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

DETECCIÓN, CONSTRUCCIÓN Y EVALUACIÓN DE POLÍGONOS NO SIMPLES A PARTIR DE IMÁGENES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

AMÉRICO IGNACIO FERRADA DÍAZ

PROFESOR GUÍA:
NANCY HITSCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
MAURICIO PALMA LIZANA
GONZALO NAVARRO BADINO

Este trabajo ha sido parcialmente financiado por Proyecto Fondecyt N° 1181506

SANTIAGO DE CHILE
2020

Resumen

En la minería, el trabajo de campo en conjunto con otras tareas se requiere contar con un análisis rápido y eficiente que permita evaluar los distintos riesgos asociados a derrumbes e inestabilidades. Con este trabajo se buscan explorar nuevos mecanismos computacionales para facilitar este análisis. Se explora el proceso de interpretación de las estructuras geométricas presentes en la roca, a partir de conjuntos de segmentos, para luego evaluar soluciones aproximadas, que permitan a su vez generar soluciones más eficientes, acelerando el proceso de predicción de los derrumbes.

Esta memoria muestra el desarrollo de una aplicación que permite detectar, construir y evaluar cortes de roca para posteriormente modelarlos en una salida de polígonos no simples y fracturas. La aplicación recibe de entrada, como mínimo un conjunto de segmentos que representan las fracturas iniciales y opcionalmente recibe además una imagen representativa del área de análisis, que contiene las distintas regiones que representan el área descrita por las roca presente en la imagen.

Durante el desarrollo de la memoria se descubren las diversas dificultades que surgen al intentar identificar los elementos de las estructuras y diferenciar los diversos componentes de la roca que requieren análisis de estabilidad. Para enfrentar las dificultades se plantea una solución adaptable y extensible para probar múltiples opciones. Específicamente se desarrollando dos alternativas para esta problemática, la primera una solución exacta sin considerar la imagen, utilizando cálculo preciso para recuperar la estructura geométrica. La segunda una solución aproximada construida para aprovechar el manejo de imágenes en el cálculo de los polígonos. Al interpretar las estructuras mediante grafos, se pueden identificar las geometrías necesarias para separar los diversos componentes de forma robusta y precisa.

Finalmente se muestran distintas validaciones para los algoritmos implementados y la corroboración de la correcta generación de estructuras a partir de la información proveniente de las rocas, la generación de un software robusto ante distintas configuraciones de segmentos. Por último, se obtuvo una diferencia en el resultado de los algoritmos desarrollados (basado en imágenes y método exacto), dado que el método basado en imagen obtiene en proporción un mayor número de polígonos encontrados, aunque no necesariamente correctos.

Este trabajo está dedicado a mi Familia que me ha empujado a seguir adelante. A mis amigos que me han mostrado un ejemplo a seguir. Y a mi profesora Nancy que me ha guiado a lo largo de toda la carrera para no rendirme.

Tabla de Contenido

1. Introducción **1**
1.1. Motivación . . . 1
1.2. Problema . . . 2
1.3. Relevancia . . . 3
1.4. Objetivos . . . 3
1.4.1. Objetivo General . . . 3
1.4.2. Objetivos Específicos . . . 3
1.5. Descripción general . . . 4
1.6. Resultados . . . 4

2. Marco Teórico **5**
2.1. Conceptos Geométricos . . . 6
2.1.1. Vértices y Segmentos . . . 6
2.1.2. Polígonos . . . 6
2.1.3. Concavidad y Convexidad . . . 6
2.2. Algoritmos de Intersección . . . 6
2.2.1. Intersección de Segmentos . . . 6
2.2.2. Punto al interior de un Polígono . . . 8
2.3. Grafos . . . 8
2.3.1. Grafo Plano . . . 9
2.4. Robustez y precisión . . . 9
2.5. PixelRegión2D . . . 10
2.6. *Detri2qt* . . . 10

3. Análisis **12**
3.1. Propuestas . . . 12
3.2. Requisitos . . . 13
3.3. Métricas . . . 13
3.4. Metodología . . . 14
3.5. Arquitectura . . . 14
3.6. Definiciones . . . 15
3.6.1. Fracturas . . . 15
3.6.2. Cortes de roca . . . 15

4. Diseño e implementación **17**
4.1. Estructuras de Datos . . . 17

4.1.1. Estructura Interna 17
4.1.2. Estructura intercambio de datos 19
4.2. Algoritmos 20
4.2.1. Carga de segmentos 21
4.2.2. Casos problemáticos 21
4.2.3. Calculo intersecciones 22
4.2.4. Recorrido 23
4.2.5. Identificación de polígonos, agujeros y fracturas 25
4.2.6. Desde imagen 27
4.3. Diseño e implementación de interfaz 28
4.3.1. Modelo 28
4.3.2. Vista 28
4.3.3. Controlador 30

5. Validación **32**
5.1. Solución exacta 32
5.2. Comparación de algoritmo exacto y algoritmo aproximado 32

6. Conclusiones **36**

Bibliografía **37**

Apéndices **39**

A. Apéndices **40**
A.1. Código implementado 40
A.2. Imágenes Validación 41

Índice de Tablas

Índice de Ilustraciones

1.1. Ejemplo de imagen 2D resultante de la intersección de segmentos. Cada color es un polígono distinto. 2
1.2. (A) Representación de líneas rasterizadas. (B) Resultado esperado: polígono en rojo, fracturas internas en verde. 4

2.1. Ejemplo intersección entre segmentos. 7
2.2. Algoritmo de Shimrat. 9
2.3. Aproximación de puntos. 10

3.1. Ejemplo de la estructura de un polígono. 15
3.2. Ejemplo de la estructura de un polígono con dos agujeros y distintos tipos de fractura. 16

4.1. Carga de segmentos. 21
4.2. Revisión de colisiones. 22
4.3. Grafo de una intersección. 23
4.4. Imagen de una fractura. 24
4.5. Imagen de una fractura entre dos polígonos. 25
4.6. Imagen de como se encuentra un bucle. 26
4.7. Ejemplo de la interfaz. 29

5.1. Ejemplo del análisis polígono símple. 33
5.2. Ejemplo del análisis polígono con dos agujeros. 34
5.3. Ejemplo del análisis de datos reales. 35

A.1. Implementacion Algoritmo interseccion. 40
A.2. Ejemplo del análisis polígono con dos agujeros. 41
A.3. Ejemplo del análisis polígono fracturas internas (cóncavo). 42
A.4. Múltiples fracturas sin polígono. 43
A.5. Ejemplo del análisis polígono múltiples fracturas y agujeros. 44
A.6. Ejemplo del análisis polígono con fracturas muy ramificadas. 45

Capítulo 1

Introducción

1.1. Motivación

El presente trabajo de memoria se sitúa en el contexto del diseño de algoritmos eficientes y robustos para simular fracturas producidas en rocas durante procesos de extracción minera y determinar su impacto en la estabilidad de las rocas restantes. Las fracturas asociadas al proceso de extracción pueden producir que ciertas rocas queden de manera inestable, apoyadas débilmente unas con otras, lo cual puede resultar peligroso si finalmente se originan derrumbes, afectando de manera significativa a personas y económicamente si además dañan la infraestructura minera.

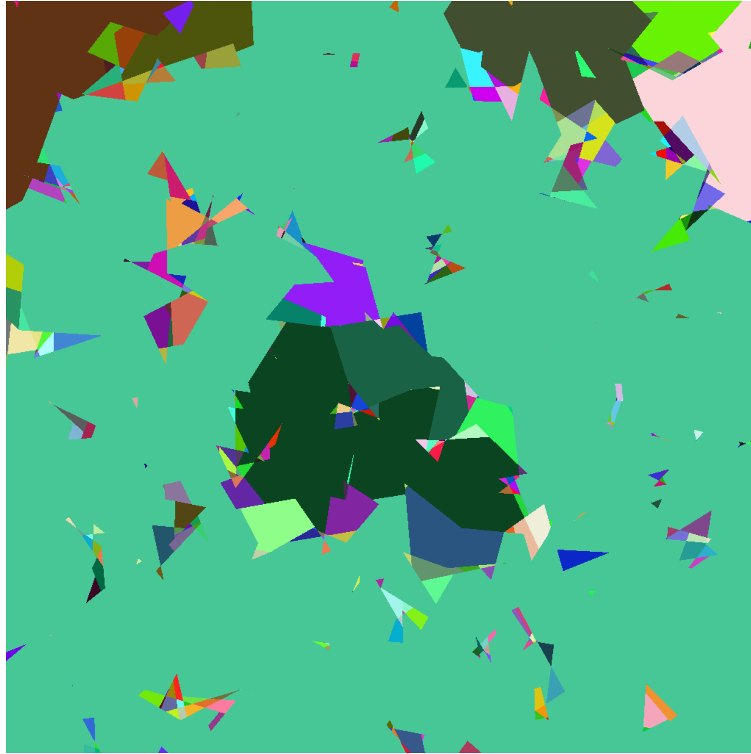
Si bien el problema real tiene características tridimensionales, el presente trabajo de memoria desarrolla un sistema computacional (bidimensional) para modelar las fallas/fracturas en una roca y determinar qué tipo (forma) de rocas más pequeñas se puede producir durante el proceso. En el ámbito de la geometría computacional, este problema se reduce a calcular la forma y volumen de poliedros. Esto último se define como un sólido cerrado, compuesto por caras (polígonos), arcos (segmentos) y vértices (puntos) formado por un número finito de planos. El poliedro resultante puede tener diversas formas, arcos y caras colgantes en el interior.

Para realizar un análisis de la problemática bidimensional, se utilizó el algoritmo *PixelRegión2D*[1] que recibe como entrada un conjunto de segmentos, cada uno define una fractura en 2D y genera como salida una imagen. En la imagen generada, cada polígono queda identificado por un color diferente (Ver Figura 1.1), donde el algoritmo aprovecha el coloreo de la pantalla y genera las regiones que representan los polígonos sin necesitar el cálculo de intersecciones de segmentos.

Como se observa en la Figura 1.1, la forma de los polígonos resultantes puede ser muy diversa y compleja: pueden existir polígonos dentro de otro, segmentos colgantes y vértices aislados. Debido a esto el algoritmo no calcula explícitamente las intersecciones, sino que

[1] *PixelRegión2D*, el cual es código está basado en *Painter's Algorithm*, y fue escrito por el Dr. Benoît Crespin

Figura 1.1: Ejemplo de imagen 2D resultante de la intersección de segmentos. Cada color es un polígono distinto.



genera una imagen que representa parte de la solución al problema. Esta solución evita los errores de precisión y casos especiales, muy comunes en los algoritmos de geometría computacional.

La salida del algoritmo usa una imagen aproximada de las regiones, pero esta información no contiene la geometría necesaria para generar los estudios de minería requeridos.

1.2. Problema

A partir de la problemática descrita, se generan dos principales líneas de trabajo. La primera consiste en construir la geometría, en formato de polígonos, que representan los segmentos(fracturas) generados en el proceso de extracción minera. Se debe por lo tanto construir, analizar y almacenar las estructuras generadas a partir de los cortes de roca. En la construcción de esta geometría, se utiliza toda la información geométrica disponible, obteniendo como resultado una representación de la manera más exacta posible.

La segunda línea de trabajo corresponde a un problema de investigación, que busca elaborar un mecanismo aproximado para generar las estructuras, aprovechando la información obtenida a partir de las imágenes, como un dato aproximado de las regiones analizadas. Se intenta, por lo tanto, comparar la calidad de las geometrías recuperadas por el mecanismo aproximado, con la solución desarrollada con mayor precisión en la primera línea de trabajo antes descrita. Con el resultado obtenido, se espera establecer si la solución aproximada es suficientemente confiable.

1.3. Relevancia

Este problema es estudiado con el objetivo de extraer propiedades geométricas de las estructuras calculadas, donde un ejemplo puede ser analizar la proporción entre el área de la cerradura convexa de los polígonos y el área del mismo. La cerradura convexa corresponde a una figura matemática que representa el polígono convexo más pequeño que envuelve al conjunto de puntos. Esto es requerido para estudiar la estabilidad de un corte en la roca que se está modelando.

El resultado de este trabajo podrá ser utilizado como entrada de un generador de mallas de triángulos. Un ejemplo es *Triangle* [8] un software que lee como entrada un conjunto de polígonos, segmentos y puntos, tales como los calculados en esta memoria, y los transforma en un conjunto de triángulos, que se pueden utilizar en simulaciones numéricas.

Encontrar un mecanismo aproximado para obtener estas geometrías es prometedor a la hora de trabajar con problemas que puedan sacrificar precisión por velocidad. Un ejemplo puede ser trabajos que requieran información en tiempo real, en el cual la velocidad de las decisiones es más importante que la precisión de estas.

1.4. Objetivos

1.4.1. Objetivo General

Generar una aplicación que permita detectar, construir y evaluar cortes de roca, representados por polígonos no simples y fracturas. La aplicación debe recibir como input un conjunto de segmentos que representan las fracturas iniciales y aceptar como entrada una imagen que represente las regiones. En base a estos datos debe entregar como resultado, un conjunto de vértices(puntos), arcos(segmentos) y polígonos que representan la geometría contenida de la imagen. Además, debe permitir calcular propiedades sobre las estructuras generadas tales como la convexidad de los polígonos. La información se almacenará en un archivo de salida. Este debe estar en formato *planar straight line graph* (pslg) [2], para poder especificar polígonos y segmentos (fracturas) dentro de los polígonos.

1.4.2. Objetivos Específicos

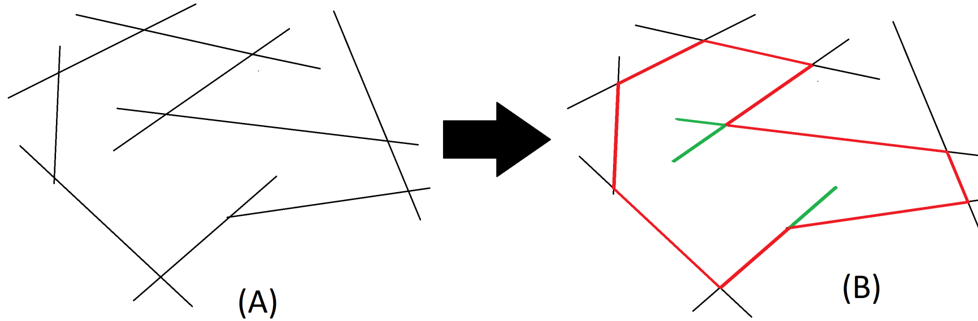
1. Desarrollar un algoritmo que permita construir los polígonos(rocas) y detectar las segmentos (fracturas) desde un conjunto de segmentos en forma exacta.
2. Desarrollar una solución que permita construir los polígonos(rocas) y detectar las segmentos(fracturas) utilizando segmentos e imágenes para aproximar las regiones.
3. Definir el conjunto de salida y su estructura.
4. Implementar Interfaz que permita analizar y comparar resultados obtenidos.
5. Añadir sistemas de pruebas y carga de ejemplos comparativos.
6. Almacenar los polígonos y segmentos encontrados en formato *pslg*[2].

[2] *Planar straight line graph*, es una forma de representar una estructura en dos dimensiones donde cada uno de los elementos del plano son segmentos de recta, puntos y polígonos. Permite puntos, segmentos aislados y polígonos con agujeros.

1.5. Descripción general

El trabajo se dividió en dos fases, la primera enfocada en la recuperación de estructuras a partir de conjuntos de segmentos para evaluar su forma y luego para almacenarlos en un formato específico de geometría vectorial. Este formato es requerido para su uso posterior en simulaciones numéricas, fuera del alcance de esta memoria. Un ejemplo de salida lo podemos ver en Figura 1.2.

Figura 1.2: (A) Representación de líneas rasterizadas. (B) Resultado esperado: polígono en rojo, fracturas internas en verde.



Un polígono es una estructura geométrica en dos dimensiones que demarca un área finita y conexa en el espacio. Un polígono simple es una polilínea cerrada, cuyos segmentos no consecutivos no se intersectan. Por otra parte, las fracturas son una estructura que definimos en este proyecto. Esta se representa con un segmento que corta la superficie interna de un polígono pero que no logra dividir su superficie completamente. En el caso de esta memoria se trabajará solo con polígonos formados por segmentos rectos, sin segmentos curvos.

Para una descripción más profunda de Polígonos o más conceptos geométricos, se puede revisar la sección Conceptos Geométricos en el Marco Teórico.

La segunda fase se centró en el desarrollo de una solución aproximada, la cual trabaja con elementos código externo para intentar aproximar una solución.

1.6. Resultados

En general el trabajo fue enfocado al desarrollo de la aplicación que cumpliera con recuperar la geometría, polígonos y fracturas, a partir del conjunto de segmentos. En esta sección se logró establecer un modelo que permite en casos generales cumplir esta labor, lo cual fue estudiado y validado con casos de prueba y estudio de las entradas y salidas del sistema.

Subsecuentemente se consiguió un caso base para poder generar las pruebas en la recuperación de datos desde las imágenes, donde se consiguió encontrar los conjuntos de datos pertenecientes a cada polígono por un mecanismo utilizando la imagen, los segmentos. Aún así no se logró establecer una geometría sin el cálculo de intersecciones.

Capítulo 2

Marco Teórico

A pesar de que existen trabajos similares, el problema abordado por esta memoria no ha sido desarrollado completamente. Las soluciones existentes para detectar polígonos trabajan calculando intersecciones entre aristas, lo cual funciona bien para aplicaciones donde se conoce la geometría esperada. Sin embargo, los algoritmos que utilizan cálculos de intersecciones sufren problemas de precisión y casos especiales, necesitando en general de soluciones complejas y propensas a errores.

Dentro de las soluciones encontradas se observaron posibles candidatos en la literatura, pero como problemas individuales, y que no abarcan completamente la solución. Dentro del principal material de estudio se usó *Computational Geometry: Algorithm and Applications*[1], en el cual se encontraron soluciones a problemas individuales, como el cálculo de intersecciones por segmentos, en el capítulo 2.1, además del estudio del algoritmo mencionado anteriormente de *PixelRegión2D*[1] el cual se basa en *Painter's Algorithm*, capítulo12 del mismo libro[1].

Otro requerimiento es tener en cuenta que la geometría esperada no es del tipo estándar; pueden existir polígonos dentro de otro, segmentos y puntos aislados, cosas que los algoritmos tradicionales no consideran y generan errores. Un algoritmo interesante a tener en cuenta es el descrito en [7]; este resuelve un problema similar, pero requiere contar con los puntos de las intersecciones.

Dentro de las limitaciones encontradas, se contabilizaron varios puntos que se tendrán en cuenta durante el desarrollo. Por una parte, se encuentra que desarrollar código más eficiente tiende a producir métodos poco legibles que dificultan la experimentación y generalización del código. Por otra parte, al ser un problema de geometría computacional, se prevé encontrar posibles errores numéricos al utilizar números reales y utilizar operaciones binarias finitas. Finalmente, este la solución generada pretende aplicarse a problemas reales y no se puede tomar como experimento completamente válido usado solo conjuntos generados de forma aleatoria; debemos probar las construcciones generadas por los algoritmos en base a algunos ejemplos reales.

2.1. Conceptos Geométricos

En esta sección abordaremos los conceptos geométricos necesarios para entender la presente memoria de título.

En primera instancia se realizará una descripción general de cada una de las definiciones, posteriormente se profundizará en las operaciones necesarias utilizadas durante la memoria.

2.1.1. Vértices y Segmentos

Un vértice es un punto en el espacio perteneciente a una figura geométrica. En un espacio dos dimensiones se puede determinar completamente un vértice con dos coordenadas. En el caso específico de esta memoria se trabaja con coordenadas cartesianas que corresponden a la denominación x,y y representan el desplazamiento en cada uno de sus ejes de coordenadas.

Los segmentos son un fragmento de una recta, este se representa como la unión entre dos puntos o vértices, tienen un origen y un punto de destino. [2]

2.1.2. Polígonos

Un polígono es un conjunto finito de segmentos, los cuales están unidos por sus extremos. Un polígono cerrado es una figura geométrica que demarca un área finita. Los polígonos pueden tener distintas características. Para las definiciones del presente trabajo se consideran los polígonos que solo poseen segmentos rectos, y que su borde siempre demarca un área cerrada no nula.

2.1.3. Concavidad y Convexidad

La concavidad y convexidad son propiedades de las figuras geométricas. En el caso de las figuras en dos dimensiones, se define como convexo a cualquier polígono que para cualquiera de los puntos pertenecientes a su borde o en el interior, si se une mediante una recta, todos los puntos de esta recta pertenecerán al polígono. Se define así, donde la concavidad son los elementos que no cumplen esta propiedad.

Para el estudio de esta memoria es importante que estas propiedades permitan obtener algunas características estructurales sobre las figuras geométricas, pues el calcularlas es relevante en el estudio de la estabilidad de las rocas.

2.2. Algoritmos de Intersección

2.2.1. Intersección de Segmentos

Supongamos que tenemos dos segmentos representados como de a a b ($\vec{ab}$) y de c a d ($\vec{c\text{d}}$). Entonces cualquiera de los puntos del primer segmento puede ser representado como $a + (b - a)t$ y el otro como $c + (\text{d} - c)s$ donde s y t son dos parámetros escalares. imagen y definición extraída con autorización de Joaquín Torres [11]

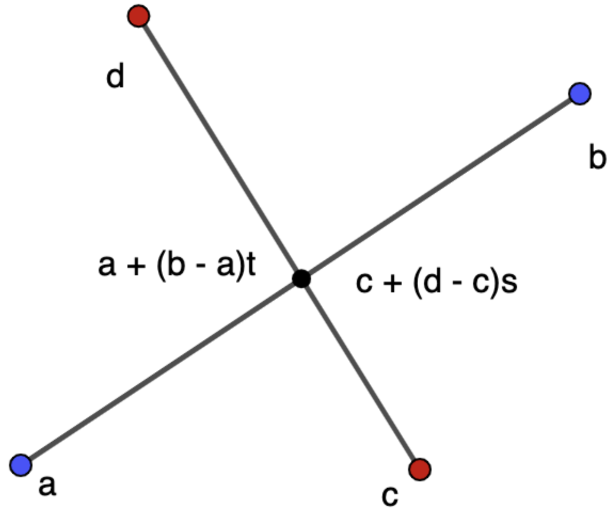


Figura 2.1: Ejemplo intersección entre segmentos.

Los dos segmentos intersectan si podemos encontrar t y s tal que $a+(b-a)t = c+(\mathrm{d}-c)s$. Para encontrar la solución a esta ecuación, necesitamos las siguientes operaciones.

Primero necesitamos encontrar el producto cruz entre $(\mathrm{d} - c)$ resultando en:

$$(a + (b - a)t) \times (\mathrm{d} - c) = (c + (\mathrm{d} - c)s) \times (\mathrm{d} - c), \tag{2.1}$$

Nosotros sabemos que $(\mathrm{d} - c) \times (\mathrm{d} - c) = 0$, esto significa

$$t((b - a) \times (\mathrm{d} - c)) = (c - a) \times (\mathrm{d} - c), \tag{2.2}$$

Además, resolviendo para t:

$$t = ((c - a) \times (\mathrm{d} - c))/((b - a) \times (\mathrm{d} - c)), \tag{2.3}$$

Dela misma manera, podemos resolver para s, y así obtenemos la siguiente ecuación:

$$s = ((a - c) \times (b - a))/((\mathrm{d} - c) \times (b - a)), \tag{2.4}$$

Para reducir el costo computacional, podemos reescribir la ecuación 2.4 cómo la siguiente $a \times b = -b \times a$:

$$s = ((c - a) \times (b - a))/((b - a) \times (\mathrm{d} - c)), \tag{2.5}$$

Reemplazando $b-a$ con p y $\mathrm{d}-c$ a q podemos escribir la ecuación cómo:

$$t = (c-a) \times q/p \times q, \tag{2.6}$$

$$s = (c-a) \times p/p \times q. \tag{2.7}$$

Para ésta ecuación, podemos encontrar cuatro posibles resultados:

1. Si $p \times q = 0$ y $(c-a) \times p = 0$, Entonces las dos rectas con colineales. En este caso, podemos expresar los valores de (c y d) en términos de una ecuación de primer grado como $(a + (b-a)t)$:

   $$t_0 = (c-a) \cdot p/(p \cdot p),$$
   $$t_1 = (c+s-a) \cdot p/(p \cdot p) = (t_0 + s) \cdot p/(p \cdot p).$$

   Si el intervalo entre t_0 y t_1 intersecta el intervalo $[0,1]$, entonces los segmentos son colineales y se sobreponen; En otro caso, estos son colineales disjuntos.
2. Si $p \times q = 0$ y $(c-a) \times p \neq 0$, entonces las dos líneas son paralelas y no intersectan.
3. Si $p \times q \neq 0$ y $0 \leq t \leq 1$ y $0 \leq s \leq 1$, los dos segmentos se interectan en el punto $a + (b-a)t = c + (\mathrm{d}-c)s$.
4. De otra forma, los segmentos no son ni paralelos ni intersectan.

Este algoritmo es el método en dos dimensiones del artículo "Intersection of two lines in three-space" de Ronald Goldman, publicado en Graphics Gems, página 304 [3].

2.2.2. Punto al interior de un Polígono

En este trabajo necesitamos un mecanismo para saber si un punto está al interior de un polígono[11]. Podemos encontrar información en [5] solución que revisaremos. La versión que más conocida que resuelva este problema es la solución de Shimrat[9].

El algoritmo de Shimrat, utiliza un rayo trazado desde el punto que necesitamos evaluar, y usualmente este continúa por el eje X. Este algoritmo busca las intersecciones entre el rayo y el borde del polígono, las cuales necesitamos contar. Si el número de intersecciones es impar, entonces el punto está dentro del polígono; de otra forma está afuera.

La figura 2.2, extraído desde [5] pagina 136, muestra el algoritmo de Shimrat. La imagen izquierda muestra el algoritmo usando los puntos P_1 y P_2. Este usa la línea horizontal hasta el ∞. Los símbolos X representan donde intersectan los polígonos. En b se muestra como este algoritmo no funciona para polígonos abiertos.

2.3. Grafos

Un grafo está definido cómo un conjunto de nodos con conexiones. La teoría de grafos describe las propiedades que poseen los grafos y las operaciones que se pueden realizar sobre

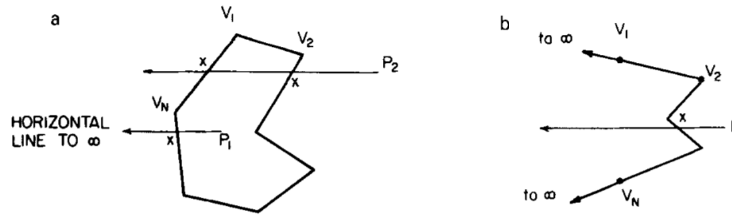


Figura 2.2: Algoritmo de Shimrat.

ellos.

En la presente memoria se utilizan para modelar la geometría e identificar los bordes de los polígonos. Los grafos han sido ampliamente estudiados y usados para resolver distinto tipos de problemas de almacenamiento de datos y la conexiones entre ellos. Sus aplicaciones van desde bases de datos y redes hasta problemas geométricos.

2.3.1. Grafo Plano

Para modelar los bordes de los polígonos esperados, se utilizaron las intersecciones entre los distintos segmentos cómo nodos de un grafo para modelar. En consecuencia, se observó que las estructuras generadas al intersecctar los segmentos corresponden a un grafo planar.

Los grafos planos o planares son grafos que al representarlos en un plano, sus conexiones(artistas) no se cruzan entre ellas. Esto se puede observar durante la construcción del grafo, dado que, si existiera una intersección, entonces se agrega un nuevo nodo al grafo para que no se crucen las conexiones.

Los grafos planos permiten aplicar operaciones garantizando que no existen segmentos que no estén correctamente representados en esta estructura de datos. Mediante la fó rmula de Euler, podemos conseguir una validación de que las estructuras están correctamente construidas, en el cual nuestro trabajo es encontrar las caras de la formula.

2.4. Robustez y precisión

La precisión de los datos es un tema importante dentro de la geometría computacional, normalmente genera problemas a la hora de trabajar con datos reales, dado que los computadores son máquinas discretas que no pueden utilizar estas unidades sin aproximaciones.

Dentro de este trabajo la precisión en general no es tan problemática, dado que la solución esperada no requiere una precisión exacta, por lo cual se ocuparon métodos para evitar los problemas de precisión cómo utilizar aproximaciones donde se definen variables de tolerancia, o unir estructuras cercanas que aporten poca información a la estructura general.

Como se observa en la figura 2.3, dos puntos separados por una distancia A y otros dos puntos separados por una distancia B, para la aproximación si los dos puntos están más cerca que el radio de tolerancia el cual es representado por el radio de los círculos, se concederá como un solo elemento.

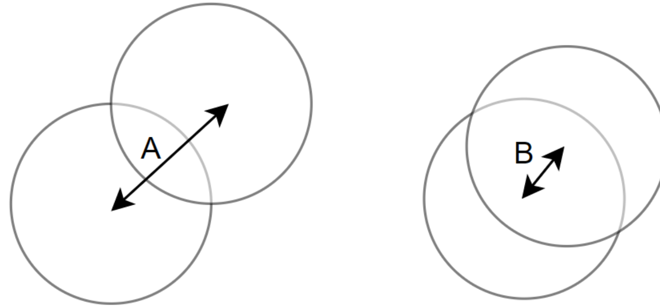


Figura 2.3: Aproximación de puntos.

Más información sobre este tema puede encontrarse en la publicación [6], donde se abarcan los problemas encontrados en la precisión, información que fue considerada pero que no implico grandes cambios en el modelo implementado.

2.5. PixelRegión2D

PixelRegión2D es un algoritmo paralelo creado por el Doctor Benoît Crespin que trabaja con segmentos para encontrar las encontrar estructuras poligonales aproximadas que fueron la inspiración al presente trabajo de memoria. Dentro de este trabajo se destacan los logros de ser sumamente eficiente y funcionar en paralelo utilizando GPU. El concepto detrás de este desarrollo es que se permite construir las regiones de un conjunto de segmentos dependiendo solo de la resolución de la pantalla, logrando dibujar con distintos colores cada una de las regiones encontradas. Sin embargo, la imagen no logra mantener la información de los segmentos ni de los bordes, y a pesar de que permite identificar las regiones no va más allá de identificarlas aproximadamente sin generar.

Este código es utilizado dentro de este proyecto como una herramienta para obtener información sobre las entradas con que se trabaja, donde uno puede conseguir una imagen representativa de las regiones que delimitan los segmentos como optimizaciones a la hora de construir las estructuras.

2.6. *Detri2qt*

Detri2qt[10] es un software con un objetivo educacional que trabaja con triangulaciones y diagramas de Voronoi. Este software ha sido desarrollado por el Doctor Hang Si, para mostrar el comportamiento de las triangulaciones de Delaunay en distintos escenarios. Este software es útil dado que, con el objeto del presente proyecto, las triangulaciones pueden representar los polígonos con que buscamos trabajar.

Este software lee archivos con formato .poly, los cuales permiten ingresar los segmentos de polígonos, en conjunto con identificadores de regiones para generar triangulaciones. Esta herramienta es utilizada en el presente proyecto a la hora de trabajar con trabajar sobre la

solución aproximada, ya que permite trabajar directamente con regiones aproximadas.

Capítulo 3

Análisis

En este capítulo se abordan las problemáticas encontradas durante el desarrollo de la memoria, también se muestran los distintos conocimientos adquiridos necesarios para entender el trabajo realizado. En primera instancia se muestra un desglose de los objetivos generales y específicos y las distintas propuestas que cumplen con los puntos planteados.

Se aborda el conocimiento adquirido para el correcto entendimiento del trabajo. Además, se explican tanto las decisiones como las distintas alternativas analizadas para los diversas problemáticas presentadas.

3.1. Propuestas

Al comenzar la memoria se tomó como base un proyecto previo enfocado básicamente en recuperar la información de la resolución de pantalla con la cual generar una imagen para capturar las regiones. Como se describió en la introducción, el objetivo consiste en: recuperar la información de un conjunto de segmentos, de los cuales sólo se conoce la información de sus coordenadas y con esta información construir la geometría, el conjunto de polígonos y fracturas.

La primera alternativa presentada consistía en modificar el código original, donde se recorrían píxeles de la pantalla para separar las regiones y con esto encontrar los elementos que fueran vecinos. Esta alternativa fue rápidamente descartada. Esto dado que al basarnos en un método aproximado no tendríamos un método de comparación con el cual analizar si la solución encuentra los polígonos representan correctamente la geometría.

Se optó finalmente por desarrollar primero una la solución exacta, con el objeto de recuperar la información de las estructuras. Además, la solución exacta permitiría generar un modelo base donde sustentar comparaciones y analizar las otras alternativas.

3.2. Requisitos

Dado que se planteó desarrollar la solución exacta como base del proyecto, surgieron varios requisitos naturalmente. Los primeros requisitos observados desde el punto de vista de robustez y extensibilidad de la aplicación a desarrollar son los siguientes:

- El diseño de la aplicación debe permitir que nuevas estrategias pueden ser fácilmente integradas y comparadas con las diseñadas e implementadas en esta memoria. Con le objetivo de tener la opción de comparar múltiples estrategias.
- Se debe diseñar una manera de comparar los polígonos generados de tal forma de encontrar posibles errores en la solución exacta. Para poder pulir la solución visualmente, dado que no existe un mecanismo para generar una solución matemáticamente exacta.
- La aplicación debe poder leer el formato de segmentos preestablecido y comunicar información a otros programas como detri2qt. Este es un requisito establecido para utilizar mallas de triángulos, las cuales son compatibles con más sistemas de modelamiento.

Lo anterior surgió dado que se requería estudiar el problema exacto comparando las soluciones manualmente, donde el usuario necesita interactuar y verificar la integridad y la calidad de los polígonos encontrados.

Luego de establecer algunas bases del código a implementar, se formaron los lineamientos para el desarrollo de los algoritmos:

- La solución exacta debe ser única, y debe ser invariable para el mismo conjunto de datos iniciales.
- La geometría debe ser resistente a problemas de precisión y debe entregar una solución válida.
- El conjunto de elementos está limitado por una caja de tamaño determinada.
- Los polígonos encontrados pueden ser no simples y contener fracturas, pero sus bordes deben describir un área no nula en todo su contorno.

3.3. Métricas

A la hora de trabajar con la solución exacta, se llegó a la conclusión de que no existe una forma simple de comparar la precisión de los datos entre distintas estructuras. Por lo cual, para el desarrollo de la solución exacta, se corrigieron los errores mediante visualización gráfica de los distintos elementos. En base a esto, las situaciones que generaron errores se almacenaron con el fin de verificar futuras soluciones similares. Estas pruebas se muestran de forma más específica en el capítulo de validación de este trabajo. Aun así, este fue un desarrollo progresivo con elementos de prueba cada vez más complejos.

Por otra parte, al utilizar de base el modelo exacto, nos permite establecer algunos medios para comparar las soluciones aproximadas. Las métricas planteadas corresponden a lo siguiente:

- Comparar el número de estructuras(polígonos) encontradas.

- Estimar la similitud entre sus estructuras, para esto establece una rúbrica donde mediante métodos exactos se comparan las estructuras generadas por los algoritmos aproximados, y se estima la diferencia de cobertura en las áreas de estos polígonos.

3.4. Metodología

Se pensó en una solución que pudiera englobar las distintas opciones analizadas. Para esto se tomó como base el lenguaje de programación de *Processing* que, utilizando el motor de Java, entrega algunas herramientas para describir elementos geométricos y hacer pruebas con un entorno gráfico. Además, este es el mismo lenguaje en que estaba implementado el algoritmo Pixel2Region el cual inspiró el trabajo de memoria.

Este lenguaje de programación fue escogido dado la simpleza a la hora de generar interfaz gráfica, y que al ser basado en Java permitía que una posible trascripción posterior fuera más simple.

La solución se planteó de la siguiente manera: en primera instancia se desarrolló el esqueleto de la solución general, el cual consistió en un software modular que resolviera todos los problemas planteados en la memoria. Este debía tomar como input los segmentos entregados y obtener como resultado los objetos que se requieren para analizar la estabilidad de las estructuras. Este código base permitiría generar el marco en el cual se pudieran comparar la alternativa de solución. El mecanismo planteado fue un desarrollo incremental con test, en el cual de inicio a partir de una solución básica y se generaron pruebas que los algoritmos debían superar, posteriormente al superarlas se generaron casos más complejos y así sucesivamente. Este mecanismo continuó hasta que se pudiera trabajar con los datos originales.

En segunda instancia con un medio de base para contrastar las posibles soluciones encontradas, se explora la opción de utilizar las regiones previamente calculadas como una aproximación. En esta sección se intenta limitar el número de elementos necesarios a comparar, aquí se debe contrastar las soluciones obtenidas mediante el cálculo exacto de los polígonos con los cálculos aproximados.

3.5. Arquitectura

Se estableció que para la interfaz con el usuario se implementaría un sistema MVC dado que se requiere una alta interacción con el usuario para generar las pruebas para el código base. El modelo MVC (Model View Controller) cuenta con tres secciones especializadas en una tarea específica y está enfocado especialmente para desarrollar interfaces humano-computador. Por una parte el Modelo(Model) se encarga de trabajar los distintos tipos de datos y sus respectivas interacciones. Por otro lado la Vista(View) es la encargada de mostrar la información en pantalla actualizando sus datos respecto a la información del modelo. Por último el Controlador(Controller) se encarga de implementar las interacciones y los cambios en el resto de esquemas. En el contexto específico de esta memoria se encarga de administrar los distintos algoritmos geométricos y modificar los valores del modelo.

3.6. Definiciones

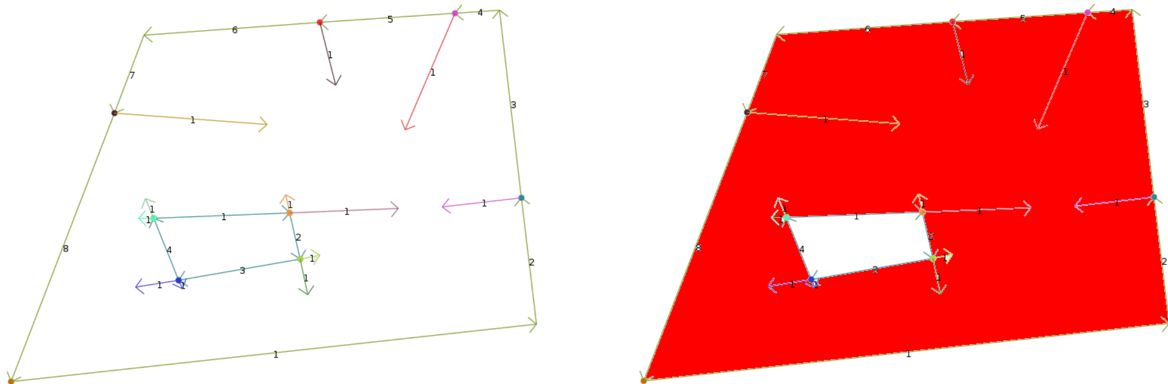
3.6.1. Fracturas

Durante el transcurso del trabajo se toma en consideración que las fracturas son: Un conjunto de segmentos que cortan un área descrita. Las fracturas pueden observarse en solitario o como un conjunto de éstas siempre y cuando no describen un área diferente.

3.6.2. Cortes de roca

Al trabajar con la salida de los datos, se construyó una estructura que pudiera interpretar correctamente los cortes de roca para cumplir la necesidad de los análisis mineros. Para modelar la geometría hay que establecer cómo funcionan los diversos elementos en los cortes de roca. Se estableció que todos los elementos del área continua, incluyendo los bordes y fracturas deben estar dentro de la estructura.

Figura 3.1: Ejemplo de la estructura de un polígono.

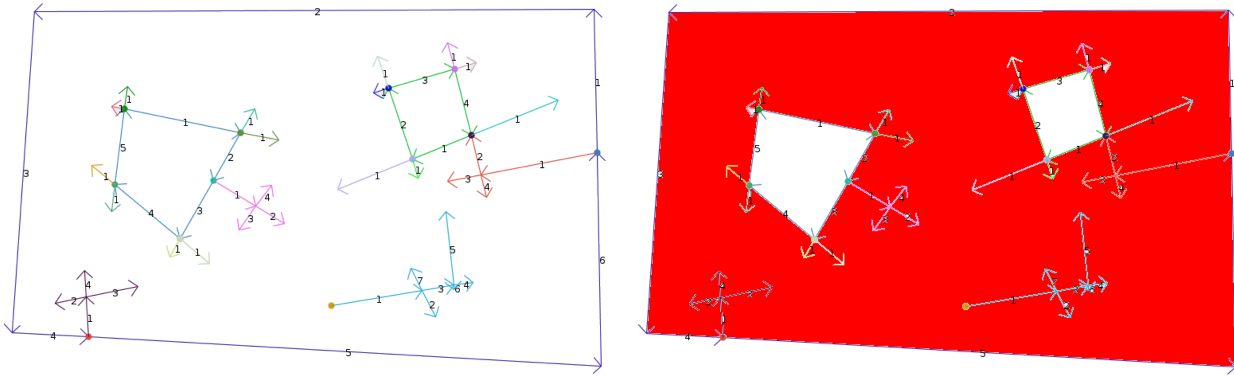


Cómo se observa en la figura 3.1, podemos observar todas las componentes de un corte de roca. Aquí notamos que en el área marcada en rojo corresponde al área que debe representar la roca que se está modelando. También se puede diferenciar que existen tres componentes para definir esta geometría. Por una parte, tenemos el borde de la estructura definido por un polígono exterior que envuelve la figura contenida. Por otra parte, tenemos un polígono interior que toma un área que no debe presentarse individualmente en otra construcción geométrica, pero que para el ejemplo que se nos muestra solo debemos considerarlo como un agujero. Por último, están las fracturas, que son los segmentos que cortan el área descrita pero que no separan la geometría analizada.

Como se muestra en la figura 3.2, una estructura puede demarcar un área, pero contener múltiples agujeros, cada uno de las áreas interiores se puede representar como un polígono secundarios al interior de nuestra representación. Además, hay que considerar que existen múltiples tipos de fracturas, en la figura 3.2, se pueden encontrar cuatro de los identificados.

- Fractura que se inicia en el borde del polígono (Esquina inferior izquierda).
- Fractura que se inicia en un agujero (Fracturas alrededor del agujero izquierdo).

Figura 3.2: Ejemplo de la estructura de un polígono con dos agujeros y distintos tipos de fractura.



- Fractura que une un agujero con el borde (Fractura entre agujero derecho y pared derechos).
- Fractura que corta el área descrita pero no intersecta con ningún polígono interior (Fractura debajo de agujero derecho).

Además de los mostrados en esta representación existen las fracturas que unen dos agujeros.

Capítulo 4

Diseño e implementación

4.1. Estructuras de Datos

4.1.1. Estructura Interna

El manejo de datos es importante para entender el funcionamiento del software implementado en las soluciones computacionales. Por lo cual en esta sección se explican los distintos elementos utilizados dentro del trabajo y cómo se interpretaron.

La estructura general de los datos se manejó manteniendo un diseño orientado a objetos, lo cual se facilita al utilizar un motor de Java como Processing para su desarrollo.

Segmentos

La primera estructura que se describe son los segmentos, esto es dado que es importante entender la entrada de datos. Los datos se reciben desde la entrada como dos puntos, el objeto aquí representado permite interpretar y almacenar los segmentos agregados por la entrada. La idea es que al generar un nuevo valor para la entrada los valores de los segmentos son inmutables, por lo cual esta información no se transforma, solo se utiliza para construir el resto de información a procesar. Los métodos aquí descritos se utilizan para generar nuevas estructuras, o para analizar si la entrada cumple con las condiciones para ser utilizada.

Clase 1: Segmento

```
p1, p2 <- Puntos del segmento;
Segmento(p1, p2);                       // Constructor de la estructura
boolean esColinear(Segmento seg);
void intersectar(Segmento seg);
```

El constructor se encarga de almacenar los puntos que representa el segmento, esta información es solamente almacenada y no se permite su edición posterior en otras partes de la aplicación. El método esColinear revisa con otro segmento si estos son colineales, este método se utiliza en un preproceso para considerar o no los segmentos correspondientes. Intersectar calcula el punto de intersección con el otro segmento y revisa si esta al interior entre estos,

este método se describe en el marco teórico 2.2.1, y más adelante en la sección Cálculo de Intersecciones 4.2.3.

Vértice

Los vértices son la estructura central de la implementación. Cada vértice contiene la información sus coordenadas y de las conexiones con otros vértices y corresponde a un nodo del grafo. Al contrario de cómo se comportan los segmentos, los vértices son mutables, necesitan actualizarse con los nuevos elementos agregados.

Funcionan mediante un esquema de grafo, cada vértice mantiene guardada su posición espacial, y almacena un mapeo a cada uno de los otros vértices con quién está conectado y a través de qué segmento lo está, esté mapeo se realiza mediante un diccionario, que almacena como llave al vértice con quien se está conectando, y como valor el segmento común a cuáles los dos vértices pertenecen. Con la información de sus vecinos el vértice implementa las funcionalidades de viajar a través de los elementos, además implementa un mecanismo para guardar el recorrido que ha realizado, lo cual es útil para identificación de bordes y fracturas.

Clase 2: Vértice

```
p1 <- Coordenada del vertice;
conexiones<Vertice, Segmento><- Mapa de los elementos con quienes está
  conectado.;
Vertice(p1);                                   // Constructor de la estructura
agregarConexion(Vertice, Segmento);
quitarConexion(Vertice);
obtenerConexiones();
siguienteVertice(Vertice);
orientar(Vertice, Orientacion, Color);
```

Lo mostrado en este pseudocódigo es una versión sumamente simplificada de los métodos que se requieren para implementar esta estructura. Los métodos de agregar, quitar y obtener están destinados a la etapa de interpretación de los segmentos, donde se generan todos los vértices y se crean todas las conexiones.

SiguienteVertice, se utiliza como un método auxiliar para encontrar el siguiente vértice a analizar. Este trabaja buscando el siguiente vértice más a la derecha posible relativo al vértice anterior.

Orientar es un método recursivo muy complejo. Este mantiene una lista de los vértices qué se han visitado y en qué dirección, su función es generar bucles y orientar los ciclos. Orientar se utiliza en la generación de los bordes de los polígonos y utiliza cómo método auxiliar el recorrido de siguienteVertice. Dentro de sus funciones tiene encontrar bucles, identificar fracturas no identificadas anteriormente, y marcar los elementos que son candidatos para visitar que no han sido catalogados.

Polígono

La estructura del polígono está construida para generar el formato de salida del sistema, almacena toda la información necesaria para describir un corte de roca. Esta información se describe más profundamente en Corte de Roca 3.6.2.

Clase 3: Polígono

```
Orientaciones <- Lista de segmentos que marcan el borde externo del polígono;
Vértices <- Vértices del borde del polígono;
Fracturas <- Lista de segmentos que cortan el interior sin traspasar el polígono;
Agujeros <- Lista de polígonos que están al interior del polígono representado;
Poligono(Orientaciones);                    // Constructor de la estructura
obtenerFracturasYAgujeros(Agujeros candidatos);
estaAdentro(Vertice);
```

La construcción de un polígono se hace en dos fases. Primero se construye el borde con una lista de segmentos llamados orientaciones. Las orientaciones permiten mantener una direccionalidad del recorrido y diferenciar si es un borde o un agujero. Más de cómo se exploran y se crean estas estructuras en 4.2.4. En la segunda fase se agregan los agujeros, donde se va revisando de los polígonos candidatos a agujero cuales son los más grandes que están dentro del polígono analizado. Aquí solo se consideran agujeros si es que no tienen otro agujero que los contenga que esté dentro del polígono.

EstaAdentro es un método que implementa el algoritmo de 2.2.2, se utiliza para identificar si un vértice está contenido dentro del polígono. Este método es importante para generar comparaciones entre estructuras y permite identificar si un agujero o una fractura están dentro de un polígono o no.

4.1.2. Estructura intercambio de datos

Existen distintos programas con los cuales es interesante comunicarse en el desarrollo del trabajo, cómo por ejemplo tenemos el software que nos permite obtener la imagen de las regiones escrito por el Doctor Benoît Crespin (PixelRegión2D). Este software originalmente no genera ningún formato válido de salida, pero fue modificado en el transcurso de la memoria para generar dos formatos.

El primer formato del archivo para almacenar los segmentos generados en forma aleetoria para probar el software PixelRegion2D. El formato que se eligio es .cvs. Estos archivos contienen los puntos utilizados para hacer pruebas de segmentos, donde los segmentos analizados con sus coordenadas. Por otro lado, como ayuda al sistema de imágenes se modificó el programa PixelRegión2D para generar un archivo de regiones en formato .matrix. Este archivo incluye la información de los píxeles de la pantalla donde se representan las regiones calculadas por PixelRegión2D, además incluye algunos indicadores como la cantidad de regiones encontradas en el formato aproximado, y la resolución de pantalla utilizada.

Por otro lado, se generó una comunicación con los sistemas de triangulaciones de [10] y [8], al trabajar con formatos de salida en estructura .poly.

4.2. Algoritmos

Para el desarrollo del algoritmo exacto se desarrollaron los siguientes métodos:

Algoritmo 4: CalculoDeLaGeometriaDefinidaPorSegmentos

```
Data: Lista de Segmentos por analizar
begin
    Limpieza();
    CargarSegmentos(Segmentos);
    CasosProblematicos(Segmentos);
    CalculoIntersecciones(Segmentos);
    RecorrerElementos(Vertices);
    GenerarEstructuras(Orientaciones, Fracturas, Vertices);
return: Cortes de rocas con sus agujeros y fracturas.;
```

Este es el mecanismo que se utiliza para generar desde los datos de los segmentos, las estructuras que se necesitan. Algoritmo 4 muestra el pseudocódigo del algoritmo para calcular la geometría generada por la intersección de segmentos dividiendo en seis pasos claves que se describen a continuación.

La limpieza se encarga de vaciar los datos de cálculos anteriores, es un paso importante para mantener la integridad entre distintas operaciones. En la limpieza se elimina todo salvo los segmentos iniciales, estos se almacenan para volver a ser interpretados.

En la carga de segmentos se vuelven a analizar los segmentos originales, aquí es donde se transforman los segmentos en vértices con conexiones.

Los casos problemáticos se analizan haciendo una revisión por los segmentos antes de calcular las intersecciones, este paso busca en particular si existe algún segmento que sea colinear, y si es posible fusionar los dos segmentos.

En el cálculo de intersecciones se intersectan los segmentos y se construyen nuevos vértices, este método es el que puebla las conexiones entre los vértices y otorga la complejidad a la geometría.

Recorrer los elementos, es una revisión que se hace por las conexiones creadas por las intersecciones. Este método es importante porque es el que marca los bucles o ciclos dentro de la estructura. Aquí ocurren dos eventos, primero se identifica qué vértices son fracturas en una primera instancia, y luego se empieza a recorrer los elementos en búsqueda de ciclos de vértices.

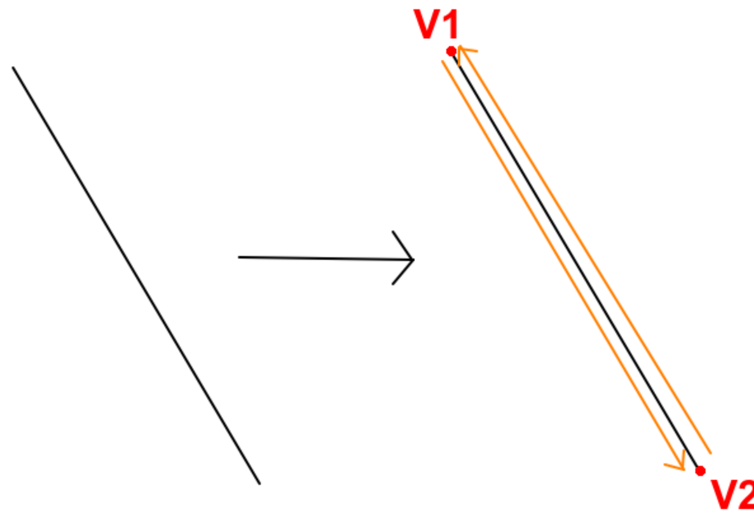
Generar estructuras se dedica a reunir la información recolectada y construir los cortes de roca. Aquí se identifican los bordes de los polígonos, se identifican sus agujeros y finalmente se definen a que polígono pertenece cada fractura.

4.2.1. Carga de segmentos

Este algoritmo es más importante, aquí ocurren dos cosas, primero se analizan los segmentos originales y luego para cada segmento se inicia la generación del grafo, los puntos que marcaban los extremos del segmento se transforman en vértices, y se conectan entre ellos como se observa en figura 4.1.

Los vértices generados necesitan ser únicos, para lo cual se implementó un patrón generador. Esta implementación revisa cada vez que se necesite generar un nuevo vértice, si existe algún otro vértice a una distancia menor que un delta definido, en caso de existir un vértice suficientemente cercano, se considera que no es necesario crear un nuevo nodo, sino que se utiliza el vértice encontrado. Esta revisión nos permite mantener un margen de seguridad para posibles problemas de precisión.

Figura 4.1: Carga de segmentos.



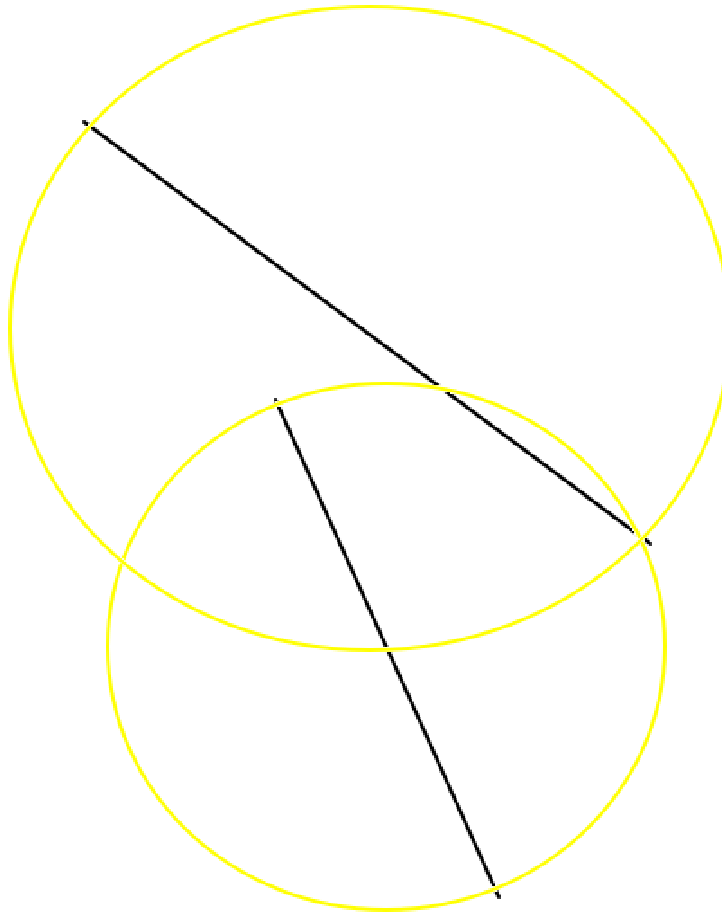
Para cada uno de los segmentos analizados se crean dos vértices conectados, estos vértices se agregan a la lista global de vértices.

4.2.2. Casos problemáticos

Cuando se revisan los casos problemáticos se busca eliminar algunas irregularidades. Dentro de los posibles conflictos se encuentra:

- Segmentos colineales, mediante una revisión de primero si los segmentos tienen colisión de sus círculos inscritos, el circulo se calcula utilizando el promedio de las coordenadas de los extremos del segmento, luego con la distancia a uno de los extremos se tiene un radio y un centro. Luego si lo anterior ocurre se revisa si ambos segmentos pertenecen a la misma recta mediante cálculo vectorial, si es así, se modifican los segmentos y se fusionan, luego sus vértices se modifican para sólo considerar los extremos.
- Vértices fuera del margen de la pantalla: en este caso se intersecta con el borde de la pantalla y se elimina el vértice exterior.

Figura 4.2: Revisión de colisiones.



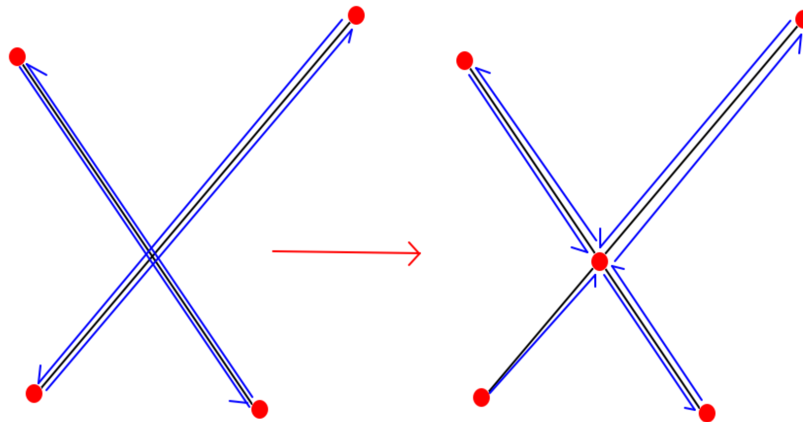
- Elementos muy pequeños: dependiendo si es un segmento, simplemente no se considera, si es un vértice, si están muy cerca se consideran el mismo vértice.

4.2.3. Calculo intersecciones

El cálculo de intersecciones es sumamente importante para el trabajo realizado: aquí es donde comienza la construcción de la geometría. El mecanismo que se utiliza es el descrito en 2.2.1, la intersección entre segmentos nos verifica si dos segmentos interactúan, y luego si es así nos entrega el punto de intersección. Ahora es importante mantener la integridad de la estructura de grafos, por lo cual si entre dos segmentos existe una intersección hay que indicarles a sus vecinos que existe un nuevo vértice con el que están conectados, además hay que eliminar las conexiones que fueran reemplazadas. Lo anterior se ilustra en la figura 4.3.

Como se logra observar en el ejemplo, consiste en encontrar los puntos en los cuales los distintos segmentos se cortan. Este primer problema puede ser sencillo de entender, pero computacionalmente puede llegar a ser complejo a grandes escalas, especialmente cuando el número de elementos que se comparan crece en demasía. Por lo cual se encontró que existen

Figura 4.3: Grafo de una intersección.



múltiples formas de abordarlo.

Hay que considerar que si una intersección genera un nuevo punto en un vértice ya existente puede generar un vértice con múltiples elementos conectados. Este comportamiento puede escalar tanto como segmentos intersectan en el mismo punto.

4.2.4. Recorrido

Este es el algoritmo más complejo del sistema ya que contiene varios casos interesantes: recorrer el grafo del sistema no es tan complicado en un caso sencillo, pero a medida que el problema escala empiezan a surgir ambigüedades y problemáticas no previsibles.

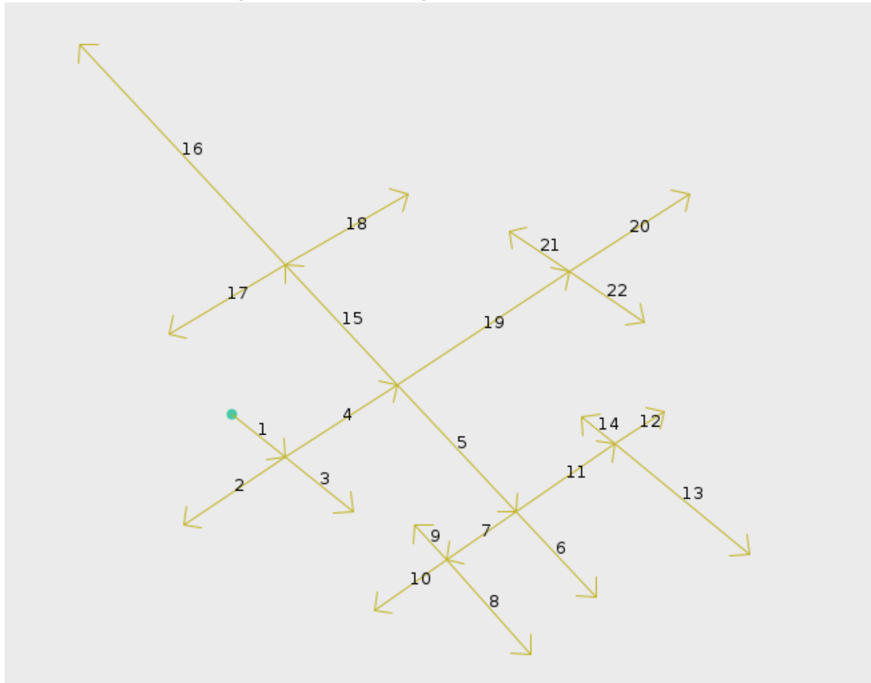
En primera instancia se necesita separar las fracturas de los bordes de los polígonos. Para esto lo primero es descartar las fracturas triviales. Para descartar estas fracturas basta marcar todos los vértices que solo conecten con un elemento, y hacer esto de forma recursiva. Aquí se observa la forma de una fractura, este en uno de los posibles ejemplos que se pueden encontrar, este caso es bien resulto por el algoritmo que marca los elementos con una conexión.

Algoritmo 5: Marcar fracturas

```
Data: Lista de Vértices por analizar
nVertices <- 5;
while Número de Vértices con una conexión nVertices >1 do
    nVertices <- 0;
    for Cada uno de los Vértices que no sean fractura do
        if vertice.nConexionesVerticesNoFractura() == 1 then
            vertice.esFractura();
            nVertices <- nVertices + 1;
```

En el código 5 se muestra cómo se marcan las fracturas triviales, este proceso se repite hasta que ya no se encuentren elementos con una conexión. Hay que destacar que para contar

Figura 4.4: Imagen de una fractura.



las conexiones entre vértices no se toman en cuenta los vértices considerados como fracturas, por lo cual al marcar un Vértice como fractura ya no cuenta a la siguiente iteración. Además, que solo se cuentan las conexiones con elementos que no sean fracturas, por lo cual en una iteración un elemento puede pasar de tener dos conexiones a una y convertirse en fractura. El ejemplo anterior ocurre en las fracturas ramificadas como en el ejemplo 4.4.

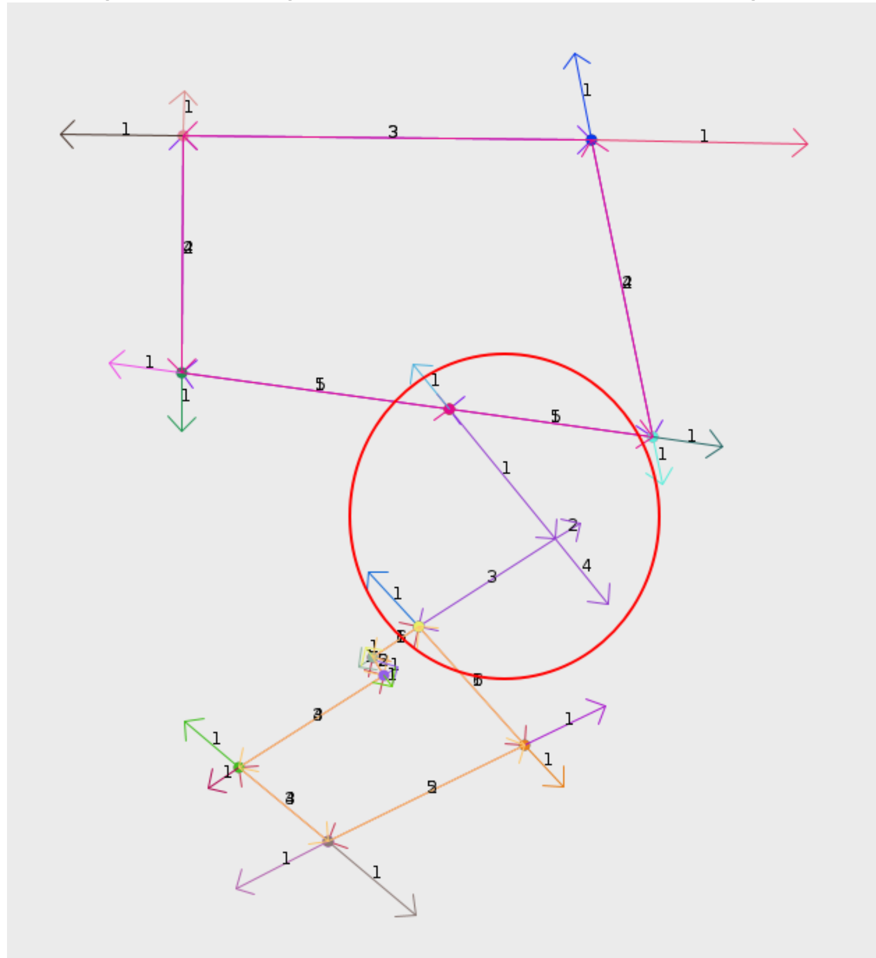
Aun así existen casos que no caen en esta categoría, los cuales son las fracturas que están entre dos polígonos, estas pueden estar entre medio de dos agujeros o dos polígonos individuales cualquiera. Este caso se debe reconocer e individualizar independientemente en un recorrido posterior.

Después de marcar los candidatos a fracturas se procede a recorrer los elementos restantes en búsqueda de ciclos, esto se realiza mediante una orientación. Se parte en un vértice y se decide una dirección aleatoria entre las opciones que no sean fracturas, luego se recorre el grafo siempre tomando la ruta más a la derecha posible. Al encontrarse algún vértice repetido se ha encontrado un bucle.

El método anterior permite encontrar los bordes interiores y exteriores de los polígonos, la única diferencia es la dirección en cual se recorre, pero eso permite utilizar esta información para separar los bordes de los agujeros. Al encontrar algún bucle es importante marcar el bucle como un elemento ya visitado y guardarlo como futura estructura. También es importante revisar el camino tomado para llegar a un bucle, dado que el camino puede contener fracturas previamente no detectadas. Para revisar las fracturas no detectadas hay que:

- Identificar el bucle, y marcar los elementos como pertenecientes a una orientación.
- Identificar los elementos visitados antes de encontrar el bucle.
- Para los elementos anteriores, consultar si una vez eliminado el bucle de la lista de

Figura 4.5: Imagen de una fractura entre dos polígonos.



conexiones validas, los vértices visitados en el recorrido, siguen teniendo más de una conexión.

- , Todos los elementos que no cumplen la condición son marcados como fracturas.

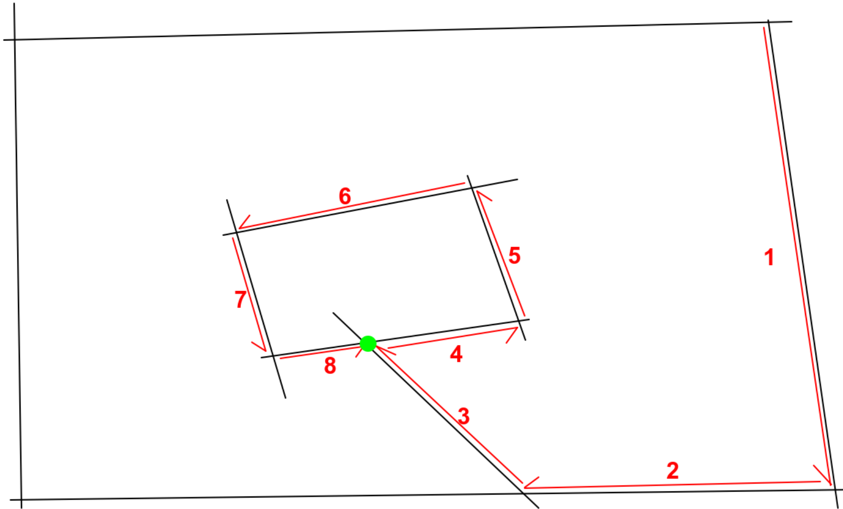
En la siguiente figura se muestra cómo se encuentra un bucle interior a un polígono. En este ejemplo se debe guardar las orientaciones 4,5, 6, 7 y 8 como una estructura, y se debe revisar las conexiones 3, 2 y 1 si son una fractura o no y marcar como fractura las conexiones 1, 2, 3 si corresponde (con la condición si tienen a lo más una conexión).

La búsqueda de orientaciones finaliza una vez se acabaron todos los candidatos a borde o a fractura. Como resultado se obtienen los bordes interiores y exteriores de los polígonos, esto dado una propiedad de la direccionalidad del recorrido.

4.2.5. Identificación de polígonos, agujeros y fracturas

La identificación de estructuras(polígonos, agujeros y fracturas) es la etapa donde se define la salida del análisis, la cual no es tan compleja como el algoritmo anterior pero tiene un par de detalles. Aquí se busca encapsular los elementos de los cortes de roca he individualizarlos.

Figura 4.6: Imagen de como se encuentra un bucle.



Dentro de la lógica del problema se puede notar que las rocas(polígono) pueden contener agujeros, pero hay que diferenciar claramente si los agujeros pertenecen a la roca(polígono) que se esté modelando, y o si pertenece a un agujero de otro modelo, donde puede ocurrir que se pueden anidar múltiples agujeros. Por lo cual para una roca que tiene un agujeró, podemos modelar su agujero como otra roca interior, este a su vez puede tener otras estructuras internas las cuales hay que distinguir de las estructuras que contiene el primer corte. Este algoritmo busca, para cada polígono identificar su borde, sus fracturas y los agujeros que contiene.

Para realizar lo anteriormente descrito se necesita la operación que permita identificar si un elemento está al interior de un polígono, aquí se utiliza el método implementado en la clase Polígono, he inspirado en lo visto en 2.2.2 , con esto se puede clarificar si primero, un polígono está dentro de otro, si una fractura está dentro de un polígono, y más precisamente si un elemento está dentro de otro, esto ayuda a diferenciar elementos con doble profundidad. Con lo anterior me refiero que una fractura o agujero de un agujero no es parte del polígono exterior.

Para realizar lo explicado se utiliza el método de trazado de rayo para determinar si un punto está al interior del polígono [4], este es un ejemplo clásico de geometría computacional el cual consiste en dibujar un rayo desde el interior de la estructura hasta fuera del contorno de la pantalla(bounding box), luego se intersecta con el polígono y se buscan cuantas veces se intersecta con un borde, si es impar esta adentro, si es par esta fuera.

Hay que destacar que las fracturas son únicas de un polígono, entonces no existen múltiples polígonos con la misma fractura, esto permite que se pueda optimizar el número de consultas, donde por cada fractura candidato, si se encuentra a donde pertenece se puede quitar de la lista de candidatos. Por último, hay que resaltar que primero se requiere encontrar los bordes y los agujeros de los polígonos, esto es necesario para identificar las fracturas internas.

El proceso de construcción de la salida se implementa en tres etapas, la primera etapa corresponde a la construcción del objeto Polígono. Para construir estos elementos se utilizan las orientaciones con giro anti-horario se asignan cómo borde de un polígono. Por otra parte

las orientaciones encontradas con giro horario se utilizan como candidatos a agujeros. Con estos elementos se soluciona la tarea de identificar cada uno de los segmentos.

Algoritmo 6: Generar estructuras

```
Data: Lista de Orientaciones generada por la búsqueda de bucles
for Cada uno de los conjuntos de orientaciones do
    if Orientación es anti-horario then
        Poligonos.add(new Poligono(orentaciones));
    else Agujeros.add(new Poligono(orentaciones));
for Cada uno de los Poligonos do
    Poligono.obtenerFracturasYAgujeros(Agujeros, Fracturas);
```

Como se muestra en el pseudo-código, cuando se identifican cada uno de los bordes como bordes de polígono o agujero, se puede identificar las fracturas que corresponden a los cortes de roca. Para el proceso de identificar a dónde pertenece cada fractura se debe consultar si la fractura está dentro del polígono, además hay que verificar que no esté dentro de un agujero del polígono, si no pasa ninguno de estos dos casos, entonces la fractura se agrega al polígono y se elimina de la lista de candidatos.

4.2.6. Desde imagen

El proceso de recuperación de estructuras desde las imágenes es más simple que el mecanismo exacto planteado. Hasta el momento no se ha podido encontrar la forma de no calcular las intersecciones entre los segmentos, aun así el mecanismo utilizado es el siguiente:

Se necesita una matriz que representa las regiones precalculadas con resolución de pantalla, y el conjunto total de segmentos. Una vez conseguido los segmentos, al igual que el método exacto se calculan las intersecciones y se consiguen nuevos segmentos.

El paso anterior se debe a que se utiliza un software externo que requiere que ningún segmento intersecte con otro para su construcción. Aquí se debe generar un documento en formato .poly que incluya, todos los puntos que se analizaron, todas las uniones entre estos puntos, y agregar al final del archivo un indicador de donde están ubicadas las regiones. Para este último punto se necesita recorrer la matriz de regiones que se tomó como entrada, aquí hay que notar que al ser una solución aproximada las regiones pueden no estar bien representadas en el borde de los píxeles, por lo cual hay que hacer un preproceso. Aquí hay que hacer un barrido por los píxeles en búsqueda de valores que estén rodeados por su misma región, al encontrar estos puntos uno se puede asegurar que no debería haber problemas al identificar las regiones.

Este algoritmo almacena como output los puntos de los segmentos, los puntos de las intersecciones, los segmentos (fracturas o lados de polígono) y cada región representada por un punto interior y su identificador. Usando detri2qt[10], leemos este input, generamos una triangulación, en donde cada triangulo tiene asociado la región a la que pertenece. A partir de esta información, reconstruimos los polígonos aproximados obtenidos desde la imagen usada como input.

Hasta el momento se ha logrado construir los polígonos desde Detriqt con el método aproximado, pero no se ha logrado confirmar la eficacia de este método.

4.3. Diseño e implementación de interfaz

En esta sección se describe la interfaz gráfica e interactiva desarrollada para ayudar a la búsqueda de errores de los algoritmos desarrollados. Esta aplicación está desarrollada usando el patrón de arquitectura MVC(Modelo-Vista-Controlador).

4.3.1. Modelo

El modelo contiene las estructuras geométricas del sistema, las cuales almacenan la lógica y como interactúan entre estas. Inicialmente se requirió implementar un modelo para los segmentos, los cuales son los elementos generados directamente desde el archivo. Pero que son útiles solamente para generar un modelo geométrico más interpretable y para guardar y cargar los archivos preliminares fuera de análisis, archivos formato csv.

Dentro de la geometría implementada dentro del sistema se encuentra:

- *Vertex*: Modelo central de la geometría utilizada, Aquí los Vértices son modelados como grafos, donde cada *Vertex* tiene un punto y sus conexiones, aquí los vértices tienen las siguientes facultades. Saben agregar y eliminar conexiones con otros vértices, saben si si están conectados con algún vértice que le preguntes. Por último, saben viajar por las conexiones para facilitar la generación de estructuras.
- *Orientation*: Estructura que guarda el recorrido por los Vertex. Se utiliza cuando se buscan los ciclos en los grafos y marca el orden en que se han visitado los distintos vértices.
- *Polygon*: Almacena la estructura final después de recorrer las estructuras. Contiene el borde, los agujeros y las fracturas del polígono. Esta es una estructura recursiva por lo cual implementa un patrón *composite*, los polígonos, dentro de la estructura implementada en el sistema, necesitan identificar qué elementos están dentro y fuera de él , y necesitan considerar que tienen un borde, y lo que está al interior del borde, y fuera de los agujeros es parte del polígono, por lo tanto necesitan almacenar, el borde como se mencionó anteriormente, los polígonos interiores que son los agujeros, y las fracturas, que son los cortes dentro del polígono pero que no logran dividir el elemento completamente.

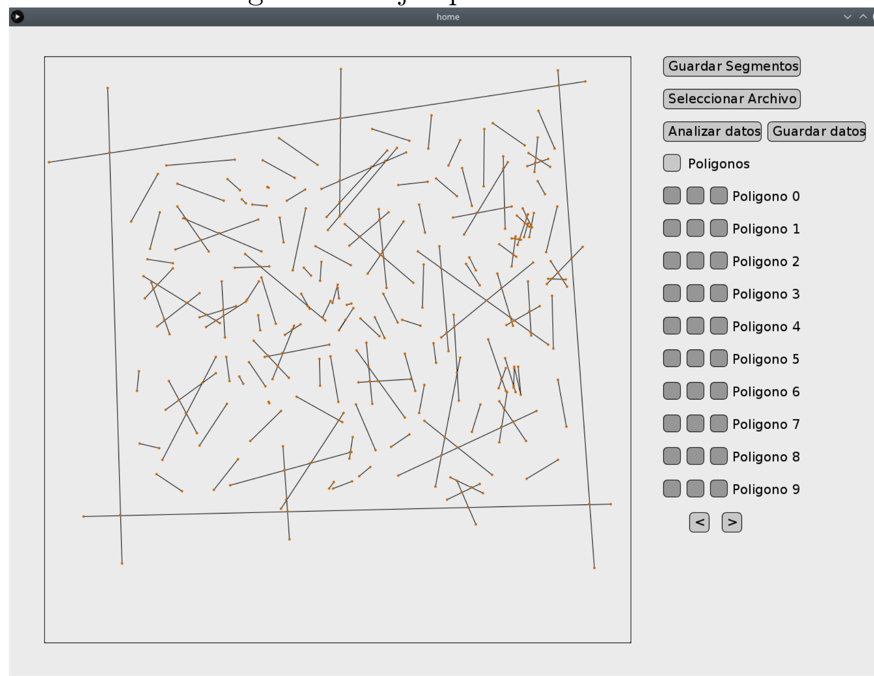
4.3.2. Vista

La interfaz de usuario es simple, pero entrega la funcionalidad requerida para generar las pruebas al sistema, esta consta de:

- Pantalla interactiva: esta tiene la función de mostrar los segmentos participantes del análisis, esta permite con el botón izquierdo generar nuevos segmentos, permite la opción de zoom con la rueda del ratón y finalmente moverse con el botón derecho del ratón.

- Botón de guardado de segmentos: este botón permite almacenar los elementos creados durante un experimento, esto permite guardar los casos que provocan problemas.
- Botón de Selección de archivos: este permite seleccionar el elemento por analizar, nos permite la carga del archivo con los segmentos.
- Analizar datos: es el botón que inicia la búsqueda de polígonos en la estructura.
- Guardar datos: permite guardar la información de las estructuras ya analizadas.

Figura 4.7: Ejemplo de la interfaz.



Finalmente existen los botones seleccionables para la visualización de los elementos, eso consiste en un botón principal que permite intercambiar la vista entre los segmentos y los polígonos y estructuras encontrados. Luego cuando se tienen los segmentos bien analizados, se pueden por cada polígono encontrado visualizar sus partes individualmente,

Aquí se implementaron los elementos pertenecientes al apartado gráfico, como se mencionó anteriormente existe una restricción con *processing* que no permite generar simplemente los llamados a elementos gráficos, y actualizaciones dado que se utilizan en el motor de refresco de la pantalla, por lo cual se tuvo que implementar parte de los llamados de los métodos al interior de la Vista, aun así, en general los métodos quedaron separados en el controlador.

En este caso la vista se encarga de mantener los objetos que están a la vista del usuario como botones y las listas, además de mostrar gráficamente los elementos analizados. Por lo tanto, en la vista se implementó todos los elementos gráficos y las interacciones que estos realizan, tales como:

- Generar nuevos segmentos en la pantalla con botón izquierdo del ratón.
- Alejarse y acercarse con la rueda del ratón.
- Desplazarse por la pantalla con el botón derecho del ratón.
- Guardar los segmentos generados en un archivo .csv.

- Cargar segmentos de un archivo .csv.
- Iniciar el cálculo de las estructuras a partir de los segmentos, (polígonos, fracturas, agujeros) de los elementos.
- Guardar las estructuras calculadas (polígonos, fracturas, agujeros) en formato .poly.
- Mostrar/Ocultar elementos de la pantalla.

4.3.3. Controlador

En este módulo se implementaron los algoritmos y las interacciones entre los llamados de las acciones de la vista y el modelo, se aprovechó de utilizar para desarrollar los distintos llamados entre las estructuras, y describir como interaccionan entre estas.

Los algoritmos implementados aquí son los métodos que interactúan con variables de ambiente, donde se utilizan llamados archivos externos, o se generan operaciones globales con los miembros del modelo. Por ejemplo, en el modelo están los segmentos que saben calculas su intersección, pero en el controlador está el llamado a quienes tienen que calcular la intersección con quien. En general aquí se implementan las actualizaciones masivas al modelo.

Los algoritmos implementados en el controlador son:

1. Leer el archivo de entrada, este es solamente un paso donde se transforma el input del archivo en los segmentos iniciales. Esto permite mantener un formato fácil de transcribir a un nuevo archivo si se desea guardar tanto como la salida o entrada, siempre se almacenan como segmentos.
2. Detección de casos problemáticos, se generó un método que recorre los segmentos en un preprocesamiento en búsqueda de casos que puedan generar errores en la geometría. Aquí se detectan los segmentos coolineales, que se intersecten, así se tomó la decisión de fusionar estos segmentos si estos cumplen las dos condiciones. El otro caso que se toma en cuenta es cuando un segmento es más pequeño que un delta, este mismo método se utiliza para los vértices, si dos vértices están más cerca que un cierto delta se concederá que son el mismo punto, esto permite ahorrar problemas de precisión, aunque no soslaya problemas en intersecciones con punto flotante lejanos a cero.
3. Búsqueda de intersecciones, la búsqueda de intersecciones es el método que más tiempo implica en el algoritmo, para esto como método inicial y el que se usa como comparación, intersecta todos los segmentos por fuerza bruta, eso sí se concederán solo los elementos que sean suficientemente cercanos puedan ser candidatos a intersectar, para esto se concederán solo segmentos que sus esferas al ser rotados no colacionen. Se utiliza este último método para optimizar la búsqueda de colisiones, esta optimización se utiliza en geometría computacional a la hora de generar *collition boxes*.
4. Marcar las fracturas, en este proceso se marcan las conexiones que no sean candidatas a pertenecer a un ciclo, esto se realiza de forma recursiva detectando los elementos que solo tienen una conexión con otro vértice y verificando los vecinos. Hay que tener en cuenta que no todas las fracturas pertenecen a esta categoría, porque pueden existir fracturas con más de una conexión cuando un polígono tiene una fractura que conecta con un agujero.

5. Búsqueda de Ciclos, en este proceso una vez marcados los candidatos a fracturas se recorren las orientaciones que se encontraron, para esto se viaja a través de las conexiones entre vértices siempre siguiendo la regla de la mano derecha, se toma en cuenta siempre el elemento que tomando en cuanta un elemento anterior indicar el elemento más hacia la derecha, cuando se hace esto se busca que las orientaciones cierren en un ciclo, si esto pasa entonces almaceno el recorrido que hice hasta encontrar el ciclo, marco los elementos del ciclo para que ya no sean candidatos a nuevos ciclos, después reviso el recorrido que hice hasta encontrar el ciclo en búsqueda de nuevas fracturas. Este es el centro de la solución implementada ya que permite diferenciar todas las estructuras necesarias.

Capítulo 5

Validación

5.1. Solución exacta

Para el proceso de validación se utilizó un modelo incremental probando los algoritmos primero sobre configuraciones de segmentos simples y en cantidad pequeña, hasta configuraciones complejas y una gran cantidad de segmentos (ilustras con las figuras ya insertadas cada caso). Con cada prueba, se fueron corrigiendo y superando fueron superando distintas dificultades y perfeccionando cada vez más el modelo a medida que se encontraban errores.

En primera instancia se desarrolló una interfaz gráfica para poder interpretar la geometría que se estuviera calculando. Luego se agregaron funcionalidades que permitieran crear y almacenar ejemplos para generar las pruebas. La idea es que cuando se encontrara algún error este no debería repetirse a futuro una vez solucionado.

El primer ejemplo fue construir un polígono simple sin elementos extraños tal cómo se muestra en la figura5.1.

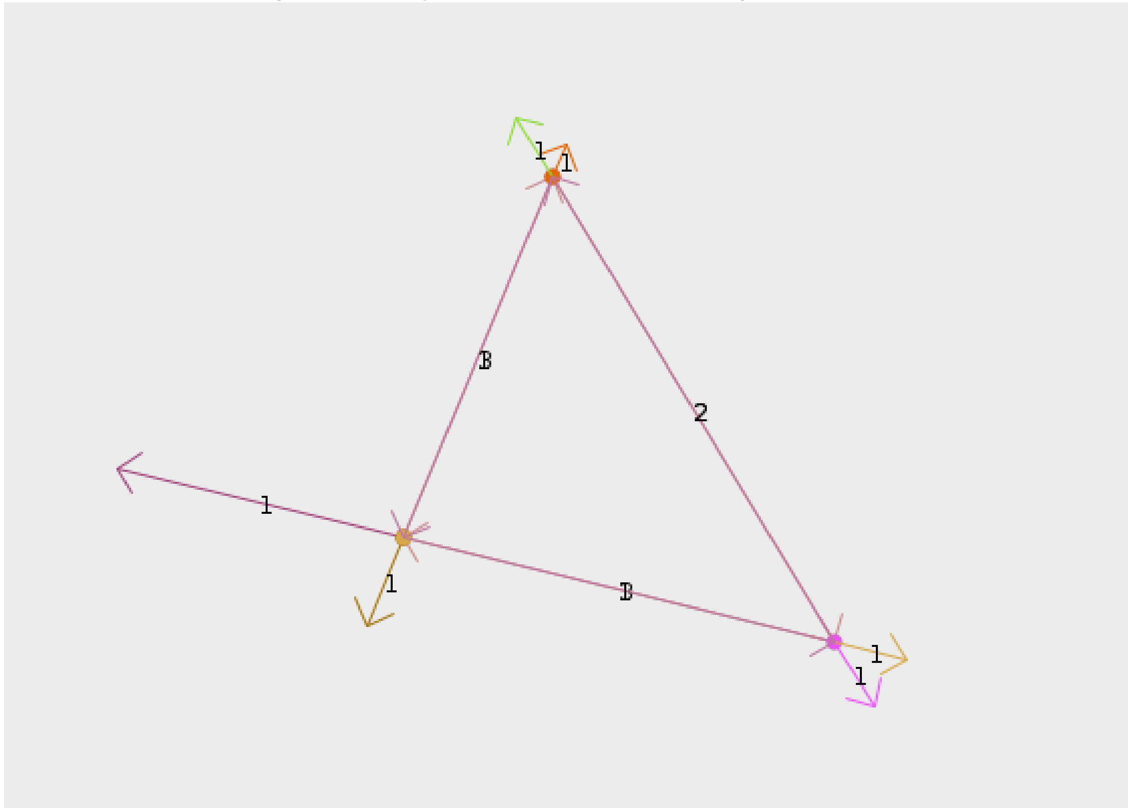
Este proceso fue incrementando paulatinamente a medida que se fueron encontrando errores, por ejemplo, el caso de fracturas no triviales que posiblemente no se puedan identificar rápidamente dado que conectan un agujero con el borde del polígono, cómo se muestra en la Figura 5.1

El resto de validaciones se incluyen en el capítulo de Apéndice donde se muestran más ejemplos de cómo fue progresando la validación de la geometría.

5.2. Comparación de algoritmo exacto y algoritmo aproximado

Finalmente, para probar la validez de la implementación se realizaron pruebas con modelos de conjuntos de segmentos que representan cortes y fracturas de roca reales tal cómo se muestra en la Figura 5.3. La cual contiene una gran cantidad de elementos. Para comprobar que el resultado final fuera correcto se realizó un análisis de regiones obtenidas por el algoritmo que

Figura 5.1: Ejemplo del análisis polígono símple.



toma como input las imágenes generadas por el algoritmo aproximando y desde el algoritmo exacto que toma como input los segmentos. Al comparar las regiones encontradas por ambos algoritmos se obtuvo que hay una pequeña diferencia en el número de regiones detectadas. Esto es esperable dada la cantidad de elementos pequeños que la solución aproximada no toma en cuenta.

La aplicación como muestra la figura 5.3, permite diferenciar correctamente las estructuras necesarias a partir de solamente los segmentos. Está estructurada para generar las pruebas de forma satisfactoria, y permite cargar, guardar, análisis y extraer propiedades de las estructuras como corresponde. Además, permite guardar esta estructura en el formato reconocible por el sistema de triangulaciones. Con esto cumple con el objetivo centrar de encontrar las estructuras básicas a partir de segmentos.

Con estos resultados se conversó con el usuario final y se validaron las alternativas para extender el análisis. También se analizaron otros elementos que se necesitan estudiar y quedan fuera del alcance de la memoria como la extensión a 3D.

Figura 5.2: Ejemplo del análisis polígono con dos agujeros.

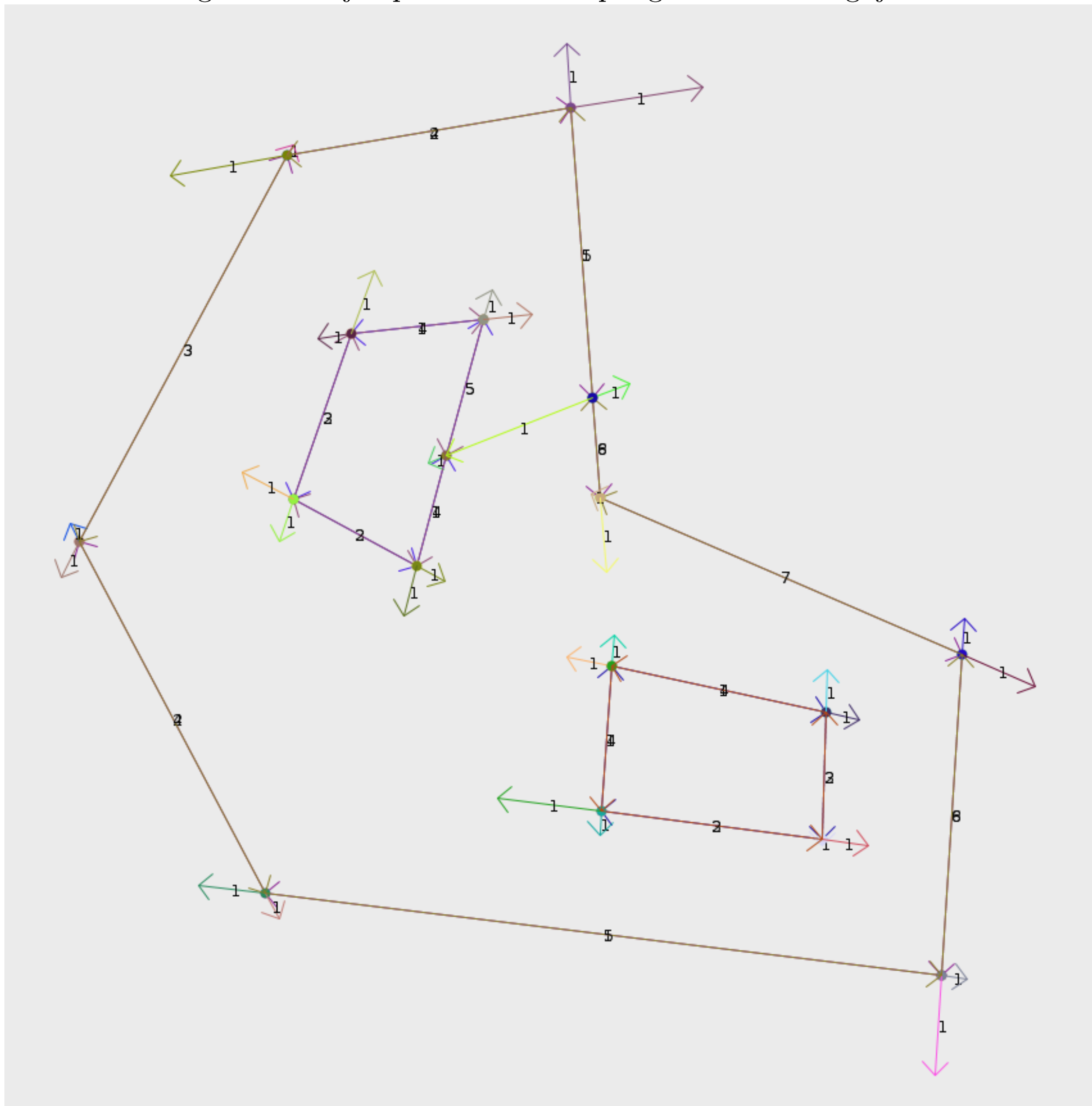
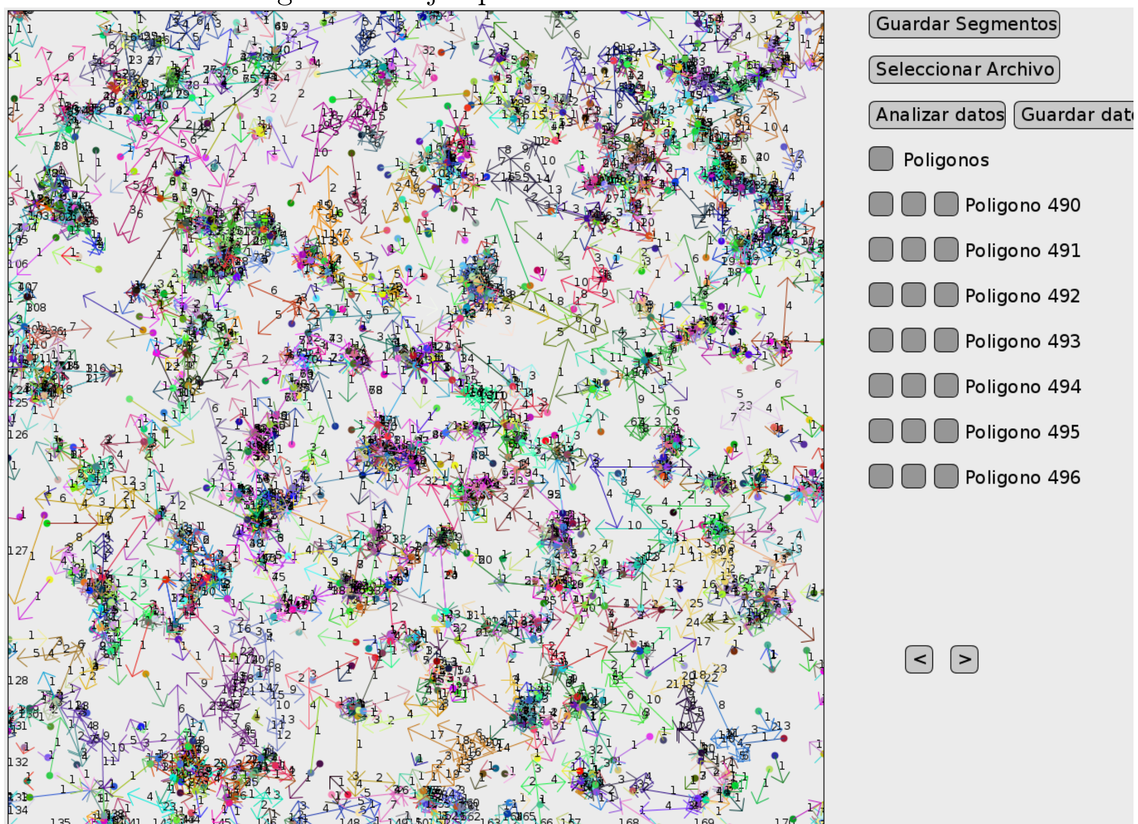


Figura 5.3: Ejemplo del análisis de datos reales.



Capítulo 6

Conclusiones

En esta memoria se implementó un algoritmo exacto para construir/detectar regiones (polígonos no simples), agujeros y fracturas a partir de un conjunto arbitrario de segmentos y otro algoritmo para construir/detectar el mismo conjunto de elementos a partir de una imagen que contiene las regiones aproximadas. Así es como se logró el objetivo de modelar cortes de roca inspirados en averiguar diversas formas de optimizar el análisis minero. Cómo resultado se dispone de una aplicación que permite definir y modelar las estructuras.

Se exploraron las distintas alternativas de cómo construir estructuras, y se generó un mecanismo para poder generar futuras comparaciones con modelos aproximados.

Entre lo que se obtuvo como resultado es la construcción de la interfaz extensible, la implementación de los algoritmos requeridos como elementos para probar soluciones nuevas, generar archivos de salida y lograr los análisis junto a generar los archivos esperados de salida. Por otra parte, se implementó un sistema que utiliza la aproximación por imagen para la obtención de las estructuras esperadas.

Como trabajo futuro queda desarrollar el análisis de las propiedades de los cuerpos de roca tales como: el cálculo de la envolvente convexa de los bordes de las rocas. Además, pruebas más exhaustivas de la correctitud de los algoritmos exactos y aproximados. Por ejemplo, comparar el número, área y forma de los elementos extraídos, entre otras características.

Una herramienta como la desarrollada en esta memoria puede tener un gran impacto en la minería, pues si un algoritmo aproximado es más rápido y eficiente que un algoritmo exacto y se logra resultados suficientemente similares de podría acelerar los procesos de alerta y monitorio en minas reales lo cual es un incentivo fuerte y de alto impacto.

Por otra parte, para la computación gráfica, los procesos aproximados pueden ser sumamente útiles a la hora de evaluar información que ya de por si es no muy precisa, con esto se busca dar nuevas herramientas para generar futuras aproximaciones esperando que se pueda proyectar a más dimensiones.

Durante el transcurso de la memoria se aprendió que el conocimiento registrado durante los años de enseñanza converge cuando uno encara proyectos reales, así como la planificación

y estimación de los tiempos no corresponde a la realidad, uno tiene las expectativas más altas de lo que puede lograr en un plazo de tiempo.

Bibliografía

[1] Cheong O. Van Kreveld M. Overmars M. Berg, M. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Alemanía, 2008.

[2] Marcel Berger. Geometry 1. *Springer Science & Business Media*, 2009.

[3] RONALD Goldman. Graphics gems. *Graphics gems*, pages 304–305, 1990.

[4] Eric Haines. I.4. - point in polygon strategies. In Paul S. Heckbert, editor, *Graphics Gems*, pages 24 – 46. Academic Press, 1994.

[5] MS Milgram. Does a point lie inside a polygon? *Journal of Computational Physics*, 84(1):134–144, 1989.

[6] Stefan Schirra. Robustness and precision issues in geometric computation. 1998.

[7] Sophie Schneider and Ivo F Sbalzarini. Finding faces in a planar embedding of a graph [extended abstract]. 2015.

[8] J. R. Shewchuk. Triangle: A two-dimensional quality mesh generator and delaunay.

[9] Moshe Shimrat. Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434–435, 1962.

[10] Hang Si. detri2.

[11] Joaquín Torres. Geometric packing for rock accumulation analysis. Master's thesis, Universidad de Chile, 9 2020.

Apéndices

Apéndice A

Apéndices

A.1. Código implementado

Figura A.1: Implementacion Algoritmo interseccion.

```
void interesec(Segment seg){

  double x1 = this.p1.getP().x;
  double x2 = this.p2.getP().x;
  double x3 = seg.p1.getP().x;
  double x4 = seg.p2.getP().x;

  double y1 = this.p1.getP().y;
  double y2 = this.p2.getP().y;
  double y3 = seg.p1.getP().y;
  double y4 = seg.p2.getP().y;

  double c = (x1 - x2)*(y3 - y4) - (y1 - y2)*(x3 - x4);

  if (Math.abs(c) < 0.0000001){
    return;
  }

  double a = x1*y2 - y1*x2;
  double b = x3*y4 - y3*x4;

  float px = (float)((a*(x3 - x4) - b * (x1 - x2))/c);
  float py = (float)((a*(y3 - y4) - b * (y1 - y2))/c);

  if ( px < this.sx
    || px > this.ex
    || py < this.sy
    || py > this.ey
    || px < seg.sx
    || px > seg.ex
    || py < seg.sy
    || py > seg.ey
    ){
    return;
    }

  PVector v = new PVector(px, py);


  VertexGenerator vg = new VertexGenerator();

  Vertex ver = vg.getVertex(v);

  this.addInterssection(ver);
  seg.addInterssection(ver);

}
```

A.2. Imágenes Validación

Figura A.2: Ejemplo del análisis polígono con dos agujeros.

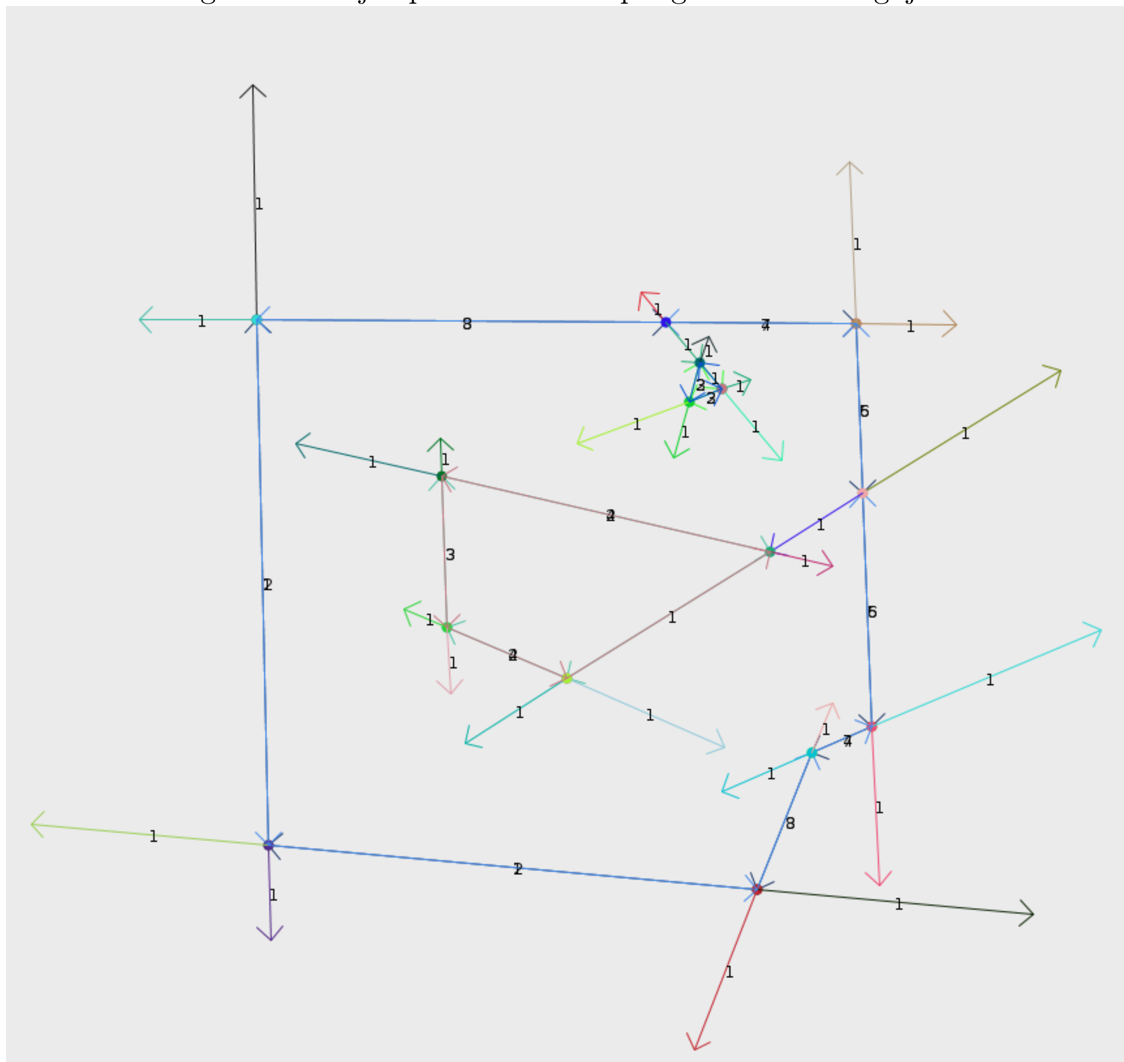


Figura A.3: Ejemplo del análisis polígono fracturas internas (cóncavo).

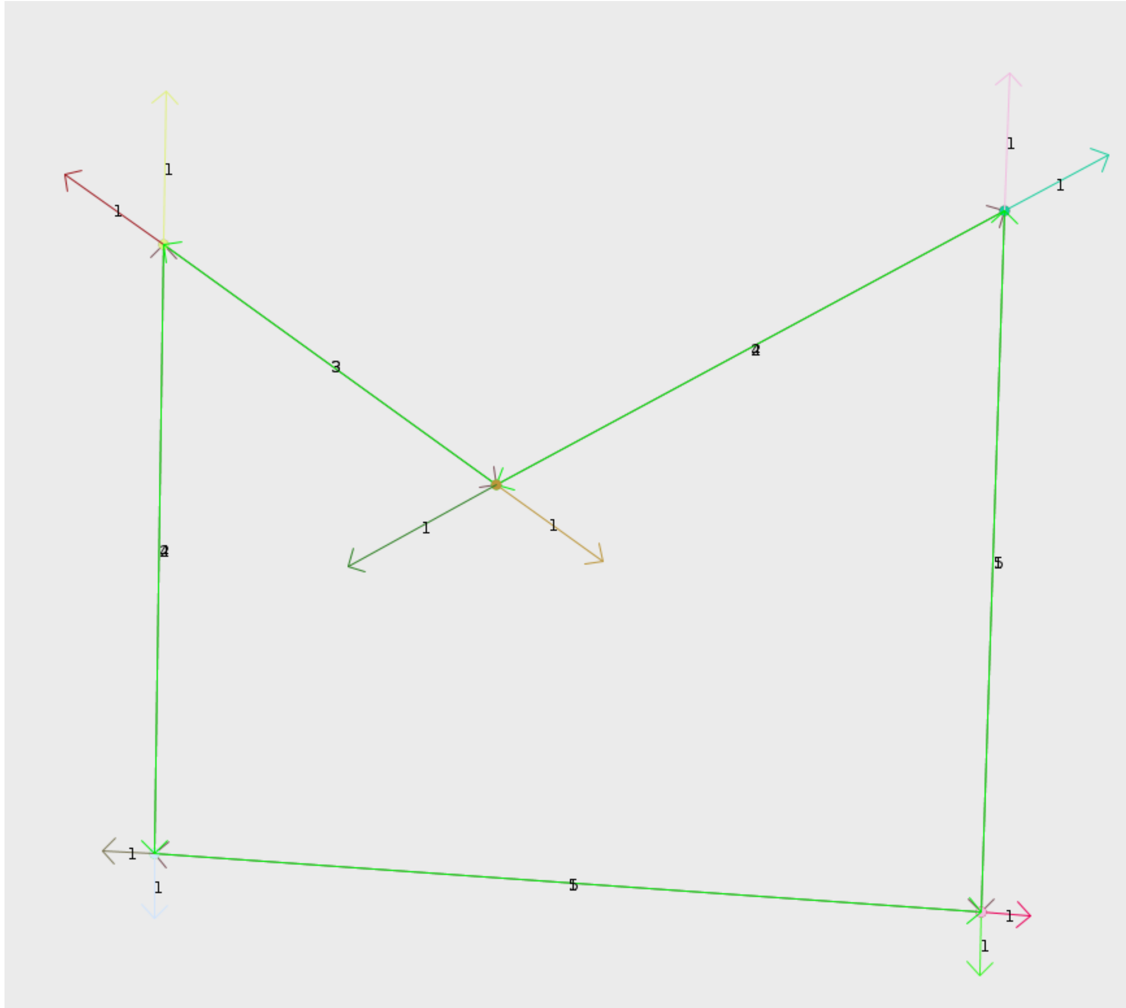


Figura A.4: Múltiples fracturas sin polígono.

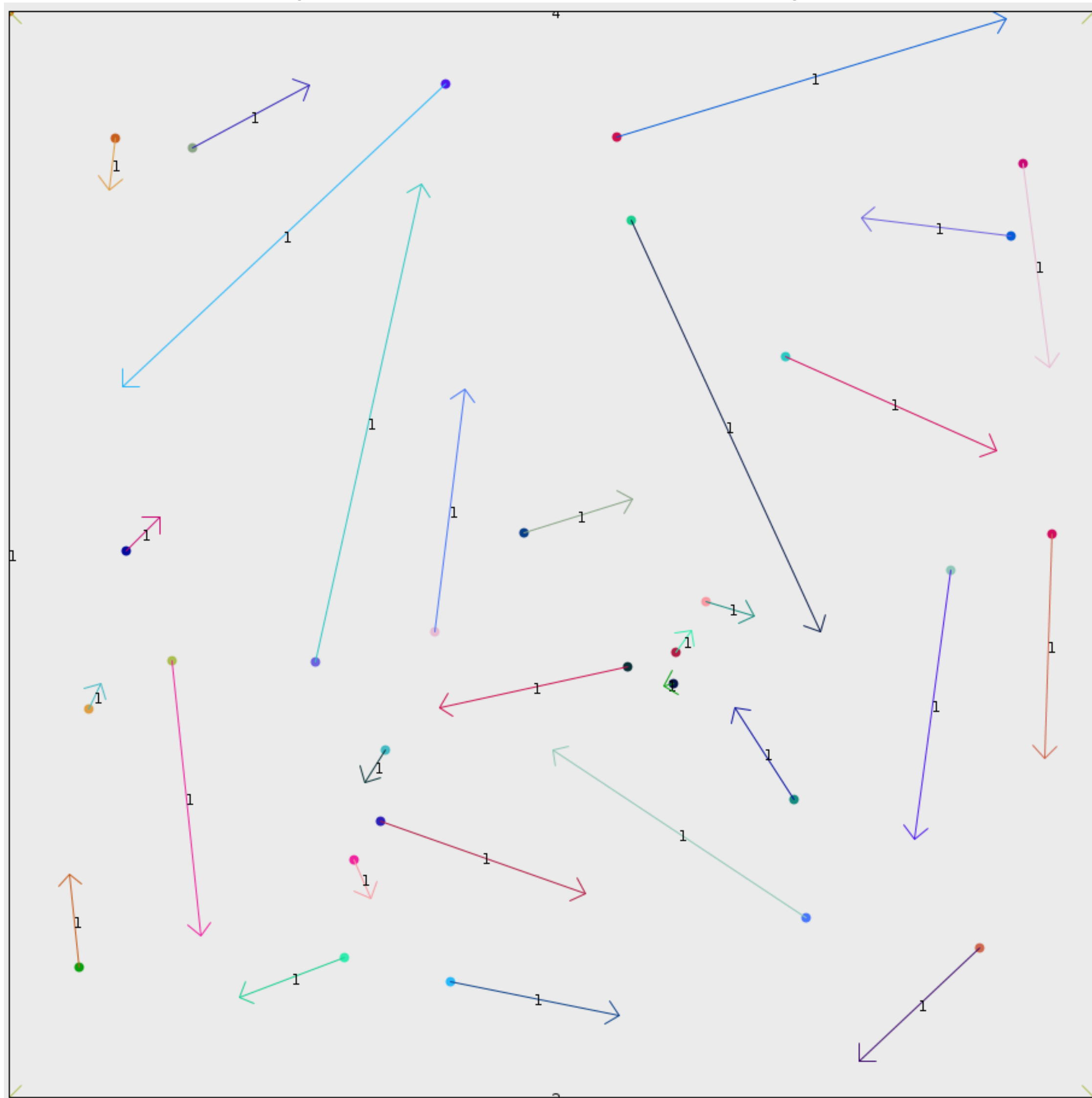


Figura A.5: Ejemplo del análisis polígono múltiples fracturas y agujeros.

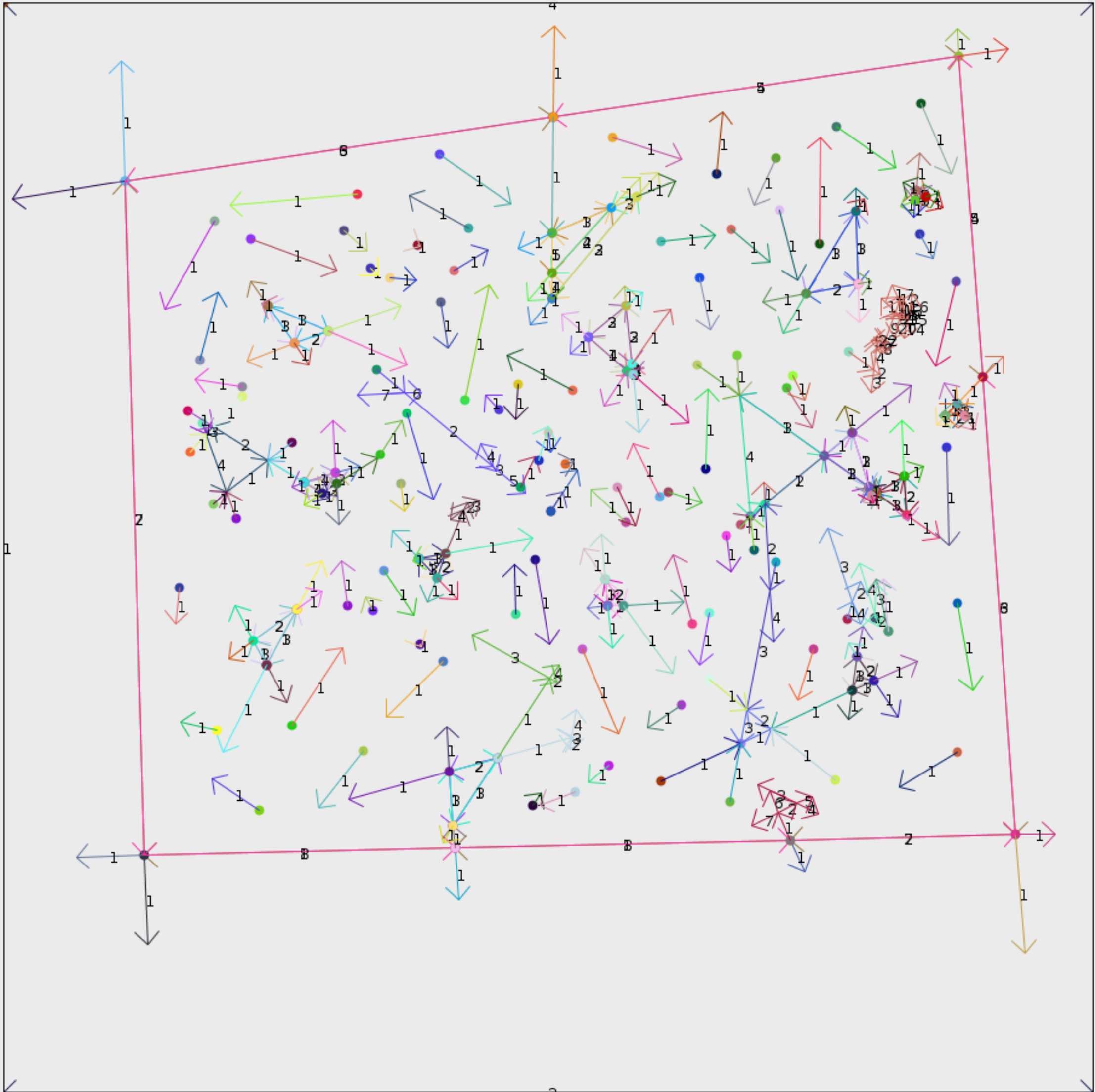


Figura A.6: Ejemplo del análisis polígono con fracturas muy ramificadas.