TRABAJO ESPECIAL DE GRADO

"IMPLEMENTACIÓN DE UN SISTEMA DE MODELADO DE SÓLIDOS PARA LA CREACIÓN DE PRIMITIVAS GEOMÉTRICAS E INTERACCIÓN GRÁFICA, CON APLICACIONES EN LA INGENIERÍA MECÁNICA"

Presentado ante la ilustre
Universidad Central de Venezuela
Por el bachiller: Laurent A., Chevron M.
Para optar al Título de
Ingeniero Mecánico

Caracas, 2008

TRABAJO ESPECIAL DE GRADO

"IMPLEMENTACIÓN DE UN SISTEMA DE MODELADO DE SÓLIDOS PARA LA CREACIÓN DE PRIMITIVAS GEOMÉTRICAS E INTERACCIÓN GRÁFICA, CON APLICACIONES EN LA INGENIERÍA MECÁNICA"

TUTOR ACADÉMICO: Prof. Antonio Barragán.

Presentado ante la ilustre
Universidad Central de Venezuela
Por el bachiller: Laurent A., Chevron M.
Para optar al Título de
Ingeniero Mecánico.

Caracas, 2008

Caracas, Abril de 2008

Los abajo firmantes miembros de Jurado designado por el Consejo de Escuela de Ingeniería Mecánica, para evaluar el Trabajo Especial de Grado presentado por el Bachiller Laurent A., Chevron M., titulado:

“Implementación de un sistema de modelado de sólidos para la creación de primitivas geométricas e interacción gráfica, con aplicaciones en la Ingeniería Mecánica”

Consideran que el mismo cumple con los requisitos exigidos por el plan de estudios conducente al Título de Ingeniero Mecánico, y sin que ello signifique que se hacen solidarios con las ideas expuestas por el autor.

Prof. Manuel Martínez
Jurado

Prof. Rodolfo Berrios
Jurado

Prof. Antonio Barragán
Tutor Académico

DEDICATORIA

- *A mi Madre del cielo, y a mi Madre terrenal. Por la vida, por su educación, por su amor incondicional.*
- *A mi Padre Jesús, y a mi Padre Alain, por enseñarme el valor del trabajo, por enseñarme que la voluntad lo puede todo, por guiarme y siempre estar allí.*
- *A mi "Auelita" Dalia Ramona, por su grandeza espiritual, por ser mi guía del alma, por haber dado a luz a mi Madre, Tías, y mi recordado Tío Oscar.*
- *Pour ma Mémé chérie que j'aime beaucoup. Ce travail...C'est pour toi mémé Alice ! Merci pour ton soutient et ton grand coeur.*
- *A mis familiares fallecidos pero cuyo legado ha sido inmenso, los quiero mucho y me hacen falta: Abuelos Oscar Boscán, y Pépé Roger, Tíos Oscar Martínez y Jean Pierre Chevron.*
- *A mi amigo Luis Isava, lo logramos mi tercio!*
- *A mi familia en Venezuela, en especial a mis tías y tíos, por estar siempre allí.*
- *A mi familia en Francia, en especial a mi hermano Fabrice Simon, por su apoyo logístico incondicional.*
- *A San Juan Bosco, San Miguel Arcángel, y Mi Ángel Guardián, por enseñarme a levantarme, por protegerme y estar presentes en las buenas y en las malas. A su Santidad Juan Pablo II, por enseñarme a "No Tener Miedo".*
- *A mi hermano de Siria y ejemplo de esta Escuela, Mahamud, por enseñarme a valorar mi formación y mis estudios, por enseñarme la bondad y la hermandad que todos debemos tener entre estudiantes; A su nación Siria, por haber engendrado un ser tan especial. A su Dios Allah, muy respetado por Mahamud.*
- *A mis amigos y futuros Ingenieros, Daniel Páez, Gustavo Rojas, y Ernesto Centeno, por apoyarme incondicionalmente; las luchas se hacen más gratas cuando se tienen hermanos así!*
- *A los que utilicen esta tesis para apoyarse en su propio trabajo, a ellos en especial, recordemos que todos necesitamos de todos.*
- *A Venezuela como país y a Francia, por ser mis Patrias queridas.*
- *A la Universidad Central de Venezuela, mi Alma Mater!*

"Somos todos personas iluminadas, y poseemos el conocimiento que deseamos. Esto es así, pero el conocimiento, por sí mismo, inspira a veces sentimientos de suficiencia. El Saber, te vuelve orgulloso, mientras que el amor es constructivo: es él quien construye, porque aquél que ama, ayuda a los demás a crecer. Si alguien cree Saber, y presume con su "ciencia", entonces no ha entendido aún de qué naturaleza es el verdadero conocimiento".

(La Biblia,1 Corinthiens 8: 1-2).

AGRADECIMIENTOS

Quiero agradecer el apoyo brindado no solamente a los que me ayudaron en mi Tesis de Grado, sino también a los que contribuyeron a lo largo de mi formación de pregrado:

- *A mi Tutor Académico, Antonio Barragán, por su calidad en la enseñanza, por enseñarme que la organización y preparación forman la base para la consecución de importantes objetivos. Mis más sinceros agradecimientos profesor, por enseñarme a calmar mi espíritu, para verlo todo con claridad. Gracias por confiar en mi persona, y darme la oportunidad de trabajar con usted.*
- *Gracias a mis tíos María Auxiliadora y Mássimo Colletta, por estar siempre allí, por ser personas de gran voluntad y por enseñarme, a través de sus actos, cómo debe encararse la vida. Agradezco a la Nación Italiana, porque a través de sus hijos los Colleta, he logrado sumar esfuerzos para la consecución de esta importante meta.*
- *A mi Madrina Marisol y Guillermo Bello, por su fuerza espiritual, por apoyarme con sus oraciones. Una parte de este logro académico se lo debo a Carabobo, gracias a ellos.*
- *A mi Madrina Milagros Martínez, por su temple, por haber hecho grandes esfuerzos por esta familia cuando mi abuela quedó sola; por su constancia y su ejemplo, soy quien soy por personas como ella.*
- *A la familia Prado Gallegos, por su apoyo moral. Cuanto bien me hace saber que existen y forman parte de mi vida. Gracias "Maitine", tía Carol querida, , mon amour, te quiero mucho. Agradezco también a través de ustedes a la Nación Española.*
- *A la Familia Agelvis, por ayudarme en la logística del día a día, por estar allí cuando necesité de un "Combibe", de un "Javier" para trasladarme, de un "Larry", de una "Bertica" que cuidó a mi abuela y colaboró en mi casa en todo y fue siempre una guerrera…Nunca olvidaré a mis tíos Mercedes y Carlos, cuando más de una vez me apoyaron económicamente.*
- *A la familia Ayala, porque hacer de su hogar, mi hogar, y prestarme la computadora de la casa cuando en el 5to semestre de mi carrera se me dañó la mía. Gracias a Cococo, porque tomé prestada su mesa de dibujo arquitectónico, y realicé mis planos de Ingeniería. Gracias A mi Tía Luisa y Jesús, por su apoyo en aquel momento.*
- *A mis compañeros Daniel Páez, Gustavo Rojas, Ernesto Centeno, por ayudarme siempre!*
- *A mi Padre, Madre, Abuelita Dalia Ramona, y Mémé Alice, por la vida y educación.*
- *A la licenciada de la Biblioteca, mi Madrina Universitaria, por su profesionalismo y visión de vida, por sus consejos.*

- *A mi amigo Manuel Pérez, por recordarme que tenía las herramientas para organizarme mejor, y de esa forma culminar con éxito esta importante meta*
- *A mi amigo Miguel Martín, por su asesoría técnica en el tema desarrollado, y sus consejos.*
- *A los Técnicos del laboratorio de la Escuela, por ser Gente, por ayudarme siempre cuando lo requerí.*
- *A mi profesor de Mecánica Automotriz, Franklin Mora, por enseñarme el valor de las cosas bien hechas, por ser un ejemplo con su actitud ante la vida.*
- *A los profesores J. Larriva,, C. Quintíni, J. Segura, por la calidad de enseñanza impartida y visión de vida.*
- *Al viejito Ángel, profesor de Electricidad Automotriz en Catia, por enseñarme que todos somos iguales antes los ojos del Señor, que debemos tener sentido de responsabilidad para con nuestra gente necesitada, por trasmitirme su conocimiento técnico, y por ser mi soporte moral cuando lo necesité. A ti mi viejo!*

*Por último, le pido al **Señor** que me permita devolver en bien todo el apoyo recibido, a las personas que me ayudaron y a sus descendientes. Maestro, enséñame a esforzarme todos los días de mi vida por superar mis limitaciones, enséñame a prepararme hasta el fin de mis días, te pido humildad y fortaleza en todo momento. Gracias Maestro por enseñarme que soy un ser humano, que cometo errores, que puedo levantarme, y que lo más importante es seguir andando y evolucionando en la vida.*

Laurent A. Chevron M.

Laurent A. Chevron M.

IMPLEMENTACIÓN DE UN SISTEMA DE MODELADO DE SÓLIDOS PARA LA CREACIÓN DE PRIMITIVAS GEOMÉTRICAS E INTERACCIÓN GRÁFICA, CON APLICACIONES EN LA INGENIERÍA MECÁNICA

Tutor académico: Prof. Antonio Barragán. Tesis. Caracas. U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Mecánica. 2008. 109 pág.

Sistema de Modelado de Sólidos, Programa SMS_EIM_UCV, Sólidos, Operadores de Euler, Estructura de datos de semiarista.

Resumen

En el presente trabajo se desarrolló un programa ejecutable bajo el sistema operativo Linux, el cual permite crear, de manera automática, modelos geométricos de los objetos físicos que son utilizados posteriormente por los programas de análisis para el diseño de piezas o componentes mecánicos.

A través de la implementación de algoritmos para la generación de primitivas geométricas y la implementación de un sistema de interacción gráfica para la manipulación de los modelos, se creó un programa denominado Programa SMS_EIM_UCV, que permite crear sólidos sencillos tales como: esferas, pirámides, conos, cilindros, prismas rectos, cubos, bloques, sólidos por extrusión, y sólidos de revolución.

La interacción con el usuario se realiza mediante una ventana gráfica, en la cual se pueden crear sólidos nuevos, ver la representación gráfica de un sólido existente, y manipular los objetos creados mediante operaciones de alejamiento, acercamiento, traslación y rotación del sólido.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En la Ingeniería Mecánica, un área de gran importancia es el diseño de piezas o componentes mecánicos, de manera que puedan soportar de forma confiable los esfuerzos que se generan en ellos al ser sometidos a condiciones de funcionamiento. Durante mucho tiempo, la evaluación de esos diseños se ha realizado mediante técnicas experimentales, lo cual resulta ser de un costo sumamente elevado, además de una gran inversión en infraestructura (laboratorios, equipos de ensayos, etc.).

En el diseño de componentes mecánicos intervienen diferentes factores que afectan la distribución de esfuerzos en los mismos, como son la forma, condiciones de apoyo, características de los materiales involucrados, cargas aplicadas y otros. La conjunción de estos factores hace que el proceso de análisis sea complejo, exigiendo métodos sofisticados, que tengan capacidad para simular diferentes condiciones de trabajo.

En la actualidad existen métodos computacionales de análisis que cumplen estos requisitos. Basados en técnicas matemáticas aproximadas, estos métodos permiten simular con grado variable de aproximación el comportamiento de piezas mecánicas bajo condiciones de trabajo. El uso de estos métodos se basa en el computador digital, el cual permite lograr resultados útiles en poco tiempo a bajo costo, siendo posible determinar los efectos de diferentes condiciones de cargas, geometrías y materiales. Fundamentalmente, los métodos utilizados son: el Método de los Elementos Finitos y el Método de los Elementos de Contorno.

En la Escuela de Ingeniería Mecánica de la UCV se han venido desarrollando un grupo de programas basados en los métodos mencionados con la finalidad de tener una plataforma computacional para la solución de algunos tipos de problemas de la física matemática (análisis de esfuerzos, transferencia de calor). Estos problemas se presentan al simular el comportamiento de piezas o componentes como parte del proceso de diseño de los mismos.

Sin embargo, la aplicación de estos métodos ha estado limitada a modelos bidimensionales y casos muy sencillos de modelos tridimensionales, debido a la

carencia relativa de Sistema de Modelado de Sólidos que permita la creación automatizada y la manipulación de modelos tridimensionales de piezas complejas. Esto constituye una limitación fundamental que obstaculiza la aplicación más amplia de las técnicas hasta ahora desarrolladas.

Al presente, en la EIM se ha realizado un desarrollo preliminar, que consistió en la creación de un sistema para la creación de sólidos sencillos (primitivas) tales como esferas, prismas, paralelepípedos; empleando operaciones de barrido de traslación y de rotación. Este sistema se encuentra obsoleto hoy día, ya que fue desarrollado en un ambiente MS-DOS, exigiendo una puesta al día una actualización, aprovechando las interfases gráficas de los computadores modernos.

Por otra parte, la riqueza, complejidad y variedad de las disciplinas involucradas hace que el desarrollo de un modelado geométrico sea un área fértil para el desarrollo intelectual donde hay espacio para la investigación y el desarrollo, constituyéndose así en un gran soporte del trabajo multidisciplinario (diseño, matemáticas, computación, manufactura).

Por lo mencionado anteriormente, se ha fijado como objeto fundamental del presente trabajo la implementación de un conjunto de técnicas de modelado geométrico que permita de manera automática la creación de modelos geométricos de los objetos a analizar, con la finalidad de automatizar la creación de los modelos que son utilizados por los programas de análisis. Ello involucrados aspectos fundamentales como:

a) Implementación de los algoritmos para la generación de primitivas geométricas;

b) Implementación de un Sistema de Interacción Gráfica para la manipulación de los modelos.

CAPÍTULO I

ANTECEDENTES DEL MODELADO DE SÓLIDOS

Desde el inicio de los años 1960, la informática gráfica ha jugado un papel cada vez más importante en sectores tan variados como la industria automotriz, la arquitectura, la cinematografía, etc. Según Marcheix [3], de manera más o menos simultánea, los primeros trabajos en relación al modelado geométrico se iniciaron en Francia (Bezier, De Casteljau) y en los Estados Unidos (Coons). Se trataba principalmente de investigaciones en la industria automotriz (Renault, Citroën, Ford), aeronáutica (Boeing, etc) y el sector naval. Los primeros sistemas de modelado geométrico (sistemas que reunían un conjunto de herramientas relativamente completas para la creación, manipulación, y la deformación de objetos) remontan del inicio de la década de los años 70.

Desde entonces, distintos métodos de modelado han visto luz, a través de cada uno de los cuales se ha buscado incrementar la diversidad de objetos modelables, al mismo tiempo que se ha tratado de aumentar la facilidad de manipulación.

En todas estas áreas se utiliza de forma directa o indirecta, el modelado geométrico. Esta etapa del proceso infográfico consiste en definir una representación informática de las distintas característica geométricas y topológicas de los objetos manipulados.

La herramienta informática que permite crear y manipular esas representaciones es llamado software de modelado geométrico, sistema de modelado, o sencillamente modelador. Un aspecto importante de los modeladores consiste en hacer de la manipulación de los objetos, algo lo más intuitivo posible. Es entonces evidente que el núcleo matemático y algorítmico a menudo complejo, debe ser ocultado al usuario, con la ayuda de herramientas sofisticadas, y de una interfaz de alto nivel.

Los roles de un tal software son diversos y dependen fuertemente del campo de utilización:

- En el marco de la CFAC (Concepción y Fabricación Asistida por Computadora), el objetivo principal de un modelador es permitir la fabricación de objetos maquinables. Por consecuencia, el utilizador debe poder manipular objetos sin incoherencia topológica o geométrica (por ejemplo, sin auto-intersección), al mismo tiempo que se verifiquen restricciones técnicas o funcionales (rigidez, separación mínima entre dos partes del objeto, etc.).
- En lo que se refiere a la síntesis de imágenes, el objetivo final es el de crear una imagen que represente una escena virtual de forma lo más realista posible. Aquí, la geometría de los objetos solo se utiliza para fines visuales.

En la Escuela de Ingeniería Mecánica de la Universidad Central de Venezuela se han venido desarrollando trabajos en relación al modelado de sólidos, para resolver problemas con aplicación a la Ingeniería Mecánica; podemos citar los siguientes autores de tesis de grado:

- Uzcátegui y Uzcátegui [2]: desarrollaron un programa computacional con aplicación a la Ingeniería Mecánica en donde implementaron un conjunto de técnicas básicas de modelado de sólidos, logrando representar sólidos sencillos (prismáticos), apoyándose en operaciones constructivas (operadores de Euler) que les permitieron crear herramientas (como la división de un sólido por un plano) que sentaron las bases para crear sólidos complejos.
- Alonzo y Oramas [4]: desarrollaron un Sistema de Representación Gráfica de entidades bi y tridimensionales, aplicables a componentes mecánicos y de Bioingeniería, el cual permite realizarle transformaciones a objetos ya creados, como operaciones de rotación, translación, proyecciones, cambio de escala, y visibilidad.
- Ordaz y Pulido [5]: trabajaron en la resolución de problemas geométricos asociados a la generación automática de mallas tridimensionales de elementos finitos en piezas de forma libre; se enfocaron en estudiar la posición de un punto respecto a un sólido, un sólido (elemento finito) respecto a un plano, intersecciones entre plano y plano, plano y sólido.

- Jaén y López [6]: desarrollaron un sistema de post procesamiento gráfico para representar resultados de esfuerzos en órganos dentarios bidimensionales mediante el método de elementos finitos, y resolver problemas de elasticidad. Se implementaron técnicas de representación de superficies isoparamétricas mediante funciones de interpolación.
- Martín H. Miguel A. [7] : desarrolló un programa de post-procesamiento de datos resultantes del análisis por métodos numéricos de problemas de ingeniería, implementando un módulo de visualización gráfica de resultados, mediante el cual el usuario puede mostrar una representación gráfica del estudio ya sea para analizar, interpretar, corregir y/o publicar estos resultados.

CAPÍTULO II

MODELADO DE SÓLIDOS

¿Qué es el modelado de sólidos?

El modelado de sólidos es una rama del modelado geométrico que busca crear representaciones “completas” de objetos físicos sólidos, es decir representaciones que permiten responder cualquier pregunta geométrica a través de algoritmos (sin la ayuda de la interacción de un usuario).

El modelado de sólidos se enfoca en la aplicabilidad general de modelos, diferenciándose así de otros tipos de modelos geométricos que son creados para cumplir una función precisa. Los modelos gráficos se enfocan en crear el dibujo de cualquier objeto.

Los modelos de superficie ofrecen información detallada de una superficie curva, pero no siempre proporciona suficiente información para determinar todas las propiedades geométricas del objeto envuelto por la superficie.

Puesto que se busca una aplicación general del modelado de sólidos, cualquier sistema de modelado de sólidos debe ser capaz de dar respuesta, a través de algoritmos, a las preguntas geométricas con aplicación a la ingeniería. Algunas de estas preguntas son:

¿Cómo se ve el objeto?

¿Cuál es la masa del objeto, su área superficial?

¿El objeto colisionará con otro objeto cuando se moverá?

¿Es el objeto capaz de soportar cierta carga?

¿Cómo puede ser fabricado el objeto a través de algunos procesos de manufactura existentes?

Se puede apreciar que el resultado de una pregunta geométrica puede ser un simple número, una constante booleana (verdadero, falso), una imagen. De hecho, puede haber otro modelo de sólido que modele el resultado de los cálculos, para responder a la

pregunta: ¿Cual es el efecto sobre el objeto cuando se le aplica tal o cual proceso de manufactura?

Es decir, es importante que un modelador geométrico incluya facilidades para modelar no solamente objetos físicos, sino también el efecto sobre los mismos al aplicarles procesos físicos (manufactura).

Sistema de Modelado de Sólidos

Unas de las ideas fundamentales del modelado geométrico, expuesta por Mäntylä [1], es la de separar el modelado propiamente dicho de la aplicación, y desarrollar técnicas de modelado que sean relativamente independientes de algún objeto en particular modelado, y de los usos pretendidos de dichos modelos.

Un sistema de modelado de sólidos persigue entonces crear modelos sólidos de objetos físicos, independiente de su aplicación, y dar respuesta a las preguntas geométricas anteriormente formuladas.

Adicionalmente a los problemas de completitud, integridad, complejidad, existen varios puntos prácticos que hacen de la elaboración de un modelador de sólidos, un problema no trivial.

Para ilustrar lo anteriormente expuesto, examinemos los componentes funcionales de un modelador de sólidos y sus roles en un sistema geométrico como el mostrado en la figura 2.1 (Mäntylä [1]).

Inicialmente, los objetos son descritos por el modelador en términos de un lenguaje descriptivo basado en los conceptos de modelado disponibles en el modelador de sólidos. El usuario debe introducir la descripción sencillamente como texto simple, o preferiblemente a través de una interfaz de usuario con la ayuda de interacción gráfica.

Una vez los datos ingresados, las descripciones del objeto son traducidas para crear la representación actual interna almacenada por el modelador. La relación entre el lenguaje descriptivo y la representación interna no necesariamente debe de ser directa: las representaciones internas pueden emplear conceptos del modelado distintos a la descripción original. Más aún, un modelador de sólidos puede perfectamente incluir varios lenguajes descriptivos que se adapten a diferentes tipos de usuarios y aplicaciones.

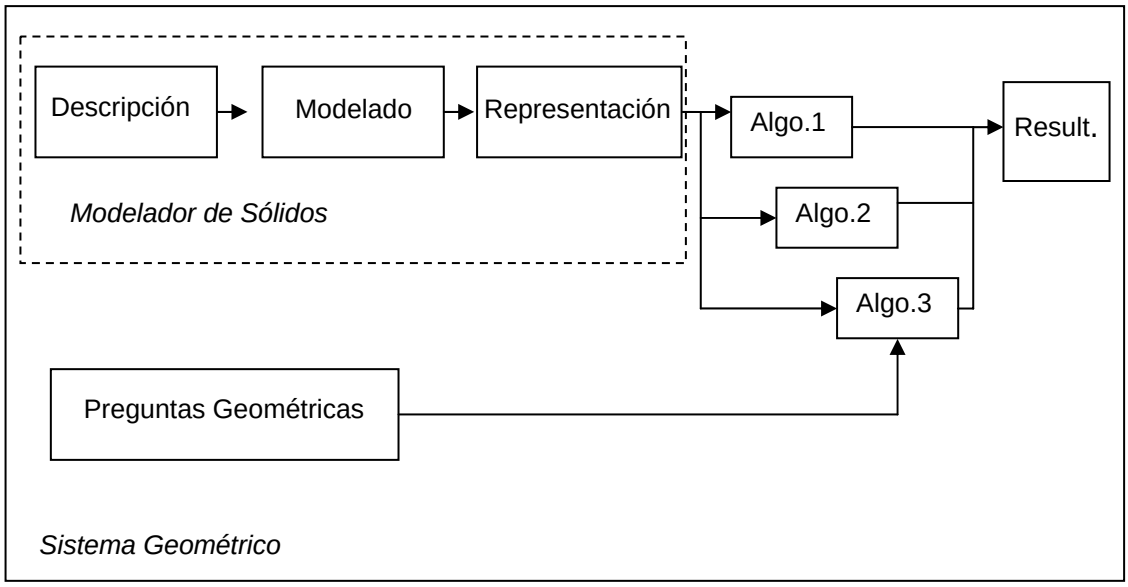


Fig. 2.1- Componentes funcionales de un modelador de sólidos (Según Mäntylä [1])

El modelador debe proveer interfaces para comunicarse con otros sistemas. Esas interfaces son utilizadas para transmitir información dirigida a varios algoritmos, o suministrar la información completa de modelos sólidos para otros sistemas de diseño.

El modelador debe también incluir facilidades para almacenar la descripción de objetos y demás datos, almacenados en permanentes bases de datos.

Con el propósito de dar respuesta a preguntas geométricas, una transformación desde la descripción externa hacia la representación interna es necesaria. De hecho, para aumentar la eficiencia de un modelador de sólidos, este debería soportar múltiples representaciones internas de los objetos; se tienen que incluir algoritmos de conversión que permitan modificar data desde una representación hacia otra.

CAPÍTULO III

CARACTERÍSTICAS DESEABLES DE UN SISTEMA DE MODELADO DE SÓLIDOS

El corazón de un sistema de modelado de sólidos es lo que llamamos el *modelo geométrico*. Basado en una abstracción matemática, se trata de una representación numérica que tiene por objetivo sacar provecho de dicha abstracción al mismo tiempo que se abra el campo a una larga variedad de aplicaciones.

Según Marcheix [3], desde un punto de vista formal, se distingue generalmente tres niveles de abstracción en la definición de objetos (figura 3.1):

- Una abstracción física, que define los objetos reales tales como el saber científico permite percibirlos, con toda su complejidad, tanto al nivel visible como microscópico.
- Una abstracción matemática, que permite definir los objetos físicos de manera formal.
- Una abstracción informática, que permite distinguir la clase de objetos de la abstracción matemática que serán representables de manera numérica.

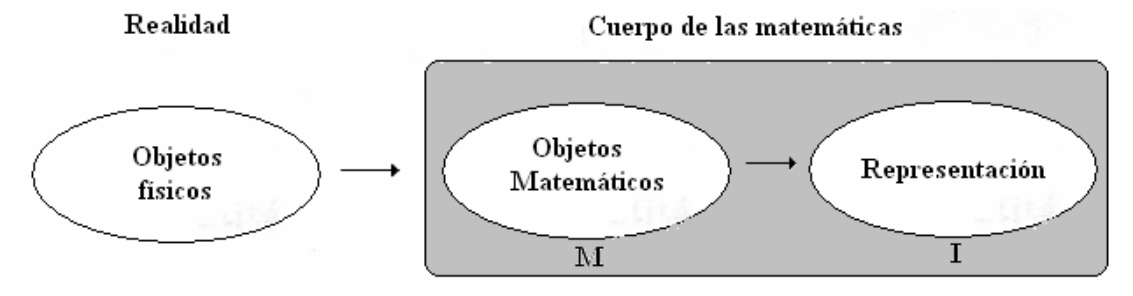


Figura 3.1: Abstracciones del modelado (Marcheix [3])

Marcheix [3] establece entonces que es posible definir un modelo geométrico a partir de la relación *s: M → I,* donde M representa el espacio de objetos matemáticos e I el espacio de representaciones informáticas. El dominio de s perteneciente a M es notado D, y la imagen de D por s perteneciente a I es notada V (figura 3.2). Un elemento de I es llamado una representación.

Podemos subrayar lo siguiente:

- D está incluido en M, puesto que todos los objetos definidos por la abstracción matemática no son necesariamente modelables por s.
- V incluido en I, puesto que toda representación de I no es necesariamente la imagen de un objeto.

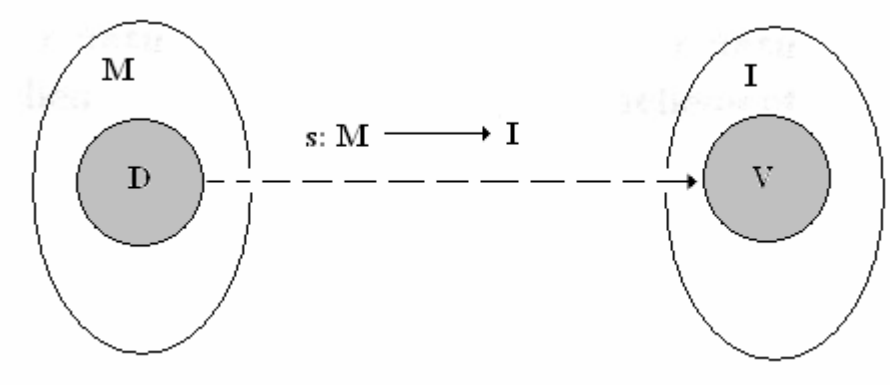


Figura 3.2: Definición de un esquema de representación (según Marcheix [3]).

A partir de estas nociones, pueden ser definidos distintos criterios para permitir caracterizar un modelo:

- Ambigüedad: si cada representación r Є V modela exactamente un objeto por s (r es la imagen de un solo elemento de M), entonces s se dice que no es ambiguo. Es decir, que un modelo que represente un único sólido a la vez, es no ambiguo. Por lo tanto, para mantener el criterio de no ambigüedad, ninguna representación válida debe modelar más de un sólido.

- Unicidad: se dice que s es único si cada objeto m Є D posee exactamente una representación r Є I (Si s (m) = r y s(m) = r´, entonces r = r´). Es decir, cualquier objeto real tendrá por representación un modelo único. Por consiguiente, ningún sólido debe tener más de una representación válida.

Las cualidades de un modelo pueden entonces ser evaluadas con la ayuda de siete características siguientes:

- Potencia: ¿Cuales objetos son incluidos en el dominio D de objetos modelables cubiertos por el modelo? ¿Es posible extender este dominio?
- Ambigüedad: ¿El modelo es ambiguo?
- Unicidad: ¿El modelo es único?
- Validez: ¿Todas las representaciones r Є I son válidas? Corresponden a un elemento de M? La mejor solución siendo que todas las representaciones sintáxicamente correctas sean válidas (V=I).
- Complejidad: ¿Cuál es el tamaño de memoria necesaria para la representación de los objetos del modelo? En general esta propiedad es inversamente proporcional a la cantidad de informaciones que contiene el modelo.
- Cierre: ¿Las operaciones de creación y manipulación de las representaciones conservan la validez del modelo?
- Aplicabilidad: ¿Que tipos de algoritmos pueden realizarse para este tipo de representación, y a que costo?

Estos son los requisitos que debe cumplir un sistema de modelado geométrico con el fin de que sea un modelo completo, es decir que posea la información geométrica necesaria para poder calcular cualquier propiedad geométrica, y que mantenga su integridad geométrica, es decir que la intersección entre sí de las caras del modelo solo debe darse en aristas o vértices comunes.

CAPÍTULO IV

TIPOS DE MODELADO GEOMÉTRICO

El modelado geométrico sólido

Existen dos aproximaciones distintas para modelar lo que llamamos intuitivamente un “sólido”. Según Marcheix [3], la primera permite representar un sólido describiendo el conjunto de puntos que definen su volumen interior, mientras que la segunda lo representa describiendo su superficie frontera. Estas dos formas de verlo, nos conducen respectivamente a las nociones de topología general y topología algebraica de superficies.

El modelado geométrico sólido por conjuntos de puntos (R-set)

La forma más natural de representar un objeto definido en el espacio real que nos rodea, consiste en determinar explícitamente el subconjunto de puntos de R3 (espacio tridimensional) que pertenecen al interior del objeto. En la mayoría de los casos se desea manipular objetos terminados. Por esta razón es importante definir la noción de sólido de la manera siguiente, tal y como lo establece Marcheix [3]:

Definición: un sólido es un subconjunto cerrado y acotado de R3.

Aunque esta definición nos permite caracterizar algún aspecto de los sólidos, no obstante no es lo suficientemente restrictiva para caracterizar de manera precisa la noción intuitiva de sólido (figura 4.1). Tres criterios suplementarios son agregados entonces a esta definición expuesta por Marcheix [3].

- La rigidez: Es natural pensar que un sólido permanezca idéntico, cuando sufre un desplazamiento. Esto puede expresarse de manera más rigurosa imponiendo que el sólido quede invariable por transformaciones rígidas (rotación y traslación).

- La regularidad: Aunque el objeto manipulado satisfaga el criterio de rigidez, varios conjuntos de puntos cerrados y acotados no definen realmente un sólido. Por ejemplo un conjunto que no contenga todos sus puntos fronteras (también llamado objeto con *borde incompleto*), o que incluya puntos aislados y elementos colgantes no representa un sólido. Por esto Marcheix [3] introduce la noción de *regularización*:

Definición: La regularización r(A) de un conjunto de puntos A es definida por: *r(A)=cl (int (A))*,

con *cl(A)* e *int (A)* definiendo respectivamente las operaciones de cierre e interior de A. Un conjunto A que verifique r(A) = A se dice regular.

Es entonces posible definir la noción de R-conjunto de la manera siguiente:

Definición: un R-conjunto es un conjunto de puntos regulares y acotados.

- Homogeneidad de la frontera: la superficie del sólido debe ser lisa, sin variación ínfimamente fina. Por esta razón, los objetos manipulados son generalmente restringidos a aquellos cuya superficie puede ser representada de manera polinomial.

La ventaja de esta definición de los sólidos proviene del hecho que se puede utilizar el concepto de topología general (point-set topology) para caracterizar de una manera rigurosa las propiedades de un objeto tridimensional. Una cierta cantidad de modelos pudieron entonces desarrollarse sobre la base de estos diferentes criterios. Citemos por ejemplo [3], los modelos volumétricos del tipo “Marching Cube” muy utilizados en campos como la generación de imágenes médicas, el modelo CSG (Constructive Solid Geometry) y las operaciones booleanas regularizadas, el modelo de instansiación de las primitivas que permite parametrizar las formas de los objetos, y los modelos de descomposición en células (la descomposición celular, etc.) o de particionamiento del espacio (la representación octree o árbol octal, etc.) que permiten respectivamente la subdivisión del objeto o del espacio que contiene al objeto.

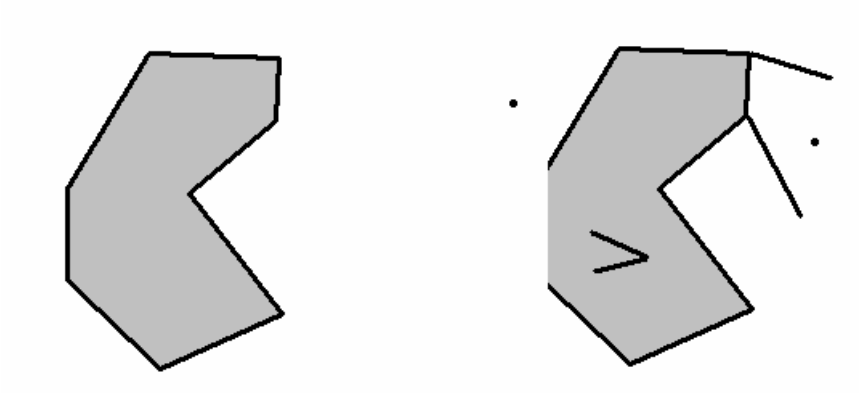


Figura 4.1: Un conjunto de puntos regulares y no regulares (según Marcheix [3])

Los modelos de particionamiento del espacio :

Estos métodos de representación, incluyendo entre otros el *modelo del árbol octal*, son más bien usados en el campo del análisis numérico, las bases de los datos geográficos y el tratamiento de imágenes. La idea básica, según explica Marcheix [3], consiste en representar los objetos a través de una subdivisión del espacio. En un primer tiempo, el espacio R3 que contiene al objeto es entonces subdividido con la ayuda de una rejilla tridimensional uniforme. En general esta rejilla es absolutamente regular, lo que conlleva a una discretización en elementos cúbicos del espacio inicial. En un segundo tiempo, los elementos que intersectan el interior del volumen son marcados, definiendo así una representación del objeto. La figura 4.2 da un ejemplo de subdivisión en elementos cúbicos, para el caso de un objeto relativamente simple.

Esta representación es muy costosa. También, con el fin de reducir la memoria necesaria, los *árboles octales* son frecuentemente usados para fusionar algunos elementos. Tomemos el ejemplo de una subdivisión cúbica. La idea consiste en reagrupar 8 cubos adyacentes que tengan el mismo tamaño en un solo cubo. Se obtiene de esta forma un cubo 8 veces más voluminoso. Este proceso se repite hasta que no existe ningún otro tipo de unión posible. (figura 4.2.a). Finalmente, estas relaciones de adyacencia son almacenadas en una estructura de árbol octal, donde los nodos representan una agregación de los cubos que no han sido marcados, y las hojas representan un cubo marcado, un cubo que no intersecta para nada el sólido, o un cubo donde se ha alcanzado el tamaño máximo de subdivisión (figura 4.2.c).

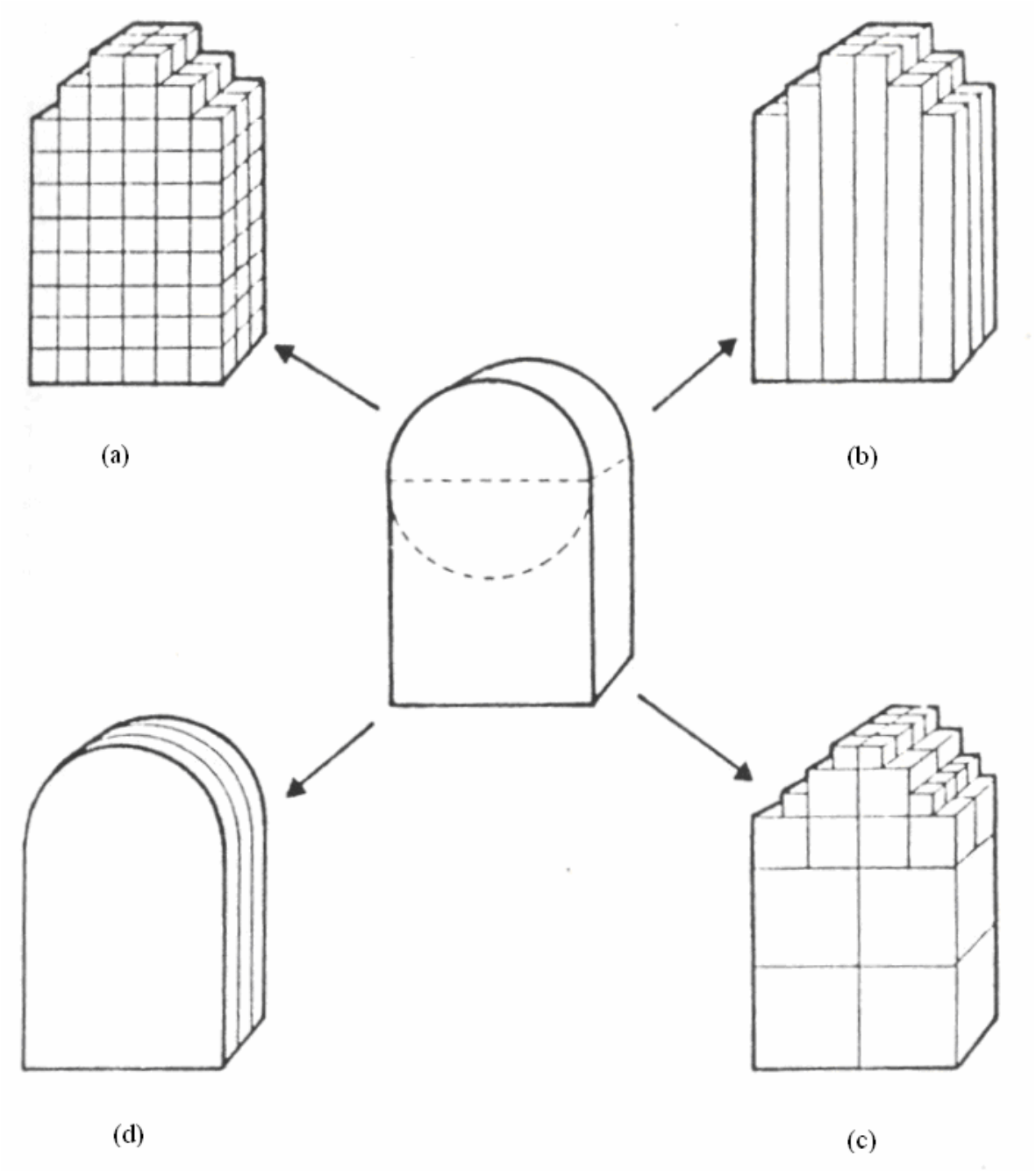


Figura 4.2: El modelo del árbol octal: El objeto matemático, y su representación (a,b,c,d), según Marcheix [3].

- Potencia: Cualquiera que sea la fineza de la rejilla de subdivisión utilizada al principio, la representación obtenida no define en general sino una aproximación a la forma real del objeto.

- Ambigüedad: Si no se toma en consideración ese aspecto aproximativo, todos los árboles octales definen un sólido de manera no ambigua.

- Unicidad: Para un nivel de resolución dada, el modelo es único.

- Validez: Todos los árboles octales son representaciones válidas de sólidos.

- Complejidad: Aunque se utilice un árbol octal, el reproche principal que se podría hacer a este método, tiene que ver con la cantidad de memoria relativamente importante que es necesaria para guardar la representación de un objeto.

- Cierre: Los modelos octree permiten el cierre de algoritmos tales como la rotación, la translación, las operaciones booleanas, etc.

-Aplicabilidad: Todas las propiedades de masa de un objeto (volumen, centro de inercia, etc.) son fácilmente calculables. No obstante, las operaciones de creación y de manipulación del modelo (rotación, etc.) son a menudo difíciles de realizar.

El modelo constructivo geométrico sólido:

Los modeladores que utilizan el modelo constructivo geométrico sólido (CSG-Constructive Solid Geometry) como base de trabajo son muy usados actualmente. Según Marcheix [3], el primer sistema de modelado que utilizó esta representación fue el sistema TIPS, desarrollado por Okino en la universidad de Hokkaido.

La idea inicial consiste en crear un objeto complejo combinando varias formas simples, llamadas primitivas (cubo, esfera, cono, etc.). Para determinar precisamente este subconjunto de puntos, es posible subdividir el espacio en semi espacios que permiten entonces definir de qué lado de la frontera se encuentran los puntos pertenecientes a la primitiva. Por ejemplo, un cubo será definido por la intersección de 6 semi-espacios de R3. La lista de operaciones de unión realizables entre dos elementos está generalmente restringida a las operaciones de base igualmente denominadas operaciones booleanas: unión, intersección y diferencia. De esta forma el objeto final será construido aplicando una serie finita de operaciones booleanas. Este proceso de

fabricación puede ser almacenado en un árbol binario en donde las hojas representen las primitivas de base, y cada nodo corresponde a una operación booleana. Para obtener la forma final, solo basta con evaluar el árbol mediante un recorrido fijado a posteriori que permite asociar a cada nodo un objeto intermedio en el proceso de construcción. La figura 4.3 representa el árbol CSG que permite crear un bolígrafo.

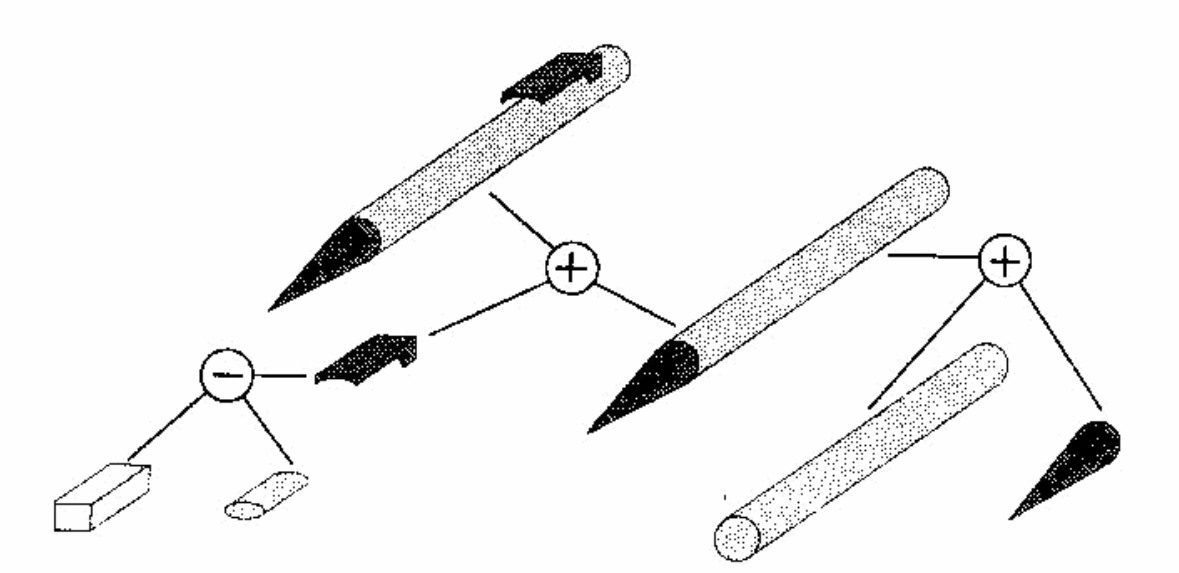


Figura 4.3: El modelo CSG (Según Marcheix [3]).

Potencia: Los objetos representables con la ayuda de este modelo, son por lo general limitados porque dependen de la clase de primitivas disponibles. No es fácil, por ejemplo, extender el dominio para cubrir objetos acotados por superficies de forma libre. En efecto, las operaciones booleanas sobre dichas superficies son relativamente complejas.

Ambigüedad: A cada árbol CSG no corresponde sino una sola interpretación física. El modelo CSG es entonces no ambiguo.

Unicidad: Esta representación no es única. Por ejemplo, El orden de aplicación de las operaciones de uniones es por lo general conmutativa.

Validez: Aunque la evaluación de un árbol CSG necesita algunas veces una interpretación particular, siempre y cuando las primitivas de base sean válidas, el resultado de esta evaluación corresponde siempre a un modelo válido cuando se utiliza las operaciones booleanas regularizadas.

Complejidad: La estructura de datos es relativamente simple y de talla reducida. No obstante, los modeladores que utilizan esta representación, guardan generalmente un cierto número de informaciones suplementarias en el modelo (por ejemplo para facilitar la visualización).

Cierre: La clase de los R-conjuntos forman un conjunto cerrado en relación a las operaciones booleanas regularizadas.

Aplicabilidad: Las informaciones explícitamente contenidas en el modelo no son numerosas. No obstante, son fáciles de manipular. Es muy fácil por ejemplo modificar los parámetros de base, definiendo la forma de los objetos. Para ello solo basta con cambiar algunos parámetros en el árbol de construcción. La figura 4.4 por ejemplo nos muestra dos árboles definiendo dos cubos agujereados por un cilindro, cuya única diferencia es la variación de un parámetro que define el radio del cilindro en el árbol CSG.

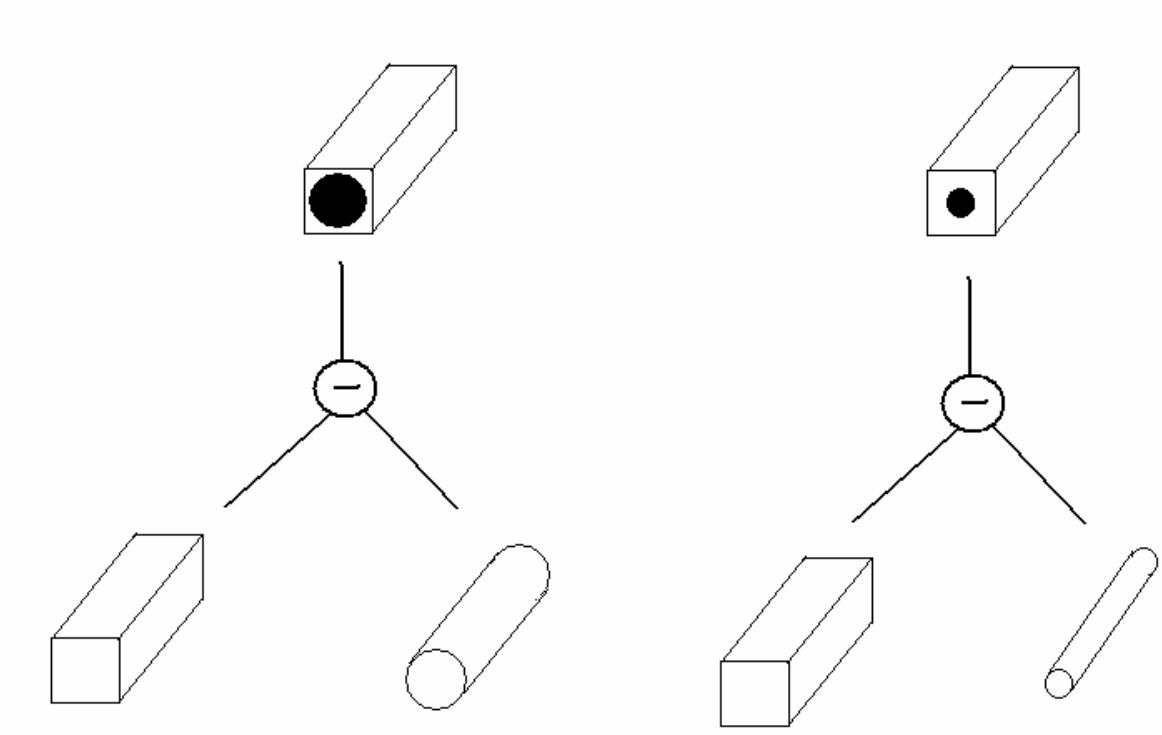


Figura 4.4: Modificación de un parámetro en un árbol de construcción (Según Marcheix [3]).

Por otra parte, la implementación de operaciones distintas a esas operaciones de base es generalmente difícil y limitada. Por ejemplo las modificaciones locales que son indispensables en un entorno interactivo, son por lo general complejas y a veces imposible de realizar. En efecto se hace necesario poder deformar cualquier parte del objeto sin estar en la obligación de reevaluarlo enteramente. Una consecuencia directa de esta apreciación es el costo elevado de los cálculos necesarios para la visualización de un objeto (modelo de alambre por ejemplo). Esto proviene del hecho de que las fronteras del objeto (caras, aristas, vértices) no son expresadas de manera explícita y por consecuencia se necesita reevaluar el árbol a cada vez que se requiera una información concerniente la superficie del sólido.

Una ventaja de la utilización de un árbol CSG es su posible conversión a una representación por frontera. Aunque esta propiedad sea teóricamente interesante, este tipo de práctica se ve restringida por dos razones principales: por un lugar el costo de tal aplicación es importante, y por otro lado, existe la limitación de que la conversión de manera inversa no siempre es realizable.

El modelado geométrico sólido por frontera:

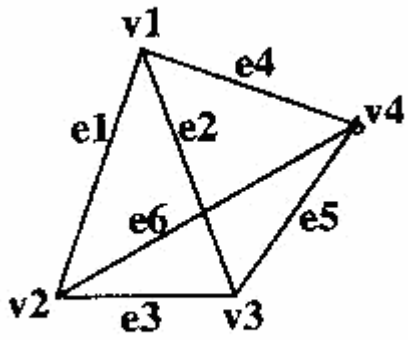
Como lo hemos visto hasta ahora, es posible representar un sólido describiendo el conjunto de puntos del espacio que el ocupa. Una segunda aproximación, consiste en definir únicamente el conjunto de puntos que conforman la frontera del objeto. Esta última puede ser considerada como una “piel” cerrada alrededor del sólido. Se habla entonces de modelado sólido por frontera (B-rep) (Boundary representation).

El Modelo de Alambre

Según Mäntylä [1], el Modelo de Alambre es una representación sólida por fronteras fácil y rápida de utilizar que permite representar por lo general objetos "simples" y en algunos casos ambiguos. La descripción de un objeto se realiza por puntos (vértices) y líneas (aristas que unen los vértices).

Este modelo fue utilizado por una gran cantidad de modeladores de concepción y fabricación asistida por computadora para definir sólidos. En realidad el modelo de Alambre puede considerarse como el esqueleto de dimensión 1 de un modelo sólido por frontera. Este modelo es de nivel inferior a los de superficie o los modelos de sólidos.

El principio de este modelo es la utilización de la arista como elemento de base de la representación. Se pueden conocer las extremidades de las aristas puesto que se tabula cada arista asociadas a sus respectivos vértices. En la figura 4.5 se muestra un tetraedro definido por su respectiva tabla de vértices y aristas.



aristas	vértices	
1	1	2
2	1	3
3	2	3
4	4	1
5	3	4
6	4	2

Figura 4.5: El modelo de alambre (Según Mäntylä [1]).

Este modelo presenta las características siguientes:

Ambigüedad: es el principal problema de este modelo. En efecto, para una misma representación pueden corresponder varios objetos reales diferentes. Además no es posible distinguir el exterior y el interior de un objeto.

Complejidad: La cantidad de memoria necesaria para esta representación es baja y las tablas son fáciles de crearse y de gestionarse. Por otro lado los accesos a los elementos que constituyen el objeto (aristas, vértices) son muy rápidos.

Aplicabilidad: Este modelo es difícil de utilizar por una persona que no tiene la costumbre de manipular objetos de alambre en un espacio 3D. Su interpretación no es necesariamente intuitiva. Además presenta la incapacidad de detectar interferencia entre objetos puesto que no reconoce las superficies de un objeto. También resulta incapaz para reconocer superficies curvas, porque las aristas, al ser rectas, no tienen la información suficiente para representar una superficia Curva.

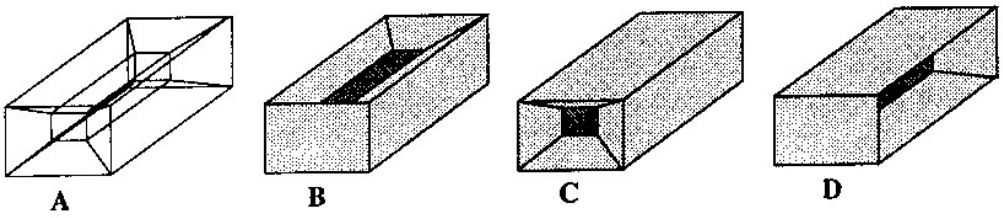
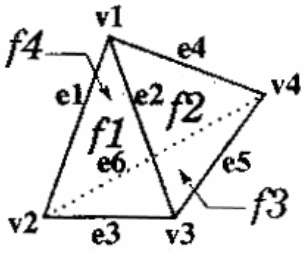


Figura 4.6: Ambigüedad del modelo de alambre (Según Mäntylä [1]).

Varios métodos que permiten reconocer y reconstruir algunos sólidos a partir del modelo de alambre han sido propuestos. Sin embargo, el hecho de que distintos sólidos pueden obtenerse a partir de una misma representación, plantea un problema de fondo que los algoritmos de más alto nivel no pueden resolver de manera satisfactoria.

El Modelo Poligonal

Este modelo es considerado, de acuerdo a lo expuesto por Mäntylä [1], como una evolución lógica del modelo de alambre. Está constituido por una lista de caras, cada una de las cuales es representada por una lista ordenada de aristas (El orden permite definir la representación de cada cara de manera no ambigua). La figura 4.7 muestra la lista de las caras asociadas a la representación de un tetraedro. En lo que respecta la estructura de datos, este modelo es generalmente representado por tablas similares a las usadas en el modelo de alambre, a las cuales se les agregan una lista ordenada de aristas que definen cada cara. En la mayoría de los casos cada una de las caras manipuladas son planas. De hecho las "caras" triangulares son frecuentemente usadas para representar la frontera de un objeto, puesto que contrariamente a las "caras" poligonales cualesquiera, estas ofrecen la ventaja de ser siempre planas.



aristas	vértices		
1	1	3	2
2	4	2	5
3	6	5	3
4	1	4	6

aristas	vértices	
1	1	2
2	1	3
3	2	3
4	4	1
5	3	4
6	4	2

Figura 4.7: El modelo poligonal (Según Mäntylä [1])

Algunos algoritmos utilizan este modelo como estándar de representación. En particular, podemos citar los trabajos de discretización de superficies de forma libre o de superficies implícitas con la ayuda del modelo poligonal con caras planas (triangulares en general).

Este modelo presenta las características siguientes:

Potencia: Puesto que en la mayoría de los casos este modelo utiliza caras planas (triangulares), las representaciones obtenidas por el mismo no son sino una aproximación de la superficies reales.

Ambigüedad: La definición explícita de las caras que caracterizan a un objeto permite la eliminación de la ambigüedad presentada en el modelo de alambre.

Unicidad: este modelo no es único puesto que la subdivisión de la superficie de un objeto no es única (caras triangulares, cuadrangulares, etc.).

Validez: la validez de este modelo es en general difícil de asegurar, puesto que esto necesita la verificación de la orientabilidad del conjunto de la superficie y el cálculo de las intersecciones entre las diferentes caras.

Complejidad: El acceso a los elementos de la estructura son muy rápidos (caras, aristas, vértices). En la medida en que solo las lista de aristas que definen cada cara son agregadas en relación al modelo de alambre, la estructura asociada a esta representación no se hace mucho más complicada, ni ocupa mucha más memoria.

Aplicabilidad: Algunos algoritmos son a veces pesados y difíciles de gestionar porque el modelo no contiene prácticamente relación de adyacencia.

Los Modelos de "Representación por Fronteras" (BREP):

Los modelos B-rep (Boundary representation) provienen de las representaciones poligonales y permiten dar una descripción relativamente completa de la superficie frontera del objeto (Mäntylä [1]) . Para esto la superficie es descompuesta de manera jerárquica en un conjunto de células de dimensión 2,1,y 0, llamadas respectivamente

caras, aristas, y vértices. La superficie del objeto debe de ser entonces subdividida de manera conveniente en un conjunto de caras orientadas. Generalmente la subdivisión es realizada de manera a obtener una representación matemática relativamente sencilla de cada cara (caras planas, cuadráticas, etc.). Por analogía, las caras son delimitadas por una lista cerrada de aristas (una lista que delimita el contorno externo de la cara, las otras pueden definir huecos), ellas mismas delimitadas por dos vértices. Generalmente, la representación de una tal descomposición permite conocer inmediatamente para cada célula la lista de células de dimensión inferior o superior que le son adyacentes; por ejemplo, encontrar la lista de caras adyacentes a un vértice, o la lista de vértices que delimitan una cara. En realidad existe en esta representación una distinción muy clara entre la topología y la geometría de un objeto.

- Por topología, se entiende las relaciones de adyacencia entre los elementos de distintas dimensiones (vértices, caras, aristas) que definen la subdivisión de la superficie frontera. Esta noción de adyacencia es muy importante. Por ejemplo ella permite calcular la característica de EULER, de probar la conectividad del objeto, definir la orientación de la superficie. En particularidad es útil cuando se desea conocer las adyacencias de algunos puntos.
- Por geometría, se entiende la profundidad de cada elemento topológico en el espacio R3.

Distintos métodos basados en la representación por fronteras fueron desarrollados. Citemos, por ejemplo, el modelo de aristas con alas, que en la actualidad parece ser el estándar en el campo de la representación por frontera de objetos que no contienen puntos singulares. El ejemplo de la figura 4.8 presenta un cubo y la estructura de arista aleteada asociada.

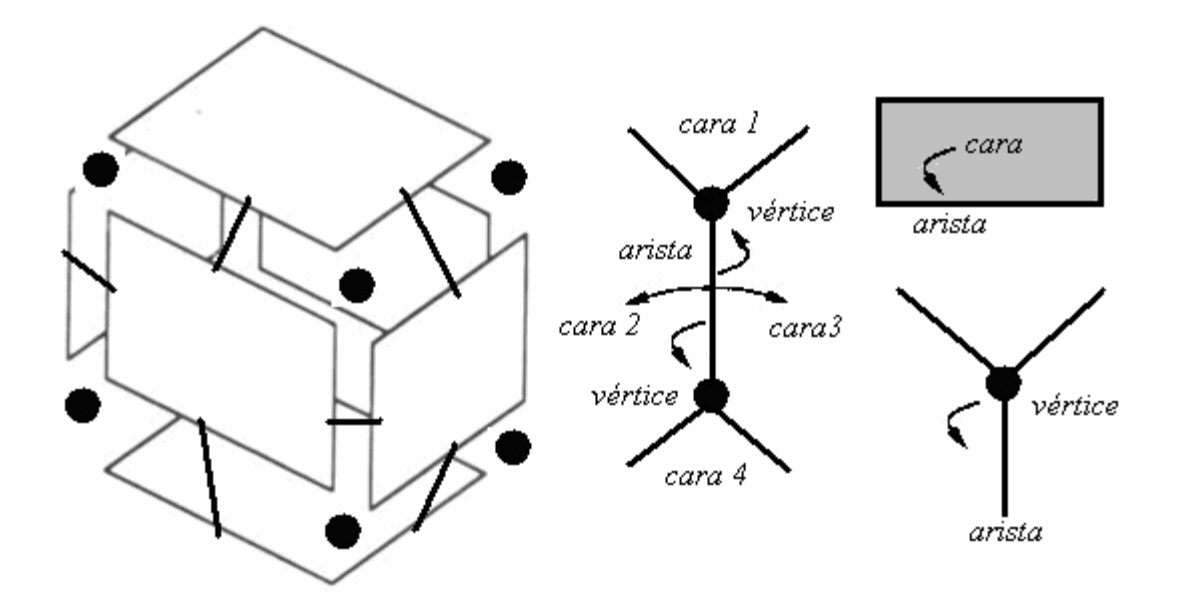


Figura 4.8: El modelo de representación por frontera (Brep), según Mäntylä [1]

Potencia: El espacio requerido para el modelo Brep está ligado a la clase de superficies que pueden ser utilizadas (planas y/o curvas). La diversidad de sólidos modelables parece ser más grande que con el modelo CSG.

Ambigüedad: la representación Brep es no ambigua.

Unicidad: Este modelo no es único ya que la subdivisión topológica de un mismo objeto no es única.

Validez: De igual manera que el modelo poligonal, la validez de la representación Brep es difícil de asegurar. Se distingue generalmente el aspecto topológico y geométrico. Por un costo razonable, la coherencia topológica de una representación es relativamente fácil de verificar. Sin embargo, es más difícil de garantizar la validez geométrica, sin penalizar la velocidad de ejecución.

Complejidad: una estructura más importante y más compleja, es el precio a pagar para una información topológica tan rica. Los algoritmos que manipulan esta representación serán por consiguiente también más complicados. Con la finalidad de resolver este problema, el acceso al modelo se realiza generalmente a través de operadores de bases que una vez escritos permiten, entre otras cosas, simplificar considerablemente los algoritmos.

Cierre: A nivel topológico, el modelo Brep es cerrado en relación a los operadores de Euler.

Aplicabilidad: Algunas operaciones son más complejas de realizar que con el modelo CSG. Por ejemplo, agrandar el tamaño de un cilindro necesita más cálculos que el simple cambio de un parámetro. En efecto será necesario recalcular el conjunto de coordenadas de los vértices y/o ecuaciones de aristas y caras que definen la profundidad geométrica del cilindro. Notemos que estas modificaciones son totalmente localizadas en una parte del objeto.

La información topológica registrada en el modelo es muy rica. Por ejemplo, es posible encontrar inmediatamente el conjunto de caras adyacentes a un vértice. La visualización del objeto es por consiguiente muy rápida. En efecto, contrariamente a la representación CSG, ningún cálculo es necesario para definir su frontera porque la misma ya está descrita explícitamente en el modelo.

Esta representación ofrece una gran libertad de manipulación (deformaciones locales, operaciones booleanas, extrusiones generalizadas, etc.), puesto que hay pocas restricciones en cuanto a las posibles operaciones. Será entonces muy fácil de insertar o de modificar las relaciones topológicas contenidas en el modelo.

El modelado geométrico sólido de objetos con puntos singulares

El modelo radial (desarrollado por Marcheix [3]):

Actualmente, una prioridad en el modelado geométrico es la representación de objetos de dimensiones no homogéneas con bordes incompletos teniendo ciertas particularidades tales como puntos singulares (picos, fisuras, aristas colgantes, fronteras internas o elementos aislados).

Estos objetos pueden ser:

- Primitivas en la construcción sólida.
- El resultado de operaciones booleanas.
- Las operaciones de barrido generalizadas.

En la figura 4.9, se muestra la operación de barrido rotacional generalizada sobre un círculo c, a lo largo de una trayectoria lineal T dadas por los segmentos T_iT_{i+1}, i=0,1…,7. Se puede apreciar que la definición incorrecta de los coeficientes de la escala en T_i, i= 2,3,4 provocan la degeneración del sólido S en un objeto de no variedad (non-manifold) Snv.

Fig.4.9: Operación de barrido generalizado sobre un círculo c (Marcheix [3]).

El objeto de la figura 4.10 es un objeto no homogéneo con bordes incompletos. Se distinguen tres partes: los sólidos A, C, y una superficie colgante B. Los elementos fronteras (representados en blanco), corresponden a fronteras internas (fint, eint,y Pint),

puntos de no variedad (env,Qnv y Rnv), y no son incluidos por el conjunto de puntos que el objeto ocupa.

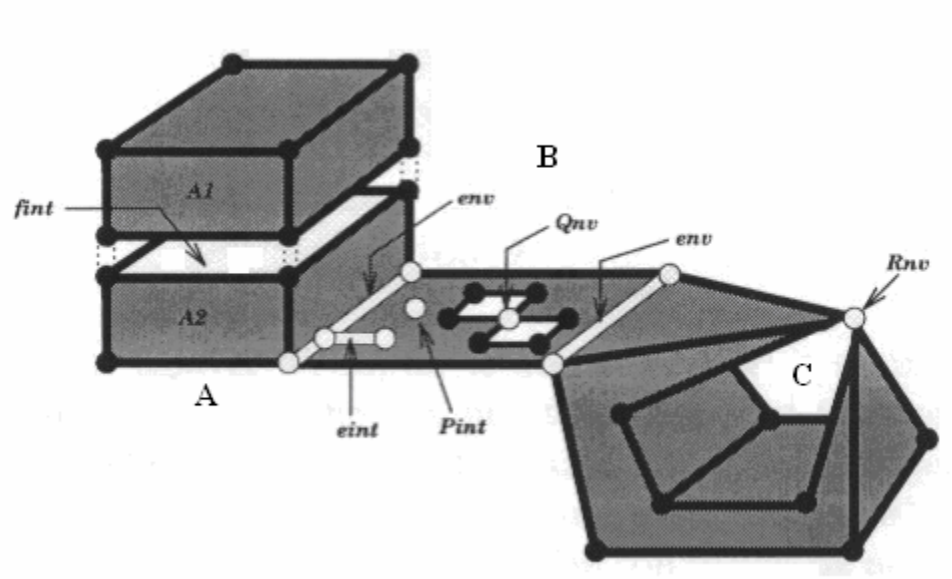


Figura 4.10: un objeto no homogéneo con bordes incompletos (Según Marcheix [3])

El modelo radial, desarrollado e implementado por Marcheix [3], se apoya en el modelo de representación por fronteras (BREP), con lo que se logra mantener la validez topológica del modelo ya que cualquier construcción y modificación se efectúan con la ayuda de los operadores de Euler.

Este modelo persigue entonces separar los puntos irregulares del resto del objeto; se introduce entonces la noción de radialidad para aislar cualquier particularidad.

En el capítulo V se describe la estructura de datos del modelo radial.

Ventajas del modelo radial:

- Permite representar objetos cuyos bordes son incompletos y que tienen ciertas particularidades, como puntos singulares, es decir, picos, fisuras, caras o aristas colgantes, fronteras internas o elementos aislados.
- Acepta la auto intersección de su superficie interior, preservando la estructura de datos interna.

- Hace posible la convergencia entre el modelado de sólidos y el modelado de superficies de forma libre.
- Permite describir y manipular la estructura interna de datos.
- Los operadores topológicos del modelo radial ofrecen gran libertad de interacción y modificación de la topología y de la geometría del objeto.
- El diseñador tiene la potestad de escoger como se va a fragmentar un objeto.
- Constituye una herramienta poderosa en el desarrollo de algoritmos para la generación automática de mallas tridimensionales utilizadas en los programas de análisis de esfuerzos.

CAPÍTULO V

ESTRUCTURA DE DATOS DE FRONTERA

Los constituyentes básicos de los modelos de frontera son los tipos de objetos *caras, aristas, y vértices,* y la información geométrica unida a ellos. Adicionalmente a la información geométrica tal como las ecuaciones de las caras, de las curvas, y las coordenadas de los vértices, un modelo de frontera debe también representar como las caras, aristas , y vértices están relacionadas unas con otras.

Todos los modelos de frontera representan caras en términos de nodos explícitos de una estructura de datos. Según Mäntylä [1], existen entonces varias alternativas para representar la geometría y la topología de un modelo de este tipo.

Modelo de frontera basado en aristas:

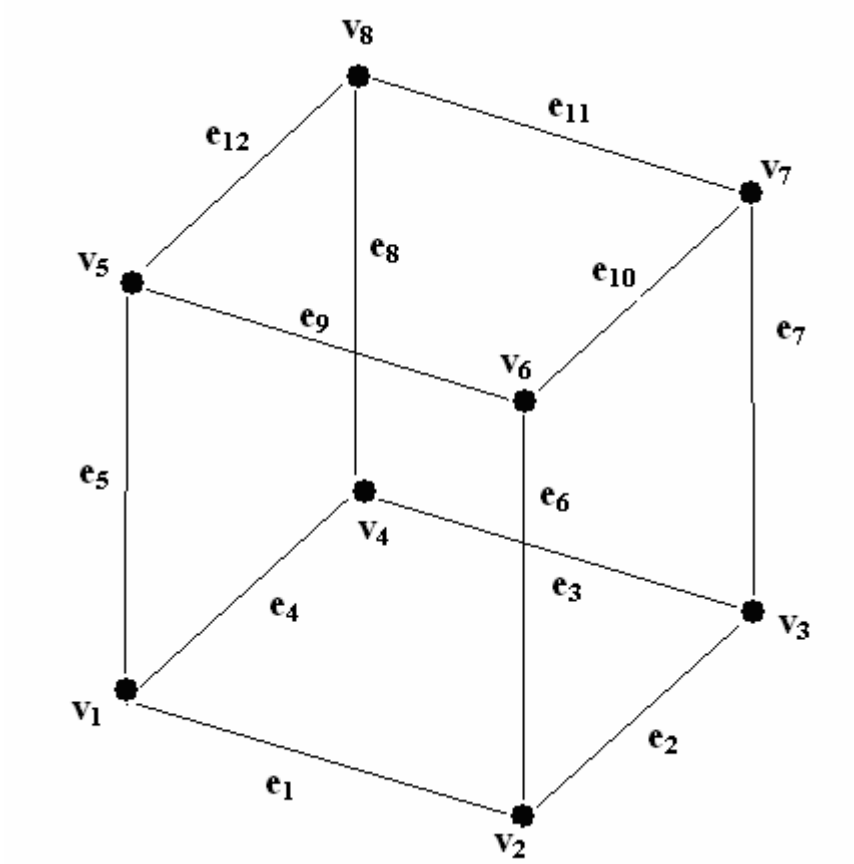
Este modelo representa una cara frontera en términos de una secuencia cerrada de aristas, lo que se denomina simplemente *lazo*. Los vértices de las caras son representados solamente a través de aristas.

En la figura 5.1 se muestra la estructura de datos de este modelo para un simple cubo.

Dicha estructura indica una orientación para cada arista; es decir, se considera que la arista e_1 está orientada positivamente desde el vértice v_1 al vértice v_2.

Las caras están orientadas en sentido horario, visto desde afuera del sólido.

Nótese que cada arista ocurre en exactamente dos caras, una en su orientación positiva, y la otra en su orientación negativa.



arista	***vértices***
e_1	v_1 v_2
e_2	v_2 v_3
e_3	v_3 v_4
e_4	v_4 v_1
e_5	v_1 v_5
e_6	v_2 v_6
e_7	v_3 v_7
e_8	v_4 v_8
e_9	v_5 v_6
e_{10}	v_6 v_7
e_{11}	v_7 v_8
e_{12}	v_8 v_5

vértice	***coordenadas***
v_1	x_1 y_1 z_1
v_2	x_2 y_2 z_2
v_3	x_3 y_3 z_3
v_4	x_4 y_4 z_4
v_5	x_5 y_5 z_5
v_6	x_6 y_6 z_6
v_7	x_7 y_7 z_7
v_8	x_8 y_8 z_8

cara	***arista***
f_1	e_1 e_2 e_3 e_4
f_2	e_9 e_6 e_1 e_5
f_3	e_{10} e_7 e_2 e_6
f_4	e_{11} e_8 e_3 e_7
f_5	e_{12} e_5 e_4 e_8
f_6	e_{12} e_{11} e_{10} e_9

Figura 5.1: Estructura de datos para un modelo basado en aristas (Según Mäntylä [1])

Modelo de frontera basado en estructura de datos de arista con alas:

Esta estructura de datos (ver figura 5.2) es más avanzada que la estructura basada en aristas, según lo expuesto por Mäntylä [1], puesto que también incluye información explícita de cada objeto básico (caras, aristas, vértices) a través de nodos relacionados entre sí, que permiten determinar, por ejemplo, cual arista perteneciente a un lazo de una cara le sigue a otra arista, o cual es la arista referencial de una cara, es decir, cual es la arista que inicia el lazo de una cara.

De igual forma se puede saber, a partir de una arista (la cual pertenece a dos caras simultáneamente), cual es arista que le sigue recorriendo una de las caras en sentido horario, o cual es la arista que le sigue haciendo un recorrido antihorario de la otra cara.

Por otra parte, las caras solamente necesitan la inclusión de una arista arbitraria inicial, y de un signo (+ o -) que acompañe a la arista, de tal forma que dependiendo del signo, se sepa qué cara se está identificando.

Modelo de frontera basado en estructura de datos de arista radial:

La estructura de datos de arista radial, desarrollada por Marcheix [3], y utilizada para la implementación del modelo radial, es presentada bajo la forma de un gráfico de incidencia descrito en la figura 5.3a.

El objeto se compone de tres primitivas (figura 5.3b): un hilo de alambre P_1, un tetraedro P_2, y una cara colgante P_3. Las condiciones de no variedad (non-manifold) son codificadas con uniones radiales a nivel del vértice V_{rad}, para V_2 y V_3, y al nivel de arista con e_{rad}, para e_3 y e_{10}.

En la estructura de datos de arista radial, se introducen nuevas nociones como la de semivértice, semiarista, y semicara. Una cara colgante, por ejemplo, poseerá dos orientaciones y en consecuencia dos semicaras, mientras que una cara que pertenezca a una primitiva volumétrica será constituida únicamente por una semicara que define el interior de la primitiva. De manera análoga se introduce los elementos *loop, loop radial* y *semiloop* para facilitar la manipulación de aristas adyacentes.

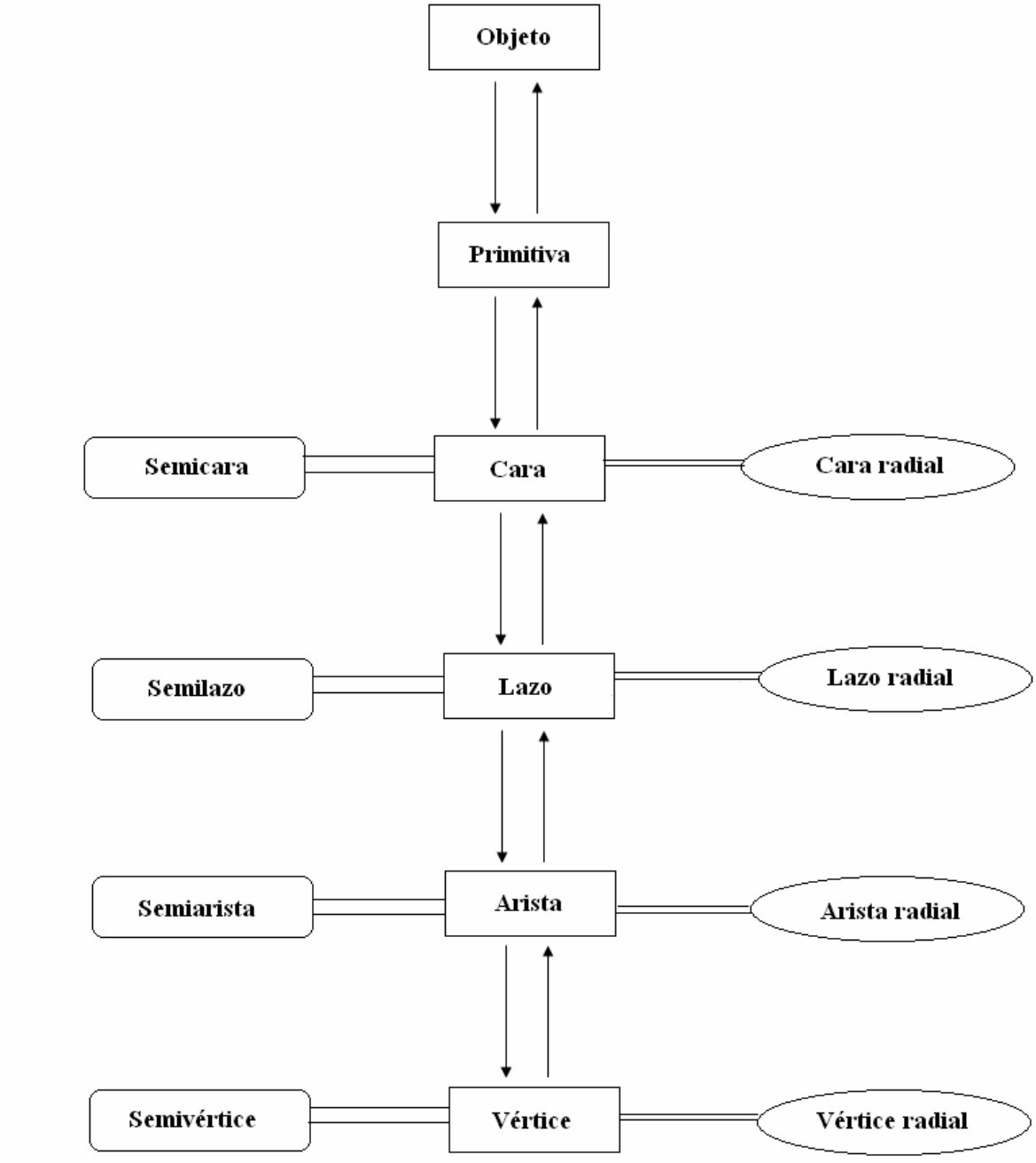
La figura 5.4 muestra como se intersectan múltiples caras en una arista radial.

arista	*v.inicial*	*v.final*	*sig_hor*	*sig_anti_hor*
e_1	v_1	v_2	e_2	e_5
e_2	v_2	v_3	e_3	e_6
e_3	v_3	v_4	e_4	e_7
e_4	v_4	v_1	e_1	e_8
e_5	v_1	v_5	e_9	e_4
e_6	v_2	v_6	e_{10}	e_1
e_7	v_3	v_7	e_{11}	e_2
e_8	v_4	v_8	e_{12}	e_3
e_9	v_5	v_6	e_6	E_{12}
e_{10}	v_6	v_7	e_7	e_9
e_{11}	v_7	v_8	e_8	e_{10}
e_{12}	v_8	v_5	e_5	E_{11}

vértice	*coordenadas*
v_1	$x_1\ y_1\ z_1$
v_2	$x_2\ y_2\ z_2$
v_3	$x_3\ y_3\ z_3$
v_4	$x_4\ y_4\ z_4$
v_5	$x_5\ y_5\ z_5$
v_6	$x_6\ y_6\ z_6$
v_7	$x_7\ y_7\ z_7$
v_8	$x_8\ y_8\ z_8$

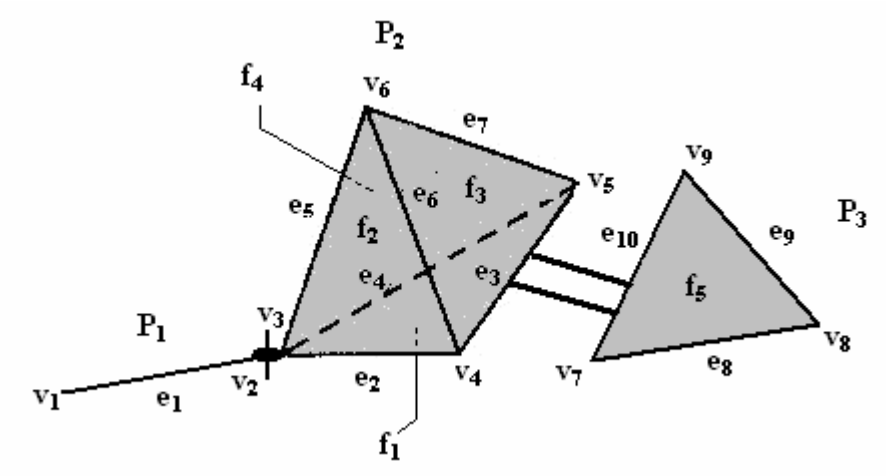
Cara	*prim_arista*	*signo*
f_1	e_1	+
f_2	e_9	+
f_3	e_6	+
f_4	e_7	+
f_5	e_{12}	+
f_6	e_9	-

Figura 5.2: Estructura de datos de arista con alas (Según Mäntylä [1])



(a)

Figura 5.3: Estructura de datos del modelo radial (Según Marcheix [3])



(b)

Figura 5.3: Estructura de datos del modelo radial (Continuación)

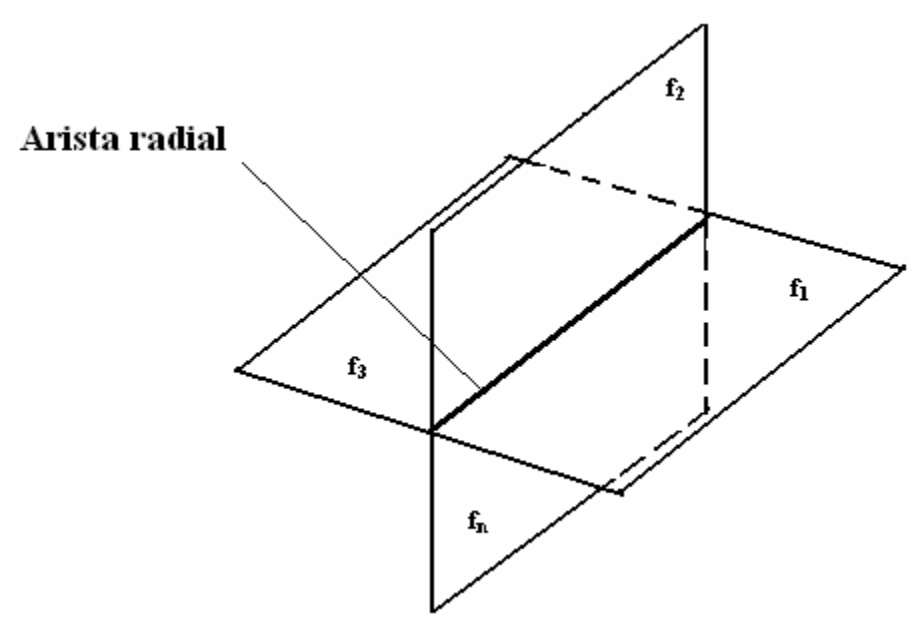


Figura 5.4: Múltiples caras se intersectan en una arista radial (Según Marcheix [3])

CAPÍTULO VI

ARQUITECTURA DEL PROGRAMA

La arquitectura del Programa SMS_EIM_UCV, desarrollado en el presente trabajo especial de grado, se fundamenta principalmente en la representación interna de datos llamada *Estructura de Datos de Semiarista*, propuesta por Mäntylä [1], y un conjunto de operaciones de bajo nivel para su manipulación. Dicha estructura se detallará en el capítulo siguiente.

La figura 6.1 muestra la arquitectura del programa desarrollado, teniendo entonces como parte central y fundamental la *Estructura de Datos de Semiarista*.

Figura 6.1: Arquitectura del programa SMS_EIM_UCV

Las siguientes capas del programa consisten en los *Operadores de Euler* que se describirán en el Capítulo VIII, y un conjunto de *Procedimientos Geométricos* básicos. Los operadores de Euler forman los procedimientos básicos de modelado del programa SMS_EIM_UCV que se usarán para implementar todas las operaciones de mayor nivel.

.

La capa siguiente (*Herramientas para Crear Sólidos)* consiste en varias herramientas de mediano nivel que permiten la creación de modelos sólidos mediante procedimientos preestablecidos tales como el barrido traslacional, el barrido rotacional, es decir que estas operaciones permiten crear distintas primitivas de sólidos. (por ejemplo, una esfera puede crearse a partir de una primitiva que se apoye en una operación de barrido rotacional).

Estos procedimientos (barrido traslacional, barrido rotacional, etc...) los denominaremos *herramientas para describir modelos.*

La capa siguiente (*Representación Gráfica*), permite obtener en pantalla gráfica el sólido en 3D (tres dimensiones). Además se incluyen varias *herramientas de manipulación de modelos* tales como acercamiento/alejamiento (Zoom), rotación de un sólido alrededor de los ejes de coordenadas, etc...

Otra capa (*Representación Matemática*): esta representación no es visible al usuario, pero permite obtener la información necesaria (coordenadas e identificación de los vértices, ecuaciones de las caras) para calcular cualquier propiedad geométrica (volumen y área superficial del sólido, masa, centro de gravedad, etc…) en trabajos posteriores.

La última capa del programa consiste en la implementación de una *Interfaz de Usuario* a través de la cual las herramientas de modelado de los distintos niveles son finalmente puestas a disposición para el usuario.

CAPÍTULO VII

ESTRUCTURA DE DATOS DE SEMIARISTA

Estructura de datos de semiarista:

A continuación se explicará la estructura de datos de semiarista, propuesta por Mäntylä [1] y desarrollada por Uzcátegui y Uzcátegui [2], la cual constituye el esqueleto del programa SMS_EIM_UCV desarrollado en el presente trabajo.

Representación gráfica:

La estructura de datos de semiarista se basa en una jerarquía de datos de cinco niveles, los cuales forman tipos de objetos llamado nodos, como por ejemplo:

- Nodo Sólido.
- Nodo Cára.
- Nodo Lazo.
- Nodo Semi-arista.
- Nodo Vértice.

Descripción de los nodos:

Sólido: El nodo *Sólido* administra la estructura de datos de semiarista de un objeto sólido. Para tener acceso a todos los datos que constituyen a un sólido, se necesita un apuntador a dicho sólido que permita identificarlo. El nodo Sólido da acceso a las caras, aristas, y vértices del modelo a través de apuntadores a tres listas doblemente entrelazadas. Todos los sólidos están relacionados mediante una lista de doble enlace conectadas mediante apuntadores al sólido siguiente o anterior de la lista.

Cara: El nodo *Cara* representa una cara plana de un poliedro representado por la estructura de datos de semiarista. En dicha estructura, se define una cara como un polígono plano cuyo interior está conectado. Resulta conveniente incluir caras con

múltiples fronteras dentro de la estructura; de hecho cada cara es asociada con una lista de lazos, cada uno de los cuales representa una frontera poligonal de la cara.

Puesto que todas las caras representan polígonos planos, uno de los lazos puede diseñarse como la frontera "externa ", mientras que los otros representan los agujeros de la cara. Esto se realiza en términos de dos apuntadores a un nodo lazo, uno que apunta al primer lazo en la lista doblemente enlazada, que consiste en todos los lazos de la cara.

Se tomará por convención llamar a los lazos-agujeros *anillos.*

Una Cara tiene un vector de cuatro puntos flotantes que representan su ecuación plana. Para realizar la lista doblemente enlazada de todas las caras del sólido, cada cara incluye apuntadores a la cara anterior o a la cara siguiente de la lista. Finalmente, cada cara tiene un apuntador hacia su sólido madre.

Lazo: El nodo lazo describe una frontera conectada tal y como se discutió anteriormente. Tiene un apuntador a su cara madre, un apuntador a una de las semiaristas que forma la frontera, y apuntadores al lazo siguiente y al lazo anterior de la cara.

Semiarista: El nodo semiarista describe un segmento de línea de un lazo. Consiste en un apuntador a su lazo madre, y un apuntador al vértice inicial del segmento de línea en la dirección del lazo. Apuntadores a las semiaristas anteriores y siguientes constituyen una lista doblemente enlazada de semiaristas de un lazo; de hecho el vértice final del segmento de línea representa el vértice inicial de la semiarista siguiente.

Vértice: El nodo vértice contiene un vector de cuatro puntos flotantes que representa un punto en sus coordenadas homogéneas. Dos apuntadores al vértice siguiente y al vértice anterior completan la lista de doble enlace de los vértices del sólido.

La jerarquía de los datos de la estructura de semiarista se muestra en la figura 7.1.

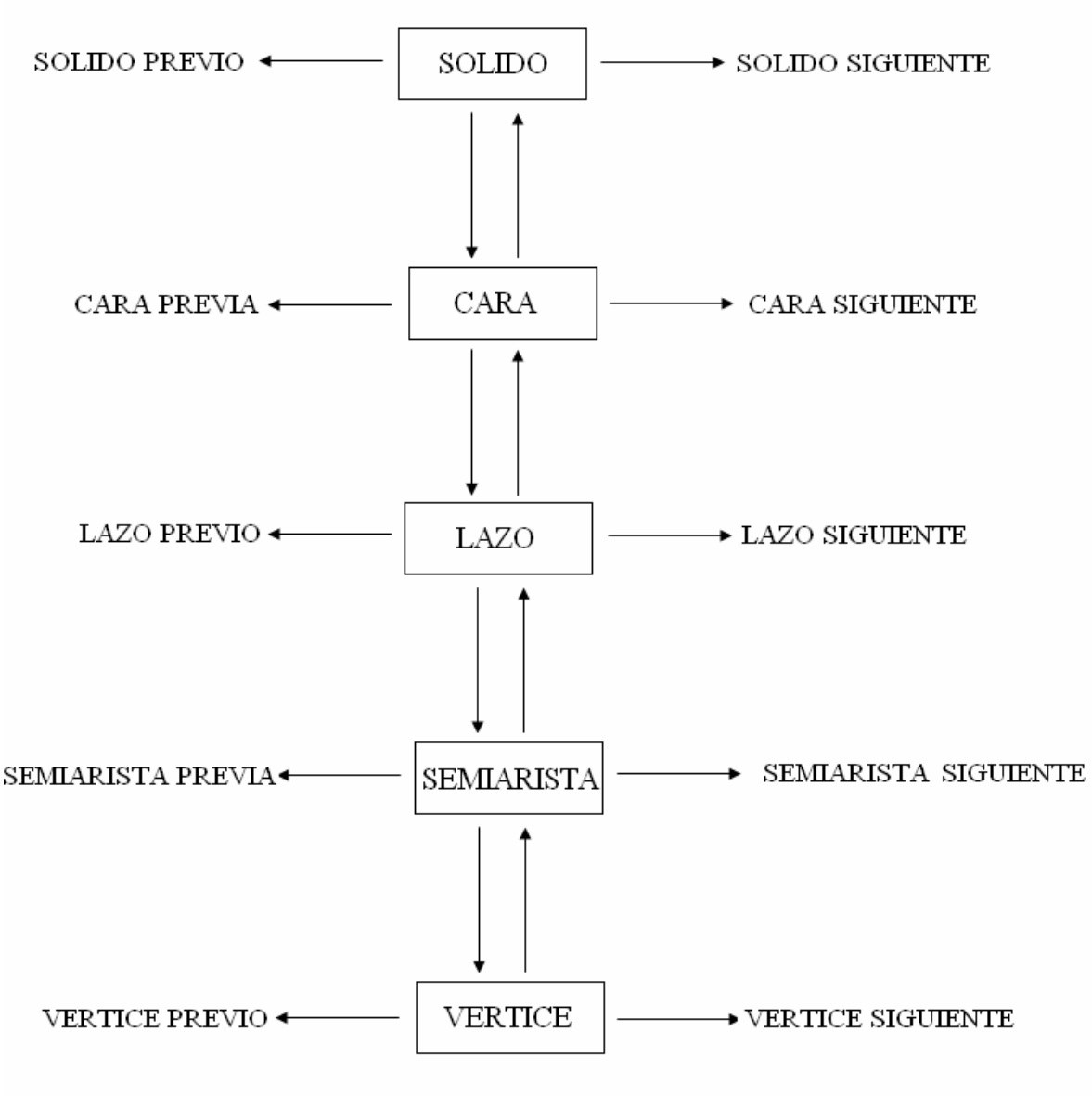


Figura 7.1: Jerarquía de datos de semiarista (Según Mäntylä [1])

Identificación de la representación:

La jerarquía estricta presentada anteriormente no indica directamente cómo las caras individuales están relacionadas una de otras, excepto cuando varios polígonos se refieren al mismo nodo vértice. Se necesita entonces agregar un tipo de nodo que haga una relación cara-cara explícita, representando la identificación de sus segmentos de línea.

Recordemos que cada arista de un modelo plano válido se identifica únicamente con otra arista. De forma similar, cada semiarista debería ser asociada con exactamente otra semiarista. Se necesita usar entonces un tipo de nodo adicional *Arista* para registrar esa información. Como se describirá a continuación, se agregará entonces una data suplementaria en algunos nodos de la jerarquía para explotar completamente la información de identificación.

Arista: El nodo arista asocia dos semiaristas una con la otra; intuitivamente, combina dos mitades juntas para formar una arista. Consiste en apuntadores hacia la semiarista "izquierda " y hacia la semiarista "derecha". La lista doblemente enlazada de una arista se completa con apuntadores hacia la arista siguiente y la arista anterior.

Semiarista: cada semiarista incluye un apuntador adicional hacia su arista madre.

Vértice: cada vértice incluye un apuntador adicional hacia una de las semiaristas que emanan de él. Para representar la identificación de los vértices, todas las caras que tienen una esquina en común en un punto en particular (a través de lazos y semiaristas) deben referirse a un nodo vértice simple que corresponde a ese punto.

La representación de la identificación de la información se muestra en la figura 7.2 que además señala los identificadores usados en lenguaje C de varios de los apuntadores requeridos. Los tres nodos semiarista, arista, y vértice forman la parte central de la estructura de datos que se utilizará para el Programa SMS_EIM_UCV.

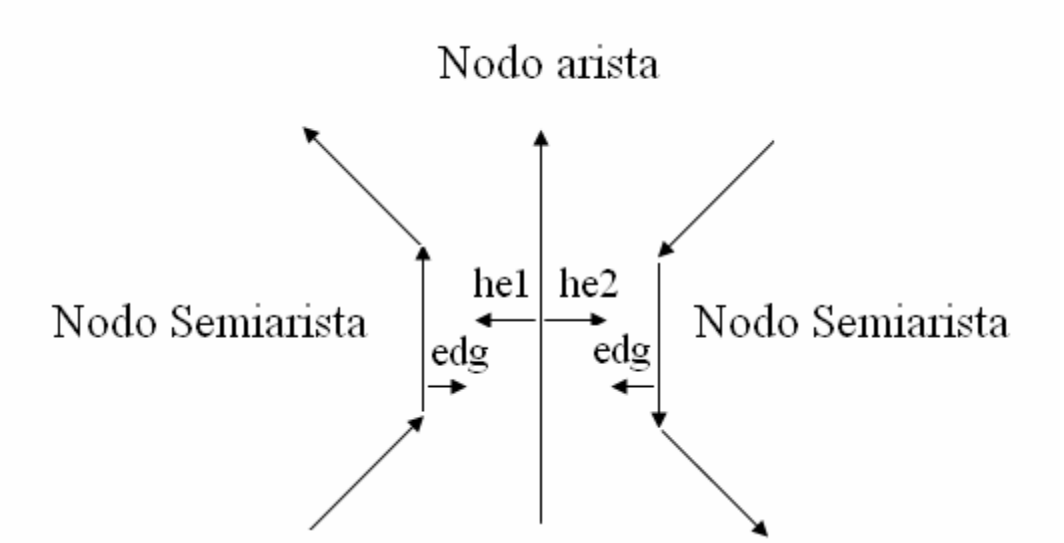


Figura 7.2: Identificadores usados en Lenguaje C (Según Mäntylä [1])

Lazos vacíos:

Resulta útil considerar un lazo vació que solo contiene un vértice, pero ninguna arista. En la estructura de datos de semiarista, los lazos vacíos se representarán en términos de una semiarista cuyo vértice apuntador se dirige hacia el vértice simple, y cuyo apuntador arista es NULL (arista nula).

El apuntador a la semiarista anterior o siguiente se refiere a su propia semiarista. Estas semiaristas excepcionales no corresponden con segmentos de línea.

A continuación se muestran las definiciones de las estructuras en lenguaje C que permiten crear los nodos anteriormente descritos:

```
struct solid
{
   Id         snum;   /*identificador del sólido*/
   Face      *sfaces; /*apuntador a una lista de caras*/
   Edge      *sedges; /*apuntador a una lista de aristas*/
   Vertex    *sverts; /*apuntador a una lista de vértices*/
   Solid     nexts;   /*apuntador al siguiente sólido*/
   Solid     prev;    /*apuntador al sólido anterior*/
};

struct face
{
   Id         fnum;     /*identificador de cara*/
   Solid     *fsolid;   /*apuntador al sólido madre*/
   Loop      *flout;    /*apuntador al lazo externo*/
   Loop      *flint;    /*apuntador a una lista de lazos internos*/
   Vector     feq;      /*ecuación de la cara*/
   Face      *nextf;    /*apuntador a la cara siguiente*/
   Face      *prevf;    /*apuntador a la cara  anterior*/
  };

struct loop
{
   HalfEdge  *lhedg;    /*apuntador a un lazo de semiaristas*/
   Face      *lface;    /*apuntador de la cara madre*/
   Loop      *nextl;    /*apuntador al lazo siguiente*/
   Loop      *prevl;    /*apuntador al lazo anterior*/
};

struct edge
{
   HalfEdge  *he1;      /*apuntador a la semiarista derecha*/
   HalfEdge  *he2;      /*apuntador a la semiarista izquierda*/
   Edge      *nexte;    /*apuntador a la arista siguiente*/
   Edge      *preve;    /*apuntador a la arista previa*/
};
```

```
struct halfedge
{
   Edge      *edg;      /*apuntador a la arista madre*/
   Vertex    *vtx;      /*apuntador al vértice inicial*/
   Loop      *wloop;    /*apuntador de regreso al lazo*/
   HalfEdge  *nxt;      /*apuntador a la siguiente semiarista*/
   HalfEdge  *prv;      /*apuntador a la semiarista anterior*/
};

struct vertex
{
   Id         vnum;     /*identificador de vértice*/
   HalfEdge  *vedge;    /*apuntador a una semiarista*/
   Vector     vcord;    /*coordenadas del vértice*/
   Vector     vnorm;
   Vertex    *nextv;    /*apuntador al vértice siguiente*/
   Vertex    *prevv;    /*apuntador al vértice anterior*/
};
```

CAPÍTULO VIII

OPERADORES DE EULER

Los operadores de Euler tienen un rol central en la arquitectura del programa SMS_EIM_UCV; estos constituyen un conjunto pequeño de operaciones que permiten la manipulación de modelos planos y son altamente poderosos ya que permiten describir cualquier modelo plano que represente un objeto sólido físico.

Toda creación de un sólido se inicia entonces a partir de un modelo plano tipo esqueleto el cual se constituye de un punto, una cara (aunque aún no esté definida por todos sus vértices), y un sólido (cuya construcción se definirá mediante operaciones posteriores.)

En la figura 8.1 (Mäntylä [1]) se muestra como comienza la creación de un cubo a partir de un modelo plano tipo esqueleto. El operador que permite realizar dicho esqueleto inicial se detallará más adelante.

Notaciones y convenciones:

Según Mäntylä [1], los operadores de Euler fueron introducidos originalmente por Baumgart en la década de los 70 en sus trabajos desarrollados en modelado geométrico computacional en la Universidad de Stanford en el contexto de la estructura de datos de arista con alas descrita en el Capitulo V del presente trabajo.

Antes de presentar los operadores de Euler, se usará por conveniencia de lenguaje el término anillo para referirse al lazo interno de una cara. Otra convención que se tomará de ahora en adelante es la de adoptar *lazos vacíos* en la estructura de datos, los cuales son constituidos por un solo vértice sin ninguna arista en absoluto.

Por convención histórica, los operadores son usualmente identificados por nombres abreviados; dichas abreviaturas se usarán de ahora en adelante en inglés con la finalidad de preservar las notaciones y convenciones de Baumgart.

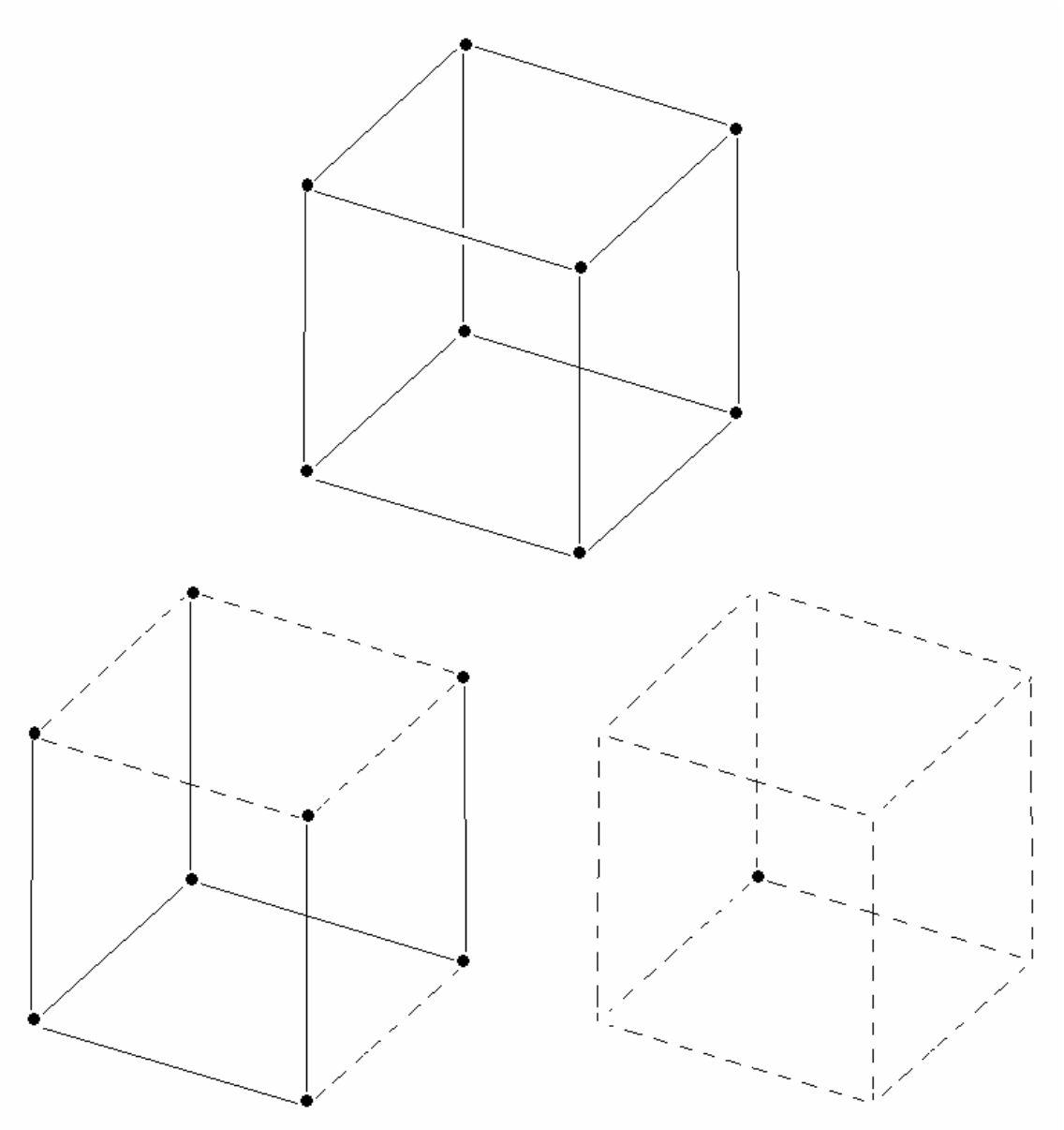


Figura 8.1: Reducción de un cubo a su modelo plano tipo esqueleto (Según Mäntylä [1])

A continuación se presentan los nombres que se utilizarán en el presente trabajo:

M-- make	V-- vertex	H-- hole
K-- kill	E-- edge	R-- ring
S-- split	F-- face	
J-- join	S-- solid	

Por ejemplo, la conjunción del nombre *mev* debe interpretarse como "make edge, vertex", es decir en español: "crea arista y vértice".

Primitivas para crear o eliminar un esqueleto: MVFS y KVFS

El operador MVFS, explica Mäntylä [1], tiene por función la de crear el modelo plano tipo esqueleto (descrito anteriormente), el cual genera un nodo sólido cuya estructura de datos solo contiene una cara un solo vértice. De hecho la nueva cara tiene un lazo vacío sin ninguna arista en absoluto. El efecto del operador MVFS puede verse en la figura 8.2.

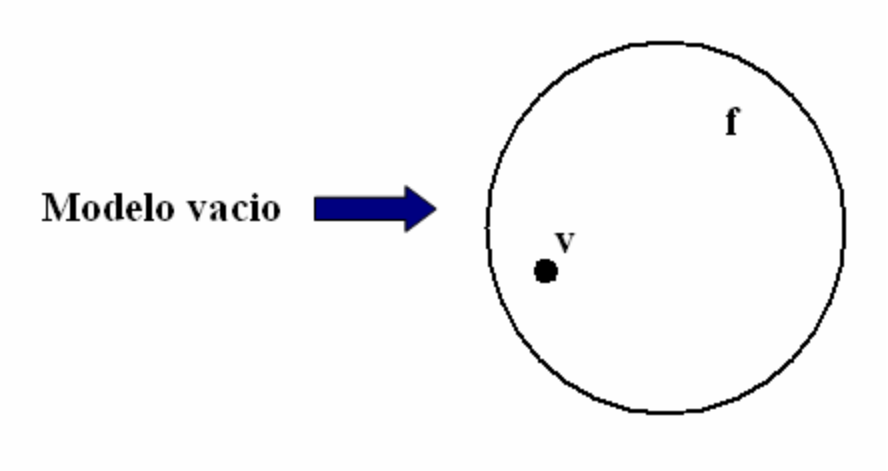


Figura 8.2: Efecto del operador MVFS (Según Mäntylä [1])

Es importante recalcar que el sólido creado por este operador no satisface la noción intuitiva que se tiene de un objeto sólido. Sin embargo, la aplicación de dicho

operador es el primer paso para crear un modelo por frontera con una secuencia de operadores de Euler que a continuación se describirán.

Todos lo operadores de Euler tienen su correspondientes operadores inversos que deshacen el efecto de un operador "positivo". El inverso de MVFS se llama KVFS, y destruye un nodo sólido tipo esqueleto eliminando su estructura de datos.

MEV, KEV

El operador MEV permite subdividir el ciclo de aristas de un vértice en dos ciclos creando un nuevo vértice, el cual se une al vértice anterior mediante una nueva arista. El efecto neto resulta en la adición de un vértice y de una arista a la estructura de dato.

La figura 8.3 muestra el resultado de la aplicación de este operador.

Figura 8.3: Efecto del operador MEV (Según Mäntylä [1])

El operador inverso KEV elimina de la estructura de datos el vértice nuevo y la arista que lo unía al vértice anterior, fusionando los ciclos de ambos vértices en un solo.

MEF, KEF

El operador MEF, según explica Mäntylä [1], subdivide un lazo uniendo dos vértices con una nueva arista; su efecto neto es el de agregar una nueva arista y cara en la estructura de datos (ver figura 8.4).

De nuevo, el operador inverso KEF puede deshacer el efecto de MEF. De hecho, en el caso de que una arista sea común a dos caras, KEF es capaz de remover la arista, y unir las dos caras en una sola cuyo lazo frontera sea el resultado de fusionar las dos fronteras originales.

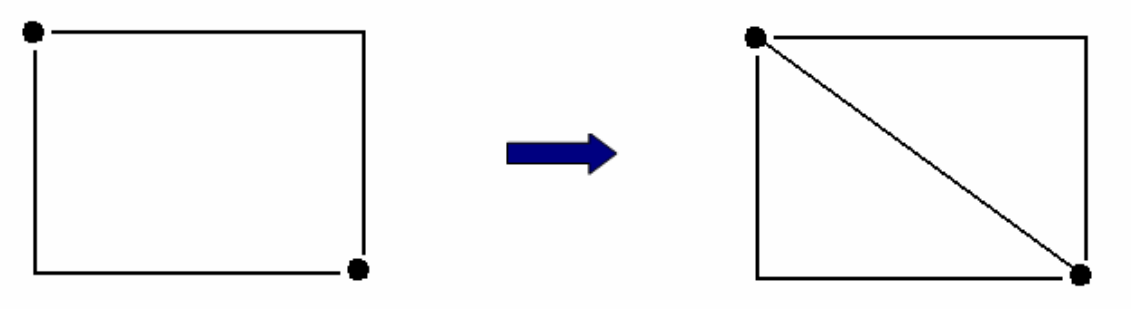


Figura 8.4: Efecto del operador MEF (Según Mäntylä [1])

KEMR, MEKR

Hasta el momento, los lazos vacíos han sido utilizados para la realización del esqueleto del modelo plano en la estructura de datos por frontera. Para explotar aún más la utilización de dichos lazos, se introduce un operador especialmente concebido para su creación.

El operador resultante KEMR transforma un lazo en dos nuevos lazos retirando una arista que aparece dos veces en el. Por consiguiente, KEMR tiene como efecto directo el de remover una arista de una cara y agregar un lazo a la estructura de datos.

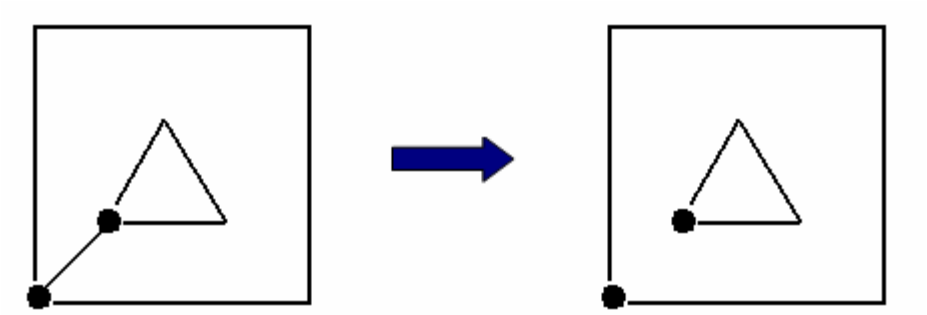


Figura 8.5: Efecto del operador KEMR (Según Mäntylä [1])

El operador inverso MEKR puede fusionar dos lazos de una cara uniendo dos vértices de cada uno de los lazos con una nueva arista.

Ejemplo de aplicación de los operadores de Euler:

A continuación se presentará la construcción paso a paso de un cubo para ejemplarizar la utilización de los operadores y mostrar su potencialidad al momento de representar sólidos mediante sus fronteras.

Dicha construcción se detalla en la figura 8.6 (Mäntylä [1]) en términos de varios modelos planos y modelos espaciales tridimensionales.

La construcción inicia creando un modelo tipo esqueleto mediante el operador MVFS (a). Aplicando tres veces seguidas MEV, tres aristas y vértices son agregadas al sólido para formar (b). Uniendo el último vértice creado, al primero, mediante MEF, se obtiene el modelo mostrado en (c). Nótese que el modelo espacial de (c) consiste en dos caras planas superpuestas como los dos lados de una hoja de papel. Este tipo de modelo se denomina *lámina.*

Cuatro MEV conducen al modelo (d) que contiene las aristas laterales del modelo emergente del cubo. Las caras laterales son formadas por cuatro MEF (e,f,g,h).

Queda entonces definido el modelo espacial del cubo.

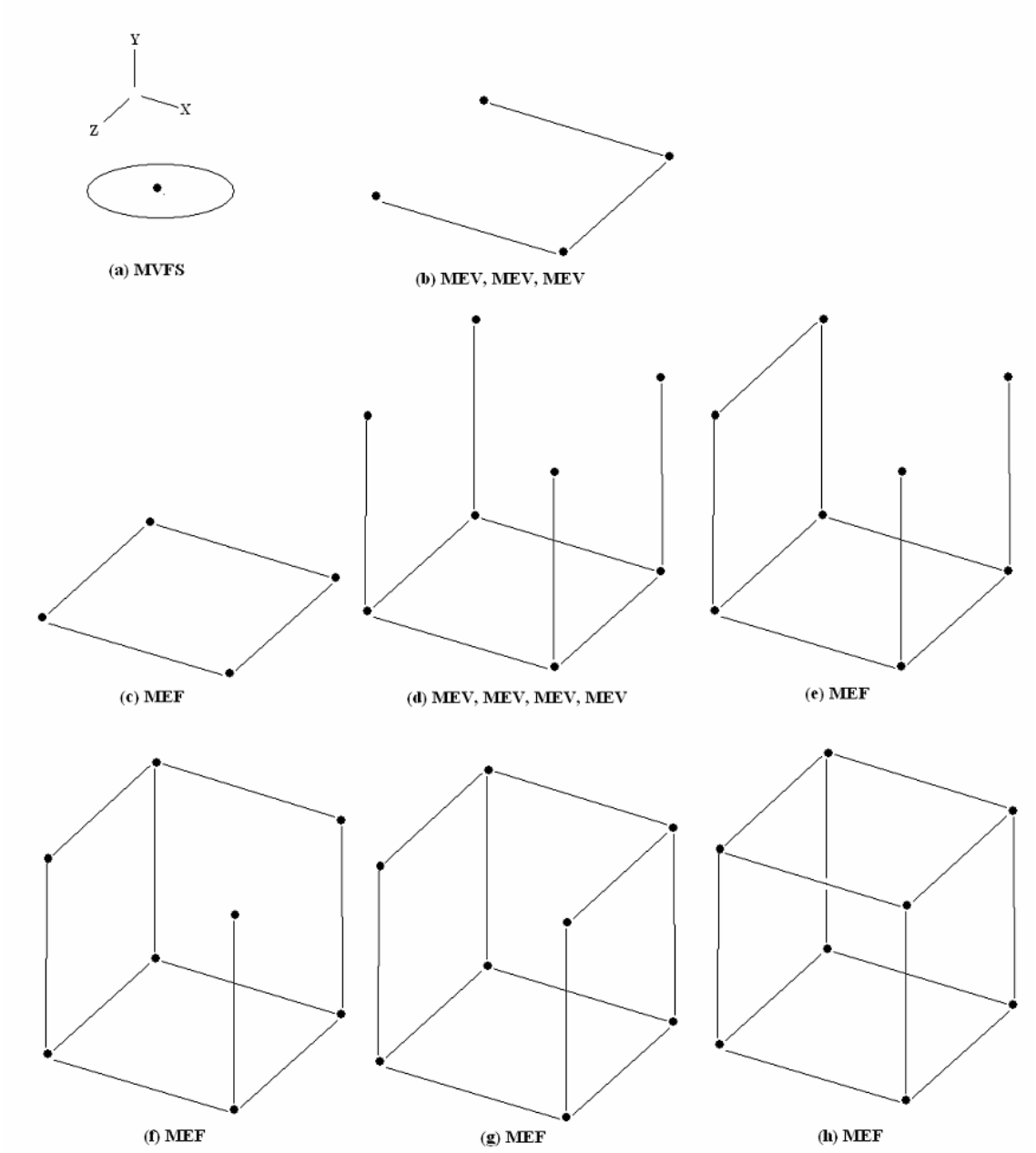


Figura 8.6: Construcción paso a paso de un cubo mediante los operadores de Euler. (Según Mäntylä[1])

CAPÍTULO IX

IMPLEMENTACIÓN DE ALGORÍTMOS DE MEDIANO NIVEL

A continuación se describen distintos algoritmos de mediano nivel implementados para la creación de semiarcos, bases circulares, bases poligonales, los cuales a través de procedimientos de barrido traslacional y barrido rotacional (algoritmos de alto nivel descritos en el Capítulo X) nos permitirán generar modelos sólidos de cilindros, esferas, figuras prismáticas, sólidos de revolución, conos y pirámides.

Generador de arco:

Esta función permite realizar una aproximación poligonal de un círculo ubicado en un plano paralelo al plano XY. En la figura 9.1 se observa la construcción paso a paso de un polígono de seis lados.

Mediante un operador MVFS se crea el vértice v_1 inicial. Teniendo el valor del radio del círculo y las coordenadas de su centro, se obtienen las coordenadas de los siguientes vértices aplicando conceptos básicos de trigonometría.

Por ejemplo las coordenadas del vértice 2 se obtienen mediante:

```
x = centrox  + cos( alfa ) * radio;
y = centroy + sin( alfa ) * radio;
```

Luego se incrementa a través de un ciclo el ángulo alfa y se calculan las coordenadas del vértice v_3, v_4, .. v_n.

Finalmente con el operador MEF se unen el último vértice (v_6 en este caso) al primero (v_1) mediante una nueva arista, quedando así definido la base poligonal.

Generalizando el algoritmo para n lados, se obtienen bases poligonales que aproximan mejor al círculo.

En el anexo B se presenta la función arco que genera pues la base circular; las observaciones realizadas van enmarcadas dentro de los símbolos /*...*/ .

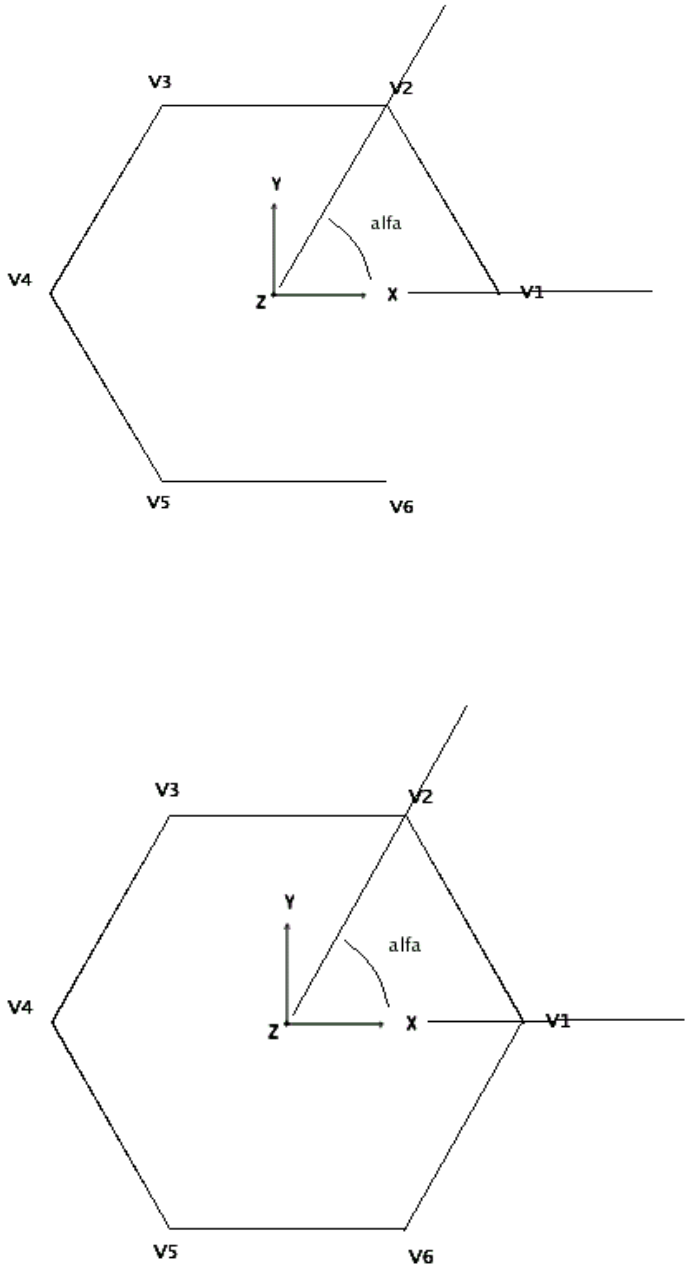


Figura 9.1: Construcción de una base poligonal para aproximar a un círculo

Otra utilización de la función arco tiene que ver con la generación de un semiarco ubicado en el plano XY (el cual no se cerrará para formar una cara), y que posteriormente se girará alrededor del eje x para generar sólidos por revolución.

En la fig.9.2 se observa un semiarco cuyo radio de curvatura se centra en las coordenadas espaciales (0, 0, 0). Se define un ángulo fi_2= 5 grados que forma el vector v1 con el eje -X, y un ángulo igualmente fi_2=5 grados que formará el último vértice del semiarco con el eje +X. Queda entonces definido un ángulo f1= 175 grados que se forma entre el eje +X y el vector v1.

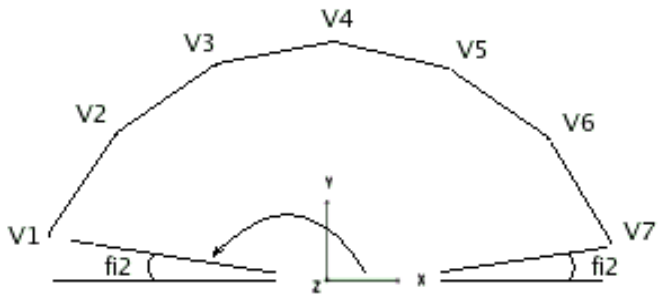


Figura 9.2: Creación de un semiarco ubicado en el plano XY

Generador de base cuadrada:

Esta función permite generar una base cuadrada ubicada en un plano paralelo al plano XY. Teniendo como datos de entrada el valor del lado del cuadrado y las coordenadas de su vector centro (cent), se genera como primer paso un vértice v_1 (ver figura 9.3):

x= cent[X]-lado/2

y= cent[Y]-lado/2

z= cent[Z]-lado/2

Nota: es decir que el cuadrado se construirá en el plano ubicado a una altura z= cent[Z]-lado/2

Posteriormente, a partir del vértice v_1, y aplicando una suma de vectores obtenemos las coordenadas del vértice v_2.

El proceso se repite a partir de v_2 para obtener v_3 y hasta construir el vértice v_4.

(Ver apéndice B para los detalles del algoritmo)

Es importante recalcar que se generalizó la función de tal forma que el centro del cuadrado pueda elegirse donde se desee, y que el lado del mismo sea el que el usuario defina.

Finalmente con el operador MEF se unen el último vértice (v_4 en este caso) al primero (v_1) mediante una nueva arista, quedando así definido el cuadrado.

En el anexo B se presenta la función cuadrado cuyo algorítmo se ha incorporado dentro de la primitiva que genera un cubo; las observaciones realizadas van enmarcadas dentro de los símbolos /*...*/ .

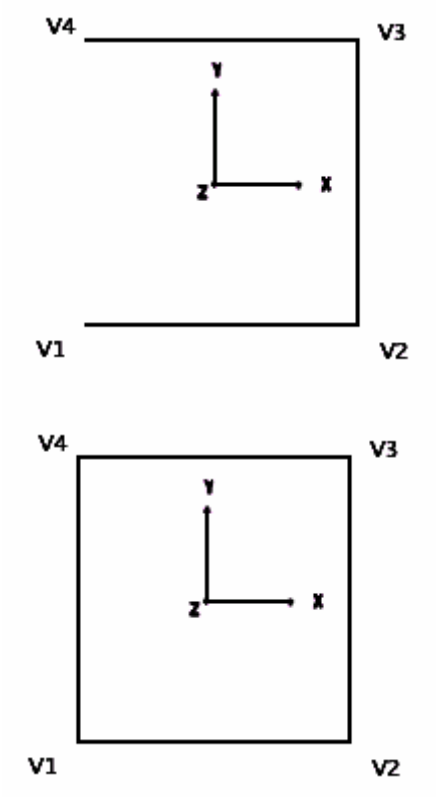


Figura 9.3: Generación de base cuadrada en un plano paralelo al plano XY

Generador de base rectangular:

Es el mismo principio utilizado para generar la base cuadrada en un plano paralelo al plano XY; En este caso, con los valores de ancho y largo de un rectángulo, se aplica el procedimiento empleado para el cuadrado para crear los 4 vértices y finalmente se cierra la cara con el operador MEF.

La figura 9.4 muestra los pasos para crear la base rectangular.

En el anexo B se presenta la función rectángulo cuyo algoritmo se ha incorporado dentro de la primitiva que genera un bloque.

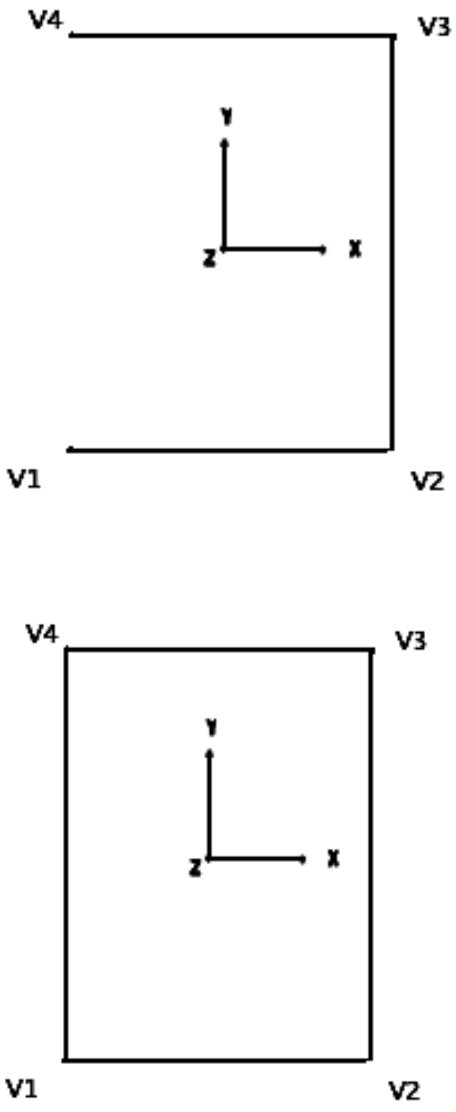


Figura 9.4: Creación de una base rectangular en el plano XY

Generador de base plana irregular:

La implementación de esta función permite que se cree una base plana irregular a partir de los vértices que el usuario introduzca en el Programa SMS_EIM_UCV.

Siendo Xp el arreglo de vectores que definen cada vértice introducido, la función permite unir los vértices y finalmente cerrar la base poligonal, creando entonces una cara plana (ver figura 9.5). Dicha base se ubica en el plano XY (cota z= 0).

Los líneas de comando de esta función se incorporaron a la función extrusión (ver apéndice B) la cual a su vez generará la extrusión de la base irregular, y también a la función revolución que genera a partir de una curva cerrada un sólido de revolución hueco en su interior.

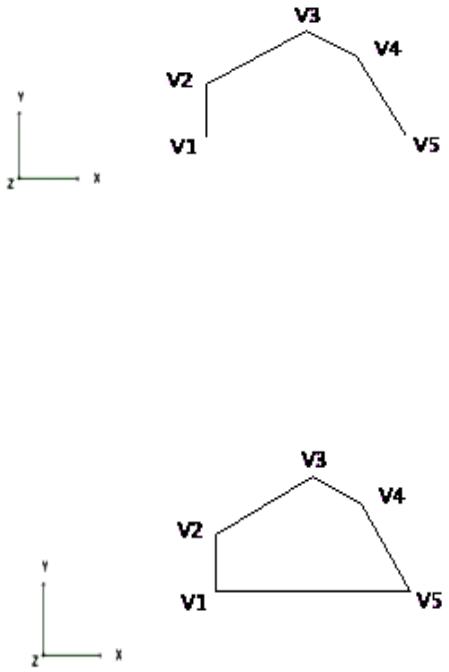


Figura 9.5: Creación de base poligonal

Generador de polígono regular:

Para generar polígonos regulares se utiliza el misma función que se detalló anteriormente para crear una base circular. En este caso el número de lados del polígono lo define el usuario.

En la figura 9.6 se muestra un pentágono regular creado con esta función.

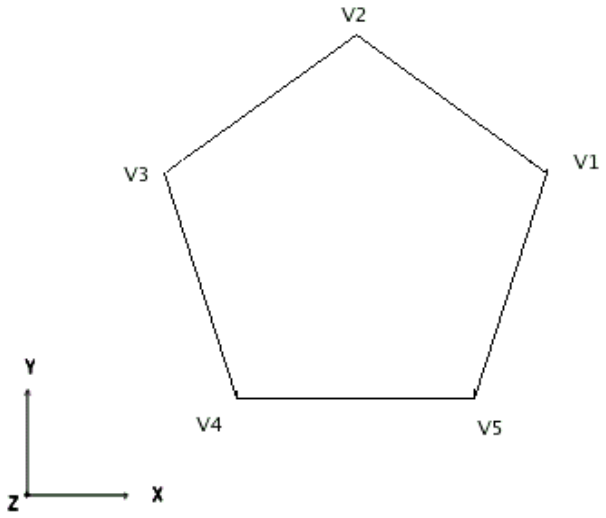


Figura 9.6: Creación de un polígono regular

Generador de caras triangulares:

La implementación de esta función nos permitirá más adelante crear algoritmos de alto nivel para la creación de las caras triangulares laterales que constituyen las pirámides o los conos.

Esta función une el vértice principal de un triangulo isósceles a los vértices de su base, definiendo así la cara triangular.

En el ejemplo de la fig. 9.7 se observa cómo se une el vértice v al vértice v_1 mediante MEV, y luego se termina de definir la cara triangular mediante el operador MEF, en donde se unen los vértices v y v_2 . Dicha función se incluye directamente en el algoritmo que permite crear pirámides (anexo B).

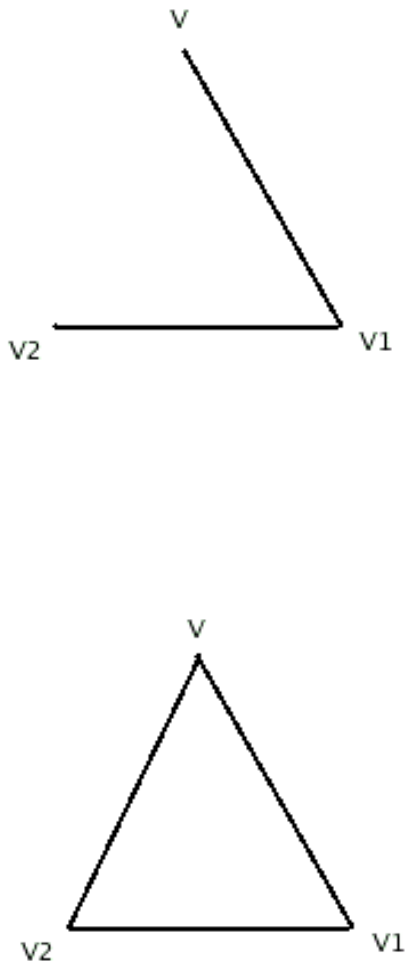


Figura 9.7: Creación de base triangular

CAPÍTULO X

IMPLEMENTACIÓN DE PRIMITIVAS

En el presente capítulo se implementarán las primitivas fundamentales con las cuales se podrán generar sólidos por extrusión a partir de un polígono plano, sólidos (macizos o huecos) obtenidos por revolución, sólidos obtenidos mediante la creación de caras triangulares a partir de un vértice común (pirámides, conos).

Operaciones de barrido:

Entiéndase por barrido de un polígono plano, aquella operación en donde el polígono es trasladado a distintas posiciones sucesivas del espacio tridimensional, siguiendo una trayectoria predefinida. Si creamos una secuencia continua de ese barrido (desplazamiento muy pequeños del polígono) se creará la capa externa de un sólido desde donde se inició el movimiento del polígono hasta donde culminó.

Este es el principio básico del barrido.

A continuación se implementarán técnicas para describir convenientemente ciertos tipos de sólido, las cuales se apoyan en las funciones de mediano nivel descritas en el Capítulo IX, y en los operadores de Euler.

Barrido de traslación:

Esta operación toma una cara f de un sólido, y le realiza un barrido a través de un vector [dx dy dz]. Recordemos que una figura cerrada (cara) es asimilada a una lámina que tiene dos caras como los dos lados de una hoja de papel. Es decir que el barrido de una figura cerrada consistirá en barrer una de las dos caras de la lámina.

La implementación de la función de barrido de traslacional se apoya en los operadores de Euler de bajo nivel.

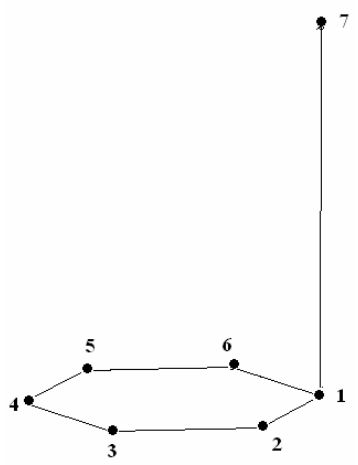
Esta función está incluida en el anexo B; el algoritmo va pasando por cada vértice de la cara base mediante un recorrido a través de las semiaristas del lazo de la cara, y le

agrega una arista desde cada vértice en la dirección del vector de barrido mediante un MEV. Los vértices de los extremos de las nuevas aristas creadas se unen mediante el operador MEF, para crear los lados de las caras de la región barrida.

En la figura 10.1 se observa los pasos que van creando un prisma de base hexagonal; primero se obtiene el vértice 7 y la arista que lo une al vértice 1 mediante el operador MEV (a). Luego se crea el vértice 8 y la arista que lo una al vértice 2 mediante MEV, y finalmente se cierra la cara formada por los vértice 1,2,8,7 con el operador MEF (b).

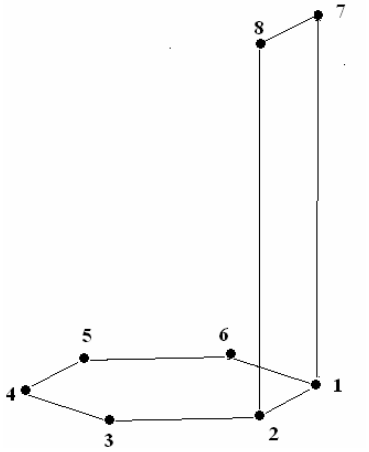
Luego se crea el vértice 9 y la arista que lo une al vértice 3 mediante MEV, y se cierra la cara formada por los vértices 2,3,9,8 mediante MEF (c). Este proceso se repite en (d) y (e) hasta crear la penúltima cara lateral del sólido (f). Finalmente se cierra la última cara lateral del sólido mediante MEF (g).

Esta función se implementó para realizar sólidos por extrusión para el programa SMS_EIM_UCV, como por ejemplo: cilindros, cubos, bloques, extrusión de bases irregulares, prismas rectos.



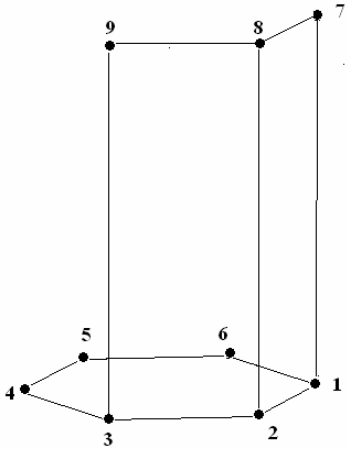
MEV

(a)



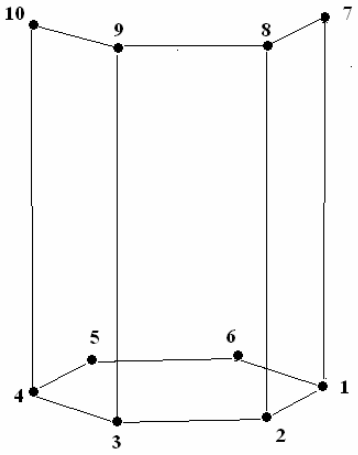
MEV y MEF

(b)



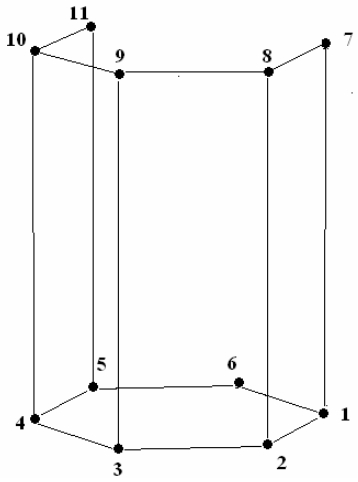
MEV y MEF

(c)



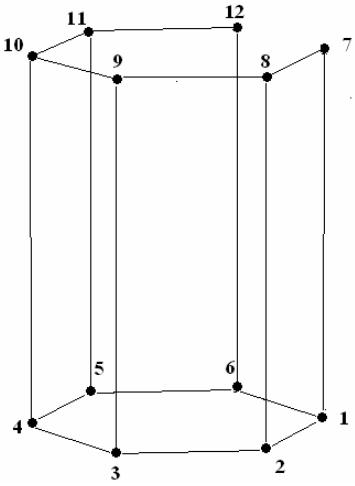
MEV y MEF

(d)



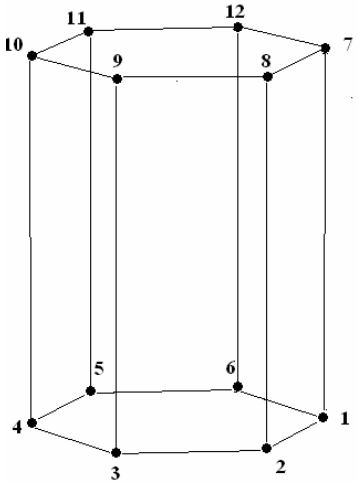
MEV y MEF

(e)



MEV y MEF

(f)



MEF (g) Figura 10.1: Aplicación de los operadores de Euler para crear un sólido mediante barrido traslacional

Barrido rotacional

El barrido rotacional permite barrer una figura plana abierta en sus extremos (polígono abierto) haciéndola girar alrededor del eje X para crear sólidos de revolución macizos. Evidentemente, el sólido resultante será un poliedro que aproximará al sólido original.

La fineza de la aproximación vendrá dada por dos parámetros:

- Una mejor aproximación de la curva plana que se hará girar.
- Una subdivisión del espacio rotacional (360 grados) los más pequeña posible.

Este algoritmo toma un conjunto de arista unidas entre sí, las cuales deben estar ubicada en el plano XY, y cuya coordenada *y* de cualquier punto de esas aristas debe ser mayor a 0; es decir que ningún vértice puede tocar el eje X ya que se generarían un conjunto de vértices repetidos ubicados en el eje X.

Supongamos que el polígono abierto (fig.10.2) está constituida por 6 aristas (y por consecuente 7 vértices). Luego, a partir del vértice v_7 (último vértice de la figura plana abierta), se genera su vértice homólogo $v_{7'}$ producto de la rotación del punto v_7 alrededor del eje x. Ese ángulo de rotación vendrá definido por el número de subdivisiones espaciales que se realicen (ángulo_rotación= 360 grados/ $n_{subdivisiones}$).

Se unen los vértices $v_7v_{7'}$ mediante el operador MEV (fig 10.3a).

Luego, con el vértice v_6 se generá su homologo $v_{6'}$ por rotación , y se unen los vértices $v_6v_{6'}$ mediante el operador MEV (b) para finalmente crear un cara con los vértices v_6,v_7,$v_{7'}$,$v_{6'}$ mediante MEF (c) .

Posteriormente con el vértice 5 se genera por rotación el vértice $v_{5'}$, y se unen los vértices $v_5v_{5'}$ mediante el operador MEV y se crea la cara v_5,v_6,$v_{6'}$,$v_{5'}$ (d). Así se repite el proceso para crear las caras restantes (e).

Posteriormente se repiten exactamente los mismos procesos para crear arista nuevas y caras nuevas pero a partir de los vértices $v_{7'}$,$v_{6'}$, ..$v_{1'}$ (f).

De esta forma se repite *nsudivisiones espaciales* veces este proceso hasta generar el sólido de revolución, en este caso una esfera (g). Es de notar que si aumentamos el número de subdivisiones del semiarco, y el número de subdivisiones espaciales, se obtiene una mejor aproximación de la esfera.

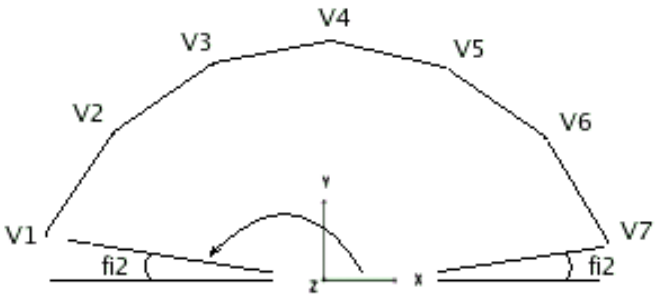
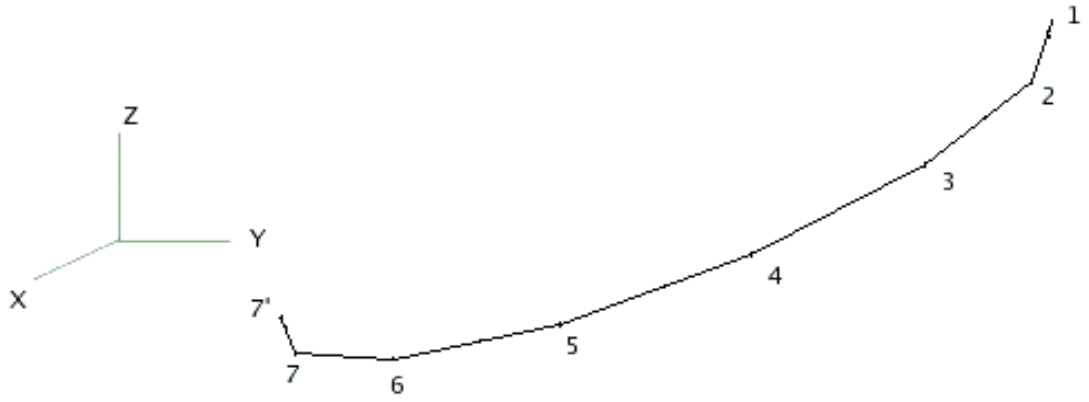
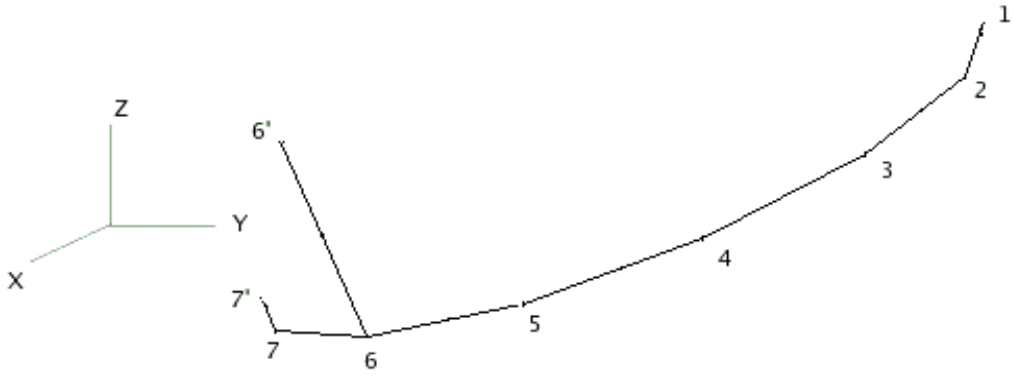


Figura 10.2: Polígono abierto ubicado en el plano XY

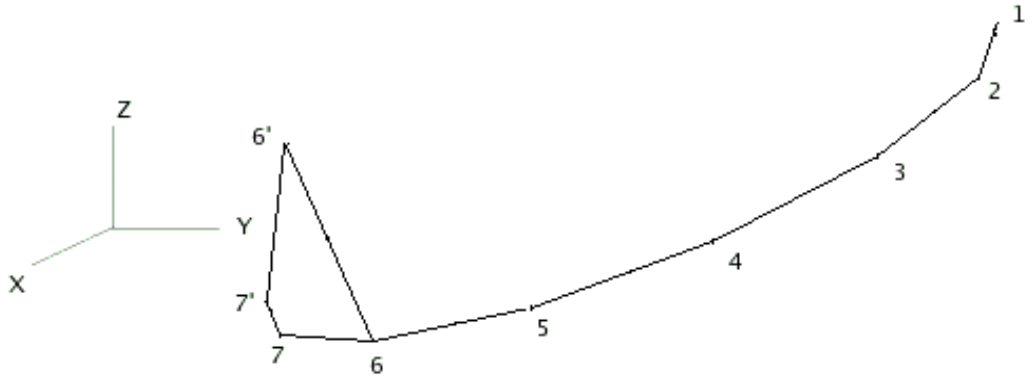


(a)

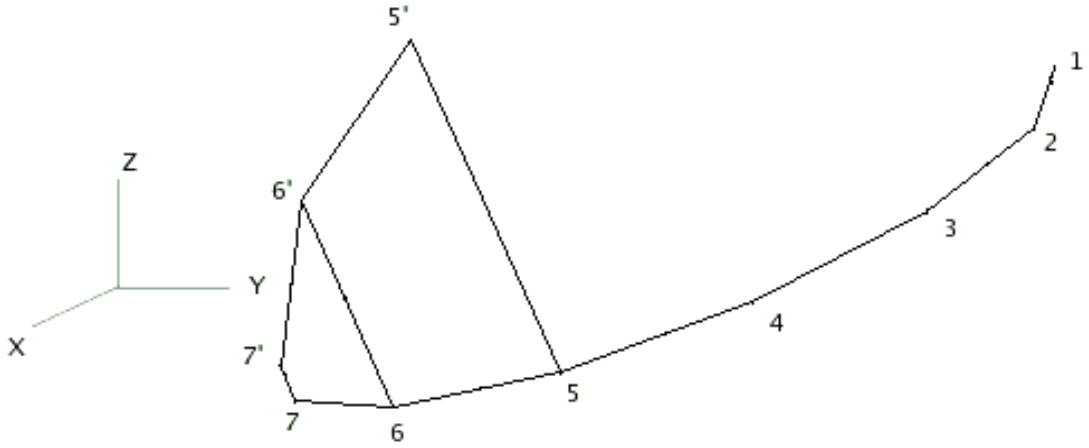


(b)

Figura 10.3: Pasos para la creación de un sólido por barrido rotacional

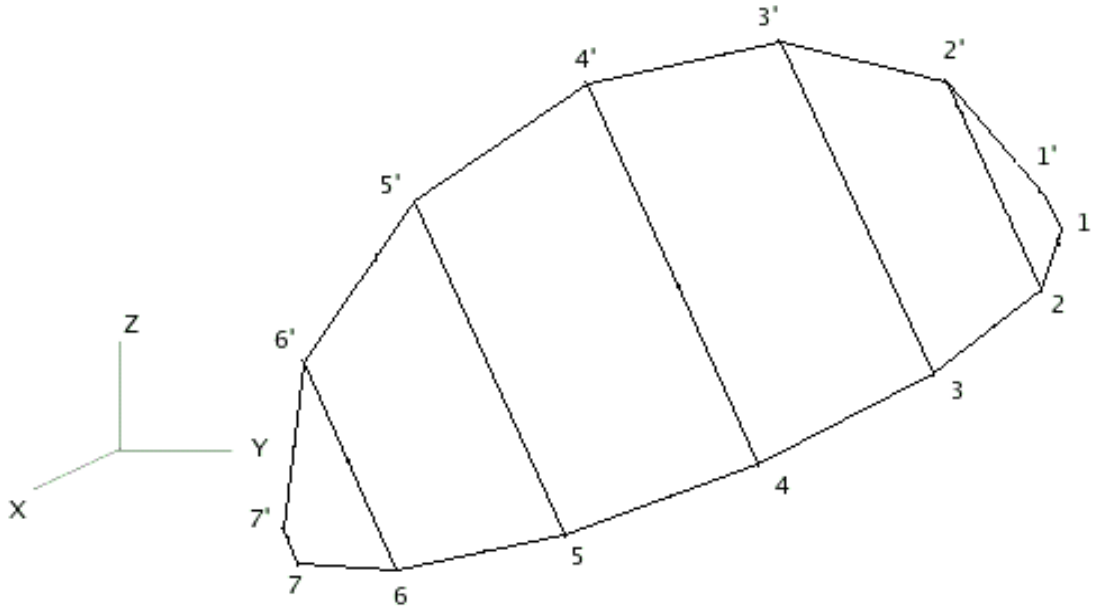


(c)

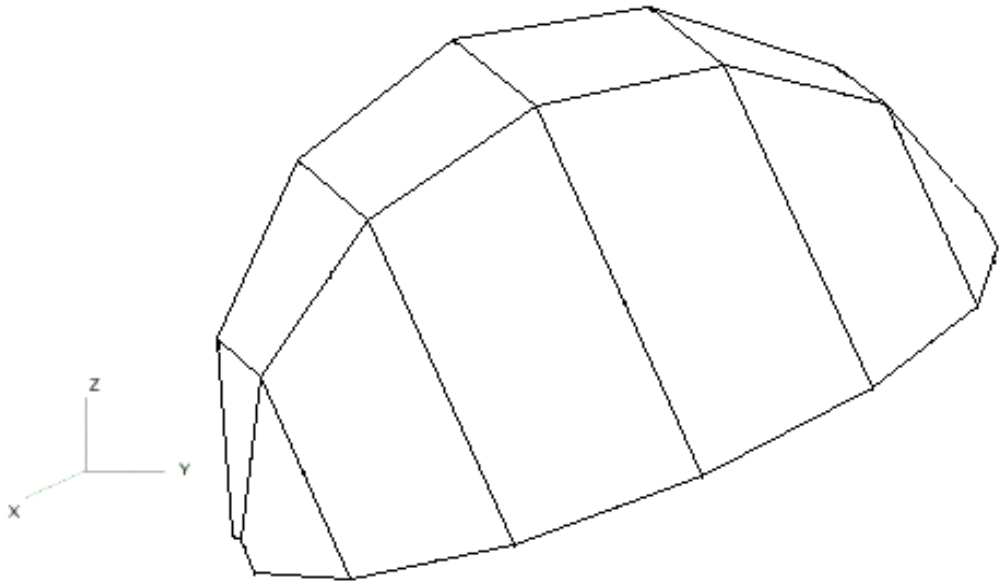


(d)

Figura 10.3: Pasos para la creación de un sólido por barrido rotacional (Continuación)



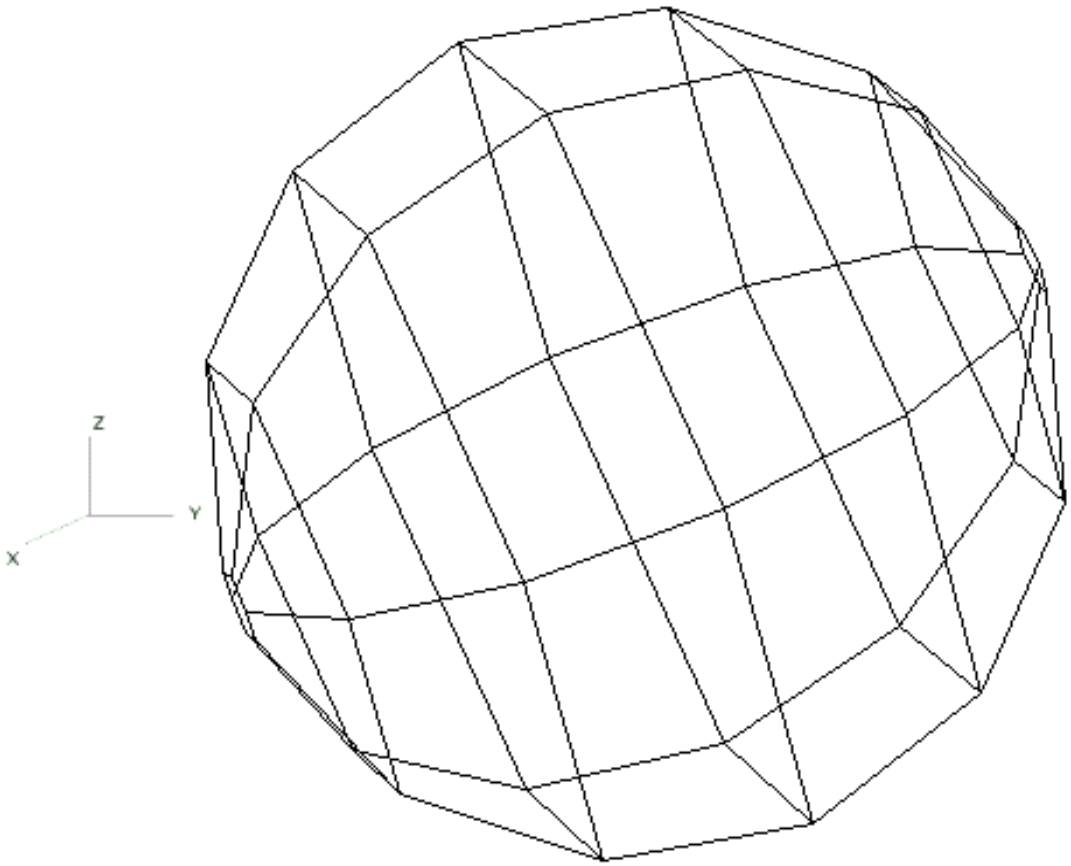
(e)



(f)

Figura 10.3: Pasos para la creación de un sólido por barrido rotacional

(Continuación)



(g)

Figura 10.3: Pasos para la creación de un sólido por barrido rotacional (Continuación)

Barrido rotacional de polígonos:

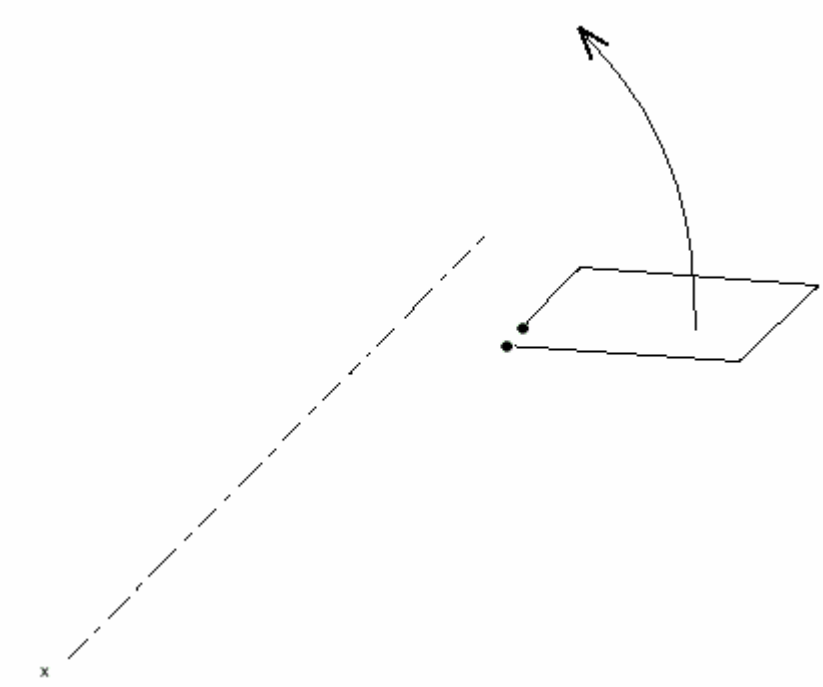
Esta función permitirá generar, a partir de un polígono ubicado en el plano XY y el cual no intersecta el eje X, un sólido de revolución hueco.

Este algoritmo es similar al barrido rotacional expuesto anteriormente; sin embargo presenta algunas modificaciones.

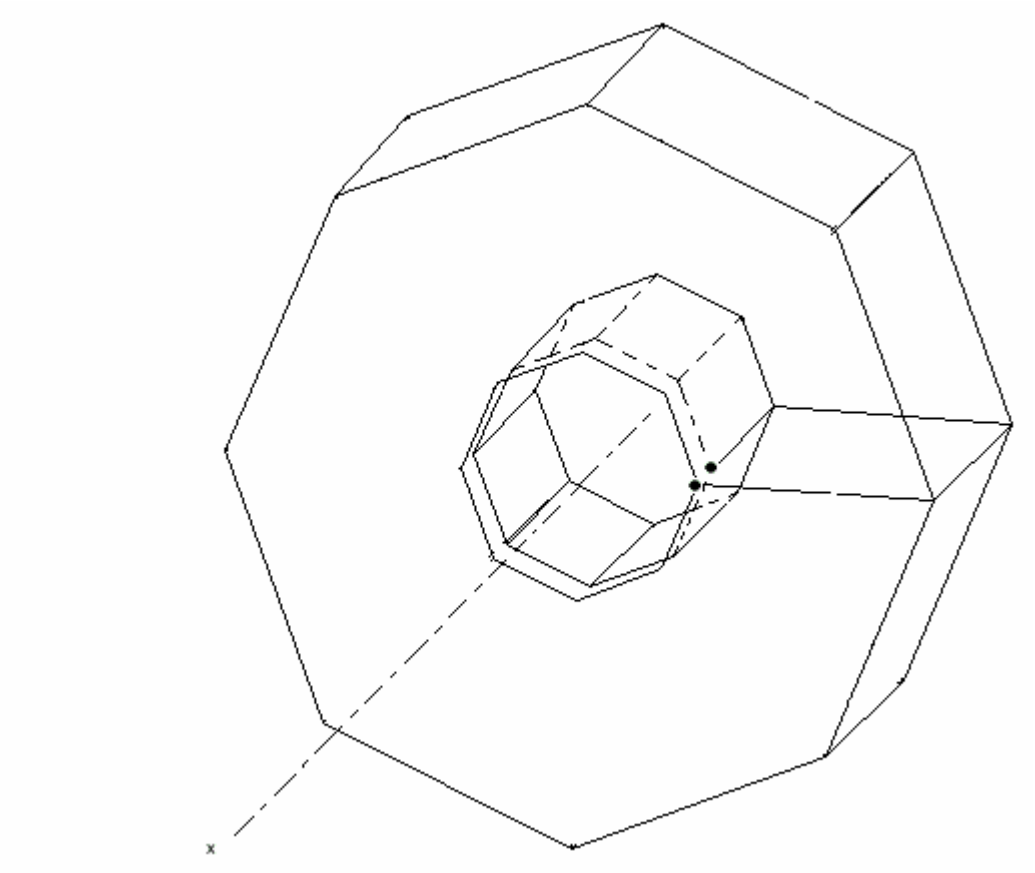
A partir del polígono ubicada en el plano XY (fig.10.4a), se recurre al artificio de abrirlo en un vértice arbitrario mediante la duplicación del mismo con un MEV, y removiendo la arista nula que conecta los dos vértices resultantes mediante KEF. Esto produce un polígono abierto (aunque pareciera que el mismo fuese cerrado ya que tiene un par de vértices superpuestos), el cual podrá ser barrido mediante la función barrido rotacional expuesta con anterioridad.

Sin embargo existe un problema que debe solucionarse: las dos copias de los vértices superpuestos causan o generan dos caras circulares (con sus respectivos lazos internos), las cuales también estarán superpuestas (b). Es por estos que se deben fusionar ambas caras circulares mediante una función que "pegue " sus respectivos lazos internos (c) .

Esta función adicional que denominaremos *pegar_lazos* se incluye igualmente en el apéndice B.

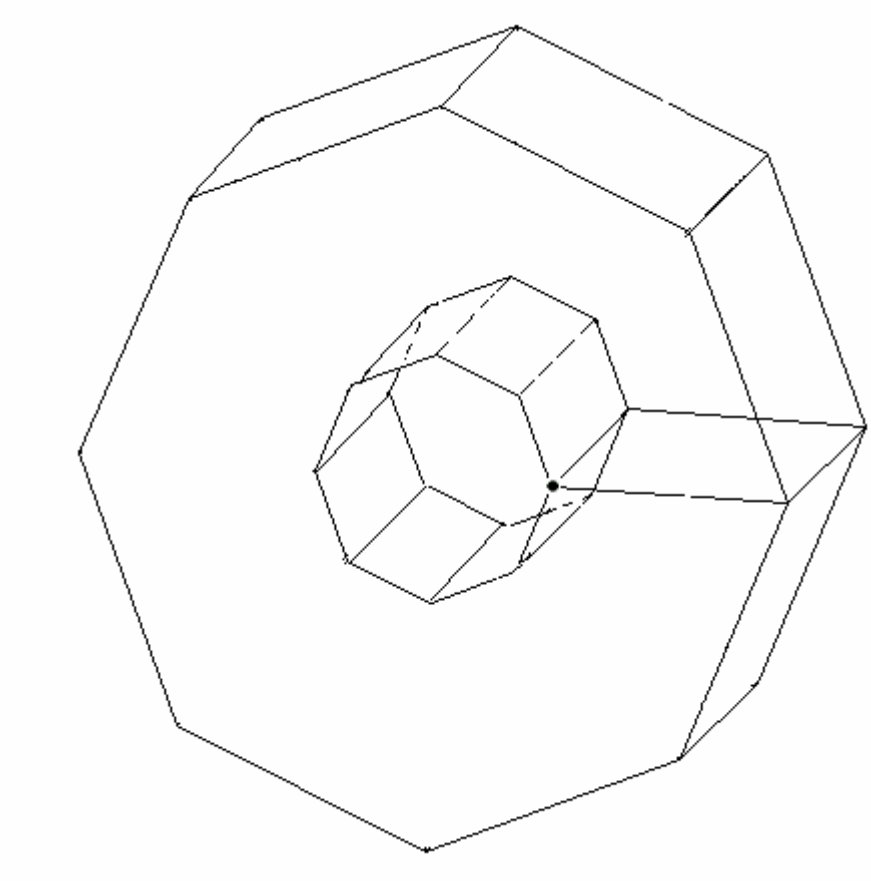


(a)



(b)

Figura 10.4: Barrido rotacional de un polígono



(c)

Figura 10.4: Barrido rotacional de un polígono
(Continuación)

Desarrollo de algoritmos para la creación de Pirámides y Conos:

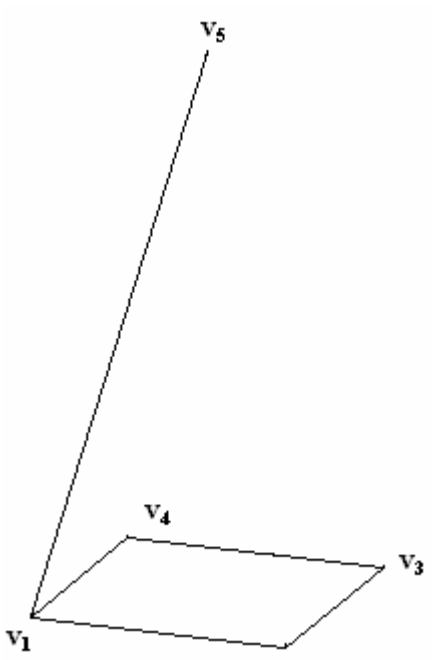
En el capítulo IX se detalló la función *caras_triangulares*, la cual permite unir la cima de un triángulo isósceles a los vértices de su base. Dicha función forma la base para la creación de pirámides y conos.

La figura 10.5 muestra la creación paso a paso de una pirámide. Una vez que se tiene constituida la base cuadrada $v_1v_2v_3v_4$ de la pirámide (mediante la función base cuadrada, Cap. IX), y teniendo las coordenadas del vértice v_5, se unen v_1 y v_5 mediante el operador MEV, tal y como se aprecia en la figura 10.5a.

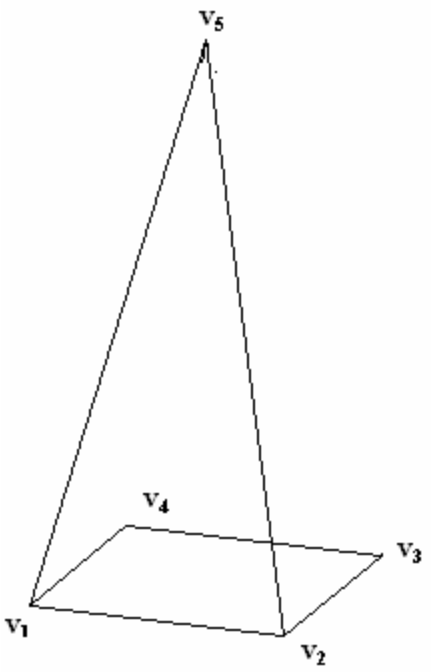
Posteriormente, se unen los vértices v_5 y v_2 mediante un operador MEF, con lo cual queda definida la primera cara lateral de la pirámide en construcción (fig.10.5b).

Luego, mediante operaciones sucesivas para crear caras, se generan las demás caras triangulares de la pirámide, con lo cual queda definido el sólido pirámide (fig. 10.5 c y d).

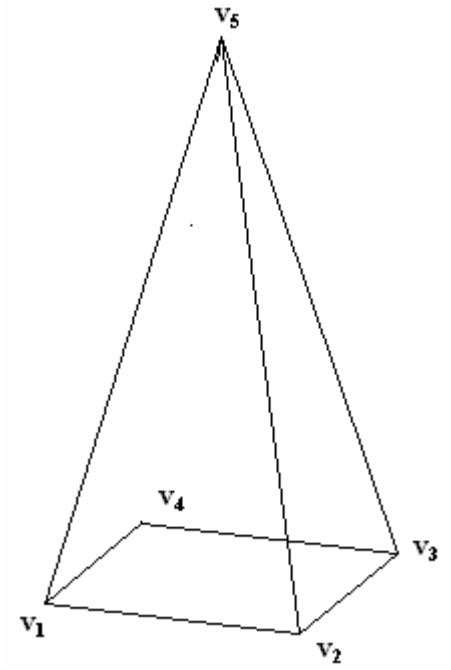
El procedimiento para crear conos, es análogo para el de las pirámides, ya que se apoya igualmente en la función caras_triangulares. La diferencia radica en que para un cono se obtendrán mayor número de caras laterales ya que la base del cono se constituye por muchas más aristas, logrando aproximarse a una base circular.



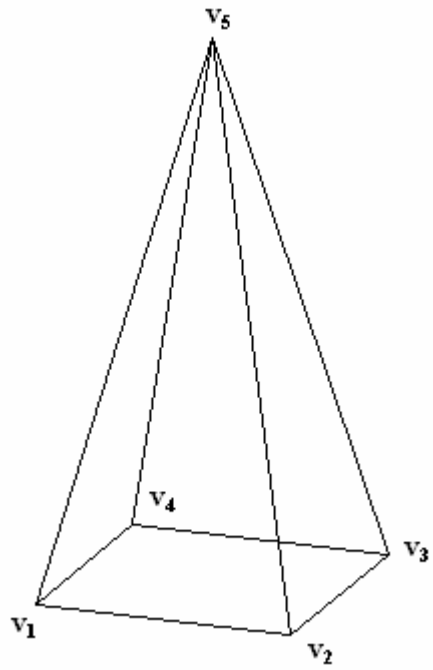
(a)



(b)



(c)



(d)

Figura 10.5 – Creación paso a paso de una pirámide

CAPÍTULO XI

PROGRAMA SMS_EIM_UCV

En este capítulo se presenta el Programa SMS_EIM_UCV, se explican los pasos que debe realizar el usuario para ejecutarlo, y se detallan las funciones implementadas en el programa.

Como ejecutar el programa:

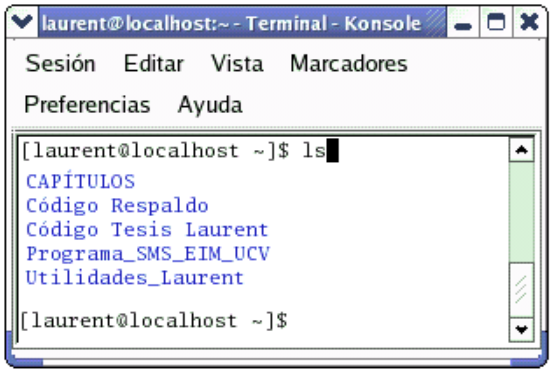
1) Copiar la carpeta "Programa SMS_EIM_UCV" ubicada en el CD del programa.

2) Pegar la carpeta "Programa SMS_EIM_UCV" en el administrador de Archivos Personales cuyo símbolo es:

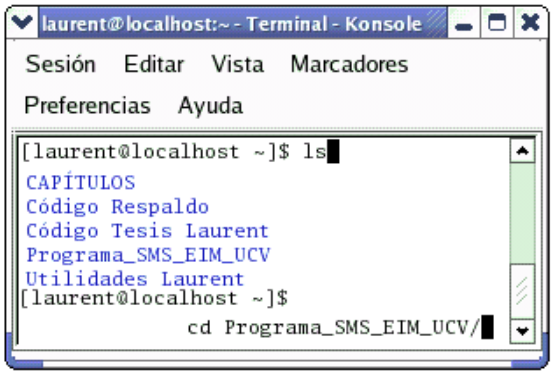
2) Abrir el Terminal de Linux cuyo símbolo es:

3) Ejecutar desde el Terminal el comando: ls [ENTER]

(Nota: Aparecerá una lista de archivos presentes en la carpeta de Archivos Personales)

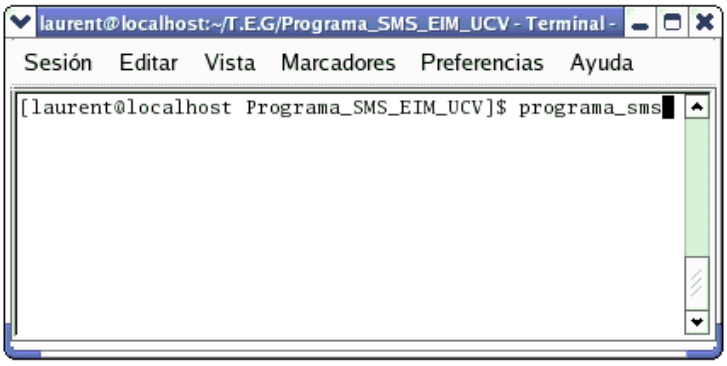


4) Ejecutar el comando: cd Programa_SMS_EIM_UCV/ [ENTER]



(Nota: Con este comando se ingresa a la carpeta del programa.)

5) Ejecutar el comando: programa_sms [ENTER]



(Nota: Al ingresar este comando se ejecuta el programa en una ventana nueva.)

Explicación del programa:

Una vez ejecutado el programa, aparece la ventana principal del mismo (fig.11.1); en ella se ubica el nombre del programa, botones para crear sólidos y manipularlos, dos botones de ayuda / salir, y una ventana de representación gráfica.

Antes de iniciar con la creación de un sólido, se invita al usuario a pulsar el botón de ayuda; al realizar esta acción, aparecerá una ventana nueva con la explicación de las funciones de cada botón (fig.11.2). El contenido completo de dicha ventana está incluido en el anexo C.

Creación de sólidos:

Solo se debe pulsar algún botón para crear algún sólido específico; por ejemplo, al pulsar el botón Cono, aparecerá una ventana de texto nueva, en la cual se ofrece al usuario una serie de opciones (fig11.3):

(1) Generar cono nuevo.

(2) Representar cono creado anteriormente.

(3) Ejemplo de cono predefinido por el SMS.

Al pulsar la opción (3), aparece la representación gráfica de un cono de ejemplo en la ventana principal (fig.11.4) . Usted puede entonces manipular el sólido con los botones destinados para tal fin. La explicación de cómo manipular un sólido creado se dará en este mismo capítulo más adelante.

Si se pulsa la opción (1), usted podrá generar un cono nuevo. Se abre entonces una ventana de texto interactiva en donde el programa le solicitará una serie de datos necesarios para tal fin (fig.11.5) :

Introduzca las dimensiones del cono una a una y presione ENTER:

-Radio de la base:

-Altura:

-Nombre del sólido que se creará:

Por ejemplo, si introducimos como valor del radio de la base 5, su altura 15, y nombramos al sólido cono_nuevo, el programa le dará la opción de ver los datos que se introdujeron, y de reingresar, de ser requerido por el usuario, nuevamente los datos del

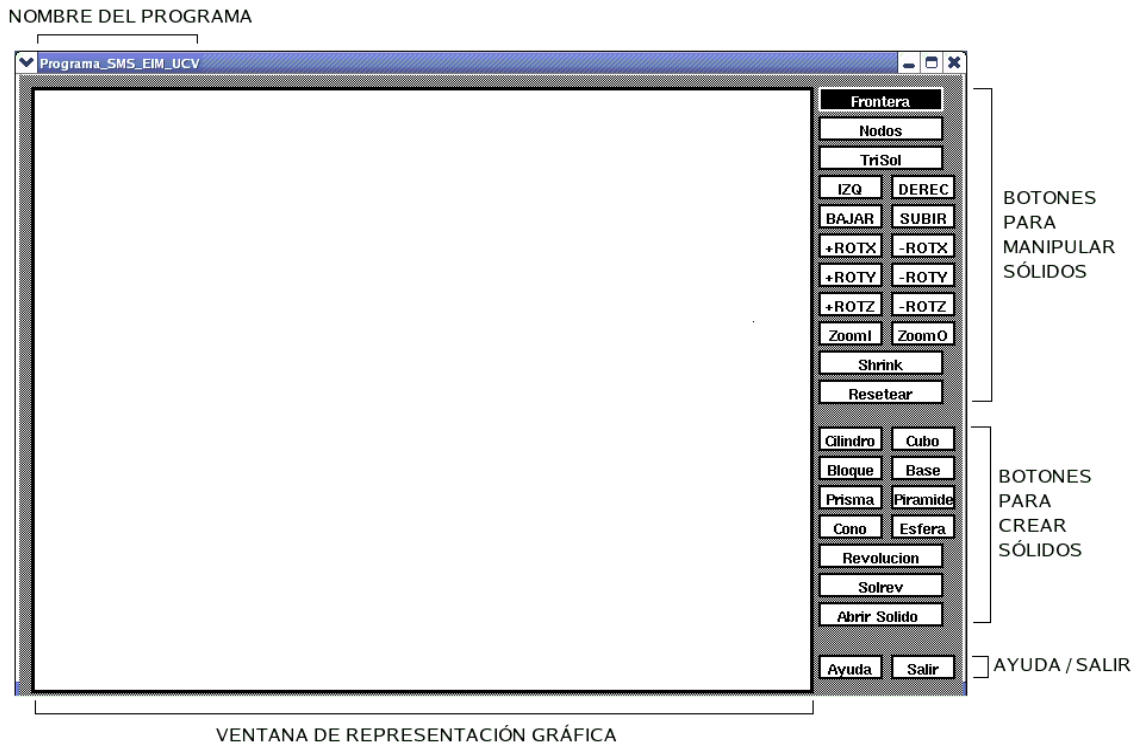


Figura 11.1: Ventana principal del Programa SMS_EIM_UCV

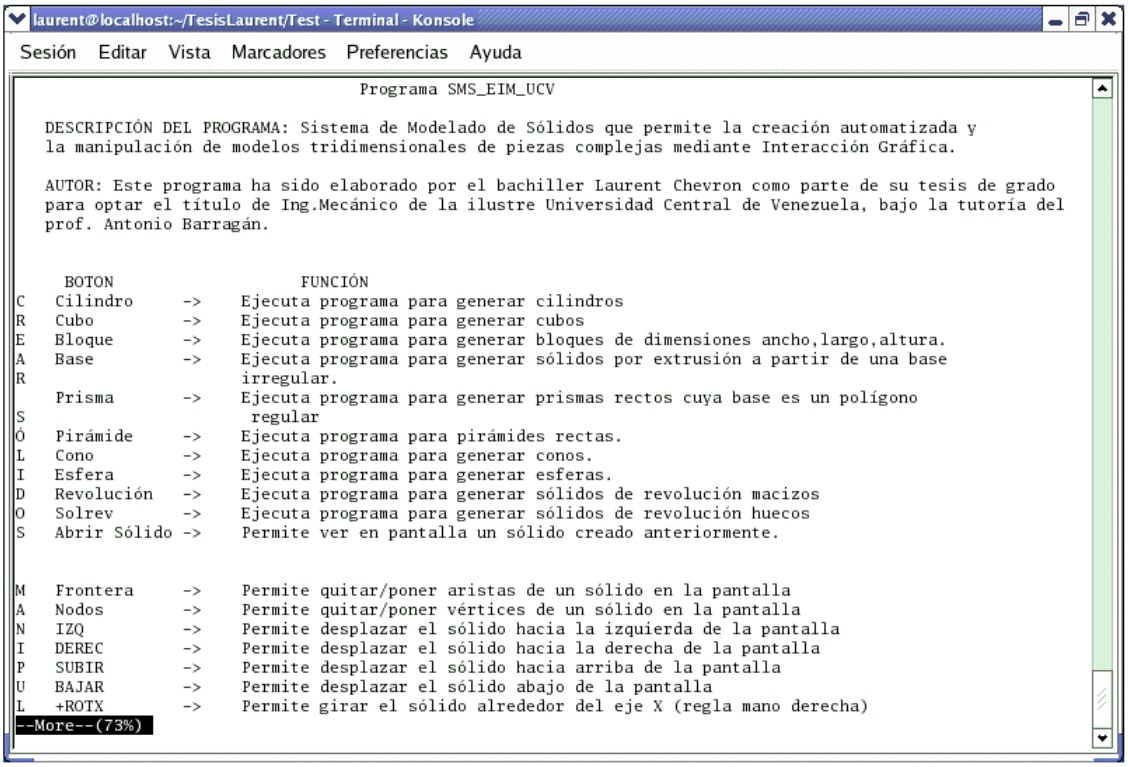


Figura 11.2: Ventana de Ayuda

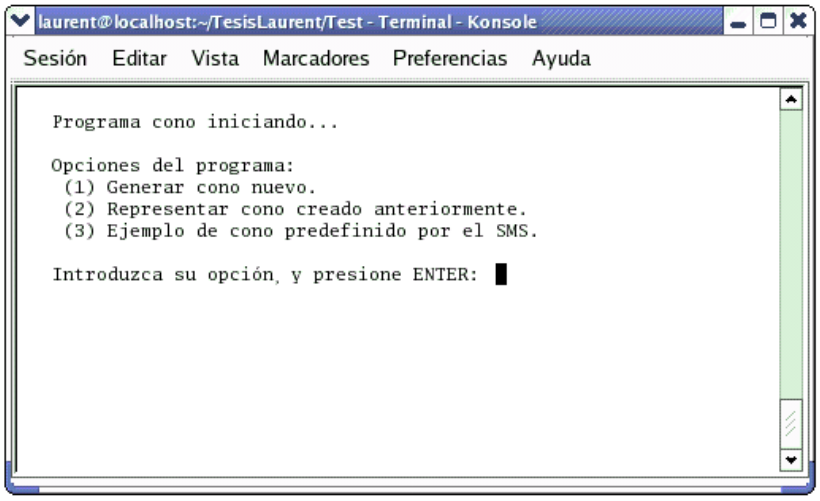


Figura 11.3: Subprograma Cono ejecutándose

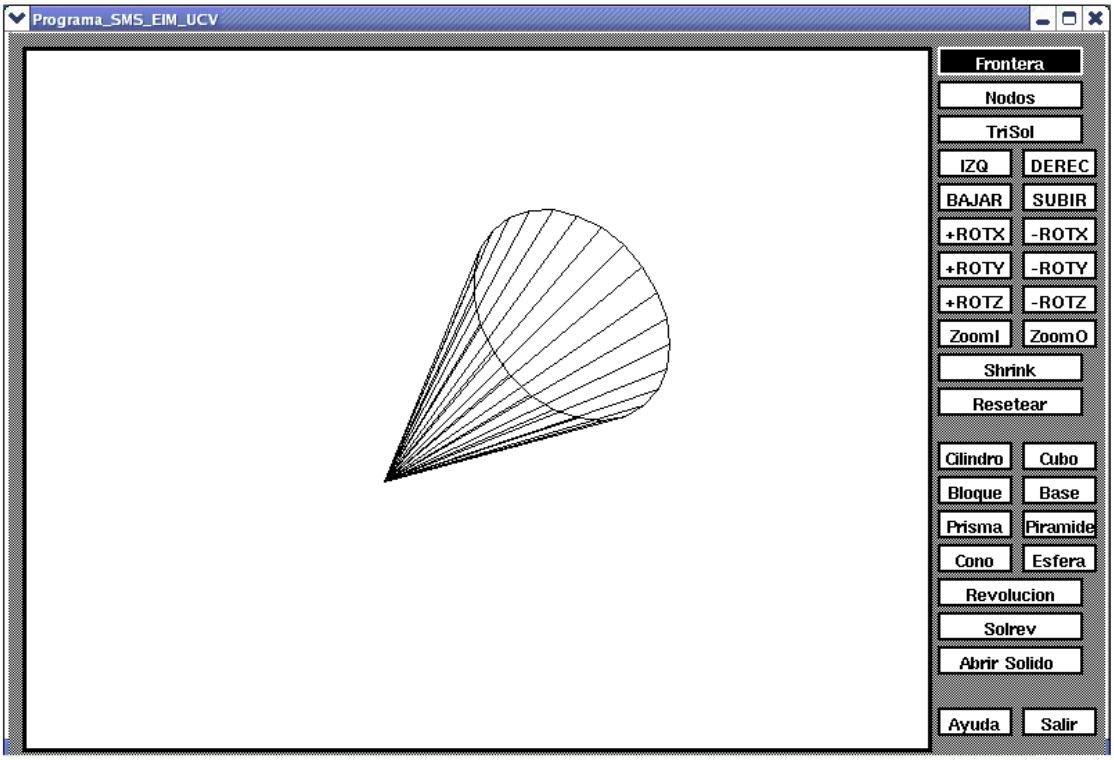


Figura 11.4: Representación gráfica de un cono de ejemplo

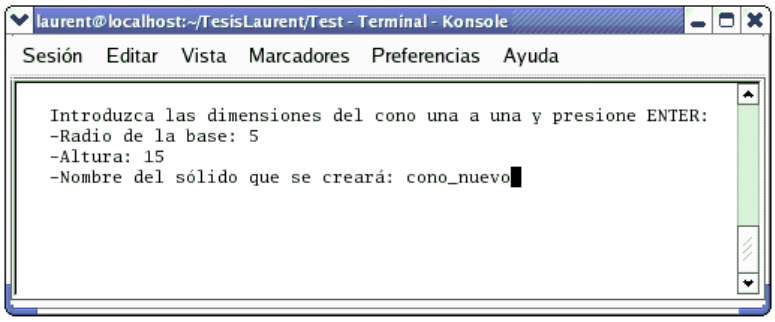


Figura 11.5: Menú para crear un cono nuevo

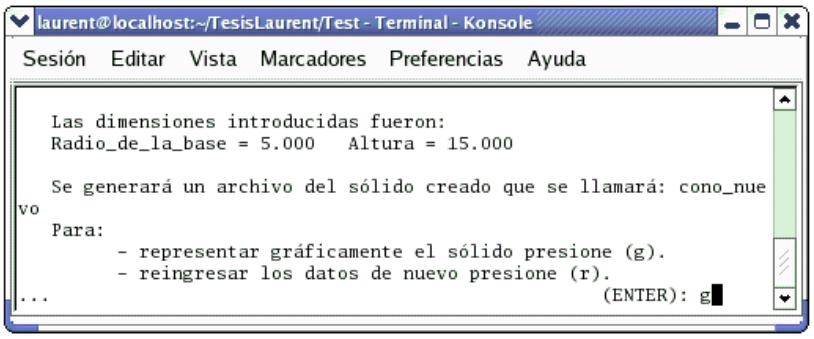


Figura 11.6: Menú de confirmación de entrada de datos

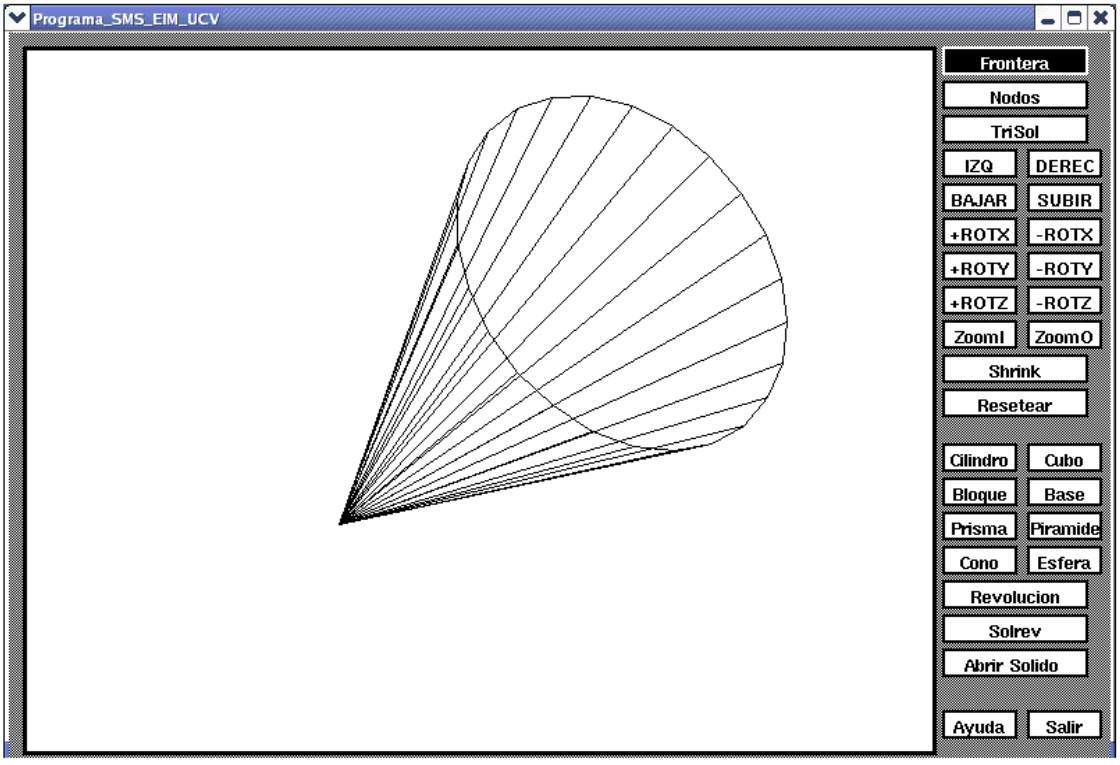


Figura 11.7: Representación gráfica de un nuevo cono

sólido (fig.11.6). Si se introdujeron correctamente los datos, el programa dará la opción de ver la representación gráfica del sólido recién creado (fig.11.7).

Por último, si se pulsa el botón (2), se desplegarán todos los archivos presentes en la carpeta del programa (fig.11.8); cabe destacar que cada vez que se genere un nuevo sólido, se guardarán sus datos en un archivo cuyo nombre será el que se le halla dado al sólido, y cuya ubicación estará dentro de la misma carpeta del programa.

Es decir, si desea abrir el sólido recién creado nuevamente o cualquier otro sólido, solo escriba en pantalla su nombre y presione ENTER, con lo cual se generará la representación gráfica del mismo.

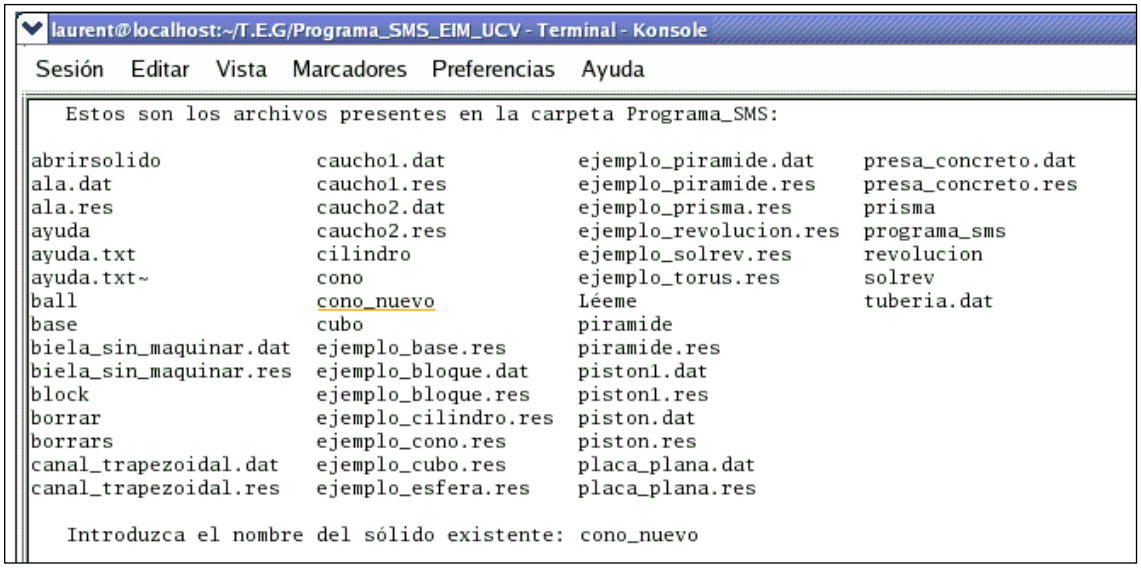


Figura 11.8: Archivos presentes en la carpeta del programa

Los botones para generar cilindros, cubos, pirámides, esferas, prismas regulares, y bloques (Botones: Cilindro, Cubo, Pirámide, Esfera, Prisma, Bloque) tienen un menú similar al presentado para el cono.

El botón que genera sólidos prismáticos a partir de la extrusión de una base irregular (botón: Base) despliega un menú distinto (fig.11.9) con las opciones siguientes:

(1) Ingresar el nombre del archivo de texto desde donde se leerán los datos del sólido.

(2) Ingresar por pantalla directamente los datos de los vértices del polígono, uno por uno.

(3) Representar sólido existente

(4) Ejemplo predefinido por el SMS

Las opciones (3) y (4) ya fueron explicadas para el cono. La opción (1) permite leer un archivo de texto que contenga los datos de una base irregular ubicada en el plano XY y constituida por un número máximo de 40 vértices; el programa abrirá y leerá dicho archivo con solo suministrarle el nombre del mismo.

Dicho archivo de texto debe contener el número de vértices, y luego la identificación de cada vértice, con sus respectivas coordenadas X y Y.

Por ejemplo: 4 (Esto indica que existen 4 vértices)

1 7.4 2.1 (Vértice 1, X=7.4, Y=2.1)

2 5.1 8 (Vértice 2, X=5.1, Y=8)

3 11 11 (Vértice 3, X=11, Y=11)

4 1 3 (Vértice 4, X=1, Y=3)

Se debe subrayar que la explicación que se está suministrando igualmente saldrá en pantalla cuando el usuario ejecute el botón Base.

El algoritmo de la función Base cerrará la base poligonal uniendo automáticamente los vértices 1 y 4 (vértice inicial y final). Posteriormente el programa le pedirá que introduzca la altura del sólido (es la dimensión requerida para aplicarle la operación de extrusión a la base).

En la fig.11.10 se ve una base irregular cuyos vértices (12 en este caso) fueron leídos desde un archivo de texto. La fig.11.11 muestra la misma base, una vez que el programa le aplica la extrusión.

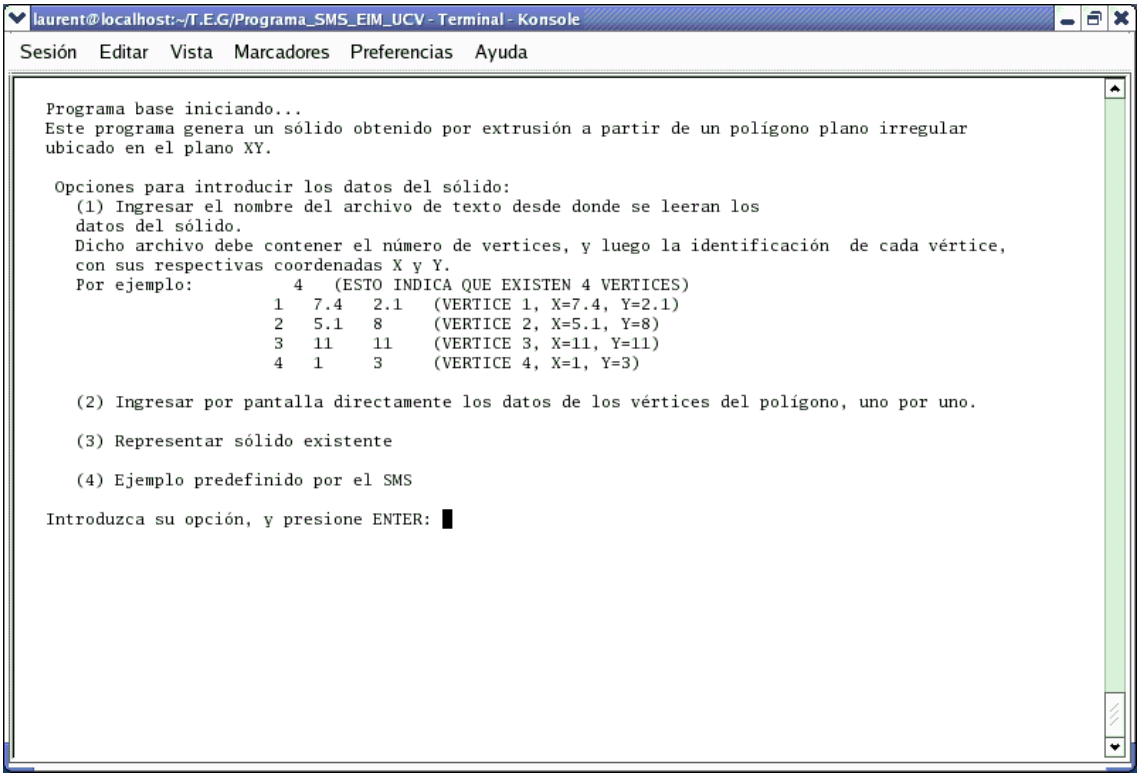


Figura 11.9: Menú para generar sólidos por extrusión

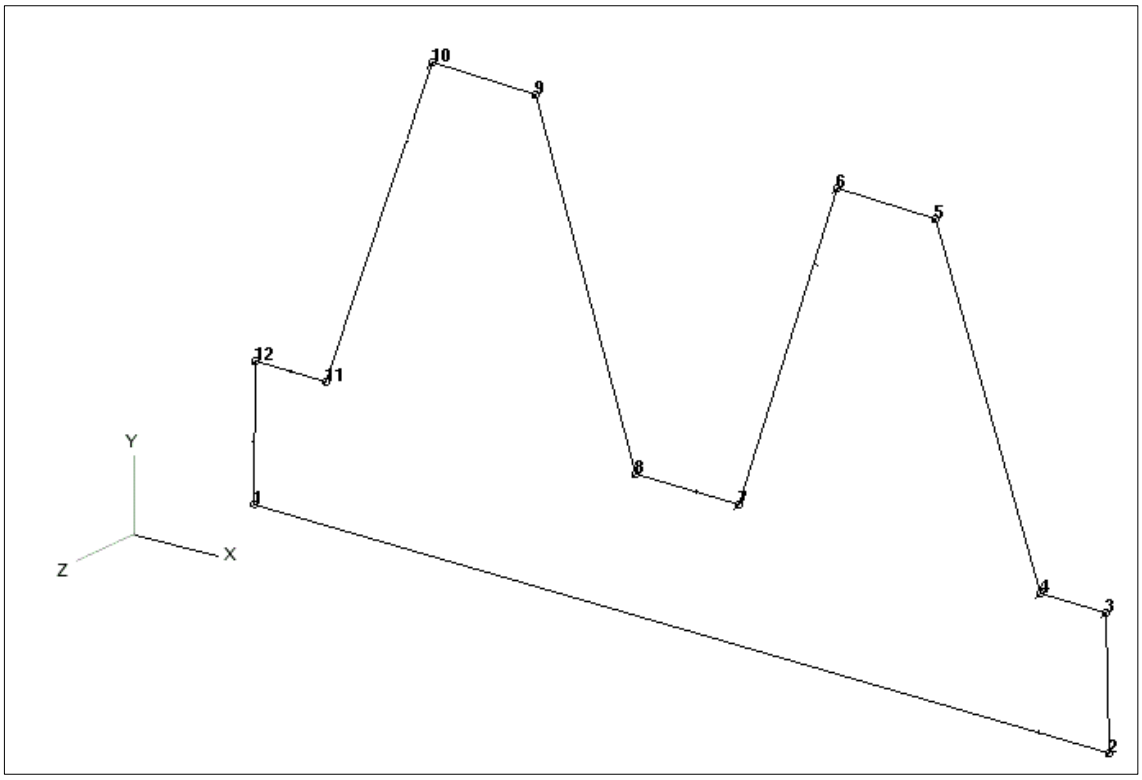


Figura 11.10: Creación de una base irregular en el plano XY

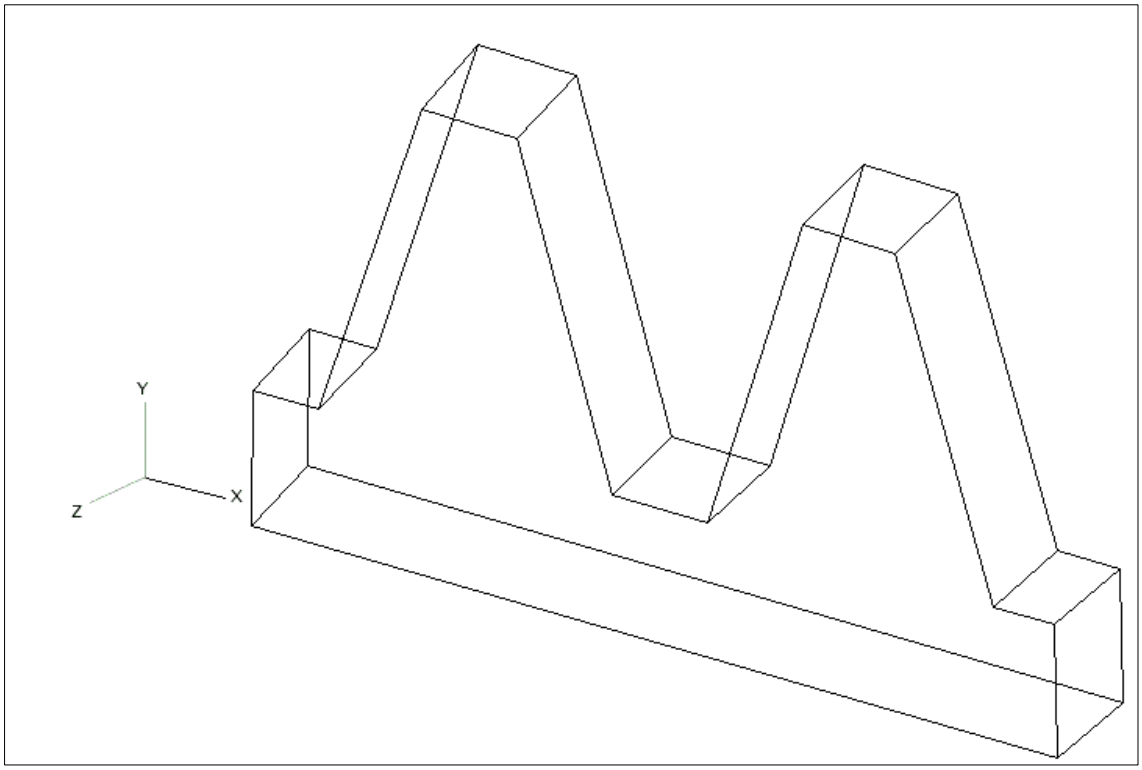


Figura 11.11: Base irregular extruída

La opción (2) permite leer los datos de los vértices de la base irregular ubicada también en el plano XY, pero esta vez la entrada de los datos se realiza por pantalla (ver figura 11.12); El programa le solicita al usuario tanto el número de vértices como las coordenadas de los mismos, también el nombre del archivo de texto donde se guardarán los datos (para una futura modificación que requiera realizar el usuario), y el nombre del sólido.

Se recomienda que el archivo de texto tenga siempre la extensión .dat (por ejemplo: base_nueva.dat), y que el nombre del archivo del sólido tenga la extensión .res (por ejemplo: base_nueva.res). El programa, cuando genera la representación gráfica del sólido, necesita solamente abrir el archivo del sólido (en este caso: base_nueva.res).

Esto lo hace de forma totalmente automática cuando el usuario pide por pantalla que se represente el sólido. El archivo .dat (base_nueva.dat) solo es de utilidad para el usuario que quiera modificar algún dato posteriormente.

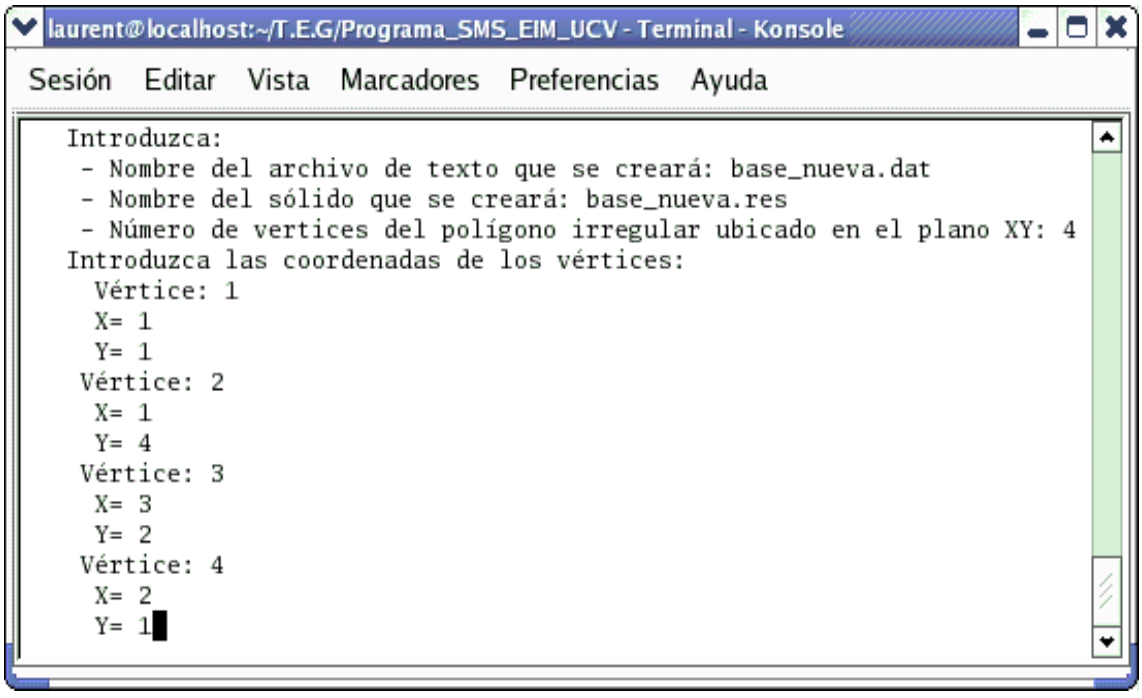


Figura 11.12: Pantalla para introducir datos de la base irregular

Por último se explicaran detalles de las funciones que permiten generar sólidos de revolución.

El botón “Revolución” permite ejecutar el subprograma mostrado en la figura 11.13, el cual fue diseñado para generar un sólido de revolución hueco (tubos por ejemplo), obtenido a partir de una curva cerrada (polígono) ubicada en el plano XY, la cual se hace girar alrededor del eje X.

Al igual que para la función que genera sólidos por extrusión a partir de una base irregular, este subprograma "Revolución" permite la entrada de los vértices del polígono directamente por pantalla o mediante la lectura de un archivo de texto. Lo importante que debe subrayarse (y que también aparecerá como información para el usuario a la hora de ejecutar dicha función), es que el programa está diseñado para unir el último vértice al primero, quedando así definida la base poligonal que permitirá, por rotación alrededor del eje X, genera un sólido de revolución hueco.

Este subprograma también permite generar tuberías (mediante la introducción de los diámetros internos y externos de las mismas, y su longitud), y toroides (o toros).

Los sólidos de ejemplo realizados con la función revolución y cualquiera de las otras funciones se incluyen en el capítulo XII.

El botón "Solrev" permite generar sólidos de revolución macizos a partir de una curva del plano XY (polígono abierto) que gira alrededor del eje X. El algoritmo esta diseñado tanto para leer datos desde un archivo de texto, como para que el usuario ingrese directamente por pantalla los vértices del polígono abierto. Cabe destacar que el algoritmo solo permite que los vértices que constituyen la poligonal abierta tengan coordenadas $y>0$, es decir que el polígono abierto no intersecte el eje X.

La figura 11.14 muestra las opciones que existen al pulsar el botón Solrev.

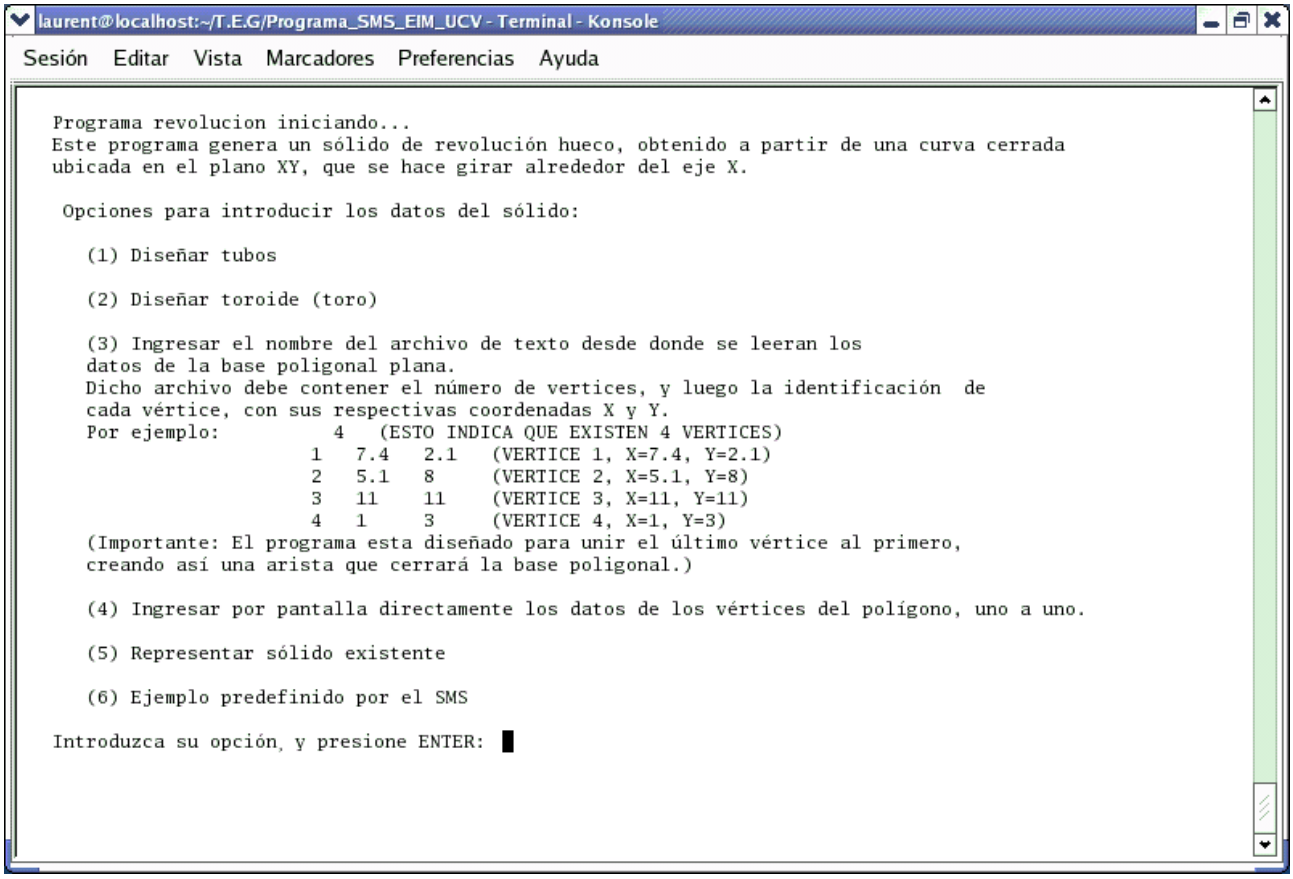


Figura 11.13: Menú del subprograma Revolución

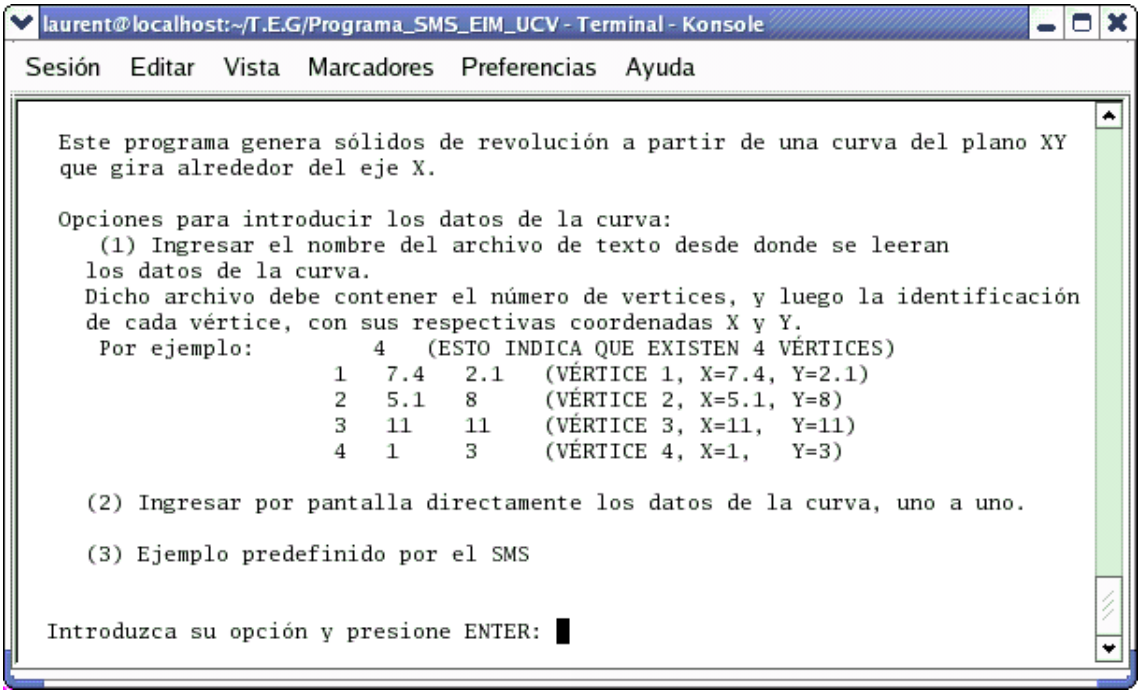


Figura 11.14: Menú del subprograma Solrev

Manipulación de sólidos:

Una vez que se genera un sólido en la ventana gráfica, es posible manipularlo a través de distintos botones que permiten rotarlo alrededor de algunos de los ejes, realizarle un acercamiento, o alejamiento, ver junto a cada vértice el número que lo identifica, desplazar el sólido hacia el marco superior de la ventana o hacia el marco inferior.

En el Capítulo XII se incluyen al detalle los botones y sus respectivas funciones para manipular un sólido; sin embargo aquí se presentan algunas explicaciones de los botones más relevantes.

La figura 11.15 muestra una esfera recién creada. Se observa que por su dimensión, la misma no se aprecia completa en la pantalla gráfica. El usuario puede aplicar varias veces el alejamiento (botón ZoomO) hasta que se aprecie la esfera dentro del marco de la ventana (fig.11.16).

Puede igualmente agrandarse la esfera aplicando la operación inversa (botón ZoomI).

Si se pulsa el botón “Resetear”, la esfera regresará a su tamaño y ubicación original en la ventana.

La figura 11.17 muestra un caucho de un vehículo realizado mediante la función revolución. Luego de aplicarle rotaciones alrededor de los ejes X y Y (botones +ROTX y +ROTY), se obtiene otra perspectiva del sólido (fig.11.18)

La figura 11.19 muestra un cubo al que se le aplicó la función dada por el botón Nodos, con lo que se observan los vértices con sus números que los identifican. Al pulsar varias veces el botón Subir, el cubo se desplaza hacia el marco superior de la ventana (fig.11.20).

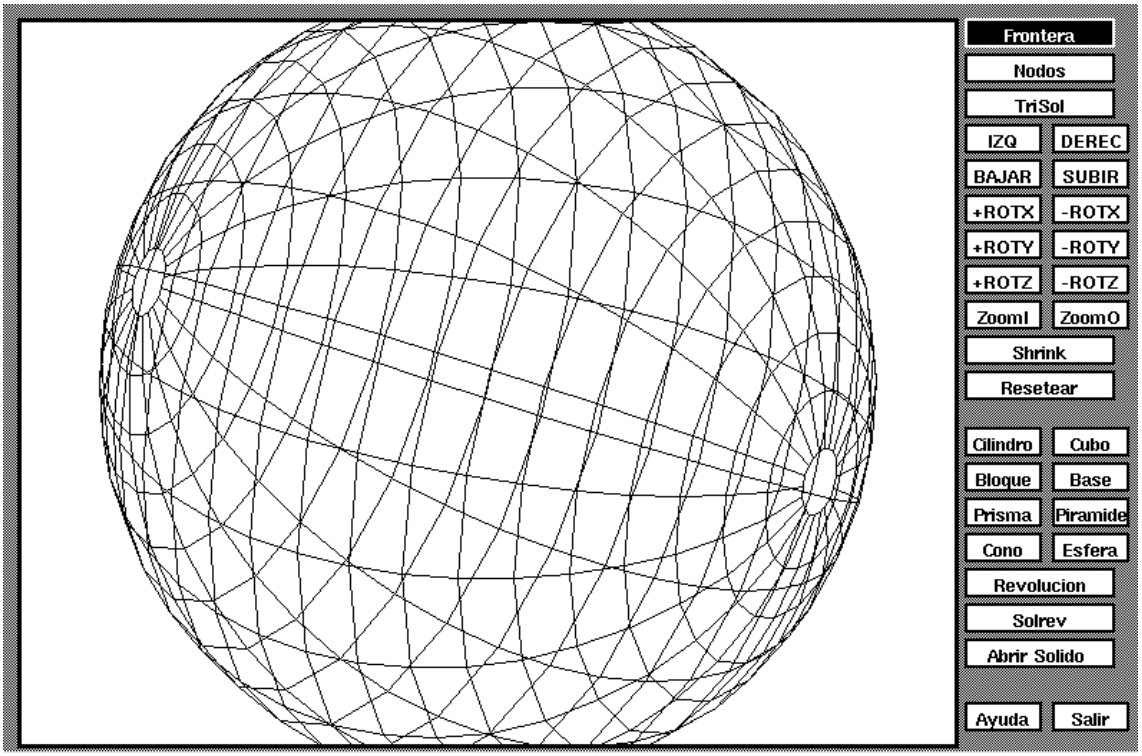


Figura 11.15: Esfera que sobresale del marco de la pantalla gráfica

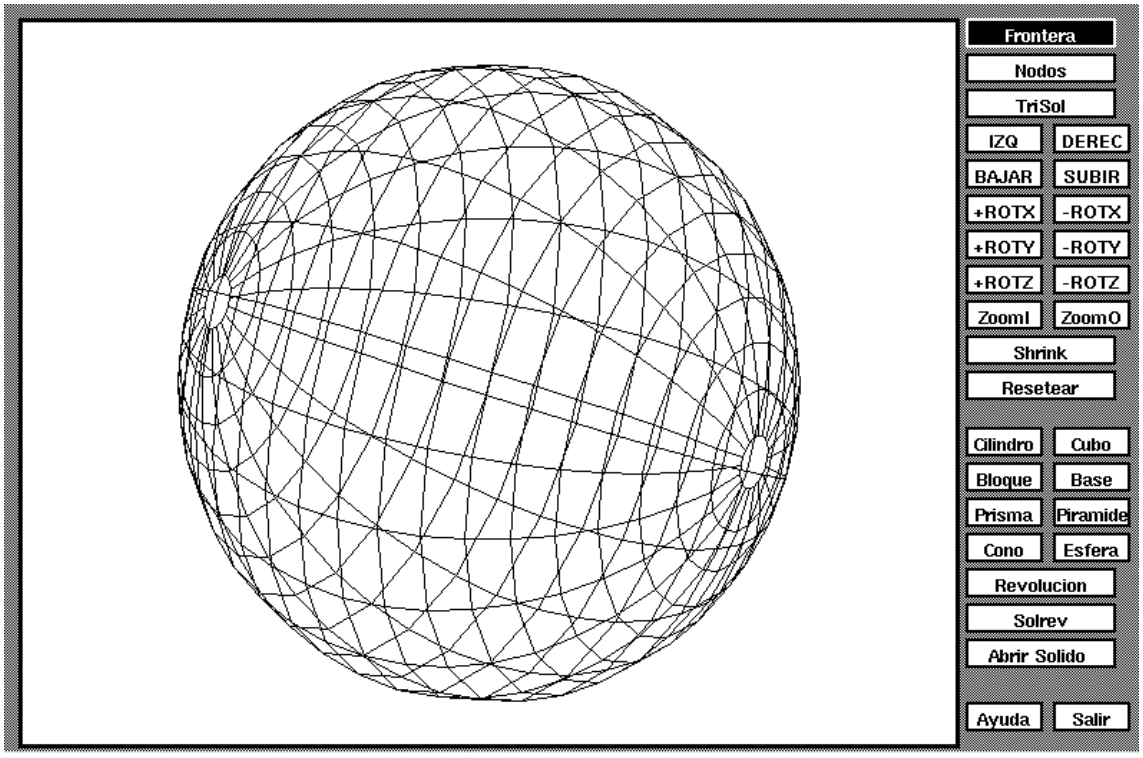


Figura 11.16: Esfera dentro del marco de la pantalla gráfica, una vez aplicada la función de alejamiento

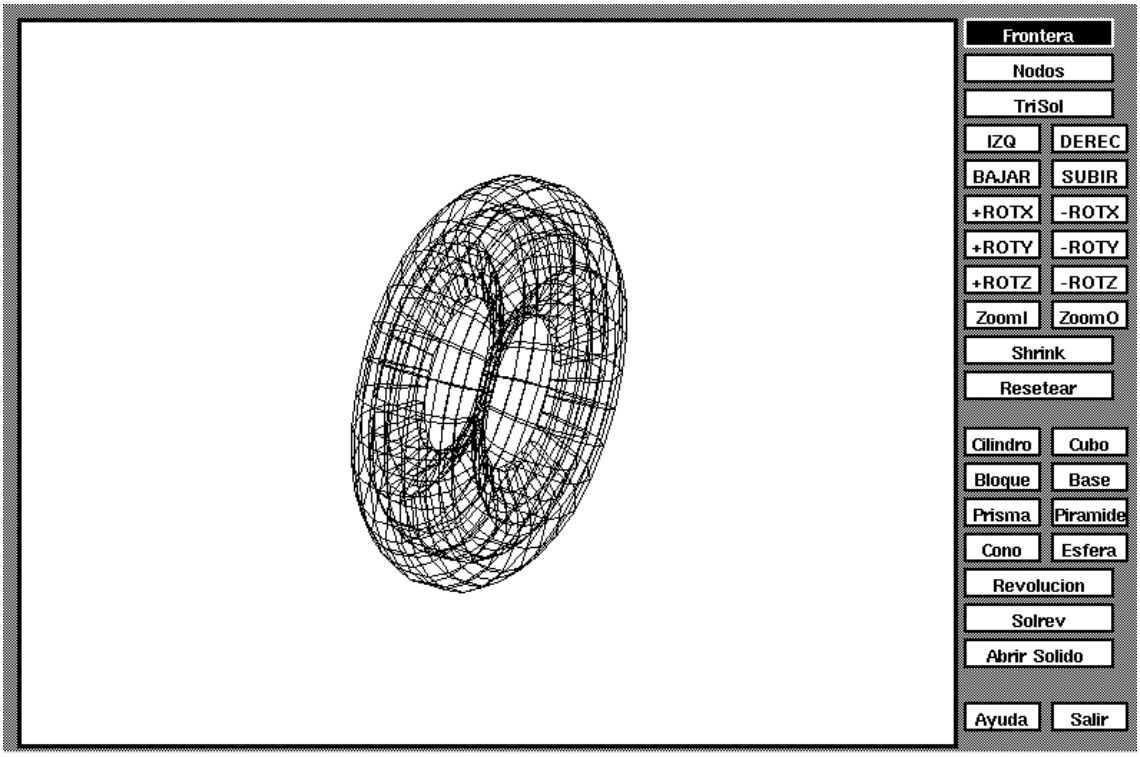


Figura 11.17: Caucho realizado mediante la función solrev (Botón Solrev)

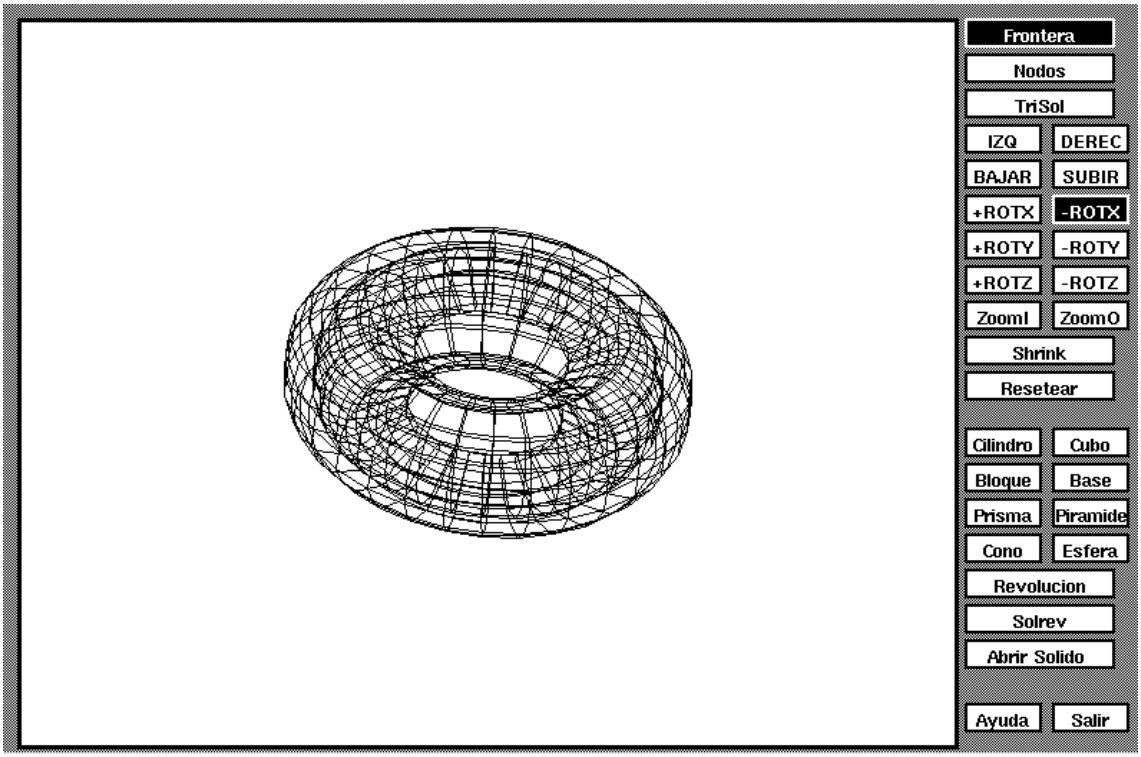


Figura 11.18: Caucho visto desde otra perspectiva

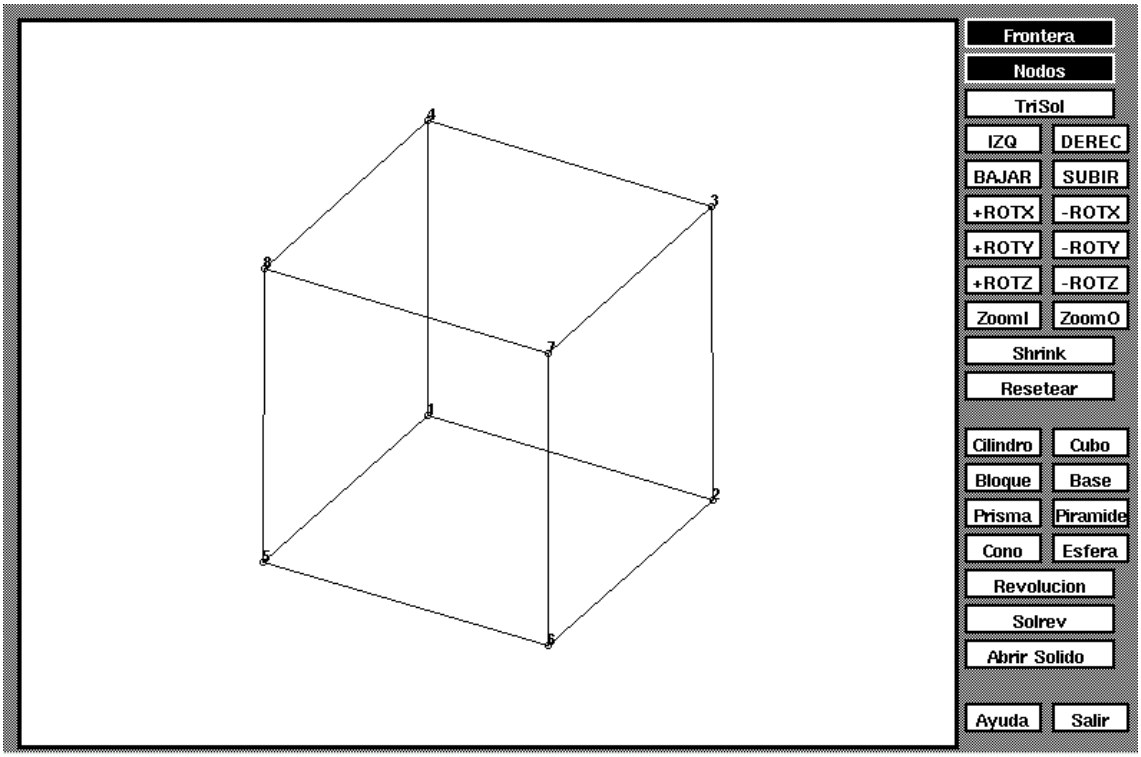


Figura 11.19: Cubo con sus vértices identificados

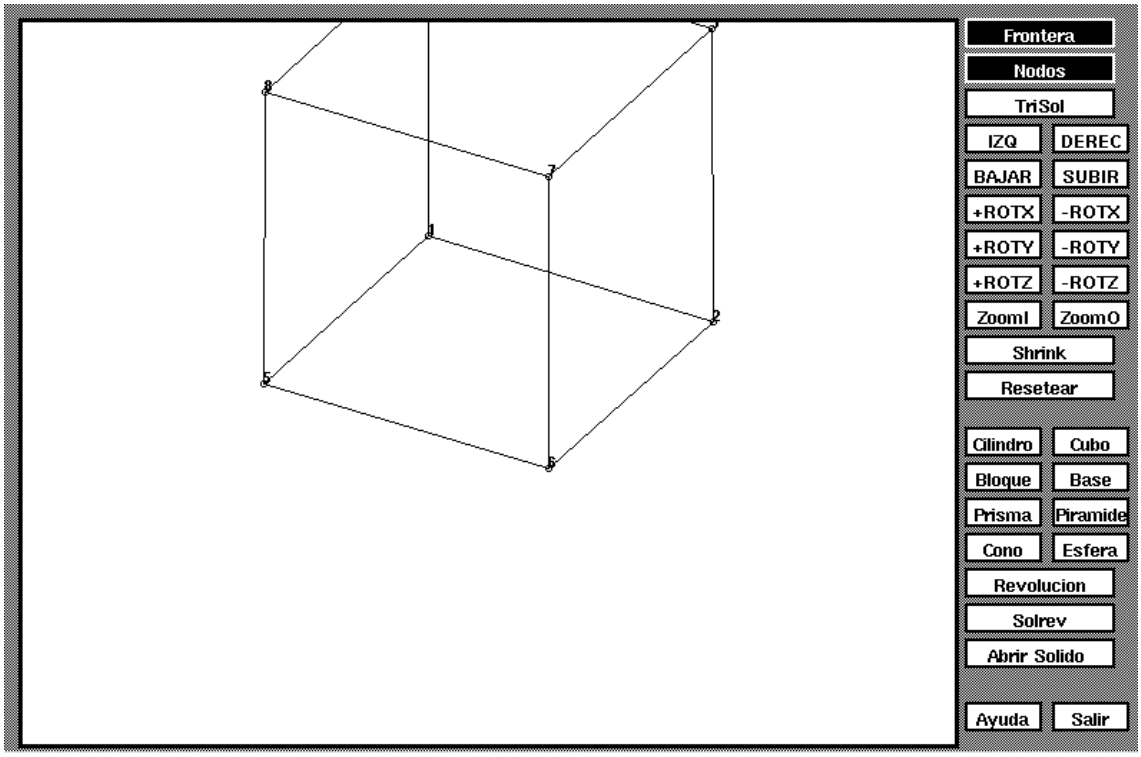
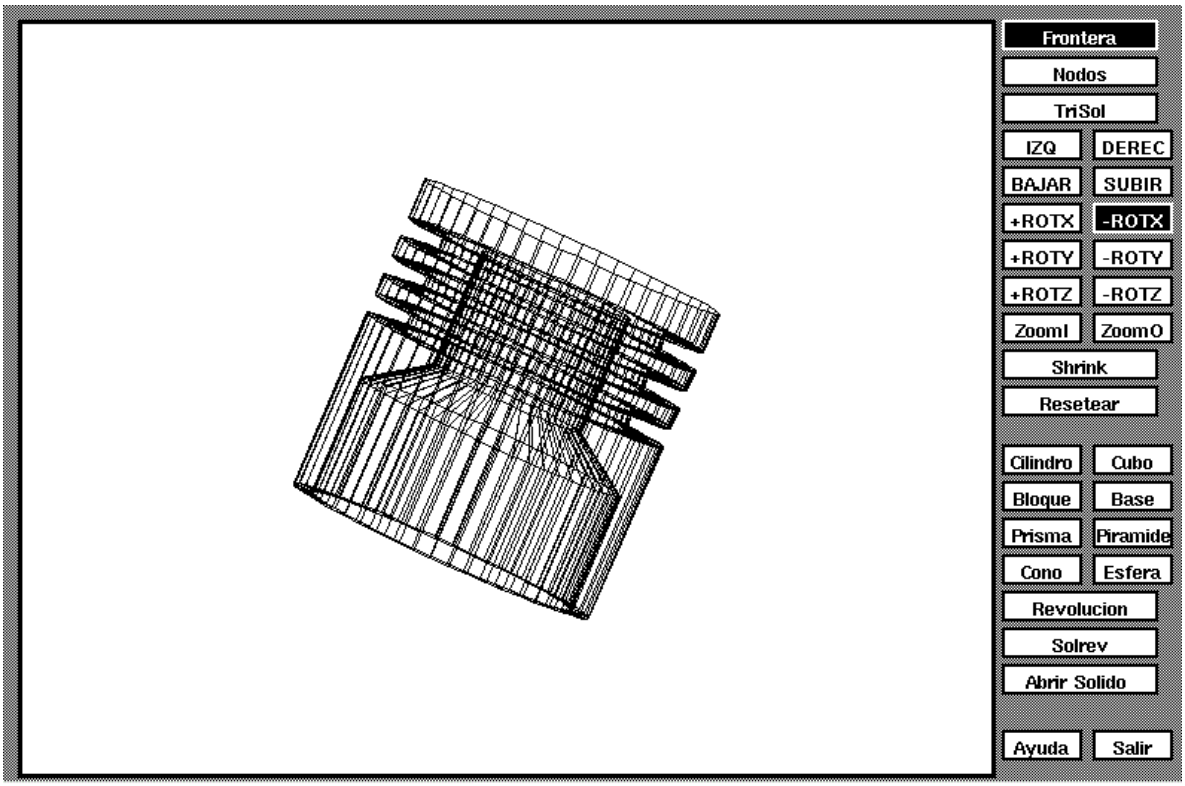


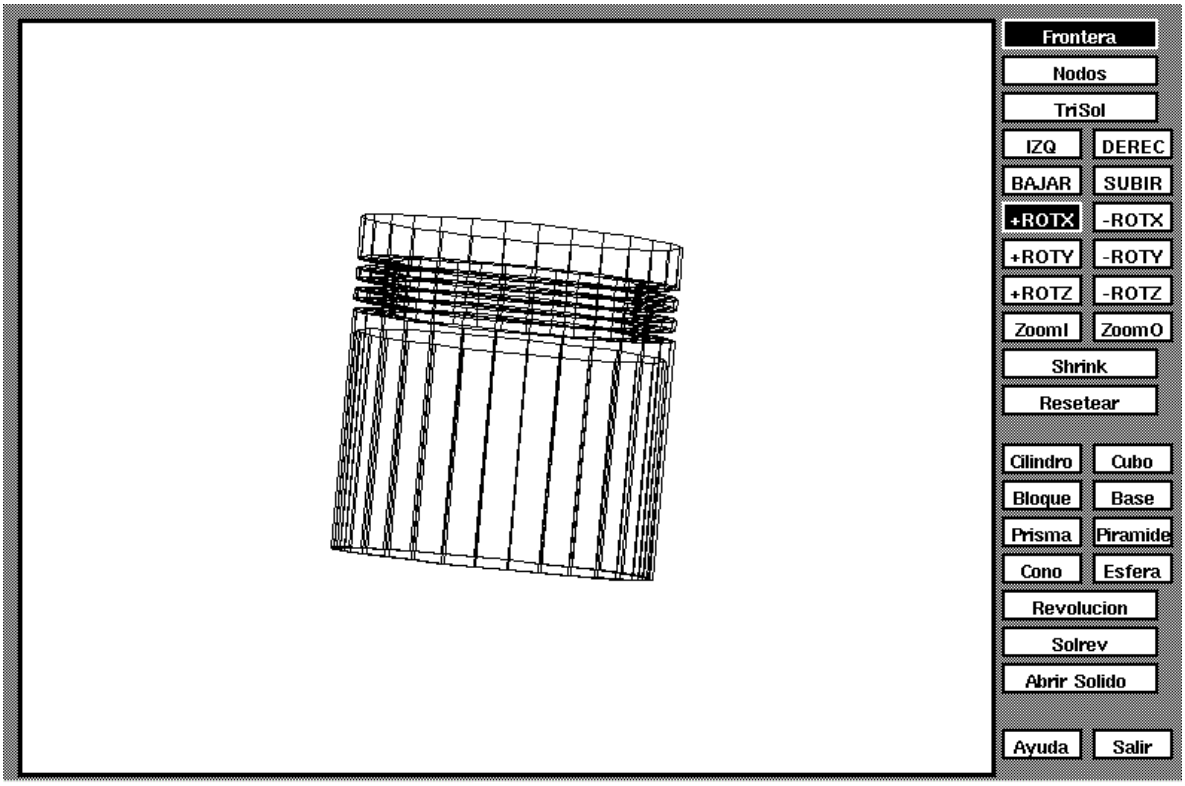
Figura 11.20: Cubo desplazado hacia el marco superior

CAPÍTULO XII

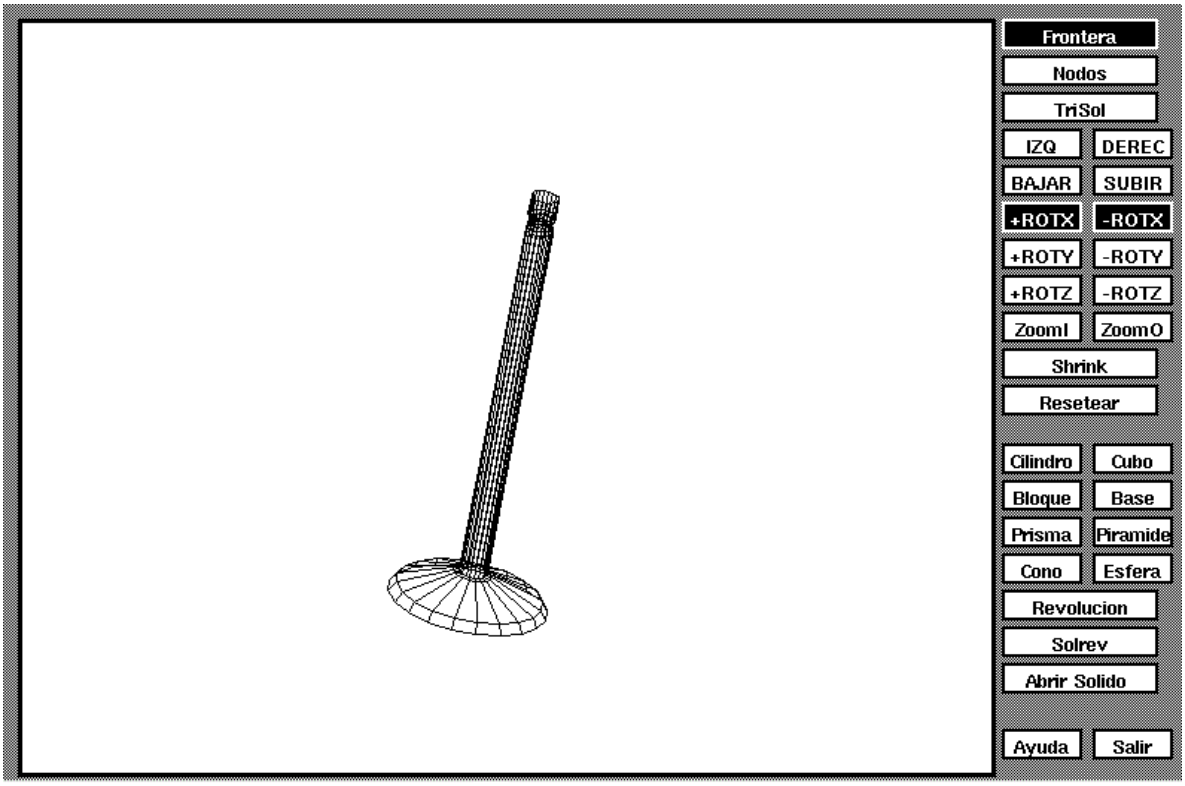
EJEMPLOS DE SÓLIDOS CREADOS CON EL PROGRAMA SMS_EIM_UCV



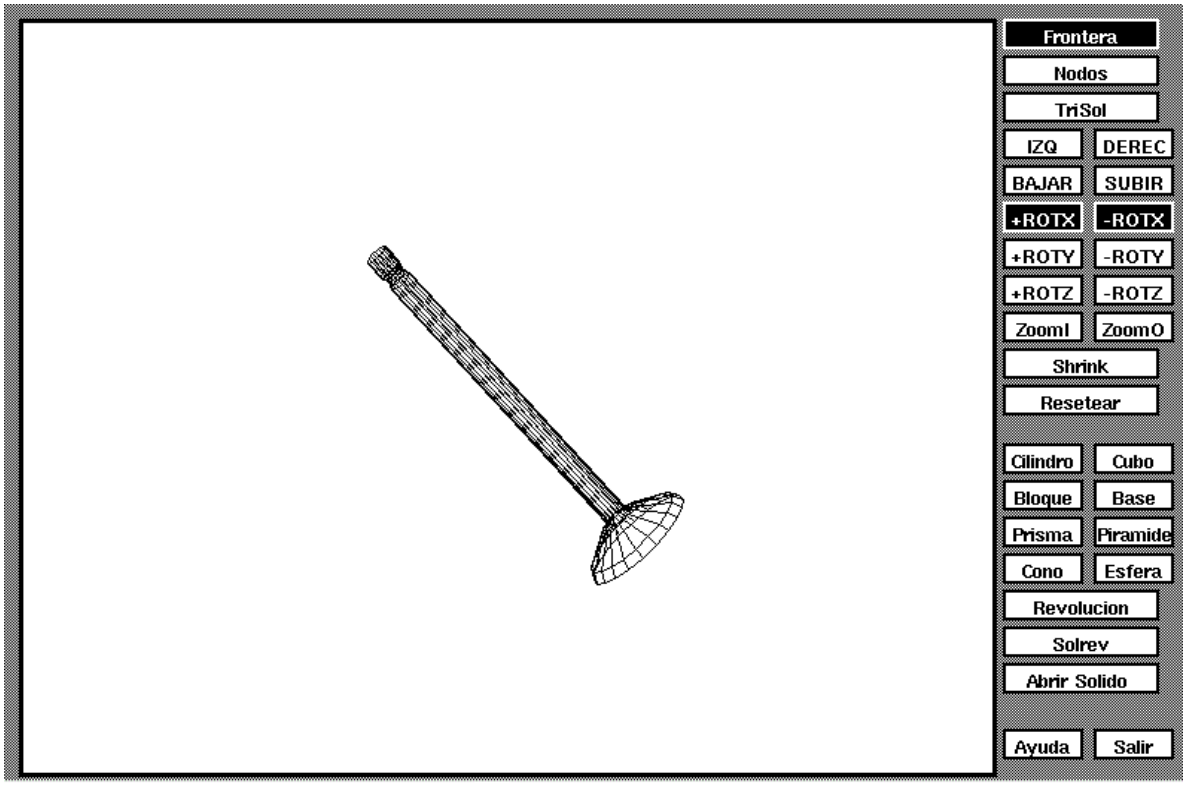
Nombre de archivo: piston1.res



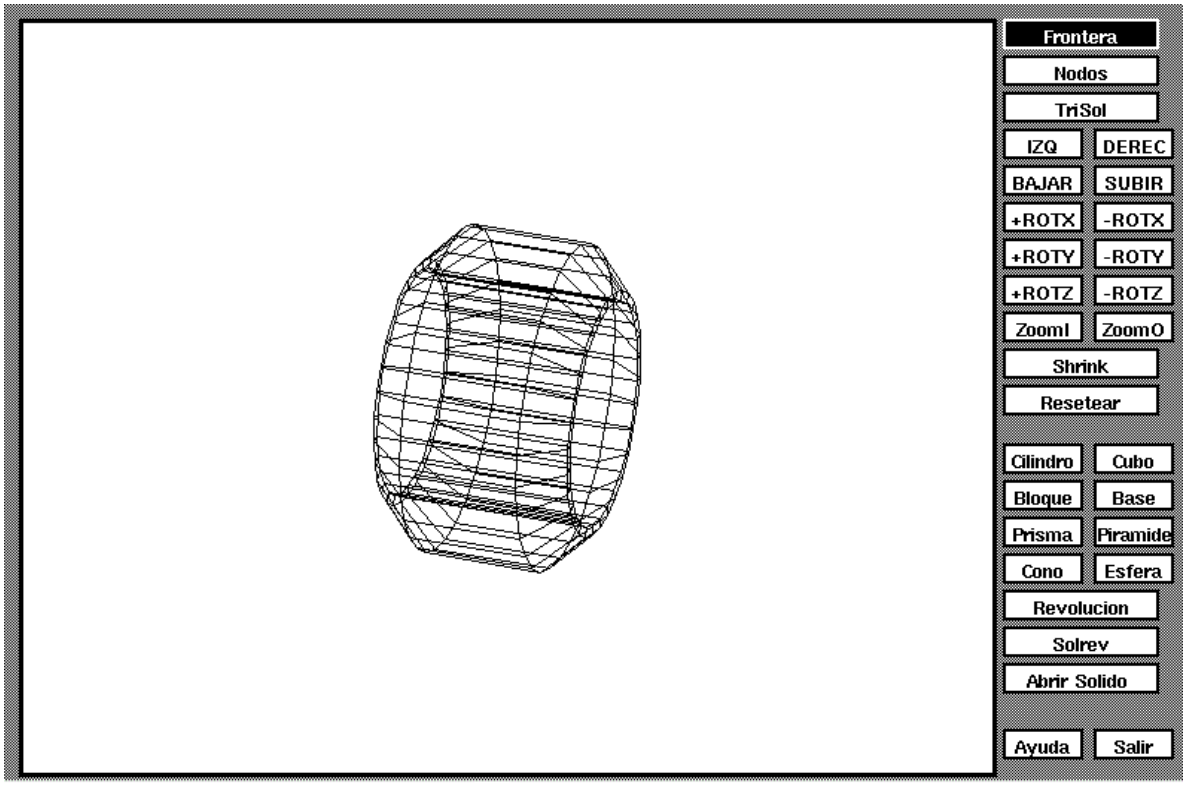
Nombre de archivo: piston2.res



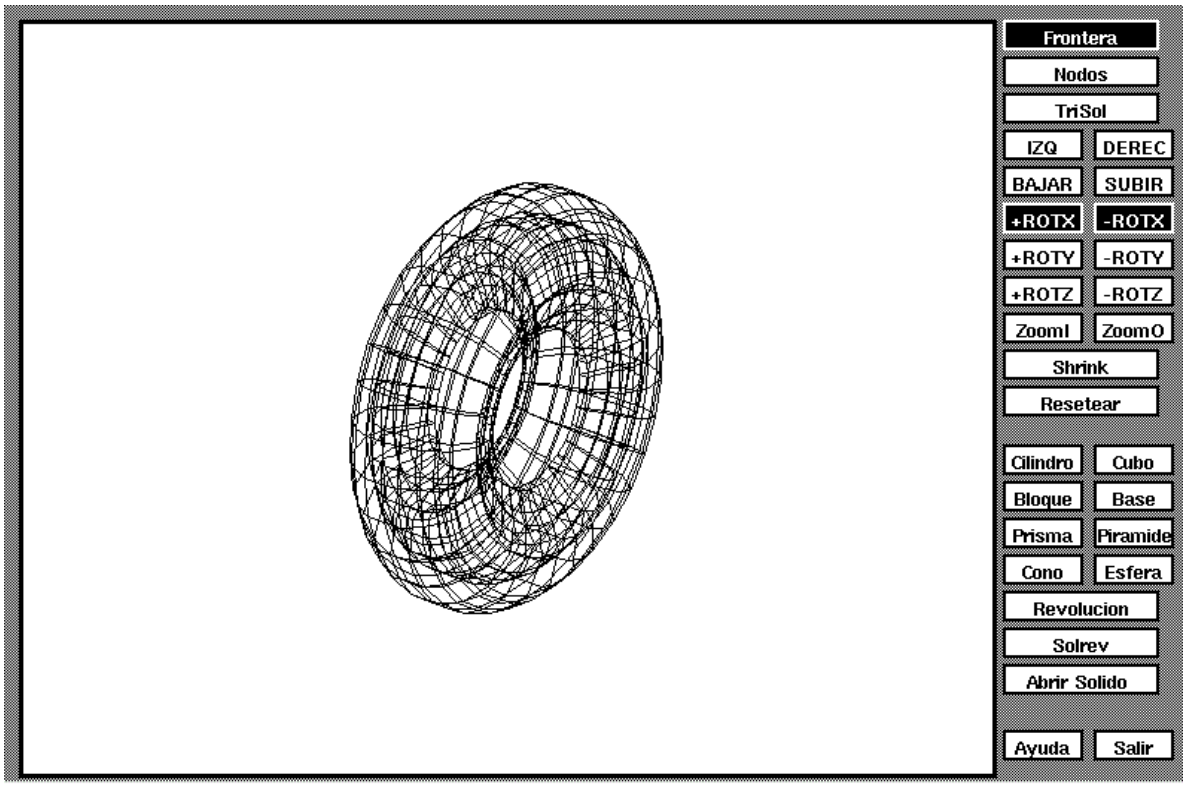
Nombre de archivo: valvula_admision.res



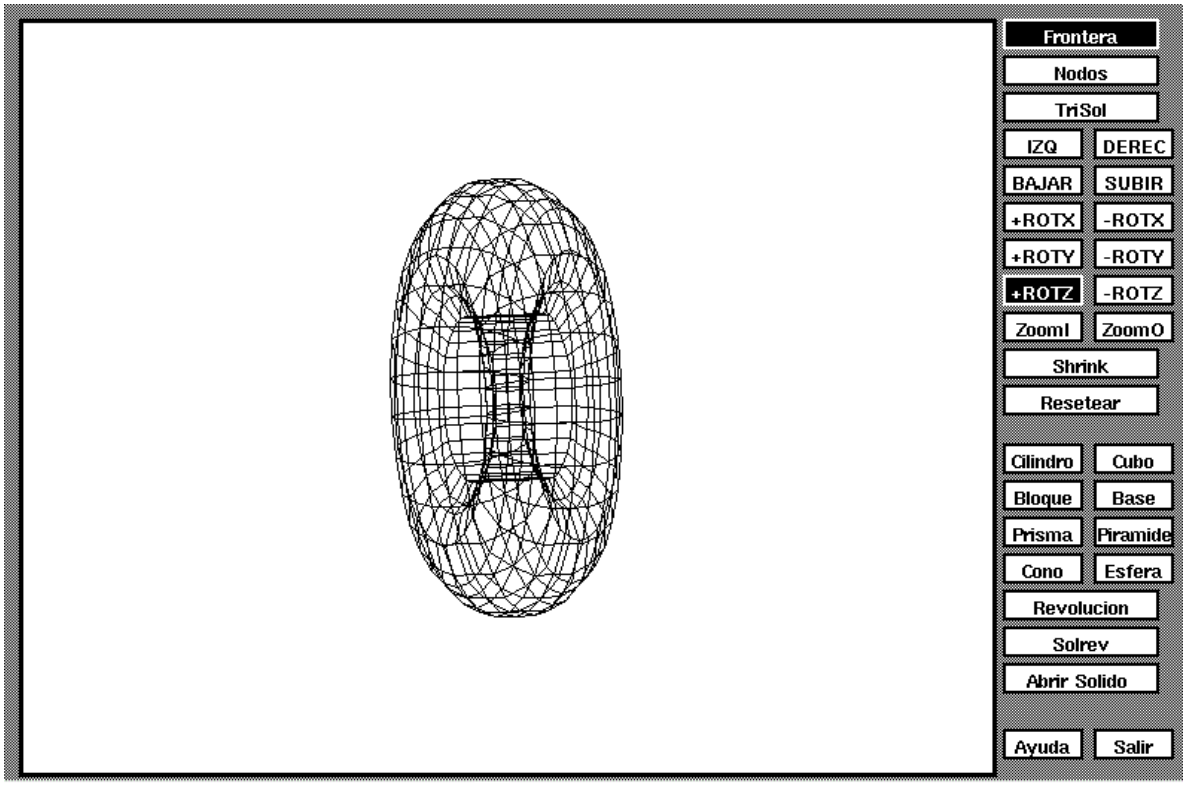
Nombre de archivo: valvula_escape.res



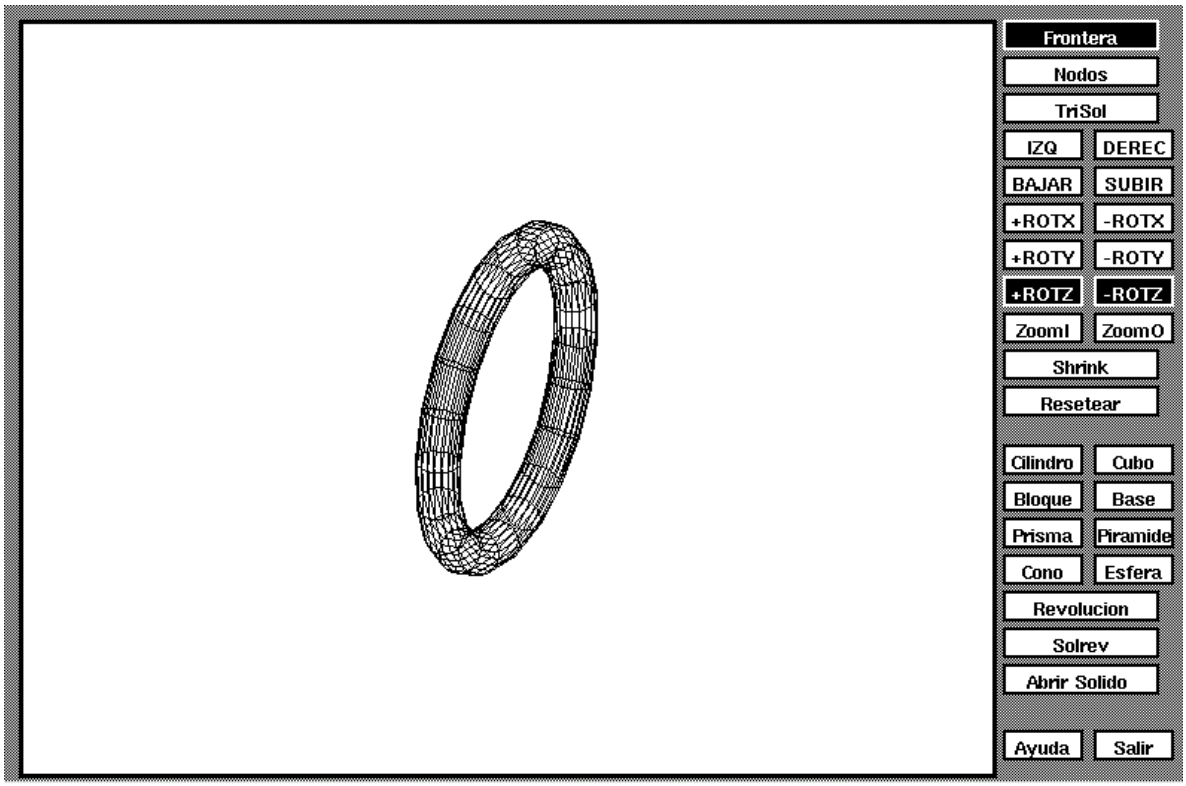
Nombre de archivo: ejemplo_revolucion.res (caucho)



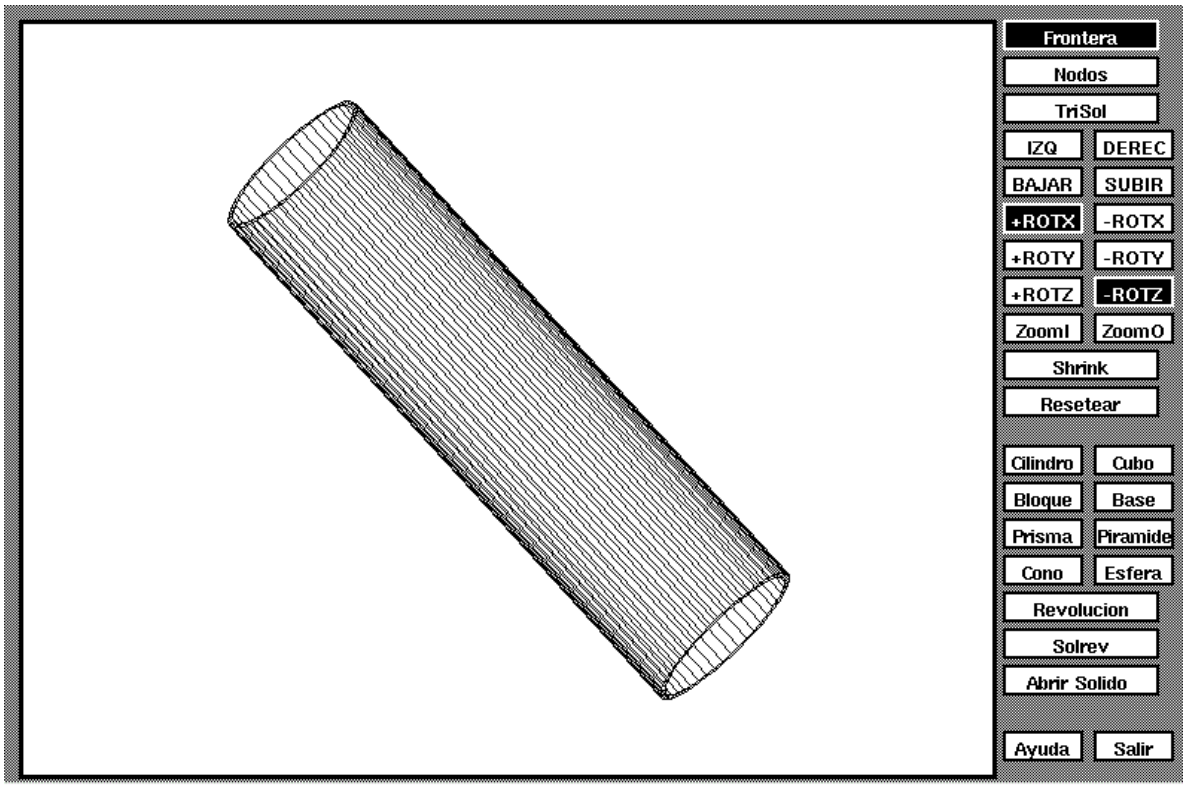
Nombre de archivo: caucho2.res



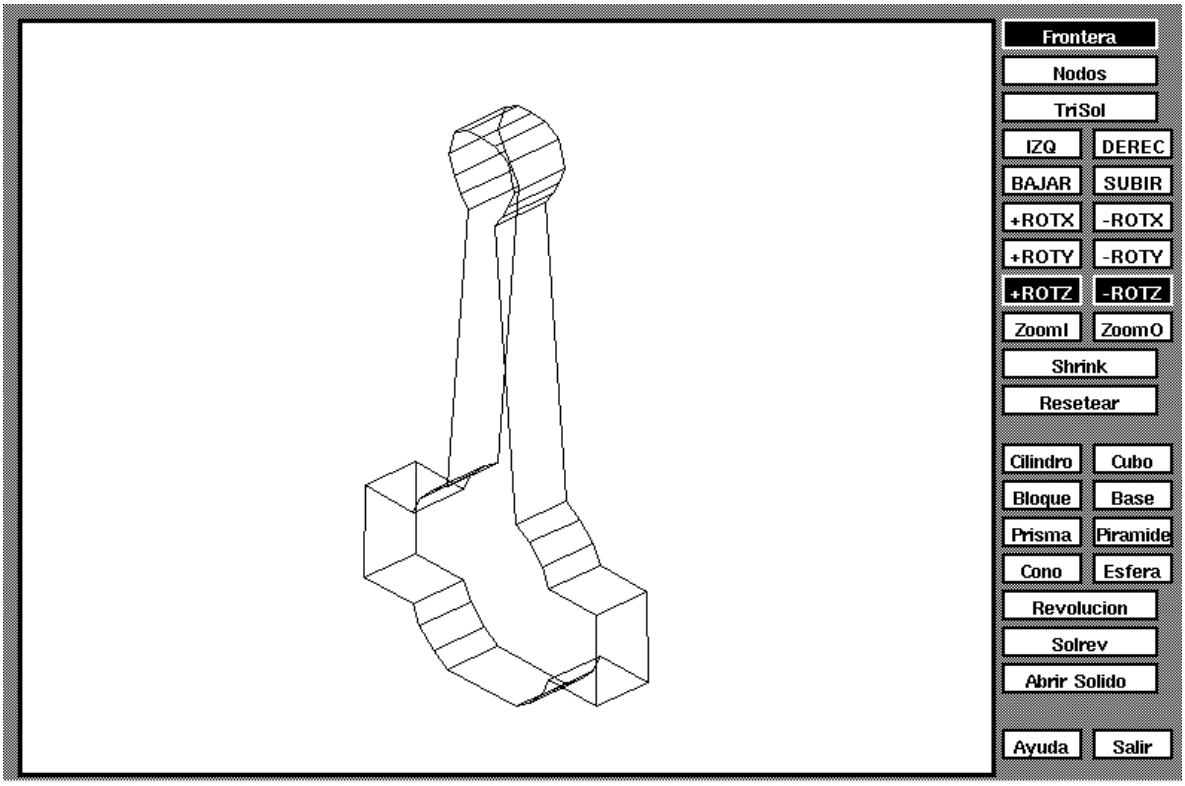
Nombre de archivo: caucho1.res



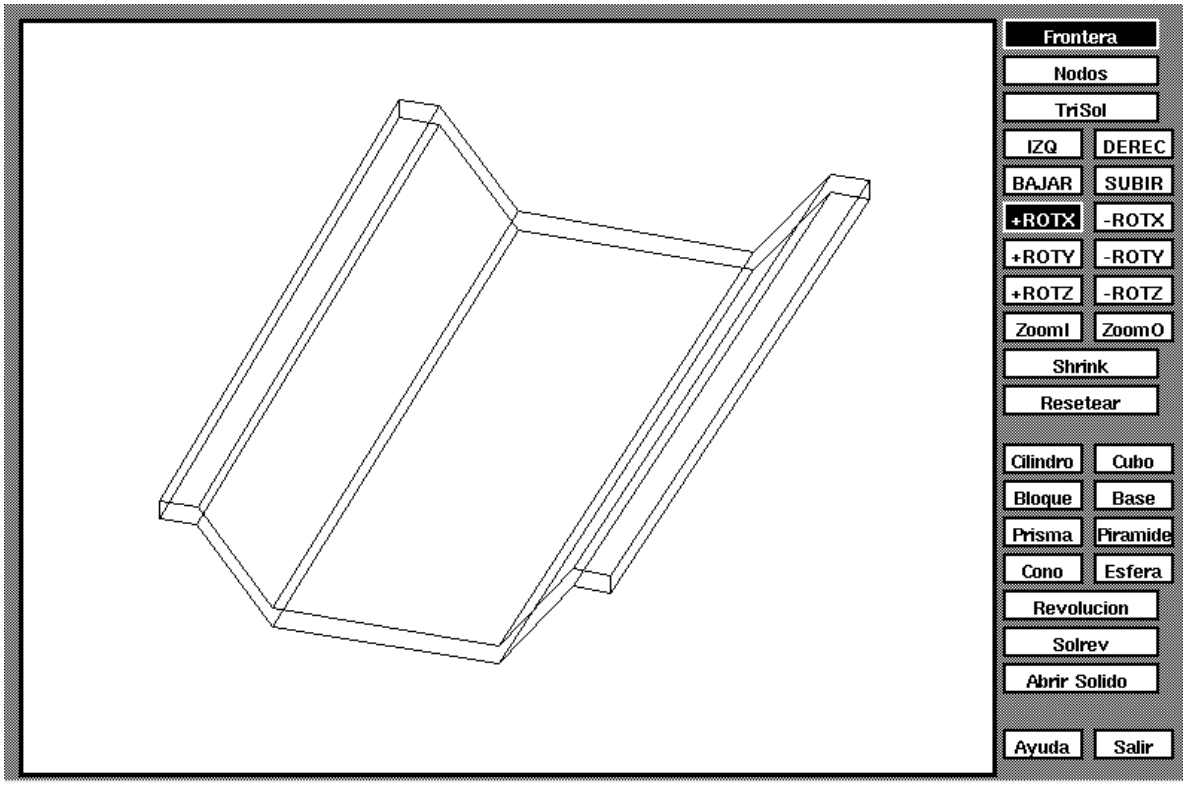
Nombre de archivo: toroide.res



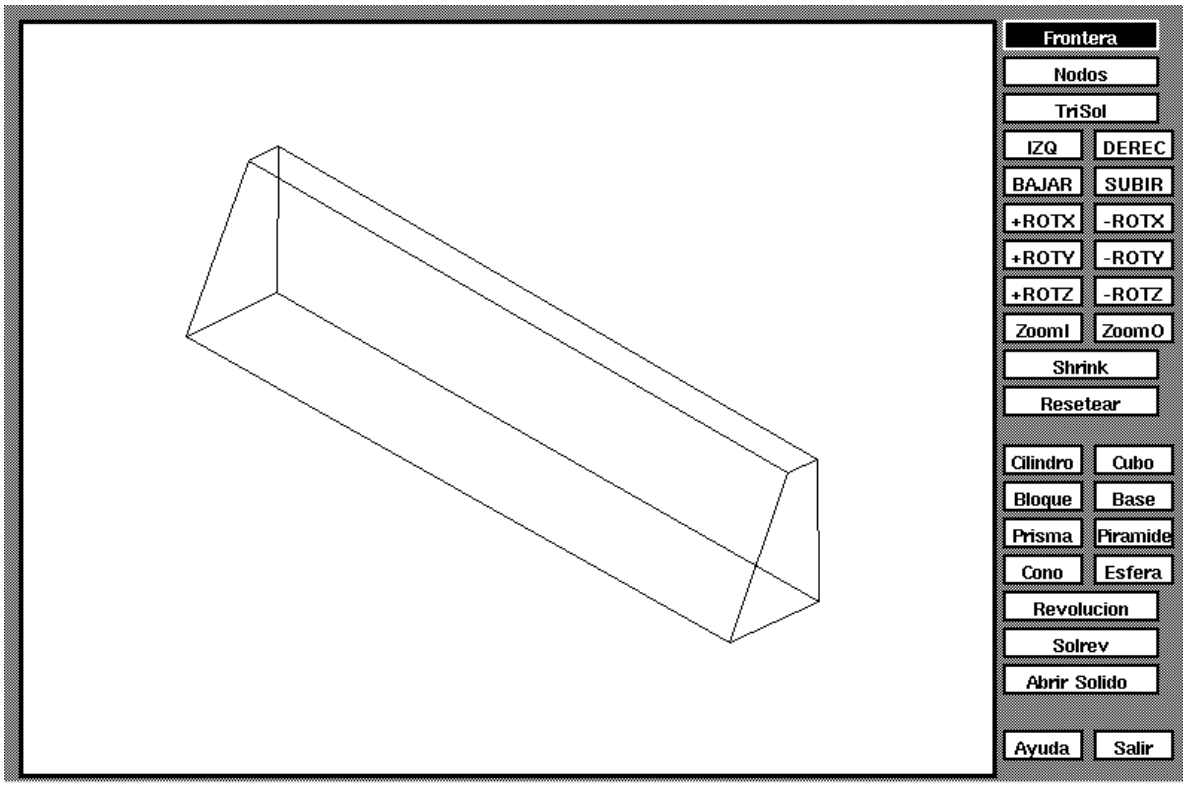
Nombre de archivo: tuberia.res



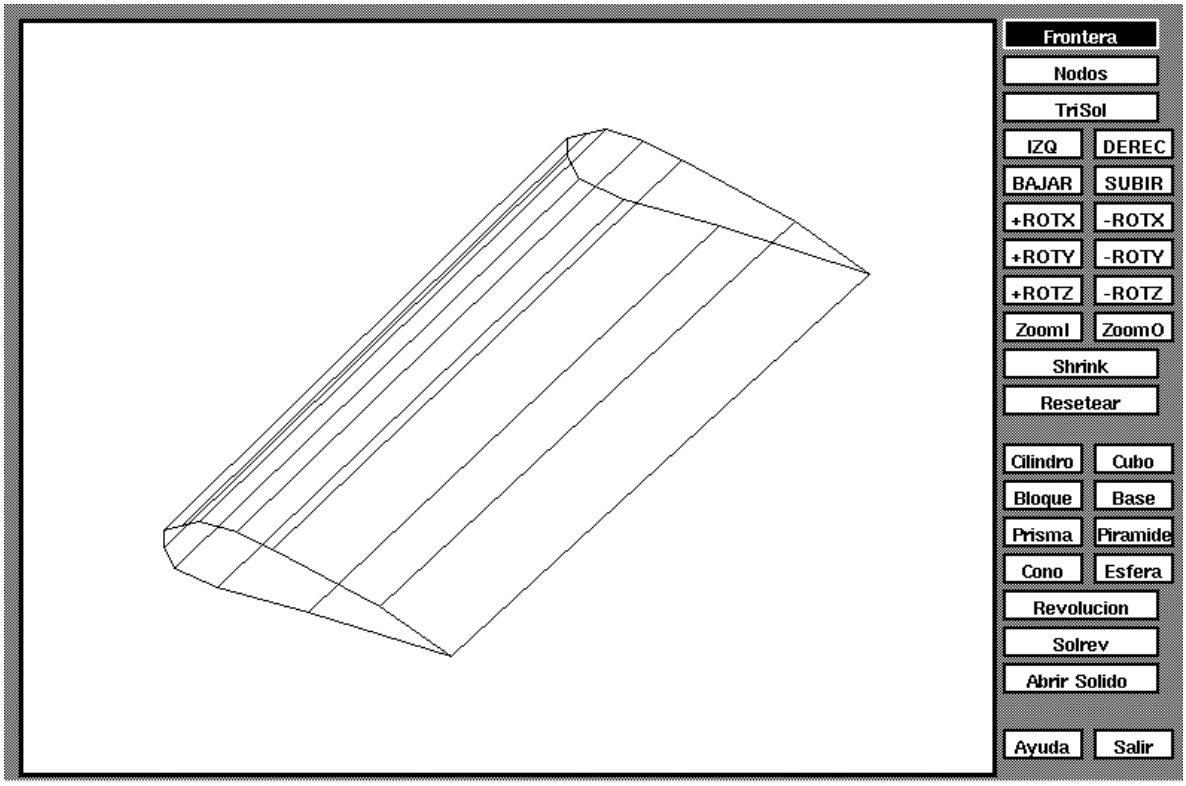
Nombre de archivo: biela_sin_maquinar.res



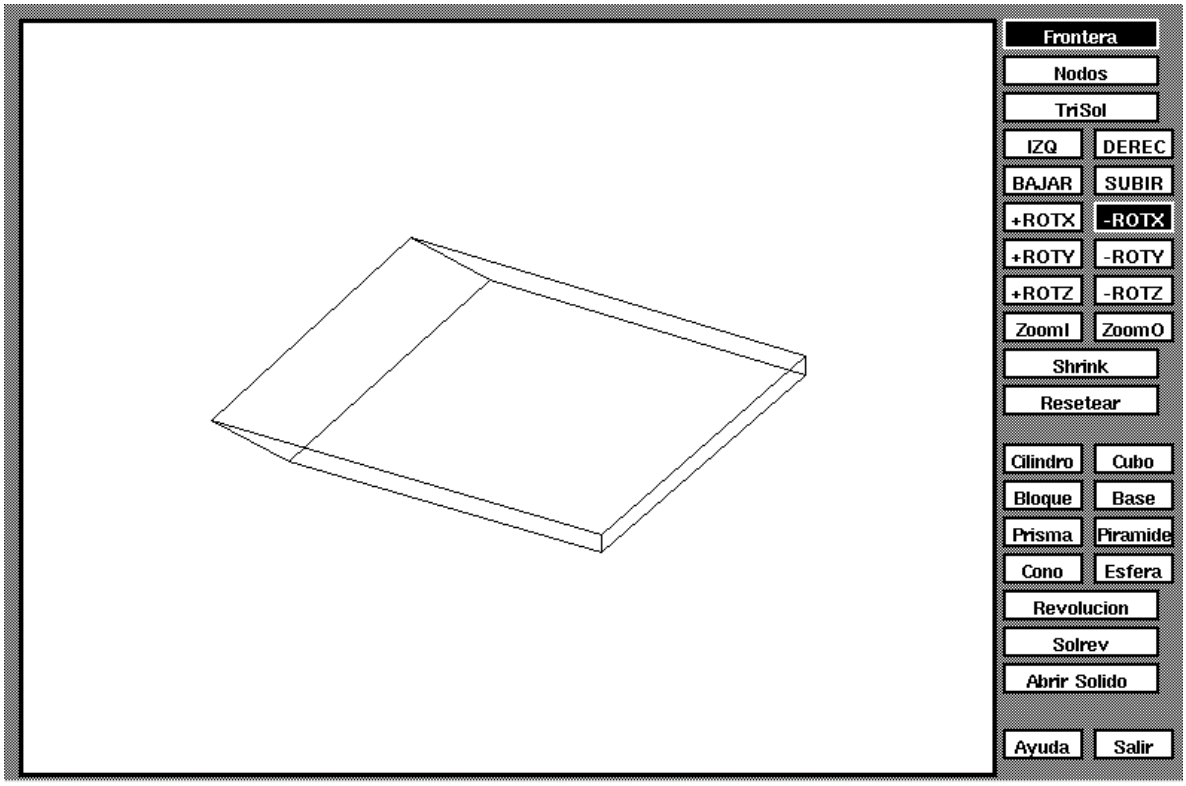
Nombre de archivo: canal_trapezoidal.res



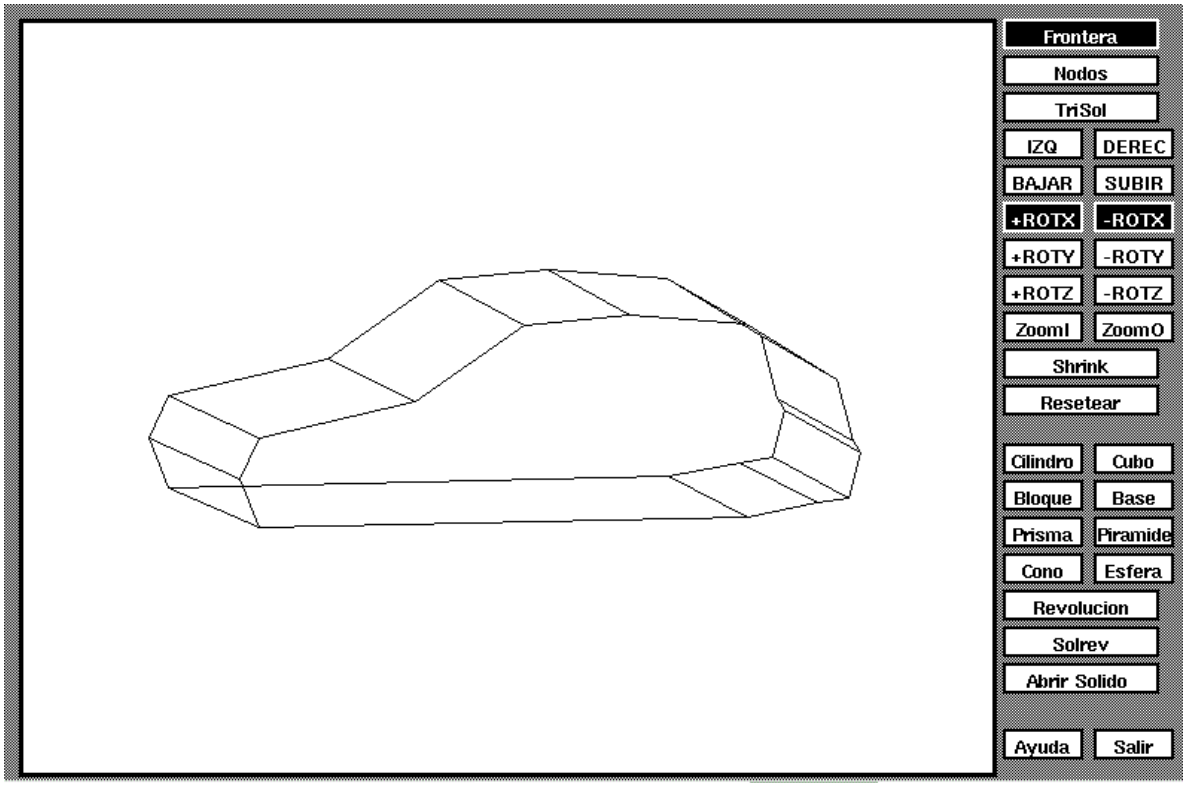
Nombre de archivo: presa_concreto.res



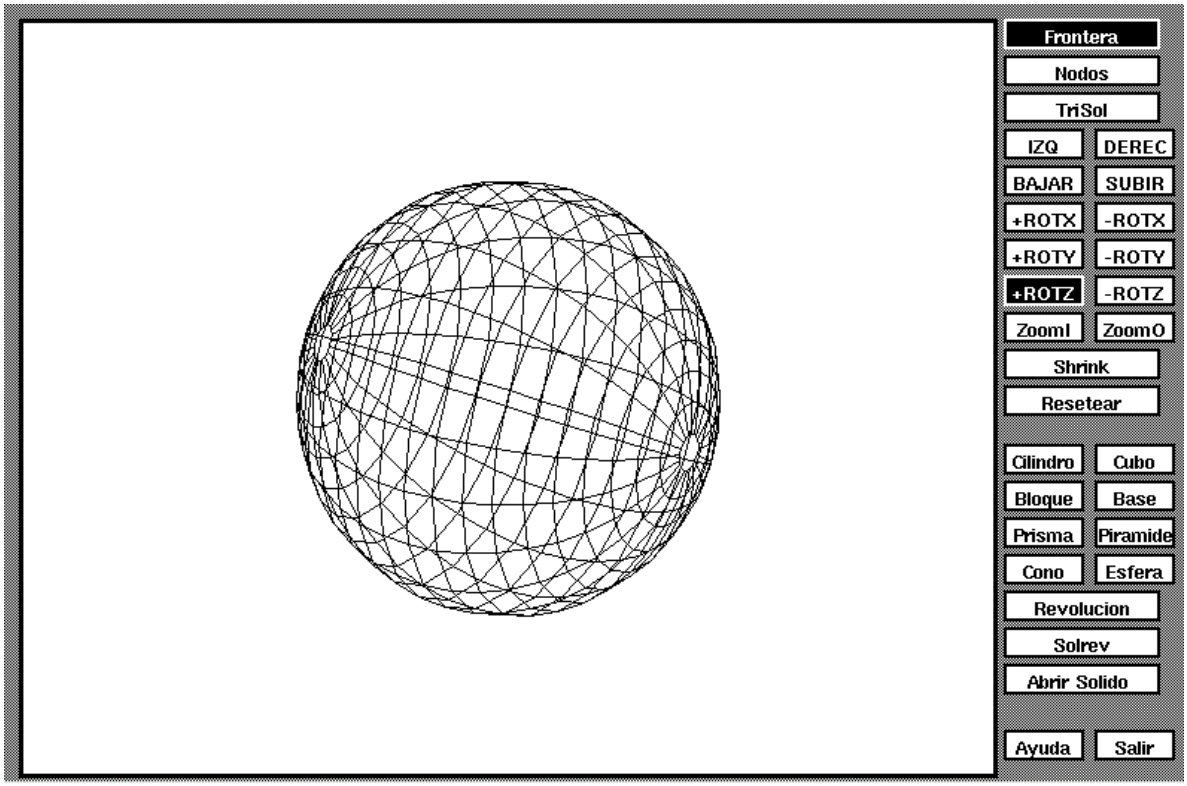
Nombre de archivo: ala.res



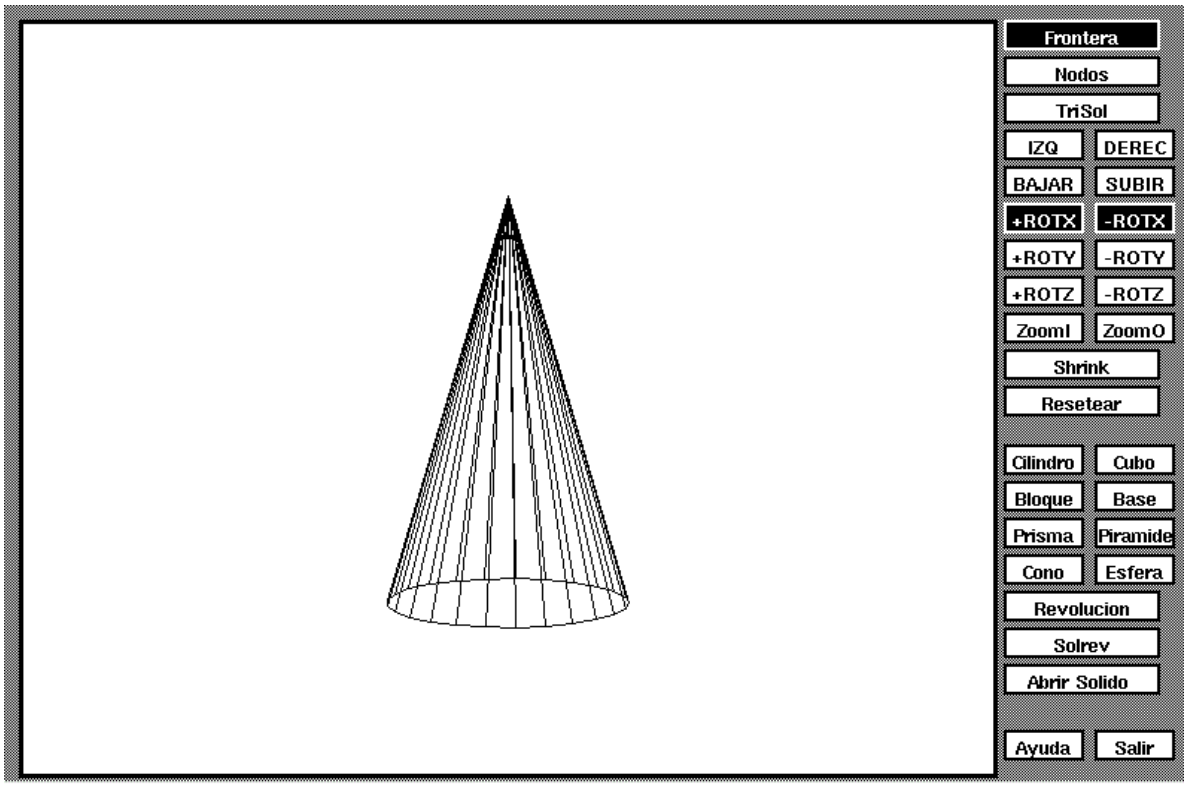
Nombre de archivo: placa_plana.res



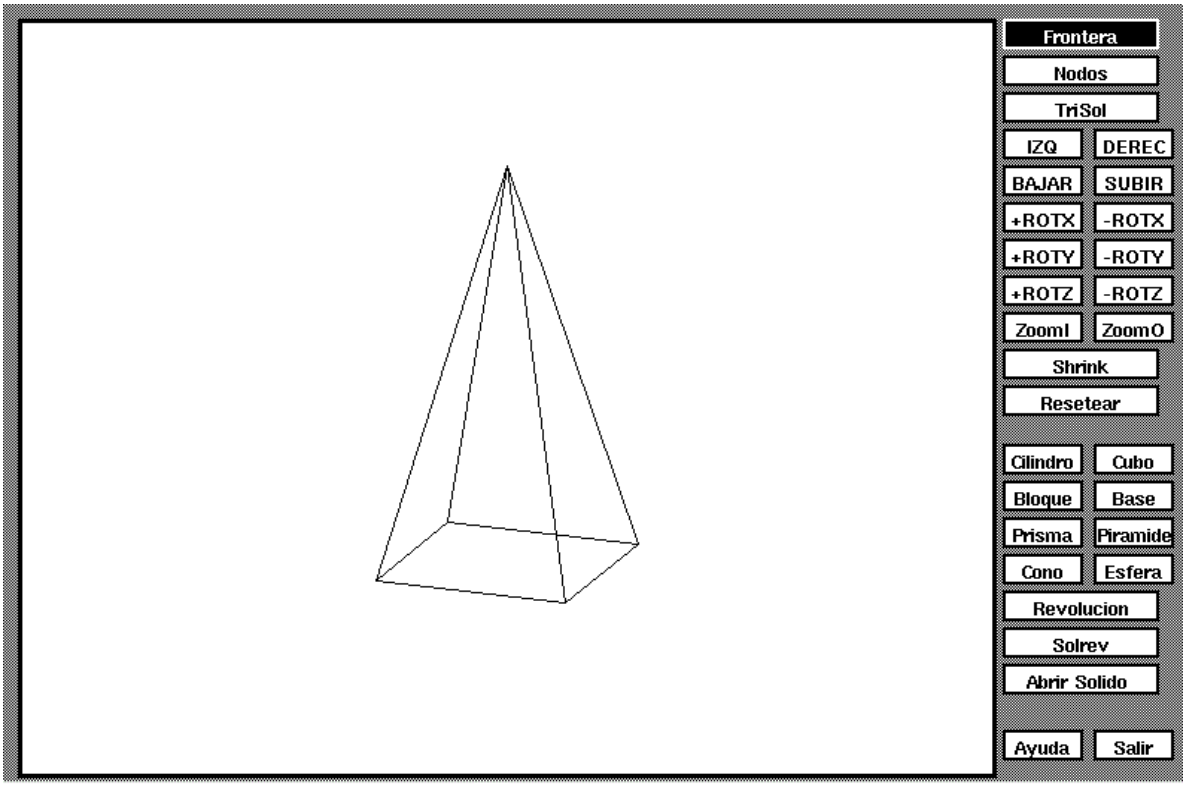
Nombre de archivo: vehiculo.res



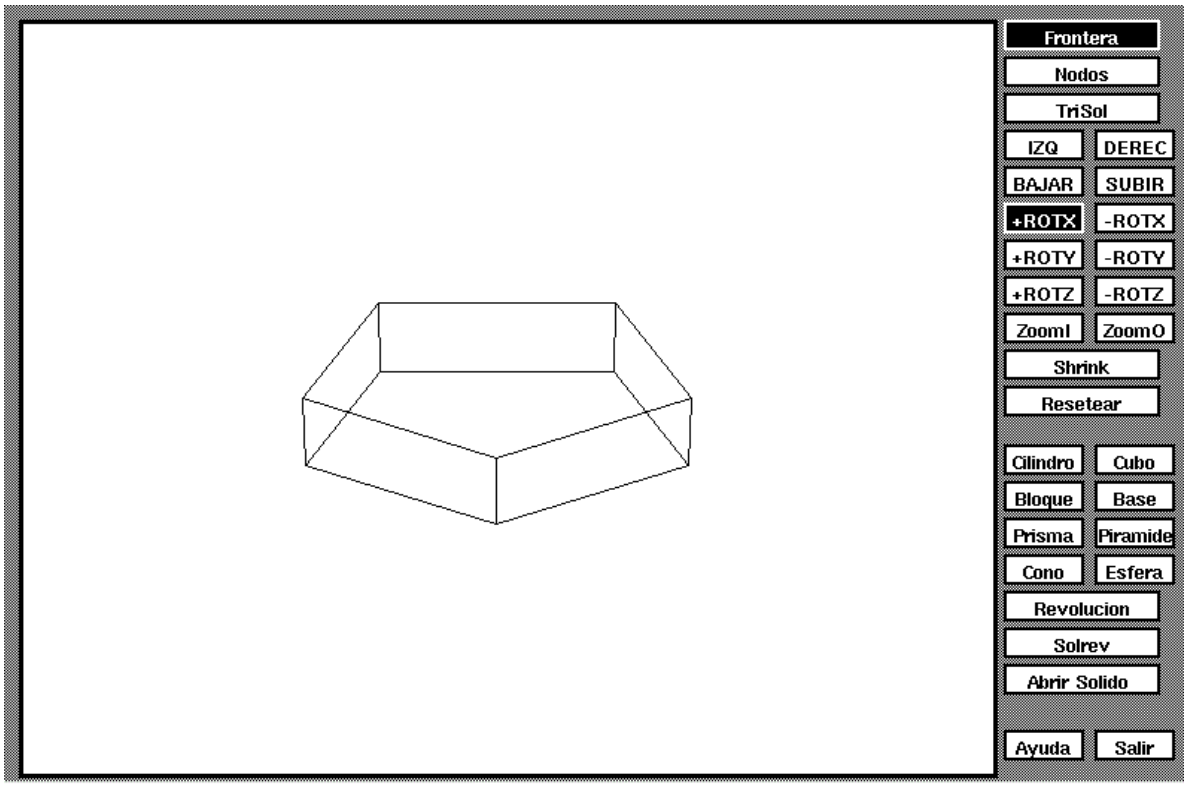
Nombre de archivo: ejemplo_esfera.res



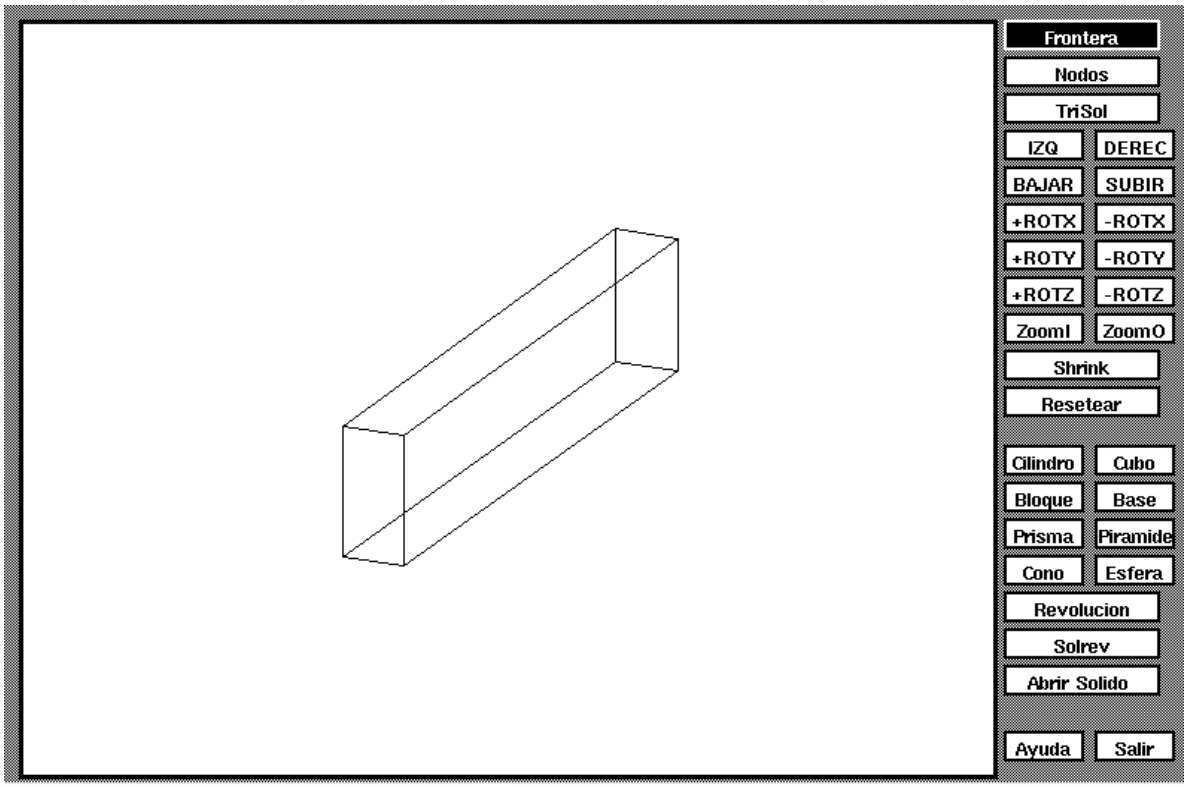
Nombre de archivo: ejemplo_cono.res



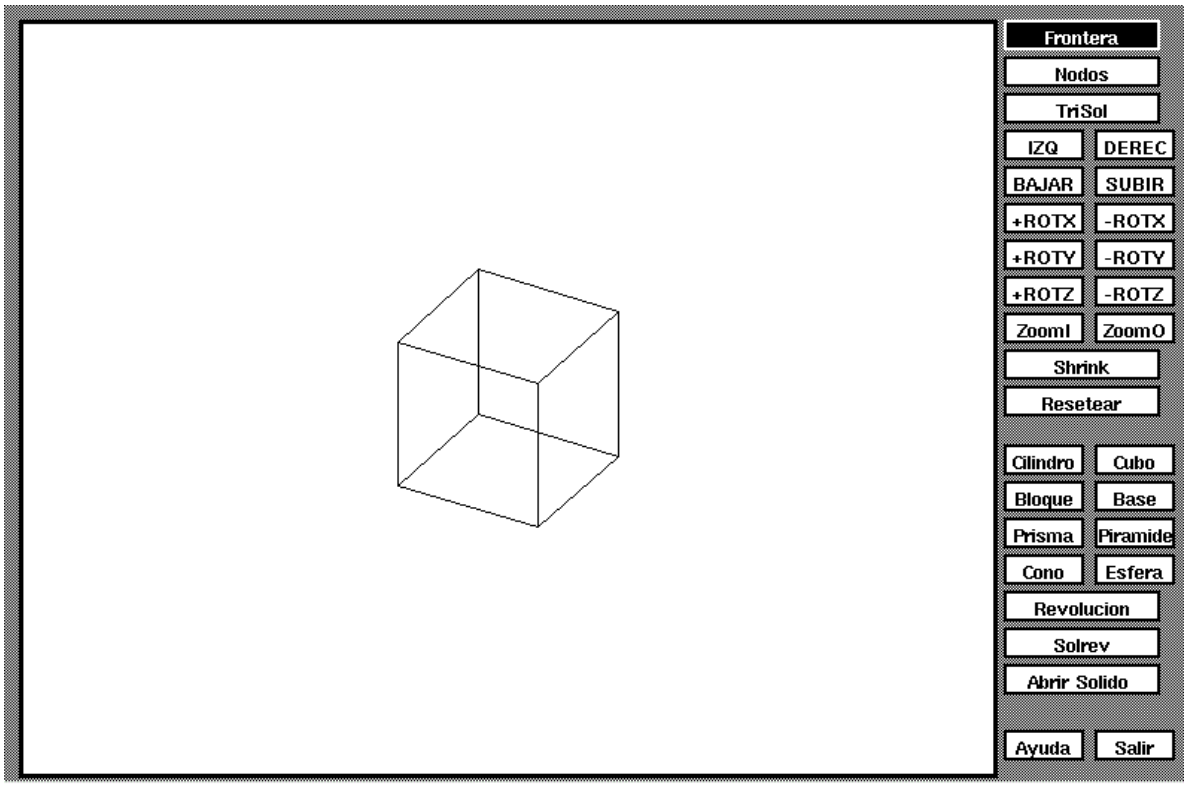
Nombre de archivo: piramide1.res



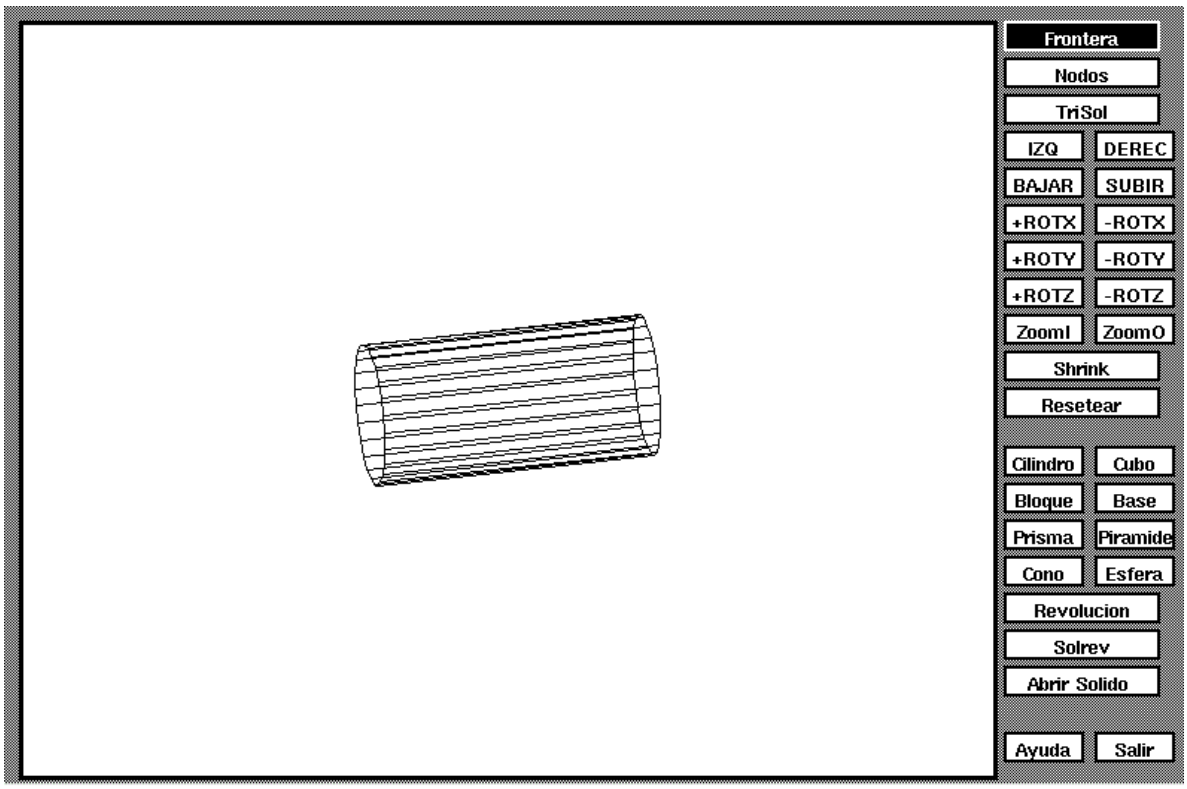
Nombre de archivo: prisma1.res



Nombre de archivo: ejemplo_bloque.res



Nombre de archivo: ejemplo_cubo.res



Nombre de archivo: ejemplo_cilindro.res

CONCLUSIONES

El programa SMS_EIM_UCV permite crear primitivas de sólidos tales como:

- Esferas: se le solicita al usuario el valor del radio de la esfera.
- Pirámides: se debe suministrar tanto el valor del lado de la base como la altura de la pirámide.
- Conos: se introducen el radio de la base y la altura del cono.
- Cilindros: se ingresan las dimensiones del cilindro tales como el radio de la base y la altura del mismo.
- Cubos: a través del valor del lado del cubo.
- Bloques: se deben ingresar las dimensiones del bloque tales como el ancho, largo, y altura.
- Prismas rectos: se genera un prisma recto regular al introducir el número de lados de la base y la altura del sólido.
- Sólidos por extrusión: a partir de una base irregular, se realiza una extrusión de altura indicada por el usuario.
- Sólidos de revolución: a partir de una base irregular o un polígono abierto, se pueden generar sólidos de revolución macizos o huecos.

Es posible guardar los datos geométricos del sólido con la finalidad de ver su representación gráfica posteriormente. El programa también permite la lectura de datos de un conjunto de puntos que representen una base irregular o un polígono abierto, a los cuales posteriormente se les puede aplicar operaciones de barrido generalizadas; Dicha lectura se realiza desde un archivo de texto, o se pueden ingresar los vértices directamente por pantalla.

La interfaz gráfica diseñada permite una interacción amigable entre el usuario y el programa. Una vez que se ejecuta el comando del programa, aparece la ventana principal del mismo, la cual contiene:

- Una ventana gráfica, en donde se plasma la representación gráfica de un sólido.
- Botones para crear tipos de sólidos nuevos.
- Botón para obtener la representación gráfica de un sólido existente.
- Botones para manipular sólidos.
- Botón de ayuda (para obtener explicación de cada uno de los botones)
- Botón de salida.

A través del botón “Abrir Sólido”, no solamente se encontrará la lista de sólidos creados por el usuario, sino también los sólidos de ejemplo creados en el presente trabajo especial de grado, todos con aplicación en la Ingeniería Mecánica, como por ejemplo: válvulas, pistones, bielas, canal abierto, tubería, caucho, ala, placa plana, presa concreto.

Los botones para manipular sólidos permiten:

- Rotar un sólido alrededor de cualquiera de sus ejes: esto resulta útil ya que se pueden observar distintos ángulos del sólido, lo que facilita su comprensión y estudio.
- Realizarle un acercamiento o alejamiento: útil cuando se desea agrandar un sólido muy pequeño, o disminuir el tamaño de un sólido que sobresalga de la pantalla gráfica por sus dimensiones.
- Desplazar el sólido respecto de su posición original en la pantalla.

La arquitectura del programa, cuya estructura de datos (semiarista) es la base fundamental, permite obtener dos tipos de representaciones:

- Representación Gráfica: permite ver en pantalla gráfica el sólido creado.
- Representación Matemática: Esta representación no es visible al usuario, pero permite obtener la información necesaria (coordenadas e identificación de los vértices, ecuaciones de las caras) para calcular cualquier propiedad geométrica (volumen y área superficial del sólido, masa, centro de gravedad…) en trabajos posteriores.

Se implementaron algoritmos de mediano nivel para el desarrollo de funciones que permiten generar polígonos (Funciones: base cuadrada, base rectangular, base plana irregular, polígono regular, caras triangulares, arco). A través de los operadores de Euler y el desarrollo de procedimientos geométricos, se implementaron algoritmos que permiten realizar procedimientos preestablecidos tales como el barrido traslacional, y el barrido rotacional (para polígonos abiertos y cerrados), lo que permitió crear distintas primitivas geométricas. Se desarrollaron algoritmos que permitieron resolver el problema que se genera al crear sólidos que contengan múltiples aristas que parten de un mismo vértice, lo que condujo a la implementación de primitivas para la creación de pirámides y conos.

La implementación del Sistema de Modelados de Sólidos permitió verificar los postulados realizados al inicio de este trabajo, referidos a las características deseables de un SMS, tales como:

- Completitud: el modelo es completo, puesto que posee información tanto geométrica como topológica de un sólido.
- Ambigüedad: el modelo no es ambiguo, ya que permite representar un único sólido a la vez.
- Validez: se pudo verificar la validez del modelo ya que este permite definir los límites de un objeto sólido en términos de superficies cerradas y orientables.
- Integridad: el criterio de integridad geométrica se mantiene ya que las caras de un modelo no se intersectan entre sí, excepto en aristas o vértices comunes, con lo cual no se permite la generación de modelos incorrectos.

Las ventajas de haber realizado un programa ejecutable bajo ambiente Linux son las siguientes:

- No existe costo alguno para descargar el sistema operativo Linux a través de la red de Internet, ya que el mismo fue desarrollado bajo la filosofía de software libre.
- Posibilidad de intercambiar información o desarrollos en el ámbito del modelado de sólidos con otros usuarios de Internet en el mundo.

El Programa SMS_EIM_UCV, orientado a la generación de la geometría de la pieza, constituye entonces un aporte para la creación posterior de una plataforma computacional que permita integrar los distintos trabajos realizados en la Escuela de Ingeniería Mecánica con el objetivo fundamental de resolver problemas de diseño de forma automatizada.

RECOMENDACIONES

El Programa SMS_EIM_UCV constituye la base para una serie de modificaciones posteriores que permitan incorporar diversas herramientas con el objetivo de crear un Sistema de Modelado de Sólidos completo y versátil.

Entre las sugerencias que puedan hacerse para mejorar el Sistema de Modelado de Sólidos desarrollado, están:

- Implementar algoritmos de alto nivel para crear funciones que permitan realizar operaciones booleanas (unión, intersección, y sustracción entre sólidos sencillos). Esto constituye un paso fundamental en el mejoramiento del programa puesto que permitiría, por ejemplo, realizarle agujeros a una pieza diseñada, incrustar una pieza dentro de otra (caso pistón-pasador), y conectar piezas entre sí (por ejemplo: caso biela-pasador-pistón).

- Incorporar algoritmos desarrollados previamente en la Escuela de Ingeniería Mecánica para añadir la función que permita intersectar un sólido con un plano. Esto resulta de gran utilidad, por ejemplo, para realizar cortes en piezas simétricas, lo que constituye una herramienta importante para futuras incorporaciones de funciones que generen planimetrías de piezas automáticamente.

- Incorporar diversas técnicas de modelado de sólido como: eliminación de aristas ocultas (para eliminar entonces las superficies ocultas del sólido), efectos de iluminación, distintos sistemas de proyección (proyección axonométrica, oblicua…), visualización de los planos coordenados, funciones que permitan incorporar color a cada una de las caras del sólido con el fin de crear la capa externa visible de una pieza.

- Implementar funciones gráficas que permitan hacer uso del ratón de la computadora directamente sobre la pantalla gráfica, para ingresar los datos de la geometría del sólido y manipular un sólido creado haciendo clic sobre la pieza directamente.

BIBLIOGRAFÍA

- Agoston, M. (2005). *Computer Graphics and Geometric Modeling*. Londres: Springer.

- Deitel, H.M. y Deitel, P.J. (1995). *Como programar en C/C++*. México: Prentice Hall.

- Kernighan, W. y Ritchie, M. (1991). *El Lenguaje de programación C.* México: Pearson Educación.

- Farina, V. (1985). *Diagramas de flujo.* México: Editorial Diana.

REFERENCIAS BIBLIOGRÁFICAS

[1]- Mäntylä, M. (1987). *An introduction to solid modeling,* (pp. 3-216). Estados Unidos: Computer Science Press.

[2]- Uzcátegui, C. y Uzcátegui J. (1993). *Implementación de técnicas de modelado de sólidos con aplicación a la Ingeniería Mecánica.* Tesis de grado, Universidad Central de Venezuela, Caracas.

[3]- Marcheix, D. (1996). *Modélisation géometrique des objets non-variétés : représentation, construction et déformation.* Tesis de doctorado, Université Bordeaux I, Francia.

[4]- Alonzo, A. y Oramas, M. (1991). *Elaboración e implementación de un sistema de representación gráfica de entidades geométricas bi y tridimensionales en computadores personales con aplicación al diseño mecánico y a la bioingeniería*. Tesis de grado, Universidad Central de Venezuela, Caracas.

[5]- Ordaz, E. y Pulido, O. (1992). *Resolución de problemas geométricos asociados a la generación automática de mallas tridimensionales de elementos finitos en piezas de forma libre.* Tesis de grado, Universidad Central de Venezuela, Caracas.

[6]- Jaén, H. y López, J. (1990). *Desarrollo de modelos computacionales de representación gráfica para problemas de elasticidad y aplicación al análisis de esfuerzos en órganos dentarios mediante el método de elementos finitos.* Tesis de grado, Universidad Central de Venezuela, Caracas.

[7]- Martín H. Miguel A (2008). *Desarrollo e implementación de un programa de visualización de resultados para programas de simulación por elementos finitos y de contorno.* Tesis de grado, Universidad Central de Venezuela, Caracas.

ANEXO A

IMPLEMENTACIÓN DE LOS OPERADORES DE EULER

A continuación se presentan los algoritmos necesarios para la implementación de los operadores de Euler de bajo nivel.

Nota: Las observaciones realizadas en los algoritmos que siguen son encerradas por los símbolos /* ... */

MVFS

Crea un sólido que consiste en una cara simple y un vértice. La cara es formada por un lazo vacío que contiene el único vértice, sin ninguna arista.

```
Solid *OP_EU_mvfs( int sn, int fn, int vn, Doble x,Doble y,Doble z)

{
   Solid    *news = SNULL;
   Face     *newf = FNULL;
   Vertex   *newv = VNULL;
   Loop     *newl = LNULL;
   HalfEdge *newh = HNULL;

   news = CreaSolid();
/*creación del un nuevo nodo sólido*/
   newv = CreaVert( news );
/*creación del nuevo nodo vértice del sólido*/
   BuildVert( newv, vn, x, y, z );
/*construcción de un nuevo vértice*/

   newf = CreaFace( news );
/*construcción de una nueva cara perteneciente al sólido*/
   newl = CreaLoop( newf );
/* construcción de un nuevo lazo perteneciente a la cara */
   newh = CreaHalfEdge();
 /*creación de semiarista*/
```

```
      news->snum  = sn;
   /* asignación de un número identificador del sólido*/

      newf->fnum  = fn;
   /*asignación de un número identificador de la cara*/
      newf->flout = newl;
   /*asignación de un número identificador del lazo externo de la cara */

      newl->lhedg = newh;

      newh->wloop = newl;
/*asignación de un puntero hacia el nuevo lazo que crea la pertenencia de la
semiarista newh al lazo newl */
      newh->nxt   = newh->prv = newh;
/*inicialización de las relaciones de precedencia entre las semiaristas */
      newh->vtx   = newv;
   /* vértice inicial de donde parte la semiarista*/
      newh->edg   = ENULL;
/*relación de pertenencia de la nueva semiarista a un puntero de arista nula*/
      return( news );
   /*regresa un puntero a un nuevo sólido ya inicializado*/
   }
```

MEV

Crea un nuevo vértice y una nueva arista, esta última conecta la arista anterior al nuevo vértice.

```
   void OP_EU_lmev( HalfEdge *he1, HalfEdge *he2, int vn, Vector vco )
   {
      HalfEdge *he   = HNULL;
      Vertex   *newv = VNULL;
      Edge     *newe = ENULL;
      Solid    *s    = he1->wloop->lface->fsolid;
   /*se crea la pertencia de la semiarista he1 al lazo wloop que a su vez
pertenece a la cara lface que a su vez pertenece al sólido fsolid */

      newe = CreaEdge( s );
```

```
      /*se crea arista nueva que pertenezca al sólido s*/
         newv = CreaVert( s );
   /*se crea vértice nuevo que pertenezca al sólido s*/
         BuildVert( newv, vn, vco[X], vco[Y], vco[Z] );
   /*asignación de coordenadas al nuevo vértice creado, y cuyo identificador
será vn*/

         he = he1;
         while( he != he2 )
   /* Divide el ciclo de semiaristas de he1->vtx en dos ciclos de manera tal
que he1 pasa a ser la primera semiarista en un ciclo y h2 también pasa a ser
la primera semiarista en el otro ciclo  */
         {
            he->vtx = newv;
/*nuevo vértice newv pasa a ser el vértice inicial de la semiarista he->vtx */
            he = OtraMitad(he)->nxt;
/*he pasa a ser la semiarista siguiente del lazo, pero en sentido contrario*/
         }

         /* Crea una nueva semiarista con vértice he2->vtx */
         addhe( newe, he2->vtx, he1, MINUS, "lmev" );
   /* he1->prv = nuevah;  nuevah->nxt = he1;         */

          /* Crea una nueva semiarista con vértice newv     */
         addhe( newe, newv    , he2, PLUS, "lmev" );
   /* he2->prv = nuevah;  nuevah->nxt = he2;*/
         newv->vedge     = he2->prv;
         he2->vtx->vedge = he2;
   }
```

MEF

Une el último vértice al primero cerrando una cara.

```
   Face *OP_EU_lmef( HalfEdge *he1, HalfEdge *he2, int nf )
   {
      Face     *newf;
```

```
   /*apuntador a una nueva cara*/
      Loop     *newl;
      /*apuntador a un nuevo lazo*/
      Edge     *newe;
   /*apuntador a una nueva arista*/
      HalfEdge *he, *nhe1, *nhe2, *temp;

      newf = CreaFace( he1->wloop->lface->fsolid );
/* creación de una nueva cara del sólido cuya semiarista inicial es he1*/

      newl = CreaLoop( newf );
   /* newl es agregado en newf->flint */
      newe = CreaEdge( he1->wloop->lface->fsolid );
/*creación de una nueva arista de la nueva cara y cuya una de sus semiaristas
es he1*/

      newf->fnum  = nf;
   /*identificador de la nueva cara*/
      newf->flout = newl;  /* newf->flout = newloop */
   /* newl es duplicado en flout */

      he = he1;
      while( he != he2 )
      {
         he->wloop = newl;
   /* se hace pertenecer he1 al nuevo lazo*/
         he        = he->nxt;
      }

      /*Se agrega a la estructura de datos del sólido las nuevas semiaristas
creadas las cuales pertenecen a la nueva cara y definen la nueva arista; a
cada semiarista se le asigna un sentido  */
      nhe2 = addhe( newe, he1->vtx, he2, PLUS , "lmef" );
      nhe1 = addhe( newe, he2->vtx, he1, MINUS, "lmef" );


      nhe1->prv->nxt = nhe2;
     /* reorganización de las asignaciones de precedencia de semiaristas*/
      nhe2->prv->nxt = nhe1;
```

```
   temp = nhe1->prv;  nhe1->prv = nhe2->prv;  nhe2->prv = temp;

   newl->lhedg       = nhe1;
   he2->wloop->lhedg = nhe2;
 return( newf );
/* apuntador de regreso a una nueva cara  */
}
```

KEV

Elimina un vértice y una arista.

```
void OP_EU_lkev( HalfEdge *he1, HalfEdge *he2 )

{
   HalfEdge *he;
   Edge     *killedge;
   Vertex   *killvert;
   Solid    *s;

   s = he1->wloop->lface->fsolid;
   killedge = he1->edg;
   killvert = he1->vtx;

   he = he2->nxt;
   while( he != he1 )
   {
      he->vtx = he2->vtx;
      he = OtraMitad(he)->nxt;
   }

   he1->wloop->lhedg = delhe( he1 );
   he2->wloop->lhedg = delhe( he2 );

   he2->vtx->vedge = he2->nxt;
   if ( he2->vtx->vedge->edg == ENULL )  he2->vtx->vedge = HNULL;

   BorraEdge( killedge, s );
```

```
/*función que elimina la arista deseada*/
   BorraVert( killvert, s );
/*función que elimina al vértice deseado*/

}
```

KEF

Elimina una arista y la cara a la cual pertenece dicha arista.

```
void OP_EU_lkef( HalfEdge *he1, HalfEdge *he2 )
{
   Face     *f1=FNULL, *f2=FNULL;
   Vertex   *v1=VNULL, *v2=VNULL;
   Loop     *l=LNULL, *lazo1=LNULL, *lazo2=LNULL;
   HalfEdge *he = HNULL;
   Solid    *s  = he1->wloop->lface->fsolid;

   f1    = he1->wloop->lface;
   f2    = he2->wloop->lface;
/* f2 es la cara a eliminar */
   lazo1 = he1->wloop;
   lazo2 = he2->wloop;
   v1    = he1->vtx;
   v2    = he2->vtx;

   while( (l = f2->flint) != LNULL )
/* (l = f2->floops) */
/* Pasa a f1 los lazos de f2 */
   {
      DelListLoop( l, f2 );
      AddListLoop( l, f1 );
   }
   he = lazo2->lhedg;


/* Los componentes del lazo2, */
```

```
   do
/* ahora pertenecen al lazo1  */
   {
      he->wloop = lazo1;
   }
   while( (he = he->nxt) != lazo2->lhedg );

/* Desconecta he1 y he2 */

   v2->vedge = he1->nxt;
   if ( v2->vedge->edg == ENULL ) v2->vedge = HNULL;

   v1->vedge = he2->nxt;
   if ( v1->vedge->edg == ENULL ) v1->vedge = HNULL;
   /* Inicia he del lazo1 */
   lazo1->lhedg = he1->nxt;
   BorraLoop( lazo2, f1 );
   BorraFace( f2, s );
   BorraEdge( he2->edg, s );
 }
```

Anexo B

ALGORÍTMOS PARA CREAR PRIMITIVAS

Algoritmos de mediano y alto nivel para la implementación de procedimientos para la creación de primitivas.

Nota: las observaciones realizadas van enmarcadas dentro de los símbolos /*...*/ .

Función arco:

Permite crear una cara circular ubicada en el plano *zplano*, de radio *rad*, y cuyas coordenadas del centro son *cx* y *cy*.

```
   void arc( Solid *s, int fac, int vert, Doble cx, Doble cy, Doble
rad, Doble zplane,Doble fi1, Doble fi2, int nsteps )
     /*nstep= número de lados de la base poligonal que aproxima al círculo*/
   {
      Doble x, y, angle, inc;
      Id    prev;
      int   i;

      angle = Deg2Rad( fi1 );
   /*ángulo inicial*/
      inc   = Deg2Rad( fi2 - fi1 ) / nsteps;
   /* incremento del ángulo*/

      prev = vert;
      getmaxnames( s );
   /*se busca el vértice semilla (v1 en este caso) */

      for ( i = 0; i < nsteps; i++ )
   /*ciclo que crea los n puntos de la base poligonal */
      {
         angle += inc;
         x = cx + cos( (Doble)angle ) * rad;
```

```
            y = cy + sin( (Doble)angle ) * rad;
            maxv++;
            OP_EU_mev( s, fac, prev, maxv, x, y, zplane );
            prev = maxv;
         }
      }
```

Función que genera un cubo:

```
   Solid *cubo( Vector cent, Doble lado )

   /* cent-> vector centro de la base cuadrada antes de aplicarle la operación de extrusión; lado-> lado del cubo */
   {
      Solid *s = SNULL;
      maxs++;
      s = OP_EU_mvfs( maxs, 1, 1, cent[X]-lado/2, cent[Y]-lado/2, cent[Z]-lado/2 );
   /*v1 se crea y se hace pertenecer a la nueva cara del nuevo sólido que está por contruirse*/
      OP_EU_mev( s, 1, 1, 2, cent[X]+lado/2, cent[Y]-lado/2, cent[Z]-lado/2 );
    /*se crea v2 y la arista que lo una al vértice 1*/
      OP_EU_mev( s, 1, 2, 3, cent[X]+lado/2, cent[Y]+lado/2, cent[Z]-lado/2 );
    /*se crea v3 y la arista que lo una al vértice 2*/
      OP_EU_mev( s, 1, 3, 4, cent[X]-lado/2, cent[Y]+lado/2, cent[Z]-lado/2 );
   /*se crea v4 y la arista que lo una al vértice 3*/
      OP_EU_mef( s, 1, 4, 1, 2 );
   /*se unen los vértices v4 y v1 para así crear la cara base*/
      sweep( fface(s, 2), 0.0, 0.0, lado );
   /*se aplica el barrido traslacional para crear el cubo por extrusión de su base*/
      return( s );
   /*regresa puntero a un sólido*/
   }
```

Función que genera un bloque:

```
Solid *block( Vector cent, Vector lado )
/* cent-> vector centro de la base rectangular antes de aplicarle la operación de extrusión; lado-> vector que contiene los datos: ancho,largo, altura del bloque*/
{
   Solid *s = (Solid *)NULL;

   s = OP_EU_mvfs( ++maxs, 1, 1, cent[X]-lado[X]/2, cent[Y]-lado[Y]/2, cent[Z]- lado[Z]/2 );
/*v1 se crea y se hace pertenecer a la nueva cara del nuevo sólido que está por construirse*/
   OP_EU_mev( s, 1, 1, 2, cent[X]+lado[X]/2, cent[Y]-lado[Y]/2, cent[Z]-lado[Z]/2 );
/*se crea v2 y la arista que lo una al vértice 1*/
   OP_EU_mev( s, 1, 2, 3, cent[X]+lado[X]/2, cent[Y]+lado[Y]/2, cent[Z]-lado[Z]/2 );
/*se crea v3 y la arista que lo una al vértice 2*/
   OP_EU_mev( s, 1, 3, 4, cent[X]-lado[X]/2, cent[Y]+lado[Y]/2, cent[Z]-lado[Z]/2 );
/*se crea v4 y la arista que lo una al vértice 3*/
   OP_EU_mef( s, 1, 4, 1, 2 );
/*se unen los vértices v4 y v1 para así crear la cara base*/
   sweep( fface(s, 2), 0.0, 0.0, lado[Z] );
 /*se aplica el barrido traslacional para crear el bloque por extrusión de su base*/
   return( s ); /*regresa puntero a un sólido*/
}
```

Función que genera la extrusión de una base plana irregular:

```
Solid *Extrusion( Doble zplano, Doble altura ,Vector Xp[], int nverts)
/*base irregular de nverts vértices ubicada en el plano XY */
{
   Id      i, vstart, prev, fstart;
```

```
      Solid  *s = NULL;
      int         maxs=1;   maxf=1;   maxv   =   1;vstart=1;   prev=1;
maxv=1;fstart=1; maxf=1;

     s = OP_EU_mvfs( maxs, maxf, maxv, Xp[0][X], Xp[0][Y], zplano );
    /*se crea un sólido y cara nueva; esta última contiene el vértice 1 */

      for ( i =1 ; i < nverts ; i++ )
   /*ciclo  para  hacer  pertenecer  los  vertices  v2,v3  ...  vn  al  sólido  en
construcción.*/
      {
         maxv++;
         OP_EU_mev( s, maxs, prev, maxv, Xp[i][X], Xp[i][Y], zplano );
         prev = maxv;

   }
      OP_EU_mef( s, 1, maxv, 1, 2 );
   /*se cierra la base poligonal, creando la cara base*/
      sweep ( fface(s, 2), 0.0, 0.0, altura );
   /*se aplica la extrusión a la base definiendo el sólido completamente*/
      return( s );
   /*regresa apuntador a un sólido*/
   }
```

Función caras triangulares

```
   void caras_triangulares ( Face *fac, Doble dx, Doble dy, Doble dz,
int nver )
   {
      Loop     *l;
      HalfEdge *first, *scan;
      Vertex   *v;
      Vector    vtmp, desp = { dx, dy, dz, 1 };
   /* vector desp define el vértice principal del triangulo*/
      int i;
      getmaxnames( fac->fsolid );
```

```
   for ( l = fac->flint; l != NULL; l = l->nextl )
   {
      first = l->lhedg;
      scan  = first->nxt;
      v     = scan->vtx;

      VecPlus( vtmp, v->vcord, desp );
      OP_EU_lmev( scan, scan, ++maxv, vtmp );
/*creación de un lado de  un triángulo*/

     for ( i = 2; i < nver; i++ )
   {

         OP_EU_lmef( scan->prv, scan->nxt, ++maxf );
/* se termina de crear la cara triangular */
         scan = OtraMitad(scan->nxt)->nxt;

   }
      OP_EU_lmef( scan->prv, scan->nxt, ++maxf );
 }
   }
}
```

Función de barrido traslacional:

```
void sweep ( Face *fac, Doble dx, Doble dy, Doble dz )
/*fac-> es la cara a la cual se le aplicará el barrido
 dx dy dz -> coordenadas del vector de traslación que se llama desp*/
{
   Loop     *l;
   HalfEdge *first, *scan;
   Vertex   *v;
   Vector    vtmp, desp = { dx, dy, dz, 1 };
   getmaxnames( fac->fsolid );
   for ( l = fac->flint; l != NULL; l = l->nextl )
   {
```

```
      first = l->lhedg;
      scan  = first->nxt;
      v     = scan->vtx;
/*se inicia traslación del vértice 1 de la cara*/

      VecPlus( vtmp, v->vcord, desp );
/* vtmp -> vector resultante de la traslación de v1  */
      OP_EU_lmev( scan, scan, ++maxv, vtmp );
 /*se une el nuevo punto trasladado al vértice v1*/

      while( scan != first )
/*en este ciclo se generan las demás traslaciones de todos los puntos de la
base*/
      {
         v = scan->nxt->vtx;
         VecPlus( vtmp, v->vcord, desp );
         OP_EU_lmev( scan->nxt, scan->nxt, ++maxv, vtmp );
         OP_EU_lmef( scan->prv, scan->nxt->nxt, ++maxf );
         scan = OtraMitad(scan->nxt)->nxt;
      }
      OP_EU_lmef( scan->prv, scan->nxt->nxt, ++maxf );
/*se cierra la última cara lateral trasladada*/
   }
}
```

Función que genera el barrido rotacional

Permite el barrido rotacional de una figura plana abierta ubicada en el plano XY con y > 0, la cual se hace girar alrededor del eje x. Se genera un sólido de revolución macizo.

```
  void sweepR( Solid *s, int npasos )
/*npasos-> subdivisiones espaciales */
{
   Matrix    m;
   Vector    v;
   HalfEdge *first, *cfirst, *last, *scan = HNULL;
   Face     *tempf, *tailf = FNULL;
```

```
      getmaxnames(s);
   /* se iniciará la rotación del máximo vértice vn*/

      first = s->sfaces->flint->lhedg;
   /* semiarista de recorrido de la figura plana abierta*/
      while( first->edg != first->nxt->edg )
 /*Se busca que first sea la semiarista que parta de vn-1 y termine en vn */
         first = first->nxt;

      last = first->nxt;
      while( last->edg != last->nxt->edg )
         last = last->nxt;

      MatIdent( m );   MatRotat( m, (360.0/npasos), 0.0, 0.0 );
    /* se definen parámetros para la matriz de rotación alrededor del eje x*/

      cfirst = first;
      while( --npasos )
   /*ciclo que generará los vértices homólogos producto de la rotación de vn,
vn-1, ...,v1 */
      {
         VecMult( v, cfirst->nxt->vtx->vcord, m );
   /* se obtiene por rotación vn' en la primera iteración*/
         OP_EU_lmev( cfirst->nxt, cfirst->nxt, ++maxv, v );
   /*se unen mediante una arista a vn' y vn*/

         scan = cfirst->nxt;
         while( scan != last->nxt )
   /*scan recorre cada vértice vn-1,vn-2,..., v1 para crear por rotación sus
homólogos vn'-1,vn'-2,...,v1'*/
         {
            VecMult( v, scan->prv->vtx->vcord, m );
            OP_EU_lmev( scan->prv, scan->prv, ++maxv, v );
            tempf = OP_EU_lmef( scan->prv->prv, scan->nxt, ++maxf );
/*se van creando las caras que se vayan generando; por ejemplo en la primera
iteración la cara formada será vn-1, vn, vn', vn'-1 */

            scan = OtraMitad(scan->nxt->nxt);
         }
```

```
         last = scan;
         cfirst = OtraMitad(cfirst->nxt->nxt);
   /* con el nuevo arco de aristas v1', ..., vn' se reiniciará el proceso para
hallar por rotación v1'',...,vn'' y así hasta generar el sólido*/
      }

      tailf = OP_EU_lmef( cfirst->nxt, OtraMitad(first), ++maxf );

      while( cfirst != scan )
      {
         tempf = OP_EU_lmef( cfirst, cfirst->nxt->nxt->nxt, ++maxf );
/*se término de cerrar el sólido*/

         cfirst = OtraMitad(cfirst->prv)->prv;
      }
      Fac2Vert( tailf );
   }
```

Barrido rotacional de polígonos:

Permite generar sólidos de revolución huecos. (por ejemplos tuberías, anillos)

```
   void rsweep( Solid *s, int npasos )
   /*npasos-> subdivión espacial del barrido*/
   {
      HalfEdge *h, *first, *cfirst, *last, *scan=HNULL;
      Matrix    m;
      Vector    v;
      Face     *tailf, *headf = FNULL;
      int       closed_figure;

      headf = s->sfaces;
   /* headf-> cara que se abrirá en un punto arbitrario para duplicar un
vértice */
      if ( s->sfaces->nextf != (Face *)NULL )
      {
         closed_figure = 1;
     /* Se asume que la figura cerrada es una lámina  */
```

```
            h = s->sfaces->flint->lhedg;
            OP_EU_lmev( h, OtraMitad(h)->nxt, ++maxv, h->vtx->vcord );
            OP_EU_lkef( h->prv, OtraMitad(h->prv) );
      /*se elimina una arista y una cara*/
            headf = s->sfaces;
      /*La cara headf ya es un polígono abierto*/

         }
         else
            closed_figure = 0;
/*De aquí en adelante se repiten los pasos del barrido rotacional ya explicado
*/
         getmaxnames( s );

         first = s->sfaces->flint->lhedg;
         while( first->edg != first->nxt->edg )
            first = first->nxt;

         last = first->nxt;
         while( last->edg != last->nxt->edg )
            last = last->nxt;

         cfirst = first;
         MatIdent( m );
         MatRotat( m, (360.0/npasos), 0.0, 0.0 );

         while( --npasos )
         {
            VecMult( v, cfirst->nxt->vtx->vcord, m );
            OP_EU_lmev( cfirst->nxt, cfirst->nxt, ++maxv, v );

            scan = cfirst->nxt;
            while( scan != last->nxt )
            {
               VecMult( v, scan->prv->vtx->vcord, m );

               OP_EU_lmev( scan->prv     , scan->prv, ++maxv, v );
               OP_EU_lmef( scan->prv->prv, scan->nxt, ++maxf );
```

```
            scan = OtraMitad( scan->nxt->nxt );
         }
         last = scan;
         cfirst = OtraMitad( cfirst->nxt->nxt );
      }

      tailf = OP_EU_lmef( cfirst->nxt, OtraMitad(first), ++maxf );

      while( cfirst != scan )
      {
         OP_EU_lmef( cfirst, cfirst->nxt->nxt->nxt, ++maxf );
         cfirst = OtraMitad(cfirst->prv)->prv;
      }
      /*finalmente se fusionan los dos caras circulares creadas por la
rotación de los dos puntos duplicados mediante la función pegar_lazo*/
      if ( closed_figure == 1 )
      {
         OP_EU_lkfmrh( headf, tailf );
         loopglue( headf );
      }
   }
```

Función pegar_lazo:

Esta función permite fusionar dos lazos de dos caras con el fin de crear una sola cara.

```
      void loopglue( Face *fac )
   {
      HalfEdge *h1, *h2, *h1next;


      h1 = fac->flint->lhedg;
   /* Inicializa lazo externo*/
      h2 = fac->flint->nextl->lhedg;
    /* inicializa lazo interno */
```

```
while( !match(h1, h2) )
/* Busca dos semiaristas coincidientes h2 y h1 */
      h2 = h2->nxt;

   OP_EU_lmekr( h1, h2 );
 /* crea arista con h1 y h2*/


   OP_EU_lkev( h1->prv, h2->prv );


   while( h1->nxt != h2 )
   {
      h1next = h1->nxt;
      OP_EU_lmef( h1->nxt, h1->prv, -1 );

      OP_EU_lkev( h1->nxt, OtraMitad(h1->nxt) );


      OP_EU_lkef( OtraMitad(h1), h1 );


      h1 = h1next;
   }
   OP_EU_lkef( OtraMitad(h1), h1 );


}
```

ANEXO C

AYUDA DEL PROGRAMA

Nota: A continuación se presenta el contenido completo del botón ayuda.

Programa SMS_EIM_UCV

DESCRIPCIÓN DEL PROGRAMA: Sistema de Modelado de Sólidos que permite la creación automatizada y la manipulación de modelos tridimensionales de piezas complejas mediante Interacción Gráfica.

AUTOR: Este programa ha sido elaborado por el bachiller Laurent Chevron como parte de su tesis de grado para optar el título de Ing.Mecánico de la ilustre Universidad Central de Venezuela, bajo la tutoría del prof. Antonio Barragán.

BOTONES PARA CREAR SÓLIDOS:

BOTÓN		FUNCIÓN
Cilindro	→	Ejecuta programa para generar cilindros
Cubo	→	Ejecuta programa para generar cubos
Bloque	→	Ejecuta programa para generar bloques de dimensiones ancho, largo, altura.
Base	→	Ejecuta programa para generar sólidos por extrusión a partir de una base irregular .
Prisma	→	Ejecuta programa para generar prismas rectos cuya base es un polígono regular
Pirámide	→	Ejecuta programa para pirámides rectas.
Cono	→	Ejecuta programa para generar conos.
Esfera	→	Ejecuta programa para generar esferas.
Revolución	→	Ejecuta programa para generar sólidos de revolución macizos
Solrev	→	Ejecuta programa para generar sólidos de revolución huecos
Abrir Sólido	→	Permite ver en pantalla un sólido creado anteriormente.

BOTONES PARA MANIPULAR SÓLIDOS:

Frontera → Permite quitar/poner aristas de un sólido en la pantalla

Nodos → Permite quitar/poner vértices de un sólido en la pantalla

IZQ → Permite desplazar el sólido hacia la izquierda de la pantalla

DEREC → Permite desplazar el sólido hacia la derecha de la pantalla

SUBIR → Permite desplazar el sólido hacia arriba de la pantalla

BAJAR → Permite desplazar el sólido hacia abajo de la pantalla

+ROTX → Permite girar el sólido alrededor del eje X (regla mano derecha)

+ROTY → Permite girar el sólido alrededor del eje Y (regla mano derecha)

+ROTZ → Permite girar el sólido alrededor del eje Z (regla mano derecha)

ZoomI → Permite realizar un acercamiento del sólido

ZoomO → Permite realizar un alejamiento del sólido

Resetear → Permite regresar un sólido a su tamaño original

Ayuda → Permite ver las funciones de cada botón

-ROTX → Permite girar el sólido alrededor del eje X (regla mano izquierda)

-ROTY → Permite girar el sólido alrededor del eje Y (regla mano izquierda)

-ROTZ → Permite girar el sólido alrededor del eje Z (regla mano izquierda)