UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE UNA APLICACIÓN PARA VISUALIZACIÓN, CREACIÓN Y MODIFICACIÓN DE MAPAS PALEOGEOGRÁFICOS.

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

SERGIO ANDRÉS ÁLVAREZ MEDINA

PROFESOR GUÍA:
FERNANDO POBLETE GÓMEZ

PROFESOR CO-GUÍA:
NANCY HITSCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
EDUARDO GODOY VEGA

Este trabajo ha sido parcialmente financiado por Fondecyt 1211484, Fondecyt 1181506 y proyecto RT_44-18 del Instituto Antártico Chileno (INACH)

SANTIAGO, CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL
EN COMPUTACIÓN
POR: **SERGIO ANDRÉS ÁLVAREZ MEDINA**
FECHA: 2021
PROF. GUÍA: FERNANDO POBLETE GÓMEZ
PROF. CO-GUÍA: NANCY HITSCHFELD KAHLER

DESARROLLO DE UNA APLICACIÓN PARA VISUALIZACIÓN, CREACIÓN Y MODIFICACIÓN DE MAPAS PALEOGEOGRÁFICOS.

En el contexto del estudio de la geografía de la Tierra hace millones de años, es normal que los geólogos tengan que modificar mapas que almacenan información sobre la Tierra en distintos periodos de estudio cada vez que se realiza un nuevo descubrimiento que detalla de mejor manera la geografía terrestre. En la actualidad existen varias ciencias que utilizan los mapas generados por los geólogos para sus investigaciones, siendo de vital importancia en los estudios que estas realizan. A pesar de esto, para realizar el proceso de modificación de los mapas solo existen herramientas que, o bien se basan en software pagado, o son complejas de usar, dificultando la obtención de nuevos resultados y la actualización de datos existentes.

La aplicación Relief Creator nace como una solución al problema anterior, permitiendo modificar mapas paleogeográficos de forma sencilla mediante el dibujo en tiempo real de polígonos, o la importación de estos en formato Shapefile. Estos polígonos serán usados para modificar las alturas almacenadas en los mapas paleogeográficos.

La aplicación está implementada en el lenguaje de programación Python y hace uso de la biblioteca PyOpenGL, la cual permite usar la funcionalidad de OpenGL en Python para poder visualizar la información de los mapas. Además, la aplicación permite la visualización de los modelos modificados en tres dimensiones en tiempo real, ayudando a los geólogos y científicos a verificar si los resultados generados son los esperados de forma más sencilla.

La aplicación permite tanto la importación como el dibujo interactivo de polígonos sobre los mapas que se carguen en el programa. Además, permite realizar modificaciones sobre los datos del mapa usando polígonos, permitiendo la actualización de la altura de los puntos al interior de estos junto con el uso de máscaras para filtrar los puntos a modificar. También permite la interpolación y suavizado de los puntos que se encuentran al borde de los polígonos. Junto a lo anterior, la aplicación permite también la exportación de los mapas con los puntos modificados y de los polígonos generados.

La aplicación genera mapas de elevación de una calidad similar a los generados con tecnología ya existente, siendo la diferencia entre las alturas de los mapas generados por la aplicación con la de los generados con tecnología existente menor a 1 metro en promedio.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

La Tierra puede entenderse como un complejo conjunto de subsistemas interconectados e interrelacionados, bajo los cuales la energía y la materia interactúan y modulan aspectos tan diversos como la formación de montañas, cambios en el clima y la evolución de la vida. Los mapas paleogeográficos, mapas que contienen información sobre la forma de la Tierra hace millones de años, cumplen un rol esencial para comprender el funcionamiento de estos subsistemas en el pasado, ya que entregan información sobre la distribución y elevación (paleotopografía y paleobatimetría) de las masas continentales y oceánicas de la Tierra en su pasado geológico (Figura 1.1). Así, proveen las condiciones de borde necesarias que permiten modelar y comprender el clima terrestre, procesos geodinámicos y su expresión en superficie, la evolución de la vida y las complejas interrelaciones que existen entre estos y otros sistemas. (Poblete y col. 2021)

A medida que avanza el estudio sobre la paleogeografía de la Tierra, se van descubriendo nuevos datos que permiten detallar de mejor manera los mapas actuales. El proceso para generar un mapa de paleoelevación consiste en llevar la topografía actual al periodo que se quiere estudiar (trasladando y rotando partes de los continentes), y luego ir modificando las alturas del terreno usando la información geológica y paleontológica disponible. Así, el estudio de las rocas permite saber si un lugar correspondía a un fondo marino o a un delta de río, por ejemplo. Esta información es obtenida principalmente de estudios que permiten identificar características de la Tierra en el pasado geológico, como lo son el descubrimiento de fósiles o la composición del terreno, tomando prioridad por sobre la información obtenida mediante el movimiento del terreno actual a la época de estudio. (Baatsen y col. 2016)

A pesar de que en la actualidad existen herramientas que permiten la manipulación y creación de mapas paleogeográficos (GPlates, GMT, QGis y Matlab), no existe un programa de código abierto y especializado en el proceso de creación y modificación de la elevación de estos mapas. Por ello, los geólogos y científicos se ven en la necesidad de usar diversos programas que no están hechos para tal labor con la finalidad de lograr la modificación de los mapas.

Esta memoria se enmarca en este último proceso, y consiste en desarrollar una aplicación gráfica interactiva que permita realizar, de forma sencilla, la generación de polígonos sobre

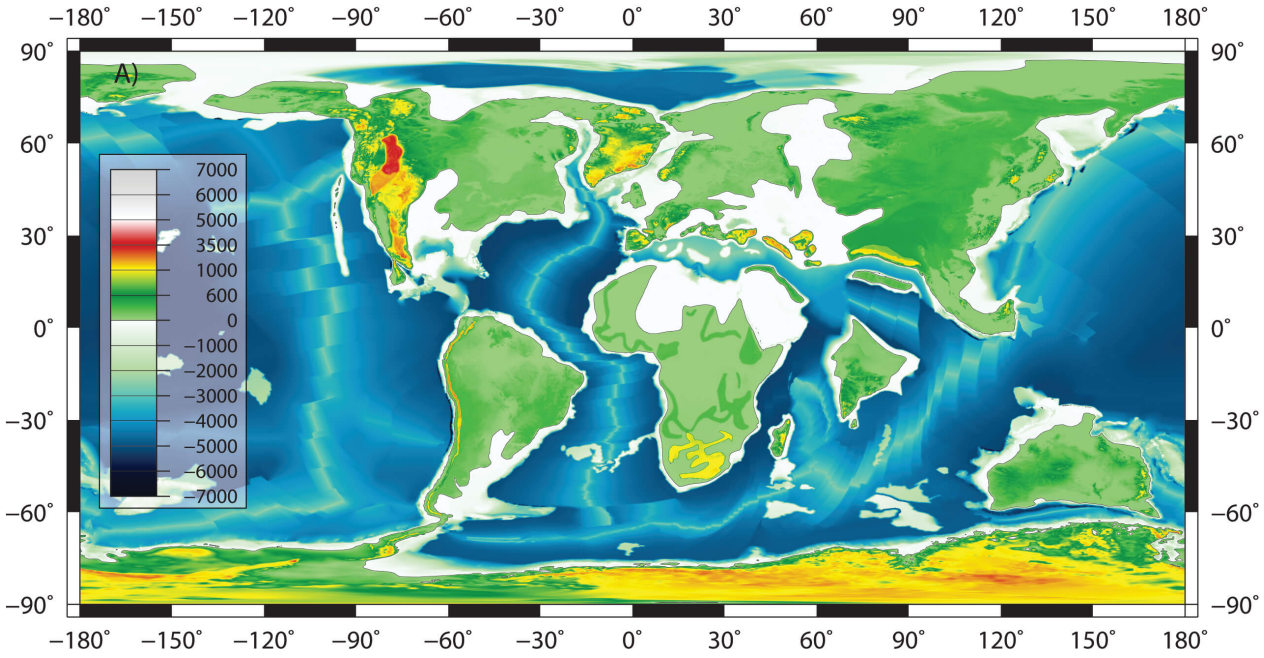


Figura 1.1: Reconstrucción paleogeográfica, donde se muestra la paleotopografía y paleobatimetría de la Tierra a los 60 Ma (Tomado de Poblete y col. 2021).

los mapas paleogeográficos y la edición de la altura de los puntos de este mediante el uso de funciones sobre estos polígonos. Además, se planea incluir una funcionalidad que permita a la aplicación el poder visualizar los mapas paleogeográficos en tres dimensiones.

En las siguientes secciones se explica en detalle cuál es el problema que se desea resolver, cuál es la relevancia que este tiene en el proceso de modificación de mapas de alturas paleogeográficas y cuáles fueron las alternativas analizadas para resolver este problema. Además, se entrega una descripción general de la solución implementada junto con un resumen de los resultados obtenidos.

1.2. Problema

El proceso de generación de un mapa paleogeográfico consiste en un conjunto de procesos que, tomando la topografía actual de la Tierra, aplican sobre esta un conjunto de transformaciones que la modifican al periodo en donde se desea estudiar. Estas modificaciones están basadas en otros estudios y simulaciones realizadas sobre la Tierra por otras ciencias, permitiendo recrear de forma más detallada la geología de la Tierra en el pasado. Este es un proceso multidisciplinario que requiere de científicos con conocimientos específicos sobre diversos temas de estudio, tomando bastante tiempo. (Poblete y col. 2021)

Además del proceso de generación anterior, es necesario actualizar los mapas creados a medida que nuevos hallazgos que detallan de mejor manera la geología de la Tierra son descubiertos. Esto se hace mediante la modificación directa de las alturas de los mapas.

(Poblete y col. 2021)

No existe un proceso o herramienta estándar a usar para actualizar la información de los mapas, usando los geólogos y científicos diversos programas, incluyendo entre estos programas que no están diseñados para la modificación de la información de mapas paleogeográficos o que se basan en software pagado, no existiendo una solución gratuita que permita realizar este proceso con facilidad.

1.3. Relevancia

Este trabajo es importante, ya que posibilitará la modificación y visualización de la altura de los mapas de paleoelevación, facilitando el proceso que tienen que hacer los geólogos y científicos para actualizar los mapas basándose en nuevos datos de paleoaltimetría, geocronológicos y estructurales, entre otros, de forma sencilla y rápida. La aplicación desarrollada permite editar y actualizar, de forma amigable y rápida, mapas paleogeográficos, aportando de forma significativa a la comunidad de geocientíficos que trabajan en estos temas.

El proceso de generación de un mapa paleogeográfico es un proceso lento. (Poblete y col. 2021) Estos mapas son de vital importancia en el estudio de otras ciencias o el estudio de la flora y fauna del pasado, siendo importante el mantener estos mapas actualizados con la información del estado del arte. (Baatsen y col. 2016)

1.4. Objetivos

A continuación se detalla el objetivo general de la solución implementada en esta memoria junto con los objetivos específicos de esta.

1.4.1. Objetivo General de la solución

El objetivo de esta memoria consiste en diseñar e implementar una aplicación gráfica interactiva que permita crear y modificar mapas paleogeográficos, junto con permitir la modificación de las alturas de los mapas haciendo uso de polígonos. Además, debe permitir visualizar en 3 dimensiones el mapa paleogeográfico que se está modificando, junto con ser desarrollado usando buenas prácticas de ingeniería de software.

1.4.2. Objetivos específicos de la solución

Los objetivos específicos que cumple la aplicación desarrollada en esta memoria son los siguientes:

1. Crear y visualizar polígonos sobre un mapa paleogeográfico.

2. Permitir el ingreso de atributos en los polígonos generados en la aplicación.
3. Interpolar los puntos que se encuentran en el borde de los polígonos generados para evitar cambios bruscos en el terreno.
4. Modificar las alturas de los puntos que se encuentran al interior de los polígonos generados en el mapa paleogeográfico mediante el uso de funciones sobre los puntos al interior de este.
5. Visualizar en 3 dimensiones un mapa paleogeográfico.
6. Leer de mapas en formato Netcdf y exportar los mapas modificados en el mismo formato.
7. Leer y crear polígonos en formato Shapefile.
8. Permitir la fácil extensión con nuevas funcionalidades en el futuro.

1.5. Resumen de los resultados

A continuación, describe un resumen de los resultados obtenidos de esta memoria, explicando de forma breve tanto el programa desarrollado como los resultados obtenidos de la verificación de este.

1.5.1. Programa desarrollado

En esta memoria se desarrolló una aplicación que posee la interfaz gráfica que se muestra en la figura 1.2. Esta ofrece al usuario, en la parte izquierda, una lista de herramientas que permiten la modificación de los mapas, como lo son la creación de polígonos, el cálculo de alturas máximas y mínimas, el cambio de las alturas de los puntos al interior de los polígonos, herramientas de interpolación y suavizado de las superficies usando polígonos, entre otras.

Las herramientas implementadas en la aplicación pueden dividirse en 4 categorías que se presentan a continuación:

- Herramientas de visualización: Herramientas que permiten al usuario ajustar la visualización de los mapas.
- Herramientas de polígonos: Herramientas que permiten al usuario crear y organizar polígonos dentro de carpetas.
- Herramientas de relieve: Herramientas que permiten al usuario modificar el relieve de los mapas cargados en la plataforma.
- Herramientas de interpolación: Herramientas que permiten modificar las alturas de los puntos que se encuentran al exterior de los polígonos.

En la parte superior, en el menú de *File* se encuentran opciones para cargar y exportar archivos tanto en formato Netcdf como Shapefile, en el menú *Edit* se muestran opciones

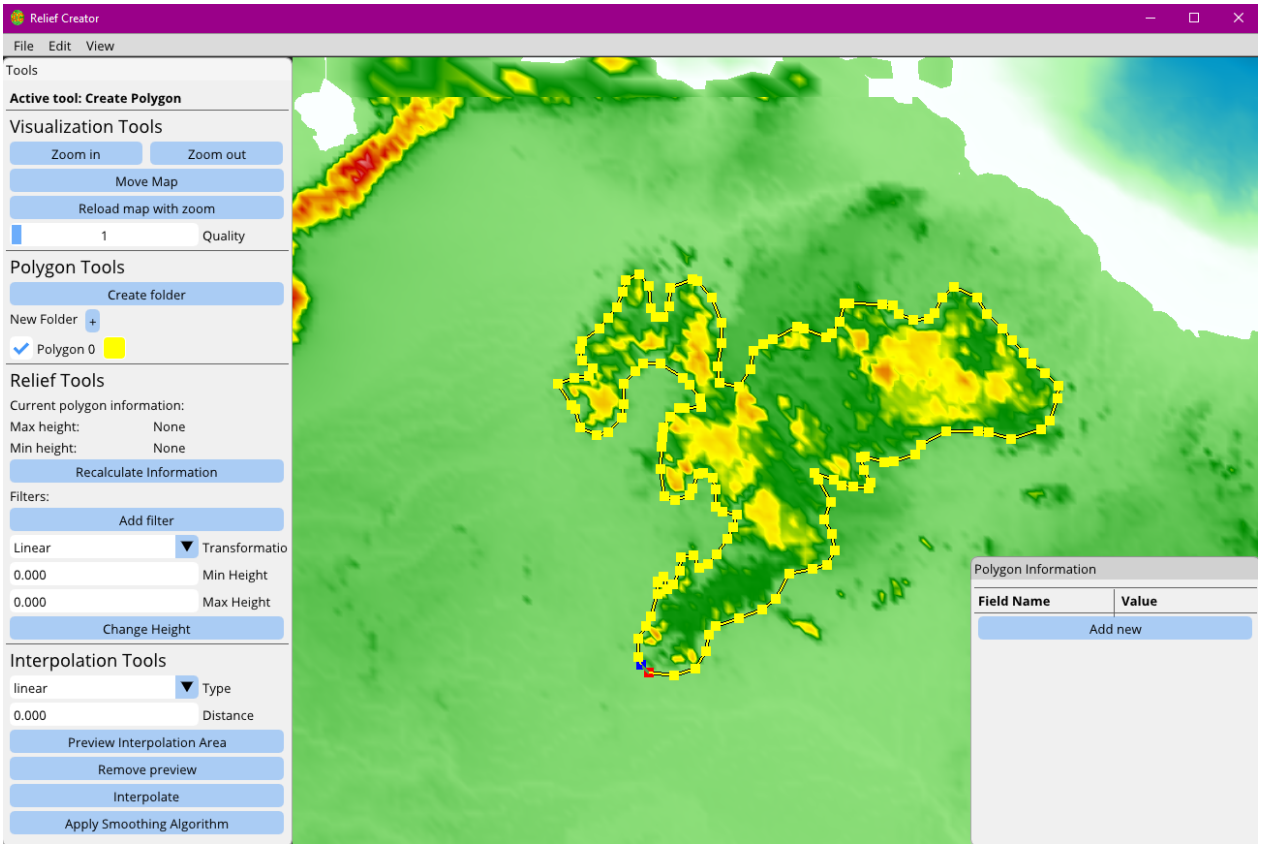


Figura 1.2: Visualización del programa, mostrando un mapa cargado y un polígono. A la izquierda, el menú de herramientas que permite al usuario generar polígonos y modificar las alturas de los mapas, a la derecha, una ventana que permite al usuario el generar atributos a los polígonos del programa.

que permiten revertir las acciones hechas en la plataforma, y en el menú *View* se muestran opciones que permiten modificar la forma en la que se muestran los datos en la escena, pudiendo seleccionar ver los triángulos (figura 1.3) que componen el modelo que se muestra, o incluso los puntos del mapa directamente (figura 1.4), además, también permite cambiar la visualización del mapa a tres dimensiones mostrándose un modelo tridimensional como el que se aprecia en la figura 1.5.

En el modo de renderizado en tres dimensiones (mostrado en la figura 1.5) se tiene a la izquierda un menú que contiene tres secciones principales, las cuales se describen a continuación:

- Información de la cámara: sección que posee la información relacionada con la posición y orientación de la cámara.
- Herramientas de visualización: Herramientas que permiten modificar la forma en la que se muestran los mapas en tres dimensiones.
- Herramientas de unidades: Herramientas que permiten modificar las medidas que se usan para generar los modelos en tres dimensiones.

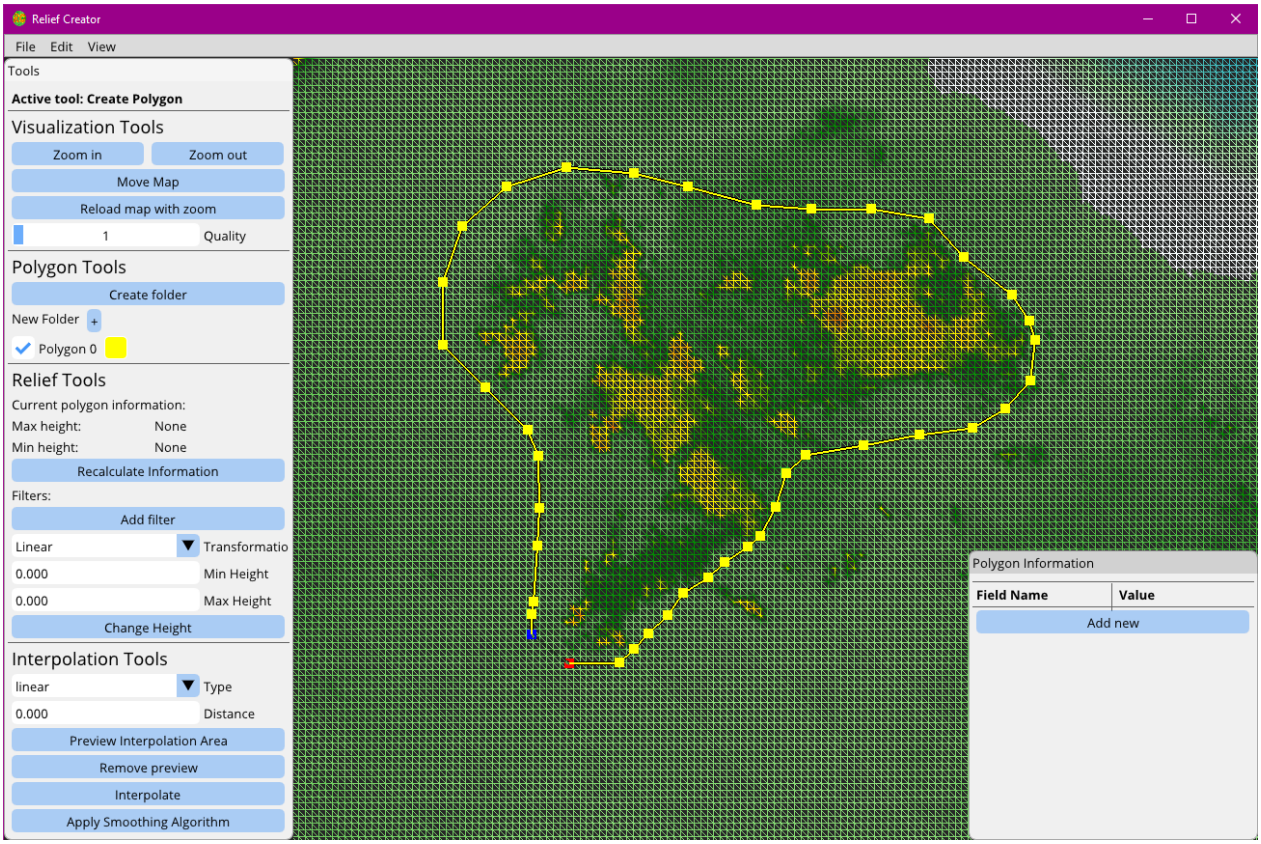


Figura 1.3: Visualización del programa, mostrando un mapa cargado y un polígono en el modo de triángulos.

1.5.2. Validación de la solución

Con la finalidad de comprobar si la aplicación implementada genera mapas de la misma calidad que los que se generan en la actualidad haciendo uso de las herramientas actuales, se decidió realizar la modificación de un mapa paleogeográfico de dos formas diferentes, usando los métodos que se utilizan en la actualidad (como se muestra en Poblete y col. 2021), y usando la aplicación implementada en esta memoria, usando los mismos polígonos en ambos casos. Se calculó la diferencia en altura de todos los puntos de los mapas resultantes, obteniéndose una diferencia en la altura de los puntos modificados inferior a 1 metro en promedio.

Entre los elementos que causan que la diferencia entre los mapas no sea cero se encuentran el hecho de que los métodos de transformación de los puntos efectuado por los programas pueden ser diferentes junto con posibles errores de redondeo generados al momento de realizar los cálculos.

1.6. Contenido de la memoria

En el capítulo 2 se describen en detalle las tecnologías utilizadas en el proceso de modificación de mapas paleogeográficos, además de describir los tipos de datos y archivos que se usan comúnmente para almacenar este tipo de información. En el capítulo 3 se describe

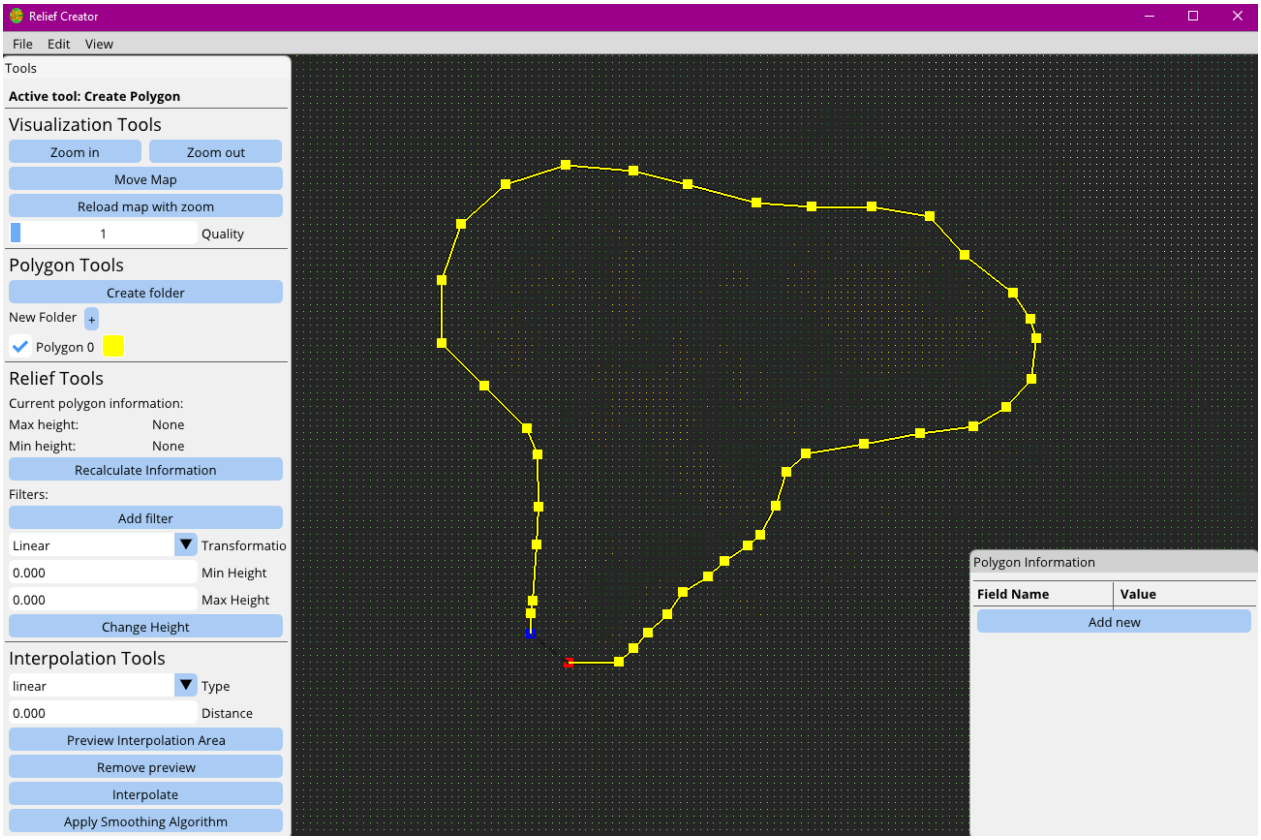


Figura 1.4: Visualización del programa, mostrando un mapa cargado y un polígono en el modo de puntos.

formalmente el problema que se intenta resolver con la aplicación desarrollada en esta memoria junto con detallar los requisitos que esta debe cumplir y la metodología de desarrollo empleada para desarrollarla. En el capítulo 4 se describe a detalle la arquitectura con la que se implementó la aplicación junto con explicar los roles de las distintas clases que se usaron en la implementación del programa. En el capítulo 5 se describe a detalle la implementación de los diferentes componentes que conforman la aplicación desarrollada. En el capítulo 6 se explica cómo utilizar la aplicación, además de como se ha validado que el programa resolvía el problema descrito en el capítulo 3. Finalmente, en el capítulo 7 se detallan las conclusiones que se obtuvieron de la memoria.

Todas las imágenes de herramientas mostradas en esta sección y en las secciones posteriores corresponden a imágenes de la aplicación desarrollada en esta memoria.

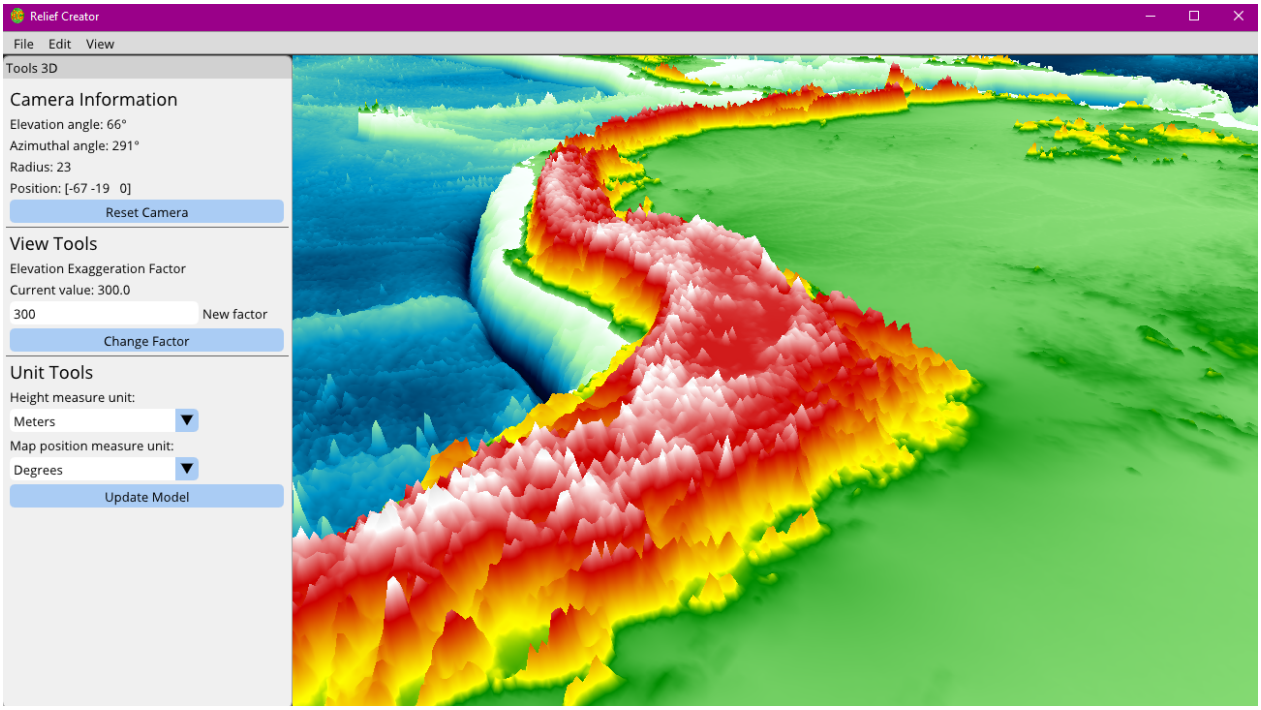


Figura 1.5: Visualización del programa, mostrando un mapa cargado en modo de tres dimensiones.

Capítulo 2

Antecedentes

Las reconstrucciones paleogeográficas son esenciales en diversas ciencias que se encargan del estudio de la Tierra y el medio ambiente, abarcando áreas tan diversas como el estudio de la dinámica terrestre (geodinámica) y el estudio del paleoclima, como también campos como la biología y la ecología, pues, proveen las condiciones de borde para el modelamiento del clima y de la geodinámica de la Tierra.(Poblete y col. 2021)

Los mapas de paleoelevación tienen un rol crucial en las ciencias que hacen estudios sobre la Tierra, pues, especifican la forma de la Tierra hace millones de años, entregando los datos fundamentales para que otras ciencias puedan realizar estudios y simulaciones sobre el comportamiento de la Tierra en el pasado. (Ruiz y col. 2020)

Para realizar las reconstrucciones de la Tierra en periodos específicos de tiempo, se utilizan mapas paleogeográficos. Estos almacenan en su interior información obtenida mediante un gran número de procesos que involucran estudios hechos en diferentes disciplinas, los cuales permiten especificar las posiciones y alturas de las masas continentales y oceánicas hace millones de años. (Baatsen y col. 2016)

A medida que avanza el estudio de la Tierra, es posible obtener cada vez mayor información sobre características particulares de la Tierra mediante la altura o forma de las masas continentales en un cierto periodo de tiempo, facilitando esto el estudio de otras disciplinas que basan sus investigaciones en esta información. Junto a lo anterior, los avances de la computación no solo han mejorado la disponibilidad de los datos al interior de la comunidad científica, sino que también ha permitido el mejorar la calidad de la información en si misma, mejorando esto la calidad de la información obtenida a partir de simulaciones y experimentos que utilizan la información de la altura y masas continentales. (Poblete y col. 2021)

La topografía de la Tierra, y en particular, la definición de las alturas que esta tiene hace millones de años, es un tema que aún está en discusión en la comunidad científica. La generación o modificación de un mapa paleogeográfico involucra diferentes procesos que abarcan desde, estudios topográficos realizados a la Tierra, reconstrucción de las placas tectónicas hasta el periodo de estudio, aplicación de los resultados obtenidos de otras ciencias, hasta modificaciones obtenidas a partir de la literatura y estudios investigativos que detallan la altura en el periodo de estudio. Estos procesos toman gran cantidad de tiempo y son tareas que requieren de un alto conocimiento en el manejo de herramientas de información geográfica

(herramientas GIS) junto con científicos con conocimientos específicos de ciencias geológicas y paleontológicas. (Poblete y col. 2021)

Dado esto, es de vital importancia la existencia de una herramienta que permita la fácil modificación de los mapas de paleoelevación, sin la necesidad de realizar nuevamente el proceso completo de generación de estos mapas.

2.1. Herramientas para modificar elevaciones de terreno

La base para todo mapa paleogeográfico es el modelo tectónico que lo sustenta y que permite reconstruir la posición en el pasado de las distintas placas tectónicas que forman el planeta. Para esta tarea, actualmente existe la herramienta de código abierto llamada GPlates, de amplio uso dentro de la comunidad de geocientíficos, que permite la visualización y manipulación de modelos de reconstrucciones de tectónica de placas. Este programa se basa en que las placas tectónicas son rígidas internamente y que su movimiento en una esfera puede ser descrito por medio de una rotación en torno a un eje vertical que pasa por el centro de una esfera.

El segundo elemento clave para la creación de un mapa paleogeográfico, es la manipulación de la topografía inicial que permite modificarla desde su elevación actual hasta la elevación deseada (figura 2.1). Este proceso requiere de la interpolación y manipulación de imágenes ráster (por ejemplo, Netcdf o Geotiff) a partir de parámetros dados por el usuario, entre ellos, la zona a modificar y los datos de elevación deseados. Para desarrollar estas modificaciones se han propuesto una serie de herramientas basadas en programas ya existentes, entre ellos Matlab (Baatsen y col. 2016), QGis (Ruiz y col. 2020) y GMT (Wessel y col. 2013).

A continuación, se describe cada uno de estos programas a detalle, señalando características importantes de cada uno, para que están diseñados y si son de pago o no.

2.1.1. MATLAB

MATLAB (abreviación de Matrix Laboratory en inglés) es un lenguaje multiparadigma y un ambiente de desarrollo diseñado e implementado por MathWorks con la finalidad de facilitar cálculos matemáticos.

MATLAB fue inicialmente diseñado con la finalidad de facilitar cálculos matemáticos en distintas áreas de la ingeniería, sin embargo, con el paso del tiempo, se han añadido nuevas funcionalidades a la herramienta, facilitando a sus usuarios el manejo de matrices, dibujo de gráficos, manejo de datos, implementación de algoritmos, entre muchas otras funcionalidades.

Desde su primer lanzamiento comercial en 1984, MATLAB ha sido altamente usado en el mundo científico con distintas finalidades. Esto, debido a su amplio uso por los estudiantes en las universidades como herramienta de aprendizaje en distintas áreas de la ingeniería, además de las distintas extensiones, llamadas toolbox en MATLAB, desarrolladas por diferentes

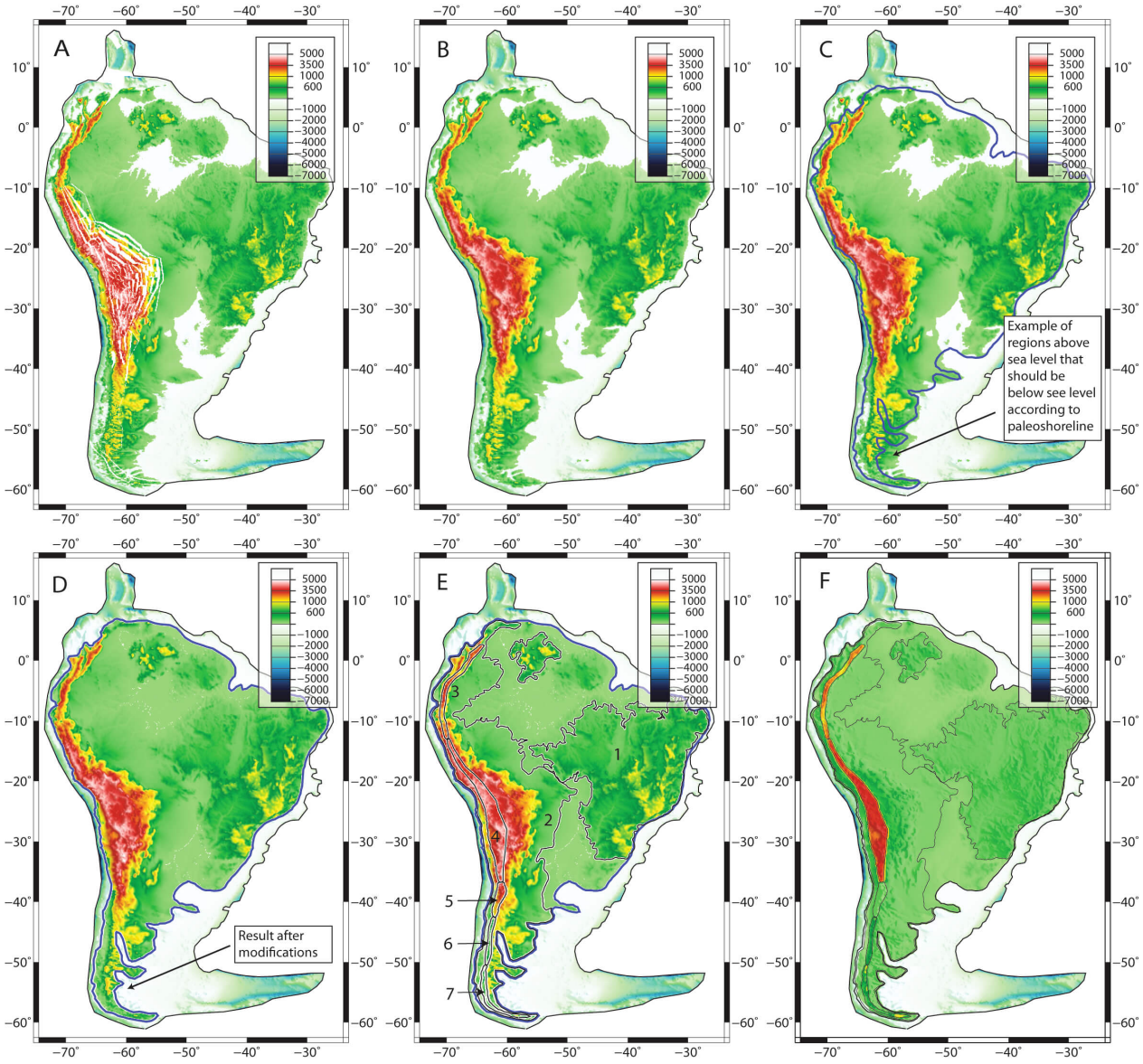


Figura 2.1: Procedimiento simplificado para realizar modificaciones en la topografía, tomando como ejemplo Sudamérica. A) Reconstrucción de Sudamérica a los 40 Ma; B) Interpolación para llenar zonas sin datos; C-D) Transformaciones en regiones al interior y exterior de la línea de costa (línea azul); E-F) Detalle de las zonas cambiadas (1-7) y resultado final.

estudiantes y científicos alrededor del mundo, las cuales agregan funcionalidad especifica de ciertas áreas de la ingeniería y ciencias a la herramienta.

Como se señaló anteriormente, MATLAB no es solo un ambiente de desarrollo, sino que también es un lenguaje de programación, teniendo los usuarios que aprender el lenguaje de programación con anterioridad para hacer uso efectivo de la herramienta, haciendo su uso particularmente difícil para usuarios con poco conocimiento en lenguajes de programación o programación en general.

MATLAB es un programa pagado, teniendo los usuarios, centros de investigación y universidades que pagar por el uso de la herramienta.

2.1.2. GPlates

GPlates, proyecto que comenzó en el año 2006 por el profesor Dietmar Müller, cuya versión 1.0 fue liberada el año 2010, es un programa de código abierto que ofrece una combinación de herramientas que permiten la reconstrucción interactiva de las placas tectónicas, facilitan el manejo de información para sistemas de información geográfica y permite realizar visualizaciones ráster.

El programa fue diseñado con la finalidad de manipular reconstrucciones de las placas tectónicas, ofreciendo diferentes funciones para crear, modificar, importar, exportar y visualizar los datos de las reconstrucciones tectónicas que se desean estudiar. La modificación de las placas tectónicas se basa en polígonos con bordes dinámicos, permitiendo a los usuarios el modificar estos polígonos con la finalidad de recrear construcciones tectónicas.

El programa es desarrollado y mantenido en la actualidad por un equipo internacional de científicos ubicados en la escuela de geociencias de la universidad de Sídney y la división de ciencias planetarias y geológicas de Caltech, siendo su uso y descarga gratuito para todos los usuarios bajo la licencia GPLv2 (GNU General Public Licence versión 2.0).

GPlates también es una herramienta que es altamente usada en la comunidad científica para realizar estudios sobre la Tierra. Sin embargo, dado lo especifico de sus funciones, requiere de conocimientos previos en herramientas de información geográfica (herramientas GIS) para poder hacer un uso efectivo de las funcionalidades que este ofrece.

2.1.3. QGIS

QGIS, inicialmente lanzado el año 2002 bajo el nombre de Quantum GIS, es una herramienta de código abierto, multiplataforma, diseñada con la finalidad de facilitar el análisis, visualización y edición de datos geoespaciales.

QGIS es un programa altamente usado no solo en las universidades y centros de investigación, sino que también en muchas organizaciones privadas (Agencia de seguridad nacional de Estados unidos, Departamento de servicios públicos de información territorial de Nueva

Zelanda, entre otros), distribuido bajo la licencia GPLv2, siendo su uso y descarga gratuitos.

QGIS funciona como un sistema de información geográfica, permitiendo a los usuarios cargar mapas en distintos formatos, incluyendo formatos web y en forma de bases de datos, y realizar modificaciones sobre estos mapas haciendo uso de las diferentes herramientas que ofrece.

El programa permite la creación y exportación de puntos, líneas y polígonos en formato Shapefile, formato altamente usado por la comunidad científica para almacenar información de tipo vectorial.

Las herramientas que ofrece el programa son variadas y bastante específicas, requiriendo un alto grado de conocimiento en sistemas de información geográfica por parte de los usuarios para poder hacer uso del programa.

2.1.4. GMT

GMT (abreviación de Generic Mapping Tools en inglés) es un conjunto de herramientas de consola, desarrolladas inicialmente en el año 1988 por Paul Wessel y Walter H. F. Smith, diseñadas con la finalidad de procesar y mostrar mapas en dos y tres dimensiones.

Estas herramientas ofrecen funcionalidades para cargar los mapas y realizar diferentes algoritmos de procesamiento de imágenes sobre estos, incluyendo entre estos rasterización y visualización de los mapas junto con diferentes tipos de proyecciones.

Sin bien las herramientas fueron inicialmente desarrolladas para ejecutarse desde la consola de comandos, existen bibliotecas que permiten la ejecución de las herramientas en otros lenguajes como Python, Java, C++, entre otros.

Las herramientas son de código abierto y están desarrolladas bajo la licencia LGPL (GNU Lesser General Public License), siendo gratuito el uso y descarga de estas.

Dado que son herramientas de consola, estas poseen una dificultad elevada de uso, no siendo apropiadas para usuarios con bajo conocimiento en computación.

2.1.5. Ventajas y desventajas de las herramientas actuales

Un problema que tiene tanto MATLAB como GMT, es que ambos requieren conocimientos básicos de programación, MATLAB requiere aprender el lenguaje de programación en donde funciona, mientras que GMT requiere que los usuarios aprendan a usar la interfaz de la consola de comandos junto con los comandos y parámetros para poder utilizar sus funcionalidades, no siendo ideales para usuarios sin conocimientos previos en programación.

Debido a que un mapa paleogeográfico puede venir definido en diferentes formatos, y el

usuario no necesita por qué conocer como están estructurados estos formatos internamente, las herramientas interactivas que permiten modificar estos mapas mediante el uso de polígonos son ideales para este proceso, pues, ofrecen una solución simple, sin necesidad de conocimiento previo sobre los mapas que se quiere modificar o de herramientas GIS, al problema de modificar las elevaciones de un mapa de elevaciones.

Si bien GPlates permite el dibujo de polígonos y la modificación de los mapas mediante polígonos, este programa está enfocado en la simulación y reconstrucción de las placas tectónicas de la Tierra, no ofreciendo funcionalidad para modificar fácilmente la elevación de mapas de paleoelevación. Además de esto, el programa posee, en general, una dificultad elevada de uso, no siendo ideal para usuarios que no poseen conocimientos previos sobre herramientas GIS.

QGIS permite editar, visualizar y analizar información geoespacial, junto con ofrecer alternativas para modificar la elevación de mapas de paleoelevación en forma de plugin (Ruiz y col. 2020), sin embargo, este programa posee una dificultad de uso muy elevada, principalmente debido al exceso de herramientas que ofrece, necesitando de un conocimiento elevado del programa para poder lograr realizar la tarea de modificar mapas paleogeográficos.

En el cuadro 2.1 se muestra un resumen de la información detallada anteriormente, comparando las distintas herramientas respecto a su dificultad de uso, si son de código abierto o no y en que se especializa cada una.

Tabla 2.1: Cuadro comparativo de las herramientas que existen actualmente para trabajar con datos geoespaciales. Se analizan 3 características de estos: (1) Su dificultad de uso en el contexto de la modificación de la altura de mapas geoespaciales, (2) si son gratuitos o no y (3) propósito para el que fueron diseñados.

	Matlab	GPlates	QGis	GMT
Dificultad de uso	Alta	Media	Alta	Alta
Gratuito	No	Sí	Sí	Sí
Especialidad	Cálculo matemático, multiproposito	Reconstrucción de placas tectónicas	Editar, visualizar y analizar información geoespacial	Set de herramientas de visualización y edición

Así, a pesar de que las herramientas anteriores permiten la manipulación y creación de mapas paleogeográficos, aún no existe una aplicación especializada en este proceso que permita modificar y manipular, mediante polígonos, una topografía dada y crear, a partir de esta, un mapa paleogeográfico.

2.2. Datos y formatos

A continuación se describen los tipos de datos con los que trabaja la aplicación desarrollada en esta memoria junto con los formatos que se usan en la actualidad para almacenarlos.

La aplicación trabaja con dos tipos de datos principalmente, polígonos y mapas de elevación.

2.2.1. Polígonos

Un polígono es definido como una lista de puntos que conforman una figura plana. Estos son usados comúnmente en herramientas de información geográfica para delimitar y separar secciones al interior de un mapa. En la aplicación desarrollada en esta memoria, y en la mayoría de las herramientas de información geográfica, los polígonos, además de tener los puntos que los componen, poseen una serie de características o atributos asociados a ellos. Estos atributos son muchas veces configurados por los mismos usuarios y poseen información relacionada con el uso del polígono.

Shapefile es un formato de archivo que permite el almacenamiento de datos de tipo vectorial, esto es, puntos, líneas y polígonos, en su interior, junto con atributos asociados a estos datos. Este formato es altamente usado por la comunidad científica y aceptado por un gran número de herramientas de información geográfica. Más información sobre el formato Shapefile se encuentra en la sección A.4.2 de los anexos.

Los polígonos definidos en las herramientas de información geográfica son polígonos que viven en un plano, y como tal, solo son definidos usando dos coordenadas. Un polígono de tales características es considerado simple si es que este divide al plano en dos conjuntos claramente diferenciables (el interior del polígono del exterior del polígono), mientras que un polígono es considerado complejo si es que posee dos aristas que se interceptan.

Los polígonos simples poseen un gran número de propiedades que permiten a las herramientas de información geográfica realizar modificaciones a los mapas de elevación, siendo una de las más importantes la existencia de una forma eficiente de calcular si un punto pertenece a un polígono o no. Esto último es usado bastante en herramientas de información geográfica, pues, permite a los programas determinar qué puntos de los mapas se encuentran al interior de los polígonos definidos por el usuario, para posteriormente realizar operaciones sobre estos puntos. Además, este método es independiente de la convexidad de los polígonos usados, entregando mayor libertad a los usuarios al momento de generar los polígonos para modificar mapas.

Cabe destacar que todo polígono complejo puede ser interpretado como un conjunto de polígonos simples, y como tal, es posible aplicar algoritmos diseñados para polígonos simples sobre polígonos complejos interpretándolos como conjunto de polígonos simples.

Si bien el uso de polígonos es común en herramientas de información geográfica, estos no son la única forma de datos vectorial usado por los geólogos y científicos, destacando

también las líneas continuas, grupos de polígonos, polígonos con agujeros en su interior, y polígonos complejos. Los archivos Shapefile permiten almacenar todos estos tipos de datos en su interior.

2.2.2. Modelo de elevación digital

Un modelo de elevación digital, también conocido como mapa de elevación, define la elevación de diferentes puntos que se definen sobre un mapa de la tierra. Para esto, se necesitan 3 arreglos principales, un arreglo que define las longitudes usadas en el mapa de elevación, otro arreglo definiendo las latitudes usadas y un último arreglo que define la elevación de los puntos.

Con la información anterior, las herramientas de información geográfica son capaces de generar una grilla en dos dimensiones en donde cada punto de esta almacena la latitud, longitud y altura de la tierra.

La representación de los puntos de los mapas de elevación en forma de grilla es bastante útil y bastante usada en herramientas de información geográfica, pues, permite representar la información en forma de un plano en dos dimensiones, facilitando la aplicación de operaciones sobre los puntos o la definición de polígonos.

Con la finalidad de renderizar los puntos de la grilla en dos dimensiones, es usual que los programas de información geográfica generen triángulos que conecten los puntos de la grilla (generen una triangulación de los puntos de la grilla), generando con esto un modelo que será posteriormente mostrado en pantalla. Este modelo facilita la interpolación de los valores de los puntos de los mapas, permitiendo a las herramientas de información geográfica el poder colorear los mapas completamente, a pesar de que los mapas no tengan una alta densidad de puntos. En la figura 2.2 se muestra un ejemplo de cómo los mapas triangulan las grillas de puntos para mostrarlas en pantalla.

Un formato bastante usado en las herramientas de información geográfica para almacenar información de los mapas corresponde al formato Netcdf, este formato permite el almacenamiento de información en forma de arreglos en su interior de forma eficiente y no dependiente de la plataforma, permitiendo también la definición de metainformación relacionada a los mapas. Mas información sobre el formato Netcdf puede encontrarse en la sección A.4.1.

Si bien para la definición de un mapa de elevación se necesitan tres arreglos principales (latitud, longitud y alturas), los mapas no están limitados a solo tres arreglos, pudiendo también definir otras características del terreno al interior de los archivos que puedan ser representadas en forma de arreglos.

Si bien el formato Netcdf permite el almacenamiento eficiente de arreglos, este no especifica la forma en la que estos tienen que ser definidos, pudiendo definir la información de un mapa paleogeográfico de diferentes formas. Mientras que lo más común es que un archivo Netcdf almacene tres arreglos principales, uno definiendo las latitudes usadas, otro definiendo las longitudes y un arreglo de dos dimensiones definiendo las alturas del mapa, también existen

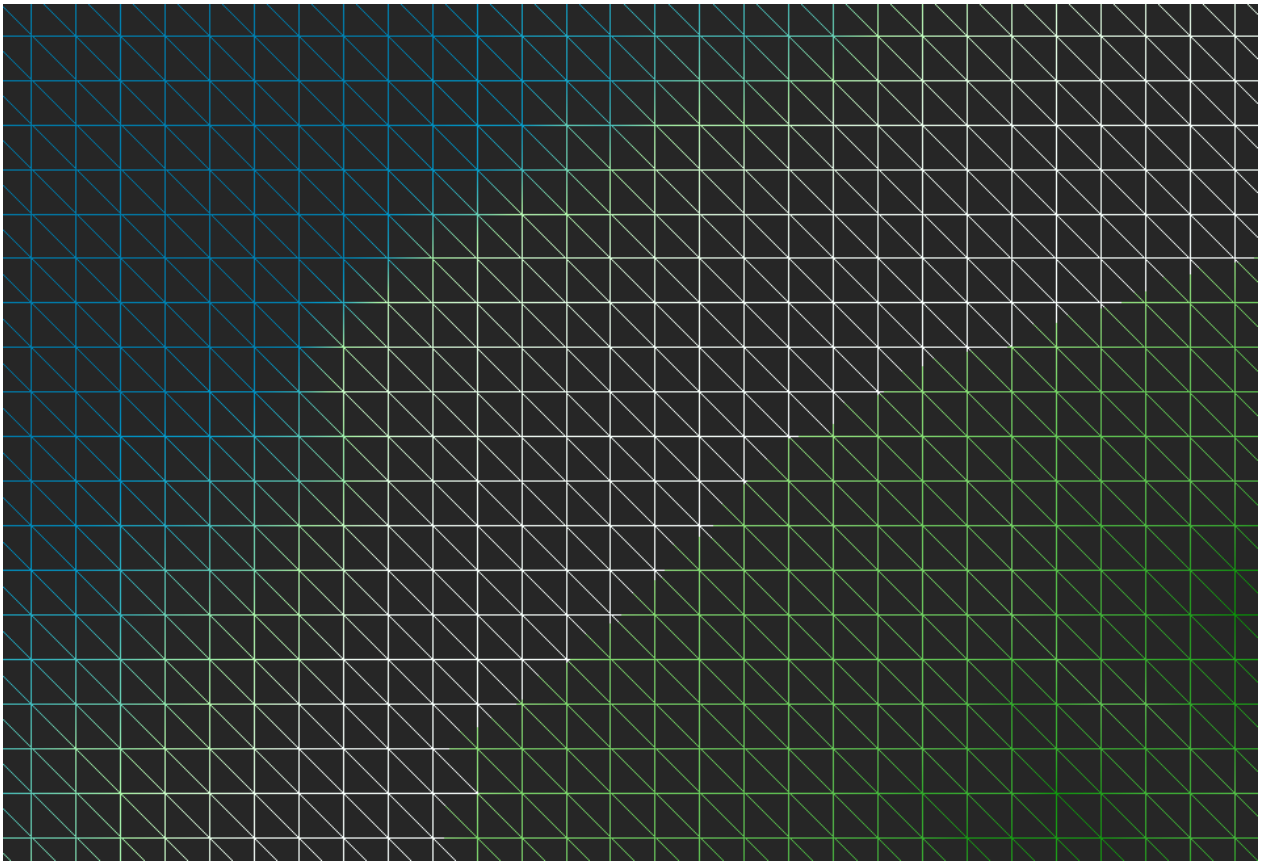


Figura 2.2: Ejemplo de triangulación generada en las herramientas de información geográfica para renderizar los puntos de un mapa en pantalla.

programas que definen en archivos Netcdf solo las longitudes iniciales y finales, y en un arreglo unidimensional las alturas, definiendo como metainformación la cantidad de datos usados para la longitud y latitud y el valor del salto entre un valor y otro.

No existe un limite en la cantidad de datos que puede almacenar un archivo Netcdf, pudiendo almacenar estos tantos datos como se requiera necesario siempre y cuando estos sigan las reglas del formato Netcdf. Esto trae consigo un beneficio para los usuarios, pues, pueden almacenar tantos datos como estimen necesario, pero un problema para las herramientas de información geográfica, quienes tienen que implementar diferentes algoritmos para evitar tener problemas de memoria al momento de mostrar los mapas a los usuarios.

Capítulo 3

Especificación del problema, requisitos y metodología de desarrollo

En este capítulo se presenta en detalle el problema que poseen los geólogos y científicos en la actualidad, junto con explicar los requerimientos y casos de uso de la aplicación que permiten resolver estos problemas. Además, se detalla la metodología de desarrollo aplicada en la implementación de la aplicación.

3.1. Problema actual

En el proceso del estudio de la Tierra, es común descubrir nuevos hallazgos que detallen de mejor forma la paleotopografía de la Tierra en un cierto periodo de tiempo, especificando con mayor detalle la posición de las placas de la Tierra o la altura de estas, como también lo es el descubrir hallazgos que generan o rectifican información ya existente, modificando completamente la información que se tiene.

En la actualidad, como se explica en el capítulo 2, ya existen herramientas especializadas en realizar reconstrucciones tectónicas. Sin embargo, no existe una herramienta de código abierto, de fácil uso, para modificar las alturas a partir de un modelo de elevación digital preexistente. Debido a esto, muchos geólogos y científicos utilizan diferentes tipos de aplicaciones para poder modificar las alturas de los mapas, siendo algunas de pago o que requieren de mucho tiempo para aprender a usarlas.

A partir de esto, surge la necesidad de un software de código abierto que permita visualizar y modificar, de forma sencilla, mapas de paleoelevación, junto con poder visualizar los cambios tanto en dos como en tres dimensiones, permitiendo a los geólogos y científicos comprobar que los cambios efectuados sobre los mapas modifican el terreno de la manera en que ellos desean.

Por ejemplo, el cliente de esta aplicación utilizaba programas como QGis para generar curvas de nivel de las cuales extraer polígonos, para posteriormente utilizar las herramientas de consola que ofrece GMT, las cuales poseen una dificultad elevada de uso, para modificar la información de los puntos que se encuentran al interior del polígono generado.

3.2. Requerimientos y casos de uso de la herramienta

A continuación se detallan los requerimientos específicos que la aplicación implementada debe cumplir junto con los casos de uso de esta.

3.2.1. Requerimientos específicos de la aplicación desarrollada

La aplicación debe cumplir con una serie de requerimientos para que esta cumpla con su propósito, a continuación se detallan los requerimientos funcionales y no funcionales, de alto nivel, que la aplicación debe cumplir.

Los requerimientos funcionales que debe cumplir la aplicación son los siguientes:

- **Leer mapas en formato Netcdf, visualizarlos y exportar los mapas modificados en el mismo formato**: El programa debe permitir la lectura de mapas de elevación en formato Netcdf, permitiendo al usuario visualizar la información de elevación que estos almacenan. Además, debe permitir la exportación de estos nuevos mapas hacia archivos en el mismo formato.
- **Leer polígonos en formato Shapefile y exportarlos en el mismo formato**: El programa debe permitir la importación de polígonos que se encuentran almacenados en archivos en formato Shapefile, pudiendo usar estos para la modificación de los mapas cargados. Además, el programa debe permitir la exportación de los polígonos generados en la aplicación en formato Shapefile.
- **Agregar atributos a los polígonos generados**: El programa debe permitir agregar atributos a los polígonos importados o generados por el mismo programa, exportando estos atributos al momento de exportar los polígonos a formato Shapefile.
- **Crear y visualizar polígonos sobre un mapa paleogeográfico**: La aplicación debe permitir visualizar la información de un mapa paleogeográfico, y, además, debe permitir la creación de polígonos sobre este mapa. Los polígonos deben poder ser visualizados sobre el mapa cargado.
- **Modificar las alturas de los puntos que se encuentran al interior de los polígonos generados en el mapa mediante el uso de funciones**: La aplicación debe permitir modificar las alturas de los puntos que se encuentran al interior de los polígonos que se dibujan o importen en esta. Esta edición de las alturas se debe efectuar aplicando funciones sobre los puntos que se encuentran al interior de los polígonos.
- **Interpolar las alturas de los puntos que se encuentran en el borde de los polígonos generados**: La aplicación debe permitir la interpolación de las alturas de los puntos que se encuentran al exterior de los polígonos, aplicando algoritmos de interpolación en 2 dimensiones, a una distancia especificada por el usuario, con la finalidad de dar un carácter más natural a las modificaciones efectuadas por el usuario en los mapas.

- **Visualizar en 3 dimensiones un mapa paleogeográfico**: El programa debe permitir la visualización en tres dimensiones de los mapas de elevación con los que se está trabajando. Esta visualización debe ser en tiempo real, con una cámara móvil controlada por el usuario, y debe mostrar los cambios hechos por el usuario a los mapas de elevación.

Los requerimientos no funcionales que debe cumplir la aplicación se presentan a continuación:

- **Permitir la fácil extensión con nuevas funcionalidades en el futuro**: El programa debe ser desarrollado utilizando buenas prácticas de la ingeniería de software, desarrollando documentación y utilizando metodologías que permitan que nuevos ingenieros puedan fácilmente añadir nuevas funcionalidades a la aplicación o modificar las ya existentes.
- **Permitir la visualización de los mapas en tiempo real**: La visualización de los mapas y las modificaciones que se hagan sobre estos deben ser visualizadas en tiempo real en la aplicación una vez aplicadas sobre los mapas.

3.2.2. Casos de uso

La aplicación debe permitir a los usuarios trabajar de forma sencilla con los mapas que estos desean modificar y con los polígonos que estos quieren usar para realizar las modificaciones. Es por esto que, a partir de los requerimientos nombrados en la sección 3.2.1, nacen los casos de uso de la aplicación a implementar. En la figura 3.1 es posible ver un diagrama que muestra los casos de uso de la aplicación junto con las relaciones que los interconectan.

La aplicación presenta solamente un actor, siendo este el usuario final de la aplicación. Este único actor es capaz de realizar todos los casos de uso de la aplicación, teniendo completo control sobre esta.

A continuación se realiza una descripción general de los casos de uso mostrados en el diagrama de la figura 3.1.

- **Visualizar un mapa**

  Para visualizar un mapa en la aplicación, el usuario debe cargar un archivo en formato NetCDF haciendo uso de la herramienta de importación de archivos NetCDF que entrega la aplicación. Si el archivo es cargado correctamente en la aplicación, se mostrará en la aplicación la información contenida en el archivo en forma de mapa, siendo coloreada con la paleta de colores que posee por defecto la aplicación. En caso de que el archivo cargado no pueda ser cargado, entonces se mostrará un mensaje de error al usuario indicando cual fue el problema que ocurrió.

  La paleta de colores usada para la coloración de los mapas puede ser cambiada por el usuario haciendo uso de la herramienta de carga de archivos en formato CPT que ofrece la aplicación.

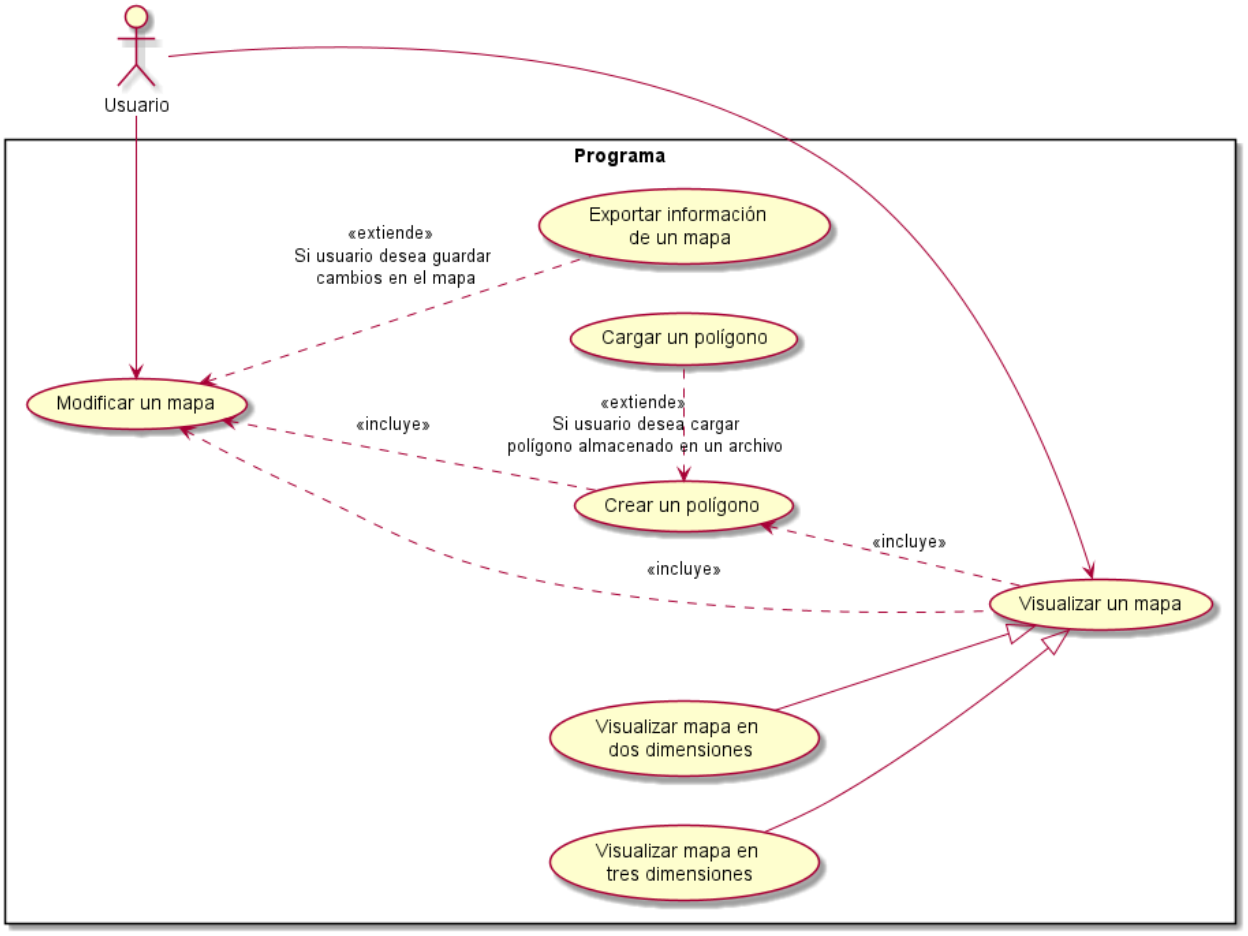


Figura 3.1: Diagrama de casos de uso de la aplicación implementada.

El mapa cargado puede ser visualizado tanto en dos dimensiones como en tres dimensiones, siendo por defecto la visualización en dos dimensiones.

- **Modificar un mapa**

  Para modificar un mapa utilizando la aplicación, es necesario, en primera instancia, cargar un mapa en la aplicación haciendo uso de la herramienta de carga de archivos NetCDF que la aplicación ofrece. La información del mapa debe estar contenida en un archivo en formato NetCDF.

  Luego, una vez cargado el mapa, es necesario crear un polígono que limite los puntos que se quieren modificar, esto se puede lograr utilizando la herramienta de polígonos que ofrece la aplicación, generando un polígono haciendo uso del ratón, o cargando uno o más polígonos de un archivo en formato Shapefile en la aplicación.

  Una vez cargado el mapa y creado el o los polígonos, el usuario puede modificar directamente la información de las alturas del mapa haciendo uso de las herramientas que ofrece la aplicación, como lo son las herramientas de transformación, interpolación o suavizado de las alturas del mapa, o puede generar filtros que limiten los puntos que se desean modificar para posteriormente hacer uso de las herramientas que ofrece la aplicación para modificar las alturas de los mapas cargados.

  Una vez modificado el mapa de elevaciones cargado inicialmente, este debe ser exportado

con la finalidad de generar un archivo con la nueva información actualizada. Para esto el usuario debe hacer uso del menú de exportación del modelo que ofrece la aplicación.

3.3. Metodología de desarrollo implementada

La metodología de desarrollo usada, en general, consiste inicialmente en el diseño de la aplicación, para posteriormente seguir con un desarrollo incremental de la funcionalidad, mientras que, en paralelo, se realizan pruebas de usuario que entregan retroalimentación y moldean el desarrollo. La metodología puede resumirse como se muestra a continuación:

1. **Diseño de la aplicación**

   En primera instancia se realizó un diseño de la aplicación que cumpliera con el requisito de ser un programa fácilmente extensible, planificando las responsabilidades de cada componente con la finalidad de lograr la mayor escalabilidad posible.

2. **Desarrollo de un mínimo producto viable**

   Luego del diseño, se desarrolló funcionalidad suficiente que permitiera a los usuarios poder ver un mapa en pantalla, crear polígonos y modificar la altura de los mapas, testeando los resultados obtenidos de esta funcionalidad mediante pruebas de usuario.

3. **Desarrollo de requerimientos**

   Posteriormente, se realizó la funcionalidad requerida para cumplir con todos los requerimientos establecidos en la memoria, desarrollando el resto de la funcionalidad restante en la aplicación. En paralelo a este proceso, se realizaron pruebas de usuario que entregaron retroalimentación de la funcionalidad implementada en la aplicación.

4. **Optimización y resolución de bugs**

   Finalmente, una vez programada toda la funcionalidad que cumple con los requisitos de la aplicación, se empezó a optimizar esta para mejorar la experiencia de usuario junto con resolver cualquier tipo de error que esta pudiera tener. Además de esto, también se realizó testing unitario de la funcionalidad clave de la aplicación. Esto con la finalidad de entregar una batería de tests que los futuros programadores puedan utilizar al momento de añadir nueva funcionalidad a la aplicación.

En la etapa de diseño de la aplicación se especificó cuáles iban a ser los componentes que conformaran la aplicación, como se iban a comunicar entre ellos, las responsabilidades que cada uno tendría, que tecnologías usaría cada componente y como iba a ser el ciclo de la aplicación. Este paso es importante, pues, facilita el desarrollo de la funcionalidad de la aplicación, ya que detalla el comportamiento que debería tener la aplicación junto con que es lo que se espera de cada componente.

En primera instancia se desarrolló un mínimo producto viable que consiste en un programa que permite la modificación básica de las alturas de un mapa. Este mínimo producto desarrollado considera solo la funcionalidad de visualización de los mapas, dibujo de polígonos y modificación básica de las alturas de los mapas, sin incluir opciones de interpolación o

suavizado. Para verificar la correctitud del mínimo producto viable desarrollado, se realizaron pruebas de usuario que validaran la funcionalidad desarrollada.

Posteriormente al diseño y prueba del mínimo producto viable, se comenzó a desarrollar el producto final. Este producto considera toda la funcionalidad relacionada con los requisitos de la memoria, incluyendo esto tanto los requisitos funcionales como los no funcionales.

Para el desarrollo de la funcionalidad de la aplicación se aplicó el paradigma de programación orientado a objetos, pues, este ofrece numerosos beneficios, como la reusabilidad de código, la escalabilidad y eficiencia en las aplicaciones. Además, el uso de esta metodología de programación facilita el entendimiento de los sistemas, haciendo más fácil la extensión de la aplicación por parte de otros desarrolladores.

En paralelo con el desarrollo del producto final, se fueron realizando pruebas de usuario de las que se obtuvo retroalimentación del producto a medida que se implementaba, moldeando con esto el desarrollo de la aplicación con la finalidad de obtener un producto final lo más perfeccionado posible. La verificación de la correctitud del producto final también fue realizada mediante pruebas de usuario.

Finalmente, una vez finalizado el desarrollo de los requerimientos de la aplicación, se procedió a realizar un testing extensivo de la funcionalidad de esta, realizando tests unitarios que comprueben la funcionalidad clave de la aplicación y que permitan a futuros desarrolladores comprobar que las nuevas funcionalidades agregadas no deshabilitan las que ya están implementadas.

El testing realizado en el paso 4 de la metodología considera tanto pruebas unitarias de bajo nivel que testean la lógica de las clases y componentes de la aplicación, como pruebas de alto nivel que consideran la correctitud del programa completo, comprobando que la funcionalidad conjunta de los componentes de la aplicación entrega los resultados esperados.

En cuanto a la documentación de la aplicación, esta se realizó en paralelo a todos los procesos enumerados anteriormente, realizando documentación tanto para el código, funcionalidad del programa y el diseño de este.

Capítulo 4

Diseño de la solución

En este capítulo se explica en detalle aspectos de diseño de la aplicación, junto con mostrar y explicar los diagramas de clases de los diferentes componentes y la comunicación que existe entre ellos.

4.1. Arquitectura general de la aplicación

La aplicación desarrollada en esta memoria implementa el patrón de programación de modelo-vista-controlador, pudiendo separar la aplicación en tres partes principales, el modelo, encargado de gestionar los datos y lógica de la aplicación, la vista, componente encargado de gestionar que es lo que se le muestra al usuario, y el controlador, componente encargado de gestionar el input del usuario.

La aplicación desarrollada corresponde a una aplicación interactiva, esto es, una aplicación que reacciona en tiempo real a las acciones del usuario, para esto, la vista está constantemente solicitando información del modelo con la finalidad de actualizar los datos que se le muestran al usuario. Esta información es almacenada en el modelo y modificada mediante el controlador, el cual, dependiendo de la acción realizada por el usuario, modifica la información almacenada en el modelo.

El componente de vista, además de ser la encargada de mostrar las interfaces gráficas al usuario, también se encarga de comunicar al controlador los eventos relacionados con la interfaz gráfica que ocurren en la aplicación. Esto con la finalidad de que el controlador realice las acciones necesarias sobre el modelo dependiendo de las acciones realizadas por el usuario en la interfaz gráfica.

La vista se actualiza un total de 60 veces por segundo, modificando la información que se muestra al usuario en tiempo real, mientras que el controlador solo realiza acciones cuando ocurre un evento en la aplicación, ya sea este evento capturado directamente por el controlador o comunicado por la vista.

La aplicación desarrollada está en todo momento actualizando la vista y esperando eventos por parte del usuario, comunicando estas acciones entre los distintos componentes como se nombró anteriormente. Este ciclo de funcionamiento es mantenido durante todo el tiempo

que la aplicación este activa.

El diseño de la arquitectura de la aplicación fue documentado utilizando diagramas que siguen el estándar UML. Los diagramas que explican la arquitectura y módulos de la aplicación junto con la comunicación entre las distintas clases implementadas fueron generados utilizando PlantUML, programa que permite la generación de diagramas que siguen el estándar UML mediante código. Más información sobre la documentación de la aplicación y las tecnologías utilizadas para generarla se encuentra en la sección A.5 de los anexos.

4.2. Descripción de los módulos principales

Es posible diferenciar 8 módulos principales que componen la aplicación, los cuales se describen a continuación:

- **Engine**: Elemento encargado de la gestión de los recursos del programa, este es el encargado de asignar y comunicar a los demás componentes que es lo que estos tienen que hacer en el momento indicado.
- **Controlador**: Elemento encargado de obtener el input ingresado por el usuario, procesarlo, y comunicar al Engine las acciones a realizar ante el ingreso de determinadas acciones en el programa. Este elemento también es el encargado de ejecutar la lógica de las acciones realizadas en la interfaz gráfica de la aplicación.
- **Escena**: Elemento encargado de crear y gestionar los modelos cargados en la aplicación. Este elemento se encarga de que los modelos cargados se representen en formatos compatibles con OpenGL para su posterior renderizado. Además, es el encargado de realizar las modificaciones de los modelos cuando se usan las funcionalidades del programa.
- **GUI**: Elemento encargado de manejar la interfaz gráfica del programa con la que el usuario puede interactuar.
- **Render**: Elemento encargado del renderizado de los componentes en la pantalla. Este elemento es el que realiza el renderizado de los modelos y de la interfaz gráfica en la ventana del programa.
- **Input**: Elemento encargado de leer y cargar los archivos de origen externo en algún formato manejable por el programa. Este es el encargado de leer archivos en formato Shapefile y Netcdf.
- **Output**: Contrario al input, este elemento es el encargado de generar archivos en formato compatible con otros programas de los elementos que se tienen al interior del programa. Este elemento es el encargado de generar archivos Shapefile con los polígonos generados por el usuario o archivos Netcdf con los mapas de elevación modificados.
- **Programa**: Este elemento es el encargado de almacenar y gestionar el estado del programa. Este elemento almacena la información de que es lo que está pasando en el programa en un determinado momento.

Tanto el Render, Programa, Escena, GUI y el Controlador dependen del Engine, como así mismo, el Engine depende de estos, siendo los únicos dos componentes independientes el Input y el Output. En la figura 4.1 es posible ver como se relacionan estos 8 componentes principales.

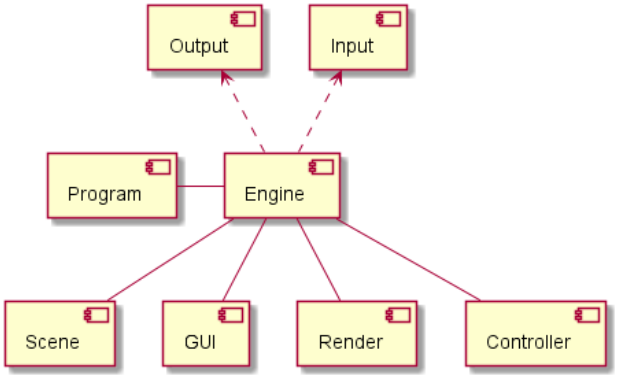


Figura 4.1: Componentes principales del programa y su forma de comunicación.

Como se explicó anteriormente, el Engine es el componente encargado de gestionar los recursos de la aplicación. El limitar la comunicación entre los componentes y hacer que estos solo se comuniquen con el Engine permite que el desarrollo de la aplicación sea más ordenada, siendo la única dependencia externa de cada componente el Engine.

Lo anterior permite que los componentes de la aplicación escalen, pues, con excepción de la dependencia al Engine que estos poseen para comunicarse con el resto del programa, cada componente es independiente en su funcionamiento interno, pudiendo escalar cada componente por separado con facilidad.

4.3. Diagramas de clases

A continuación, se presentan los diagramas de clases de los componentes importantes del programa, junto con una pequeña descripción de estos.

4.3.1. GUI

En la figura 4.2 es posible observar el diagrama de clases del componente GUI. Este componente se comunica con el componente Engine mediante la clase GUIManager. El componente GUI puede ser dividido en tres funcionalidades principales, las carpetas, los iconos y los frames.

Las carpetas son las encargadas de gestionar y agrupar los polígonos en la interfaz de usuario, administrando la ubicación de los polígonos en las distintas secciones que el usuario puede crear.

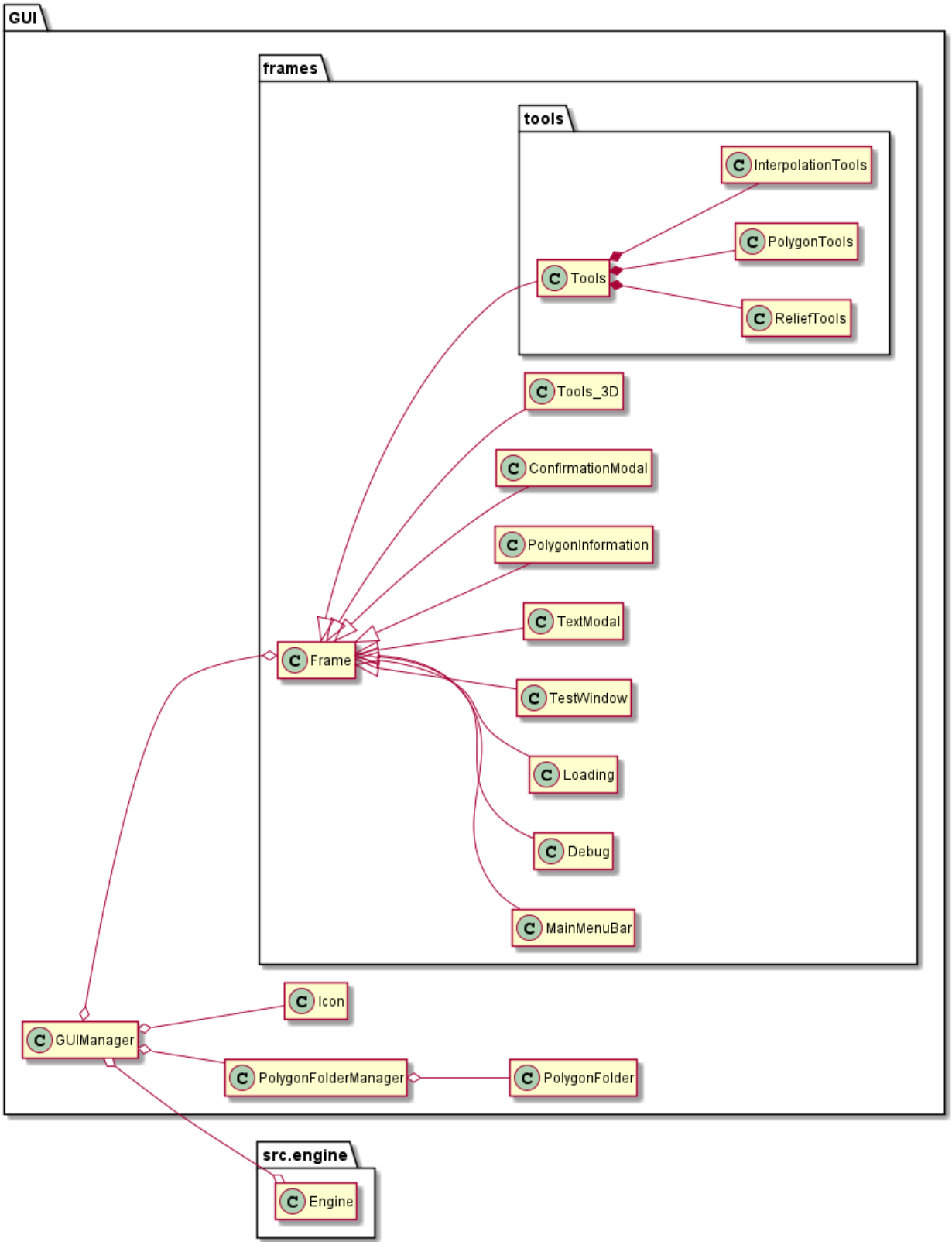


Figura 4.2: Diagrama de clases del componente GUI de la aplicación implementada en la memoria.

La clase Icon es la encargada de gestionar los iconos usados en la interfaz de usuario, esta se encarga de la definición y configuración de iconos para mostrar en la interfaz de usuario.

El paquete frames es el encargado de definir las diferentes ventanas que el usuario visualiza, tanto en el modo 2D como en el modo 3D. En este se define la clase Frame, clase base que debe ser usada para todas las ventanas creadas en la aplicación y todo el conjunto de ventanas disponibles que el usuario puede visualizar.

No todas las ventanas definidas en el paquete frames son visibles por el usuario en todo momento, es la clase GUIManager la que gestiona cuáles son las ventanas visibles de cuáles no y en qué momento deben ser visibles. Además, la clase GUIManager es la encargada de conectar las carpetas de polígonos, los iconos y las diferentes ventanas definidas en la aplicación con el Engine, funcionando como intermediario entre la interfaz de usuario y la funcionalidad definida en el programa.

4.3.2. Escena

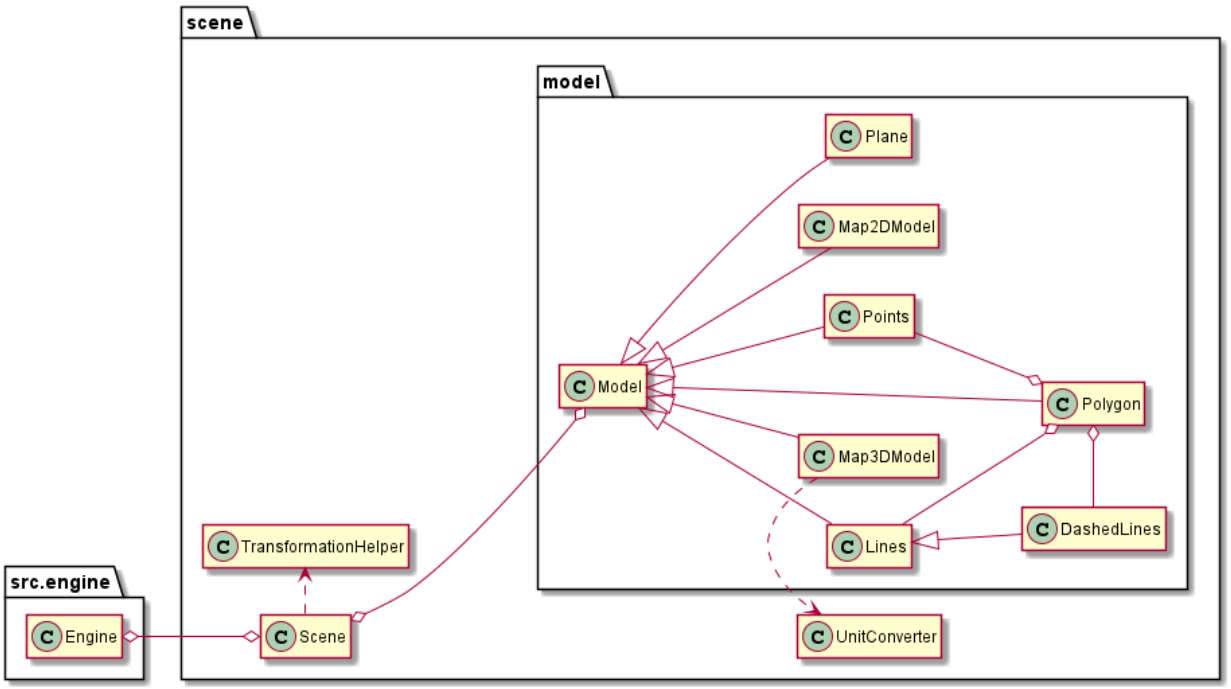


Figura 4.3: Diagrama de clases del paquete Scene del programa implementado.

En la figura 4.3 es posible observar el diagrama de clases del componente de la escena junto con los paquetes y clases que la componen.

La clase Scene es la encargada de gestionar todos los elementos de la escena, esto es, los modelos en dos y tres dimensiones junto con los polígonos. Esta también es la encargada de comunicarse con el Engine y de realizar las transformaciones a los modelos cuando sea necesario.

La clase TransformationHelper es una clase que implementa en su interior los algoritmos de suavizado e interpolación que se detallan en las secciones 5.9 y 5.10 respectivamente, junto con implementar algoritmos para la generación de máscaras y manejo de arreglos que faciliten el realizar las modificaciones a los mapas usando los polígonos definidos en la Escena.

La clase UnitConverter es la encargada de gestionar la conversión de unidades de las medidas usadas en los modelos. Esta clase contiene en su interior los factores de conversión necesarios para cambiar de una unidad a otra.

Al interior del paquete model se realiza la definición de los diferentes modelos que pueden ser mostrados en la Escena. Todos los modelos deben heredar de la clase Model, clase que realiza las configuraciones básicas para que los datos almacenados en los modelos puedan ser dibujados en OpenGL.

Cada modelo dentro del paquete model define en el método *draw* como debe ser dibujado. Este método es llamado en cada frame del programa para realizar el renderizado de los elementos en pantalla. Además, cada modelo debe tener asociado tanto un vertex shader como un fragment shader para poder ser usado por OpenGL en el proceso de dibujo.

Como se logra ver en el diagrama de clases de la figura 4.3, los modelos pueden heredar unos de otros sin ningún problema siempre que estos definan de forma correcta el cómo deben ser dibujados en la escena.

4.3.3. Engine

En la figura 4.4 es posible apreciar el diagrama de clases del componente Engine. En este pueden apreciarse las clases que componen al componente, además de las clases que componen a otros componentes de la aplicación.

El paquete engine puede separarse en tres partes principales, las clases que conforman al componente Engine, las dependencias de otros componentes que se encuentran dentro del mismo paquete que el componente Engine y las dependencias que se encuentran fuera del paquete Engine.

Las clases que corresponden al componente Engine corresponden a la clase Engine, Settings, ProcessManager y ThreadManager. La clase Engine es la encargada de la gestión de los recursos del programa, y, con tal finalidad, esta es la encargada de conectar todos los componentes del programa. Las clases ProcessManager y ThreadManager son las encargadas de gestionar la ejecución de los procesos y threads de la aplicación respectivamente. La clase Settings es una clase estática que se encarga de almacenar las configuraciones presentes en el programa, modificando sus valores cuando ocurren eventos que impactan en las configuraciones del programa, como lo son el cambio en el tamaño de la ventana de la aplicación, cambios en los tamaños de las ventanas de las herramientas, cambios en el tamaño de la fuente usada en la GUI, entre otras configuraciones.

Las dependencias del componente Engine que se encuentran dentro del mismo paquete

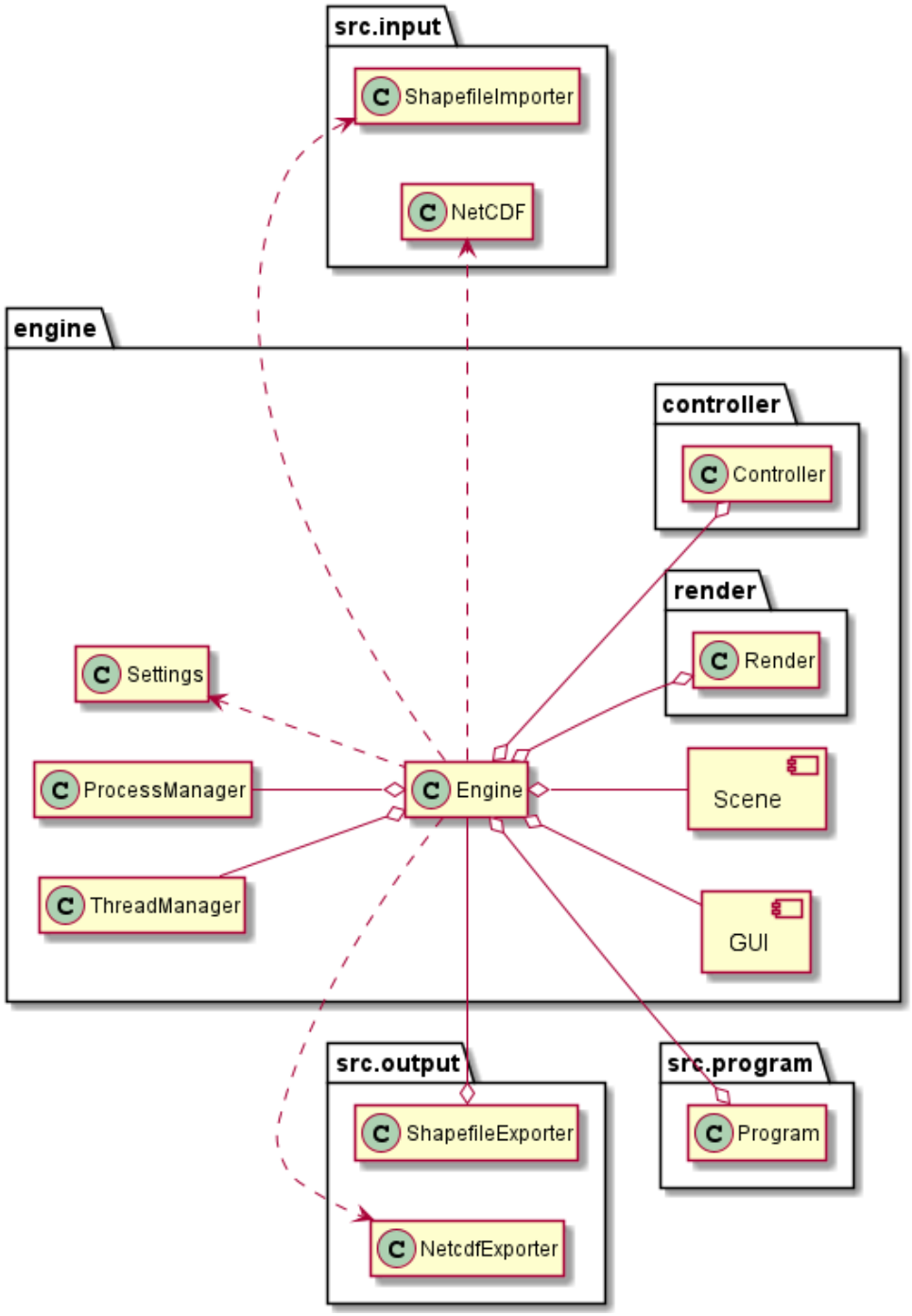


Figura 4.4: Diagrama de clases del paquete Engine de la aplicación implementada. Componentes Escena y GUI no se muestran para hacer la visualización más clara.

corresponden a los componentes de la Escena, GUI, Render y Controlador. Los diagramas de clases correspondientes a la GUI y la Escena son explicados en las secciones 4.3.1 y 4.3.2 respectivamente. El controlador y el render son componentes que están constituidos únicamente de una sola clase que realiza la lógica necesaria que debe ejecutar cada uno.

Las dependencias del componente Engine que se encuentra al exterior del mismo paquete

corresponden a los componentes de Input, Output y Programa. El componente de Input corresponde a un conjunto de módulos que almacenan diferentes funciones para la importación de datos en la aplicación. El componente de Output funciona de la misma forma, ofreciendo un conjunto de funcionalidad para exportar información del programa. El componente del Programa, al igual que el Render y el Controlador, corresponde a un componente constituido únicamente por una sola clase, este almacena el estado del programa mientras está siendo ejecutado.

La clase Engine es el centro de la aplicación implementada. Esta actúa como intermediario entre el componente Engine y todo el resto de las componentes que conforman el programa, teniendo los otros componentes que acudir a esta clase para ejecutar lógica almacenada en los otros componentes. Este además es el encargado de realizar el manejo de errores del programa, ejecutando la lógica que corresponda cuando ocurren errores en los diferentes componentes que componen la aplicación.

4.3.4. Errores

A pesar de que Python contiene clases definidas en su interior que pueden ser usadas para levantar excepciones, en ocasiones, estas son poco específicas en el tipo de error que estas definen, dificultando esto el manejo de errores de la aplicación, y con ello, el entendimiento del programa para futuros desarrolladores.

Con la finalidad de especificar de forma clara los posibles errores que pueden ocurrir en el programa, y la forma en que el Engine maneja estos errores, se decidió generar clases internas a la aplicación que sirvan como clases que puedan ser levantadas cuando el programa identifica algún tipo de error. El diagrama de clases de los errores definidos en el programa se muestra en la figura 4.5.

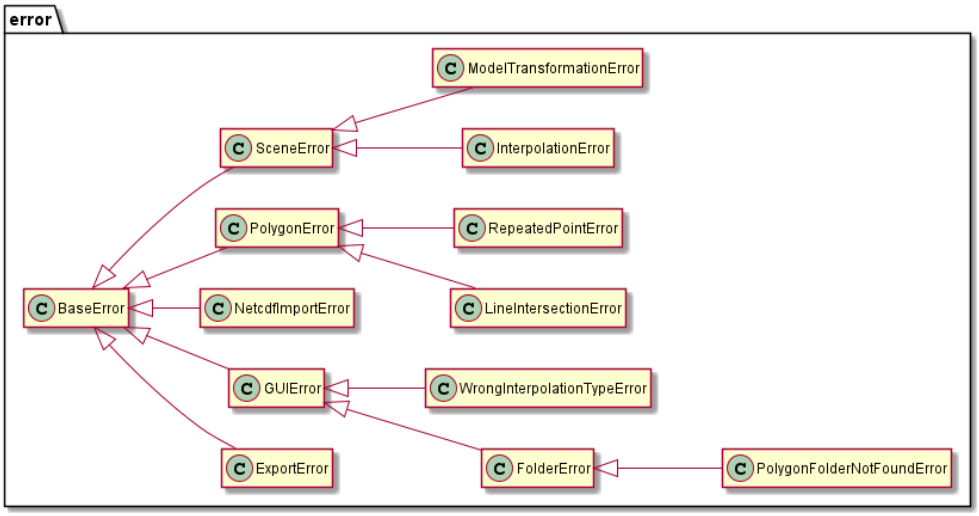


Figura 4.5: Diagrama de clases del paquete error de la aplicación implementada.

Todas las clases definidas con la finalidad de levantar errores en la aplicación deben heredar

de la clase BaseError, esta clase define las configuraciones básicas que tienen implementados los errores.

Todos los errores definidos en la aplicación, además de heredar de la clase BaseError, definen en su interior un diccionario con códigos de error y una descripción del error. Estos diccionarios tienen la finalidad de especificar aún más el tipo de error encontrado, facilitando la labor de corrección de errores a los programadores.

Cada error levantado en la aplicación, siempre y cuando pertenezca a una clase heredada de la clase BaseError, tiene un código asociado, la clase BaseError define un método que permite obtener la descripción del código de error asociado a la clase, facilitando el obtener información sobre la excepción ocurrida.

Capítulo 5

Implementación de la solución

A continuación, se describe la licencia usada en la aplicación implementada, además de detallar la implementación de los componentes principales que conforman la aplicación junto con una explicación de los algoritmos y metodologías utilizadas para su implementación. además, también se explica de forma general el funcionamiento de las tecnologías que utiliza el programa para la implementación de las funcionalidades que ofrece.

5.1. Licencia usada

La licencia de uso de un programa es un documento que detalla de forma explícita quien o quienes pueden hacer uso de una herramienta de software y en qué condiciones.

Esto considera no solo el uso de las aplicaciones, sino también el uso del código fuente, ya sea para fines de extensión o de uso personal.

La aplicación implementada en esta memoria corresponde a una aplicación de código abierto que utiliza la licencia GPLv3, licencia que especifica que tanto el código fuente como la aplicación en si son entregados de forma gratuita, especificando que todo producto derivado tanto de la aplicación en si como del código fuente deben también ser entregados en forma gratuita y bajo las mismas condiciones.

En la sección A.3 de los anexos se encuentra más información sobre las licencias de uso.

5.2. Tecnologías usadas

A continuación, se describen brevemente las tecnologías más importantes usadas en el programa. Descripciones detalladas pueden ser encontradas en la sección A.2 de los anexos.

5.2.1. Python

La aplicación fue desarrollada en lenguaje Python, esto dado que este presenta un gran número de bibliotecas escritas en lenguaje C y C++ que permiten el desarrollo eficiente de las funcionalidades del programa, junto con ser multiparadigma, haciendo más fácil la extensibilidad y mantención de la aplicación. En la sección A.2.1 se encuentra más información sobre el lenguaje de programación Python.

5.2.2. OpenGL

OpenGL es una biblioteca de procesamiento gráfico que permite el renderizado de polígonos en dos y tres dimensiones haciendo un uso efectivo de los recursos de los programas. OpenGL permite el uso eficiente de la GPU en el proceso de renderizado de los modelos, siendo ideal para el desarrollo de aplicaciones interactivas que deben actualizar constantemente la información gráfica que muestran.

PyOpenGL es una biblioteca escrita en Python que ofrece las funcionalidades de OpenGL en Python, siendo la biblioteca usada en esta memoria para el desarrollo de la funcionalidad relacionada con el renderizado de los mapas.

Mas información sobre OpenGL se encuentra en la sección A.2.2 de los anexos.

5.2.3. GLFW

GLFW es una biblioteca complementaria a OpenGL que ofrece una API que facilita la creación de ventanas y el manejo de eventos en la aplicación.

PyGLFW es una biblioteca escrita en lenguaje Python que ofrece las funcionalidades de GLFW en Python, siendo la biblioteca usada en la aplicación desarrollada para el manejo de las ventanas y eventos.

5.3. GPU, Shaders y pipeline gráfico de OpenGL

A continuación se detallan conceptos importantes de computación gráfica que son necesarios para el renderizado de modelos en dos y tres dimensiones en la aplicación.

5.3.1. GPU

Las GPU (de sus siglas en ingles Graphic Processing Unit) son elementos de hardware diseñados específicamente con la finalidad de acelerar el procesamiento gráfico de las aplicaciones.

Las GPU son otro tipo de procesador que, a diferencia de los procesadores normales que

poseen los computadores (CPU), poseen un gran número de núcleos en su interior, permitiendo la ejecución masiva de threads en paralelo. Esto último es ideal para las aplicaciones gráficas, en donde hay que ejecutar una misma pieza de código de forma paralela sobre muchos vértices o sobre muchos píxeles.

Las GPU fueron creadas en el año 1970 con la finalidad de apoyar a las CPU en el procesamiento de buffers y el manejo de los elementos a mostrar en pantalla. Desde entonces, las GPU han avanzado considerablemente en el ámbito tecnológico, especializándose en el procesamiento de buffers y renderizado de elementos en pantalla, pasando a ser en la actualidad, un componente común en computadores, celulares, videoconsolas y otros sistemas que realizan procesamiento gráfico.

Aunque las GPU fueron creadas con la finalidad de apoyar en el procesamiento de los gráficos, en la actualidad también son usadas para una variedad de fines que aprovechan el masivo paralelismo que estas ofrecen, encontrándose entre estos procesos la codificación o decodificación de vídeos, entrenamiento de redes neuronales, cálculo de simulaciones físicas, entre otros.

5.3.2. OpenGL y Shaders

Como se explica en la sección A.2.2, OpenGL es una herramienta que ofrece funcionalidad que facilita el renderizado de objetos en dos y tres dimensiones, es open-source y escrita originalmente en lenguaje C.

Si bien OpenGL ofrece funcionalidad que facilita el proceso de dibujo de modelos en dos y tres dimensiones, los programas y aplicaciones que utilicen OpenGL deben estar diseñados para adaptarse al proceso que sigue OpenGL para realizar el dibujo de estos modelos, a este proceso de dibujo que realiza OpenGL para poder mostrar los elementos se le conoce como el pipeline gráfico de OpenGL.

El pipeline gráfico de OpenGL se encarga de definir los modelos geométricos que se quieren mostrar a partir de los datos que se le proporcionen, posicionar y orientar los modelos frente a la cámara, aplicar propiedades a estos modelos para modificar la visualización de estos y, finalmente, mostrar los elementos en pantalla. Este proceso de renderizado posee distintas etapas, en donde cada una puede ser subdividida en varias etapas internamente. Un resumen de este, que muestra los procesos que se usan en la aplicación implementada, se muestra en la figura 5.1. Cada etapa de este proceso se ejecuta en GPU, mejorando la eficiencia del proceso gracias al masivo paralelismo que estas ofrecen.

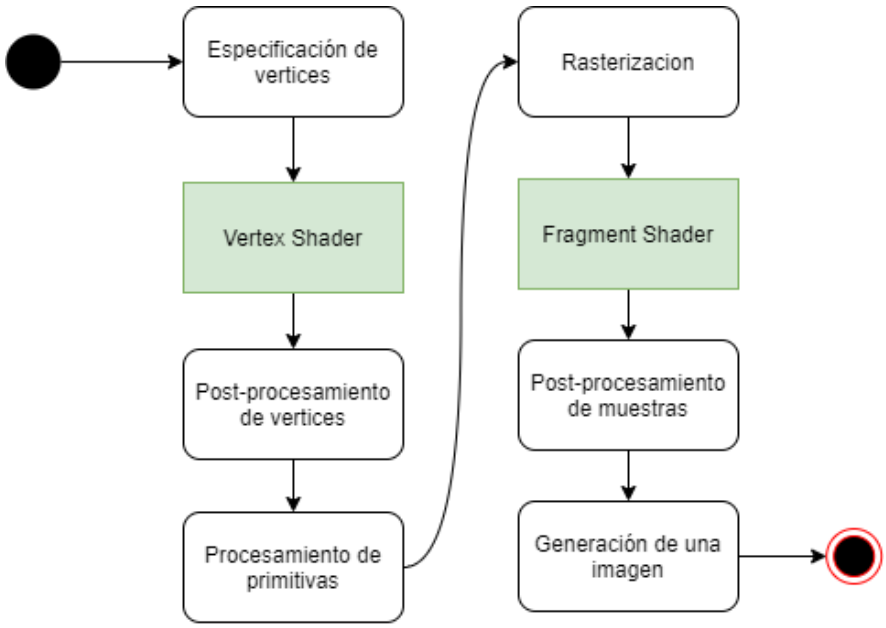


Figura 5.1: Resumen simplificado del proceso de rendering efectuado por OpenGL. Se especifica en verde los procesos que pueden ser programados por el usuario.

A continuación se realiza un resumen de que es lo que OpenGL realiza en cada paso del pipeline mostrado en la figura 5.1.

- **Especificación de vértices**: En este paso del pipeline, OpenGL define los vértices que se usaran en el proceso de renderizado junto con los diferentes atributos que se pudieron haber asociado a estos (atributos como el color de los vértices, vector normal, transparencia, entre otros) para su posterior uso. Para realizar esto, OpenGL utiliza arreglos previamente definidos por el usuario que definen los vértices y los posibles atributos que se les asigna a estos.
- **Vertex Shader**: En este paso se ejecuta un programa definido previamente por el usuario, escrito en lenguaje GLSL (OpenGL Shading Languaje), que recibe como entrada las primitivas de vértices y atributos asociados en el paso anterior, junto con otros atributos extras que pudo haber definido el usuario con anterioridad, para generar el output de este proceso. Los outputs de este proceso pueden ser definidos por el usuario, pudiendo ser más de solo uno, pero debe haber al menos uno que defina la posición final de los vértices en la pantalla. Es en este paso en donde se aplican las transformaciones de los modelos y la cámara a los vértices.
- **Post-procesamiento de vértices**: En este paso se realiza una copia del estado de los buffers con la finalidad de mantener el estado de estos para su posterior uso. En este paso también ocurre la generación de primitivas (triángulos, líneas, puntos, entre otros) a partir de los vértices.
- **Procesamiento de primitivas**: En este paso se realizan una serie de procesos a las primitivas generadas en el paso anterior, entre los que destacan, clipping, esto es, separar las primitivas que están al interior del espacio visual definido de las que se encuentran

al exterior, aplicación de las transformaciones de perspectiva y viewport, además del descarte de los triángulos que no presentan un vector normal que apunte en la dirección de la cámara.

- **Rasterización**: En esta etapa se rasterizan las primitivas resultantes del proceso anterior, resultando de este proceso una serie de fragmentos, objetos que representarán posteriormente los píxeles en pantalla. Los fragmentos incluyen información sobre su posición en la pantalla junto con una lista de parámetros originados a partir del output de los shaders anteriores. Para obtener el valor que estos parámetros tienen asignado a cada fragmento, se realiza una interpolación de los valores de los outputs de los shaders anteriores, tomando como referencia los vértices a los cuales fueron asignados y que tanto influyen estos en los fragmentos generados.

- **Fragment Shader**: En esta etapa se ejecuta un programa previamente definido por el usuario, escrito en GLSL, que recibe como entrada los valores rasterizados de la etapa anterior, los fragmentos, y los parámetros asignados a cada uno. Debe definir un color de salida para cada fragmento generado en la etapa anterior.

- **Post-procesamiento de muestras**: En esta etapa se realizan una serie de tests que deciden si los fragmentos generados son descartados o no. Entre estos destaca el test de profundidad, en donde se descartan los fragmentos que están detrás de otros, el test de pertenencia, en donde se descartan los fragmentos que no pertenecen a la aplicación, como lo es en el caso de cuando una ventana se superpone a la ventana de la aplicación, y el test de recorte, en donde se descartan los fragmentos que se encuentran al exterior de un determinado rectángulo especificado por el usuario.

- **Generación de una imagen**: Finalmente, usando los resultados generados por el proceso anterior, se genera una imagen que posteriormente se mostrará en pantalla.

El proceso anterior ocurre cada vez que se genera una nueva imagen, y, dado que la aplicación implementada en esta memoria se recarga 60 veces por segundo, el proceso anterior se repite 60 veces por segundo, generando una nueva imagen en cada iteración.

5.4. Renderizado en 2 dimensiones

En esta sección se explica a detalle como se renderizan los mapas en dos dimensiones desde el momento en que son cargados en memoria hasta que son mostrados en la pantalla.

5.4.1. Carga y lectura de datos

Cuando el usuario carga un mapa en el programa, es el componente de Input el encargado de obtener los datos del archivo cargado. Este componente comprueba que el archivo cargado y los datos sean correctos antes de pasar estos datos al Engine, de otra forma, solicita al Engine que despliegue un error notificándole al usuario que el archivo que intenta cargar no es válido.

Si el archivo cargado es correcto, los datos del mapa le son entregados a la Escena para que esta genere un nuevo modelo. Los datos corresponden a dos arreglos, representando las coordenadas x e y en un sistema cartesiano, y una matriz que representa las alturas de los puntos en el mapa cargado.

5.4.2. Generación del modelo

Luego de haber cargado y procesado los datos, la Escena genera una instancia de objeto de mapa en dos dimensiones (instancia de la clase Map2DModel) y la asigna como el modelo actual a mostrar en el programa.

A pesar de que la Escena acepta el tener cargado más de un modelo al mismo tiempo, el Programa tiene definido en todo momento un único modelo activo. Es este modelo activo el que se renderiza y muestra en la ventana, y es también este modelo el que el usuario modifica al momento de aplicar operaciones sobre los polígonos.

Para crear una instancia de la clase Map2DModel en el programa, la Escena debe entregarle como input a la clase los arreglos y la matriz obtenida de la lectura del archivo, junto con el archivo de colores que se está usando y las configuraciones actuales de la ventana y el programa, como lo son el tamaño designado para el renderizado del modelo, el nivel de zoom a usar por defecto y la calidad de la renderización a usar.

El archivo de colores es administrado por el Programa, el Programa almacena en todo momento un archivo de colores activo que contiene información sobre que colores deben recibir los puntos de los mapas según la altura que estos posean. Mientras que las configuraciones actuales de la ventana y el programa son obtenidas del Engine.

Con la información anterior, el constructor de la clase Map2DModel realiza una serie de procesos para generar arreglos compatibles con OpenGL para poder ser dibujados.

Primero, a partir de los arreglos y la matriz cargada, la clase Map2DModel genera un arreglo unidimensional que contiene los vértices del modelo y que entrega a OpenGL. Dado que el mapa generado será representado en dos dimensiones, todos los vértices tienen como coordenada z un valor constante. Este arreglo no es modificado durante toda la vida útil del modelo en el programa.

Posterior al proceso anterior, se genera otro arreglo que se entrega a OpenGL con las alturas de los modelos. Este arreglo es generado usando las alturas definidas en la matriz que se recibe como input, y es asignado como un atributo extra de los vértices que se cargan en OpenGL.

Aunque los vértices hayan sido cargados a OpenGL, se debe definir como se comunican estos vértices para poder generar las primitivas que dibuje OpenGL, esto es, hay que especificar cuáles son los triángulos que utilizan los vértices anteriores para que OpenGL pueda dibujarlos. Para esto, se define un arreglo de índices que indica como se generan estas primitivas y se le entrega este arreglo a OpenGL para que realice el proceso de renderizado.

Como los mapas cargados en la plataforma pueden tener muchos puntos, muchos más que los que pueden ser vistos en la pantalla al mismo tiempo, la clase Map2DModel utiliza el valor de la resolución que le fue entregado por la Escena para generar el arreglo de índices solo con los triángulos que sean necesarios para una correcta visualización de los modelos. Esto con la finalidad de ahorrar en los recursos que utiliza el programa al momento de mostrar el modelo.

5.4.3. Matriz de proyección

Si bien el modelo tiene todos sus elementos definidos, es necesario definir un elemento más para poder renderizar el modelo correctamente.

Dado que los mapas en dos dimensiones deben aceptar el poder aplicarles Zoom, junto con poder moverlos por la ventana, es necesario definir una matriz de proyección que modifique la posición de los puntos en el vertex shader con la finalidad de dar la ilusión de movimiento y acercamiento a los mapas.

La matriz anterior es generada utilizando los datos de la ventana entregados por la Escena, y es entregada al vertex shader para que este modifique la posición de los vértices mostrados en la pantalla, dejando solo al interior del espacio visual los vértices que correspondan al nivel de zoom y la posición del mapa que se tienen en el momento. Dado que la aplicación de esta matriz se realiza al interior del vertex shader, esta se realiza en paralelo en la GPU por cada punto definido en el modelo y no modifica el valor almacenado en los arreglos almacenados en GPU que definen el modelo.

La matriz de proyección define que puntos de los modelos son los que deben ser mostrados en la escena, modificando las coordenadas de los vértices del modelo, en el vertex shader, de forma que solo los puntos que deben mostrarse en la escena posean coordenadas dentro del rango aceptado por OpenGL (coordenadas con valores entre -1 y 1). Dado esto, si la matriz se encuentra definida de forma incorrecta, por ejemplo, los puntos que se deciden mostrar poseen una proporción que no se corresponde con las proporciones de la ventana en que se muestran, pueden crearse inconsistencias visuales (modelos aplanados o alargados en alguna dirección).

Esto presenta un problema, pues, dado que la aplicación desarrollada es una aplicación de escritorio, el tamaño de la ventana puede ser modificado por el usuario en todo momento, pudiendo provocar inconsistencias en la visualización de los modelos.

Como solución al problema anterior, el programa actualiza la matriz de proyección cada vez que el usuario modifica el tamaño de la ventana de la aplicación, manteniendo de esta forma una correcta visualización de los modelos en la aplicación.

5.4.4. Coloración y shaders

El vertex shader de los modelos en dos dimensiones, además de recibir los vértices del modelo, recibe la altura asociada a cada vértice. Esta altura no es usada por el vertex shader, pero es interpolada y entregada como input al fragment shader, el cual, haciendo uso de la información almacenada en el archivo de colores, le entrega el color correspondiente a cada píxel de la pantalla dependiendo de la altura que este represente.

Dado que la altura es interpolada antes de ser pasada al fragment shader, este ultimo es capaz de designar colores en zonas que no tienen un punto definido en el mapa, ya que estos poseen una altura que es calculada realizando una interpolación de las alturas definidas en la primitiva (el triángulo) que se quiere renderizar.

5.4.5. Actualización de los modelos

La escena es la encargada de actualizar los modelos cuando ocurre algún cambio en la ventana o en el archivo de colores, actualizando los arreglos necesarios para mostrar de forma visual los cambios efectuados.

En el caso de un cambio en la altura de los puntos de un mapa, se actualiza el arreglo de alturas de estos, y, dado que el arreglo de alturas y el arreglo de vértices son distintos, no es necesaria la modificación de los vértices del modelo para representar este cambio, además, dado que la coloración de los mapas ocurre en el fragment shader, esta cambia automáticamente a los colores correspondientes una vez se modifica el arreglo de alturas. Si el cambio es respecto a la posición o nivel de Zoom del programa, se actualiza la matriz de proyección que se usa para el renderizado y que es pasada al vertex shader.

5.5. Renderizado en 3 dimensiones

Para la generación de modelos en tres dimensiones, se utilizan los datos que ya se tienen cargados de los modelos en dos dimensiones, con la diferencia de que estos se procesan de forma distinta para poder visualizar los modelos en tres dimensiones. Además, se utilizan una serie de matrices para lograr la correcta visualización de los modelos en pantalla que dependen tanto de la escena como de la cámara. A continuación se explican a detalle procesos importantes para la generación y visualización de estos modelos en el programa.

5.5.1. Procesamiento de datos

Los modelos en tres dimensiones (instancias de la clase Map3DModel) son generados a partir de modelos en dos dimensiones (instancias de la clase Map2DModel). De estos últimos, los datos necesarios para generar el modelo en tres dimensiones son, el arreglo de vértices de estos, el arreglo de alturas de estos y el archivo de colores.

Como se explicó en la sección 5.4, en los mapas en dos dimensiones, la coordenada z de los

vértices contiene un valor fijo, por lo que es necesaria cambiarla para poder realizar el renderizado en tres dimensiones de los mapas. Gracias al uso de la librería Numpy, este proceso puede realizarse de forma eficiente, ya que Numpy se encuentra programada originalmente en C, manteniendo la eficiencia de los lenguajes compilados.

Posterior al proceso anterior, se genera un arreglo unidimensional con los vértices y se entrega a OpenGL para su posterior renderizado.

De forma análoga al proceso de generación de modelos en dos dimensiones, es necesario especificar cómo se comunican los vértices del arreglo anterior con la finalidad de que OpenGL pueda dibujarlos en pantalla. Para esto, se genera otro arreglo unidimensional que especifica los triángulos a dibujar y que vértices usar para ello y se le entrega a OpenGL. A diferencia del modelo en dos dimensiones, en este caso no se especifica la calidad del renderizado, utilizando todos los vértices del modelo para la generación de los índices de los triángulos que se dibujarán pantalla.

Además de lo anterior, también se genera un arreglo unidimensional con las alturas de los vértices que es entregado como atributo extra a los vértices que se cargan en OpenGL. Este arreglo será utilizado posteriormente en el fragment shader para realizar la asignación de los colores a los vértices.

Es normal que, en los mapas de elevación, la unidad de medida de los puntos en el plano y la elevación sean diferente (por ejemplo, grados o UTM para los puntos en el plano y metros para las alturas de los puntos), por lo que es necesario realizar una conversión con la finalidad de poder visualizar correctamente los modelos. Esta conversión se realiza mediante la multiplicación de un factor a la coordenada z de todos los vértices del modelo en tres dimensiones, cambiando la unidad de este valor a la unidad usada en las coordenadas x e y. Como la coloración de los vértices se realiza usando el arreglo de alturas proporcionado con anterioridad y no la altura de los vértices del modelo, la coloración no se ve afectada por el cambio en las coordenadas de los vértices.

5.5.2. Matrices de modelo, vista y proyección

Con la finalidad de lograr la representación correcta de los modelos en tres dimensiones, es necesario modificar los vértices del modelo para que estos puedan ser vistos desde distintos ángulos. Para esto, se generan una serie de matrices que son usadas en el vertex shader.

A continuación se explican los tres tipos de matrices usadas en el proceso de renderizado en tres dimensiones para poder representar de forma correcta los modelos en la pantalla.

5.5.2.1. Perspectiva

La perspectiva es la forma de representar los objetos en una superficie plana, que da idea de la posición, volumen y situación que ocupan en el espacio con respecto al ojo del observador. En este caso, el objeto es el mapa en tres dimensiones, mientras que el observador es la cámara definida en el programa.

Con la finalidad de dar la sensación de perspectiva a los modelos mostrados en tres dimensiones, esto es, mostrar de forma más pequeña los triángulos que se encuentran más lejos de la cámara y más grandes los que se encuentran más cercanos a la cámara, se genera una matriz que, aplicada a los vértices del modelo en el vertex shader, modifica la posición de estos, entregando la ilusión de perspectiva en la visualización de los modelos.

Para la generación de esta matriz, el modelo en tres dimensiones utiliza información de la cámara, como lo es el campo de visión, e información del tamaño del área de renderizado. Toda esta información es obtenida de la Escena.

5.5.2.2. Vista

El modelo renderizado en tres dimensiones es visto a través de una cámara, esta cámara, puede variar su posición y ángulo desde donde mira al modelo.

Los vértices del modelo son fijos, los arreglos que los definen no varían su contenido, pues, de hacerlo, modificarían la información del modelo directamente. Por esta razón, para modificar los vértices de forma que estos solo varíen en el proceso de visualización, se utiliza una matriz llamada matriz de vista.

La matriz de vista es una matriz que se aplica sobre los vértices de los modelos, en el vertex shader, que se encarga de rotar y trasladar los modelos de tal forma que estos simulen el ser vistos desde la posición en la que se encuentra la cámara.

La generación de esta matriz no depende del modelo que se esté visualizando, pudiendo ser generada solamente con la información de la posición y orientación de la cámara.

El modelo obtiene esta matriz de la Escena, la cual es la encargada de gestionar la cámara de la visualización en tres dimensiones, y la utiliza en el vertex shader para modificar la información de los vértices.

5.5.2.3. Modelo

La matriz de modelo, es una matriz que se usa para modificar la posición, orientación y escala de los modelos de la escena, especificando en que coordenadas del sistema cartesiano que representa la totalidad de la escena se ubican los modelos. Esta posición impacta en donde debe apuntar la cámara del programa para visualizar correctamente los modelos. Esta matriz es útil cuando se tiene más de un modelo en la escena, pudiendo separar la ubicación de estos en el espacio.

En la aplicación implementada, como solo se visualiza un modelo al mismo tiempo, no es necesaria la aplicación de esta matriz, pues, las coordenadas del modelo y su ubicación en la escena son las mismas. Por esta razón, la matriz que se usa como modelo es la matriz identidad en el programa, la cual, no modifica la ubicación de los vértices.

5.5.2.4. Aplicación de las matrices

La aplicación de las matrices toma lugar en el vertex shader, en donde las matrices son aplicadas a los vértices de los modelos de la escena en el siguiente orden, matriz de modelo, vista y perspectiva. Este orden es importante, ya que cada matriz espera recibir como input los puntos ya modificados de la transformación anterior, no siendo intercambiables los ordenes.

Dado que la aplicación de estas matrices ocurre en el vertex shader, esta es efectuada haciendo uso del paralelismo masivo que ofrecen las GPU, resultando esto en un proceso eficiente de renderizado.

5.5.3. Coloración

Para la coloración de los modelos, de forma análoga a los mapas en dos dimensiones, se usa el arreglo de alturas proporcionado a OpenGL con anterioridad junto con la información proporcionada por el archivo de colores para asignar un color a los vértices dependiendo de la altura de estos.

El vertex shader, además de recibir las coordenadas de los vértices, recibe la altura de estos en un arreglo. Este arreglo no es usado por el vertex shader, pero es interpolado y entregado como input al fragment shader, el cual le asigna un color a los píxeles dependiendo de la altura que estos representen.

5.5.4. Cámara

Con la finalidad de que los usuarios puedan visualizar sus modelos desde cualquier posición y ángulo, es necesario la implementación de una cámara que le entregue esta funcionalidad al usuario.

A continuación se define como fue implementada la cámara en el programa.

5.5.4.1. Sistema coordenado

La cámara en el programa fue implementada usando coordenadas esféricas, esto es, el movimiento de la cámara se basa en el cambio en el radio, el ángulo que se tiene en torno al eje z (ángulo azimutal) y el ángulo de elevación, que es cero cuando los puntos están sobre el eje z.

Dado que en la escena solo está renderizando un modelo a la vez, el cual se encuentra en el centro de las coordenadas esféricas, el uso de estas coordenadas permite al usuario la visualización del modelo desde distintos ángulos de forma sencilla, pues, dado que el modelo se encuentra en el centro, el moverse usando ángulos permite rotar en torno al modelo.

5.5.4.2. Movimiento de la cámara

Un problema que tiene el uso de coordenadas esféricas es que estas giran en torno a un punto específico. En este caso, el punto en donde las coordenadas giran es el centro del modelo en 3 dimensiones.

Lo anterior supone un problema, pues, no siempre se quiere visualizar el centro del modelo. Como solución a esto, el programa implementa, particularmente en el Controlador, funcionalidad para mover la posición en donde se ubica el centro de las coordenadas esféricas usadas por la cámara.

La funcionalidad anterior se diferencia de mover el modelo usando la matriz de modelo, ya que, en este caso, como es la cámara la que se mueve, es la matriz de vista la que, aplicada sobre la posición de los vértices, entrega la ilusión de estar moviendo el modelo por la escena.

5.5.5. Factor de exageración

El mapa del mundo, cuando es representado en dos dimensiones, presenta una extensión demasiado grande cuando se compara con la altitud del terreno que este presenta. Debido a esto, si el mapa se representa con la ubicación de sus puntos originales, al visualizar el mapa del mundo completamente, no es posible visualizar con detalle el relieve de los puntos del mapa.

Como solución al problema anterior, el programa implementa un factor de exageración que aumenta las alturas mostradas en el modelo en tres dimensiones, mostrando un relieve que, aunque no realista, permite a los usuarios ver con mayor detalle los relieves del mapa.

Las nuevas alturas del modelo en tres dimensiones son obtenidas multiplicando las alturas originales del modelo por el factor de exageración, modificando el arreglo que se entrega a OpenGL con las alturas de este.

El factor de exageración se aplica después de efectuada la transformación de la unidad de medida de las alturas del modelo, siendo independiente de la unidad de medida que usen estos.

5.5.6. Actualización de la información

La Escena es la encargada de actualizar la información de los modelos en tres dimensiones cuando ocurren cambios en el programa.

Cuando ocurre un cambio en la información del modelo en dos dimensiones, se modifican los arreglos de vértices y de alturas en el modelo en tres dimensiones, mientras que si ocurre un cambio en la posición u orientación de la cámara, se modifica la matriz de vista usada por el vertex shader para representar este cambio en la visualización.

Dado que los modelos en tres dimensiones siempre están usando todos los vértices del modelo en el renderizado, no es necesario modificar el arreglo de índices de los modelos, pues, en ningún momento cambia el número de triángulos renderizados en la pantalla.

5.6. Polígonos

Una de las principales herramientas que ofrece la aplicación desarrollada es la de creación y carga de polígonos. Esta herramienta permite al usuario seleccionar que puntos, del mapa que se encuentra cargado en la aplicación, este desea modificar.

En geología, es normal el uso de herramientas externas para la generación de los polígonos, ya sea mediante la generación de curvas de nivel o usando información de mapas geológicos, como también es normal la especificación de polígonos de forma manual haciendo uso del ratón para agregar puntos a estos.

El formato Shapefile permite almacenar la información de los polígonos junto con atributos que definen metainformación sobre estos. En el estudio de la Tierra, es de vital importancia, con la finalidad de que los estudios puedan ser reproducidos, el almacenar información en los polígonos, tales como, que información estos intentan delimitar, alturas máximas y mínimas, periodo de estudio en el cual fue generado, entre muchas otras características importantes.

En la aplicación desarrollada en esta memoria, y en la mayoría de las aplicaciones de información geográfica, los polígonos son almacenados como un arreglo de puntos en dos dimensiones. En la aplicación desarrollada, asociado a este arreglo de puntos en dos dimensiones, se encuentra un diccionario que almacena los diferentes parámetros que el usuario puede definir para el polígono.

5.6.1. Creación de polígonos

Los polígonos pueden ser creados en la herramienta haciendo uso de la herramienta de polígonos, esta herramienta permite la creación de polígonos utilizando el ratón, agregando un nuevo punto al polígono exactamente en la posición en donde se encuentra el ratón en el mapa que se tiene cargado.

Para poder obtener la posición en la que se encuentra el ratón con respecto al mapa que se está mostrando en la aplicación, se hace uso de los datos que almacena la escena, en particular, se hace uso de los datos que almacenan la información sobre que coordenadas del mapa son las que se están mostrando y sobre la posición que posee la escena con respecto a la ventana de la aplicación. Además de esto, se hace uso de la información que entrega el controlador sobre la posición que tiene el ratón relativa a la ventana de la aplicación. Con estos datos, es posible obtener de forma exacta la posición del ratón en el mapa que se está mostrando en la escena, logrando de esta forma, agregar un nuevo vértice al polígono que se esté creando.

5.6.2. Carga de polígonos

Los polígonos, además de poder ser creados en la aplicación, también pueden ser cargados desde archivos en formato Shapefile.

Como se explica en la sección 2.2.1, los archivos en formato Shapefile permiten, además del almacenamiento de los puntos que componen al polígono, almacenamiento de parámetros asociados a los polígonos.

La aplicación desarrollada utiliza la biblioteca Shapefile del lenguaje de programación Python para realizar la correcta importación de los polígonos en la aplicación, importando los puntos que componen los polígonos junto con los parámetros que estos tienen asociados.

Los archivos en formato Shapefile pueden definir más de un polígono en su interior, pudiendo almacenar un número ilimitado de polígonos. La aplicación desarrollada no presenta problemas con estos archivos, pues, la aplicación importa cada polígono, junto con sus parámetros asociados, de forma independiente.

Dado que los polígonos generados con herramientas externas pueden tener un número ilimitado de puntos en su interior, se hizo uso de la funcionalidad que ofrece la biblioteca Numpy para optimizar el proceso de importación de los polígonos, mejorando de esta forma la experiencia de usuario al utilizar polígonos generados de fuentes externas.

5.6.3. Uso de polígonos simples

Como se explica en la sección 2.2.1, el uso de polígonos simples (polígonos que dividen el plano en dos secciones claramente diferenciables) es bastante común en herramientas GIS. En la herramienta implementada, no se admite la carga o creación de polígonos complejos, informando al usuario en caso de que este este intentando generar un polígono complejo.

Lo anterior genera un problema, pues, en el proceso de creación de un polígono usando el ratón, es posible que en un momento dado los polígonos sean complejos, pero que luego de agregar puntos estos sean simples. En la imagen 5.2 se puede apreciar un polígono complejo que luego de agregar puntos se convierte en un polígono simple.

Como solución al problema anterior, la aplicación desarrollada entrega una advertencia al usuario, que se muestra al lado derecho del nombre del polígono en la aplicación, en forma de un triángulo amarillo, indicándole al usuario que el polígono que esta generando no es simple, y, por ende, no podrá ejecutar la funcionalidad de modificación de puntos.

Para identificar si los polígonos son simples o no, se utiliza la funcionalidad ofrecida por la biblioteca shapely, biblioteca que entrega una funcionalidad para identificar de forma eficiente si un polígono es simple o si no lo es.

En caso de que el usuario intente, deliberadamente, generar un polígono complejo, como lo es en el caso de que el usuario intente agregar puntos al polígono de tal forma que dos

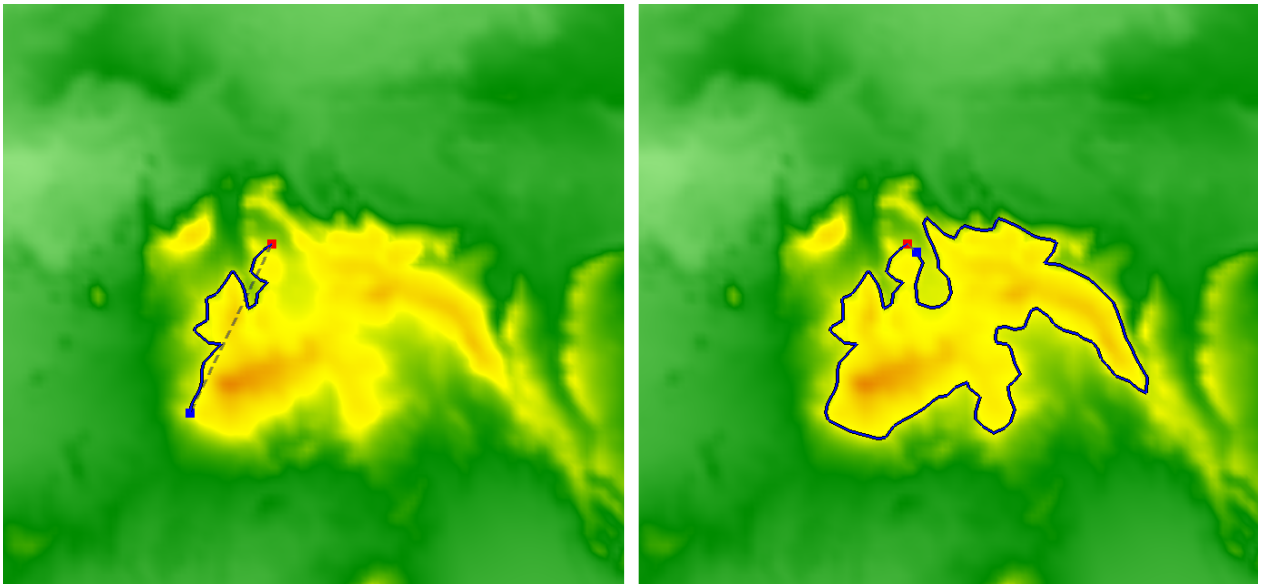


Figura 5.2: A la izquierda, imagen de un polígono complejo generado en la aplicación. A la derecha, imagen del mismo polígono, pero con más puntos que permiten al polígono ser un polígono simple.

líneas del polígono se intercepten, el programa mostrará una ventana de error indicándole al usuario que no es posible agregar el punto en la posición especificada.

Para identificar este último caso, se usa la funcionalidad de intersección que ofrece la biblioteca Shapely, funcionalidad que permite identificar si una línea intercepta con otra línea perteneciente al polígono de forma eficiente.

En caso de que los polígonos se importen desde un archivo Shapefile a la aplicación, el programa solo aceptara polígonos simples, mostrando un mensaje que informe al usuario sobre esto en caso que se intente cargar un polígono complejo en la aplicación. En caso de que el archivo Shapefile defina a más de un polígono, entonces solo se importaran en la aplicación los polígonos que sean simples.

5.7. Selección de los puntos

Con la finalidad de modificar la información de las alturas que se almacenan al interior de los mapas cargados en el programa, el usuario debe hacer uso de polígonos para delimitar claramente que puntos son los que desea modificar y cuáles no. Estos polígonos pueden o bien ser cargados en la plataforma o creados haciendo uso de las funcionalidades que esta ofrece.

Para que la aplicación pueda diferenciar que puntos del mapa son los que se encuentran al interior del polígono y cuáles son los que se encuentran al exterior, se hace uso de un algoritmo que resuelve el problema llamado punto-en-polígono.

El problema de punto-en-polígono consiste en, dado un punto y polígono simple, determinar si el punto pertenece o no al área interna polígono. Este problema posee diferentes soluciones, siendo una de las más eficientes la que consiste en lanzar un rayo desde el punto en cuestión en cualquier dirección y contar la cantidad de lados del polígono con la que este rayo intercepta, estando el punto al interior del polígono si el número de lados interceptados es impar, o estando al exterior si el número es par.

Con la finalidad de resolver el problema anterior de forma óptima para todos los puntos de los mapas, se hace uso de la biblioteca Shapely, biblioteca que ofrece una implementación en paralelo de la solución al problema de punto-en-polígono, permitiendo realizar los cálculos necesarios para determinar si los puntos de los mapas se encuentran al interior o exterior de un polígono de forma eficiente.

Dada la importancia de esta operación en la aplicación implementada, se decidió medir la eficiencia del algoritmo implementado por la biblioteca Shapely. Para esto se generaron varias grillas de puntos con diferentes números de filas y columnas, representando mapas de elevación con diferentes cantidades de datos en su interior, y varios polígonos con un número variado de vértices. Todos los polígonos generados son simples y almacenan un 64 % de los puntos de las grillas en su interior, quedando el 36 % de los puntos de las grillas al exterior de los polígonos.

Se realizaron dos comparaciones principales para medir la eficiencia del algoritmo de punto-en-polígono implementado por la biblioteca Shapefile. La primera consiste en una comparación del tiempo de ejecución del algoritmo sobre todas las grillas generadas utilizando siempre el mismo polígono, esto con la finalidad de analizar como aumenta el tiempo de ejecución del algoritmo en función de la cantidad de datos que poseen los mapas. Mientras que la segunda consistió en una comparación del tiempo de ejecución del algoritmo sobre todos los polígonos generados usando siempre la misma grilla de datos, esto con la finalidad de analizar como aumenta el tiempo de ejecución en función del número de puntos que poseen los polígonos usados.

La finalidad de la aplicación del algoritmo de punto-en-polígono es saber si un punto pertenece a un polígono o no, y por ende, en las dos comparaciones anteriores, el algoritmo es aplicado sobre todos los puntos de las grillas usadas, independiente de si estos puntos están inicialmente al interior o exterior del polígono usado. Los resultados obtenidos de las comparaciones se muestran en las gráficas de las figuras 5.3, 5.4 y 5.5.

Los resultados obtenidos del proceso muestran que el tiempo de ejecución del algoritmo aumenta en función del número de elementos que tienen las grillas analizadas como del número de vértices que poseen los polígonos, en lo que pareciera ser de forma lineal. Esto tiene sentido, pues el algoritmo aplicado por Shapely para la resolución del problema de punto-en-polígono corresponde al algoritmo de lanzar un rayo y contar el número de intersecciones que este posee con el polígono analizado, algoritmo que posee un costo lineal en el número de lados del polígono. Como el algoritmo usado debe ser aplicado para todos los puntos de la grilla analizada, tiene sentido que el tiempo de ejecución del algoritmo también se relacione de forma lineal con el número de puntos que se tiene en la grilla.

Además, los gráficos también muestran que los tiempos de ejecución del algoritmo, tanto para grillas con un gran número de elementos como para polígonos con un gran número de vértices, son buenos.

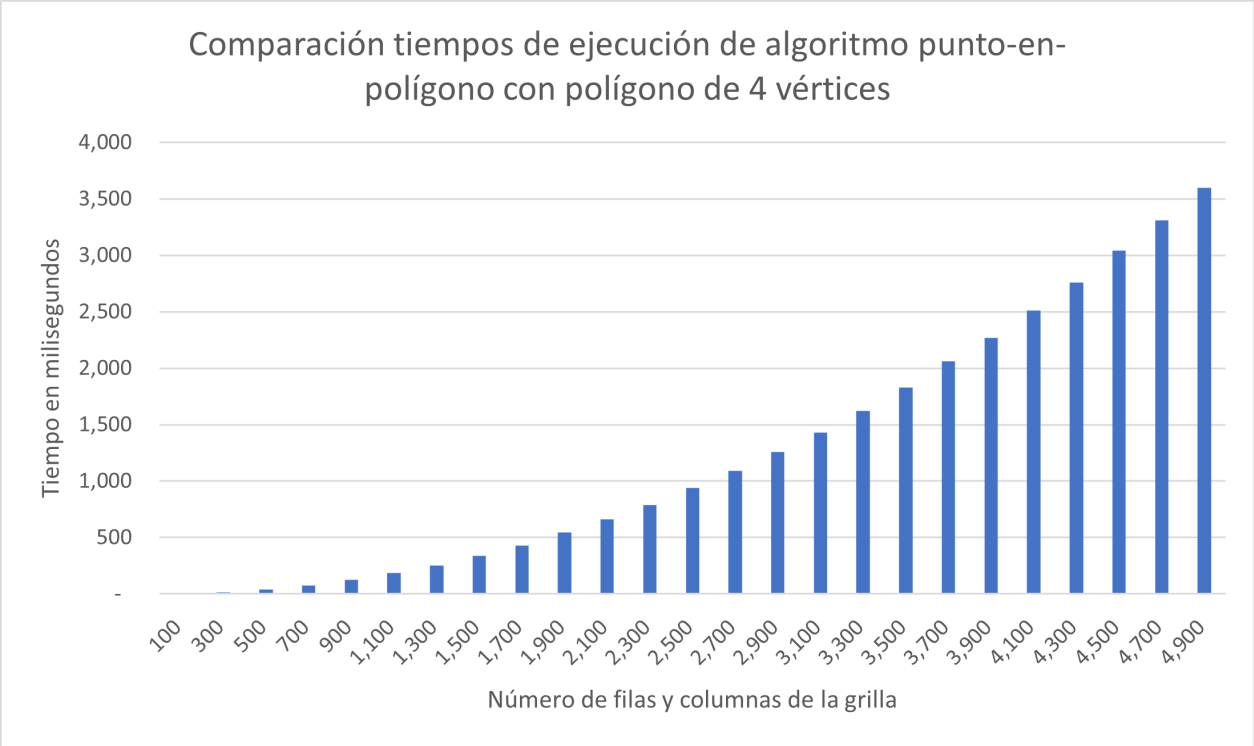


Figura 5.3: Gráfico que muestra el número de filas y columnas de las grillas y el tiempo que demora el algoritmo de la biblioteca Shapely en determinar si los puntos de la grilla se encuentran al interior del polígono analizado.

5.8. Transformación de los puntos y filtros

A continuación, se explica cómo fue implementada la transformación de los puntos y cómo son aplicados los filtros para seleccionar qué puntos del mapa modificar y cuáles no modificar.

5.8.1. Transformación de los puntos

Para modificar los puntos al interior de los polígonos se hace uso de la fórmula que se muestra en la ecuación 6.1. Esta fórmula permite calcular la nueva altura de los puntos usando como dato, además de las constantes definidas por el usuario y calculadas por el programa como lo son las alturas máximas y mínimas, solamente la altura actual de los puntos, siendo perfecta para ser aplicada de forma paralela sobre los puntos de los mapas, y de esta forma, mejorar de forma considerable el tiempo de ejecución que esta operación demora.

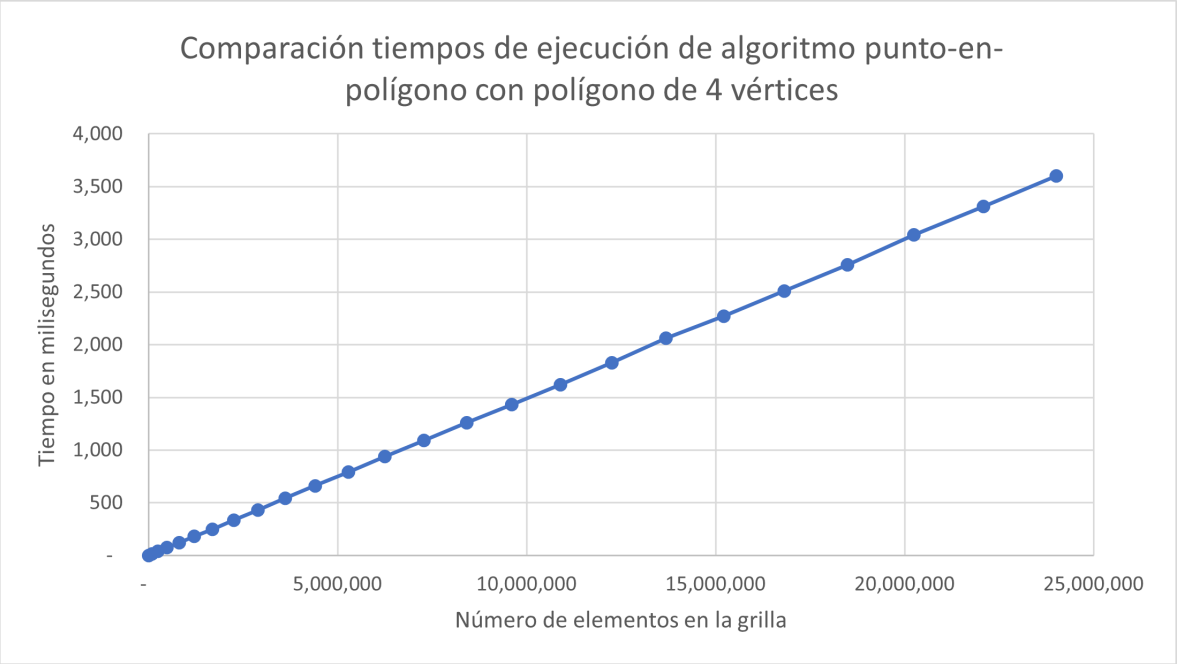


Figura 5.4: Gráfico que muestra el número de puntos almacenados en las grillas y el tiempo que demora el algoritmo de la biblioteca Shapely en determinar si los puntos de la grilla se encuentran al interior del polígono analizado.

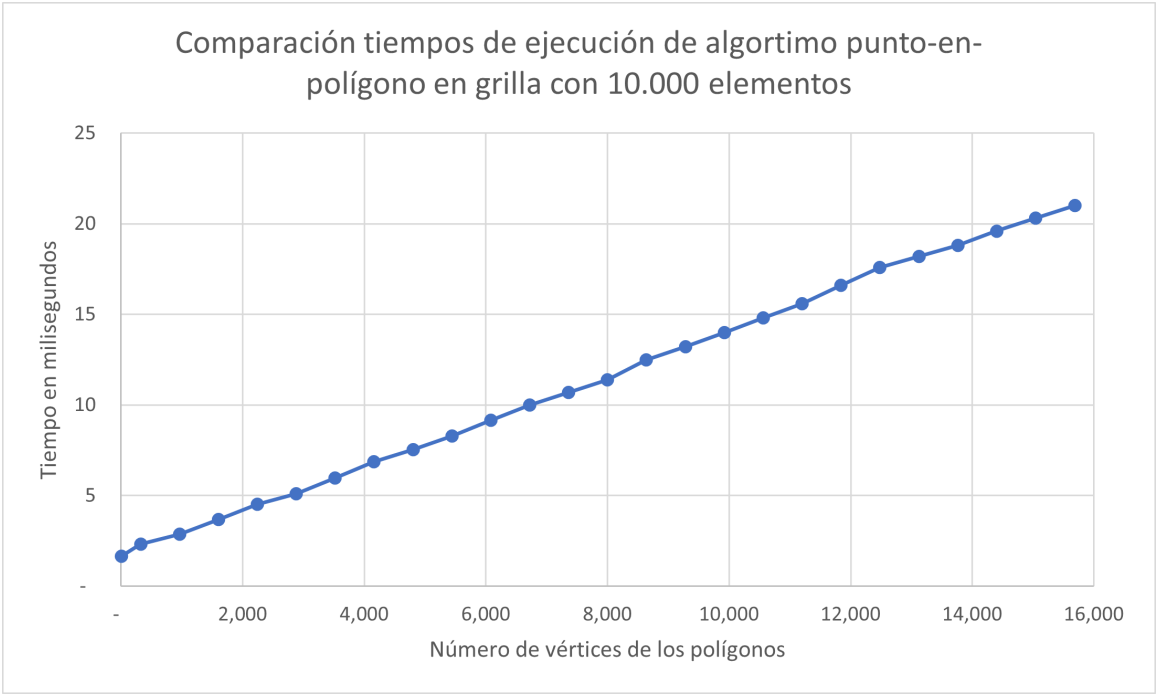


Figura 5.5: Gráfico que muestra el número de vértices de los polígonos analizados y el tiempo que demora el algoritmo punto-en-polígono de la biblioteca Shapely en determinar si los puntos de una grilla se encuentran al interior o exterior del polígono analizado. La grilla utilizada contiene 10.000 puntos.

Con la finalidad de hacer el cálculo de las nuevas alturas de los puntos de forma paralela, se hace uso de la biblioteca Numpy. Esta biblioteca se especializa en cálculo matricial, estando optimizada para el manejo de operaciones matemáticas entre matrices. El uso de esta biblioteca permite realizar el cálculo de las nuevas alturas de los puntos que se encuentran al interior de los polígonos de forma paralela, mejorando de forma considerable el tiempo que demora la operación, y con ello, la experiencia de usuario.

La fórmula presentada en la ecuación 6.1 requiere de 4 parámetros, dos de estos son las alturas máxima y mínima que se espera tener al interior del polígono y que son proporcionadas por el usuario, mientras que los otros dos parámetros son las alturas máxima y mínima de los puntos que se encuentran al interior del polígono. Estas últimas alturas son obtenidas haciendo uso de la biblioteca Numpy, la cual permite realizar el cálculo del mínimo y del máximo de los puntos al interior de una matriz de forma eficiente.

5.8.2. Filtros

A pesar de que los puntos que el usuario pueda modificar puedan ser delimitados mediante polígonos, en variadas ocasiones esto no es suficiente. Esto puede deberse a diversos motivos, entre estos se encuentran: que los puntos a modificar deben cumplir una determinada condición, el polígono necesario para realizar la modificación es muy difícil de dibujar o generar, o simplemente no se necesita modificar todos los puntos al interior de un polígono, sino que solo algunos de los que se encuentran al interior. Como solución a estos problemas se implementaron filtros a la aplicación.

Los filtros implementados en la aplicación son opciones que se ejecutan antes de modificar la altura de los puntos del mapa, que seleccionan bajo diferentes criterios (escogidos por el mismo usuario) que puntos son los que se modificaran y que puntos no se modificaran.

Actualmente la herramienta de filtros cuenta con dos tipos de filtros: filtros de altura, que permiten al usuario seleccionar que puntos modificar dependiendo de la altura que estos posean, y filtros de polígonos, que permiten al usuario seleccionar si modificar o no modificar los puntos que se encuentran al interior de los polígonos que este seleccione.

Con la finalidad de implementar los filtros en la aplicación, se hace uso de la biblioteca Numpy, ya que esta permite la ejecución en paralelo de la lógica que permite la selección de los puntos en matrices, permitiendo realizar la selección de los puntos a modificar de forma eficiente.

El proceso de selección de los puntos ocurre antes de ejecutarse la transformación de los puntos (descrita en la sección anterior). Este genera una máscara, representada en formato de matriz de Numpy, que contiene en su interior información sobre qué puntos deben ser modificados y cuáles no. Esta máscara es usada por el proceso de transformación para aplicar la respectiva fórmula sobre los puntos que correspondan.

Dado que la transformación de los puntos es ejecutada de forma paralela por cada punto al interior del polígono, la aplicación de la máscara generada a partir de los filtros no afecta el

tiempo de ejecución, pues, como la máscara generada y los puntos del mapa son representados en forma de matrices de Numpy, la comprobación de si los puntos deben o no ser modificados es realizada de forma paralela por cada punto.

5.9. Interpolación

En el proceso de modificación de las características de mapas que almacenan las propiedades de la Tierra, como lo son la posición de las placas tectónicas, diferentes condiciones climáticas, o incluso la altura del terreno, es normal el uso de polígonos para la modificación de los datos que estos almacenan.

El uso de polígonos permite delimitar qué partes de los mapas deben ser modificadas de cuáles no deben ser modificadas, permitiendo establecer de forma precisa la zona a modificar. Sin embargo, en el proceso de modificación de alturas de un mapa de paleoelevación, el uso de estos genera un problema, pues, cuando las modificaciones del terreno que se efectúan al interior del polígono difieren mucho de las alturas que se presentan al exterior de este, se genera un terreno con cambios muy bruscos en la altura de los puntos, siendo esto poco natural.

Para solucionar estos problemas, el programa implementa una herramienta que permite al usuario interpolar las alturas de los puntos que se encuentran al exterior del polígono a una distancia específica, ingresada por el usuario, del borde del polígono. Esta interpolación es realizada eliminando las alturas de los puntos del mapa que se encuentran al interior del área de interpolación, área que es mostrada al usuario antes de realizar los cambios, para posteriormente calcularlas, aplicando algoritmos de interpolación en dos dimensiones, usando como base las alturas de los puntos que se encuentran en los bordes del área de interpolación. En la figura 5.6 es posible ver un ejemplo del área de interpolación generada por el programa y como esta es presentada al usuario.

Scipy es una biblioteca de Python que utiliza diferentes bibliotecas de cálculo matemático y científico como Numpy, Sympy y Pandas, entre otras, con la finalidad de entregar un conjunto de herramientas que entreguen soporte a programas y aplicaciones que necesiten funcionalidad relacionada con las matemáticas, ciencias e ingeniería.

Entre las herramientas que entrega Scipy se encuentra la herramienta de interpolación en dos dimensiones. Esta herramienta recibe como entrada una matriz de números, con datos faltantes en algunas celdas, y genera, aplicando algoritmos de interpolación, los datos faltantes en las celdas correspondientes. Scipy permite la aplicación de tres algoritmos de interpolación, lineal, que realiza una interpolación lineal en dos dimensiones de los valores para determinar el valor de las celdas faltantes, nearest, que usa el vecino más cercano para determinar el valor de las celdas faltantes, y cubic, que intenta generar una superficie continuamente diferenciable, con aproximadamente la curvatura mínima necesaria para unir los puntos faltantes con los datos ya existentes, obteniendo los valores faltantes a partir de la superficie generada.

El programa implementado en esta memoria permite al usuario el escoger entre estos tres tipos de interpolación, permitiéndole modificar las alturas que se encuentran al exterior de

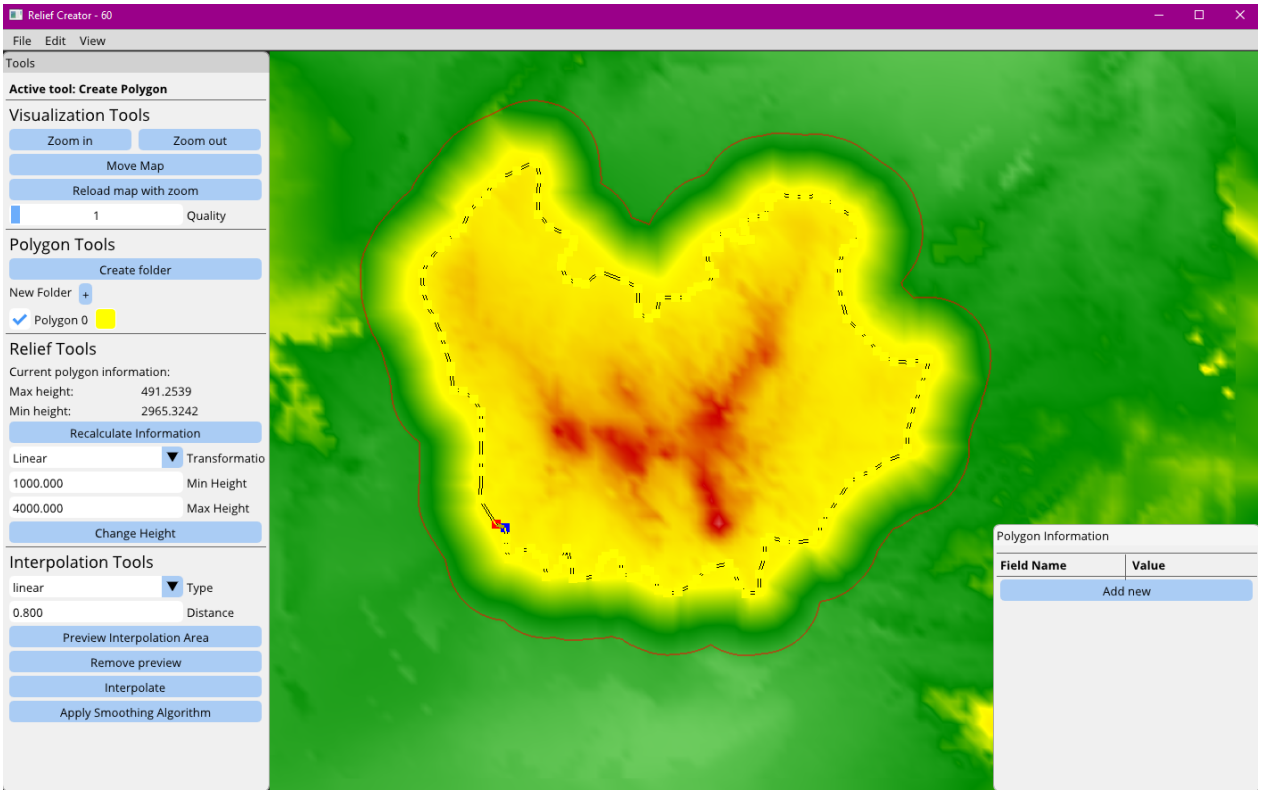


Figura 5.6: Imagen que muestra la aplicación implementada en esta memoria y un polígono sobre el cual fue aplicado el algoritmo de interpolación de la aplicación.

los polígonos de tres diferentes formas.

Cabe destacar que, como el proceso de interpolación primero elimina los puntos que se encuentran al interior de la zona de interpolación, usando solo los puntos del borde de la zona de interpolación como base para la generación de los valores faltantes, la aplicación reiterada de interpolaciones sobre un mismo polígono no afecta a las alturas que este genera, obteniendo siempre los mismos valores.

Scipy usa como base bibliotecas de Python que están originalmente escritas en lenguaje C, siendo estas eficientes al momento de realizar cálculos matemáticos. No obstante, el proceso de interpolación en dos dimensiones es costoso matemáticamente, siendo el tiempo de demora del proceso perceptible al usuario. Esto le es comunicado al usuario mediante ventana que indica que la operación que se está realizando es costosa en tiempo.

5.10. Suavizamiento

La interpolación de la altura de los puntos al exterior de los polígonos ayuda en gran medida en la generación de terrenos con apariencia más natural, sin embargo, en algunos casos, dependiendo de las condiciones del terreno, la interpolación aún puede generar terrenos de apariencia poco natural.

Los filtros de suavizado, usualmente usados en procesos de procesamiento de imágenes con la finalidad de generar un efecto de imagen borrosa, son herramientas que permiten disminuir el gradiente que poseen las imágenes, definiendo gradiente como la diferencia en el color de una celda o píxel con las celdas vecinas, eliminando de esta forma cambios bruscos en el color en las imágenes donde se aplican, generando una imagen que da la impresión de estar difuminada.

Existen distintos tipos de filtros que se aplican sobre imágenes, siendo unos mejores que otros dependiendo de la finalidad para la cual sean usados. Estos tienen la característica de que se aplican en cada celda o píxel, pudiendo ser aplicados de forma paralela a cada celda, resultando esto en una forma eficiente de modificar las imágenes. Los filtros, para determinar el nuevo color de una celda o píxel, necesitan la información de las celdas vecinas a la celda a la cual se le está aplicando el filtro, por lo que, si bien pueden ser aplicados en paralelo, no pueden ser aplicados en la misma imagen, siendo necesario la generación de una nueva imagen temporal con los datos filtrados. Un filtro bastante usado para difuminar imágenes corresponde al filtro gaussiano.

El filtro gaussiano utiliza una función gaussiana para su funcionamiento, determinando a partir de esta que celdas considerar y la importancia que estas tienen en la generación del color de la celda sobre la cual se está aplicando el filtro.

La biblioteca skimage es una biblioteca escrita en Python que entrega herramientas para el procesamiento de imágenes, en particular, entrega herramientas con las cuales poder aplicar filtros sobre imágenes, teniendo definido una variedad de filtros conocidos, entre estos, el filtro gaussiano.

La biblioteca skimage trabaja las imágenes como matrices de valores, esto es, una matriz en donde el valor de cada celda contiene información sobre el color del píxel que representa. Si las imágenes son a color, entonces las celdas contienen un objeto que representa el color en algún formato, usualmente RGB, mientras que, si las imágenes son en escala de grises, entonces las celdas contienen un solo valor que es el que representa en que punto entre el color negro y el blanco la celda se encuentra.

Para aplicar el filtro gaussiano sobre los mapas de elevación, se aplica el filtro proporcionado por la biblioteca skimage sobre los mapas, simulando que los mapas de elevación son imágenes en escala de grises, representando la altura el valor del color de los pixeles. Esto genera un nuevo mapa de alturas, el cual es usado para extraer los valores que deben ser modificados del mapa de elevaciones original, permitiendo esto modificar solo las alturas del mapa que se encuentran al interior de la zona marcada por el programa.

5.11. Input y Output del programa

La aplicación implementada permite la importación y exportación de la información de los mapas en formato Netcdf y de los polígonos en formato Shapefile. A continuación se detalla como se implementó esta funcionalidad junto con nombrar las bibliotecas usadas para lograrlo.

5.11.1. Netcdf

Para importar los archivos en formato Netcdf al programa se utilizó la biblioteca netCDF4, biblioteca escrita en Python que entrega una API para obtener información de los archivos en formato Netcdf.

La biblioteca netCDF4 utiliza una serie de bibliotecas escritas en C para lograr que la lectura de los archivos se haga de forma eficiente, aprovechando la eficiencia de los lenguajes compilados. Esta biblioteca permite la fácil y eficiente lectura y escritura de archivos en formato Netcdf.

5.11.1.1. Exportación

Para realizar la exportación de los modelos modificados en el programa en formato Netcdf, es necesario tener los arreglos que definen las coordenadas usadas en el mapa junto con la matriz de alturas de este. Los arreglos que definen las coordenadas se encuentran ya almacenados en el modelo en dos dimensiones, mientras que la matriz de altura se almacena en formato de arreglo unidimensional en el programa y debe ser modificada para ser correctamente exportada.

Luego de convertir los arreglos al formato de exportación adecuado, la información es exportada usando la biblioteca netCDF4, generando un archivo en formato Netcdf, permitiendo el uso de este en otros programas de información geoespacial.

5.11.1.2. Importación

Los archivos en formato Netcdf almacenan en su interior información en forma de arreglos. Estos arreglos tienen, además de su contenido, información que ayuda a identificarlos, como la cantidad de datos que estos poseen y etiquetas que ayudan a determinar la finalidad de su contenido.

Los mapas de elevación tienen en su interior, usualmente, tres arreglos principales. Dos de estos arreglos son usados para definir las medidas de los ejes coordenados que usa el mapa, por ejemplo, latitud y longitud, y un tercer arreglo define las alturas de los puntos del mapa. No existe una limitante a la unidad de medida que usen estos arreglos, pudiendo incluso no especificar que unidad de medida utilizan los arreglos.

Lo anterior genera un problema al momento de importar los arreglos, pues, para poder representar el mapa correctamente en la escena, hay que reconocer que arreglo del archivo debe alinearse con el eje y del sistema coordenado y cuál con el eje x, pues, de otra forma, la representación de las alturas en el mapa resultarían erróneas.

Como se dijo anteriormente, los arreglos almacenados en los archivos Netcdf contienen meta-información sobre el contenido que estos almacenan, encontrándose entre estos un identificador de los arreglos que sirve para saber que representa la información del arreglo (por ejemplo, x si el arreglo almacena la información usada en el eje x, o *latitude* si el arreglo

contiene las latitudes usadas en el mapa). Como solución al problema de identificación de los arreglos, el programa implementa listas, fácilmente modificables, que poseen los identificadores de los arreglos usados para las latitudes y longitudes de distintos mapas en formato Netcdf. De esta forma, el programa, al leer un nuevo archivo Netcdf, solo debe buscar en las listas los identificadores almacenados en el archivo para saber si los arreglos representan la información del eje x, eje y o la altura de los puntos.

Lo anterior fue testeado usando varios archivos Netcdf de orígenes distintos, esto con la finalidad de hacer que el programa acepte la mayor cantidad de archivos diferentes sin la necesidad de que el usuario tenga que intervenir el programa.

Luego de identificados los arreglos, el programa utiliza la biblioteca netCDF4 para realizar la lectura de los archivos, obteniendo como resultado dos arreglos y una matriz. Estos serán usados en procesos posteriores para representar el modelo en dos y tres dimensiones en el programa.

5.11.2. CPT

Los mapas paleogeográficos, dado que son visualizados en dos dimensiones, necesitan de algún tipo de coloración que entregue información sobre la altura que estos representan. No existe una coloración de las alturas de los mapas de elevación estándar, variando esta dependiendo de los mapas o la característica en particular que se quiera visualizar en estos. Con la finalidad de colorear los mapas, la aplicación hace uso de archivos en formato CPT (Color Pallete Tables), los cuales contienen en su interior información sobre la coloración que deberían recibir los puntos de los mapas de paleoelevación en relación con la altura que estos poseen.

El formato de archivo CPT es un formato bastante usado en software GIS para realizar la coloración de mapas de elevación. En su interior se especifica una tabla en donde cada fila de esta contiene información sobre la altura inicial y el color con el cual debería ser representada y la altura final con el color que debería ser representada. Los valores de altura que están entre las dos alturas especificadas en la fila obtienen un color interpolado entre los dos valores especificados. No hay límite de filas para la tabla que se define en estos archivos. En el cuadro 5.1 puede verse un ejemplo de las tablas almacenadas en el interior de este tipo de archivos.

Tabla 5.1: Cuadro de ejemplo representando el contenido almacenado al interior de un archivo CPT.

Altura inicial	Color	Altura final	Color
0	(0, 255, 0)	1000	(0, 255, 255)
1000	(0, 255, 255)	2000	(100, 255, 100)
...	...	...	...

La aplicación implementada en esta memoria, con la finalidad de entregar al usuario la libertad de escoger los colores con los cuales visualizar los mapas cargados en la aplicación,

implementa un parser de archivos en formato CPT que permite la lectura de estos para la posterior coloración de los modelos.

El programa contiene un archivo de coloración por defecto, el cual se usa para visualizar los mapas que se carguen en la plataforma, sin embargo, el programa también implementa funcionalidad que permite a los usuarios el usar sus propios archivos de coloración en los mapas, modificando la visualización de los mapas.

5.11.3. Shapefile

Para la importación y exportación de archivos en formato Shapefile se usó la biblioteca pyshp, biblioteca que entrega una API para el manejo de archivos en este formato.

5.11.3.1. Importación

Los archivos en formato Shapefile definen en su interior información de tipo vectorial, esto es, puntos, lineas y polígonos. Además, estos almacenan meta-información sobre las formas almacenadas en el interior, como lo son descripciones de los polígonos, significado de los puntos o identificadores de las formas almacenadas.

En la aplicación implementada, los polígonos generados con esta no pueden tener líneas que se intercepten entre sí, pues, dado que los polígonos serán usados para la modificación de los mapas, esto generaría problemas en la identificación y selección de los puntos del mapa.

Los archivos en formato Shapefile pueden contener en su interior más de una sola forma, esto es, un solo archivo puede contener varios polígonos, puntos y lineas. Además, los archivos también pueden almacenar polígonos no planares en su interior, polígonos que no pueden ser usados en la aplicación. Dado lo anterior, se vuelve necesario filtrar la información de los archivos importados antes de poder cargarlos en el programa.

El programa, antes de cargar las formas almacenadas en los archivos Shapefile, realiza un filtro verificando si es que estas son polígonos, en caso de no ser, la forma no se carga en el programa, entregándole un aviso al usuario sobre esto. En caso de ser un polígono, la aplicación comienza a recrear el polígono internamente usando los vértices definidos en el archivo Shapefile, en caso de que ocurra un error en el proceso, ya sea relacionado al hecho de que existen puntos repetidos en el polígono o el polígono no es planar, el polígono no se carga en la plataforma, entregándole un aviso al usuario sobre esta situación y el motivo de porque no pudo ser cargado el polígono. El proceso anterior es independiente para cada forma almacenada en el archivo Shapefile, no afectando un error en una forma al proceso de importación de otra.

Posterior al proceso de importación, los polígonos quedan definidos al interior del programa, dando al usuario la libertad de visualizar y modificar la información que estos almacenan en su interior.

5.11.3.2. Exportación

Para la exportación de los polígonos a archivos en formato Shapefile se usa también la biblioteca pyshp. Esta ofrece una API para definir los polígonos que se quieren almacenar en los archivos, permitiendo la escritura de información asociada a estos.

La aplicación permite la exportación de polígonos generados en esta de forma separada, generando un archivo en formato Shapefile con el polígono y sus datos asociados, como también permite la exportación de los polígonos en conjunto, esto es, almacenando un grupo de polígonos en un mismo archivo Shapefile con sus datos asociados.

5.12. Testing

Con la finalidad de comprobar el correcto funcionamiento de la aplicación, se realizaron dos tipos de tests sobre esta, tests unitarios, escritos en código en lenguaje Python que comprueban que los diferentes componentes del programa ejecuten la funcionalidad que ofrecen de forma correcta, y tests funcionales, ejecutados por personas, que comprueban empíricamente que el programa funciona correctamente.

5.12.1. Tests unitarios

La aplicación contiene implementados tests unitarios que comprueban la funcionalidad de forma lógica de los distintos componentes de esta. Estos tests pueden ser ejecutados mediante la línea de comandos y, a no ser que algún componente varíe su comportamiento, estos entregan siempre el mismo resultado.

Este tipo de testing es bueno, ya que, al momento de extender los componentes con nuevas funcionalidades, permite comprobar que las funcionalidades que se encontraban anteriormente en el programa mantienen su funcionamiento, y con ello, comprobar que no se introdujeron errores en el programa.

Para la implementación de los tests unitarios se utilizó la biblioteca unittest que ofrece el lenguaje Python. Esta biblioteca entrega una API que permite definir tests mediante la creación de clases que hereden de la clase TestCase, clase que entrega la biblioteca.

Si bien los tests unitarios comprueban el funcionamiento del sistema, existen componentes dentro del programa que no pueden ser testeados haciendo uso de tests unitarios, ya sea porque estos componentes solo realizan lógica que modifica elementos visuales, no pudiendo representar su comportamiento con elementos internos del programa, por que son componentes que requieren de la interacción directa del usuario con el programa, o por algún otro motivo. Es por esto que se hace necesario, en conjunto con los tests unitarios, la ejecución de tests de funcionamiento que involucren personas.

Un ejemplo de los tests unitarios programados en la aplicación se encuentra en la sección A.6 de los anexos.

5.12.2. Tests de funcionamiento

Con la finalidad de testear la funcionalidad que requiere de la interacción directa del usuario, se realizaron tests de funcionamiento. Estos tests consistieron en que personas utilizaran el programa para realizar ciertas labores en específico y detectaran si es que el programa funcionaba correctamente o si este tenía algún tipo de bug.

Este tipo de tests sirve para comprobar la funcionalidad que no puede ser testeada haciendo uso de tests unitarios, por ejemplo, comprobar que la GUI muestra los elementos en el orden correcto o que el programa muestra los mensajes de errores respectivos cuando el usuario realiza alguna acción incorrecta, comprobando de esta forma la funcionalidad de los componentes que los test unitarios no pueden.

Capítulo 6

Solución implementada y validación

En este capítulo se describe la solución implementada en esta memoria de forma general junto con explicar como se realizó la validación de esta, esto es, como se comprobó que los resultados que esta entrega son los esperados.

En la sección 6.1 se explican de forma general las características que posee la solución implementada en esta memoria. En la sección 6.2 se explica de forma general como usar la aplicación desarrollada. En la sección 6.3 se explica como se realizó la validación de la solución y las métricas usadas.

6.1. Características de la solución implementada

Dados los problemas que se especifican en la sección 3.1, la solución implementada en esta memoria debe cumplir, en general, con ser una aplicación gratuita, fácil de usar y que permita a los geólogos y científicos el modificar la elevación de mapas de paleoelevación. Además, debe permitir visualizar los mapas y sus cambios en tres dimensiones.

Lo anterior permite a la aplicación reemplazar el proceso actual efectuado por los geólogos y científicos para la modificación de los mapas de elevación, en donde utilizan una o más herramientas, de dificultad elevada de uso, no diseñadas para tal labor.

A continuación, se explican, de forma general, las características de la solución implementada que solucionan los problemas nombrados en la sección 3.1.

6.1.1. Aplicación interactiva

La aplicación desarrollada corresponde a una aplicación interactiva, esto significa que el usuario puede interactuar en tiempo real con la aplicación y recibir respuesta inmediata de esta.

Este tipo de aplicaciones es ideal para personas con bajo conocimiento en programación, ya que abstrae el funcionamiento e implementación de la aplicación del modo de uso de esta, no necesitándose ningún conocimiento previo para hacer uso de la aplicación, pues, la

aplicación le informa al usuario de los posibles problemas que ocurran en el proceso.

6.1.2. Modificación de alturas

La aplicación implementada utiliza polígonos para la modificación de las alturas de los mapas de elevación, permitiendo a los usuarios visualizarlos junto a los mapas de paleoelevación que se carguen en la aplicación.

Los polígonos pueden ser importados mediante un archivo Shapefile, o bien pueden ser dibujados de forma interactiva en la misma aplicación usando el ratón y haciendo clic en el mapa que se quiera usar como base para la posición de los puntos. Los polígonos se muestran sobre el mapa que esté cargado en el programa, permitiendo al usuario el identificar y diferenciar las zonas que son afectadas por el polígono y las que no son afectadas.

La modificación de las alturas se realiza mediante la aplicación de una función lineal a la altura de todos los puntos del mapa que se encuentren al interior del polígono especificado. La función lineal aplicada depende de la altura máxima y mínima de los puntos que se encuentren al interior del polígono, junto con la nueva altura máxima y mínima deseada que se quiere obtener después de la modificación, esta se realiza aplicando la siguiente formula:

$$nueva\_altura = (altura - mínimo_{actual}) * \frac{maximo_{deseado} - minimo_{deseado}}{maximo_{actual} - minimo_{actual}} + mínimo_{deseado} \tag{6.1}$$

En donde *nueva_altura* es la nueva altura que tendrá el punto analizado y *altura* es la altura actual del punto.

La aplicación de esta función sobre los puntos al interior de los polígonos modifica el terreno de tal forma que, los puntos al interior del polígono cambian su altura máxima y mínima, pero a su vez mantienen el relieve relativo que tenían entre ellos mismos antes de la aplicación de la función, esto es, si un punto P es más alto que otro punto Q y menos alto que otro punto R, entonces, luego de aplicada la transformación, el punto P seguirá siendo más alto que el punto Q y menos alto que el punto R, logrando crear un nuevo relieve en el mapa sin aplanar el terreno completamente.

Además de lo anterior, el programa permite el uso de filtros que permiten seleccionar, de los puntos que se encuentran al interior del polígono, que puntos modificar y cuáles no, permitiendo la selección de puntos según su altura (modificar los puntos cuya altura es mayor o menos a un valor dado) o si estos se encuentran en algún otro polígono (por ejemplo, modificar todos los puntos que se encuentran al interior del polígono seleccionado pero que no se encuentran al interior de otro polígono).

Como se explica en el capítulo 2, un formato bastante usado para almacenar información de paleoelevación corresponde al formato de archivo Netcdf, mientras que un formato bastante usado para almacenar información de polígonos es el de Shapefile. Es por esto que, con la finalidad de que la aplicación desarrollada se adapte de la mejor forma posible con los otros

procesos y modificaciones que realizan los geólogos y científicos a los mapas de paleoelevación en la actualidad, el programa implementado acepta tanto la importación como la exportación de mapas de paleoelevación en formato Netcdf y de polígonos en formato Shapefile.

En la figura 6.1 se muestra un ejemplo del resultado obtenido al modificar las alturas de los puntos al interior de un polígono usando el programa implementado en esta memoria.

6.1.3. Visualizacion en 3D

Como se detalla en la sección 5.5, la aplicación implementada en esta memoria permite la visualización de los mapas paleogeográficos en tres dimensiones. Esto permite a los usuarios de la aplicación el poder verificar de forma más sencilla el efecto que tienen las modificaciones que estos realizan en los mapas cargados en la aplicación.

6.1.3.1. Exageración de las alturas

Debido a lo grande que es el mapa terrestre en comparación con las alturas que este presenta, es necesario exagerar de alguna forma la elevación de los terrenos con la finalidad de que los usuarios de la aplicación sean capaces de ver en tres dimensiones los relieves que presenta el terreno.

Para esto, la aplicación desarrollada contiene una opción que permite a los usuarios el exagerar las elevaciones del terreno, permitiendo a los usuarios el observar con más detalle las elevaciones del terreno y los cambios que estos realicen sobre los mapas.

6.1.3.2. Conversión de unidades

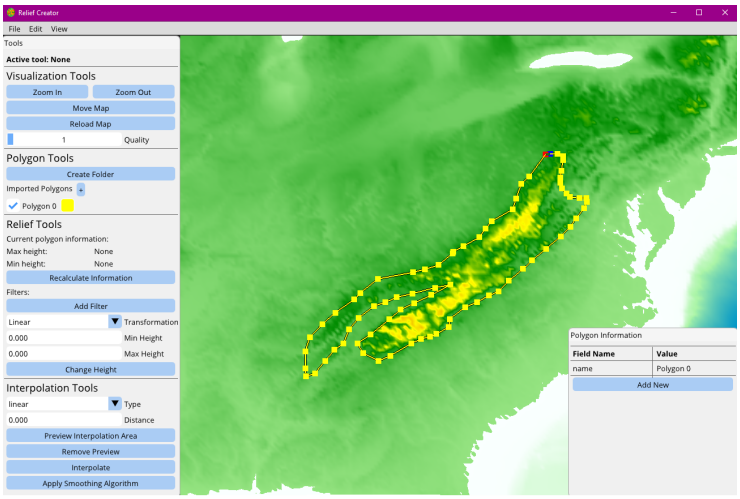
No todos los mapas de elevación utilizan las mismas unidades. Por otro lado, es normal que los mapas de elevación utilicen una unidad de medida para definir los puntos del mapa, grados o UTM por ejemplo, y otra unidad de medida para definir la altura de estos.

Es por esto que, para representar el modelo en tres dimensiones de forma precisa, es necesario realizar la conversión de unidades de la altura de los puntos para que coincida con la medida usada en la posición de los puntos.

Para solucionar este problema, el programa permite a los usuarios, en el menú de visualización en tres dimensiones, el especificar las medidas usadas en el mapa, convirtiendo automáticamente la posición de los puntos en la visualización a la medida especificada.

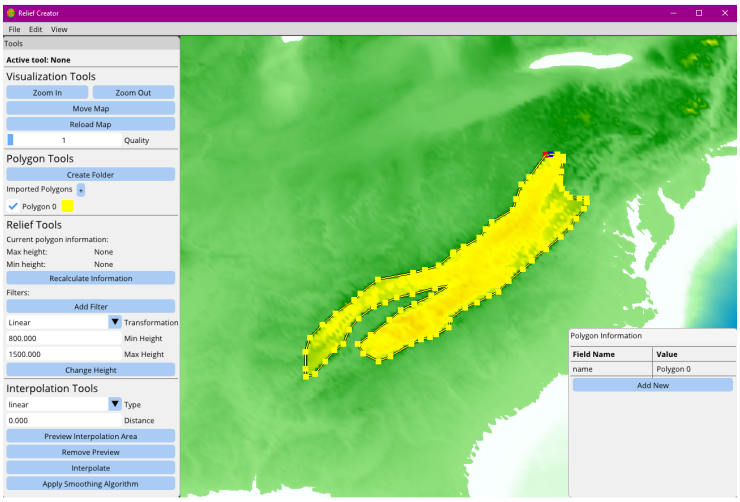
6.1.3.3. Cámara

Uno de los beneficios que tiene el representar los mapas en tres dimensiones es el poder ver las alturas de los mapas desde diferentes ángulos. Para esto, es necesario una cámara interactiva que permita al usuario el poder moverse alrededor de los mapas.



A) Original

B) Alturas modificadas entre 185 y 414 metros



C) Alturas modificadas entre 800 y 1500 metros

Figura 6.1: Imagen A, mapa original, sin modificaciones y el polígono usado para realizar las modificaciones de alturas. Imagen B, mapa modificado usando el polígono mostrado y seleccionando 185 y 414 metros como alturas mínimas y máximas respectivamente. Imagen C, mapa modificado usando el mismo polígono seleccionando 800 y 1500 metros como alturas mínimas y máximas respectivamente.

La aplicación implementa una cámara interactiva que permite al usuario el visualizar los modelos desde diferentes ángulos. En la figura 6.2 es posible ver ejemplos de la visualización en tres dimensiones de los mapas generados por la aplicación.

6.1.4. Interpolación

La aplicación ofrece también dos formas de mejorar la calidad de las modificaciones, permitiendo la generación de mapas que parezcan más naturales, la primera corresponde a una herramienta de interpolación de puntos, mientras que la segunda corresponde a una herramienta de suavizado de las alturas. En esta sección se describirá la herramienta de interpolación.

La herramienta de interpolación de puntos es una herramienta que permite interpolar de diferentes formas los puntos que se encuentran al exterior del polígono seleccionado con los puntos que se encuentran al interior del mismo polígono. Esta herramienta modifica los puntos al exterior de los polígonos, haciendo uso de algoritmos de interpolación en dos dimensiones, con la finalidad de poder generar una conexión más natural entre el terreno modificado al interior del polígono utilizado y el terreno que se encuentra al exterior del polígono.

La aplicación permite al usuario especificar la distancia a usar para realizar la interpolación de los puntos, permitiendo también realizar una previsualización del área de interpolación que se utilizará.

En la figura 6.3 se muestran los resultados obtenidos de la interpolación de un polígono usando el programa.

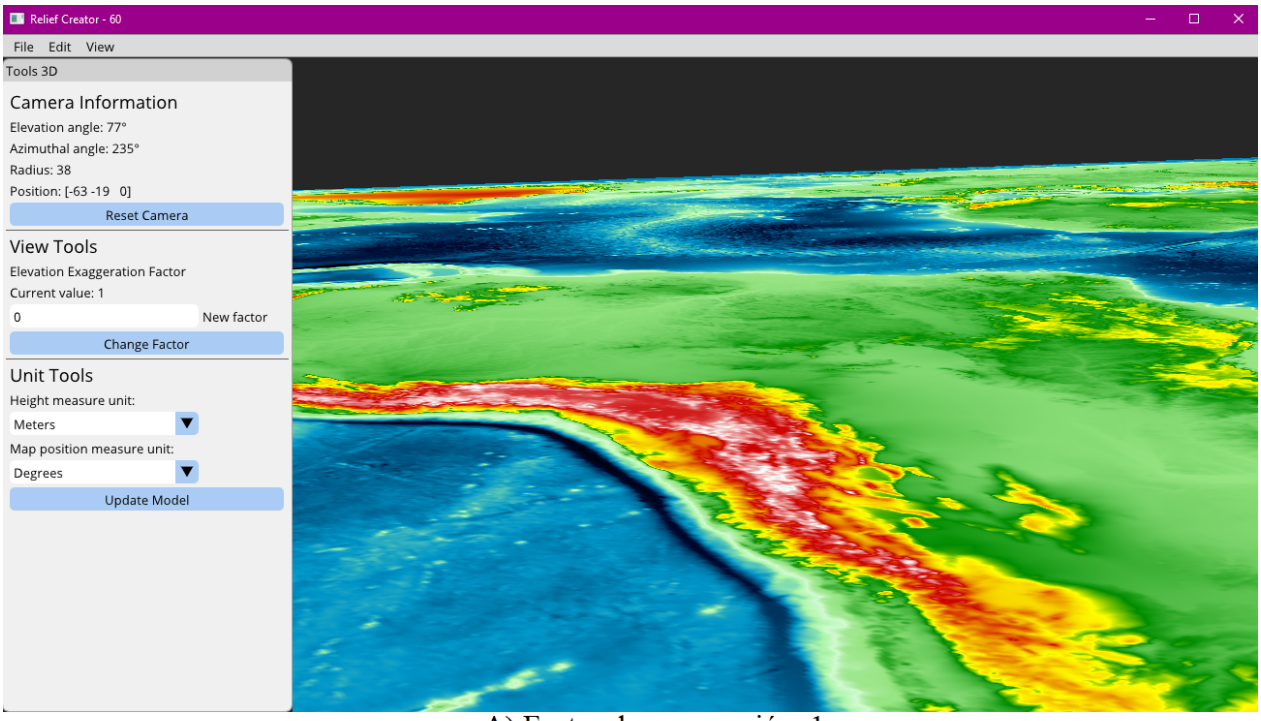
6.1.5. Suavizado

Incluso después de aplicada la interpolación de los puntos al exterior de los polígonos, es posible que, en algunos casos, debido a las características del terreno, aún resulten terrenos que poseen características poco naturales.

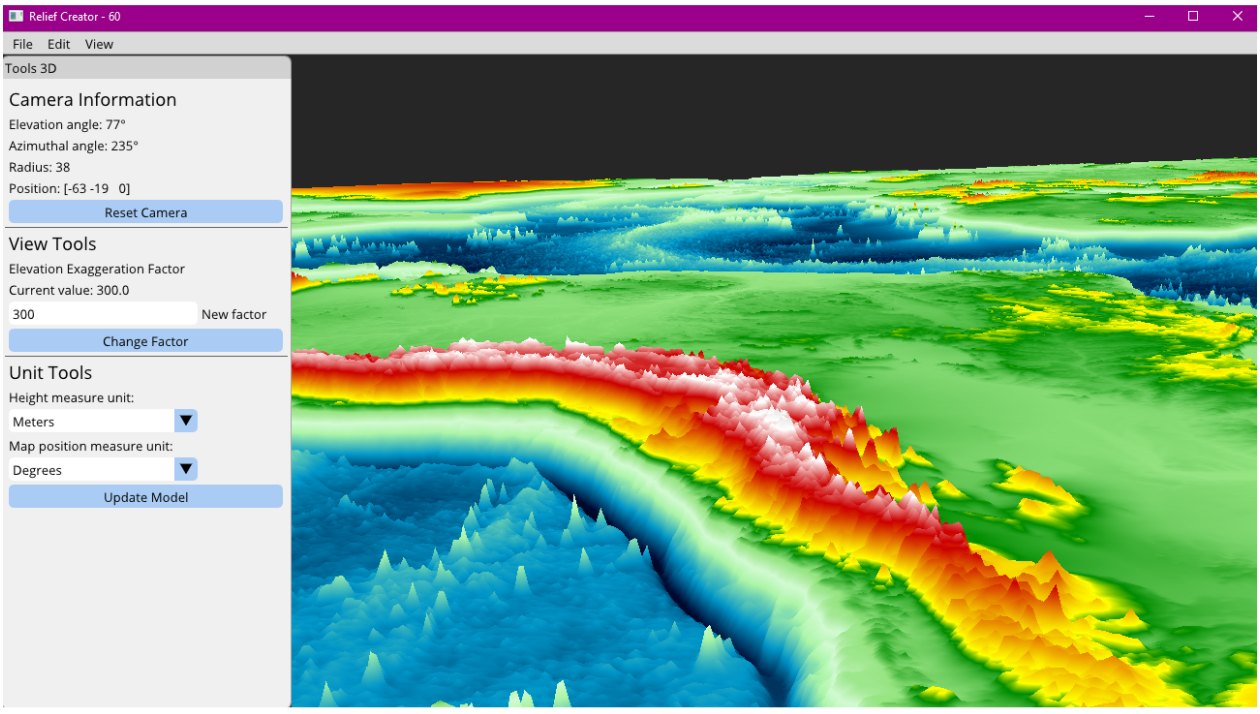
Como solución a estos problemas, y como alternativa a la herramienta de interpolación, se implementó en el programa una herramienta de suavizado.

La herramienta de suavizado ejecuta un filtro gaussiano sobre los puntos al exterior del polígono, modificando la misma área que modifica el realizar el proceso de interpolación. El filtro gaussiano disminuye las repentinas diferencias del terreno en las zonas donde se aplica, generando con esto el efecto de una superficie más lisa.

Esto difiere de la interpolación, ya que la interpolación genera nuevos valores de alturas a partir de las alturas que presentan los puntos que se encuentran al exterior del área de interpolación, mientras que la herramienta de suavizado calcula la nueva altura de los puntos a partir de las alturas ya existentes en el terreno, utilizando los valores presentes en el área de suavizado para calcular la nueva altura de los puntos.



A) Factor de exageración: 1



B) Factor de exageración: 300

Figura 6.2: Imagen A, previsualización en tres dimensiones sin exageración de las alturas, Imagen B, previsualización en tres dimensiones exagerando las alturas por un factor de 300.

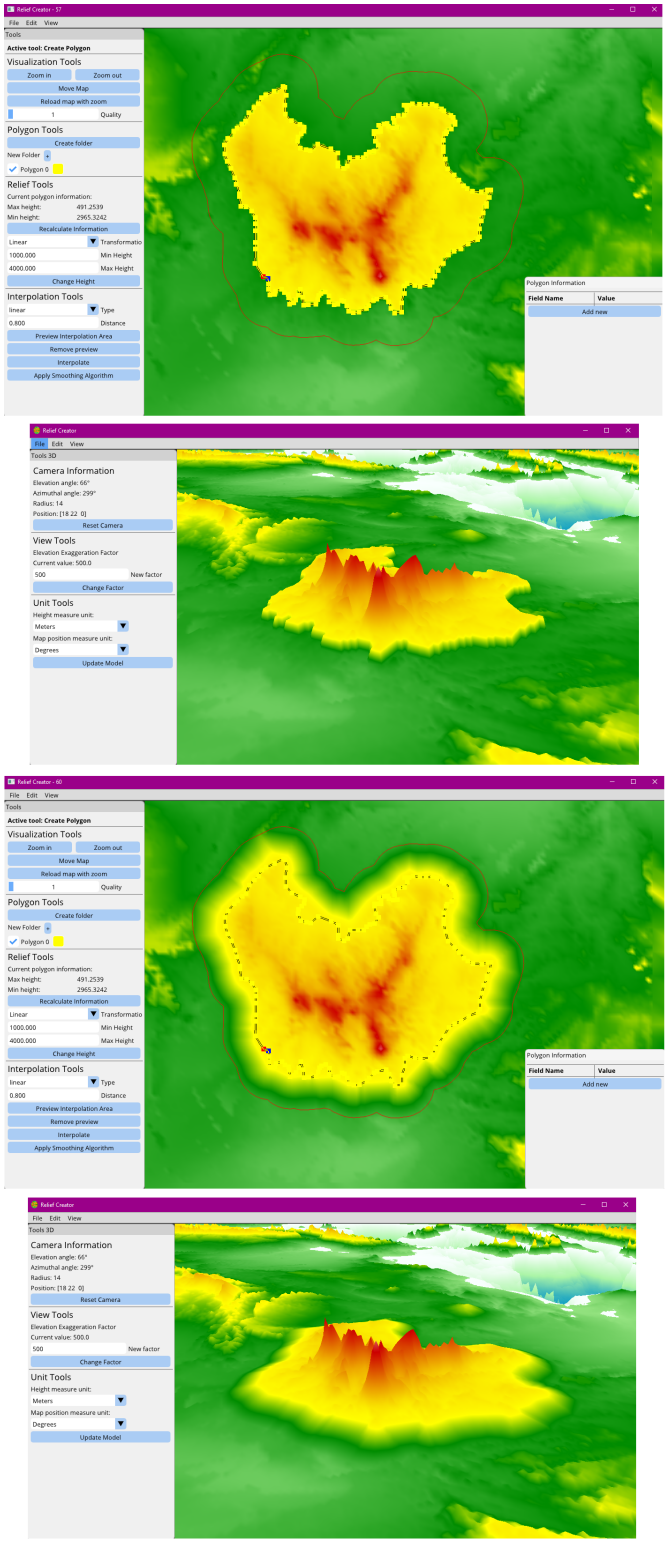


Figura 6.3: En la parte superior, imagen del programa con un polígono que tiene su altura modificada y área de interpolación marcada, segunda imagen de arriba hacia abajo, modelo 3D de la sección modificada, tercera imagen de arriba hacia abajo, imagen del programa con puntos del borde del polígono interpolados de forma lineal, en la parte inferior, modelo 3D de la sección modificada e interpolada.

En la figura 6.4 puede observarse un ejemplo del uso de suavizado en el programa. Más detalles sobre cómo funciona el filtro de suavizado se explican en el capítulo 4.

6.2. Modo de uso de la aplicación

A continuación, se presenta un resumen del modo de uso de la aplicación implementada, mostrando como efectuar las principales funcionalidades de esta. Instrucciones completas sobre el uso de la aplicación se encuentran en la sección A.1 de los anexos.

6.2.1. Carga de mapas

Para cargar mapas en la plataforma, hay que hacer uso del menú *File* que ofrece el programa en la barra de navegacion (mostrado como número 7 en la figura A.1) y luego hacer clic en la opción *Load NetCDF file...* o hacer uso del atajo de teclado *CTRL+O*. Esto abrirá una ventana interactiva (que varía dependiendo del sistema operativo en donde se ejecute el programa) que permitirá al usuario el seleccionar el archivo que desea cargar.

Una vez cargado el mapa, es posible usar las herramientas de visualización mapa mover o acercar el mapa (mostradas como número 2 en la figura A.1).

Es posible hacer un acercamiento en el mapa haciendo uso de los botones *Zoom In* y *Zoom Out* que se muestran en el menú de visualización, o haciendo *Scroll* con el mouse.

Es posible mover el mapa que se está visualizando usando las teclas *WASD*, o haciendo clic sobre el botón *Move Map* y arrastrando el mouse por sobre el mapa.

Es posible recargar la visualización del mapa haciendo clic en el botón de *Reload map with zoom*, o presionando la tecla *R*. Esto recarga la visualización del mapa en mayor detalle, ajustándose al nuevo nivel de zoom que se tenga en el mapa. La calidad de esta visualización puede ser modificada cambiando el valor del parámetro *Quality* en el menú de visualización, siendo 1 la mejor calidad posible y 30 la peor.

6.2.2. Creación y carga de polígonos

Tanto para la creación como para la carga de un polígono es necesario que el programa tenga un mapa cargado. En caso contrario, el programa mostrará mensajes de error indicando que es necesario cargar un polígono antes de realizar las acciones.

6.2.2.1. Creación de polígonos

Para la creación de un polígono, hay que presionar el botón *Create folder*, ubicado en el menú de herramientas de polígonos (mostrado como número 3 en la figura A.1). Esto generará

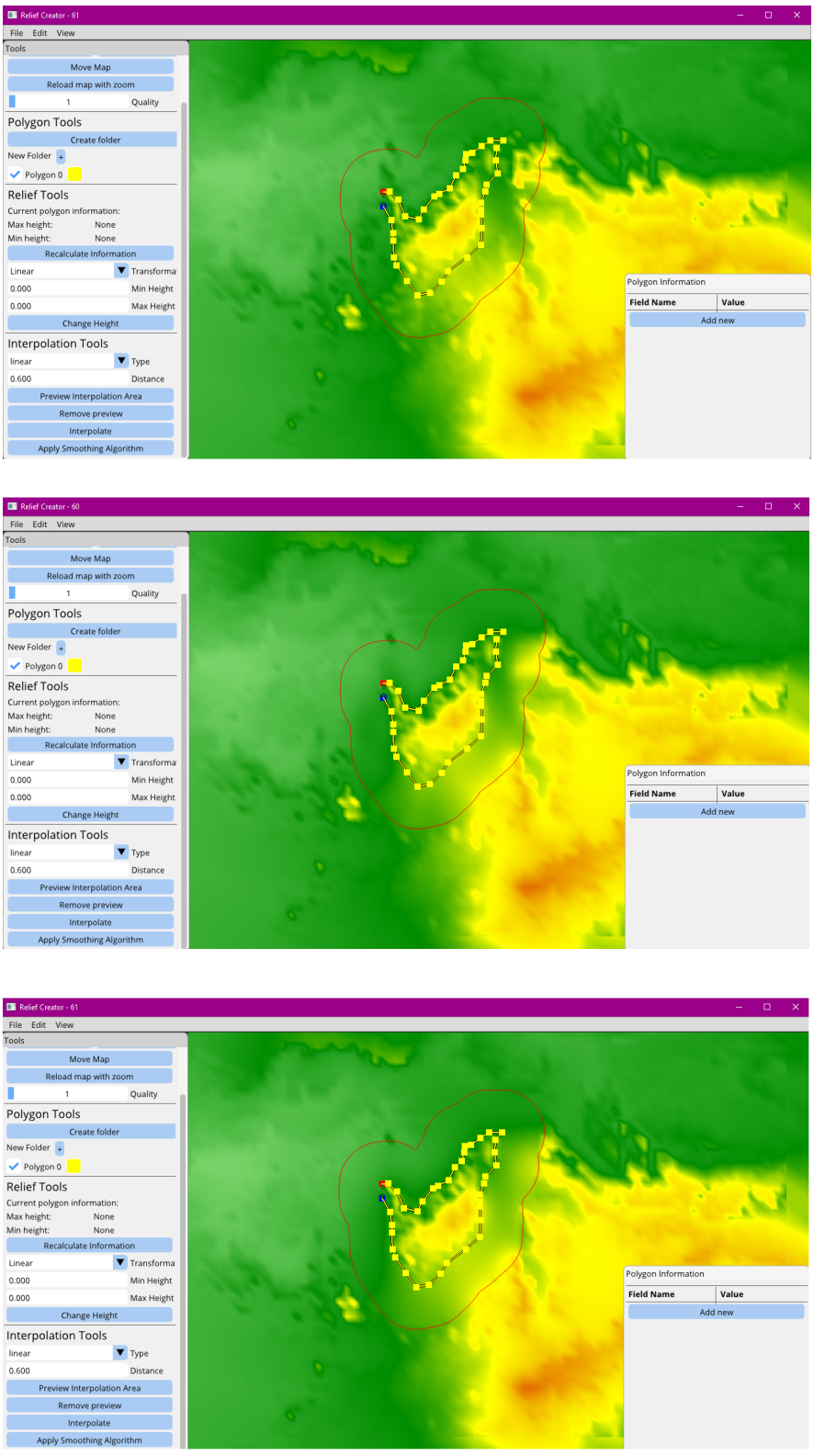


Figura 6.4: En la parte superior, imagen del mapa sin modificar con el polígono que se usara y el área de interpolación que se modificara, en medio, imagen del mapa después de aplicar el algoritmo de suavizado 1 vez, en la parte inferior, imagen del mapa después de aplicado el algoritmo de suavizado 5 veces.

una nueva carpeta en el programa en caso de que no exista una antes y colocará el nuevo polígono al interior de esta.

Luego, es posible añadir nuevos puntos al polígono recién creado haciendo clic directamente en el mapa cargado. En caso de necesitar borrar puntos añadidos al polígono, es posible hacerlo haciendo uso del menú de retroceso en el menú de edición o del atajo de teclado *CTRL+Z*.

Para crear nuevos polígonos al interior de una carpeta, solamente hay que presionar el botón que tiene un símbolo "+" que se ubica al lado derecho del nombre de las carpetas.

6.2.2.2. Carga de polígonos

Para cargar polígonos en la plataforma, hay que hacer uso del menú *File* que ofrece el programa en la barra de navegación (mostrado como número 7 en la figura A.1) y luego hacer clic en la opción *Load shapefile file...* o hacer uso del atajo de teclado *CTRL+L*. Esto abrirá una ventana interactiva (que varía dependiendo del sistema operativo en donde se ejecute el programa) que permitirá al usuario el seleccionar el archivo que desea cargar.

6.2.3. Modificación de la altura de los puntos en un polígono y uso de máscaras

Para realizar la modificación de la altura de los puntos que se encuentran al interior de un polígono, hay que hacer uso de las herramientas de relieve (mostradas como número 4 en la figura A.1).

Para utilizar estas herramientas es necesario tener cargado en el programa un mapa y un polígono con tres o más puntos. De otra forma las herramientas no se mostrarán o se desplegara un mensaje de error al intentar interpolar.

Para modificar las alturas de los puntos que se encuentran al interior de un polígono, hay que seleccionar el tipo de interpolación a efectuar (modificando el parámetro *Transformation*), seleccionar las nuevas alturas máximas y mínimas esperadas (modificando los parámetros *Max Height* y *Min Height*) y presionar el botón *Change Height*. Esto modificara todos los puntos al interior del polígono seleccionado.

Con la finalidad de entregar mayor información al usuario al momento de modificar las alturas de un polígono, en el menú de relieve, en la sección *Current polygon information* se encuentra un botón que permite al usuario el obtener el valor de las alturas máximas y mínimas de los puntos que se encuentran al interior del polígono seleccionado.

Para agregar filtros antes de realizar la modificación de los puntos de un polígono, hay que hacer clic en el botón *Add Filter*. Esto agrega un nuevo filtro en el programa, que será mostrado en la parte superior del botón de agregar filtro. Este filtro será aplicado antes de realizar la modificación de los polígonos. Los parámetros por seleccionar para los filtros varían

dependiendo del filtro que se agregue, teniendo que seleccionar un polígono si el filtro es de polígonos o ingresar un valor numérico si el filtro es de alturas. (ver imagen A.4).

Los filtros agregados se aplicarán automáticamente al momento de presionar el botón *Change height*.

6.2.4. Interpolación y suavizamientos de puntos

Para hacer uso de las herramientas de suavizado e interpolación, es necesario tener cargado el programa junto con tener un polígono de 3 o más vértices, en caso contrario, las herramientas no se mostrarán o se desplegará un mensaje de error en el programa.

Las herramientas de interpolación y suavizado se muestran en la sección de *Interpolation Tools* de las herramientas del programa (mostradas como número 5 en la figura A.1).

En esta sección el usuario puede seleccionar el tipo de interpolación junto con la distancia a usar para calcular el área de interpolación, un ejemplo de como se muestra el área de interpolación se muestra en la figura A.5.

Para generar la previsualización del área de interpolación de un polígono, hay que ingresar un valor mayor que 0 en la sección de *Distance* y hacer clic sobre el botón de *Preview Interpolation Area*, esto generará la previsualización en color rojo en el mapa. Para eliminar la previsualización, hay que hacer clic en el botón *Remove preview* teniendo seleccionado el polígono al cual se le desea eliminar la previsualización.

La distancia ingresada en el campo *Distance* es usada en el proceso de generación de las previsualizaciones y su unidad de medida corresponde a la unidad de medida usada en el mapa que se tenga cargado en el momento de realizar la previsualización.

Para realizar la interpolación, hay que hacer clic en el botón *Interpolate*, esto comenzará el proceso de interpolación de los puntos que se encuentran al exterior del polígono. De la misma forma, para aplicar algoritmos de suavizado hay que hacer clic sobre el botón *Apply Smoothing Algorithm*, comenzando esto con el proceso de suavizado.

6.2.5. Visualización en tres dimensiones

Para visualizar el programa en tres dimensiones, hay que seleccionar, a partir del menú de *View* de la barra de navegación (mostrada como número 7 en la figura 6.5) la opción *Change to 3D view*. Esto cambiará el menú del programa al que se muestra en la figura 6.6, el cual puede ser separado en tres secciones, información de la cámara, herramientas de vista y herramientas de unidades.

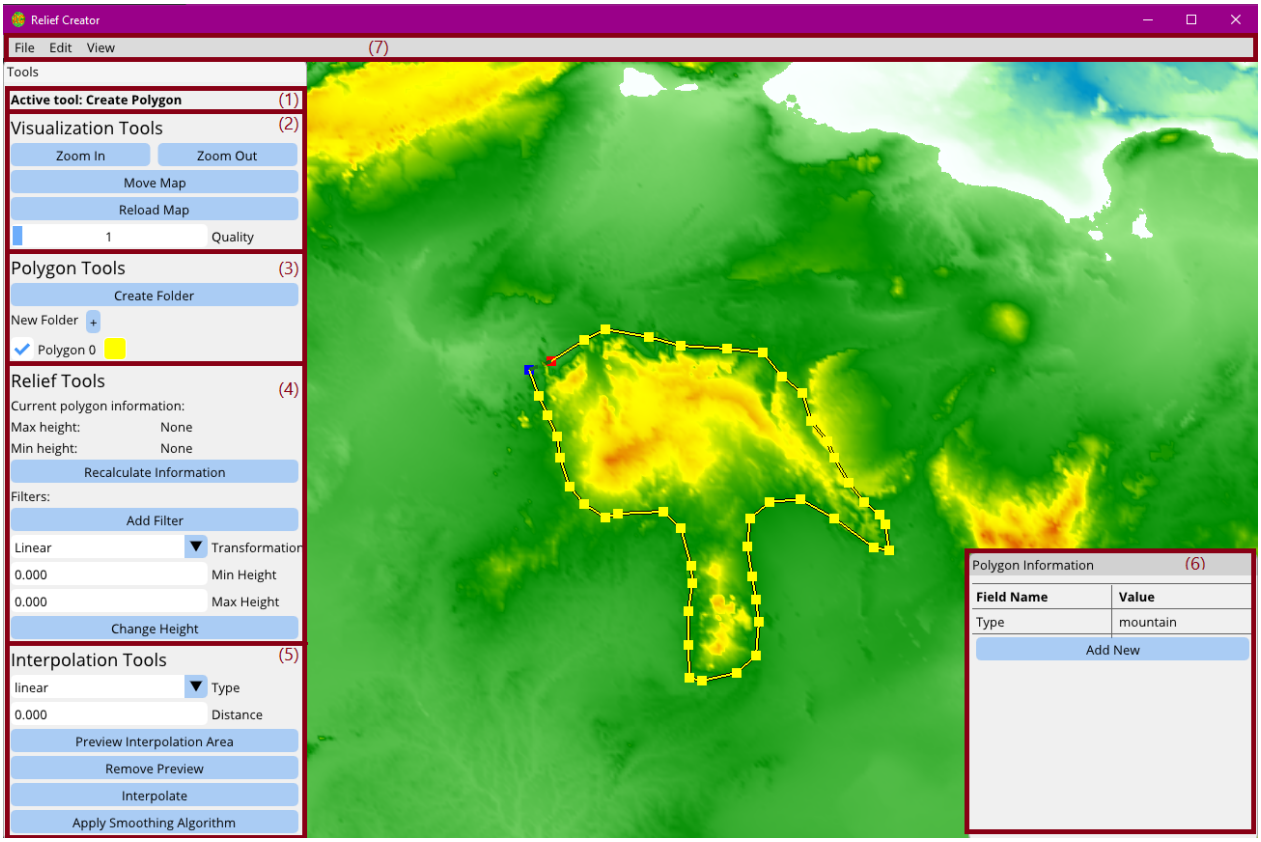


Figura 6.5: Imagen de la aplicación mostrando el menú de esta seccionado en 7 partes.

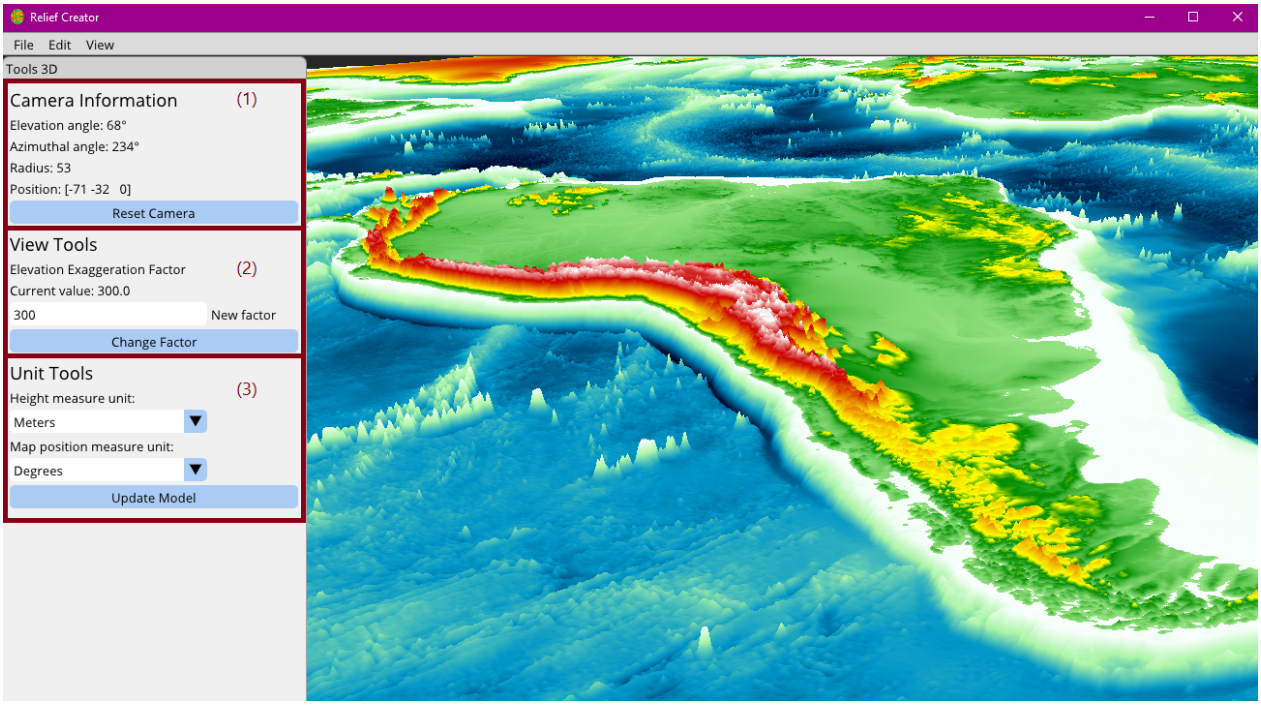


Figura 6.6: Imagen de la aplicación mostrando el menú en el modo en tres dimensiones.

La sección de información de la cámara muestra información relativa a la cámara que se está usando para visualizar el modelo. El radio de la cámara puede ser cambiado haciendo *Scroll* con el ratón sobre el mapa en tres dimensiones, acercándose o alejándose del modelo según corresponda. El ángulo de la cámara puede ser cambiado usando las teclas *WASD*, modificando esto el ángulo azimutal y el ángulo de elevación de la cámara, o dejando apretado el botón central del mouse y arrastrando este sobre el mapa. Tanto el radio como los ángulos de elevación y azimutal pueden ser restaurados a su estado inicial haciendo clic sobre el botón *Reset Camera*.

El factor de exageración usado en las alturas del modelo puede ser cambiado ingresando un valor en el campo *New factor* de la sección de herramientas de vista y luego presionando el botón *Change Factor*, modificando esto como se visualiza el modelo.

El valor ingresado por el usuario aplicado directamente sobre las alturas de todos los puntos del modelo cargado en la aplicación, esto es, si el usuario ingresa un factor de exageración de 300, entonces, las alturas del modelo serán multiplicadas por un valor de 300.

Las unidades usadas en el mapa tanto para las alturas como para la posición de los puntos pueden ser modificadas usando los menús desplegables de la sección de herramientas de unidades. Para que los cambios sean efectivos se debe hacer clic sobre el botón *Update Model*.

6.2.6. Exportación de mapas y polígonos

Para exportar un mapa, hay que presionar la opción *Export current model* en el menú *File* de la barra de navegación (mostrada como número 7 en la figura A.1). Esto abrirá un menú de selección de directorio en donde el usuario puede ingresar el nombre y el lugar en donde almacenar el archivo.

Para exportar un polígono, hay que hacer clic derecho, ya sea sobre una carpeta de polígonos si se quiere exportar un conjunto de polígonos, o sobre un polígono si solo se quiere exportar un único polígono. Luego, hay que presionar sobre la opción *Export to shapefile* o *Export polygons*, esto desplegará un menú de selección de directorio en donde el usuario puede ingresar el nombre y el lugar en donde almacenar el archivo.

6.3. Validación de la solución

Con la finalidad de validar que la aplicación desarrollada en esta memoria entrega resultados válidos, esto es, que entrega los resultados esperados por los geólogos y científicos al momento de modificar un mapa de paleoelevación, se decidió comparar los mapas generados por la aplicación desarrollada en esta memoria y los mapas generados utilizando el método desarrollado por Poblete y col. 2021.

El proceso de verificación que se uso es como se describe a continuación:

- Generación de un polígono que abarque una zona de alturas conocidas previamente.
- Modificación de un mapa de paleoelevación usando el polígono anterior con la aplicación desarrollada en esta memoria.
- Modificación del mismo mapa de paleoelevación y polígono usado en la modificación anterior, pero con la tecnología que existe actualmente.
- Comparación de los mapas obtenidos haciendo uso de la aplicación desarrollada en esta memoria con los mapas obtenidos usando la tecnología existente.

El polígono usado fue generado usando la herramienta Global Mapper, herramienta que permite la visualización de mapas de elevación y creación de polígonos sobre estos.

Para modificar las alturas del mapa de elevación utilizando la aplicación desarrollada, solo fue necesario cargar el mapa de elevaciones y el polígono en la aplicación, aplicar el cambio en los puntos del mapa haciendo uso de las herramientas que ofrece la aplicación y luego exportar el mapa resultante. Mientras que, para modificar las alturas del mapa utilizando las herramientas existentes, fue necesario generar un script con instrucciones que utilicen los comandos de GMT para poder modificar el mapa. Cabe destacar de este último proceso, que, dado que GMT son herramientas de consola, el mapa generado debe ser cargado en otra herramienta para poder visualizar si los cambios que se hicieron corresponden con los que se esperaban.

6.3.1. Datos y mapas comparados

El polígono usado para la modificación de los mapas corresponde a un polígono que toma gran parte de los puntos del continente de África. Este polígono fue generado calculando una curva de nivel de altura 500 metros en el territorio de África. El polígono puede apreciarse en la figura 6.8.

El mapa que se usó para la comparación se muestra en la figura 6.7. Este mapa fue modificado varias veces usando el polígono que se muestra en la figura 6.8.

6.3.2. Resultados obtenidos

En primera instancia, se modificaron los puntos que se encuentran al interior del polígono de la figura 6.8 hasta una altura entre 2000 y 6000 metros. Esto fue hecho utilizando la aplicación desarrollada como también utilizando herramientas ya existentes.

El proceso anterior genera dos mapas, uno generado usando la aplicación desarrollada, y otro generado usando la tecnología ya existente. Con la finalidad de obtener las diferencias entre cada mapa, se procedió, utilizando las herramientas ya existentes, a restar las alturas de los mapas generados, obteniéndose un mapa como el que se muestra en la figura 6.9.

El mapa obtenido del proceso anterior solo difiere en la altura de los puntos que se muestran en blanco. Los puntos en blanco corresponden a puntos cuya altura es distinta de cero,

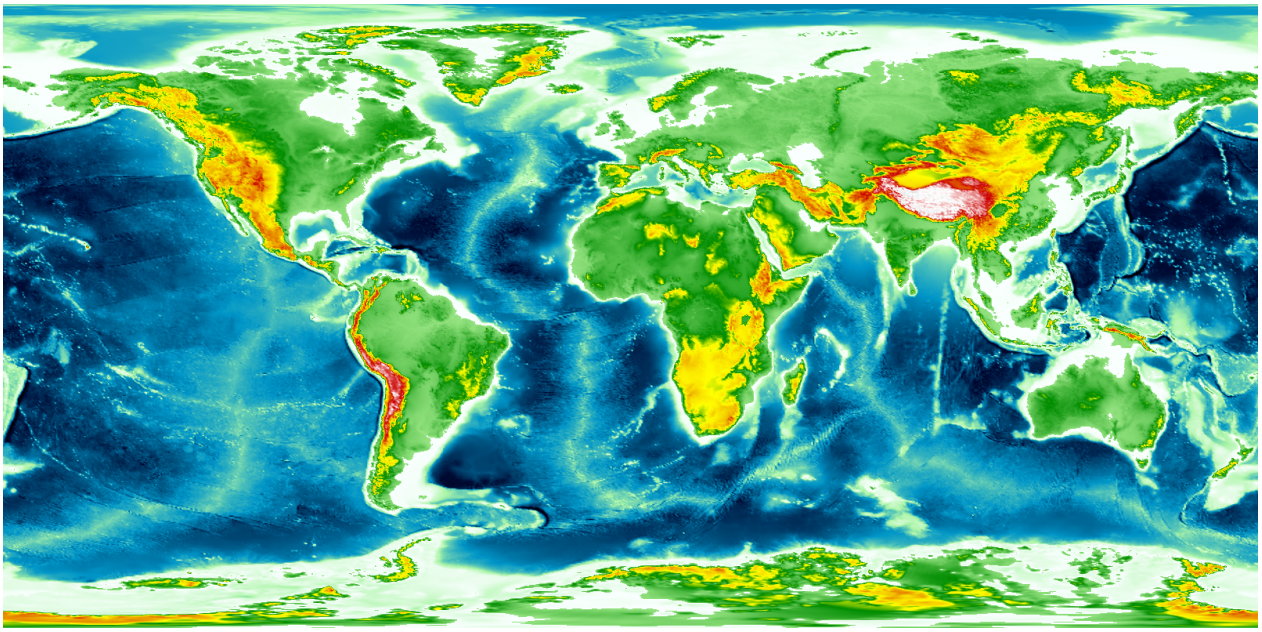


Figura 6.7: Mapa de elevaciones utilizado para la verificación del programa. Mapa contiene 1800 filas y 3600 columnas de información sobre las alturas de los puntos.

representando la altura de estos la diferencia o error obtenido entre un mapa generado haciendo uso de la aplicación desarrollada en esta memoria y uno generado con tecnología existente. Luego de procesados los puntos del mapa anterior para que todos tengan un valor positivo, se tiene que, de todos los puntos en blanco, la máxima diferencia de alturas entre un mapa y otro corresponde a 0,799 metros, mientras que el promedio de las diferencias obtenidas entre un mapa y otro corresponde a 0,644 metros.

Se realizó el mismo proceso anterior, con el mismo mapa y polígono, pero modificando las alturas entre 500 y 780 metros, obteniéndose un mapa como el que se muestra en la figura 6.10.

La máxima diferencia de alturas obtenidas del mapa generado del proceso anterior corresponde a 1,209 metros, mientras que el promedio de las diferencias corresponde a 0,286 metros.

Esta diferencia de alturas generadas puede deberse a la diferencia en los algoritmos que usan las tecnologías actuales para la modificación de los puntos y el algoritmo usado por la aplicación desarrollada en esta memoria. A pesar de esto, la diferencia de alturas obtenidas es bastante pequeña, entregando buenos resultados en general.

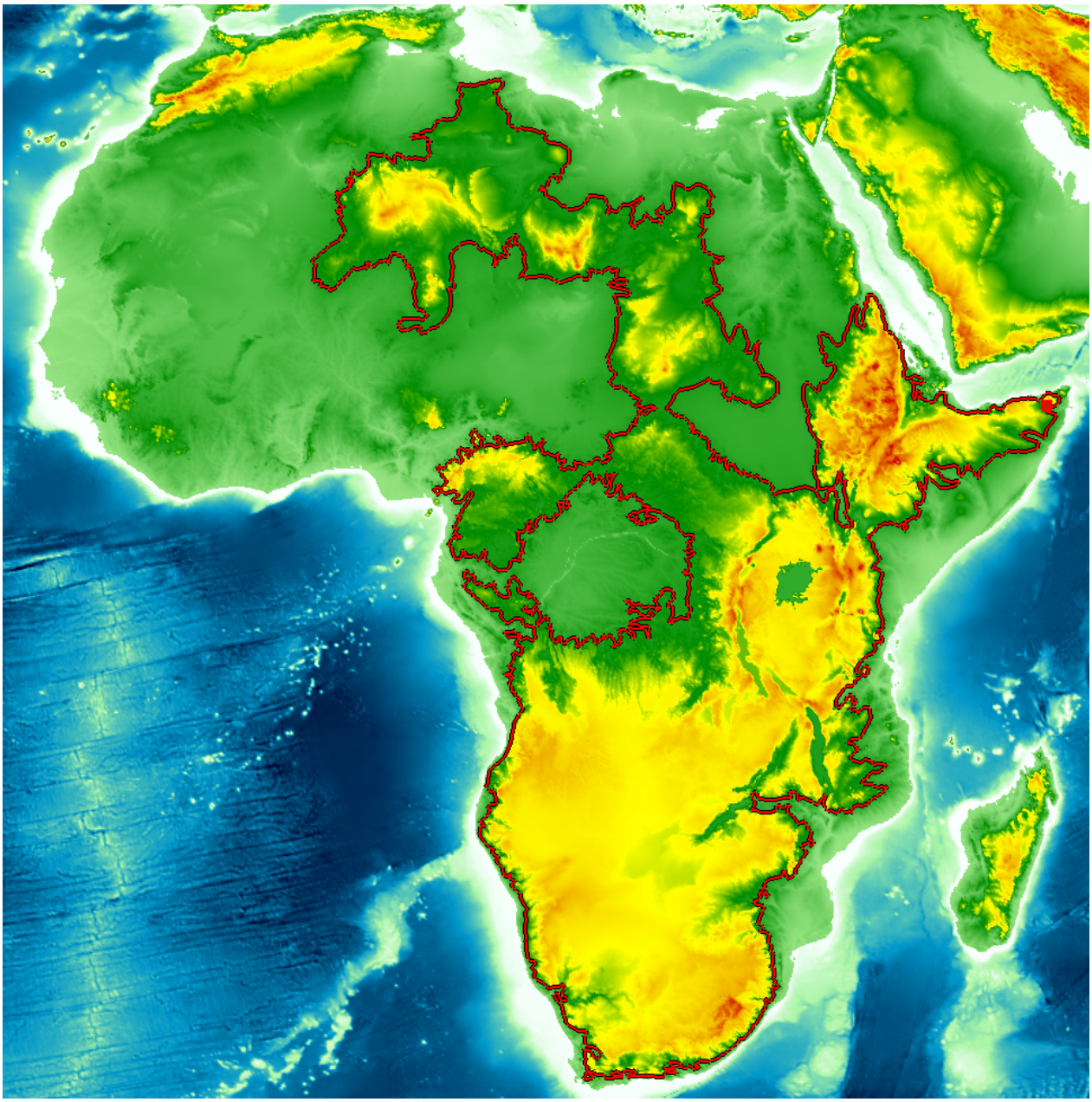


Figura 6.8: En rojo, polígono usado para el proceso de verificación de la aplicación implementada en la memoria. Polígono fue generado haciendo uso de una curva de nivel en 500 metros sobre África. Polígono conformado por 4543 vértices.

Figura 6.9: Mapa obtenido de la diferencia entre un mapa modificado usando los métodos definidos en Poblete y col. 2021 y un mapa modificado usando la aplicación desarrollada en esta memoria. La altura de los puntos fue modificada de su altura original hasta una altura entre 2000 y 6000 metros. En negro, puntos del mapa que poseen un valor 0 de altura. En blanco, puntos del mapa que poseen un valor distinto de cero. Todo el resto del mapa corresponde a puntos en color negro.

Figura 6.10: Mapa obtenido de la diferencia entre un mapa modificado usando los métodos definidos en Poblete y col. 2021 y un mapa modificado usando la aplicación desarrollada en esta memoria. La altura de los puntos fue modificada de su altura original hasta una altura entre 500 y 780 metros. En negro, puntos del mapa que poseen un valor 0 de altura. En blanco, puntos del mapa que poseen un valor distinto de cero. Todo el resto del mapa corresponde a puntos en color negro.

Capítulo 7

Conclusión

7.1. Aplicación desarrollada

Fue posible crear una aplicación que permite modificar mapas paleogeográficos de forma interactiva y fácil usando polígonos simples, junto con permitir exportar los resultados obtenidos a archivos Netcdf para su almacenamiento o posterior uso en otro software de sistemas de información geográfica.

La aplicación permite la importación y exportación de archivos en formato Netcdf, que almacenan la información de los mapas, y en formato Shapefile, que almacenan la información de los polígonos, de forma eficiente. Permitiendo a los usuarios de la aplicación usar los mismos archivos que utilizan la mayoría de las herramientas de información geográfica que existen en la actualidad.

La aplicación también permite la visualización de los mapas tanto en dos como en tres dimensiones, entregando mayor retroalimentación a los usuarios que hacen uso de la aplicación. Además, dadas las optimizaciones realizadas en la aplicación para el manejo de la información de los mapas y polígonos, esta permite trabajar tanto con mapas que almacenan una gran cantidad de datos como con polígonos que poseen un gran numero vértices, entregando buenos tiempos de ejecución.

En cuando a la modificación de los puntos, el software permite la modificación de los puntos que se encuentran al interior de los polígonos generados o importados, especificando las alturas máximas y mínimas esperadas por el usuario, como también la modificación de los puntos que se encuentran en el borde de los polígonos, ya sea aplicando algoritmos de interpolación, o aplicando algoritmos de suavizado. Además, la aplicación también permite la aplicación de filtros sobre los puntos a modificar, entregando mayor libertad a los usuarios al momento de seleccionar que puntos modificar.

La aplicación desarrollada simplifica el proceso actual de modificación de un mapa de elevaciones, proceso en el cual se hace uso de una o más aplicaciones, de una dificultad de uso generalmente elevada y no diseñadas para la modificación de la elevación de los puntos de un mapa, para la generación de polígonos y posterior modificación de los mapas, permitiendo a los usuarios el generar o importar polígonos y modificar mapas de elevación haciendo uso de una sola aplicación diseñada para tal proceso.

La aplicación fue testeada comparando mapas de alturas generados por la aplicación con mapas generados utilizando las herramientas que existen en la actualidad, utilizando en ambos casos el mismo mapa y polígonos iniciales, obteniendo buenos resultados.

El uso de librerías escritas en lenguaje C, y el uso de OpenGL, permiten hacer un uso efectivo de la GPU, permitiendo que el software desarrollado realice las operaciones de transformación e interpolación en tiempos competitivos con el software existente.

El uso de programación orientada a objetos, buenas prácticas de programación, testing y documentación permiten al software escalar en cuanto a funcionalidad y junto con facilitar su mantenimiento, permitiendo la fácil extensión de las funcionalidades que este posee en el futuro.

Considerando los puntos anteriores, todos los objetivos específicos se cumplen, logrando el objetivo general planteado en esta memoria.

Entre las dificultades encontradas en el desarrollo de la memoria, está el hecho de tener que desarrollar una aplicación cuyos tiempos de respuesta sean rápidos y comparables con las herramientas ya existentes. Dado que los mapas de elevación usados en geología usualmente contienen en su interior una gran cantidad de datos, el hacer que las operaciones de modificación, interpolación y suavizado se realicen en tiempo real es un desafío, incluso contando con una gran cantidad de bibliotecas como las que ofrece el lenguaje Python.

7.2. Trabajo futuro

Como trabajo futuro en la aplicación desarrollada, es posible extender las herramientas ya implementadas, agregando nuevos algoritmos de modificación de alturas, nuevos algoritmos de interpolación y suavizado, como también nuevos filtros. Esto entregaría mayor libertad a los usuarios de la aplicación para que estos puedan modificar los mapas de elevación como estos lo requieran.

Si bien tanto los algoritmos implementados como el sistema en general están optimizados, entregando tiempos de respuesta aceptables, aún es posible optimizar los algoritmos que se encuentran en la herramienta de interpolación de la aplicación.

Dado que la aplicación está desarrollada en el lenguaje de programación Python, y este posee una gran cantidad de bibliotecas junto con mecanismos que permiten la integración de funcionalidad escrita en otros lenguajes como Java o C++, la integración de nueva funcionalidad a la aplicación no debería ser un problema para desarrolladores que deseen mejorar la aplicación en un futuro.

La herramienta actual se encuentra diseñada con la finalidad de que nuevos desarrolladores puedan fácilmente cambiar los componentes de esta en caso de ser necesario. Esto trae consigo numerosos beneficios, entre estos, el que la aplicación sea fácilmente extendida por nuevos desarrolladores o que los módulos sean fácilmente reemplazables por otros completamente nuevos. Sin embargo, esto pone demasiada responsabilidad en el componente Engine,

componente encargado de conectar todos los otros componentes que conforman la aplicación.

Dado lo anterior, es recomendable que futuros desarrolladores evalúen la posibilidad de crear nuevas funcionalidades o módulos que se comuniquen con los otros componentes del programa, además del componente Engine, repartiendo de forma más equitativa las responsabilidades de los componentes que conforman la aplicación.

Bibliografía

Booch, G., Rumbaugh, J. O. & Jacobson, I. (1999). *The unified modeling languaje: the definitive reference to the UML from the original designers*. Addison-Wesley.

Shreiner, D. (2013). *OpenGL programming guide: the official guide to learning OpenGL, Versions 4.3*. Addison-Wesley.

Wessel, P., Smith, W. H., Scharroo, R., Luis, J. & Wobbe, F. (2013). Generic mapping tools: Improved version released. *Eos*, *94*(45). https://doi.org/10.1002/2013EO450001

Baatsen, M., van Hinsbergen, D. J. J., von der Heydt, A. S., Dijkstra, H. A., Sluijs, A., Abels, H. A. & Bijl, P. K. (2016). Reconstructing geographical boundary conditions for palaeoclimate modelling during the Cenozoic. *Climate of the Past*, *12*(8), 1635-1644. https://doi.org/10.5194/cp-12-1635-2016

O'Rourke, J. (2017). *Computational geometry in C* (2.ª ed.). Cambridge University Press.

Ruiz, D., Dupont-Nivet, G., Aminov, J., Poblete, F., Van Der Linden, T. & Van Hinsbergen, D. (2020). "Terra Antiqua": a paleogeographic reconstruction plugin for QGIS. https://doi.org/10.5194/egusphere-egu2020-20362

Poblete, F., Dupont-Nivet, G., Licht, A., van Hinsbergen, D. J., Roperch, P., Mihalynuk, M. G., Johnston, S. T., Guillocheau, F., Baby, G., Fluteau, F., Robin, C., van der Linden, T. J., Ruiz, D. & Baatsen, M. L. (2021). Towards interactive global paleogeographic maps, new reconstructions at 60, 40 and 20Ma. https://doi.org/10.1016/j.earscirev.2021.103508

de Vries, J. (s.f.). Welcome to OpenGL. https://learnopengl.com/

What is a digital elevation model (DEM)? (s.f.). Consultado el 24 de agosto de 2021, desde https://www.usgs.gov/faqs/what-a-digital-elevation-model-dem?qt-news_science_products=0#qt-news_science_products

Anexos

A continuación se presenta información sobre el programa complementaria a la presentada en la memoria.

A.1. Manual de uso

La interfaz gráfica de la aplicación puede dividirse en 7 secciones principales, las cuales pueden verse en la imagen A.1.

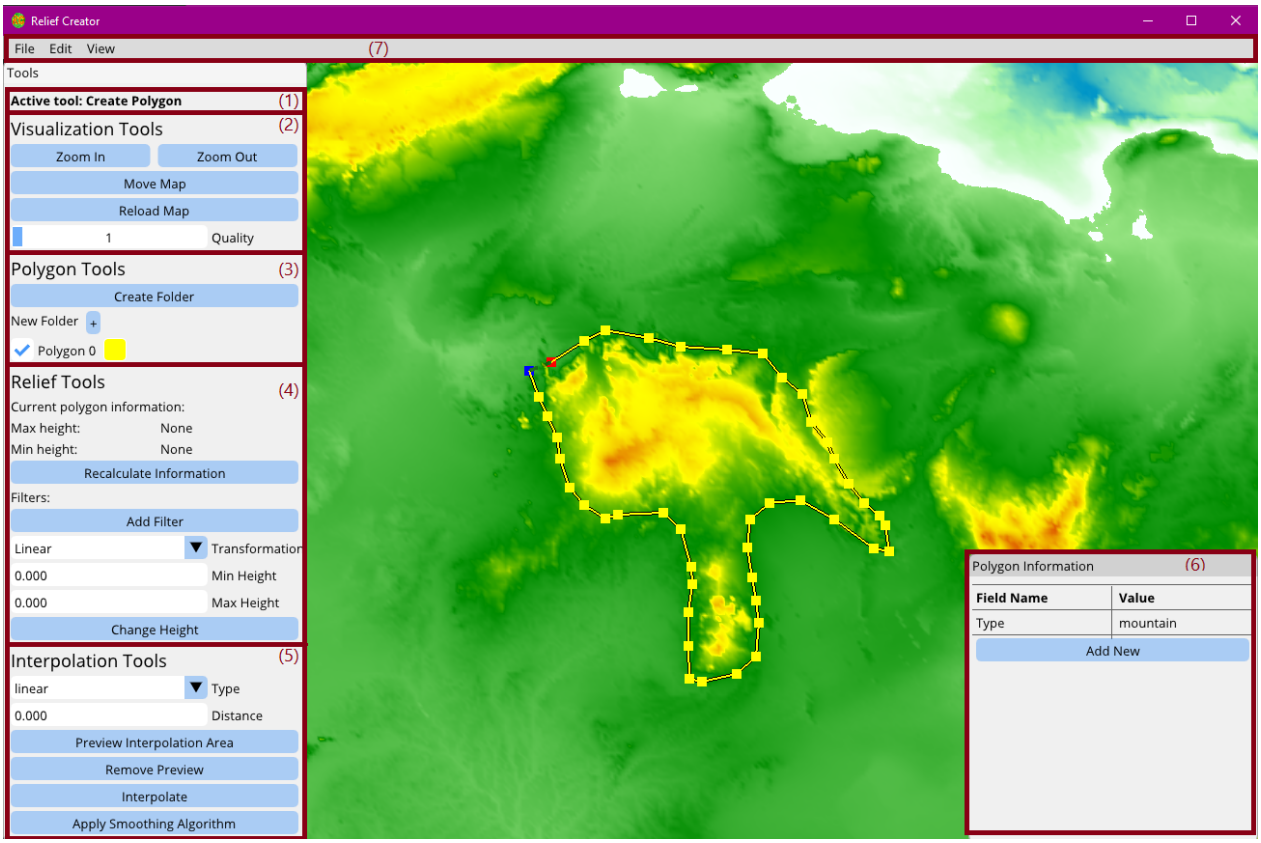


Figura A.1: Imagen de la aplicación mostrando el menú de esta seccionado en 7 partes.

1. Muestra la herramienta que se encuentra activa en la plataforma, su valor puede variar entre *None*, *Move Map* o *Create Polygon*.

2. Herramientas de visualización, herramientas que controlan el cómo se visualiza el mapa en la pantalla.

3. Herramientas de polígonos, herramientas que permiten crear y organizar los polígonos en el programa.

4. Herramientas de relieve, herramientas que permiten modificar el relieve de los puntos que se encuentran al interior de los polígonos. No se muestran a menos que se tenga un polígono en el mapa.

5. Herramientas de interpolación, herramientas que permiten interpolar o suavizar los puntos que se encuentran al exterior de los polígonos. No se muestran a menos que se tenga un polígono en el mapa.

6. Información del polígono, ventana que muestra los parámetros que se tienen configurados en el polígono, permite la creación y eliminación de estos parámetros.

7. Barra de navegación, barra con opciones útiles y que se muestra en todas las pantallas del programa.

A continuación se detallará de mejor maneja cada una de estas secciones.

A.1.1. *Visualization Tools*

A continuación, se listan los botones de la herramienta de visualización junto con una descripción de que es lo que estos hacen:

- *Zoom In*: Realiza un acercamiento sobre el mapa que se esté visualizando. Su accion tambien puede realizarse haciendo uso *Scroll Up* en el mouse.
- *Zoom Out*: Realiza un alejamiento sobre el mapa que se esté visualizando. Su acción también puede realizarse haciendo uso *Scroll Up* en el mouse.
- *Move Map*: Cambia la herramienta activa a la herramienta *Move Map*, esto hace que el arrastrar el mouse sobre el mapa dejando presionado el clic izquierdo mueva la posición de este. El mapa también se puede mover haciendo uso de los botones *WASD*.
- *Reload Map* y *Quality*: El botón *Reload Map* recarga el mapa que se tiene cargado haciendo uso del nivel de zoom que se tenga, mejorando la calidad de la imagen mostrada. *Quality* es un valor configurable, variable entre 1 y 30, que configura la calidad a la que se deben hacer las recargar de los mapas, siendo 1 la mejor calidad y 30 la peor. Los mapas también se pueden recargar presionando la tecla *R*.

A.1.2. *Polygon Tools*

Esta sección cuenta inicialmente solo con el botón *Create Folder*, botón que permite la creación de una nueva carpeta con un polígono en su interior.

Las carpetas son listadas directamente abajo del botón *Create Folder* con el nombre *New Folder*. A la derecha del nombre de las carpetas se encuentra un botón con un símbolo "+" que permite la creación de nuevos polígonos al interior de estas. Los polígonos que perteneces a una carpeta son mostrados directamente abajo del nombre de las carpetas.

Al hacer clic derecho sobre el nombre de una carpeta se muestran las opciones que se listan a continuación:

- *Export polygons*: Opción que permite exportar todos los polígonos que se encuentran al interior de una carpeta en formato Shapefile. Al hacer clic se abre una ventana de selección de directorio que permite al usuario escoger donde almacenar los archivos Shapefile generados.
- *Move Up/Down*: Opción que permite modificar el orden de las carpetas en la interfaz gráfica. Permitiendo modificar que polígonos se muestran sobre otros.
- *Rename*: Opción que permite modificar el nombre de la carpeta.
- *Delete folder*: Opción que permite eliminar la carpeta de la herramienta junto con todos los polígonos que esta almacena en su interior. Al presionarla se abre un menú de confirmación.

En la herramienta, solo puede haber un solo polígono activo en un determinado momento, esto es representado en el programa con un icono a la izquierda de los nombres de los polígonos, como se muestra en la figura A.2.

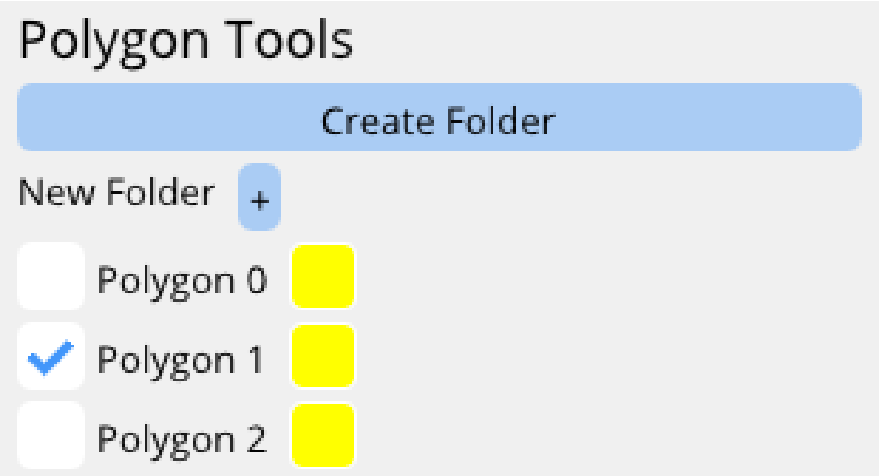


Figura A.2: Imagen de la aplicación mostrando el menú de polígonos y la diferencia entre los polígonos activos y los inactivos. Polígono con nombre *Polygon 1* es el polígono activo.

A la derecha del nombre de los polígonos se muestra un rectángulo del color de las líneas con las que se dibuja el polígono. Al hacer clic sobre este rectángulo se despliega un menú que permite al usuario modificar el color con el que se mostrarán los polígonos en el mapa. La imagen A.3 muestra el menú que se despliega al hacer clic sobre los rectángulos.

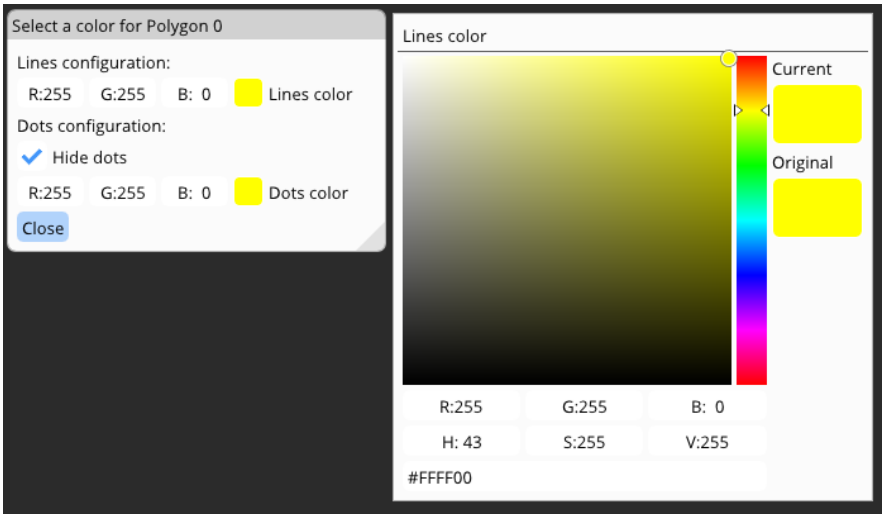


Figura A.3: Imagen de la aplicación mostrando el menú desplegado al hacer clic sobre el rectángulo de la derecha del nombre de los polígonos. A la izquierda, menú desplegable de selección de color, a la derecha, menú desplegado al hacer clic sobre los botones *Lines color* o *Dots color*.

Al hacer clic derecho sobre el nombre de los polígonos se muestran una serie de opciones que se listan a continuación:

- *Export to shapefile*: Exporta el polígono seleccionado a formato Shapefile. El presionar esta opción abre un menú de selección de directorio para almacenar el archivo Shapefile.
- *Move Up/Down*: Opción que permite modificar el orden de los polígonos al interior de una carpeta. Esto modifica el orden en el que se muestran los polígonos en los mapas.
- *Change folder*: Menú que permite cambiar el polígono de una carpeta a otra. Al pasar el ratón por sobre la opción se desplegará una lista con todas las carpetas disponibles a las cuales mover el polígono.
- *Rename*: Opción que permite cambiar el nombre del polígono.
- *Delete*: Opción que permite borrar el polígono de la herramienta. Al presionarla se abre una ventana de confirmación de la acción.

A.1.3. *Relief Tools*

La herramienta *Relief Tools* permite modificar la altura de los puntos que se encuentran al interior de los polígonos. Esta herramienta solo se encuentra disponible si es que se tiene algún polígono activo.

El botón *Recalculate information* permite calcular la altura máxima y mínima de los puntos que se encuentran al interior de un polígono, mostrando los resultados en las secciones *Max height* y *Min height* que se encuentran en la parte superior al botón.

El botón *Add Filter* añade un nuevo filtro a la herramienta. Los filtros son ejecutados antes de cualquier modificación de puntos y permiten seleccionar de mejor manera que puntos pueden ser modificados.

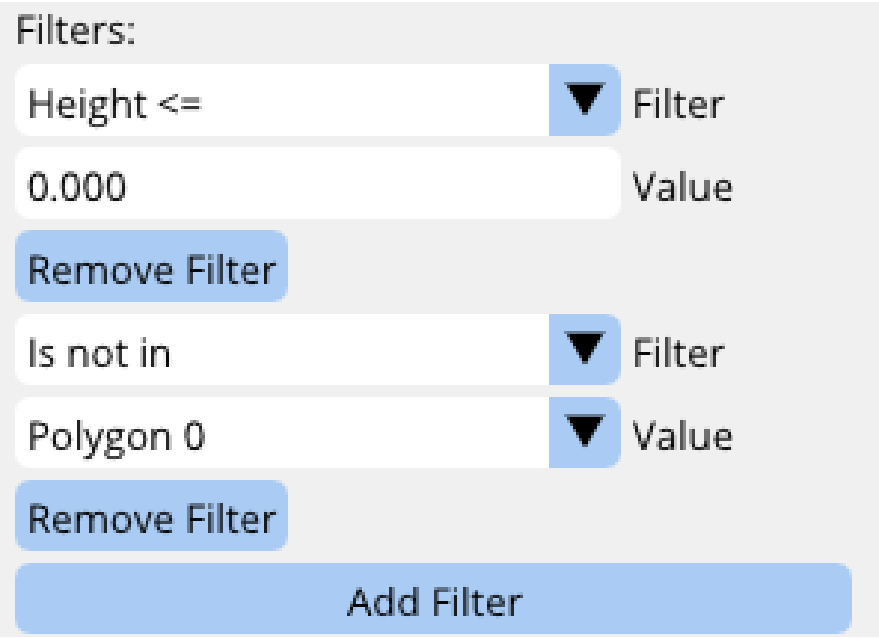


Figura A.4: Imagen de la herramienta mostrando el menú que se despliega al presionar el botón *Add Filter*.

En la figura A.4 se muestra el menú que se despliega al hacer clic en el botón *Add Filter*. Por cada filtro añadido a la herramienta se añaden dos parámetros configurables (*Filter* y *Value*) y un botón *Remove Filter*, los que se describen a continuación.

- *Filter*: Filtro a usar, actualmente el programa posee 4 implementados *Height* <=, *Height* >=, *Is in* y *Is not in*.
- *Value*: Valor a usar en el filtro añadido, el valor de este puede variar dependiendo del filtro, siendo una entrada numérica en caso de que se usen los filtros *Height* <= o *Height* >=, o un menú de selección de polígono en caso de que se usen los filtros *Is in* o *Is not in*.
- *Remove Filter*: Botón que permite borrar un filtro de la herramienta.

Finalmente, al final de la herramienta, se encuentra el menú de selección *Transformation* que permite al usuario seleccionar el tipo de transformación a realizar a los puntos (actualmente solo contiene la opción *Linear*), dos entradas numéricas *Min Height* y *Max Height* que permiten al usuario especificar las nuevas alturas mínimas y máximas esperadas después de la transformación respectivamente y el botón *Change Height*, que comienza el proceso de transformación de las alturas.

A.1.4. *Interpolation Tools*

La herramienta *Interpolation Tools* permite realizar modificaciones sobre los puntos que se encuentran al exterior de los polígonos con la finalidad de entregar una visualización más realista de los mapas. Esta herramienta solo se muestra si hay un polígono activo en la aplicación.

Esta posee dos parámetros configurables y 4 botones que se detallan a continuación:

- *Type*: Permite seleccionar el tipo de interpolación a efectuar, sus valores pueden variar entre *linar*, *nearest* y *cubic*.
- *Distance*: Permite seleccionar la distancia desde los polígonos que se debe usar para seleccionar que puntos modificar. La unidad de medida usada en esta distancia es la misma que se esté usando en el mapa que se tenga cargado en el momento.
- *Preview Interpolation Area*: Permite visualizar el área de interpolación que se usará para seleccionar los puntos a modificar, en la imagen A.5 se muestra un ejemplo de como se muestra el área.
- *Interpolate*: Interpola los puntos que se encuentran al interior del área de interpolación usando el tipo de interpolación seleccionado.
- *Apply Smoothing Algorithm*: Aplica un algoritmo de suavizado sobre los puntos que se encuentran al interior del área de interpolación. Este puede ser aplicado varias veces sobre los mismos puntos, resultando en un nivel de suavizado cada vez mayor.

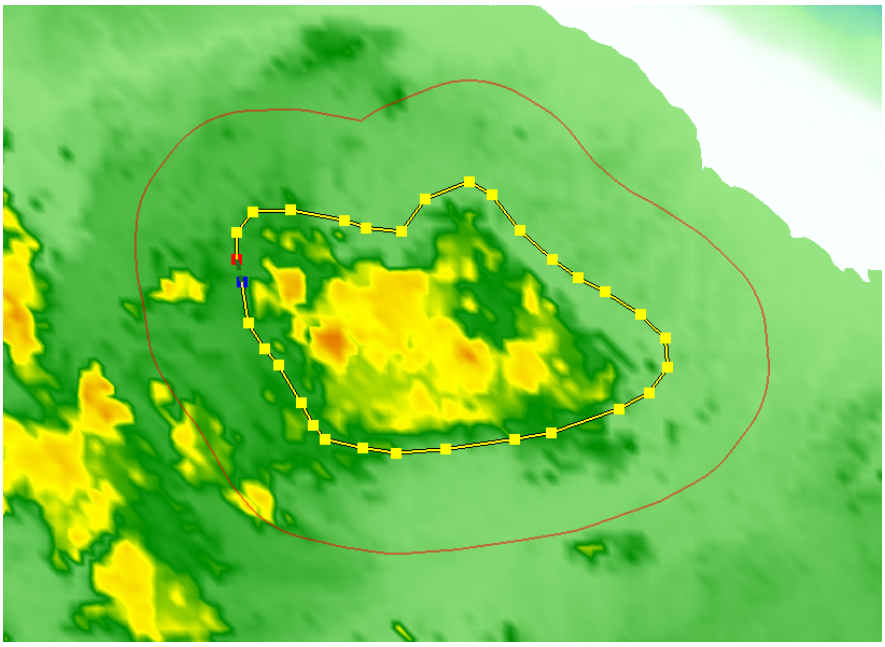


Figura A.5: Imagen de la aplicación mostrando el área de interpolación calculada para el polígono que se muestra de color amarillo.

A.1.5. *Polygon Information*

La herramienta *Polygon Information* muestra los parámetros que se almacenan en los polígonos y que serán exportados al interior de los archivos en formato Shapefile. Esta herramienta solo se muestra si hay un polígono activo en la aplicación.

La herramienta consiste en una tabla de dos columnas, en donde la primera columna muestra el nombre de los parámetros almacenados en el polígono y en la segunda columna se muestra el valor de estos parámetros.

El botón *Add New* abre un menú que permite añadir un nuevo parámetro al polígono (ver figura A.6). Este menú permite añadir tres tipos de parámetros a los polígonos, parámetros de tipo texto, parámetros de tipo numérico y parámetros de tipo booleano.

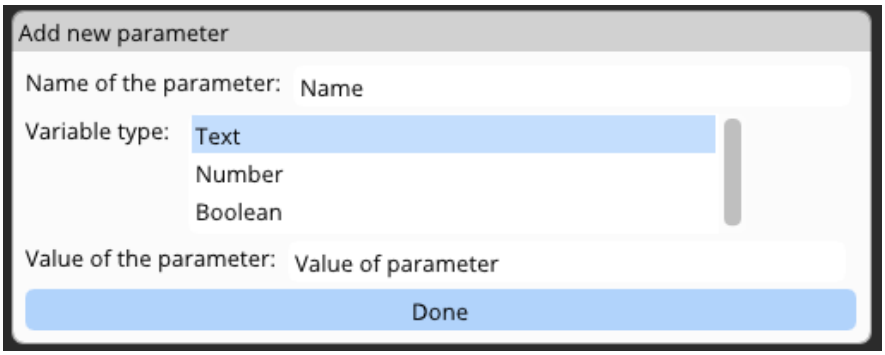


Figura A.6: Imagen de la aplicación mostrando el menú que se despliega al presionar el botón *Add Filter* de la herramienta *Polygon Information*.

Al hacer clic derecho sobre algún parámetro en la herramienta se muestran dos opciones que se listan a continuación:

- *Edit*: Opción que permite editar el tipo de variable y el valor que esta almacena.
- *Delete*: Opción que permite eliminar un parámetro del polígono.

A.1.6. Barra de navegación

La barra de navegación es un menú que se encuentra disponible en todas las pantallas de la herramienta que permite realizar diversas acciones. Este puede dividirse en tres secciones principales *File*, *Edit* y *View*.

A continuación se detallan las opciones que se presentan en la sección *File* de la barra de navegación.

- *Open NetCDF file...*: Opción que permite cargar un nuevo mapa a la plataforma. El presionarla abre un menú de selección de archivo que permite al usuario seleccionar el mapa a cargar. Funcionalidad también puede ser ejecutada con el atajo de teclado *CTRL+O*.

- *Change CTP file...*: Opción que permite cargar un nuevo archivo de colores a la plataforma. El presionarla abre un menú de selección de archivo que permite al usuario seleccionar el archivo a cargar. Funcionalidad también puede ser ejecutada con el atajo de teclado *CTRL+T*.

- *Load shapefile file...*: Opción que permite cargar uno o varios polígonos a la plataforma. El presionarla abre un menú de selección de archivo que permite al usuario seleccionar el archivo Shapefile a cargar. Funcionalidad también puede ser ejecutada con el atajo de teclado *CTRL+L*.

- *Export current model...*: Opción que permite exportar el mapa cargado a un archivo Netcdf. El presionarla abre un menú de selección de directorio para almacenar el nuevo archivo generado.

A continuación se detallan las opciones que se encuentran en la sección *Edit*:

- *Undo*: Opción que permite deshacer la última acción realizada en la plataforma por la herramienta activa. Funcionalidad también puede ser ejecutada con el atajo de teclado *CTRL+Z*.

A continuación se detallan las opciones que se encuentran en la sección *View*:

- *Unfix windows positions*: Permite que las ventanas mostradas en la aplicación sean móviles y de tamaño configurable. El presionar esta opción hace que esta cambie a la opción *Fix windows positions*, opción que permite volver a dejar las ventanas en su posición y configuración inicial.

- *Use points*: Cambia el modo de visualización de los mapas para ver solamente los puntos que componen los mapas.

- *Use wireframe*: Cambia el modo de visualización de los mapas para ver solamente las líneas que componen los triángulos generados en el modelo en dos dimensiones.

- *Fill polygons*: Cambia el modo de visualización de los mapas al modo por defecto, colorando los modelos completamente.

- *Change to 3D view*: Cambia el modo de visualización de los mapas al modo 3D. El presionar esta opción hace que la opción cambie a *Change to 2D view*, opción que regresa al modo de visualización por defecto.

A.1.7. Modo 3D

Desde la barra de navegación es posible cambiar el programa a un modo en tres dimensiones. Este modo permite a los usuarios el visualizar de mejor manera los mapas. El modo de navegación posee herramientas que se pueden dividir en tres secciones (ver figura A.7), *Camera Information*, *View Tools* y *Unit Tools*. A continuación, se detallan las tres secciones.

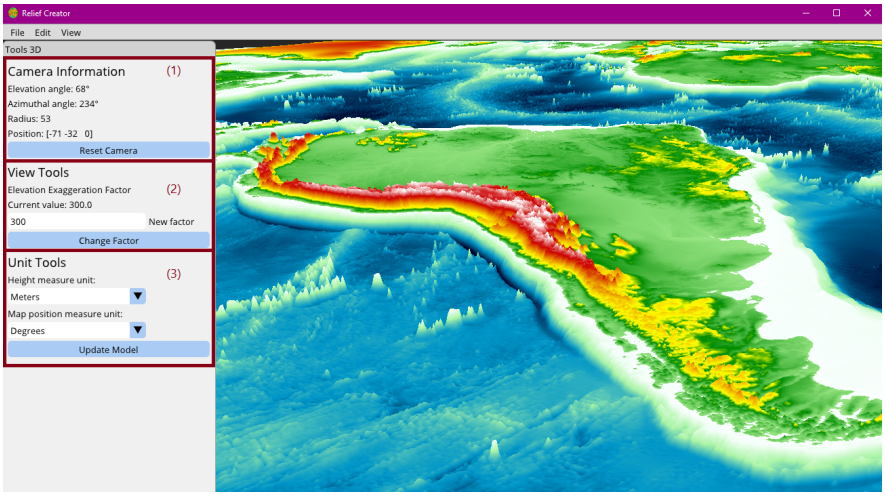


Figura A.7: Imagen de la aplicación mostrando el menú en el modo en tres dimensiones.

A.1.7.1. *Camera Information*

Esta sección muestra información relacionada con la cámara que se está usando para visualizar el modelo en tres dimensiones.

Los controles de la cámara son como siguen:

- *WASD*: Modifican los ángulos de la cámara, botones *WS* modifican el ángulo de elevación, mientras que *AD* modifican el ángulo azimutal usado.
- *Scroll up/down*: Modifica el valor del radio usado por la cámara al foco que de esta.
- *Arrows*: Modifican el foco al cual está apuntando la cámara. Por defecto el foco se encuentra en el centro del modelo en tres dimensiones.
- *Mouse middle button*: Permite modificar ambos ángulos de la cámara al mismo tiempo. Para hacer esto hay que dejar presionado el botón de en medio del ratón y arrastrar este para modificar los ángulos de visión.

Los cambios hechos en la cámara son mostrados en la herramienta, mostrándose los ángulos de elevación y azimutal, el radio de la cámara al foco y la posición del foco de la cámara respectivamente.

El botón *Reset Camera* regresa los parámetros de la cámara a su valores iniciales.

A.1.7.2. *View Tools*

La herramienta *View Tools* permite modificar el cómo se visualiza el modelo en tres dimensiones. Esta posee un parámetro configurable *New Factor* y un botón *Change Factor* que permiten modificar el factor de exageración usado en el modelo en tres dimensiones.

El factor de exageración consiste en un parámetro que se aplica al modelo en tres dimensiones que intensifica el relieve de los mapas, generando un modelo con montañas mucho más altas de lo que deberían.

El parámetro *New factor* es una entrada numérica que permite al usuario ingresar el nuevo factor de exageración que se quiere usar en el modelo, mientras que el botón *Change Factor* aplica este factor en el modelo, modificando el cómo se visualiza el modelo en tres dimensiones.

A.1.7.3. *Unit Tools*

La herramienta *Unit Tools* permite modificar las unidades usadas en el modelo en tres dimensiones. Esta herramienta presenta tres opciones que se detallan a continuación:

- *Height measure unit*: Opción que permite modificar la unidad de medida que se usa para representar las alturas del mapa. Su valor por defecto es metros.
- *Map position measure unit*: Opción que permite modificar la unidad de medida que se usa para indicar la posición de los puntos en el mapa. Su valor por defecto es grados.
- *Update Model*: Opción que permite aplicar los cambios con las nuevas unidades de medida indicadas en las opciones anteriores.

A.2. Tecnologías usadas

En la herramienta desarrollada se hizo uso de diferentes tecnologías y herramientas para poder implementar toda la funcionalidad deseada, a continuación, se explica cada una de las tecnologías usadas en el desarrollo de esta.

A.2.1. Python

Python es un lenguaje de programación de alto nivel, no tipado, dinámico y multiparadigma, que permite a sus usuarios el desarrollo de diferentes tipos de aplicaciones, ya sean estas aplicaciones de consola o aplicaciones interactivas.

Python, desarrollado por Guido van Rossum y liberado por primera vez en su versión 0.9.0 el año 1991, es un lenguaje que se caracteriza por permitir a los desarrolladores escribir programas claros y de fácil lectura (lectura de código) tanto para proyectos grandes como pequeños.

Además de lo anterior, Python es conocido por su gran número de bibliotecas que este posee implementadas, siendo las más conocidas las relacionadas con el ámbito de inteligencia artificial y manejo de operaciones matemáticas.

Python es un lenguaje altamente usado en el mundo científico, siendo usado no solo por programadores, sino que también por científicos de diferentes áreas de la ingeniería.

La aplicación desarrollada usa de base el lenguaje de programación Python, haciendo uso de las diferentes bibliotecas que este ofrece para poder implementar las diferentes funcionalidades del programa.

El uso de este lenguaje para la aplicación, dado el alto uso que este tiene en el mundo científico, permite que no solo programadores o desarrolladores se involucren en el desarrollo de esta, sino que abre la posibilidad de que geólogos y científicos también se involucren en el proceso de desarrollo.

A.2.2. OpenGL

OpenGL, abreviación para Open Graphic Library, es una biblioteca de procesamiento gráfico, inicialmente programada en lenguaje C, que ofrece una API (de sus siglas en inglés Application Programming Interface) que facilita el desarrollo de aplicaciones gráficas interactivas. Entre las principales funciones ofrecidas por OpenGL se encuentra el dibujo en dos y tres dimensiones de polígonos, líneas y puntos.

OpenGL es desarrollada y mantenida actualmente por Khronos Group, siendo su lanzamiento inicial en el año 1992, y que ha recibido bastantes actualizaciones en el tiempo, encontrándose actualmente en su versión 4.6.

A partir de la versión 3.3, OpenGL permite la definición de vertex shaders y fragment shaders, estas son piezas de código que son compiladas en tiempo de ejecución y que son ejecutadas por la GPU (de las siglas Graphic Processor Unit) de forma paralela, generando un hilo de ejecución por cada vértice especificado en el programa para el vertex shader, y un hilo de ejecución por cada píxel a dibujar en la pantalla para el fragment shader.

La especificación de estos programas que se ejecutan en la GPU no solo alivia la carga que recibe el procesador al momento de dibujar los píxeles en la pantalla, sino que también, dado como están diseñadas las GPU, mejoran en gran medida la eficiencia de los programas implementados en OpenGL. La información sobre cómo funcionan los shaders en OpenGL se describe en la sección 5.3.

OpenGL es una biblioteca altamente usada en la actualidad para el desarrollo de aplicaciones gráficas, esto debido la eficiencia que esta presenta al momento de realizar las operaciones de dibujo, principalmente debido al uso de la GPU para ayudarse en las tareas de dibujo.

PyOpenGL es una biblioteca escrita en Python que ofrece las funcionalidades de OpenGL en Python. Esta biblioteca ejecuta las funciones originalmente programadas en lenguaje C, manteniendo la eficiencia que estas tienen al ser programadas en un lenguaje compilado.

La aplicación desarrollada en esta memoria utiliza PyOpenGL para efectuar el procesamiento gráfico de los mapas y polígonos.

A.2.3. GLFW

GLFW (abreviación de Graphic Library Framework) es una biblioteca complementaria a OpenGL, desarrollada por "The GLFW Development Team", multiplataforma, que ofrece una API que facilita la creación de ventanas, contextos y superficies de aplicaciones gráficas que utilicen OpenGL.

La biblioteca está escrita en lenguaje C, y, además de ofrecer funcionalidad para el manejo de ventanas y superficies, también ofrece funcionalidad para el manejo de eventos en las aplicaciones, entregando una API para programar funcionalidad relacionada con los eventos que sucedan en la aplicación.

La biblioteca es de código abierto bajo la licencia zlib/libpng, siendo su uso gratuito para fines gratuitos y comerciales.

El uso de esta herramienta permite crear ventanas y manejar eventos en distintos sistemas operativos bajo una misma API, haciendo que las funcionalidades desarrolladas en la aplicación de esta memoria puedan ser usadas en distintos sistemas operativos sin ninguna diferencia en la implementación.

PyGLFW es una biblioteca escrita en Python que implementa la funcionalidad de GLFW en Python, permitiendo el manejo de ventanas y contextos gráficos. Como la biblioteca original está programada en lenguaje C, se mantiene la eficiencia que esta tiene al ser programada en un lenguaje compilado en Python.

A.2.4. Dear IMGUI

IMGUI (abreviación de Immediate Mode Graphical User Interface) es un patrón de diseño de interfaces de usuario que, a diferencia del modo que usualmente se usa en aplicaciones de escritorio llamado Retained Mode, ejecuta la lógica de la interfaz gráfica al mismo tiempo que el proceso de renderizado, entregando más libertad a los desarrolladores para configurar el comportamiento de la interfaz gráfica.

Las interfaces gráficas del estilo inmediato son ideales para aplicaciones gráficas, ya que permiten analizar en cada momento los eventos que han sucedido en la aplicación, haciendo más sencillo el entregarle retroalimentación al usuario sobre sus acciones.

Dear IMGUI es una implementación de una interfaz gráfica de modo inmediato, programada en C++, que se renderiza en conjunto con OpenGL, por lo que aprovecha el uso de la GPU para hacer más eficiente el proceso de renderizado de la interfaz gráfica, solo usando la CPU para realizar la lógica asociada a la aplicación.

Dear IMGUI, y las interfaces gráficas de modo inmediato en general, dado que definen en cada fotograma que es lo que se renderizará en pantalla, permiten un fácil escalamiento de su funcionalidad, siendo sencillo el modificar o agregar nuevas ventanas y botones a estas.

La aplicación implementada en esta memoria utiliza la biblioteca PyIMGUI. Esta biblioteca está implementada en lenguaje Python y ofrece las funcionalidades de la biblioteca Dear IMGUI en Python, y, dado que esta última se encuentra originalmente programada en C++, aprovecha todos los beneficios en rendimiento que tiene asociado el uso de funcionalidad escrita en un lenguaje compilado.

A.2.5. PlantUML

PlantUML es una herramienta de código abierto, desarrollada por Arnaud Roques, lanzada por primera vez el año 2009, que permite la generación de diagramas que siguen el estándar UML a partir de archivos de texto plano.

Esta herramienta permite la generación de diagramas de clases, procesos, objetos, entre otros, en formato de código, haciendo mucho más fácil la manutención y modificación de estos diagramas.

La herramienta también ofrece diferentes comandos que permiten importar código de unos archivos a otros, permitiendo cierto grado de modularidad en la generación de diagramas.

En la aplicación desarrollada en esta memoria, parte de la documentación está hecha utilizando PlantUML. Esto ya que PlantUML facilita el mantenimiento de la documentación, haciendo más fácil el mantenerla actualizada con los últimos cambios de la plataforma.

A.2.6. Shapely

Shapely es una biblioteca de Python, desarrollada bajo la licencia BSD, que permite el fácil manejo de información de tipo vectorial, en particular, facilita el manejo de polígonos, ofreciendo una gama de operaciones para realizar sobre estos.

La licencia BSD, acrónimo de Berkeley Software Distribution, usada por Shapely, permite el uso de la biblioteca y redistribución de esta tanto en contextos comerciales como no comerciales.

Shapely está programada en lenguaje Python, pero esta basada en la biblioteca GEOS (Geometry Engine Open-Source), la cual está programada en C++, manteniendo de esta forma la eficiencia de los lenguajes compilados en la implementación de sus funciones.

El hecho de que las bibliotecas OpenGL, GLFW, Dear IMGUI y Shapely estén programadas en C o C++ representa un gran beneficio en el rendimiento del programa, ya que, el programa, al ser una aplicación interactiva, necesita dibujar el mapa en cada frame de la aplicación, acto que sucede 60 veces por segundo, siendo importante el uso de bibliotecas de dibujo optimizadas.

A.3. Open Source

Open-Source, en el contexto de desarrollo de *software*, se refiere principalmente al desarrollo de código completamente disponible de forma gratuita para su posible modificación o redistribución. Esto no está limitado al código fuente solamente, pudiendo agregar a lo anterior documentos de diseño, documentación o contenidos derivados del producto mismo.

Para lograr esto, se hace uso de diferentes licencias que permiten o limitan la privatización y uso del *software* o producto, encontrándose entre las licencias de código abierto más conocidas la licencia MIT, inicialmente creada por el instituto de tecnologías de Massachusetts, y la licencia GPTv2.

Aunque open-source normalmente se refiere al desarrollo de software sin cargos para su uso o modificación, incluyendo fines comerciales, este no tiene por qué ser el caso para todos los programas, pudiendo añadir los desarrolladores o distribuidores sus propias restricciones a las licencias que estos implementan.

Entre los principales beneficios del desarrollo de software open-source se encuentran el hecho de que el software es gratuitamente distribuido a los usuarios, no limitando el acceso a ningún usuario, ayudando esto en la detección de errores y testing de las aplicaciones. Otro beneficio del desarrollo open-source es el hecho de que el código fuente se encuentra disponible de forma gratuita, descentralizando el desarrollo del software, acelerando el proceso de desarrollo y corrección de errores de la aplicación.

El software desarrollado en esta memoria está licenciado bajo la licencia GPLv3. Esto quiere decir que el uso de la aplicación como también el código fuente son distribuidos de forma gratuita. Esto permite que la aplicación pueda seguir siendo desarrollada en el futuro con nuevas funcionalidades o corregida por nuevos desarrolladores.

Con la finalidad de facilitar el desarrollo de software para desarrolladores externos, es importante que el software desarrollado presente una buena documentación que explique fácilmente cómo funciona el software desarrollado y los procesos para ejecutarlo y modificarlo. En la sección A.5 se detalla cómo se efectuó la documentación de la aplicación desarrollada en esta memoria con la finalidad de facilitar el uso y extensión del programa.

A.4. Formatos

En el proceso de modificación de la elevación de los mapas paleogeográficos, se usan varios formatos de archivos para almacenar la información de estos o de los polígonos que se usan para especificar diferentes áreas de los mapas. A continuación, se detallan los formatos más conocidos y usados para almacenar este tipo de información.

A.4.1. Netcdf

Los mapas de paleoelevación almacenan información sobre la geografía de la Tierra en un periodo específico de tiempo. Existen diferentes formas de almacenar esta información de forma digital, ya sea en forma de arreglos, tablas, en bases de datos geoespaciales, o en alguna otra representación. Un formato bastante usado por la comunidad de geólogos y científicos, y aceptada en la mayoría de software GIS, para almacenar datos sobre la paleoelevación de un terreno, corresponde al formato NetCDF.

NetCDF, acrónimo para *Network Common Data Form*, es un formato de archivo de especificación abierta, aceptado por la OGC (Open Geosppatial Consortium), organización internacional que se compromete con la creación de estándares abiertos para la comunidad geoespacial en el marco de los sistemas de información geográfica (GIS), bastante usado por la comunidad geocientífica para almacenar la información de los mapas de paleoelevación.

Con la finalidad de ser independiente de la máquina en donde se lea el archivo, este contiene en su interior metainformación en formato de texto con especificaciones de las características que tienen los datos que el mismo archivo almacena, logrando de esta manera, ser un archivo autocontenido e independiente de la máquina en donde se lea.

El formato Netcdf permite el almacenamiento de información en forma de arreglos, siendo ideal para almacenar información de mapas de elevación geográfica.

A.4.2. Shapefile

El generar polígonos con la finalidad de marcar ciertas áreas en el terreno para su procesamiento, ya sea dibujándolos o generándolos mediante operaciones sobre el mismo terreno, es una práctica común en el estudio de la paleogeografía de la Tierra. Un formato de almacenamiento de polígonos bastante usado por la comunidad científica, y por un gran número de software GIS, corresponde al formato Shapefile.

Shapefile es un formato de archivo de especificación abierta, desarrollado y regulado por ESRI (de sus siglas en ingles *Environmental Systems Research Institute*), que permite el almacenamiento de datos vectoriales geoespaciales, esto es, puntos, líneas y polígonos, junto con atributos que permiten describir características de estos.

ESRI, es una empresa internacional privada de software que proporciona servicios GIS, web GIS y de bases de datos geoespaciales, inicialmente fundada en el año 1969 como el Instituto de investigación de sistemas medioambientales (Environmental Systems Research Institute) con la finalidad de actuar como consultora para proyectos que involucren la modificación y el manejo de ambientes naturales.

Este formato de archivo fue diseñado con la finalidad de interconectar diferentes soluciones GIS (Geographical Information Systems), siendo un formato aceptado por la mayoría de *software* GIS y bastante usado por la comunidad de geólogos y científicos para el almacenamiento de polígonos y otras formas de datos vectoriales.

Este tipo de formato permite almacenar en su interior parámetros asociados a cada figura almacenada en su interior, permitiendo el almacenamiento de datos numéricos, booleanos, y de cadenas de texto. Estos parámetros son definidos en la mayoría de los casos por los usuarios para agregar información complementaria a los polígonos.

Cada figura almacenada al interior de un archivo Shapefile contiene asociado a ella un tipo, este tipo determina la información que la figura tiene almacenada en su interior y como esta se encuentra almacenada.

A.5. Documentación

La documentación es de vital importancia en el proceso de extensión de un programa. Esta especifica el comportamiento que deberían tener los componentes, la forma en la que estos se comunican junto con los protocolos que estos usan. Además, esta es de vital importancia en el proceso de testing de las aplicaciones, pues, especifica el comportamiento esperado del programa, creando una barrera entre que es lo que el programa debería y no debería hacer. La existencia de documentación facilita la modificación del software en el futuro, facilitando el proceso a nuevos desarrolladores que deseen extender con nueva funcionalidad algún programa.

La documentación usada en el programa implementado en esta memoria puede dividirse en dos secciones, documentación de código y documentación de la arquitectura del programa.

A.5.1. Documentación de la arquitectura

La documentación de la arquitectura del programa, esto es, documentación de los componentes principales que usa el programa, las dependencias de estos y su comunicación, ayuda a los desarrolladores a entender el funcionamiento, de forma general, de la lógica del programa. Este tipo de documentación ayuda a desarrolladores externos a entender la lógica detrás de como están ordenados los componentes, no teniendo estos que entender la totalidad del código fuente para poder realizar modificaciones o mejoras en componentes específicos.

Si bien la documentación de la arquitectura de los programas puede ser escrita, es común el uso de diagramas que expliquen de forma gráfica como se comunican los componentes entre sí, facilitando el entendimiento del sistema a los nuevos desarrolladores.

La documentación de un programa explica el funcionamiento de este, sin embargo, a medida que el desarrollo de los programas avanza, estos pueden presentar cambios en su funcionalidad, la forma en la que se comunican los componentes, e incluso, en la arquitectura que estos presentan. Esto genera un problema, pues, la documentación debe ser actualizada junto con la modificación del programa, proceso que toma mucho tiempo, pues, hay que rectificar toda la documentación escrita y rehacer toda la documentación gráfica del programa.

A.5.1.1. UML y PlantUML

UML (Unified Modeling Languaje) es un lenguaje de modelamiento que provee un método estándar de visualizar el diseño de un sistema. UML fue desarrollado entre los años 1994 y 1995 con la finalidad de estandarizar las diferentes notaciones que eran usadas para representar los diseños de los programas.

PlantUML es un software Open-Source bajo la licencia GNU que permite la generación de diagramas que siguen el estándar UML mediante código escrito en texto plano. Esto tiene el beneficio de convertir los diagramas gráficos en código, reduciendo el costo de mantención de la documentación de los diagramas, ya que esta mantención pasa a ser parte de la mantención del código de la aplicación.

PlantUML define su propio lenguaje de programación para la generación de los diagramas. Este lenguaje permite la modularidad e importación de código de un archivo a otro, permitiendo añadir lógica a los diagramas que se quieren visualizar. Esto se traduce en que los componentes usados en los diagramas generados por PlantUML no tienen la necesidad de ser escritos dos veces, pues, estos son importables, ayudando esto en la manutención de la documentación, pues, las características de los componentes no se encuentran repartidas entre múltiples archivos.

El programa desarrollado en esta memoria implementa la documentación gráfica del sistema usando PlantUML, generando un diagrama para cada clase y paquete del programa. Además, se generan, usando los diagramas nombrados anteriormente, diagramas de la arquitectura del programa, mostrando de forma general como se comunican los principales componentes de este y como dependen entre ellos.

En la figura 4.3 es posible apreciar un ejemplo de como PlantUML representa los diagramas de clase, mostrándose en la imagen el paquete de la Escena.

A.5.1.2. Documentación de código

Además de la documentación de la arquitectura de la aplicación, se desarrolló documentación escrita al interior del código explicando que es lo que deberían hacer las funciones y clases programadas. Esta fue escrita para cada método, clase y archivo definido en la aplicación.

La documentación escrita sobre el código ayuda a futuros desarrolladores a saber qué es lo que hacen las distintas clases y métodos programados sin necesidad de tener que entender en su totalidad la lógica programada, facilitando procesos como la extensión o modificación de la aplicación.

En Python existe un componente llamado docstring que permite la escritura de texto explicativo de distintos elementos del lenguaje, como lo son clases, métodos y archivos. Este componente, a diferencia de los comentarios, si es considerado al momento de interpretar el lenguaje, almacenándose al interior del objeto en el cual fue definido. Esto permite que usuarios del programa puedan, en una consola de Python, llamar al método predefinido "help" con el nombre del componente del cual desean saber más y el componente pueda

entregarle información de este al usuario, no necesitando el usuario el saber cómo o donde está implementado el componente.

Los docstring no poseen un formato estándar con el cual deben ser escritos, sin embargo, existen varios formatos conocidos como NumPy, Epytext, reStructuredText y Google. En la aplicación implementada en esta memoria se hizo uso de este último formato estándar para la escritura de los docstrings.

A.6. Tests unitarios

Con la finalidad de comprobar que el funcionamiento de la aplicación desarrollada funciona correctamente, junto con desarrollar un mecanismo que permita a futuros desarrolladores el poder comprobar que los cambios que estos hagan a la aplicación no modifiquen la lógica ya existente en esta, se implementaron test unitarios en la aplicación.

Estos tests fueron desarrollados usando la biblioteca unittest que ofrece en lenguaje Python. Un ejemplo de estos se muestra a continuación:

Código A.1: Ejemplo de test unitario desarrollado en la aplicación.

```
class TestModifyHeight(TestCase):

    def setUp(self) -> None:
        # Genera variables necesarias para ejecutar los tests
        self.engine = Engine()
        self.program = Program(self.engine)
        ...

    def tearDown(self) -> None:
        # Elimina archivos generados por los tests
        self.program.remove_temp_files()
        ...

    def test_linear_transformation(self):
        # Ejecuta lógica del programa
        self.program.transform_linear(...)
        ...

        # Comprueba si los resultados obtenidos se corresponden con los esperados
        self.assertTrue(...)
        self.assertTrue(...)
```

Como se puede apreciar en el ejemplo, para la creación de un test, es necesario crear una clase que herede de clase TestCase, y definir en su interior métodos que definan los tests que se deseen ejecutar. Todos los métodos de la clase que tengan un nombre que comience con "test_" son los que la biblioteca unittest considerará como tests ejecutables, permitiendo esto la definición de otros métodos al interior de la clase.

La biblioteca también permite la definición de otros métodos como setUp, método que se ejecutará siempre antes de cada test definido al interior de la clase, o el método tearDown, método que se ejecutará siempre al final de cada test definido en la clase.

Como se aprecia en el ejemplo, al interior del método setUp se definen las variables que permiten acceder a la lógica de la aplicación, mientras que en el método de testing solamente se comprueba la funcionalidad de la aplicación.