Proyecto Final de Carrera

Visor BRep

Un programa para visualizar modelos 3D

Autor: Vicente Matoses

Director: Eduardo Vendrell

Mayo de 2012

Ingeniería Informática (Universidad Politécnica de Valencia)

ÍNDICE

1 Introducción 3

1.1 Resumen y palabras clave 4

2 El Diseño Asistido por Computador (CAD) 5

2.1 Modelado alámbrico 5

2.2 Modelado de superficie 6

2.3 Modelado sólido 7

2.4 Modelado por representación de fronteras (Boundary Representation o B-Rep) 11

3 La aplicación 15

3.1 Software utilizado 15

3.1.1 Desarrollo de la aplicación 15

3.1.2 Iconos de la aplicación 16

3.1.3 Vídeo explicativo de la aplicación 16

3.2 Descripción de las características y de la funcionalidad de la aplicación 18

3.2.1 Inicio del programa 18

3.2.2 Vista Código 20

3.2.3 Vista Datos 22

3.2.4 Vista Dibujo 23

3.2.4.1 Controles de la representación gráfica 24

3.2.4.2 Otras opciones 25

3.2.5 Menú 27

3.2.6 Menú contextual 29

3.3 Ayuda en el programa 31
3.4 Algunas consideraciones sobre la ejecución del programa 35
3.4.1 Gestión de memoria 35
3.4.2 La clase tessellator 36
3.5 Instalación y requisitos de la aplicación 36
3.6 Versiones de la aplicación según el sistema operativo 40
4 Control de errores 42
4.1 Código 42
4.2 Objetos 43
4.3 Vértices 43
4.4 Aristas 44
4.5 Caras 45
4.6 Colores 45
4.7 Otros comentarios 46
4.8 Descripción de cómo se han comprobado algunos errores 48
4.8.1 Aristas coincidentes 48
4.8.2 Aristas paralelas 48
4.8.3 Aristas superpuestas 48
4.8.4 Cálculo del vector normal a las caras 52
4.8.5 Valor del plano 53
4.8.6 Caras planas 55
4.8.7 Aristas que se cortan 55
4.8.8 Aplicación de las fórmulas de Euler 62
5 Formato propio que usa la aplicación 65
5.1 Objetos 65
5.2 Vértices 65

5.3 Aristas 66
5.4 Caras 67
5.5 Colores 67
5.6 Orden 68
5.7 Estructuras que se generan 68
5.7.1 Listas de elementos 68
5.7.2 Matriz de vértices de dibujo 69
5.7.3 Clase vértice 69
5.7.4 Clase arista 70
5.7.5 Clase cara 70
5.7.6 Clase objeto 71
5.7.7 Clase color 72
5.7.8 Clase vectores 72
5.7.9 Secuencia de análisis 73
6 Conclusiones 74
7 Agradecimientos 75
8 Bibliografía y otros recursos 76
8.1 Java 76
8.2 OpenGL 76
8.3 Java for OpenGL (JOGL) 76
8.4 Diseño asistido por computador 77
8.5 Otros recursos 77
9 Índice de ilustraciones 79
10 Índice de tablas 81

1 Introducción

En el entorno de la enseñanza del Diseño Asistido por Computador, se necesitan herramientas para poder visualizar los diseños realizados.

Por esto, para las clases de prácticas de DAC de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia, se propuso actualizar la aplicación que se usaba.

Basándose en un formato que se había definido y que estaba usándose, se estableció un formato nuevo, que incluía información sobre colores y que se ajustaba al formato que se estudia en las clases de teoría.

La aplicación, además de mostrar los diseños, debía permitir modificar los parámetros en este tipo de aplicaciones (orientación de la cámara, zoom, giros en los objetos, etc) todo ello controlado desde el ratón.

Por otro lado, otro requisito importante era que la aplicación tenía que ser multiplataforma, es decir, que se pudiese ejecutar en los sistemas operativos más usados (windows, linux y mac).

Además, el programa tenía que controlar el máximo número de errores posibles, tanto desde el punto de vista del formato definido (formato de las definiciones de cada elemento, orden, etc), como de la representación gráfica (que las aristas no se crucen, que los polígonos sean planos, etc).

Por lo tanto, el objetivo principal de este proyecto final de carrera se plantea como la generación de una aplicación que pueda representar gráficamente objetos definidos con este formato BRep propio, y que controle el mayor número posible de errores en la representación.

Como un objetivo secundario, se propone generar documentación explicativa de la aplicación. Esta consistirá básicamente en la memoria del PFC, una ayuda dentro del programa, y un vídeo subtitulado que explique la funcionalidad básica del programa en cualquier ordenador, incluyendo las salas de prácticas, en las que puede resultar complicado oir el sonido del vídeo.

1.1 Resumen y palabras clave

En este proyecto final de carrera se genera una aplicación gráfica multiplataforma con la que se pueden representar objetos definidos en un formato BRep.

También incluye un vídeo explicativo de la funcionalidad de la aplicación.

PALABRAS CLAVE:

CAD, DISEÑO ASISTIDO POR ORDENADOR, MODELADO SÓLIDO, BREP, BOUNDARY REPRESENTATION, JOGL

2 El Diseño Asistido por Computador (CAD)

El Diseño Asistido por Computador, más conocido por su abreviatura en inglés CAD (Computer Aided Design) es un conjunto de técnicas que permiten optimizar el diseño de productos.

Su objetivo consiste en obtener representaciones gráficas de los objetos mediante el uso de ordenadores. De esta forma se consigue que la representación sea más rápida, limpia, exacta, reproducible y editable que por los medios tradicionales.

Además, en los últimos años se han desarrollado aplicaciones que han ido incorporando nuevas posibilidades. Hoy en día se pueden añadir funcionalidades que incorporan las propiedades físicas y mecánicas de los materiales con los que se construirán los objetos.

También se pueden obtener animaciones en las que se representa el funcionamiento de los objetos que forman parte de mecanismos, o el proceso de mecanizado que se ha de seguir para obtenerlos.

Con estas técnicas se consigue que el diseño de los objetos sea más rápido y que se puedan comprobar fallos o realizar modificaciones antes de pasar a la producción.

Para obtener la representación gráfica de los objetos, necesitamos usar alguna técnica de modelado geométrico. Dentro del CAD se distinguen tres tipos de modeladores geométricos: alámbricos, de superficies y de sólidos.

2.1 Modelado alámbrico

En primer lugar tenemos los modeladores alámbricos. En este tipo de modelado los objetos se representan mediante líneas.

En los primeros años del CAD se usó el modelado en dos dimensiones (2D), representando los objetos normalmente según sus vistas en sistema diédrico (alzado, planta y perfil).

Luego se pasó al modelado 2½D, que añadía al modelado 2D líneas verticales para dar sensación de volumen.

Finalmente llegó el modelado puramente 3D, en el que se supera la limitación de representar líneas solamente verticales.

Estos modeladores presentan algunos problemas, como son la ambigüedad, la pérdida de líneas de silueta, modelos sin sentido y modelos imposibles.

La ambigüedad se da cuando no podemos determinar claramente el objeto a partir de la representación gráfica. Un ejemplo muy claro se ilustra en la siguiente figura, en la que no se puede determinar si la cara frontal del cubo está orientada hacia la esquina inferior izquierda o hacia la esquina superior derecha.

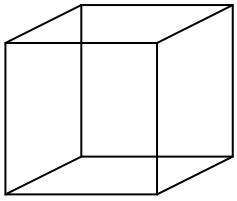

Figura 1. Representación ambigua en modelado alámbrico

La pérdida de líneas de silueta se produce cuando en la representación no se incluye la información necesaria para determinar la silueta.

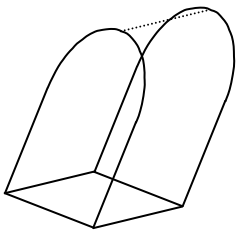

Figura 2. Pérdida de líneas de silueta en modelado alámbrico

2.2 Modelado de superficie

En segundo lugar tenemos los modeladores geométricos de superficies. Estas pueden ser poliédricas (cuando el objeto se puede representar o aproximar mediante poliedros) o libres (usando ecuaciones matemáticas para describirlas).

Al igual que el modelado alámbrico, este tipo de modelado también presenta problemas, sobre todo a la hora de representar bordes o superficies curvas, aunque mejora al modelado alámbrico. Es lo que se llaman imprecisiones por el facetado. Si se representa un cilindro, y se junta con otro cilindro, puede que no se aprecie bien la línea de unión. Se ilustra en la siguiente figura.

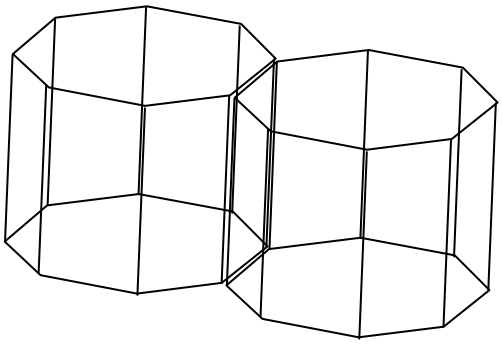

Figura 3. Imprecisiones por facetado en el modelado de superficies

De todas formas, el problema más importante es que el modelado por superficies no garantiza que el objeto representado describa un volumen cerrado.

Además se plantea la necesidad de distinguir entre interior, exterior y superficie de un objeto. Por esto es por lo que se introdujo el modelado sólido.

2.3 Modelado sólido

Dentro del modelado sólido tenemos varias posibilidades: modelos constructivos, modelos de enumeración espacial y modelos de representación de fronteras.

Modelos constructivos

Dentro de los modelos constructivos, los más conocidos son los barridos, la instanciación y parametrización, y el modelo de Geometría de Construcción de Sólidos o CSG.

El modelado sólido mediante barridos genera sólidos mediante el desplazamiento de superficies, quedando definido como el espacio por el que se ha desplazado la superficie. El movimiento puede ser de traslación, rotación, o libre, sobre una línea cualquiera.

Este modelado tiene sus limitaciones. Además de que su dominio es limitado, puede provocar intersecciones consigo mismo, áreas como resultado (si se desplaza la línea paralelamente al plano que la contiene), y no hay una formulación simple del resultado, pero tiene una utilidad indudable en el análisis de mecanizados o en los ensamblajes para evitar interferencias.

En la instanciación y parametrización partimos de una serie de primitivas que se pueden instanciar modificando los parámetros que las definen. Por ejemplo, a partir de un cubo podemos generar cualquier prisma de base rectangular, modificando sus dimensiones. Se ilustra en la figura siguiente.

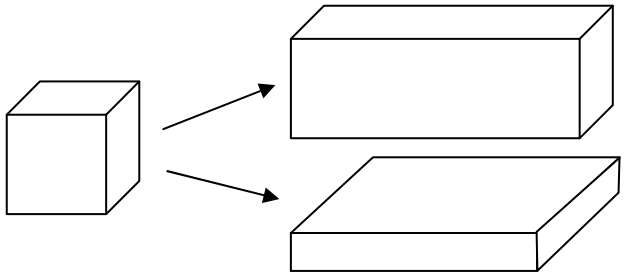

Figura 4. Instanciación y parametrización

Este modelado presenta el problema de la riqueza de primitivas de las que dispongamos.

Otro tipo de modelado importante es la geometría de construcción de sólidos o CSG. En este caso, partimos de una serie de primitivas que podemos combinar con una serie de operaciones lógicas, tales como la unión, la sustracción o la intersección.

En la siguiente figura se representan los casos en que a partir de un prisma rectangular y un cilindro se genera una pieza con un agujero por sustracción, o una pieza cilíndrica con base rectangular por unión.

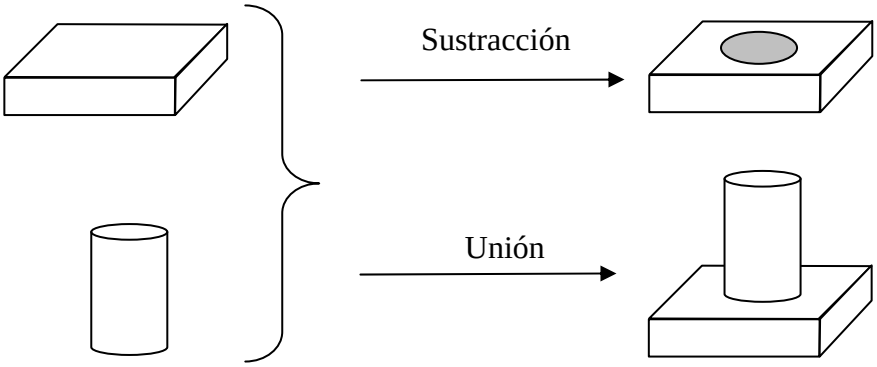

Figura 5. Ejemplo de modelado CSG

El mayor problema de este modelado es que las operaciones lógicas se han de regularizar, ya que en algunos casos aparecen fronteras que no engloban ningún espacio. Se ilustra en la siguiente figura, en la que al unir dos paralelepípedos, queda una frontera entre ellos que no engloba ningún espacio.

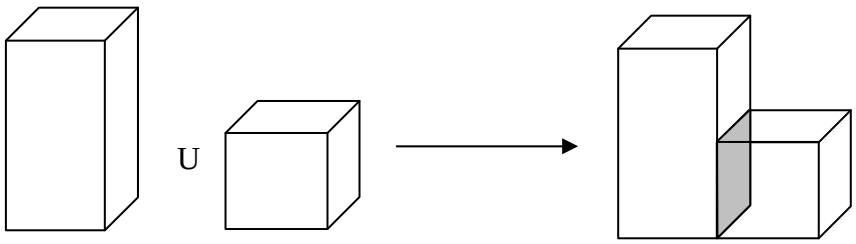

Figura 6. Problemas con las fronteras en CSG

Modelos de enumeración espacial

En la descomposición espacial lo que se hace es indicar qué partes del espacio están ocupadas por el objeto. Para especificar las partes en que dividimos el espacio, se pueden usar células (descomposición celular, en la que las células son básicamente primitivas), cubos localizados en una rejilla (enumeración de la ocupación espacial) u octrees (en este caso los cubos se agrupan cuando es posible).

Como todos los modelados, éste también tiene sus limitaciones, ya que resulta complicado representar objetos cuando estos no se ajustan a las células o cubos elegidos.

En la siguiente figura se muestra un ejemplo muy sencillo en una rejilla formada por cubos, en el que se representa una pieza en forma de T.

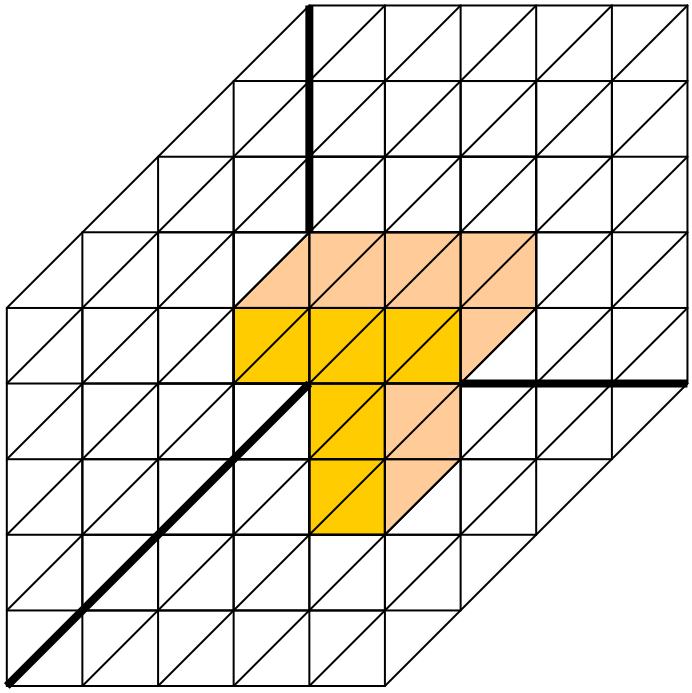

Figura 7. Ejemplo de enumeración espacial

. La pieza se definiría indicando qué cubos son los que ocupa.

Representación de fronteras

Por último, tenemos el modelado por representación de fronteras (Boundary Representation o B-Rep).

Como este modelado es el que va a utilizar la aplicación, se desarrolla más a fondo en un nuevo subapartado.

2.4 Modelado por representación de fronteras (Boundary Representation o B-Rep)

En este modelado, los objetos se representan indicando las fronteras que los definen, pero incluyen información topológica que no permite definir objetos no cerrados.

Para especificar los objetos se usa la representación de arista-alada (winged-edged). Los objetos se consideran poliedros.

La información que recoge la estructura arista alada (además de la de vértices y aristas que se recoge en tablas aparte) se ilustra en la siguiente figura.

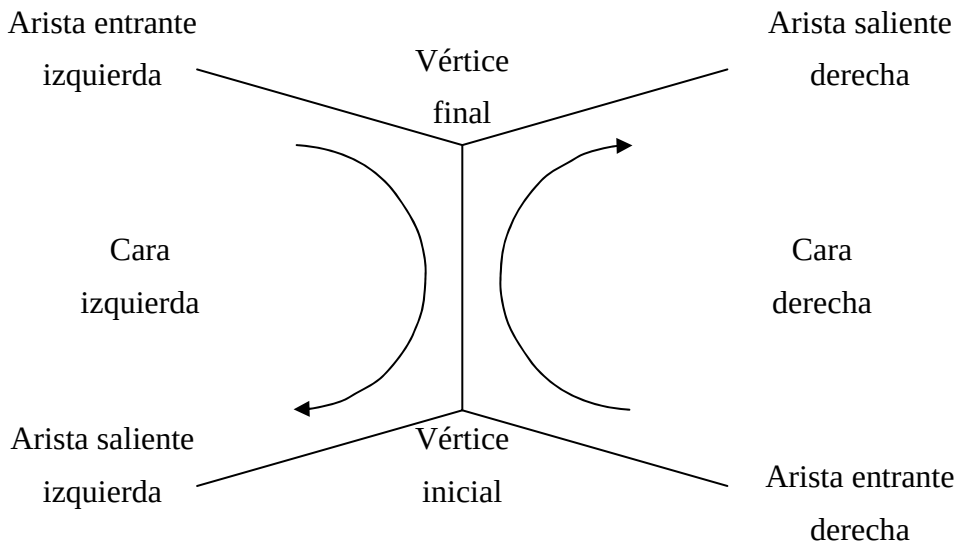

Figura 8. Información de la estructura arista alada

Para guardar la información, se usan tres tablas, una para los vértices, otra para las caras y otra para las aristas. Con un ejemplo se entenderá mejor.

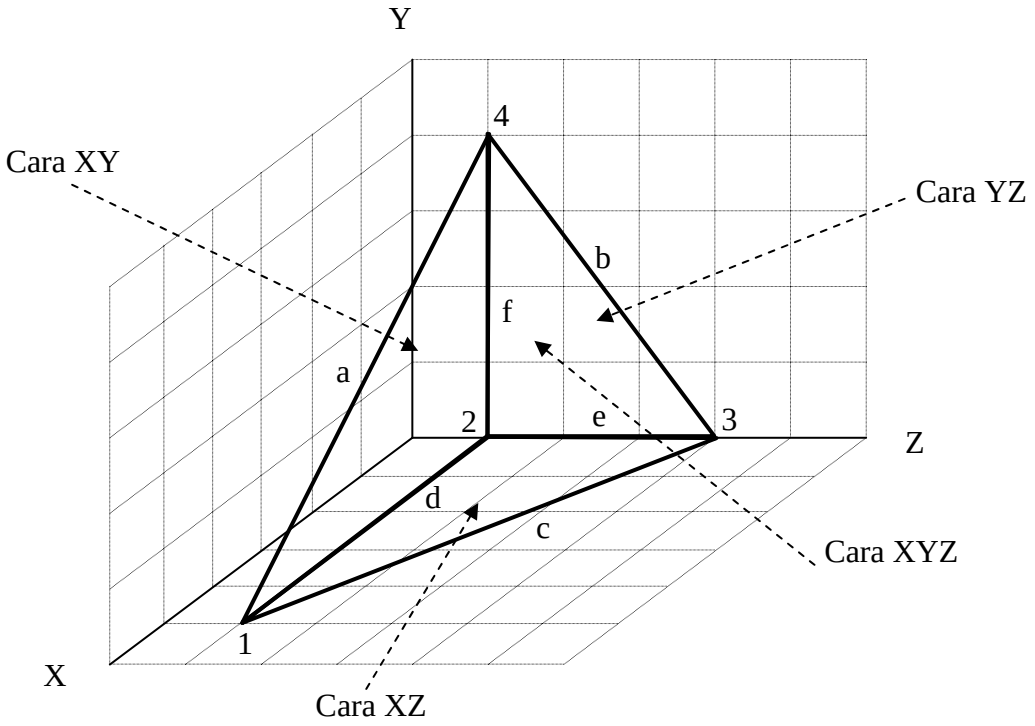

Figura 9. Ejemplo del modelado B-Rep

En el ejemplo se representa un tetraedro no regular, cuyos vértices son el 1 (5,0,1), 2 (0,0,1), 3 (0,0,4) y 4 (0,4,1). Las aristas son la a, b, c, d, e y f. Las caras son la XY (paralela al plano XY), XZ (paralela al plano XZ), YZ (paralela al plano YZ) y la cara XYZ, que estaría inclinada y taparía a las otras tres desde el punto de vista desde el que se ha representado el ejemplo.

La tabla de vértices incluye solamente los nombres y coordenadas de los mismos.

Tabla de vértices			
Vértice	X	Y	Z
1	5	0	1
2	0	0	1
3	0	0	4
4	0	4	1

Tabla 1. Ejemplo BRep Tabla de vértices

La tabla de caras incluye el nombre de las mismas y la secuencia ordenada de las aristas que la definen, en sentido horario visto desde el exterior para las fronteras externas, y antihorario para las internas.

Tabla de caras	
Cara	Aristas
XY	a-d-f
XZ	c-e-d
YZ	b-f-e
XYZ	a-b-c

Tabla 2. Ejemplo BRep Tabla de caras

La tabla de aristas es la más importante, ya que recoge la información que relaciona cada arista con los vértices que la limitan, las caras que la definen, y las aristas que la preceden o suceden en las caras. Todo ello, definido en un cierto orden, en sentido horario visto desde el exterior, y en sentido antihorario cuando se trata de fronteras interiores.

Arista	Vértices		Caras		Aristas			
	desde	hacia	izda	dcha	en cara izda		en cara dcha	
					prec	suc	prec	suc
a	1	4	XY	XYZ	f	d	c	b
b	4	3	YZ	XYZ	e	f	a	c
c	3	1	XZ	XYZ	d	e	b	a
d	1	2	XZ	XY	e	c	a	f
e	2	3	XZ	YZ	c	d	f	b
f	2	4	YZ	XY	b	e	d	a

Tabla 3. Ejemplo BRep Tabla de aristas

Como ocurre con los otros modelados, éste también tiene sus debilidades. La principal es la validez, ya que cualquier conjunto de caras no define un sólido válido. Por ejemplo, en la tabla no se tiene en cuenta si una cara corta a otra cara.

Pero a su favor tiene que asegura la consistencia topológica, ya que para poder representar un sólido mediante B-Rep, éste debe cumplir las siguientes propiedades:

- Cada arista está delimitada por dos vértices
- Cada arista separa dos caras (sólidos múltiples)
- Las caras que coinciden en una arista tienen una orientación conforme
- Las aristas solo se intersectan en los vértices
- Las caras solamente se intersectan en los vértices y aristas

3 La aplicación

3.1 Software utilizado

A continuación se comenta todo el software usado en el desarrollo de la aplicación.

Nota: Todos los programas usados son gratuitos y en el apartado de bibliografía se indican las direcciones desde las que se pueden descargar.

3.1.1 Desarrollo de la aplicación

Como la aplicación tenía que ser multiplataforma se eligió el lenguaje java, ya que es el lenguaje multiplataforma más extendido. De esta manera, mediante la máquina virtual java, instalada en la mayoría de ordenadores personales, se puede usar en diferentes plataformas.

En primer lugar, para programar en Java hace falta un IDE de programación (entorno de desarrollo integrado de programación).

Se eligió NetBeans porque se iba a trabajar en Windows, la instalación es muy sencilla, e incorpora de serie la mayoría de elementos necesarios. Se instaló la versión 7.0 Java SE.

Por otro lado, al tratarse de una aplicación gráfica, se necesitan librerías gráficas que faciliten la representación. Unas de las más extendidas, y que cumplen el requisito de ser multiplataforma, son las librerías OpenGL. Por eso se decidió utilizar estas librerías en su versión para Java (JOGL).

Esta decisión provoca dos problemas. La implementación de OpenGL para Java (JOGL) no es un estándar universalmente aceptado, y de hecho, resulta difícil encontrar documentación al respecto.

Por otra parte, las librerías JOGL usan código nativo, con lo cual, la principal ventaja de Java, el hecho de ser multiplataforma, se pierde en parte.

La consecuencia de usar estas librerías es que no se genera una sola versión del programa, si no que se crean versiones para diferentes sistemas operativos.

De todas formas, esta es la opción más recomendable y NetBeans cuenta con un plugin para instalar fácilmente las librerías JOGL. Accediendo a la página de plugins de NetBeans y buscándolo, se descarga, se descomprime y se instala desde NetBeans.

Hay un video explicativo de cómo se instala, al que se hace referencia en la bibliografía.

Por supuesto, se puede utilizar cualquier otro IDE, y las librerías se pueden descargar desde la página de java.net.

Las librerías JOGL que se han instalado fueron la versión 1.1.1.a

Para poder desarrollar la aplicación también hace falta tener instalado el kit de desarrollo java (o JDK). Se puede descargar e instalar fácilmente desde la página java.com, que incluye versiones y documentación para varios sistemas operativos. La versión que se usó para este proyecto fue la 6 (1.6.0_25).

También hay que tener instalado el Java Runtine Environment (JRE) que incluye la máquina virtual java para ejecutar la aplicación. La versión instalada en el ordenador de desarrollo es la 1.6.0_30.

Por otro lado, son necesarias algunas librerías para la generación de las ventanas (frames), paneles, menús, botones, etc. En este caso se usaron las librerías SWING, que ya vienen incorporadas al IDE NetBeans.

3.1.2 Iconos de la aplicación

Para crear los iconos de la aplicación se utilizó el programa IconFX en su versión 1.6.4. Esta es la última versión gratuita, y se puede descargar desde la página de descargas 321download.com.

En la página oficial de IconFX se puede descargar una versión de prueba que se puede usar 15 veces, pero la versión gratuita es más que suficiente para crear los iconos.

3.1.3 Vídeo explicativo de la aplicación

Con el software que hemos comentado hasta este momento (NetBeans con SWING, JDK y JRE de Java, las librerías gráficas JOGL e IconFX) ya se puede desarrollar la aplicación, pero se ha utilizado más software para completar el proyecto.

Se elaboró un vídeo en el que se muestran las características de la aplicación, y para ello se han utilizado otros programas.

En primer lugar se usó el programa Camstudio Recorder en su versión 2.6 para capturar la pantalla, y así poder grabar el vídeo explicativo de la aplicación. La grabadora de sonidos de Windows se utilizó para grabar el sonido.

En segundo lugar se usó el programa Subtitle Workshop en su versión 2.51 de urusoft.net. Con este programa, de gran versatilidad, se crearon crear los subtítulos del vídeo explicativo.

Para editar el vídeo se usó el programa spanishDub 1.5 en su versión portable. A este programa se tuvo que hay que añadir el filtro subtitle para poder manejar los subtítulos. SpanishDub es la versión española de virtualDub, una excelente aplicación de edición de vídeo. El filtro subtitle se puede descargar de la página de virtualdub.

El último programa usado para la generación del vídeo fue Zoomit. Se trata de una aplicación para poder hacer zoom de la pantalla, y así hacer más visibles los botones y el menú de la aplicación, tal como se puede apreciar en el vídeo. Se puede descargar desde la página de Microsoft.

3.2 Descripción de las características y de la funcionalidad de la aplicación

3.2.1 Inicio del programa

Cuando se arranca la aplicación (con un doble clic sobre el archivo Visor BRep.jar si se tiene asociado a este tipo de archivos la JVM), se muestra una ventana en la que solamente están activos unos pocos botones y las opciones correspondientes del menú. Además, se muestra solamente un panel de color gris.

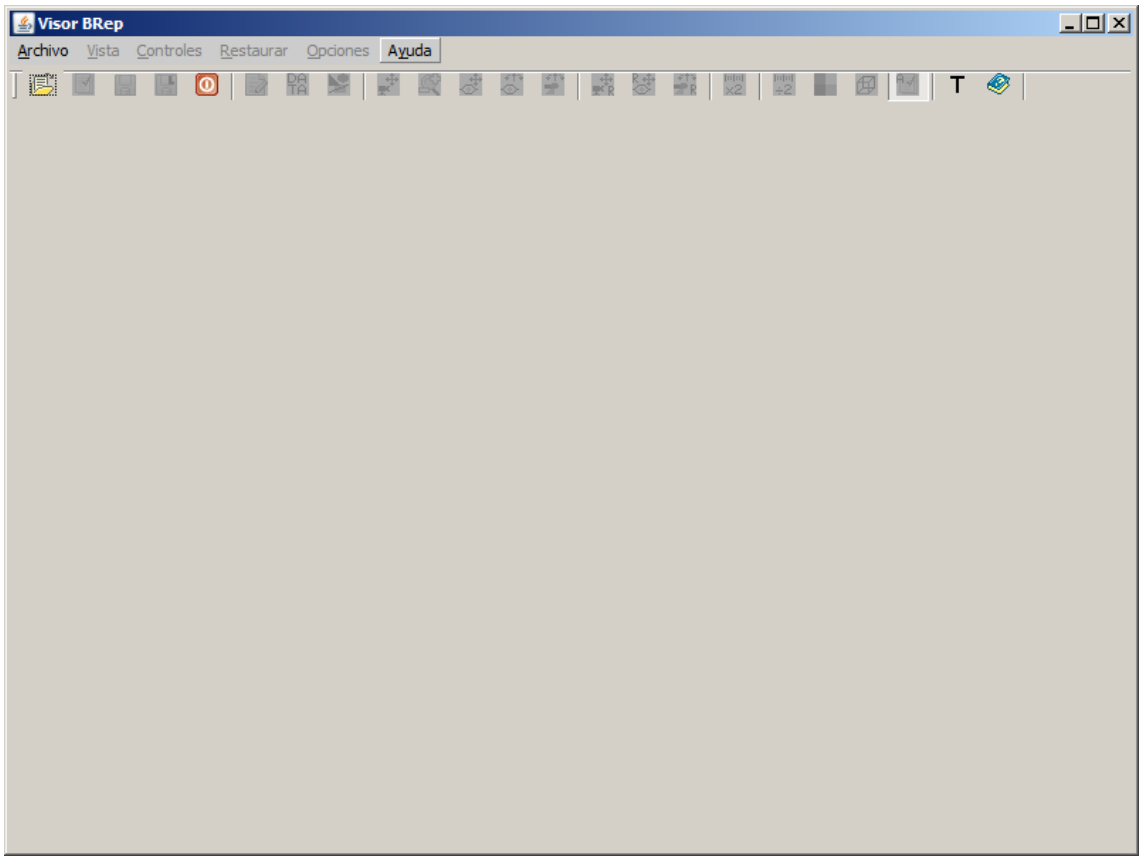

Figura 10. Ventana inicial

Como es habitual en aplicaciones basadas en ventanas, ésta se puede redimensionar, aunque tiene un tamaño mínimo de 380x200 píxeles.

A partir de este momento, el programa se puede manejar con el ratón usando la barra de herramientas donde se encuentran todos los botones. Como ayuda, al dejar unos segundos el ratón sobre los botones, aparece un texto con un nombre descriptivo de la funcionalidad del botón.

Aparte de los botones para salir de la aplicación ( ) y de la ayuda ( ), (y el de la tolerancia, que se explicará más adelante), solamente está activa la opción para abrir archivo ( ).

Cuando se elige esta opción, aparece un diálogo para elegir el archivo que se quiere abrir.

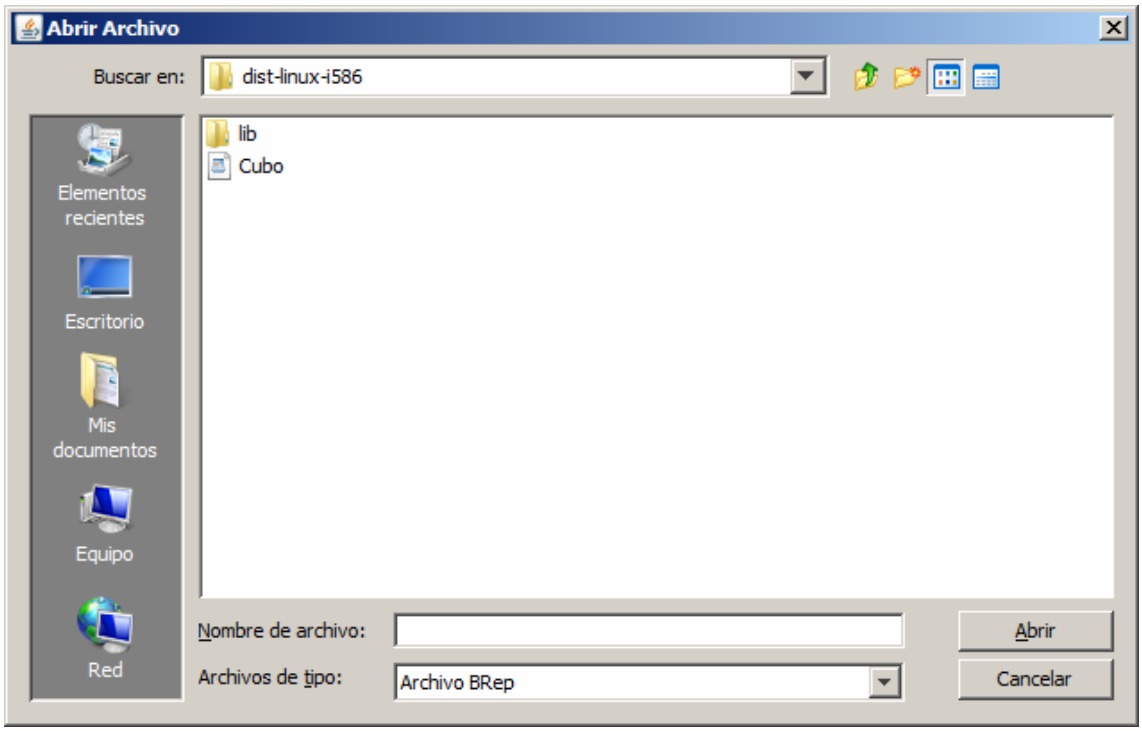

Figura 11. Diálogo para abrir archivos

Este diálogo ofrece la posibilidad de moverse por los diferentes directorios y unidades, y permite abrir archivos en dos formatos.

Estos dos formatos son archivos del tipo BRep (extensión .brp) o archivos de tipo texto (extensión .txt).

En realidad, lo único que cambia es la extensión, pero de esta forma, se puede usar un procesador de textos cualquiera para generar una primera versión del archivo, y luego guardarlo y editarlo en formato BRep con la aplicación para identificarlo mejor.

3.2.2 Vista Código

Una vez abierto el archivo, se muestra la vista Código.

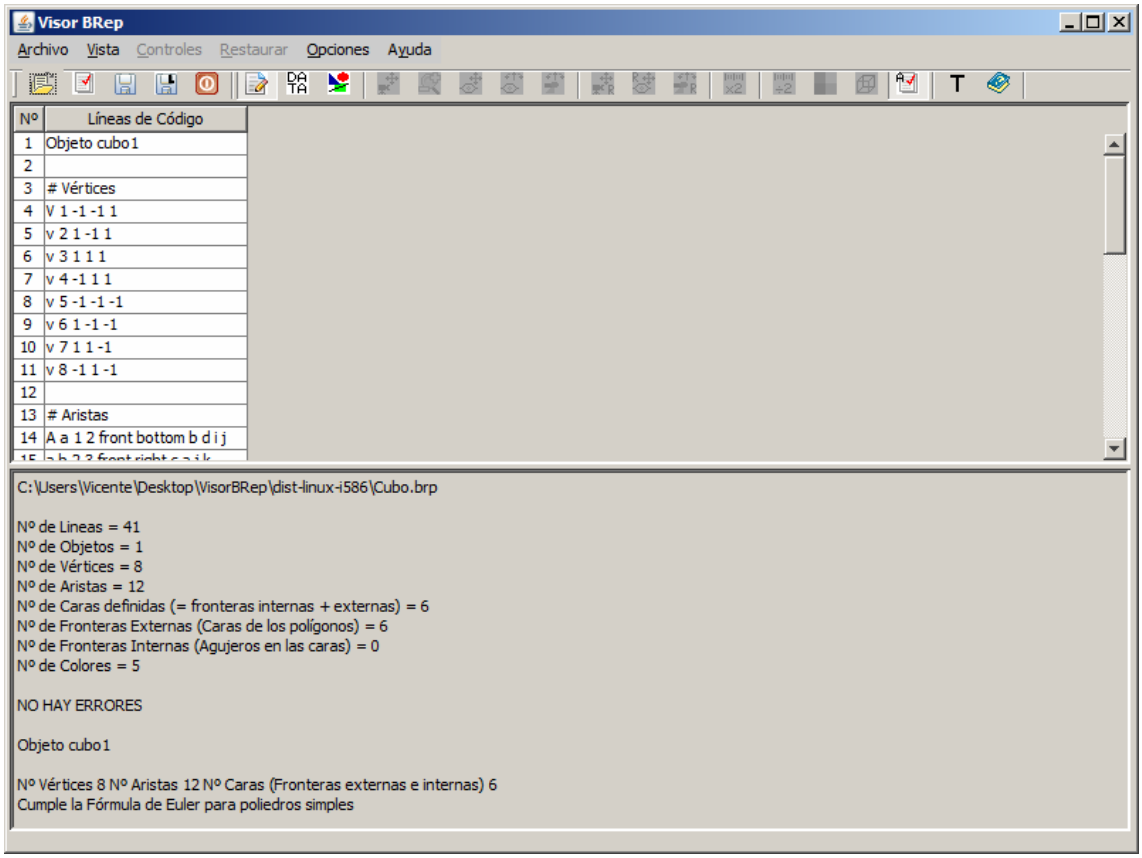

Figura 12. Vista Código

En esta vista, hay más botones activos, y se ve un panel dividido verticalmente en dos partes. En la parte superior tenemos el contenido del archivo. En la parte inferior, un log en el que se muestra el nombre (con la ruta completa) del archivo, una serie de estadísticas del mismo (número de líneas del texto, número de objetos, vértices, aristas, caras, fronteras internas y externas, y número de colores), junto con los errores que se hayan detectado. Después de esto, añade las estadísticas individuales de cada objeto y si cumple la ley de Euler de poliedros simples (sin agujeros) o la de Euler-Poincaré (de poliedros con agujeros).

Mientras que el código es editable, el log no lo es. La división entre ambos se puede desplazar.

Respecto a los botones de la barra de herramientas activados al abrir un archivo (y sus correspondientes opciones en el menú), tenemos uno para guardar el contenido del archivo (). Con este botón se accede a un diálogo que permite almacenar el archivo, tanto en formato texto como en formato BRep.

Hay otro botón () con el que podemos guardar el contenido del log (para analizarlo con más calma cuando aparecen muchos errores), pero éste solamente se puede guardar en formato texto.

También tenemos dos botones más que gestionan el análisis del fichero y que se comentan a continuación.

Por defecto está activado el análisis automático del archivo (). Esto significa que cuando se carga el archivo y cuando se modifica su contenido, se analiza el archivo y se muestran en el log los errores correspondientes.

Cuando hay pocos errores, esto no supone un problema, pero si hubiese muchos errores, el log se ampliaría mucho con cada edición, lo que haría difícil seguir el proceso de corrección.

Para ello, se puede desactivar el análisis automático, corregir todos los errores, y luego hacer un análisis manual con el botón correspondiente (), y activar a continuación, si se quiere, el análisis automático.

En otro grupo de botones tenemos los de las tres vistas: vista Código, vista Datos y vista Dibujo.

La vista Código () ya se ha comentado, ya que es la que se ve cuando se abre el archivo.

3.2.3 Vista Datos

En la vista Datos (DA TA), se muestran una serie de tablas con los datos del objeto (u objetos) definidos en el archivo.

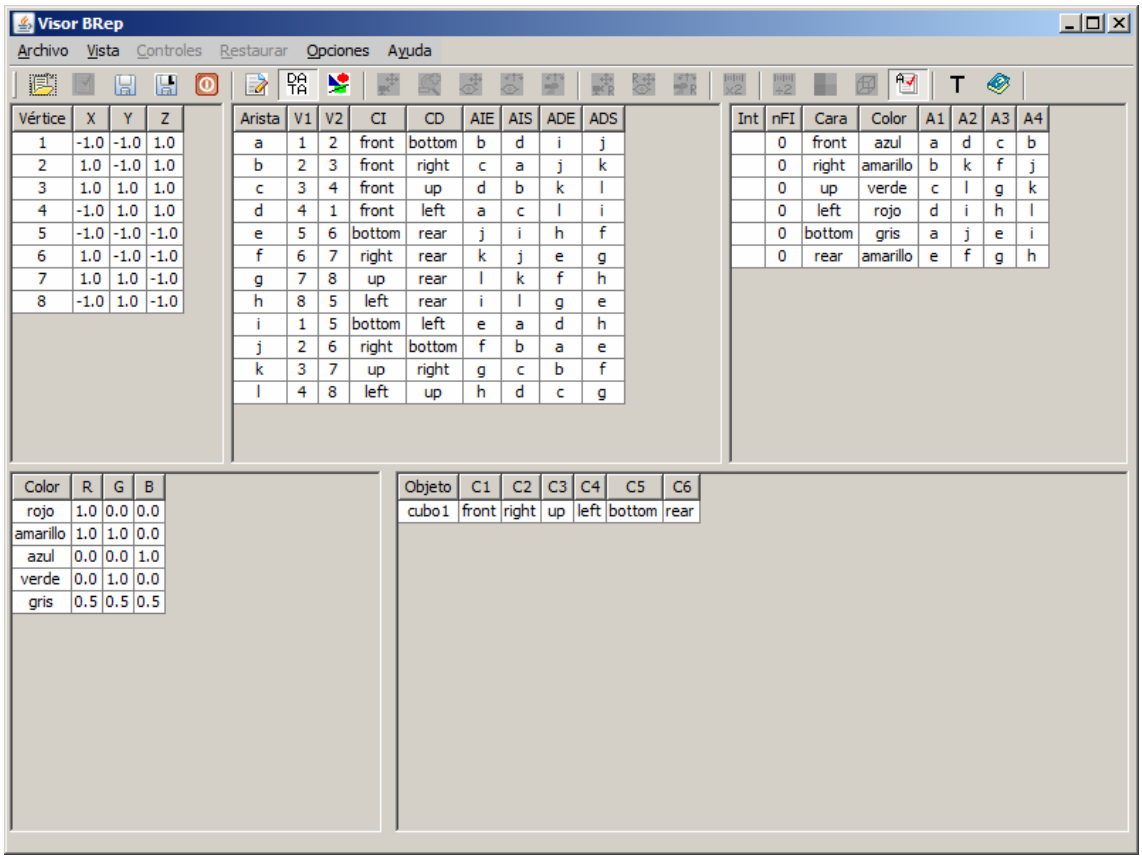

Figura 13. Vista Datos

En la tabla de vértices se muestran el nombre de los vértices y sus coordenadas (X, Y, Z).

En la tabla de aristas tenemos la información del formato de arista alada, es decir, los nombres de las aristas, los vértices inicial y final (V1 y V2), caras izquierda y derecha (CI y CD), y aristas izquierdas de entrada (AIE) y de salida (AIS), y aristas derechas de entrada (ADE) y de salida (ADS).

En la tabla de caras se indica si se trata de una frontera interna (Int), el número de fronteras internas que tiene esa cara (nFI), su nombre, color y la lista ordenada de aristas que la definen (en sentido horario vistas desde fuera para las caras o fronteras externas, y antihorario para las fronteras internas)

En la tabla de colores se muestra el nombre del color y sus componentes RGB.

Por último, en la tabla de objetos se muestra el nombre y la lista de caras (tanto fronteras internas como externas) que lo componen.

3.2.4 Vista Dibujo

La otra vista es la vista Dibujo () en la que se ve la representación gráfica de los objetos definidos en el archivo, junto con los semiejes cartesianos positivos X, Y, y Z, de colores rojo, verde y azul respectivamente.

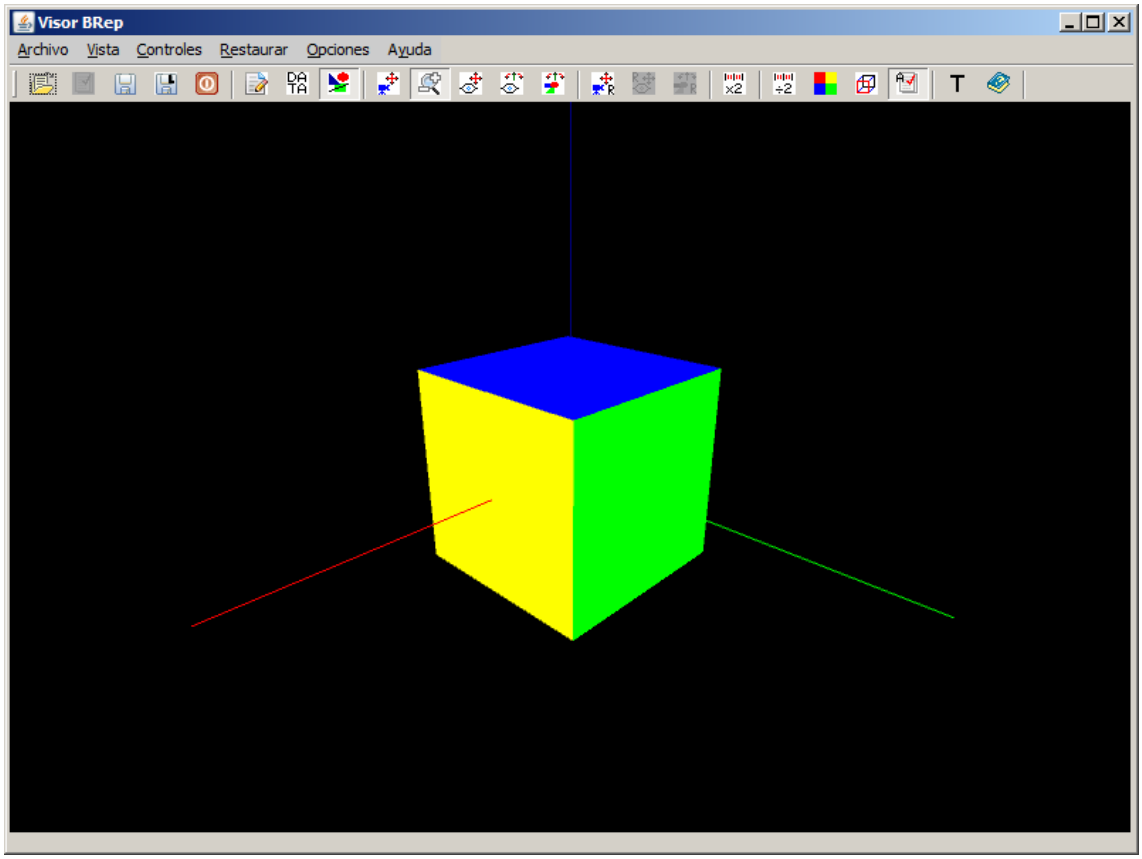

Figura 14. Vista Dibujo

Si se desactiva el análisis automático puede darse la circunstancia que el contenido de las tablas y de la vista Dibujo no se correspondan con el contenido del archivo (si se ha modificado y no se ha vuelto a analizar).

En todo momento las vistas Datos y Dibujo se corresponden con la última versión analizada sin errores. Es más, si el archivo contiene errores no se puede acceder a ninguna de estas dos vistas.

3.2.4.1 Controles de la representación gráfica

En la vista Dibujo están accesibles los botones que controlan la representación gráfica.

Por un lado tenemos el botón Posición Cámara () con el que se establece que el movimiento del ratón desplace vertical y horizontalmente la posición de la cámara, manteniendo su orientación.

El botón Zoom () provoca que el movimiento vertical del ratón desplace la cámara hacia delante o hacia atrás, manteniendo la orientación. Este movimiento da la sensación de estar haciendo un zoom, aunque en realidad mueve la cámara y podemos "atravesar" los objetos y verlos desde su interior.

El siguiente botón es el botón Mirar Hacia (), y éste provoca que con el movimiento del ratón se gire horizontal y verticalmente el vector hacia donde mira la cámara, manteniendo su posición.

El botón Ladear Cámara () provoca que al mover el ratón, la cámara gire alrededor del eje hacia donde mira, manteniendo la posición.

El botón Girar Objeto () hace que el objeto gire alrededor de dos de sus ejes cuando movemos horizontal y verticalmente el ratón. En realidad, lo que gira es la cámara alrededor del objeto, pero el efecto es el mismo.

Tanto si hay un objeto como varios, se calcula la media de todos los vértices, y ese "centro de masas" es el punto alrededor del cual se hacen los giros de la escena.

Después de realizar muchos giros y desplazamientos, podemos querer volver a la situación inicial. Por eso se han añadido tres botones más.

El botón Restaurar Posición Cámara y Zoom () devuelve a su configuración inicial el desplazamiento lateral y el zoom de la cámara.

El botón Restaurar Mirar Hacia y Ladeo () devuelve a su configuración inicial los giros del vector hacia donde mira la cámara, y el giro de la cámara alrededor de ese vector.

Por último, el botón Restaurar Girar Objeto () devuelve a su configuración inicial los giros efectuados sobre los objetos.

Los botones para restaurar la configuración inicial están habilitados en función del control que esté activado.

3.2.4.2 Otras opciones

Todos los movimientos de la representación gráfica se controlan con el ratón. Pero, dependiendo de los casos, podemos querer que esos cambios se produzcan más rápida o más lentamente.

Para eso se añadieron los botones Aumentar Sensibilidad Ratón () y Disminuir Sensibilidad Ratón (), que multiplican o dividen por dos el efecto que cada desplazamiento de un píxel produce en la imagen.

Los últimos botones dan control sobre ciertas opciones del programa.

El botón Color de Fondo () muestra un diálogo en el que se puede elegir el color de fondo de la imagen.

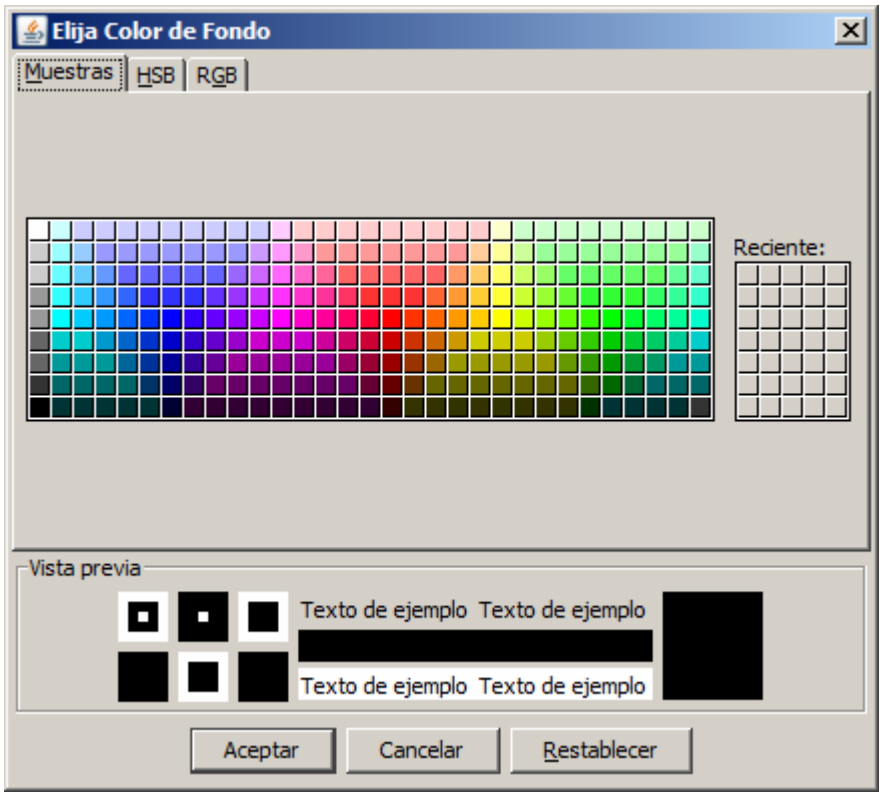

Figura 15. Diálogo para elegir el color de fondo

El botón de Vista Alámbrica ( ), representa los objetos solamente mediante sus aristas. En este caso, se dibujan todas las aristas de color rojo.

Para controlar algunos errores (como por ejemplo si las caras son polígonos planos o no) se realizan cálculos, tales como productos vectoriales y productos escalares, y se compara su resultado con cero. Como a veces no podremos dar las coordenadas con total precisión, puede darse el caso de que una operación matemática que debiera dar cero no lo dé, o viceversa. Por eso admitimos un pequeño margen de error en esos cálculos.

A este valor es al que llamamos tolerancia, y su valor se establece con el botón del mismo nombre (T).

En algunos casos, estos errores se producen y provocan que no se pueda acceder a la vista Dibujo. Modificando el valor de la tolerancia se corrige este problema, pero puede producirse otro.

Podemos tener una cara que en vez de ser plana, se haya definido como una superficie curva muy retorcida sobre sí misma. En las siguientes imágenes se ilustran tres posibles situaciones.

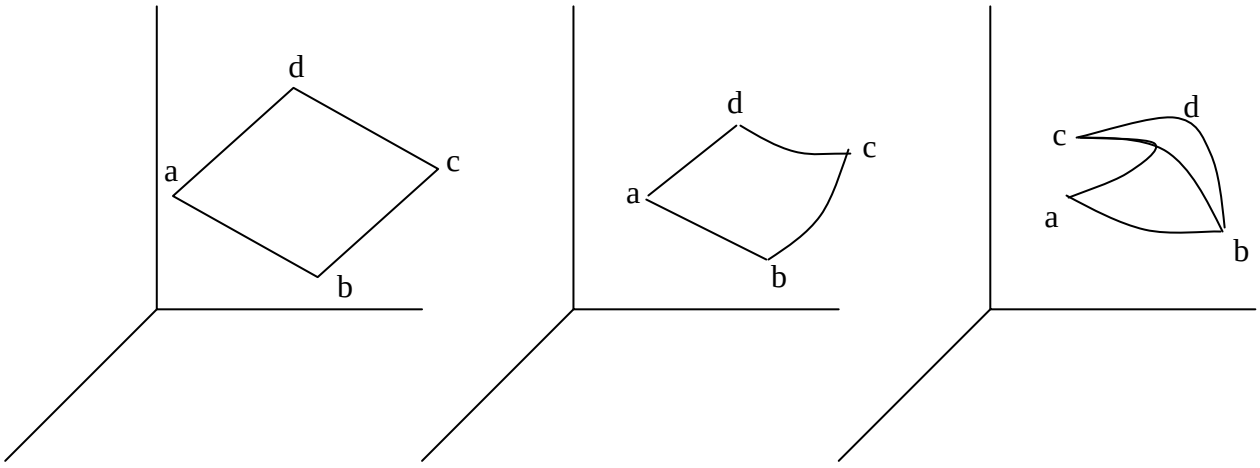

Figura 16. Limitaciones en los polígonos a representar

Para representar las caras, se utiliza la clase tessellator (que se comenta más adelante). Esta clase puede representar polígonos siempre que no se retuerzan demasiado sobre sí mismas.

Por eso, cuando variamos la tolerancia y tenemos alguna cara definida de esta forma, el programa se puede colapsar y cerrarse. La única solución en este caso es volver a arrancar el programa y trabajar sin modificar la tolerancia hasta que se detecte el error que produjo que se cerrara la aplicación.

3.2.5 Menú

Además de los botones que se encuentran en la barra de herramientas, tenemos la barra de menús. En ella tenemos todas las mismas opciones que con los botones.

Al igual que ocurre en la barra de herramientas, las opciones del menú están habilitadas o no en función del estado de la aplicación. Por ejemplo, si hay errores en el archivo, no se puede acceder a las vistas Datos y Dibujo.

Para movernos por el menú, tenemos varias opciones.

La más habitual es acceder con el ratón, haciendo clic sobre la opción elegida.

También se puede acceder al menú pulsando la tecla Alt. Con esto, entramos en el menú. Después, podemos movernos a través de él con los cursores, o pulsando las teclas de las letras subrayadas en la opción.

La última forma de acceder al menú es a través de teclas de acceso rápido o shorcuts. Se trata de combinaciones de teclas con la tecla de Control (Ctrl), y con la tecla de mayúsculas en algunos casos.

A continuación se muestran imágenes de todas las opciones del menú.

En la primera de ellas (opciones de Archivo) podemos observar lo que hemos comentado.

La opción Analizar Archivo está deshabilitada porque esta imagen se tomó en la vista Dibujo.

Para acceder solamente con el teclado, pulsaríamos la tecla Alt y nos moveríamos con los cursores. También podríamos pulsar la tecla A para entrar en el menú de Archivo, y luego podríamos acceder a Abrir Archivo pulsando otra vez la A, a Guardar Archivo pulsando la G, a Guardar Log Errores pulsando la E, o a Salir pulsando la S.

Por último, al lado de las opciones aparece la combinación de teclas de acceso rápido (Control + A para Abrir Archivo, Ctrl + N para Analizar Archivo, etc).

Figura 17. Opciones del menú Archivo

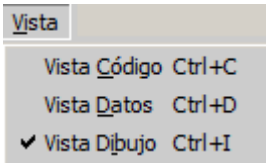

Figura 18. Opciones del menú Vista

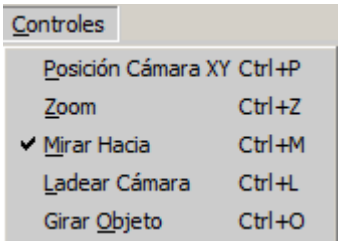

Figura 19. Opciones del menú Controles

Figura 20. Opciones del menú Restaurar

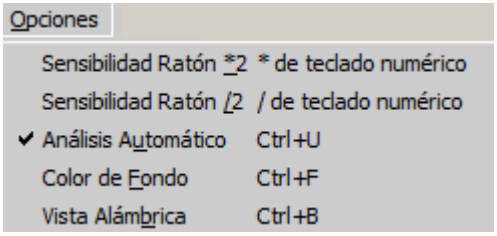

Figura 21. Opciones del menú Opciones

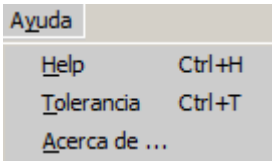

Figura 22. Opciones del menú Ayuda

3.2.6 Menú contextual

En vista Dibujo tenemos un menú contextual. Se accede a él pulsando el botón derecho del ratón.

Desde él se pueden acceder a las opciones más comunes de control sobre la representación gráfica.

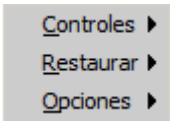

Figura 23. Menú contextual

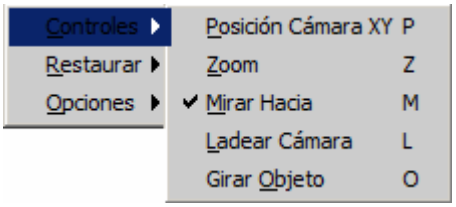

Figura 24. Menú contextual Controles

Figura 25. Menú contextual Restaurar

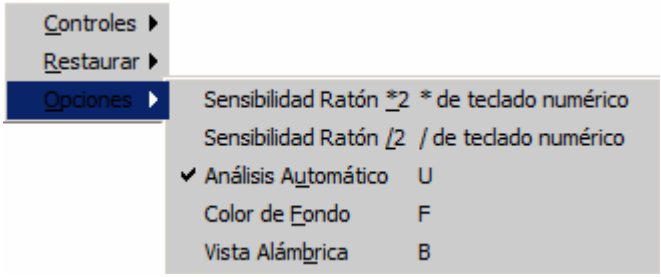

Figura 26. Menú contextual Opciones

3.3 Ayuda en el programa

El programa tiene una opción que abre la ayuda del programa () en un panel separado de la aplicación.

Este panel tiene 5 pestañas: características del programa, menús y shorcuts, controles de visión, control de errores y formato propio.

El contenido de las pestañas sobre las características del programa y los controles de visión ya se ha comentado.

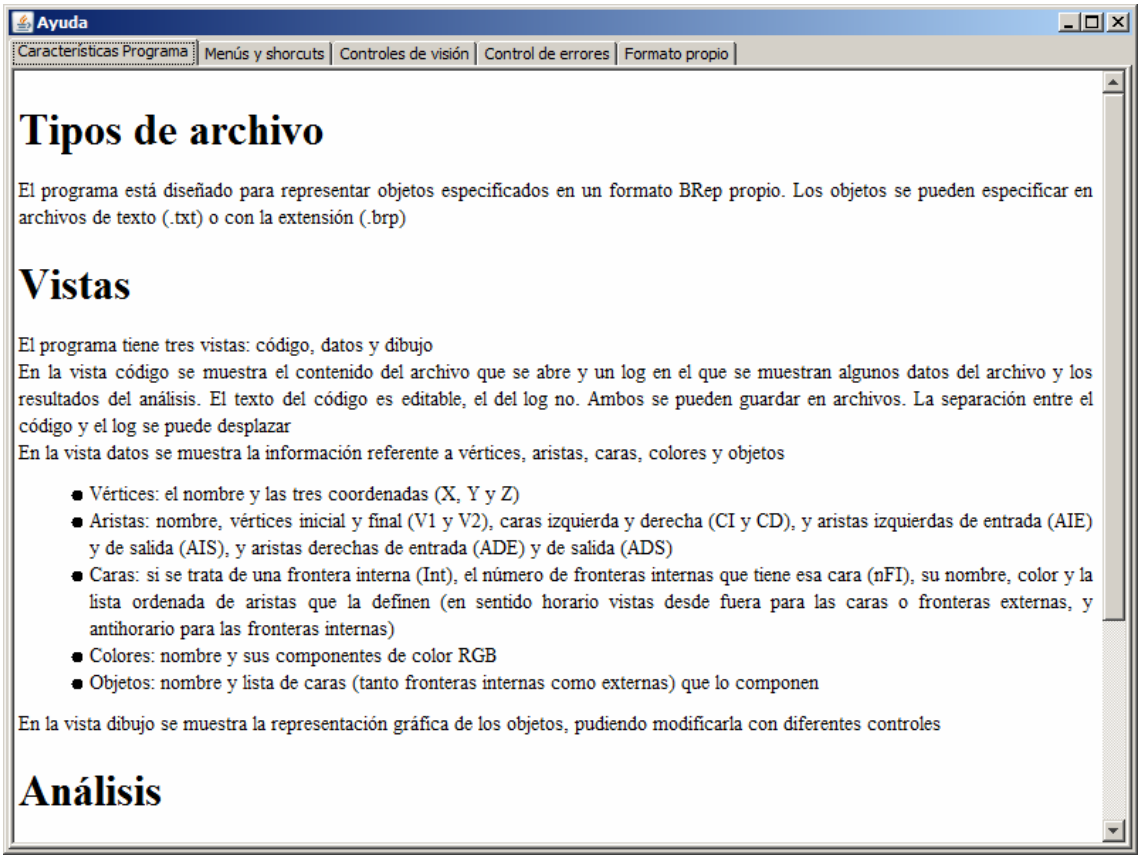

Figura 27. Ayuda Características programas

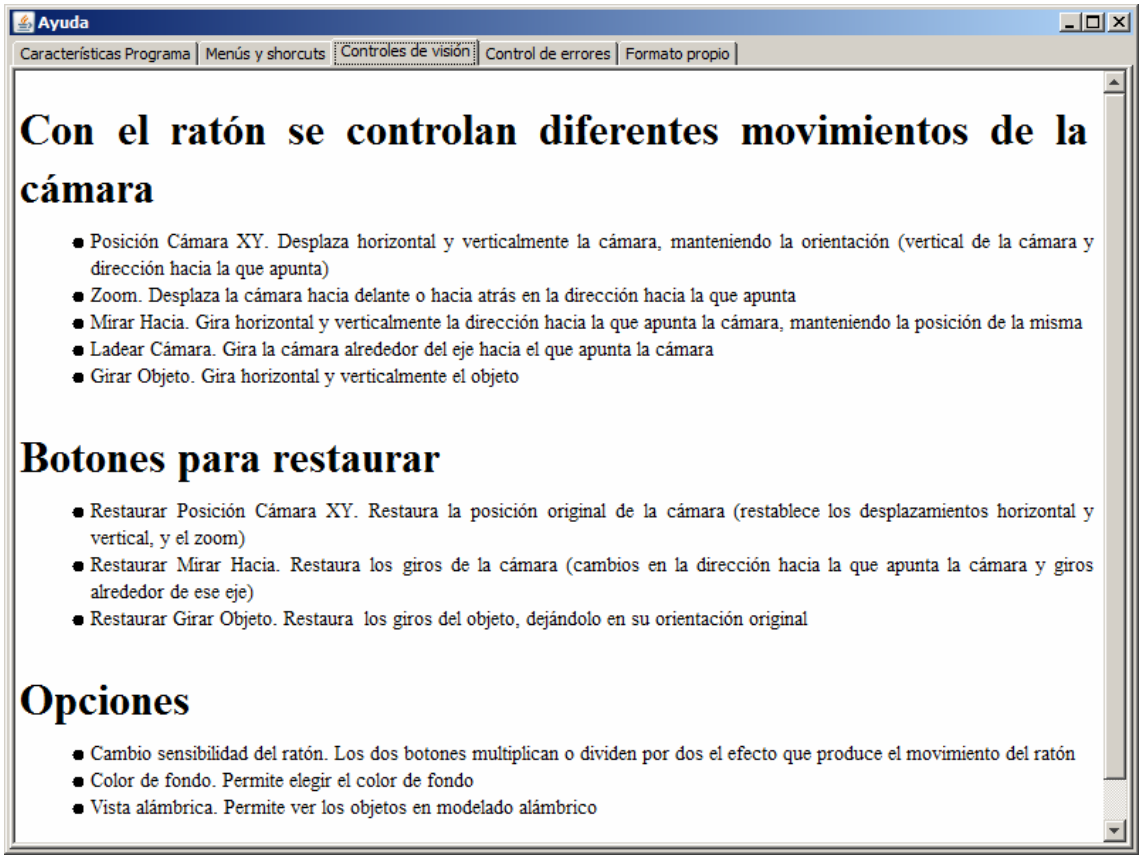

Figura 28. Ayuda Controles de visión

En la pestaña de menús y shorcuts tenemos una tabla con el listado completo de las teclas de acceso rápido.

ARCHIVO	
Ctrl + A	Abrir Archivo
Ctrl + N	Analizar Archivo
Ctrl + G	Guardar Archivo
Ctrl + E	Guardar Log Errores
Ctrl + S	Salir
VISTAS	
Ctrl + C	Vista Código
Ctrl + D	Vista Datos
Ctrl + I	Vista Dibujo

CONTROLES	
Ctrl + P	Posición Cámara XY
Ctrl + Z	Zoom
Ctrl + M	Mirar Hacia
Ctrl + L	Ladear Cámara
Ctrl + O	Girar Objeto
RESTAURAR	
Ctrl + Shift + P	Restaurar Posición Cámara
Ctrl + Shift + M	Restaurar Orientación Cámara
Ctrl + Shift + O	Restaurar Girar Objeto
OPCIONES	
*	Sensibilidad Ratón *2
/	Sensibilidad Ratón /2
Ctrl + U	Análisis Automático
Ctrl + F	Color Fondo
Ctrl + B	Vista Alámbrica
AYUDA	
Ctrl + H	Help
Ctrl + T	Tolerancia

Tabla 4. Lista de teclas de acceso rápido o shorcuts

En la pestaña de control de errores tenemos una lista con todos los errores que se comprueban cuando se analiza el archivo. Estos se comentarán más adelante.

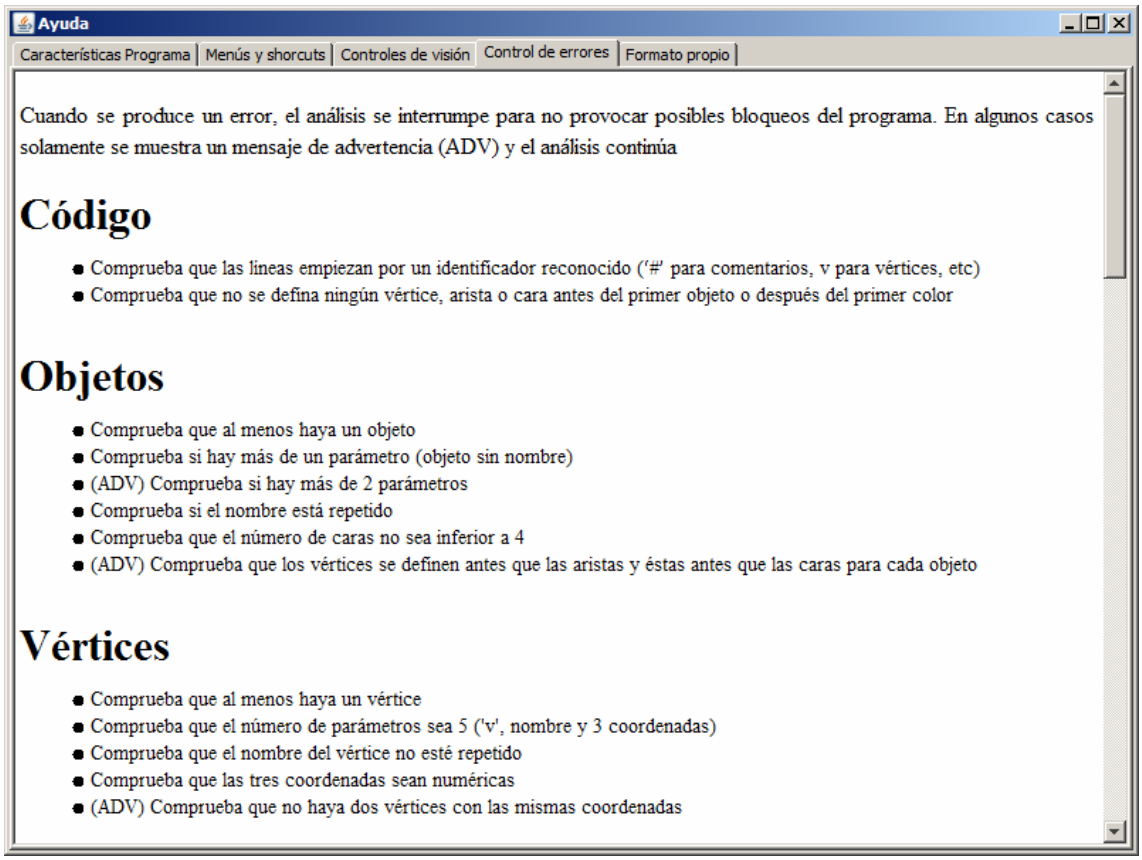

Figura 29. Ayuda Control de errores

La última pestaña de la ventana de ayuda es la del formato propio. En ella se especifica el formato BRep utilizado. También se comenta en otro apartado.

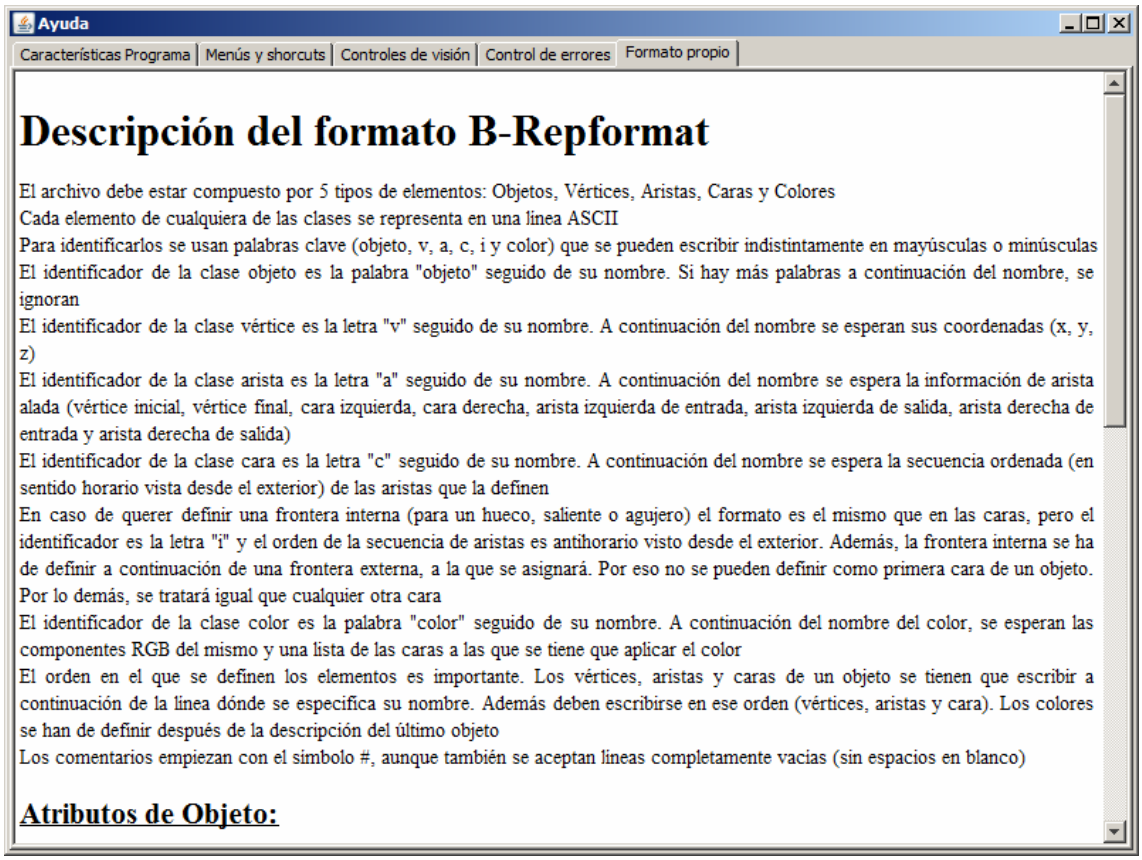

Figura 30. Ayuda Formato Propio

3.4 Algunas consideraciones sobre la ejecución del programa

3.4.1 Gestión de memoria

Como ya se ha comentado anteriormente, con cada análisis del archivo se crean (y previamente se destruyen si ya existían) las estructuras que recogen la información necesaria para la representación gráfica.

Para gestionar la memoria se usa el garbage collector de la Máquina Virtual de Java. Este proceso se ejecuta automáticamente, y lo que hace es controlar la memoria que ocupan los objetos (en el sentido del lenguaje Java, como instancias de las clases, no como objetos definidos en formato BRep). Si ya no se usan, se libera la memoria correspondiente.

El garbage collector se puede invocar, forzando su ejecución, aunque, como cualquier otro proceso, se ejecuta cuando el procesador le asigna su uso. En esta aplicación se invoca al garbage collector cuando se acaba de analizar el código, es decir, después de haber liberado (si hace falta) y reservado el espacio de memoria necesario.

De esta forma, si el garbage collector tarda en ejecutarse, la cantidad de memoria necesaria puede aumentar, pero en cuanto es posible, se vuelve a liberar, y esto "garantiza" que los recursos necesarios no se van a disparar.

3.4.2 La clase tessellator

Las librerías gráficas OpenGL ofrecen varias funciones para dibujar distintas formas geométricas, tanto 2D como 3D.

Habitualmente se usa la función glPolygon, pero solamente dibuja polígonos convexos. Para poder dibujar polígonos cóncavos tenemos que utilizar la clase tessellator.

Esta clase lo que hace es dibujar el polígono como una combinación de triángulos, permitiendo de esta forma dibujar polígonos convexos, aunque como se comentó anteriormente, tiene sus limitaciones cuando se representa una superficie curvada.

3.5 Instalación y requisitos de la aplicación

La aplicación no necesita ninguna instalación especial. Los archivos se pueden situar en cualquier directorio y ejecutarse directamente.

Para que una aplicación que usa las librerías JOGL funcione, necesita dos cosas.

La primera es que esté instalada en el sistema la máquina virtual de Java (JVM) en su versión 1.4.2 o posterior. La JVM forma parte del JRE o Java Runtime Environment.

De la Máquina Virtual Java hay muchas implementaciones. La que se ha elegido para generar la aplicación y para documentarla, es la de Sun (hoy en día, Sun forma parte de Oracle), por su gran compatibilidad.

Se puede descargar desde la página de Oracle. En esta página se insiste en instalar la última versión (6.0) tras desinstalar previamente las versiones anteriores. Por

eso resulta difícil encontrar información sobre requisitos al instalar versiones anteriores y se comentan los requisitos de la última versión, que no son muy exigentes.

A continuación se describen los requisitos para los sistemas operativos windows y linux.

Plataforma	Versión	Memoria	Espacio en HD
Windows (32-bit)	Windows 7 Windows Vista Windows Server 2008 Windows Server 2003	128 MB	98 MB
	Windows XP Windows 2000	64 MB	98 MB
Windows x64 32-bit mode	Windows 7 Windows Server 2008 Windows vista Windows XP Windows Server 2003	128 MB	98 MB
Windows x64 64-bit mode	Windows 7 Windows Server 2008 Windows Vista Windows XP Windows 2003 Standard Edition (R2, SP1, SP2)	128 MB	98 MB
Linux x86 32-bit		64 MB	58 MB
Linux x64 64-bit		64 MB	58 MB

Tabla 5. Requisitos para la Máquina Virtual de Java (JVM)

El JRE 1.4.2 (la versión mínima del JRE y de la JVM necesaria para ejecutar JOGL) necesita windows 98 como mínimo, procesador pentium a 166 MHz y 32 MB RAM (recomendados 48 MB).

JRE 1.4 se publicó en febrero de 2002 y la siguiente versión (5.0) en mayo de 2004. La fecha de aparición de la versión 1.4.2 del JRE, posiblemente sea el 5 de mayo de 2003.

Por otro lado, la JVM reserva una cantidad de memoria determinada, normalmente lo que especifica en sus requisitos. Por ejemplo, para windows 7, reserva 128 MB. Si necesita más espacio, aumenta su espacio reservado.

Como prueba, se hizo funcionar el programa dibujando un cubo, y dibujando 10 cubos. La diferencia no se apreció, ya que la cantidad de memoria usada por la JVM incluye espacio suficiente para esos dos casos.

Por eso presentamos los siguientes cálculos aproximados para establecer el consumo de memoria.

Haciendo cuentas aproximadamente (tomando 4 Bytes para representar un entero, un carácter, un número real y un puntero, que son las cantidades usadas por el ordenador de desarrollo), tenemos los siguientes espacios necesarios para cada estructura (añadiendo un puntero para apuntar al elemento).

Elemento	Componentes	Tamaño total en bytes
Arista	10 int 1 vectores 10 String	618
Cara	10 int 1 double 6 string 1 vectores	464
Colores	1 int 3 float 1 string	60
Objeto	7 int 2 string	112
Vectores	10 float	44
Vertice	2 int 3 float 2 string	104

Tabla 6. Tamaño aproximado de las estructuras que se generan

Como se puede ver, la cantidad de memoria para representar los elementos es muy pequeña, apenas media KB para una arista.

Si suponemos un poliedro de 1000 caras triangulares tendríamos unos 3000 vértices, aristas y colores, 1000 caras y un objeto, que sumando de forma aproximada, da menos de 3 MB. Por lo tanto, con las cantidades de memoria RAM que tienen los ordenadores actuales, no supone ningún problema el uso del programa.

Lo que sí que hay que tener en cuenta es que el programa llama repetidamente al garbage collector de la máquina virtual de java (el proceso que se encarga de liberar la memoria que no se usa). Si la aplicación está ejecutando otras tareas, puede que el garbage collector tarde en ejecutarse y la cantidad de memoria aumente temporalmente, pero con el uso normal (si la memoria del ordenador no está saturada), la cantidad de memoria requerida no varía significativamente (en el ordenador portátil menos potente que se usó, la memoria usada no supuso ningún problema – se comenta más adelante-).

La otra condición necesaria para las aplicaciones que usan las librerías JOGL es que la tarjeta gráfica soporte directamente las librerías gráficas OpenGL en su versión 1.1 o superior.

Existen programas para determinar la capacidad de la tarjeta gráfica. Uno de ellos (el usado en el proyecto) es OpenGL Viewer (que tiene versiones para Mac, Windows XP, Vista 7 o posterior), en la tienda de Itunes y en el Android Market, aunque tiene el inconveniente de tener que instalarse.

Como se puede ver, los requisitos no son nada exigentes. OpenGL 1.1 se publicó en enero de 1997, y desde aquella fecha se incorporó en la mayoría de tarjetas gráficas que se construyeron.

Dentro de las pruebas, y para comprobar su ejecución en un ordenador lo menos potente posible, se probó, funcionando correctamente, en un portátil Packard Bell EasyNote E, con tarjeta gráfica S3 Pro SavageDDR de 32 MB de memoria gráfica (que soporta openGL 1.1 completamente, y solo un 12% de las core features de openGL 1.2), con 224 MB de memoria RAM, procesador Mobile AMD Athlon XP de 1800 MHz, y sistema operativo Windows XP. La aplicación solamente usaba 44 MB de la memoria RAM disponible.

También llegó a funcionar en un portátil Toshiba Satellite 4070 CDT con 64 MB de RAM, procesador Intel Celeron-A, a 188 MHZ Mendocino, y con sistema operativo Windows 98, aunque las imágenes se veían invertidas verticalmente (lo de arriba se veía abajo y viceversa), posiblemente debido al sistema de referencia de las ventanas.

Como se puede observar, el equipo necesario no es nada potente.

3.6 Versiones de la aplicación según el sistema operativo

Como se ha comentado anteriormente, a pesar de tratarse de una aplicación Java, el programa hace uso de librerías nativas, y eso provoca que se generen diferentes versiones del programa.

En concreto se generan las siguientes:

- Windows-amd64
- Windows-i586
- Linux-amd64
- Linux-i586
- Solaris-i586
- Solaris-sparc
- Solaris-sparcv9
- Macosx-ppc
- Macosx-universal

Las versiones i586 indican que trabajan sobre procesadores compatibles con la arquitectura x86 de 32 bits, es decir, procesadores Intel de 32 bits.

Las versiones amd64 indican que trabajan sobre procesadores compatibles con la arquitectura x86 de 64 bits.

El sistema operativo Solaris fue desarrollado por Sun (ahora parte de Oracle), es decir, los mismos creadores de Java. Por eso encontramos más versiones para este sistema operativo.

Además de la versión para microprocesadores x86 de 32 bits, tenemos la versión para microprocesadores SPARC, en su denominación sparc (versiones de la arquitectura hasta la 8, es decir, de 32 bits) y sparcv9 (versiones de la arquitectura a partir de la 9, es decir, de 64 bits).

Las versiones para ordenadores MAC son la ppc, que trabajan sobre procesadores Power PC, y la universal, que trabaja sobre procesadores con la arquitectura x86.

La elección de la versión de 32 ó 64 bits de la aplicación, depende de la versión de la Máquina Virtual Java instalada.

4 Control de errores

Cuando abrimos un fichero, el programa analiza su contenido. Lo mismo ocurre cuando se ejecuta un análisis manual, o cuando se hace cualquier modificación del código y el análisis automático está habilitado.

El análisis de los errores se hace en una secuencia determinada. Si se produce algún error, el análisis se detiene y se muestran los errores en el log de la vista código.

Esto se hace así para que la aplicación no se bloquee o se cierre. Si se produjese un error, éste puede afectar a la comprobación de errores posteriores. Por ejemplo, si no se definiese ninguna cara, se produciría un error al intentar acceder a la lista de caras. En ese caso, la aplicación dejaría de responder. Por eso no se continúa comprobando más errores.

En otros casos, los fallos detectados no son importantes y se consideran solamente como advertencias. Se muestra el mensaje correspondiente en el log de errores con la marca (ADV), pero el proceso continúa.

A continuación se detallan y se comentan los errores que se controlan. Si los comentarios no tienen la etiqueta (ADV) significa que son errores que interrumpen el análisis.

4.1 Código

Se eliminan los espacios en blanco duplicados, ya que pueden dar problemas al identificar los items (datos o palabras) que componen la frase. Esto no aparece como un mensaje de error, sino que simplemente se realiza con todas las líneas.

Se comprueba que las líneas empiezan por un identificador reconocido (‘#’ para comentarios, v para vértices, etc) .

Se comprueba que no se defina ningún vértice, arista o cara antes del primer objeto o después del primer color.

4.2 Objetos

Se comprueba que al menos haya un objeto (en todo el archivo).

Se comprueba si hay más de un parámetro (objeto sin nombre).

(ADV) Se comprueba si hay más de 2 parámetros. En este caso se habría escrito alguna palabra detrás del nombre, y sencillamente, no se tiene en cuenta.

Se comprueba si el nombre está repetido. Es importante que no haya nombres repetidos, ya que de lo contrario el programa asigna los elementos (vértices, aristas, etc), al primer objeto definido con ese nombre, y provocaría errores graves.

Se comprueba que el número de caras no sea inferior a 4, es decir, como mínimo debe tratarse de un tetraedro.

(ADV) Se comprueba que los vértices se definen antes que las aristas y éstas antes que las caras para cada objeto. Lo recomendable es mantener el orden que establece el formato, pero dentro de cada objeto no hay duda a qué objeto pertenece los elementos.

4.3 Vértices

Se comprueba que al menos haya un vértice (en todo el archivo). Si faltase en algún objeto se produciría otro error al analizar las aristas).

Se comprueba que el número de parámetros sea exactamente 5 ('v', nombre y 3 coordenadas).

Se comprueba que el nombre del vértice no esté repetido.

Se comprueba que las tres coordenadas sean numéricas.

Se comprueba que no haya dos vértices con las mismas coordenadas. Si se permitiera esto, podríamos modelar objetos no múltiples, es decir, que coincidieran dos aristas, y en el formato BRep cada arista separa dos caras.

4.4 Aristas

Se comprueba que al menos haya una arista.

Se comprueba si el nombre de la arista está repetido.

Se comprueba que exista el vértice inicial.

Se comprueba que exista el vértice final.

Se comprueba que el número de parámetros sea 10 ('a', nombre de la arista, caras izquierda y derecha, y las aristas entrantes y salientes de ambas caras).

Se comprueba que la cara izquierda existe.

Se comprueba que la cara derecha existe.

Se comprueba que la cara izquierda y derecha no son la misma.

Se comprueba que las 4 aristas de entrada y salida existen.

Se comprueba que ninguna de las 4 aristas de entrada o salida coinciden con la arista que se define.

Se comprueba que ninguna de las 4 aristas del final no coinciden entre sí.

Se comprueba que una arista y la siguiente tienen un vértice en común.

Se comprueba que la arista existe en la cara de la derecha.

Se comprueba que la arista saliente es la siguiente en la cara de la derecha.

Se comprueba que la arista entrante es la anterior en la cara de la derecha.

Se comprueba que la arista existe en la cara de la izquierda.

Se comprueba que la arista saliente es la siguiente en la cara de la izquierda.

Se comprueba que la arista entrante es la siguiente en la cara de la izquierda.

Se comprueba que no haya dos aristas coincidentes, que tengan los mismos vértices (ambos).

Se comprueba que no haya ninguna arista incluida en otra o que compartan un mismo segmento de las mismas (objetos múltiples).

Indirectamente, se comprueba que una arista solamente está en dos caras, ya que antes de llegar a este extremo, se producen otros errores.

4.5 Caras

Se comprueba que al menos haya una cara.

Se comprueba que haya al menos 5 parámetros ('c', nombre de la cara y al menos 3 aristas).

Se comprueba que el nombre de la cara no esté repetido.

Se comprueba que las aristas de la cara existan.

Se comprueba que las aristas no se repitan en la definición de la cara.

Se comprueba que la primera cara que se define para un objeto no sea una frontera interna (no tendría frontera externa asignada correctamente).

(ADV) Se comprueba que no haya dos vértices repetidos en una misma cara.

Se comprueba que las caras sean planas. El mensaje indica qué vértices no son coplanares con el plano que define la cara, pero hay que tener en cuenta las aristas y vértices que se usan para definir ese plano.

Se comprueba que todas las aristas de una cara no estén en la misma recta (no se trataría de un polígono; teóricamente, este error no se puede dar, ya que las aristas se superpondrían unas a otras y ese error se comprueba antes).

Indirectamente se comprueba que se trata de polígonos cerrados, ya que si dos aristas consecutivas no tienen ningún vértice en común, se produce ese error.

4.6 Colores

Se comprueba que al menos haya un color.

Se comprueba que el número de parámetros sea mayor de 5 ('color', nombre, componentes RGB y nombre de al menos una cara).

Se comprueba que el color no esté repetido.

Se comprueba que el tercer, cuarto y quinto parámetros (RGB) sean numéricos.

Se comprueba que el tercer, cuarto y quinto parámetros (RGB) estén comprendidos entre 0.0f y 1.0f.

Se comprueba que las caras a las que se aplica el color existan.

Se comprueba que no haya ninguna cara que no tenga color asignado.

4.7 Otros comentarios

Cuando representamos paralelepípedos con sus aristas paralelas a los ejes, situar los vértices de las caras en un mismo plano, resulta relativamente sencillo.

Pero si queremos representar caras que estén inclinadas respecto a esos ejes, puede resultar complicado dar las coordenadas adecuadas. Por eso, en algunos cálculos se permite una cierta tolerancia. Como ya se ha comentado anteriormente, hay una opción del menú y un botón que permiten modificar ese valor. Por defecto vale 1E-05 (0,00001 unidades). Aunque la correspondencia no es exacta (se usa en comparaciones directas de valores, pero también en productos escalares y vectoriales), la tolerancia coincide aproximadamente con la diferencia de coordenadas que provocaría un error. Es decir, si un vértice lo desplazamos 1E-05 unidades, provocará (o anulará) un error.

Esto puede provocar dos efectos negativos.

El programa se puede cerrar de repente si se intenta dibujar un polígono en el que las aristas se cortan entre sí, o si dibujamos caras que no sean planas y se retuerzan mucho en el espacio. Estos posibles errores se comprueban dentro de la lógica del programa e interrumpen el análisis sin dibujar la figura, pero en el punto anterior hemos comentado el uso de la tolerancia. Esto significa que si variamos el valor de la tolerancia, algunos errores que se comprueban, dejarán de hacerlo (podría considerar aristas superpuestas como no superpuestas, o lo que es peor, que polígonos que no son planos, considere que sí lo son), con la posibilidad de que el programa se cierre.

El otro efecto consiste en que pueden aparecer errores que no se correspondan con la situación real de la definición de los objetos.

En estos casos, la única solución es hacer un uso adecuado de la ayuda que supone la tolerancia.

Un error que no se comprueba es si las caras se cortan entre sí. Esto da pie a diseñar objetos que no se puedan fabricar, pero el programa permite dibujarlos. Sin embargo, este hecho provoca una posibilidad de representación errónea. Si situamos todos los vértices de una frontera interior fuera de la frontera externa a la que pertenece, y en el mismo plano (si no es así, provocaría un error de cara no plana o de aristas que se cortan), el objeto se representa, pero el agujero se ve como una cara externa enlazada al objeto al que pertenece por el agujero que definía. En estos casos, es el usuario el que debe corregir las coordenadas de los vértices para una correcta representación

La situación se ilustra en la siguiente figura.

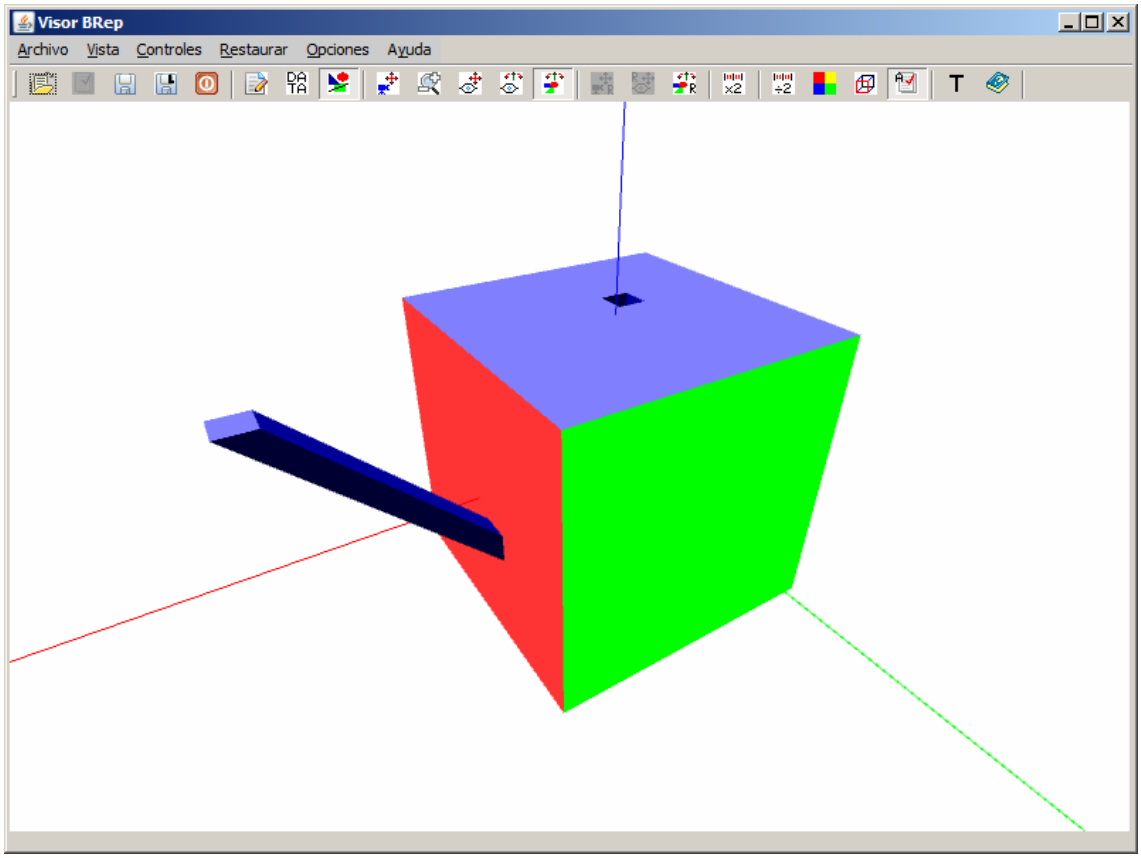

Figura 31. Poliedro con una frontera interna definida fuera de su cara

La figura representa un cubo con dos agujeros que lo atraviesan desde la cara superior azul, a la cara inferior.

Tal como se observa en la figura, una de las fronteras internas de la cara azul, se ha definido fuera de esta cara. La aplicación la representa de esta forma.

Si solamente se hubiese definido un vértice (o varios de la frontera interna), se produciría un error porque se cortarían aristas entre sí, o el vértice estaría definido fuera del plano de la cara. Pero en este caso, los cuatro vértices de la frontera interna se definen fuera de la cara y están en el mismo plano.

La aplicación lo permite, y es responsabilidad del usuario hacer un uso apropiado de sus diseños.

4.8 Descripción de cómo se han comprobado algunos errores

4.8.1 Aristas coincidentes

Esta comprobación es muy sencilla, ya que solamente hay que comprobar que los vértices inicial y final coinciden.

Pero se ha de tener la precaución de comparar los vértices iniciales entre sí y los vértices finales entre sí, y luego comprobar los vértices cruzados, vértice inicial con final de la otra arista y viceversa.

4.8.2 Aristas paralelas

Para comprobar si dos aristas son paralelas, se usan sus vectores directores (vectores unitarios).

Se calcula el producto escalar de ambos vectores. Si el resultado es positivo y mayor que la tolerancia, significa que los dos vectores apuntan hacia la misma mitad del espacio.

En ese caso se compara la diferencia de coordenadas de los vectores (hay que recordar que están normalizados). Si esta diferencia es menor que la tolerancia, se considera que son aristas paralelas.

En caso de que el producto escalar sea negativo, significa que los vectores apuntan hacia partes diferentes del espacio, y lo que se compara con la tolerancia es la suma de sus componentes.

Si el producto escalar es nulo, son perpendiculares.

4.8.3 Aristas superpuestas

Esta función está implementada en la clase Vectores. Las instancias de esta clase se crean a partir de una arista y se genera el vector director de la misma. Recoge la información de los vértices inicial y final de la arista, y del vector director de la misma.

Para diferenciar las dos aristas que vamos a ver si se superponen, nos referiremos a ellas como la primera y la segunda arista.

En primer lugar se comprueba si las aristas son paralelas. Si no lo fuesen, no podría tratarse de aristas superpuestas. Para ello, se calcula el producto vectorial de los vectores directores de ambas aristas. Si el producto vectorial resultante es mayor que un cierto error (tolerancia) no son paralelas.

En el caso de que se trate de vectores paralelos, comprobamos si están en la misma línea. Construimos dos nuevos vectores que unen el punto inicial de una de las aristas con los extremos de la otra arista.

Se podría elegir el segundo punto de la primera arista, o tomar cualquiera de los puntos de la segunda arista para unirlos con los dos vértices de la primera arista.

Multiplicamos vectorialmente estos dos nuevos vectores con el vector director de la primera arista. Si alguno de estos productos es distinto de cero (mayor que la tolerancia) significa que las aristas no están situadas sobre la misma línea.

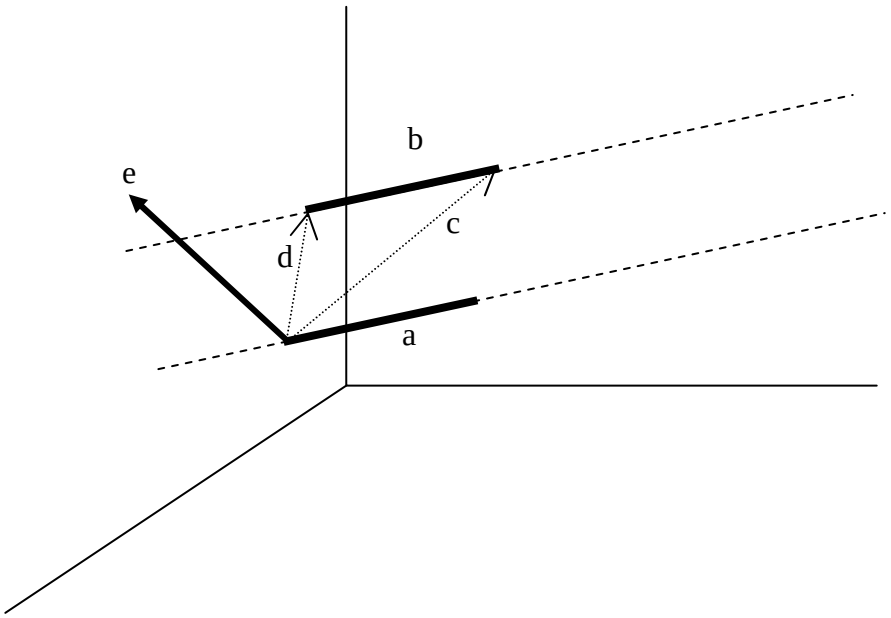

Figura 32. Aristas situadas sobre líneas paralelas diferentes

Tal como se ve en la figura anterior el producto vectorial del vector director de la primera arista (a) por los vectores que hemos generado (c y d) nos da un vector perpendicular al plano que contiene a las aristas a y b, y a los vectores c y d, es decir, paralelo al vector e.

Si las aristas a y b estuviesen sobre la misma línea, los vectores c y d también se situarían sobre esa recta, y todos los productos vectoriales darían el vector nulo.

En la siguiente figura se representa esta situación, con los vectores c y d desplazados ligeramente para apreciarlos mejor.

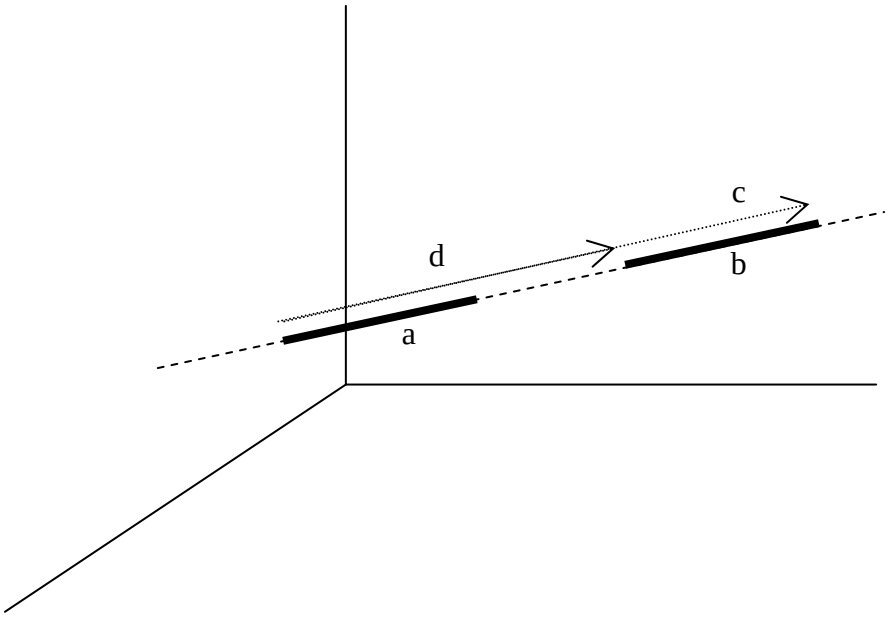

Figura 33. Aristas paralelas situadas sobre la misma línea

Tenemos que comprobar los dos extremos, ya que podría darse el caso de que uno de ellos estuviese en la recta en la que está definida la primera arista.

Si se trata de una arista que va antes o después en la secuencia de aristas, uno de sus vértices coincide con uno de los vértices de la primera arista, pero también podría darse el caso de que simplemente estuviese en la misma recta.

La siguiente figura ilustra esta situación.

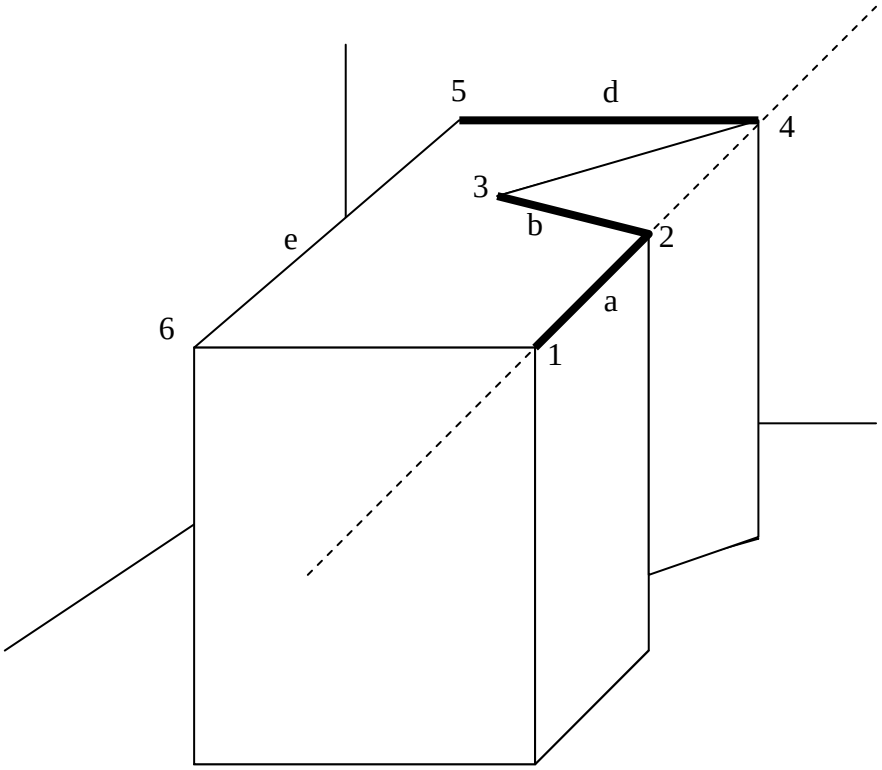

Figura 34. Aristas que podrían dar problemas si se tuviese en cuenta solamente uno de sus vértices

Como se puede apreciar en la figura, si partimos de la arista a y solamente comprobamos uno de los vértices de las otras aristas para ver si están sobre la misma línea, podría producirse un error en el caso de que se tomase el vértice 2 de la arista b, o el vértice 4 de la arista d, y considerar que están sobre la misma recta, cuando en realidad no es así.

Si tomamos en cuenta los dos vértices, el producto vectorial del vector director de la arista a por el vector que une el vértice 1 (o el 2) y el vértice 3 no es nulo. Con la arista d, el producto vectorial no nulo sería el del vector director de la arista con el vector que une el vértice 5 con el 1 (o con el 2).

En el caso de tratarse de aristas paralelas, pero situadas sobre líneas diferentes (como en el caso de la arista a y e), el resultado que se obtiene es correcto, ya que cualquiera de los vectores generados con las parejas de vértices 1-5, 1-6, 2-5 ó 2-6 da un producto vectorial distinto de cero, y por tanto se comprueba que no están situadas sobre la misma recta.

Una vez comprobado que las dos aristas están sobre la misma recta, solamente queda por determinar si las aristas se superponen o están separadas.

Esto se hace sencillamente tomando la primera arista (se podría tomar la segunda) y comprobando que las coordenadas del primer vértice de la segunda arista está situado entre los vértices de la primera. También se ha de comprobar el segundo vértice de la segunda arista. Si se da uno de estos dos casos, las aristas se superponen.

Aquí se plantea un problema. Esta comprobación se realiza con una sola coordenada. Si la arista fuese paralela a uno de los ejes del sistema de referencia (X, Y, Z), una (o hasta dos si están en uno de los planos XY, XZ o YZ) serían la misma para todos los puntos que analizamos (vértices inicial y final de ambas aristas).

Por eso, esta comprobación de si algún vértice de una arista está situado entre los de la otra arista, se ha de realizar para las tres coordenadas.

4.8.4 Cálculo del vector normal a las caras

Para calcular la normal a una cara se toma la primera arista y se busca la siguiente arista que no sea paralela a ella. Se podría dar el caso de un polígono en el que dos o más aristas consecutivas sean paralelas. Un ejemplo sería el siguiente polígono, en el que las aristas a (que va del vértice 1 al 2) y b (que va del vértice 2 al 3) están en la misma recta

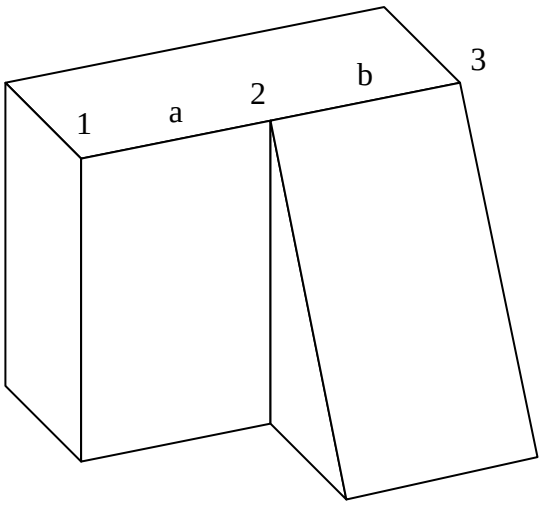

Figura 35. Aristas consecutivas en una misma cara

Si tomásemos esas dos aristas para calcular la normal, el producto vectorial daría el vector nulo, provocando otros errores más adelante.

Es evidente que debe haber al menos una arista (en realidad 2 es el número mínimo) que no sea paralela a la primera, ya que si todas fuesen paralelas, significaría que en vez de tener un polígono, tendríamos todas las aristas sobre una misma línea, y antes de llegar a este punto, se comprueba que no haya aristas superpuestas.

4.8.5 Valor del plano

Esto no es un error que se comprueba, sino una función que se utiliza para comprobar varios errores.

La ecuación que define un plano tiene la forma

$$Ax + By + Cz + D = 0$$

Los coeficientes A, B y C se corresponden con las componentes de un vector perpendicular al plano. Este vector sería común a todos los planos paralelos entre sí. El coeficiente D representa la posición en la que se encuentra el plano.

De esta forma, las ecuaciones siguientes

$$Ax + By + Cz + D_1 = 0$$

$$Ax + By + Cz + D_2 = 0$$

$$Ax + By + Cz + D_3 = 0$$

Definen diferentes planos paralelos, tal como se representa en la figura siguiente (siendo los valores D_1, D_2, D_3, etc, diferentes)

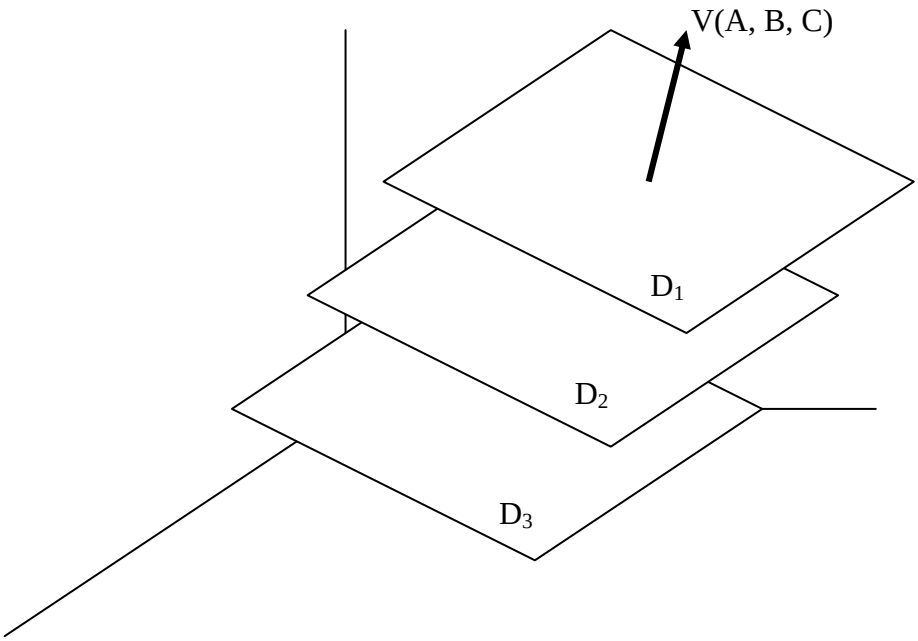

Figura 36. Ecuación del plano

Esta ecuación se puede transformar en una función

$$F(x, y, z) = Ax + By + Cz + D$$

De esta forma, si calculamos el valor de la función de un plano para un punto cualquiera, podemos saber si pertenece al plano o no. Si el valor de la función es cero, pertenece al plano. Si es positivo estará a un lado del plano, y si es negativo, al otro lado.

Esta función se calculará para cada cara y nos servirá para determinar si las caras son planas o no.

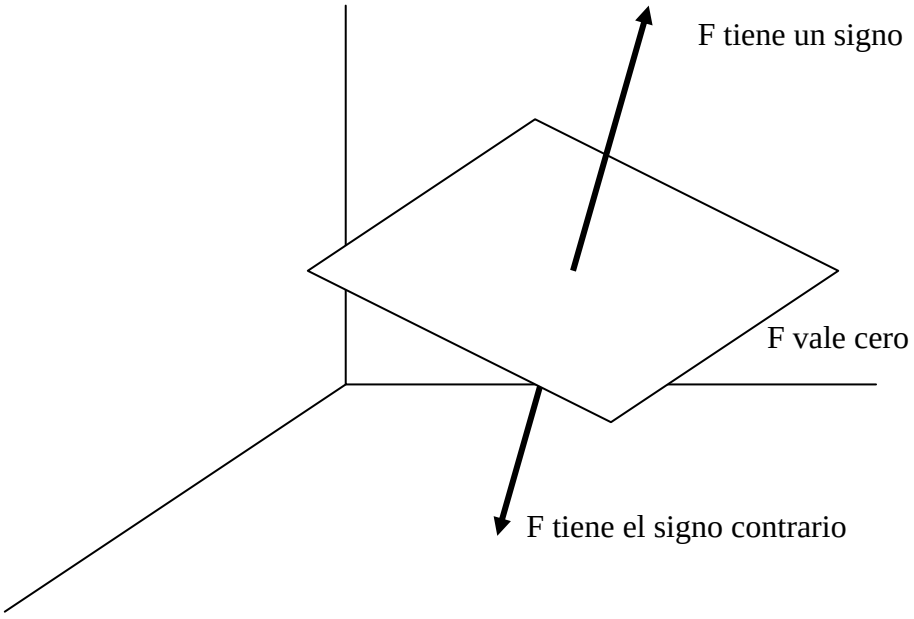

Figura 37. Función asociada al plano

4.8.6 Caras planas

Para comprobar que las caras son planas, se calcula en primer lugar el vector normal a la cara, tal como se ha comentado en un apartado anterior. Para ello, se tomaba la primera arista y la siguiente que no sea paralela a esta primera.

Con este vector y el punto inicial de la primera arista se calculan los coeficientes correspondientes al plano que incluye esa cara.

A continuación, se calcula el valor de la función de ese plano para todos los vértices de la cara, incluyendo las fronteras interiores. Si algún valor se aparta del cero más que el valor de la tolerancia, se produce el error correspondiente.

4.8.7 Aristas que se cortan

Esta es una de las comprobaciones más laboriosas. Se aplica a todas las aristas.

En primer lugar, se calculan y normalizan los vectores directores de todas las aristas.

A continuación se comprueba si se corta cada arista con todas las siguientes. Para distinguirlas, vamos a llamar i y j a cada pareja de aristas que se compara.

En primer lugar se mira si son consecutivas. En ese caso, se salta esta comprobación, ya que anteriormente se ha visto si las aristas se superponen, y dos aristas consecutivas tendrían que superponerse para considerar que se cortan.

Si no se da este caso (no son consecutivas), se comprueba si son paralelas. Hay una función que se encarga de ello que se comenta en este mismo bloque. Si son paralelas, o no se cortan, o son consecutivas, o se superponen, y este último caso ya se ha comprobado.

Una vez considerados estos casos, tenemos dos aristas que se cortan en un punto que no es un extremo, o que no se cortan.

En este caso se calcula, mediante el producto vectorial de los vectores directores de las dos aristas que se comparan, un vector perpendicular a ellas. Con esto tendremos un vector perpendicular a ambas aristas, independientemente de dónde estén situadas u orientadas (excepto en los casos que ya hemos comentado de ser paralelas, consecutivas o superpuestas). Este vector se normaliza para que su módulo valga la unidad. Le llamaremos vector perpendicular 1 para distinguirlo, y sus coordenadas serán (A, B, C).

En este momento calculamos los dos planos que contienen a cada arista que analizamos y que son perpendiculares al vector perpendicular a ellas. Las ecuaciones correspondientes a cada plano son:

$$F_i(x, y, z) = Ax + By + Cz + D_i$$

$$F_j(x, y, z) = Ax + By + Cz + D_j$$

Los coeficientes A, B y C son iguales para las dos ecuaciones, ya que coinciden con las coordenadas del vector perpendicular 1.

Esta operación se usa para comprobar si las dos aristas están en el mismo plano. Si es así, el término independiente de las ecuaciones (D_i y D_j) coincidirá, y si están en planos diferentes no, con lo cual, resulta imposible que se puedan cortar.

Ambas situaciones se ilustran en las siguientes figuras.

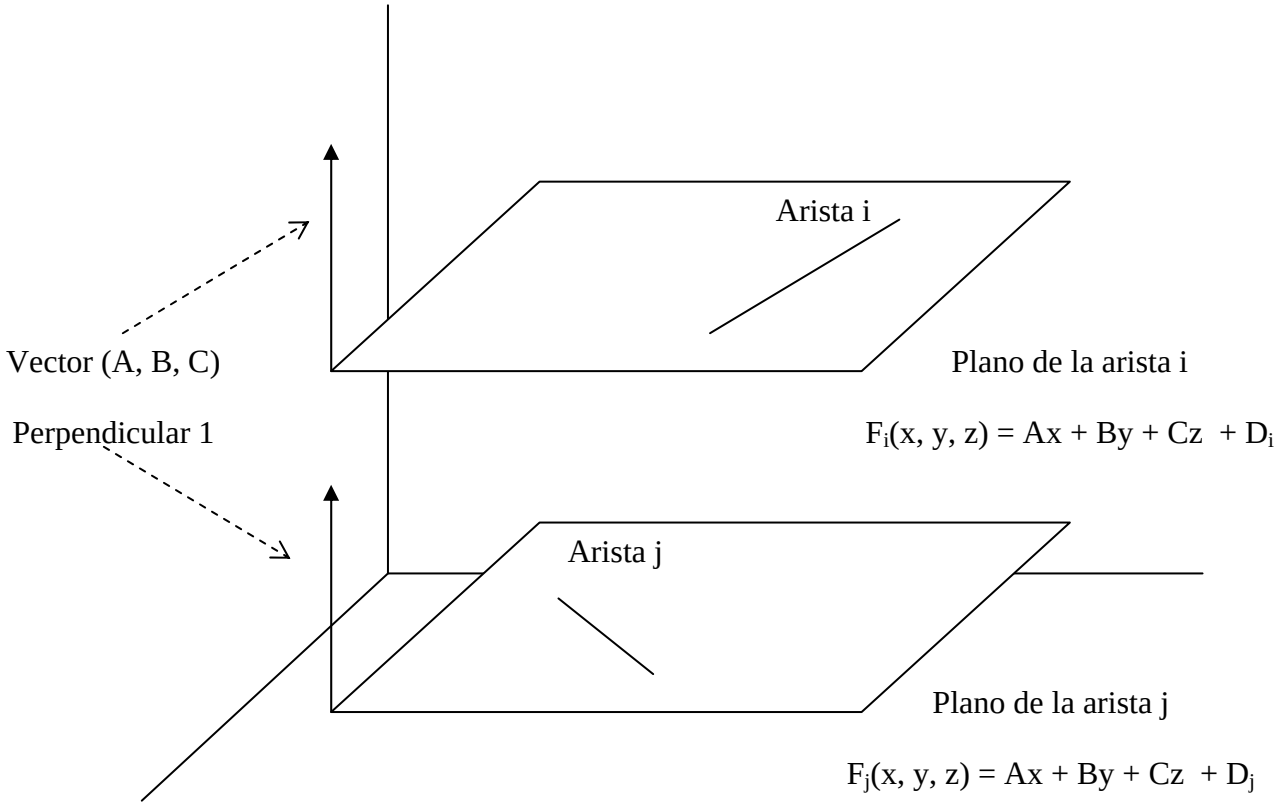

Figura 38. Aristas en planos paralelos

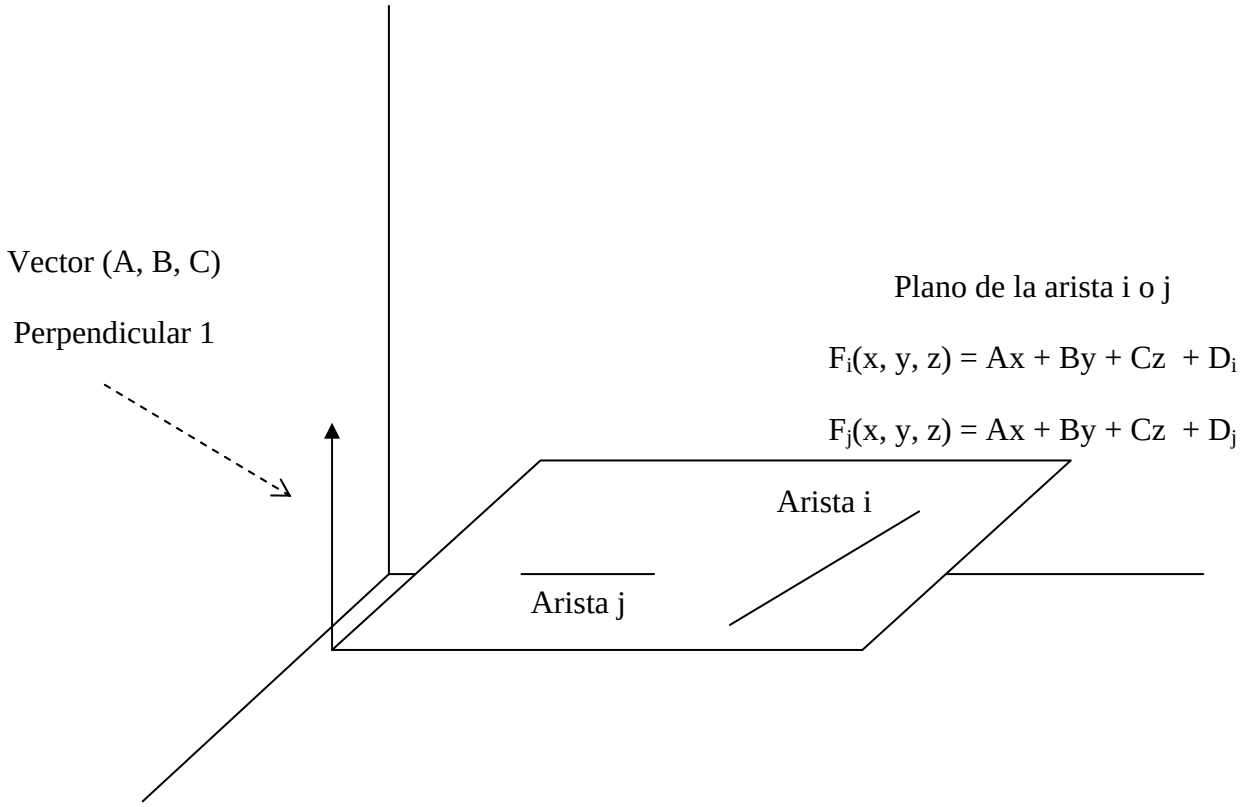

Figura 39. Aristas en el mismo plano

Una vez llegado al caso en que las dos aristas se encuentran en el mismo plano, calculamos un vector perpendicular a una de las aristas (la arista i, por ejemplo) y al vector perpendicular 1, al que llamaremos vector perpendicular 2. Sus coordenadas serán (A', B', C').

La siguiente figura muestra este nuevo vector.

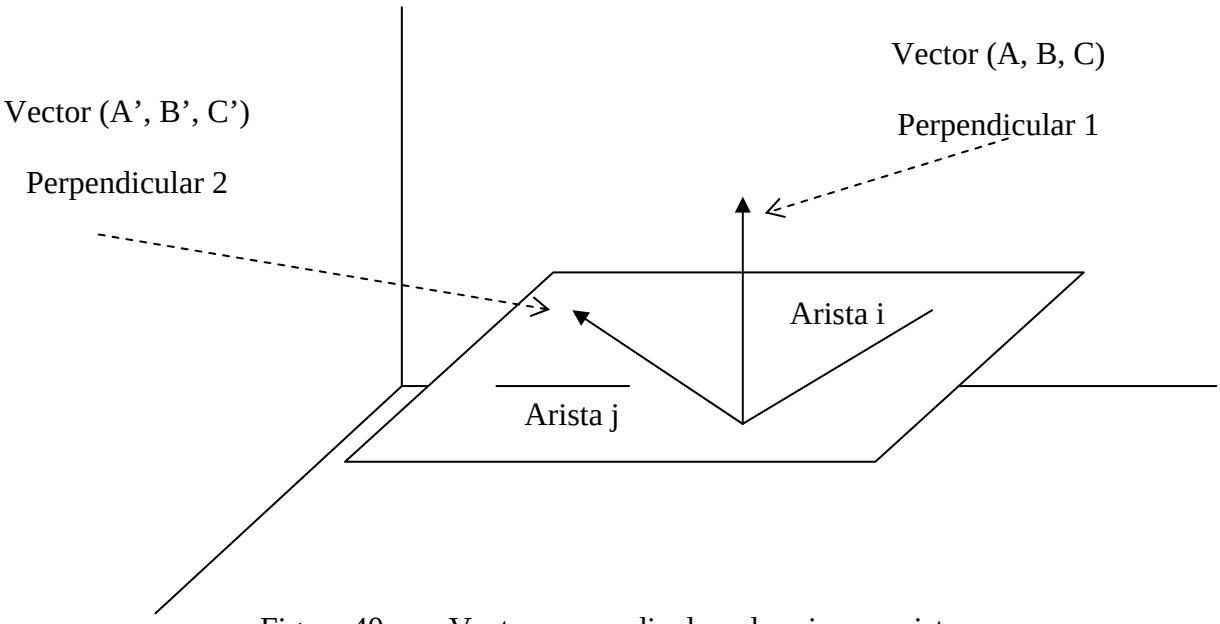

Figura 40. Vector perpendicular a la primera arista

Con este nuevo vector perpendicular 2 calculamos tres nuevos planos perpendiculares a él.

El primer plano contendrá a la arista i. Para calcular sus coeficientes se puede usar cualquiera de sus vértices. Su ecuación será la siguiente.

$$F'_i(x, y, z) = A'x + B'y + C'z + D'_i$$

Los otros dos planos serán paralelos a éste, y contendrán respectivamente los vértices inicial y final de la otra arista, la arista j. Sus ecuaciones son las siguientes.

$$F'_{j1}(x, y, z) = A'x + B'y + C'z + D'_{j1}$$

$$F'_{j2}(x, y, z) = A'x + B'y + C'z + D'_{j2}$$

Con este cálculo, podemos ver si los dos vértices de la arista j están a un lado de este plano, o están en ambos lados. Si están en el mismo lado no se pueden cortar. En caso contrario, sí que se pueden cortar. Las siguientes figuras ilustran las dos situaciones.

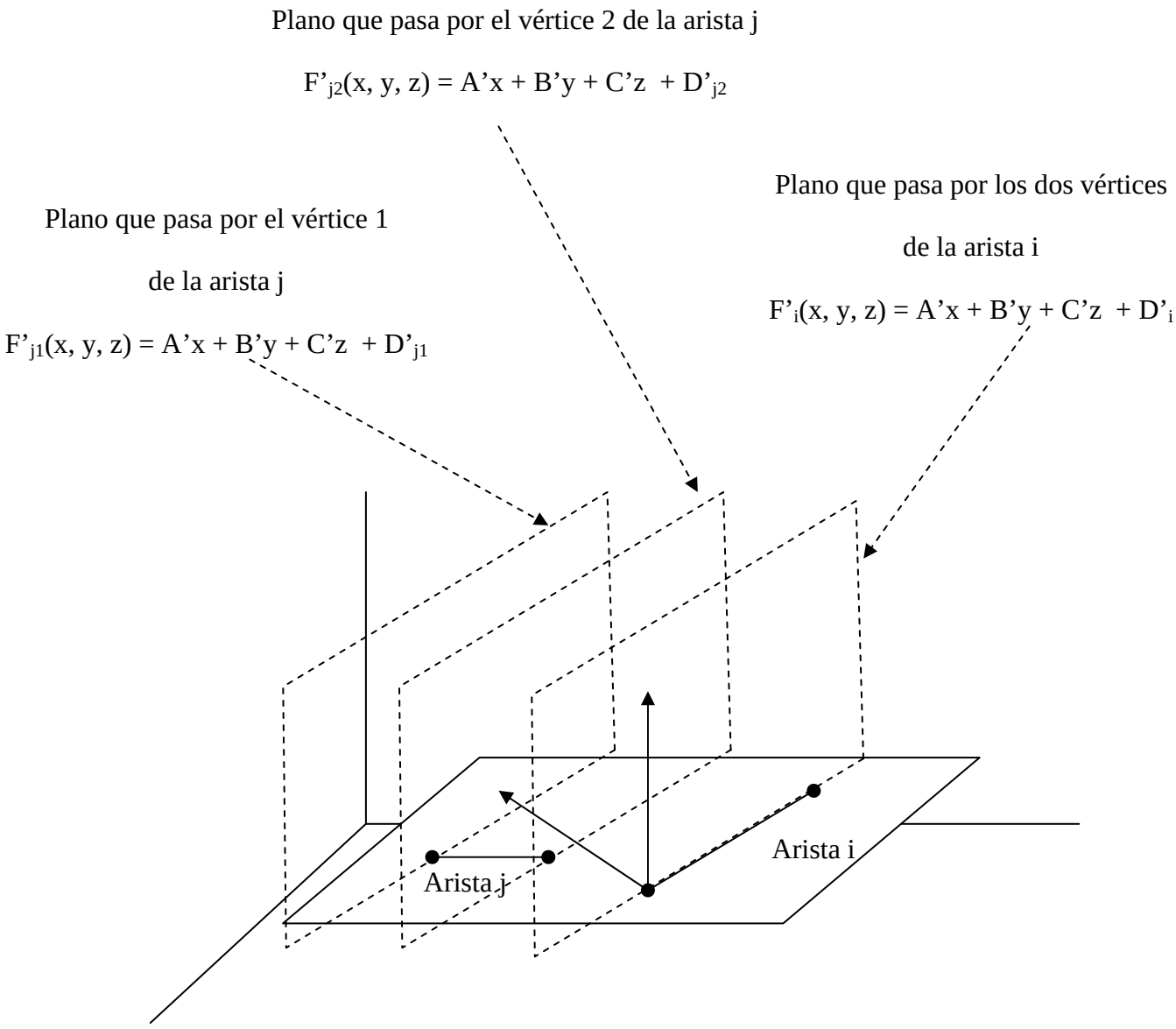

Figura 41. Planos que pasan por los extremos de las aristas. Los vértices de una arista están a un lado de la otra arista

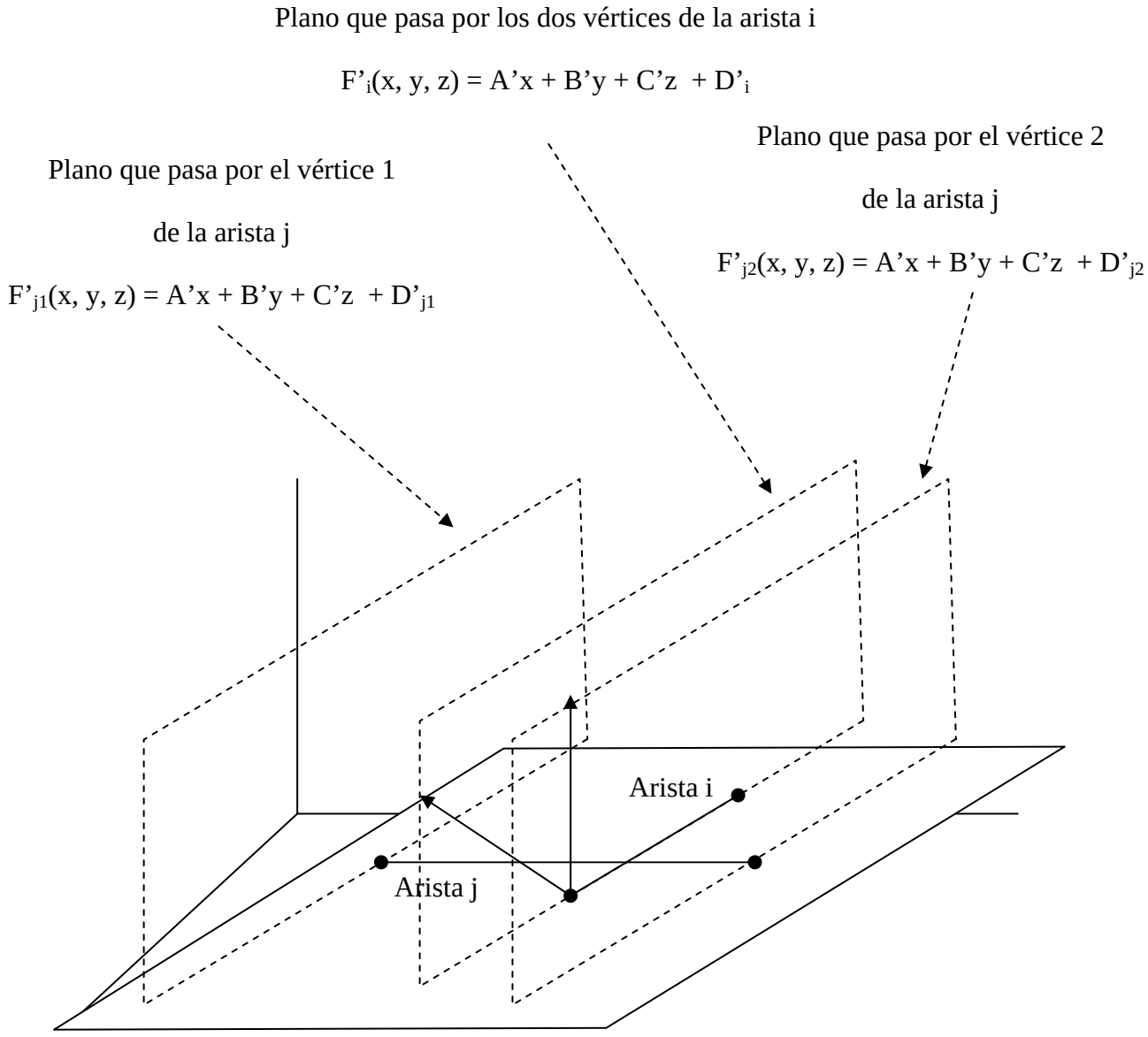

Figura 42. Planos que pasan por los extremos de las aristas. Los vértices de una arista están a ambos lados de la otra arista

Si estamos en este último caso, puede que se corten las aristas, pero no necesariamente. Si la arista j está desplazada, se cruzará con la prolongación de la arista i, pero no con ésta. La siguiente figura ilustra esta situación.

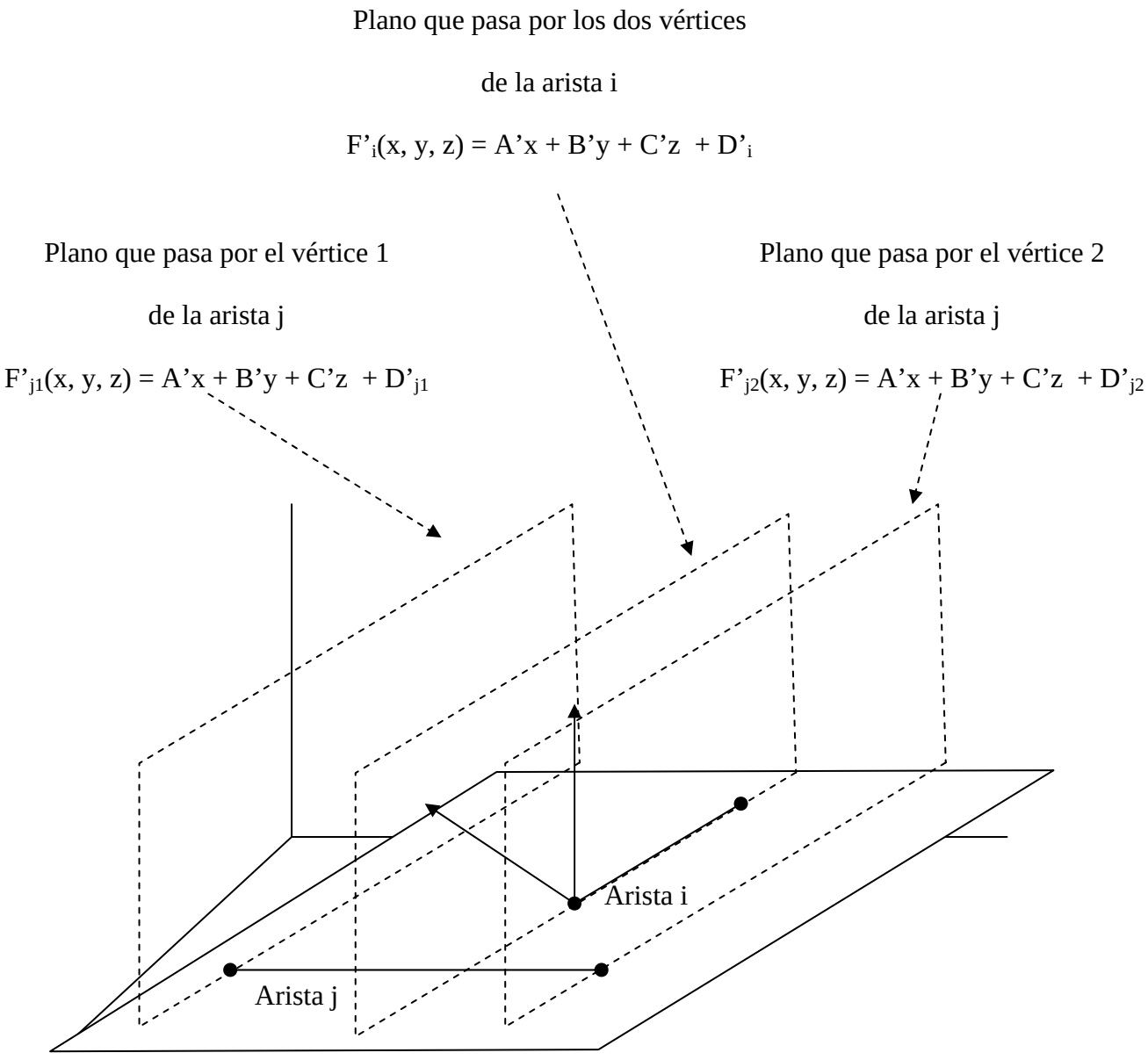

Figura 43. Planos que pasan por los extremos de las aristas. Los vértices de una arista están a ambos lados de la otra arista, pero no la corta

Para comprobar si definitivamente se cortan, lo que hacemos es calcular el punto en el que la arista j, corta a la línea que contiene la arista i.

Este punto puede calcularse por interpolación. Hay la misma proporción entre las distancias que hay de los planos que contienen a los vértices de la arista j al plano que contiene a la arista i, que las distancias que hay desde los vértices de la arista j al punto de corte. La siguiente figura lo ilustra.

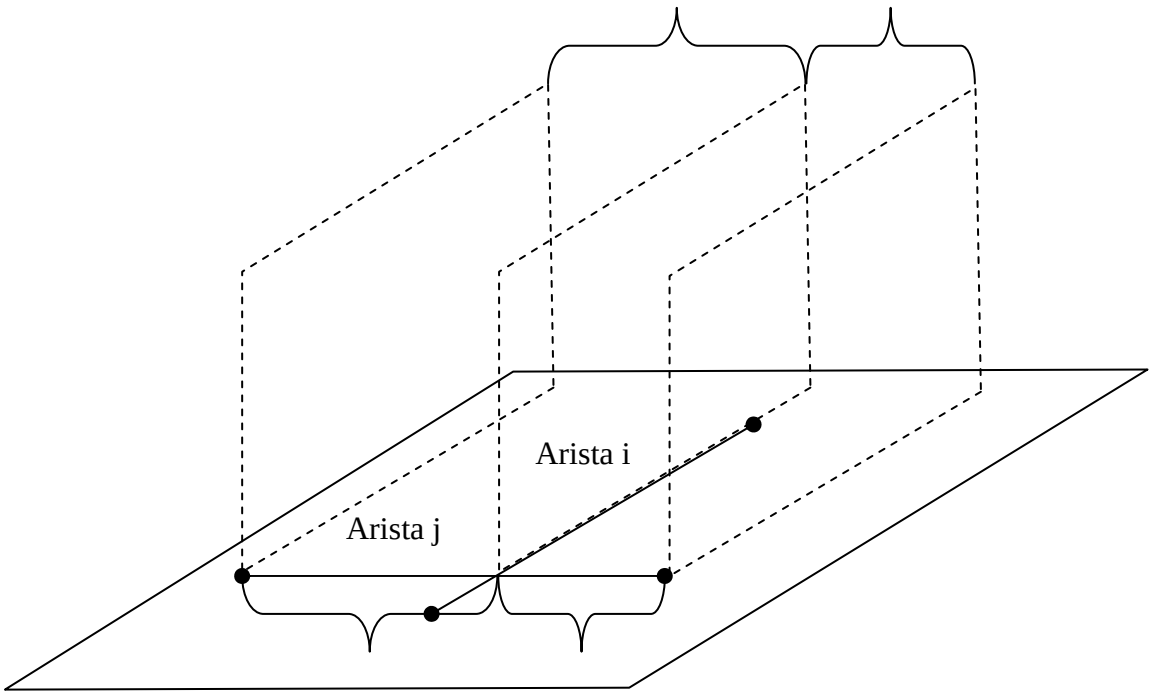

Figura 44. Interpolación para calcular el punto de corte

Una vez tenemos calculado este punto, solamente queda comprobar que las coordenadas de este punto se encuentran entre las de los vértices de la arista i. Si es así, las aristas se cortan. En caso contrario, no.

4.8.8 Aplicación de las fórmulas de Euler

La fórmula de Euler para poliedros simples relaciona el número de vértices, aristas y caras de objetos que no tienen agujeros.

Si llamamos V al número de vértices, A al número de aristas y C al número de caras, tenemos la siguiente expresión.

$$V + C = A + 2$$

Cuando el objeto tiene agujeros se tiene que aplicar la fórmula de Euler-Poincaré. En este caso tenemos que tener en cuenta el género o número de agujeros, G. La expresión de la fórmula es la siguiente.

$$V + C = A + 2\,(1 - G)$$

El problema es que para aplicar esta segunda fórmula, no se pueden contabilizar las caras con agujeros como si no lo tuvieran. Si se hace así, la fórmula no funciona.

Una solución para contabilizar correctamente el número de vértices, aristas y caras, consiste en transformar todas las caras con agujeros en caras sin agujeros.

Para ello, se han de añadir dos aristas y una cara por cada frontera interna, sin modificar el número de vértices. Esta transformación se ilustra en la siguiente figura. Los agujeros se muestran sombreados en gris.

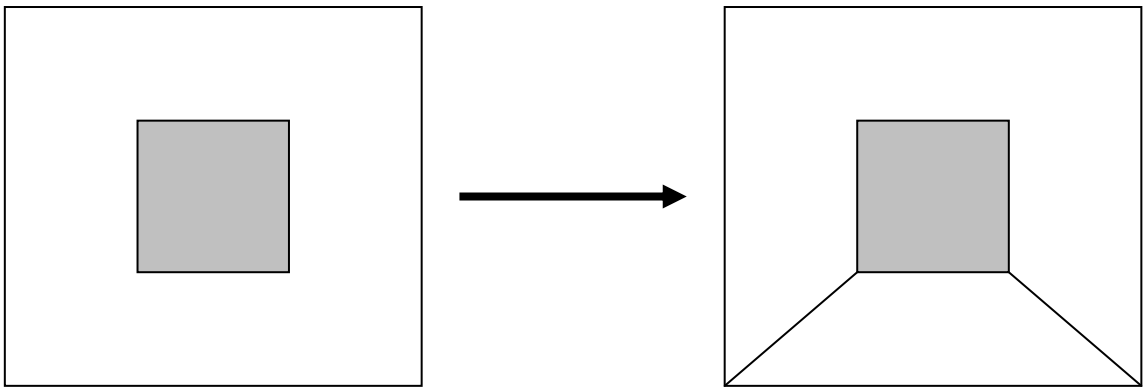

Figura 45. Transformación de caras con agujeros a caras sin agujeros

En este caso estamos uniendo dos vértices de la frontera exterior con dos vértices de la frontera interior. De esta forma aumenta en 1 el número de caras y en 2 el número de aristas, pero pasamos a contabilizar todas las caras sin agujeros.

Lo que hemos hecho ha sido unir vértices de la frontera exterior con vértices de la frontera interior, pero si hay varios agujeros, no tiene por qué ser así.

En el siguiente ejemplo, se podría pensar que no se puede unir ningún vértice de la frontera externa con ningún vértice de la frontera interna A, pero se puede unir desde cualquier vértice de las fronteras internas B o C.

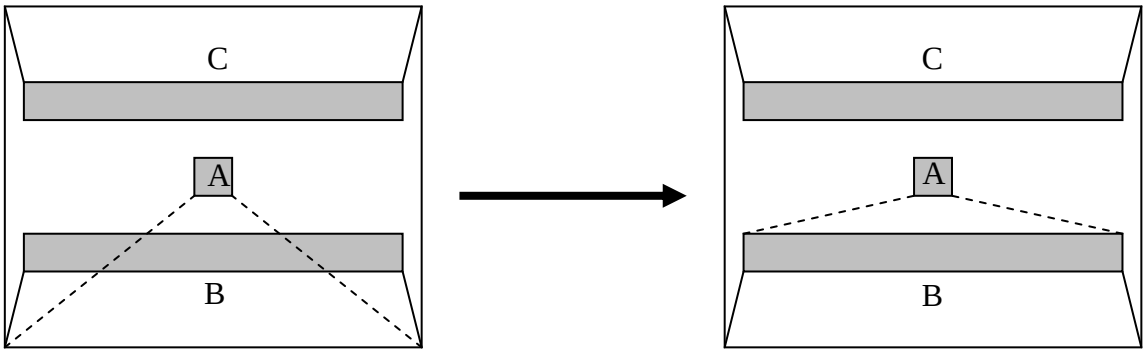

Figura 46. Transformación de caras con agujeros a caras sin agujeros con varios agujeros

Como se puede ver, la contabilidad sigue la misma regla. Por cada frontera interna se suma una cara y dos aristas.

De esta forma, el programa lo que hace es aplicar, para cada uno de los objetos, la fórmula de Euler para poliedros simples, y comprobar de esta forma si no tiene agujeros. Si es así, muestra el mensaje correspondiente en el log de errores.

Si no cumple esta fórmula, muestra un mensaje en el log, y aplica la fórmula de Euler-Poincaré, siguiendo la contabilidad de vértices, aristas y caras que se ha comentado, para calcular el número de agujeros. Y lo muestra como otro mensaje en el log de errores.

5 Formato propio que usa la aplicación

El archivo debe estar compuesto por 5 tipos de elementos: Objetos, Vértices, Aristas, Caras y Colores.

Cada elemento de cualquiera de las clases se representa en una línea ASCII.

Para identificarlos se usan palabras clave (objeto, v, a, c, i y color) que se pueden escribir indistintamente en mayúsculas o minúsculas.

Los comentarios empiezan con el símbolo #, aunque también se aceptan líneas completamente vacías (sin espacios en blanco).

5.1 Objetos

El identificador de la clase objeto es la palabra "objeto" seguido de su nombre. Si hay más palabras a continuación del nombre, se ignoran.

Atributos de Objeto:

Objeto nombre_objeto

Ej: Objeto cubo1 (En este caso estaríamos definiendo un objeto cuyo nombre es cubo1)

5.2 Vértices

El identificador de la clase vértice es la letra "v" seguido de su nombre. A continuación del nombre se esperan sus coordenadas (x, y, z).

Atributos de Vértices:

v nombre_vértice x y z

Ej: v 1 -1 -1 1 (Aquí estaríamos definiendo el vértice '1' cuyas coordenadas son x = -1, y = -1, z = 1)

5.3 Aristas

El identificador de la clase arista es la letra "a" seguido de su nombre. A continuación del nombre se espera la información de arista alada (vértice inicial, vértice final, cara izquierda, cara derecha, arista izquierda de entrada, arista izquierda de salida, arista derecha de entrada y arista derecha de salida).

Atributos de Arista:

a nombre_arista vertice1 vertice2 caraIzquierda caraDerecha aristaIzqEntrada aristaIzqSalida aristaDerEntrada aristaDerSalida

Ej: a a 1 2 fachada fondo b d i j (Aquí estaríamos definiendo la arista a, que va del vértice 1 al 2 – esto es para definir la parte derecha y la izquierda de la arista, una decisión arbitraria, ya que las aristas no tienen sentido -, con la cara fachada a la izquierda, y la cara fondo a la derecha. En la cara de la izquierda – fachada – la arista b sería la de entrada – llega al vértice 2 – y la arista d sería la de salida – sale del vértice 1. En la cara la derecha – fondo – la arista i sería la de entrada – llega al vértice 1 – y la arista j sería la de salida – sale del vértice 2. La siguiente figura ilustra esta situación).

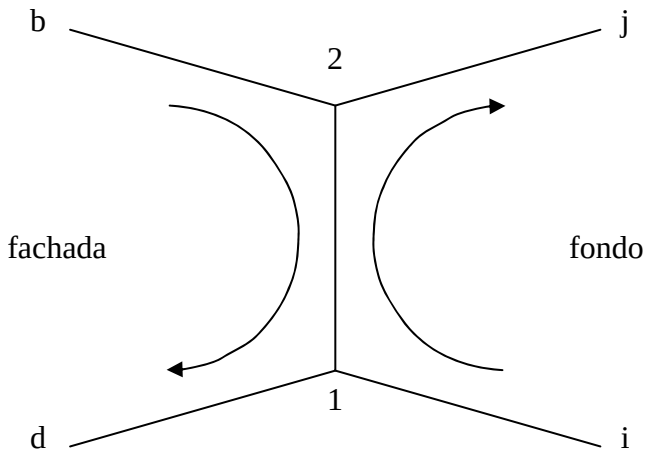

Figura 47. Ejemplo de definición de una arista (arista alada)

5.4 Caras

El identificador de la clase cara (o frontera externa) es la letra "c" seguido de su nombre. A continuación del nombre se espera la secuencia ordenada (en sentido horario vista desde el exterior) de las aristas que la definen.

Atributos de Cara:

c nombre_cara arista0 arista1 ... aristaN

Ej: c fachada a b c d (En este caso estaríamos definiendo la cara fachada cuyas aristas son a, b, c y d)

En caso de querer definir una frontera interna (para un hueco, saliente o agujero) el formato es el mismo que en las caras, pero el identificador es la letra "i" y el orden de la secuencia de aristas es antihorario visto desde el exterior.

Además, la frontera interna se ha de definir a continuación de una frontera externa, a la que se asignará. Por eso no se pueden definir como primera cara de un objeto. Por lo demás, se tratará igual que cualquier otra cara.

Atributos de Frontera interna:

i nombre_frontera_interna arista0 arista1 ... aristaN

Ej: i ifachada ia ib ic id (En este caso estamos definiendo la frontera interna ifachada, cuyas aristas son ia, ib, ic e id)

5.5 Colores

El identificador de la clase color es la palabra "color" seguido de su nombre. A continuación del nombre del color, se esperan las componentes RGB del mismo y una lista de las caras a las que se tiene que aplicar el color.

Atributos de Color:

Color nombre_color r g b cara0 cara1 ... caraN

Ej: color rojo 1 0 0 izq der fondo (En este caso estaríamos definiendo el color rojo, de componentes 1 (R), 0 (G), 0 () y que se aplicaría a las caras izq, der y fondo

5.6 Orden

El orden en el que se definen los elementos es importante.

Los vértices, aristas y caras de un objeto se tienen que escribir a continuación de la línea dónde se especifica el nombre del objeto, porque se asignan al último objeto definido.

El orden de estos elementos dentro de un mismo objeto no es importante, y se considera como una advertencia y no como un error, pero es recomendable para una mejor legibilidad.

Los colores se han de definir después de la descripción del último objeto.

5.7 Estructuras que se generan

Para almacenar la información de cada elemento (vértices, aristas, etc) se definen una serie de estructuras de datos (clases en lenguaje Java).

Además de la información propia de los elementos se añade información para seguir la lógica del programa.

5.7.1 Listas de elementos

Cada vez que se analiza un archivo, se genera una lista para cada tipo de elementos (vértices, aristas, caras, objetos y colores). Si la lista existe, se libera la memoria a la que está asociada y se vuelve a crear.

Estas listas se rellenan en el mismo orden en el que se leen del código. Es decir, el primer objeto que se define en el texto, es el primer objeto de la lista.

Estas listas se recorren normalmente en orden ascendente (para realizar alguna operación con esos datos), excepto cuando se mira a qué objeto pertenece cada elemento. En este caso, se recorre la lista de objetos en orden descendente y el elemento se asigna al primer objeto que se define en una línea anterior al elemento.

Como se puede ver en los apartados que siguen, cuando en un elemento se hace referencia a otros elementos, se hace mediante su nombre, pero para hacer más rápido el programa, se guarda también la referencia al número correspondiente en su lista. En caso contrario, tendríamos que buscar en cada momento en qué posición se encuentra cada elemento.

5.7.2 Matriz de vértices de dibujo

Se trata de una matriz (de tantas filas como vértices a dibujar y 3 columnas, una por coordenada) que se crea para la representación gráfica. En ella se guardan las coordenadas de los vértices que componen cada cara, ordenados por cara.

Esto significa que en primer lugar tenemos los vértices de la primera cara (ordenados), a continuación los de la segunda, etc. Es decir que los vértices se repiten (un vértice puede pertenecer a muchas caras, tres como mínimo), pero simplifica mucho la representación gráfica.

5.7.3 Clase vértice

La información básica de los vértices es la siguiente:

- Nombre del vértice
- Coordenadas cartesianas X, Y, Z del vértice

A esta información se le añade la siguiente:

- Número de línea en que se define
- Nombre del objeto al que pertenece
- Número del objeto en la lista de objetos

5.7.4 Clase arista

En la clase arista, la información básica es la siguiente:

- Nombre de la arista
- Nombres de los vértices inicial y final
- Nombres de las caras izquierda y derecha
- Nombres de las aristas de entrada y salida, tanto de la cara izquierda como de la derecha

A esa información se le añade la siguiente:

- Número de línea en que se define
- Nombre del objeto al que pertenece
- Número del objeto al que pertenece en la lista de objetos
- Números en las listas correspondientes de los vértices, aristas y caras del formato de arista alada
- Una instancia de la clase vectores que recoge la información de la arista y permite realizar comprobaciones sobre ésta

5.7.5 Clase cara

En la clase cara, la información básica es la siguiente:

- Nombre de la cara
- Lista ordenada (según el sentido de las agujas del reloj visto desde la parte exterior en el caso de la frontera externa de la cara, y al revés en las fronteras internas) de las aristas que la definen
- Color de la cara
- Si la cara es interna o no
- Número de fronteras internas que tiene la cara

Los últimos tres datos (color e información sobre fronteras internas) no tienen nada que ver con la información de arista alada, pero son necesarios.

A esta información se le añade la siguiente:

- Número de línea en que se define
- Nombre del objeto al que pertenece
- Número del objeto en la lista de objetos
- Número del color en la lista de colores
- Número de vértices (o de aristas) de la cara
- Lista ordenada (en el mismo orden que las aristas) de los nombres de los vértices de la cara
- Listas ordenadas de los números de las aristas y de los vértices de sus caras (en sus listas correspondientes)
- Número de la última frontera externa (este valor sirve para asignar las fronteras internas a la cara a la que pertenecen)
- Dos índices numéricos que indican el primer y último número en la lista de vértices de dibujo
- Una instancia de la clase vectores que guarda la información del vector normal a la cara
- Los coeficiente A, B, C y D que definen la función asociada al plano

5.7.6 Clase objeto

En la clase objeto, la información básica es la siguiente:

- Nombre del objeto
- Lista de caras que definen al objeto

A esta información se le añade la siguiente:

- Número de línea en que se define
- Lista de los números de las caras (en la lista de caras) que definen al objeto
- Número de caras del objeto

- Número de vértices del objeto
- Número de aristas del objeto
- Número de fronteras internas del objeto
- Número de fronteras externas del objeto

Los últimos cinco valores se utilizan para hacer cálculos con las fórmulas de Euler (Ley de Euler para poliedros simples y Ley de Euler-Poincaré).

5.7.7 Clase color

En la clase color, la información básica es la siguiente:

- Nombre del color
- Componentes RGB

A esta información se le añade la siguiente:

- Número de la línea en la que se define el color

5.7.8 Clase vectores

Esta clase se creó para implementar la mayoría de las funciones asociadas a las aristas.

Entre ellas tenemos el producto escalar, necesario para saber si las aristas son paralelas o no.

También tenemos el producto vectorial, usado para calcular el vector normal a la cara y varios cálculos más (si las caras son planas, o para la función asociada al plano).

La información que guarda esta clase es:

- Coordenadas de los vértices inicial y final de la arista
- Coordenadas del vector diferencia de los dos vértices
- Módulo del vector (que se usa básicamente para normalizarlo)

5.7.9 Secuencia de análisis

Para crear estas estructuras, se analiza el código en un orden determinado.

En primer lugar se analizan los objetos, seguidos de los vértices y un primer análisis de las aristas.

Como la información de arista alada hace referencia a vértices, aristas y caras, no se puede completar toda la información en el primer análisis de las aristas. Por eso solamente se analiza la información referente a los nombres (para registrarlas) y los vértices que las definen.

Después se analizan las caras, y una vez hecho esto, se completa el análisis de las aristas incluyendo las caras izquierda y derecha, y las aristas entrantes.

Los colores es lo último que se analiza.

6 Conclusiones

Una vez finalizado el proyecto, podemos afirmar que se han cumplido todos los objetivos propuestos.

- Se ha generado una aplicación que es multiplataforma, aunque se ha tenido que pagar el precio de crear versiones diferentes para cada sistema operativo.
- La aplicación representa correctamente los objetos definidos en el formato propuesto, y controla un gran número de posibles errores.
- Por otro lado, el vídeo explicativo consigue describir la funcionalidad del programa de una forma suficientemente intuitiva y sin extender excesivamente su duración.

Pero esto no significa que se trate de una aplicación cerrada. Algunas de las propuestas que se pueden plantear, para continuar en el desarrollo de esta aplicación son las siguientes.

- En primer lugar, se puede ampliar el número de errores que se controlan, en especial, controlar si alguna cara corta a otra. Además, siempre pueden aparecer errores nuevos en los que no se haya pensado.
- También se podrían ampliar los controles de la representación gráfica. Esto incluye desde incluir las vistas ortográficas de las piezas (alzado, planta y perfil), a incluir iluminación (controlando tipo de luces, posiciones y/u orientación, etc), tener en cuenta más parámetros de la cámara (ángulo de visión, etc) o mejorar los menús (añadiendo la posibilidad de cambiar el tamaño de letra, por ejemplo).
- Otra alternativa sería ofrecer la posibilidad de definir elementos de los diseños (vértices, aristas, caras, objetos y/o colores) para generar archivos que recojan esta información y faciliten un diseño definitivo.
- Por último, se podría añadir un formato más completo que incluyese herramientas y trayectorias de mecanizado, con lo que uniríamos el diseño asistido por computador con la fabricación asistida por computador.

7 Agradecimientos

En primer lugar tengo que agradecerle a Eduardo Vendrell Vidal, como director de este proyecto final de carrera, la oportunidad que me dio para realizar este proyecto.

En segundo lugar, tengo que dar las gracias a Manuel Herrero Mas por sus clases de Java que tanto me han ayudado.

Por último, tengo que agradecer a Juan Verdú Cueco, técnico de los laboratorios de la Escola Tècnica Superior d'Enginyeria Informàtica, su ayuda y comprensión para realizar las pruebas en los ordenadores de prácticas, y aclarar dudas.

8 Bibliografía y otros recursos

8.1 Java

APUNTES DE CLASE de Manuel Herrero Mas, Profesor de la asignatura "Ingeniería de la Programación" de la Universidad Politécnica de Valencia, curso 2007-08.

8.2 OpenGL

APUNTES DE CLASE de Roberto Vivó, Profesor de la asignatura "Gráficos por Computador" de la Universidad Politécnica de Valencia, curso 2009-10.

Página de referencia de la clase tesselator en la web de OpenGL.org. Disponible en : http://www.opengl.org/sdk/docs/man/xhtml/gluTessCallback.xml

OpenGL 1.2 Reference Manual de IBM. Disponible en: http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.opengl/doc/openglrf/gluBeginPolygon.htm

8.3 Java for OpenGL (JOGL)

Davison, Andrew. Pro Java 3D Game Development (Java 3D, JOGL, JInput and JOAL APIs. 1ª Edición. New York: Apress, 2007. 498 páginas. ISBN-13: 978-1-59059-817-7. ISBN-10: 1-59059-817-2. Contenido accesible desde http://fivedots.coe.psu.ac.th/~ad/jg2/

Página de Jogamp.org, organización que se encarga del mantenimiento del proyecto JOGL. Disponible en http://jogamp.org/jogl/www/

Especificación JSR 231 (JOGL). Disponible en: http://download.java.net/media/jogl/builds/archive/jsr-231-beta5/javadoc_public/

Jogl User's Guide. Disponible en: http://download.java.net/media/jogl/doc/userguide/

Repositorio de JOGL en la página de download.java.net. Disponible en: http://download.java.net/media/jogl/

8.4 Diseño asistido por computador

APUNTES DE CLASE de Eduardo Vendrell Vidal, Profesor de la asignatura "Diseño Asistido por Computador" de la Universidad Politécnica de Valencia, curso 2009-2010.

8.5 Otros recursos

Además de los anteriores recursos que se han comentado (libros y páginas web), también se usaron las demostraciones de programas JOGL que vienen incluidos en el plugin de NetBeans.

Página de descarga de NetBeans. Disponible en: http://netbeans.org/

Página de descarga de plugins de NetBeans. Disponible en: http://plugins.netbeans.org/

Página de descarga del plugin OpenGL para NetBeans. Disponible en: http://plugins.netbeans.org/plugin/3260/netbeans-opengl-pack

Video explicativo de la instalación de las librerías JOGL en NetBeans. Disponible en: http://www.youtube.com/watch?v=tHcsaJVWmxU

Página de descarga de las librerías OpenGL para Eclipse u otros IDE. Disponible en: http://download.java.net/media/jogl/builds/archive/jsr-231-1.1.0/

Página de descarga de la Máquina Virtual Java (JVM). Disponible en: http://www.java.com/es/download/

Requisitos para instalar la JVM o JRE 6.0. Disponible en: http://java.com/en/download/help/sysreq.xml

Requisitos para JVM 1.4.2. Disponible en: http://java.sun.com/j2se/1.4.2/jre/install-windows.html

Página de descarga del jdk. Disponible en: http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html#javasejdk

Página de descarga de Camstudio Recorder 2.6. Disponible en: http://camstudio.org/

Página de descarga de Subtitle Workshop 2.51. Disponible en: http://www.urusoft.net/downloads.php

Página de descarga de Spanishdub 1.5. Disponible en: http://sourceforge.net/projects/spanishdub/files/SpanishDub/SpanishDub%20v1.5/

Página de descarga del filtro subtitler.vdf. Disponible en: http://www.virtualdub.org/virtualdub_filters.html

Página de descarga de Zoomit. Disponible en: http://technet.microsoft.com/es-es/sysinternals/bb897434

Página de descarga de IconFX versión 1.6.4. Disponible en: http://www.321download.com/LastFreeware/page40.html

9 Índice de ilustraciones

Figura 1. Representación ambigua en modelado alámbrico 6

Figura 2. Pérdida de líneas de silueta en modelado alámbrico 6

Figura 3. Imprecisiones por facetado en el modelado de superficies 7

Figura 4. Instanciación y parametrización 8

Figura 5. Ejemplo de modelado CSG 9

Figura 6. Problemas con las fronteras en CSG 9

Figura 7. Ejemplo de enumeración espacial 10

Figura 8. Información de la estructura arista alada 11

Figura 9. Ejemplo del modelado B-Rep 12

Figura 10. Ventana inicial 18

Figura 11. Diálogo para abrir archivos 19

Figura 12. Vista Código 20

Figura 13. Vista Datos 22

Figura 14. Vista Dibujo 23

Figura 15. Diálogo para elegir el color de fondo 25

Figura 16. Limitaciones en los polígonos a representar 26

Figura 17. Opciones del menú Archivo 28

Figura 18. Opciones del menú Vista 28

Figura 19. Opciones del menú Controles 28

Figura 20. Opciones del menú Restaurar 29

Figura 21. Opciones del menú Opciones 29

Figura 22. Opciones del menú Ayuda 29

Figura 23. Menú contextual 29

Figura 24. Menú contextual Controles 30

Figura 25. Menú contextual Restaurar 30

Figura 26. Menú contextual Opciones 30

Figura 27. Ayuda Características programas 31

Figura 28. Ayuda Controles de visión 32

Figura 29. Ayuda Control de errores 34

Figura 30. Ayuda Formato Propio 35

Figura 31. Poliedro con una frontera interna definida fuera de su cara 47

Figura 32. Aristas situadas sobre líneas paralelas diferentes 49

Figura 33. Aristas paralelas situadas sobre la misma línea 50

Figura 34. Aristas que podrían dar problemas si se tuviese en cuenta solamente uno de sus vértices ……51

Figura 35. Aristas consecutivas en una misma cara 52

Figura 36. Ecuación del plano 54

Figura 37. Función asociada al plano 54

Figura 38. Aristas en planos paralelos 56

Figura 39. Aristas en el mismo plano 57

Figura 40. Vector perpendicular a la primera arista 57

Figura 41. Planos que pasan por los extremos de las aristas. Los vértices de una arista están a un lado de la otra arista 59

Figura 42. Planos que pasan por los extremos de las aristas. Los vértices de una arista están a ambos lados de la otra arista 60

Figura 43. Planos que pasan por los extremos de las aristas. Los vértices de una arista están a ambos lados de la otra arista, pero no la corta 61

Figura 44. Interpolación para calcular el punto de corte 62

Figura 45. Transformación de caras con agujeros a caras sin agujeros 63

Figura 46. Transformación de caras con agujeros a caras sin agujeros con varios agujeros ……64

Figura 47. Ejemplo de definición de una arista (arista alada) 66

10 Índice de tablas

Tabla 1. Ejemplo BRep Tabla de vértices 12

Tabla 2. Ejemplo BRep Tabla de caras 13

Tabla 3. Ejemplo BRep Tabla de aristas 13

Tabla 4. Lista de teclas de acceso rápido o shorcuts 33

Tabla 5. Requisitos para la Máquina Virtual de Java (JVM) 37

Tabla 6. Tamaño aproximado de las estructuras que se generan..... 38