ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"CREACIÓN DE UN SOFTWARE CON PROGRAMACIÓN CONCURRENTE PARA LA ILUMINACIÓN Y SOMBREADO DE SUPERFICIES VECTORIALES UTILIZANDO GRADIENTES VECTORIALES"

Trabajo de titulación presentado para optar al grado académico de:

INGENIERO EN SISTEMAS INFORMÁTICOS

AUTOR: MICHAEL CHRISTIAN GARCÍA ROBLES

TUTOR: DR. ALONSO WASHINGTON ALVAREZ OLIVO

Riobamba-Ecuador

2017

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

El Tribunal del Trabajo de Titulación certifica que: El trabajo técnico: **"CREACIÓN DE UN SOFTWARE CON PROGRAMACIÓN CONCURRENTE PARA LA ILUMINACIÓN Y SOMBREADO DE SUPERFICIES VECTORIALES UTILIZANDO GRADIENTES VECTORIALES** "de responsabilidad del señor Michael Christian García Robles, ha sido minuciosamente revisado por los Miembros del Tribunal del Trabajo de Titulación, quedando autorizada su presentación.

	FIRMA	**FECHA**
ING. WASHINGTON LUNA **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	_____	_____
ING. PATRICIO MORENO **DIRECTOR DE LA ESCUELA DE INGENIERÍA EN SISTEMAS**	_____	_____
DR. ALONSO ALVAREZ **DIRECTOR DEL TRABAJO DE TITULACIÓN**	_____	_____
DRA. NARCISA SALAZAR **MIEMBRO DEL TRIBUNAL**	_____	_____

Yo, Michael Christian García Robles, soy responsable de las ideas, doctrinas y resultados expuestos en este trabajo y el patrimonio intelectual del Trabajo de Titulación pertenece a la Escuela Superior Politécnica de Chimborazo.

Michael Christian García Robles

DEDICATORIA

Sin duda alguna y con el mayor amor que se pueda expresar dedico este trabajo a mis padres, Verónica Robles y Cesar García que con el sacrificio y perseverancia diaria me enseñaron que siempre tengo que luchar por mis ideales y sueños.

A mis amigas y amigos que siempre me acompañan en los buenos y malos momentos, demostrándome con una sonrisa la verdadera solución a un problema

Michael

AGRADECIMIENTO

Antes que nada, Gracias a Dios por haberme brindado la salud para poder realizar este trabajo de titulación.

Un agradecimiento al Tribunal de Tesis, Dr. Alonso Álvarez y Dra. Narcisa Salazar que con el conocimiento y la capacidad supieron guiarme en los procesos más difíciles de esta tesis.

Michael

TABLA DE CONTENIDO

Página

RESUMEN ..... **xiii**

SUMMARY ..... **xiv**

INTRODUCCIÓN **2**

CAPÍTULO I

1. MARCO REFERENCIAL **3**

1.1. Antecedentes **3**

1.2. Justificación **4**

1.2.1. Justificación Teórica ***4***

1.2.2. Justificación Aplicativa ***5***

1.3. Objetivos **5**

1.3.1. Objetivo General ***5***

1.3.2. Objetivos Específicos ***5***

CAPÍTULO II

2. MARCO TEÓRICO **6**

2.1. Programación Concurrente **6**

2.1.1. Introducción ***6***

2.1.2. Definición ***8***

2.2. Iluminación y Sombreado **10**

2.2.1. Introducción ***10***

2.2.2. Fuentes de Luz ***13***

2.2.3. Fuentes de Color ***14***

2.2.4. Luz Ambiente ***15***

2.2.5. Fuentes ***15***

2.3. Gradientes Vectoriales..... 17

2.3.1. Introducción..... 17

2.3.2. Definición 18

CAPÍTULO III

3. MARCO METODOLÓGICO.....30

3.1. Investigación Aplicada.....30

3.1.1. Método de Investigación para la resolución de casos 30

3.1.2. Entrevista..... 30

3.1.3. Técnica de recolección de requerimientos en base a Casos de Uso..... 30

3.2. Metodología de desarrollo SCRUM.....31

3.2.1. Planeación y estimación 33

3.2.2. Implementación..... 34

3.2.3. Revisión y retrospectiva 35

3.2.4. Liberación..... 35

3.3. Análisis de la programación concurrente en la CPU y GPU.....36

3.3.1. Gráficas en el Computador. 36

3.3.2. El sistema gráfico. 36

3.3.3. Ventana Real y Ventana Pantalla..... 37

3.4. Clases y Métodos a utilizar.....39

3.4.1. Repositorio de Modelos Matemáticos..... 39

3.4.2. Clase vector 3D 39

3.4.3. Segmento 3D..... 39

3.4.4. Clase Superficie..... 39

CONCLUSIONES.....40

RECOMENDACIONES.....41

GLOSARIO DE TERMINOS

BIBLIOGRAFÍA

ANEXOS

ÍNDICE DE TABLAS

Tabla 1-3: Fases scrum.....31

Tabla 2-3: Roles scrum 32

Tabla 3-3: Historia de usuario 33

Tabla 4-3: Formato pruebas de aceptación.....35

ÍNDICE DE FIGURAS

Figura 1-2: Representación de la aplicación con programación orientada a objetos. 6
Figura 2-2: Paralelismo de procesos en un determinado tiempo 7
Figura 3-2: Tiempo en la CPU en sistemas monoprocesador 8
Figura 4-2: Proceso de iluminación de objetos 10
Figura 5-2: Plano de proyección de los gráficos por computadora 11
Figura 6-2: Sombreado de objetos dependiente de su orientación 12
Figura 7-2: Fuente de luz de un objeto 13
Figura 8-2: Iluminación de un objeto 14
Figura 9-2: Intensidad de iluminación 16
Figura 10-2: Contraste de la iluminación de objetos con una sola fuente 16
Figura 11-2: Gradientes Vectoriales 17
Figura 12-2: Representación de gradientes 18
Figura 13-2: Planos equidistantes formando curvas de nivel 21
Figura 14-2: Cortes de los planos proyectados sobre el papel 22
Figura 15-2: Cortes de los planos proyectados sobre una depresión 23
Figura 16-2: Curvas de nivel de la depresión 23
Figura 17-2: Diferentes itinerarios a cumbre 24
Figura 18-2: Equidistancia entre curvas de nivel 25
Figura 19-2: Curvas de nivel auxiliares 27
Figura 20-2: Sombreado de pendiente 28
Figura 21-2: Sombreado oblicuo 29
Figura 1-3: Diagrama de casos de usos general 31
Figura 2-3: Sistema gráfico 37
Figura 3-3: Ventana real y de la pantalla 38

ÍNDICE DE ILUSTRACIONES

Gráfico 1-3: Tiempos de desarrollo de los sprints 34

ÍNDICE DE ABREVIATURAS

API	Application Programming Interface (Interfaz de Programación de Aplicaciones)
CPU	Central Processing Unit (Unidad Central de Proceso)
FPS	Frames Per Second (Fotogramas por segundo)
GPGPU	General-Purpose Computing on Graphics Processing Units
GPU	Graphic Processing Unit (Unidad de Procesamiento Gráfico)
OpenCL	Open Computing Language
SP	Stream Processor

RESUMEN

En el presente trabajo de titulación se creó un software con programación concurrente para la iluminación y sombreado de superficies vectoriales utilizando gradientes vectoriales. Para el desarrollo de la aplicación y del prototipo se usaron las herramientas: C++, OpenGL y Visual Studio como entorno de desarrollo integrado (IDE). Para el análisis de la programación concurrente sobre la unidad de procesamiento gráfico (GPU) y unidad central de proceso (CPU), se utilizaron las siguientes herramientas: gDebuuger, Fraps y el monitor del sistema de Windows, con las cuales se pudo realizar las mediciones sobre el prototipo. Se hizo uso del método analítico con el cual se analizó los datos obtenidos tras las mediciones realizadas. Los resultados obtenidos fueron: en tiempo de ejecución: GPU ejecutó el algoritmo en 20 segundos frente a los 45 segundos que se tardó en CPU; tramas por segundo (FPS): 8 fps en CPU y 18 fps en GPU; uso de procesador: 67% en CPU frente a 10% en GPU; uso de memoria en megabytes: 48 MB en GPU y 86 MB en CPU. Las mediciones de los parámetros se las hicieron sobre un prototipo construido para el efecto, con lo cual se obtuvo que la tecnología GPU es la más adecuada para implementar la aplicación Fractal Build, siendo superior en cada uno de los parámetros analizados, obteniendo 75 puntos en la sumatoria, sobre 14.5 puntos que obtuvo la tecnología CPU. Por lo analizado se concluye que el uso de la tecnología GPU es la más adecuada debido a las ventajas que brinda al momento de ejecutar una aplicación gráfica. Se recomienda continuar con el estudio de las implementaciones sobre GPU como procesador de propósito general debido a sus múltiples beneficios.

PALABRAS CLAVE: <TECNOLOGÍA Y CIENCIAS DE LA INGENIERÍA>, <INGENIERÍA DE SOFTWARE>, <UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)>, <FRAPS (SOFTWARE)>, <SCRUM (METODOLOGÍA DE DESARROLLO ÁGIL)>, <INFORMACIÓN ACADÉMICA>.

SUMMARY

In the present work of qualification a software with Concurrent programming was created for the lighting and shading of vector surfaces using vector gradients. For development of application and of the prototype used tools: C++, OpenGL and Visual Studio as an integrated Development Environment (IDE). For the analysis of concurrent programming on the Graphics Processing Unit (GPU) and Central Programming Unit (CPU), used for the following tools: Debugger, Fraps and the Windows system monitor, with which was possible to perform the measurements of the prototype. The analytical method was used with which the data obtain after the measurements were analyzed. The result obtained were: at runtime; GPU executed the algorithm in 20 seconds versus the 45 seconds that took in CPU; Frames Per Second (FPS): 8 fps on CPU and 18 fps on GPU; Processor usage: 67 % on CPU versus 10% on GPU; Memory usage in megabytes: 48 MB in GPU; and 86 MB in CPU. The measurements of the parameters were made on a prototype built for the purpose, with which it was obtained that the GPU technology is most suitable to deploy the application Fractal Build, being superior in each one of the parameters analyzed, obtaining 75 points in the summation, over 14.5 points obtained by the CPU technology. For the analyzed it was concluded that the used use of the GPU technology is most suitable because of the advantages that is offers when executing a graphical application. It is recommended to continue with the study of the implementations on GPU as General-Purpose Processor due to its multiple benefits.

Keywords: <TECHNOLOGY AND THE ENGINEERING SCIENCES >, <SOFTWARE ENGINEERING>, < GRAPHICS PROCESSING UNIT (GPU)>, <FRAPS (SOFTWARE)>, <SCRUM (AGILE DEVELOPMENT METHODOLOGY)>, <ACADEMIC INFORMATION>,

INTRODUCCIÓN

En este trabajo de titulación tuvo como objetivo desarrollar un producto software con programación concurrente para la iluminación y sombreado de superficies vectoriales utilizando gradientes vectoriales, ¿Cómo se logró realizarlo?, - Mediante el proceso de estudio de programación concurrente, determinación de parámetros y pruebas de comparación en programación concurrente entre GPU y CPU, análisis de modelos sobre iluminación y sombreado, finalmente investigando la aplicación gradiente de un vector.

SCRUM, fue la metodología de desarrollo software mediante la cual se realizó el proceso de desarrollo utilizando los procesos encaminados a los responsables que notablemente influyo para la óptima relación de los involucrados del proyecto desde la fase de planificación hasta la fase de cierre, listando requisitos, capturando requisitos abstractos formalizándolos en historias de usuario finalmente dando el cumplimiento eficiente de los requisitos del sistema.

El presente trabajo de titulación está compuesto por tres capítulos; en el primer relata cómo se recolectaron los antecedentes así también porqué se justificó reflejándolo en los objetivos planteados, a continuación, ya en el segundo capítulo se detalla la descripción y sustentación teórica motivo de estudio para el presente trabajo y el tercer capítulo se expone el proceso de desarrollo del trabajo de titulación además la investigación aplicativa y análisis de la programación concurrente.

CAPÍTULO I

1. MARCO REFERENCIAL

1.1. Antecedentes

La programación concurrente es el nombre dado a notaciones de programación y técnicas para expresar paralelismo potencial y resolver los problemas resultantes de sincronización y de comunicación.

Existen varios aspectos en nuestro mundo inherentemente distribuido que hacen necesaria la programación concurrente. En primera instancia es más fácil modelar de una manera concurrente un sistema del mismo tipo, que hacerlo encajar dentro del paradigma secuencial que de ninguna manera esquematiza el comportamiento de ese sistema. También es necesario considerar que el programar concurrentemente permite que los sistemas sean más fácilmente escalables debido al modularidad de su desarrollo y que también estos pueden ser mucho más eficientes debido a que permiten la ejecución en paralelo de múltiples instrucciones. La programación concurrente es usada para modelar y simular sistemas físicos, inclusive si esos sistemas no están controlados directamente por un computador. La simulación es una herramienta importante en la optimización de sistemas físicos; la programación concurrente brinda una forma natural de asignar segmentos del programa para representar objetos físicos y por eso ayuda mucho a representar simulaciones.

Se piensa que la concurrencia como un tópico avanzado mucho más difícil que la programación serial, por lo que necesita ser estudiado muy detenidamente y muy ampliamente, sin embargo, al estudiar más a fondo podemos darnos cuenta que esto no es verdad y podemos entonces percibir todas las ventajas que esta nos ofrece.

Al programar concurrentemente y por ello compartir recursos surgen algunos problemas que necesitan ser resueltos para así aprovechar al máximo todas las ventajas que la programación concurrente nos puede brindar. Entre los problemas más importantes podemos mencionar algunos como:

- ✓ La ejecución de un proceso que pueda afectar la información perteneciente a otro proceso que se ejecuta en paralelo a menos que esté autorizado a hacerlo (datos compartidos).
- ✓ El abrazo mortal, que es el estado en el que dos transacciones se quedan bloqueadas, una

esperando por recursos que está utilizando la otra.

✓ Inanición: Estado al que llegan una transacción cuando es seleccionada repetidamente para abortar y así evitar un abrazo mortal.

✓ Libelo: Estado en donde una transacción cambia continuamente de estado en respuesta a cambios en otra transacción mientras la otra hace lo mismo, sin conseguir ningún resultado con ello.

1.2. Justificación

1.2.1. Justificación Teórica

Al hablar del origen de la programación concurrente definitivamente hablamos de los Sistemas Operativos de multiprogramación, en la que solamente un procesador de "gran capacidad" en aquel entonces generaba o repartía tiempo a un gran número de usuarios. Para cada usuario, la sensación era que el procesador estaba dedicado para él. Durante la década de los sesenta y setenta esto fue así. La programación de sistemas con capacidades de concurrencia se hacía a bajo nivel, en ensamblador, pues aparte de no disponer de lenguajes de alto nivel con capacidades de concurrencia, se primaba la supuesta eficiencia del código escrito directamente en ensamblador. La aparición en 1972 del lenguaje de alto nivel Concurrent Pascal (Brinch-Hansen, 1975), desarrollado por Brinch Hansen, se encargó de romper este mito y abrir la puerta a otros lenguajes de alto nivel que incorporaban concurrencia.

Desde entonces la programación concurrente ha ido ganando interés y actualmente se utiliza muy a menudo en la implementación de numerosos sistemas. Tres grandes hitos se nos antojan importantes para que la programación concurrente actualmente sea tan importante:

➢ La aparición del concepto de thread o hilo que hace que los programas puedan ejecutarse con mayor velocidad comparados con aquellos que utilizan el concepto de proceso.

➢ La aparición más reciente de lenguajes como Java, lenguaje orientado a objetos de propósito general que da soporte directamente a la programación concurrente mediante la inclusión de primitivas específica.

➢ La aparición de Internet que es un campo abonado para el desarrollo y la utilización de programas concurrentes. Cualquier programa de Internet en el que podamos pensar tales como un navegador, un chat, etc. están programados usando técnicas de programación concurrente.

1.2.2. Justificación Aplicativa

Actualmente existe una carencia de herramientas de diseño de superficies que cubra todas las expectativas para la enseñanza del análisis vectorial en 3D, como por ejemplo las curvas de nivel de una superficie.

Debido al avance tecnológico en nuestro país y al desear competitividad en aspectos de gráficas en el computador, es necesario crear aplicaciones que nos puedan ayudar en el proceso de enseñanza aprendizaje del nivel superior, para lo cual es importante conocer diferentes técnicas y componentes que ayudan a la representación y descripción de las herramientas pedagógicas como el de los tipos de gráficos por computadora bajo diferentes estructuras geométricas y lenguajes de programación, para crear una aplicación interactiva en el computador con el propósito de facilitar al usuario, estudiante o profesor en el desarrollo de laboratorios virtuales, como es el caso de este trabajo.

Hablamos de educación virtual hoy en día, gracias a las aplicaciones informáticas que pretenden acortar distancias geográficas simulando laboratorios interactivos mediante la web, permitiendo desarrollar prácticas en el laboratorio en tiempo real

1.3. Objetivos

1.3.1. Objetivo General

Crear un software con programación concurrente para la iluminación y sombreado de superficies vectoriales utilizando gradientes vectoriales.

1.3.2. Objetivos Específicos

- Estudiar cómo se desarrolla la programación concurrente en CPU y GPU.
- Determinar los parámetros y pruebas para comparar programación concurrente en GPU y CPU.
- Analizar modelos existentes sobre iluminación y sombreado para poderlos aplicar en objetos 3D.

CAPÍTULO II

2. MARCO TEÓRICO

2.1. Programación Concurrente

2.1.1. Introducción

La idea de programación concurrente siempre ha estado asociada a los sistemas operativos: Un sólo procesador de gran capacidad debía repartir su tiempo entre muchos usuarios. La programación de estos sistemas se hacía a bajo nivel (ensamblador) (Flórez et al., 2016, p. 2). Tiempo después aparecerían lenguajes de alto nivel con soporte para este tipo de programación. Su uso y potencial utilidad se apoya en: threads o hilos, java e internet. Se habla de concurrencia cuando ocurren varios sucesos de manera contemporánea. (Flórez et al., 2016, p. 2) (Rodríguez, n.d., p. 2)

Con este detalle, la concurrencia en computación está directamente coligada a la ejecución de distintos procesos que pueden coexisten temporalmente. (Flórez et al., 2016, p. 2-3) Para comprender y dar una definición más formal, debemos tener claro la diferencia entre programa y proceso:

Programa: Conjunto de sentencias y/o instrucciones que se ejecutan secuencialmente. Se asemeja al concepto de clase dentro de la POO. Es por tanto un concepto estático. (Flórez et al., 2016, p. 4)

Proceso: Básicamente, se puede definir como un programa en ejecución. Líneas de código en ejecución de manera dinámica. Se asemeja al concepto de objeto en POO, la misma que se encuentra representada en **la Figura 1-2.** (Flórez et al., 2016, p. 5)

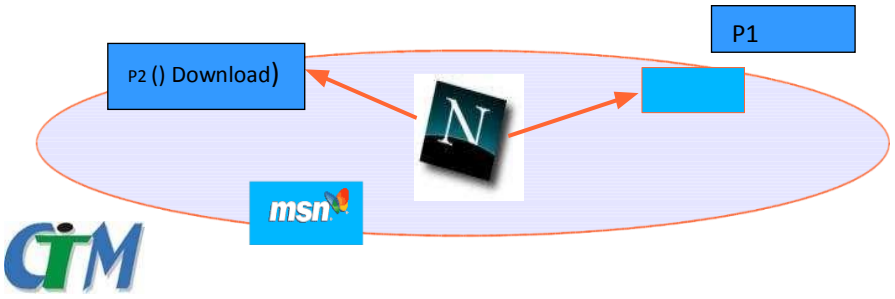


Figura 1-2: Representación de la aplicación con programación orientada a objetos.

Fuente: (Rodríguez, 2013, p. 3) Introducción a la Programación Concurrente

Concurrencia

La concurrencia surge en el preciso momento cuando unas asociaciones de dos o más procesos son simultáneas. Un caso particular es el paralelismo matemáticamente, sus "pendientes" de ejecución son iguales (programación paralela), así como se puede apreciar en la **Figura 2-2.**

Figura 2-1: Paralelismo de procesos en un determinado tiempo

Fuente: (Rodríguez, 2013, p. 4) Introducción a la Programación Concurrente

Los procesos pueden "competir" o colaborar entre sí por los recursos del sistema. Por lo tanto, existen tareas de colaboración y sincronización. (Flórez et al., 2016, p. 6-7) La programación concurrente se encarga del estudio de las nociones de ejecución concurrente, así como sus problemas de comunicación y sincronización. (Rodríguez, n.d., pp. 2–4)

Beneficios de la Concurrencia

Velocidad de ejecución. Al subdividir un programa en procesos, éstos se pueden "repartir" entre procesadores o gestionar en un único procesador según importancia. (Flórez et al., 2016, p. 7-8)

Solución a problemas de esta naturaleza. Existen algunos problemas cuya solución es más fácil utilizando esta metodología: (Rodríguez, n.d., p. 5-6)

- Sistemas de control: ingreso de datos, validaciones y respuesta actualizada (por ejemplo, autenticación de un sistema). (Flórez et al., 2016, p. 9)

- Tecnologías Web: Hablamos de los servidores web que tienen la capacidad de recibir simultaneas peticiones de parte del frontend, recibir aquellos datos y generarlos en información de vuelta. servidores de redes, chat, email y muchos más. (Flórez et al., 2016, p. 10)

- Aplicaciones basadas en GUI: El usuario realiza muchas a la aplicación grafica (navegadores web, por ejemplo). (Flórez et al., 2016, p. 11)

- Simulación: Podríamos mencionar la inteligencia artificial, programas que modelan sistemas físicos con autonomía, y además que son capaces de aprender. Un ejemplo de ello podemos mencionar la domótica en hogares inteligentes. (Flórez et al., 2016, p. 12)

- Sistemas gestores de Bases de Datos: Cada usuario un proceso. (Flórez et al., 2016, p. 7-8)

Concurrencia y Hardware

Hasta ahora sólo hemos hablado del software, aunque el hardware y su topología es importante para abordar cualquier tipo de problema.

- **Sistemas Monoprocesador.** Podemos tener concurrencia, gestionando el tiempo de procesador para cada proceso. como Miguel ángel Rodríguez (Rodríguez, n.d., p. 7) muestra en la **Figura 3-2.**

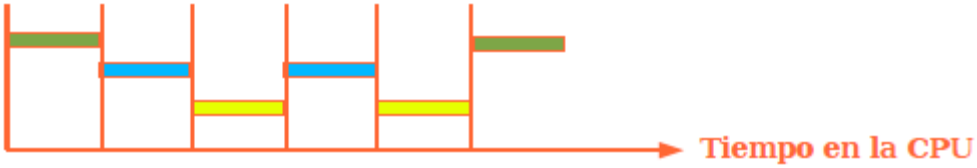


Figura 3-2: Tiempo en la CPU en sistemas monoprocesador

Fuente: (Rodríguez, 2013, p. 7) Introducción a la Programación Concurrente

- **Sistemas multiprocesador.** Un proceso en cada procesador. Estos pueden ser de memoria compartida (fuertemente acoplados) o con memoria local a cada procesador (debidamente acoplados). Un ejemplo muy conocido y útil son los sistemas distribuidos (por ejemplo, Beowulfs). En relación a la concurrencia se pueden clasificar en aquellos que funcionan con variables/memoria compartida o paso de mensajes.

2.1.2. Definición

Se puede definir a la programación concurrente como las notaciones y técnicas empleadas para expresar el paralelismo potencial y para resolver los problemas de comunicación y sincronización resultantes. La programación concurrente proporciona una abstracción sobre la que estudiar el paralelismo sin tener en cuenta los detalles de implementación. Esta abstracción ha demostrado ser muy útil en la escritura de programas claros y correctos empleando las facilidades de los lenguajes de programación modernos.

Existen dos formas de concurrencia:

Concurrencia implícita: Es la concurrencia interna al programa, por ejemplo, cuando un programa contiene instrucciones independientes que se pueden realizar en paralelo, o existen operaciones de Entrada/Salida que se pueden realizar en paralelo con otros programas en ejecución. Está relacionada con el paralelismo hardware.(Albuja, 2011, p. 2)

Concurrencia explícita: Es la concurrencia que existe cuando el comportamiento concurrente es especificado por el diseñador del programa. Está relacionada con el paralelismo software. Características Principales. (Albuja, 2011, p. 2)

Los procesos concurrentes tienen las siguientes características:

- *Indeterminismo*: Las acciones que se especifican en un programa secuencial tienen un orden total, pero en un programa concurrente el orden es parcial, ya que existe una incertidumbre sobre el orden exacto de ocurrencia de ciertos sucesos, (Anónimo, s.n., parr. 1)esto es, existe un indeterminismo en la ejecución. De esta forma si se ejecuta un programa concurrente varias veces puede producir resultados diferentes partiendo de los mismos datos. (Rodríguez, n.d., p. 10) (Anónimo, s.n., parr. 1)

- *Interacción entre procesos*: Los programas concurrentes implican interacción entre los distintos procesos que los componen:(Rodríguez, n.d., p. 2) (Anónimo, s.n., parr. 6)

  - Los procesos que comparten recursos y compiten por el acceso a los mismos.
  - Los procesos que se comunican entre sí para intercambiar datos.

En ambas situaciones se necesita que los procesos sincronicen su ejecución, para evitar conflictos o establecer contacto para el intercambio de datos. La interacción entre procesos se logra mediante variables compartidas o bien mediante el paso de mensajes. Además, la interacción puede ser explícita, si aparece en la descripción del programa, o implícita, si aparece durante la ejecución del programa. (Rodríguez, n.d., p. 13) (Anónimo, s.n., parr. 8)

- *Gestión de recursos:* Los recursos compartidos necesitan una gestión especial.

Un proceso que desee utilizar un recurso compartido debe solicitar dicho recurso, esperar a adquirirlo, utilizarlo y después liberarlo. Si el proceso solicita el recurso, pero no puede adquirirlo en ese momento, es suspendido hasta que el recurso está disponible. (Rodríguez, n.d., p. 16-18) La

gestión de recursos compartidos es problemática y se debe realizar de tal forma que se eviten situaciones de retraso indefinido (espera indefinidamente por un recurso) y de deadlock (bloqueo indefinido o abrazo mortal).

- *Comunicación*: La comunicación entre procesos puede ser síncrona, cuando los procesos necesitan sincronizarse para intercambiar los datos, o asíncrona, cuando un proceso que suministra los datos no necesita esperar a que el proceso receptor los recoja, ya que los deja en un buffer de comunicación temporal. (Anónimo, s.n., parr. 8-10) (Rodríguez, n.d., p. 13)

2.2. Iluminación y Sombreado

2.2.1. Introducción

Desde una perspectiva física, una superficie puede emitir luz por su propia emisión, como focos de luz, o reflejar luz de otras superficies que la iluminan. Algunas superficies pueden reflejar y emitir luz. El color que se ve en un punto de un objeto está determinado por las múltiples interacciones entre las fuentes de luz y superficies reflectivas. Este es un proceso recursivo.

El problema consiste de dos aspectos:

1. Modelar las fuentes de luz en una escena.

2. Construir un modelo de reflexión que trate con las interacciones entre materiales y luz.

Para comprender el proceso de iluminación, se puede comenzar siguiendo los rayos de luz de un punto fuente, donde el observador ve solamente la luz que emite la fuente y que llega a los ojos; probablemente a lo largo de complejos caminos y múltiples interacciones con objetos en la escena (Sandoval, 2013, parr. 2), como se muestra en la **Figura 4-2.**

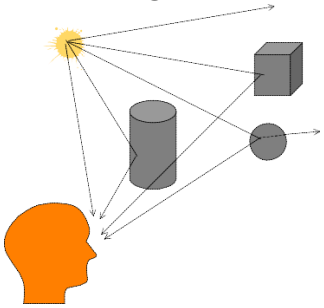


Figura 4-2: Proceso de iluminación de objetos

Fuente: (Weitzenfeld, n.d., p. 2) Gráfica: Iluminación y sombreado

- Si un rayo de luz entra al ojo directamente de la fuente se verá el color de la fuente.
- Si un rayo de luz pega en una superficie que es visible al observador, el color visto se basará en la interacción entre fuente y el material de la superficie: se verá el color de la luz reflejado de la superficie a los ojos. (Weitzenfeld, n.d., p. 2)

En término de gráfica por computadora se reemplaza el observador por el plano de proyección como se ve en la siguiente **Figura 5-2.**

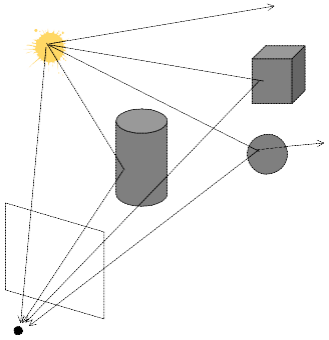


Figura 5-2: Plano de proyección de los gráficos por computadora

Fuente: (Weitzenfeld, n.d., p. 2) Gráfica: Iluminación y sombreado

La ventana de recorte en este plano se mapea a la pantalla. El recorte del plano de proyección y su mapeo a la pantalla significa un número particular de pixeles de despliegue. El color de la fuente de luz y las superficies determinan el color de uno o más pixeles en el frame buffer.

Se debe considerar solo aquellos rayos que dejan las fuentes y llegan al ojo del observador, el COP, después de pasar por el rectángulo de recorte. Nótese que la mayoría de los rayos que dejan la fuente no contribuyen a la imagen y por lo tanto no son de interés aquí. (Weitzenfeld, n.d., p. 2-4)

La naturaleza de interacción entre los rayos y las superficies determina si un objeto aparece rojo o rosado, claro u obscuro, reluciente o no. Cuando la luz da en una superficie, parte se absorbe y parte se refleja.

✓ Si la superficie es **opaca**, reflexión y absorción significa de toda la luz quede en la superficie.
✓ Si la superficie es **translucida**, parte de la luz será transmitida a través del material y podrá luego interactuar con otros objetos.

Un objeto iluminado por la luz blanca se ve rojo porque absorbe la mayoría de la luz incidente, pero refleja luz en el rango rojo de frecuencias. Un objeto relumbrante se ve así porque su superficie es regular, al contrario de las superficies irregulares.

El sombreado de los objetos también depende de la orientación de las superficies, caracterizado por el vector normal a cada punto (Weitzenfeld, n.d., p. 3). Las interacciones entre luz y materiales se pueden clasificar en tres grupos, como se ve en la siguiente **Figura 6-2.**

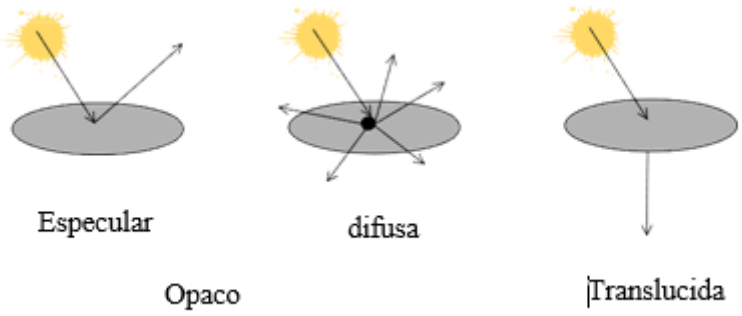


Figura 6-2: Sombreado de objetos dependiente de su orientación

Fuente: (Weitzenfeld, n.d., p. 3) Gráfica: Iluminación y sombreado

Superficies especulares (espejos) - se ven relumbrantes porque la mayoría de la luz reflejada ocurre en un rango de ángulos cercanos al ángulo de reflexión. Espejos son superficies especulares perfectas. La luz del rayo de luz entrante puede absorberse parcialmente, pero toda la luz reflejada aparece en un solo ángulo, obedeciendo la regla que el ángulo de incidencia es igual al ángulo de reflexión. (Weitzenfeld, n.d., p. 3)

Superficies Difusas - se caracterizan por reflejar la luz en todas las direcciones. Paredes pintadas con mate son reflectores difusos. Superficies difusas perfectas dispersan luz de manera igual en todas las direcciones y tienen la misma apariencia a todos los observadores. (Weitzenfeld, n.d., p. 3)

Superficies translucidas - permiten que parte de la luz penetre la superficie y emerja de otra ubicación del objeto. El proceso de **refracción** caracteriza al vidrio y el agua. Cierta luz incidente puede también reflejarse en la superficie. (Weitzenfeld, n.d., p. 3)

2.2.2. Fuentes de Luz

La luz puede dejar una superficie mediante dos procesos fundamentales:

- Emisión propia

- Reflexión

Normalmente se piensa en una fuente de luz como un objeto que emite luz solo mediante fuentes de energía interna, sin embargo, una fuente de luz, como un foco, puede reflejar alguna luz incidente a esta del ambiente. Este aspecto no será tomado en cuenta en los modelos más sencillos. (Weitzenfeld, n.d., p. 4)

Si se considera una fuente como en la **Figura 7-2**, se lo puede ver como un objeto con una superficie.

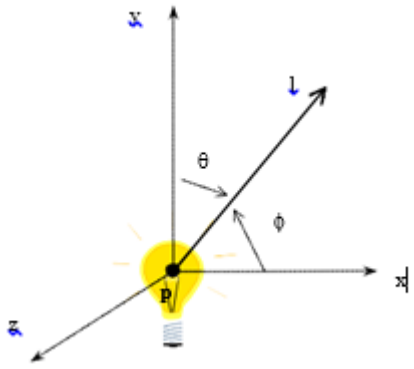


Figura 7-2: Fuente de luz de un objeto

Fuente: (Weitzenfeld, n.d., p. 4) Gráfica: Iluminación y sombreado

Cada punto (x, y, z) en la superficie puede emitir luz que se caracteriza por su dirección de emisión (φ, θ) y la intensidad de energía emitida en cada frecuencia λ. Por lo tanto, una fuente de luz general se puede caracterizar por la **función de iluminación** *I* (x, y, z, φ, θ, λ) de seis variables. Nótese que se requiere dos ángulos para especificar una dirección, y que se asume que cada frecuencia puede considerarse de manera independiente (Weitzenfeld, n.d., p. 4). Desde la perspectiva de una superficie iluminada por esta fuente, se puede obtener la contribución total de la fuente al integrar sobre la superficie de la fuente, un proceso que considera los ángulos de emisión que llegan a la superficie bajo consideración y que debe considerar también la distancia entre la fuente y la superficie, en la **Figura 8-2** se puede apreciar la iluminación del objeto.

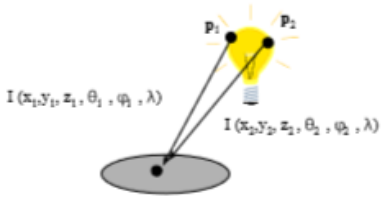


Figura 8-2: Iluminación de un objeto

Fuente: (Weitzenfeld, n.d., p. 4) Gráfica: Iluminación y sombreado

Para una fuente de luz distribuida, como un foco de luz, la evaluación de este integral es difícil, usando métodos analíticos I numéricos. A menudo, es más fácil modelar la fuente distribuida con polígonos, cada una de las cuales es una fuente simple, o aproximando a un conjunto de fuentes de punto. (Weitzenfeld, n.d., pp. 2–3)

Se considerarán cuatro tipos básicos de fuentes, que serán suficientes para generar las escenas más sencillas:

1. Luz ambiente.
2. Fuentes de punto.
3. Spotlights (Luces direccionales).
4. Luces distantes.

2.2.3. Fuentes de Color

No solamente las fuentes de luz emiten diferentes cantidades de luz en diferentes frecuencias, pero también sus propiedades direccionales varían con la frecuencia. Por lo tanto, un modelo físicamente correcto puede ser muy complejo. Para la mayoría de las aplicaciones, se puede modelar fuentes de luz en base a tres componentes primarios, RGB, y puede usar cada uno de los tres colores fuentes para obtener el componente de color correspondiente que un observador humano vería. (Weitzenfeld, n.d., pp. 2–3)

Se describe una fuente mediante una función de **intensidad** o **luminancia** de tres componentes:

$$I = \begin{bmatrix} I_T \\ I_g \\ I_b \end{bmatrix}$$

Cada uno de los componentes es la intensidad de los componentes rojo, verde y azul independientes. Como las computaciones de color involucran tres cálculos similares pero independientes, se presentarán ecuaciones escalares sencillas, con el entendimiento de que pueden representar cualquiera de los tres colores. (Weitzenfeld, n.d., pp. 2–3)

2.2.4. Luz Ambiente

En algunos cuartos, las luces se diseñan y ubican para proveer iluminación uniforme en el cuarto. Tal iluminación se logra mediante fuentes grandes con difusores cuyo propósito es esparcir la luz en todas las direcciones. Se puede crear una simulación precisa de tal iluminación, modelando todas las fuentes distribuidas, y luego integrando la iluminación de estas fuentes en cada punto de una superficie reflectora. Hacer tal modelo y generar la escena sería una tarea formidable para un sistema gráfico, especialmente si se desea ejecución en tiempo real (Weitzenfeld, n.d., p. 5). De manera alternativa, se puede ver el efecto deseado de las fuentes: lograr un nivel de luz uniforme en el cuarto. Esta iluminación uniforme se llama **luz ambiente**. Si se sigue este segundo enfoque, se puede postular una intensidad ambiente en cada punto del ambiente. Por lo tanto, iluminación ambiente se caracteriza por una **intensidad Ia**, que es idéntica en cada punto de la escena.

La fuente de ambiente tiene tres componentes:

$$I_a = \begin{bmatrix} I_{ax} \\ I_{ag} \\ I_{ab} \end{bmatrix}$$

Aunque cada punto en la escena recibe la misma iluminación ambiente de **Ia**, cada superficie la refleja de manera diferente.

$$I_a(p) = I_a$$

2.2.5. Fuentes

Una **fuente de punto** ideal emite luz de manera igual en todas las direcciones. Se caracteriza una fuente ideal ubicada en un punto *p0* por un vector color de tres componentes:

$$I(p_0) = \begin{bmatrix} I_{r\,(p_0)} \\ I_g(p_0) \\ I_b(p_0) \end{bmatrix}$$

La intensidad de iluminación recibida de una fuente de punto es proporcional al inverso del cuadrado de la distancia entre la fuente y la superficie. Por lo tanto, en un punto *p*, como se ve en la siguiente **Figura 9-2**.

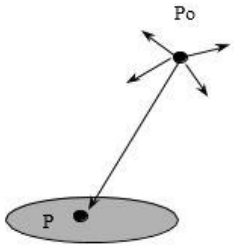


Figura 9-2: Intensidad de iluminación

Fuente: (Weitzenfeld, n.d., p. 5) Gráfica: Iluminación y sombreado

El uso de fuentes de punto en la mayoría de las aplicaciones se determina más por su facilidad de uso que por su similitud a la realidad física. Escenas generadas solo con fuentes de punto tienden a tener un alto contraste; los objetes se ven brillantes u obscuros. En el mundo real, el tamaño grande de las fuentes de luz contribuye a las escenas más suaves, como se ve en la siguiente **Figura 10-2.**

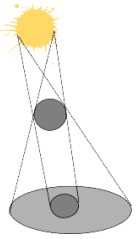


Figura 10-2: Contraste de la iluminación de objetos con una sola fuente

Fuente: (Weitzenfeld, n.d., p. 6) Gráfica: Iluminación y sombreado

Algunas áreas están totalmente en la sombra, o en la **umbra**, mientras que otras están parcialmente en la sombra, o **penumbra**. Se puede mitigar el efecto de alto contraste al añadir luz ambiente a una escena.

La distancia también contribuye al efecto abrupto en las fuentes de punto. Aunque el término inverso al cuadrado de la distancia es correcto para fuentes de punto, en la práctica normalmente se reemplaza por un término del tipo $(a+bd+cd2)$-1, donde *d* es la distancia y las constantes *a*, *b*, y *c* se escogen para suavizar la luz, lo que significaría una ecuación de la forma:

$$I_p(p, p_0) = \frac{1}{a + b|p - p_0| + c|p - p_0|^2} I(P_0)$$

Nótese que, si la fuente de luz está lejana de las superficies en la escena, la intensidad de la luz de la fuente será suficientemente uniforme que el término de distancia será constante sobre las superficies.

2.3. Gradientes Vectoriales

2.3.1. Introducción

Las magnitudes como el peso y la temperatura consisten en un número, como 15 grados o 1.1 kilogramos. Los científicos llaman a estas magnitudes escalares. Las medidas como la velocidad y la fuerza, por otra parte, son vectores, y tienen dos datos: una magnitud y una dirección. Por ejemplo, el reporte del clima dice que el viento sopla del este a siete kilómetros por hora. Los científicos indican a los vectores con flechas, ya que las flechas tienen una longitud (que indica la magnitud o intensidad de la medida) y apuntan en una dirección específica. El gradiente es un vector que resulta de una operación delta en una superficie. Si la superficie es plana, el gradiente es cero, su forma no cambia. Si la superficie tiene una colina, el gradiente apunta hacia arriba. Cuando la superficie tiene depresiones y valles, el gradiente apunta hacia abajo. Cuanto más grande sean las elevaciones o depresiones, mayor será la magnitud del gradiente. La **Figura 11-2,** Representa los gradientes vectoriales.

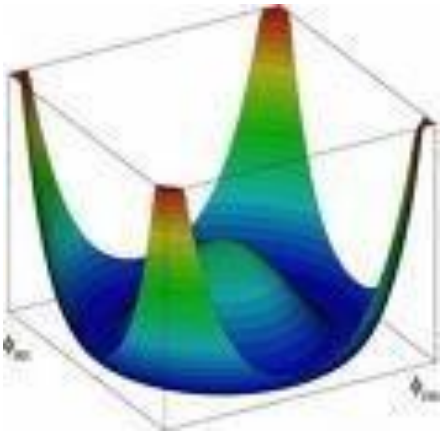


Figura 11-2: Gradientes Vectoriales

Fuente: (Silvera, 2015, para. 1) La simetría CP y otros aspectos de la Física

2.3.2. Definición

Normalmente el gradiente denota una trayectoria en el espacio por la cual se aprecia una variación de una determinada propiedad o magnitud física. (Esteves, 2009, parras. 1–2) En otros contextos se usa informalmente gradiente, (Esteves, 2009, parras. 1–2) para indicar la existencia de gradualidad o variación gradual en determinado aspecto, no necesariamente relacionado con la distribución física de una determinada magnitud o propiedad como se puede apreciar en la **Figura 12-2.**

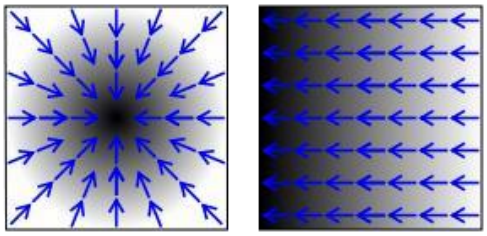


Figura 12-2: Representación de gradientes

Fuente: (Troya, 2016, parr. 3) Algoritmo de minimización de función convexa del coste

En esta **Figura 12-2**, el campo escalar se aprecia en blanco y negro, representando valores bajos o altos respectivamente, y el gradiente correspondiente se aprecia por flechas azules. (Avendaño, 2010, parras. 1–3) El gradiente de un campo escalar, que sea diferenciable en el entorno de un punto, es un vector definido como el único que permite hallar la derivada direccional en cualquier dirección como: (Avendaño, 2010, parr. 4)

$$\frac{\partial_{\emptyset}}{\partial_n} = (grad\emptyset).\hat{n}$$

siendo $\hat{n}$ un vector unitario y $\partial_{\emptyset}/\partial_n$ la derivada direccional de $\emptyset$ en la dirección de $\hat{n}$,que informa de la tasa de variación del campo escalar al desplazarnos según esta dirección: (Avendaño, 2010, parr. 5)

$$\frac{\partial_{\emptyset}}{\partial_n} \equiv \lim_{\in \to 0} \frac{\emptyset(\vec{r} - \in \hat{n}) - \emptyset(\vec{r})}{\in}$$

Una manera semejante de definir el gradiente es como el único vector que, multiplicado por cualquier desplazamiento infinitesimal, da el diferencial del campo escalar: (Avendaño, 2010, parr. 6)

$$d\emptyset = \emptyset(\vec{r} + d\vec{r}) - \emptyset(\vec{r}) = \nabla\emptyset.d\vec{r}$$

Así pues, de la definición anterior, el gradiente está distinguido de forma unívoca. El gradiente se expresa alternativamente mediante el uso del operador nabla: (Avendaño, 2010, parr. 7)

$$grad\emptyset = \nabla\emptyset$$

Interpretación del gradiente

De forma geométrica el gradiente es un vector que se encuentra normal (perpendicular) a la curva de nivel en el punto que se está estudiando, llámese (x, y), (x, y, z), (tiempo, temperatura), etcétera. Algunos ejemplos son:

- Considere una habitación en la cual la temperatura se define a través de un campo escalar, de tal manera que en cualquier punto (x, y, z), la temperatura es $\emptyset(x, y, z)$. Asumiremos que la temperatura no varía con respecto al tiempo. Siendo esto así, para cada punto de la habitación, el gradiente en ese punto nos dará la dirección en la cual la temperatura aumenta más rápido. La magnitud del gradiente nos dirá cuán rápido aumenta la temperatura en esa dirección.

- Considere una montaña en la cual su altura en el punto (x, y) se define como $H(x, y)$. El gradiente de H en ese punto estará en la dirección para la que hay un mayor grado de inclinación. La magnitud del gradiente nos mostrará cuán empinada se encuentra la pendiente.

Propiedades

Las más relevantes que se pude determinar del gradiente vectorial es que se encarda de verificar que:

- Es ortogonal a las superficies equiescalares, definidas por $\emptyset$ =cte. (Avendaño, 2010, parr. 10) (Esteves et al., 2009, parr. 21)

- Apunta en la dirección en que la derivada direccional es máxima. (Avendaño, 2010, parr. 10) (Esteves et al., 2009, parr. 21)
- Su módulo es igual a esta derivada direccional máxima. (Avendaño, 2010, parr. 10) (Esteves et al., 2009, parr. 21)

- Se anula en los puntos estacionarios (máximos, mínimos y puntos de silla). (Avendaño, 2010, parr. 10) (Esteves et al., 2009, parr. 21)

- El campo formado el gradiente en cada punto es siempre ir rotacional, (Avendaño, 2010, parr. 10) (Esteves et al., 2009, parr. 21) esto es:

$$\nabla\, x\, (\nabla_{\emptyset}) \equiv \vec{0}$$

Expresión en diferentes sistemas de coordenadas

A partir de su definición puede hallarse su expresión en diferentes sistemas de coordenadas. En coordenadas cartesianas, su expresión es simplemente. (Avendaño, 2010, parr. 11)

$$\nabla_{\emptyset} = \frac{\partial_{\emptyset}}{\partial_x}\hat{x} + \frac{\partial_{\emptyset}}{\partial_x}\hat{y} + \frac{\partial_{\emptyset}}{\partial_z}\hat{z}$$

En un sistema de coordenadas ortogonales, el gradiente requiere los factores de escala, mediante la expresión. (Avendaño, 2010, parr. 12)

$$\nabla_{\emptyset} = \frac{1}{h_1}\frac{\partial_{\emptyset}}{\partial_{q_1}}\widehat{q_1} + \frac{1}{h_2}\frac{\partial_{\emptyset}}{\partial_{q_2}}\widehat{q_2} + \frac{1}{h_3}\frac{\partial_{\emptyset}}{\partial_{q_3}}\widehat{q_3}$$

Para coordenadas cilíndricas ($h_p = h_z = 1, h_\varphi = \rho$) resulta (Avendaño, 2010, parr. 13)

$$\nabla_{\emptyset} = \frac{\partial_{\emptyset}}{\partial_\rho}\hat{\rho} + \frac{1}{\rho}\frac{\partial_{\emptyset}}{\partial_\varphi}\hat{\varphi} + \frac{\partial_{\emptyset}}{\partial_z}\hat{z}$$

y para coordenadas esféricas ($h_r \quad h_\varphi = r\sin\theta = 1, h_\theta = \mathrm{r}$) (Avendaño, 2010, parr. 14)

$$\nabla_{\emptyset} = \frac{\partial_{\emptyset}}{\partial_r}\hat{r} + \frac{1}{r}\frac{\partial_{\emptyset}}{\partial_\theta}\hat{\theta} + \frac{1}{r\sin\theta}\frac{\partial_{\emptyset}}{\partial_\varphi}\hat{\varphi}$$

Gradiente de un campo vectorial

En un espacio euclídeo, el concepto de gradiente también puede extenderse al caso de un campo vectorial, $\vec{F}$ siendo el gradiente de un tensor que da el diferencial del campo al realizar un desplazamiento (Esteves et al., 2009, parr. 27) (Avendaño, 2010, parr. 11)

$$d\vec{F} = \vec{F}(\vec{r} + d\vec{r}) - \vec{F}(\vec{r}) = \left(\nabla\vec{F}\right).d\vec{r}$$

Este tensor podrá representarse por una matriz (3x3), que en coordenadas cartesianas está

formada por las tres derivadas parciales de las tres componentes del campo vectorial. (Esteves et al., 2009, parr. 28) (Avendaño, 2010, parr. 12)

Curvas de Nivel

El sistema de representación de curvas de nivel consiste en cortar la superficie del terreno mediante un conjunto de planos paralelos entre sí, separados una cierta distancia unos de otros. (AristaSur, 2015, parras. 1) Cada plano corta al terreno formando una figura (plana) que recibe el nombre de **curva de nivel** o **isohipsa**. La proyección de todas estas curvas de nivel sobre un plano común (el mapa) da lugar a la representación buscada. (AristaSur, 2015, parras. 1–3) (Esteves et al., 2009, parr. 23-28)

En la **Figura 13-2** se ve la construcción para representar mediante curvas de nivel una montaña. La montaña es cortada mediante planos paralelos separados una cierta distancia que se llama **equidistancia entre curvas de nivel**. (AristaSur, 2015, parras. 2-3)

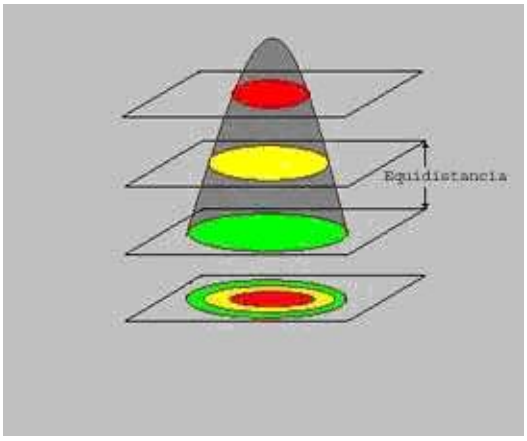


Figura 13-2: Planos equidistantes formando curvas de nivel

Fuente: (AristaSur, 2015, parra. 3) Qué son las curvas de nivel de un mapa tipográfico

Las intersecciones de los planos con la superficie de la montaña determinan un conjunto de secciones que son proyectadas sobre el plano inferior, que representa al mapa. (AristaSur, 2015, parras. 4) (Esteves et al., 2009, parr. 23-28) El resultado final que observaremos sobre el mapa es algo como se ve en la **Figura 14-2.** (AristaSur, 2015, parra. 4)

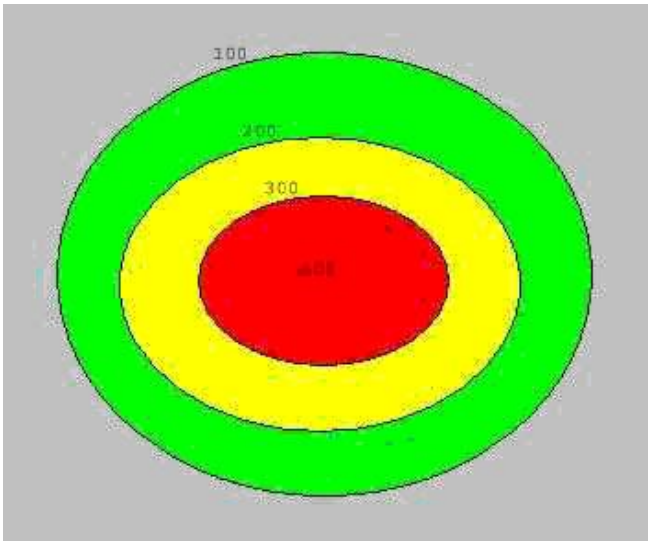


Figura 14-2: Cortes de los planos proyectados sobre el papel

Fuente: (AristaSur, 2015, parr. 5) Qué son las curvas de nivel de un mapa tipográfico

Al observar la figura nos puede quedar la duda sobre qué secciones están por encima de otras. Es decir, ¿está realmente la sección roja por encima de la amarilla y de la verde? (AristaSur, 2015, parra. 6) (Esteves et al., 2009, parr. 23-28) , el problema anterior se resuelve fácilmente si para cada sección indicamos su altura con respecto a un plano de referencia, y como tal plano se toma el nivel del mar. De este modo la sección verde se ha obtenido cortando la montaña mediante un plano paralelo al nivel del mar y una altura (o nivel) de 100 metros con respecto a aquél (Esteves et al., 2009, parr. 25-28). La sección amarilla se ha obtenido mediante la intersección con un plano a 200 metros sobre el nivel del mar (s.n.m.). Y la sección roja con un plano a 300 metros s.n.m. Para cada curva de nivel indicaremos esa altitud y le denominaremos **cota**. (AristaSur, 2015, parras. 6-8) (Esteves et al., 2009, parr. 25-28)

La equidistancia entre curvas de nivel se puede deducir ahora con facilidad para el ejemplo dado: 100 metros.

En la **Figura 15-2** se ve cómo se efectúa la construcción de curvas de nivel de una depresión, que es el caso opuesto al monte de la **Figura 14-2.**

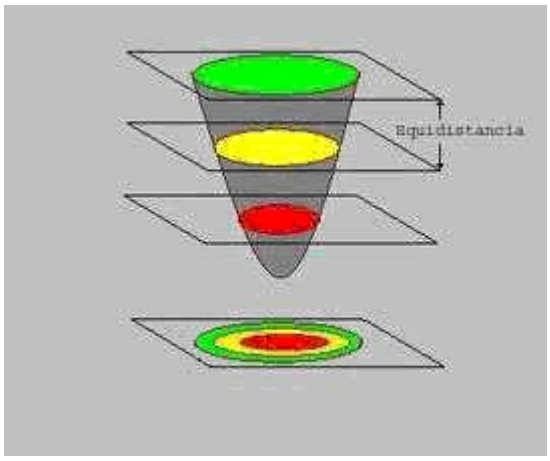


Figura 15-2: Cortes de los planos proyectados sobre una depresión

Fuente: (AristaSur, 2015, parr. 9) Qué son las curvas de nivel de un mapa tipográfico

Puede observarse que el procedimiento (AristaSur, 2015, parr. 10) a seguir es exactamente el mismo y que se obtiene la misma representación, de la **Figura 16-2** (Esteves et al., 2009, parr. 25-30)

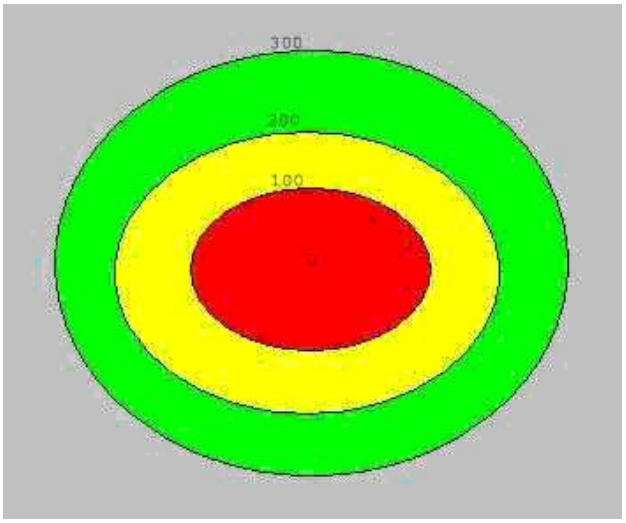


Figura 16-2: Curvas de nivel de la depresión

Fuente: (AristaSur, 2015, parr. 10) Qué son las curvas de nivel de un mapa tipográfico

Sin embargo, la acotación de las curvas de nivel no deja lugar a dudas. Podemos observar que las curvas de mayor cota encierran a las curvas de cota menor, señal inequívoca de una depresión en el terreno. En un monte ocurre justo lo contrario, las curvas de nivel de menor cota encierran a las de cota mayor. (AristaSur, 2015, parras. 10-11) (Esteves et al., 2009, parras. 25-30)

Las curvas de nivel verifican las siguientes **premisas** de manera general como lo expone en su sitio web AristaSur (2015) así como también Martínez (S.N.):

- Las curvas de nivel no se cortan ni se cruzan (sólo ocurre esto cuando queremos representar una cueva o un saliente de roca). (Martinez, n.d., parr. 11)

- Las curvas de nivel se acumulan en las laderas más abruptas y están más espaciadas en las laderas más suaves. (Martinez, n.d., parr. 12)

- La línea de máxima pendiente entre dos curvas de nivel es aquella que las une mediante la distancia más corta. (Martinez, n.d., parr. 13)

En la **Figura 17-2** tenemos dos itinerarios para alcanzar una cumbre desde dos puntos A y B. Desde el punto A (itinerario rojo) es más largo que desde el punto B (recorrido azul (AristaSur, 2015, parras. 12-15) .Sin embargo, el itinerario azul es mucho más duro ya que las curvas de nivel se hallan más apretadas o, si se prefiere, el camino atraviesa las curvas de nivel en menos espacio.

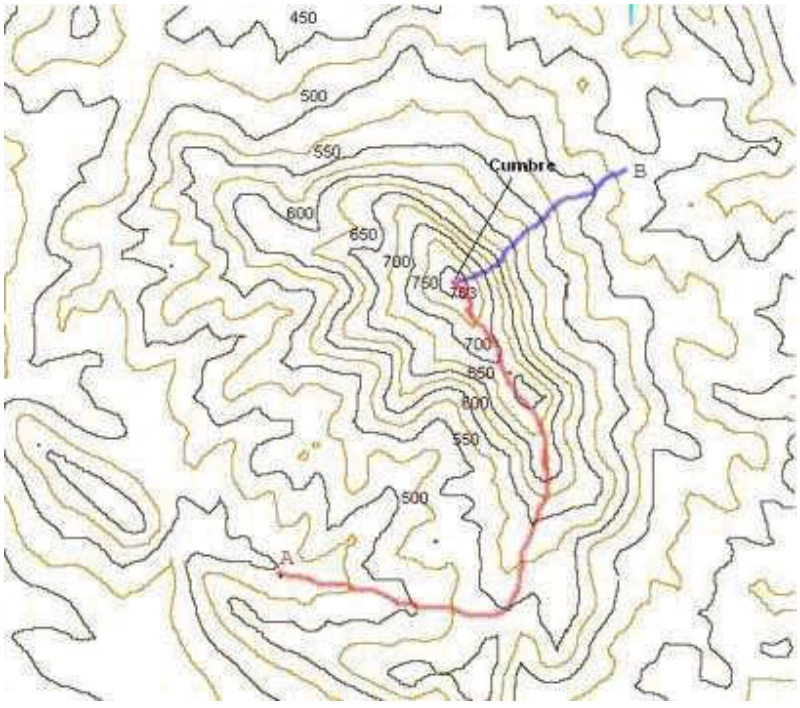


Figura 17-2: Diferentes itinerarios a cumbre

Fuente: (AristaSur, 2015, parra. 17) Qué son las curvas de nivel de un mapa tipográfico

Equidistancia entre curvas de nivel

La distancia entre los diversos planos imaginarios que cortan el terreno es siempre la misma para un mapa dado y se llama **equidistancia entre curvas de nivel**. (Martinez, n.d., parras. 1-4) (AristaSur, 2015, parras. 12-15)

En el plano anterior **(Figura 17-2)** (AristaSur, 2015, parras. 14) la equidistancia entre curvas de nivel es de 25 metros las mismas que se pueden apreciar en la **Figura 18-2.** Se puede notar que se están utilizando dos colores para poder contar mejor las curvas de nivel. Así mismo, las líneas más oscuras aparecen cada 50 metros, y se puede apreciar que entre dos de ellas contiguas surge una línea más clara. (AristaSur, 2015, parras. 15) En cualquier caso, entre dos curvas de nivel tendremos una diferencia de altitud de 25 metros. A las líneas más oscuras se les suele llamar **curvas de nivel maestras**.

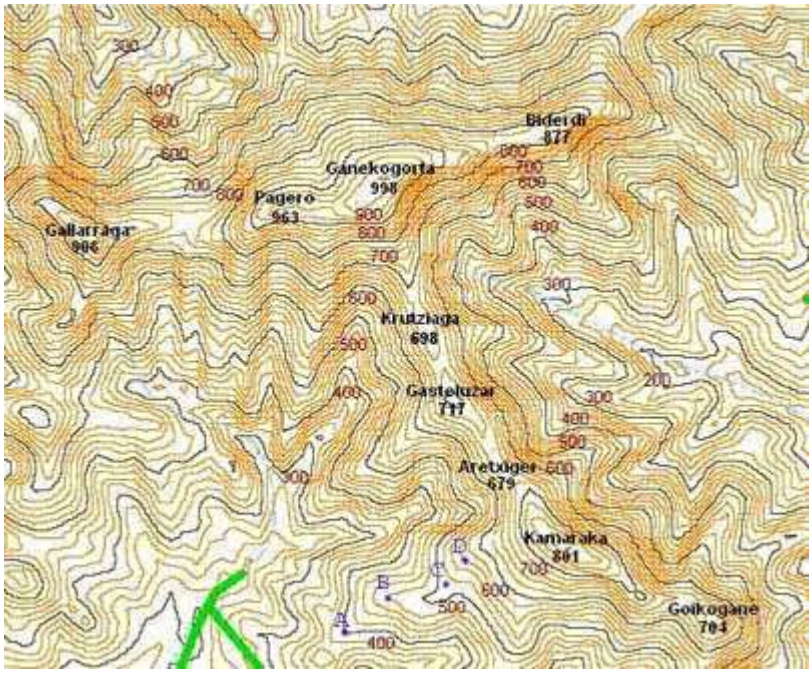


Figura 18-2: Equidistancia entre curvas de nivel

Fuente: (AristaSur, 2015, parra. 20) Qué son las curvas de nivel de un mapa tipográfico

Se puede apreciar en plano un mapa con equidistancia entre curvas de nivel de 20 metros. (AristaSur, 2015, parras. 16) Las curvas maestras surgen en tono oscuro cada 100 metros. Entre dos

curvas maestras consecutivas tenemos, por tanto, cuatro curvas de nivel en tono más claro. Entre dos curvas cualesquiera existe una diferencia de nivel de 20 metros. (AristaSur, 2015, parras. 17-21)

Curva Bézier

Sean $\{(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)\}$ (n+1) puntos soporte y sea $t \in [0,1]$ entonces la función paramétrica P(t)=(x(t), y(t)) es la curva conocida como Bezier, donde

$$x(t) = \sum_{i=0}^{n} x_i B_{i,n}(t),$$

$$y(t) = \sum_{i=0}^{n} y_i B_{i,n}(t),$$

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

y teniendo en cuenta que

$$\binom{n}{i} = \frac{n!}{(n-i)!.i!}.$$

Estas funciones pueden ser implementadas en el computador con los siguientes procedimientos y funciones.

Cota de un punto

Cada punto de un mapa se instala a una altitud precisa que se denomina **cota**. Cota de un punto es la longitud vertical que lo aparta del plano de comparación, regularmente el nivel del mar. (AristaSur, 2015, parras. 18)

En AristaSur (2015) menciona que en vista al plano anterior **Figura 18-2** podemos ver que la cota del punto A es 400 metros, ya que se ubica sobre la curva maestra de 400 metros. La cota del punto B es 480 metros, pues se encuentra a cuatro curvas de nivel por encima de la curva maestra de 400 metros (400 + 4 x 20 = 480 m). (AristaSur, 2015, parra. 19) Igualmente, se puede determinar su cota observando que está en la curva de nivel anterior a la curva maestra de 500 metros (500 - 20 = 480 m). El punto C se encuentra entre las curvas de nivel 500 y 520 metros. Su cota estará pues comprendida entre estos dos valores, pero no lo podemos saber con certeza. En tal caso se puede tomar como valor aproximado el valor medio, 510 metros. Finalmente, la cota del punto D es 560 metros (500 + 3 x 20 = 560 m). (AristaSur, 2015, parras. 18-19)

Curvas de nivel auxiliares

En las regiones muy planas hallamos las curvas de nivel fuertemente distanciadas por lo que tan solo tendremos información relativa a la topografía del terreno. (AristaSur, 2015, parra. 21)

Sospechemos, por ejemplo, un plano con una equidistancia entre curvas de nivel de 25 metros. Cualquier accidente que sea de menor altura sobre el terreno que 25 m quedará sin representar. Pero bastará una franja rocosa vertical de, por ejemplo, 4 metros, para que nos resulte infranqueable. (AristaSur, 2015, parra. 22)

Estas dos situaciones (AristaSur, 2015, parras. 22-23)nos empujan a aumentar el número de curvas de nivel en ciertas zonas de los mapas añadiendo curvas de nivel de menor equidistancia y que se dibujan entre dos curvas de nivel consecutivas. Reciben, estas curvas, el nombre de **curvas de nivel auxiliares**. (AristaSur, 2015, parra. 23)

Las curvas de nivel auxiliares (AristaSur, 2015, parra. 24) suelen estar representadas por trazos discontinuos como se puede apreciar en la **Figura 19-2**. En los mapas de equidistancia entre curvas de nivel de 20m, aparecen entre curvas de nivel consecutivas con una equidistancia de 10 m. Por tanto, si entre las curvas de nivel de 340 y 360 metros de cota se nos muestra una curva discontinua, sabremos que es una curva de nivel auxiliar de 350 metros. (AristaSur, 2015, parra. 23-24)

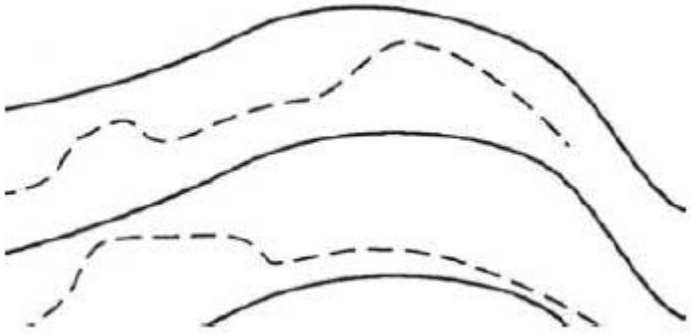


Figura 19-2: Curvas de nivel auxiliares

Fuente: (AristaSur, 2015, parra. 29) Qué son las curvas de nivel de un mapa tipográfico

Sombreado

Consiste en crear unos efectos de sombra e iluminación similar al que originaría un "sol artificial" situado a cierta altitud sobre el relieve. (AristaSur, 2015, parra. 27)

Como conocemos y se afirma en AristaSur (2015) El sombreado no se constituye en una herramienta "cuantitativa" de representación del relieve, Sin embargo, ayuda en gran medida a su entendimiento como superficie tridimensional. (AristaSur, 2015, parra. 28)

Se conoce o existe dos sistemas básicos de sombreado:

- **Sombreado de Pendiente**: Basado en toda la cantidad de luz que receptan las superficies en función de la pendiente propia, estas, cuando están iluminadas por un foco ubicada en la vertical denominada: CENIT. (AristaSur, 2015, parra. 30). La Superficies que se manifiestan planas se exponen con más claridad, aunque, según cómo van generando ganancia de inclinación se tornan más oscuros. Ver la **Figura 20-2.** (AristaSur, 2015, parras. 29-30)

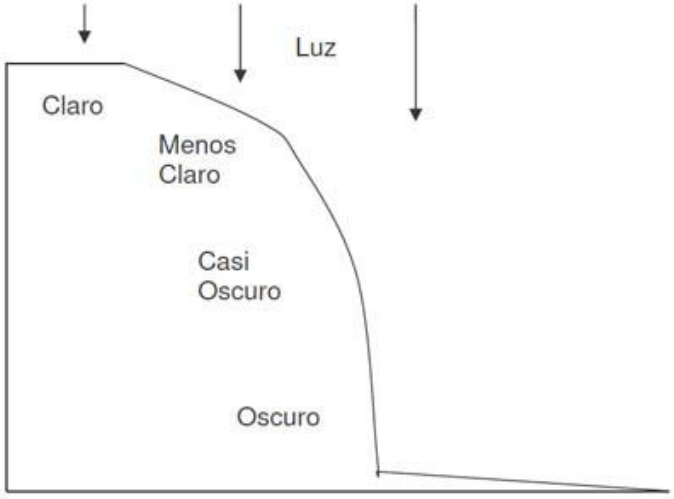


Figura 20-2: Sombreado de pendiente

Fuente: (AristaSur, 2015, parra. 36) Qué son las curvas de nivel de un mapa tipográfico

- **Sombreado Oblicuo**: Denominado aquel sombreado que se crea a partir de cuándo un objeto recibe iluminación por un foco de luz ubicado de manera oblicua con respecto a dicho objeto. (AristaSur, 2015, parr. 31) Ver la **Figura 21-2.**

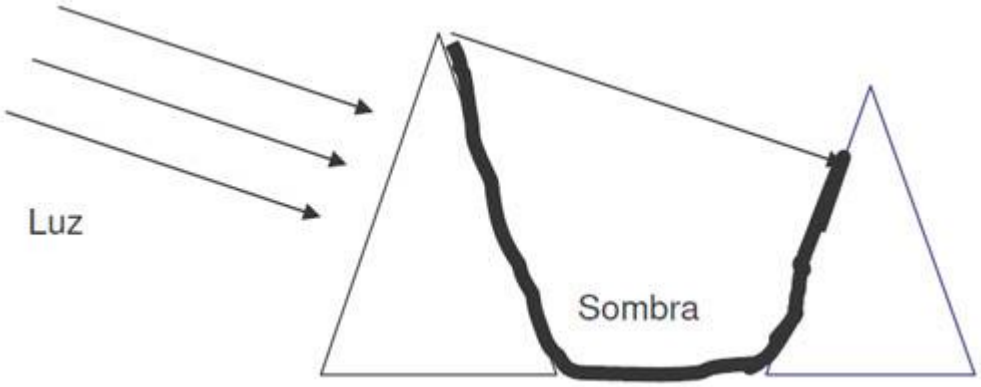


Figura 21-2: Sombreado oblicuo

Fuente: (AristaSur, 2015, parra. 38) Qué son las curvas de nivel de un mapa tipográfico

Este sombreado oblicuo seguramente es el más usado por particulares beneficios y características propias, como, por ejemplo: su interpretación es más sencilla, definitivamente. (AristaSur, 2015, parras. 29-31)El poder utilizarlo implica inicialmente por definir su posición del foco, ya sea; luz imaginario o luz por sol artificial. A si pues, de manera estándar se acostumbra a ubicar en el ángulo superior izquierdo del mapa (al N.W.), a una elevación virtual de 45º sobre el horizonte (plano del papel). De hecho, que el sol jamás logra alcanzar dicha posición en el cielo. (AristaSur, 2015, parr. 31)

Regularmente dentro de los mejores mapas se armoniza el sombreado con las tintas hipsométricas como herramientas accesorias para la comprensión del relieve. (AristaSur, 2015, parras. 31-32)

CAPÍTULO III

3. MARCO METODOLÓGICO

3.1. Investigación Aplicada

3.1.1. Método de Investigación para la resolución de casos

El método de investigación para la resolución de casos es un método que permite tener una independencia de aprendizaje ya que se puede explorar el conocimiento científico, y surge a partir del Aprendizaje Basado en Problemas el cual se originó en la Escuela de Medicina de la Universidad de McMaster, Canadá, hace unos 35 años, con el objetivo de mejorar la formación de los profesionales médicos. (Francisco, 2013)

Este método se puede aplicar de diversas formas como: la utilización de registros de investigación e informaciones necesarias, utilizando las relaciones entre los conocimientos científicos, sociales, humanísticos, medioambientales, políticos y de forma experimental, permitiendo así poder desarrollar prácticas de investigación, manipulación y comunicación de los datos o resultados (Francisco, 2013), permite estudiar a la ves un tema varios temas determinados, fenómenos desde múltiples perspectivas y no desde la influencia de una sola variable, y sobre cada fenómeno permite tener un conocimiento más amplio.

3.1.2. Entrevista

La entrevista es una técnica de recolección de requerimientos, que trata de forma más personal y formal al usuario final, con el principal objetivo de tener información sobre el funcionamiento del producto software a desarrollarse (Romero & Domenech, s.f.).

Esta técnica se utilizará en primera instancia para proceder al desarrollo de los casos de uso mediante los cuales se explicará de mejor manera el funcionamiento a tener el producto software, los mismos que serán utilizados para su posterior desarrollo en base a la planificación desarrollada el cual se puede ver en el **Anexo A**.

3.1.3. Técnica de recolección de requerimientos en base a Casos de Uso.

Los casos de uso como son conocidos como una herramienta en el desarrollo de software, ya que

pertenecen al UML estándar, mediante la utilización de estos diagramas se define de forma clara y precisa cada uno de los casos a implementar en la aplicación, para ello se tiene un diagrama general el mismo que se puede apreciar en la **Figura 1-3.**

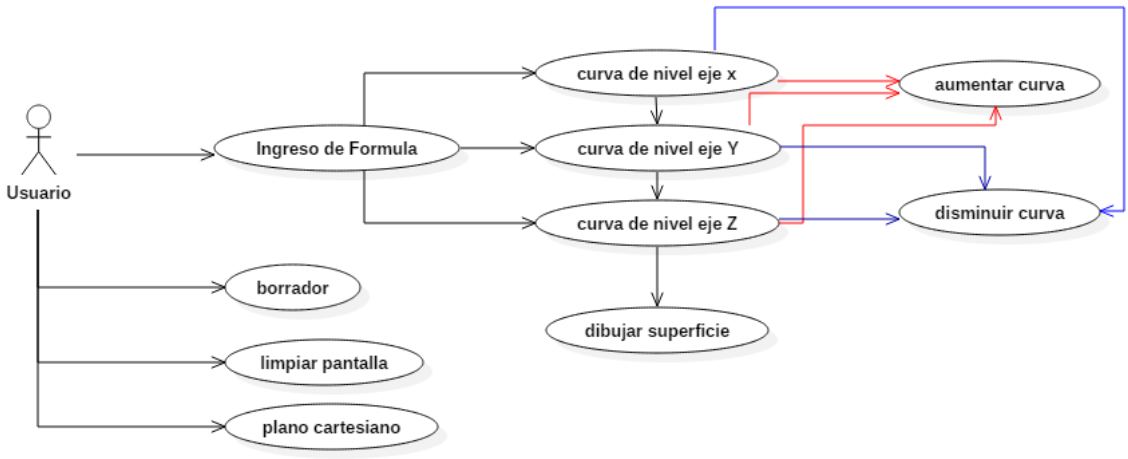


Figura 3.1-3: Diagrama de casos de usos general

3.2. Metodología de desarrollo SCRUM

Esta metodología de desarrollo de software se aplica de manera regular permitiendo así mejorar la relación interpersonal entre el desarrollador (es) y el usuario final, promover un trabajo colaborativo entre el equipo, avanzando positivamente en el desarrollo de un proyecto software. (Gallego., 2006) (Merino, 2017, p. 19).

Melé (todos a la vez) como empezó o donde surge Scrum como metodología de desarrollo ágil, inserta intervalos de desarrollo denominados Sprints, para agilizar y simultáneamente hace una retroalimentación entre los involucrados, de los procesos, Dentro del ciclo del proceso de desarrollo scrum es importante saber y comprender las fases que envuelven dicho proceso, teniendo en cuenta que su orden es de suma importancia para culminar con éxito el proyecto de software. (Merino, 2017, p. 19) en la **Tabla 1-3** se describe las fases scrum.

Tabla 1-3: Fases scrum

FASES	REPRESENTACIÓN	DETALLE

Product Backlog	CONCEPTO	✓ Listado de requerimientos. ✓ La lista lo elabora el Product Owner y es la persona encargada de priorizarlas, según la prioridad que clasifique el cliente.
Sprint Backlog	FORMALIZACIÓN DE REQUERIMIENTOS	✓ Asociación de tareas que se realizan en un Sprint. ✓ Se establece en el camino de la planificación y la permanencia de cada uno de los Sprint.
Sprint Planning Meeting	REVISIÓN y RETROALIMENTACIÓN DEL EQUIPO SCRUM	✓ Realizar una reunión entre los involucrados y comprometidos del proyecto para seleccionar de las listas Backlog del producto. ✓ Previa a dicha reunión **e**l Product Owner es quien elaborará el Backlog.
Daily SCRUM		✓ Equipo Scrum. ✓ La reunión no durará más de 15 minutos. ✓ El equipo se auto cuestiona tres aspectos y/o parámetros: **¿Qué se ha hechor?, ¿Qué se hará hoy?, ¿Qué problema hay para realizarlo?**
Sprint Review		✓ Los desarrolladores del equipo Scrum, exhiben el producto entregable (Sprint), que han desarrollado. ✓ El Product Owner analiza y recibe identificación sobre problemas en el proceso, en caso de haberlos. ✓ Tiempo máximo 4 horas.
Sprint Retrospective	CIERRE	✓ Es un debate profesional, donde el equipo de desarrollo sustenta sobre el sprint realizado. ✓ Es donde se examina posibles variantes, buscando mejorar el Sprint entrante, deseando alcanzar su máxima productividad. ✓ Tiempo máximo 3 horas.

Realizado por: García Michael. 2017

Fuente: (Merino, 2017, p. 19) Fases scrum

Gracias a la facilidad y garantías en el proceso de desarrollo de software que brinda Scrum, basados en los requisitos que fueron recolectados y recapturado conjuntamente entre los involucrados del proyecto los mismos que son alojados en el Product Backlog que adjunto en el **Anexo B**. Los roles scrum como Cristian Merino lo describe en la **Tabla 2-3**

Tabla 3.2-3: Roles scrum

ROL	DESCRIPCIÓN
ProductOwner	Dueño del producto, canaliza las necesidades y maximiza el valor

	de las mismas para el negocio.
ScrumMaster	Líder, no jefe, garantiza la correcta aplicación de scrum.
Equipo de Desarrollado	Grupo de 5-9 personas, todos tienen autoridad y toma de decisiones para conseguir el objetivo.
Usuarios	Beneficiado final del Producto
Stakeholders	Son todos los Interesados en el producto que les generará un beneficio. Forman parte en los chequeos del Sprint correspondiente.

Realizado por: García Michael. 2017

Fuente: (Merino, 2017, p. 18) Roles scrum

3.2.1. Planeación y estimación

Con los requerimientos recolectados y una vez realizado el Product Backlog es necesario realizar una planificación del proyecto en el cual se determinará el tiempo de desarrollo del producto software el cual se encuentra en el **Anexo C.**

Historias de Usuario

Son las especificaciones de las funcionalidades que tendrá el producto software con un tiempo estimado y el responsable a cargo de la historia, Le contenido de la tarjeta de historia de usuario, es un identificador de la historia, nombre de la historia, descripción tiempo estimado, prioridad si tiene dependencia o no con otra historia y las pruebas de aceptación de la historia como se puede apreciar en la **Tabla 3-3**, en el presente proyecto cuenta con un total de 10 historias de usuario, las cuales se encuentran desarrolladas en un 100%, las tarjetas se pueden apreciar en el **Anexo D.**

Cada punto estimado equivale a una semana de 40 horas de trabajo, esta estimación se reflejará en puntos en las historias de usuario.

Tabla 3-3: Historia de usuario

Nº: USR-001	**Ingreso de fórmula matemática**		
Como usuario, requiero realizar el ingreso por teclado de una formula, con la finalidad de representarla en el plano X, Y, Z			
Estimación:	1 punto	**Real:**	1 punto
Prioridad:	Alta	**Dependencia:**	Ninguna
Pruebas de Aceptación:			

- ✓ La fórmula es registrada por la aplicación.
- ✓ La fórmula no es registrada por la aplicación

Realizado por: García Michael. 2017

Cada una de las historias de usuario pertenecen a un Sprint y por cada uno de estos se ha realizado la implementación, revisión retrospectiva y liberación del sprint permitiendo de esta forma que al finalizar cada uno de ellos se tenga un entregable funcional para el usuario.

3.2.2. Implementación

Para la implementación de la aplicación se lo realizo en seis sprint los mismos de los cuales se realizó un análisis sobre el avance del desarrollo en base a los valores estimados y los valores reales que se obtuvieron durante el desarrollo del producto software, el cual se puede apreciar en la **Gráfico 1-3.** De los cuales se obtuvieron que de los dos primeros sprint estuvieron dentro del rango estimado de tiempo, mientras que el sprint 3 y 4 se los realizo en menor tiempo, para el sprint 5 se superó el tiempo en un punto, y el ultimo sprint se encontraba dentro del tiempo estimado.

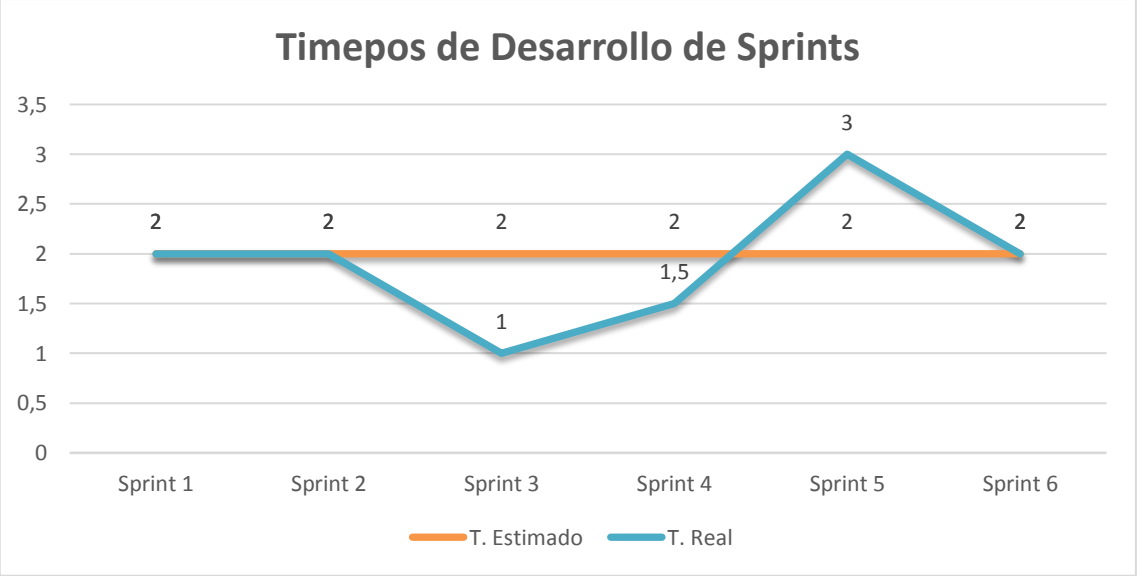


Gráfico 1-3: Tiempos de desarrollo de los sprints

Realizado por: García Michael. 2017

3.2.3. Revisión y retrospectiva

Dentro de las revisiones y retrospectiva de cada uno de los sprint se realizaron cambios pequeños en cuanto a presentación, los mismos que se dieron en el sprint 5, los mismos que se resolvieron de forma rápida permitiendo así que la planificación no se vea muy afectada por dichos cambios.

3.2.4. Liberación

El desarrollo del producto software se lo realizo dentro del tiempo planificado con pequeños cambios, cada uno de los sprint se han ido liberando paulatinamente teniendo entregables funcionales de la aplicación para su respectivo uso, realizando una comprobación de integración y de que los siguientes sprint desarrollados no se afecten al funcionamiento ya implementado con anterioridad, cada uno de los requerimientos cuenta con una historia de usuario en la cual se encuentran redactadas las pruebas de aceptación de la funcionalidad.

Se tiene un total de 20 pruebas de aceptación, de las cuales se han cumplido el 100% de lo propuesto, las mismas que se encuentran en el **Anexo C**, en cada una de las tarjetas contara con un identificador de la prueba nombre de la prueba, descripción número de historia de usuario a la que pertenece, datos de salida de la prueba datos obtenidos, los pasos que se deben realizar para la prueba, y las condiciones que debe tener para poder ejecutarla, como se puede apreciar en la **Tabla 4-3.**

Tabla 4-3: Formato pruebas de aceptación

Código: P-001	**Historia de usuario:** USR-001 Como usuario, requiero realizar el ingreso por teclado de una formula, con la finalidad de representarla en el plano X, Y, Z.
Nombre: La fórmula es registrada por la aplicación	
Descripción: La fórmula ingresada en la caja de texto es registrada por la aplicación	
Condiciones de Ejecución: • Iniciar la aplicación • Caja de texto en blanco	
Entrada/Pasos de Ejecución: • Iniciar la aplicación • Clic en la caja de texto • Escribir la formula a ingresar hasta en tres coordenadas **X, Y, Z** • Dar clic en el botón de ingresar.	
Resultados Esperados:	

• Se muestra la formula ingresada por la caja de texto.
Evaluación de la prueba: • Aceptada

Realizado por: García Michael. 2017

3.3. Análisis de la programación concurrente en la CPU y GPU

3.3.1. Gráficas en el Computador.

Podemos sintetizar la graficación por computadora como la creación de imágenes en dos y tres dimensiones mediante un computador para múltiples usos, como investigación científica, artes gráficas, en la industria, en la educación. Pero vale la pena indicar que la graficación en tres dimensiones no es otra cosa que una simulación en dos dimensiones.

El campo de la graficación por computadora comprende todos los aspectos relacionados con el uso del computador para generar imágenes, que pueden ser fijas o inclusive animadas.

El uso de la gráfica por computadora se ha incrementado notablemente en los últimos años, debido al estudio del software y al crecimiento en el hardware computacional.

Como habíamos indicado, cuando decimos que el computador realiza una imagen 3D en realidad es sólo una simulación en 2D, pero que da una apariencia tridimensional, es lo que matemáticamente se conoce como un isomorfismo del espacio de tres dimensiones en el espacio de dos dimensiones. Para lo cual debemos tener en cuenta algunos aspectos netamente matemáticos como las operaciones geométricas para la modelación y transformación, y los procesos algorítmicos o de presentación para todo lo que tiene que ver con la iluminación, sombreado, texturizado, eliminar superficies escondidas, rasterización, etc.

3.3.2. El sistema gráfico.

Iniciamos describiendo las características graficas de nuestro computador. La pantalla del computador está formada por un retículo de puntos llamados *Pixels* (Contracción de *Picture Elements*) que son individuados por una pareja de coordenadas enteras (*sx, sy)* en un sistema de referencia donde el origen *O= (0,0),* se encuentra en el ángulo superior izquierdo de la pantalla y el punto *M* (Punto de coordenadas máximas) en el ángulo inferior derecho y tiene coordenadas (sx_M, sy_M). Ve la **Figura 2-3**.

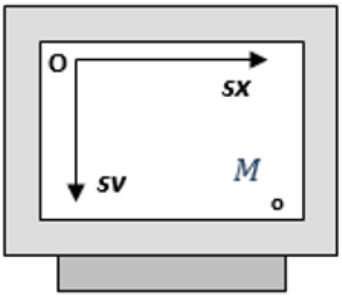


Figura 2-3: Sistema gráfico

De esta forma cada pareja de enteros (*sx, sy)* con $0 \leq sx \leq sx_M$ e $0 \leq sy \leq sy_M$ individua un píxel. Los valores sx_M y sy_M (el número total de pixels) son distintos, depende de las características de la tarjeta gráfica interna del computador. Cada tarjeta está asociada a un programa, llamado *driver gráfico* que le permite su funcionamiento.

Cada píxel puede ser "encendido" con un color distinto, utilizando oportunas instrucciones desde el Visual Studio, como

Canvas.Pixels[sx, sy] =ClColor;

La que nos permite pintar un píxel en las coordenadas *(sx, sy)*, con el color que queramos.

3.3.3. Ventana Real y Ventana Pantalla

Dentro de los pasos en la línea de ensamblaje que se utiliza para generar imágenes en un computador es fundamental diferenciar entre ventana real y ventana del computador. Es decir, tenemos dos sistemas de referencia: el sistema de referencia real donde se desarrolla el fenómeno a estudiar, y el sistema de referencia de la pantalla.

En el sistema de referencia real se identifica la ventana que queremos transportar sobre la pantalla *(ventana real)*, y sobre la pantalla se identifica el rectángulo en el que tal ventana vendrá transferida *(ventana pantalla)*. Es decir, debemos determinar la transformación (función) que asocia las coordenadas de del sistema de referencia real a las coordenadas de referencia de la pantalla en modo que la ventana real venga transformada en la ventana pantalla y viceversa, estos sistemas se pueden apreciar en la **Figura 3-3.**

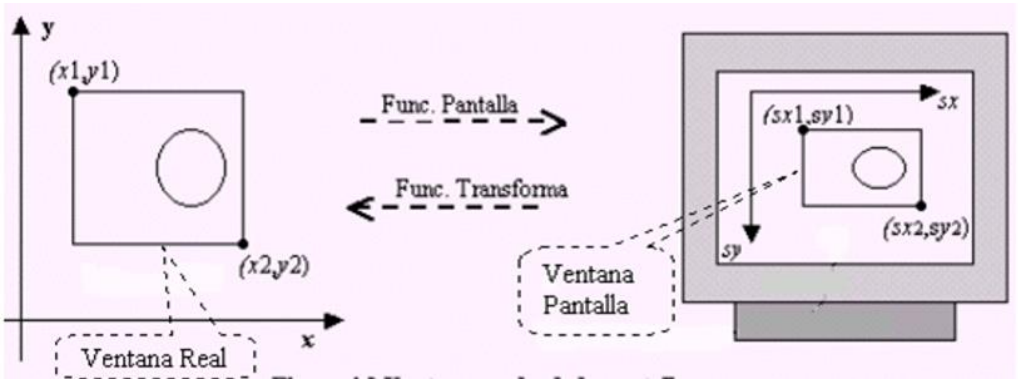


Figura 3.3: Ventana real y de la pantalla

Llamamos con (x_1, y_1) las coordenadas del ángulo superior izquierdo de la ventana real, y con (x_2, y_2) las coordenadas del ángulo inferior derecho de la ventana real. Llamamos con (sx_1, sy_1) las coordenadas del ángulo superior izquierdo de la ventana pantalla, y con (sx_2, sy_2) las coordenadas del ángulo inferior derecho de la ventana pantalla.

Buscamos relaciones que nos permitas expresar las coordenadas de referencia real con las coordenadas de referencia de la pantalla, es decir, relaciones de la forma

$$\begin{cases} x = \alpha.sx + \beta \\ y = \gamma.sy + \delta \end{cases}$$

con α y β tales que:

$$\begin{cases} x_1 = \alpha.sx_1 + \beta \\ x_2 = \alpha.sx_2 + \beta \end{cases}$$

y ϒ, δ tales que:

$$\begin{cases} y_1 = \gamma.sy_2 + \delta \\ y_2 = \gamma.sy_1 + \delta \end{cases}$$

Resolviendo los dos sistemas lineales se obtienen las relaciones

$$x = x_1 + \frac{(x_2 - x_1)(sx - sx_1)}{(sx_2 - sx_1)}$$

$$y = y_2 - \frac{(y_2 - y_1)(sy - sy_1)}{(sy_2 - sy_1)}$$

3.4. Clases y Métodos a utilizar

3.4.1. Repositorio de Modelos Matemáticos

Este repositorio contiene diferentes métodos que se utilizaran para poder representar objetos del mundo real o imaginario, en la pantalla de un computador, y poder realizar acciones sobre el mismo como rotarlo en diferentes ejes, sombrear, etc. En esta clase también es importante la utilización del proceso de la factorial de un número, dicho proceso se puede apreciar en el **Anexo E.**

3.4.2. Clase vector 3D

Esta clase es la principal ya que con ella se puede realizar la representación de cada una de las gráficas mediante las coordenadas de tres dimensiones (x, y, z), con la utilización de ecuaciones vectoriales y la herencia las demás clases dependerán de está para poder graficar **Anexo E.**

3.4.3. Segmento 3D

Esta clase permite poder trazar líneas desde un punto inicial a u punto final en coordenadas de tres dimensiones (x, y, z), mediante la utilización de vectores de las mismas dimensiones, su representación se la realiza con una ecuación vectorial **Anexo E.**

3.4.4. Clase Superficie

La clase permite representar las superficies, ingresadas por medio de una caja de texto, las mismas que deben estar representadas en ecuaciones de componentes de X, Y, Z para su adecuado funcionamiento. Trabaja con una clase matemática para poder capturar cada una de las dimensiones que se desean graficar **Anexo E.**

CONCLUSIONES

- Se utilizó GPU para ejecutar aplicaciones, fue necesario tomar en cuenta algunos factores, como el hecho de que el algoritmo a implementar fue, preferiblemente paralelo. Además, se determina que al usar estas herramientas es un requisito indispensable para el programador, no exclusivamente para los desarrolladores de software de sistemas.

- El proceso para la determinación de los parámetros para aprender de estas herramientas, fue un poco lento y mientras tanto el hardware de que se dispuso estuvo muy infrautilizado, ya que muy pocas aplicaciones aprovechan su potencia, en consecuencia, la comparación de programación concurrente entre GPU y CPU se realizó y/o comprobó con un poco de dificultad, sin embargo, no insidió significativamente con este objetivo.

- El sistema operativo utiliza lo múltiples núcleos de los actuales microprocesadores para repartir la carga de trabajo, pero apenas hace uso de la GPU. De hecho, las GPU, salvo en el caso de los juegos y algunas aplicaciones específicas de gráficos/vídeo, son el recurso más desaprovechado. La última generación de navegadores, no obstante, hace uso de esa potencia a través de la aceleración por hardware de la composición de páginas.

- Tras la indagación de ciertos modelos de iluminación y sombreado, se analizó el uso de un estándar nuevo, como lo es OpenCL, requiere de mucho tiempo, pero se puede convertir en una gran herramienta por los beneficios que puede dar al usarse para acelerar aplicaciones de propósito general. El paralelismo, cuando se trabajó en el procesador gráfico, si se lo hacía de la forma tradicional hubiese requerido el rediseño de los algoritmos, de tal forma que requería mapear los datos y representarlos como texturas; el uso de API's como OpenCL constituye un gran avance en este ámbito, pues ya no se requiere el mapeo.

RECOMENDACIONES

- Esta tesis sirva de base para el estudio y utilización de las arquitecturas heterogéneas (CPU+GPU), es decir que la GPU sea utilizada como co-procesador que sirve de apoyo en las tareas más exigentes y que no necesariamente se traten de aplicaciones gráficas o de videojuegos.

- Se recomienda un estudio en el cual se analice las dos tecnologías (CPU y GPU) desde el punto de vista de la escalabilidad para determinar cuán escalable puede ser un sistema al desarrollarse sobre una u otra plataforma.

GLOSARIO DE TÉRMINOS

ESCALABILIDAD. La escalabilidad es la capacidad de mejorar recursos para ofrecer una mejora (idealmente) lineal en la capacidad de servicio. La característica clave de una aplicación es que la carga adicional sólo requiere recursos adicionales en lugar de una modificación extensiva de la aplicación en sí.

RENDERIZAR. Es un término usado en informática para referirse al proceso de generar una imagen desde un modelo. Este término técnico es utilizado por los animadores o productores audiovisuales y en programas de diseño en 3D.

FOTOGRAMAS POR SEGUNDO. Las imágenes por segundo o en inglés frames per second (FPS) es la medida de la frecuencia a la cual un reproductor de imágenes genera distintos fotogramas o frames.

THREAD. En sistemas operativos, es un hilo de ejecución o hebra o subproceso y es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

KERNEL. Es utilizado en la programación bajo OpenCL para referirse a una función que será ejecutada en un procesador que soporte dicho estándar.

BIBLIOGRAFÍA

1. **ALBUJA, Pablo.** *PROGRAMACIÓN CONCURRENTE* [blog]. prezi.com, 28 enero, 2011. [Consulta: 2017-05-13 20:57:54]. Disponible en: https://prezi.com/xp7orqpanwyz/programacion-concurrente_pablo-albuja_-4tosis_iii-parcial/

2. **ANILEMA, José.** *Análisis de la Programación Concurrente sobre la CPU y GPU en el Desarrollo de Fractal Build [*En línea]. (tesis pregrado). Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica, Escuela Ingeniería en Sistemas. Riobamba, Ecuador. 2012. pp. 136- 145[Consulta: 2017-05-14 10:28:17]. Disponible en: http://dspace.espoch.edu.ec/handle/123456789/2534

3. *Historia de Scrum* [En línea]. Proyectosagiles.org, 19 agosto, 2008. [Consulta: 2017-03-22 08:49:19]. Disponible en: https://proyectosagiles.org/historia-de-scrum/

4. *PROGRAMACIÓN CONCURRENTE [*En línea]. www.pegasus.javeriana.edu.co, [Consulta: 2017-05-25 23:20:58]. Disponible en: http://pegasus.javeriana.edu.co/~scada/concurrencia.html

5. **ARISTA.** *QUÉ SON LAS CURVAS DE NIVEL EN UN MAPA TOPOGRÁFICO [*En línea]. www.aristasur.com, 2012. [Consulta: 2017-05-16 20:55:28]. Disponible en: http://www.aristasur.com/contenido/que-son-las-curvas-de-nivel-en-un-mapa-topografico

6. **AVENDAÑO, Andrea.** *OPERADORES DIFERENCIALES: GRADIENTE, DIVERGENCIA Y ROTACIONAL [*En línea]. www.documents.mx, 18 febrero, 2016. [Consulta: 2017-05-13 15:51:51]. Disponible en: http://documents.mx/documents/proceso-de-comunicacion-secuencial.html

7. **COLLA, Pedro. "**Marco para evaluar el valor en metodología SCRUM" [En línea], 2012, (Argentina), pp. 1-3. [Consulta: 24-02-2017 11:41:12]. Disponible en: http://41jaiio.sadio.org.ar/sites/default/files/086_ASSE_2012.pdf

8. **DOMENECH, Lourdes.** *MATERIALES DE LENGUA Y LITERATURA [*En línea]. www.materialesdelengua.org, 2005. [Consulta: 2017-05-13 19:08:49]. Disponible en: http://www.materialesdelengua.org/

9. **ESTEVEZ, Oswaldo et. al.** *GRADIENTES [*En línea]. www.mate3.foroactivo.com, 26 noviembre, 2009. [Consulta: 2017-05-16 13:08:14]. Disponible en: http://mate3.foroactivo.com/t4-gradientes

10. **FLÓRES, Julián et. al.** *PROGRAMACIÓN [*En línea]. www.powtoon.com, 5 abril, 2016. [Consulta: 2017-05-25 22:28:29]. Disponible en: https://www.powtoon.com/online-presentation/eIhkcXDqN8W/programacion/?mode=Movie

11. **FLORES, Lesly et. al.** *CURVAS DE NIVEL* [blog]. www.slideshare.net, 4 julio, 2015. [Consulta: 2017-05-16 20:55:17]. Disponible en: https://es.slideshare.net/Leslyaylin/curvas-de-nivel-36638100?next_slideshow=1

12. **GUZMAN, José.** *PROCESO DE COMUNICACIÓN SECUENCIAL [*En línea]. www.teoriaelectromagneticated502.pbworks.com, 2010. [Consulta: 2017-05-26 00:01:17]. Disponible en: http://teoriaelectromagneticated502.pbworks.com/w/page/20548734/Operadores%20Diferenciales%3AGradiente%2C%20divergencia%20y%20rotacional

13. **MAMNI, Gladys.** *PROGRAMACION CONCURRENTE* [blog]. www.slideshare.net, 19 noviembre, 2008. [Consulta: 2017-05-13 15:34:05]. Disponible en: https://es.slideshare.net/gladysmamani/programacion-concurrente-presentation-768002

14. **MARTINEZ, Nestor.** *CURVAS DE NIVEL [*En línea]. www.academia.edu, 2017. [Consulta: 2017-05-26 12:21:14]. Disponible en: http://www.academia.edu/17580955/Curvas_de_nivel

15. **MERINO, Cristian.** *DESARROLLO DEL SISTEMA ACADÉMICO DEL SINDICATO DE CHOFERES PROFESIONALES 4 DE OCTUBRE APLICANDO EL FRAMEWORK JSF [*En línea]. (tesis pregrado). Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica, Escuela Ingeniería en Sistemas. Riobamba, Ecuador. 2017. pp. 17- 19 [Consulta: 2017-05-22 10:18:07]. Disponible en:

16. **MONTAÑO, Deivid.** *SCRUM: Ensayo sobre Scrum* [blog]. Blogger, 20 julio, 2011. [Consulta: 2017-03-22 08:52:00]. Disponible en: http://scrum-ing-soft.blogspot.com/2011/07/ensayo-sobre-scrum.html

17. **PALACIO, Juan.** *Gestión de proyectos Scrum Manager* (Scrum Manager I y II) [En línea]. Versión 2.5. de la edición: Scrum Manager, 2014. pp. 13-69 [Consulta: 26-02-2017 12:46:49]. Disponible en: http://www.scrummanager.net/files/sm_proyecto.pdf

18. **RODRÍGUEZ, Miguel.** *INTRODUCCIÓN A LA PROGRAMACIÓN CONCURRENTE [*En línea]. www.www2.ulpgc.es, 2013. [Consulta: 2017-05-13]. Disponible en: http://www2.ulpgc.es/hege/almacen/download/20/20233/tema1.pdf

19. **RODRIGUEZ, Ricardo.** *Errores comunes de programación: Segunda Parte* [blog]. PMOinformatica.com, 24 octubre, 2012. [Consulta: 2017-03-07 09:40:49]. Disponible en: http://www.pmoinformatica.com/2012/10/errores-de-programacion-comunes-segunda.html

20. **SANDOVAL, Aziel.** *ILUMINACIÓN Y SOMBREADO* [blog]. www. sandovalpandha.blogspot.com, septiembre, 2013. [Consulta: 2017-05-13 21:11:24]. Disponible en: https://es.slideshare.net/gladysmamani/programacion-concurrente-presentation-768002

21. **SILVERA, Emilio.** *LA SIMETRÍA CP Y OTROS ASPECTOS DE LA FÍSICA* [En línea]. www.emiliosilveravazquez.com, 2015. [Consulta: 2017-05-16 13:02:15]. Disponible en: http://www.emiliosilveravazquez.com/blog/2015/09/20/la-simetria-cp-y-otros-aspectos-de-la-fisica/

22. **TACHOIRES, Rodrigo,** *Conceptos de Scrum* [blog]. SlideShare, 19 abril, 2016. [Consulta: 2017-02-28 18:54:15]. Disponible en: https://www.slideshare.net/RodrigoTachoires/conceptos-de-scrum

23. **TRIGAS, Manuel.** *Metodología Scrum* [En línea]. Ana Cristina Domingo Troncho, 2012. pp. 33-52 [Consulta: 22-02-2017 22:43:39]. Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf

24. **TROYA, Cesar.** *ALGORITMO DE MINIMIZACIÓN DE FUNCIÓN CONVEXA DEL COSTE* [blog]. www. cesartroyasherdek.wordpress.com, marzo, 2016. [Consulta: 2017-05-16 13:27:32]. Disponible en: https://cesartroyasherdek.wordpress.com/2016/03/04/algoritmo-de-minimizacion-de-funcion-convexa-del-coste/

25. **ZACARRO, Jorge.** *ARQUITECTURA DE COMPUTACIÓN DISTRIBUIDA PARA LA WEB 3D [*En línea]. (tesis postgrado). Pontificia Universidad Javeriana, Departamento de Ingeniería Electrónica. Bogotá D.C., Colombia. 2012. pp. 26- 67 [Consulta: 2017-05-13 20:54:07]. Disponible en: http://repository.javeriana.edu.co/handle/10554/12715

26. **WEITZENFELD, Alfredo.** *GRÁFICA: ILUMINACIÓN Y SOMBREADO [*En línea]. www.cannes.itam.mx, s.n. [Consulta: 2017-04-13]. Disponible en: http://www.cannes.itam.mx/Alfredo/Espaniol/Cursos/Grafica/Sombreado.pdf

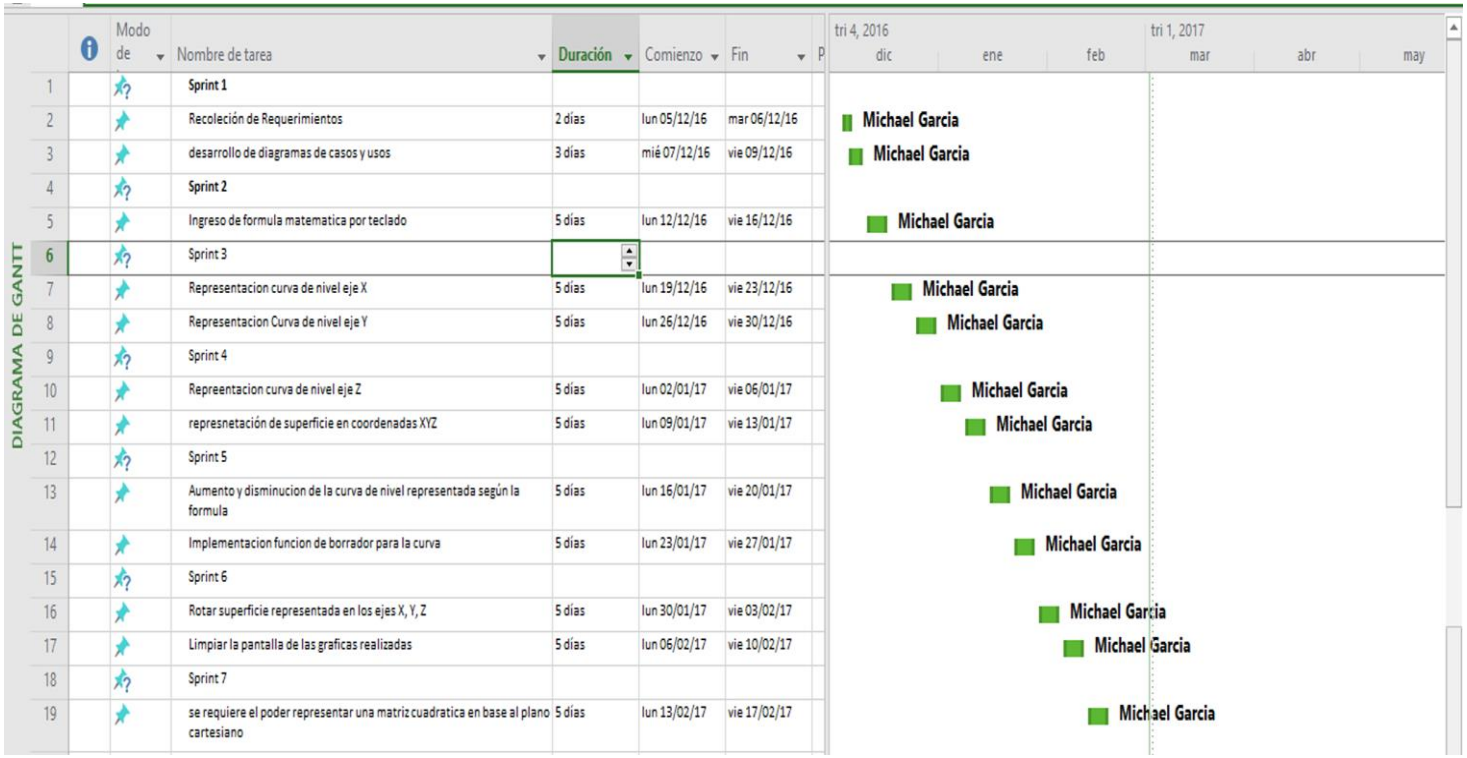
27. **WELINGTON, Francisco,** "El uso de un caso de investigación para el estudio de los métodos electrolíticos: Una experiencia en la educación superior". [En línea], 2013, (Brasil). ISSN: 3, 419-439 [Consulta: 2017-05-13]. Disponible en: http://reec.uvigo.es/volumenes/volumen12/REEC_12_3_3_ex709.pdf

28. **ZOLLER, Uri,** "The eclectic examination: A model for simultaneous formative evaluation of teacher-training programs and consequent student performance". [En línea], 1983, (Great Britain). DOI: 10.1016/0191-491X (83)90018-4. ISSN: 0191-491X [Consulta: 2017-05-13 19:23:03]. Disponible en: http://www.sciencedirect.com/science/article/pii/0191491X83900184

ANEXOS

Anexo A

Planificación y diagrama Gantt para el desarrollo del producto software.



Realizado por: García, Michael, 2017

Anexo B

Pila de Producto (Product Backlog): Instrumento de la metodología de desarrollo de software SCRUM en el que se lista las características y funcionalidades de producto software a desarrollar priorizadas de acuerdo a las necesidades.

A continuación, se muestran los posibles estados de la funcionalidad durante su ciclo de vida:

- ✓ **Vacío:** La historia fue identificada pero aún no ha sido asignada a una iteración.
- ✓ **Planificada:** La historia fue asignada a una iteración y aún no ha comenzado su ejecución. Puede tener este estado incluyendo en la iteración donde está planificado ejecutarla (pero que aún no ha comenzado).
- ✓ **En Proceso:** La historia fue seleccionada por el equipo y está en proceso de desarrollo (en ejecución).
- ✓ **Finalizada:** La historia fue desarrollada. Es importante clarificar la definición de "Finalizada" con el equipo de trabajo. "Finalizada" no sólo incluye el desarrollo sino la integración y pruebas integrales del Software. Una historia hecha puede presentarse al dueño de producto para sus pruebas de aceptación.
- ✓ **Descartada:** Se determinó que la historia ya no es relevante, su contenido se incluyó en otro grupo de historias o fue cancelada.

A continuación, se detalla la nomenclatura de los identificadores usados para las funcionalidades del sistema:

USR: Usuario de la aplicación.

Funcionalidades para la Aplicación de Escritorio

Identificador (ID) de la Historia	Enunciado de la Historia	Estado	Iteración (Sprint)	Prioridad
USR-001	Como usuario, requiero realizar el ingreso por teclado de una formula, con la finalidad de representarla en el plano **X, Y, Z.**	FINALIZADO	2	ALTA
USR-002	Como usuario, requiero representar la curva de nivel de la formula ingresada en el eje **X**.	FINALIZADO	3	ALTA
USR -003	Como usuario, requiero representar la curva de nivel de la formula ingresada en el eje **Y**.	FINALIZADO	3	ALTA
USR -004	Como usuario, requiero representar la curva de nivel de la formula ingresada en el eje **Z**.	FINALIZADO	4	ALTA
USR -005	Como usuario, requiero representar la superficie de la formula ingresada en un plano **X, Y, Z.**	FINALIZADO	4	ALTA
USR -006	Como usuario, requiero ampliar o disminuir la curva de nivel representada.	FINALIZADO	5	ALTA
USR -007	Como usuario, requiero borrar partes de la curva o superficie dibujada.	FINALIZADO	5	MEDIA
USR -008	Como usuario, requiero rotar la superficie en el eje X, Y o Z.	FINALIZADO	6	BAJA
USR -009	Como usuario, requiero limpiar la pantalla donde se grafica la superficie o curva de nivel	FINALIZADO	6	BAJA

USR -010	Como usuario, requiero dibujar un matriz del plano cartesiano.	FINALIZADO	7	BAJA

Realizado por: García, Michael, 2017

Anexo C

Historias de Usuario del Proyecto

Nº: USR-002	**Representación curva de nivel eje** X		
Como usuario, requiero representar la curva de nivel de la formula ingresada en el eje X.			
Estimación:	1 punto	**Real:**	2 puntos
Prioridad:	Alta	**Dependencia:**	USR-001
Pruebas de Aceptación:			
* La fórmula se gráfica en la aplicación. * La fórmula no se gráfica en la aplicación.			

Realizado por: García, Michael, 2017

Anexo D

Anexo E

Repositorio de modelos matemáticos:

```
Function  Factorial (n:integer):real;
   var
    i:integer;
    fact:real;
   begin
    if n=0 then fact:=1
      else
        begin
          fact:=1;
          for i:=2 to n do fact:=fact*i;
        end;
     factorial:=fact;
   end;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using System.Collections;
using System.Reflection;
using System.Diagnostics;

namespace NewCompu
{
   class RepositorioMM
   {
      public static int pxmax = 560;
      public static int pymax = 440;
      public static int pxmin = 0;
      public static int pymin = 0;
      public static double xmax = 14;
      public static double xmin = -14;
      public static double ymax = 11;
      public static double ymin = -11;
      PictureBox VentanaP;
```

```
public static void Pantalla(double x, double y, out int px, out int py)
{
   px = (int)Math.Truncate((pxmax - pxmin) / (xmax - xmin) * (x - xmax) + pxmax);
   py = (int)Math.Truncate((pymin - pymax) / (ymax - ymin) * (y - ymax) + pymin);
}

public static void Carta(int px, int py, out double x, out double y)
{
   x = ((px - pxmax) * ((xmax - xmin) / (pxmax - pxmin))) + xmax;
   y = ((py - pymax) * ((ymax - ymin) / (pymin - pymax))) + ymin;
}

public static double CartaX(double px)
{
   double x;
   x = ((px - pxmax) * ((xmax - xmin) / (pxmax - pxmin))) + xmax;
   return x;

}

public static double CartaY(double py)
{
   double y;
   y = ((py - pymax) * ((ymax - ymin) / (pymin - pymax))) + ymin;
   return y;
}


public void Rotar(double x, double y, double teta, out double rx, out double ry)
{
   rx = ((x  * (Math.Cos(teta))) - (y  * (Math.Sin(teta))));
   ry = ((x  * (Math.Sin(teta))) + (y  * (Math.Cos(teta))));

}

public void Rotar2(double X, double Y, double Teta,  double Xc, double Yc, out double VX, out double
VY)
{
   VX = (X - Xc) * Math.Sin(Teta) - (Y - Yc) * Math.Cos(Teta) + (Xc);
   VY = (X - Xc) * Math.Cos(Teta) + (Y - Yc) * Math.Sin(Teta) + (Yc);
}
```

```
public void RotarXYZ(double x, double y, double z, int eje, int ban, out double rx, out double ry, out double
rz)
{
   double gama = Math.PI / 10;
   rx = 0;
   ry = 0;
   rz = 0;
   if (eje == 1)
   {
      rx = x;
      ry = y * Math.Cos(gama) - z * Math.Sin(gama);
      rz = y * Math.Sin(gama) + z * Math.Cos(gama);
   }
   if (eje == 2)
   {
      rx = x * Math.Cos(gama) + z * Math.Sin(gama);
      ry = y;
      rz = -x * Math.Sin(gama) + z * Math.Cos(gama);
   }
   if (eje == 3)
   {
      rx = x * Math.Cos(gama) - y * Math.Sin(gama);
      ry = x * Math.Sin(gama) + y * Math.Cos(gama);
      rz = z;
   }
   if (eje == 0)
   {
      rx = x;
      ry = y;
      rz = z;
   }
}


public double Lagrange(double x, int nD, double[] vx, double[] vy)
{
   double S, P;
   int i, j;
   S = 0;
   for (i = 0; i < nD; i++)
   {
      P = 1;
      for (j = 0; j < nD; j++)
```

```
            {
                if (j != i)
                {
                    P = P * (x - vx[j]) / (vx[i] - vx[j]);
                }
            }
            S = S + vy[i] * P;
        }
        return S;
    }

    public void Bezier(double t, int ndatos, double[] vx, double[] vy, out double xt, out double yt)
    {
        int Nd = ndatos - 1;
        xt = 0;
        yt = 0;
        for (int i = 0; i <= Nd; i++)
        {
            xt = xt + (vx[i] * (fact(Nd) / (fact(i) * fact(Nd - i))) * (Math.Pow(1 - t, Nd - i) * Math.Pow(t, i)));
            yt = yt + (vy[i] * (fact(Nd) / (fact(i) * fact(Nd - i))) * (Math.Pow(1 - t, Nd - i) * Math.Pow(t, i)));
        }
    }

    public double fact(double real)
    {
        double resp = 1;
        for (double i = 1; i <= real; i++)
        {
            resp = resp * i;
        }
        return resp;
    }

    public static void Axonometria(double x, double y, double z, out double ax, out double ay)
    {
        ax = y - 0.5 * x * Math.Cos(Math.PI / 4); //los valores de ro = 0.5 y alfa = PI / 4;
        ay = z - 0.5 * x * Math.Sin(Math.PI / 4);
    }


    public void Cuadrilatero(double Px, double Py, double Qx, double Qy, double Rx, double Ry, double Sx,
double Sy, int Tipo, Bitmap grafico)
    {
```

```
int SPx, SPy, SQx, SQy, SRx, SRy, SSx, SSy;
Pantalla(Px, Py, out SPx, out SPy);
Pantalla(Qx, Qy, out SQx, out SQy);
Pantalla(Rx, Ry, out SRx, out SRy);
Pantalla(Sx, Sy, out SSx, out SSy);
Point p1 = new Point(SPx, SPy);
Point p2 = new Point(SQx, SQy);
Point p3 = new Point(SRx, SRy);
Point p4 = new Point(SSx, SSy);
Point[] p = { p1, p2, p3, p4 };
switch (Tipo)
{
   case 0:
      {
         Pen pn = new Pen(Color.Green, 1);
         Graphics g = Graphics.FromImage(grafico);
         g.DrawPolygon(pn, p);
      }
      break;
   case 1:
      {
         Pen pn = new Pen(Color.Green, 2);
         Graphics g = Graphics.FromImage(grafico);
         g.DrawPolygon(pn, p);
         g.FillPolygon(new SolidBrush(Color.Blue), p);

      }
      break;
   case 2:
      {
         Pen pn = new Pen(Color.Fuchsia, 2);
         Graphics g = Graphics.FromImage(grafico);
         g.DrawPolygon(pn, p);
         g.FillPolygon(new SolidBrush(Color.DarkGray), p);
      }
      break;
   case 3:
      {
         Pen pn = new Pen(Color.Black, 1);
         Graphics g = Graphics.FromImage(grafico);
         g.DrawPolygon(pn, p);
      }
      break;
```

```
        }
      }

  }
}
```

Clase vector 3D

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;

namespace NewCompu
{
   class Vector3D:Vector
   {
      public double z0;

      public Vector3D() { }

      public Vector3D(double x, double y, double z, Color c)
      {
         this.x0 = x;
         this.y0 = y;
         this.z0 = z;
         this.color0 = c;
      }

      ~Vector3D()
      { }

      public override void Encender(Bitmap grafico)
```

```
        {
            double xr, yr;
            int sx, sy;
            RepositorioMM.Axonometria(x0, y0, z0, out xr, out yr);
            RepositorioMM.Pantalla(xr, yr, out sx, out sy);
            if (sx > 0 && sx < 560 && sy > 0 && sy < 440)
            {
                grafico.SetPixel(sx, sy, this.color0);
            }

        }
    }
}
```

Segmento 3D

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;

namespace NewCompu
{
    class Segmento3D:Vector3D
    {
        public double xf, yf, zf;

        public Segmento3D() { }

        public Segmento3D(double x, double y, double z, double x1, double y1, double z1, Color c)
        {
            this.x0 = x;
            this.y0 = y;
```

```
            this.z0 = z;
            this.xf = x1;
            this.yf = y1;
            this.zf = z1;
            this.color0 = c;
        }
        ~Segmento3D()
        { }

        public override void Encender(Bitmap grafico)
        {
            double t = 0, dt = 0.001;
            Vector3D v1 = new Vector3D();
            while (t <= 1)
            {
                v1.x0 = x0 + (xf - x0) * t;
                v1.y0 = y0 + (yf - y0) * t;
                v1.z0 = z0 + (zf - z0) * t;
                v1.Encender(grafico);
                v1.color0 = this.color0;
                t += dt;
            }

        }
    }
}
```

Clase Superficie

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using info.lundin.math;
```

```
using System.Collections;
using System.Reflection;
using System.Diagnostics;


namespace NewCompu
{
   class Superficie : Vector3D
   {
      int Tipo;
      double Escalar;
      double Aux;
      int ban;
      int eje, giro;
      double dx = 0.01;
      string FunMate;
      int R = 0;
      PictureBox VentanaP;
      int cara;
      static double[] Rx = new double[10000],
         Ry = new double[10000],
         Rz = new double[10000];


      public int Tipo1
      {
         get
         {
            return Tipo;
         }

         set
         {
            Tipo = value;
         }
      }

      public double Escalar1
      {
         get
         {
            return Escalar;
         }
```

```
        set
        {
            Escalar = value;
        }
    }

    public double Aux1
    {
        get
        {
            return Aux;
        }

        set
        {
            Aux = value;
        }
    }

    public int Ban
    {
        get
        {
            return ban;
        }

        set
        {
            ban = value;
        }
    }

    public double Dx
    {
        get
        {
            return dx;
        }

        set
        {
            dx = value;
```

```
    }
}

public string FunMate1
{
    get
    {
        return FunMate;
    }

    set
    {
        FunMate = value;
    }
}

public PictureBox VentanaP1
{
    get
    {
        return VentanaP;
    }

    set
    {
        VentanaP = value;
    }
}

public int Cara
{
    get
    {
        return cara;
    }

    set
    {
        cara = value;
    }
}

public int Eje
```

```
{
    get
    {
        return eje;
    }

    set
    {
        eje = value;
    }
}

public int Giro
{
    get
    {
        return giro;
    }

    set
    {
        giro = value;
    }
}
double formula(string sFunction, double Vx, double Vy, double Vz)
{
    ExpressionParser oParser = new ExpressionParser();
    oParser.Values.Add("x", Vx);
    oParser.Values.Add("y", Vy);
    oParser.Values.Add("z", Vz);
    int iIterations = 1000;
    double fResult = 0f;
    Stopwatch watch = new Stopwatch();
    watch.Start();
    fResult = oParser.Parse(sFunction);
    Expression expression = oParser.Expressions[sFunction];
    for (int i = 0; i < iIterations; i++)
    {
        fResult = oParser.EvalExpression(expression);
    }
    watch.Stop();
    return (fResult);
}
```

```
public override void Encender(Bitmap grafico)
{
   double[,] Mx = new double[120, 120];
   double[,] My = new double[120, 120];
   double[] Vx = new double[1200];
   double[] Vy = new double[1200];
   double[] Vz = new double[1200];
   int i, j, ni, nj;
   double rx, ry;
   Vector3D V3d = new Vector3D();
   Vector3D T = new Vector3D();
   double x, dxx, y,dy;
   x = -10;
   dxx = dx; //No viene de afuera
   i = 0;
   V3d.color0 = color0;
   if (ban == 1)
   {
      dxx = 0.001;
      y = -7;
      dy = 0.001;
      j = 0;
      do
      {
         V3d.x0 = Aux;
         V3d.y0 = y;
         V3d.z0 = formula(FunMate, V3d.x0, V3d.y0, 0)*Escalar1;
         V3d.Encender(grafico);
         y += dy;
      } while (y <= 7);
   }

   if (ban == 2)
   {
      dxx = 0.001;
      y = -7;
      dy = 0.4;
      j = 0;
      do
      {
         V3d.x0 = x;
```

```
        V3d.y0 = Aux;
        V3d.z0 = formula(FunMate, V3d.x0, V3d.y0, 0) * Escalar1;

        V3d.Encender(grafico);
        y += dy;
        x += dx;

    } while (x <= 10);
}

if (ban == 4)
{
    j = 0;
    dxx = 0.2;
    do
    {
        y = -7;
        dy = 0.06;

        do
        {
            V3d.x0 = x;
            V3d.y0 = y;
            V3d.z0 = formula(FunMate, V3d.x0, V3d.y0, 0) * Escalar1;

            if ((V3d.z0 >= (Aux-0.1)) && (V3d.z0 <= (Aux + 0.1)))
            {
                V3d.z0 = Aux;
                V3d.Encender(grafico);
                for (double p = 0; p <= 0.12; p += 0.01)
                {
                    V3d.y0 = y + p;
                    V3d.Encender(grafico);
                }

                for (double p = 0; p <= 0.12; p += 0.01)
                {
                    V3d.x0 = x + p;
                    V3d.Encender(grafico);
                }


                j = j + 1;
```

```
            }
            y += dy;


        } while (y <= 7);
        x += dxx;
        i = i + 1;
    } while (x <= 10);
    ni = j - 1;
    nj = j - 1;

}

if (ban == 3)
{
    R = 0;
    dxx = 0.4;
    do
    {
        y = -7;
        dy = 0.4;
        j = 0;
        do
        {


            if (Eje == 0)
            {
                V3d.x0 = x;
                V3d.y0 = y;
                V3d.z0 = formula(FunMate, V3d.x0, V3d.y0, 0) * Escalar1; ;
            }
            else
            {
                V3d.x0 = (Rx[R]);
                V3d.y0 = (Ry[R]);
                V3d.z0 = Rz[R];
            }



            T.RotarXYZ(V3d.x0, V3d.y0, V3d.z0, Eje, Giro, out T.x0, out T.y0, out T.z0);
```

```
                Rx[R] = T.x0;
                Ry[R] = T.y0;
                Rz[R++] = T.z0;
                T.color0 = this.color0;
                T.Encender(grafico);

                Axonometria(T.x0, T.y0, T.z0, out rx, out ry);
                Mx[i, j] = rx;
                My[i, j] = ry;
                y += dy;
                j = j + 1;

            } while (y <= 7);
            x += dx;
            i = i + 1;
        } while (x <= 10);
        ni = i - 1;
        nj = j - 1;

        //mallado

        if (cara == 0)
        {
            for (i = 0; i < ni; i++)//mallado
            {
                for (j = 0; j < nj; j++)
                {
                    Cuadrilatero(Mx[i, j], My[i, j], Mx[i + 1, j], My[i + 1, j], Mx[i + 1, j + 1], My[i + 1, j + 1],
Mx[i, j + 1], My[i, j + 1], 0, grafico);
                }
            }

        }
        if (cara == 1)
        {
            int malla = 0;
            for (i = 0; i < ni; i++)
            {
                for (j = 0; j < nj; j++)
                {
                    double xx = (Mx[i + 1, j] - Mx[i, j]) * (My[i + 1, j + 1] - My[i + 1, j]);
                    double yy = (Mx[i + 1, j + 1] - Mx[i + 1, j]) * (My[i + 1, j] - My[i, j]);
```

```
                    if (xx - yy > 0)
                        malla = 1;
                    else
                        malla = 2;

                    Cuadrilatero(Mx[i, j], My[i, j], Mx[i + 1, j], My[i + 1, j], Mx[i + 1, j + 1], My[i + 1, j + 1],
Mx[i, j + 1], My[i, j + 1], malla, grafico);

                }
            }
        }
    }
  }
}
```