CENTRO DE INGENIERIA Y DESARROLLO INDUSTRIAL

SOFTWARE PARA REPRESENTACIÓN VIRTUAL DE DUCTOS E IDENTIFICACION DE DEFECTOS

Tesis

QUE PARA OBTENER EL GRADO ACADÉMICO DE

Maestro en Ciencia y Tecnología en la Especialidad de Mecatrónica

PRESENTA

Ing. Leonardo Barriga Rodríguez

POSGRADO INTERINSTITUCIONAL CIENCIA Y TECNOLOGIA

Santiago de Querétaro, Qro., México, Abril del 2009.

RESUMEN

En el lenguaje utilizado en el mantenimiento e inspección de ductos de hidrocarburos, un diablo o pig es un artefacto empleado para limpiar o inspeccionar un ducto. Es insertado en la tubería y es arrastrado por el flujo del petróleo o gas.

De acuerdo con la información que generan los diablos instrumentados durante la inspección de una tubería en la cual se transportan hidrocarburos principalmente, es necesario la extracción, organización, análisis y tratamiento de ésta, así como de la presentación gráfica de los resultados obtenidos.

En este trabajo se desarrolla un software que permite el tratamiento de la información en un diablo geómetra, desarrollado por CIDESI, así como la reconstrucción virtual de una tubería a partir de la información analizada.

El desarrollo de la tubería virtual se realiza utilizando las librerías de OpenGL, bajo una plataforma del lenguaje C++ de Builder.

En el software se crea una escena donde el usuario puede ir recorriendo la tubería como si estuviera recorriendo (dentro o fuera), una cámara de video a través o sobre la tubería en cuestión.

La creación e identificación de los defectos que durante la inspección fueron detectados son representados a través de un código de colores o incluso agregando una marca a la tubería virtual, para que al usuario le sea más sencillo la ubicación de las mismas.

Aquí se presenta cómo se crea el mallado de la tubería, así también, cómo se identifica el grado de defecto encontrado. Por otro lado se plantea cómo permite interactuar con el usuario por medio de selección de zonas de la tubería para poder realizar un mallado más fino de pequeñas zonas que se requiere examinar con mayor detalle.

Para realizar un detalle mayor de la tubería se prueban algunos métodos para el suavizado de las curvas, como son la duplicación de puntos de vértices, la parametrización de las líneas y la utilización de curvas de Bezier.

CONTENIDO

Lista de tablas

Lista de Figuras.

CAPÍTULO 1

INTRODUCCIÓN

1. INTRODUCCIÓN

1.1 ANTECEDENTES

La industria gráfica ha proporcionado paquetes estándar que sirven como intermediarios entre el hardware de gráficos y aplicaciones de los usuarios. Sin ellos, las aplicaciones gráficas serían muy difíciles de desarrollar y mantener. Uno de ellos, OpenGL (Open Graphics Library), ha sido adoptado por gran un número de proveedores de hardware y software. Como una interfaz de software para hardware de gráficos, el cual permite a los usuarios especificar los objetos y las operaciones de producción de color de alta calidad de imágenes en 3D de objetos gráficos.[2]

En México existen más de 50,000 Km de ductos de gas y petróleo, donde algunas de estas tuberías fueron colocadas en las décadas de 1930 y 1940 y muchas más se construyeron en las siguientes dos décadas 1950 y 1960. [4]

Es necesario diagnosticar periódicamente con ayuda de un equipo especial denominado “diablo” o conocido mundialmente como “pig”, las fallas en las tuberías como son: abolladuras, arrugas, pliegues, ovalamientos, picaduras, grietas, laminaciones, corrosiones, entre otras.

En el lenguaje utilizado en el mantenimiento e inspección de ductos de hidrocarburos, un diablo o pig es un artefacto empleado para limpiar o inspeccionar un ducto. Es insertado en la tubería y es arrastrado por el flujo del petróleo o gas.

El diagnostico de las tuberías en México, es realizado por compañías extranjeras principalmente BJ Services Company, Rosen, GE (General Electric) y Tuboscope, las cuales únicamente se dedican a ofrecer el servicio de inspección, es decir, sus equipos de inspección no los ofrecen a la venta.

Ya que CIDESI, se ha dado a la tarea de involucrarse en el desarrollo e investigación de estos equipos de inspección (diablos instrumentados y diablos geómetra), es necesario contar con software que permita la interpretación de la información que es adquirida y almacenada por estos equipos.

Existen varios artículos internacionales en cuanto a los sistemas que involucran un diablo geómetra o un diablo instrumentado, desde los sistemas de navegación inercial como hasta los sistemas de ultrasonido para los diablos instrumentados.

Con respecto al tratamiento de la información adquirida por los diablos instrumentados o mejor aún a la interpretación de la información para crear un ambiente en 3D de una tubería inspeccionada existen muy pocos trabajos (artículos, tesis, reportes técnicos, manuales), entre estos se encuentran el de Martin Johnson y G. Sen Gupta [3] , en el cual se presenta la reconstrucción de una tubería inspeccionada mediante un robot que proyecta una línea laser alrededor de la tubería para que una cámara adquiera las imágenes y por medio de perfilometria se reconstruya virtualmente la tubería.

Debido a que la información resulta ser de grandes volúmenes [1], hay que procesarla y presentarla de forma que se visualice con un grado de detalle en el cual el usuario tenga de forma clara y con una buena resolución la vista de la tubería y sus defectos.

1.2 OBJETIVOS

Realizar un software que permita visualizar de manera sencilla y clara la estructura geométrica de los ductos que se hayan inspeccionado mediante un diablo geómetra, además identificar los defectos detectados por medio de los diferentes sensores del diablo.

Identificar la posición de cada defecto en la tubería, en coordenadas cartesianas (x,y,z), además de relacionarlos a mapas georreferenciados mediante un Sistema Global de Posicionamiento.

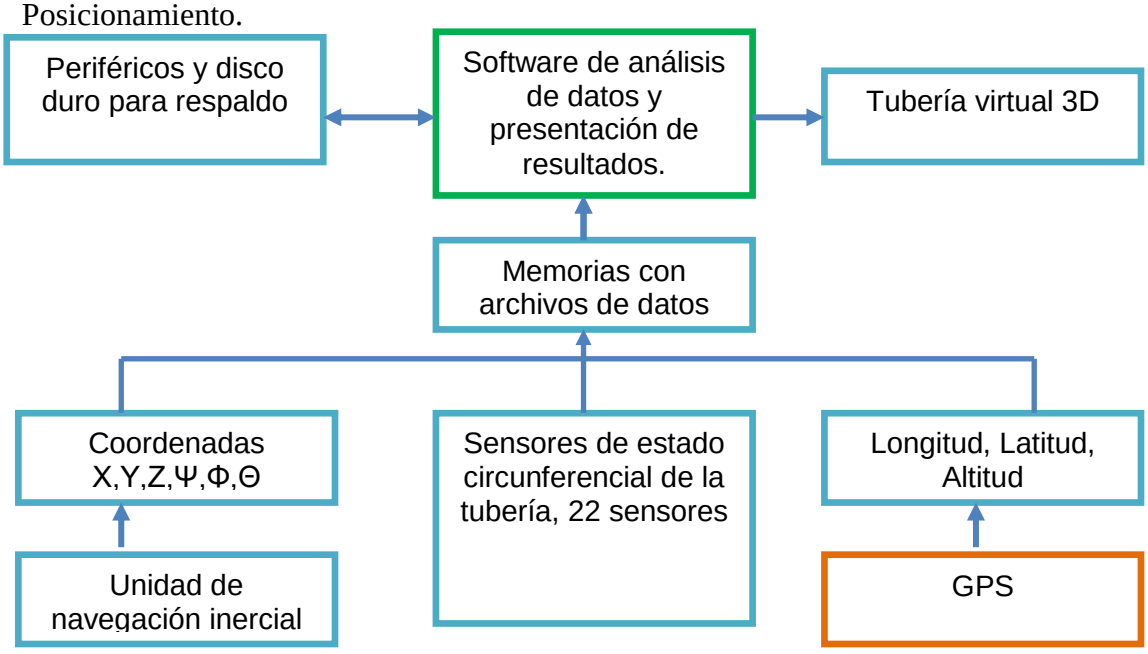


Figura 1: Diagrama de bloques del flujo de la información para representar la tubería virtual.

1.3 JUSTIFICACIÓN

- Las compañías prestadoras del servicio de inspección son propietarias de la tecnología la cual no está en venta. Todas las empresas son extranjeras.
- Ya que CIDESI se ha involucrado en la construcción de un diablo geómetra, es necesario el desarrollo de un software para generación de informes e interpretación de la información, por lo que se requiere del desarrollo de un software a la medida de las necesidades del propio diablo geómetra.
- El costo de la inspección y además de generación de reportes oscila entre USD $1,200-3,000 / Km., una vez construido y probado CIDESI podría disminuir esos costos ya que se contaría con tecnología propia desarrollada en México.
- Las compañías que contratan el servicio de inspección necesitan saber con un grado muy pequeño de error, por donde pasan las tuberías y donde se encuentran los defectos.
- La información generada por todos los sensores que se encuentran en el diablo geómetra, no representan nada por si solos para el usuario, es necesario un sistema que reúna toda la información y la presente de una forma fácil de interpretar.

1.4 ORGANIZACIÓN DE LA TESIS

La organización del trabajo se presenta en 7 capítulos y 1 anexo. En los primeros 5 capítulos se presenta una introducción a los fundamentos teóricos de las herramientas y técnicas utilizadas. En los capítulos 6 y 7 se presenta el desarrollo paso a paso del procedimiento que se propone y se siguió para el desarrollo del software que crea las tuberías virtuales a partir de la información contenida en un diablo geómetra. En el capítulo 7 se presentan las pruebas y resultados obtenidos del procedimiento.

En el capítulo 1 se presentan las generalidades y direcciones de la tesis, en donde se presenta una introducción para tener una idea general del proyecto y conocer las consideraciones que se tomaron para llevarlo a cabo.

En el capítulo 2 se presentan los fundamentos para la representación de objetos virtuales en 3D, dando una breve introducción de diferentes herramientas que existen para realizarlo a través de programación, tales como Java3D, Direct3D, VRML y OpenGL, de este último se explica con mayor profundidad debido a que es el que se utiliza en el desarrollo del proyecto.

En el capítulo 3 se describen los principales tipos de diablos que se utilizan durante la inspección de una tubería de transporte de hidrocarburos principalmente.

En el capítulo 4 se presentan los fundamentos para la representación de curvas y superficies, como curvas paramétricas, splines y curvas de Bezier, las cuales se utilizan para representar tramos de la tubería virtual de forma más detallada y con un suavizado en las curvas detectadas.

En el capítulo 6 se muestran el desarrollo del proyecto y los pasos para realizar la reconstrucción de la tubería virtual, así como la forma de representación y la interacción con el usuario. También se muestra los resultados obtenidos al representar varios tramos de tubería con información puesta de manera aleatoria dentro de los rangos para cada sensor del diablo geómetra.

CAPÍTULO 2

REPRESENTACIÓN DE OBJETOS VIRTUALES 3D

CAPITULO II

2. REPRESENTACIÓN DE OBJETOS VIRTUALES 3D

Con el aumento de la potencia gráfica de los sistemas actuales, cada vez es más frecuente basar la interacción persona-computador en la utilización de modelos y representaciones tridimensionales; el conjunto de técnicas denominadas Realidad Virtual son un ejemplo típico de esta situación. Sin embargo, la consideración de objetos y escenarios detallados precisa de la utilización de métodos que disminuyan fuertemente la complejidad de los modelos manipulados, manteniendo sin embargo la calidad visual de la representación.

2.1 JAVA 3D

Si un programador de Java quiere añadir en sus aplicaciones o applets, gráficos y animaciones en 3D, tiene dos opciones:

- Incluir métodos nativos escritos en OpenGL u otro lenguaje
- Utilizar el API 3D de Java (J3D en adelante)

La segunda opción es mejor y más fácil, siempre que el programador no sea ya un experto programador de OpenGL, de integrar con el resto del programa Java. Es una extensión de JDK2 de Java. Los creadores son los mismos que hicieron Java (SUN).

Como cabe de esperar en un lenguaje orientado a objetos como Java, J3D se basa en objetos. Existen objetos para transformaciones, luces, sonidos, formas, sombras, etc. Éstos tienen una estructura de nodos y arcos. Un nodo se representa con un objeto de las clases J3D y un arco es una relación entre nodos.

El crear un escenario gráfico en J3D no es más que el diseñar una estructura de nodos y arcos. Un nodo puede ser una luz, un sonido, una transformación, etc. Una relación dice dónde afectan estos objetos.

Existen dos clases de nodos. Por un lado están las clases nodo que extiende a la clase Node. Éstos tienen la particularidad de poder referenciar a varios objetos nodo hijos, pero solo

puede ser referenciado por un objeto nodo. Con esto se consigue que la estructura de nodos y arcos tenga la forma de árbol.

Por otro lado están las clases componente, que extienden a la clase NodeComponent. Estos pueden ser referenciados por más de un objeto nodo.

De forma general se puede decir que los objetos componente describen como es el objeto visual, y las clases nodo dice donde se colocan dentro del escenario gráfico.

En la figura 2 se muestra un ejemplo de una escena realizada en Java 3D.

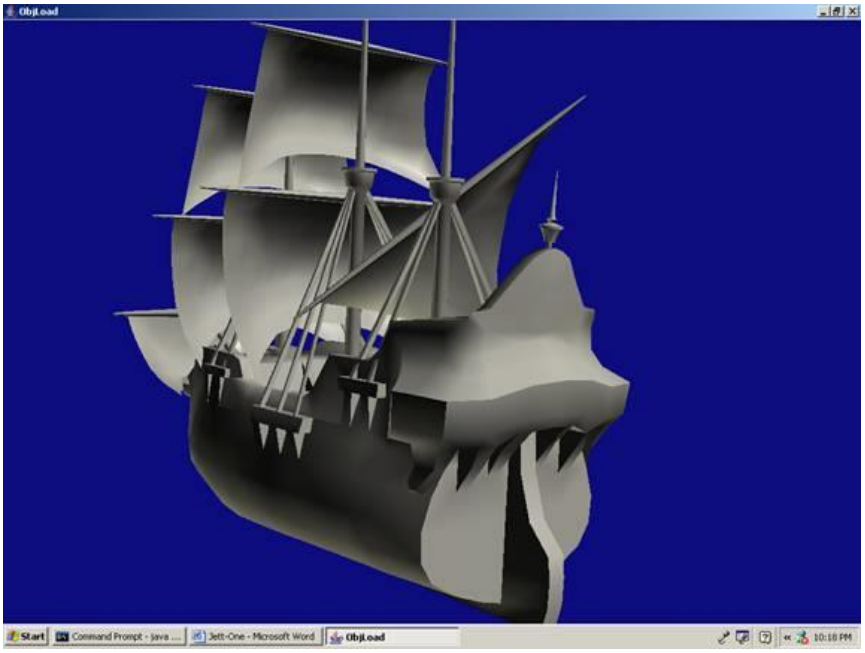


Figura 2: Escena tridimensional realizada con Java 3D

2.2 Direct3D

Direct3D es parte de DirectX, una API propiedad de Microsoft disponible tanto en los sistemas Windows de 32 y 64 bits, como para sus consolas Xbox y Xbox 360 para la programación de gráficos 3D.

El objetivo de esta API es facilitar el manejo y trazado de entidades gráficas elementales, como líneas, polígonos y texturas, en cualquier aplicación que despliegue gráficos en 3D, así como efectuar de forma transparente transformaciones geométricas sobre dichas entidades. Direct3D provee también una interfaz transparente con el hardware de aceleración gráfica.

Se usa principalmente en aplicaciones donde el rendimiento es fundamental, como los videojuegos, aprovechando el hardware de aceleración gráfica disponible en la tarjeta gráfica.

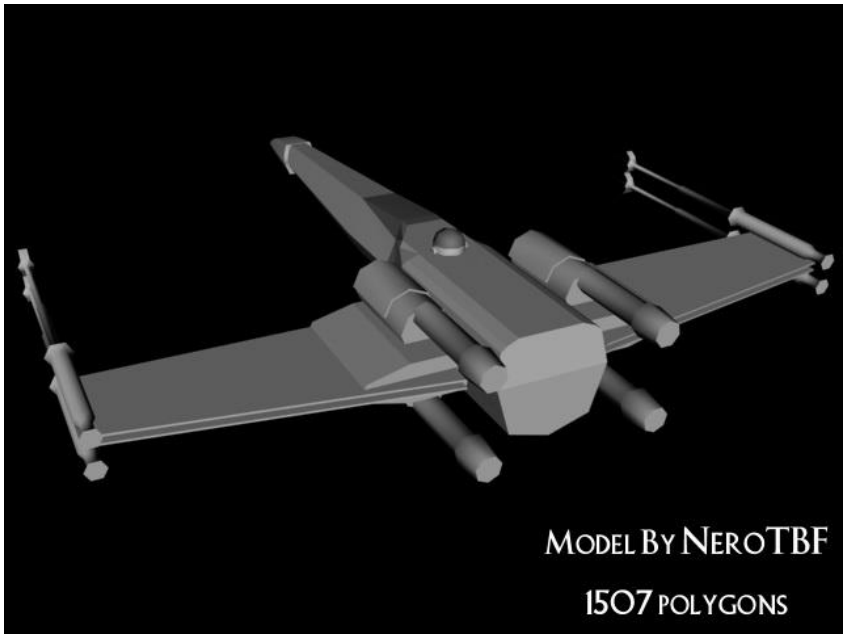


Figura 3: Escena creada mediante Direct3D

El principal competidor de Direct3D es OpenGL, desarrollado por Silicon Graphics Inc.

2.2.1 Arquitectura.

Direct3D es uno de los múltiples componentes que contiene la API DirectX de Windows. Se le podría situar al nivel del GDI de Windows, presentando un nivel de abstracción entre una aplicación de gráficos 3D y los drivers de la tarjeta gráfica Con arquitectura basada en el COM de Microsoft, la mayor ventaja que presenta Direct3D frente al GDI es que Direct3D se comunica directamente con los drivers de pantalla, consiguiendo mejores resultados en la representación de los gráficos por pantalla que aquel.

Direct3D está compuesto por dos grandes APIs. El modo retenido y el modo inmediato. El modo inmediato da soporte a todas las primitivas de procesamiento 3D que permiten las tarjetas gráficas (luces, materiales, transformaciones, control de profundidad, etc). El modo retenido, construido sobre el anterior, presenta una abstracción de nivel superior ofreciendo funcionalidades preconstruidas de gráficos como jerarquías o animaciones. El modo retenido ofrece muy poca libertad a los desarrolladores, siendo el modo inmediato el que más se usa.

El modo inmediato de Direct3D trabaja fundamentalmente con los llamados dispositivos (devices). Son los encargados de realizar la renderización de la escena. El dispositivo ofrece un interfaz que permite diferentes opciones de renderización. Por ejemplo un dispositivo mono permite la renderización en blanco y negro mientras que un dispositivo RGB permite el uso de colores. Podemos clasificar los dispositivos en tres clases principales:
Dispositivo HAL (hardware abstract layer): permite aceleración hardware. Se trata del dispositivo más rápido.
Dispositivo de referencia: es necesaria la instalación previa del SDK de Direct3D para poder usar este tipo de dispositivo. Permite la simulación software de un tipo de renderización y resulta muy útil cuando el hardware todavía no tiene incorporadas nuevas características de renderización. Un caso muy concreto del dispositivo de referencia es el

Null reference device cuya función es presentar la pantalla en negro. Se usa por defecto cuando se intenta usar un dispositivo de referencia y no se encuentra el SDK.
Dispositivos de conexión software (pluggable software device): permite opciones de rasterización software. Previamente se ha tenido que obtener el dispositivo mediante el método RegisterSoftwareDevice. Este tipo de dispositivos no fueron usados hasta DirectX 9.0.1

Cada dispositivo tiene asociada una o más cadenas de intercambio (swap chains). Dichas cadenas están compuestas por varios buffers de superficies, considerando a una superficie como un conjunto de píxeles más todos los atributos asociados a cada uno de ellos como la profundidad, el color, la transparencia (canal alfa), etc.

Además, los dispositivos tienen asociados también una colección de recursos (resources) o datos concretos necesarios para realizar la renderización. Cada uno de estos recursos tiene los siguientes atributos:
Tipo (type): define el tipo de recurso del que se trata: superficie, volumen, textura, textura de cubo, textura de volumen, textura de superficie, buffer de vértices o buffer de índices.
Almacén (pool): describe dónde se almacena el recurso durante la ejecución. Default indica que se almacena junto con el dispositivo; managed que se guarda en la memoria del sistema y se copia en el dispositivo cuando éste lo necesita; system memory que se encuentra exclusivamente en la memoria del sistema, al igual que scratch, ignorando este último las restricciones de la tarjeta gráfica.
Formato (format): describe el formato en que se almacena el recurso en memoria. La información más importante es respecto al almacenamiento de los píxeles en memoria. Un ejemplo de valor de format es D3DFMT_R8G8B8 que indica que una profundidad de color de 24 bits (los 8 de más peso para el rojo, los 8 de en medio para el verde y los 8 de menos peso para el azul).
Uso (usage): mediante una lista de flags, indica cómo se va a usar el recurso. También permite distinguir los recursos estáticos, aquellos que una vez cargados solo interesa su valor, de los recursos dinámicos, cuyo valor se modifica repetidamente.

En la figura 4 se presenta una versión simplificada de la pipeline de Direct3D.

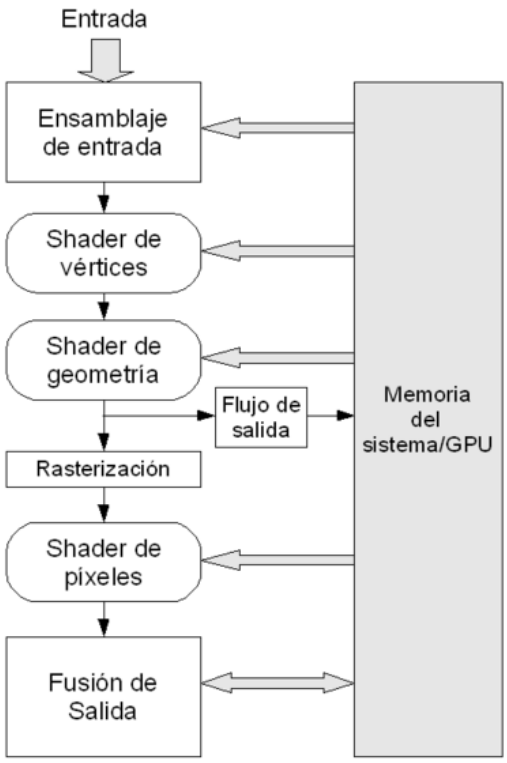


Figura 4: Proceso en la pipeline de gráficos

Las diferentes etapas del proceso de renderización son:

- Input Assembler: aporta los datos de entrada (líneas, puntos y triángulos).
- Vertex Shader: se encarga de las operaciones de vértices (iluminación, texturas, transformaciones). Trata los vértices individualmente.
- Geometry Shader: realiza operaciones con entidades primitivas (líneas, triángulos o vértices). A partir de una primitiva, el geometry shader puede descartarla, o devolver una o más primitivas nuevas.
- Stream Output: almacena la salida de la etapa anterior en memoria. Resulta útil para realimentar la pipeline con datos ya calculados.
- Rasterizer: convierte la imagen 3D en píxeles.
- Pixel Shader: operaciones con los píxeles.

- Output Merger: se encarga de combinar la salida del pixel shader con otros tipos de datos, como los patrones de profundidad, para construir el resultado final.

Direct3D permite la reconfiguración de todas las etapas, aumentando considerablemente la flexibilidad de esta pipeline. [15]

2.3 VRML

VRML (sigla del inglés Virtual Reality Modeling Language. "Lenguaje para Modelado de Realidad Virtual") - formato de archivo normalizado que tiene como objetivo la representación de gráficos interactivos tridimensionales; diseñado particularmente para su empleo en la web. Igual que el HTML nos sirve para maquetar páginas web, VRML sirve para crear mundos en tres dimensiones a los que accedemos utilizando nuestro navegador, igual que si visitáramos una página web cualquiera, con la diferencia que nuestras visitas no se limitan a ver un simple texto y fotografías, sino que nos permite ver todo tipo de objetos y construcciones en 3D por los que podemos pasear o interactuar. Consiste en un formato de fichero de texto en el que se especifican los vértices y las aristas de cada polígono tridimensional, además del color de su superficie. Es posible asociar direcciones web a los componentes gráficos así definidos, de manera que el usuario pueda acceder a una página web o a otro fichero VRML de Internet cada vez que pique en el componente gráfico en cuestión.

Este modo de visitar sitios en Internet es mucho más avanzado y posee grandes ventajas. Para empezar la navegación se desarrolla de una manera mucho más intuitiva, dado que la forma de actuar dentro del mundo virtual es similar a la de la vida real. Podemos movernos en todas las direcciones, no solo izquierda y derecha sino también adelante, atrás, arriba y abajo. Podemos tratar con los objetos como en la vida misma, tocarlos, arrastrarlos, etc. También los escenarios son mucho más reales, pensemos en un ejemplo como podría ser una biblioteca virtual. En ella podríamos andar por cada una de las salas, tomar determinados libros y leerlos, como en la imagen que se muestra a continuación, figura 5.

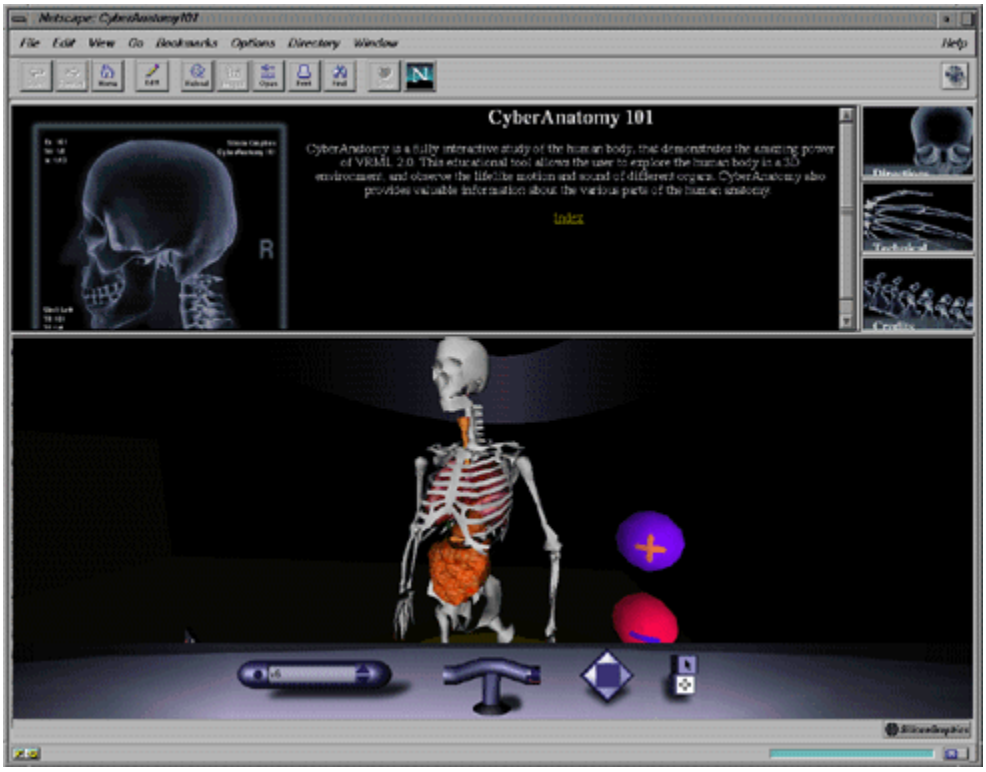


Figura 5: Ejemplo de la utilización de VRML para la creación de escenarios virtuales.

2.3.1 Materiales necesarios

Los materiales necesarios para comenzar con VRML son pocos, y probablemente se cuente con todo sin saberlo para el desarrollo y edición de mundos virtuales. Estos son:

Un editor de textos sencillo. El Block de notas es ideal o cualquier otro editor en modo ASCII. También podemos utilizar editores especializados como el VRML PAD.

Un visualizador VRML para ver los resultados, que se instala en el navegador como cualquier otro plug-in. Posiblemente el navegador instalado en la computadora a utilizar ya esté preparado para ver los mundos en VRML, si no es así, se tendría que instalar. Un visualizador muy conocido es el Cosmo Player.

2.4 OPENGL

OpenGL se define estrictamente como "una interfaz de software para hardware gráfico". Básicamente, se trata de una biblioteca de modelado y gráficos 3D que se pueden exportar muy rápidamente. Con OpenGL podemos crear gráficos 3D elegantes y con una calidad visual muy buena.

Al principio usaban algoritmos desarrollados y optimizados cuidadosamente por Silicon Graphics, Inc. (SGI), un reconocido líder mundial en gráficos y animación por ordenador. Con el tiempo OpenGL ha evolucionado a medida que otros proveedores han contribuido, con su experiencia y propiedad intelectual, a desarrollar implantaciones propias de alto rendimiento. [8]

Durante años OpenGL se ha consolidado como la librería por excelencia para desarrollar aplicaciones 2D y 3D con independencia de la plataforma o el hardware gráfico. Cuenta con más de nueve años de vida, por lo que existe una extensa base de conocimientos a su alrededor. Durante todo este tiempo se han producido cambios en la librería, pero OpenGL siempre asegura una compatibilidad "marcha atrás". De esta forma, una aplicación que se desarrolló usando la primera implementación de la librería, compilaría y funcionaria con la última versión de la misma.

Gracias a la portabilidad de OpenGL nuestras aplicaciones podrán ejecutarse en una amplia variedad de arquitecturas y de soportes gráficos, sin que el resultado se vuelva inconsistente. Por ejemplo, nuestro código fuente va a ser el mismo para una máquina Sparc corriendo Linux, que para un PC corriendo Windows y usando una aceleradora gráfica. Este punto dota a nuestras aplicaciones de una gran escalabilidad. Actualmente OpenGL está disponible para una gran variedad de sistemas operativos, tales como Unix, Windows, Mac OS, BeOS, etc.

2.4.1 Introducción a los gráficos en 3D

Los objetos reales poseen tres dimensiones: altura, anchura y profundidad. Para que estos objetos puedan representarse en una dimensión menos tenemos que transformar la información visual del objeto original con el objetivo de producir la ilusión de ver un objeto tridimensional en un sistema bidimensional. Esta transformación se basa en la perspectiva, la eliminación de las caras ocultas, los cambios de color, iluminación e intensidad de las texturas o colores de los objetos.

Para poder realizar todas estas operaciones necesitamos un sistema de coordenadas sobre el que trabajar. El más común es el sistema de coordenadas cartesianas. En nuestro caso, contamos con tres ejes: X, Y, Z. De tal forma que dos a dos son perpendiculares. Los ejes X e Y representan la posición horizontal y vertical respectivamente, y el eje Z representa la profundidad.

Trasladándolo al monitor de un ordenador, el eje X se extiende a lo ancho, el Y a lo alto, y el Z se dirige hacia el observador desde el centro de la pantalla.

Una vez que se tiene definido el espacio de coordenadas podemos posicionar objetos dentro de él, simplemente con especificar el punto del espacio donde lo deseamos colocar.

2.4.2 El vertice: una posición en el espacio

Tanto en 2D como en 3D, cuando se dibuja un objeto, en realidad se compone de diversas formas más pequeñas denominados primitivos. Los primitivos son entidades o superficies de una o dos dimensiones con puntos, líneas y polígonos (una forma de multiples lados plana) que se montan en un espacio 3D para crear objetos 3D. Por ejemplo, un cubo tridimencional está compuesto de seis cuadrados bidimencionales ,cada unno colocado con una cara independiente. Cada esquina del cuadrado (o de cuelquier forma primitiva) se denomina vértice. Estos vértices se especifican a continuación para ocupar una coordenada determinada en el espacio 3D. La creación de un objeto geométrico sólido es un poco más que un juego de conexión de puntos.

2.4.3 Coordenadas de trabajo y vistas

Como ya hemos visto por cada objeto tridimensional que vayamos a representar OpenGL va a realizar una proyección de este sobre un espacio 2D. Este espacio es nuestra área de trabajo, es decir, una ventana.

Así pues, lo primero que debemos hacer para que OpenGL pueda empezar a funcionar es crear una nueva ventana de trabajo.

OpenGL va a trasladar las coordenadas tridimensionales a coordenadas bidimensionales a través de la proyección de los objetos en la ventana de trabajo. Disponemos de dos clases de proyecciones:

- Proyección paralela o proyección ortogonal: En este tipo de proyección no se tiene en cuenta el punto de vista. Es decir, dos objetos con el mismo tamaño se ven igual en la pantalla, independientemente que de uno esté más alejado del punto del vista del observador. Por otra parte si un objeto no entra en el espacio de trabajo no se dibuja.

  Para especificar el volumen de visualización en una proyección ortogonal, se especifica los planos de recorte lejos, cerca, izquierdo, derecho, superior e inferior. Los objetos y figuras que se coloquen dentro de este volumen de visualización se proyectan en una imagen 2D que aparece en la pantalla.

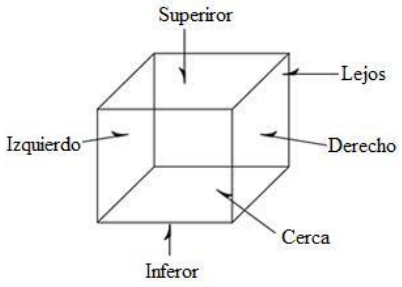


Figura 6: Volumen de recorte para una proyección ortogonal.

- Proyección en perspectiva: La principal característica de esta proyección es que los objetos más cercanos al observador se ven más grandes que los que están más alejados. Este tipo de proyección ofrece más realismo para la simulación y la animación 3D.

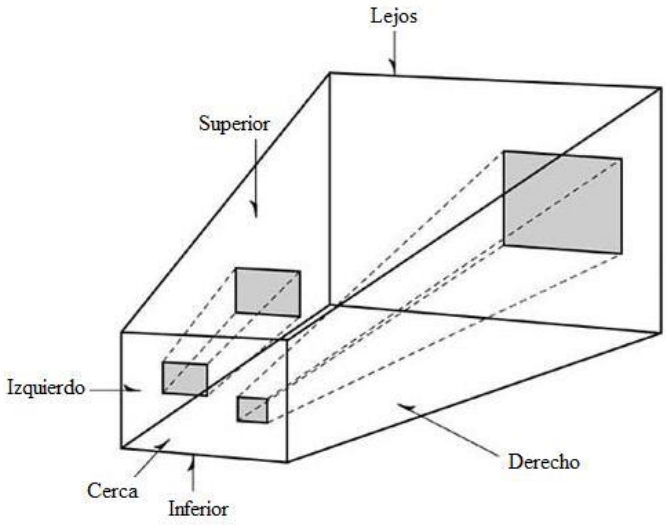


Figura 7: El volumen de recorte en una proyección de perspectiva.

2.4.4 Tipos de datos de OpenGL

Uno de los puntos fuertes de OpenGL es su gran portabilidad. Debido a esto, en las librerías de OpenGL se definen una serie de tipos de datos para que no haya problemas entre

diferentes arquitecturas, y a cada uno de ellos se le asocia un sufijo que posteriormente veremos que es usado en las funciones de gl.

Sufijo literal de C	**Tipo de datos OpenGL**	**Representación interna**	**Definido como tipo C**
B	GLbyte	Entero de 8 bits	signed char
s	GLshort	Entero de 16 bits	Short
i	GLint GLsizei	Entero de 32 bits	int / long
Ub	GLubyte GLboolean	Entero de 8 bits sin signo	unsigned short
Us	GLushort	Entero de 16 bits sin signo	unsigned short
Ui	GLunit GLenum GLbitfield	Entero de 32 bits sin signo	unsigned int / unsigned long
F	GLfloat GLclampf	Flotante de 32 bits	Float
D	GLdouble GLclampd	Flotante de 64 bits	Doublé

Tabla 1: Tipos de variable de OpenGL y sus tipos de datos C correspondientes

Es muy recomendable que todas las variables, cabeceras de funciones, nuevos tipos, etc., se hagan con los de OpenGL como base. Al hacerlo así se asegura la portabilidad sin ningún tipo de problemas.

El sufijo de tipo de usa porque hay funciones en el API de la librería que tienen muchas variantes, como por ejemplo **glVertex2i().** Con esta nomenclatura se pueden distinguir de antemano los argumentos que se le pasan a la función.

La primera parte, es decir "gl", hace referencia a la librería. "Vertex" es el nombre del comando que se va a ejecutar. El "2" indica el número total de argumentos de entrada que tiene la función, y la "i" hace referencia al tipo de estos argumentos, en este caso números enteros de 32 bits.

2.4.5 Dibujando con OpenGL

OpenGL dispone de métodos para dibujar primitivas gráficas de multitud de clases (como puntos, líneas, polígonos, etc.) y nos brinda una forma uniforme de dibujar todos estos objetos, basada en el uso de las funciones glBegin() y glEnd(). Para dibujar cualquiera de ellos lo primero que hay que hacer es llamar a glBegin con un argumento que describa el tipo de primitiva a dibujar: GL_POINT, GL_LINES, GL_LINES_LOOP, GL_LINE_STRIP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP o GL_POLYGON.

A continuación definir los datos sobre la(s) primitiva(s), y finalmente llamar a glEnd(). Por ejemplo, el siguiente fragmento de código dibuja un punto en la posición 250, 200.

```
glBegin(GL_POINTS);
   glVertex2i(250,200);
glEnd();
```

Para poder probar todos estos ejemplos vamos a tener que realizar unas modificaciones en la función de inicialización, quedando como sigue:

```
void
inicializacion (void) {
```

```
    glClearColor (1.0, 1.0, 1.0, 0.0);

    glColor3f (0.0,0.0,0.0);

    glPointSize (4.0);

    glMatrixMode (GL_PROJECTION);

    glLoadIdentity ();

    gluOrtho2D (0, ANCHO, 0, ALTO);

}
```

En esta función se han introducido las siguientes llamadas a funciones:

- glColor3f: Especifica el color actual de dibujo mediante su valor RGB. Por ejemplo: glColor3f (1.0, 0, 0) fija como color predefinido el rojo, o glColor3f (0.5, 0.5, 0.5) fija un gris.

- glPointSize: Esta función especifica el grosor de los puntos que se van a dibujar a través del numero en coma flotante que toma como argumento.

2.4.6 Definición del espacio de trabajo

OpenGL posee un nivel de abstracción mayor, que va a facilitar mucho el trabajo con gráficos y escenas en tres dimensiones, ya que en lugar de trabajar con las coordenadas de la pantalla (o ventana), lo va a hacer con unas coordenadas que fija arbitrariamente el programador dependiendo de sus necesidades; es decir, nuestro volumen de trabajo va a será tan grande o tan pequeño como queramos, y tendremos la posibilidad de escoger el sistema de coordenadas que más de adapte a las necesidades de nuestro programa.

Por ejemplo: Se puede crear un volumen de trabajo de con una longitud en el eje de las X de 200, una longitud de 500 en el eje Y, y de 30 en el eje Z.

A la hora de crear el volumen de trabajo se debe tener en cuenta que a OpenGL no le es suficiente con conocer las longitudes de la escena, tiene que saber cómo colocar los ejes de coordenadas sobre la misma: No es lo mismo una escena entre los valores -100 y 100, que una otra entre 0 y 200, aunque las dos tengan la misma dimensión.
La definición de las dimensiones del volumen de trabajo se realiza mediante la función glOrtho.

El prototipo de esta función es el siguiente:

void glOrtho (GLdouble izquiera, GLdouble derecha, GLdouble abajo, GLdouble arriba, GLdouble cerca, GLdouble lejos)

Mediante la llamada a esta función se está definiendo el espacio de trabajo de la escena, es decir, los puntos a partir de los cuales la escena se sale de la ventana (o la pantalla) y por lo tanto no es proyectado sobre esta. Estos dos puntos son (izquierda, abajo, -cerca) y (derecha, arriba, -cerca), y están mapeados sobre la esquina inferior izquierda y la esquina superior derecha de la pantalla respectivamente.

Detallando un poco más este mecanismo observamos que la llamada a glOrtho crea una matriz que multiplicará por la actual, es decir la llamada a glOrtho no crea un volumen de trabajo nuevo que reemplaza por el actual, si no que modifica el volumen de trabajo existente en ese momento mediante la multiplicación por esta matriz.

$$\begin{bmatrix} \frac{2}{(izquierda - derecha)} & 0 & 0 & \frac{derecha + izquierda}{derecha - izquierda} \\ 0 & \frac{2}{(arriba - abajo)} & 0 & \frac{(arriba + abajo)}{(arriba - abajo)} \\ 0 & 0 & \frac{-2}{(lejos - cerca)} & \frac{(lejos + cerca)}{(lejos - cerca)} \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Como norma general antes de realizar una llamada a glOrtho, se realiza una anterior a glLoadIdentity para cargar una matriz identidad sobre la que realizar la multiplicación. De esta forma el resultado es el señalado en glOrtho, claro está independientemente de la matriz anterior.

El prototipo de glLoadIdentity es:

void glLoadIdentity (void)

La llamada a esta función tiene exactamente el mismo resultado que la llamada a glLoadMatrix con la matriz identidad como parámetro, aunque en este caso la forma más rápida es mediante la llamada a glLoadIdentity.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 8: Matriz Identidad.

2.4.7 Vistas

Una vista define las coordenadas de un área dentro de la ventana sobre la cual OpenGL puede dibujar. Por lo tanto, el espacio de trabajo actual se mapeará sobre el área definida por estas coordenadas. Un detalle importante que hay que señalar son: las vistas se fijan teniendo en cuanta las coordenadas de la ventana, no las de el espacio de trabajo.

Hay que hacer una distinción entre coordenadas del espacio de trabajo de OpenGL, y las coordenadas de la ventana sobre la que se proyecta el resultado.

Gracias a las vistas es posible realizar tareas de una cierta complejidad con un simple cambio de vista sobre la venta. Por ejemplo, supongamos que nuestro objetivo es crear un programa de diseño de modelos 3D, en el cual sea posible ver estos modelos desde varias perspectivas al mismo tiempo. En principio para esta tarea las vistas son la herramienta fundamental.

Para dibujar los diferentes objetos sobre la pantalla bastaría con cambiar la vista, fijar un tamaño de vista menor que el de la ventana e ir cambiando su localización de una zona a otra. Es decir, dibujar el modelo 3D con el que trabajemos, cambiar la vista a otra porción de la pantalla, realizar los cambios necesarios sobre el modelo y volver a dibujarlo sobre la ventana.

Otro de los procesos que las vistas simplifican enormemente es el hacer zoom sobre una porción de la ventana. Al contrario que en el ejemplo anterior, si se define una vista mayor que el tamaño de la ventana, al mapearse sobre esta ultima el resultado resultará agrandado.

Las vistas se fijan mediante la llamada a la función de la librería gl, glViewport. El prototipo de esta función es:

void glViewport (GLint x, GLint y, GLsizei ancho, GLsizei alto)

Los parámetros *x* e *y* especifican las esquina inferior izquierda de la vista dentro de la ventana, y los parámetros ancho y largo especifican las dimensiones en pixeles.

Un error muy frecuente es llamar a esta función con dos parejas de números especificando el área de dibujo en la ventana. Como hemos señalado, se produciría un resultado erróneo, ya que en realidad se trata de las dimensiones en lugar de un segundo punto.

2.4.8 Transformaciones 3D

Un punto A con coordenadas Ax, Ay y Az respecto de los ejes X, Y y Z se representa mediante la siguiente matriz: {Ax, Ay, Az, 1}.

2.4.8.1 Translaciones

El caso de las translaciones en el espacio es idéntico al de las translaciones en el plano.

1	0	0	Mx
0	1	0	My
0	0	1	Mz
0	0	0	1

Figura 9: Matriz de translación

En esta ocasión también es posible realizar el desplazamiento sin necesidad de multiplicar por la matriz de transformación, basta simplemente con sumar a cada coordenada del punto el desplazamiento en el eje correspondiente que se desea realizar.

La función encargada en OpenGL de generar matrices de transformación para la translación es glTranslatef. El prototipo de esta función es el siguiente:

void glTranslate (GLfloat x, GLfloat y, GLfloat x)

Los tres parámetros especifican la cantidad que se va a desplazar en cada uno de los ejes de coordenadas.
Aparte de generar la matriz, glTranslatef también la aplica.

2.4.8.2 Escalamiento

Existen tres factores de escalado, uno para cada eje: Ex, Ey y Ez.

Ex	0	0	0
0	Ey	0	0
0	0	Ez	0
0	0	0	1

Figura 10: Matriz de escalado

Si los tres factores son iguales la imagen resultante guardará proporcionalidad con la original. En el caso especial de que los tres factores tengan valor 1, la imagen no sufrirá cambios.

En este caso las funciones de escalado en OpenGL son las siguientes: glScalef y glScaled.

void glScalef (GLfloat x, GLfloat y, GLfloat z)

void glScaled (GLdouble x, GLdouble y, GLdouble z)

Los tres argumentos de estas funciones representan los factores por los que se van a multiplicar cada una de sus tres dimensiones.

2.4.8.3 Rotaciones

En este caso, al disponer de 3 ejes de coordenadas podemos realizar tres tipos de rotaciones: una rotación en el eje X, en el eje Y o en el eje Z.

Por ejemplo, una rotación en el eje X quiere decir que se realiza la rotación moviendo el eje Y hacia el eje Z, de forma que el X permanece inmóvil.

Las matrices de transformación para las rotaciones 3D de un ángulo 'σ' son las siguientes:

Para rotaciones en el eje X:

1	0	0	0
0	cos(σ)	-sen(σ)	0
0	sen(σ)	cos(σ)	0
0	0	0	1

Figura 11: Matriz de rotación para el eje X

En el eje Y:

cos(σ)	0	sen(σ)	0
0	1	0	0
-sen(σ)	0	cos(σ)	0
0	0	0	1

Figura 12: Matriz de rotación para el eje Y

Y, en el eje Z:

cos(σ)	-sen(σ)	0	0
sen(σ)	cos(σ)	0	0
0	0	1	0
0	0	0	1

Figura 13: Matriz de rotación para el eje Z

Al igual que con las translaciones, OpenGL dispone de una función que genera y aplica matrices de transformación para la rotación: glRotatef. El prototipo de la función es el siguiente:

void glRotatef (GLfloat angulo, GLfloat x, GLfloat y, GLfloat z)

El primer argumento especifica el ángulo que se ha de rotar (en grados). Los siguiente tres argumentos especifican el vector sobre el que se va a rotar. De esta forma es posible realizar rotaciones sobre cualquier eje de giro con un coste mínimo.

El programador puede optar por hacer uso, o no, de las funciones de translación, rotación o escalado. En OpenGL se ha contemplado la posibilidad de que las matrices de transformación en algún caso se puedan generar a mano por el programador.
Una vez que se ha creado una matriz de transformación, hay que cargarla sobre la matriz de transformación de OpenGL mediante una de estas dos funciones: glLoadMatrixf o glMultMatrixf.
La primera de ellas sobrescribe la matriz de transformación actual con la que se le provee por parámetro. Y la segunda, se multiplica la matriz actual por la que se pasa por parámetro.
Las matrices se pueden definir de dos formas diferentes:

- Como un array de 16 elementos (recordemos que se trata de matrices de 4x4): GLfloat M[16]
- Como un array de 4x4 elementos: GLfloat M[4][4]

En ambos casos hay que tener en cuenta que OpenGL espera matrices definas por columnas, es decir:

A0	A4	A8	A12
A1	A5	A9	A13
A2	A6	A10	A14
A3	A7	A11	A15

Figura 14: Definición de una matriz de 4 x 4 elementos

Los prototipos de las funciones son los siguientes:

void glLoadMatrixd (const GLdouble matriz)*

void glLoadMatrixf (const GLfloat matriz)*

y

void glMultMatrixd (const GLdouble matriz)*

void glMultMatrixf (const GLfloat matriz)*

2.4.9 La pila de matrices

La pila de transformaciones funciona como una pila normal de cualquier programa. Esta pila en concreto está dedicada a guardar matrices (de ahí que sea la pila de matrices de transformación). Hay que señalar que esta pila de matrices, por defecto en la ejecución normal de un programa contiene en su creación un elemento: una matriz unidad.

La principal funcionalidad de la pila es la de poder almacenar estados de la matriz de transformación en cualquier momento de la ejecución. De esta forma, la matriz actual se puede modificar a gusto del programador y en cualquier otro momento rescatar la copia original de la matriz y seguir trabajando con ella.

En caso de no disponer de esta pila, la tarea descrita anteriormente resultaría mucho más larga y tediosa, ya que en lugar de meter y sacar de la pila, sería necesario comenzar de nuevo a crear la matriz de transformación a través de la función glLoadIdentity y todas las de transformación.

Los prototipos de las funciones de trabajo con la pila son las siguientes:

void glPushMatrix (void)

void glPopMatrix (void)

La principal funcionalidad de la pila es la de poder almacenar estados de la matriz de transformación en cualquier momento de la ejecución. De esta forma, la matriz actual se puede modificar a gusto del programador y en cualquier otro momento rescatar la copia original de la matriz y seguir trabajando con ella.
Por otra parte, es preciso comentar que esta pila no es ilimitada, sino que tiene un tamaño restringido. En caso de intentar introducir una matriz en la pila si ya está llena, o de sacar un elemento de la pila si ya está vacía, se producirá un error de GL_STACK_OVERFLOW y GL_STACK_UNDERFLOW respectivamente.

2.4.10 SELECCIÓN

La selección es en realidad un modo de interpretación, pero en este modo, no se copia ningún pixel en el búfer de fotogramas. En su lugar, los primitivos que se dibujan dentro del volumen de visualización (y por lo tanto, los que normalmente aparecerían en el búfer de fotogramas) producen registros de aciertos en un búfer de selección. Este búfer, al contrario que el resto de los búfers de Opengl, es una matriz de valores enteros.
Se debe establecer este búfer de selección de antemano y de nominar nuestros primitivos o grupos de primitivos (nuestros objetos o modelos) para que se puedan identificar en el búfer de selección. A continuación, analizamos el búfer de selección para determinar qué objetos se intersecan con el volumen de visualización. Un uso potencial para esta opción es la determinación de la visibilidad. Los objetos denominados que no aparecen en el buffer de selección se encuentran fuera del volumen de visualización y puede que no se dibujen en el modo de interpretación. Aunque el modo de selección es lo bastante rápido como para seleccionar un objeto, si se usa para un propósito general de elección de troncos piramidales se ejecuta significativamente más lentamente que cualquier otra técnica aprendida. Normalmente, modificamos el volumen de visualización antes de introducir el modo de visualización y llamamos a nuestro código de dibujo para determinar qué objetos se encuentran en una zona restringida de nuestra escena. Para elegir, especificamos un volumen de visualización que se corresponde con el puntero del ratón y a continuación comprobamos qué objetos denominados se interpretan por debajo del puntero del ratón.

2.4.10.1 Denominar los primitivos

Podemos denominar todos los primitivos usados para interpretar nuestra escena de objetos , pero hacerlo es muy poco útil. Es más normal denominar grupos de primitivos, creando así nombres para objetos o elementos de objetos específicos en nuestra escena. Los nombres de los objetos, igual que los nombres de las listas de presentación, no son más que enteros sin firmar.

Las listas de nombres se conservan en la pila de nombres. Una vez inicializada unja pila de nombres, podemos introducir nombres en la pila o simplemente reemplazar el nombre que se encuentra actualmente en la parte superior de la pila. Cuando se produce un acierto durante la elección, todos los nombres que se encuentran actualmente en la pila de nombres se adjuntan al final del búfer de selección. Por lo tanto, un solo acierto puede devolver más de un nombre, si es necesario.

Los términos e instrucciones mencionadas en este capítulo, específicamente los referidos a OpenGL son los utilizados en el capítulo IV para reconstruir la tubería virtual a partir de los datos que arroja un diablo geómetra, del cual se hablara a continuación en el capitulo III.

CAPÍTULO 3

DIABLOS INSTRUMENTADOS

CAPITULO III

3. DIABLOS INSTRUMENTADOS

3.1 DIABLO DE LIMPIEZA

En el lenguaje utilizado en el mantenimiento e inspección de ductos de hidrocarburos, un diablo o pig es un artefacto empleado para limpiar o inspeccionar un ducto. Es insertado en la tubería y es arrastrado por el flujo del petróleo o gas.

Existen diferentes tipos de diablos, un diablo de limpieza es utilizado únicamente para limpiar de residuos los ductos. Para conocer el estado de las tuberías es necesaria la utilización de diablos geómetra para conocer la geometría de la tubería, además de diablos instrumentados para conocer el estado de la pared de las mismas.

Su objetivo es de asegurarse que no habrá obstrucciones que impidan el paso del diablo geómetra en el diámetro interno, así como para remover cualquier formación de cera que pudiera estar presente.

Figura 15: Algunos tipos de diablos de limpieza.

Al igual que con cualquier inspección, los mejores resultados se obtienen si la tubería está limpia. La inspección por ultrasonido es una técnica precisa que puede ser afectada por la suciedad, los desechos y las burbujas al igual que una inspección con un equipo magnético. Es por esto que resulta de gran importancia el uso de los diablos de limpieza.

Previamente de colocar un diablo de limpieza se deben checar el diámetro interno de las válvulas y verificar que cumpla con las tolerancias mínimas, las tees deben contener barras guías para evitar que la herramienta de inspección se atore, los radios de curvatura de los

codos deben exceder los tres diámetros de la tubería (1.5 veces el diámetro para diablo de ultrasonido y 3 veces el diámetro para diablos de flujo magnético)

3.2 DIABLO GEOMETRA

El diablo geómetra (o caliper) es un diablo que se utiliza para evaluar los defectos geométricos del ducto tales como ovalamientos, arrugas o abolladuras, utiliza brazos flexibles como sensores, y es introducido a la tubería antes de utilizar el diablo instrumentado.

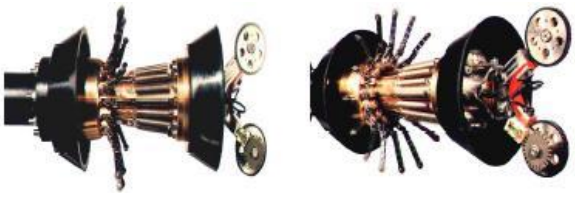


Figura 16: Diablo geómetra, BJ Pipeline Inspection Services

Una herramienta de este tipo también cuenta con un sistema de navegación inercial a bordo, el cual genera información que es utilizada para generar las coordenadas X, Y, y Z de los ejes centrales de la tubería. Los brazos colocados alrededor del diablo geómetra (colocados externamente entre las dos copas del geómetra), responden a los cambios en el diámetro interno de la tubería, mientras que el sistema electrónico interno registra todos estos datos en memoria. La figura 17 muestra el diablo geómetra construido en CIDESI.

Figura 17. Diablo geómetra construido en CIDESI.

3.3 DIABLO INSTRUMENTADO ULTRASONICO

El diablo instrumentado es un dispositivo de inspección que se introduce en la tubería de distribución de hidrocarburos. Durante su viaje efectúa una inspección con técnicas no destructivas de la sanidad estructural del material de la tubería.

Figura 18: Diablo instrumentado, BJ Pipeline Inspection Services

Un diablo instrumentado es introducido a la tubería normalmente después de que se ha introducido un diablo geómetra y se ha supuesto que la tubería está libre de grandes obstáculos.

El diablo geómetra y el diablo instrumentado deben ser capaces de discernir y almacenar la información adquirida durante el trayecto recorrido y una vez fuera del ducto, transferir tal información a una computadora con un software de análisis. La información adquirida está relacionada con la sanidad del material así como con la posición y orientación del diablo.

Los diablos de diámetros muy grandes pueden colectar más de 500 gigabytes de datos [5]. A bordo, se lleva a cabo una compresión de datos antes de ser almacenados ya sea en medios magnéticos o en medios de estado sólido como las memorias flash utilizadas para altas temperaturas.

3.4 Características del diablo geómetra a probar

Este trabajo se enfoca en la interpretación de la información generada por un diablo geómetra para tuberías de 10 pulgadas de diámetro, cuenta con 22 sensores de tipo LVDT

(Transformador Diferencial Variable Lineal), los cuales están conectados en brazos abatibles que se contraen y expanden dependiendo del estado de la tubería.

La información generada por el diablo geómetra, es almacenada en memorias internas, que permiten su lectura vía USB para su posterior análisis, figura 18.

La capacidad de autonomía del diablo geómetra tanto en alimentación eléctrica como en almacenamiento de datos es de 36 horas o 100 km de inspección, lo que ocurra primero, la velocidad máxima a la que es capaz de viajar en una corrida es de 2.7 m/s, aproximadamente 10 km/h. Es capaz de almacenar información a una velocidad aproximada a 1ms.

El tamaño del dato que se almacena en el diablo geómetra para cada sensor se muestra en la siguiente tabla:

LVDTs	16 bits
Encoders	16 bits
Acelerómetros	16 bits
Girómetros	16 bits
Inclinómetros	16 bits

Tabla 2: Tamaño del dato de cada sensor.

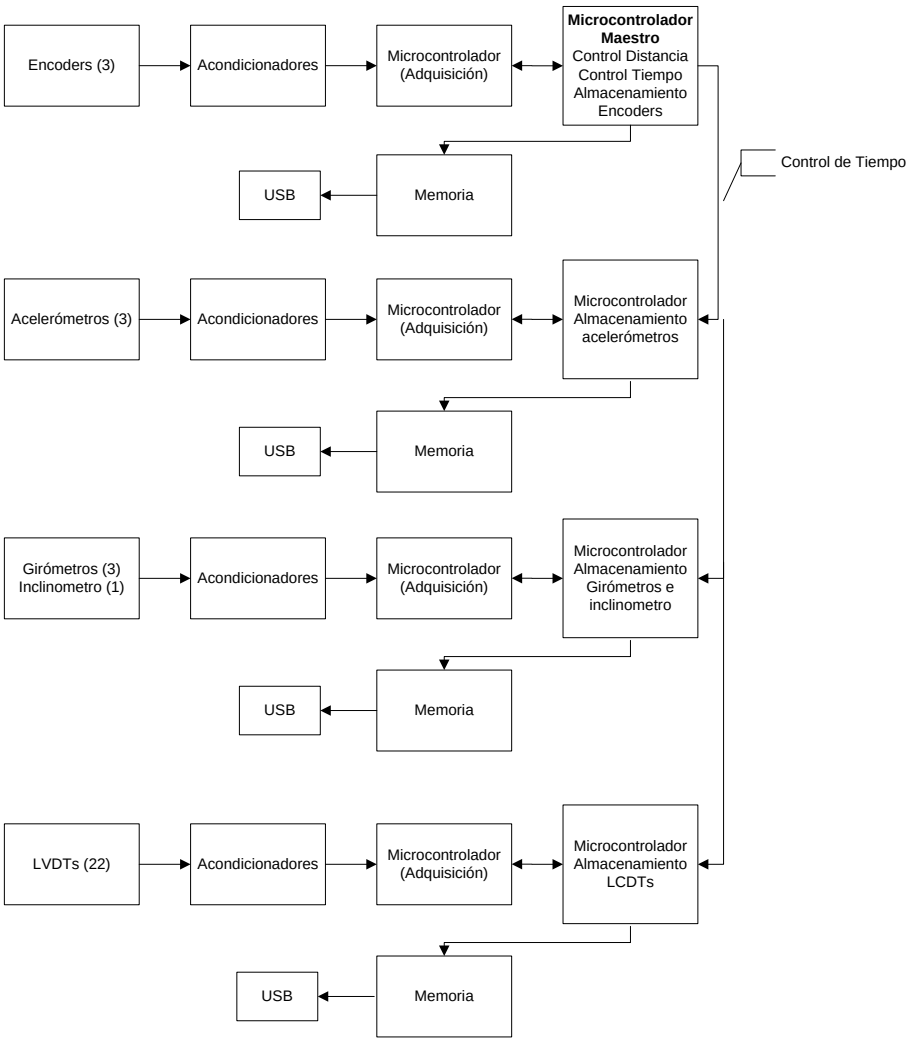


Figura 19: Diagrama de bloques de la parte electrónica que adquiere información de los sensores en el diablo geómetra.

La ubicación de los sensores LVDT, así como de los odómetros en el cuerpo del diablo geómetra se muestra en la figura 20, en la cual también se visualiza la posición del transmisor del tracker.

El tracker es un sistema de seguimiento del diablo que consiste en un transmisor y un receptor de seguimiento que es colocado en varios puntos de la tubería. El transmisor utiliza señales electromagnéticas de baja frecuencia para penetrar las paredes de la tubería y cuando son recogidas por el receptor este emite una notificación de audio y visual para indicar que el diablo ha pasado por ese punto.

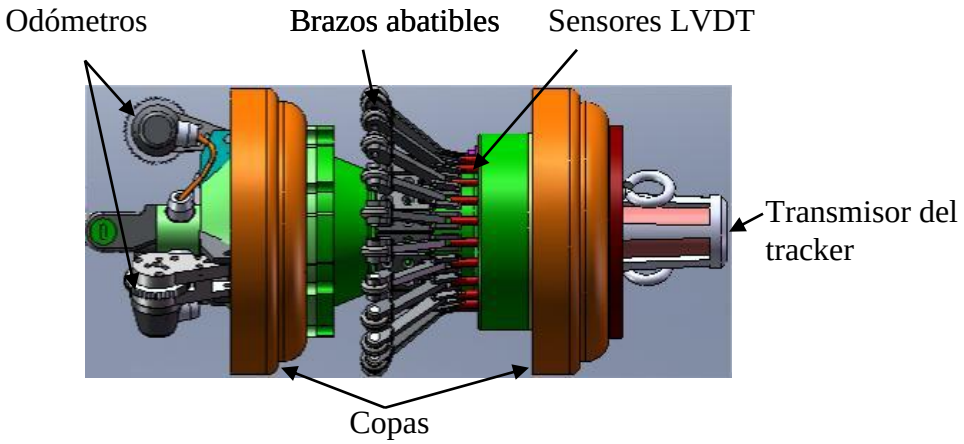


Figura 20: Cuerpo principal del diablo geómetra de CIDESI.

Los brazos abatibles del diablo geómetra se encuentran a 16.36° de separación entre ellos (figura 21), y la posición que adoptan en una tubería de 10 pulgadas de diámetro es su posición inicial, es decir si se expanden se incrementara un valor positivo que va de 0 a 10 mm, y se es que se contraen se incrementará un valor negativo, que va de 0 a -10 mm. Estos valores representan las máximas deformaciones que se registrará el diablo.

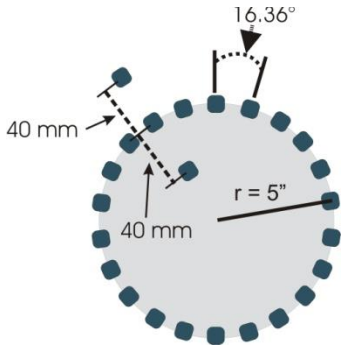


Figura 21: Distribución de los sensores alrededor del diablo geómetra.

En la figura 22 se muestra el diseño de un brazo abatible del diablo geómetra.

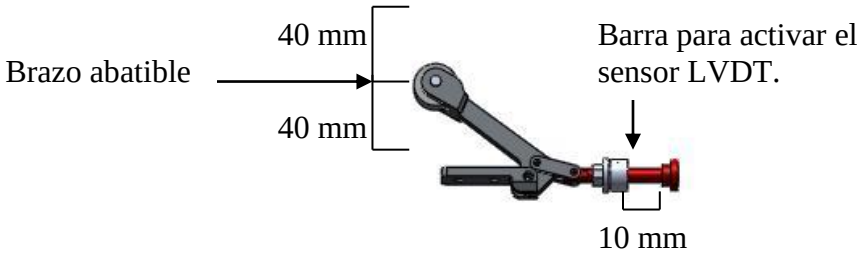


Figura 22: Brazo para registrar deformaciones.

3.4.1 Navegación Inercial

El principio de funcionamiento de este sistema se basa en el posicionamiento relativo a partir de la integración de las aceleraciones registradas por los acelerómetros, utilizando las velocidades angulares de los giroscopios para determinar la dirección del recorrido. Al conjunto de sensores inerciales se le conoce como IMU que, junto con las ecuaciones de mecanización, conforman un sistema de navegación inercial (INS). [4] Ver figura 23.

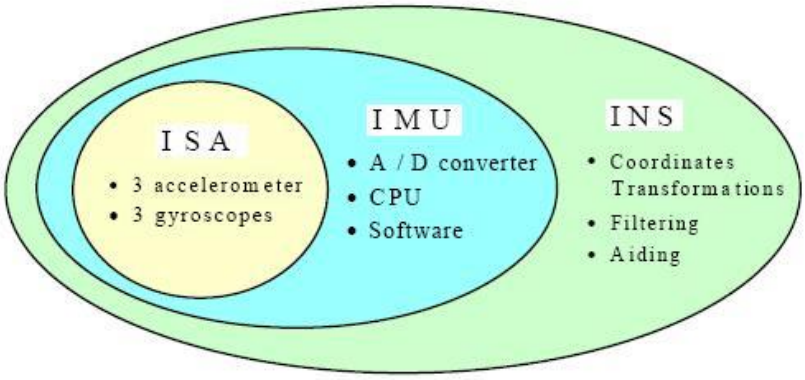


Figura 23: Estructura de sistemas de navegación inercial.

ISA (Inertial System Assembly)

IMU (Inertial Measurement Unit)

INS (*Inertial Navigation System)*

Generalmente los sistemas inerciales se clasifican en:

- Con plataforma o Gimballed.
- Sin plataforma o Strapdown.

En este proyecto se empleará el de tipo Strapdown, por ser el más usado por los sistemas de inspección de tuberías.

Las mediciones de los sensores son filtradas para eliminar ruido antes de ser integradas. El algoritmo de integración comúnmente usado es el Filtro de Kalman. Gracias a éste es posible calcular la posición, orientación y curvatura de la tubería. A continuación se muestra un diagrama del método de navegación inercial Strapdown, figura 24.

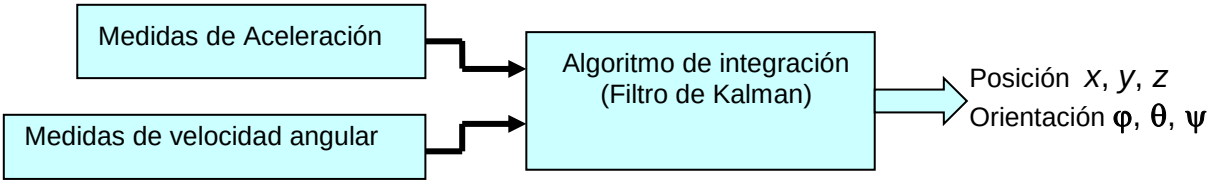


Figura 24: Diagrama del método de navegación inercial Strapdown

CAPÍTULO IV

SUAVIZADO Y REPRESENTACION DE CURVAS Y SUPERFICIES

CAPITULO IV

4. SUAVIZADO Y REPRESENTACIÓN DE CURVAS Y SUPERFICIES

La necesidad de representar curvas y superficies proviene de modelar objetos a partir de cero o representar objetos reales. En este último caso, normalmente no existe un modelo matemático previo del objeto, y el objeto se aproxima con "pedazos" de planos, esferas y otras formas simples de modelar, requiriéndose que los puntos del modelo sean cercanos a los correspondientes puntos del objeto real.

4.1 Representación paramétrica

Una curva solo tiene un solo punto inicial, una longitud y un punto final. En realidad es como una línea que se dibuja en un espacio 3D. Por otro lado, una superficie tiene anchura y longitud y, por lo tanto, un área de superficie.

Cuando pensamos en líneas rectas, podemos pensar en esta famosa ecuación:

$$y = mx + b$$

Aquí, m es igual a la pendiente de la línea y b es la intercepción de y con la línea. Esta explicación puede recordarle a sus tiempos de colegio, donde seguramente también aprendió las ecuaciones de parábolas, hipérbolas, curvas exponenciales, etc. Todas esas ecuaciones expresan *y* (o *x*) en función de alguna función de *x* (o *y*). Otra forma de expresar la ecuación para una curva o línea es una ecuación paramétrica, que expresa tanto *x* como *y* en función de otra variable que varía por un rango predefinido de valores que no forma explícitamente parte de la geometría de la curva. Por ejemplo, algunas veces, en Física, las coordenadas x, y y z de una partícula pueden expresarse según algunas funciones de tiempo, donde el tiempo se expresa en segundos. En el siguiente ejemplo, f(), g() y h() son funciones únicas que varían con el tiempo (t):

x = f(t)

y = g(t)

z = h(t)

Cuando definimos una curva en OpenGL, también definimos como una ecuación paramétrica. El parámetro paramétrico de la curva, que denominamos u, y su rango de valores forman el dominio de dicha curva. Las superficies se describen usando dos parámetros paramétricos: u y v. La figura 25 muestra tanto una curva como una superficie definidas en función de los dominios de u y v. Lo importante es advertir que los parámetros paramétricos (u y v) representan las extensiones de las ecuaciones que describen la curva; no reflejan los valores de las coordenadas reales.

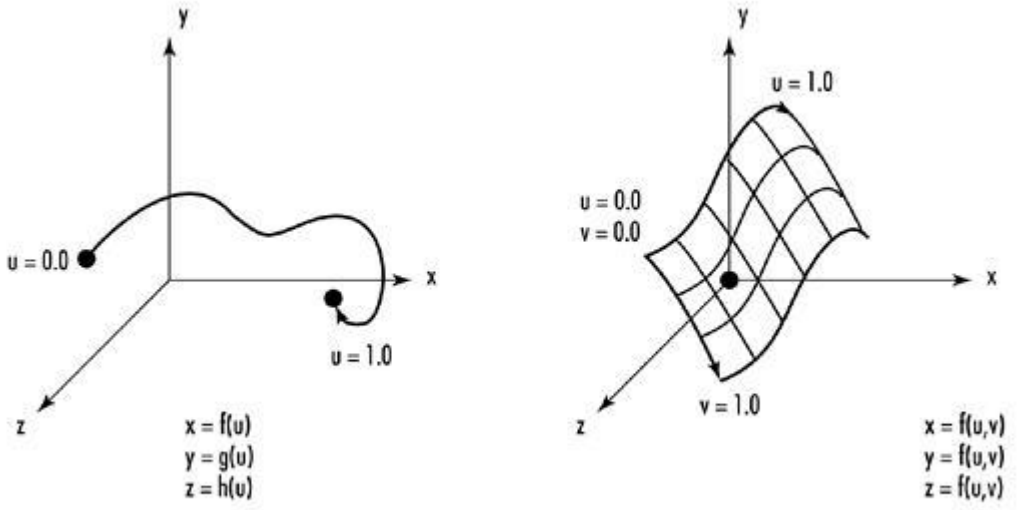


Figura 25: Representación paramétrica de curvas y superficies.

4.1.1 Puntos de control

Una curva se representa mediante un número de puntos de control que influyen en la forma de dicha curva. Para una curva de Bézier, el primer punto de control y el último, en realidad forman parte de la curva. Los otros puntos de control actúan como imanes, atrayendo la curva hacia ellos. La figura 26 muestra algunos ejemplos de este concepto con una cantidad variable de puntos de control.

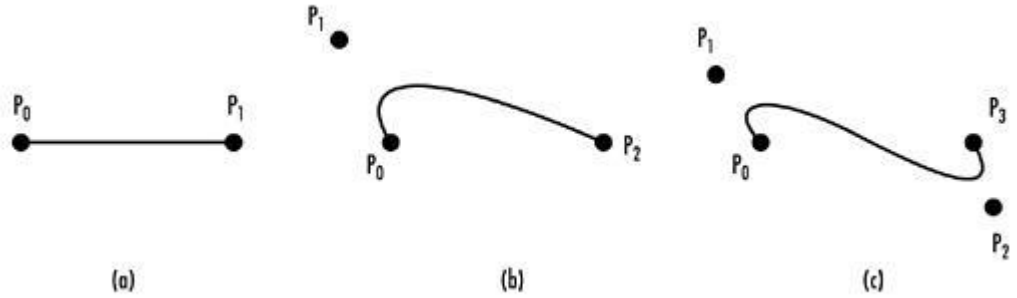


Figura 26: Afectación de los puntos de control en curvas

El orden de la curva se representa mediante la cantidad de puntos de control usados para describir su forma. El grado es uno menos el orden de la curva. El significado matemático de estos términos pertenece a las ecuaciones paramétricas que describen exactamente la curva, siendo el orden la cantidad de coeficientes y el grado el exponente más alto del parámetro paramétrico.

La curva de la figura 26(b) se denomina curva cuadrática (grado 2) y la figura 26(c) se denomina cubica (grado 3). Las curvas cúbicas son las más típicas. Teóricamente, podemos definir una curva de cualquier orden, pero curvas de orden superior empiezan a oscilar incontrolablemente y pueden variar mucho con el cambio más insignificante de los puntos de control.

4.1.2 Continuidad

Si dos curvas colocadas una junto a otra comparten un punto final (denominado punto de corte), ambas forman una curva de una pieza. La continuidad de dichas curvas en el punto de corte describe la suavidad de la transición entre ellas. Las cuatro categorías de continuidad son ninguna, posicional (C0), tangencial (C1) y curvatura (C2).
Como se puede comprobar en la figura 27, no se produce ninguna continuidad cuando dos curvas no se unen del todo. La continuidad posicional se consigue cuando dos curvas, al menos, se reúnen y comparten un punto de corte común. La continuidad tangencial se produce cuando las dos curvas tienen la misma tangente en el punto de corte.

Por último, la continuidad de curvatura significa que las tangentes de las dos curvas también tienen la misma tasa de cambio en el punto de corte (por lo tanto, una transición incluso más suave).

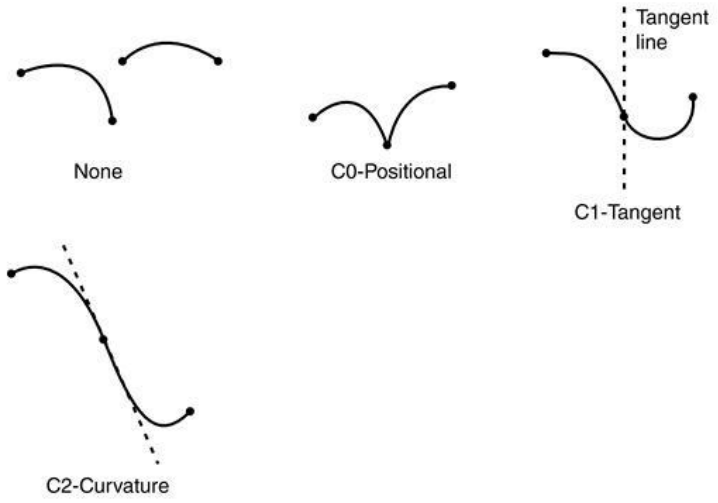


Figura 27: Continuidad

Cuando montamos superficies o curvas complejas a partir de muchas piezas, normalmente nos esforzamos por obtener una continuidad tangencial o de curvatura.

4.1.3 Evaluadores

OpenGL contiene varias funciones que facilitan el dibujo de curvas y superficies Bézier. Para dibujarlas, especificamos los puntos de control y el rango para los parámetros paramétricos u y v. A continuación, mediante una llamada a la función de evaluación apropiada (el evaluador), OpenGL genera los puntos que forman la curva o la superficie.

4.2 NURBS

Las NURBS, B-splines racionales no uniformes, son representaciones matemáticas de geometría en 3D capaces de describir cualquier forma con precisión, desde simples líneas

en 2D, círculos, arcos o curvas, hasta los más complejos sólidos o superficies orgánicas de forma libre en 3D. Gracias a su flexibilidad y precisión, se pueden utilizar modelos NURBS en cualquier proceso, desde la ilustración y animación hasta la fabricación.

La geometría NURBS tiene cinco cualidades esenciales que la convierten en la opción ideal para el modelado asistido por ordenador.
Existen varias formas estándar industriales para intercambiar la geometría NURBS. Los usuarios pueden y deberían ser capaces de transportar todos sus modelos geométricos entre los diferentes programas de modelado, renderizado, animación e ingeniería de análisis que hay en el mercado. Estos programas pueden almacenar información geométrica que podrá ser utilizada durante más de 20 años.

Las NURBS tienen una definición precisa y muy conocida. La geometría NURBS se enseña en las facultades de matemáticas e informática de las universidades más importantes. Eso significa que los vendedores de software especializado, los equipos de ingenieros, las empresas de diseño industrial y las empresas de animación que necesitan crear aplicaciones de software específicas para sus proyectos podrán encontrar programadores capacitados para trabajar con la geometría NURBS.
Las NURBS pueden representar con precisión objetos geométricos estándar tales como líneas, círculos, elipses, esferas y toroides, así como formas geométricas libres como carrocerías de coches y cuerpos humanos.
La cantidad de información que requiere la representación de una forma geométrica en NURBS es muy inferior a la que necesitan por separado las aproximaciones comunes.
La regla de cálculo de las NURBS, que se describe a continuación, se puede implementar en un ordenador de manera eficaz y precisa.

4.2.1 ¿Qué es la geometría NURBS?

Las curvas y superficies NURBS se comportan de maneras similares y comparten mucha terminología. Una curva NURBS se define mediante cuatro elementos: grados, puntos de control, nodos y regla de cálculo.

4.2.2 Grado

Un grado es un número entero positivo.

Este número normalmente es 1, 2, 3 o 5, pero puede ser cualquier número entero positivo. A veces se utilizan los siguientes términos: lineal, cuadrático, cúbico y quintico. Lineal significa de grado 1, cuadrático significa de grado 2, cúbico significa de grado 3 y quintico significa de grado 5.

Es posible que vea referencias del orden de una curva NURBS. El orden de una curva NURBS es un número entero positivo igual a (grado+1). En consecuencia, el grado es igual a orden-1.

Existe la posibilidad de incrementar los grados de una curva NURBS sin cambiar su forma. Generalmente, no es posible reducir el grado de una curva NURBS y no cambiar su forma.

4.2.3 Puntos de control

Los puntos de control son una lista de puntos de grado+1 como mínimo.

Una de las maneras más sencillas de cambiar la forma de una curva NURBS es mover los puntos de control.

Los puntos de control tienen un número asociado denominado peso. Con algunas excepciones, los pesos son números positivos. Cuando todos los puntos de control de una

curva tienen el mismo peso (normalmente 1), la curva se denomina no racional; de lo contrario, se trataría de una curva racional. En NURBS, la R significa racional e indica que una curva NURBS tiene la posibilidad de ser racional. A la práctica, la mayoría de las curvas NURBS son no-racionales. Algunas curvas, círculos y elipses NURBS, ejemplos significativos, son siempre racionales.

4.2.4 Nodos

Los nodos son una lista de números de grado N-1, donde N es el número de puntos de control. A veces esta lista de números se denomina vector nodal. En este contexto, la palabra vector no significa una dirección 3D.

Esta lista de números de nodos debe cumplir varias condiciones técnicas. El modo estándar para asegurar que las condiciones técnicas se cumplan es requerir que el número se mantenga igual o aumente a medida que vaya bajando en la lista y limitar el número de valores duplicados a que no sea superior al grado. Por ejemplo, para una curva NURBS de grado 3 con 15 puntos de control, la lista de números 0,0,0,1,2,2,2,3,7,7,9,9,9 es una lista de nodos satisfactoria. La lista 0,0,0,1,2,2,2,2,7,7,9,9,9 no es aceptable porque hay cuatro 2, y cuatro es un número mayor que el grado.

El número de veces que un valor nodal se duplica se denomina multiplicidad nodal. En el ejemplo anterior de lista satisfactoria de nodos, el valor nodal 0 tiene una multiplicidad de tres, el valor nodal 1 tiene una multiplicidad de uno, el valor nodal 2 tiene una multiplicidad de tres, el valor nodal 3 tiene multiplicidad de uno, el valor nodal 7 tiene una multiplicidad de dos y el valor nodal 9 tiene una multiplicidad de tres. Se dice que un valor nodal es un nodo de multiplicidad total si se multiplica por su grado varias veces. En el ejemplo, los valores de nodo 0, 2, y 9 tienen multiplicidad total. El valor de un nodo que aparece una sola vez se denomina nodo simple. En el ejemplo, los valores del nodo 1 y 3 son nodos simples.

Si una lista de nodos se inicia con un nodo de multiplicidad completa, la siguen nodos simples, termina con un nodo de multiplicidad completa y los valores se espacian uniformemente, entonces los nodos son uniformes. Por ejemplo, si una curva NURBS de grado 3 con 7 puntos de control tiene nodos 0,0,0,1,2,3,4,4,4, la curva tendrá nodos uniformes. Los nodos 0,0,0,1,2,5,6,6,6 no son uniformes. Los nodos que no son uniformes se denominan no uniformes. Las letras N y U de la palabra NURBS significan no uniforme e indican que los nodos de una NURBS puede ser no uniformes.

Los valores duplicados del nodo en la mitad de la lista del nodo hacen que una curva de NURBS sea menos suave. En caso extremo, un nodo de completa multiplicidad en la mitad de la lista de nodos significa que hay un lugar en la curva NURBS que se puede doblar en un punto de torsión. Por esta razón, a algunos diseñadores les gusta agregar y quitar nodos y luego ajustar los puntos de control para hacer curvas más suaves o figuras torsionadas. Debido a que el número de nodos es igual a (N+grado+1), donde N es el número de puntos de control, si se agregan nodos también se agregan puntos de control, y si se quitan nodos se quitan puntos de control. Los nodos se pueden añadir sin cambiar la forma de la curva de NURBS. En general, quitar nodos cambiará la forma de una curva.

4.2.5 Nodos y puntos de control

Un error frecuente se produce cuando cada nodo se empareja con un punto de control, y ocurre sólo en las NURBS de grado 1 (polilíneas). Para curvas NURBS de grados más altos, existen grupos de nodos de 2 x grado que corresponden a grupos de puntos de control de grado+1. Por ejemplo, suponga que tiene curvas NURBS de grado 3 con 7 puntos de control y nodos 0,0,0,1,2,5,8,8,8. Los primeros cuatro puntos de control están agrupados con los primeros seis nodos. Del segundo hasta el quinto punto de control están agrupados con los nodos 0,0,1,2,5,8. Del tercer al sexto punto de control están agrupados con los nodos 0,1,2,5,8,8. Los últimos cuatro puntos de control están agrupados con los últimos seis nodos.

Algunos modeladores que utilizan algoritmos más antiguos para el cálculo de curvas NURBS necesitan dos valores de nodos extra para un total de nodos grado+N+1.

4.2.6 Regla de cálculo

La regla de cálculo de una curva utiliza una fórmula matemática que coge un número y asigna un punto.
La regla de cálculo NURBS es una fórmula que comprende el grado, los puntos de control y los nodos. En la fórmula hay lo que se llama funciones básicas de B-spline. Las letras B y S de la palabra NURBS significan “basis spline”. El número de cálculo con que empieza la regla de cálculo se denomina parámetro. Puede imaginarse la regla de cálculo como una caja negra que se come un parámetro y produce un punto. El grado, los nodos y los puntos de control determinan el funcionamiento de la caja negra.

4.3 Curvas y superficies de bezier

Pierre Bézier, diseñador de automóviles para Renault en los años setenta, reconociendo esta necesidad fundamental en el arte de los gráficos generados por ordenador, creó un conjunto de modelos matemáticos que podían representar curvas y superficies mediante la especificación de un pequeño conjunto de puntos de control. Además de simplificar la representación de superficies curvas, los modelos facilitaban ajustes interactivos para la forma de la curva o la superficie.
Pronto comenzaron a evolucionar otros tipos de curvas y superficies y, de hecho, un nuevo y completo vocabulario para superficies generadas por computador. Las matemáticas que se esconden detrás de esta presentación mágica no son más complejas que las manipulaciones de matrices y es fácil adquirir un conocimiento intuitivo de estas curvas. Vamos a explicarle la solución para que pueda hacer mucho con estas funciones sin necesidad de adquirir un conocimiento profundo de sus matemáticas.

4.4 Utilización de extensiones de OPENGL

El estándar OpenGL permite a los fabricantes añadir nuevas funcionalidades adicionales mediante extensiones conforme aparecen nuevas tecnologías. Dichas extensiones pueden introducir nuevas funciones y constantes, y suavizar o incluso eliminar restricciones en funciones ya existentes. Cada fabricante dispone de una abreviatura que le identifica en el nombre de sus nuevas funciones o constantes. Por ejemplo, la abreviatura de NVIDIA (NV) aparece en la definición de su función glCombinerParameterfvNV() y su constante GL_NORMAL_MAP_NV.

Es posible que varios fabricantes se pongan de acuerdo en implementar la misma funcionalidad extendida. En ese caso, se usa la abreviatura EXT. Incluso puede ocurrir que el ARB adopte la extensión, convirtiéndose así en estándar y utilizando la abreviatura ARB en sus nombres. La primera extensión ARB fue GL_ARB_multitexture, presentada en la versión 1.2.1. Siguiendo el camino marcado por la extensión, el multitexturing no es ya una extensión opcional, sino que entró a formar parte del núcleo de OpenGL desde la versión 1.3.

Antes de usar una extensión, los programas deben comprobar su disponibilidad y, después, acceder a las nuevas funcionalidades ofrecidas. Este proceso es dependiente de la plataforma, pero bibliotecas como GLEW y GLEE lo simplifican.

Para obtener acceso a una función de extensión que no es parte de la biblioteca estándar OpenGL, se llama a la función wglGetProcAddress. Si existe la función de extensión en la implementación actual, a continuación, wglGetProcAddress proporciona un puntero a función que puede utilizar para tener acceso a la función. WglGetProcAddress devuelve NULL de lo contrario.

Por ejemplo, para obtener acceso a la función de extensión glAddSwapHintRectWIN, se llama a wglGetProcAddress como sigue:

```
// Get a pointer to the extension function.
```

```
typedef void (WINAPI *FNSWAPHINT)(GLint, GLint, GLsizei, GLsizei);
fnSwapHint = (FNSWAPHINT)wglGetProcAddress("glAddSwapHintRectWIN");
// Actual call to glAddSwapHintRectWIN.
if (fnSwapHint != NULL)
   (*fnSwapHint)(0, 0, 100, 100);
```

CAPÍTULO V

RESULTADOS EXPERIMENTALES

CAPITULO V

5. RESULTADOS EXPERIMENTALES

5.1 Cálculos para la representación de tubería

Se utiliza la ecuación paramétrica del círculo para generar la ubicación de los puntos alrededor de la tubería en la posición dada por *x*, *y*, *z*. En donde cada uno de los puntos de un círculo de radio "*r*", se le suma el desplazamiento registrado en cada uno de los brazos abatibles, ecuación 1 y 2.

$$x(\theta) = R \cdot \cos(\theta) + j \qquad \text{Ec. 1}$$

$$y(\theta) = R \cdot \sin(\theta) + k \qquad \text{Ec. 2}$$

Donde:

$R = r$ radio nominal de la tubería + desplazamiento detectado por brazo abatible n.

j, k = centro del circulo.

θ = ángulo de ubicación de los brazos abatibles.

Con las ecuaciones 1 y 2 es posible saber el estado de la tubería alrededor de los sensores de desplazamiento, los cuales se encuentran ubicados en cierta posición *X*, *Y*, *Z*, coordenadas que pueden y deberán ser proporcionadas por el sistema de navegación inercial (ver figura 28) el cual es parte fundamental del cualquier diablo de inspección.

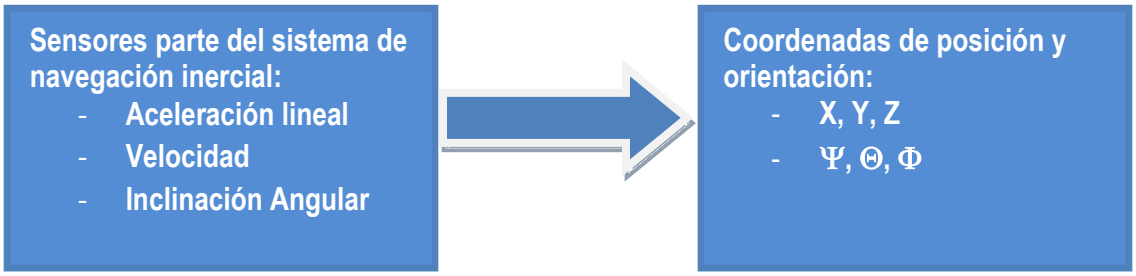


Figura 28: Sensores utilizados y coordenadas arrojadas por el Sistema de Navegación Inercial.

Las coordenadas cartesianas X,Y,Z, permiten conocer la posición central del diablo geómetra, durante el recorrido realizado, por lo que a partir de este punto central es posible representar la circunferencia de la tubería con el daño geométrico si así fue detectado por los sensores LVDT (Linear variable differential transformer).

Una vez que es identificado el centro del diablo instrumentado, es necesario conocer la orientación del mismo, ya que durante el trayecto de inspección el diablo geómetra se encontrará seguramente con curvas de hasta 3 veces el diámetro de la tubería, así como también girará sobre su propio eje transversal, debido al flujo y la presión del combustible que trasporte los ductos. Esta información de inclinación y orientación, son proporcionadas a través de las coordenadas llamadas Yaw, Pitch y Roll, así como se muestra a continuación (ver figura 29):

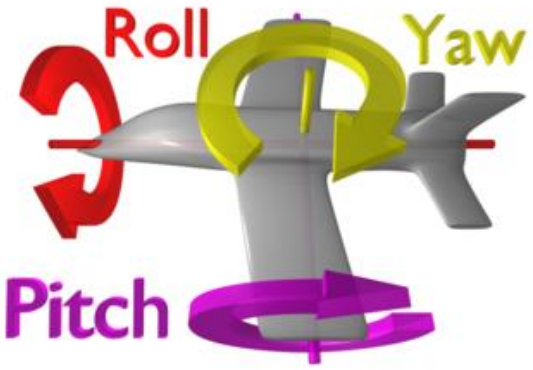


Figura 29: Coordenadas de orientación del diablo geómetra.

5.2 Almacenamiento de los datos en memoria.

El diablo geómetra construido por CIDESI utiliza como medio de almacenamiento, memorias de tipo flash, específicamente memorias SD de 1GB. La manera de almacenar los datos en estas memorias es a través del protocolo de comunicación SPI (Serial Peripherical Interfase) y teniendo una estructura de almacenamiento en la memoria de tipo FAT16 (File Allocation Table), la cual proporciona la facilidad de que se almacene la información de modo que la memoria pueda ser leída por una computadora con sistema operativo de tipo DOS (Disk Operating System) o Windows.

Para realizar este tipo de almacenamiento es necesario preparar todas las memorias utilizadas en el diablo formateándolas en modo FAT16, esto es realizado en una computadora.

Una vez que se ha realizado lo anterior con las memorias, es necesario que un micro controlador con capacidad de comunicación SPI, envíe las señales necesarias para escribir o leer información en la memoria.

5.3 Datos adquiridos desde archivo

Los archivos de datos donde se concentra la información una vez que se ha extraído de las memorias del diablo geómetra y que se ha realizado el algoritmo de integración tienen la siguiente organización, ver figura 30:

		Coordenadas GPS			Coordenadas cartesianas			Coordenadas de orientación			Detección de defectos
		Latitud	Longitud	Altitud	X	Y	Y	Yaw	Pitch	Roll	22 sensores de desplazamiento
Tiempo ó desplazamiento	1	…	…	…	…	…	…	…	…	…	…
	2	…	…	…	…	…	…	…	…	…	…
	3										

Figura 30: Organización de los archivos de información

Esta organización permite saber en determinado momento en qué posición se encontraba el diablo geómetra, así como la posición de sus sensores de desplazamiento para conocer si hubo o no un defecto geométrico en la tubería.

La estructura utilizada para adquirir y almacenar la información es la siguiente:

```
struct sPoint{
    float latitud, longitud, altitud;
    float x, y, z;
    float fi,theta,psi;
     float s[22];
};
```

La representación de una tubería con la ayuda de OpenGL puede hacerse con el uso de las instrucciones:

- glVertex (x, y, z) para la colocación de los vértices que componen el contorno de la tubería inspeccionada (ver figura 31).
  Donde x = x(θ), y=y(θ)

Para *theta=0°* hasta *360°*, con incrementos de *16.36°*, considerando que no hay defectos que se puedan identificar con *R*. glVertex3d (R*cos(theta)+j, R*sin(theta)+k, 0);	

Figura 31: Generación de vértices sobre una circunferencia a partir de la información de los sensores

- glTranslated (0,0,z) utilizado para la ubicación y reconstrucción de los diferentes contornos obtenidos de la tubería (ver figura 32).

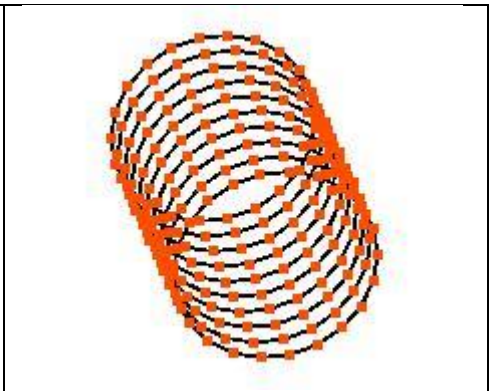
La variable "*d*" indica la distancia recorrida del diablo geómetra entre cada que se registra la información de los sensores.
 	
..
glVertex3d(R*cos(theta)+j,R*sin(theta)+k,0);
..
..
glTranslated(0,0,d);
	

Figura 32: Generación de vértices representando una tubería.

- glRotated (Yaw, Pitch, Roll) utilizado para dar la orientación de la información de los contornos de la tubería (ver figura 33).

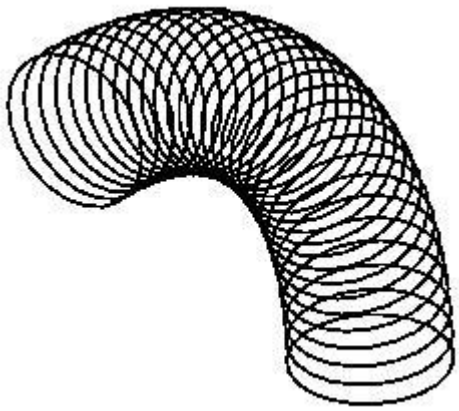
Para este ejemplo *Pitch* y *Roll* no varían, *Yaw* varia en *3°* por cada circunferencia que se representa.
 	
..
glVertex3d(R*cos(theta)+j,R*sin(theta)+k,0);
..
..
glRotated(yaw,1,0,0);
glRotated(pitch,0,1,0);
glRotated(roll,0,0,1);
glTranslated(0,0,d);
	

Figura 33: Generación de vértices representando un codo en la tubería virtual.

Aunque estas funciones de OpenGL permiten realizar transformaciones de rotación, translación, etc. Se utilizan matrices de transformación para realizar las respectivas operaciones realizando el trabajo a "mano", para contar con las matrices de posición. Realizar solo una vez los cálculos de rotación y translación y poder representar los tramos de tubería solo los de interés.

Esta parte es clave para la manipulación y actualización de las vistas de las tuberías que se generan.

Para realizar esto se sigue el siguiente procedimiento:
Iniciar la Matriz identidad,

$$Matriz\ Act = Matriz\ New = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dependiendo de la orientación que haya tomado el diablo geómetra, de acuerdo a la información almacenada y procesada, se va actualizando la matriz "Matriz New", para conocer la posición del diablo geómetra y poder reconstruir la tubería.

La forma en actualizar la matriz es realizando las operaciones con matrices respectivas, cuando se trate de una rotación o una translación.

La translación representa el avance del diablo geómetra hacia adelante, las rotaciones representan las orientaciones que haya tomado el diablo geómetra en los ejes X, Y, Z

```
Mtranslat(0,0,Avance hacia adelante);
Mrotatex(Valor de fi en el instante t);
Mrotatey(Valor de theta en el instante t);
Mrotatez(Valor de psi en el instante t);
```

.

.

.

```
PVertixx[ncircle][nsens]=MATRIZ_NEW[0][0]*x + MATRIZ_NEW[0][1]*y + MATRIZ_NEW[0][2]*z + MATRIZ_NEW[0][3]*1;
PVertixy[ncircle][nsens]=MATRIZ_NEW[1][0]*x + MATRIZ_NEW[1][1]*y + MATRIZ_NEW[1][2]*z + MATRIZ_NEW[1][3]*1;
```

PVertixz[ncircle][nsens]=MATRIZ_NEW[2][0]*x + MATRIZ_NEW[2][1]*y + MATRIZ_NEW[2][2]*z + MATRIZ_NEW[2][3]*1;

5.4 Suavizado y detallado de la superficie

Una vez que se tienen los vértices es posible la realización de una representación con mayor detalle de la tubería, una de las formas para hacer esto es la generación de más vértices para realizar un mallado mucho más fino, tal como se muestra en la figura 34:

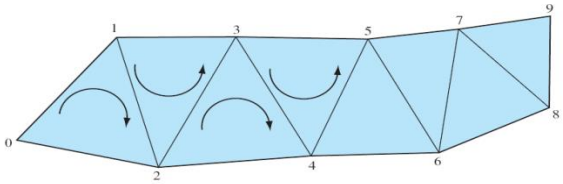


Figura 34: Conexión de vértices.

Este método consiste en dividir en la mitad cada uno de los lados de los triángulos que conectan a tres de los vértices, para considerar ahora esos puntos como nuevas coordenadas y realizar un mallado entre ellos, ver figura 35.

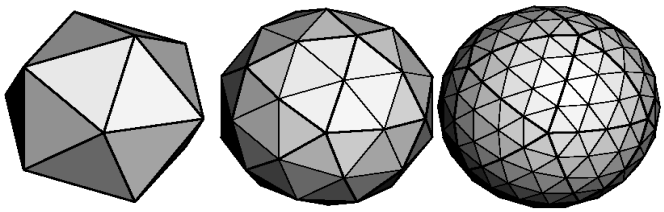


Figura 35: suavizado de superficies con el aumento de vértices intermedios.

Una zona que se encuentra fuera del diámetro nominal, se le asigna un color a la tubería virtual en esa zona que identifique el grado de defecto detectado, para esto es necesario determinar cuál es el color que representa el máximo grado de defecto, es decir aquel en el que un brazo abatible sea desplazado hasta su máxima capacidad, así sea por una disminución en el diámetro de la tubería o un aumento en el mismo.

Aunque se puede escoger cualquier color para los defectos, para este caso de estudio se decidió representar con un color gris (R=0.29, V=0.29, A=0.29), la tubería cuando se

encuentra dentro de su diámetro nominal y con un aumento o disminución del 10% del máximo que puede representar los brazos abatibles (ver figura 36).

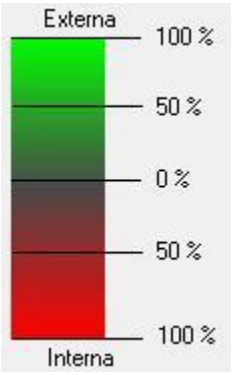


Figura 36: Escala de colores para representar los defectos.

Cuando cualquiera de los brazos abatibles registra una variación mayor al 10% de su máximo desplazamiento, el color Gris se va transformando hacia un Verde (R=0, V=1, A=0); si el defecto es hacia el exterior y se transforma hacia un Rojo (R=1,V=0, A=0), si el defecto es hacia el interior, ver figura 37.

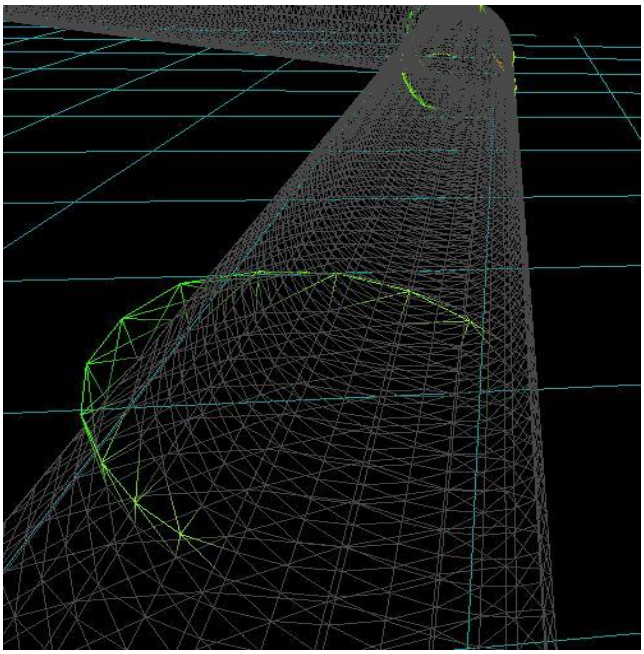


Figura 37: Representación de un defecto con escala de colores.

5.5 Movimiento en el espacio de trabajo.

Para poder desplazarse en la escena virtual e ir recorriendo la tubería virtual reconstruida, se definen y permiten los siguientes movimientos; adelante y atrás, arriba y abajo, izquierda y derecha y rotación sobre el eje Y y sobre el eje Z visto como se muestra en la figura 38:

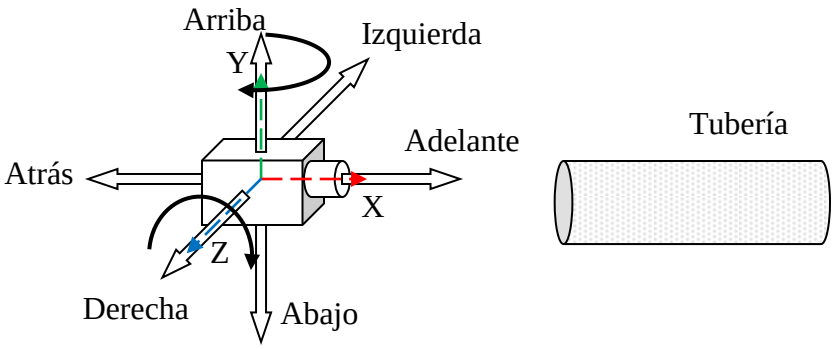


Figura 38: Movimientos posibles del usuario con la cámara virtual.

5.6 Presentación de resultados a través del desarrollo de la tesis.

A continuación se muestran diferentes vistas de lo que ha evolucionado el software para la presentación de resultados del diablo geómetra durante el desarrollo de este trabajo de tesis, principalmente en la parte de presentación de los resultados.

La primer imagen (figura 39) muestra una aplicación desarrollada para reconstruir una pequeña tubería con varias curvas y defectos, estos últimos representados con colores verde y rojo, dependiendo si se detectaban hacia adentro o hacia afuera respectivamente. Con la posibilidad de hacer clic sobre varios botones para moverse a través del espacio de trabajo, mostrando las coordenadas X, Y y Z de la posición relativa de la cámara virtual.

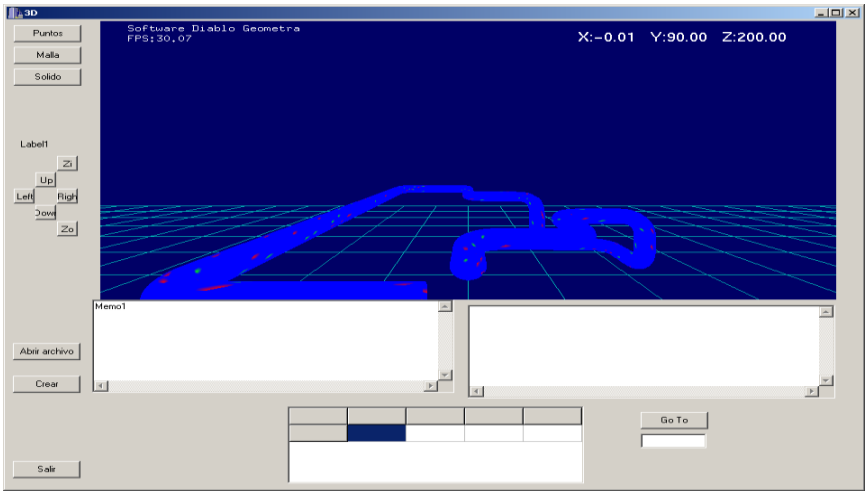


Figura 39: Primera generación de una tubería virtual a través del desarrollo de una aplicación de software.

En la figura 40 se muestra una aplicación para la reconstrucción de la tubería virtual con la posibilidad de animar un modelo real del diablo geómetra desarrollado. Moviéndose a través de la tubería según la información de posición y orientación adquirida. Sin embargo esto resultaba más una animación que una simulación por lo que se evitó tener esta posibilidad en el software ya que al realizar esto se consume bastantes recursos de una computadora lo cual significaría limitar su uso a equipos de cómputo de alto rendimiento.

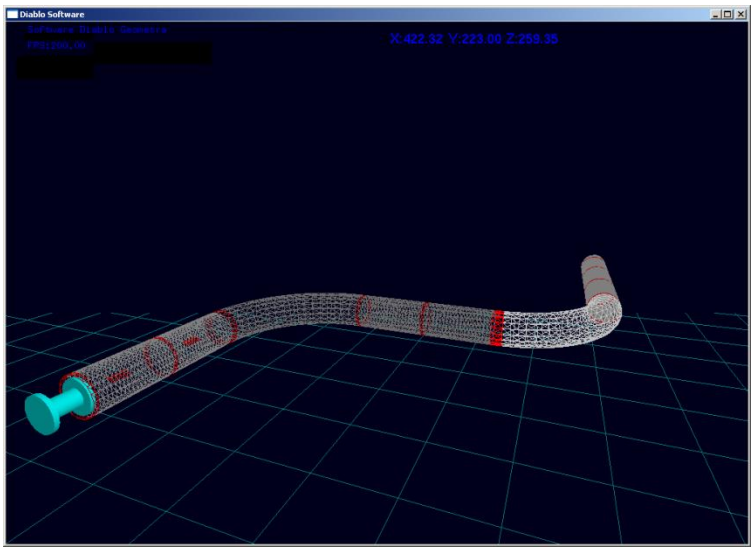


Figura 40: Realización de una animación de recorrido del diablo sobre una tubería reconstruida.

5.7 Publicacaciones

- L. Barriga Rodríguez, J. C. Pedraza Ortega, ***"Reconstrucción Virtual en 3D de tubería inspeccionada mediante un diablo geómetra"***, Cuarto Congreso Internacional de Ingeniería, Abril del 2008 Querétaro Qro.

- N. Tovar, C. Rubio Gonzalez, L. Barriga Rodríguez, J. C. Pedraza, "Adquisición y procesamiento de micro-deformaciones en pruebas dinámicas utilizando LabVIEW", Sexto Congreso Internaciónal Sobre Innovación y Desarrollo Tecnológico (CIINDET 2008), Octubre 2008. Cuarnavaca, Mor.

- J. C. Pedraza Ortega, J. A. Soto Cajiga, J. M. Ramos Arreguin, L. Barriga Rodríguez, F. Hernandez, A. Rivas Velazquez. ***"Digitalizador de sólidos utilizando FTP modificado"***, Tercer Congreso Internacional de Ingeniería, Abril del 2007 Querétaro Qro.

- J. C. Pedraza Ortega, J. A. Soto Cajiga, J. M. Ramos Arreguin, L. Barriga Rodríguez, F. Hernandez, A. Rivas Velazquez. ***"Image Processing for 3D Reconstruction using a Modified Fourier Transform Profilometry Method"***, MICAI, Octubre 2007, ISBN 978-3-540-76630-8.

CAPÍTULO VI

TRABAJO FUTURO

CAPITULO VI

6. TRABAJO FUTURO

El paso inmediato es la comprobación de los resultados obtenidos en un banco de pruebas donde pueda circular el diablo geómetra y detectar una serie de diferentes defectos en la tubería con una ubicación conocida para comparar la visualización obtenida a través de la tubería virtual con la tubería verdadera.

La representación de tuberías a través de un diablo geómetra se limita a la representación de defectos geométricos que pueden ser detectados en su interior a través de los sensores distribuidos alrededor del diablo. Aunque esta información es de suma importancia, es necesario enriquecerla mediante el análisis de la tubería con un diablo instrumentado.

El uso de un diablo instrumentado (con sensores ultrasónicos), además de contar con número de sensores mayor al del diablo geómetra, también permite conocer el espesor de una tubería, si existe un defecto externo o interno, por lo que el software para representar esos defectos deberá incluir la representación de una tubería con espesor y de la identificación de otro tipo de defectos, como pueden ser grietas, picaduras, laminaciones, entre muchas otras, esto sugiere un procesamiento de una cantidad mayor de información desde su análisis hasta su representación de manera virtual, en la siguiente figura se muestra un ejemplo en esta representación.

También la realización de un software que permita llevar un registro de las diferentes corridas que ha hecho uno o diferentes diablos con el paso del tiempo para verificar que la afectación de los mismos tramos de tubería con respecto del tiempo y presente un informe de daños ocurridos entre una corrida y otra.

CAPÍTULO VII

CONCLUSIONES

CAPITULO VII

7. CONCLUSIONES

Se tiene un software para registrar, procesar y presentar en pruebas de laboratorio e incluso en pruebas de campo los resultados de los recorridos que realice el diablo geómetra, diseñado por CIDESI.

Se logra representar una buena cantidad de puntos de forma visual que permite al usuario ver el estado de una tubería mediante código de colores y de una manera interactiva, ya que es posible utilizar ya sea un joystick o simplemente el teclado de la computadora para realizar un recorrido virtual a través de toda la tubería que se ha inspeccionado.

Se tiene un punto de partida para iniciar trabajos en presentación de resultados de mayor resolución con diablos de ultrasonido. Además de la utilización de otro tipo de mapas digitales para conocer la ubicación en un sistema georeferenciado.

ANEXOS Y REFERENCIAS

8. ANEXOS

8.1 SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)

El término SIG procede del acrónimo de Sistema de Información Geográfica (en inglés GIS, Geographic Information System).

Técnicamente se puede definir un SIG como una tecnología de manejo de información geográfica formada por equipos electrónicos (hardware) programados adecuadamente (software) que permiten manejar una serie de datos espaciales (información geográfica) y realizar análisis complejos con éstos siguiendo los criterios impuestos por el equipo científico (personal).

Son por tanto cuatro los elementos constitutivos de un sistema de estas características:

1. Hardware.
2. Software.
3. Datos geográficos.
4. Equipo humano.

Aunque todos ellos han de cumplir con su cometido para que el sistema sea funcional, existen diferencias en cuanto a su importancia relativa. A lo largo del tiempo, el peso de cada uno de los elementos dentro de un proyecto S.I.G. ha ido cambiando mostrando una clara tendencia: mientras los equipos informáticos condicionan cada vez menos los proyectos S.I.G. por el abaratamiento de la tecnología, los datos geográficos se hacen cada vez más necesarios y son los que consumen hoy día la mayor parte de las inversiones en términos económicos y de tiempo.

Así, hoy día el condicionante principal a la hora de afrontar cualquier proyecto basado en SIG lo constituye la disponibilidad de datos geográficos del territorio a estudiar, mientras que hace diez años lo era la disponibilidad de ordenadores potentes que permitieran afrontar los procesos de cálculo involucrados en el análisis de datos territoriales.

Pero además de ser un factor limitante, la información geográfica es a su vez el elemento diferenciador de un Sistema de Información Geográfica frente a otro tipo de Sistemas de Información; así, la particular naturaleza de este tipo de información contiene dos vertientes diferentes: por un lado está la vertiente espacial y por otro la vertiente temática de los datos.

Mientras otros Sistemas de Información (como por ejemplo puede ser el de un banco) contienen sólo datos alfanuméricos (nombres, direcciones, números de cuenta, etc.), las bases de datos de un S.I.G. han de contener además la delimitación espacial de cada uno de los objetos geográficos.

Por ejemplo, un lago que tiene su correspondiente forma geométrica plasmada en un plano, tiene también otros datos asociados como niveles de contaminación. Pongamos otro ejemplo para que esto se entienda mejor: supongamos que tenemos un suelo definido en los planos de clasificación de un planeamiento urbanístico como "urbanizable". Este suelo urbanizable tiene una serie de atributos, tales como su uso, su sistema de gestión, su edificabilidad, etc. Pero es que además, el urbanizable tiene una delimitación espacial concreta correspondiente con su propia geometría definida en el plano.

Por tanto, el SIG tiene que trabajar a la vez con ambas partes de información: su forma perfectamente definida en plano y sus atributos temáticos asociados. Es decir, tiene que trabajar con cartografía y con bases de datos a la vez, uniendo ambas partes y constituyendo con todo ello una sola base de datos geográfica.

Esta capacidad de asociación de bases de datos temáticas junto con la descripción espacial precisa de objetos geográficos y las relaciones entre los mismos (topología) es lo que diferencia a un SIG de otros sistemas informáticos de gestión de información.

8.2 LA CONSTRUCCIÓN DE BASES DE DATOS GEOGRÁFICAS.

La construcción de una base de datos geográfica implica un proceso de abstracción para pasar de la complejidad del mundo real a una representación simplificada asequible para el lenguaje de los ordenadores actuales. Este proceso de abstracción tiene diversos niveles -

como iremos viendo- y normalmente comienza con la concepción de la estructura de la base de datos, generalmente en capas; en esta fase, y dependiendo de la utilidad que se vaya a dar a la información a compilar, se seleccionan las capas temáticas a incluir.

Pero la estructuración de la información espacial procedente del mundo real en capas conlleva cierto nivel de dificultad. En primer lugar, la necesidad de abstracción que requieren las máquinas implica trabajar con primitivas básicas de dibujo, de tal forma que toda la complejidad de la realidad ha de ser reducida a puntos, líneas o polígonos.

En segundo lugar, existen relaciones espaciales entre los objetos geográficos que el sistema no puede obviar; es lo que se denomina topología, que en realidad es el método matemático-lógico usado para definir las relaciones espaciales entre los objetos geográficos.

Aunque a nivel geográfico las relaciones entre los objetos son muy complejas, siendo muchos los elementos que interactúan sobre cada aspecto de la realidad, la topología de un S.I.G. reduce sus funciones a cuestiones mucho más sencillas, como por ejemplo conocer el polígono (o polígonos) a que pertenece una determinada línea, o bien saber qué agrupación de líneas forman una determinada carretera.

Existen diversas formas de modelizar estas relaciones entre los objetos geográficos o topología. Dependiendo de la forma en que ello se lleve a cabo se tiene uno u otro tipo de Sistema de Información Geográfica dentro de una estructura de tres grupos principales:

- S.I.G. Vectoriales.
- S.I.G. Raster.
- S.I.G. Orientados a Objetos.

No existe un modelo de datos que sea superior a otro, sino que cada uno tiene una utilidad específica, como veremos a continuación.

8.3 TOPOLOGÍAS, MODELOS DE DATOS Y TIPOS DE SIG

En función del modelo de datos implementado en cada sistema, podemos distinguir tres grandes grupos de Sistemas de Información Geográfica: SIG Vectoriales, SIG Raster y SIG con modelo de datos Orientados a Objetos. En realidad, la mayor parte de los sistemas existentes en la actualidad pertenecen a los dos primeros grupos (vectoriales y raster).

Aunque veremos posteriormente las diferencias entre ambos con más detalle, adelantaremos que los vectoriales utilizan vectores (básicamente líneas), para delimitar los objetos geográficos, mientras que los raster utilizan una retícula regular para documentar los elementos geográficos que tienen lugar en el espacio.

8.3.1 Los SIG Vectoriales

Son aquellos Sistemas de Información Geográfica que para la descripción de los objetos geográficos utilizan vectores definidos por pares de coordenadas relativas a algún sistema cartográfico.

Con un par de coordenadas y su altitud gestionan un punto (e.g. un vértice geodésico), con dos puntos generan una línea, y con una agrupación de líneas forman polígonos. De entre todos los métodos para formar topología vectorial la forma más robusta es la topología arco-nodo, cuya lógica de funcionamiento trataré de detallar en los siguientes esquemas:

La topología arco-nodo basa la estructuración de toda la información geográfica en pares de coordenadas, que son la entidad básica de información para este modelo de datos. Con pares de coordenadas (puntos) forma vértices y nodos, y con agrupaciones de éstos puntos forma líneas, con las que a su vez puede formar polígonos. Básicamente esta es la idea, muy sencilla en el fondo.

Para poder implementarla en un ordenador, se requiere la interconexión de varias bases de datos a través de identificadores comunes. Estas bases de datos, que podemos imaginarlas

como tablas con datos ordenados de forma tabular, contienen columnas comunes a partir de las cuales se pueden relacionar datos no comunes entre una y otra tabla.

Hemos visto en el esquema anterior cómo se forman las líneas a partir de puntos (pares de coordenadas). Veámos ahora cómo se forman los polígonos a partir de la agrupación de líneas:

En general, el modelo de datos vectorial es adecuado cuando trabajamos con objetos geográficos con límites bien establecidos, como pueden ser fincas, carreteras, etc.

8.3.2 Los SIG Raster

Los Sistemas de Información Raster basan su funcionalidad en una concepción implícita de las relaciones de vecindad entre los objetos geográficos. Su forma de proceder es dividir la zona de afección de la base de datos en una retícula o malla regular de pequeñas celdas (a las que se denomina pixels) y atribuir un valor numérico a cada celda como representación de su valor temático. Dado que la malla es regular (el tamaño del pixel es constante) y que conocemos la posición en coordenadas del centro de una de las celdas, se puede decir que todos los pixels están georreferenciados.

Lógicamente, para tener una descripción precisa de los objetos geográficos contenidos en la base de datos el tamaño del pixel ha de ser reducido (en función de la escala), lo que dotará a la malla de una resolución alta. Sin embargo, a mayor número de filas y columnas en la malla (más resolución), mayor esfuerzo en el proceso de captura de la información y mayor costo computacional a la hora de procesar la misma.

No obstante, el modelo de datos raster es especialmente útil cuando tenemos que describir objetos geográficos con límites difusos, como por ejemplo puede ser la dispersión de una nube de contaminantes, o los niveles de contaminación de un acuífero subterráneo, donde los contornos no son absolutamente nítidos; en esos casos, el modelo raster es más apropiado que el vectorial.

8.3.3 Los SIG Orientados a Objetos

No existe una definición clara ni un acuerdo general en la comunidad de usuarios acerca de la entidad de los modelos orientados a objetos, pero sí existe unanimidad en cuanto a las características que debe tener un S.I.G. de este tipo.

En primer lugar, los S.I.G. orientados a objetos plantean un cambio en la concepción de la estructura de las bases de datos geográficas; mientras los modelos de datos vectorial y raster estructuran su información mediante capas -como ya hemos dicho anteriormente- los sistemas orientados a objetos intentan organizar la información geográfica a partir del propio objeto geográfico y sus relaciones con otros. De este modo, los objetos geográficos están sometidos a una serie de procesos y se agrupan en clases entre las cuales se da la herencia.

En segundo lugar, los S.I.G. orientados a objetos introducen un carácter dinámico a la información incluida en el sistema, frente a los modelos de datos vectoriales y raster que tienen un carácter estático.

Por ello, el modelo orientado a objetos es más aconsejable para situaciones en las que la naturaleza de los objetos que tratamos de modelar es cambiante en el tiempo y/o en el espacio.

Para poner un ejemplo de organización de la información con este modelo de datos, pensemos en un subcompartimento forestal, dentro del cual se dan muchos árboles, cada uno de ellos sometido a unos procesos (por ejemplo el crecimiento); este crecimiento es heredado por el subcompartimento y da como resultado que la altura del mismo sea cambiante con el tiempo.

Por lo tanto, en este caso los atributos temáticos de cada objeto geográfico son el resultado de aplicar unas determinadas funciones que varían según las relaciones del objeto de referencia con su entorno.

Sin duda alguna, este modelo de datos es más aconsejable que cualquier otro para trabajar con datos geográficos, pero se encuentra con dificultades de implementación en los actuales Sistemas de Gestión de Bases de Datos (SGBD), y por lo tanto también con dificultades de implementación en los S.I.G. Hoy en día comienzan a verse implementaciones de este tipo de organización de datos en algunos GIS comerciales, si bien son a nuestro entender aproximaciones en cierto modo incompletas cuya su funcionalidad tiene que ser mejorada en los siguientes años.

La ventaja fundamental que permite esta estructura de datos frente a las demás es la dinamicidad de los datos. Es decir, a partir de una serie de parámetros establecidos en el comportamiento de los objetos geográficos, podemos simular su evolución futura, lo que constituye un gran avance si se trabaja en entornos en los que se requiere simulación de situaciones potenciales.

9. REFERENCIAS

[1] Point Based Rendering for Massive Data Sets: A Case Study. Stephan Mantler Anton L. Fuhrmann, VRVis Center for Virtual Reality and Visualization Vienna, Austria, 2004

[2] Manipulation of Volume & Graphics Objects for the Visualization of Medical Data under OpenGL, Zhenghong Lee, Pedro Diaz, Errol Bellon; Biomedical Engineering Department Case Westem Reserve University, 1996

[3] A Robotic Laser Pipeline Profiler, Martin Johnson, G. Sen Gupta, IIMS, Massey University, Albany, Auckland, New Zealand, IIS&T, Massey University, Palmerston North, New Zealand.

[4] NAVIGATION SYSTEM FOR AUTOMATION OF PIPELINES INSPECTION MISSIONS, *Sergiy Sadovnychiy1, Volodymyr Ponomaryov, Tomás Ramírez, Eduardo Herrera*, Gerencia de Geofísica de Explotación, Instituto Mexicano del Petróleo,

[5] COMPUTATION OF PIPELINE-BENDING STRAINS BASED ON *GEOPIG* MEASUREMENTS, By Dr. Jaroslaw A. Czyz and John R. Adams Nowsco Pipeline Services, Calgary, Alberta, Canada

[6] PREVENTION OF PIPELINE FAILURES IN GEOTECHNICALLY UNSTABLE AREAS BY MONITORING WITH INERTIAL AND CALIPER IN-LINE INSPECTION, Jaroslaw Czyz, BJ Pipeline Inspection Services

[7] MULTI-PIPELINE GEOGRAPHICAL INFORMATION SYSTEM BASED ON HIGH ACCURACY INERTIAL SURVEYS, Jaroslaw A. Czyz and Chris Pettigrew BJ Pipeline Inspection Services Calgary, Alberta, Canada.

[8] Programación OpenGL, Richard S. Wright, Benjamin Lipchak Ed. Anaya Multimedia.

[9] Análisis Numérico, Richard L. Burden Ed. Thomson Learning. Pp 146-149

[10] Curvas y superficies para modelado geométrico, Juan M. Cordero Valle, Ed. Alfaomega. Capitulo 6, 7.

[11] High accuracy pipeline depth of cover survey in channel crossing using inertial navigation Jaroslaw A. Czyz, BJ pipeline inspection service.

[12] Monitoring pipeline movement and its effect on pipe integrity using inertial/caliper in-line inspection. Jaroslaw A. Czyz, Sergio E. Wainselboin, PIPELINE

[13] Monitoring pipeline movement and its effect on pipe integrity using inertial/caliper in-line inspection. Jaroslaw A. Czyz, Sergio E. Wainselboin, BJ pipeline inspection service.

[14] Computation of pipeline-bending strains based on geopig measurements Dr. Jaroslaw A. Czyz and John R. Adams, Nowsco Pipeline Services, Calgary, Alberta, Canada