id: QString
layer: QgsRasterLayer*
mapCanvas: QgsMapCanvas*
mapList: QList<QgsMapCanvasLayer>*
mapLoaded: bool
max: double
min: double
name: QString
nameLabel: QLabel*
path: QString
setLayerButton: QPushButton*
+ deleteHandler() : void
+ disableButtons(bool) : void
+ getDeleteButton() : QPushButton*
+ getExtendsButton() : QPushButton*
+ getFileExtension() : QString
+ getId() : QString
+ getMax() : double
+ getMin() : double
+ getName() : QString
+ getNameLabel() : QLabel*
+ getPath() : QString
+ getRasterLayer() : QgsRasterLayer*
+ getSetLayerButton() : QPushButton*
+ isMapLoaded() : bool
+ loadMap(QString&) : void
- rasterDeleted(QString) : void
+ RasterImporter(QString&, QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~RasterImporter()
- rasterSetted(QString) : void
+ setLayerHandler() : void
+ zoomToLayer() : void
1
0..*
0..1
QObject
TMesh
eshBuffer*
() : void
HeightMap&) : void
(HeightMap&) : void
ort() : CDynamicMeshBuffer&
h*
portBuffer(s32) : S3DVertex&
eshBuffer(s32) : S3DVertex&
f32, IVideoDriver*, u16, bool) : void
32) : void
, char*)
0..1
QDialog
ModelGenUI
- area: SquareArea*
- interpolatedLayer: QgsRasterLayer*
- irrWidget: QIrrlichtWidget*
- layerCollectionQHash: QHash<LayerType,ILayerCollection*>*
- mapCanvas: QgsMapCanvas*
- mapList: QList<QgsMapCanvasLayer>*
- modelExporter: IModelExporter*
- ui: Ui::ModelGenUIClass
- vectorialHeightLoaded: bool
- wakeUp: bool
+ cancelButtonHandler() : void
+ generateButtonHandler() : void
+ getTestFrame() : QFrame*
+ getTestGraphicView() : QGraphicsView*
+ horizontalCameraHandler(int) : void
+ irrlichtTerminoHandler(int) : void
+ keyPress(QKeyEvent*) : void
+ keyRelease(QKeyEvent*) : void
+ ModelGenUI(QgsMapCanvas*, QHash<LayerType,ILayerCollection*>*, SquareArea*, QList<QgsMapCanvasLayer>*, QgsRasterLayer*, QWidget*, char*, bool, Qt::WFlags)
+ ~ModelGenUI()
+ mouseMove(QMouseEvent*) : void
+ mousePress(QMouseEvent*) : void
+ mouseRelease(QMouseEvent*) : void
+ numberColumnsSpinBoxHandler(int) : void
+ numberRowsSpinBoxHandler(int) : void
+ saveButtonHandler() : void
+ textureSizeSpinBoxHandler(int) : void
+ vertexSpinBoxHandler(int) : void
+ verticalCameraHandler(int) : void
- waitIrrlicht() : void
+ wheel(QWheelEvent*) : void
1
1
ormalizator
ectF
nt
: QRectF
ed(QgsPoint) : QgsPoint
gsPoint
zator(QRectF)
lizator()
m(QgsPoint) : QgsPoint
1
1


QObject
ILayerCollection
deleteButton: QPushButton*
directory: QDir
mapCanvas: QgsMapCanvas*
mapList: QList<QgsMapCanvasLayer>*
rasterQHash: QHash<QString,RasterImporter*>*
title: QString
type: LayerType
vectorQHash: QHash<QString,VectorImporter*>*
deleteCollection(LayerType, QString) : void
deleteHandler() : void
getDeleteButton() : QPushButton*
getRasterQHash() : QHash<QString,RasterImporter*>*
getTitle() : QString
getType() : LayerType
getVectorQHash() : QHash<QString,VectorImporter*>*
ILayerCollection(QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
~ILayerCollection()
interpolationFinished(bool, QgsRasterLayer*) : void
interpolationFinishedHandler(bool, QgsRasterLayer*) : void
rasterDeleted(QString) : void
rasterDeletedHandler(QString) : void
rasterSetted(QString) : void
rasterSettedHandler(QString) : void
setupFields() : void
vectorialDeleted(QString) : void
vectorialDeletedHandler(QString) : void
vectorialInterpolating() : void
vectorialInterpolatingHandler() : void
vectorialSetted(QString) : void
vectorialSettedHandler(QString) : void
RoadLayerCollection
+ RoadLayerCollection(QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~RoadLayerCollection()
BuildingLayerCollection
+ BuildingLayerCollection(QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~BuildingLayerCollection()
RailLayerCollection
+ RailLayerCollection(QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~RailLayerCollection()
TerrainLayerCollection
+ TerrainLayerCollection(QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~TerrainLayerCollection()
TreeLayerCollection
+ TreeLayerCollection(QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~TreeLayerCollection()
LayerType
L_TERRAIN
L_ROAD
L_RAIL
L_TREE
L_BUILDING
1..5
1
1
QMainWindow
ModelatorUI
- boxLayout: QVBoxLayout*
- comboBoxItemsQHash: QHash<LayerType,QString>*
- emitPointTool: QgsMapTool*
- lastDirectory: QString
- layerCollectionQHash: QHash<LayerType,ILayerCollection*>*
- mapCanvas: QgsMapCanvas*
- mapList: QList<QgsMapCanvasLayer>*
- panTool: QgsMapTool*
- rasterInterpolated: QgsRasterLayer*
- selectedArea: SquareArea*
- ui: Ui::ModelatorUIClass
- zoomInTool: QgsMapTool*
- zoomLayerIndex: int
- zoomOutTool: QgsMapTool*
+ addLayerCollectionHandler() : void
+ deleteCollectionHandler(LayerType, QString) : void
+ disableButtons(bool) : void
+ exitActionHandler() : void
+ generateModelHandler() : void
+ interpolationFinishedHandler(bool, QgsRasterLayer*) : void
+ mapCollectionComboBoxHandler(int) : void
+ ModelatorUI(QWidget*, Qt::WFlags)
+ ~ModelatorUI()
+ mouseClickHandler(QgsPoint&, Qt::MouseButton) : void
+ panHandler() : void
+ rasterDeletedHandler(QString) : void
+ rasterSettedHandler(QString) : void
+ squarePositionHandler() : void
+ squareSizeHandler(int) : void
+ vectorialDeletedHandler(QString) : void
+ vectorialInterpolatingHandler() : void
+ vectorialSettedHandler(QString) : void
+ zoomFullExtentHandler() : void
+ zoomInHandler() : void
+ zoomOutHandler() : void
1
0..1
QObject
VectorImporter
deleteButton: QPushButton*
extendButton: QPushButton*
id: QString
layer: QgsVectorLayer*
mapCanvas: QgsMapCanvas*
mapList: QList<QgsMapCanvasLayer>*
mapLoaded: bool
nameLabel: QLabel*
nameLayer: QString
path: QString
setLayerButton: QPushButton*
+ deleteHandler() : void
+ disableButtons(bool) : void
+ getDeleteButton() : QPushButton*
+ getExtendsButton() : QPushButton*
+ getId() : QString
+ getNameLabel() : QLabel*
+ getNameLayer() : QString
+ getPath() : QString
+ getSetLayerButton() : QPushButton*
+ getVectorLayer() : QgsVectorLayer*
- interpolationFinished(bool, QgsRasterLayer*) : void
+ isMapLoaded() : bool
# loadInterpolatedMap(QString&) : QgsRasterLayer*
+ loadMap(QString&) : void
+ setLayerHandler() : void
- vectorialDeleted(QString) : void
- vectorialInterpolating() : void
- vectorialSetted(QString) : void
+ VectorImporter(QString&, QgsMapCanvas*, QList<QgsMapCanvasLayer>*, QObject*, char*)
+ ~VectorImporter()
+ zoomToLayer() : void
0..1
1
0..1
1
- cellSizeX: double
- cellSizeY: double
- interpolator: QgsTINInterpolator*
- layerData: QgsTINInterpolator::LayerData
- layerDataList: QList<QgsTINInterpolator:
- numberColumns: int
- numberRows: int
- path: QString
- showProgressDialog: bool
+ getPath() : QString
+ interpolate(long) : bool
+ Interpolator(QgsVectorLayer*, int, QObjec
+ ~Interpolator()
1
- layer: QgsVe
- ui: Ui::Interpo
+ cancelButton
+ init() : Interpo
+ InterpolatorU
+ ~Interpolator
+ numberColun
+ numberRows
+ okButtonHan
- elevationFieldN
- inputType: Qg
- numberColum
- numberRows:
+ getElevationFi
+ getInputType(
+ getNumberCol
+ getNumberRo
+ InterpolatorCo
+ ~InterpolatorC
+ setElevationFie
+ setInputType(
+ setNumberCol
+ setNumberRo


ModelFormat
M_OBJ
M_COLLADA
M_IRR
QT
Todas las Clases usan QT.
QGIS
Todas las Clases usan QGIS.
QDialog
InterpolatorUI
ectorLayer*
olatorUIClass
Handler() : void
olatorConfig*
I(QgsVectorLayer*, QWidget*, char*, bool, Qt::WFlags)
UI()
mnsSpinBoxHandler(int) : void
sSpinBoxHandler(int) : void
dler() : void
InterpolatorConfig
Name: QString
gsTINInterpolator::InputType
ns: int
int
eldName() : QString
) : QgsTINInterpolator::InputType
lumns() : int
ws() : int
nfig(QString, int, int, QgsTINInterpolator::InputType)
Config()
eldName(QString) : void
QgsTINInterpolator::InputType) : void
lumns(int) : void
ws(int) : void
QObject
Interpolator
:LayerData>
ct*, char*, bool, int, int, double, double, QString, QgsTINInterpolator::InputType, bool)

La clase SquareArea es usada para almacenar y seleccionar las coordenadas del área y su tamaño en el mapa para la generación del modelo 3D, esta usa una QgsRubberBand de QGIS que sirve para dibujar polígonos sobre el mapa, en este caso un cuadrado que representa el área a utilizar.

Para la generación de la altura del modelo 3D a partir de un mapa Vectorial, se usa una clase dedicada a la interpolación de mapas de altura vectoriales con QGIS, la cual genera un mapa Raster de altura para ser usado en la generación del modelo.

Existe una enumeración llamada LayerType que representa los tipos de conjuntos de capas existentes, son usados por las distintas clases, principalmente por las Tablas *Hash* para identificar el tipo de conjunto. Por ejemplo el LayerType L_TERRAIN corresponde al conjunto de capas sobre Terrenos, aplicado seria:

```
01     hashDeLayerCollections->obtener(L_TERRAIN);
```

El cual devuelve un TerrainLayerCollection.

Un punto relevante es que hay tres clases que extienden de QWidgets que son los elementos de las interfaces gráficas de QT, estas son ModelatorUI, la cual extiende de QMainWindow que es usada para ventanas principales. La otra es ModelGenUI que extiende de QDialog, esta es usada para en configuración del modelo y la ultima es InterpolatorUI que es el dialogo para la configuración de la interpolación. Todas estas clases que extienden de QWidgets, tienen una variable que contiene los elementos creados gráficamente por QT, como por ejemplo Ui::ModelatorUIClass modelatorUI en el caso de ModelatorUI.

3.7 Diagramas de Secuencia

- **Diagrama de Secuencia General del Sistema**

El presente diagrama en la Figura 3.10 muestra cómo interactúan las distintas clases del sistema para cumplir las funcionalidades principales.

Acá se puede apreciar como los principales elementos de la interfaz de usuario son creados. Se crea un SquareArea para ser usado en la selección del área para generar el modelo 3D y se crean los distintos botones con sus métodos asociados para manejar los conjuntos de capas.

Cuando el usuario presiona el botón de agregar un conjunto se ejecuta el *Handler* “addLayerCollectionHandler” con el parámetro que indica el tipo de LayerType seleccionado, ModelatorUI en base al LayerType crea un conjunto de capas correspondiente a ese LayerType y se remueve de una lista de LayerType, para no ser agregado nuevamente, ya que se puede tener solamente 1 conjunto de capas por tipo. Por ejemplo el usuario selecciono agregar un conjunto sobre Terreno, se busca el entero correspondiente al LayerType (L_TERRAIN) y

ModelatorUI crea un ILayerCollection correspondiente a Terreno, o sea un TerrainLayerCollection. Esto esta expandido en el Diagrama de Secuencia de Importar Capa.

En el caso de presionar el botón de exportar el modelo, se ejecuta el *Handler* "generateModelHandler" y se crea un ModelGenUI el cual recibe como parámetro el SquareArea que corresponde al área del mapa seleccionada por el usuario para generar el modelo 3D, ModelGenUI muestra un dialogo para seleccionar la configuración del modelo y posteriormente se exporta el modelo. Para ver con más detalle este proceso ver el Diagrama de Secuencia Exportar Modelo.

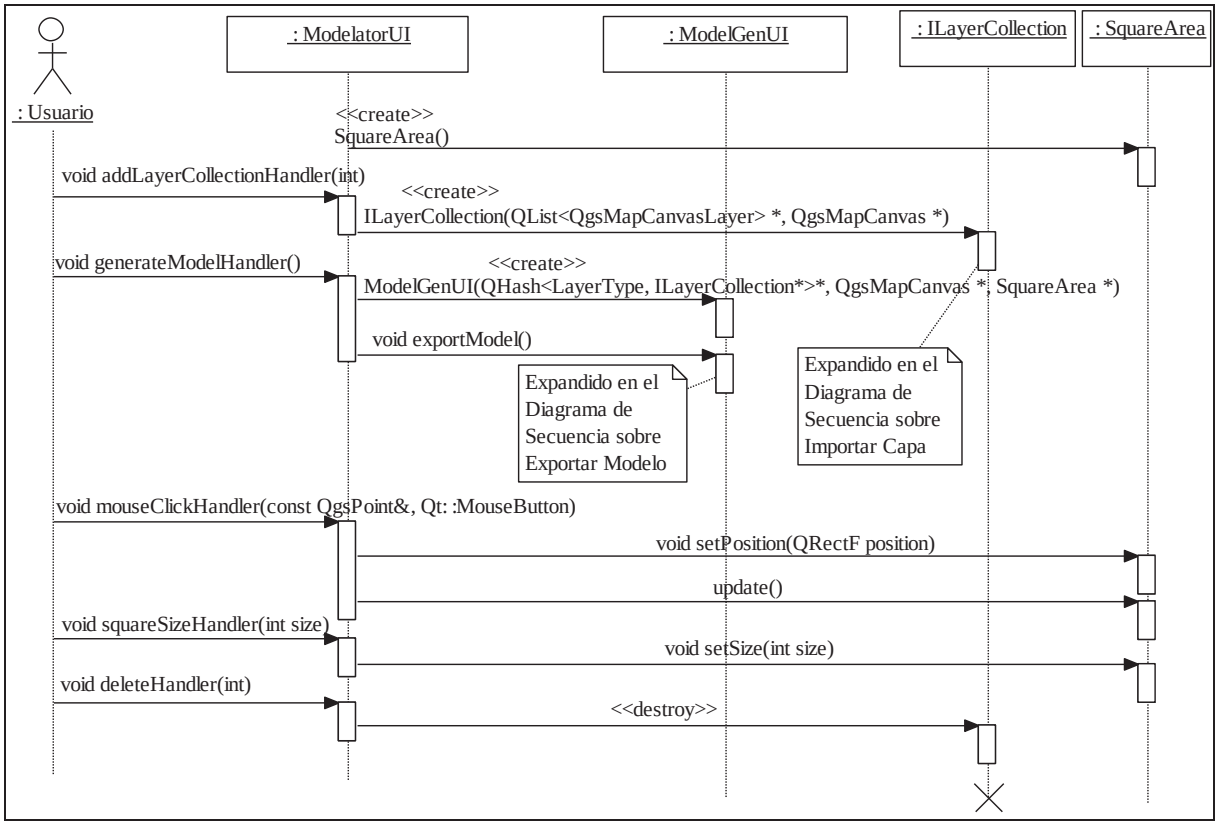


Figura 3.10: Diagrama de Secuencia General del Sistema

Otro caso es cuando el usuario entra en el modo de seleccionar el área del mapa para generar el modelo 3D, al hacer click con el mouse en el mapa ejecuta el ClickHandler() que entrega la posición del mouse en el mapa, que es usada para posicionar un cuadrado representando el área seleccionada en el mapa y actualizar la información en la instancia de SquareArea. En relación a este caso está el squareSizeHandler que es llamado por un *Slider* para elegir el tamaño del área del mapa a generar.

En el caso de presionar el botón de eliminar conjunto, se ejecuta el *Handler* "eliminarButtonHandler" con el parámetro que indica el LayerType a ser eliminado, pos eliminación se agrega de nuevo a la lista para poder ser agregado nuevamente.

- **Diagrama de Secuencia Importar Capa**

El diagrama de secuencia que se ve en la Figura 3.11 muestra cómo se importan las capas en una colección de capas de algún tipo.

El usuario presiona el botón de agregar nuevo conjunto de capas y se ejecuta el *Handler* "addLayerCollectionHandler" de ModelatorUI, el cual instancia un ILayerCollection o mejor dicho un conjunto de capas del tipo correspondiente al LayerType entregado por el usuario. Posteriormente el ILayerCollection instancia dos Tablas *Hash* con el formato <QString, RasterImporter> o <QString, VectorImporter>, una almacena capas *Raster* con RasterImporter y la otra capas Vectoriales con VectorImporter. Se usa un QString para identificar cada capa dentro de cada Tabla *Hash* con un nombre significativo de su uso y estos son inicializados dependiendo de la clase que hereda a ILayerCollection. Ejemplo de algunas de estas son:

- TerrainLayerCollection: que implementa las capas de Altura Vectorial o *Raster* y una capa de Textura.

- RoadLayerCollection: este implementa la capa vectorial de Camino.

- TreeLayerCollection: esta implementa la capa vectorial de Árboles.

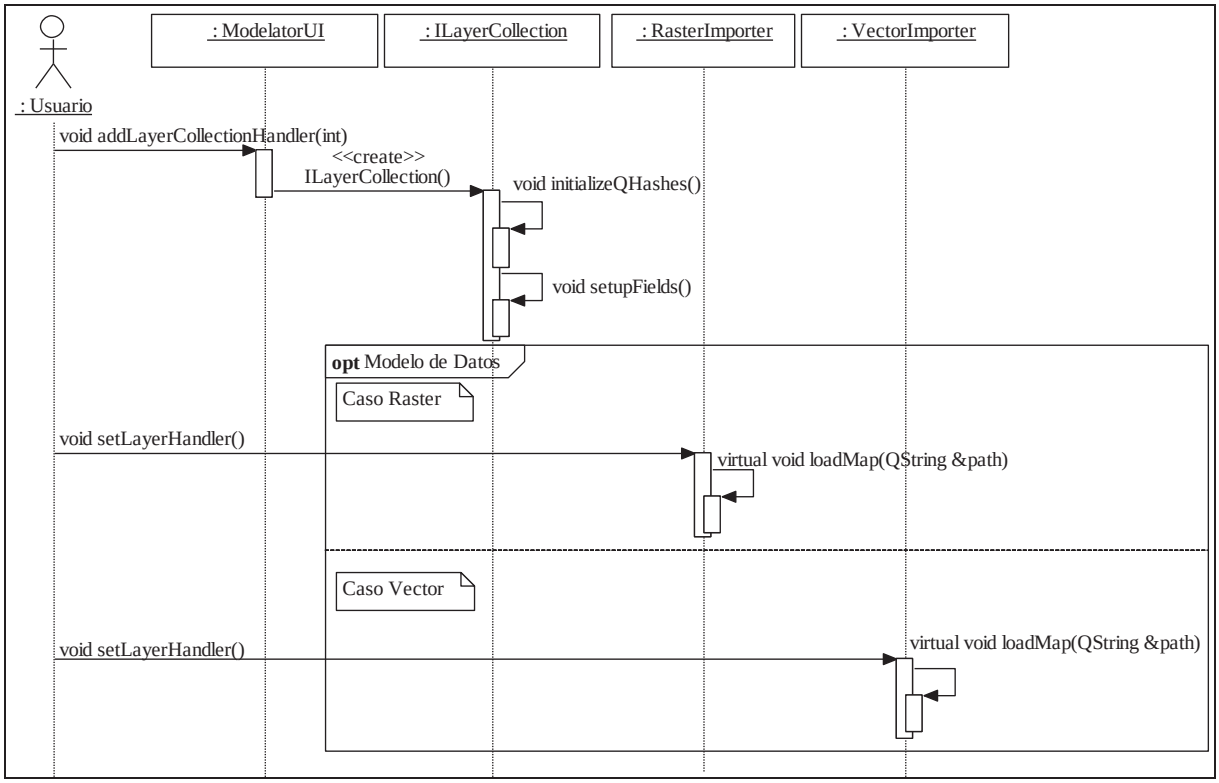


Figura 3.11: Diagrama de Secuencia Importar Capa

Estas son configuradas en el constructor de la clase que hereda a ILayerCollection. Tomando como ejemplo el TerrainLayerCollection estas se insertan de la siguiente forma:

```
01      rasterQHash->insert("Altura",NULL);
02      rasterQHash->insert("Textura",NULL);
03      vectorQHash->insert("Altura",NULL);
```

Posteriormente el ILayerCollection llama al metodo setupFields que se encarga de crear los QWidget correspondientes en base a las Tablas *Hash Raster* y Vectorial, que consiste en un botón. La funcionalidad del botón es la elección del mapa, creando un RasterImporter o un VectorImporter dependiendo si es un mapa *Raster* o Vectorial.

Una vez generados los botones si el usuario quiere editar o agregar una capa, en el caso de que esta sea *Raster* se obtendrá el directorio del archivo con el método setLayerHandler, con el directorio se cargara el mapa con loadMap que recibe como parámetro el directorio elegido y se instancia un QgsRasterLayer de QGIS usando GDAL para cargar el mapa Raster.

Ahora en el caso de que esta sea Vectorial se obtendrá el directorio del archivo con el método setLayerHandler, luego se asigna la capa con loadMap, el cual recibe como parámetro el directorio de la capa y se instancia un QgsVectorLayer de QGIS usando OGR para cargar el mapa Vectorial.

- **Diagrama de Secuencia Exportar Modelo**

El diagrama presentado en la Figura 3.12 muestra la exportación de un modelo, desde que el usuario presiona el botón exportar, hasta que se crea el IModelExporter.

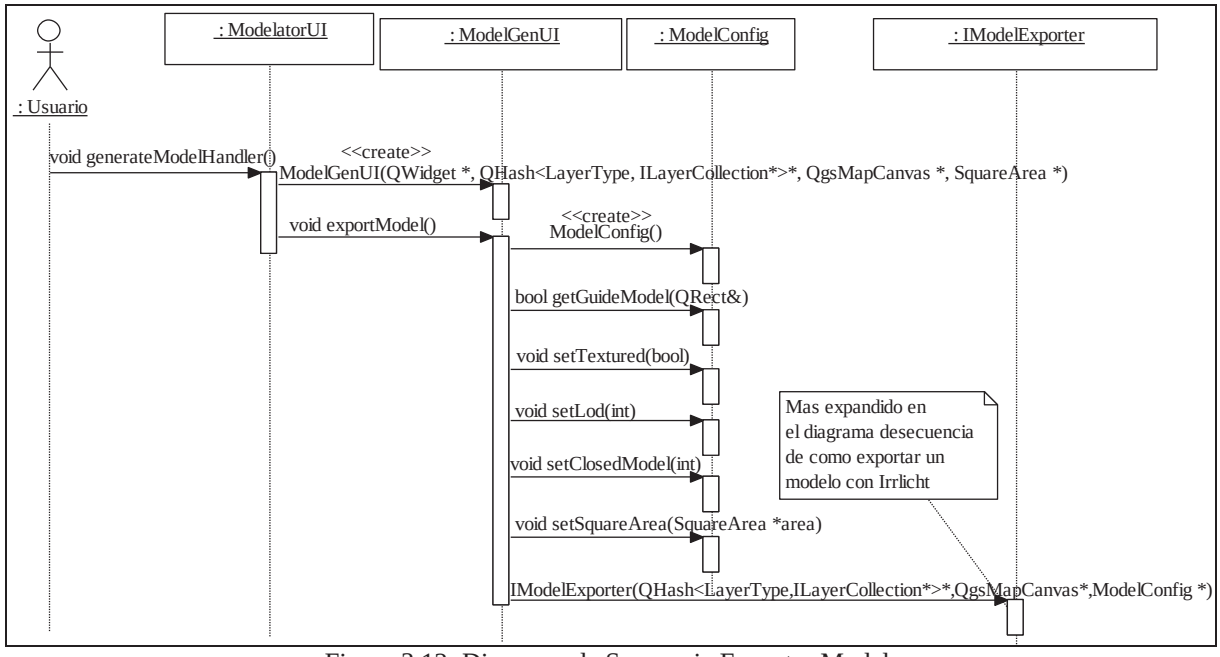


Figura 3.12: Diagrama de Secuencia Exportar Modelo

El usuario presiona el botón correspondiente a exportar un modelo, este lanza el *Handler* de ModelatorUI, el cual crea una instancia de ModelGenUI que recibe como parámetros el

conjunto de capas, el *Canvas* donde se dibuja y el SquareArea representando el área seleccionada por el usuario para generar el modelo 3D.

El ModelGenUI muestra una interfaz de configuración de modelo, el cual al finalizar ejecuta exportModel, que instancia un ModelConfig que almacena toda la configuración del modelo, asigna el área del mapa que se va a usar para la generación del terreno en el método setSquareArea, asigna variables de configuración general, como si se genera el modelo de guía (setGuideModel), configurar el suavisado del modelo (setSmooth), o sea que tan detallado se quiere el modelo (setNumberVertex), si se quiere el modelo cerrado o abierto (setClosedModel).

Con todo configurado se procede a crear una instancia de IModelExporter y se entrega la Tabla *Hash* con el conjunto de capas (ILayerCollection), además de esto se pasa la configuración del modelo creada y asignada anteriormente (ModelConfig). Para una visión más profunda de cómo el IModelExporter genera el modelo, ver el Diagrama de Secuencia Exportar Modelo con Irrlicht (Figura 3.13).

- **Diagrama de Secuencia Exportar Modelo con Irrlicht**

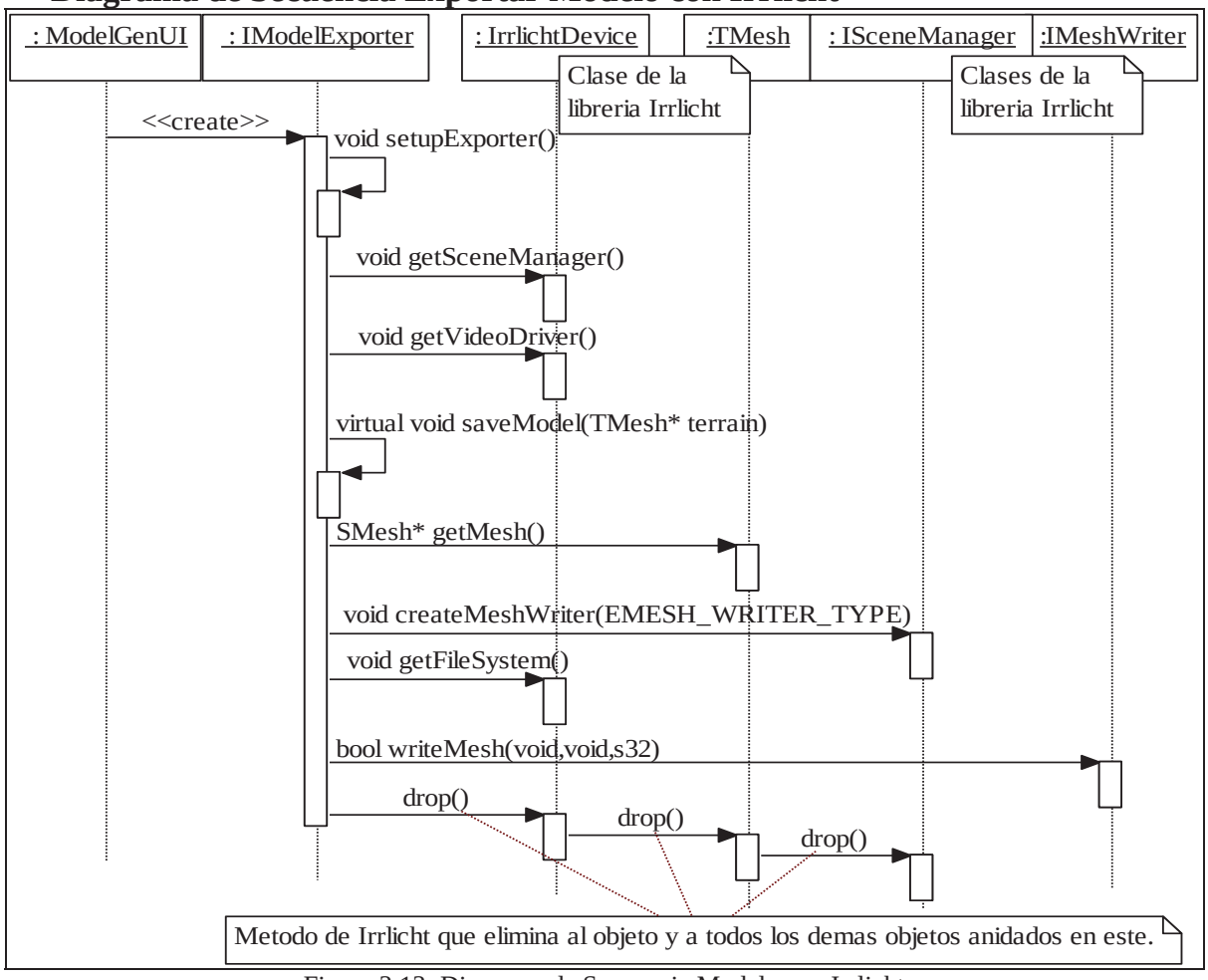


Figura 3.13: Diagrama de Secuencia Modelo con Irrlicht

Esta secuencia comienza en el Diagrama de Secuencia Exportar Modelo donde se crea una instancia de la clase IModelExporter con un QHash como parámetro de construcción, el cual contiene la información de todas las capas que han sido agregadas para ser adicionadas al modelo a exportar.

El siguiente paso es llamar al método setupExporter() el cual adapta las opciones del modelo al formato del exportador en uso. Una vez configurado el exportador se procede a obtener desde la biblioteca dinámica de Irrlicht una instancia del administrador de escena ISceneManager el cual maneja la creación del mapa de relieve (*heightmap*) al cual se le realizaran diferentes operaciones según los conjuntos de mapas que se quieran agregar.

Una vez terminadas las operaciones de creación y optimización del modelo, se obtiene el nivel de detalle exigido por el usuario para exportar el modelo con más o menos detalle según corresponda utilizando para esto el método getMesh(), luego se obtiene la instancia del exportador de mallas correspondiente al exportador escogido utilizando el método createMeshWriter(…). Se genera el archivo vació del modelo 3D previo uso del método getFileSystem() el cual retorna una instancia del manejador de sistemas de archivos, el que posteriormente realiza la creación del archivo.

Finalmente se utiliza el método writeMesh(…), el cual exporta el modelo 3D generado a lo largo de esta secuencia en el formato de archivos que corresponda al exportador escogido. Al finalizar la operación de exportación se llama al método drop() para limpiar las referencias e instancias creadas y/o utilizadas por Irrlicht.

3.8 Diagramas de Actividad

- **Diagrama de Actividad de Creación de un Conjunto de Capas**

El diagrama en la Figura 3.14 muestra el proceso de crear un ILayerCollection, en el diagrama se puede ver el caso de agregar un conjunto de capas, como en el sistema solo puede haber un conjunto de capas por cada tipo, se procede a comprobar si ya existe el tipo conjunto de capas nuevo, retornando en caso contrario.

Posteriormente se procede a configurar las Tablas *Hash* de capas *Raster* y Vectorial, y configurar la interfaz para la generación de los QWidgets en base a las Tablas *Hash Raster* y Vectorial configuradas anteriormente. Si las Tablas *Hash* están vacías se da un aviso que la clase que está instanciando no tiene ninguna Tabla de *Hash* configurada y retorna.

En caso que las Tablas de *Hash* estén configuradas y tengan elementos se procede a recorrer la Tabla *Hash* de capas *Raster* y a generar los respectivos QWidgets, que es el Botón y el Textbox para cambiar el mapa cargado, se procede a hacer lo mismo para la Tabla *Hash* de capas Vectoriales. Cuando se finalizó de recorrer la Tabla *Hash* de capas Vectoriales se procede a finalizar el método.

- **Diagramas de Actividad de Importar una Capa Raster o una Vectorial**

Los dos diagramas presentados en la Figura 3.15 y Figura 3.16 tienen características similares que explican cómo cargar una capa *Raster* y/o una capa Vectorial, en ambos

diagramas se obtiene la dirección del archivo a cargar y se comprueba que esta dirección sea válida, en caso incorrecto se retorna. Posteriormente se comprueba que sea una extensión valida, en el caso de los mapas *Raster* que sea una extensión soportada por las librerías implementadas en el sistema para cargar mapas *Raster*, como la librería implementada para mapas *Raster* es GDAL, se soportan los formatos de esta.

Así mismo es el caso de los mapas Vectoriales, en que se comprueba que sea una extensión de un mapa Vectorial válida para las librerías para cargar mapas Vectoriales del sistema, en este caso se ha implementado OGR, así que tiene que ser un formato soportado por OGR. Hasta este punto si todo va bien se procede a realizar la carga del mapa Vectorial o del mapa *Raster* dependiendo del caso, si esta carga falla, se liberan los recursos como el mapa creado y las clases creadas y se retorna, si no se carga el mapa *Raster* o Vectorial exitosamente y se guarda.

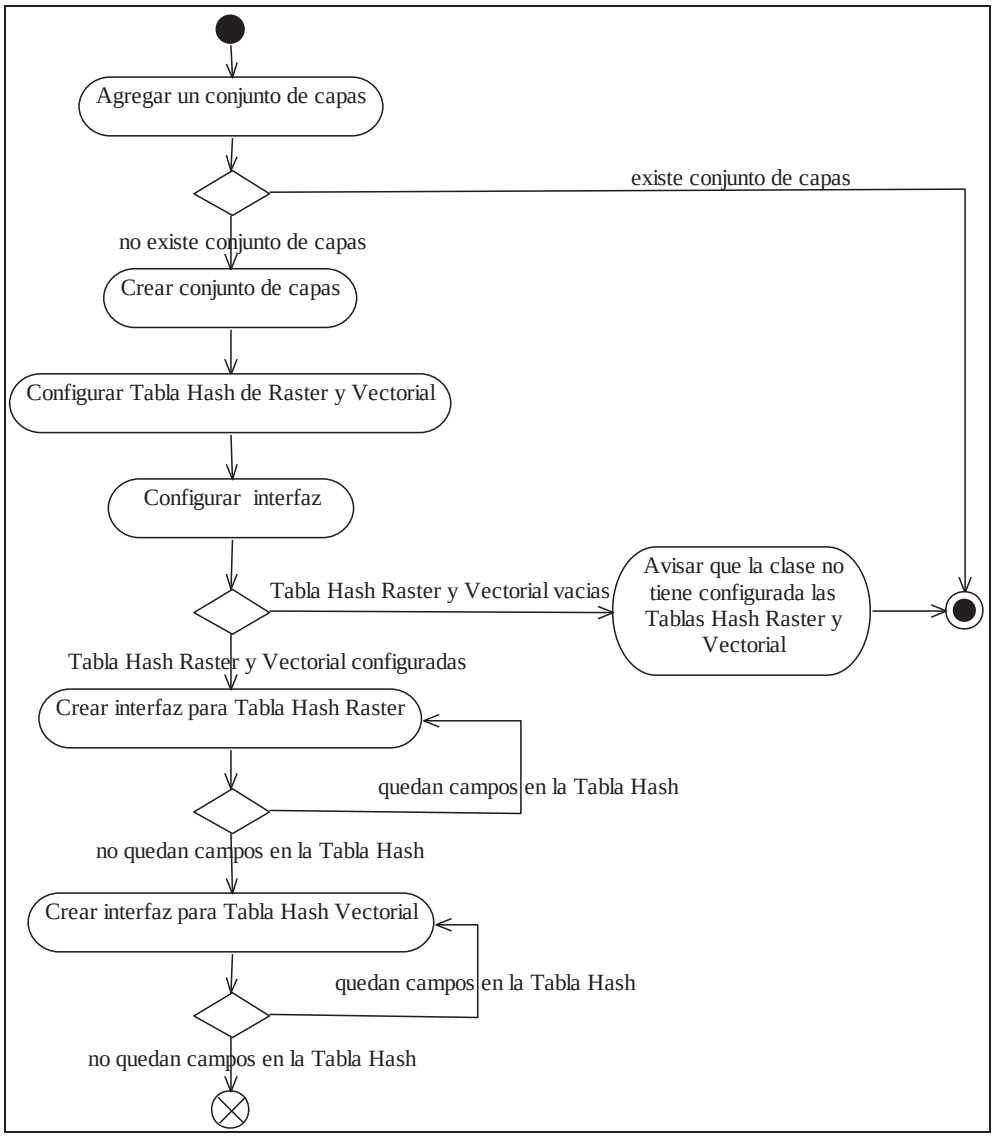


Figura 3.14: Diagrama de Actividad de la Creación de un Conjunto de Capas

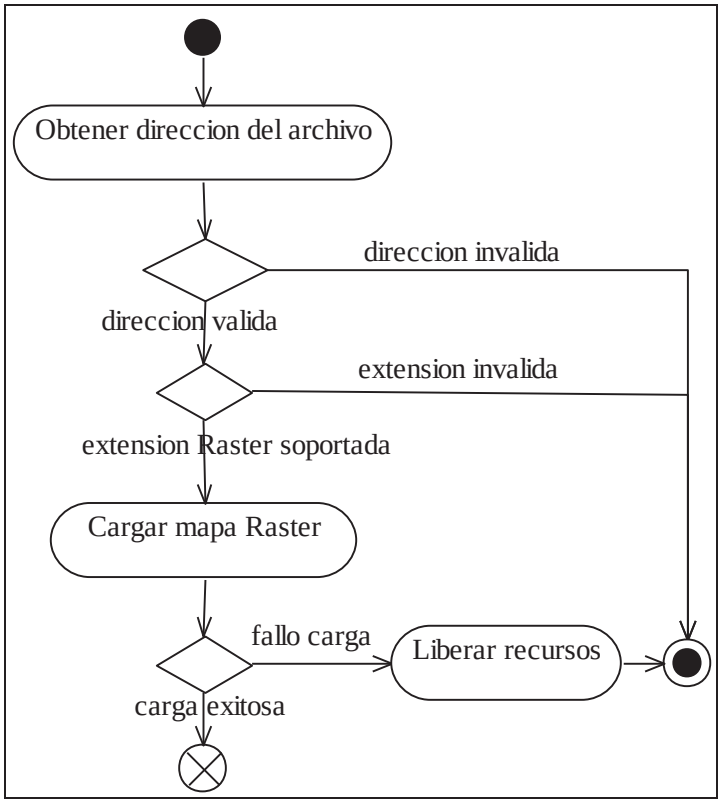


Figura 3.15: Diagrama de Actividad sobre Importe de una Capa Raster

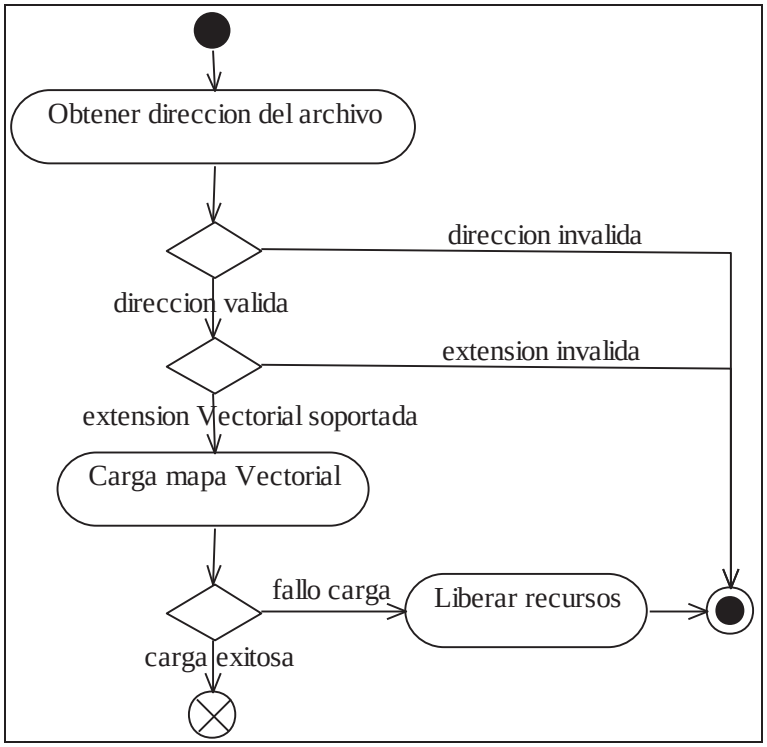


Figura 3.16: Diagrama de Actividad sobre Importe de una Capa Vectorial

- **Diagrama de Actividad de Exportar Modelo**

En el diagrama en la Figura 3.17 se presentan los distintos caminos a tomar a la hora de exportar un modelo.

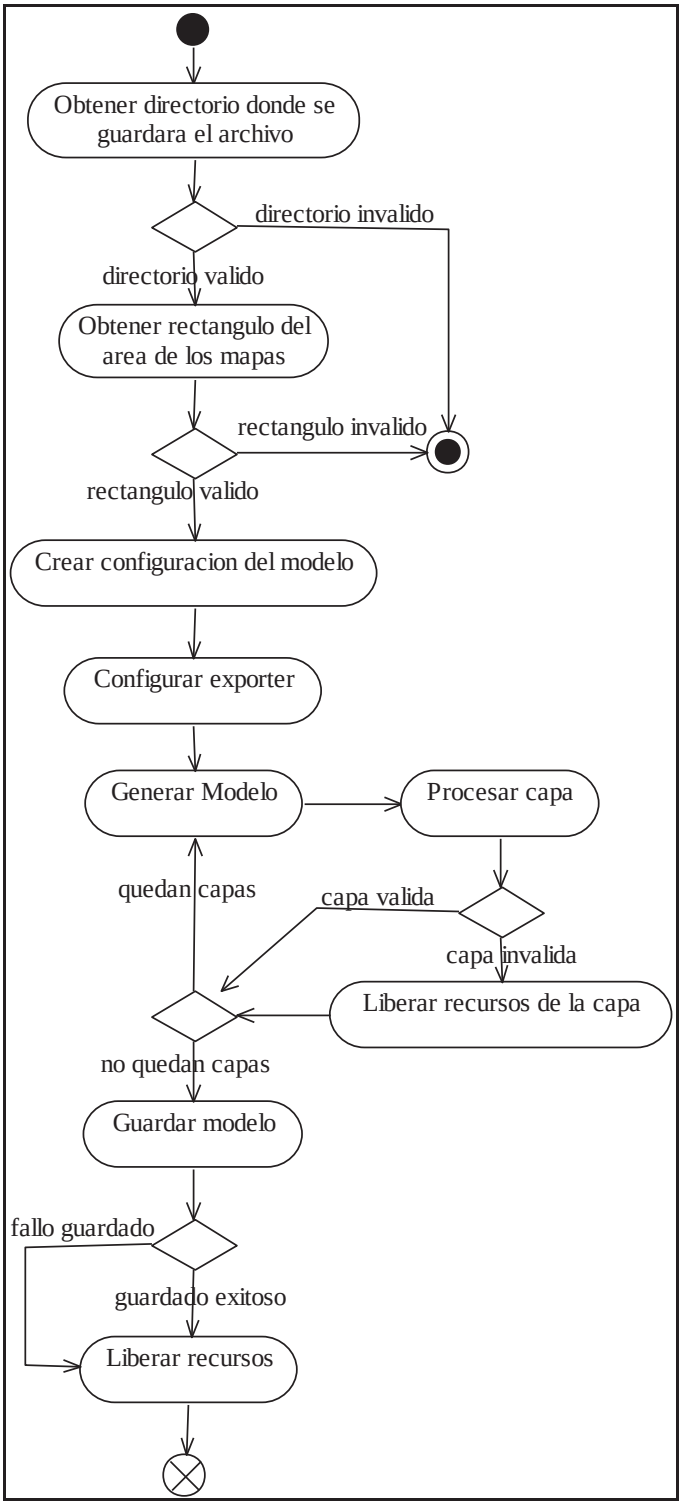


Figura 3.17: Diagrama de Actividad Sobre Exportar Modelo

En primer lugar se obtiene el directorio donde se quiere almacenar el archivo del modelo y se comprueba si este es válido, si es invalido retorna, sino se obtiene el rectángulo que indica el área del mapa que va a ser usada en la generación del modelo y se comprueba si esta área es válida, en caso de no serla, retorna.

Ahora se procede a crear la configuración del modelo, que es el área, y otras configuraciones generales del modelo indicadas por el usuario. Luego recibe esta configuración el *Exporter* elegido y comienza la generación del modelo.

En esta etapa se procesan todos los conjuntos de capas para generar los distintos elementos que conforman el modelo, en caso de encontrarse una capa inválida se liberan los recursos de esta, se desecha y se prosigue en la creación del modelo con lo que queda, hasta que ya no existen más capas. Por último se realiza el guardado del modelo ya creado, si falla o es exitoso el almacenamiento del modelo se liberan los recursos y retorna.

3.9 Plan de Pruebas

La necesidad de comprobar el correcto funcionamiento del sistema hace que sea imprescindible un plan de pruebas, con el cual se procederá a realizar una serie de ensayos que permitan obtener resultados correctos y erróneos con el fin de analizar el proceso de ejecución. Con este conjunto de pruebas será posible determinar si el software es erróneo sobre todo en casos extremos y particulares, o sea si estos fallos se producen por una mala implementación del programa o bien por un uso específico que realiza el usuario.

En el caso de la aplicación a desarrollar en este proyecto, se deben realizar pruebas intensivas en cuanto al procesamiento interno de los datos dada la precisión con la que se debe trabajar cuando se trata de mapas y sobre todo de la transformación desde un modelo 2D de datos a uno 3D. Para este propósito se realizaron pruebas de caja blanca para asegurar dentro de lo posible el correcto procesamiento de los datos.

Otro caso de importancia es el de la interacción con la aplicación mediante la interfaz de usuario, ya que esta interfaz y los datos que ingresen por ella son imprescindibles para llevar a cabo los objetivos del proyecto. Para este caso se realizaron pruebas de caja negra para asegurar que la aplicación sea capaz de trabajar con todos de los valores que puedan ser ingresados a través de la interfaz de usuario y también se debe verificar lo que la interfaz de usuario interpreta de los datos del sistema.

Por lo explicado anteriormente se da a entender que las pruebas a realizar son del tipo:

- **Pruebas de Caja Blanca**: se basa en el minucioso examen de los detalles procedimentales, es decir se comprueban los caminos lógicos del software.
- **Pruebas de Caja Negra**: diseñadas para considerar exclusivamente las entradas y salidas del sistema, es decir se centra principalmente en el análisis de la interfaz de usuario.

Para el diseño individual de cada prueba se especificará información que será documentada de la siguiente forma:

- El propósito de la prueba.
- Los pasos para la ejecución de la prueba.
- El resultado que se espera obtener.

Para el diseño general de las pruebas se considerará la entrada de los mapas al sistema desarrollado, como se administran estos dentro de la interfaz de usuario y cuál es el resultado de generación del modelo en relación a los datos que refleja la interfaz de usuario.

3.9.1 Pruebas de Caja Blanca.

También son conocidas como pruebas estructurales. En este tipo de prueba se evalúan los caminos lógicos del código y se usan las estructuras de control del diseño para obtener los casos de prueba, a través de estos puede comprobarse que todos los caminos lógicos se ejecuten por lo menos una vez.

Entre las pruebas de caja blanca se encuentran:

- **Pruebas de Segmentos:** este tipo de pruebas se enfoca en las partes del código que no tienen puntos de decisión, como el computador está obligado a ejecutarlas una tras otra, se puede decir que se han probado todos los segmentos o sentencias.

- **Pruebas de Ramas:** estos casos son un refinamiento de las pruebas de segmentos, ya que consiste en recorrer todas las posibles salidas de los puntos de decisión (*if else*) y basta con ejecutar una vez con éxito la condición para probarlas.

- **Pruebas de Ciclos o Bucles:** Los ciclos no son más que segmentos controlado por decisiones por lo que en teoría las pruebas de ramas bastarían, pero esto es solo en teoría ya que los ciclos podrían ser una fuente inagotable de errores. Ya que un ciclo se ejecuta un cierto número de veces; pero ese número debe ser muy preciso, y basta con que se ejecute una vez más o menos para que esto tenga consecuencias indeseables. Los ciclos *while* deben probarse con cero ejecuciones, una ejecución y más de una ejecución. En cambio un ciclo *for* basta con ejecutarlo solo una vez.

  Ya que las pruebas sobre el código o de caja blanca se realizaron a medida que se desarrollaba el software, donde se corrigieron los errores lógicos del sistema a medida que fueron apareciendo mediante la depuración del programa hecha con la IDE, aplicando estas pruebas cada vez que se agregó una nueva funcionalidad al sistema, se harán pruebas de caja negra donde se dará énfasis a la verificación de los datos de entrada y salida con el afán de comprobar que el sistema funcione correctamente y que las salidas correspondan con lo que se espera.

3.9.2 Pruebas de Caja Negra.

También conocidas como pruebas funcionales o de entrada y salida. Las pruebas de caja negra se centran en lo que se espera de las funcionalidades del sistema, es decir, intentan encontrar casos en que alguna funcionalidad del sistema no cumple su especificación. Estas

están centradas principalmente en el análisis de la interfaz de usuario (como el teclado, pantalla o ficheros), en donde un probador se limita suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el software internamente. Se realizaron las siguientes pruebas de caja negra:

Tabla 3.3: Prueba 1 de Caja Negra.

Prueba 1.	
Descripción de la prueba:	Agregar un conjunto de caminos, cargar un mapa vectorial de caminos, sin agregar un mapa de elevación de terreno y exportarlo.
Propósito de la prueba:	Comprobara el comportamiento del sistema al faltarle datos.
Pasos para la ejecución:	1. Agregar un conjunto de caminos. 2. Agregar un mapa vectorial de caminos. 3. Exportar modelo como OBJ. 4. Importarlo en Blender.
Resultados esperados:	1. Generar un modelo que consiste en un plano con los caminos dibujadas en la textura. 2. Modelo visualizado en Blender correctamente.

Tabla 3.4: Prueba 2 de Caja Negra.

Prueba 2.	
Descripción de la prueba:	Agregar un conjunto de terreno, cargar un mapa Vectorial, uno Raster y exportarlo.
Propósito de la prueba:	Comprobar comportamiento del sistema al cargar la capa Raster y Vectorial de altura.
Pasos para la ejecución:	1. Agregar un conjunto de terreno. 2. Agregar un mapa Vectorial de altura. 3. Agregar un mapa Raster de altura. 4. Exportar modelo como OBJ. 5. Importarlo en Blender.
Resultados esperados:	1. Generar el modelo con la información del mapa vectorial. 2. Modelo visualizado en Blender correctamente.

Tabla 3.5: Prueba 3 de Caja Negra.

Prueba 3.	
Descripción de la prueba:	Agregar todos los conjuntos de capas soportados por el sistema, cargar todos los mapas para cada capa, exportar modelo con textura de guía.
Propósito de la prueba:	Generar el modelo con la textura guía acorde a la información proporcionada por las capas.
Pasos para la ejecución:	1. Agregar todos los conjuntos de capas disponibles. 2. Cargar todos los mapas requeridos por cada conjunto de capa. 3. Exportar modelo con textura de guía como OBJ. 4. Importar modelo con textura guía en Blender
Resultados esperados:	1. Generar un modelo y un modelo de guía con la información de todas las características expuestas en los mapas cargados. 2. Modelo con textura guía visualizado correctamente en Blender.

Tabla 3.6: Prueba 4 de Caja Negra.

Prueba 4.	
Descripción de la prueba:	Agregar cualquier conjunto de terreno, cargar un mapa vectorial de altura y seleccionar un área vacía.
Propósito de la prueba:	Comprobar cómo se comporta el sistema al seleccionar un área sin datos de un mapa.
Pasos para la ejecución:	1. Agregar un conjunto de terreno al sistema. 2. Cargar un mapa de terreno. 3. Seleccionar un área vacía (no haya mapa dibujado) en el mapa. 4. Exportar modelo como OBJ. 5. Importar modelo en Blender.
Resultados esperados:	1. Generar un modelo de un área plana, sin elevación. 2. Modelo visualizado en Blender.

Tabla 3.7: Prueba 5 de Caja Negra.

Prueba 5.	
Descripción de la prueba:	Importar mapas con distintas proyecciones y comprobar si concuerdan en la visualización.
Propósito de la prueba:	Comprobar cómo se comporta el sistema al importar mapas de distintas proyecciones.
Pasos para la ejecución:	1. Agregar un mapa con un sistema de proyección A. 2. Agregar un mapa con un sistema de proyección B.
Resultados esperados:	1. Los mapas se posicionan correctamente en la visualización.

3.9.3 Resultados de las Pruebas.

Se obtuvieron los siguientes resultados al realizar las pruebas:

- **Prueba 1**: Agregar un conjunto de caminos, cargar un mapa vectorial de caminos, sin agregar un mapa de elevación de terreno y exportarlo.

Se agregó un conjunto de caminos, con su respectiva capa Vectorial y se procedió a importarlo en Blender. Después de realizar todos los procedimientos de la prueba se obtuvo un modelo con las calles posicionadas correctamente en un modelo 3D sin altura. Esto se puede ver en la Figura C.1, en la esquina superior izquierda el mapa cargado en el software, en la esquina superior derecha la textura y abajo el modelo 3D generado cargado en Blender.

- **Prueba 2**: Agregar un conjunto de terreno, cargar un mapa Vectorial, uno Raster y exportarlo.

Se agregó el conjunto de terreno, con su respectiva capa Vectorial y Raster y se procedió a importarlo en Blender. Al realizar los procedimientos de la prueba se obtuvo un modelo 3D con elevación y textura creadas en base a la información de elevación provista por estas capas. Esto se puede ver en la Figura C.2, en la esquina superior izquierda los mapas cargados en el software, en la esquina superior derecha la textura generada y abajo el modelo 3D generado cargado en Blender.

- **Prueba 3**: Agregar todos los conjuntos de capas soportados por el sistema, cargar todos los mapas para cada capa, exportar modelo con textura de guía.

Se agregaron todos los conjuntos con mapas cargados y se procedió a importarlo en Blender. Al realizar los procedimientos de la prueba se obtuvo un modelo 3D con elevación y textura guía creadas en base a la información de los mapas provistos. Como se ve en la Figura C.3: en la esquina superior izquierda los mapas cargados en el software; en la esquina superior derecha la textura guía generada, donde se marcan con un circulo verde la posición de los árboles, con un circulo morado la posición de los edificios, con una línea naranja las ferrovías y los caminos en gris; abajo el modelo 3D generado cargado en Blender.

Prueba 4: Agregar cualquier conjunto de terreno, cargar un mapa Vectorial de altura y seleccionar un área vacía.

Se agregaron un conjunto de terreno con una capa Raster de altura, se seleccionó un área vacía y se procedió a importarlo en Blender. Al realizar los procedimientos de la prueba se obtuvo un modelo 3D plano con una textura de desierto. Como se aprecia en la Figura C.4, arriba el área seleccionada y abajo el modelo 3D generado cargado en Blender.

Prueba 5: Importar mapas con distintas proyecciones y comprobar si concuerdan en la visualización.

En las pruebas anteriores se han importado mapas con distintas proyecciones y han concordado en la visualización, los mapas vectoriales tienen la proyección "Hito XVIII 1963 / UTM zone 19S" y el mapa Raster de altura cargado tiene la proyección "WGS 84".

3.10 Generación de mapas para Unreal Engine 3.

En esta sección se describe como se realizó la demo del sistema utilizando Unreal Engine 3 en base al modelo 3D generado con los datos entregados por distintos mapas. Para esto se procedió a cargar en la aplicación distintos mapas que contenían datos de altura, árboles, caminos y edificios, y se seleccionó un área de 256 pixeles equivalente a 1581 metros cuadrados.

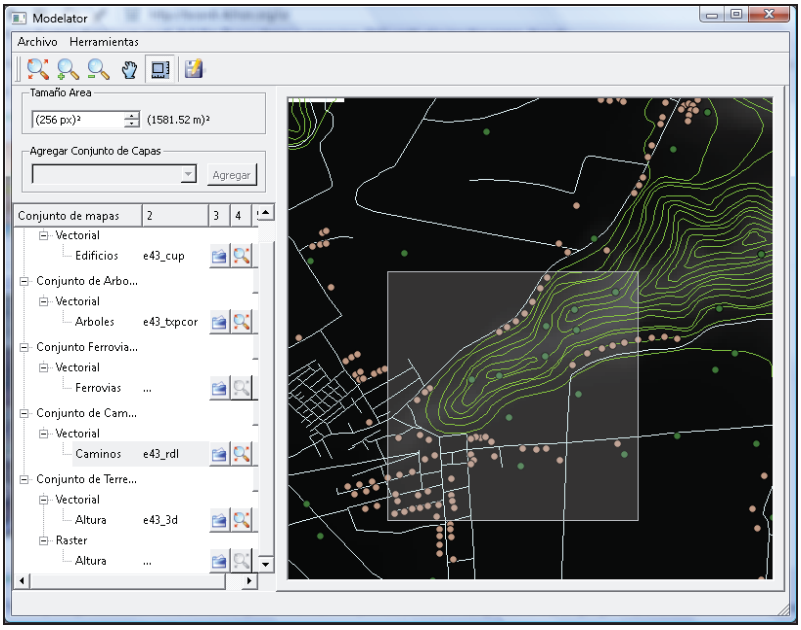


Figura 3.18: Selección del área para generar el modelo 3D.

Para crear el modelo 3D se debe seleccionar las propiedades del modelo a generar con el sistema, como la cantidad de vértices, la escala del modelo y malla abierta o cerrada. Para Unreal Engine 3 se deben proveer de preferencia modelos cerrados para usarlos como terreno, también se genera la textura a usar en el modelo en base a la altura del terreno, y se puede generar una textura guía con los distintos puntos que representan las ubicaciones de los árboles, edificios o caminos.

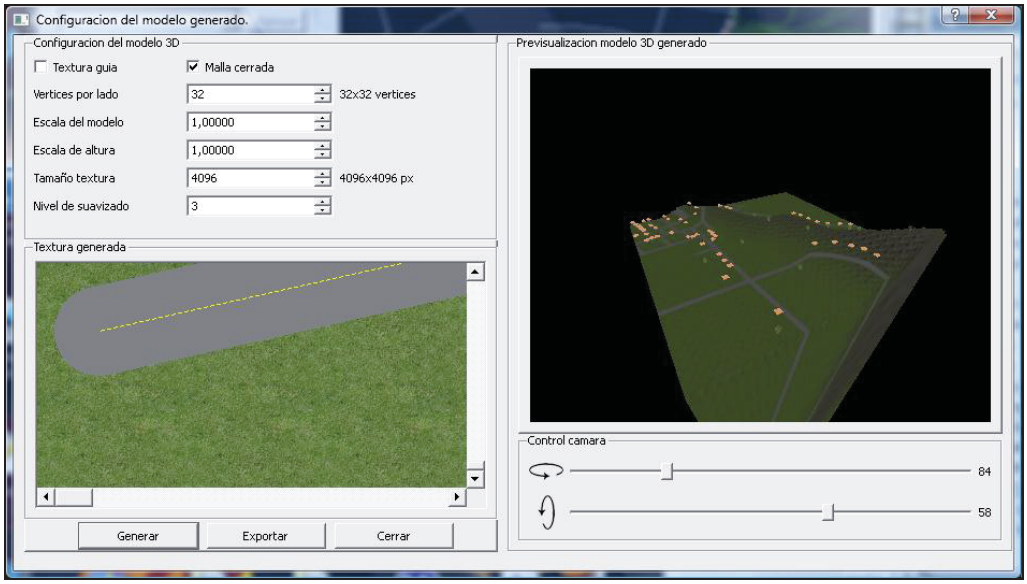


Figura 3.19: Generación del modelo 3D.

Una vez generado el modelo 3D, se exporto con el formato OBJ y dicho modelo se importó a Blender 2.54. A su vez en Blender se exporto el modelo utilizando un *plugin* para convertir la malla al formato ASE el cual es compatible con el editor de Unreal Engine 3.

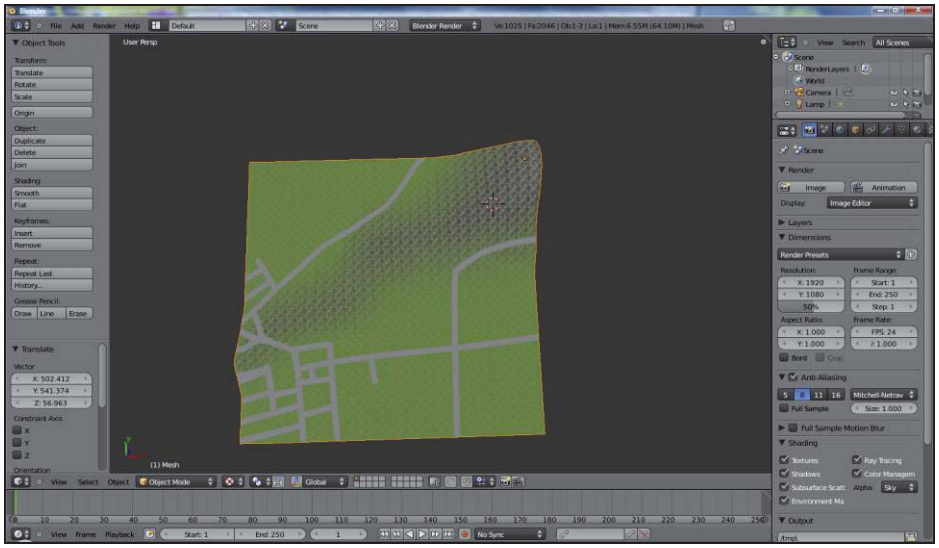


Figura 3.20: Modelo importado en Blender

En el editor de Unreal Engine 3 se importó el modelo 3D en formato ASE, el cual se agregó a la escena como un *brush* sólido. Para la textura del terreno en Unreal Engine 3 es necesario crear un nuevo paquete del tipo material en el *Content Browser*, se cargó la textura como *diffuse* y se aplicó el material al *brush* agregado a la escena. En este paso se usó las siguientes propiedades al aplicar el material: escala U y V en 512, alineamiento planar, rotación 180°, pan U en 70 y pan V en 55. En base a la textura de guía se procede a posicionar los árboles representados por puntos verdes, los edificios representados por puntos morados y los caminos.

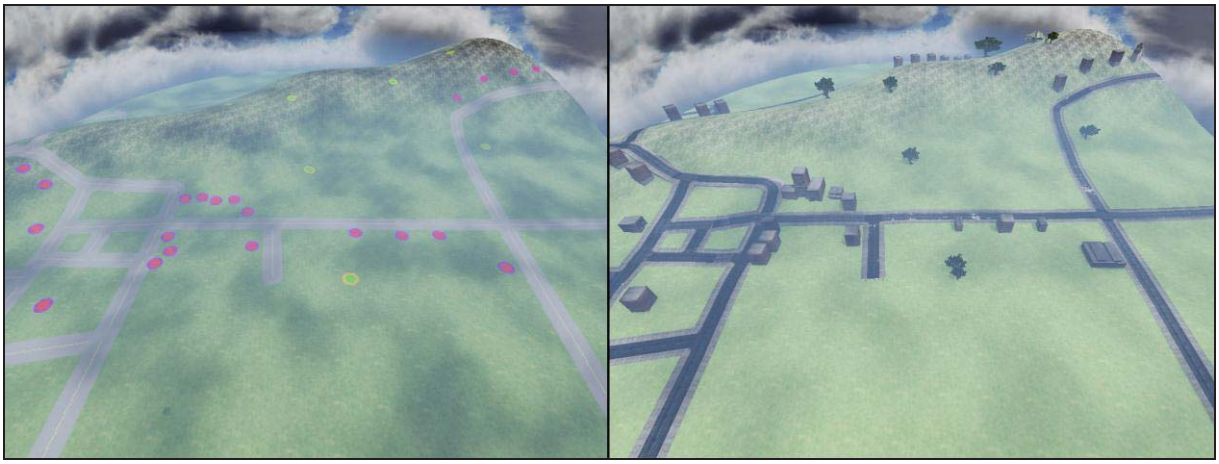


Figura 3.21: modelo 3d en Unreal Engine 3 con textura guía

Una vez agregado los modelos en Unreal Engine 3 se procede a configurar las distintas propiedades del motor como fuentes de luz, puntos de inicio y por último se cocina para generar el mapa para producción, obteniendo como producto un mapa listo para ser utilizado en Unreal Tournament 3.

Figura 3.22: mapa en Unreal Engine 3 en base al modelo 3D

4 Conclusiones

Este proyecto se presentó como una idea innovadora para la utilización de los datos utilizados por los SIG, en la creación de modelos 3D, y su aplicación en otras áreas, ya que obtener un mapa como base desde un sistema de información geográfica facilita el trabajo para el desarrollo de una aplicaciones con entorno 3D. Por ejemplo en el área de simulación, con datos verídicos y actualizados.

Durante la primera fase de investigación, se hizo notar la escasez de *Papers* acerca de proyectos relacionados, por esta razón que la mayor parte de la información se basó en otras fuentes bibliográficas y en información encontrada en proyectos comerciales y *Open Source*, como referencia para la elaboración del diseño de este sistema.

Se formó un marco teórico inicial para el proyecto en cuanto a los SIG, donde se estudió las distintas tecnologías existentes para ser usadas en la identificación de los datos útiles para la simulación 3D. A partir de esto se realizó el diseño y posterior desarrollo de un sistema de software, el cual es capaz de utilizar distintos mapas digitales para creación de modelos 3D y la generación de texturas básicas para estos. Para el desarrollo se utilizaron las librerías de *Quantum GIS* para la administración de los mapas digitales y la información contenida en estos, e *Irrlicht Engine* para la generación y exportación del modelo.

El modelo del terreno puede ser utilizado por aplicaciones de juegos y de simulación 3D, simplemente abriéndolo con una herramienta de diseño como *Blender 3D* y su posterior exportación en el formato de modelos compatible con el juego o sistema de simulación a utilizar, como el formato ASE que corresponde a lo *meshes* soportados por *Unreal Engine 3*, en el caso de este motor la textura generada por el software sirve como marca para posicionar los árboles, caminos y edificaciones provistos por este motor de videojuegos.

Este proyecto está hecho de tal forma que pueda ser retomado por otros alumnos para utilizarlo en sus proyectos o para extender sus funcionalidades como agregar formatos de archivos para la exportación de modelos, como base para un sistema de simulación o videojuego utilizando el código de Irrlicht implementado en el software, o hacer extensible la creación de las distintas capas generadas implementando algún lenguaje de *scripting* o dentro del mismo código implementar alguna forma genérica de cargar tipos similares de capas con sus respectivos modelos, como por ejemplo la capa de árboles y edificios siguen el mismo procedimiento para ser generados, se puede implementar una asociación entre una capa de puntos y un modelo para hacer más representaciones de este tipo de capas.

Referencias

[gvSIG, 11] Información sobre gvSIG en la página del manual de usuario de gvSIG http://www.gvsig.org/web/docusr/acceso-editores/manual-de-usuario/introduccion-a-gvsig/copy_of_queesgvsig/. Revisada por última vez en Junio del 2011.

[ESRI, 10] Información sobre la extension de Arcview, desarrollada por ARCGis conocida como 3D Analyst en la página http://www.esri.com/software/arcgis/extensions/3danalyst/key-features.html. Revisada por última vez en Septiembre del 2010.

[Autodesk, 11] *3ds Max - 3D Modeling, Animation, and Rendering Features and Demos – Autodesk*. http://usa.autodesk.com/3ds-max/features/. Revisada por última vez en Junio del 2011.

[Blender, 11] *Blender 3D Home*. http://www.blender.org/. Revisada por última vez en Junio del 2011.

[Unreal Engine, 11] Información sobre las características de unreal engine en la página http://www.unrealengine.com/features. Revisada por última vez en Junio del 2011.

[DeMers, 09] DeMers, Michael**.** *GIS For Dummies.*Wiley Publishing, 2009.

[Khronos, 10] Información referente a la especificación del formato de archivos COLLADA a través de la página http://www.khronos.org/collada/ mediante el enlace "*COLLADA 1.4.1 Specification* (*Second edition*)". Revisada por última vez en Junio del 2010.

[Reddy, 11] Información referente a la especificación del formato de archivos OBJ a través de la página http://www.martinreddy.net/gfx/3d/OBJ.spec. Revisada por última vez en Junio del 2011.

[Funkhouser, 02] Funkhouser, Thomas. Polygonal Meshes. Princeton University, 2002.

[GDAL, 11] Información sobre GDAL y OGR en la página oficial http://www.gdal.org/. Revisada por última vez en Junio del 2011.

[Goizueta, 05] Goizueta, Javier. *Sistemas de Referencia Geodésicos. Proyecciones Cartográficas*, 2005.

[Google, 11] *Google SketchUp - 3D modeling for everyone*. http://sketchup.google.com/. Revisada por última vez en Junio del 2011.

[gvSIG, 10] gvSIG Association. *gvSIG Desktop 3D Extension Manual*. gvSIG Association, 2010.

[Harvey, 08] Harvey, Francis. *A Primer of GIS - Fundamental Geographic and Cartographic Concepts*. The Guilford Press, 2008.

[Hearn et al, 96] Hearn, Donald y Pauline, Baker. *Computer Graphics, C Version (2nd Edition).* Prentice Hall, 1996.

[Helton et al., 06] Helton Nogueira Uchôa1, Maurício Carvalho Mathias de Paulo, Luiz Carlos Teixeira Coelho Filho, Paulo Roberto Ferreira. *The Open 3D GIS Project - A Free Tool to Enable 3D Geographic Systems on the Web.* OpenGeo, Workshop de Software Libre, Brasil, 2006.

[Hyam, 02] Hyam, Danny. *The GIS files.* Ordnance Survey, 2002.

[ERDAS, 11] Detalles del producto IMAGINE VirtuaGIS en la página http://www.erdas.com/products/ERDASIMAGINE/IMAGINEVirtualGIS/Details.aspx. Revisada por última vez en Junio del 2011.

[GRASS, 11] Información sobre GRASS en la página http://grass.fbk.eu/intro/general.php. Revisada por última vez en Junio del 2011.

[Intergraph, 11] Información sobre la familia de productos de Intergraph Geomedia en la página http://www.intergraph.com/sgi/products/productFamily.aspx?family=10&country=. Revisada por última vez en Junio del 2011.

[Irrlicht, 11] *Irrlicht Engine - A free open source 3d engine*. http://irrlicht.sourceforge.net/. Revisada por última vez en Junio del 2011.

[Jacobson et al., 99] Jacobson, Ivar; Booch, Grady y Rumbaugh, James. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.

[James, 04] James Westervelt. *GRASS Roots.* FOSS/GRASS Users Conference, Septiembre 2004.

[Jason et al., 04] Jason Busby, Zak Parrish, Jeff Wilson. *Mastering Unreal Technology, Volume I.* Sams Publishing, Julio del 2009.

[jMonkeyEngine, 11] *jMonkeyEngine – Serious Monkeys. Serious Engine*. http://jmonkeyengine.org/. Revisada por última vez en Junio del 2011.

[Microsoft, 11] Información sobre las bases de datos espaciales de Microsoft SQL Server 2008 en la página http://www.microsoft.com/sqlserver/2008/en/us/spatial-data.aspx. Revisada por última vez en Junio del 2011.

[PENNSTATE, 10] The UTM Grid and Transverse Mercator Projection. Información sobre proyecciones geográficas en la página https://www.e-education.psu.edu/natureofgeoinfo/c2_p22.html. Revisada por última vez en Septiembre de 2010.

[OGRE, 11] *OGRE - Open Source 3D Graphics Engine.* http://www.ogre3d.org/. Revisada por última vez en Junio del 2011.

[Opengeo, 94] Opengeo Consultoria de Informática LTDA, Paper e información sobre aplicacion SIG 3D. Información sobre la aplicacion en la página http://www.opengeo.com.br/. Revisada por última vez en Marzo del 2010.

[Oracle, 11] Información sobre las bases de datos espaciales de Oracle en la página http://www.oracle.com/us/products/database/options/spatial/index.html. Revisada por última vez en Junio del 2011.

[Pilouk, 07] Pilouk, Alias Abdul-Rahman Morakot. *Spatial Data Modelling for 3D GIS.* Springer, 2007.

[PostGIS, 11] Información sobre PostGIS y su soporte para archivos Raster en la página oficial de PostGIS http://postgis.refractions.net/. Revisada por última vez en Junio del 2011.

[OSGeo, 10] Pagina oficial del proyecto Quantum GIS http://www.qgis.org/en/, con información para desarrolladores y usuarios. Revisada por última vez en Septiembre del 2010.

[Reed, 08] Reed, Aaron. *Learning XNA 3.0.* O'Reilly Media, 2008.

[Sanchez et al, 00] Sanchez, Julio y Canton, Maria. *DirectX 3D Graphics Programming Bible.* Hungry Minds, 2000.

[Sarría, 06] Francisco Alonso Sarría. Sistemas de Información Geográfica, 2006.

[ESRI, 11] Información sobre ARCGIS en la página http://www.esri.com/software/arcgis/index.html. Revisada por última vez en Junio del 2011.

[SIRGAS, 10] Definición de SIRGAS e información adicional en las páginas http://www.sirgas.org/ y http://www.cartografia.cl/. Revisadas por última vez en Junio del 2010.

[Steve et al., 00] Steve Streeting, Framework 3D. Información sobre el framework en la página http://www.ogre3d.org/about. Revisada por última vez en Abril del 2010.

[Ambler, 11] Ambler, Scott W. *The Agile Unified Process (AUP) Home Page*. http://www.ambysoft.com/unifiedprocess/agileUP.html. Revisada por última vez en Junio del 2011.

[DarkMod, 10] *The Dark Wiki, Documenting The Dark mod and DarkRadiant.* http://modetwo.net/darkmod/wiki/index.php?title=DrVertexBlend_%28tutorial%29. Revisada por última vez en septiembre de 2010.

[The Game Creators, 99] *The Game Creators, Framework 3D.* Información sobre el *framework* en la página http://www.thegamecreators.com/?m=view_product&id=2128. Revisada por última vez en Abril de 2010.

[SOEST, 10] School of Ocean and Earth Science and Technology, *Universal Transverse Mercator (UTM) projection (-Ju -JU).* http://gmt.soest.hawaii.edu/gmt/doc/gmt/html/GMT_Docs/node106.html. Revisada por última vez en Septiembre del 2010.

[Panda3D, 08] *The Panda3D Development Team, Framework 3D.* Información sobre el framework en la página http://www.panda3d.org/. Revisada por última vez en Abril del 2010.

[Uchoa et al, 06] Uchoa, Helton Nogueira, y otros. *The Open 3D GIS Project – A Free Tool to Enable 3D.* OpenGEO, 2006.

[Epic Games, 11] Información sobre Unreal Development Kit en la página oficial http://www.udk.com/. Revisada por última vez en Junio del 2011.

[Wright et al, 04] Wright, Richard y Lipchak, Benjamin. *OpenGL SuperBible (3rd Edition).* Sams, 2004.

Anexos

A: Glosario de Términos

Driver: Interfaz que permite a los sistemas de software comunicarse con el hardware.

Framework: Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos.

Handheld: Describe a un ordenador portátil para diversas aplicaciones, que puede ser llevado a cualquier parte mientras se utiliza.

Open Source: Es el término con el que se conoce al software distribuido y desarrollado libremente.

Plugin: Es una pieza de software que agrega funcionalidades y características a un programa.

Renderizado: Es el proceso de dibujar un objeto por pantalla.

B: Lista de Abreviaturas

SIG: Sistema de Información Geográfica.
GIS: Geographic Information System.
TIN: Triangulated Irregular Network.
Lat: Latitud
Long: Longitud

C: Resultados de las Pruebas

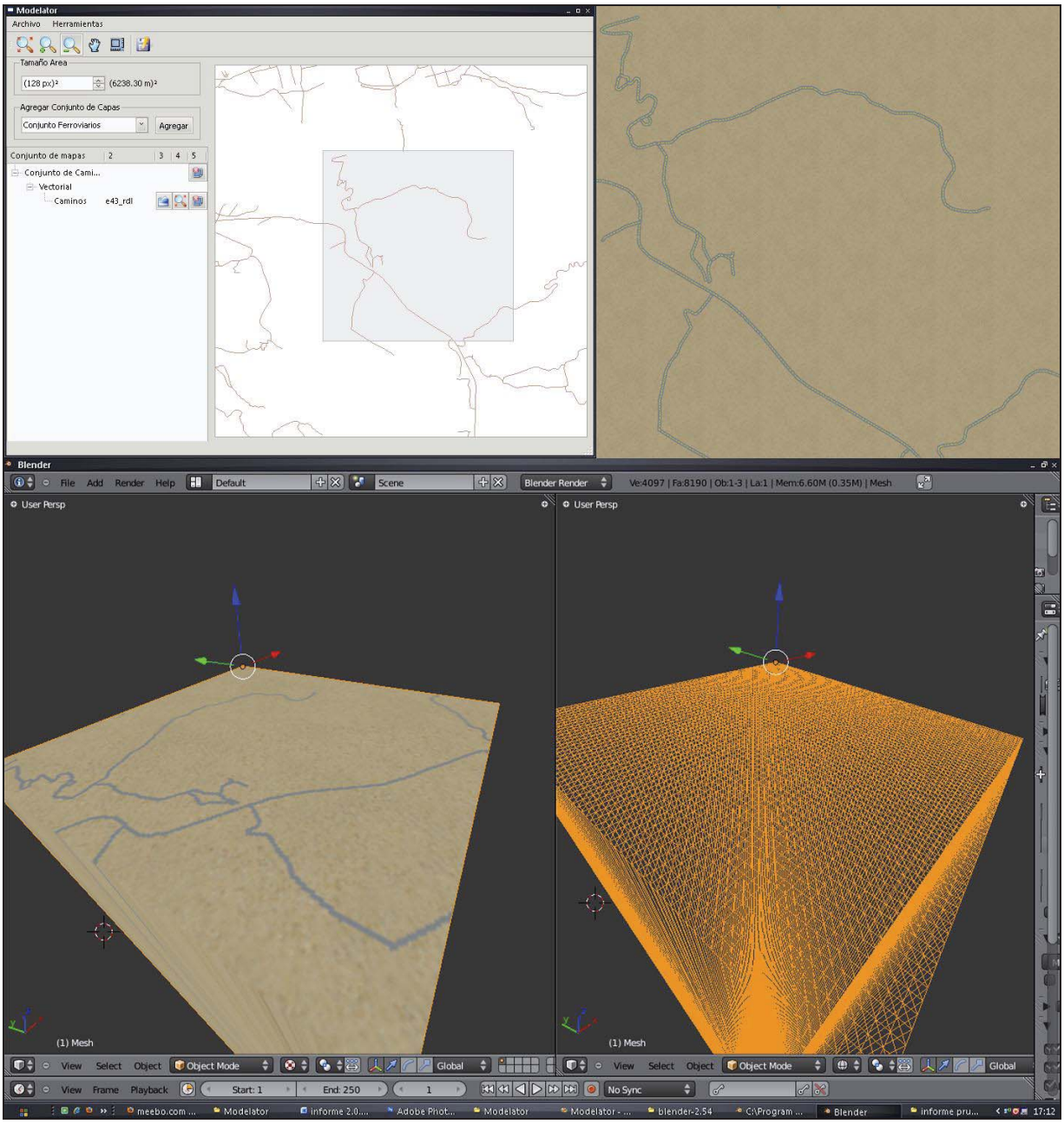


Figura C.1: Resultado prueba 1.

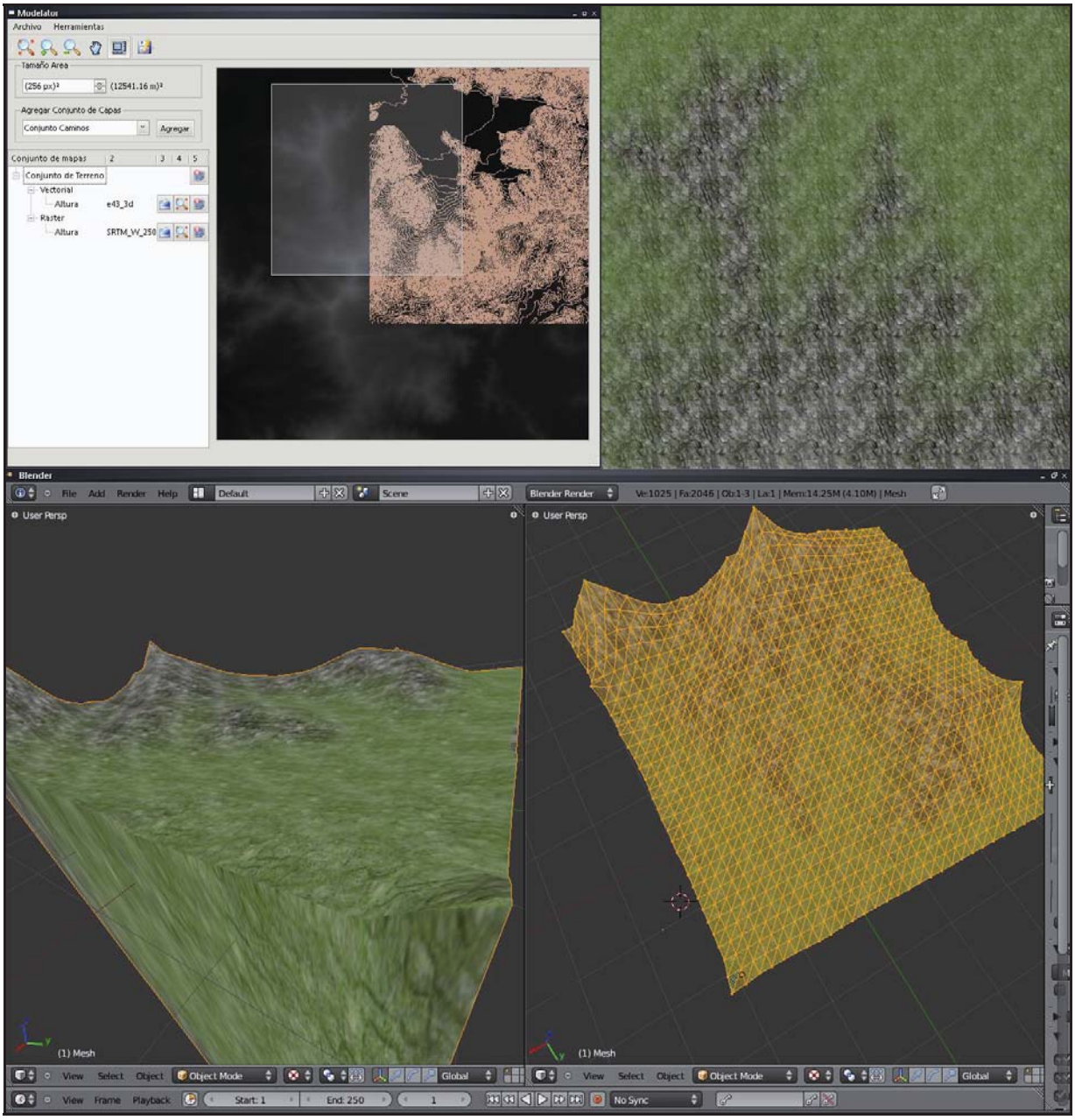


Figura C.2: Resultado prueba 2.

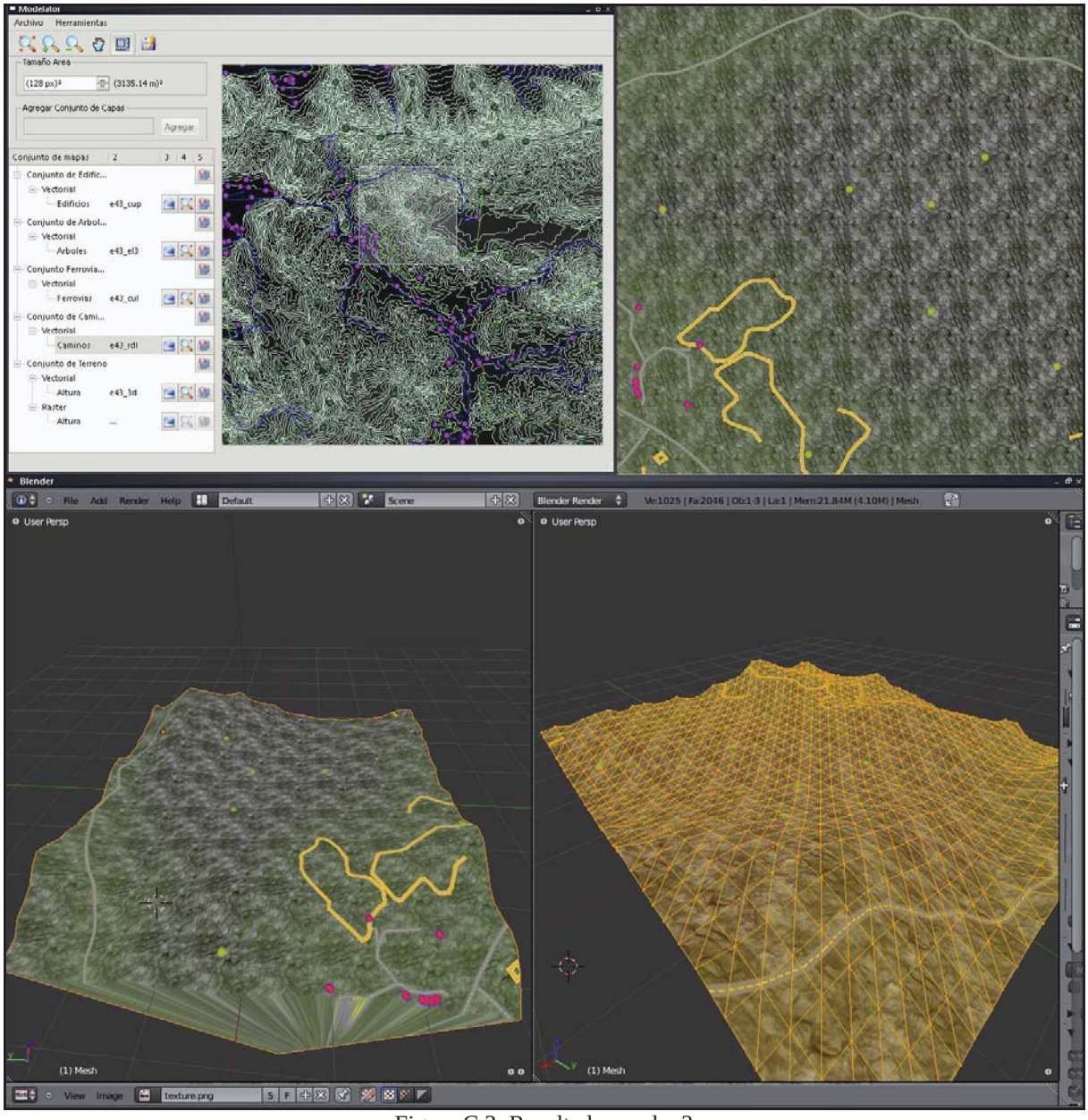


Figura C.3: Resultado prueba 3.

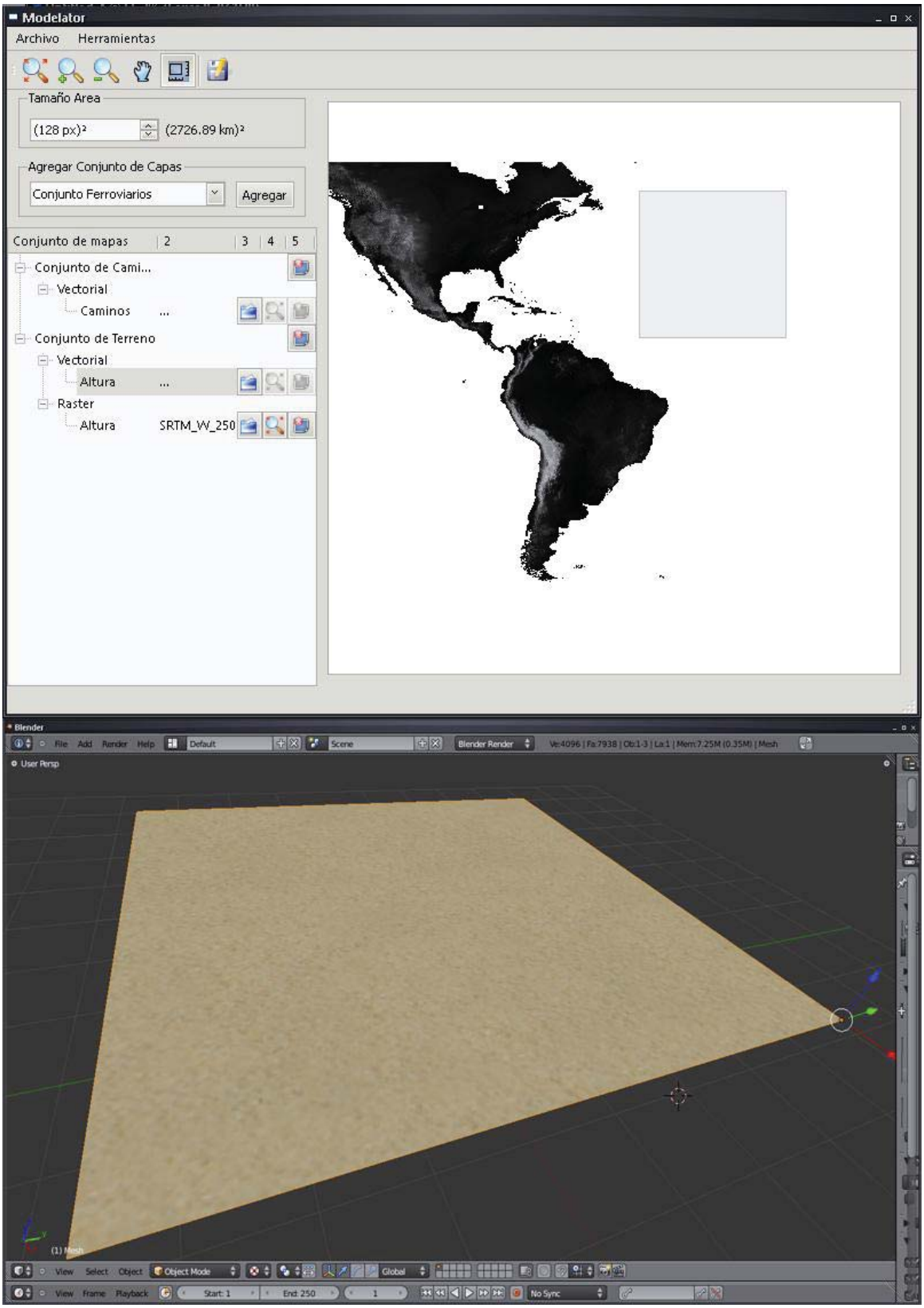


Figura C.4: Resultado prueba 4.