UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ANÁLISIS Y DISEÑO DE UN SISTEMA DE MODELADO Y ANIMACIÓN 3D GPL

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

(Computación)

P R E S E N T A:

MARCELO PÉREZ MEDEL

DIRECTORA DE LA TESIS: DRA. MARÍA GARZA DE JINICH

MÉXICO, D.F. **2002.**

Agradecimientos:

A Dios: Por la vida y la oportunidad. Por la libertad y la posibilidad. Y por él.

A mis padres Marcelo Pérez Arroyo y Raquel Medel Morales ya que en gran medida soy lo que ellos lograron con un gran esfuerzo.

A Carmen Delia: Porque ha sido y es mi compañera en la vida.

A mis hijos Marcelo Daniel, Dulce Raquel y Lucía Alexandra porque ellos son mi gran motivación y espero ser un gran ejemplo para ellos.

A la UNAM, que es la institución que me ha formado en el ámbito académico y es mi segunda casa.

A mis profesores del IIMAS, y de manera especial al Dr. Enrique Daltabuit, a la Dra. María Garza y al Dr. Fernando Gamboa.

Al Centro de Instrumentos por las facilidades prestadas para utilizar el laboratorio de usabilidad para las pruebas requeridas por este trabajo.

Al Laboratorio de Computo del Centro Tecnológico Aragón y sus tesistas quiénes son parte fundamental de este proyecto.

Índice

Índice de tablas y figuras

Figuras

Tablas

Introducción

En la actualidad existen varias herramientas que permiten el modelado y animación en tercera dimensión, sin embargo la mayoría están orientadas a usos comerciales, principalmente publicidad, además de ser herramientas muy costosas.

Estos sistemas no son aptos para visualización científica o de ingeniería, ya que buscan realismo visual más que fidelidad de modelado o en los movimientos, a pesar de que el realismo logrado sería mayor empleando ecuaciones y modelos físicos.

El objetivo de este trabajo es diseñar un software libre para modelado y animación tridimensional, que sea fácil de utilizar. Existen muchos trabajos relacionados con el modelado y la animación y éstos son citados dentro de este trabajo (principalmente en el capítulo 4) y también trabajos sobre "diseño centrado en el usuario" (Explicado en el capítulo 1) Sin embargo se ha escrito muy poco referente al software libre de forma seria.

Uno de los mejores análisis respecto a software libre se encuentra en el texto *The Cathedral & the Bazaar* de Erick S. Raymond, **[25]**. El texto explica por medio de una metafora cómo se contruye software propietario y software libre, comparando el primero con la construcción de una catedral y el segundo con la colaboración cuasi-caotica de un bazar.

Las compañias que desarrollan programas de computo son las dueñas de los mismos, y venden a los usuarios, sólo la licencia de uso y esto bajo condiciones muy restrictivas, tales como la imposibilidad de copiarlo, o el impedimento de intentar modificarlo. A este tipo de programas se les denomina "software propietario"

Los programas desarrollados por la comunidad o por algunas empresas y que tiene el concentimiento expreso de los autores para modificarse o distribuirse se denomina

"software libre", aunque las condiciones de modificación y distribución varian de acuerdo a la licencia con la que el programa se libera (esto se explica en el capítulo 2).

Las ventajas del software propietario son:

- Existe el respaldo de una empresa detrás del producto
- Normalmente existe garantía
- Soporte técnico

Y sus desventajas son:

- Esta ligado a una empresa, si ésta desaparece el esfuerzo invertido en desarrollo se pierde.
- Los usuarios no pueden modificar (para mejorar, corregir o adecuar) los programas.
- Tienden a ser caros

Respecto al software libre sus ventajas son:

- No esta ligado a una empresa o persona, si el autor original abandona un desarrollo, siempre es posible que alguien más lo retome o utilice las piezas del programa que le sean útiles. De este modo el esfuerzo invertido no se desperdicia.
- El código fuente esta disponible, los usuarios pueden modificarlo de acuerdo a sus necesidades. También permite que los problemas sean corregidos muy rápidamente.
- Tiende a ser gratis. Puede conseguirse gratis y distribuir copias sin infringir ninguna ley. Sin embargo tampoco se impide su venta, estos programas se pueden vender a condición de que no se obstaculice su distribución.

Sus desventajas son:

- No cuenta con respaldo de grandes empresas. Hasta años recientes ha comenzado a recibir apoyo de empresas importante, la tendencia muestra claramente como grandes empresas como IBM, HP y otras apoyarán de manera importante este tipo de programas.

Programas libres para modelado tridimensional existen, pero hasta ahora han sido proyectos efímeros y de éxito limitado. Puede verse que proyectos de software libre de modelado 3D existen en la página freshmeat [1].

Como antecedente importante existe un proyecto de representación de superficies 3D, desarrollado por un alumno de la Facultad de Ciencias. Dicho proyecto se llama "superficie[2]" y sirvió para entender y salvar algunos problemas en el presente trabajo.

Otros proyectos importantes y más exitosos son "PovRay[3]" el cual es un poderoso motor de generación de imágenes sintéticas (render engine). Es uno de los mejores en su clase, pero no dispone de una interfaz para modelado, éste se realiza en archivos de texto de forma muy semejante a un programa de cómputo. Es gratis y el código fuente está disponible, pero los autores conservan todos los derechos, por lo cual no es libre.

Sart[4] es otro motor de generación de imágenes. Es libre, pero a pesar de ello aún no compite con PovRay.

Las motivaciones para desarrollar este proyecto son las siguientes:

1) Aplicar diseño centrado en el usuario. Los proyectos libres de modelado 3D son desarrollado típicamente por programadores que se enfocan más en las estructuras de datos y los algoritmos y descuidan los aspectos de la interfaz y los procesos realizados por el usuario. Es en gran medida por este descuido que los programas comerciales y libres difieren tanto en calidad.

2) La necesidad de disponer de un programa que sirva como base para desarrollos propios y evite la dependencia de productos comerciales. Sobre todo en ámbitos académicos donde en muchas ocasiones el presupuesto es muy limitado, es importante tener alternativas libres.

[1] http://freshmeat.net/browse/792/
[2] http://superficie.sourceforge.net/
[3] http://www.povray.org/
[4] http helga.zesoi.fer.hr/~silovic/sart/://

3) Desde el punto de vista académico. Se pensó en desarrollar el programa "Material 3D" para que los alumnos de Graficación por Computadora de la carrera de Ingeniería en Computación puedan desarrollar los contenidos del temario de una manera más activa. Esto en lugar de limitarnos a aspectos teóricos o a ejercicios muy simples.

4) Para contribuir con la Universidad, ya que en muchas dependencias carecen de los recursos para comprar programas comerciales. Para contribuir con la comunidad de software libre, de la cual he sido beneficiario al poder utilizar los programas desarrollados por ellos.

El objetivo general de esta tesis es la realización de un sistema de modelado y animación que permita la realización de modelos y animaciones tridimensionales con una amplia variedad de usos, desde los simples modelos para páginas web, hasta la visualización de datos científicos y de ingeniería pasando por modelos esquemáticos de utilidad académica.

El sistema propuesto deberá cumplir adicionalmente con los siguientes objetivos particulares:

- De uso sencillo, basado en la forma en que los usuarios realizan sus tareas.
- Interfaz gráfica de usuario sencilla y de manejo natural, producto de un análisis bajo condiciones controladas de la tarea del usuario.
- Ampliable, debe ser diseñado de forma que cualquier parte de la comunidad de usuarios pueda mejorar o agregar funcionalidades.
- Capacidad de manejar módulos adicionales programados por terceros. Dichos módulos deben poder ser cargables / descargables por el usuario y deben tener una interfaz consistente.
- Debe poder ejecutarse en Linux. Linux es una plataforma sólida y de amplio uso en la comunidad académica. En particular en esta plataforma no hay ofertas gratuitas de programas serios para el manejo de modelado y animación en 3D con

funcionalidades como las aquí planteadas. Sería deseable que pudiese correr en otras plataformas también.

- Diseño de fácil traducción, para que el sistema pueda tener una gran base de usuarios. Es necesario que pueda traducirse a varios idiomas de manera sencilla. Incluso seria recomendable que fuese posible el cambio de alfabetos.
- Que el proyecto garantice continuidad; esto, a través de dos aspectos fundamentales: además de usar una bien planteada licencia (por ejemplo GPL - Licencia Pública General) y un conjunto de bibliotecas de amplio uso y de fuentes abiertas.
- Debe tener capacidad de importar – exportar al menos un formato gráfico de vector, que sea estándar de facto. Este objetivo obedece a la necesidad de facilitar la migración a los usuarios.
- Usar un formato de documentación estándar y que tenga facilidades para ser impreso o publicado en la web.

Todos los objetivos anteriores hacen de éste un proyecto descomunal para poder realizarse por una sola persona, baste poner un ejemplo: el programa de modelado y animación 3D "blender" requirió 10 años hombre para su desarrollo[5]; la forma natural de realizar este tipo de proyectos es por un grupo de personas, donde el análisis y diseño se realizan por un líder de proyecto o un pequeño grupo y la implementación de los módulos es realizada por personas bajo su supervisión.

La forma de trabajo que se plantea en este proyecto es: la realización del análisis y diseño de la aplicación, así como una implementación mínima será el objeto de esta tesis y la implementación de los módulos se realizará como tesis de licenciatura de alumnos de la ENEP Aragón. Estamos conscientes del trabajo adicional de coordinación y de los riesgos de hacer tesis interdependientes, pero se cree que es una forma interesante en la que la Universidad puede generar soluciones de esta magnitud.

El programa se ha llamado "Material 3D", primero porque trabaja con objetos tridimensionales, pero además porque la palabra "material" se escribe de manera muy similar en varios idiomas.

[5] http://www.blender.org

Metodología

El sistema se desarrolló con base en tres aspectos principales:

a) Análisis de los usuarios potenciales;
b) Análisis de la tarea del usuario: Cómo modelan los usuarios habituados a los sistemas de graficación, y aquéllos que ignoran el uso de paquetes informáticos, pero que intentan modelar;
c) Análisis de las técnicas y paradigmas más poderosos y flexibles en la actualidad, y que además sean susceptibles de permanecer vigentes.

Estos tres aspectos llevan a desarrollar la siguiente metodología:

FASE	DESCRIPCIÓN	
	Análisis de requerimientos	**Análisis del usuario**
I	Definición del problema	Análisis de los usuarios tipo.
II	Análisis de las técnicas informáticas más adecuadas para el desarrollo del sistema	Análisis de la tarea del usuario.

	Especificación del sistema	**Validación con usuario**
III	Definición de los diferentes operadores y comandos que debe contener el sistema, sus prioridades, jerarquías, coordinación, así como de las informaciones que deben presentarse a los usuarios en cada estado de la aplicación.	Generación y evaluación de un prototipo semántico (informaciones e instrucciones a presentar en cada pantalla).
	Definición de estructuras	
	Definición de los formatos nativos	
	Definición de la estructura de los cartuchos (plugs-in)	

IV	**Desarrollo del sistema**	**Validación del sistema**
	Implementación de las primitivas básicas para insertar y manipular objetos en 3D.	Generación y evaluación con usuarios de un primer prototipo funcional.
	Implementación de las funciones básicas para la manipulación de texturas.	Pruebas con usuarios.
	Implementación de las funciones básicas para animar.	Pruebas con usuarios.

V	**Liberación del sistema a la comunidad Gnu para su ampliación**	

Tabla 0.1 Metodología y etapas del proyecto

Capítulo 1 Diseño centrado en el usuario

La importancia de los usuarios como elementos de los sistemas interactivos que involucran a maquinas y a personas, está siendo reconocida hasta muy recientemente. Muestra de ello se observa en los ciclos de diseño clásicos en donde el usuario no es tomado en cuenta, y la parte con la que interactúa (la interfaz) se diseña al final del desarrollo y refleja la arquitectura interna de la aplicación y no el proceso que realiza el usuario.

En este capítulo se mostrarán algunos de los principios del diseño centrado en el usuario, algunas consideraciones referentes a su importancia y algunos métodos utilizados en este trabajo.

Dado que el tema es muy amplio y rebasa los objetivos del presente proyecto, nos limitaremos sólo a explicar los conceptos y métodos de relevancia para el trabajo.

1.1 Principios del diseño basado en el usuario

Anteriormente el papel desempeñado por los usuarios no era tomado cuenta durante la etapa de diseño (en muchos lugares las cosas no han cambiado). Recientemente se ha revalorizado a los usuarios en el diseño de sistemas interactivos, hasta el punto de incluirlos de forma participativa en esta etapa del desarrollo.

El diseño basado en el usuario está muy relacionado con la ergonomía, la cual puede ser definida de la siguiente manera:

"Ergonomía es una disciplina nacida con el propósito de integrar, en la concepción de los sistemas de producción, los conocimientos existentes sobre el hombre en situación de trabajo[6]"

[6] http://www.ucm.es/info/csegae/exerg.html

La ACM[7] (Association for Computer Machinery) es una organización internacional de profesionales, estudiantes e investigadores interesados en los aspectos de la computación. Esta asociación cuenta con un grupo especial de trabajo en temas relativos a interfaz humano – maquina denominado SIGCHI[8] (Special Interest Group in Computer Human Interaction) el cual propuso la siguiente definición de Interacción Persona–Computadora:

"Es la disciplina relacionada con el diseño, evaluación e implementación de sistemas informáticos interactivos para el uso de seres humanos y con el estudio de los fenómenos más importantes con los que está relacionado".

La ergonomía es multidisciplinaria **[19]**, ya que aprovecha los conocimientos y métodos de diversas disciplinas, tales como: fisiología, psicología, sociología y biomecánica entre otras. Tiene varias ramas, cada una de las cuales se enfoca a ciertos aspectos del ser humano en su entorno de trabajo. La rama que más interesa para este proyecto es la ergonomía cognitiva.

La ergonomía se relaciona con otras disciplinas que buscan optimizar la relación humano – sistema, a continuación se mencionan dichas relaciones:

1) El estudio de tiempos y movimientos: Una tarea compleja se puede realizar de diferentes maneras, pero la forma en que se desarrolla, y el orden de las subtareas afecta el tiempo requerido para desarrollarla. El estudio de tiempos y movimientos busca encontrar la configuración óptima requerida para desarrollar un proceso en el menor tiempo posible, adicionalmente determina los tiempos requeridos para poder realizar planeación, fijar sueldos etc.

2) El estudio del trabajo: Se interesa por el proceso realizado por el usuario, en determinar sus acciones, condiciones y los problemas inherentes al trabajo realizado, sus objetivos son más limitados que los de la ergonomía, ya que no se preocupa demasiado por el usuario en sí.

[7] http://www.acm.org
[4] SIGCHI (Special Interest Group in Computer Human Interaction http://www.acm.org/sigchi/).

3) La investigación de operaciones: Intenta realizar predicciones confiables de necesidades futuras y planear las cargas de trabajo necesarias para cubrir dichas necesidades. El usuario es considerado como un recurso y se programa y reasigna de acuerdo a las necesidades. Es importante mencionar que en investigación de operaciones no se considera importante la opinión de los usuarios dentro de los modelos.

1.1.1 Breve historia de la ergonomía

El término "ergonomics" se creó en Oxford el 12 de julio de 1949, durante una reunión de un grupo relacionado con varias disciplinas con interés en los problemas de ámbito laboral. El interés por la ergonomía se había iniciado desde la primera guerra mundial, pero fue durante la segunda guerra mundial cuando se evidenció la importancia de entender y mejorar la interacción entre humanos y sistemas, debido a la creciente complejidad de la maquinaria bélica **[27]**.

Existen diferentes enfoques para la ergonomía dependiendo del país que la desarrolle. Mientras que en Estados Unidos está orientada a aspectos fisiológicos, en Europa se refiere a la adaptación de los sistemas a las necesidades de los usuarios, esto en todos los sentidos, desde fisiológicos, hasta los cognitivos.

En Europa se considera a Inglaterra como el padre de la ergonomía Europea, sin embargo Francia es el país con más agrupaciones.

1.1.2 Ergonomía cognitiva

La ergonomía cognitiva **[28]** se interesa principalmente por los siguientes temas:

- Proceso de recepción de señales e información.
- Cómo la procesan los seres humanos
- Cómo actúan con base a: la información, los conocimientos y la experiencia.

En la interacción entre una persona y un sistema está en juego la recepción de señales: Las que el operador humano envía al sistema y las que el sistema regresa al operador. El estudio de este proceso de envío y recepción de señales es de gran importancia para comprender que es lo qué permite una comunicación más clara y dónde se presentan las fuentes de error.

Las señales provenientes del sistema son interpretadas por el operador, para lo cual este último hace uso de sus conocimientos y experiencia. Estos dos aspectos deben ser tomados muy en cuenta, ya que por ejemplo, en algunas culturas ciertos símbolos pueden ser interpretados de forma diferente a como lo hacemos nosotros.

1.2 MAD*

MAD (Méthode Analytique de Description des tâches – Método analítico de descripción de tareas) **[13]**. Es un formalismo basado en la tarea del usuario. Utiliza una estructura gráfica jerárquica para representar las tareas, y sus subtareas como ramas (Fig. 1.1)

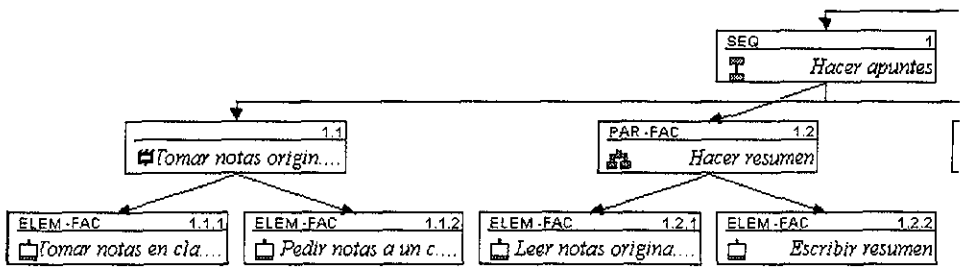


Figura 1.1Una tarea y sus subtareas

MAD* o MAD-STAR. Es el producto de una revisión de MAD: incorpora mejoras para evitar algunas confusiones que se presentaban utilizando MAD en el análisis de tareas muy complejas. IMAD* es una implementación de MAD* para ser utilizada en una PC.

1.2.1 Cuerpo de la tarea

MAD* se basa en tareas y toma estas como unidades. De esta forma una tarea es una unidad que tiene los componentes mostrados en la figura 1.2.

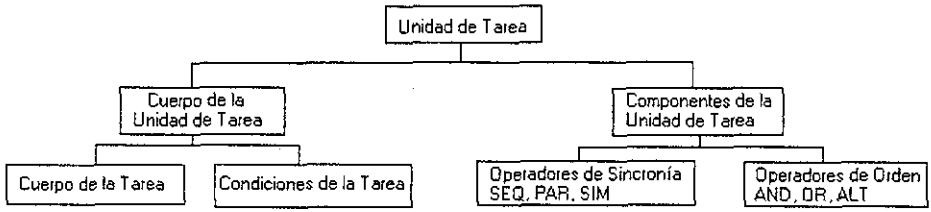


Figura 1.2 Componentes de una tarea

El cuerpo de la tarea contiene la información propia de la tarea en sí. Cada atributo del cuerpo de la tarea, así como el tipo de valor se muestran en la tabla 1.1

Atributo	Tipo de Valor o Valores Posibles
Número de identificación	Alfanumérico
Nombre	Alfanumérico
Objetivo	Alfanumérico
Comentarios	Alfanumérico
Grado de libertad	[Opcional, Obligatorio]
Prioridad	Entero
Interrumpible	[Verdadero, Falso]
Tarea superior	Apuntador a la tarea superior (en la implementación de MAD*)
Modalidad	[Manual, Automática, Interactiva]
Tipo	[Senso-motora, Cognitiva]
Centralidad	[Importante, No importante]
Frecuencia	[Alta, Media Baja]
Entidades importantes	Cantidad de tareas
Experiencia	Usuario = [Novato, Ocasional, Experto]
Estado	[Esperando para ejecución, Inaccesible, En Ejecución, Interrumpida, Terminada, Ignorada]
Subtareas obligatoriamente terminadas	Entero

Tabla 1.1 Atributos de la tarea y sus posibles valores

Condiciones de la tarea

Las condiciones de IMAD* están basadas en:

1) El estado del mundo, el cual incluye los objetos utilizados por las tareas.
2) La "estructura condicional" la cual expresa la gramática formal de las reglas. Estas reglas expresan limitantes sobre los valores de atributos en los objetos.

Existen tres tipos de precondiciones:

1) **Precondición de arranque:** Establece las condiciones para arrancar la ejecución de la tarea. Es útil cuando se administran eventos del sistema o para ejecutar en un orden diferente al definido por los operadores de sincronía.

   Aún cuando esta condición sea valida, el que la tarea se ejecute depende de la precondición de ejecución.

2) **Precondición de ejecución:** Verifica el valor correcto de los objetos en la tarea. Debe ser evaluada para poder ejecutar la tarea.

3) **Precondición de paro:** Indica bajo qué circunstancias termina la ejecución de la tarea.

Sólo hay una Postcondición: **Postcondición final**, la cual especifica los valores de los objetos involucrados con el fin de la ejecución.

1.2.2 Componentes de la tarea

Los componentes de IMAD* son los operadores de sincronía, los cuales se refieren al orden en que se ejecutan las tareas y la dependencia que guardan. Los operadores de orden, se refieren a las alternativas de ejecución para diferentes tareas.

A continuación se explica cada uno de los diferentes operadores:

Operadores de sincronía:

- **Secuencial** : Si una tarea tiene este operador, indica que debe ejecutar cada una de sus subtareas en orden para poder terminar.

- **Paralela**: Las subtareas se pueden ejecutar en cualquier orden.

- **Simultanea**: N subtareas se pueden ejecutar al mismo tiempo, por n operadores.

Operadores de orden:

- **Alternativa:** Sólo una de las subtareas puede ser ejecutada.

- **OR:** Una o más subtareas pueden ser ejecutadas.

- **AND:** Todas las tareas deben ser ejecutadas.

Por último, la tarea "**elemental**" no tiene subtareas y por lo tanto no requiere indicar cómo se ejecutarían éstas.

1.3 Pruebas con usuarios

Las pruebas con los usuarios se realizan durante todo el desarrollo, empezando por las pruebas iniciales para determinar el proceso, hasta las pruebas del desempeño con el sistema final, pasando por todos los prototipos intermedios que permiten afinar y mejorar el diseño.

Existen varios tipos de prototipos, por mencionar algunos:

a) **Prototipos en papel [20]**: Del bosquejo que se ha desarrollado se determinan las pantallas y su contenido. Cada pantalla se "dibuja" en una hoja y se le pide a un usuario que realice una tarea por medio de las "pantallas de papel". Al final se tendrá una pila de hojas que son la simulación de una corrida.

   Este tipo de prototipos tiene como ventaja principal el hecho de que se pueden preparar en poco tiempo y el costo del material es mínimo.

b) **Prototipos en software**: Se "dibujan" las pantallas con ayuda de un programa RAD (Desarrollo Acelerado de Aplicaciones por sus siglas en ingles), tal como Visual Basic o Delphi. Dichas pantallas no contienen código relacionado y en realidad se comportan de forma semejante a los prototipos en papel, pero con la ventaja de hacer sentir a los usuarios un sistema "real" (considere que para la mayoría de los usuarios la interfaz es la aplicación).

   Existen tres tipos de prototipos de software:

   - **El desechable**: El prototipo se desecha después de las pruebas, se utiliza sólo para determinar ciertos aspectos de la tarea o estudiar ciertos efectos sobre el usuario.
   - **El incremental**: Se desarrolla la aplicación sobre el prototipo agregando módulos, los cuales son probados primero.
   - **Evolutivo**: El prototipo se modifica cada prueba para eliminar errores y para agregar las funcionalidades que se determinen con cada prueba.

En las primeras pruebas, cuando el objetivo es determinar la estructura del proceso se pueden aplicar varios métodos. Para este trabajo se utilizó una variante del Mago de Oz, la cual a continuación se describe:

1.3.1 Mago de Oz

El "Mago de Oz" es un procedimiento que consiste en:

Una persona actuará como el usuario del hipotético sistema, otra persona (se le llamará operador) actuará como el sistema en sí. De este modo el usuario indica al operador las acciones y éste las realiza.

Este procedimiento es útil para determinar qué tareas realiza el usuario de manera natural cuando no está bajo las restricciones que impondría un prototipo. Existen variantes del mago de Oz, donde el operador puede trabajar con un sistema real o bien con un sistema que simula la presentación visual del sistema.

1.4 Criterios ergonómicos [15]

Para poder evaluar un sistema se necesitan lineamientos que ayuden a determinar de manera objetiva qué aspectos se han realizado satisfactoriamente y con ello tener elementos de juicio para comparar aplicaciones.

Los criterios ergonómicos evalúan ocho aspectos. A continuación se menciona en qué consisten:

1) **Guía**: se refiere a los medios disponibles para orientar, informar y guiar al usuario a través de su interacción con una computadora. Incluye los siguientes aspectos:

   a) Indicadores
   b) Agrupamiento y distinción de elementos.
   c) Retroalimentación inmediata.
   d) Legibilidad

2) **Carga de trabajo**: Se refiere a la cantidad de información que recibe un usuario, es importante que sea la mínima necesaria. Incluye:

   a) Brevedad
   b) Densidad de la información

3) **Control explícito**: El usuario debe poder ver reflejadas sus acciones de manera inmediata y debe poder cambiar de opinión durante la ejecución de una tarea:

   a) Acción explícita del usuario
   b) Control por parte del usuario

4) **Adaptabilidad**: Los sistemas deben poder adaptarse al tipo de usuario:

   a) Flexibilidad
   b) Experiencia del usuario

5) **Manejo de errores**: ¿Los errores se evitan?, ¿Entiende el usuario el mensaje de error?:

   a) Protección contra errores
   b) Calidad de los mensajes de error
   c) Corrección de errores

6) **Consistencia**: Es el aspecto y lógica de trabajo uniforme en toda la aplicación

7) **Códigos significativos**: ¿Los letreros de los menús y los iconos son fácilmente asociados a sus acciones?

8) **Compatibilidad**: ¿Se comporta la aplicación de manera semejante al resto de las aplicaciones en su entorno?

Aquí sólo se presentó la lista de los criterios ergonómicos y a que aspectos se refieren. Una lectura mucho más detallada puede ser encontrada en el anexo B de este trabajo.

Una gran cantidad de sistemas nunca llega a utilizarse a pesar de funcionar como se planeó, esto se debe en gran medida a no haber considerado la tarea del usuario. Los seres humanos tenemos un comportamiento complejo y aunado a esto se presenta en mayor o menor medida una resistencia a los cambios. El no tomar en cuenta a los usuarios complica el problema. Para la mayoría de las personas es más fácil aceptar una solución en la que participaron.

Dado que uno de los objetivos principales de este trabajo de tesis es "que el sistema sea fácil de usar", se ha invertido gran cantidad de tiempo en realizar pruebas, mismas que son descritas en el capítulo tres.

Capítulo 2 Análisis de las arquitecturas y herramientas

Dado que uno de los objetivos de este trabajo fue que el sistema final sea útil a la comunidad de usuarios de software libre, es importante la selección de la arquitectura, herramientas y bibliotecas que se utilizarán. Se seleccionaron tecnologías con base en un compromiso de portabilidad, difusión y facilidad de acceso y de uso, que permitan que la comunidad de usuarios pueda obtener un sistema flexible, adecuado a sus necesidades, y con garantía de continuidad.

En este capítulo se revisarán los diferentes tipos de licencias de software, entre las que destacan la GPL (Licencia Pública General). También se revisarán las bibliotecas para la creación de interfaces gráficas de usuario, destacando GTK+, sobre la que esta construido todo el escritorio GNOME; las bibliotecas Qt, sobre las que se construye el escritorio KDE y algunas otras bibliotecas menos conocidas como FLTK.

Para la creación y manipulación de los modelos en dos y tres dimensiones se revisarán las bibliotecas OpenGL y Mesa3D, siendo ambas compatibles y con características casi idénticas.

Para el manejo de archivos se aborda la biblioteca LibXML, la cual implementa funciones de alto nivel para el manejo de archivos en formato XML. Una de las tendencias actuales para el almacenamiento de información y el intercambio de datos entre aplicaciones, es el manejo de formatos legibles para las personas y manejo de estructuras de información, requisitos cumplidos por XML.

Por último se revisa brevemente la herramienta de desarrollo de aplicaciones con interfaz gráfica de usuario, Glade. Glade genera código sobre GTK+ y puede ser ligado a muchos de los principales lenguajes de programación, tales como:

- Lenguaje "C"
- C++
- Perl

2.1 Revisión de los tipos de licencias de software

Con el marco legal actual, la licencia bajo la que se distribuye un programa delimita exactamente los derechos que tienen sobre él sus usuarios. Por ejemplo, en la mayoría de los programas propietarios la licencia priva al usuario de los derechos de copia, modificación, préstamo, alquiler, uso en varias máquinas, etc. De hecho, las licencias suelen especificar que el propietario del programa es la empresa editora del mismo, que simplemente vende derechos restringidos de uso del mismo.

En el mundo del software libre, la licencia bajo la que se distribuye un programa también va a ser de gran importancia. Normalmente, las condiciones de las licencias de software libre son el resultado de un compromiso entre varios objetivos hasta cierto punto contrapuestos. Entre ellos pueden citarse los siguientes:

- Garantizar a los usuarios algunas libertades básicas (de redistribución, de modificación, de uso).
- Asegurar algunas condiciones impuestas por los autores (cita del autor en trabajos derivados, por ejemplo).
- Procurar que los trabajos derivados sean también software libre.

Los autores pueden elegir proteger su software con distintas licencias según el grado con que quieran cumplir cada uno de estos objetivos, y los detalles que quieran asegurar. De hecho, el autor puede distribuir su software con licencias diferentes por medios diferentes y con tarifas diferentes.

Aunque cada autor puede utilizar una licencia diferente para sus programas, la gran mayoría del software libre usa una de las siguientes licencias:

- ``estilo" BSD
- GPL
- LGPL
- Artistic
- ``estilo" Netscape

2.1.1 BSD (Berkeley Software Distribution)

La "licencia BSD"[9] cubre, entre otros, las entregas de BSD (Berkeley Software Distribution). Estas entregas fueron hechas por el CSRG (Computer Science Research Group) de la Universidad de California en Berkeley. Las entregas de BSD fueron la forma en que el CSRG distribuía su trabajo alrededor del sistema operativo Unix. De hecho, Unix era un sistema operativo propietario, y por ello durante mucho tiempo los usuarios de las entregas de BSD necesitaban también una licencia Unix. Las entregas de BSD se usaron como la base de muchos sistemas operativos propietarios, como SunOs de Sun Microsystems o Ultrix de DEC. Todos los sistemas derivados de las entregas de BSD componen la "rama BSD" del árbol Unix.

A principios de los años noventa el CSRG liberó versiones que no incluían código propietario, por lo tanto los usuarios ya no requerían una licencia de Unix. Después de algunos litigios con los dueños de la licencia Unix, se llegó a un acuerdo con la entrega de BSD-Lite, que fue reconocida como completamente libre de código propietario. Esa entrega fue el origen de NetBSD, FreeBSD y OpenBSD. La entrega de BSD-Lite fue el último trabajo hecho por el CSRG antes de desaparecer.

La licencia BSD es un buen ejemplo de una licencia permisiva, que casi no impone condiciones sobre lo que un usuario puede hacer con el software, incluyendo cobrar a los clientes por distribuciones binarias, sin la obligación de incluir el código fuente. Probablemente esta licencia (además de la excelencia técnica del software), fue una de las razones principales para su uso en tantos sistemas propietarios derivados de Unix durante los años ochenta. Los principales puntos que establece la licencia son:

- Se permite la redistribución, uso y modificación del software.
- Las distribuciones deben incluir copias literales de la licencia, anuncio de copyright y una "negación de responsabilidad" (disclaimer).
- Debe incluirse reconocimiento del origen del software (la Universidad de California) en cualquier anuncio.

[9] http://www.opensource.org/licenses/bsd-license.html

Resumiendo, los redistribuidores pueden hacer casi cualquier cosa con el software, incluyendo usarlo para productos propietarios. Los autores sólo quieren que su trabajo sea reconocido. En cierto sentido, esta restricción asegura un cierto grado de "mercadeo (marketing) gratis". Es importante darse cuenta que este tipo de licencia no incluye ninguna restricción orientada a garantizar que los trabajos derivados sigan siendo libres. De hecho, ya se ha mencionado anteriormente cómo muchos sistemas operativos derivados de versiones de BSD han sido distribuidos como software no libre. Puede argumentarse que esta licencia asegura "verdadero software libre", en el sentido que el usuario tiene libertad ilimitada con respecto al software, y puede decidir incluso redistribuirlo como no libre. Otras opiniones están más orientadas a destacar que este tipo de licencia no contribuye al desarrollo de más software libre, incluso cuando es básicamente la versión libre original redistribuida como un producto propietario. Otras licencias similares son las que cubren el Sistema X Window X11R6, XFree86 y Tcl/Tk

2.1.2 GPL (Licencia Pública de GNU)

La Licencia Pública de GNU (GPL[10], de sus iniciales en inglés, General Public License) es la licencia bajo la cual se distribuye el software del proyecto GNU. Sin embargo, hoy día pueden encontrarse toneladas de software no relacionado con el proyecto GNU distribuido bajo la GPL (un ejemplo notable es el kernel Linux). La GPL se diseñó cuidadosamente para promover la producción de más software libre, y por ello prohíbe explícitamente algunas acciones sobre el software que podrían llevar a la integración de software protegido por la GPL en programas propietarios. La GPL usa como base legal la legislación sobre copyright, haciendo de esa forma un uso muy interesante de ella, ya que se usa el copyright para promover la distribución de software que garantiza mucha más libertad a los usuarios que los trabajos habitualmente protegidos por copyright. Por lo tanto, algunas se veces dice que el software cubierto por la GPL está protegido no por copyright sino por "copyleft". Esto como un juego de palabras en inglés que indica contraposición.

[10] http://www.fsf.org/licenses/licenses.html

Las principales características de la GPL son las siguientes:

- Permite la redistribución binaria, pero sólo si se garantiza también la disponibilidad del código fuente. Esta disponibilidad puede garantizarse bien mediante su distribución simultánea, o mediante el compromiso de distribución a solicitud de quien recibe la versión binaria.
- Permite la redistribución fuente (y obliga a ella en caso de distribución binaria).
- Permite las modificaciones sin restricciones siempre que el trabajo derivado quede cubierto también por la GPL.
- La integración completa sólo es posible con software cubierto por la GPL.

2.1.3 LGPL (GPL para bibliotecas)

La Licencia Pública de GNU para Bibliotecas (LGPL[11], según las iniciales de su nombre en inglés, Library General Public License) fue diseñada por la Free Software Foundation para proteger las bibliotecas que se estaban desarrollando en el proyecto GNU, pero que permite que puedan ser enlazadas con programas propietarios.

- Usada en bibliotecas del proyecto GNU y muchos proyectos más.
- Diseñada para permitir el uso de bibliotecas libres con software propietario (por ejemplo, en el caso de un compilador libre).
- Como la GPL cuando se redistribuye la biblioteca como tal.
- Permite la integración con cualquier otro software. En este caso, no hay prácticamente limitaciones.

2.1.4 Otras licencias de software libre

- Artistic[12] (Perl, similar a la de BSD).
- Aladdin (Aladdin Ghostscript, en caso de redistribución con ánimo de lucro, las condiciones son similares a las del software propietario).
- NPL (Licencia Pública de Netscape, incluye ciertos privilegios para el ``primer autor'')

[11] http://www.fsf.org/licenses/licenses.html
[12] http://www.opensource.org/licenses/artistic-license.php

2.2 Bibliotecas para el diseño de interfaces

En este proyecto se enfatizó el análisis y diseño de una interfaz bien diseñada, por lo que es fundamental la correcta elección de las bibliotecas para el desarrollo de ésta. De esta elección dependen el estilo y el aspecto de la aplicación, los limites de la interacción y la portabilidad **[11]** de la aplicación. Existen varias bibliotecas para crear interfaces, pero cada una presenta sus ventajas y desventajas.

A continuación se mencionan algunas características de las principales bibliotecas para el desarrollo de interfaces gráficas de usuario.

2.2.1 Qt

Qt[13] fue la primera biblioteca en popularizarse dentro del mundo de Linux, gracias a que ofrecía funciones de alto nivel, fáciles de manipular y escritas en "C++". El principal impulso para Qt se debió a un ambicioso proyecto que busca crear un atractivo "escritorio" basado en el CDE (Common Desktop Enviroment), llamado KDE[14] (Kool Desktop Enviroment). Adicionalmente al escritorio se crearían varios programas de uso común en la computadoras personales.

Como ventaja adicional existen versiones para Windows, con lo cual aligeraban el trabajo de portar programas de un sistema a otro. Cabe mencionar que a pesar de que este objetivo se logró relativamente bien (el 90% del código se conserva al pasar de una plataforma a otra) no llega al nivel de GTK+, que permite migrar aplicaciones prácticamente sin cambiar nada.

Lo que impidió que Qt se convirtiera en el estándar fue su licencia original, que básicamente decía: si la aplicación que se va a escribir con Qt es libre, entonces la biblioteca Qt es gratis, pero si el programa es comercial, entonces la biblioteca también es comercial y se requiere efectuar un pago. Dicha licencia causó la incomodidad de varios sectores de la comunidad del "Open Source", entre ellos Miguel de Icaza, el cual junto con Federico Mena iniciaron el proyecto GNOME, el cual consiste en crear un escritorio con aplicaciones (lo mismo que KDE), pero que estuviese basado en una biblioteca sin esas restricciones.

[13] http://www.trolltech.com/

Ahora, debido a la competencia entre GNOME y KDE y sus bibliotecas GTK+ y Qt, esta última ha cambiado su licencia a GPL, con lo cual terminan con los problemas potenciales que acarreaba la licencia original. A pesar de ya no presentar problemas de licencia, esta carrera por el "escritorio de Linux" ha dejado dos corrientes que cuentan con sus propios seguidores, herramientas y aplicaciones, afortunadamente ya se realizan acciones para permitir la integración de ambos mundos y así poder evitar duplicidad de esfuerzos.

2.2.2 GTK+

GTK+ es básicamente una biblioteca de funciones para la realización de interfaces gráficas de usuario.

GTK+ **[2]** significa *GIMP Toolkit*, es decir, conjunto de herramientas para GIMP (http://www.gimp.org), en tanto que GIMP significa *GNU Image Manipulation Program* o Programa de Manipulación de Imágenes GNU. Cuando la comunidad GNU estaba trabajando en él, decidieron escribir una biblioteca para interfaces que fuera también de licencia GPL, dado que estaban trabajando originalmente con las bibliotecas Motif, las cuales no son libres.

Así fue como nació el proyecto GTK+ en el cual se escogió el lenguaje C como lenguaje de desarrollo, debido a que en esos momentos era mucho más portable que C++, además de la gran base de programadores de C que hay en el mundo. Sin embargo, no se ignoraron las ventajas de la programación orientada a objetos, se implementaron las funciones de tal forma que quién utiliza la biblioteca GTK+ puede trabajar con objetos.
GTK+ es una biblioteca que consta de tres partes:

- GLib : Biblioteca de funciones que comprenden la definición de tipos y manejo de estructuras de datos.
- GDK : Biblioteca de funciones gráficas.
- GTK : Es la parte más alta en cuanto nivel de abstracción, contiene las funciones para crear los botones, ventanas, y en general todos los elementos de interfaz gráfica.

[14] http://www.kde.org/

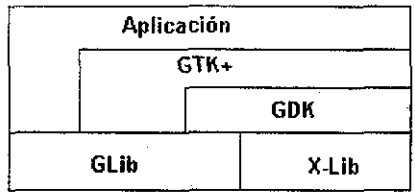


Figura 2.1 Estructura de capas de GTK+

Glib

Como se aprecia en la figura 2.1. Glib es una biblioteca sobre la cual se construye el resto de GTK+. Básicamente está pensada en proporcionar portabilidad y funciones para el manejo de estructuras de datos.

El lenguaje "C" **[1]** es bastante portable, es decir, puede pasarse un programa de una plataforma a otra con cambios mínimos o incluso sin cambio. Uno de los aspectos que dificultan la portabilidad es que "C" define sus tipos básicos en función del tamaño de la palabra del procesador donde se ejecuta, así pues un int no es del mismo tamaño en una PC y en una estación de trabajo.

Glib redefine los tipos básicos para que puedan ser utilizados independientemente de la plataforma, algunos de estos tipos son:

gboolean;	Booleano (puede ser TRUE o FALSE)
gchar;	Equivalente a char
guchar;	Equivalente unsigned char
gint;	Equivalente a int
guint;	Equivalente a unsigned int
gint16;	Equivalente a int de 16 bits
gint32;	Equivalente a int de 32 bits

Como se puede apreciar, existe más de una definición del tipo int. De hecho hay declaraciones de int de 64 bits y cualquiera de estos tipos puede ser utilizado en cualquier plataforma. También hay tipos que no existen en "C" pero que han

demostrado mucha utilidad como gboolean, el cual es el equivalente al boolean de Pascal.

Adicionalmente Glib incluye una gran variedad de funciones que facilitan la vida del programador. Por citar algunos:

- Manejo de Cadenas
- Manejo de Hora y Fecha
- Gestión de memoria
- Listas simplemente ligadas
- Listas doblemente ligadas
- Arboles
- Tablas Hash
- Autocompletado de cadenas
- Analizador léxico
- Cargador dinámico de módulos

GDK

GDK (GIMP Drawing kit o equipo de dibujo de GIMP) contiene funciones para crear áreas de dibujo, seleccionar propiedades de los pinceles y para crear primitivas gráficas tales como arcos, puntos, mapas de bits, segmentos de recta, texto, etc.

En GDK están declaradas las señales para el manejo de eventos típicos en dibujo, como por ejemplo, redimensionamiento de las ventanas que contienen elementos gráficos o el redibujo de las mismas.

GDK sirve de base para crear los elementos de interfaz, los cuales se encuentran en GTK. En la figura 2.1 se observa que las aplicaciones pueden directamente utilizar funciones de GTK. Eso no impide que pueda también utilizar funciones de GDK y de Glib.

GTK

GTK contiene las declaraciones de constantes y las funciones que manejan los "widgets" (botones, etiquetas, ventanas y en general cualquier elemento de una interfaz gráfica de usuario).

GTK+ está diseñado para trabajar de forma orientada a eventos, es decir, las aplicaciones se dibujan en pantalla y esperan a que ocurran eventos para ejecutar una acción. Un evento es cualquier acción del usuario sobre la interfaz, por ejemplo hacer clic sobre un botón o maximizar una ventana.

Todas las aplicaciones que se desarrollan con esta biblioteca deben inicializar un ciclo por medio de una llamada a la función gtk_init(), ésta debe ser llamada antes que cualquier otra función GTK, debido a que es necesario que el programa esté "a la escucha" de cualquier cambio que ocurra sobre la interfaz y pueda responder en consecuencia. El ciclo termina hasta que se ejecuta la función gtk_main_quit():

2.2.3 FLTK

FLTK[15] (Fast Light Tool Kit) es una biblioteca de funciones para crear interfaces gráficas de usuario, escrito por Bill Spitzak. La biblioteca tiene una licencia LGPL (GNU Library General Public License) con una restricción: permite que las aplicaciones ligadas estáticamente sean distribuidas sin proveer el código fuente de la aplicación ni de la biblioteca.

FLTK es multiplataforma y trabaja con X (UNIX), OpenGL, y Microsoft Windows NT 4.0, 95, o 98, está escrito en C++ y está diseñado para ligarse de forma estática, con lo cual las aplicaciones resultantes son fáciles de instalar.

[15] http://www.fltk.org/

Características más importantes de FLTK:

- Ejecutables compactos: el programa "Hola Mundo" compilado y ligado con la biblioteca estática FLTK y el compilador gcc ocupa sólo 82K.

- Escrito directamente sobre Xlib (Versión Unix) o sobre WIN32 (Versión Microsoft Windows), lo cual logra mayor eficiencia y desempeño.

- Compatibilidad buena (sin llegar a ser completa); cerca del 90% del código se conserva sin cambios.

- Posee un programa interactivo para construir las interfaces gráficas, la salida es legible y está diseñada para ser comprensible por las personas.

- Posee un elemento gráfico denominado "drawing area" para visualizar ventanas de OpenGL/Mesa.

Programar con FLTK es sencillo y de escritura breve; a continuación se muestra un programa que crea una ventana que dice "Hola Mundo":

```
#include <fltk/Fl.h>
#include <fltk/Fl_Window.h>
#include <fltk/Fl_Box.h>
int main(int argc, char **argv) {
Fl_Window *window = new Fl_Window(300,180);
Fl_Box *box = new Fl_Box(20,40,260,100,"Hola Mundo!");
box->box(FL_UP_BOX);
box->label_font(FL_HELVETICA_BOLD_ITALIC);
box->label_type(FL_SHADOW_LABEL);
window->end();
window->show(argc, argv);
return Fl::run();
}
```

La forma de programar, el crear ejemplares de los elementos gráficos y el manejo de retrollamadas es muy semejante a como lo manejan otras bibliotecas, en particular GTK+.

2.3 Bibliotecas gráficas

Desde los inicios de la Graficación por Computadora se han creado conjuntos de funciones gráficas (núcleos o kernel's). Sin embargo a pesar de los esfuerzos, no se ha podido obtener un estándar.

Lo más parecido a un estándar es la biblioteca OpenGL, que es una marca registrada y con derechos de copia de Silicon Graphics, Inc. Existe una biblioteca libre, la cual es muy semejante y posee una API sumamente parecida, se llama Mesa3D.

Existen otras bibliotecas gráficas, pero ninguna tiene la difusión, ni la flexibilidad como las anteriormente mencionadas.

2.3.1 OpenGL

OpenGL (Open Graphics Library) es una biblioteca gráfica 2D y 3D, desarrollada por la empresa Silicon Graphics, Inc, e introducida en 1992 a la industria **[6][7]**. Desde entonces esta biblioteca se ha convertido en un estándar.

Algunas de las principales funciones gráficas de OpenGL son:

Buffer de Acumulación: Es un buffer en el que múltiples cuadros pueden ser compuestos para producir una imagen promediada, útil para representar efectos de movimiento y eliminar los artefactos de discretización.

Mezcla Alpha: Permite grados de transparencia en los objetos. El canal Alpha se puede entender como una capa que se coloca sobre la imagen y puede cambiarla de desde completamente transparente hasta completamente opaca, pasando por una variedad de niveles de transparencia.

Anti-aliasing: El aliasing o artefacto de discretización es un defecto debido a los algoritmos de generación de imágenes sintéticas. Se expresa como fronteras muy marcadas y con un efecto de "escalones" en los píxeles. El antialiasing es un mecanismo que elimina dicho efecto, logrando imágenes de bordes suavizados.

Modo de índice de color: Los buffers (almacén temporal) pueden almacenar un índice de la paleta de colores en lugar de los canales R, G, B y Alpha.

Listas de despliegue: Las listas de despliegue contienen comandos OpenGL y pueden ser ejecutados mientras se despliega otra imagen. El resultado de la ejecución de la lista de despliegue puede ser mostrado en cuanto sea requerido, evitando parpadeos.

Almacén temporal doble (double buffer): Permite tener en memoria una imagen antes de ser requerida y cargar o generar la siguiente al intercambiar la pantalla activa con el almacén.

Sombreado Gouraud: Calcula la iluminación en cada vértice e interpola el color para cada píxel del polígono, permitiendo una variación suave de los colores y evitando los cambios bruscos entre cada cara.

Operaciones sobre píxel: Almacenamiento, transformación, mapeado y zoom.

Evaluador Polinomial: Soporta B-splines racionales no uniformes (NURBS).

Primitivas: Punto, línea, bitmap, o imagen. **Primitivas raster**: Bitmaps y pixel rectangulares.

Modo RGBA: Los almacenamientos de color pueden almacenar los componentes rojo, verde, azul y alpha o el índice de la paleta que indica su color.

Mapeado de textura: Es el proceso de aplicar una imagen a una primitiva gráfica. Esta técnica es utilizada para lograr realismo en las imágenes.

Transformación: Habilidad para cambiar el tamaño, la posición, la orientación y la perspectiva de un objeto en un espacio cartesiano.

Almacén Z (Z-buffering): Este almacén contiene la profundidad de los punto para determinar cuáles están más cerca que otros con lo cual es posible eliminar superficies ocultas.

La principal ventaja de OpenGL es la portabilidad. Actualmente es soportado por las siguientes plataformas:

Todas las versiones de UNIX incluido Linus y FreeBSD, MacOS, OS/2, Windows 95/98, Windows 2000, Windows NT, OPENStep, y BeOS. Puede ligarse con diferentes lenguajes de programación tales como Ada, C, C++, Fortran, Python, Perl, Java y ofrece completa independencia de protocolos de red y topologías.

Arquitectura del OpenGL

OpenGL está desarrollado como una estructura de capas (Figura 2.2), donde se facilita la independencia de la plataforma gracias a la reducción de la cantidad de código que requiere cambiarse, ya que sólo requieren cambios las capas que están en contacto con los niveles más bajos.

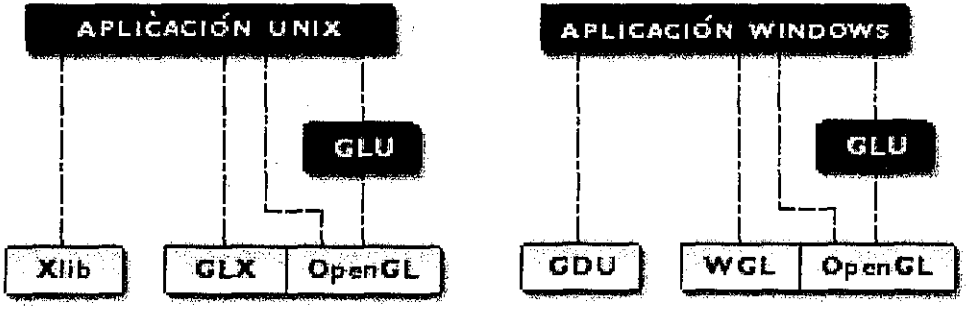


Figura 2.2 Arquitectura de capas para Unix y Windows

2.3.2 Mesa3D

Mesa[16] es una biblioteca gráfica 3D con una API[17] muy semejante a OpenGL. Por cuestiones legales no se puede decir que Mesa 3D es un reemplazo para OpenGL, pero lo cierto es que muchos de los programas que utilizan OpenGL puede ser compilados utilizando Mesa con cambios mínimos.

La parte principal de Mesa tiene una licencia que permite la copia, modificación, redistribución e incluso la venta de las aplicaciones construidas con ella sin cargo. Para los fines de este trabajo ésta es la diferencia principal con OpenGL.

Mesa 3D Graphics Library puede ser descargada libremente de http://www.mesa3d.org, y se encuentra disponible en prácticamente todas las distribuciones de Linux.

2.4 Bibliotecas para el manejo del contenido de los archivos

El desarrollar un formato propio siempre implica tomar decisiones en el diseño, refleja las estructuras de datos del sistema (y en ocasiones las limita), determina el espacio que los archivos de trabajo ocuparán y la complejidad para que terceros escriban o lean nuestro formato.

Hasta hace muy poco tiempo casi todo los programas tenían sus formatos propietarios y sólo los creadores sabían qué significaba cada parte del archivo. Leer y escribir en éstos era una labor desde sencilla hasta prácticamente imposible. En la actualidad, el manejo de estándares y la posibilidad de compartir información entre aplicaciones nos está llevando a formatos más comprensibles, tales como HTML, RTF y más recientemente a XML.

2.4.1 XML

El XML[18] (eXtensibe Markup Languaje - Lenguaje de Marcas Extensible) **[8][10]** es un metalenguaje que permite definir nuevos lenguajes de uso específico, en los que la información adicional es agregada por medio de etiquetas encerradas entre picoparéntesis.

[16] http://www.mesa3d.org
[17] Interfaz de Programación de Aplicaciones por sus siglas en Inglés
[18] http://www.xmlsoft.org/

Ejemplo de un archivo XML que contiene una caja tridimensional:

```
<?xml version="1.0"?>
<m3d:Helping xmlns:m3d="http://tigre.aragon.unam.mx/m3d/">
  <m3d:Objects>
   <m3d:Object>
   <m3d:M3Dobject Name="Box01" NumVertex="18" NumTriangles="32"/>
      <m3d:Vertex>0, 0.5, 0</m3d:Vertex>
      <m3d:Vertex>-0.5, 0.5, -0.5</m3d:Vertex>
      <m3d:Vertex>0, 0.5, -0.5</m3d:Vertex>
       ...
      <m3d:Vertex>0, -0.5, 0</m3d:Vertex>
      <m3d:Triangle>2, 0, 4</m3d:Triangle>
      <m3d:Triangle>4, 3, 2</m3d:Triangle>
       ...
      <m3d:Triangle>10, 17, 16</m3d:Triangle>
    </m3d:Object>
  </m3d:Objects>
</m3d:Helping>
```

A diferencia de HTML, XML almacena no sólo el contenido sino también la estructura. Esto se ejemplifica en estos dos fragmentos de código:

```
<! HTML>
<bold>Cámara</bold>
<bold>1200</bold>

<! XML>
<producto>Cámara</producto>
<precio>1200</precio>
```

En la versión HTML se tienen dos cadenas de texto que serán presentadas con negritas, pero no contiene información adicional. En la versión XML se sabe que Cámara es un producto, adicionalmente se puede indicar al programa encargado de presentar la información que coloque todos los productos con el tipo de letra deseada, que haga búsquedas o convierta el archivo de datos en una tabla.

Los archivos de formato binario propietario presentan el inconveniente de ser difíciles de leer e interpretar; aquí es donde XML presenta una gran ventaja y se convierte en la tendencia de las nuevas aplicaciones.

2.4.2 LibXML

LibXML es una biblioteca para el manejo de XML desarrollada en ANSI "C" para el proyecto GNOME, consta de varios módulos, de entre los que destacan:

- Capa de Entrada/Salida
- Capa cliente de FTP y HTTP
- Capa de Internacionalización
- El parser XML y su interfaz SAX
- Un parser HTML

Para poder hacer uso de esta biblioteca, es necesario incluir dentro de los fuentes las directivas #include **<libxml/xmlmemory.h>** , **#include <libxml/parser.h>** y pasar algunas opciones y banderas al compilador. Para facilitar esta labor, existe un pequeño programa que se encarga de determinar dichas opciones. La forma de utilizarlo dentro de un archivo de configuración configure.in es:

```
dnl Get libxml flags & libs
  AC_PATH_PROG(xml_config, xml-config)
  if test "x$xml_config" = "x";   then
      AC_MSG_ERROR([*** xml-config not found.])
  fi
  XML_CFLAGS=`$xml_config --cflags 2/dev/null`
  XML_LIBS=`$xml_config --libs 2/dev/null`
  CPPFLAGS="$CPPFLAGS $XML_CFLAGS"
  LIBS="$LIBS $XML_LIBS"
```

LibXML posee varias funciones de alto nivel para la validación y manipulación de documentos y las etiquetas que contiene. Las que se utilizaron para este proyecto son:

xmlGetProp();	Obtiene la cadena que se encuentra como atributo en una etiqueta
xmlNodeListGetString();	Obtiene la cadena correspondiente a un nodo.
XmlParseFile();	Revisa un documento para verificar que sea XML válido y extrae su estructura.

2.5 Glade

Glade[19] es un programa de desarrollo visual de aplicaciones. Es parte del escritorio GNOME, pero puede también servir para desarrollar aplicaciones que corran en cualquier plataforma, a condición de que se tenga instalado GTK+.

La filosofía de trabajo de Glade es muy semejante a la de otros programas RAD (Diseño Rápido de Aplicaciones) tales como Visual Basic o Delphi, en el sentido que el usuario diseña la aplicación de forma visual y posteriormente asigna funciones a cada evento de interés.

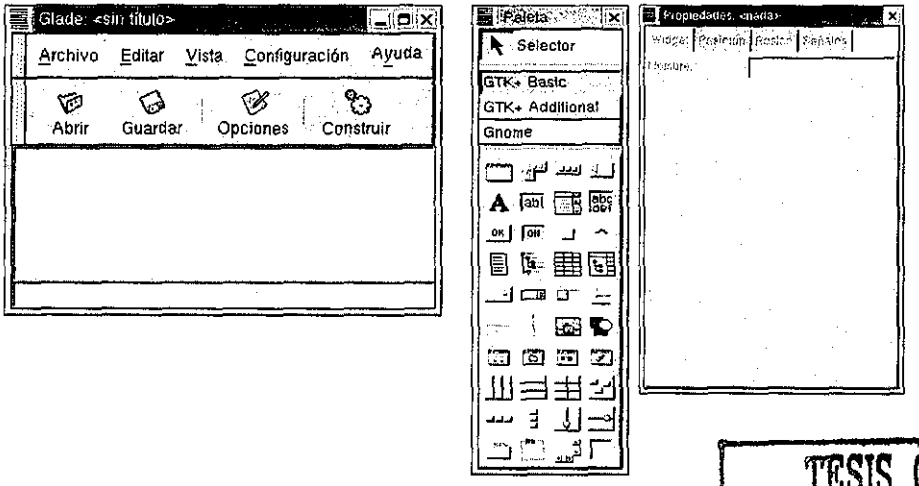


Figura 2.3 Aspecto de Glade

[19] http://glade.gnome.org

Glade no es una herramienta terminada, está en etapa de desarrollo y aún no es fácil de usar. Por ejemplo, para escribir el código de las funciones que corresponden a los eventos es necesario recurrir a un editor de textos externo. La ejecución de los guiones de configuración y la compilación se tienen que realizar desde línea de comandos. A pesar de esto, es posible crear aplicaciones de buena calidad con Glade.

2.6 Componentes para Material 3D

De todas las bibliotecas y herramientas mencionadas en este capítulo se eligieron únicamente siguientes:

GTK+ para el desarrollo de la interfaz. Debido a que las aplicaciones realizadas sobre GTK+ son muy fáciles de migrar a casi cualquier plataforma. Además el número de personas que programan con esta biblioteca es considerable y sé encuentra en aumento.

Mesa3D para la previsualización de los modelos. La decisión se debe al tipo de licencia, dado que el uso de Mesa3D no impone restricciones para utilizarlo en proyectos de software libre.

Como herramienta de diseño para la interfaz se utilizó Glade, porque es la única que soporta GTK+ de forma nativa. Ya que Glade puede generar código en diferentes lenguajes, se tuvo que elegir uno y éste fue el lenguaje "C" **[1]** porque es el lenguaje en el que están escritas las bibliotecas utilizadas.

Capítulo 3 Análisis de "Material 3D"

En este capítulo se describen los métodos utilizados y los resultados obtenidos en el análisis de los usuarios de modelado y animación 3D. Se describen los tipos de usuario que se contemplan y sus características. Se presentan los resultados de la evaluación de algunos programas comerciales con respecto a criterios ergonómicos. En cuanto al análisis de la tarea del usuario se incluyen los resultados de pruebas con usuarios realizadas en condiciones controladas **[17]**.

Estos análisis serán útiles en la determinación de las tareas principales y en cómo las realiza el usuario, adicionalmente brindará información relacionada con los intereses, expectativas y problemas comunes en el uso de este tipo de programas. Todo lo anterior servirá de base para el diseño de "Material 3D"

3.1 Análisis de programas de modelado y animación 3D.

Una primera aproximación con los programas de modelado y animación 3D se realizó con dos programas: Moray 3.1 y con Blender 1.0, los cuales fueron revisados de acuerdo con los criterios ergonómicos descritos en el apéndice "C".

3.1.1 Análisis del programa Moray 3.1

Moray[20] es un programa de modelado en 3D, "shareware" (copia de evaluación). El programa es completamente funcional, y no caduca como es el caso de otros programas de evaluación. La restricción radica en que si un archivo se guarda en disco cierta cantidad de veces, el programa envía un mensaje indicando que está haciendo uso intensivo del programa y que debiera registrarse. En caso de no hacerlo se perderá el acceso a dicho archivo.

En general el programa está bien hecho, la pantalla de inicio se muestra en la figura 3.1.

[20] http://www.stmuc.com/moray/

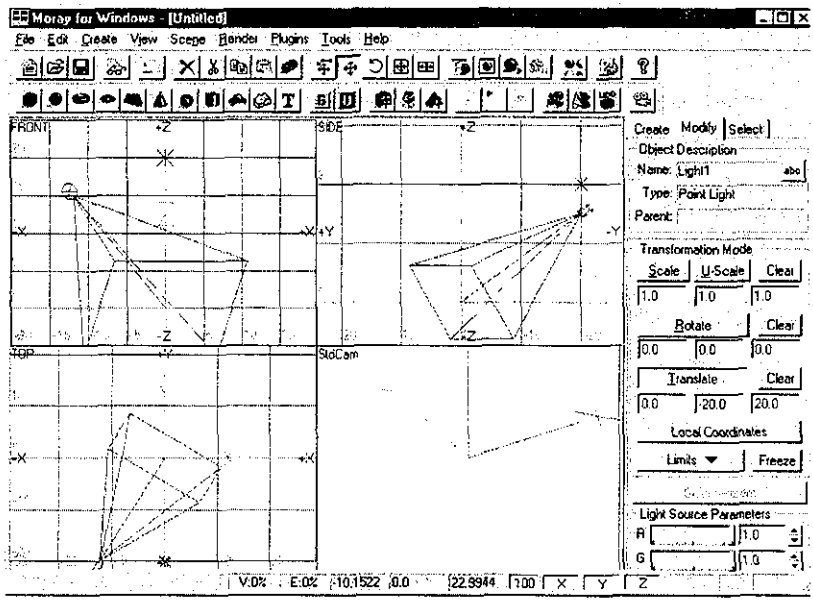


Figura 3.1 Pantalla principal del programa Moray 3.1

Los resultados de la evaluación son los siguientes:

Prompting: es adecuado, ya que se presentan de entrada las posibles acciones a seguir: crear, modificar, seleccionar, o las posibles alternativas del menú crear (opción por defecto).

Retroalimentación Inmediata: Es muy adecuada, ya que toda acción es inmediatamente reflejada en la pantalla.

Legibilidad: La legibilidad es moderadamente buena, salvo que parte de las opciones tienen que verse desplazando la parte derecha de la pantalla (algo no muy evidente).

Concisión: La mayor parte de las acciones está entre 1 y 2 pasos, pero en particular existe un problema cuando se desea colocar textura en un objeto, entonces se requiere de realizar las siguientes tareas:

1.- Llamar al editor de materiales (figura 3.2)

2.- Dado que el editor de materiales no contiene ningún material (por defecto así aparece al inicio), es necesario importar o crear un material, se continúa el ejemplo importando una librería de texturas:

3.- Seleccionar la opción de importar material (icono carpeta con signo +).

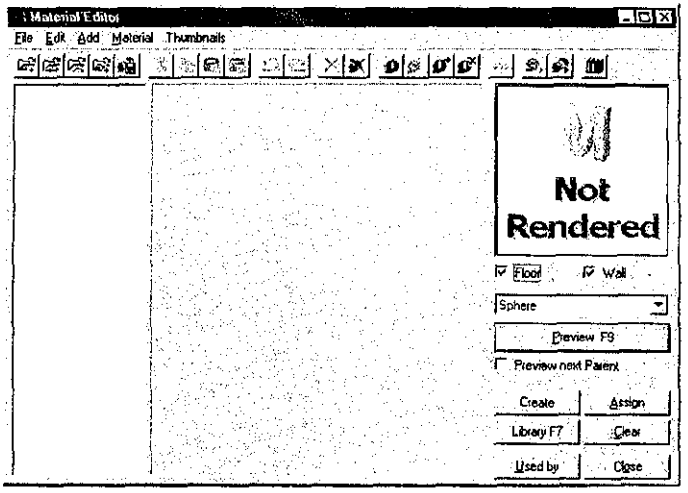


Figura 3.2 Ventana del editor de materiales

4- Seleccionar una textura y la previsualizarla (figura 3.3).

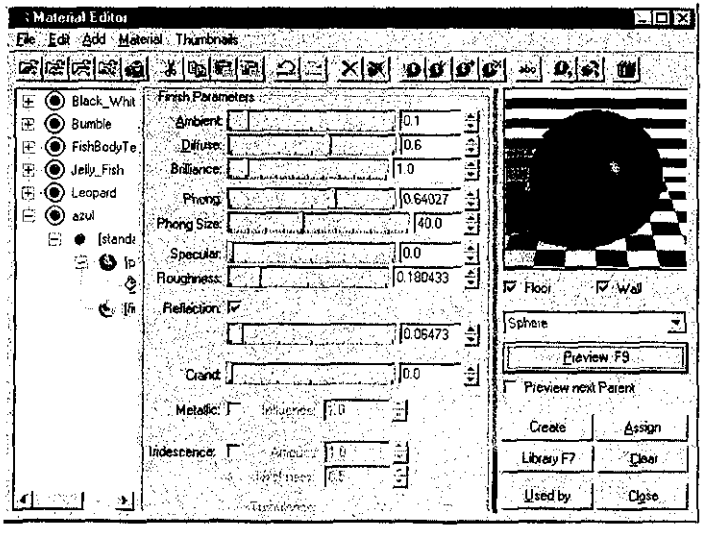


Figura 3.3 Previsualizando una textura

5.- asignar el material con el botón "assign"
6.- Cerrar el editor de materiales con el botón "close"

Como puede verse son muchos pasos para la realización de una tarea que es central y frecuente (el resto de las tareas importantes se realizan en muy pocos pasos).

Manejo de Errores: La mayoría de los mensajes de error son de muy buena calidad, ya que indican de forma clara cuál es la causa del error y la forma más probable de eliminarlo, como se ejemplifica en la figura 3.4.

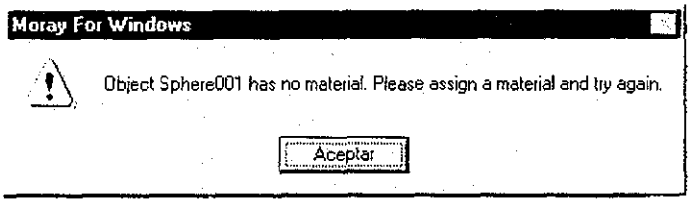


Figura 3.4 Mensaje de error que indica la causa y posible solución.

El error anterior es producido por intentar generar la imagen (rendering) de una escena en la que algún objeto no tiene una textura asignada, el mensaje muestra qué generó el error (en este caso lo generó la esfera, al no tener textura), e indica la forma de solucionar el problema.

Respecto a la prevención de los errores, también maneja de manera correcta la mayoría de las situaciones. Por ejemplo si una operación no está permitida dentro de cierta parte de la tarea, dicha opción queda desactivada. Las cajas de diálogo corrigen muchos de los posibles errores, por ejemplo al poner un factor de escala de 0 todas las coordenadas del objeto coincidirían y generaría un problema en el algoritmo de generación de la imagen. Así, cuando uno teclea scale = 0.00000 es automáticamente cambiado a 0.00001.

Consistencia: El manejo de los objetos es consistente en todos los menús y el manejo de los modificadores y vectores de transformación se manejan de manera homogénea.

Compatibilidad
Los menús están colocados en las mismas posiciones que la mayoría de los programas para MS-Windows (file, edit, etc).

3.1.2 Análisis del programa Blender

Blender es un programa realizado sobre OpenGL, es orientado a objetos y con él se pueden hacer trabajos muy profesionales. El programa es gratuito, pero la empresa que lo desarrolla sigue conservando los derechos sobre él.

Su uso es complicado y prácticamente imposible sin un manual. Los manuales para el programa existen, pero los vende la empresa creadora.

Al momento de ejecutarlo aparece una pantalla como la mostrada en la figura 3.5, debido a lo pequeño de los botones y las etiquetas poco significativas no resulta sencillo encontrar la lógica de trabajo.

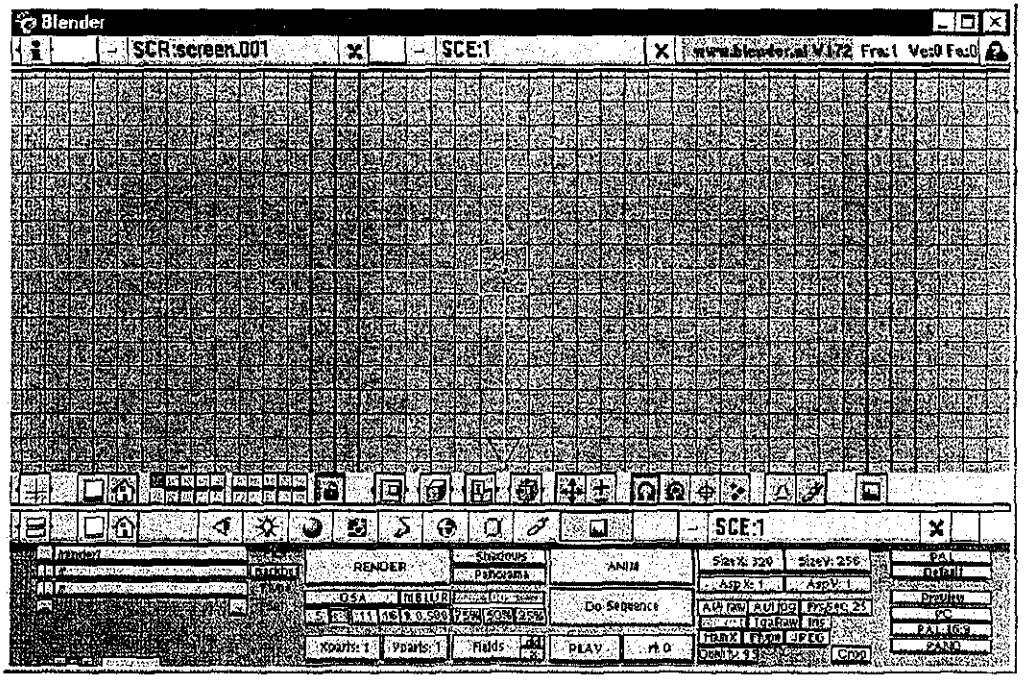


Figura 3.5 Pantalla de trabajo del programa "Blender"

Los principales errores de los que adolece son:

- **Prompting** : Inadecuado: No es evidente lo que se puede hacer ni lo que el programa espera del usuario.

- **Guía**: Mala. No es posible determinar en qué etapa de un proceso se encuentra. Cuando se presiona un botón, cambia la pantalla, pero no se puede determinar qué acciones son posibles a partir de ésta, ni cómo se puede regresar al estado anterior.

- **Densidad de información:** La cantidad de íconos es muy grande, así como muchas ventanas pequeñas (muy pequeñas) y muchos valores.

3.2 Análisis de los usuarios

Como parte del diseño centrado en usuario se analizó qué tipo de usuarios utilizan programas de modelado y animación, así como sus necesidades y expectativas y sus objetivos principales con respecto a la utilización del modelado en tres dimensiones.

3.2.1 Clasificación de usuarios

Los usuarios de programas de modelado 3D fueron clasificados en tres tipos principales:

- **Usuarios comerciales**: Su objetivo principal es la publicidad, buscan realismo visual, es decir, que se vea de forma semejante a la realidad.
- **Usuarios científicos**: Su interés principal consiste visualizar objetos que se apeguen a modelos matemáticos. El realismo que buscan se refiere más a la fidelidad con el modelo.

- **Usuarios de ingeniería**: Sus objetivos son muy semejantes a los objetivos de los científicos: representar modelos por medio de objetos tridimensionales. Dos de sus metas son: visualizar cómo se ven o se verán los modelos una vez construidos, y visualizar valores de una forma más fácil de entender, Por ejemplo, la representación de los esfuerzos en una presa por medio de colores.

3.2.2 Análisis de usuarios de tipo comercial

Para determinar las necesidades de este grupo se utilizó el cuestionario que se encuentra en el anexo A7. El resumen de los resultados de dichos cuestionarios se muestra en las figuras 3.6 a 3.9:

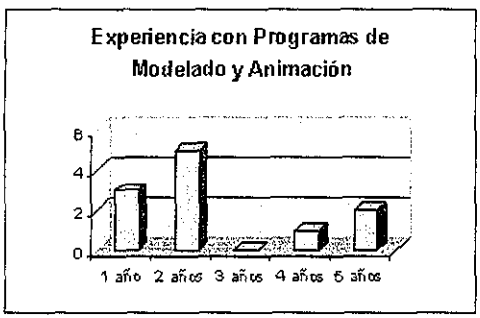


Figura 3.6 Experiencia con programas de modelado

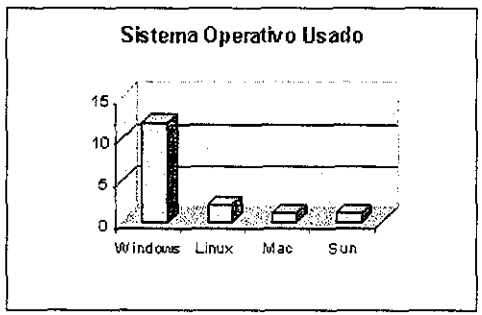


Figura 3.7 Sistema operativo usado

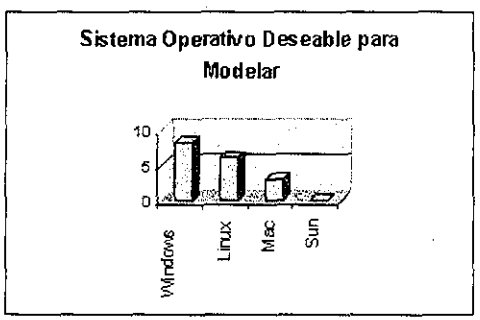


Figura 3.8 Sistema operativo deseable para utilizar programas de modelado 3D.

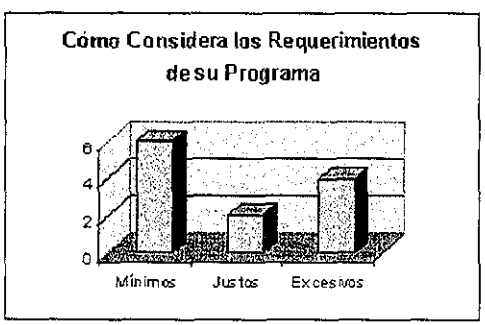


Figura 3.9 Consideración con respecto a los recursos

De estos resultados se puede concluir que el sistema operativo más usado es MS-Windows y las plataformas más deseables para modelado son MS-Windows y Linux.

Adicionalmente a las herramientas que ya poseen sus programas, los usuarios desean las siguientes:

- Previsualización de animaciones.
- Render (generación de imágenes) más rápido.
- Más texturas y transparencias.
- Facilidades para crear texturas metálicas.
- Metamorfosis entre objetos. Con facilidades para configurar el número de cuadros intermedios.
- Editor de explosiones.
- Detección de colisiones de fácil manejo.
- Un lenguaje de programación integrado que permita crear objetos por medio de procedimientos y que facilite movimientos complejos. Todo esto con un lenguaje sencillo, que no requiera mucho tiempo para entenderse.

3.2.3 Análisis de usuarios de tipo científico

Para obtener información referente a este punto se recurrió a entrevistas con el Biólogo José Luis Villareal, Jefe del Laboratorio de Visualización de la Dirección General de Computo Académico (DGSCA) de la UNAM, y con Armando Ricalde, quién labora en el Instituto de Astronomía realizando animaciones para el proyecto "Telescopio Infrarrojo Mexicano" (TIM)[21], quién también ha realizado animaciones de simulaciones de fenómenos astronómicos[22].

De las entrevistas se concluyó que: la mayoría de los programas comerciales son inadecuados para aplicaciones de corte científico, debido sobretodo a que los programas de modelado 3D brindan poco control sobre ciertos elementos. Por ejemplo, si un investigador desea realizar una simulación consistente en colocar algunas miles de esferas en un medio gravitatorio de prueba, requiere control sobre cada una de las esferas, algo que los programas comerciales no permiten.

Los programas comerciales más utilizados (3D Studio Max **[9]** y Maya) poseen herramientas para el manejo de dinámica de partículas, pero están pensadas para generar efectos que incrementen el realismo de una imagen, por lo que el control del movimiento para cada partícula no está contemplado.

Otra de las necesidades de este tipo de usuarios es referente a visualización de mallas, por ejemplo para representar modelos de elevación digital o esfuerzos sobre estructuras. En ambos casos es indispensable tener información asociada a cada elemento. Dicha información debería poder ser definida por el usuario y eso no ocurre en los programas comerciales.

[21] http://www.astroscu.unam.mx/tim/index.html
[22] http://tonatiuh.astroscu.unam.mx/tesis/index.html

3.2.4 Usuarios de ingeniería

Los usuarios del área de ingeniería tienen necesidades específicas, Una de las más importantes es la representación de modelos con datos asociados, por ejemplo, las mallas de elemento finito.

Este análisis está basado en las experiencias de colaboración con el área de geotecnia del Instituto de Ingeniería[23]. Uno de los principales problemas abordados fue la representación de mallas de elemento finito donde los esfuerzos se visualizaran por medio de un código de colores.

A pesar de la diferencia de los objetivos y procedimientos entre usuarios de ingeniería y usuarios de orientación comercial, se pueden notar similitudes en la forma de crear los modelos:

En ambos casos el orden es el siguiente:

- Creación del modelo
- Asignación de textura (o color que representa un valor, en el caso de ingeniería)
- Opcionalmente la creación de una animación

La creación del modelo difiere en que mientras en el modelado comercial se toman medidas de la realidad, o se crean para ajustarse a la imaginación de quien diseña, en el área de ingeniería los datos se obtienen a partir de medidas de campo o de modelos matemáticos. A pesar de la diferencia de los procedimientos en el fondo son muy semejantes.

3.3 Análisis de la tarea del usuario

Una parte fundamental del análisis se realizó sobre la tarea del usuario, esto es, qué es lo que los usuarios hacen y cómo. Es muy importante determinar cuál es el orden lógico de las acciones que realiza mentalmente, así como las reacciones que tiene el usuario con respecto al comportamiento de la interfaz **[18]**.

[23] Un agradecimiento especial al M. En I. Roberto Magaña del Toro por su valioso tiempo y experiencia.

3.3.1 Metodología usada

Se utilizó una variante del método "del mago de Oz" **[26]**, con la disposición que se muestra en la figura 3.10 para observar cómo desarrollaban una tarea diferentes usuarios bajo diferentes condiciones.

Figura 3.10 Configuración utilizada para las pruebas de análisis de la tarea

La metodología fue la siguiente: se invitó a un par de personas, una que fuese experta en Maya y otra que no supiese utilizarlo, y se le pidió al novato que estuviese del lado del observador y mediante un transmisor le indicará al experto qué hacer. El novato tenia el dibujo del modelo que se tenía que elaborar. Para explicarles qué hacer se utilizaron los protocolos descritos en el anexo A como A3 y A4.

Toda la interacción quedó registrada en video y al terminar los usuarios contestaron el cuestionario A5.

Después se le pidió al experto que realizara otro modelo pero en esta ocasión sólo. Para esto se empleó el protocolo A6.

Al terminar se tenían videos de la pantalla de la aplicación, es decir, qué hacía el usuario con la aplicación y el video del usuario y sus reacciones registradas por la cámara, así como el audio de las sesiones.

Otras pruebas se realizaron en el Laboratorio de Computo del Centro Tecnológico Aragón[24], empleando la configuración que se muestra en la figura 3.11.

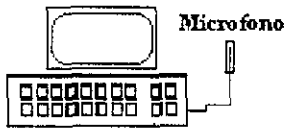


Figura 3.11 Configuración con registro en la PC del sujeto de observación

Dicha configuración es muy austera, ya que consta únicamente de una computadora, un micrófono y un programa que captura la pantalla en formato AVI. Dicho programa se llama Snag-it y puede ser descargado de la red de forma gratuita.

3.3.2 Resultados de las pruebas

Los resultados en cuanto a uso de herramientas se aprecian en la figura 3.12 y con un análisis de la lógica de trabajo se propone un orden de dichas herramientas como aparece en la tabla 3.1.

Archivo	Edición	Modificar		Construir	Objetos	Vista	Render
Salvar	Seleccionar	Transladar	Nombrar	Agrupar	Esfera	Cambiar	Render
	Copiar	Escalar	obj.	Unir	Cilindro	vista	
	Pegar	Rotar		Diferencia	Cubo	Zoom	
	Editar obj.	Ajustar a			Cono		
	Borrar	borde			Spline		
	Duplicar				Toroide		
	Espejo				Cámara		
	deshacer						

Tabla 3.1 Herramientas más utilizadas agrupadas por función

[24] Gracias a la colaboración de Felipe de Jesús Gutiérrez.

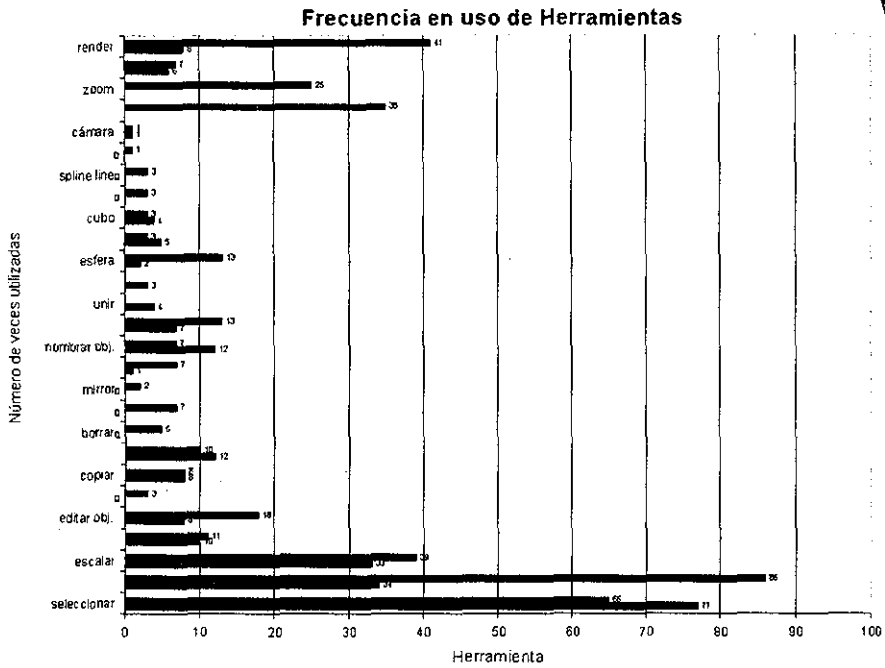


Figura 3.12 Gráfica de frecuencia de uso de herramientas

Del análisis de las pruebas realizadas en el laboratorio de interacción humano – máquina se obtuvo el árbol MAD* general, figura 3.13.

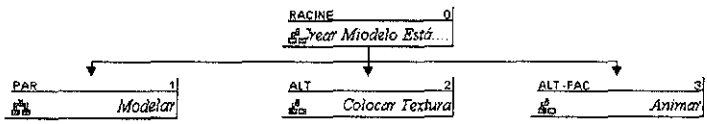


Figura 3.13 Arbol del usuario de orientación comercial

Cada una de estas tareas es todo un proceso y puede desglosarse en una rama para cada tarea principal, en la figura 3.14 se muestra la rama de la tarea "Modelar".

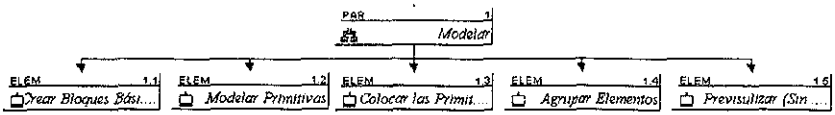


Figura 3.14 Rama para la tarea "Modelar"

La tarea "Colocar textura" se muestra en la figura 3.15. En ella es fácil observar que algunas subtareas son facultativas, es decir, pueden o no realizarse dependiendo del objetivo del usuario.

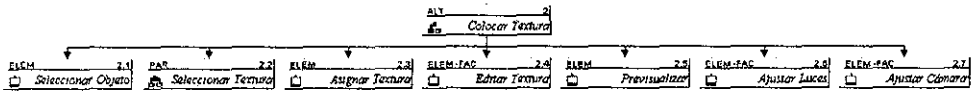


Figura 3.15 Rama para la tarea "Colocar Textura"

La tarea "Animar" es facultativa, esto es, debido a que un usuario puede hacer un modelo tridimensional estático o puede animarlo en una etapa posterior, pero no puede realizar la tarea "Animar" si no ha realizado antes la tarea "Modelar". La rama del árbol MAD* correspondiente a la tarea "Animar" se muestra en la figura 3.16.

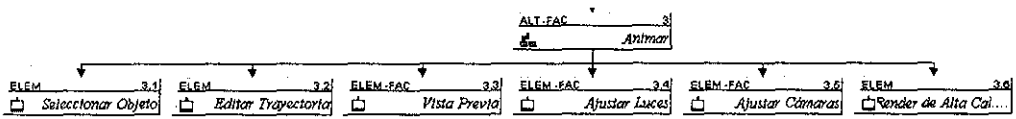


Figura 3.16 Rama para la tarea "Animar"

El análisis de este árbol, con respecto a los obtenidos a partir de entrevistas para usuarios de orientación científica y de ingeniería, muestra semejanzas en la estructura básica, ya que en todos es necesario primero crear el modelo, colocar textura y opcionalmente animar. Pero al extender las ramas se aprecia que la forma en que se realizan dichas tareas varía.

Dado que la interfaz debe reflejar la misma lógica de estos árboles, es evidente que se tendrá un menú general, que corresponde al árbol principal y acceso a diversas herramientas dependiendo del tipo de usuario de que se trate.

Capítulo 4 Diseño e implementación de "Material 3D"

En esta etapa se diseñaron y construyeron tanto las estructuras que contendrán los objetos, los algoritmos que generan las primitivas gráficas y las funciones básicas que manipulan dichos objetos.

En algunos casos, como el que se refiere al tipo de representación de los objetos dentro de la computadora, se requirió tomar decisiones entre diferentes alternativas, seleccionando aquellas que se ajustaran mejor a los objetivos del proyecto.

La elección de primitivas y herramientas de "Material 3D" para diseñar e implemetar es producto de los análisis realizados en el capítulo tres. La elección de las bibliotecas se basa en los resultados del capítulo dos.

No se menciona mucho respecto a la animación, porque está en un estado preliminar y todavía es susceptible de cambios importantes.

4.1 Consideraciones relativas a la representación de superficies

Para representar modelos tridimensionales por computadora predominan dos enfoques: Representar los objetos por medio de ecuaciones, o bien representarlos por medio de mallas poligonales **[21]**.
Por ejemplo, para representar una esfera:

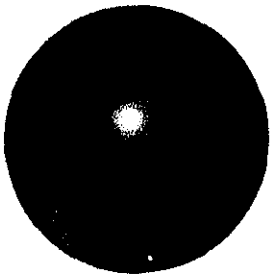


Figura 4.1 Imagen generada por medio de una ecuación

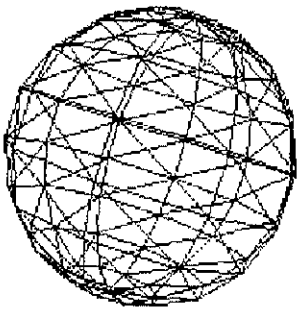


Figura 4.2 Malla de alambre para una esfera

En el ejemplo que se muestra en las figura 4.1 se encuentra una imagen generada por medio de la ecuación $x^2 + y^2 + z^2 = 1$, que es la ecuación de una esfera con centro en el origen y radio 1.0. En la figura 4.2 se tiene una malla de alambre que representa una esfera de radio 1.0.

Dependiendo de qué método se utilice, la representación del modelo en la computadora será diferente: mientras que en la representación por ecuaciones sólo se almacenan los parámetros, en la malla se debe almacenar una lista de vértices y una lista de polígonos; cada polígono almacena los índices de los vértices que lo conforman.

Cada uno de estos métodos presenta ventajas y desventajas. Por ejemplo, las representaciones de mallas tienden a presentar poco detalle. Para obtener detalles semejantes a las representadas por ecuaciones se requieren mallas finas, las cuales contienen un elevado número de polígonos. (figura 4.3).

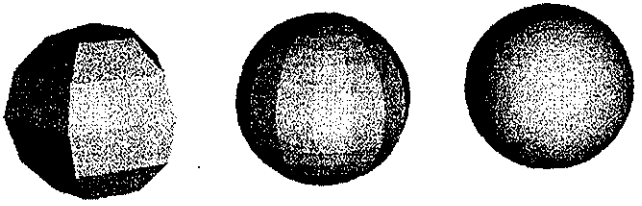


Figura 4.3 Esferas generadas respectivamente con 50, 450 y 20,000 Triángulos

Como se observa en la figura 4.3, a mayor número de polígonos, mayor nivel de detalle. Esto por supuesto también implica un mayor uso de recursos de almacenamiento.

En general las ventajas y desventajas de estas dos formas de representación de superficies son:

Basadas en ecuaciones	**Basadas en mallas poligonales**
Ventajas	**Desventajas**
Ocupan muy poco espacio	Ocupan mucho espacio
Nivel de detalle muy alto	Bajo nivel de detalle
Desventajas	**Ventajas**

Muy difíciles de modelar	Fáciles de modelar
Dependen por completo del procesador para la generación de imágenes.	Sacan provecho de las tarjetas aceleradoras de gráficos, que están optimizadas para generar triángulos

Tabla 4.1 Ventajas y desventajas de los distintos tipos de representación de superficies

Para este proyecto se ha elegido la representación por medio de mallas, sobretodo por que permiten una manipulación más sencilla, y se adecua mejor a los datos de ingeniería y ciencias, como lo son las mallas de elemento finito, la reconstrucción en volumen de imágenes médicas o científicas.

4.2 Creación de las primitivas gráficas

Los modelos que el programa manejará serán obtenidos como resultado de procesos numéricos en problemas de ciencias e ingeniería, pero también debe tener la capacidad de crear sus propios modelos, tanto para aplicaciones de ciencia e ingeniería, como para aquellas de modelos con fines comerciales o didácticos.

Para poder crear modelos es necesario construir las piezas elementales, es decir, las primitivas gráficas, como lo son: la esfera, el cilindro, el paralelepípedo, etc.

4.2.1 Creación de la malla para el paralelepípedo

Objetivo: Crear una malla tridimensional que envuelva una caja y permita un número variable de divisiones en cada eje. Los vértices de la malla que son compartidos por dos o tres caras no deben considerarse más de una vez.

Análisis.

Una caja tiene 6 caras, cada una puede tener un número variable de divisiones para cada eje. Para ejemplificar se creó una caja con cuatro puntos de división por cara (Figuras 4.4 y 4.5):

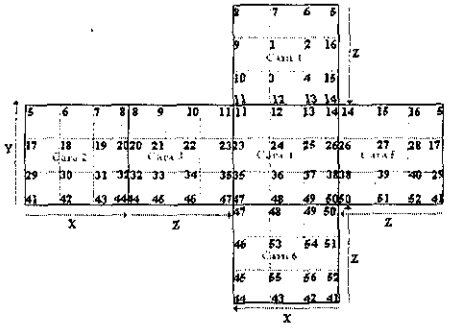


Figura 4.4 Arreglo para recortar

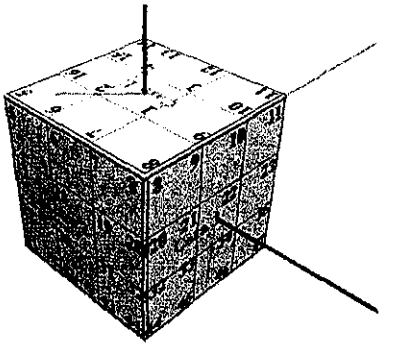


Figura 4.5 Caja armada

El primer paso es calcular los vértices de la caja, que para este ejemplo son 56. Debido a que los vértices compartidos sólo pueden ser contados una vez, es necesario encontrar una configuración fácil de construir en la computadora por medio de iteraciones. Dicha configuración se muestra en la figura 4.4, donde se observa una caja para recortar y armar (como las que se utilizan en las primarias), a la que se le han agregado etiquetas para las caras, se han numerado sus vértices y se indica el sentido de los ejes.

Dado que las caras 2, 3, 4, y 5 tienen vértices numerados de forma consecutiva son relativamente fáciles de calcular. Las caras 1 y 6 son las "tapas" de la caja, para éstas sólo es necesario calcular los vértices internos, ya que los externos ya han sido calculados por ser compartidos con otras caras. El algoritmo se muestra a continuación.

```
Delta2= 2*(numZ + numX)-4;
 delta1 = (numX-2)*(numZ-2);                              {Cara número 2}
 for contY = 1 to numY do
   for contX = 1 to numX do
     begin
        puntos[delta1+(contY-1)*delta2 + contX].x = (contX-1)/(numX-1)-0.5;
        puntos[delta1+(contY-1)*delta2 + contX].y = 0.5-(contY-1)/(numY-1);
        puntos[delta1+(contY-1)*delta2 + contX].z = -0.5;
     end;



delta1 = (numX-2)*(numZ-2) + numX -1;              {Cara número 3}
for contY = 1 to numY do
   for contZ = 2 to numZ do
     begin
        puntos[delta1+(contY-1)*delta2 + contZ].x = 0.5;
        puntos[delta1+(contY-1)*delta2 + contZ].y = 0.5-(contY-1)/(numY-1);
        puntos[delta1+(contY-1)*delta2 + contZ].z = (contZ-1)/(numZ-1)-0.5;
    end;

  delta1 = (numx-2)*(numZ-2) + numX+ numZ -2;          {Cara número 4}
  for contY  = 1 to numY do
   for contX = 2 to numX do
     begin
        puntos[delta1+(contY-1)*delta2 + contX].x = 0.5 - (contX-1)/(numX-1);
        puntos[delta1+(contY-1)*delta2 + contX].y = 0.5-(contY-1)/(numY-1);
        puntos[delta1+(contY-1)*delta2 + contX].z = 0.5;
     end;



delta1 = (numX-2)*(numZ-2) + 2*numX+numZ -3;         {Cara número 5}
 for contY = 1 to numY do
   for contZ = 2 to numZ-1 do
     begin
        puntos[delta1+(contY-1)*delta2 + contZ].x = -0.5;
        puntos[delta1+(contY-1)*delta2 + contZ].y = 0.5-(contY-1)/(numY-1);
        puntos[delta1+(contY-1)*delta2 + contZ].z = 0.5-(contZ-1)/(numZ-1);
     end;
```

```
for contZ = 2 to numZ-1 do                          {Cara número 1}
   for contX = 2 to numX-1 do
     begin
        puntos[(contZ-2)*(numX-2) + contX-1].x = 0.5 - (contX-1)/(NumX-1);
        puntos[(contZ-2)*(numX-2) + contX-1].y = 0.5;
        puntos[(contZ-2)*(numX-2) + contX-1].z = (contZ-1)/(numZ-1)-0.5;
     end;

  delta1 = (numZ - 2) * (numX - 2)+ numY ((2(numX+numZ))-4);       {Cara 6}

  for contZ = 2 to numZ-1 do
   for contX = 2 to numX-1 do
     begin
      puntos[delta1+(contZ-2)*(numX-2) + contX-1].x = 0.5- (contX-1)/(NumX-1);
      puntos[delta1+(contZ-2)*(numX-2) + contX-1].y = -0.5;
      puntos[delta1+(contZ-2)*(numX-2) + contX-1].z = (contZ-1)/(numZ-1)-0.5;
     end;
```

donde :

- **numX, numY y numZ**: son el número de divisiones para los ejes X, Y y Z respectivamente.
- **Puntos[ ]** es un arreglo donde se almacenan los vértices, el arreglo es de puntos3D.
- **delta1** es el desplazamiento en la numeración debido a los vértices internos de la cara 1.
- **delta2** es el desplazamiento debido a cada avance de fila en las caras 2, 3, 4 y 5, se calcula de la siguiente forma: Delta2 = 2*(numX+numZ-2).

4.2.2 Creación de la malla para la esfera

Análisis.

Se puede considerar a una esfera como una sucesión de *polígonos que aproximan a* circunferencias con una variación en su radio, el cual es una función del ángulo que forma la línea que une el radio con el nivel.
Ver figura 4.6

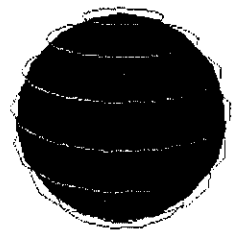


Figura 4.6 Aproximación por circunferencias

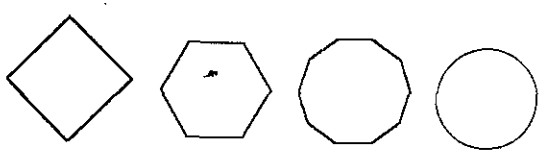


Figura 4.7 Aproximación a una esfera por medio de polígonos.

Una circunferencia se puede aproximar por medio de un polígono de n lados, la aproximación es más fiel en la medida en que n es mayor. (Figura 4.7)

Las formulas para calcular las coordenadas de los vértices de los lados del polígono son (Figura 4.8):

$x = r \cos(t)$

$y = r \sin(t)$ con $t = 360º / n$

donde : 'n' es el número de lados del polígono,
'r' es el radio de la circunferencia.

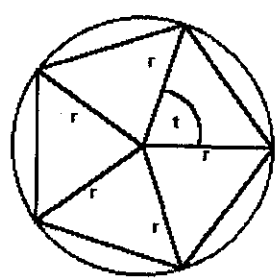


Figura 4.8 Cálculo de las coordenadas de los vértices del polígono.

Para calcular el radio de las circunferencias se utiliza el siguiente razonamiento:

Teniendo en cuenta que la circunferencia ecuatorial es la circunferencia máxima, se utiliza la función coseno que posee una curva que crece de 0 a 1 en el rango de 0 a ¶/2, y decrece de 1 a 0 en el rango de ¶/2 a ¶. Ajustando los valores de la función coseno desplazándola -90º. (Figura 4.9).

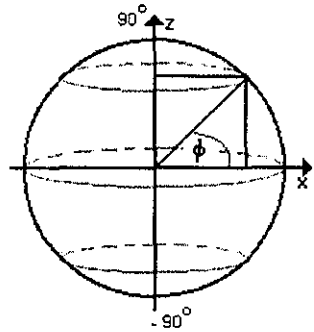


Figura 4.9 Cálculo de los radios de las circunferencias

Entonces se tiene que para calcular el radio de las circunferencias, se calcula la proyección sobre el eje X mediante la formula:

radio = R Cos (ø) ;

donde R es la magnitud del radio ecuatorial, y ø es el ángulo del centro de la esfera a la tangente de la circunferencia a la que se calcula el radio.

Para calcular la altura a la que se va a ubicar la circunferencia se calcula la proyección sobre el eje 'Z', mediante la formula: altura = R Sin (ø).

Teniendo el número de circunferencias que contendrá la esfera a lo largo de 'Z', ø variará de acuerdo a la siguiente formula: ø = (¶ / (1 + num_niveles)) * n - ¶/2

donde: 'num_niveles' es el número de circunferencias que habrá a lo largo de la altura de la esfera, y 'n' variará de 1 a 'num_niveles'.

Dados los vértices, se pueden crear los polígonos tomando cada punto p de una circunferencia del nivel n, y se une con los puntos m + 1 y n + 1 de modo que se obtenga un triángulo. De esta forma se obtiene para cada punto dos triángulos que son (p,n), (p,n+1), (p+1,n) y (p+1,n), (p,n+1), (p+1,n+1) (Figura 4.7). Sin embargo este procedimiento crea una malla de una circunferencia sin las tapas polares (Figura 4.11):

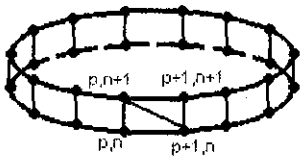


Figura 4.10 Forma de unir los puntos para obtener los triángulos

Figura 4.11 Esfera terminada pero sin las tapas polares

Las tapas polares se realizan calculando los puntos máximo (0,0,radio) y mínimo (0,0,-radio) y se unen por líneas a cada uno de los vértices del primer y último nivel (Figuras 4.12 y 4.13). Los triángulos están formados por las líneas que unen los vértices p_n con p_{n+1} y con p_{polar}.

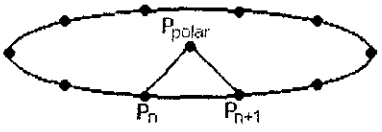


Figura 4.12 Triángulo formado por p_n, p_{n+1} y p_{polar}

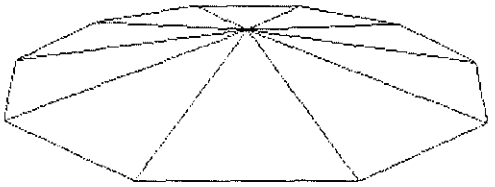


Figura 4.13 Casquete polar para completar la esfera

El resultado final es el siguiente (Figura 4.14):

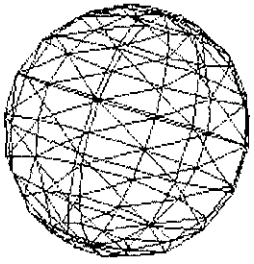


Figura 4.14 la malla de alambre terminada

4.2.3 Creación de la malla para el cilindro

Se puede considerar a un cilindro como una sucesión de circunferencias sobre una trayectoria lineal (Figura 4.15).

Figura 4.15 Aproximación a un cilindro por medio de circunferencias

Para calcular la altura de las circunferencias se utiliza el siguiente razonamiento (Figura 4.16):

Teniendo la altura del cilindro 'h', la altura de las circunferencias variara de 0 a 'h', suponiendo que variará a lo largo de 'Z'. Esto se puede calcular con la siguiente formula:

$$Zi = h / (num\_niveles - 1) * n$$

donde 'num_niveles' es el número de circunferencias que habrá a lo largo de la altura del cilindro, y 'n' variará de 0 a 'num_niveles'-1

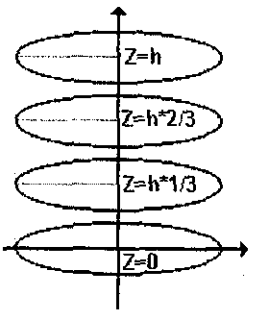


Figura 4.16 Cálculo de la coordenada Z para las circunferencias

El procedimiento anterior proporciona los puntos y los polígonos de la pared del cilindro. Las tapas se crean de forma muy semejante a los casquetes polares de la esfera (Figura 4.17).

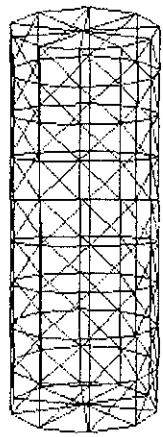


Figura 4.17 Malla de alambre para el cilindro

4.2.4 Creación de la malla para el cono

Se puede considerar a un cono como una sucesión de circunferencias sobre una trayectoria lineal, con una variación lineal de su radio (Figura 4.18).

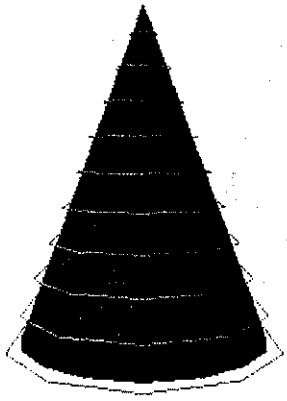


Figura 4.18 Esferas que aproximan un cono

Para calcular la altura de las circunferencias se utiliza el siguiente razonamiento:

Teniendo la altura del cilindro 'h', la altura de las circunferencias variará de 0 a 'h', suponiendo que variará a lo largo de 'Z', se puede calcular con la siguiente formula (Figura 4.19):

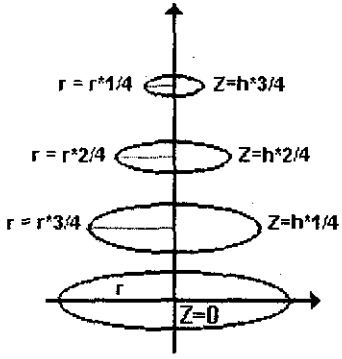


Figura 4.19 Cálculo de las alturas

$$hi = h / (num\_niveles) * n$$

Para el cálculo de los radios de las circunferencias, es mediante una proporción lineal, en la cual al variar la altura de 0 a 'h', simultáneamente variará el radio de 'r' a 0, calculándolo mediante la siguiente formula:

ri = r / (num_niveles) * m

donde 'num_niveles' es el número de circunferencias que habrá a lo largo de la altura del cilindro, 'n' variará de 0 a 'num_niveles'-1, y 'm' variara de 'num_niveles' a 1.

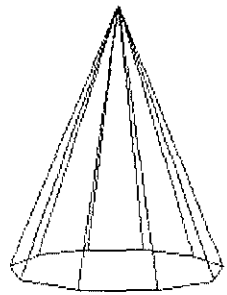


Figura 4.20 Detalle de la "tapa" del cono

En el procedimiento anterior se obtienen los puntos y los polígonos de la pared del cono. Sin embargo, lo que se obtiene es la pared de un cono truncado, sin los polígonos de la punta ni los de la tapa inferior.

La tapa se crea de manera similar a las tapas del cilindro, mientras que la malla de la punta se crea de manera similar a las tapas polares de la esfera.

El resultado final se muestra en la figura 4.21

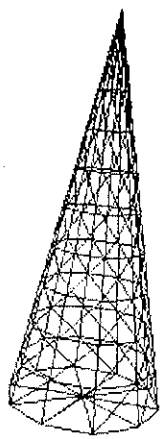


Figura 4.21 Malla final para aproximar un cono

4.2.5 Creación de la malla para el toroide[25]

Se puede considerar a un toroide como una sucesión de circunferencias con radio fijo, girando alrededor de un eje. La creación del toroide se basa en el concepto de figuras de revolución. Dada una figura, en este caso una circunferencia, formada sobre un plano determinado, se toma cada punto de ésta y se rota alrededor de un punto, que es el radio del toroide, como se muestra en la figura 4.22.

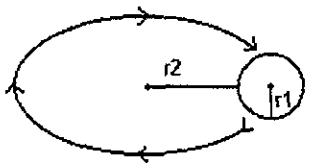


Figura 4.22 Elementos básicos del toroide

El toroide en particular tiene dos características básicas: el radio propio de la circunferencia en revolución (r1), y el radio de giro del objeto en revolución (r2).

Así como la circunferencia en revolución se puede aproximar mediante un polígono, también la trayectoria de giro se puede aproximar mediante un polígono de 'n' lados. Dicho polígono girará alrededor de eje restante, en este caso 'z' (Figuras 4.23 a la 4.25)

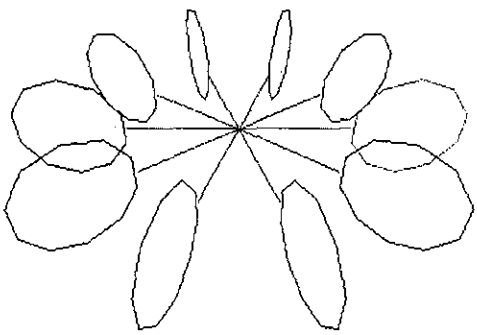


Figura 4.23 Polígonos en revolución

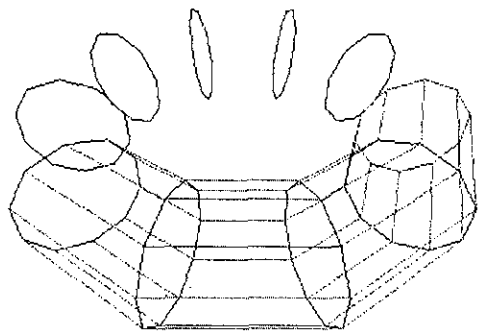


Figura 4.24 Mallado de los polígonos

[25] Un agradecimiento a Arturo Hernádez Sánchez por este planteamiento

De esta forma, es posible obtener la figura 4.25, a la que se puede triangular de manera muy sencilla.

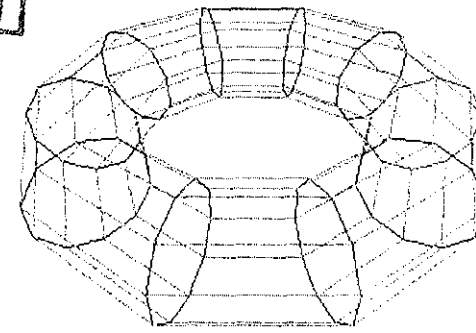


Figura 4.25 Mallado completo, previo a la triangulación

El razonamiento para girar los puntos de la circunferencia es como sigue: se crea la circunferencia inicial y se desplaza su centro del eje de giro ('z') 'r1+r2' unidades (sobre el eje 'x'). Una vez teniendo los puntos de ésta, se toma la distancia de cada punto al eje de giro (distancia en 'x') y se calcula la posición de éste según las siguientes ecuaciones.

$Xn,i = X0,i * Cos(2 * ¶ / pasos\_por\_giro * n)$

$Yn,i = Y0,i$

$Zn,i = X0,i * Sin(2 * ¶ / pasos\_por\_giro * n)$

con 'n' variando de 1 hasta (pasos_por_giro -1), e 'i' variando de 0 hasta el número de puntos por circunferencia.

Una vez que se tienen los puntos de las circunferencias, para componer los polígonos se sigue el siguiente método: se toma cada punto de una circunferencia dada situada en un determinado nivel, y se une con los puntos que se encuentran consecutivos a él, tanto del mismo nivel como del siguiente, de modo a que se forme un polígono. De esta forma quedará un rectángulo definido por los vértices (p,n), (p,n+1), (p+1, n+1), (p+1,n), en el caso de triángulos, el polígono anterior se descompone en dos: (p,n), (p,n+1), (p+1,n) y (p+1,n), (p,n+1), (p+1,n+1).

El resultado final se muestra en la figura 4.26

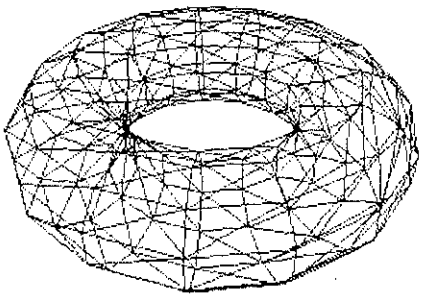


Figura 4.26 Malla que aproxima un toroide

Las primitivas bosquejadas en este capítulo no son las únicas posibles, Existe una amplia variedad de objetos susceptibles de ser implementados bajo la representación de mallas. Sin embargo, se optó por empezar con estas debido a que son las utilizadas, de acuerdo con los resultados de las pruebas mencionadas en 3.3.2.

Las siguientes primitivas a implementar en trabajos posteriores deberían ser algunas relacionadas con modelado orgánico:

- Parches Bezier
- Curvas Nurbs
- Metaesferas (Blobs)

4.3 Estructuras de datos

Para almacenar los objetos tridimensionales fue necesario diseñar e implementar estructuras de datos apropiadas **[12][21]**. Se optó por un esquema constructivo, es decir, primero se crea una estructura básica (el punto 3D) y posteriormente se crearon estructuras cada vez más complejas, las cuales utilizaban como componentes las estructuras diseñadas en el paso previo.

La importancia del diseño de las estructuras de datos para almacenar los objetos radica en la capacidad de representar las escenas y su contenido. En otras palabras, si las estructuras limitan la capacidad de representación, queda igualmente limitado lo que el programa puede hacer. Por ejemplo, si nuestro programa impone restricciones a los objetos (para que se ajusten a la estructura de datos), y dichas restricciones no existen en otros programas de modelado 3D, será prácticamente imposible importar los objetos de esos programas sin sufrir pérdidas en algunos atributos.

El formato de archivos de "Material 3D" depende en gran medida de las estructuras creadas para almacenar los objetos y las escenas.

El punto 3D es la estructura básica más simple y consta de 3 números reales para representar un punto en el espacio:

```
Typedef struct {
  gdouble x, y, z;              /* Vertex components */
} point3D;
```

Para almacenar los triángulos se empleó la siguiente estructura y declaración de tipo[26]:

```
typedef sruct {
  gint v1;              /* First vertex of triangle  */
  gint v2;              /* Second vertex of triangle */
  gint v3;              /*  Trhird vertex of triangle */
  guchar  r, g, b;      /* RGB color components */
}typTriangle;
```

La estructura principal es la que contiene a los objetos tridimensionales. Su contenido incluye el nombre del objeto, el número de vértices y de triángulos y arreglos de los mismos. Agregar nuevos atributos es sencillo y requiere de muy pocas modificaciones en el código. Su implementación es:

```
typedef struct {                /*  Material 3D object definition     */
   gchar objectName;            /* Object name                        */
   gint numVertext;             /* Number of vertex in the object     */
   gint numTriangles;           /* Number of triangles in the object  */
  gint typeObject;              /* Object type (sphere, box, etc      */
   point3D *Vertex;             / * Pointer to vertex array          */
   typTriangle *Triangles;      / * Pointer to triangles array       */
}M3Dobject;
```

Para almacenar todos los elementos de la escena se creó un arreglo de objetos. Por medio de un apuntador se puede manipular cualquier objeto de dicho arreglo:

```
m3Dobject  *listObjects;   /*        Object list in scene        */
```

Se tomó la decisión de que el formato nativo del programa Material 3D se definiría con XML, (esto se explica en el punto 2.4). Debido a esto, se crearon las estructuras que permitieran recuperar estos datos:

[26] Se utilizó ingles en los comentarios y en los nombres de las variables por ser lengua franca de la comunidad GNU, misma que continuará este proyecto.

```
typedef struct {
   xmlChar *name;
   xmlChar *numVertex;
   xmlChar *numTriangles;
   xmlChar *type;
   xmlChar *vertexArray[2000];
   xmlChar *trianglesArray[12000];
 }xmlM3Dobject, *xmlM3DobjectPtr;

 typedef struct {
   int numObjects;
   xmlM3DobjectPtr m3Dlist[50];
 }m3Dscene, *m3DscenePtr;
```

Estas declaraciones se encuentran en el archivo m3d_file.h y las funciones necesarias para llenar estas estructuras y convertirlas en estructuras del tipo vértices y triángulos se encuentran en el archivo m3d_file.c.

En esencia son las equivalentes a las utilizadas para almacenar los objetos y la lista de objetos, pero todos los elementos básicos son del tipo xmlChar, el cual es una variante del apuntador a caracter o cadena, esto debido a que XML es un formato de texto. Primero se lee el archivo, posteriormente se convierten a los tipos apropiados y por último se copian a sus equivalentes, las cuales sí se pueden utilizar para operaciones.

Para almacenar las transformaciones y movimientos útiles para animación se tienen la siguiente estructura:

```
typedef struct {
  gint transType;    /* Transformation type: Rotation, scale or translate  */
  gfloat x, y, z;    /* values for transformation                          */
 }motion;
```

Estas estructuras son la base que permite contener los objetos y están diseñados de manera que es posible agregar atributos adicionales. En estos momentos el diseño de la estructura que permite manejar animación es preliminar, y posiblemente sea sustituida por una estructura más compleja que permita no sólo movimiento simple a los objetos, sino también movimientos basados en física.

4.4 Modelado

La tarea de modelado consiste en la creación y modificación de objetos virtuales que sean semejantes a la realidad o a los objetos imaginados por el diseñador. Normalmente se desea que la apariencia y comportamiento de estos modelos sea lo más parecido posible a la realidad.

4.4.1 Modelado Jerárquico

Cuando se crean algunos objetos primitivos, en ocasiones es deseable poderlos manipular como un sólo objeto. Al conjunto etiquetado de primitivos se le denomina estructura **[22]**.

Una estructura puede contener varios objetos, por ejemplo es un conjunto de cajas y cilindros puede agruparse y etiquetarse como "mano", después se realiza lo mismo con otros grupos para crear "pie", "tronco" y "cabeza". Estas estructuras pueden a su vez agruparse y etiquetarse como "cuerpo". A esto se le denomina modelado jerárquico.

La utilización de modelado jerárquico facilita mucho la manipulación y comprensión de los modelos. La implementación de modelado jerárquico en este trabajo se realizó con la declaración de un tipo de dato denominado M3Dcgs, el cual contendrá el nombre de la estructura, el número de objetos en la estructura, el tipo de operación (0 = unión, 1 = intersección, 2 = diferencia) y por último la lista de objetos.

La declaración del tipo de dato M3Dcgs se muestra a continuación:

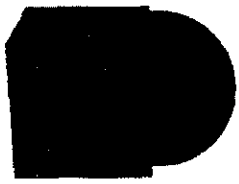
```
typedef struct {                /*  Material 3D object definition    */
  gchar objectName;             /* CGS name                          */
  gint numObjects;              /* Number of objects in the CGS      */
  gint typeCGS                  /* Operation type                    */
  m3Dobject  *listObjects;      /* Object list in CGS                */
}M3Dcgs;
```

4.4.2 Geometría sólida constructiva

La geometría sólida constructiva (CGS por sus siglas en inglés) permite crear objetos más complejos a partir de objetos primitivos u otros objetos CGS, empleando operaciones de conjuntos.

Las operaciones CGS básicas son:

- Unión: (A unión B), el resultado es lo que está en A o lo que está en B. Es equivalente a la suma de los objetos (figura 4.27).
- Intersección: (A intersección B), el resultado es lo que está en A y además está en B, es decir, lo que esta simultáneamente en ambos objetos (figura 4.28).
- Diferencia: (A menos B), el resultado es que lo que está en A pero no está en B (figura 4.29).



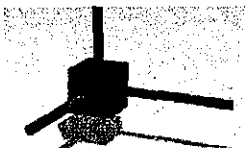
4.27 Unión

4.28 Intersección

4.29 Diferencia

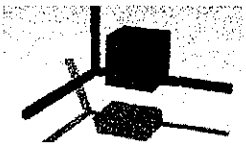
4.4.3 Modificaciones y transformaciones del modelo

Las transformaciones se refieren a los cambios que se realizan sobre el modelo en su conjunto, estas pueden ser translación, rotación y escala (figuras 4.30 a 4.33).

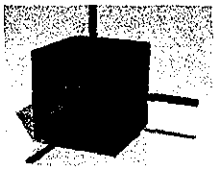


4.30 Imagen original

4.31 Rotación

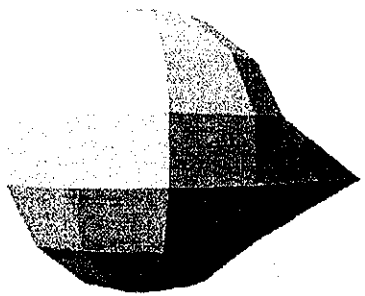


4.32 Translación



4.33 Escala

Las modificaciones involucran cambios en el interior del modelo, por ejemplo el mover un vértice (figura 4.34) o escalar sólo algunos vértices.



4.34 Modificación de sólo algunos vértices

4.5 Diseño de la interfaz de usuario

Una de las partes más importantes de una aplicación es la interfaz **[16]**, ya que para quien usa un programa, **la interfaz es el programa**, es decir, los usuarios consideran que lo que ven en pantalla es lo que pueden hacer. Es por ello que cuando algún programa posee funciones que no se muestran en la primera pantalla, sean utilizadas muy poco o nada.

El que un programa tenga una interfaz gráfica de usuario no es garantía de que sea fácil de usar. La facilidad de uso depende de un buen diseño, que refleje la forma en la que el usuario realiza sus tareas **[3][4]**.

Para realizar el diseño de la interfaz de "Material 3D" se utilizaron como base los resultados obtenidos en el capítulo 3.

Del árbol MAD* que se muestra en la figura 3.13 se observa que la tarea "modelar y animar" se divide en 3 subtareas principales, las cuales son:

- Modelar
- Colocar Textura
- Animar

Cada una de estas tareas es todo un proceso, en el cual se pueden utilizar varias herramientas. Debido a esto se decidió agrupar cada una de las diferentes herramientas bajo la pestaña correspondiente, obteniendo un bloc de herramientas (fig. 4.35).

Figura 4.35 Bloc de herramientas

Se creó una barra de herramientas que contuviera las herramientas convencionales, tales como abrir, guardar, copiar, pegar, etc., a la que adicionalmente se agregaron herramientas que se requieren en cualquier momento, ya sea mientras se modela o se anima. (figura 4.36).

Figura 4.36 Herramientas estándar y las comunes en modelado 3D.

Los menús de opciones se basan en los que son familiares para la mayoría de las personas que utilizan programas con interfaz gráfica de usuario tanto en Windows como en Linux, es decir, los típicos menús que contienen Archivo, Editar, Ver, etc. A estos

menús se agregaron los elementos propios de "Material 3D", como son crear, transformar, etc. los elementos principales se observan en la figura 4.37.

File Edit Create Transform View Render Settings Help

Figura 4.37 Cabeceras de los menúes de "Material 3D"

Para poder modelar objetos de tres dimensiones sobre un área de trabajo de sólo dos dimensiones (el monitor), se necesita del apoyo de alguna forma de representación que ayude a visualizar los cambios de cada objeto respecto a cada eje. La forma tradicional que se ha utilizado es mostrar los modelos en 3 vistas, para los planos xy, xz, yz y una vista adicional para una representación en perspectiva. Esta solución es la misma que se adoptó para el proyecto.

La implementación (Figura 4.38) se realizó en una tabla de 2x2 y en cada celda se colocó un marco que a su vez contiene un área de dibujo, excepto la vista en perspectiva, la cual en lugar de contener un área de dibujo contiene un área de dibujo Mesa3D.

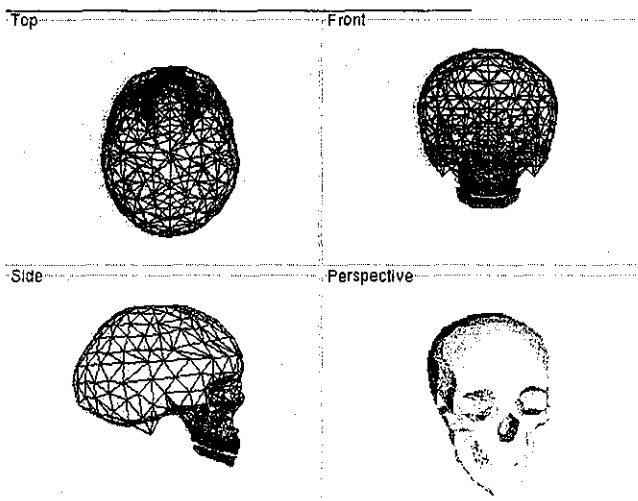


Figura 4.38 Areas de dibujo

Cuando en una escena se encuentran muchos objetos, la selección puede resultar complicada. Para facilitar esta labor se creó un árbol de objetos, del que pueden seleccionarse los objetos por su nombre (figura 4.39).

El árbol de objetos se encuentra dentro de un bloc, donde se pueden agregar pestañas para configurar diferentes aspectos de la escena.

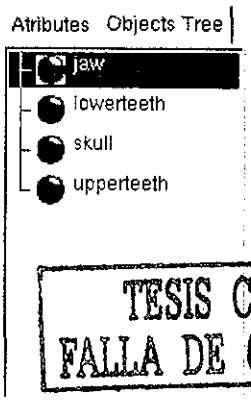


Figura 4.39 Árbol de objetos

El aspecto final de la aplicación en este momento, es el mostrado en la figura 4.40.

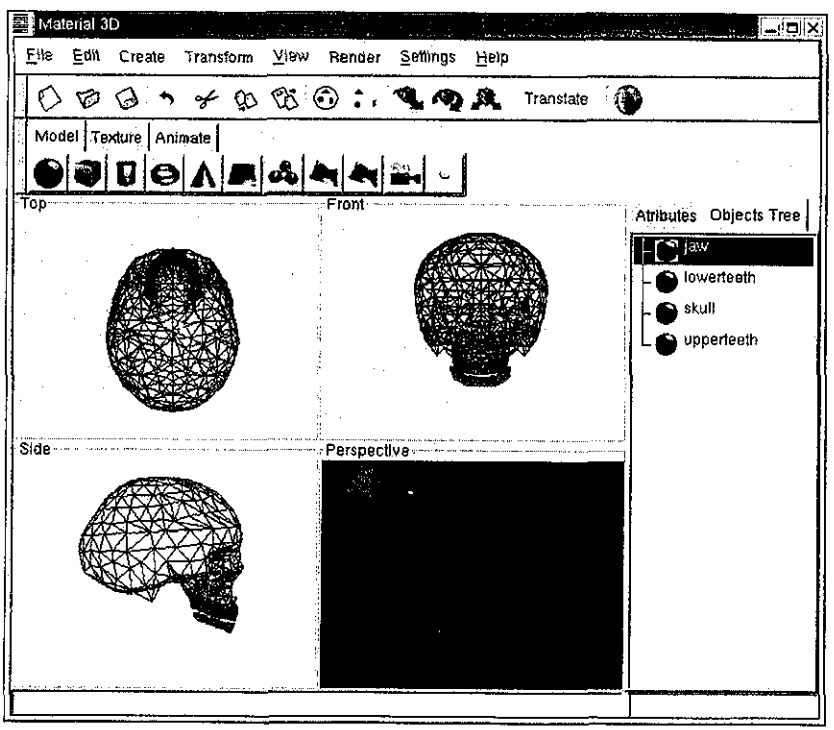


Figura 4.40 Aspecto final de la interfaz de usuario

Los diálogos son los mismos que utilizan los programas del escritorio Gnome de Linux, debido a que es la plataforma en que corre de forma nativa y se ha preferido respetar su aspecto y estilo.

En este capítulo se describió la construcción del sistema mínimo o base de "Material 3D". Es una versión funcional y extensible. En el capítulo 6 se muestran varios trabajos que demuestran el desempeño del sistema para diferentes tipos de trabajos.

De manera resumida se puede afirmar: las estructuras que almacenan los objetos y los principales objetos primitivos están terminados. Adicionalmente, es posible agregar atributos nuevos a los objetos, por medio de agregar variables a las estructuras y también se pueden agregar nuevas primitivas de manera muy sencilla.

Los aspecto que se refieren a los procesos de asignar textura y animar están en etapas tempranas de su implemetación y es probable que cambien. Sin embargo, el programa es operacional y es una base sobre la cual se puede construir un sistema verdaderamente profesional, útil y fácil de usar.

Capítulo 5 Diseño del sistema de módulos insertables

Uno de los objetivos de este trabajo de tesis es permitir el manejo de módulos insertables. Este objetivo nace del deseo de poder proporcionar una herramienta que sea extensible, de tal forma, que cualquier usuario o grupo de ellos, capaz de programar moderadamente bien, puedan desarrollar sus propios aditamentos.

Por otro lado, si llegase a existir un número elevado de aditamentos para nuestra aplicación, no sería conveniente que los usuarios tuviesen instalados todos. Es mucho mejor si el usuario puede instalar sólo aquellos módulos que requiera. De esta forma no se sobrecarga de información visual innecesaria.

La correcta elección de la tecnología para la elaboración de los módulos insertables es de suma importancia para fomentar el desarrollo por parte de la comunidad de usuarios, para que estos escriban sus propias funciones para el programa. Es muy importante que los "plug-ins" sean sencillos de programar, que puedan ser programados sin conocer mucho del programa "Material 3D" y que se puedan cargar / descargar de forma sencilla.

5.1 ¿Qué es un Plug-in?

Un "Plug-in" **[24]**, también conocido como módulo insertable o cartucho, es una pieza de software que se agrega a una aplicación para proporcionar nuevas funcionalidades. Típicamente es desarrollado por terceros.

Desde la etapa de diseño de la aplicación, se deben considerar las conexiones que pudiera tener la misma para permitir la integración de piezas o módulos adicionales de cualquier tipo.

5.2 Aplicaciones ampliables y no ampliables

Para poder entender el concepto de módulo insertable, es necesario ver cómo se diseñan diferentes tipos de aplicaciones, qué permiten, y qué características tienen, para lo cual se explican las aplicaciones ampliables y no ampliables.

Aplicaciones no ampliables

Las aplicaciones no ampliables son aquéllas que fueron diseñadas para mantener una estructura sin cambios. En realidad, sí es posible ampliarlas, pero sólo por los desarrolladores, y es necesario tener todo el código fuente y entenderlo bien.

Se puede distinguir entre aplicaciones con una estructuración deficiente (figura 5.1) (tienen muchas relaciones entre componentes, las cuales tienen dependencias entre sí, de manera poco clara) y las aplicaciones bien estructuradas (figura 5.2) (en las cuales son pocas las relaciones internas y son muy claras).

Figura 5.1 Aplicación monolítica mal estructurada

Figura 5.2 Aplicación monolítica bien estructurada

Aplicaciones ampliables

Las aplicaciones ampliables se parecen en principio a las aplicaciones no ampliables bien estructuradas, donde los elementos tienen pocas dependencias entre ellos e interactúan por medio de parámetros,

Básicamente las aplicaciones ampliables permiten agregar módulos por medio de una interfaz, que son los parámetros que permiten la comunicación con la pieza que se agrega.

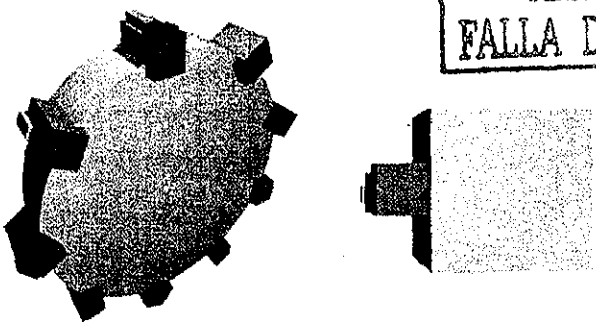


Figura 5.3 Aplicación ampliable

No es necesario conocer demasiado de la aplicación para poder programar estos módulos, sólo es necesario conocer la interfaz.

5.3 Características deseables en una arquitectura de Plug-ins

- Deben ser fáciles de agregar (enchufar)
- Deben ser fáciles de quitar (desenchufar)
- Debe poder agregarse más de uno simultáneamente.
- Deben ser consistentes con la apariencia de la aplicación.
- Entre menos se requiera conocer de la aplicación para poder programarlos es mejor

5.4 Arquitecturas de módulos insertables

Existen muchas formas diferentes de implementar plug-ins, algunas son más apropiadas sólo para cierto tipo de aplicaciones. Para poder tomar una decisión es importante conocer las ventajas y desventajas que ofrecen cada una de las diferentes alternativas.

5.4.1 Módulos en lenguaje nativo

La aplicación se parece mucho a una aplicación no ampliable bien estructurada, salvo que se definen un conjunto de variables y funciones (API) que pueden ser utilizados por el desarrollador del módulo, con lo que el conocimiento de los detalles internos de las funciones es menor.

Ejemplos de este tipo de arquitectura:
El kernel o núcleo de Linux permite la programación de módulos (por ejemplo para el manejo de tarjetas y otro tipo de periféricos) y al recompilarlo se obtiene un núcleo con el módulo ya integrado. El servidor de paginas web "Apache" también emplea módulos en código nativo.

Desventajas:

- A pesar de todo, todavía es mucho lo que se tiene que saber para realizar un módulo.
- Es prácticamente forzoso que el módulo sea programado en el mismo lenguaje que la aplicación.
- No son fáciles de agregar y quitar.

Este tipo de implementación no permite cargar / descargar módulos en tiempo de ejecución, por lo que queda descartado su uso en este proyecto al no cumplir con uno de los objetivos principales.

5.4.2 Módulos enlazados por librerías dinámicas

Emplean piezas de código compilado de tal forma que otros programas puedan hacer uso de las funciones contenidas dentro de éstas. Para ello, sólo es necesario conocer los parámetros que la función necesita y los valores que devuelve.

Son el mismo concepto que las bibliotecas ligadas dinámicamente (DLL´s) de Windows, y bibliotecas dinámicas de Linux. Se pueden programar los módulos en un lenguaje diferente al de la aplicación e incluso diferente al del resto de los módulos.

Ejemplos:

El kernel de Linux y Apache también soportan este tipo de módulos.
Las aplicaciones hechas para Microsoft Windows que soporten DLL's .

Ventajas:

- Se pueden programar en otros lenguajes
- El conocimiento requerido es pequeño (sólo la interfaz de las funciones y variables)

Desventajas:

- Se necesita saber exactamente qué biblioteca se requiere, qué servicio proporciona y qué parámetros requiere.

Los módulos enlazados dinámicamente tienen de hecho muy pocas desventajas y son una buena alternativa para muchas aplicaciones.

5.4.3 Módulos realizados por guiones (scripts)

Algunas aplicaciones incorporan lenguajes de scripts o guiones, que permiten la extensión de la funcionalidad, al programar nuevas herramientas e interpretarlas desde el propio programa.

Un ejemplo de uso está en el programa de dibujo 2D llamado Gimp. Dentro del software libre es relativamente común el uso de Guile[27], una biblioteca que implementa un intérprete de guiones basada en Scheme, el cual es un dialecto de Lisp.

Ventajas:

- Al ser un lenguaje de programación permite mucha flexibilidad y construcciones complejas, que de otra forma serian difíciles de crear.

Desventajas:

- Requiere que los desarrolladores aprendan un lenguaje de programación adicional.
- Es interpretado por lo tanto es lento.
- Es difícil de implementar dentro de la aplicación por el manejo del intérprete.

En general, a pesar de requerir aprender un nuevo lenguaje, proporciona unas enormes posibilidades y vale la pena considerar esta tecnología. Por otro lado siempre es posible crear un intérprete para algún subconjunto de algún lenguaje de alta difusión.

[27] http://www.gnu.org/software/guile/guile.html

5.4.4 Módulos externos

Realizan procesos directamente sobre los archivos de datos que la aplicación utiliza. Así, cuando se requiere que un módulo realice cierta operación, se le manda llamar y éste modifica directamente los datos. El programa puede entonces seguir trabajando con una versión ya transformada.

No existen muchos ejemplos de este tipo de implementaciones, ya que presenta muchos inconvenientes debido a que no permite trabajar de manera simultánea al programa y al módulo o a dos módulos, por lo cual no se utilizará dentro de "Material 3D".

5.4.5 Módulos conectados por CORBA (Object Request Broker)

En 1989 se formó el OMG (Object Management Group) conformado por más de 700 empresas **[24]**. Su función es adoptar y promover estándares para el desarrollo de aplicaciones en entornos distribuidos heterogéneos.

La arquitectura de administración de objetos pretende definir en un alto nivel de abstracción las reglas de la computación orientada a objetos distribuida. Se divide en Modelo de Objetos y Modelo de Referencia

En este modelo un Objeto es una entidad encapsulada, es decir, una caja negra de la cual sólo se conocen sus "enchufes" o "cabos" (interfaz). El programador no requiere conocer más. Aplicando un criterio de conocimiento mínimo el programador puede concentrarse en programar lo que necesita y no en detalles internos de los módulos que ya existen.

Modelo de Referencia

Está formado por (figura 5.4):

- Interfaces
- Interfaces de facilidades comunes
- Interfaces de dominio
- Interfaces de Aplicación

- Mediador de Peticiones de Objeto (ORB)
- Servicios de Objeto

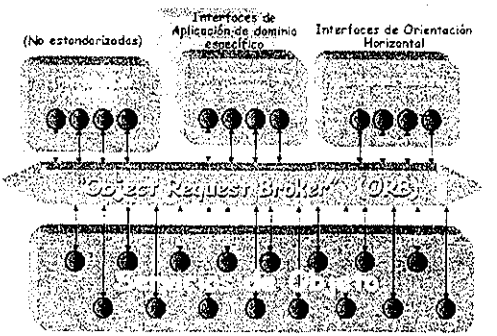


Figura 5.4 Modelo de referencia

Facilidades comunes

Son de orientación horizontal, por ejemplo:

- Interfaces de documentos distribuidos
- Facilidades comunes para agentes móviles
- Intercambio de datos
- Internacionalización

Interfaces de dominio

Son propuestas de estándares de interfaces de dominio especifico. Por ejemplo: interfaces para aplicaciones en medicina, telecomunicaciones, etc..

Interfaces de aplicación

Son específicas para la aplicación, por lo tanto no hay estándares. Son diseñadas por los programadores finales de la misma.

Mediador de peticiones de objeto

El ORB se encarga de aislar al programador de los detalles de las implementaciones de los objetos (lenguajes, estructura, etc.) y de los mecanismos de acceso, tales como protocolos de red, etc.

El ORB recibe la petición del cliente; se encarga de localizar el objeto; hacerlo disponible y, regresar un mensaje al cliente (figura 5.5).

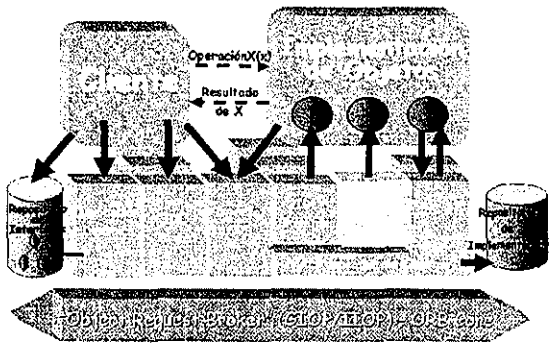


Figura 5.5 Mediador de objetos

Lenguaje de definición de interfaces

El lenguaje de definición de interfaces (IDL) es en estructura semejante a C++. Permite especificar el "contacto" o "cabo" de un objeto y ofrece como ventaja el separar la declaración de la interfaz de la propia implementación del objeto. Como consecuencia, permitir el uso de casi cualquier lenguaje de programación.

Interoperabilidad de objetos

Los IDL's son los encargados de enviar y recibir señales desde y hacia los objetos. Pueden ser IDL's "Stubs" e IDL "Skeletons" (figura 5.6).

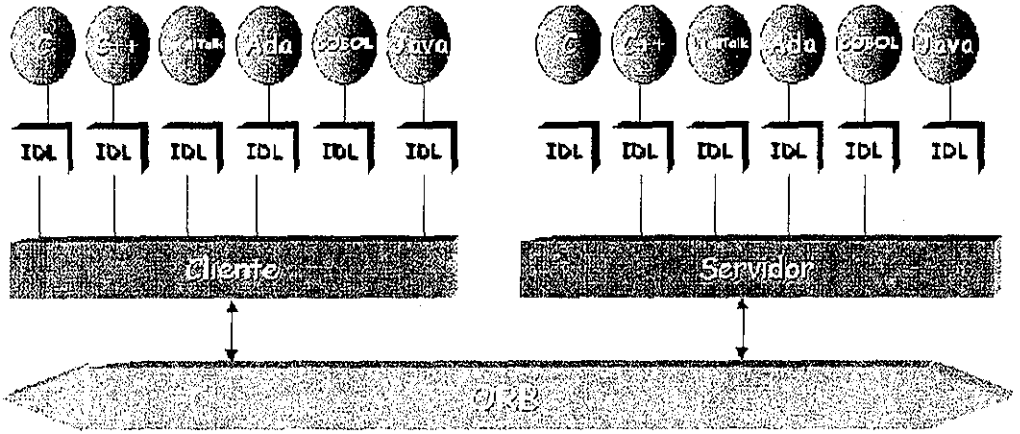


Figura 5.6 Interoperabilidad de objetos

IDL "Stubs" e IDL "Skeletons"

Un "stub" es el encargado de enviar las peticiones del cliente al servidor por medio del ORB, mientras en el lado del servidor está el "skeleton" que es el encargado de colaborar en la recepción de las peticiones. Cuando se regresa la respuesta el "skeleton" hace el envío al stub.

Este proceso es denominado estático, dado que tanto el cliente como el objeto de invocación conocen bien las interfaces IDL en cuestión

Repositorio de interfaces

El repositorio de interfaces (IR) es una base de datos distribuida modificable en tiempo de ejecución, la cual contiene información de las interfaces IDL. Puede usarse para conocer qué interfaces soporta un objeto registrado, así como para conocer las interfaces de objetos, de modo a poder usarlos de forma dinámica (figura 5.7).

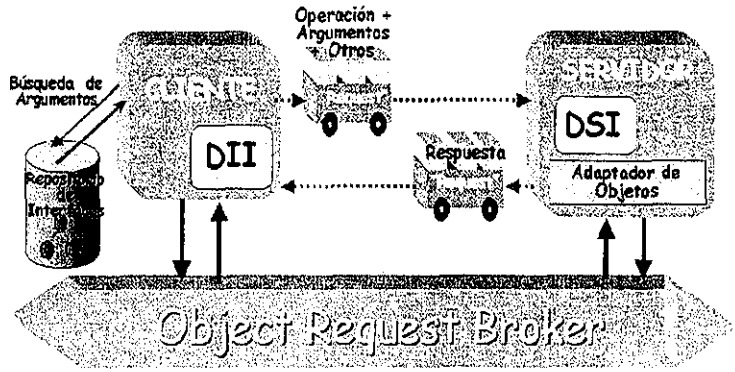


Figura 5.7 Interfaz de invocación dinámica

Adaptador de objetos

Dentro de sus principales actividades están la de activar, "instanciar", y crear referencias a los objetos, así como colaborar con el ORB cuando se realizan múltiples solicitudes a un objeto.

Repositorio de implementación

Es una base de datos generada en tiempo de ejecución y que brinda información relativa a:

- Qué clases soporta un servidor
- Qué objetos están "instanciados"
- Cuáles son identificadores
- Información de trazas y seguridad

Implementaciones de CORBA

Actualmente existen muchas versiones para los diferentes sistemas operativos, algunas son comerciales y algunas son libres, pero todas están basadas en el mismo estándar y pueden trabajar de forma colaborativa gracias a que utilizan los mismos protocolos.

Las distribuciones más extendidas son:

- OrbixWeb (para Java)
- NEO y JOE (de SUN)
- HP ORB (HP)
- SOM (IBM)
- VisiBroker (Visigenic)
- Orbit (Linux)

Como ejemplo del manejo de CORBA utilizando lenguaje "C" y Orbit, que es la implemetación del estándar en Linux, se tiene:

Se escribe el IDL. Como se puede observar, el lenguaje se parece mucho a C++. El archivo se puede crear con cualquier editor de texto, pero es importante que la extensión del archivo sea .idl.

```
/*** begin print_string.idl ***/

module appname {
    interface object {
        /* accept a string and a long. and return a long. */
        void print_string (in string some_string, in long someval, out long retval);
    };
};
```

Para convertir el archivo IDL a código en "C" se ejecuta el programa orbit-idl, que es parte de la implementacion de CORBA para Linux.

```
$ orbit-idl print_string.idl
```

Esto genera los siguientes archivos:
print_string.h
print_string-skels.c
print_string-common.c
print_string-stubs.c

Por último, para compilarlo se le pasan las banderas para que incluya las bibliotecas correspondientes.

```
gcc print_client.c -Wall -c -g `gtk-config --cflags`
gcc print_string-stubs.c -Wall -c -g `gtk-config --cflags`
gcc print_string-common.c -Wall -c -g `gtk-config --cflags`
```

Básicamente se invoca al compilador gcc para cada uno de los archivos, con la inclusión de una instrucción (`gtk-config --cflags`). gtk-config es un pequeño programa que lo único que hace es desplegar las banderas adecuadas para poder compilar programas que utilicen gtk.

El archivo client.c es el archivo que contiene el código que corresponde al cliente del ejemplo, mientras que los archivos print_string-stubs.c y print_string-common.c son los "cabos" que permiten unirlo con otros programas.

```
gcc print_string-stubs.o print_client.o print_string-common.o -o \
print_client `gtk-config --libs` -lIIOP -lORBit -lORBitutil
```

Dado que los archivos se compilaron de forma independiente, se obtuvieron tres archivos de código objeto (con extensión .o). Para ligarlos y crear un ejecutable se utilizó la instrucción anterior, donde se agregaron las banderas para ligarlo con las bibliotecas gtk (`gtk-config --libs`) y las de CORBA (-lIIOP -lORBit -lORBitutil)

Para compilar el programa que actuará como servidor, se utilizaron las siguientes llamadas al compilador, las cuales son idénticas a la parte del cliente.

```
gcc print_server.c -Wall -c -g `gtk-config --cflags`
gcc print_string-skels.c -Wall -c -g `gtk-config --cflags`
```

Para ligar los dos archivos de código objeto y las bibliotecas gtk y CORBA se utilizó la siguiente instrucción:

```
gcc print_string-skels.o print_server.o print_string-common.o -o \
print_server `gtk-config --libs` -lIIOP -lORBit -lORBitutil
```

CORBA es una poderosa herramienta que permite el uso de casi cualquier servicio de una aplicación en otra, en tiempo de ejecución, escritos en cualquier lenguaje y de hecho en cualquier plataforma, con la única condición de que ambas aplicaciones soporten el estándar.

De todos los posibles métodos de implementar módulos insertables, el más prometedor es CORBA y es el que se utilizará para desarrollar "plug-ins" para "Material 3D". Dadas las grandes posibilidades que brinda un lenguaje interpretado, se implementará un interprete, pero éste servirá para definir modelos y comportamientos de animación, más no para manejar módulos adicionales.

Capítulo 6 Conclusiones y trabajo futuro

6.1 Análisis del programa Material 3D

En este capítulo se muestran los resultados del uso del programa desarrollado durante la tesis, las conclusiones, los puntos mejorables y las líneas de acción para continuar su desarrollo.

6.1.1 Trabajos de tipo didáctico y comercial

Para probar si Material 3D podía trabajar con modelos de tipo comercial o didáctico se realizó un programa que gráfica funciones del tipo z = f(x, y) en tres dimensiones y guarda los resultados en formato nativo de material 3D. Los resultados son satisfactorios y se puede apreciar un ejemplo en la figura 6.1, donde se muestra la gráfica de la función z = Sen (x) * Cos (y).

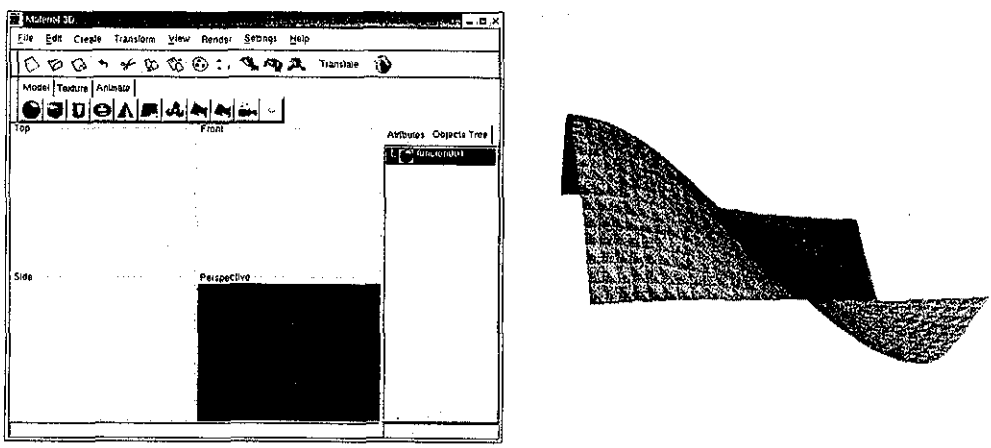


Figura 6.1 Ejemplo de la gráfica z = Sen X * Cos Y utilizando Material 3D

A partir del programa libre 3ds2pov, el cual es un convertidor de formatos, se creó una modificación que permitiera convertir archivos de 3D Studio a formato de Material 3D. Cabe mencionar que la mayoría de los programas de modelado y animación comercial utilizan el mismo método de representación de superficies, por medio de mallas, las

cuales son almacenadas como arreglos de vértices y polígonos, por lo que modificar el programa 3ds2pov para que exportara a Material 3D fue sencillo.

Una vez que los modelos estuvieron en formato de Material 3D, fue posible manipular cada uno de sus objetos internos, como se aprecia en la figura 6.2, donde se encuentra seleccionada la mandíbula y es posible manipularla. Del lado derecho de la misma imagen se encuentra la imagen generada por el programa.

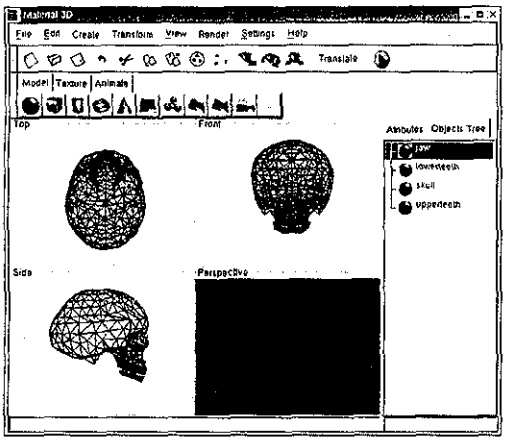


Figura 6.2 Ejemplo de manipulación de un modelo compuesto de varios objetos

Otro de los modelos importados de formato 3D Studio es el de la Venus que se muestra en la figura 6.3. Es un modelo sumamente detallado, compuesto por más de 67,000 polígonos. El programa maneja sin problemas esa cantidad de datos, debido a que asigna de forma dinámica la memoria.

Se pueden realizar transformaciones sobre los objetos, tales como transladarlos, rotarlos, escalarlos o agregar más objetos y componer una escena. Para ejemplificar la posibilidad de apreciar los modelos desde varios puntos de vista se generaron dos vistas (figura 6.3). En ellas se puede ver el nivel de detalle que se puede alcanzar con modelos basados en mallas cuando se está dispuesto a utilizar muchos recursos de almacenamiento. Este modelo en formato nativo ocupa poco más de 8 Mbytes.

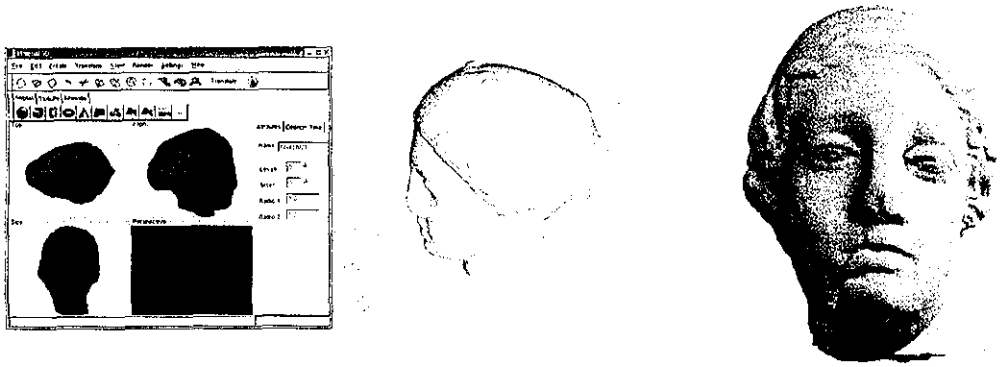


Figura 6.3 Generación de dos imágenes a partir de datos importados

6.1.2 Trabajos para aplicaciones de Ingeniería

Para hacer pruebas con aplicaciones de ingeniería, se utilizaron datos proporcionados por el Instituto de Ingeniería de la UNAM, por el Maestro en Ingeniería Roberto Magaña. Dichos datos son mallas de elemento finito, cuya estructura esta basada en "elementos", cada uno de los cuales es un volumen de forma idealmente cúbica, definido por ocho vértices o nodos.

Los algoritmos numéricos de análisis de elemento finito calculan los esfuerzos en cada uno de estos elementos. El método para convertir las mallas de elemento finito a mallas de Material 3D fue el siguiente:

Los datos de la malla de elemento finito son como los que se muestran:

Elemento y vértices que lo forman:

1 863 863 864 864 861 1 1 861

2 864 57 2 1 863 39 2 1

3 57 58 3 2 39 40 3 2

..... (en este caso hasta 1047)

Lista de vértices y sus coordenadas:

1 145.0000 -4.5000 110.0000

2 116.0000 -4.5000 110.0000

..... (En este caso hasta 1106)

Para convertir el "elemento" en polígonos se utilizó el esquema de conversión mostrado en la figura 6.4, donde se aprecian los índices de los vértices y a qué triángulos corresponderán.

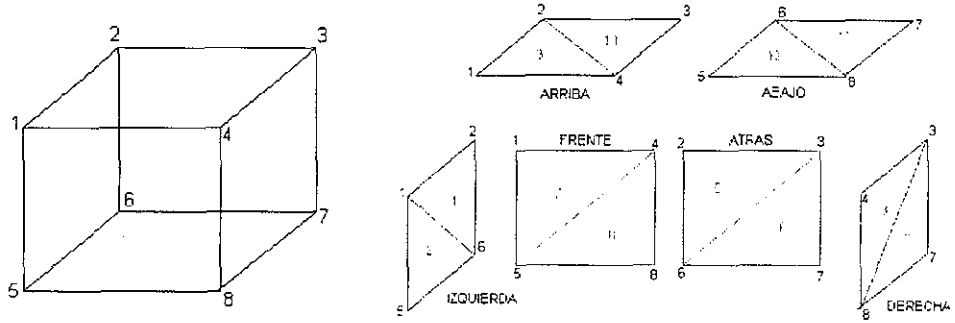


Figura 6.4 Esquema para convertir un elemento cúbico en triángulos.

Para representar los esfuerzos sobre cada elemento se tomaron los archivos de resultado, se calcularon valores máximos y mínimos y se asignó un color a cada valor empleando la escala que se muestra en la figura 6.5.

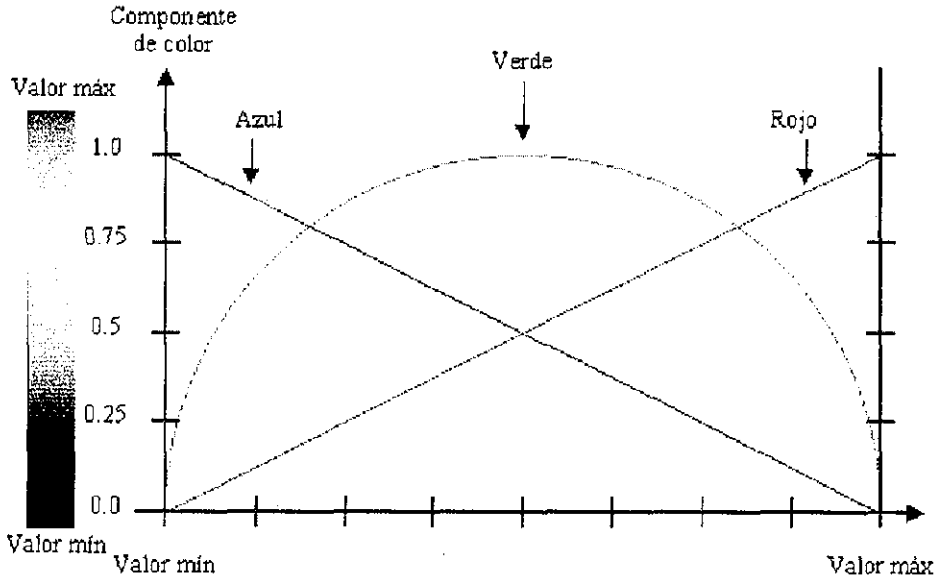
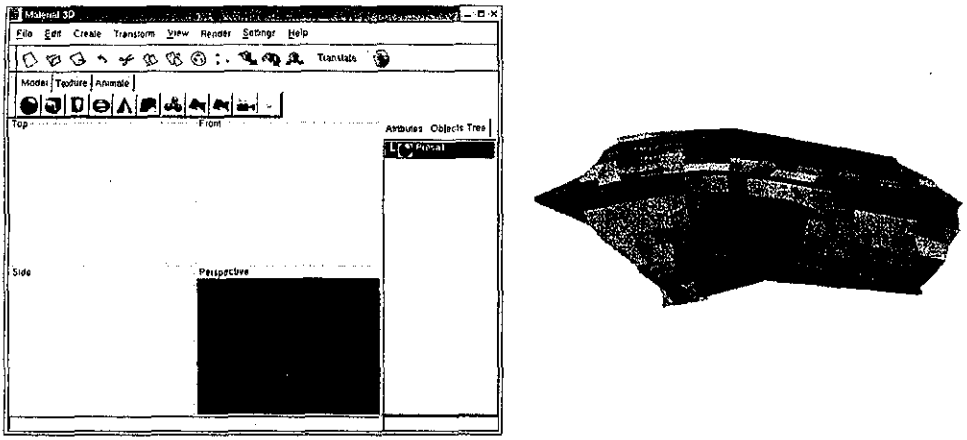


Figura 6.5 Escala para convertir en colores un valor dentro de un rango.

Es muy importante mencionar que anteriormente los resultados de dichas simulaciones eran enormes listados de texto. El usuario para poder visualizarlos utilizaba programas propios para exportar los datos a programas comerciales muy costosos.

El resultado final se muestra en la figura 6.6. Donde se muestra una presa de tierra, representada en tres dimensiones, primero en malla de alambre (izquierda) y posteriormente como un modelo donde el color de cada elemento representa un valor esfuerzo al que esta sometido (derecha).



6.6 Visualización y generación de la imagen del análisis de una presa

6.1.3 Trabajos para aplicaciones científicas

Para ejemplificar la utilidad del programa se escogió la reconstrucción en volumen (volume rendering) el cual es un tema de interés a varias disciplinas, desde la medicina hasta la astronomía.

Para crear un volumen a partir de imágenes individuales segmentadas, se modificó el algoritmo del cilindro (ver 4.2.3), que es la primitiva que por su estructura más se presta para ello.

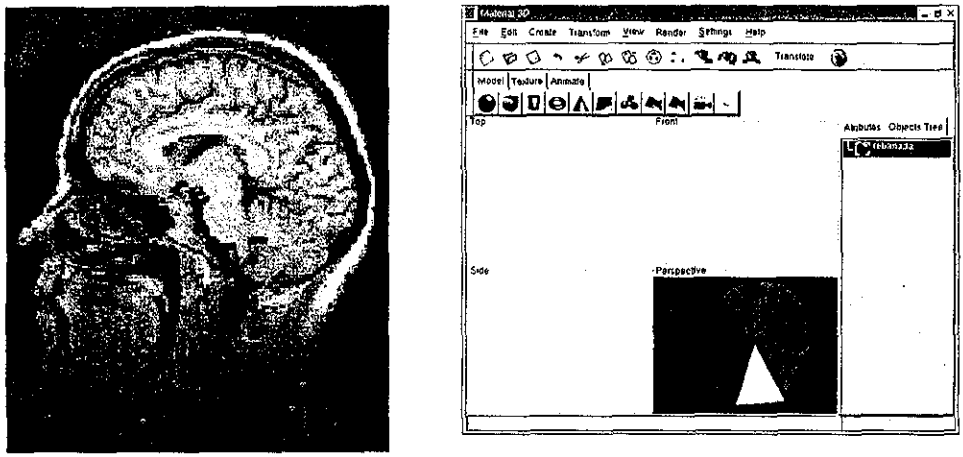


Figura 6.7 Imágenes bidimensionales segmentadas

Así, para una serie de imágenes segmentadas como las que se muestran en la figura 6.7, se obtiene la lista de las coordenadas de los puntos que las segmentan (Tabla 6.1) y se genera un volumen como el que se aprecia en la figura 6.8.

Punto	X	Y	Punto	X	Y	Punto	X	Y
1	132	254	**17**	212	232	**33**	32	166
2	126	17	**18**	204	261	**34**	12	161
3	146	16	**19**	197	278	**35**	16	151
4	168	17	**20**	94	286	**36**	24	142
5	184	22	**21**	80	265	**37**	34	129
6	201	29	**22**	67	253	**38**	34	119
7	216	38	**23**	45	252	**39**	37	104
8	227	50	**24**	27	249	**40**	38	89
9	237	65	**25**	23	235	**41**	42	74
10	243	82	**26**	25	225	**42**	49	62
11	24	99	**27**	20	213	**43**	56	51
12	253	117	**28**	30	203	**44**	64	61
13	253	140	**29**	25	199	**45**	80	32
14	250	161	**30**	19	195	**46**	98	25
15	235	186	**31**	23	182	**47**	113	19
16	223	206	**32**	30	172			

Tabla 6.1 Lista de vértices que segmentan una de las imágenes bidimensionales

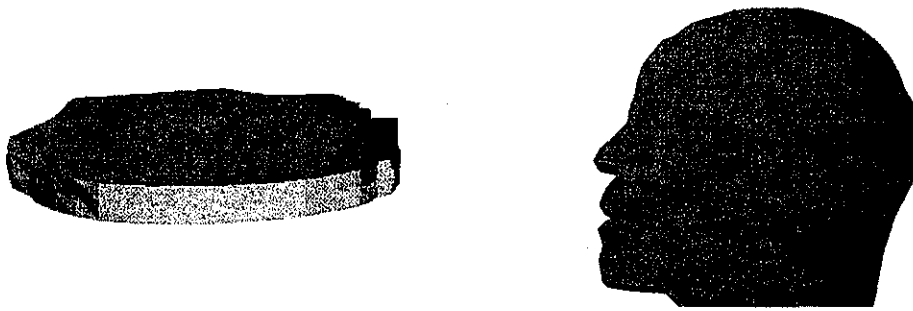


Figura 6.8 Reconstrucción en volumen de una pila de imágenes médicas

6.2 Trabajo futuro

Los objetivos que se plantearon para este proyecto se alcanzaron, sin embargo falta mucho por hacer para que Material 3D pueda considerarse completo. Con el fin de agregarle más funciones, se desarrolló un esquema de trabajo donde se integraron varios alumnos de la carrera de Ingeniería en Computación, de la Escuela Nacional de Estudios Profesionales (ENEP) "Aragón" de la UNAM.

Los alumnos de la ENEP están desarrollando este trabajo como proyectos de tesis en colaboración del Ingeniero Marcelo Pérez Medel, quien funge como asesor del trabajo.

Los proyectos están divididos de tal forma que pueden desarrollarse de manera independiente y después integrarse entre sí. Los trabajos

6.2.1 Editor de materiales

Todos los programas comerciales de modelado y animación 3D cuentan con editores de texturas y materiales. Los materiales permiten que los objetos adquieran una apariencia mucho más real. Para incrementar el realismo de los modelos será importante que entre los atributos de los materiales se incluyan masa, elasticidad y en general atributos que permitan manejar animación basada en física.

El proyecto del diseño e implementación de un editor de materiales se encuentra en proceso, desarrollado por Guadalupe Amezcua y Carlos de la Rosa.

6.2.2 Editor de trayectorias

Un editor de trayectorias permite definir la ruta que seguirá un objeto o un conjunto de objetos dentro de una escena. Poder realizar una animación requiere del movimiento de algunos elementos de la escena. Estos pueden ser objetos, o la cámara, la fuente de luz o el objetivo de la cámara.

El alumno Jesús Hernández se encuentra desarrollado un editor de trayectorias con las siguientes características:

- Permite definir trayectorias por medio de curvas bezier
- Puede leer las coordenadas de las trayectorias de los objetos de un archivo de texto, facilitando que los cálculos científicos o de ingeniería puedan ser visualizados en forma de animación.
- Permite definir funciones para calcular las coordenadas de las rutas. Definir rutas por medio de ecuaciones permitirá realizar animaciones basadas en física. De este modo, definir que un objeto (una pelota, por ejemplo) va a caer siguiendo un camino definido por la ecuación de caída libre.

6.2.3 Módulo de metamorfosis en 3D

Este módulo tiene aplicaciones principalmente en el ámbito de las animaciones comerciales y permite la transformación tridimensional de un objeto en otro pasando por todos los estados intermedios. Pudiese tener algunas aplicaciones didácticas.

Felipe de Jesús Gutiérrez ha desarrollado esté módulo y se encuentra en proceso de depuración y pruebas.

6.2.4 Manejo de sistemas de partículas

En muchas aplicaciones se requieren sistemas de partículas, como por ejemplo, para simular el humo de un cigarro o el movimiento de partículas en el espacio.

Este proyecto no ha sido asignado a ninguna persona.

6.3 Conclusiones

- El Diseño Centrado en el Usuario permite crear sistemas que sean utilizables además de útiles, esto es: si se desarrolla un sistema capaz de realizar la tarea para la cual fue diseñado, entonces es útil. Sin embargo, es común que muchos sistemas útiles no sean utilizables porque el usuario no encuentra su manejo sencillo.

- MAD* es un formalismo muy útil para modelar tareas, a pesar de ser una herramienta muy poco conocida. Tiene un enorme potencial, ya que describe las tareas de tal manera que con facilidad podrían utilizarse sistemas automáticos capaz de procesar sus atributos, los cuales son más completos que los utilizados en casos de uso del modelo UML.

- El análisis de las posibles tecnologías y herramientas puede proporcionarnos una garantía razonable de continuidad. Por ejemplo, si las bibliotecas en las que se apoya un sistema fuesen descontinuadas y sustituidas por otras, esto involucraría una nueva inversión de tiempo.

- Entre los juegos de herramientas para interfaces gráficas GTK+ y Qt, la elección fue muy difícil, ya que las dos tienen casi las mismas ventajas y el principal problema de Qt (su licencia) fue arreglado. Sin embargo aún con los últimos cambios, GTK+ sigue siendo una mejor alternativa, debido sobretodo a la facilidad de ser llevada a cualquier plataforma con menos esfuerzo que Qt.

- Los usuarios no consideran excesivos los requerimientos de los principales programas de modelado y animación comerciales.

- El sistema operativo que prefieren como entorno de trabajo es Windows, pero también desearían tener programas de modelado y animación en Linux.

- Tener las mismas aplicaciones en diferentes plataformas es algo muy apreciado por los usuarios, porque permite que personas con preferencias diferentes para su entorno de trabajo puedan realizar proyectos en conjunto utilizando el mismo programa y archivos de trabajo.

- Los usuarios de las áreas científica y de ingeniería compran productos comerciales para visualizar sus cálculos, pero descubren que no resuelven la mayoría de sus necesidades. Incluso, los programas específicos para estas áreas son muy limitados en cuanto a lo que se puede hacer con ellos.

- Las principales quejas de los usuarios son referentes a: demasiado tiempo para generar la previsualización de las escenas y las dificultades para manejar texturas.

- La principal dificultad durante el modelado es el manejo de la tercera dimensión sobre una ventana de dos dimensiones (el monitor). Es difícil para los usuarios rotar un objeto con respecto a un eje. La mayoría de los usuarios hacían ensayos del tipo "prueba y error" hasta lograr el giro en el sentido que desean.

- El proceso de modelar y animar un objeto es típicamente en el orden:
  - Modelar
  - Colocar textura
  - Animar

- La representación de objetos tridimensionales por medio de mallas de polígonos permite la importación de programas comerciales y de ingeniería de forma mucho más sencilla de lo que sería si los objetos fuesen representado por medio de ecuaciones.

- Al utilizar un formato nativo utilizando XML, se facilita la creación de programas que graben en nuestro formato.

- Los módulos insertables permiten personalizar las capacidades y funcionalidades de los programas.

- Tecnologías de elementos distribuidas, tal como CORBA, tienen un enorme potencial, en un mundo donde la infraestructura de comunicaciones es enorme. Vale la pena aprovecharla en aplicaciones distribuidas.

- "Material 3D" es útil para las aplicaciones que fueron planteadas al inicio del proyecto, tanto en aplicaciones didácticas, comerciales, científicas y de ingeniería.

- Para que los usuarios puedan utilizarlo, hace falta documentación clara y que no existe en este momento.

Anexo A

Protocolos y cuestionarios utilizados en el análisis del usuario y el modelado de la tarea del usuario

A1 Protocolo de usuario novato solo

Bueno Días
Gracias por venir a colaborar con nosotros

De lo que se trata es que pruebes un programa de diseño en 3D, y la prueba consiste en modelar estos objetos (Mostrar dibujos)

Es importante que mientras vayas utilizando el programa vayas mencionando en voz alta qué es lo que estas haciendo y pensando. Por ejemplo cuando estás en una pantalla trata de predecir para qué sirve cada botón y cada herramienta. Si cuando lo presionas hace algo diferente a lo que habías imaginado, por favor menciónalo también.

Si tienes dudas, menciónalas, pero no te voy a poder ayudar porque yo tampoco conozco el funcionamiento del programa.

Por favor avísame cuando termines.

(Esperar a que termine)

Gracias. Por ultimo te pido que por favor llenes este cuestionario con tu opinión sobre el programa.

A2 Cuestionario para usuario novato solo

1.- ¿Sabes usar este programa?
2.- ¿Sabes usar algún programa parecido?
3.- ¿Te pareció fácil de usar?
4.- ¿Qué tareas o acciones te parecieron fáciles?
5.- ¿Qué acciones se te dificultaron?
6.- ¿Qué te gustaría cambiar del programa?

A3 Protocolo (Versión de usuario avanzado con novato)

Hola, buenos días.

Gracias por colaborar con nosotros. Te hemos pedido que vinieras para que nos ayudes a probar la facilidad de uso de unos programas de diseño en 3D.

Tu estarás sentado aquí frente a la computadora y vas a ayudar a otra persona a realizar un modelo en (Maya o 3D-Max según el caso). La persona que va a realizar el modelo con tu ayuda estará detrás del cristal y puede ver la pantalla de la computadora en un monitor. No conoce el programa y va a hablar contigo para indicarte qué acciones debes ir realizando. No le propongas que herramientas usar, si te pide hacer algo para lo cual no tengas una herramienta adecuada díselo y espera que encuentre otra forma de solucionarlo. Si no se le ocurre nada en un par de minutos entonces si propones alguna solución.

Cuando terminen les pediremos que contesten un cuestionario muy sencillo.

(Esperar a que terminen)

A4 Protocolo (Versión de usuario experto apoyando a usuario novato)

Hola, buenos días.

Gracias por colaborar con nosotros. Te hemos pedido que vinieras para que nos ayudes a probar la facilidad de uso de unos programas de diseño en 3D.

Tu estarás sentado aquí frente a este monitor y vas a realizar un modelo en (Maya o 3d-max). La persona que va a realizar el modelo con tu guía estará detrás del cristal y por medio del micrófono le vas a ir indicando qué acciones tomar para realizar el modelo. Esta persona sabe usar el programa, así que puedes indicarle que hacer, pero no puedes pedirle que te explique, ya que básicamente ejecutará tus instrucciones. Si no puedes avanzar, pídele alguna sugerencia.

Cuando terminen les pediremos que contesten un cuestionario muy sencillo.
(Esperar a que terminen)

A5 Cuestionario (Versión de usuario novato con apoyo de experto)

Por favor contesta a las siguientes preguntas:
1.- ¿Qué tareas te parecieron más fáciles de realizar? y ¿por qué?
2.- ¿Qué tareas te parecieron más difíciles de realizar? y ¿por qué?
3.- ¿Qué herramientas adicionales te gustaría que tuviera el programa?
4.-¿Cambiarías el comportamiento de alguna de las herramientas? ¿De cuál y de qué forma?
5.- Escribe un comentario general del programa

A6 Protocolo (Versión de usuario avanzado solo)

Gracias por tu paciencia. Ahora te pedimos que hagas este otro modelo, pero ahora tu solo. Es importante que vayas diciendo en voz alta lo que vas haciendo.
Cuando terminen les pediremos que contesten un cuestionario muy sencillo.
(Esperar a que termine)

A7 Cuestionario para los usuarios de orientación comercial

1- ¿Cómo fue que te iniciaste en el modelado 3D?
2- ¿Porqué te gusta el modelado en 3D?
3- ¿Cuántos años llevas practicando el modelado 3D?
4- ¿Cómo consideras tu experiencia en el modelado 3D?
5- ¿Qué programas para el modelado en 3d conoces?
6- ¿Cuál de ellos es de tu predilección y porque?
7- ¿Cuáles son los requerimientos de Hardware para el programa de tu preferencia?
8- ¿Cómo consideras estos requerimientos?
9- Bajo que sistemas operativos operan los paquetes que usas
10- Bajo que sistema operativo te gustaría que tu programa corriera
11- ¿Qué opinas de los paquetes de licencia y distribución libre?
12- ¿Hay suficiente información para el aprendizaje de los programas que usas?
13- Has trabajado con algún programa de los siguientes y cómo calificas su interfaz:
- ❖ Maya
- ❖ 3D Studio
- ❖ 3D Studio MAX
- ❖ Softimage

14- ¿Que herramientas quisieras que tuviera un modelador en 3D?
15- ¿Desearías que el programa que usas contuviera un editor de trayectorias para facilitar la animación?
16- ¿Desearías que el programa que usas contuviera un editor de texturas?

Anexo B

Traducción Española de la Licencia Pública GNU

Esta es la conocida GNU Public License (GPL), versión 2 (de junio de 1.991), que cubre la mayor parte del software de la Free Software Foundation, y muchos más programas.

Los autores de esta traducción son:

- Jesús González Barahona
- Pedro de las Heras Quirós

NOTA IMPORTANTE:

Esta es una traducción no oficial al español de la GNU General Public License. No ha sido publicada por la Free Software Foundation, y no establece legalmente las condiciones de distribución para el software que usa la GNU GPL. Estas condiciones se establecen solamente por el texto original, en inglés, de la GNU GPL. Sin embargo, esperamos que esta traducción ayude a los hispanohablantes a entender mejor la GNU GPL.

IMPORTANT NOTICE:

This is an unofficial translation of the GNU General Public License into Spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL--only the original English text of the GNU GPL does that. However, we hope that this translation will help Spanish speakers understand the GNU GPL better.

Preámbulo

Las licencias que cubren la mayor parte del software están diseñadas para quitarle a usted la libertad de compartirlo y modificarlo. Por el contrario, la Licencia Pública General de GNU pretende garantizarle la libertad de compartir y modificar software libre, para asegurar que el software es libre para todos sus usuarios. Esta Licencia Pública General se aplica a la mayor parte del software del la Free Software Foundation y a cualquier otro programa si sus autores se comprometen a utilizarla. (Existe otro software de la Free Software Foundation que está cubierto por la Licencia Pública General de GNU para Bibliotecas). Si quiere, también puede aplicarla a sus propios programas.

Cuando hablamos de software libre, estamos refiriéndonos a libertad, no a precio. Nuestras Licencias Públicas Generales están diseñadas para asegurarnos de que tenga la libertad de distribuir copias de software libre (y cobrar por ese servicio si quiere), de que reciba el código fuente o que pueda conseguirlo si lo quiere, de que pueda modificar el software o usar fragmentos de él en nuevos programas libres, y de que sepa que puede hacer todas estas cosas.

Para proteger sus derechos necesitamos algunas restricciones que prohiban a cualquiera negarle a usted estos derechos o pedirle que renuncie a ellos. Estas restricciones se traducen en ciertas obligaciones que le afectan si distribuye copias del software, o si lo modifica.

Por ejemplo, si distribuye copias de uno de estos programas, sea gratuitamente, o a cambio de una contraprestación, debe dar a los receptores todos los derechos que tiene. Debe asegurarse de que ellos también reciben, o pueden conseguir, el código fuente. Y debe mostrarles estas condiciones de forma que conozcan sus derechos.

Protegemos sus derechos con la combinación de dos medidas:

Ponemos el software bajo copyright

1. le ofrecemos esta licencia, que le da permiso legal para copiar, distribuir y/o modificar el software.

También, para la protección de cada autor y la nuestra propia, queremos asegurarnos de que todo el mundo comprende que no se proporciona ninguna garantía para este software libre. Si el software se modifica por cualquiera y éste a su vez lo distribuye, queremos que sus receptores sepan que lo que tienen no es el original, de forma que cualquier problema introducido por otros no afecte a la reputación de los autores originales.

Por último, cualquier programa libre está constantemente amenazado por patentes sobre el software. Queremos evitar el peligro de que los redistribuidores de un programa libre obtengan patentes por su cuenta, convirtiendo de facto el programa en propietario. Para evitar esto, hemos dejado claro que cualquier patente debe ser pedida para el uso libre de cualquiera, o no ser pedida.

Los términos exactos y las condiciones para la copia, distribución y modificación se exponen a continuación.

Términos y condiciones para la copia, distribución y modificación

1. Esta Licencia se aplica a cualquier programa u otro tipo de trabajo que contenga una nota colocada por el tenedor del copyright diciendo que puede ser distribuido bajo los términos de esta Licencia Pública General. En adelante, «Programa» se referirá a cualquier programa o trabajo que cumpla esa condición y «trabajo basado en el Programa» se referirá bien al Programa o a cualquier trabajo derivado de él según la ley de copyright. Esto es, un trabajo que contenga el programa o una porción de él, bien en forma literal o con modificaciones y/o traducido en otro lenguaje. Por lo tanto, la traducción está incluida sin limitaciones en el término «modificación». Cada concesionario (licenciatario) será denominado «usted».

   Cualquier otra actividad que no sea la copia, distribución o modificación no está cubierta por esta Licencia, está fuera de su ámbito. El acto de ejecutar el Programa no está restringido, y los resultados del Programa están cubiertos únicamente si sus contenidos constituyen un trabajo basado en el Programa, independientemente de haberlo producido mediante la ejecución del programa. El que esto se cumpla, depende de lo que haga el programa.

2. Usted puede copiar y distribuir copias literales del código fuente del Programa, según lo has recibido, en cualquier medio, supuesto que de forma adecuada y bien visible publique en cada copia un anuncio de copyright adecuado y un repudio de garantía, mantenga intactos todos los anuncios que se refieran a esta Licencia y a la ausencia de garantía, y proporcione a cualquier otro receptor del programa una copia de esta Licencia junto con el Programa. Puede cobrar un precio por el acto físico de transferir una copia, y puede, según su libre albedrío, ofrecer garantía a cambio de unos honorarios.

3. Puede modificar su copia o copias del Programa o de cualquier porción de él, formando de esta manera un trabajo basado en el Programa, y copiar y distribuir esa modificación o trabajo bajo los términos del apartado 1, antedicho, supuesto que además cumpla las siguientes condiciones:

   1. Debe hacer que los ficheros modificados lleven anuncios prominentes indicando que los ha cambiado y la fecha de cualquier cambio.
   2. Debe hacer que cualquier trabajo que distribuya o publique y que en todo o en parte contenga o sea derivado del Programa o de cualquier parte de él sea licenciada como un todo, sin carga alguna, a todas las terceras partes y bajo los términos de esta Licencia.
   3. Si el programa modificado lee normalmente órdenes interactivamente cuando es ejecutado, debe hacer que, cuando comience su ejecución para ese uso interactivo de la forma más habitual, muestre o escriba un mensaje que incluya un anuncio de copyright y un anuncio de que no se ofrece ninguna garantía (o por el contrario que sí se ofrece garantía) y que los usuarios pueden redistribuir el programa bajo estas condiciones, e indicando al usuario cómo ver una copia de esta licencia. (Excepción: si el propio programa es interactivo pero normalmente no muestra ese anuncio,

no se requiere que su trabajo basado en el Programa muestre ningún anuncio).

Estos requisitos se aplican al trabajo modificado como un todo. Si partes identificables de ese trabajo no son derivadas del Programa, y pueden, razonablemente, ser consideradas trabajos independientes y separados por ellos mismos, entonces esta Licencia y sus términos no se aplican a esas partes cuando sean distribuidas como trabajos separados. Pero cuando distribuya esas mismas secciones como partes de un todo que es un trabajo basado en el Programa, la distribución del todo debe ser según los términos de esta licencia, cuyos permisos para otros licenciatarios se extienden al todo completo, y por lo tanto a todas y cada una de sus partes, con independencia de quién la escribió.

Por lo tanto, no es la intención de este apartado reclamar derechos o desafiar sus derechos sobre trabajos escritos totalmente por usted mismo. El intento es ejercer el derecho a controlar la distribución de trabajos derivados o colectivos basados en el Programa.

Además, el simple hecho de reunir un trabajo no basado en el Programa con el Programa (o con un trabajo basado en el Programa) en un volumen de almacenamiento o en un medio de distribución no hace que dicho trabajo entre dentro del ámbito cubierto por esta Licencia.

4. Puede copiar y distribuir el Programa (o un trabajo basado en él, según se especifica en el apartado 2, como código objeto o en formato ejecutable según los términos de los apartados 1 y 2, supuesto que además cumpla una de las siguientes condiciones:
    1. Acompañarlo con el código fuente completo correspondiente, en formato electrónico, que debe ser distribuido según se especifica en los apartados 1 y 2 de esta Licencia en un medio habitualmente utilizado para el intercambio de programas, o
    2. Acompañarlo con una oferta por escrito, válida durante al menos tres años, de proporcionar a cualquier tercera parte una copia completa en formato electrónico del código fuente correspondiente, a un coste no mayor que el de realizar físicamente la distribución del fuente, que será distribuido bajo las condiciones descritas en los apartados 1 y 2 anteriores, en un medio habitualmente utilizado para el intercambio de programas, o
    3. Acompañarlo con la información que recibiste ofreciendo distribuir el código fuente correspondiente. (Esta opción se permite sólo para distribución no comercial y sólo si usted recibió el programa como código objeto o en formato ejecutable con tal oferta, de acuerdo con el apartado b anterior).

Por código fuente de un trabajo se entiende la forma preferida del trabajo cuando se le hacen modificaciones. Para un trabajo ejecutable, se entiende por código fuente completo todo el código fuente para todos los módulos que contiene, más cualquier fichero asociado de definición de interfaces, más los guiones utilizados para controlar la compilación e instalación del ejecutable. Como excepción especial el código fuente distribuido no necesita incluir nada que sea distribuido normalmente (bien como fuente, bien en forma binaria) con los componentes principales (compilador, kernel y similares) del sistema operativo en el cual funciona el ejecutable, a no ser que el propio componente acompañe al ejecutable. Si la distribución del ejecutable o del código objeto se hace mediante la oferta acceso para copiarlo de un cierto lugar, entonces se considera la oferta de acceso para copiar el código fuente del mismo lugar como distribución del código fuente, incluso aunque terceras partes no estén forzadas a copiar el fuente junto con el código objeto. No puede copiar, modificar, sublicenciar o distribuir el Programa excepto como prevé expresamente esta Licencia. Cualquier intento de copiar, modificar sublicenciar o distribuir el Programa de otra forma es inválida, y hará que cesen automáticamente los derechos que te proporciona esta Licencia. En cualquier caso, las partes que hayan recibido copias o derechos de usted bajo esta Licencia no cesarán en sus derechos mientras esas partes continúen cumpliéndola.

5. No está obligado a aceptar esta licencia, ya que no la ha firmado. Sin embargo, no hay hada más que le proporcione permiso para modificar o distribuir el Programa o sus trabajos derivados. Estas acciones están prohibidas por la ley si no acepta esta Licencia. Por lo tanto, si modifica o distribuye el Programa (o cualquier trabajo basado en el Programa), está indicando que acepta esta Licencia para poder hacerlo, y todos sus términos y condiciones para copiar, distribuir o modificar el Programa o trabajos basados en él.

6. Cada vez que redistribuya el Programa (o cualquier trabajo basado en el Programa), el receptor recibe automáticamente una licencia del licenciatario original para copiar, distribuir o modificar el Programa, de forma sujeta a estos términos y condiciones. No puede imponer al receptor ninguna restricción más sobre el ejercicio de los derechos aquí garantizados. No es usted responsable de hacer cumplir esta licencia por terceras partes.

7. Si como consecuencia de una resolución judicial o de una alegación de infracción de patente o por cualquier otra razón (no limitada a asuntos relacionados con patentes) se le imponen condiciones (ya sea por mandato judicial, por acuerdo o por cualquier otra causa) que contradigan las condiciones de esta Licencia, ello no le exime de cumplir las condiciones de esta Licencia. Si no puede realizar distribuciones de forma que se satisfagan simultáneamente sus obligaciones bajo esta licencia y cualquier otra obligación pertinente entonces, como consecuencia, no puede distribuir el Programa de ninguna forma. Por ejemplo, si una patente no permite la redistribución libre de derechos de autor del Programa por parte de todos aquellos que reciban copias directa o indirectamente a través de usted, entonces la única forma en que podría satisfacer tanto esa condición como esta Licencia sería evitar completamente la distribución del Programa.

Si cualquier porción de este apartado se considera inválida o imposible de cumplir bajo cualquier circunstancia particular ha de cumplirse el resto y la sección por entero ha de cumplirse en cualquier otra circunstancia. No es el propósito de este apartado inducirle a infringir ninguna reivindicación de patente ni de ningún otro derecho deç propiedad o impugnar la validez de ninguna de dichas reivindicaciones. Este apartado tiene el único propósito de proteger la integridad del sistema de distribución de software libre, que se realiza mediante prácticas de licencia pública. Mucha gente ha hecho contribuciones generosas a la gran variedad de software distribuido mediante ese sistema con la confianza de que el sistema se aplicará consistentemente. Será el autor/donante quien decida si quiere distribuir software mediante cualquier otro sistema y una licencia no puede imponer esa elección.

Este apartado pretende dejar completamente claro lo que se cree que es una consecuencia del resto de esta Licencia.

8. Si la distribución y/o uso de el Programa está restringida en ciertos países, bien por patentes o por interfaces bajo copyright, el tenedor del copyright que coloca este Programa bajo esta Licencia puede añadir una limitación explícita de distribución geográfica excluyendo esos países, de forma que la distribución se permita sólo en o entre los países no excluidos de esta manera. En ese caso, esta Licencia incorporará la limitación como si estuviese escrita en el cuerpo de esta Licencia.

9. La Free Software Foundation puede publicar versiones revisadas y/o nuevas de la Licencia Pública General de tiempo en tiempo. Dichas nuevas versiones serán similares en espíritu a la presente versión, pero pueden ser diferentes en detalles para considerar nuevos problemas o situaciones.

   Cada versión recibe un número de versión que la distingue de otras. Si el Programa especifica un número de versión de esta Licencia que se refiere a ella y a «cualquier versión posterior», tienes la opción de seguir los términos y condiciones, bien de esa versión, bien de cualquier versión posterior publicada por la Free Software Foundation. Si el Programa no especifica un número de versión de esta Licencia, puedes escoger cualquier versión publicada por la Free Software Foundation.

10. Si quiere incorporar partes del Programa en otros programas libres cuyas condiciones de distribución son diferentes, escribe al autor para pedirle permiso. Si el software tiene copyright de la Free Software Foundation, escribe a la Free Software Foundation: algunas veces hacemos excepciones en estos casos. Nuestra decisión estará guiada por el doble objetivo de de preservar la libertad de todos los derivados de nuestro software libre y promover el que se comparta y reutilice el software en general.

AUSENCIA DE GARANTÍA

12. Como el programa se licencia libre de cargas, no se ofrece ninguna garantía sobre el programa, en todas la extensión permitida por la legislación aplicable. Excepto cuando se indique de otra forma por escrito, los tenedores del copyright y/u otras partes proporcionan el programa «tal cual», sin garantía de ninguna clase, bien expresa o implícita, con inclusión, pero sin limitación a las garantías mercantiles implícitas o a la conveniencia para un propósito particular. Cualquier riesgo referente a la calidad y prestaciones del programa es asumido por usted. Si se probase que el Programa es defectuoso, asume el coste de cualquier servicio, reparación o corrección.

13. En ningún caso, salvo que lo requiera la legislación aplicable o haya sido acordado por escrito, ningún tenedor del copyright ni ninguna otra parte que modifique y/o redistribuya el Programa según se permite en esta Licencia será responsable ante usted por daños, incluyendo cualquier daño general, especial, incidental o resultante producido por el uso o la imposibilidad de uso del Programa (con inclusión, pero sin limitación a la pérdida de datos o a la generación incorrecta de datos o a pérdidas sufridas por usted o por terceras partes o a un fallo del Programa al funcionar en combinación con cualquier otro programa), incluso si dicho tenedor u otra parte ha sido advertido de la posibilidad de dichos daños.

FIN DE TÉRMINOS Y CONDICIONES

Apéndice: Cómo aplicar estos términos a sus nuevos programas.

para el público en general, la mejor forma de conseguirlo es convirtiéndolo en software libre que cualquiera pueda redistribuir y cambiar bajo estos términos.

Para hacerlo, añada los siguientes anuncios al programa. Lo más seguro es añadirlos al principio de cada fichero fuente para transmitir lo más efectivamente posible la ausencia de garantía. Además cada fichero debería tener al menos la línea de «copyright» y un indicador a dónde puede encontrarse el anuncio completo.

<una línea para indicar el nombre del programa y una rápida idea de qué hace.>

Copyright (C) 19aa <nombre del autor Este programa es software libre. Puede redistribuirlo y/o modificarlo bajo los términos de la Licencia Pública General de GNU según es publicada por la Free Software Foundation, bien de la versión 2 de dicha Licencia o bien (según su elección) de cualquier versión posterior.

Este programa se distribuye con la esperanza de que sea útil, pero SIN NINGUNA GARANTÍA, incluso sin la garantía MERCANTIL implícita o sin garantizar la CONVENIENCIA PARA UN PROPÓSITO PARTICULAR. Véase la Licencia Pública General de GNU para más detalles.

Debería haber recibido una copia de la Licencia Pública General junto con este programa. Si no ha sido así, escriba a la Free Software Foundation, Inc., en 675 Mass Ave, Cambridge, MA 02139, EEUU.

Añada también información sobre cómo contactar con usted mediante correo electrónico y postal.

Si el programa es interactivo, haga que muestre un pequeño anuncio como el siguiente, cuando comienza a funcionar en modo interactivo:

Gnomovision versión 69, Copyright (C) 19aa nombre del autor

Gnomovision no ofrece ABSOLUTAMENTE NINGUNA GARANTÍA. Para más detalles escriba «show w».

Los comandos hipotéticos «show w» y «show c» deberían mostrar las partes adecuadas de la Licencia Pública General. Por supuesto, los comandos que use pueden llamarse de cualquier otra manera. Podrían incluso ser pulsaciones del ratón o elementos de un menú (lo que sea apropiado para su programa).

También deberías conseguir que su empleador (si trabaja como programador) o tu Universidad (si es el caso) firme un «renuncia de copyright» para el programa, si es necesario. A continuación se ofrece un ejemplo, altere los nombres según sea conveniente:

> Yoyodyne, Inc. mediante este documento renuncia a cualquier interés de derechos de copyright con respecto al programa Gnomovision (que hace pasadas a compiladores) escrito por Pepe Programador.
>
> <firma de Pepito Grillo>, 20 de diciembre de 1996
>
> Pepito Grillo, Presidente de Asuntillos Varios.

Esta Licencia Pública General no permite que incluya sus programas en programas propietarios. Si su programa es una biblioteca de subrutinas, puede considerar más útil el permitir el enlazado de aplicaciones propietarias con la biblioteca. Si este es el caso, use la Licencia Pública General de GNU para Bibliotecas en lugar de esta Licencia.

Anexo C

Criterios ergonómicos para la evaluación de interfaces humano-computadora

(Traducido por Ma. Dolores Mendoza Guzmán, bajo la dirección del Dr. Fernando Gamboa Rodríguez)

Los criterios que a continuación se presentan, fueron desarrollados por Scapin y Bastien (1997). Si bien no constituyen un método de evaluación formal, sí nos ofrecen una guía para la evaluación de nuestras interfaces-usuario desde un punto de vista no técnico. Podemos verlos como un suplemento de los métodos formales de evaluación.

Estos criterios ergonómicos tienen dos objetivos:

1. **Definir o formalizar las diferentes dimensiones que conforman el concepto de "utilizable" (c.f. Capítulo 2), sustento de lo que denominamos "software de calidad".**

2. **Proporcionar una herramienta que facilite, mejore y documente el proceso de evaluación de las interfaces-usuario.**

El conjunto de criterios consiste de ocho criterios principales, algunos de los cuales, se encuentran subdivididos en criterios más específicos. De tal manera, el conjunto de criterios ergonómicos está compuesto de 18 criterios. Para cada uno de ellos, se proporciona una definición, un sustento sobre el cual se basa el criterio y ejemplos para su aplicación.

1. GUÍA

La guía del usuario se refiere a los medios disponibles para aconsejar, orientar, informar, instruir y guiar a los usuarios a través de su interacción con la computadora (mensajes, alarmas, etiquetas, etc.).

Sustento:
Una buena guía facilita el aprendizaje y el uso de un sistema. Permite a los usuarios saber en cualquier momento dónde se encuentran al realizar una secuencia de interacciones, o en el cumplimiento de una tarea.

El criterio de guía está subdividido en cuatro sub-criterios:

a) **Incitación.** Se refiere a los medios disponibles para llevar a los usuarios a la fabricación de acciones específicas, sea una entrada de datos u otras tareas. Así mismo, este criterio se refiere a todos los medios que ayudan a los usuarios a conocer las alternativas posibles y aquellas que le ayudan a identificar el lugar donde se encuentra dentro de la aplicación.

   Sustento:
   Una buena incitación guía a los usuarios en sus interacciones y permite que sepan el modo actual de la aplicación, lo cual les ayuda a navegar de manera efectiva reduciendo los errores en su uso.

   Ejemplos:
   - Para entradas de datos, conviene proveer al usuario los formatos requeridos y valores aceptables, por ejemplo, incluir en un campo la etiqueta que señale la estructura de los datos: (dd/mm/aa): __/__/__.
   - Despliegue de las unidades de medida para la entrada de los datos.
   - Proveer señales sobre la longitud aceptable de las entradas.
   - Proveer un título por cada entrada.
   - Proveer una ayuda en línea y una guía.

b) **Agrupación/distinción de elementos.** Concierne a la organización visual de campos de información. Toma en cuenta la topología, distribución y características gráficas de los datos desplegados. El agrupamiento o distinción de elementos puede ser realizado en base a dos criterios diferentes: Agrupación/distinción por localización y Agrupación/distinción por formato.

   Sustento:
   El entendimiento de una pantalla depende, entre otras cosas, del orden, posición y la distinción de los objetos (imágenes, textos, comandos, etc.), en que son presentados. Los usuarios pueden ubicar los diferentes campos o grupos de campos y aprenderán sus relaciones más fácilmente si se presentan de una manera organizada.

   Ejemplos:
   - Organizar los campos en una lista jerárquica.
   - Al presentar varias opciones, la organización de éstas debe ser lógica, es decir, debe tener una organización funcional relevante (orden alfabético, funcional, frecuencia de uso, etc.).
   - Proveer una distinción visual clara de las áreas que tienen funciones diferentes (zona de comandos, zona de mensajes, etc.).
   - Proveer una distinción visual clara de los campos de datos y sus etiquetas.

c) **Retroalimentación inmediata.** Se refiere a las respuestas que el sistema brinda para cada acción del usuario.

Sustento:
La calidad de retroalimentación y rapidez son dos factores importantes para tener la confianza del usuario y su satisfacción. Permiten a los usuarios ganar un mejor entendimiento del funcionamiento del sistema. Por otro lado, la ausencia de retroalimentación puede ser desconcertante para el usuario, ya que podría pensar que el sistema tiene una falla, trayendo como consecuencia la interrupción de las tareas.

Ejemplos:
- Cuando exista un proceso largo, es conveniente denotar el estado del sistema.
- Las entradas de datos por parte del usuario deben ser desplegadas, excepto cuando se trate de entradas de seguridad (passwords).

d) **Legibilidad.** Concierne a las características de la información en pantalla que puedan facilitar o dificultar su lectura (caracteres brillantes, contrastes entre la letra y fondo, tamaño de las letras, espacios entre palabras, párrafos, etc.).

Sustento:
Mantener una buena legibilidad en las pantallas, facilita la lectura de la información presentada.

Ejemplos:
- Diferenciar los títulos de las pantallas con los títulos o palabras que tienen una interacción.
- Hacer uso de letras mayúsculas y minúsculas.
- Tener en cuenta el justificado, espacio, tamaño, color, etc. de las letras.

2. CARGA DE TRABAJO

Concierne a todos los elementos de la interfaz que juegan un papel en la reducción de la carga perceptual y cognoscitiva del usuario, y en el incremento de la eficiencia del diálogo.

Sustento:
Es conveniente aportar sólo la información necesaria para que el usuario no se distraiga y realice su tarea eficientemente; de esta manera, se evita que cometa errores.

Este criterio está subdividido en dos sub-criterios:

a) Brevedad. **Se refiere a la carga de trabajo perceptual y cognoscitiva para entradas y salidas individuales, y para un conjunto de entradas (conjunto de acciones necesarias para realizar una meta o tarea). La brevedad corresponde a la meta de limitar la lectura y entrada de la carga de trabajo y el número de acciones a seguir.**

Sustento:
Usando términos cortos, reduce la probabilidad de cometer errores y el tiempo de lectura es más corto.

Este criterio se subdivide a su vez en dos sub-criterios:

- Concisión. **Concierne a la carga perceptual y cognoscitiva para entradas y salidas de información. Por definición, este criterio no toma en cuenta la retroalimentación de los mensajes de error.**

  Sustento:
  Usando términos cortos, reduce la probabilidad de cometer errores y el tiempo de lectura es más corto.

  Ejemplos:
  - Para datos numéricos, los ceros en las entradas no deben ser necesarios.
  - Si los códigos son más largos de 4 o 5 caracteres, usar mnemónicos o abreviaciones.
  - Permitir al usuario entradas cortas de datos.
  - Cuando una unidad de medida es asociada con un campo de datos particular, es mejor incluir esa unidad como parte de la etiqueta del campo que pedírsela al usuario.

- **Acciones Mínimas.** Concierne a la carga de trabajo con respecto al número de acciones necesarias para completar una meta o tarea. Se busca limitar lo más posible los pasos que el usuario realiza en una tarea.

  Sustento:
  Las numerosas y complejas acciones necesarias para completar una meta, incrementan la carga de trabajo del usuario y consecuentemente, hay más probabilidad de cometer errores.

  Ejemplos:
  - Reducir el número de pasos para hacer una selección en un menú.
  - No solicitar entradas de datos al usuario cuando los datos pueden ser deducidos por la computadora.
  - Evitar entradas del usuario de comandos que incluyan puntuación.
  - Para entradas de datos, se sugiere desplegar valores definidos por omisión en sus campos de datos apropiados.
  - Para despliegues largos y con varias páginas, es conveniente pedir una página particular directamente, sin tener que ir por todas las páginas intermedias.

b) **Densidad de la información.** Concierne a la carga de trabajo del usuario desde un punto de vista perceptual y cognoscitivo ocasionada por los grupos de elementos, y no por elementos asilados como en el caso de Brevedad.

Sustento:
En muchas tareas, las ejecuciones de los usuarios son empeoradas cuando la densidad de la información es demasiada alta o muy baja. En estos casos, los errores llegan a aparecer. Los elementos que no estén relacionados a la tarea deberían ser removidos. Por otra parte, la carga de memoria sobre el usuario debe ser minimizada. Los usuarios no deben memorizar largas listas de datos o procedimientos complicados. Ellos no deben encargarse de actividades cognoscitivas complejas cuando éstas no sean requeridas por la tarea.

Ejemplos:

- Proveer al usuario solamente de lo necesario (datos útiles) e inmediatamente, para cualquier transacción.
- Los datos no deben requerir transacciones únicas.
- El lenguaje de consulta debe usar el mínimo de cuantificadores en la formulación de la consulta.
- No pedir al usuario que recuerde exactamente los datos de una ventana a otra.
- Proveer de una computación automática de datos derivados, para que el usuario no tenga que calcular e introducir cualquier número que pueda ser derivado de datos ya accesibles a la computadora.

3. CONTROL EXPLÍCITO

Concierne al procesamiento por parte del sistema de acciones explícitas del usuario, así como el control que debe tener el usuario sobre un proceso.

Sustento:
Cuando los usuarios explícitamente definen sus entradas y cuando esas entradas están bajo su control, los errores también como sus ambigüedades son limitadas. Además, el sistema será mejor aceptado por los usuarios si ellos tienen control sobre el diálogo.

Este criterio esta subdividido en dos sub-criterios:

a) **Acciones explícitas del usuario.** Se refiere a las relaciones entre el procesamiento de la computadora y las acciones de los usuarios. Esta relación debe ser explícita, esto es, que la computadora debe procesar solamente aquellas acciones solicitadas por el usuario y sólo cuando se necesiten.

   Sustento:
   Cuando el procesamiento de la computadora responde a las acciones explícitas del usuario, los usuarios aprenden y entienden mejor el funcionamiento de la aplicación, por lo que se observan menos errores.

   Ejemplo:
   - Permitir al usuario iniciar los procesos mediante una acción explícita (como pulsar una tecla) y no iniciar una tarea antes.

b) **Control del usuario.** Se refiere al hecho de que los usuarios siempre tendrán el control del procesamiento del sistema (como interrumpir, cancelar, pausar y continuar). Cada acción posible por un usuario será anticipada, proporcionando las opciones apropiadas.

   Sustento:
   El control sobre las interacciones favorece el aprendizaje y entonces disminuye la probabilidad de cometer errores. Como consecuencia de esto, la computadora llega a hacerse más predecible.

   Ejemplo:
   - Permitir a los usuarios el control sobre avance de pantallas, impresión o cancelación de una transacción.

4. ADAPTABILIDAD

La adaptabilidad de un sistema se refiere a su capacidad para comportarse de manera contextual y de acuerdo a las necesidades y preferencias del usuario.

Sustento:
Hay diversas maneras de llevar a cabo una tarea dada, un usuario particular encontrará la forma de adecuada para él. Los diferentes procedimientos, opciones y comandos, deben ser disponibles para el usuario cuando realice su tarea.

Este criterio de subdivide en dos sub-criterios:

a) **Flexibilidad.** Es la capacidad de la interfaz para adaptarse a las necesidades particulares de los usuarios.

   Sustento:
   Al ofrecer diferentes maneras de realizar una tarea dada, se da más probabilidad de que el usuario elija y aprenda una de ellas durante su aprendizaje. Una buena flexibilidad permite al usuario adaptar la interfaz a sus necesidades.

   Ejemplos:
- Cuando las necesidades del usuario sean inciertas, conviene proveerle de algunos medios para controlar la configuración del despliegue.
- Cuando los diseños de la interfaz no puedan predecir qué valores por omisión serán útiles, permitir al usuario definir, cambiar o remover los valores por omisión para entradas de datos.
- Cuando algunos despliegues sean innecesarios, el usuario debe ser habilitado para removerlos temporalmente.
- Proveer de algunos medios para que el usuario cambie la secuencia de las entradas de datos y respetar la secuencia elegida.
- Cuando los formatos de texto no puedan ser pronosticados en adelante, permitir al usuario especificar y almacenar para futuros usos los formatos que pueda necesitar.
- Los usuarios deben ser habilitados para asignar nombres a campos de datos que hayan creado.

b) **Experiencia del usuario.** Se refiere a los medios disponibles para tomar en cuenta el nivel de experiencia del usuario.

   Sustento:
   La experiencia e inexperiencia de los usuarios requieren diferentes necesidades de información. Para usuarios no experimentados, es conveniente permitirle acciones simples paso a paso. Para usuarios expertos, los diálogos iniciales de la computadora pueden ser aburridos y pueden alentar sus interacciones; los comandos cortos deben permitirle acceder a las funciones del sistema de una forma más rápida. Los diferentes niveles de interacción deben tomar en cuenta la experiencia del usuario. Sin embargo, muchos sistemas tendrán usuarios con diferentes niveles de experiencia. Los usuarios pueden llegar a ser expertos conforme incrementan su experiencia, o tal vez menos expertos, después de un largo período de desuso. La interfaz debe ser diseñada para acomodar la variedad de niveles de experiencia de los usuarios.

Ejemplos:

- Permitir a los usuarios experimentados introducir entradas de comandos equivalentes o teclas cortas directamente a una serie de selecciones de menú.
- Permitir a los usuarios experimentados teclear una serie de comandos en un tiempo, y a los usuarios no experimentados permitir la entrada paso a paso.
- Los tipos de diálogo deben ser diseñados para igualar las necesidades de diferentes usuarios.
- Diversos tipos de diálogo deben ser provistos como una función de la experiencia de los varios grupos de usuarios (ejemplo: entradas de datos como una característica de pauta que puede ser seleccionada por usuarios novatos pero que pueden ser omitidos por usuarios experimentados.
- Cuando las técnicas adoptadas por las pautas de los usuarios puedan alentar al usuario experimentado, proveer de rutas o modos alternativos permitiendo a un usuario evitar los procedimientos estándar.
- Después de un mensaje de error, permitir a los usuarios solicitar una explicación con más detalle que sea ajustable a su nivel de conocimiento.

5. MANEJO DE ERRORES

Se refiere a los medios disponibles para prevenir o reducir errores y recuperarlos a partir de cuando ellos suceden. Los errores se definen en este contexto como entrada de datos inválidos, formatos inválidos en la entrada de datos, sintaxis de comando incorrecta, etc.

Sustento:
Las interrupciones causadas por los errores del usuario pueden traer consecuencias negativas en las actividades del usuario. En general, este tipo de interrupciones incrementa el número de interacciones y rompen la organización y el cumplimiento de la tarea.

Este criterio se subdivide en tres sub-criterios:

a) **Protección contra errores.** Se refiere a los medios disponibles para detectar y prevenir errores en la entrada de datos, errores en comandos o en acciones con consecuencias destructivas.

   Sustento:
   Es preferible detectar errores antes de la validación y no después.

   Ejemplos:
   - Cuando el usuario requiere la terminación de una tarea, y si alguna transacción está pendiente no será terminada, o si los datos se perdieron, es conveniente desplegar un mensaje de advertencia requiriendo confirmación por parte del usuario.
   - Proteger los campos por los cambios accidentales que pueden hacer los usuarios.
   - Los campos diseñados para el desplegado de información deben ser protegidos: a los usuarios no se les debería permitir cambiar la información contenida en esos campos.
   - Asegurarse que el software para la interfaz-usuario funcionará apropiadamente con los posibles errores que puedan ocurrir, incluyendo las entradas de datos accidentales.

b) **Calidad en los mensajes de error.** Se refiere a la frase y contenido de los mensajes de error, esto es: relevancia, facilidad en la lectura y especificación acerca de la naturaleza de los errores (formato, sintaxis) y las acciones necesarias para corregirlos.

Sustento:
La calidad en los mensajes de error promueve el aprendizaje del usuario en cuanto a sistemas indicando las razones de sus errores, su naturaleza y enseñándoles las maneras de prevenirlos.

Ejemplos:
- Especificar los mensajes tanto como sea posible.
- Redactar mensajes breves e informativos.

c) **Corrección de errores.** Se refiere a los medios disponibles para que los usuarios corrijan sus errores.

Sustento:
Los errores son menos dañinos cuando son rápidamente corregidos.

Ejemplo:
- Si la transacción de la entrada de datos ha sido terminada y los errores han sido detectados, se debe permitir a los usuarios hacer correcciones inmediatamente.

6. CONSISTENCIA

Se refiere a la manera en que el diseño de una interfaz se mantiene para contextos similares, y se diferencia para contextos diferentes.

Sustento:
Los procedimientos, etiquetas, comandos, etc., serán mejor nombrados, localizados y reconocidos, si su formato, localización y sintaxis son establecidos de una a otra pantalla, o de una sesión a otra. La falta de consistencia es una de las razones importantes para el rechazo de los usuarios en cuanto al software.

Ejemplos:
- Los títulos de las pantallas deberían asignarse en el mismo lugar.
- Usar formatos de pantallas similares.
- Usar procedimientos similares para acceder a las opciones de los menús.

7. SIGNIFICADO DE CÓDIGOS

Califica la relación entre un término y/o un signo, y el objeto o comando al que hace referencia. Los códigos y nombres son importantes para los usuarios cuando existe una relación clara entre tales códigos y acciones.

Sustento:
Cuando los códigos se pueden manipular, éstos se pueden recordar e identificar con facilidad. En contraste, cuando se utilizan códigos o nombres sin significado puede llevar a que los usuarios realicen operaciones inapropiadas, lo que conduce a errores.

Ejemplos:
- Los títulos deben ser distintos y manejables.
- Hacer de las abreviaciones reglas explícitas.
- Los códigos deben ser manejables y familiares.

8. COMPATIBILIDAD

Se refiere a la relación que existe entre las características del usuario (memoria, capacidad cognoscitiva, capacidad perceptual, experiencia, preferencias, etc.) y su tarea (qué hace, cómo lo hace, qué objetos utiliza, en qué momento, etc.), con respecto a la organización de las entradas/salidas, y en el diálogo de la aplicación.

Sustento:
La información que es transferida de un contexto a otro, se realiza de forma rápida y de manera más eficiente cuando el volumen de la información que recuperan los usuarios es limitado. La eficiencia se aumenta cuando los procedimientos diseñados para acompañar y completar las tareas son compatibles con las características psicológicas de los usuarios; los procedimientos y las tareas se organizan de acuerdo a lo que los usuarios esperan que haga, así como a sus costumbres.

Ejemplos:
- Cuando los datos que se introducen involucran transcripciones de documentos, se debe de asegurar que las formas de llenado sean las mismas que las del documento original.
- Los mensajes deben de reflejar la estructura de los datos o su organización, para que los usuarios las perciban de forma natural.
- Los formatos de los calendarios deben seguir la costumbre del lugar.
- Las etiquetas, entradas y mensajes de ayuda, deben ser familiares al contexto del usuario.
- El despliegue de mensajes, datos textuales o instrucciones deben seguir un mismo diseño o seguir los diseños convencionales de impresión.

Bibliografía

[1]. B. W. Kernighan y D. Ritchie, *El lenguaje de Programación "C"*, Segunda edición, Prentice Hall, México, 1991.

[2]. E. Harlow, *Desarrollo de aplicaciones Linux con GTK+ y GDK*, Prentice Hall Iberia, Madrid, España, 1999.

[3]. K. Mullet y D. Sano, *Designing Visual Interfaces*, Prentice Hall, New Jersey, USA, 1995.

[4]. Ch. Strothotte y T. Strothotte, *Seeing Between the Pixels*, Springer, New York, USA, 1997.

[5]. J. Foley, A van Dam, S. K. Feiner, J. F Hughes, R. L. Phillips , *introducción a la graficación por computador*, Segunda edición, Addison Wesley, Argentina ; México 1996.

[6]. M. Woo, J. Neider, T. Davis, D. Shreiner , *OpenGL Programming Guide*, Second edition, Addison Wesley, Massachusetts , USA, 1997.

[7]. C. Walnum, *3-D Graphics Programming With Opengl*, Que, Indiana, USA, 1997.

[8]. C. F. Goldfarb, P. Prescod , *Manual de XML*, Prentice Hall Iberia, Madrid, España, 1999.

[9]. D. Pescador , *3D Studio Max versión 2.5*, Anaya multimedia, Madrid, España, 1999.

[10]. N. Walsh and L. Muellner, *3D DocBook: The Definitive Guide*, First edition, O'Reilly, California, USA, 1999.

[11]. B. W. Kernighan y R. Pike , *La Práctica de la Programación*, Prentice Hall, México, 2000.

[12]. A. M. Tenenbaum, Y. Langsam y M. A. Augenstein, *Estructuras de Datos en C*, Prentice Hall, México, 1993.

[13]. Gamboa, F. and. Scapin D. L, *Editing MAD* task descriptions for specifying user interfaces at both, semantic and presentation levels. Design, Specification and Verification of Interactive Systems* '97. M. D. Harrison and J. C. Torres. Springer-Verlag, Wien, Austria, pp. 193-208, 1997.

[14]. Rubin, J., *Handbook of Usability Testing.*, John Willey & Sons, Inc., New York, USA, 1994.

[15]. Scapin, D. L. and J. M. C. Bastien, "*Ergonomic Criteria for evaluating the ergonomic quality of interactive systems.*" Special Issue of Behavior and Information Technology on usability methods. 16(4 / 5): pp. 220-231, Wien, Austria, 1997.

[16]. Shneiderman, B., *Designing the user interface.* Reading Massachusets, Berkeley California, Mexico City, Addison Wesley Longman, Inc, USA, 1998.

[17]. Treu, S., *User Interface Evaluation.* A structured approach., Plenum Press, New York, USA, and London England, 1994.

[18]. Wood, L. E., *User Interface Design.* Bridging the Gap from User Requirements to Design., CRC Press, New york, USA, 1998.

[19]. Faultkner, C., *The Essense of Human-Computer Interaction.* Prentice Hall, New Jersey, USA, 1998.

[20]. Wagner , *The art of Human-Computer Interface Design*, Addison Wesley, Cuarta Impresión, MA-USA, 1992.

[21]. Watt, A., *Three – Dimensional Computer Graphics*, Addison Wesley, pp 1-43, MA-USA , 1989.

[22]. Hearn, D., *Gráficas por Computadora*, Prentice Hall Hispanoamericana, segunda edición, Madrid, España pp. 319-427, 1995.

[23]. Young, D., *Netscape Developer's Guide to Plug-Ins*, Prentice Hall, New Jersey, USA 1997.

[24]. Siegel, J., *CORBA 3 Fundamentals and Programming*, John Wiley & Sons, New Jersey, USA , 2000.

[25]. Raymond. E, *The Cathedral & the Bazaar*, O'Reilly, Sebastopol, USA, 2001.

[26]. DICKERSON K., BENTON R., ATYEO M., *Rapid prototyping tools and techniques.* B T Technol. Journal; Vol. 6, No 4, October, London, England, 1988.

[27]. Lorés J., Granollers T., Lana S., *Introducción a la Interacción Persona–Ordenador*, Jesús Lorés, Editor, Madrid, España, 2001.

[28]. Cañas j., Salmerón L., Gámez P., *El Factor humano*,. Jesús Lorés, Editor, Madrid, España, 2001