# Universidad de las Ciencias Informáticas

“Facultad 6”

**“Desarrollo de un componente para modelación y visualización en tres dimensiones de objetos geológicos utilizando el *framework* VTK.”**

**Trabajo de Diploma para optar por el título de**
**Ingeniero Informático**

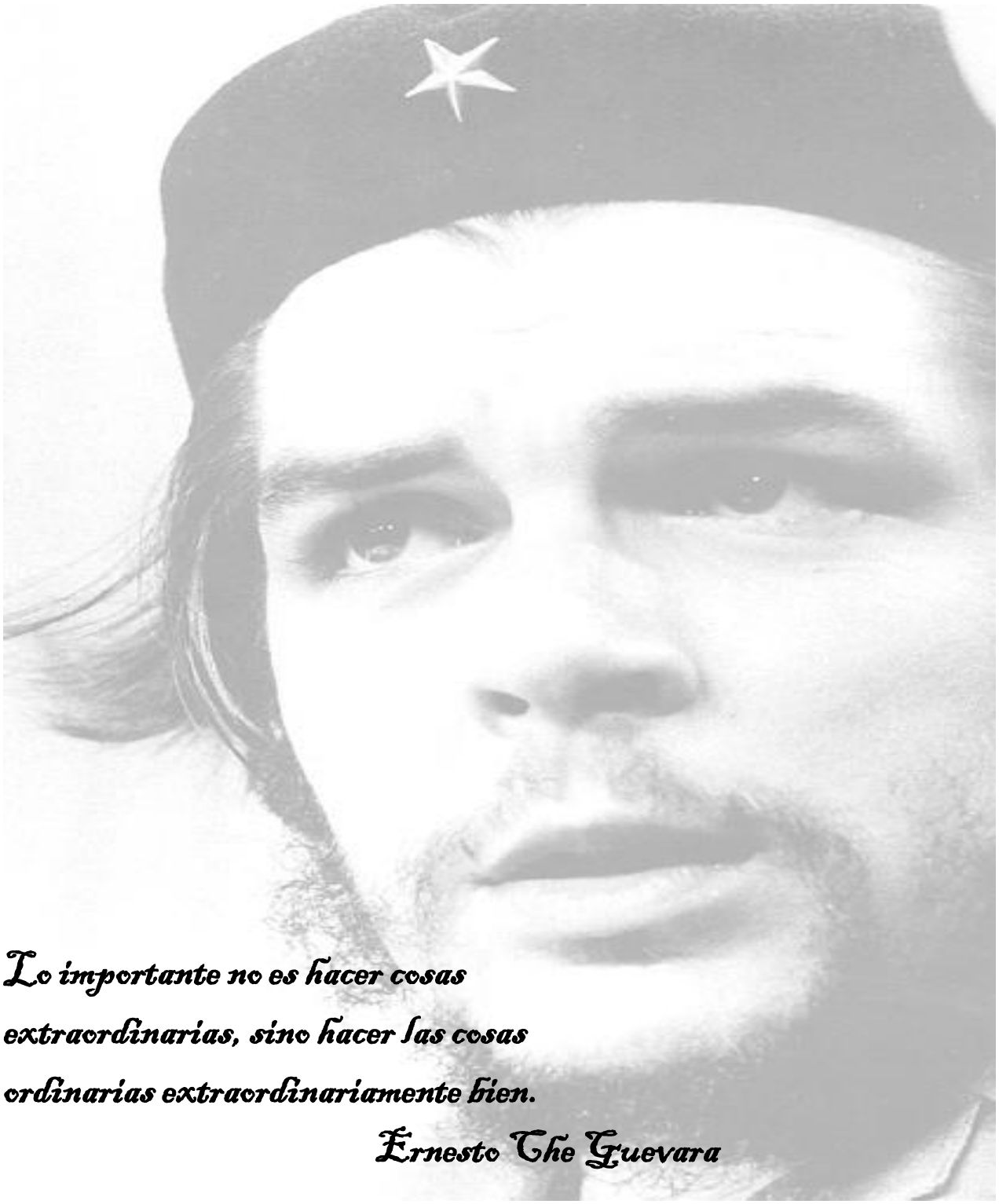
**Autor:** Alain López Jiménez

**Tutor:** Rocny Morales Delgado

**Co-tutor:** Eddy Dangel Quezada Rodríguez

**La Habana, Junio 2011**

**“Año 53 de la Revolución”**



*Lo importante no es hacer cosas extraordinarias, sino hacer las cosas ordinarias extraordinariamente bien.*

*Ernesto Che Guevara*

Declaro que soy el único autor de este trabajo: **“Desarrollo de un componente para modelación y visualización en tres dimensiones de objetos geológicos utilizando el framework VTK”** y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

**Alain López Jiménez**

_____ _____

**Ing. Rocny Morales Delgado** **Ing. Eddy Dangel Quezada Rodríguez**

**Autor:** Alain López Jiménez
Universidad de las Ciencias Informáticas
La Habana, Cuba
E-mail: aljimenez@estudiantes.uci.cu

**Tutor:** Rocny Morales Delgado
Universidad de las Ciencias Informáticas
La Habana, Cuba
E-mail: rmdelgado@.uci.cu

**Co-tutor:** Alain López Jiménez
Universidad de las Ciencias Informáticas
La Habana, Cuba
E-mail: edquezada@.uci.cu

*Para algunos comienza con satisfacción, para otros con obligación, pero al final del camino, ese instante en que cinco duros años llenos de sacrificio y dedicación, se convierte en un sueño hecho realidad. Es el momento de sentirse orgulloso por haber culminado el recorrido que tan lejos parecía, pero en un abrir y cerrar de ojos se asomaba para marcar el comienzo de una nueva vida.*

***Agradezco a mi madre Paula:*** *por saber cultivar en mí la educación y el conocimiento que condujeron a mi formación como profesional.*

***A mi abuela Anilda:*** *por quererme cada día de su vida sin importar cuán difícil se tornara su situación.*

***A mi novia Cecilia:*** *por permitirme conocer el amor verdadero, que renace con la unión de dos almas gemelas. Por estar a mi lado cada vez que la necesité, guiando mis pasos con mucha seguridad y la satisfacción de saberse importante en mi vida. Por confiar en mí sin importar la grandeza del obstáculo. Por fortalecer mi ser en cada momento difícil que afrontamos. Pero sobre todo, por ser la mujer que quiero junto a mí para el resto de la vida.*

***A mis tíos Margarita, Rafael y Jorge:*** *por su entrega para con la familia. Por ser hombres y mujeres, padres y madres, amigos y amigas siempre que lo necesité.*

***A mis primos Jaymí e Ives:*** *por llenar el vacío que queda cuando la vida te depara un destino sin hermanos.*

***A mi familia:*** *por su preocupación ante los exámenes y la confianza que me trasmitieron durante este período. En especial a mi abuela* **Consuelo**, *que se desvive con sus hijos y nietos sin importar el peso del sacrificio consumado.*

***A mis amistades:*** *por su incondicionalidad hacia mí. En especial a los que han estado al tanto de mis dificultades, corrigiendo mis errores y aconsejándome de buena fe.*

***A mi tutor y co-tutor:*** *por estar dispuestos cada vez que los abarroté de dudas. Por ser ejemplos y guiarme en esta difícil tarea. Por cuidar de mi prestigio como estudiante y enfrentar cualquier opinión errada. Por ser mis amigos en cualquier plano y por confiar en mí como protagonista de esta labor.*

***A la Revolución Cubana:*** *por hacer realidad la posibilidad de estudiar en una escuela de excelencia con profesionales comprometidos con su deber.*

***Al comandante Fidel Castro Ruz:*** *por anteponer las necesidades de su pueblo a las personales y arriesgar su vida por el bien de los cubanos.*

***Al comandante Ernesto The Guevara:*** *por ser ejemplo en mi desempeño como cubano y estudiante. Por constituir el héroe y la inspiración de mis ideales revolucionarios.*

*Dedico este resultado a las personas que jugaron un papel fundamental en mi carrera e inspiraron mi capacidad de continuar en la búsqueda por un futuro mejor.*

***A mi madre:*** *por ser la mujer más linda del mundo. Por darme la gracia de vivir con humildad. Por apoyarme incondicionalmente en todas mis etapas sin importar la distancia. Por llenar el vacío que queda cuando se pierde a ese ser masculino que defiende la familia, el padre. Pero sobre todo, por arriesgar su vida para que yo pudiera tener la mía. Te amo.*

***A mi abuela:*** *por no descansar ni un minuto en hacerme un hombre de bien. Por estar presente en mi vida y permitirme ser la razón de la suya. Por forjar un hombre orgulloso de la mujer. Por ser fuerte y enfrentar a la muerte hasta el último instante. Te amo.*

***A mi novia Cecilia:*** *por aportar ese cariño femenino que necesita un hombre para enfrentar sus batallas. Por constituir la mujer perfecta en un mundo de imperfecciones. Por cuidar mis sueños y responder a mis anhelos. Por dedicar su tiempo a conquistar al padre de sus hijos. Te amo.*

RESUMEN

Los sistemas para la planificación y explotación minera, permiten entre sus aplicaciones: la estimación de reservas, modelamiento de relieves, levantamiento topográfico y la conversión de los complejos datos recogidos en la fase de exploración minera en información visual. Este proceso facilita la comprensión del entorno que se estudia a partir de imágenes interactivas y dinámicas en tres dimensiones (3D) de los objetos geológicos y sus propiedades. En la actualidad, el país no cuenta con un software de este tipo que cubra todas las necesidades, sólo existen algunas soluciones que han sido personalizadas para satisfacer objetivos específicos, y no cuentan con la posibilidad de visualización de los datos. El objetivo del presente trabajo de diploma es desarrollar un componente que permita modelar y visualizar objetos geológicos en tres dimensiones usando el *framework* VTK, para apoyar el proceso de desarrollo del primer sistema cubano de planificación y exploración minera mediante la reutilización de sus funcionalidades.

PALABRAS CLAVES

Componente, diseño, exploración minera, modelado, patrones, planificación, propiedades visuales, transformaciones visuales, visualización.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN ..... 1
1 CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA ..... 5
1.1 Introducción ..... 5
1.2 Análisis de los principales software mineros en Cuba ..... 5
1.3 Conceptos y características del modelamiento 3D ..... 8
1.3.1 Modelado tridimensional ..... 8
1.3.2 Fundamentos de la escena ..... 11
1.3.3 Visualización de objetos geológicos en 3D ..... 17
1.4 Conclusiones ..... 19
2 CAPÍTULO 2: TENDENCIA Y TECNOLOGÍAS ACTUALES ..... 20
2.1 Introducción ..... 20
2.2 Acerca de la Arquitectura de Software (AS) ..... 20
2.3 Metodologías de software ..... 21
2.4 Tecnologías y herramientas ..... 25
2.4.2 *Frameworks* de desarrollo ..... 27
2.5 Conclusiones ..... 34
3 CAPÍTULO 3: PLANIFICACIÓN Y DISEÑO DEL SISTEMA ..... 35
3.1 Introducción ..... 35
3.2 Historias de Usuarios ..... 35
3.3 Estimación de esfuerzo por historias de usuarios ..... 36
3.4 Plan de iteraciones ..... 37
3.5 Plan de duración de las iteraciones ..... 39
3.6 Arquitectura del sistema ..... 40
3.6.1 Patrones de diseño ..... 41
3.7 Clases-Responsabilidades-Colaboración ..... 43
3.8 Conclusiones ..... 44
4 CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA ..... 45
4.1 Introducción ..... 45
4.2 Estándar de codificación ..... 45

4.2.1 Comentarios....45
4.2.2 Nombres de identificadores....48
4.2.3 Sangrías....49
4.2.4 Líneas y espacios en blanco....49
4.3 Pruebas....50
4.3.1 Desarrollo dirigido por pruebas....50
4.3.2 Validación mediante la encuesta....54
4.4 Conclusiones....55
5 CONCLUSIONES....56
6 RECOMENDACIONES....57
7 REFERENCIAS BIBLIOGRÁFICAS....58
8 BIBLIOGRAFÍA....61
9 GLOSARIO....62

ÍNDICE DE FIGURAS

Figura 1: Secuencia de modelado....8
Figura 2: Secuencia de modelado....8
Figura 3: Traslación de un triángulo y un punto....10
Figura 4: Rotación de un cubo en el espacio....11
Figura 5: Escalado de un cubo en los distintos ejes de coordenadas....11
Figura 6: Representación de un sistema de coordenadas tridimensionales....12
Figura 7: Sistema de coordenadas de diferentes vistas....12
Figura 8: Sistemas de coordenadas locales....13
Figura 9: Proyección en perspectiva....15
Figura 10: Proyección en paralelo....16

ÍNDICE DE TABLAS

Tabla 1: Estimación de la duración de las Historias de Usuarios .....36
Tabla 2: Plan de duración de las iteraciones.....40
Tabla 3: Resultados de la encuesta a los miembros del proyecto Minería .....55

INTRODUCCIÓN

Desde que el hombre comenzó a practicar la minería como una de las actividades fundamentales para su desarrollo, se apoyó en el empleo de diferentes técnicas como la recopilación de información, teledetección, geología, geoquímica, sondeos mecánicos y geofísica. Estas han ido evolucionando a lo largo de las diferentes etapas por las que ha transcurrido la extracción de minerales que comenzó en la Era de Piedra y se mantiene en la actualidad. El surgimiento y evolución de las Tecnologías de la Información y las Comunicaciones (TICs) dio paso a la utilización de un conjunto de sistemas especialmente diseñados para la planificación de la explotación minera. Esto generó grandes cambios que marcaron un hito en la transformación de la industria.

Las principales alternativas de estos sistemas que imperan en el mercado son DATASTREAM 7I, DATAMINE, GEMCOM, HONEYWELL, MAPTEK, MINCOM, NOVELLNOVELL, SURPAC y WENCO. Cada uno de estos sistemas son desarrollados y comercializados por las empresas Datamine Latin America S.A, Datastream, Gemcom Software Internacional, Honeywell, Maptek, Mincom, Surpac Minex Group y Wenco International Mining Systems. Se puede encontrar su aplicación en todos los aspectos geológicos y de planificación minera como la estimación de reservas, modelamiento de relieves y levantamiento topográfico. Además permiten convertir los complejos datos en información visual que se reflejan en la creación de imágenes interactivas y dinámicas en tres dimensiones (3D), facilitando su comprensión y entendimiento.

En Cuba la industria minera también ha tenido una evolución que se vio fuertemente influenciada por los sucesos políticos y sociales que afectaron la isla. Con el triunfo revolucionario en 1959 y con la colaboración del Consejo de Ayuda Mutua Económica (CAME) se prestó gran atención a esta actividad económica explotando importantes yacimientos polimetálicos de cromo, níquel, cobalto, hierro y otros. A partir de la década de los 90 y con la conexión a Internet, el país comienza el proceso de informatización de la sociedad. Por las necesidades existentes y el avance que viene mostrando la exploración minera con la ayuda de estas tecnologías, se introduce el uso de las mismas en la actividad geológica minera en Cuba.

Los geólogos y mineros cubanos utilizan fundamentalmente los softwares privativos GEMCOM, DATAMINE, MICROLIN y SURPAC. De aquí surge la necesidad de aprovechar el conocimiento adquirido

por los especialistas en las diferentes ramas para el desarrollo de soluciones informáticas cubanas con el fin de lograr la independencia tecnológica. Se han desarrollado en el país algunas herramientas informáticas como TIERRA, SIM, CORTE y MICRONIQ. Las principales entidades que han tenido el compromiso del desarrollo de las mismas son la Universidad de Pinar del Rio, Instituto Superior Metalúrgico de Moa (ISMM), Industrias del Níquel, Empresas Geomineras, Instituto Superior Politécnico José Antonio Echeverría (ISPJAE) y en los últimos dos años la Universidad de la Ciencias Informáticas (UCI).

A pesar de los esfuerzos por explotar estas tecnologías siguen presente algunas deficiencias que surgen del mal aprovechamiento de las mismas y frenan el desarrollo progresivo en esta rama. Si se analiza cómo se ha llevado a cabo este proceso en el país, sobresale la dependencia tecnológica de las empresas hacia las grandes compañías desarrolladoras y comercializadoras de estos softwares mineros. Esto se produce a partir del uso de estas soluciones privativas que implementan estrategias para fidelizar a los clientes con la necesidad de las constantes actualizaciones y la protección de sus códigos fuentes para impedir que se conozca la base de su funcionamiento o hacer otros más eficientes. Otra variante que consume grandes cantidades de dinero es la adquisición de licencias para legalizar el uso y aprovechamiento de estos medios informáticos. Por último, los productos cubanos cuentan con una débil representación de la información en tres dimensiones (3D) de objetos geológicos, lo que impide un análisis más profundo y real de los especialistas mineros.

Debido a la necesidad de solucionar la situación descrita se plantea el siguiente **problema a resolver**: ¿Cómo modelar y visualizar objetos geológicos en tres dimensiones?

La investigación tiene como **objeto de estudio** las técnicas de representación y modelado en 3D de objetos geológicos enmarcado en el **campo de acción** las técnicas de representación y modelado en 3D de objetos geológicos haciendo uso del *framework* VTK.

Para dar solución a los problemas antes expuestos se define como **objetivo general**: Desarrollar un componente que permita modelar y visualizar objetos geológicos en tres dimensiones usando el *framework* VTK.

Como **objetivos específicos** se plantean:

1. Diseñar el componente para modelar y visualizar objetos geológicos en tres dimensiones.
2. Implementar el componente para modelar y visualizar objetos geológicos en tres dimensiones.
3. Validar la solución propuesta.

Para guiar el desarrollo de la investigación se define la siguiente **idea a defender**: Con la implementación de un componente para el modelado y visualización de objetos geológicos en 3D y la incorporación del mismo a los softwares cubanos, se fortalecerá la posibilidad de análisis de los especialistas mineros que trabajan con estos.

Como parte de la investigación que se debe llevar a cabo se utilizan para su desarrollo los siguientes métodos científicos:

- Observación: para determinar la realidad objetiva de la investigación. Permite orientar la investigación a través de la definición del objeto de estudio y el campo de acción de la misma.
- Analítico-Sintético: mediante el cual se extrae la información conceptual más relevante, necesaria para realizar un completo proceso de representación y visualización de objetos geológicos en 3D.
- Histórico–Lógico: para fundamentar la evolución y desarrollo de los sistemas de planificación minera en el mundo y en Cuba.
- Modelación: permite realizar una abstracción del objeto para organizar el flujo de información de la investigación.

En la investigación se esperan como **posibles resultados**:

1. La herramienta de modelado y visualización en 3D de objetos geológicos.
2. La documentación técnica asociada al desarrollo de la herramienta anterior.

El presente trabajo de diploma consta de los siguientes **capítulos**:

**Capítulo 1.-** Fundamentación Teórica: Este capítulo contiene un estudio de las principales soluciones mineras que se utilizan en Cuba y una explicación de los conceptos necesarios en el proceso de modelado y visualización en tres dimensiones de objetos geológicos.

**Capítulo 2.-** Tendencias y Tecnologías Actuales: Este capítulo contiene un análisis de las principales metodologías, tecnologías y herramientas que se utilizan en la actualidad, así como la fundamentación de las escogidas para el desarrollo del componente de visualización de objetos geológicos en 3D.

**Capítulo 3.-** Planificación y Diseño del sistema: En este capítulo se abordan las fases de planificación y diseño propias de la metodología de desarrollo utilizada para la implementación del componente de visualización de objetos geológicos en 3D y se exponen los artefactos generados durante el transcurso de las mismas.

**Capítulo 4.-** Implementación y Pruebas: Este capítulo constituye la última fase del proceso investigativo. En él se abordan las etapas de codificación del componente y ejecución de las pruebas propuestas por la metodología de desarrollo escogida.

# 1 CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## 1.1 Introducción

Este capítulo contiene un repaso histórico sobre la incorporación de las principales soluciones mineras que se utilizan en Cuba. Se exponen los conceptos asociados al dominio como el modelado tridimensional, los fundamentos del trabajo y creación de la escena y el proceso de visualización de objetos en tres dimensiones. Además permite crear los conocimientos básicos que posibiliten una comprensión lógica del contenido y los términos que se exponen en el trabajo.

## 1.2 Análisis de los principales software mineros en Cuba

La minería se ha convertido en un negocio muy competitivo a nivel internacional por lo que se trabaja en pos de lograr una mayor eficiencia. El mayor aprovechamiento de las capacidades y los ahorros tienen un peso importante en el éxito y viabilidad de los proyectos. De ahí la relevancia de los productos orientados a una mejor gestión del proceso minero, a fin de contar en forma clara y amigable con los datos adecuados y oportunamente para la toma de decisiones. Diversas alternativas ofrece la industria de los software mineros disponibles en el mercado, desde soluciones orientadas a aplicaciones específicas hasta productos que ofrecen servicios más integrales, que abarcan y controlan disímiles áreas del negocio minero.

El uso de estos en la actividad geólogo minera en Cuba data de los años 80 con la demanda de un software para el cálculo de las reservas de níquel donde se le dio la tarea al grupo de trabajo de Computación y Geomatemática de la Empresa Geominera de Oriente. Surge entonces MICRONIQ que se utiliza para el cálculo de recursos en la prospección geológica de níquel. Esta solución de origen nacional cuenta con un algoritmo de modelación de recursos conocido como CORTE para el cálculo de los resultados de los trabajos de prospección.

Sobre 1995, la Geominera Oriente y el despacho Che Guevara desarrollan el Software Integral Minero (SIM) que posibilita la modelación de recursos y reservas con introducción de algoritmos de modelación matemática. Hasta entonces el desarrollo de aplicaciones en el país para tratar de cubrir las necesidades de esta rama económica no superaba las barreras de simples algoritmos o pequeños programas que

resolvían problemas específicos. Fue entonces cuando el Instituto Superior Metalúrgico de Moa y el despacho Che Guevara juegan un papel importante con la programación e introducción en la práctica del software TIERRA, de muy amplia riqueza algorítmica de estimación de recursos, planificación y control de la producción (5).

Ante el compromiso que planteaba la apertura de las negociaciones con capital extranjero muchas empresas cubanas tomaron la decisión hacia la adquisición de licencias de productos como GEMCOM que abarca desde las fases de exploración, evaluación de recursos, diseño de mina, optimización, planeamiento minero y control de leyes de producción, hasta la reconciliación y balance metalúrgico a lo largo de la línea de producción. Más tarde se incorpora SURPAC que tiene aplicación en el área geológica minera, específicamente en: exploración geológica, estimación de reservas, diseño de mina a cielo abierto y mina subterránea, planificación minera, modelamiento de relieves y levantamiento topográfico.

Otra de las adquisiciones importantes fue Surpac Minex Group que cuenta con una serie de soluciones computacionales para el trabajo geológico minero, entre las cuales se puede mencionar: Surpac Vision y Minesched. El primero cubre desde las tareas de exploración hasta la planificación de la mina y se caracteriza por tener un fácil manejo y gran potencialidad al manejar información de distintos formatos, además de poder realizar conexiones de trabajos múltiples desde internet. El segundo permite la planificación minera de desarrollo y planificación de la producción desde un corto hasta un largo plazo en minería subterránea y de cielo abierto. Además tiene interfaz directa con MS Project y Excel, permitiendo al planificador obtener flexibilidad y un mejor manejo del plan minero al analizar múltiples opciones en un corto período tiempo.

El último de este conjunto de programas propietarios que se incorporó como herramienta de apoyo a los especialistas cubanos fue DATAMINE. Este cuenta con todas las características fundamentales para asegurar el éxito del negocio minero, tales como integrabilidad, versatilidad, potencialidad y confiabilidad. Estas particulares permiten un trabajo rápido, eficiente, que considera el riesgo asociado, generando resultados auditables y repetibles a lo largo del tiempo, o en cualquier sitio de la compañía.

A partir de las posibilidades que ofrecen estas soluciones en cuanto a modelación y visualización de objetos geológicos en tres dimensiones, se pueden determinar las funcionalidades que son comunes en la

mayoría de los softwares que se venden en el mercado y que constituyen una deficiencia en los que han sido desarrollados en el país hasta este momento. Entre las principales posibilidades de modelación y visualización identificadas aparecen:

- Visualización de pozos de sondeos.
- Visualización de las propiedades de los pozos de sondeos: identificador, concentración de minerales, litología o tipos de rocas.
- Visualización del Modelo de Terreno Digital (MTD) o superficie.
- Modelación del cuerpo de minerales.
- Generación del modelo de bloques.
- Visualización de la concentración de minerales en el modelo de bloques.
- Creación de secciones de análisis.
- Representación de las capas litológicas.
- Diseño de la mina.

Después de exponer el proceso por el que transcurrió la introducción de los diferentes software que se utilizan en el país y algunas de sus principales características, se hace difícil establecer un análisis más profundo de las principales ventajas y desventajas que ofrece el uso de los mismos en la fase de exploración del proceso de extracción de minerales. Esto se debe a la falta de información de los productos cubanos que no existe o no está publicada de manera accesible por el autor. Se conoce el proceso de su introducción y uso expresados anteriormente gracias a la información brindada por el especialista vinculado al desarrollo de estas soluciones en la UCI, el compañero Máster en Ciencias y Geólogo Orestes Gómez.

### 1.3 Conceptos y características del modelamiento 3D

Con la vertiginosa evolución que tienen las computadoras en la actualidad, la generación de objetos en tres dimensiones ha dejado de ser una actividad de grandes computadores para entrar en el mundo cotidiano de las computadoras personales. Para lograr una comunicación y una expresión más técnica o aprovechar al máximo y sin ningún inconveniente las posibilidades que ofrece el mundo de la representación y modelado 3D hay que tener conocimiento de los principales fundamentos geométricos.

#### 1.3.1 Modelado tridimensional

Para modelar un objeto se crea una o varias partes virtuales del mismo que luego pueden ser ensambladas dando como resultado la estructura final. Esta se puede crear mediante un conjunto de figuras geométricas conocidas como primitivas dentro del cual se encuentran: el punto o vértice, la línea o arista, el polígono y las superficies curvas. En las Figura 1 y 2 se puede ver cómo se realiza este proceso en algunos pasos.

**Figura 1:** Secuencia de modelado

**Figura 2:** Secuencia de modelado

Es posible realizar el **Modelado Conceptual** donde lo importante es la representación tridimensional de la idea o concepto. Este puede ser muy rápido y útil para las primeras etapas del desarrollo de un producto, componente o estructura y para fines de visualización, mercadotecnia y venta (2). Otra posibilidad es el **Modelado de Detalle** que incluye todos los detalles con miras al desarrollo de ensambles, animación y

planos para la fabricación o construcción (2). En el proceso de modelado tridimensional intervienen además otros factores como el **espacio de trabajo o escenario** que define los límites del espacio donde se modelará la figura u objeto.

Una vez definida la estructura de los objetos falta proporcionar el acabado superficial. Para esto se puede emplear el **color,** que es quizá lo que más claramente perciben las personas. Normalmente se maneja más de una variable para definir el color, como la difusión[1] o el color ambiente[2]. Otra variante a utilizar es la **especularidad** que controla los brillos o destellos que produce la luz en un objeto. La **reflectividad** interviene en los reflejos del entorno en la superficie del objeto. Muchas veces cuando se mira un objeto no se ve el color de ese material, sino lo que refleja (el caso más extremo sería un espejo). También está el nivel de **transparencia**, que permite observar objetos o no que se encuentran detrás de otros. Un ejemplo cotidiano y real son los cristales que permiten ver a través de ellos lo que hay detrás. Por último la **refracción** controla los efectos donde la luz viaja de un medio a otro, por ejemplo: el cristal de una lupa que deforma lo que hay debajo aumentándolo (3).

Se pueden utilizar otros medios para darle características superficiales a los objetos que se modelan. Aquí entra a jugar su papel el **texturizado**, que consiste en aplicar una imagen como superficie de la estructura. No necesariamente la textura tiene que ser plana, puede ser un cilindro, una esfera o cualquier otra. La imagen puede cubrir completamente la superficie del objeto o repetirse progresivamente. Entre los diferentes tipos de texturas están: la **textura bitmap** que es una imagen real o creada en un programa de edición de imágenes. En esta es muy importante controlar la resolución, adaptándola a las necesidades; si no se hace podría ocurrir que al acercar mucho el objeto aparecieran los píxeles de la imagen. La otra es la **textura procedural** o **shader** que son otros sistemas de texturizados, es decir, algoritmos internos que el mismo programa realiza (3). Según (3), existen cuatro técnicas de texturizado básico:

- Texturizado planar: La textura aparece muy bien definida en la cara donde se aplica pero en las adyacentes aparece proyectada de forma longitudinal.

[1] Controla la cantidad y el color de la luz dispersada por el objeto.
[2] Controla la sensibilidad del material a la luz ambiente (la cantidad de luz que hay presente en las sombras de un objeto).

- Texturizado cúbico: Se proyecta la textura en las 6 direcciones de las caras de un cubo. Se utilizaría para darle textura a objetos cúbicos como un armario por ejemplo.

- Texturizado cilíndrico: Es muy usado por sus posibilidades para aplicar texturas a objetos cilíndricos como por ejemplo una botella.

- Texturizado esférico: Este tiene gran aplicación en objetos esféricos como por ejemplo los mares y continentes del planeta Tierra.

En el modelo 3D que se genera, cada objeto se crea en su sistema de coordenadas local. Por otro lado, su ubicación en el mundo, está dada por su posición en un sistema de coordenadas global[3]. Las **transformaciones** se utilizan para variar el comportamiento inicial de un objeto o el ambiente 3D en una escena, como su posición, tamaño y ángulo de vista. Así se tienen transformaciones locales y transformaciones globales. Además se pueden realizar operaciones complejas como combinación de otras más sencillas. Básicamente para pasar de unas coordenadas a otras se puede aplicar secuencialmente una serie de transformaciones. Entre las transformaciones más usadas se encuentran: la traslación, la rotación y el escalado.

Se aplica una **traslación** cuando se quiere hacer un cambio de posición de un objeto a lo largo de la trayectoria de una línea recta, de una dirección de coordenada a otra (1).

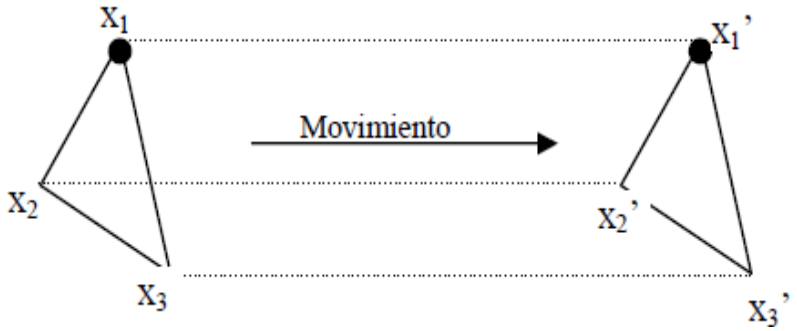



**Figura 3:** Traslación de un triángulo y un punto

[3] Sistema de coordenadas del mundo.

La **rotación** de un objeto es el cambio de su posición siguiendo la trayectoria de una circunferencia, es decir se realiza un giro sobre uno de los ejes de coordenadas XYZ del espacio tridimensional (1). Por ejemplo, si se rota un cubo sobre uno de los ejes quedaría de la siguiente manera:

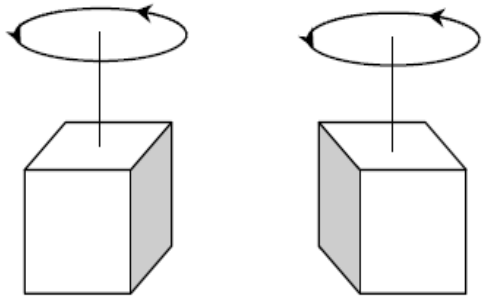


**Figura 4:** Rotación de un cubo en el espacio

La otra transformación es el **escalado** que se utiliza para alterar el tamaño de los objetos. Se puede escalar sobre determinado eje de coordenada o sobre varios ejes a la vez (1).

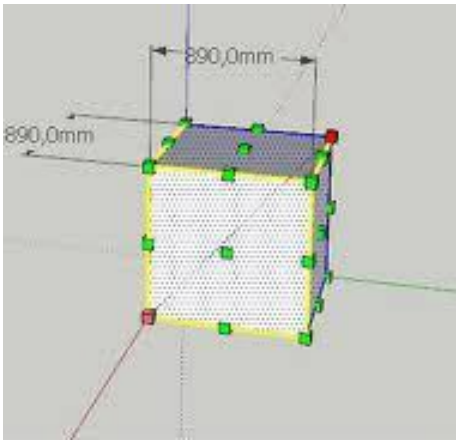



**Figura 5:** Escalado de un cubo en los distintos ejes de coordenadas

### 1.3.2 Fundamentos de la escena

Una vez expuestos los conceptos relacionados con la modelación es necesario explicar los fenómenos externos que influyen en determinadas características del modelo. Es necesario conocer el sistema de

coordenadas que identifica la escena donde tiene lugar la renderización. Este espacio a menudo se llama sistema de coordenadas universal, o "*world*" (mundo).

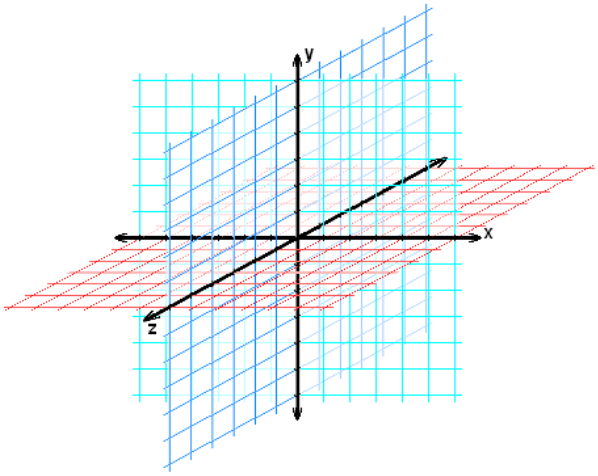



**Figura 6:** Representación de un sistema de coordenadas tridimensionales

El sistema de coordenadas se puede cambiar en función de la vista tridimensional[4] (arriba, frente, izquierda y perspectiva) que se representa muy bien en la siguiente imagen.

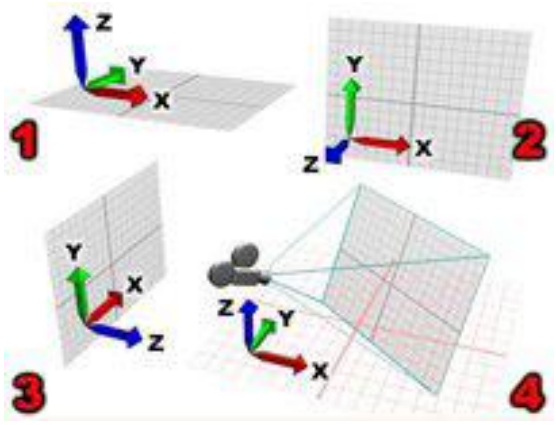



**Figura 7:** Sistema de coordenadas de diferentes vistas

Además cada objeto cuenta con su propio sistema de coordenadas denominado local.

[4] Visualización de una escena desde diferentes puntos, desde el frente, el lateral, arriba y atrás.

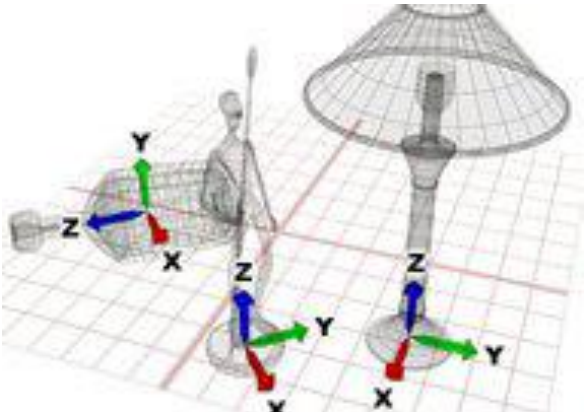



**Figura 8:** Sistemas de coordenadas locales

Una de las técnicas más difíciles que se emplean en la concepción de la escena es la **iluminación**, pues en el mundo real la luz tiene un comportamiento complejo que no resulta fácil imitar en el ordenador. La principal dificultad deriva del hecho de que la luz es emitida desde un determinado punto (el Sol, una bombilla, la llama de una vela, etc) y al chocar con los cuerpos los ilumina, pero también se refleja en ellos, iluminando otros puntos que, en principio, parecería que no deberían verse afectados por ella.

En cualquier programa 3D se dispone de diferentes tipos de luces para iluminar una escena. Por lo general siempre se habla de 4 clases de luces (existen otras, pero éstas son las más importantes) (3):

- Radial: una luz que procede de un punto que se sitúa en la escena y emite sus rayos en todas las direcciones. Sería la luz idónea para una bombilla que cuelga de la pared, o una llama.

- Spot o foco: las típicas luces de los teatros o espectáculos. Están dirigidas en una dirección concreta y se puede controlar la mayor o menor apertura del cono de luz, así como su difusión y otros factores.

- Paralela: es la luz ideal para simular el comportamiento de los rayos del Sol. Éste es un astro que se encuentra en un punto concreto y que emite luz en todas las direcciones, por lo que se puede emplear una luz radial para representarlo. Se llaman paralelas porque aunque se sitúen a muy poca distancia de la escena los rayos que emiten son paralelos, como prácticamente lo son los del Sol cuando llegan a la Tierra.

- Ambiente: La luz no sólo procede de un determinado punto y llega a un objeto en una dirección, iluminándolo desde un cierto ángulo, sino que además rebota. En una habitación con las paredes blancas o claras la luz que entra por una ventana rebota en todas las paredes y objetos que se encuentra a su paso, de modo que puede haber un sofá que está levemente iluminado en una zona en la que debería estar en sombra. Al aire libre también sucede otro fenómeno, que es la dispersión de la luz al atravesar la atmósfera, las nubes o la contaminación.

Un proceso fundamental en la representación 3D es el trabajo con la **cámara**, que se encarga de la forma en que se visualiza la imagen definiendo el punto desde el cual se está viendo la escena. Una cámara decide qué se ve en una toma, desde dónde se ve y cómo se ve. Al igual que una cámara real, la virtual saca una foto de la escena. En general, todos los programas de rendering 3D proveen una por defecto o estándar. Esta tiene un sistema óptico: filtros, lentes y apertura. La orientación de la cámara está determinada por la dirección de un vector y el ángulo en que está rotada con respecto a este (24). Otros elementos importantes son el campo de visión y la profundidad o distancia focal. La última es la encargada de determinar la distancia a que están los objetos que deben verse en el foco. Según (24) se pueden escoger planos cercanos o lejanos.

- Planos cercanos: los objetos que estén demasiado cerca de la cámara no requieren dibujarse porque bloquearían al resto y seguramente estarían muy distorsionados. Tampoco se requieren dibujar los objetos que están detrás de la cámara ya que no se ven.

- Planos lejanos: los objetos muy alejados de la cámara no requieren dibujarse porque aparecerían muy pequeños para ser visualmente significativos. Por otro lado tomaría demasiado tiempo renderizarlos. Descartando estos objetos se pierde detalle pero se gana mucho en lo que respecta a tiempo de renderizado. Por otro lado, puede ser que la escena esté llena de objetos significativos y se requiera despejar la escena de modo tal que se rendericen los cercanos y se descarte el resto.

Los **lentes** de la cámara, reales o simulados, son tal vez la componente más importante en cualquier sistema de cámara porque definen la forma en que el mundo 3D se proyecta en el plano de la imagen. En las cámaras fotosensibles, la película está ubicada exactamente en el plano de proyección. El plano de

proyección de las cámaras simuladas en la computadora puede posicionarse en cualquier lugar de la escena 3D (24). Existen varias formas muy usadas de proyectar un objeto en un espacio que se conocen como: proyección en perspectiva, en paralelo, oblicua (proyecciones caballera y gabinete) y ortogonal. En la siguiente sección de la investigación se decidió describir solo las dos primeras (1).

- Proyección en perspectiva: Se genera una vista de una escena tridimensional proyectando puntos en el plano de despliegue a lo largo de trayectorias convergentes. Esta proyección cambia el tamaño de los objetos de modo que los que se encuentran más alejados del origen de visión se proyectan con menor tamaño que los más cercanos. Esta técnica es la que más se asemeja a la visión natural del ojo y lentes de cámaras.

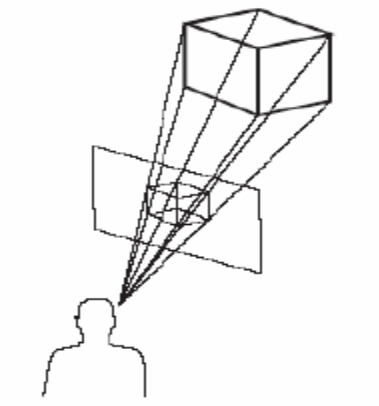


**Figura 9:** Proyección en perspectiva

- Proyección en paralelo: Se genera una vista de una escena tridimensional proyectando puntos en el plano de despliegue a lo largo de trayectorias paralelas. Esta proyección mantiene el tamaño de los objetos y permite obtener distintas vistas bidimensionales de los mismos cambiando el origen de observación.

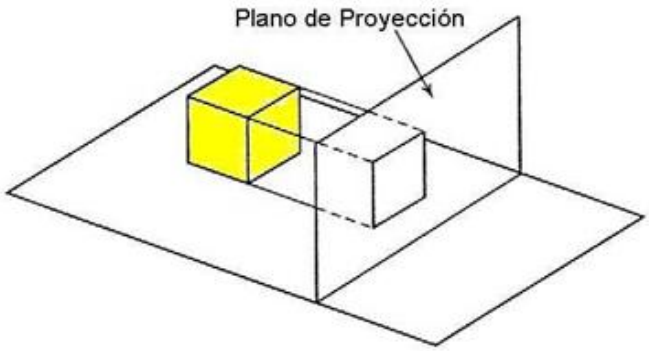



**Figura 10:** Proyección en paralelo

El ordenador se encarga de la fase final realizando el **renderizado** de la escena. Para agilizar este proceso se aplica el recorte, con el objetivo de eliminar objetos o partes de ellos que se encuentran fuera del campo de vista.

Existen varios sistemas (algoritmos) de renderizado, pero los más importantes son (3):

- *Wireframe*: es el más rápido, y lo que muestra es tan sólo unas líneas que definen los polígonos de cada elemento. No se distingue ningún tipo de textura sino tan sólo la estructura de los objetos, pero resulta de enorme utilidad para testear la calidad de los movimientos en una animación antes de pasar a usar otros sistemas mucho más lentos.

- *Phong*: en varios programas éste es un algoritmo bastante tosco, que ni siquiera puede representar sombras arrojadas ni otros muchos fenómenos físicos y en esos casos también se utiliza sólo para testear la animación. Pero debido a su gran velocidad de cálculo algunos programas lo han convertido en su motor de renderizado de más alto nivel, implementándole algunas prestaciones para suplir esas carencias. De hecho, y a pesar de sus muchas limitaciones, es el más utilizado en grandes producciones, donde el tiempo de renderizado no puede dispararse excesivamente.

- *Raytracing*: aquí las reflexiones, las sombras proyectadas o las refracciones son calculadas de acuerdo con parámetros asimilables al mundo real dando un resultado bastante aproximado a la realidad. Lo malo es que resulta muy lento, incluso mucho más que el Phong y normalmente se

utiliza imágenes estáticas. En este sistema cada rayo visual que sale de la cámara llega a los objetos y, en función de los índices de reflexión, transparencia o refracción de aquí pasa a otros objetos o luces.

- *Radiosity*: el sistema de renderizado más realista, pero también el más lento. Aquí se calculan también las interacciones entre la luz y el color de objetos más o menos próximos, de manera que, si por ejemplo, se coloca una pelota roja cerca de una pared blanca se observa como la zona de la pared más cercana a la pelota se tiñe de rojo. Este es un sistema perfecto para simulaciones muy realistas en el campo de la arquitectura, especialmente en interiores y para crear los escenarios de algunos videojuegos en 3D para aportar realismo (con la particularidad de que la escena ya está previamente calculada y guardada en el disco, de lo contrario sería imposible jugar en tiempo real).

### 1.3.3 Visualización de objetos geológicos en 3D

La visualización es el proceso a través del cual se construyen representaciones visuales interactivas de datos, con el objetivo de que su manipulación facilite la extracción de información y el entendimiento de los procesos que los generan. El proceso de visualización puede entenderse como una serie de transformaciones que comienzan con un conjunto de datos y finalizan con una representación visual que los codifica (4). En un sistema de visualización, el usuario puede controlar los parámetros de dichas transformaciones y de ese modo explorar distintos aspectos de los datos. Un modelo de referencia simple para describir las etapas del proceso de visualización está compuesto por tres transformaciones, transformaciones de datos, transformaciones visuales y transformaciones de vista.

Las **transformaciones de datos** pueden ser de dos tipos básicos: conversión de formatos particulares a tablas de datos y manipulación de dichas tablas. En primer lugar, los conjuntos de datos, generalmente multivariados y de gran tamaño, se encuentran en formatos específicos del domino de aplicación y deben ser organizados con una estructura relacional para lograr mayor flexibilidad. Luego, contando con tablas de datos, se pueden aplicar distintas transformaciones que actúan sobre los valores de las tablas o sobre sus estructuras (eliminación de variables, derivación de nuevas variables, clasificación, ordenamiento, etc) (4).

El centro del proceso de visualización son las **transformaciones visuales**, ya que, mediante ellas los datos adquieren una forma gráfica. Las estructuras visuales poseen propiedades gráficas, las cuales son moduladas para codificar información. Ejemplo de estructuras visuales son volúmenes, polígonos, superficies implícitas, curvas y puntos. Por otro lado, entre las propiedades gráficas de éstas se encuentran color, transparencia, textura, forma, tamaño, orientación y posición, siendo algunas más efectivas que otras para la codificación de determinados tipos de datos. El principal desafío en el desarrollo de aplicaciones es encontrar una representación efectiva y eficiente para representar los datos que se desean visualizar. Cabe destacar que después de realizar las transformaciones visuales, sólo se dispone de las estructuras de datos en las cuales se almacena la información geométrica y las propiedades gráficas de cada estructura visual (4).

- Generación de bloques: En el modelo de bloques, estos se describen por la posición de su centro geométrico y su tamaño (definido en forma global). A partir de estos datos, se construye la representación gráfica del modelo, para lo cual es necesario generar los polígonos que componen los bloques. Además, se debe determinar el color asociado a cada bloque, el cual se utiliza para codificar la variable que el usuario desee estudiar.

- Generación de planos cortantes: Se generan planos alineados con los ejes coordenados con los cuales se muestrea el volumen de datos. Los planos permiten reducir la dimensión del problema tridimensional a dos dimensiones, evitando los problemas de oclusión de datos.

- Generación de superficie topográfica: Adicionalmente al modelo de bloques y la secuencia, es posible contar con información topográfica que se puede combinar con las representaciones anteriores. Los datos topográficos están disponibles como puntos de altura ubicados en el plano latitud - longitud. Luego, estos puntos aislados se utilizan para generar una malla de elementos triangulares y finalmente, se calculan las normales promedio en los vértices de los triángulos para generar una representación de buena calidad gráfica.

Las **transformaciones de vista** permiten generar diferentes vistas de las estructuras visuales. Este tipo de transformación involucra rotaciones, traslaciones, acercamientos, recorte geométrico y distorsiones.

Estas transformaciones son aplicadas durante la proyección de las estructuras visuales en imágenes, utilizando algoritmos de computación gráfica (4).

- Transformaciones geométricas: En computación gráfica rotaciones, traslaciones y acercamientos se conocen como transformaciones geométricas y son elementos estándar de las aplicaciones. Desde el punto de vista de la interacción, proveen un medio de manipulación mediante el cual se pueden observar modelos desde distintas posiciones, facilitando el entendimiento de la geometría estudiada.

- Filtros geométricos: El denominado filtro geométrico permite limitar la representación del modelo de bloques, de modo que incluya sólo aquellos bloques cuyas posiciones se encuentran dentro de un rango de coordenadas. Esto permite lidiar con la complejidad tridimensional de los modelos, en particular con el problema de oclusión de bloques al representarlos como tales.

## 1.4 Conclusiones

El estudio de las principales soluciones informáticas para la planificación minera existentes en el país destacó las grandes posibilidades que generan para las industrias cubanas la adquisición y empleo de estas técnicas en la explotación minera. También expresa el esfuerzo por sustituir los softwares adquiridos en el mercado internacional, la necesidad de aumentar el uso y desarrollo de productos nacionales, así como fortalecer las opciones de visualización y modelado para un análisis más profundo de los especialistas. La caracterización del proceso de modelado y visualización en 3D proporcionó los conocimientos básicos necesarios para continuar con el desarrollo de la investigación.

# 2 CAPÍTULO 2: TENDENCIA Y TECNOLOGÍAS ACTUALES

## 2.1 Introducción

Este capítulo contiene un análisis de las principales metodologías, arquitecturas, tecnologías y herramientas que se utilizan en la actualidad. Se establece una comparación entre estas con el objetivo de resaltar algunas ventajas y desventajas, así como un estudio que justifique la selección de las herramientas necesarias para realizar el componente de visualización y modelado de objetos geológicos en 3D.

## 2.2 Acerca de la Arquitectura de Software (AS)

La arquitectura se ha visto fuertemente influenciada por las tendencias de la industria del desarrollo de software existiendo un desfasaje entre los avances de la academia y los conceptos que maneja la industria, citando a Clements y a Northrop: “la arquitectura está siguiendo a la práctica no liderándola”. Es necesario entonces mencionar importantes acontecimientos como lo son la introducción de los principios de factoría y líneas de productos, la aparición de las metodologías ágiles, la ingeniería de software basada en componentes y la aparición de los patrones. En este marco se aprecian tendencias en los estudios arquitectónicos que se van a concretar con posturas que llegan a tomar el carácter de escuelas (6), las más significativas son (7):

- Arquitectura como etapa de ingeniería y diseño orientada a objetos: existe una corriente muy específica que es la conformada por: Rumbaugh, Jacobson y Booch, también conocida como el grupo de los Tres, es una postura, en la cual la arquitectura se ve restringida en sus fases iniciales y preliminares del proceso. La arquitectura se debe tener muy en cuenta ya que identifica una etapa en el diseño orientado a objetos y en general de la ingeniería de software. Lo anterior tiene que ver con temas tales como la abstracción, objetos, clases y subclases, encapsulamiento, herencia y polimorfismo. Para el grupo de los tres la arquitectura se mezcla con el modelado y el diseño, los cuales son los conceptos más importantes.

- Arquitectura estructural, basada en un modelo estático de estilos, Lenguajes de Descripción de Arquitectura (ADLs) y vistas: teniendo en cuenta que los modelos estáticos son los que describen

la estructura estática del sistema en términos de las clases y relaciones, se ha fundamentado en una arquitectura estructural y al igual que en la anterior descripción, esta arquitectura cuenta con importantes representantes tales como Mary Shaw profesora de ciencias de la computación de la universidad Carnegie Mellon de Pittsburgh de Estados Unidos, Paul C. Clements profesor de la misma universidad. Ellos utilizan las descripciones verbales, los ADLs y lenguajes formales de especificación (Cham y Z).

- Estructuralismo arquitectónico radical: se trata de un desprendimiento de la corriente estructuralista, mayoritariamente europea, que asume una actitud más enfrentada con el UML. Se visualizan dos tendencias, una que excluye la relevancia del modelado orientado a objetos y otra que procura definir nuevos metamodelos y estereotipos de UML como correctivos de la situación.

- Arquitectura basada en patrones: este tipo de arquitectura por patrones se utilizó inicialmente en el campo de la arquitectura de software, por Christopher Alexander, a finales de los años setenta. El conocimiento obtenido es llevado al desarrollo de software orientado por objetos y se le aplica específicamente al diseño, seguidamente es aplicado al desarrollo del mismo, se reconoce en ella la importancia del modelo aplicado mucho tiempo después al diseño orientado a objetos.

Teniendo en cuenta la evolución histórica de la disciplina y las diferentes corrientes que han devenido escuelas de la arquitectura, en el presente trabajo se adoptará una alineación con la última de las corrientes, la arquitectura basada en patrones. La misma está estrechamente vinculada al desarrollo basado en componentes y permite realizar un diseño orientado a la resolución de problemas recurrentes en la programación orientada a objetos. Es muy utilizada en la industria de desarrollo de software para lograr mayor flexibilidad y reutilización de las aplicaciones ya que aporta una riqueza conceptual en función de la práctica.

## 2.3 Metodologías de software

La metodología de software se desarrolla con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas incluyen procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software. Una

metodología es un proceso. Se debe mencionar que no existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable y las diversas metodologías existentes son aplicables a diferentes proyectos según sus características.

### 2.3.1 Proceso Unificado de Desarrollo (RUP)

Constituye una metodología pesada, dirigida a grandes proyectos, está centrado en la arquitectura, guiado por casos de uso y es iterativo e incremental. Es un proceso bien definido, estructurado y adaptable a las características y necesidades de cada proyecto específico. Define como sus principales elementos:

- Trabajadores ("quién"): Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto con un equipo.
- Actividades ("cómo"): Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- Artefactos ("qué"): Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- Flujo de Actividades ("cuándo"): Secuencia de actividades realizadas por los trabajadores y que produce un resultado de valor observable.

Las actividades son agrupadas en grupos lógicos, donde se definen 9 flujos de trabajo fundamentales, de los cuales los 6 primeros son considerados flujos de ingeniería y los 3 restantes como flujos de apoyo. Estos flujos son: Modelado de Negocio, Requerimientos, Análisis y Diseño, Implementación, Pruebas, Despliegue, Configuración y Administración de Cambios, Administración de Proyecto y Entorno. Además, RUP define cuatro fases importantes en el desarrollo, las cuales son: Conceptualización (Concepción o Inicio), la cual describe el negocio y delimita el proyecto, se definen sus alcances con la identificación de los casos de uso del sistema; Elaboración, donde se define la arquitectura que tendrá el sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen; Construcción,

donde se obtiene un producto documentado y listo para su utilización, y por último la fase de Transición, en la cual el *release* queda listo para su instalación en condiciones reales.(9)

### 2.3.2 *Open Unified Process* (OpenUP)

Toma las mejores prácticas de RUP, está dirigido por casos de uso, centrado en la arquitectura, es iterativo e incremental y divide el ciclo de vida de un proyecto en las mismas 4 fases de desarrollo. Busca cubrir el mayor número de necesidades para los proyectos de desarrollo en un cierto plazo. Es un proceso iterativo para un desarrollo de software que es: mínimo por solo incluir el contenido fundamental del proceso, completo pues manifiesta un proceso entero para desarrollar un sistema y es extensible ya que puede ser utilizado como base para agregar o adaptar más procesos. Los beneficios de esta metodología se destacan de la siguiente forma: permite disminuir las probabilidades de fracaso en los proyectos pequeños, las detecciones tempranas de errores, evita la elaboración de documentación innecesaria y permite un enfoque centrado en el cliente y con iteraciones cortas. Tiene como principios fundamentales la colaboración en busca de alinear intereses y compartir conocimientos, balancear las necesidades con el fin de maximizar las necesidades de los *stakeholder*[5], así como estar centrado en la arquitectura y llevar a cabo un desarrollo iterativo.(10)

OpenUP propone 6 flujos de Trabajo:

1. **Requerimientos**: en este flujo de trabajo se realizan entrevistas con el cliente para comprender el problema a resolver y se definen los requerimientos.

2. **Análisis y Diseño**: se realiza el diseño de los requisitos que serán después implementados.

3. **Implementación**: en esta disciplina se realiza la implementación del sistema basándose en el diseño realizado.

4. **Prueba**: busca los defectos a los largo del ciclo de vida.

5. **Gestión del Proyecto**: involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

[5] Cualquier persona que sea afectada materialmente por el resultado del proyecto.

6. **Gestión de Configuración y Cambios**: describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización y actualización, control de versiones, etcétera.

Existe una forma básica y fácil de manejar OpenUP que es OpenUP/Basic, objetivos más pequeños y para equipos interesados en el desarrollo ágil e iterativo. OpenUP/Basic es un *framework* de procesos de desarrollo de software de código abierto que permite un abordaje ágil al proceso de desarrollo de software. Provee un conjunto simplificado de contenidos fundamentalmente relacionados con orientación, productos de trabajo, roles, y tareas. OpenUP/Basic es un proceso interactivo de desarrollo de software simplificado, completo y extensible. Es un proceso que valora los beneficios de la colaboración y de los involucrados con el resultado del proyecto, por encima de formalidades innecesarias. (11)

### 2.3.3 Programación Extrema (XP)

Es una metodología de desarrollo de software ágil basada en una serie de valores y de prácticas que posibilitan comunicación, realimentación y reutilización del código con el cual se trabaja, además tiene como objetivo aumentar la productividad a la hora de desarrollar programas. XP procura la sencillez en el software. La Programación Extrema asume que la planificación nunca será perfecta, y que variará en función de cómo varíen las necesidades del negocio. Por tanto, el valor real reside en obtener rápidamente un plan inicial y contar con mecanismos de retroalimentación que permitan conocer con precisión dónde están. En XP la planificación es iterativa por lo que un representante del negocio decide al comienzo de cada iteración qué características concretas se van a implementar (12). Propone como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. Es utilizada en proyectos de corto plazo y pequeño equipo de desarrollo.

**¿Qué es lo que propone XP?**

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.

- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo. (13)

Este proceso de desarrollo de software tiene como principales normas:

1. Planificación

2. Diseño

3. Codificación

4. Prueba (14)

Para determinar el uso de una metodología a utilizar como guía para el desarrollo del componente propuesto es necesario tocar varios aspectos como (tiempo, dinero, personal) que son muy importantes. Teniendo en cuenta las características propias del proceso de desarrollo del trabajo, se exponen a continuación las razones por las que se selecciona XP. Dicha metodología está pensada para proyectos cortos con equipos pequeños y rotables en cuanto a roles. Se basa en Historias de Usuarios (HU) que definen los detalles técnicos mediante un lenguaje de comunicación simple, entendible y lejos del uso de palabras técnicas complejas. Además esta metodología propone un proceso más ligero generando menos documentación e incorpora al cliente como miembro del proyecto flexibilizando los cambios que puedan ocurrir durante el proceso de desarrollo. Está más orientada a la implementación por ser una metodología de software ágil que posibilita la reutilización del código.

### 2.4 Tecnologías y herramientas

El empleo de un conjunto de tecnologías y herramientas informáticas facilita el desarrollo del software aumentando la eficiencia y disminuyendo el coste de este proceso. Además enriquece los medios de trabajo lo que contribuye a un desarrollo basado en estándares, escalable y orientado a la implementación de buenas prácticas. La investigación tecnológica que incluye herramientas, librerías, *frameworks*, estilos, patrones y prácticas ayuda a alcanzar una arquitectura de trabajo sólida. Para ello se debe realizar un

estudio exhaustivo de las herramientas y lenguajes que se adecuan a los requerimientos y presentan un espectro de funciones correspondientes a las necesidades.

### 2.4.1 Lenguaje de programación

Los lenguajes de programación son aquellos sistemas de comunicación que, con una cierta estructura sintáctica y semántica, indica distintas instrucciones a un programa de computadora, permitiendo así crear nuevos programas y software. Entre los principales lenguajes están: Delphi, Visual Basic, Pascal, Java, C, C++, C#, etc. De acuerdo a su nivel de abstracción, se habla de lenguaje de máquina, lenguaje de bajo nivel, lenguaje de medio nivel o lenguaje de alto nivel. Estos facilitan la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar. Existen diversos lenguajes de programación, lo que ha llevado al desarrollo de intérpretes[6] y compiladores[7]. Existen lenguajes informáticos que no son, en realidad, lenguajes de programación, como es el caso del HTML (un lenguaje de marcas) (21).

**C++**

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. El nombre C++ fue propuesto por Rick Masciatti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Es un lenguaje híbrido, que se puede compilar y resulta más sencillo de aprender para los programadores que ya conocen C. El programador tiene el control total de lo que está haciendo, permitiendo una máxima eficiencia al no incorporar verificación de errores en tiempo de ejecución. Las principales características del C++ son: el soporte para programación orientada a objetos y el soporte de plantillas o programación genérica (*templates*). Por ende, se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

6 Programas que adaptan las instrucciones encontradas en otro lenguaje.
7 Programas que traducen de un lenguaje a otro.

Posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel: posibilidad de redefinir los operadores (sobrecarga de operadores) e identificación de tipos en tiempo de ejecución (RTTI). Está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que dificulta su aprendizaje. Los compiladores de C++ generan código nativo con un alto grado de optimización en memoria y velocidad, minimizando la implementación de recursos tales como el polimorfismo y la expansión de patrones lo que lo convierte en uno de los lenguajes más eficientes. La libre utilización de punteros por parte de los programadores, por un lado aporta eficiencia, pero por otro es una fuente de errores de lógica. Por este motivo, lenguajes derivados de C++, como C# y Java, quitaron este recurso y solo permiten referencias a objetos. Otra fuente de errores es la administración de la memoria ya que la asignación y liberación de memoria dinámica es responsabilidad del programador (22).

Basándose en las ventajas que ofrece el lenguaje en cuanto a eficiencia, soporte para programación orientada a objeto, optimización y velocidad de generación de código de los compiladores y la implementación rápida y fácil de programas con el uso de la biblioteca VTK, se toma C++ para construir las sentencias que indiquen a la aplicación que hacer y como visualizar la información en 3D.

### 2.4.2 *Frameworks* de desarrollo

Un *framework* es una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. Se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede añadir las últimas piezas para construir una aplicación concreta. En la actualidad son muy usados para simplificar y agilizar el proceso de desarrollo de aplicaciones. Proporcionan un conjunto de librerías con funcionalidades que aumentan la facilidad del trabajo y disminuyen su complejidad. Además permiten reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. Existen *frameworks* para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y muchos otros (15).

***Visualization Toolkit* (VTK)**

Conjunto de librerías de clases de C++ desarrollado por la compañía *Kitware* fundada en 1998 que posee intérpretes para lenguajes script como lo son Tcl, Java y *Python*. Consiste en un sistema de visualización por software que permite la representación de geometrías 2D y 3D, soportando una amplia variedad de algoritmos de visualización y modelado. Además se ejecuta bajo distintas plataformas como Windows, GNU/Linux y Macintosh. Entre los principales algoritmos que ofrece están: escalar, vector, tensor, la textura, los métodos volumétricos, técnicas de modelización avanzadas como modelado implícito, la reducción de polígonos, suavizado de malla, corte, contorno, y la triangulación de Delaunay. Proporciona también una gran diversidad de representaciones de datos, incluyendo conjunto de puntos desorganizados, datos poligonales, imágenes, volúmenes, mallas estructuradas, rectilíneas y desestructuradas.

Como VTK es una herramienta de código abierto ha impulsado una gran difusión de la misma, extendiendo su aplicación a prácticamente todos los campos en los que se emplean objetos 3D entre los que destacan la medicina, las herramientas industriales, procesamiento paralelo a gran escala, la exploración petrolífera, la mecánica de fluidos, la reconstrucción de superficies a partir de la digitalización con láser.

**Ventajas**

- El modelo gráfico de VTK posee un nivel de abstracción mucho mayor que el de otras librerías de renderización de imágenes como OpenGL. Esto se traduce en una mayor sencillez a la hora de implementar aplicaciones gráficas o de visualizar con VTK.

- Las aplicaciones creadas pueden ser escritas directamente en los lenguajes Tcl, Java, *Python* o C++ lo que aumenta y facilita la posibilidad de implementar aplicaciones en poco tiempo.

- Posee la capacidad de leer una gran cantidad de formatos de imágenes como los son los archivos Dicom, Vtk, Obj, Bmp, Png, así como ingresar a la escena tridimensional texto plano, como

también texto tridimensional y brinda además varios *Widget*[8] para interfaces de usuarios y controles de cámara como son acercamiento, alejamiento y recorrido de la escena.

**Desventajas**

- Bajo rendimiento al intentar visualizar conjuntos de datos grandes (de más de 100 nodos).
- La instalación de VTK requiere soporte para OpenGL.
- Es bastante grande por lo que se recomienda el uso de hardware de aceleración gráfica con soporte para OpenGL y Z-buffer, así como el uso de una computadora con multiprocesador y suficiente *Random Acces Memory* (RAM) para manejar los datos (16).

**OpenGL**

Es un software de interfaz con el hardware gráfico. Esta interfaz consiste de 150 comandos o rutinas que permiten especificar las características de los objetos en el espacio. Esta biblioteca de rutinas está diseñada para el desarrollo de aplicaciones gráficas interactivas y es independiente del hardware, por lo tanto, OpenGL no hace manejo de ventanas ni de interfaces de usuario. Se le considera sin lugar a dudas la *Application Programming Interface* (API) que prevalece en la industria para desarrollar aplicaciones gráficas 2D y 3D. Las aplicaciones OpenGL pueden ser ejecutadas en cualquier plataforma del mercado producto de su diseño independiente de sistemas operativos. Es perceptiva a la red, de manera que es posible separar la aplicación en un servidor y un cliente que verdaderamente produzca los gráficos.

Existe un protocolo para mover por la red los comandos OpenGL entre el servidor y el cliente. Gracias a su independencia del sistema operativo, el servidor y el cliente no tienen que ejecutarse en el mismo tipo de plataforma, muy a menudo el servidor será un supercomputador ejecutando una compleja simulación y el cliente una simple estación de trabajo mayormente dedicada a la visualización gráfica. Hay varias tarjetas gráficas aceleradoras y especializadas en 3D que implementan primitivas OpenGL a nivel de hardware.

[8] Componente gráfico o control con el cual el usuario interactúa.

**Ventajas**

- Ofrece una alta calidad visual y un excelente desempeño.
- Presenta gran estabilidad.
- Todas las aplicaciones OpenGL producen resultados de despliegue visuales consistentes en cualquier API compleja de OpenGL o hardware, sin tener en cuenta el sistema operativo.
- Se han publicado numerosos libros sobre OpenGL, y hay mucho código como muestra disponible, haciendo de la información sobre OpenGL barata y fácil de obtener.

**Desventajas**

- OpenGL no es una biblioteca de alto nivel, ya que no posee rutinas que permitan la descripción de objetos complejos, de hecho las primitivas del OpenGL son: puntos, líneas y polígonos.
- El desarrollador tiene que construir sus propios modelos basándose en unas pocas y simples primitivas (16).

***Framework* Qt**

Es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario producida por la división de software Qt de la empresa Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega *Trolltech*. Actualmente pertenece a la empresa Digia producto de un acuerdo entre Nokia y Microsoft, lo que acarreó la venta de las licencias comerciales de Qt para librarse del desarrollo del mismo. Este sistema de desarrollo es robusto y ha posibilitado la implementación de grandes proyectos como Entorno de Escritorio K (KDE), *Google Earth*, *Skype*, *VirtualBox*, etc. Distribuida bajo los términos de GNU *Lesser General Public License*, es software libre y de código abierto. Cuenta actualmente con un sistema de triple licencia: GPL v2/v3 para el desarrollo de software de código abierto y software libre y la licencia de pago *Q Public License* (QPL) para el desarrollo de aplicaciones comerciales.

Actualmente se encuentra en la versión 4.7.1, liberada en Noviembre del 2010, lógicamente incorporando mejoras en el rendimiento y optimizaciones. Se encuentra programado en forma nativa por el lenguaje C++, pero a través de *bindings*[9] es posible extenderlo para otros lenguajes como *Python*, *Ruby*, *Perl*, etc. Desde hace un tiempo comenzaron a desarrollar sus propias herramientas ideales para los programadores, como es el caso de Qt Creator para programadores C++, Qt Quick para desarrolladores de interfaces y Qt para dispositivos Nokia. Una de las principales ventajas para los programadores es su extensa documentación disponible, y lo fácil que puede ser internacionalizar una aplicación (17).

Teniendo en cuenta las ventajas que ofrece el empleo de *frameworks* para simplificar el trabajo se decide utilizar VTK como herramienta gráfica, pues trae implementado un conjunto de algoritmos necesarios para la visualización y modelado de los objetos geológicos en 3D como: los métodos volumétricos, técnicas de modelización avanzadas como modelado implícito, la reducción de polígonos, suavizado de malla, corte, contorno y la triangulación de Delaunay. Provee una escena con todos los eventos de interacción del usuario con la misma automatizados. Esta tecnología es libre, así que facilita un desarrollo más independiente y se cumple con las nuevas políticas que viene implementando el país sobre la utilización de software libre. Además debido a que su nivel de abstracción es mucho mayor que el de otras, permite implementar aplicaciones en C++ en poco tiempo.

### 2.4.3 Entornos de Desarrollo Integrado

Un entorno de desarrollo integrado (en inglés *Integrated Development Environment* o IDE) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos. Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones. Hoy en día los entornos de desarrollo proporcionan un marco de trabajo para la mayoría de los lenguajes de programación existentes en el mercado (por ejemplo C, C++, C#, Java, Python y Visual Basic entre otros).

[9] Adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquél en el que ha sido escrita.

**Qt Creator**

Qt Creator es un IDE multiplataforma que se ajusta a las necesidades de los desarrolladores Qt. Proporciona características que ayudan a los nuevos usuarios a aprender y comenzar a desarrollar rápidamente, también aumenta la productividad de los desarrolladores con experiencia en Qt. El depurador visual (visual debugger) para C++ es consciente de la estructura de muchas clases de Qt, lo que aumenta la capacidad de mostrar los datos con claridad. Posee un entorno integrado para la creación y diseño de formas para proyectos C++ que permite diseñar rápidamente *widgets* y diálogos usando los mismos *widgets* que se usarán en su aplicación. Las formas son totalmente funcionales y se pueden previsualizar inmediatamente para asegurarse de que se verá y sentirá exactamente como se pensó. Es distribuido bajo tres tipos de licencias: *Qt Commercial Developer License*, Qt GNU LGPL v. 2.1, Qt GNU GPL v. 3.0 y está disponible para las plataformas: GNU/Linux, Mac OSX; Windows, Windows CE, Symbian y Maemo (18).

**Principales características**

- Soporta el lenguaje: C++.
- Posee un avanzado editor de código C++.
- Herramientas para la rápida navegación del código.
- Control estático de código y estilo a medida que se escribe.
- Posee también un Diseñador de Interfaces Gráficas de Usuario (GUI) y un diseñador de formularios.
- Herramienta para proyectos y administración.
- Ayuda integrada sensible al contexto.
- Depurador visual.
- Resaltado y auto-completado de código.
- Soporte para refactorización de código (19).

**C++ Builder 6**

C++ Builder 6 simplifica dramáticamente el desarrollo de aplicaciones Windows, de Internet y distribuidas. Trae más de 70 nuevas características que le permitirán desarrollar más rápido que nunca y ganar interoperabilidad entre plataformas reutilizando proyectos de C++ existentes. El IDE de C++ Builder ofrece todo lo necesario para escribir nuevas aplicaciones C++, incluyendo un compilador de 32 bits, un editor, un depurador, una biblioteca de componentes y un diseñador de formularios integrado. Permite crear una amplia gama de aplicaciones C++ de alta velocidad que van desde software comerciales, de base de datos y de Internet hasta controladores de dispositivo, aplicaciones de consola y sistemas comerciales. Satisface las necesidades de un amplio segmento de programadores de C++: desde desarrolladores a nivel de sistemas hasta desarrolladores de aplicaciones que confían en la productividad de las bases de datos, los componentes reutilizables y el desarrollo de Internet.

La nueva Tecnología de Compilador Adaptable y el Encadenador Incremental trabajan en colaboración para acelerar el tiempo de desarrollo. La Tecnología de Compilador Adaptable crea automáticamente encabezados precompilados basados en patrones de utilización. Con el compilador de 32 bits, se compila y encadena en cuestión de segundos. C++Builder es el único entorno de desarrollo C++ que proporciona rendimiento y productividad con un solo lenguaje, entorno y conjunto de herramientas. Permite además crear aplicaciones visuales y no visuales.

Se determinó como IDE a utilizar el QT Creator que trae integrado el conjunto de bibliotecas para el desarrollo de interfaces gráficas de QT que ofrece una gama amplia de clases para el trabajo con el lenguaje de programación C++. Esto hace más cómodo el trabajo del programador a través de la reutilización de código y se logra una muy buena fluidez con el lenguaje. Además que por sus características puede ejecutarse sobre diferentes sistemas operativos como GNU/Linux, presenta gran estabilidad y desempeño al implementar programas con un código más legible y organizado. Ofrece una interfaz gráfica para abstraer aún más el trabajo. También integra completamiento de código y permite evacuar las dudas que surgen en el proceso de trabajo con el mismo a través de una amplia ayuda muy bien estructurada y ejemplificada. (20).

### 2.4.4 Sistema Operativo

El sistema operativo seleccionado para el desarrollo del trabajo investigativo es GNU/Linux en su distribución Ubuntu 10.04 gracias a las facilidades que ofrece para el desarrollo de software con su gestor de paquetes, que permite descargar e instalar cualquier herramienta necesaria desde los repositorios. Así se realiza el proceso de obtener e instalar las herramientas en el menor tiempo posible, a diferencia de Windows que se descargan los programas si se encuentran en la Internet y luego se instalan. Sobre GNU/Linux se ejecutan con un menor tiempo de respuesta y consumo de recursos los programas seleccionados para el desarrollo del componente propuesto, ya que fueron creados con los lenguajes nativos Python y C++ para cumplir ese propósito. Esto agiliza el trabajo del desarrollador en cuanto a ejecución y actualización de código. Este sistema operativo es mucho más seguro, pues se encuentra fuera de la mira de los atacantes y creadores de virus lo que garantiza la estabilidad en los archivos y evita la pérdida intencionada de la información. Además con su selección se da cumplimiento a las políticas que se adoptaron en el país sobre el uso de software libre y gratuito.

## 2.5 Conclusiones

En el capítulo que finaliza se expusieron las características, ventajas y desventajas de las principales tecnologías y herramientas empleadas en el desarrollo de software influenciando en la utilización del software libre. Quedan explicados los conceptos fundamentales para entender los medios y las tecnologías actuales de desarrollo de softwares. Después de un estudio organizado y fundamentado, se determinan las tecnologías que van a ser utilizadas en el proceso práctico de la investigación. Se crean las bases para dar comienzo a la siguiente fase de la investigación.

# 3 CAPÍTULO 3: PLANIFICACIÓN Y DISEÑO DEL SISTEMA

## 3.1 Introducción

En este capítulo se abordan las fases de planificación y diseño propias de la metodología de desarrollo utilizada para la implementación del componente de visualización de objetos geológicos en 3D. Se identifican las historias de usuarios, se realiza la estimación del esfuerzo que costará implementar cada historia de usuario, las iteraciones en las que se dividirá el proceso y se exponen las tarjetas de Clases, Responsabilidades y Colaboración (CRC) utilizadas durante el diseño del sistema.

## 3.2 Historias de Usuarios

Las HU son utilizadas en la metodología de desarrollo ágil XP para representar una breve descripción del comportamiento del sistema. Emplea terminología del cliente sin lenguaje técnico, se realiza una por cada funcionalidad del sistema y se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos. Reemplazan un gran documento de requisitos y también se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia.

La diferencia más importante entre estas historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuario deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Deben poder ser programadas en un tiempo entre una y tres semanas. Si la estimación es superior a tres semanas, debe ser dividida en dos o más historias. Si es menos de una semana, se debe combinar con otra historia de usuario.

Las Historias de Usuario tienen tres aspectos:

- Tarjeta: en ella se almacena suficiente información para identificar y detallar la historia.
- Conversación: cliente y programadores discuten la historia para ampliar los detalles (verbalmente cuando sea posible, pero documentada cuando se requiera confirmación).

- Pruebas de Aceptación: permite confirmar que la historia ha sido implementada correctamente (12).

Como resultado del trabajo realizado se identifican las siguientes historias de usuarios:

- Visualización de objetos geológicos.
- Construcción de la caja del modelo.
- Visualización de objetos geométricos.
- Ejecución de transformaciones visuales.
- Modelación del plano de trabajo.
- Proyección de figuras geométricas.
- Edición de objetos geométricos.
- Diseño de objetos geológicos (cuerpo de minerales).

### 3.3 Estimación de esfuerzo por historias de usuarios

Para el buen desarrollo del sistema propuesto, se realizó una estimación de la duración para cada una de las historias de usuarios identificadas, llegando a los resultados que se muestran a continuación:

**Tabla 1:** Estimación de la duración de las Historias de Usuarios

| Historias de usuarios | Puntos de estimación |
|---|---|
| Visualización de objetos geológicos | 1 semana |
| Construcción de la caja del modelo | 1 semana |

| | |
|---|---|
| Visualización de objetos geométricos | 1 semana |
| Ejecución de transformaciones visuales | 1 semana |
| Modelación del plano de trabajo | 1 semana |
| Proyección de figuras geométricas | 1 semana |
| Edición de objetos geométricos | 1 semana |
| Diseño de objetos geológicos (cuerpo de minerales) | 1 semanas |

### 3.4 Plan de iteraciones

El Plan de entrega está compuesto por iteraciones de no más de tres semanas. En cada iteración el cliente decide qué historias se implementarán (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del plan de iteraciones son las historias de usuario. El trabajo de cada iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores. Estas tareas no tienen que necesariamente ser entendidas por el cliente, pueden ser escritas en lenguaje técnico y son para el uso estricto de los programadores.

Una vez definidas las HU y estimado el esfuerzo propuesto para la realización de cada una de ellas, se decide realizar el sistema en 6 iteraciones, las cuales se describen detalladamente a continuación:

**Iteración 1**

Esta iteración tiene como objetivo darle cumplimiento a las HU que se consideraron de mayor importancia para el desarrollo del componente de visualización de objetos geológicos en 3D. Al concluir dicha iteración se contará con las funcionalidades descritas en las HU 1, 2 y 3 permitiendo la interacción con la superficie y los pozos de sondeos que se encuentran dentro de la caja del modelo tridimensional formado.

**Iteración 2**

Esta iteración tiene como objetivo darle cumplimiento a las funcionalidades descritas en las HU 4 y 5 que permitirán interactuar con el modelo a partir de transformaciones visuales como el paneo, la rotación, el *zoom* y el *clippin*. Además se genera el plano sobre el que se estará diseñando y se implementa el movimiento hacia delante y atrás del mismo en la dirección de proyección.

**Iteración 3**

Durante el transcurso de esta iteración se le da cumplimiento a las funcionalidades descritas en las HU 6 y 7. Una vez terminada la misma, será posible dibujar figuras geométricas en el plano de trabajo a partir de puntos, así como corregir el diseño en caso de errores.

**Iteración 4**

Durante el transcurso de esta iteración se le da cumplimiento a la funcionalidad descrita en la HU 8. Una vez terminada la misma, el componente permitirá generar el cuerpo de minerales a partir de los puntos de las figuras diseñadas en cada plano de trabajo.

Después de explicar cada una de las iteraciones en las que queda dividido el ciclo de desarrollo del componente propuesto es necesario simplificar el desarrollo de cada una de las HU dividiéndolas en tareas de ingeniería.

Como resultado del estudio realizado se obtienen las siguientes tareas de ingeniería:

- Construir geometría del objeto geológico.
- Visualizar en el espacio la geometría construida.
- Obtener los puntos extremos.
- Construir la geometría de la caja.
- Trasladar el modelo sobre el plano de trabajo.

- Rotar el modelo en el espacio.
- Acercar o alejar el modelo en la profundidad del espacio.
- Cambiar la ubicación de los planos de la cámara.
- Trasladar plano de análisis.
- Crear el plano de proyección.
- Mover el plano de proyección.
- Trasladar las coordenadas de las figuras.
- Seleccionar la figura del espacio de trabajo.
- Cambiar propiedad visual.
- Mover figura geométrica.
- Eliminar figura geométrica.
- Construir la figura geométrica.
- Capturar coordenadas de los puntos.
- Triangular los puntos.
- Visualizar el resultado.

### 3.5 Plan de duración de las iteraciones

Como parte del ciclo de vida de un proyecto guiado por la metodología de desarrollo de software XP, se crea el plan de duración de cada una de las iteraciones que se llevan a cabo durante el desarrollo del

proyecto. En este plan se muestra la duración de cada iteración y el orden en que serán implementadas las HU en cada una de las mismas.

**Tabla 2:** Plan de duración de las iteraciones

| Iteraciones | Historias de usuarios | Duración total de iteraciones |
|---|---|---|
| Iteración 1 | Visualización de objetos geológicos | 3 semanas |
| | Construcción de la caja del modelo | |
| | Visualización de objetos geométricos | |
| Iteración 2 | Ejecución de transformaciones visuales | 2 semanas |
| | Modelación del plano de trabajo | |
| Iteración 3 | Proyección de figuras geométricas | 2 semanas |
| | Edición de objetos geométricos | |
| Iteración 4 | Diseño de objetos geológicos (cuerpo de minerales) | 2 semanas |

### 3.6 Arquitectura del sistema

Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. Establece los fundamentos para que analistas, diseñadores, programadores, etc, trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. Es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación (25). Además define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación (módulos principales) y sus responsabilidades, sus interfaces, la comunicación entre ellos (protocolos de interacción y comunicación) y la ubicación en el hardware (26).

En el capítulo anterior se expusieron las razones de una alineación con la corriente arquitectónica “Arquitectura basada en patrones”.

### 3.6.1 Patrones de diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, basadas en la experiencia y que se ha demostrado que funcionan. Son descripciones de cómo resolver problemas que pueden ser utilizadas en diversas situaciones. Son técnicas para flexibilizar el código haciéndolo satisfacer ciertos criterios. Estos proveen una forma efectiva de compartir experiencias con el resto de programadores de softwares orientados a objetos (27).

**Experto**

Este es un patrón que se usa para asignar las responsabilidades a las clases en dependencia de la información que poseen. Da origen al diseño donde el objeto del software realiza las operaciones que se aplican normalmente al objeto real que representa. Se conserva el encapsulamiento ya que los objetos se valen de su propia información para hacer lo que se les pide. El comportamiento se distribuye entre las clases que cuentan con la información requerida siendo las estas más sencillas, cohesivas y fáciles de entender y mantener.

**Creador**

Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. Es idóneo para asumir la responsabilidad de crear lo que contiene o registra. Brinda soporte a un bajo acoplamiento ya que la clase creada tiende a ser visible en el creador, lo que supone menos dependencias en cuanto al mantenimiento y aumenta las posibilidades de reutilización.

**Bajo acoplamiento**[10]

Soporta el diseño de clases más independientes y reutilizables lo que reduce el impacto de los cambios y aumenta la productividad. Debe incluirse en la decisión de asignar responsabilidades y nunca

[10] Una medida de la fuerza con que una clase depende o está conectada a otras, con qué las conoce y con qué recurre a estas.

considerarse de forma independiente de otros patrones como Experto o Alta cohesión. Han de tener escaso acoplamiento las clases muy genéricas y con grandes posibilidades de reutilización.

**Alta cohesión**[11]

Se da una alta cohesión funcional cuando los elementos de algún componente colaboran para producir algún comportamiento bien definido. Se obtiene una clase con responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Una clase donde se implementa este patrón es más fácil darle mantenimiento, entenderla y reutilizarla.

**Controlador**

Se deben elegir los controladores que manejen los eventos de entrada externa que reciben los sistemas. Un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. Una categoría de controlador es de fachada que representa el sistema global, es decir una clase que representa el comportamiento del sistema entero. Otra categoría es el controlador de casos de usos y se manifiesta cuando el mismo empieza a saturarse con demasiadas responsabilidades y asigna las mismas a otras clases individuales controlables.

**Polimorfismo**

Para asignar las responsabilidades cuando varían las alternativas o comportamientos en dependencia de los tipos. Un diseño basado en la asignación de responsabilidades mediante el polimorfismo puede ser extendido fácilmente para que realice nuevas variantes.

**Observador**

Brinda un mecanismo que permite a un componente transmitir de forma flexible mensajes a aquellos objetos que hayan expresado interés en él. Define una dependencia uno a muchos entre objetos, de modo que cuando el estado de un objeto cambia, se les notifica el cambio a todos los que dependen de él y se actualizan de forma automática.

[11] Una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase.

### 3.7 Clases-Responsabilidades-Colaboración

Las tarjetas Clases-Responsabilidades-Colaboración (CRC) son una de las principales piezas de diseño que propone la metodología XP. Permite al programador centrarse y apreciar el desarrollo orientado a objetos a través de la representación de las clases. Estas tarjetas constan de tres secciones donde se recogen el nombre de la clase, las funcionalidades o responsabilidades y los colaboradores con otras clases. Una clase es cualquier persona, cosa, evento, concepto, pantalla o reporte. Las responsabilidades o funciones de una clase son las características y las acciones que realizan, sus atributos y métodos. Los colaboradores de una clase son las demás clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades. En la práctica conviene tener pequeñas tarjetas de cartón por ejemplo, que se llenarán y que se mostrarán al cliente, de manera que se pueda llegar a un acuerdo sobre la validez de las abstracciones propuestas. En la siguiente tabla se muestra una propuesta de la plantilla:

A continuación se muestran las clases identificadas en el componente:

- Geometric.
- Geometric3DTriangulation.
- GeometricMultiPoint.
- GeometricPoint.
- GeometricPolygon.
- GeometricPolyline.
- GeometricTriangulation.
- VisualObject.
- Designer.
- DesignInteractor.
- Viewer.
- ViewerBase.

## 3.8 Conclusiones

En este capítulo se explicó el proceso de planificación donde se definieron cada una de las historias de usuarios, la estimación del esfuerzo, las iteraciones y las tareas necesarias para completar la solución, así como el plan de duración. Se explicó la arquitectura utilizada para elaborar el diseño de la solución mediante los diferentes patrones propuestos. En esta fase se obtuvieron las tarjetas CRC que permitieron modelar las funcionalidades y colaboradores referentes a cada una de las clases que conformarán el componente.

# 4 CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

## 4.1 Introducción

El presente capítulo contiene la documentación de las últimas fases propuestas por la metodología de desarrollo XP. Se exponen las técnicas que se utilizan durante el proceso de implementación de la solución, como el estándar de codificación necesario para lograr un código entendible y muy bien documentado. Además, se realizan las pruebas seleccionadas, donde se muestran algunos ejemplos y los resultados del componente ante las mismas.

## 4.2 Estándar de codificación

Los estándares de codificación, también llamados estilos de programación o convenciones de código, no son más que convenios para escribir código fuente en ciertos lenguajes de programación. Estos estándares elevan la mantenibilidad del código, sirven como punto de referencia para los programadores, mantienen un estilo de programación y ayudan a mejorar el proceso de codificación, haciéndolo, entre otras cosas, mucho más eficiente. Para entender su elaboración y resultado práctico, se describe a continuación el estándar de codificación utilizado durante la implementación del componente:

### 4.2.1 Comentarios

Para los comentarios se empleó el estilo de Javadoc que permite generar una ayuda a través de la herramienta Doxygen.

**Normas comunes:**

- Cada bloque de comentario comienza con /** y luego un salto de línea.
- Cada línea siguiente comienza con * que debe estar alineado al primero de la línea inicial.
- Se realiza una descripción breve en la segunda línea concluyendo en un punto.

- Se deja una línea en blanco para comenzar con la descripción más detallada. Esta finaliza en un punto también.
- Se cierra el bloque con */ en una nueva línea.

Ejemplo:

```
/**
 * Variable temporal.
 *
 * Esta variable es utilizada para almacenar las propiedades
 * en cada iteración del ciclo.
 */
vtkSmartPointer<vtkProp3D> temp;
```

**Funciones**

- Después de la descripción detallada hay que describir los parámetros.
- Para esto se comienza con @param seguido del nombre del parámetro y la descripción del mismo.
- Se deja una línea en blanco y con @return se describe el valor devuelto.

Ejemplo:

```
/**
 * Funcionalidad que recibe una propiedad y devuelve su posición.
 *
 * Obtiene los puntos que se encuentran en el plano
 * y busca la posición que representa el punto centro
 * de la propiedad en la lista.
 *
 * @param p argumento de tipo propiedad.
```

```
 *
 * @return número entero que representa la posición.
 */
int DesignInteractor::getPropPosition(vtkSmartPointer<vtkProp3D> p)
{
    QList<double *> temp = getPtsToPlaneProps();
    return temp.indexOf(p->GetCenter());
}
```

**Clases**

- Las clases deben de documentarse en su declaración, es decir, en los ficheros de cabecera.
- Los atributos y funciones privados no se documentan.
- Los atributos y funciones públicos o protegidos se documentan en el fichero fuente.

Ejemplo:

```
/**
 * Clase abstracta geometría.
 *
 * Esta clase permite implementar geometrías como el punto,
 * polilínea, polígono, etc.
 */
class Geometric
{
protected:
    vtkSmartPointer<vtkPoints> points;
    vtkSmartPointer<vtkCellArray> cell;
    vtkSmartPointer<vtkPolyData> poly;
public:
    Geometric();
```

```
    ~Geometric();
    virtual vtkSmartPointer<vtkPolyData> getGeometric() = 0;
};
```

### 4.2.2 Nombres de identificadores

Se considera como identificador a los nombres de variables, arreglos, matrices, apuntadores, funciones, así como cualquier tipo de dato definido por el usuario.

- Deberán tener un nombre significativo en el idioma Inglés.
- Al leerlo debe inducir en su función, sin tener que consultar manuales o hacer demasiados comentarios.
- Para nombres que se usen con frecuencia o para términos largos, se pueden usar abreviaturas estándar para que éstos tengan una longitud razonable.
- Para distinguir palabras dentro del nombre se deberá emplear un guión bajo (_).
- Para las funciones y clases, todas las palabras deberán comenzar con las primeras letras en mayúscula.
- Para las funciones del tipo lectura de datos o cambio de datos, deberá comenzar con el prefijo get o set en minúsculas.

Ejemplo:

```
double cliprange;
double *obj_center;
virtual void OnLeftButtonDown();
virtual double* getModelCenter();
```

### 4.2.3 Sangrías

- Las sangrías serán establecidas a través del tabulador en el teclado.
- Las llaves deben estar alineadas con el comienzo de la instrucción a la que definen el cuerpo.
- Las instrucciones contenidas en el cuerpo de otras deben de tener un nivel más de sangría que la contenedora.

Ejemplo:

```
if(ev == "pan")
{
   DeleteEvent("rotate");
   this->GetInteractor()->Render();
}
```

### 4.2.4 Líneas y espacios en blanco

- Insertar una línea en blanco para separar instrucciones diferentes.
- Se deben incluir espacios en blanco a ambos lados de los operadores.
- Es posible distribuir una instrucción grande sobre varias líneas. Si se hace, se deben seleccionar puntos de ruptura que tengan sentido, como después de una coma (,) en el caso de una lista, o después de un operador (+, -, etc.) en el caso de una expresión larga.
- Agregar un espacio en blanco después de cada coma (,).

Ejemplo:

```
double *obj_center = moveprop->GetCenter();
```

```
this->ComputeDisplayToWorld(this->GetInteractor()->GetLastEventPosition()[0],
                            this->GetInteractor()->GetLastEventPosition()[1],
                            disp_obj_center[2], old_pick_point);
```

### 4.3 Pruebas

Uno de los pilares de la metodología XP es el uso de las pruebas para comprobar el funcionamiento de los códigos que se vayan implementando. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones (24).

La metodología ágil XP divide las pruebas en dos grupos: pruebas unitarias y pruebas de aceptación. Las pruebas unitarias son desarrolladas por los programadores y se encargan de verificar el código automáticamente y las pruebas de aceptación están destinadas a verificar que al final de cada iteración las HU cumplen con la funcionalidad asignada y satisfagan las necesidades del cliente.

Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado y el final de una iteración y el comienzo de la siguiente, por esto el cliente junto a una persona del proyecto encargada de esta fase son los encargados de diseñar las pruebas de aceptación.

#### 4.3.1 Desarrollo dirigido por pruebas

El desarrollo dirigido por pruebas (TDD por sus siglas en ingles), es una práctica de programación que involucra otras dos prácticas: Escribir las pruebas primero (*Test First Development*) y refactorización (*Refactoring*). Para escribir las pruebas generalmente se utiliza la Prueba Unitaria (*Unit Test*). Primeramente se escribe una prueba y se verifica que las pruebas fallen, luego se implementa el código que haga que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio y que funcione. La idea es que los

requerimientos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que los requerimientos se hayan implementado correctamente.

TDD permite un diseño más robusto y fácil de mantener a través de la noción de pruebas. Estas obligan a reflexionar sobre el comportamiento del código y la forma de garantizar que funciona según lo previsto. La mayoría de las veces, el código influenciado por TDD es relativamente seguro, y sin duda, es bastante simple (25).

A continuación se explican las fases del ciclo de vida para el desarrollo dirigido por pruebas:

- Escribir la prueba: Para escribir la prueba, el desarrollador debe entender claramente las especificaciones y los requisitos. El diseño del documento deberá cubrir todos los escenarios de prueba y condición de excepciones.
- Escribir el código haciendo que pase la prueba: Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces. Ésta es la parte conducida por el diseño, del TDD. Como parte de la calibración de la prueba, el código debe fallar la prueba significativamente las primeras veces.
- Ejecutar las pruebas automatizadas: Si pasan, el programador puede garantizar que el código resuelve los casos de prueba escritos. Si hay fallos, el código no resolvió los casos de prueba.
- Refactorización y limpieza en el código: Después se vuelven a efectuar los casos de prueba y se observan los resultados.
- Repetición: Después se repetirá el ciclo y se comenzará a agregar las funcionalidades adicionales o a arreglar cualquier error (25).

**Pruebas Unitarias**

Una prueba unitaria es la verificación de una unidad lógica[12] determinada dentro de un sistema. Se verifica que una unidad funciona correctamente por sí misma, sin tener en cuenta las relaciones que pueda tener

[12] Las partes que constituyen un programa (módulos, paquetes, clases, subsistemas, funciones, etc).

con otras partes del sistema. Se crean pequeños módulos auxiliares, que se encargan de verificar el funcionamiento de otras unidades lógicas. Cada una de las pruebas individuales que se hacen a una unidad lógica del software se denominan “Casos de pruebas”, y “Colección de pruebas” al conjunto comprendido por la unión de las mismas. Los responsables de la realización de las pruebas unitarias en un software son los programadores encargados de cada unidad lógica del mismo. Se recomienda su desarrollo antes que la propia unidad. Esto permite definir como se quiere utilizar la unidad, es decir, los parámetros de entrada que serán necesarios y los parámetros de salida esperados.

**Ventajas**

- Los errores son más fáciles de localizar.
- Los errores están más acotados
- Se reducen los efectos secundarios[13].
- Se da más seguridad al programador.
- Las pruebas funcionales se hacen más sencillas.
- El programador escribe código de una forma más lógica.

Se diseñaron 5 colecciones de pruebas con algunos casos de pruebas cada una, que fueron aplicadas sobre 5 de las unidades lógicas del componente, obteniéndose resultados satisfactorios. A continuación se listan cada una de las mismas:

- GeometricPointTest: colección de pruebas que se aplican para comprobar la unidad GeometricPoint del componente.
- GeometricPolylineTest: colección de pruebas que se aplican para comprobar la unidad GeometricPolyline del componente.

[13] Errores producidos al intentar arreglar otros.

- GeometricPolygonTest: colección de pruebas que se aplican para comprobar la unidad GeometricPolygon del componente.
- GeometricTriangulationTest: colección de pruebas que se aplican para comprobar la unidad GeometricTriangulation del componente.
- Geometric3DTriangulationTest: colección de pruebas que se aplican para comprobar la unidad Geometric3DTriangulation del componente.

**Pruebas de Aceptación**

Las pruebas de aceptación no son más que validaciones que se le realizan al sistema para ver si cumple con el funcionamiento esperado y le permite al usuario determinar su aceptación en cuanto a su funcionalidad y rendimiento. Son pruebas de caja negra que se realizan partiendo de las historias de usuarios.

Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Para su ejecución, se deben especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada. En caso de que fallen varias pruebas, se debe indicar el orden de prioridad de resolución. Es recomendable publicar los resultados de las pruebas de aceptación de manera que todo el equipo esté al tanto de esta información.

Las pruebas funcionales que se aplicaron a las historias de usuario implementadas, fueron concebidas y aprobadas por el cliente y ejecutada por el equipo de desarrollo. A continuación se listan cada una de las mismas:

- Prueba de aceptación Visualizar objeto geológico.
- Prueba de aceptación Construir la caja del modelo.
- Prueba de aceptación Visualizar objeto geométrico.
- Prueba de aceptación Ejecutar transformación visual.

- Prueba de aceptación Modelar el plano de trabajo.
- Prueba de aceptación Proyectar figura geométrica.
- Prueba de aceptación Editar objeto geométrico.
- Prueba de aceptación Diseñar objeto geológico (cuerpo de minerales).

### 4.3.2 Validación mediante la encuesta

Para comprobar el impacto del componente en el cliente, se diseñó y aplicó una encuesta sobre los principales líderes de los equipos de desarrollo del proyecto productivo Minería, perteneciente al centro de desarrollo GEYSED de la facultad 9. El proceso consistió en recopilar información numérica a partir de evaluaciones entre 1 y 5 puntos, que escogieron los encuestados sobre diferentes parámetros a los que sometieron el componente. A continuación se muestra la estructura de las preguntas utilizadas en la encuesta:

Analice y evalúe el comportamiento del componente después de interactuar con el ejemplo. Para esto debe escoger una puntuación entre 1 y 5 puntos para cada una de las siguientes variantes:

a) Tiempo de respuesta en la ejecución de las funcionalidades __

b) Cantidad y frecuencia de fallos __

c) Facilidad de uso __

d) Capacidad para integrarse a otros sistemas __

Los valores numéricos resultantes fueron agrupados en una tabla para determinar la media de cada parámetro medido en el procedimiento y representados visualmente en una gráfica de barras. A continuación se ilustra dicha tabla:

**Tabla 3:** Resultados de la encuesta a los miembros del proyecto Minería

| Encuestados | Tiempo de respuesta | Ocurrencia de fallos | Facilidad de uso | Capacidad de integrarse | Media resultante |
|---|---|---|---|---|---|
| **Eddy** | 5 | 4 | 5 | 4 | **4,5** |
| **Rocny** | 5 | 4 | 4 | 5 | **4,5** |
| **Dagoberto** | 3 | 4 | 5 | 4 | **4** |
| **Armando** | 4 | 3 | 4 | 5 | **4** |

Se obtuvo una valoración numérica del componente por cada personal encuestado. La misma representa una media que oscila entre los 4 y 4,5 puntos lo que aprueba la siguiente conclusión: los datos obtenidos demuestran una aceptación positiva de la herramienta por parte del cliente, representado por los principales miembros del proyecto Minería.

## 4.4 Conclusiones

En el capítulo se definió el estilo o estándar necesario para lograr uniformidad en la codificación, que facilitará el soporte del código así como la comprensión del mismo. Se induce la posibilidad de crear una documentación a partir de las descripciones hechas con el apoyo de la herramienta Doxygen y la interpretación del estilo de comentarios Javadoc. Se diseñaron y realizaron distintas pruebas que permitieron corregir los errores cometidos y concluir una versión más estable del componente propuesto.

# 5 CONCLUSIONES

- Se realizó un estudio sobre las posibilidades que genera el empleo de software en la fase de exploración minera y la necesidad de incluir una herramienta informática de esa envergadura y de origen nacional en el trabajo diario de los especialistas cubanos de esta rama económica. Esto permitió elaborar un diseño metodológico con objetivos dirigidos a contribuir en la solución de la situación anteriormente descrita.

- Se documentó en 4 capítulos el proceso de desarrollo de la herramienta para modelado y visualización de objetos geológicos en 3D.

- Las herramientas seleccionadas y estrategias utilizadas a partir de un análisis comparativo de las tendencias y tecnologías actuales, están alineadas con las políticas de seguridad y uso de software libre de la universidad y el país.

- Se realizó el diseño del componente para lo que se generaron todos los artefactos propuestos por la metodología utilizada.

- Se implementó el componente siguiendo un conjunto de normas, concebidas para lograr uniformidad en el código, facilitando la comprensión del mismo a través del estilo de comentarios Javadoc, que permitirá generar una documentación completa mediante Doxygen y aumenta la posibilidad de soporte y su reutilización en el desarrollo de sistemas futuros.

- Se diseñaron y aplicaron las pruebas necesarias para corregir los errores cometidos en la codificación de la herramienta. Este proceso garantizó la satisfacción del cliente hacia el componente, demostrada en los resultados positivos arrojados por la encuesta aplicada a los principales miembros del proyecto Minería.

# 6 RECOMENDACIONES

Después de cumplir con los objetivos trazados, se recomienda incorporar al componente las siguientes funcionalidades:

- Permitir crear objetos visuales compuestos a partir de varias geometrías.
- Realizar operaciones booleanas entre los objetos.
- Generar el modelo de bloques.

# 7 REFERENCIAS BIBLIOGRÁFICAS


1. **Arévalo, Cristina Charro y Armijo, Vinicio Washington Valencia.** *Modelo tridimensional de la historia geológica del volcán Cotopaxi.* Escuela Politécnica Nacional. Quito : s.n., 2007. Proyecto previo a la obtención del título de ingeniero en sistemas informáticos y de computación.

2. **Morales, Luis Javier Iturriaga.** Modelado 3D | Visualización Fotorrealista. [En línea] 2004. http://webspace.webring.com/people/ui/iturriagal/modelado.html.

3. **Vila, Cristobal.** Introducción a la animación 3D. [En línea] Noviembre de 2000. http://www.etereaestudios.com/training_img/intro_3d/intro_3d_eng.htm.

4. **Soto, Leonardo, Sánchez, Ricardo y Amaya, Jorge.** *Desarrollo de un Sistema de Visualización para la Planificación Minera.* Universidades Talca, Concepción y Chile. Chile : s.n.

5. *Taller sobre Software de Minería.* 2008.

6. Introducción a la Arquitectura de Software. **Billy Reynoso, Carlos. 2004.** Buenos Aires, Argentina : http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp, 2004.

7. **Guerrero, Lugo Manuel Barbosa.** *arquitectura de software como eje temático de investigación.* Grupo de Investigación DAVINCI del Programa de Ingeniería de Sistemas, Universidad Libre. s.l. : AVANCES Investigación en Ingeniería - 2006 No. 4, 2006.

8. **Universidad de las Ciencias Informáticas.** Entorno Virtual de Aprendizaje. [En línea] 2009. [Citado el: 20 de enero de 2011.] http://eva.uci.cu/course/view.php?id=102.

9. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** El Proceso Unificado de Desarrollo de Software. España : Adison-Wesley.

10. **Epf.eclipse.org.** OpenUP/Basic. [En línea] 2006. [Citado el: 20 de enero de 2011.] http://epf.eclipse.org/wikis/openupsp/.

11. CBASQA. [En línea] 02 de Septiembre de 2008. [Citado el: 26 de Enero de 2011.] http://cbasqa.wordpress.com/2008/09/02/proceso-de-desarrollo-openup/.

12. **Escribano, G. F. (2002).** Introducción a Extreme Programming.

13. Extreme Programming. [En línea] 28 de Septiembre de 2009. [Citado el: 26 de Enero de 2011.] http://www.extremeprogramming.org/.

14. Extreme Programming. [En línea] [Citado el: 26 de Enero de 2011.] http://www.extremeprogramming.org/rules.html.

15. **Euphoria IT Ltda. 2005-2008.** euphoriait. [Online] 2005-2008. [Cited: enero 20, 2011.] http://euphoriait.com/articulos/framework_web.

16. **FREDY, CARRANZA ATHÓ y LAURA, FLORIAN CRUZ.** *OPENGL Y VTK .* UNIVERSIDAD NACIONAL DE TRUJILLO. Trujillo Perú : s.n., 2006. COMPUTACIÓN GRÁFICA II.

17. **Maldonado, Daniel M. 2011.** El CoDiGo K. [Online] enero 17, 2011. [Cited: enero 25, 2011.] http://www.elcodigok.com.ar/2011/01/10-datos-librerias-qt/#more-2354.

18. **Inventa Network. 2011.** Pixelco Blog. *Un blog sobre diseño y desarrollo web, Internet y tecnología.* [Online] 2011. [Cited: enero 28, 2011.] http://pixelcoblog.com/qt-creator-completo-entorno-de-desarrollo-multiplataforma/.

19. **Nokia Corporation. 2008-2009.** Developer Network. [Online] 2008-2009. [Cited: enero 25, 2011.] http://developer.qt.nokia.com/wiki/Category:Tools::QtCreator_Spanish.

20. **Buenas Tareas. 2011.** buenas tareas. *Caracterisaticas De C++ Builder.* [Online] 2011. [Cited: enero 28, 2011.] http://www.buenastareas.com/ensayos/Caracterisaticas-De-C-Builder/776285.html.

21. **Definición.de. 2008.** Definición.de. [Online] 2008. [Cited: enero 20, 2011.] http://definicion.de/lenguaje-de-programacion/.

22. **Lenguajes de programación. 2009.** Lenguajes de programación. [Online] 2009. [Cited: enero 23, 2011.] http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

23. **Amaro Calderón, Sarah Dámaris y Valverde Rebaza, Jorge Carlos.** *Metodologías Ágiles.* Facultad de Ciencias Físicas y Matemáticas, Universidad Nacional de Trujillo. Trujillo-Perú : s.n., 2007.

24. **Castro, Silvia.** *ESCENAS 3D.* Dpto. de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur. 2008.

25. **Morales, Sergio.** http://carloscar7.files.wordpress.com/2008/02/carlos-david-carballo-espana.doc. [En línea]

26. **Casanovas, Joseph.** http://www.willydev.net/descargas/UsabilidadArquitectura.pdf. [En línea]

27. **Largman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Facultad de ingeniería, Universidad de la república. Montevideo : s.n., 2001. pág. 265.

# 8 BIBLIOGRAFÍA

**Paredes, Ignacio García y De Cáceres García, José Luis.** *Fundamentos de los gráficos en 3D.* Ingeniería Informática. 2002-2003. Informática Gráfica.

Minería Chilena. *El mercado de los software mineros.* [En línea]

**S.A, SOCIEDAD MINERA TOTE;.** areaminera. *La industria minera en Cuba.* [En línea] Septiembre de 2003. http:/www.areaminera.com.

**Centro de Información para la Prensa, de la Unión de Periodistas de Cuba.** Cuba por el camino de la ciencia y la tecnología. [En línea] http://www.cip.cu.

*Taller sobre la necesidad y posibilidad de la creación de un software de minería cubano.* **Oficina Nacional de Recursos Minerales.** Moa : s.n., 2008.

**Menéndez, Rosa.** Proyectos. [En línea] 2009. [Citado el: 23 de enero de 2011.] Disponible en: http://www.usmp.edu.pe/publicaciones/boletin/fia/info36/proyectos.html#a.

**Free Download Manager.** Free Download Manager. [En línea] 2010. [Citado el: 20 de enero de 2011.] Disponible en: http://www.freedownloadmanager.org/.

**Simple Machines LLC. 2006-2009.** PortalHacker.Net. [Online] 2006-2009. [Cited: enero 25, 2011.] http://www.portalhacker.net/index.php/topic,22211.0.html.

**Yahoo. 2011.** Yahoo! Respuestas. [Online] 2011. [Cited: enero 23, 2011.] http://es.answers.yahoo.com/question/index?qid=20070811200816AAA6CvJ..

# 9 GLOSARIO

Escalado: Altera el tamaño de un objeto en dirección de los ejes de coordenadas.

Escena: archivo que contiene toda la información necesaria para identificar y posicionar todos los modelos, luces y cámaras para su renderización.

Espacio geométrico: el conjunto de todos los puntos del universo físico.

La programación: proceso de escritura del código fuente de un software. De esta forma, la programación le señala al programa informático qué tiene que hacer y cómo realizarlo.

Lenguaje de alto nivel: formado por elementos del lenguaje humano.

Lenguaje de bajo nivel: el lenguaje de programación que se acerca al funcionamiento de una computadora.

Lenguaje de máquina: cadenas binarias que pueden ser legibles de manera directa por la computadora.

Lenguaje de medio nivel: comparte características con los lenguajes de bajo nivel pero también con los más avanzados.

Línea o arista: Segmento de recta donde se cruzan dos planos, definido por la posición de las aristas que la limitan (1).

Modelado: proceso para crear una parte virtual que represente exactamente la geometría de una pieza, componente, estructura o producto, puede constar de una o más piezas (2).

Polígono: es una figura plana y cerrada formada por tres o más segmentos de línea unidos en sus extremos.

Proyección: proceso de reducir un espacio de n dimensiones en otro de n-1 dimensiones (1).

Punto o vértice: Punto definido por una posición en los ejes XYZ dentro del mundo tridimensional (1).

Renderizado**:** proceso que realiza el ordenador para calcular cada una de las imágenes de la escena (3).

Se refiere a tridimensional cuando se está analizando algo donde están presentes las tres dimensiones del espacio: largo, ancho y profundidad.

Un Sistemas de coordenadas tridimensionales no es más que un sistema de referencia formado por un conjunto de valores que permiten definir unívocamente la posición de cualquier punto de un espacio geométrico respecto a un punto denominado origen[14]**.**

Vector: todo segmento de recta dirigido en el espacio.

[14] Punto centro que permite posicionar el modelo.