211-16

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES
DE MONTERREY

CAMPUS ESTADO DE MEXICO

SISTEMA DE GENERACION AUTOMATICA DE MACROOPERACIONES DE MAQUINADO

TRABAJO DE INVESTIGACION

QUE PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS DE MANUFACTURA

ESPECIALIDAD EN SISTEMAS INDUSTRIALES

PRESENTA :

ING. GUILLERMO GRANADOS

SIENDO INTEGRADO EL JURADO POR
DR. ISAAC RUDOMIN
DR. HECTOR SALDAÑA

DIRECTOR DE TESIS :
DR. PEDRO L. GRASA

DICIEMBRE, 1992

Dedico este esfuerzo:

A mi madre, mi padre y Fernando.

A mis amigos.

A quienes la búsqueda por ser
no les impide ser.

Quiero agradecer por dedicar su tiempo en esta tesis al Dr. Isaac Rudomin y al Dr. Héctor Saldaña, sinodales de esta Tesis.

También a Miguel Salaís, Juan López y Pepe Pantoja por su orientación, regaños y lecciones que forjan a triunfadores

Un agradecimiento muy especial a mi gente, mi familia y mis amigos de siempre, que son míos.

Un agradecimiento muy cariñoso a la generación de Ingenieros en Electrónica y Comunicaciones de diciembre 1992, porque crecimos juntos y me enseñaron lo satisfactorio que es entregarseen lo que uno hace.

Al creador, orientador, informador, impulsador de esta Tesis, que no se dió por vencido le expreso mi agradecimiento y mi admiración, al Dr. Pedro Luis Grasa Soler.

RESUMEN:

La evolución contínua de la computación, ha hecho accesible el desarrollo de sistemas complejos de información y su procesamiento para la toma de decisiones, que en principio fue realizada por personas y es cada vez más común que sean los mismos sistemas computacionales los que realicen esta actividad.

En el área de manufactura, como en otras, se pretende crear un único sistema que integre la información que generan diferentes subsistemas como nóminas, planeación y control de inventarios, cuentas por cobrar, diseño, planeación de procesos, control numérico, etc Algunos esfuerzos se enfocan al seguimiento lógico de un producto, desde que es conceptualizado, hasta que es una realidad física buscando integrar los sistemas de diseño, planeación de procesos, control y prueba de manufactura y control de calidad.

La interfaz entre diseño y procesos es de gran complejidad, porque traslada la imagen de un diseño y sus características tecnológicas en información directa para manufactura. Aún no ha sido comprendido el razonamiento espacial del ser humano para trasladar la imagen y sus características técnicas en información directa para manufactura, "no sabemos como, pero la verdad es que lo hacemos", es una respuesta común al preguntarnos cómo entendemos una escena visual.

¿Cómo hace un experto en procesos de producción para asignar una secuencia de maquinado a un diseño? ¿Qué hace que su decisión sea "la mejor"? La respuesta a estas preguntas es el enfoque de los esfuerzos de personas que quieren ver entrelazados los sistemas de apoyo al diseño con los de apoyo a la planeación del proceso de manufactura. Al conseguir esta respuesta, daremos un gran paso para la integración de los sistemas de diseño y los de planeación de proceso: sistemas de diseño para manufactura.

Esta tesis propone un esquema para la modelación y síntesis geométrica de un sólido básico y le relaciona una secuencia ordenada de macrooperaciones, en base a reconocimiento de características primitivas - barrenos, ranuras, etc.- , para que junto con un análisis dimensional del diseño se obtengan posibles secuencias ordenadas de maquinado, y se realice la planeación de procesos.

INDICE.

§ 1. Introducción.

1.1 Sistemas de manufactura

1.1.1 Manufactura

En un principio, la vida se regía por principios fáciles de describir y de entender, las relaciones entre los pequeños grupos de personas eran sencillas, y aisladas. La naturaleza fue la única fuente de riqueza, mediante actividades productivas básicas como son la cacería, la pesca, la agricultura o la minería, que satisfacían las necesidades locales de cada comunidad. Al existir comunicación entre comunidades, se dan cuenta que no es posible encontrar todos los bienes en todos los lugares todo el tiempo, existe entonces una distinción en dos clases de bienes: los bienes gratuitos son los que se encuentan en cantidades ilimitadas, y no representa un costo el poseerlos, como son el aire o el agua de un río (si se está cerca de él, por supuesto); los bienes económicos son los que representa un costo el obtenerlo en el lugar y tiempo requeridos.

La labor de algunas personas se concentra en distribuir los bienes económicos mediante alguna retribución. Esta actividad comienza como trueque, evoluciona en comercio y más tarde en producción

Producción se acepta modernamente como "El aumento en la utilidad de los bienes, mediante la labor coordinada de los factores o agentes de producción" [11]. Para esta definición: utilidad es una medida de satisfacción deseada por el humano; factores o agentes de producción son los bienes -económicos o libres-, con los medios y el conocimiento adecuados; el aumento en la utilidad se refiere a un cambio en las características de los bienes, o en su temporalidad, posicionalidad o su interrelación con agentes económicos. Hitomi [16], extiende esta definición señalando que se puede tratar de bienes tangibles, o intangibles. (Fig. 1.1).

Cuando limitamos este concepto a bienes tangibles cuyo aumento en la utilidad está determinado por una transformación física o química, nos referimos a la definición de **manufactura.** Al bien tangible sin transformar se le denomina **materia prima,** y cuando ha sido transformado se le define como **producto terminado.**

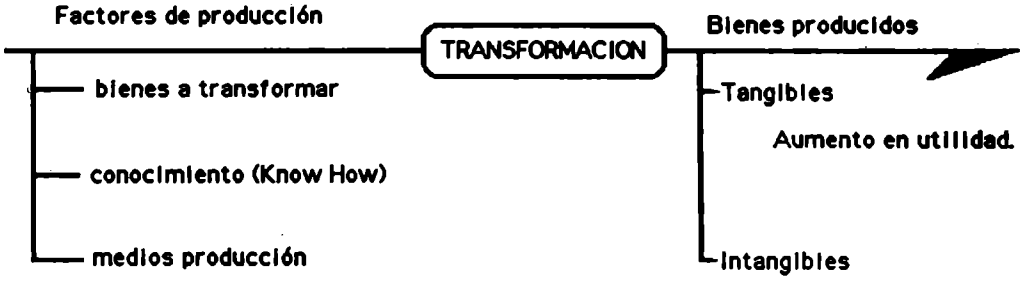


Fig. 1.1. Visualización de manufactura como proceso de entrada-salida

1.1.2. Sistemas.

Existe una gran variedad en las definiciones de sistema, estas coinciden con los siguientes atributos [16]:

a) Divisibilidad. Un sistema es la agrupación de elementos distinguibles, tangibles o intangibles. Se agrupan componentes, máquinas, personas, u otros sistemas.

b) Relación. Esta agrupación, debe de tener alguna relación o relaciones físicas o lógicas entre sí; entre estas, el motivo de su agrupación

c) Dirección. Este conjunto de elementos relacionados, para ser considerado como sistema, ha de cumplir una función, o persigue un objetivo (simple o múltiple).

d) Pertenencia a un medio ambiente. No podemos pensar en un conjunto de elementos aislado del universo -¿De dónde se obtuvo ese conjunto?-. Un sistema pertenece a otro conjunto de elementos llamado medio ambiente, al que afecta, y por el cual es afectado.

Al combinar estos conceptos, llegamos a algunas observaciones:

- Un sistema siempre es parte de otro sistema. Al sistema contenido se le llama subsistema, y al contenedor suprasistema.

- Teóricamente, debemos concluir la existencia de un Megasistema, o más estrictamente hablando, de "El Sistema" que contuviese a todos los demás, y no sea contenido por alguno. La discusión de la existencia o no existencia de este Sistema, carece de practicidad y no afecta al estudio de los sistemas.

- Un mismo conjunto de elementos físicos, puede ser agrupado en muchos sistemas, más aún, un sistema se puede dividir en subsistemas de tantas formas, como relaciones físicas, lógicas o conceptuales podamos definir en el.

- Los efectos que el medio ambiente tiene en un sistema son llamados entradas al sistema, y los efectos que tiene un sistema en el medio ambiente, son llamados salidas del sistema. (Fig. 1.2)

- El estudio de los sistemas resalta el hecho que un sistema persiga su objetivo de una forma óptima. Si los sistemas carecieran de dirección, no tendría sentido su estudio; para que esto sea posible, debe de existir una forma de medir la proximidad lograda, es decir, el objetivo debe ser mesurable.

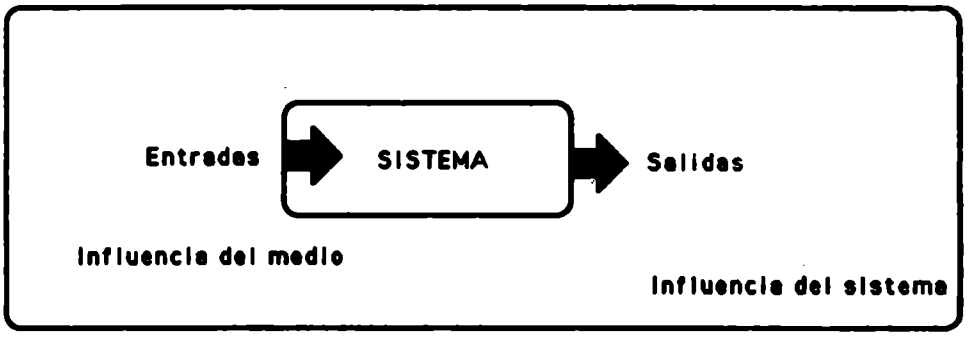


Fig. 1.2. Interacción de un sistema con su medio ambiente

- Un sistema que es capaz de realizar cambios en sí mismo para permanecer óptimo pese a los cambios en su medio ambiente, es llamado **sistema adaptativo, o cibernético.**

1.1.3. Sistemas de manufactura.

Nos referiremos como centro de manufactura, a todo lugar donde se realice una actividad de manufactura, sin importar tamaño, capacidad o lugar físico. Resulta claro que un centro de manufactura, es un sistema, donde las entradas son los factores de producción, y las salidas son los bienes producidos. (Fig. 1.1). El objetivo de los sistemas de manufactura debe ser lograr la máxima satisfacción conjunta de las necesidades del cliente, de la sociedad, y de sus socios. Las personas dedicadas al estudio de estos sistemas, deben de visualizar, desarrollar y mejorar subsistemas de manufactura para que este objetivo se logre en cada centro de manufactura.

Se han creado infinidad de subsistemas y se seguirán creando más, para facilitar el análisis -y posteriormente, la síntesis- de

estos subsistemas. Podemos clasificar las divisiones como sigue [11]:

- **Aspecto social.** Donde el principio de división es el factor humano, al realizar esta división, logramos una visión mas objetiva de la psicología del trabajo, las relaciones laborales, y el aspecto social de una empresa.

- **Aspecto transformacional.** Estos sistemas siguen el trayecto físico de la materia prima hasta que es convertida en producto terminado, a esto se le conoce como flujo de material. Es muy importante comprender esta división para facilitar el estudio de los tiempos y movimientos de los trabajadores y del material, o la estructura estática espacial (layout) del centro de manufactura.

- **Aspecto procedural.** Se busca que un sistema de manufactura sea cibernético, pero lograr esta adecuación al cambio no es tarea fácil, dada la incertidumbre que se tiene sobre la dirección del medio ambiente. Los esfuerzos enfocados a lograr esta adaptabilidad se pueden dividir en dos grupos: los encargados a la planeación estratégica de la producción, que diseñan nuestra respuesta al cambio; los encargados al control dada una planeación, esto es la administración de la producción que verifica que la respuesta sea la mejor y que se esté realizando como fué diseñada.

En la administración de la producción, es útil resaltar los procedimientos que se aplican a la materia prima para convertirla en producto, siguiendo su trayectoria lógica, desde que es conceptualizado, hasta que es una realidad física (Fig. 1.3). Esto es conocido como procedimiento de manufactura, y su eficiencia depende directamente del grado de sistemización que logremos

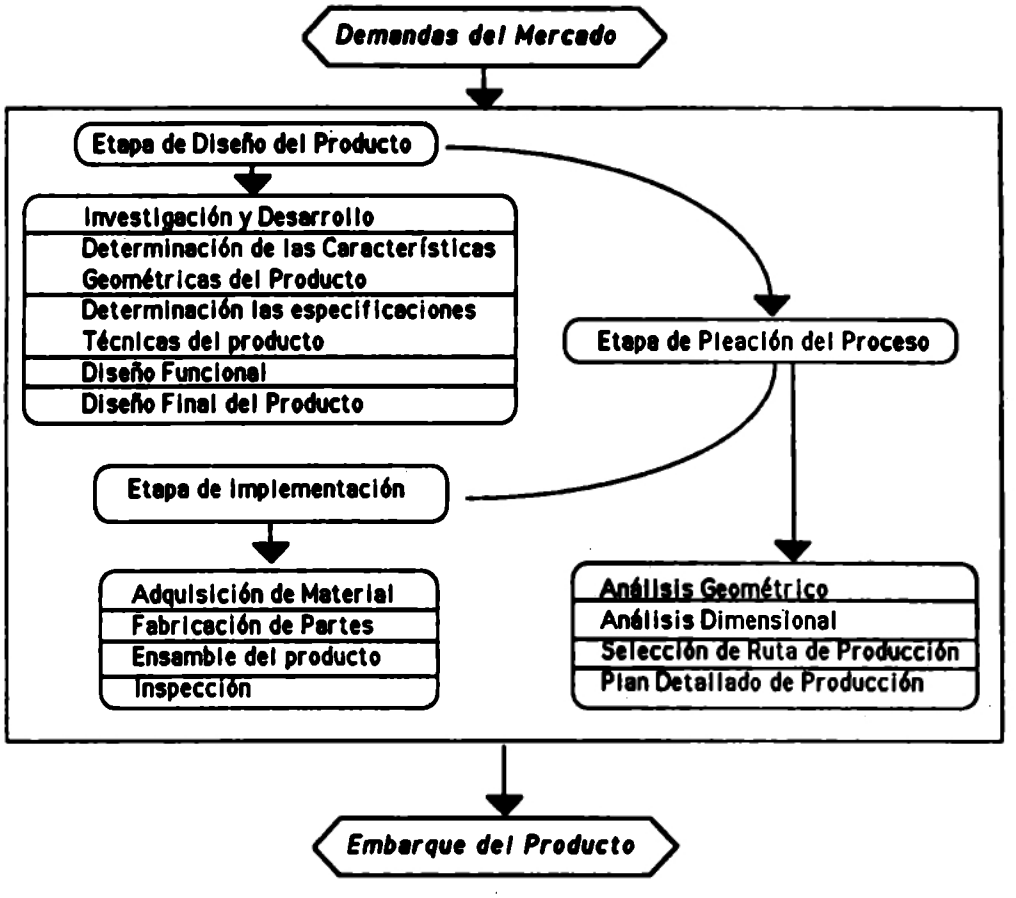


Figura 1.3. Análisis Procedural de un Sistema de Manufactura

entre sus partes.

Esta tesis está enfocada a la integración de los sistemas de diseño y los de planeación de procesos.

1.2. Sistemas de información computarizados para manufactura.

1.2.1. Sistemas de información.

Gaylord [11], compara un sistema de manufactura con una pequeña sociedad, donde personas de diversas ideologías, costumbres, creencias, conocimientos, y -quizá lo más grave- entendimiento diverso están reunidas con un objetivo común, pero en grupos con metas no siempre iguales, y en ocasiones encontradas, como en toda sociedad. Esto crea conflictos, y el éxito de una empresa está directamente relacionado con la rapidez y efectividad con que estos conflictos son resueltos, lo mejor será que no existan estos conflictos. Para esto es necesario pensar que las partes de la empresa se comuniquen de la manera más clara y veraz posible, al tiempo debido, para que puedan negociar sus objetivos locales con el fin de lograr de la mejor manera el objetivo global.

La comunicación es la transimisión de información entre un emisor y un receptor. La información son datos reunidos que tienen un significado como conjunto. Un sistema de información se define como: "Un ensamble de componentes sistemático y formal que realiza operaciones de procesamiento de datos para (a) Cubrir requerimientos legales y transaccionales, (b) proveer información a la gerencia para soportar actividades de planeación, control, y toma de decisiones, y (c) proveer una variedad de reportes, como es requerido para los constituyentes externos de la empresa." [4]. Al referirnos a un sistema de información, sabemos que esta tiene que ser clasificada en conjuntos menores, verificados, arreglados mediante modelos de procesamiento de datos.

La mesurabilidad del objetivo de un sistema de información, está determinada por el tiempo en que son presentados los

resultados, su confiabilidad, su consistencia, su alcance, su flexibilidad y ajuste a las necesidades, para esto es necesario diseñar correctamente el flujo de información, además de corregirlo para lograr un sistema cibernético.

1.2.2. La computación en los sistemas de información.

Las sociedades, como las empresas, para funcionar han desarrollado lenguajes, sistemas de comunicación, técnicas de transporte, y reglas de comportamiento propias, que influyen en su ventaja o desventaja frente a otras sociedades -competitividad- [11]. Resulta claro el pensar que algunas técnicas son más funcionales que otras, y que su funcionalidad esta sujeta a un espacio temporal. El Dr. José M. Sánchez [27], visualiza las estrategias globales de competitividad en manufactura de las últimas cuatro décadas, de la siguiente forma:

60's Economía de escala.
70's Costos.
80's Calidad.
90's Velocidad.

Lo cual es una respuesta obvia al vertiginoso crecimiento de la relación poder de computo/costo en los últimos 15 años.

Estamos en la era de la información, los productos son informativamente complejos, Sánchez cita que el tanque M1 de la Armada Americana, requiere de más de 40 000 páginas de documentación técnica para apoyar su fabricación, y cita a Naisbitt diciendo que "La nueva fuente de poder no es dinero en manos de pocos, sino información en manos de muchos" [27]. La competitividad en estos días está relacionada con la capacidad de manejar los complejos sistemas de información tan rápida e inteligentemente como sea posible.

El sueño de una empresa de manufactura, es lograr un manejo de información óptimo en toda la planta. Imaginemos por un momento, que vivimos en un mundo que cuenta con un centro de información global, donde toda la información estuviera reunida (no necesariamente en el mismo lugar físico) y nuestro nombre apareciera solamente una vez, con toda la información que cualquier persona, asociación, o país pudiera necesitar en cualquier momento. Esto suena irreal, pretencioso, y a algunas personas les parecerá molesto, pero detengámonos a pensar algunas de las ventajas:

- Sería fácil mantener este sistema siempre actualizado, porque al modificarse cualquier información, sabríamos inmediatamente a dónde acudir una única vez, por lo mismo, nunca existiría duplicidad de información, ni inconsistencia de ésta.

- Podríamos obtener información veraz rápidamente, sabríamos exactamente el alcance de la información para cada persona, no habría que estar recolectando información. Obviamente, debe de existir privilegios de acceso a infomación confidencial.

- Tendríamos una visión global de lo que sucediera en el orbe en cualquier momento, y sería fácil detectar elementos singulares - deseables y no deseables- para tomar acciones adecuadas.

- Tendríamos una gran asertividad al proyectar hacia el futuro, para tomar acciones de dirección.

En otras palabras, el sistema de información global, se traduciría en planeación, organización, dirección, y control - ADMINISTRACION- de la población.

Afortunadamente, más por política que por poder de cómputo, ese día está bastante lejano y por ahora en la mente de pesimistas autores de ciencia ficción, pero imaginemos un sistema de información así en una empresa...

El sueño se llama CIM (computer Integrated Manufacturing) y se traduce en la administración ideal de un centro de manufactura, información depurada, única, rápida: PODER sobre el cumplimiento del objetivo de un centro de manufactura.

INTEGRACION DE SISTEMAS
Manufactura Integrada por Computadora - CIM
SISTEMAS DE SOPORTE AL DISEÑO
Diseño Asistido por Computadora - CAD
Ingeniería Asistida por Computadora - CAM
CONTROL DE PROCESOS
Manufactura Asistida por Computadora - CAM
Planeación de Procesos Asistida por Computadora - CAPP
Monitoreo y Control de Procesos - PMC
Sistemas para Manufactura Flexible - FMS
Sistemas de calendarización - Scheduling
PROBADO
Probado Asistido por Computadora
ADQUISICION DE MATERIALES
Materials Requirements Planning - MRP

Tabla 1.1. Clasificación de sistemas de información

En la actualidad existen sistemas de cómputo para subsistemas de un centro de manufactura: CAD, CAM, CAE, CAPP, CAT, CAR, etc. y cualquier división en subsistemas (sección 1.1.3.) tiene sus propios sistemas de información, que Gaylord clasifica según su función genérica (Tabla 1.1). Así encontramos sistemas de información de personal, sistemas de simulación de layout, sistemas de mercadotecnia, de planeación y control de inventarios, de diseño y de manufactura del producto, y el esfuerzo de muchas personas está enfocado en reunir todos estos sistemas en uno solo.

1.3. Sistemas enfocados al proceso de manufactura.

En esta sección, ampliaremos los conceptos involucrados en el proceso de manufactura (Fig 1.3.).

1.3.1. Fase de diseño del producto.

Una empresa de manufactura, se consolida gracias a la necesidad de un producto, ya sea creada o existente. El mercado demanda un producto tangible que tiene que cumplir con ciertas características, que quizá aún no haya sido descrito técnicamente, o no se conozca su forma física. Al final de esta etapa debe existir el concepto del producto tanto en su forma física, como en sus características tecnológicas.

En su significado más obvio, diseñar significa crear un concepto y determinar sus características físicas, químicas y de forma (medidas, tolerancias, materiales). Es necesario incluir el concepto de rediseño, esto es, modificar un diseño existente por las exigencias del mercado, o para reducir costos, lo que le da al diseño caracter de ciclicidad.

Esto nos describe al diseño como la retroalimentación del

análisis: Dado un primer diseño, el análisis de este nos permite predecir su desempeño, y compararlo con las necesidades a cubrir, y mejorarlo las veces que sea necesario, hasta que haya satisfecho las necesidadesdemandadas por el diseño de la mejor forma respecto al objetivo de la empresa.

El proceso de diseño es la cuna donde nace un producto, y conceptualmente ha sido dividido de varias formas, nosotros tomaremos un enfoque hacia el diseño de piezas obtenibles utilizando técnicas de maquinado -no ensambles ni piezas obtenibles por otras técnicas de manufactura-. En lo posterior, nos referiremos solo a este tipo de piezas, por lo que no se mencionará esta distinción.

El proceso de diseño parte, desde la creación de un concepto, de una idea de un algo físico que va a cumplir una función. Debemos de conocer los alcances que pretendamos tenga el producto (durabilidad, consistencia, rigidez), o cuando menos los mínimos que debe satisfacer. Entre más información tengamos sobre lo que tiene que hacer nuestro producto, menos rediseños llevará su creación.

Los sistemas computacionales que están enfocados a esta etapa, requieren de gran capacidad gráfica y aritmética, y son el Diseño Asistido por Computadora (CAD-"Computer Aided Design/ Drafting"), y la IngenieríaAsistidapor Computadora (CAE-"Computer Aided Engineering").

1.3.1.1. CAD- Diseño Asistido por Computadora.

A primera vista, un sistema CAD, es un conjunto de dispositivos computacionales, capaz de procesar información referente a diseños lógicos de un producto, por lo que entre los dispositivos, se esperan monitores de alta resolución,

graficadores y dispositivos de entrada adecuados para el dibujo, como tabletas y ratones. En sus principios, el CAD solo soportaba realizar diseños de dos dimensiones, y su uso se limitaba a planos y vistas, pero con la ventaja de poder editarlos. El diseño por computadora fué al restirador de dibujo, lo que el procesador de palabras fué a la máquina de escribir. La capacidad de edición de un diseño, es una de los grandes apoyos de CAD, esto nos permite además de corregir errores sin redibujar, modificar sus dimensiones, rotarlo, proyectarlo, deformarlo, etc

Pero la ayuda de la computadora a un sistema de diseño no queda ahí, y pronto los sistemas computacionales fueron capaces de soportar la construcción en 3 dimensiones. Esta ampliación es mas poderosa de lo que parece, en primer lugar, el monitor de la computadora hasta hoy cuenta con 2 dimensiones y el proyectar la imagen tridimensional al monitor se presentan complicaciones de ambiguedad visual. Para solucionar esta complicación surgen varias técnicas de representación de sólidos, que se tratan en el siguiente capítulo.

La labor del diseño asistido por computadora no se detiene ahí, una vez que ha sido posible conceptualizar un sólido, será deseable conocer sus propiedades fundamentales, como momentos de inercia, centro de gravedad, centroides en las superficies, cortes seccionales, y propiedades geométricas en general.

1.3.1.2. CAE, Ingeniería Asistida por Computadora.

La modelación sólida por CAD tuvo un surgimiento posterior y al principio fué posible utilizar la computadora para realizar análisis sobre un diseño, considerando que para estos efectos, se puede prescindir de la vista del producto.

Se desarrollaron algoritmos para profundizar en las

características físicas de un producto, como el análisis dinámico, resultados ante aplicaciones de fuerzas, simulación con distintos materiales, etc.

Hoy en día, es una realidad la extracción de información volumétrica directamente de un modelo geométrico, y las funciones de CAE vienen integradas un sistema CAD.

1.3.2. Fase de Planeación del Proceso del Producto.

La planeación de procesos se define como "El acto de preparar información detallada para transformar un diseño de ingeniería en una pieza final" [5], esta información detallada, toma la forma de una hoja de procesos, y contiene la siguiente información:

1. Encabezado.
2. Herramientas.
3. Maquinaria.
4. Operaciones de maquinado.
5. Secuencia de operaciones.
6. Parámetros de maquinado.
7. Tiempo de maquinado.
8. Instrucciones especiales.

La planeación de procesos, se hace de forma manual por un experto en el área, y se basa en su experiencia y habilidad, por lo que una modelación matemática resulta muy compleja, si no es que imposible. Los acercamientos computacionales buscan simular el proceso de decisión del experto, más que modelarlo.

Podemos dividir este sistema en tres partes:

a) Sumario de planeación. Donde se examinan tamaños, tolerancias

y terminado requerido, y se obtiene una lista ordenada de operaciones en base a las máquinas y herramientas disponibles.
b) Análisis dimensional. Aquí el planeador selecciona una superficie de referencia para un grupo de operaciones similares, y puede hacer cambios al diseño si resulta imposible o muy caro manufacturar el producto.
c) Planeación detallada. Después de evaluar varias alternativas, se obtiene una lista definitiva que garantice la calidad deseada. Este proceso se auxilia del sistema computacional CAPP ("Computer Aided Process Planning").

1.3.2.1. CAPP, Planeación del Proceso Asistido por Computadora.

Su objetivo es capturar la experiencia de la "mejor práctica" para tener planes de proceso estandarizados. La forma de determinar la "mejor práctica" es precisamente practicandola, ya que se trata de conocimiento resultado de la destreza y la experiencia, esto trae como consecuencia que el experto de procesos sea una persona importante para la corporación y su conocimiento se irá con él, aun que intentase transmitirlo.

El problema es más grave, normalmente no es solamente una persona la encargada de procesos, lo que puede generar una proliferación de planes de procesos, lo que es muy difícil de controlar y detectar, y los cambios que se hacen al diseño, por lo general son indocumentados. Es muy usual que no exista una consistencia entre los planos de las piezas y las piezas fabricadas, que se hagan modificaciones de palabra y aparentemente las cosas siguan funcionando bien, hasta que el que entendía las palabras deja el lugar, o hasta que se quiere hacer una modificación escrita al diseño, y se observa que el plano es completamente obsoleto.

1.3.3. Fase de Implantación.

Esta es la fase donde el producto va a pasar de ser un concepto a un objeto tangible, donde ya existe perfectamente diseñado un plan de fabricación (y de ensamble, en su caso). Es la etapa donde hay que controlar los procesos, observar que su desempeño este conforme a lo establecido.

Todo el trabajo planeado se hará realidad en el piso de la fábrica, y se tiene que realizar de la mejor manera posible.

1.3.3.1. CAM, Manufactura Asistida por Computadora.

La tarea del CAM, es precisamente verificar que la manufactura sea conforme a lo planeado, y se puede dividir en varios subsistemas.

Las máquinas de control numérico NC, son máquinas herramientas de gran precisión, donde el trabajo no se realiza manualmente, sino que es programado con anterioridad, ya sea meidante captura -lo cual representa un gran tedio- o mediante un interfaz con una computadora. Gracias a esto se logra una variancia mínima en un proceso repetitivo, lo que nos asegura poder soportar tolerancias bajas. También se expande a la robótica, al monitoreo y control de procesos, al control numérico etc.

1.4 Algunos acercamientos a CAPP

1.4.1. Planeación de procesos.

Una vez que se tiene un diseño y sus características geométricas y dimensionales, así como el material que se utilizará en su manufactura, lo siguiente es definir las operaciones que van a transformar a un cuerpo del mismo material -materia prima- en esta pieza o ensamble, mediante procesos de manufactura. Este trabajo, trata únicamente sobre procesos de maquinado, es decir, aquellos que mediante el desvaste que produce una el filo de una herramienta a la materia prima causado por un movimiento repetitivo y direccionado, obtenga el producto deseado.

La planeación de procesos, es el acto de relacionar a una pieza o ensamble un conjunto ordenado de actividades de transformación que se aplican a la materia prima para obtenerla. El objetivo es lograr esta transformación utilizando el mínimo número de recursos -tiempo de maquinado, herramienta, refrigerante- . Entonces, podemos visualizar a la planeación de procesos, como un conjunto de elementos: máquinas-herramienta, herramientas, dispositivos de sujección, materia prima, conocimiento para transformarla, unidos con el fin de manufacturar un conjunto de piezas con el mínimo consumo o desgaste de estos mismos elementos, esto es un **sistema.**

Esta serie de transformaciones físicas aplicadas a una materia prima, se debe documentar en una hoja de procesos, que generalmente contiene:

1. Encabezado (con la información necesaria para ubicar la pieza)
2. Herramientas utilizadas
3. Maquinaria utilizada.
4. Operaciones de maquinado.
5. Secuencia de operaciones
6. Parámetros de maquinado
7. Tiempo de maquinado.
8. Instrucciones especiales

El grado de detalle en una planeación de procesos, varía según el tamaño y tipo de la empresa. Cuando se tiene un taller casero, la planeación de procesos es responsabilidad -y entendimiento- del experto de manufactura, y normalmente empírico, si se trata de una industria con grandes volúmenes de producción de muchas piezas, como la industria automotriz, entonces además de que requiere ser detallado, debe ser estandarizado, ya que una persona no puede ser realizador -ni ser el único con el conocimiento- de una producción a gran escala. Houtzeel [17], declara que el típico planeador de procesos, tiene en promedio 40 años, y una larga experiencia. También indica que hacen falta personas con este tipo de conocimiento.

La planeación de procesos hasta ahora es una actividad predominantemente humana, y es un trecho entre el diseño y la manufactura, que se debe cerrar para realizar el concepto de manufactura integrada por computadora. Sin embargo, presenta varias complicaciones:

Dada su naturaleza empírica, se tienen que desarrollar técnicas de ingeniería del conocimiento; aunque el verdadero problema consiste en generar la "mejor práctica de manufactura" para todas las piezas que se produzcan, que no necesariamente ha de ser la unión de las "mejores prácticas" de cada pieza.

La actividad de un planeador de procesos, se puede clasificar en las siguientes fases (Fig 1.4.).

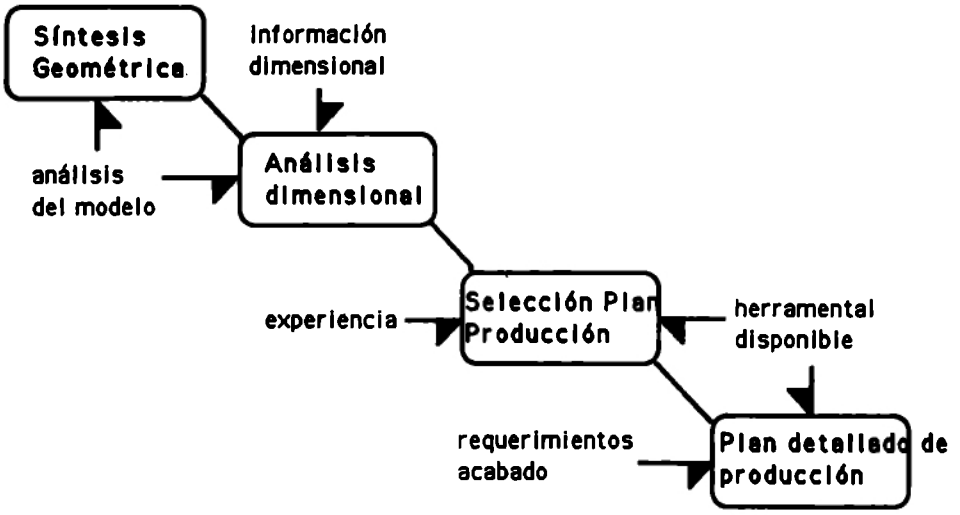


Figura 1.4. Esquema de la Planeación de Procesos

a)**Síntesis Geométrica.** El planeador analizará la figura, observando las partes que la componen, sus características, su tamaño respecto al resto de la pieza, luego sintetizará estas características logrando una comprensión geométrica del diseño. En estos momentos ha detectado las superficies exteriores de la pieza, aquellas que se han de maquinar; ya ha reconocido la naturaleza geométrica, sabe si es una figura prismática para realizar macrooperaciones de fresado o si es un sólido de revolución, para realizar macrooperaciones de torneado. Ha analizado si existen discontiuidades en las superficies, si son intrusiones o poltrusiones, o se se trata de barrenados, o de un hueco no cilíndrico. Le ha asignado a un conjunto de superficies la

clasificación de ranura, o bien de muesca. Ya tiene una idea general de como manufacturar la pieza.

b) **Análisis Dimensional.** Una vez que comprende la figura, el planeador debe conocer las dimensiones de las características geométrica y su tolerancia. También debe saber las características físicas y químicas material del que se pretende hacer la pieza, para evaluar la factibilidad de las opciones de herramienta, máquina y dispositivos que había pensado. También debe de considerar las limitaciones técnicas de la máquina-herramienta, de la misma herramienta, de los dispositivos de sujección, del sistema de producción, así como la resistencia del material , sus propiedades de deflexión, eltc.

c) **Selección de Plan de Producción.** Hasta aquí el planeador tiene varias alternativas para maquinar la pieza, o incluso varios órdenes de un mismo conjunto de operaciones. Debe seleccionar el proceso que le parezca que consuma menos recursos, o que no genere tiempos muertos en alguna máquina, o que utilize la máquina ineficiente que sólo puede hacer operaciones imprecisas, pero que deja libre otras máquinas para otros procesos, etc. Su criterio se basa en la experiencia que ha adquirido con el paso del tiempo.

El planeador puede modificar el diseño de una parte, ya sea porque es imposible de manufacturar, porque resulta muy costoso, o porque es ineficiente. Un enfoque sistémico de producción, utiliza el **diseño para manufactura** para evitar estos ciclos.

d) **Plan detallado de producción.** Una vez que ha sido determinada la ruta a seguir, se especifican las operaciones que finales que darán a la pieza los requerimientos de calidad -terminado de superficie, tolerancia, etc. y se se hace un plan detallado de producción, se documenta este conocimiento en una hoja de procesos.

La información que se requiere para hacer una planeación de procesos es la siguiente:

+ Geometría. De la información geométrica conceptualizamos la pieza, pero sólo es posible extraer dimensiones nominales.

+ Tolerancias. Todo proceso de manufactura requiere además de una dimensión nominal, el grado de variancia que es permisible.

+Material empleado en su fabricación. Se requieren las propiedades físicas y químicas del material o aleación para determinar los parámetros de maquinado (velocidad y profundidad de corte, herramientas que se pueden utilizar, dispositivos de sujección).

+ Disponibilidad de máquinas-herramienta, de herramientas y dispositivos, no sólamente se refiere a su existencia o no, sino a su grado de uso.

+ Capacidades Tecnológicas de el material de la pieza y del herramental. Deflexión de la pieza, y uso posible de dispositivos auxiliares -contrapunto-, dimensiones máximas y mínimas de cada máquina-herramienta, superficies maquinables por cada herramienta, etc.

Los esfuerzos en automatizar la planeación de procesos, se pueden clasificar en aquellos que simulan la decisión de un experto en manufactura mediante la comparación con decisiones ya tomadas -variante-, y aquellos que pretenden simular el razonamiento abstracto del experto -generativo-.

1.4.2. Enfoque variante.

La idea en que se basa el enfoque variante es que piezas similares han de tener secuencias similares. Un problema al que se enfrenta es la agrupación de partes "similares", o en la asignación de una pieza nueva a alguna familia de partes.

Para esto, se han desarrollado propuestas e implantaciones de técnicas para agrupar partes que tengan similaridades de forma, dimensión y/o rutas de proceso, asignando un código a cada pieza. Estas técnicas reciben el nombre de **Tecnología de Grupos (GT)**. De acuerdo a las características de cada fábrica, se formarán familias de partes, posiblemente basados en técnicas ya existentes. Se debe de tener muy claro el vínculo que une a los grupos, ya que se formarán planes

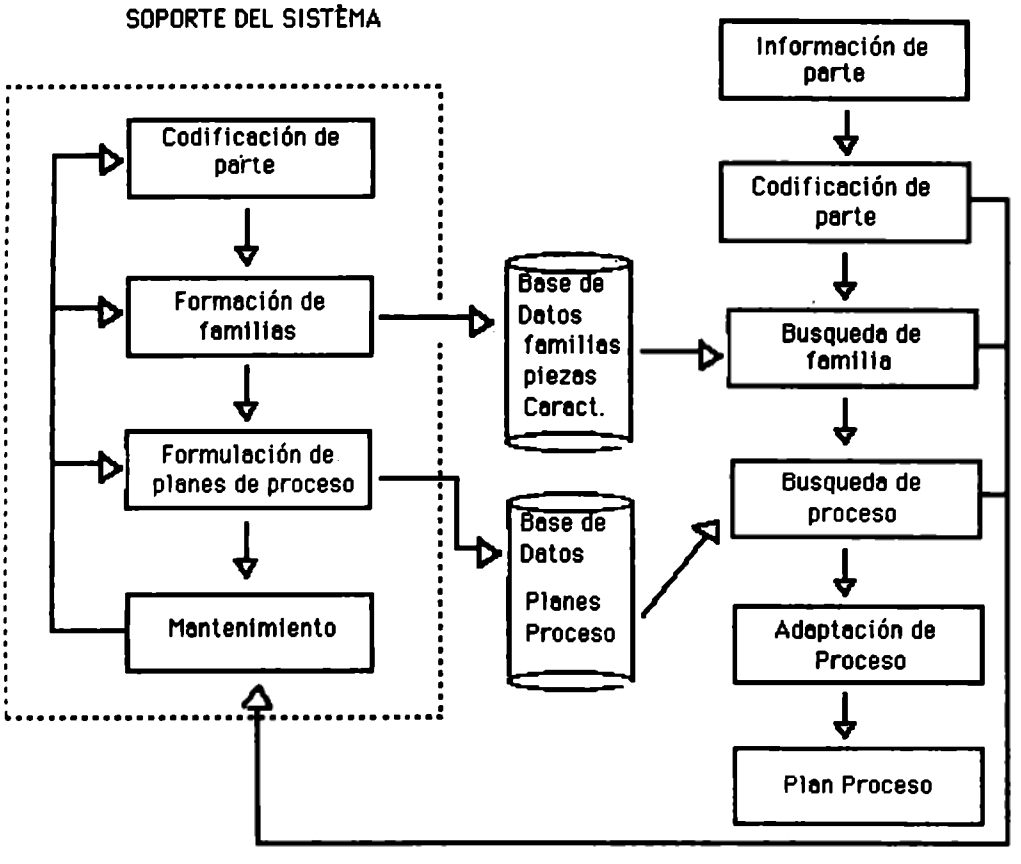


Figura 1.5 Enfoque variante de CAPP

genéricos de proceso para cada familia. Entonces el mejor proceso de producción sigue siendo el generado a través del tiempo y no es necesario evaluarlo, pero sí darle un mantenimiento periódico tanto al criterio de formación de familias, como a la elaboración de planes estándar. Esto se hará con la ayuda de un planeador de procesos.

Este método consume muchos recursos el diseño del soporte, (Fig 1.5) se tiene que hacer una formación de familias, y para cada familia encontrar el mejor plan genérico de proceso, y los parámetros de variación. Se deben de definir también un algoritmo de reconocimiento para ubicar una parte en una familia, incluso se puede implantar una base de datos de las piezas que ya han sido reconocidas como parte de una familia, y lograr el reconocimiento de partes nuevas por iteración directa con el usuario.

Una vez que el sistema se ha implantado, es entendible, y fácil de operar, sin embargo no es transportable de una fábrica a otra, y la efectividad del sistema es dependiente completamente del diseño del sistema de soporte. Este enfoque no ha sido muy popular en los desarrollos realizados hasta ahora en CAPP. [7]

1.4.3. Enfoque Generativo.

El enfoque generativo, pretende representar en la computadora el conocimiento y la experiencia de manufactura generados hasta hoy, y en un enfoque de sistemas expertos, generar nuevo conocimiento o mejorar el existente.

Este enfoque, busca simular el razonamiento de un experto de procesos, representando la forma conocimiento que ha adquirido, no el resultado. Un sistema generativo se puede conceptualizar en las siguientes fases (Fig. 1.6.)

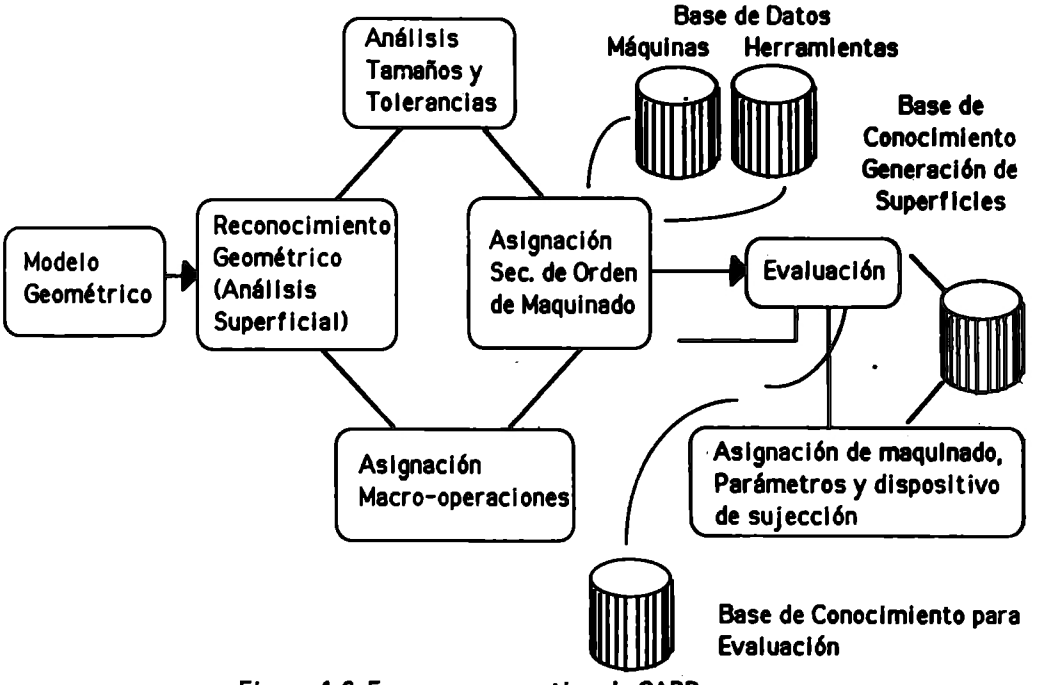


Figura 1.6. Esquema generativo de CAPP

a) **Reconocimiento Geométrico.** En esta fase, se debe sintetizar los criterios que sigue un planeador de proceso para formarse una idea de un plan general. Uno de los criterios principales es la generalidad geométrica -prisma, sólido de revolución, híbrido-. Para lograr esta síntesis, se debe de contar con una modelo geométrico adecuado, donde sea posible relacionar las superficies espacialmente, detectar su tipo e identificar discontinuidades que serán interpretadas como ranuras, o barrenos, o poltrusiones. La salida de esta fase es la representación de la evaluación de estos criterios.

b) Asignación de Macrooperaciones. Las macrooperaciones van directamente relacionadas con el tipo de superficie que queremos generar. Su orden depende de la relación espacial de estas superficies. Debemos de tener bien reconocidas las superficies que deseamos del modelo. Esta fase debe de representar y evaluar los criterios para asignarles una secuencia de macrooperaciones.

c) Análisis de tamaños y tolerancias. Un sistema de CAD, arroja información geométrica -y quizá física- de un diseño. Sin embargo, sólo es posible extraer dimensiones nominales. Chang afirma que no existe un estándar aún para incluir esta información en CAD, por consiguiente no existe una interfase directa entre CAD y CAPP. Esta información tiene que ser entrada al sistema, y se hace en forma de código o de diálogo iterativo entre el usuario y la máquina. [7] .

d) Asignación de secuencias ordenadas de maquinado. Con la información de macrooperaciones, máquinas-herramienta, herramientas disponibles y de tamaños y tolerancias, podemos obtener todas las secuencias ordenadas de maquinado factibles.

Una máquina herramienta es un dispositivo, que mediante la devastación mecánica -desprendimiento de viruta- , mediante una herramienta de corte, es capaz de transformar las características superficiales de un objeto (materia prima). Nosotros conocemos el tipo y características de la superficie que es capaz de hacer una máquina herramienta con cada una de las herramientas de corte y dispositivos de sujección que puede usar, conocemos también sus limitaciones tecnológicas.

El proceso de asignar una secuencia ordenada de maquinado, parte de haber establecido en la pieza una secuencia de proceso, determinado por precedencias. Lo demás es el proceso inverso a lo explicado en el párrafo anterior, esto es, a cada superficie, con ciertas características geométricas, de material, acabado y tolerancia, determinar la máquina herramienta y los dispositivos con que es posible producirla.

e) Evaluación. El asignarle una calificación a una secuencia de maquinado a fin de determinar mejorabilidad, no es una tarea trivial, se deben considerar todos los consumos de los bienes de producción en que se incurre. Aún así es posible que al combinar los planes de menor consumos saturemos una máquina herramienta (la que era mas rentable), generando un cuello de botella. Se deben considerar además disponibilidad de las máquinas, trayectoria de la pieza, etc. Esta información debe de estar representada de alguna forma en una base de conocimiento.

f) Plan detallado. Aquí se asignará una lista de operaciones que garantice la calidad deseada, -acabado de superficie, tolerancia-. Los parámetros de corte, dispositivos de sujección y el tipo de herramienta son determinantes para definir la calidad final.

1.5. Hipótesis.

1.5.1. Definición del problema

El problema que este trabajo trata, es la propuesta de un esquema de reconocimiento geométrico, para que a partir de la información geométrica de un sólido representada en un modelo de armazón, genere el conocimiento suficiente para relacionarle una secuencia ordenada de macrooperaciones, y apoyarlo mediante el desarrollo de un sistema computacional.

1.5.2 Justificación de la tesis.

La Planeación de Procesos asistida por computadora, es un área que requiere de recursos computacionales importantes para lograrse, y es una brecha que se está cerrando para lograr el concepto CIM.

La capacidad de razonamiento espacial del hombre es un gran misterio todavía[5], y no es fácil describir el proceso que sigue nuestra mente para poder reconocer las características geométricas y dimensionales de un diseño. Nosotros sabemos que "una imagen vale más que mil palabras", y la mejor comprobación es el esfuerzo que mucha gente está haciendo actualmente para reconocer una escena visual.

Reconocer una imagen se refiere al extraer información utilizable de una escena visual para algún fin, y es un área de estudio de la Inteligencia Artificial. La escena visual puede estar en dos dimensiones, como la imagen de una cámara, o el mapa de un sonar, o en tres dimensiones, como un modelo de armazón que genera un paquete de diseño. El fin, puede ser el reconocimiento de una pieza en particular, de un punto donde aplicar soldadura, de algún componente electrónico o de las características de un solido, que nos permiten asignarle una secuencia de maquinado.

El conocimiento de procedimientos (Know-How), es la información que un experto de manufactura posee, mucha de la cual, proviene del empirismo de su experiencia y su intercambio de información con otros expertos, además este conocimiento se ha ido haciendo particular de la empresa, cada empresa tiene sus propias formas de manufactura, sus propios cánones, y entre más grande sea la empresa, más se tienen que documentar y estandarizar. Esto vitaliza al experto de manufactura -¿Qué hacemos si se nos va?-, además que uno de los más grandes problemas a los que se enfrenta un centro de manufactura, es la actualización de sus diseños, generalmente se llega a un acuerdo entre el experto de procesos, y el ingeniero de diseño, olvidándose de los incomódos formalismos del papel. Cuando consultamos nuestro archivo de partes, nos podemos encontrar desde partes que ya no existen, o aquellas que sus especificaciones han sido modificadas.

¿Cómo hace un experto en procesos para determinar la mejor secuencia de maquinado?, o una mejor pregunta: Dada una igualdad de condiciones de diseño, disponibilidad de máquinas, herramientas y dispositivos de sujección, ¿Todos los expertos de procesos establecerían la misma secuencia de maquinado?. La proliferación de secuencia de

procesos es un problema en todas las industrias de transformación.

Esta tesis, pretende sumarse a los esfuerzos del área de computación en manufactura, para integrar los sistemas de información y conocimiento para manufactura.

1.5.3. Alcances de la tesis.

La labor del experto de procesos, -que es el enfoque que utilizo para dividir un sistema generativo de planeación de procesos asistida por computadora en el siguiente capítulo-, se puede dividir en la síntesis geométrica del diseño -análisis-, para la asignación de una secuencia ordenada de macrooperaciones, que junto con la información dimensional, y la disponibilidad de máquinas y de herramientas, servirá para asignar una secuencia ordenada de maquinado.

Esta tesis propone un esquema de datos y algoritmos, para el reconocimiento de características geométricas de maquinado, para figuras que se puedan generar con líneas y arcos.

§ 2 Propuesta de modelación.

2.1 Concepto.

Dado que el objetivo es automatizar la interfaz entre el diseño y la planeación de procesos asistidos por computadora y no proponer un nuevo esquema de diseño, partiremos de la modelación en 2.5 dimensiones, que es la más usual dada la herencia de diseño en base a proyecciones ortogonales proveniente de la ingeniería mecánica. Enfocándonos en la información extraíble de CADKEY llamaremos entidad a una unidad básica de dibujo en dos dimensiones, una línea, un punto, un arco, un círculo, un polígono, líneas especiales (que definan funciones creadas por el diseñador o por una interpolación). Cada entidad tendrá atributos que pueden ser geométricos para definir su posición espacial, o bien gráficos que le darán las características de despliegado.

La estructura de datos de un modelo gráfico se concentrará en manejar la información proveniente de las entidades. La figura 2.1. nos muestra los atributos para las entidades de CADKEY. Sin embargo, no podemos considerar esto como una modelación sólida, ya que no posee ninguna característica volumétrica, ni siquiera superficial. Dirigimos nuestro esfuerzo a lograr una modelación sólida a partir de esta información.

Conviene mencionar también que la inclusión de arcos y círculos complica la recuperación de la información, ya que la información proveniente de la base de datos de CADKEY no es suficiente para definir de forma única un círculo, además se le relaciona un número de vista. Una vista es una transformación isomorfa a un sistema nuevo de coodenadas donde el plano xy define el plano de vista y el eje z define la profundidad. CADKEY utiliza 8 prefijadas: frontal, superior, inferior, trasera, derecha, izquierda, isométrica y axonométrica -un caso especial de transformación dimétrica y

tipo de entidad	atributos geométricos
PUNTO	Posición, Identificador
LINEA	Posición inicio. Posición final. Identificador
ARCO o CIRCULO	Posición centro. Radio. Angulo de inicio. Incremento angular. Identificador.
POLIGONO	Número de Vertices. Posición de cada vertice (ordenado).

Figura 2.1. Atributos de algunas entidades de CADKEY

trimétrica. Si un usuario crea una figura no lineal (arco, círculo, spline) que no sea paralela a alguna de estas vistas, automáticamente se genera una nueva vista con esta característica, y se le asigna un número secuencial a partir del nueve.
El archivo de vistas se traduciré a un arreglo rectangular de 10 columnas, donde la primera indica el número de vista (identificador), y las otras nueve representan a la matriz de transformación de la base ortonormal. La figura 3.3 indica el funcionamiento de estas matrices. También hay que indicar que aunque CADKEY trabaja de la misma forma a un arco que a un círculo, nuesro concepto es

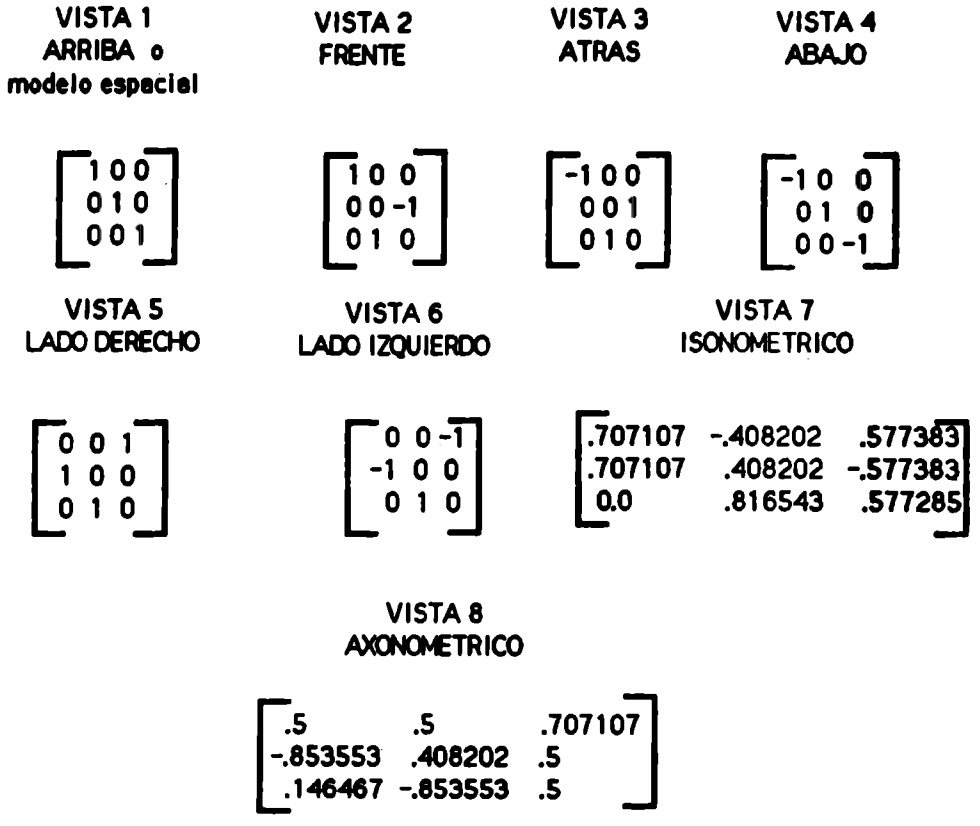


Figura 2.2 Vistas predefinidas de CADKEY

diferente, ya que un círculo espera que otra entidad lo enlace a otro círculo, y no se sabe con precisión por que punto pasará, un arco espera la unión de cuatro entidades, dos en su superficie, y dos en otra, que lo conectarán con un otro arco pero es seguro el enlace por sus puntos finales. Por esto, hay que dar un trato diferente a los ciclos, y cambiaremos su tipo de 3 a 30 (para no interferir con alguna ampliación del sistema).

Se considerarán únicamente los sólidos que sea posible generar con líneas, arcos y círculos, entonces tendremos cinco tipos diferentes de superficies, que conviene separar, porque representan distintos tipos de macrooperaciones.

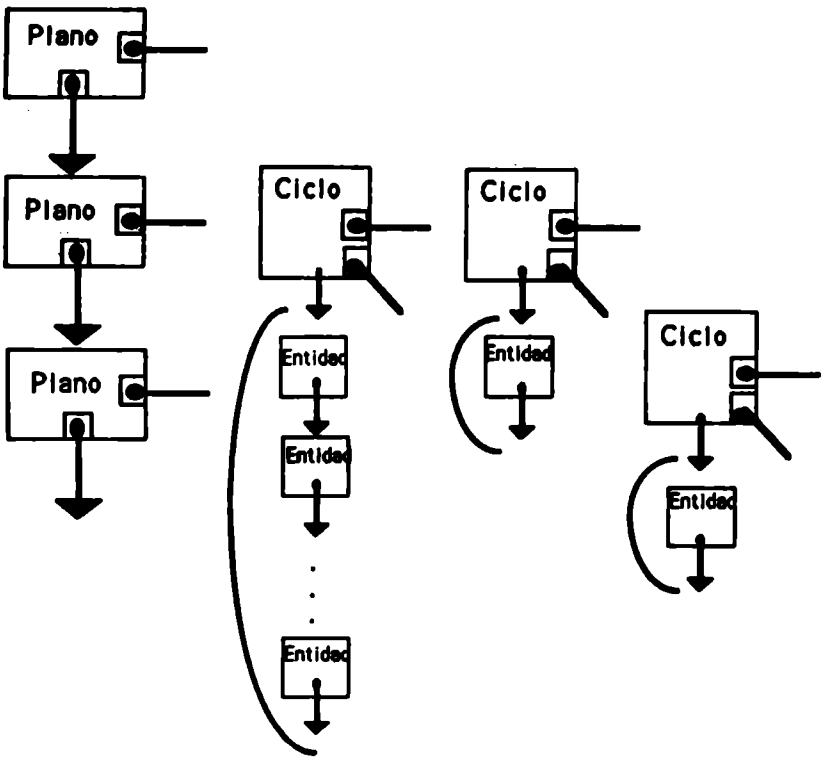


Figura 2.3. Estructura dinámica de datos para un plano

Lamaremos a estas superficies plano, túnel, túnel de arco, revolución y buje. Todas las superficies, excepto el plano, tienen un número fijo de entidades que las componen, por lo que es posible crear un registro para cada una de ellas, sin embargo, el plano requiere de una estructura más compleja, ya que no tiene un número predeterminado de entidades que lo compongan, más aún, estas entidades estarán ordenadas en ciclos (Fig. 2.3.)

Diremos que un ciclo es interno respecto a otro ciclo, si pertenece a el, y que un ciclo es hermano de otro ciclo, si ambos están en la misma superficie, pero ninguno pertenece al otro.

Para facilitar los cálculos, es conveniente conceptualizar las superficies planas, como una traslación de un subespacio vectorial (hiperplano), de esta forma, podemos asignarle una base ortonormal (y un desplazamiento), y hacer los cálculos en coordenadas relativas al plano (2 dimensiones).

Las superficies planas serán representadas como una lista lineal, donde cada nodo representará un plano, mediante su ecuación normalizada, a su vez, este nodo señalará a una lista ortogonal de ciclos, que se enlazarán ya sea por el vínculo de pertenencia, o por el de hermano. Así mismo, cada uno de estos ciclos debe apuntar a una lista circular que contiene las entidades ordenadas expresadas en coordenadas relativas al plano desplazado. Estas entidades contienen un ciclo, indicando su dirección respecto a la dirección de su información original (coordenadas globales).

En el caso de un arco, necesitaremos tres coordenadas, una representará al centro, y las otras dos al punto de inicio y de final de acuerdo al orden del ciclo. Esta información es necesaria para evaluar la pertenencia de los ciclos.

.2.2. Descripción de estructura de datos.

Para procesar la información proveniente de la base de datos de Cadkey, que han sido resumidas en dos listas dinámicas lineales, una de entidades y otra de vistas, hay que tomar en cuenta lo siguiente:

Cadkey trabaja de la misma forma a un arco que a un círculo, pero nuesro concepto es diferente, ya que un círculo por sí mismo define a un ciclo, y un arco es parte de un ciclo. Un círculo espera que otra entidad lo enlace a otro círculo, y no se sabe con precisión por que punto pasará, un arco espera la unión de cuatro entidades, dos en su superficie, y dos en otra, que lo conectarán con un otro arco pero es seguro el enlace por sus puntos finales. Por esto, hay que dar un trato diferente a los ciclos, y cambiaremos su tipo de 3 a 30 (para no interferir con alguna ampliación del sistema).

Se considerarán únicamente los sólidos que sea posible generar con líneas, arcos y círculos, entonces tendremos cinco tipos diferentes de superficies, que conviene separar, porque representan distintos tipos de macrooperaciones.

Llamaremos a estas superficies plano, túnel, túnel de arco, revolución y buje, de acuerdo a la figura 2.6.

Todas las superficies, excepto el plano, tienen un número fijo de entidades que las componen, por lo que es posible crear un registro para cada una de ellas.

El plano requiere de una estructura más compleja, ya que no tiene un número predeterminado de entidades que lo compongan, más aún, estas entidades estarán ordenadas en ciclos (Fig. 2.7.)

Diremos que un ciclo es interno respecto a otro ciclo, si pertenece a el, y que un ciclo es hermano de otro ciclo, si ambos están en la misma superficie, pero ninguno pertenece al otro.

Para facilitar los cálculos, es conveniente conceptualizar las superficies planas, como una traslación de un subespacio vectorial (hiperplano), de esta forma, podemos asignarle una base ortonormal (y un desplazamiento), y hacer los cálculos en coordenadas relativas al plano (2 dimensiones).

Las superficies planas serán representadas como una lista lineal, donde cada nodo representará un plano, mediante su ecuación normalizada, a su vez, este nodo señalará a una lista ortogonal de ciclos, que se enlazarán ya sea por el vínculo de pertenencia, o por el de hermano. Así mismo, cada uno de estos ciclos debe apuntar a una lista circular que contiene las entidades ordenadas expresadas en coordenadas relativas al plano desplazado. Estas entidades contienen un ciclo, indicando su dirección respecto a la dirección de su información original (coordenadas globales).

En el caso de un arco, necesitaremos tres coordenadas, una representará al centro, y las otras dos al punto de inicio y de final de acuerdo al orden del ciclo. Esta información es necesaria para evaluar la pertenencia de los ciclos.
Un ejemplo de la estructura de datos generada para un modelo sencillo, es mostrado en la siguiente tabla.

.2.3. Descripción de los algoritmos utilizados.

2.3.1. Creación de las listas iniciales

Este es el módulo que convertirá la información de la base de datos de CADKEY en estructuras de datos lineales.

Como se mencionó es factible que el acceso a la base de datos de entidades de CADKEY, se realice mediante un módulo CDE, y un compilador comercial Metaware, conceptualmente la única diferencia estriba en evitar la escritura y lectura en un archivo de datos de todas las entidades, y de todas las vistas.

El módulo genera dos listas lineales una que contiene la información de las entidades (únicamente líneas y círculos), y otra que contiene la información de las vistas predefinidas y creadas, que es información complementaria de las entidades circulares.

Para nuestros fines de modelación, es necesario distinguir entre el concepto de arco y el de círculo. Un arco es una parte de un ciclo, y un círculo es un ciclo por sí mismo. Un círculo genera un vértice suave -y en nuestro caso, la única posibilidad de que exista un vértice de esta naturaleza, es el vértice que une a dos círculos para definir un sólido de revolución. El módulo asignará un tipo diferente a los círculos.

Respetando la nomenclatura de cadkey, el tipo de una entidad está definido como un entero, a las líneas les asigna un valor de 2, a los arcos y a los círculos 3. Este módulo asigna a las líneas y a los arcos su mismo valor, y a los círculos les asigna un valor de 30, ya que los números anteriores los utiliza cadkey para otros tipos de entidad.

Se genera también una lista de vértices, extrayendo la información de entidades de la siguiente forma: Cuando se trata de líneas o arcos, con las coordenadas de cada uno de sus extremos se hace una búsqueda para ver si ha sido registrado como vértice, si no se registra. En ambos casos se agrega como atributos el número de entidad que contiene al vértice, cuál de sus extremos finales es el vértice. Aun que ha sido limitado el número de entidades que coinciden en un vértice a tres, se implantó una estructura dinámica para expandir esta limitación.

Los círculos se tratan de una forma especial, ya que no tienen definido algún vértice, se espera que sean referenciados por una línea. Entonces, para cada círculo se hace una búsqueda exhaustiva con los vértices existentes para saber qué vertice está contenido en él, en cuyo caso se coloca en sus atributos al círculo.

El enfoque propuesto es basado en vértices, y esta lista será base para la síntesis geométrica. También se guarda en esta lista el tipo de vértice que se trata, aunque esta información es obtenible hasta analizar más características del modelo.

2.3.2. Generación de planos.

La finalidad de este módulo, es identificar todos los planos de una figura.

Recibe como entrada, la información generada por el módulo anterior, y como salida genera una lista lineal de ecuaciones planares diferentes, donde cada elemento de la lista apunta a su vez, a una lista lineal de entidades que pertenecen a ese plano.

Tres puntos no colineales son suficientes para generar a un plano, estos puntos pueden provenir de dos líneas que coincidan en uno de sus extremos, o de una sola línea curva.

El algoritmo de generación de planos, selecciona primero a todos los pares de líneas que coincidan en uno de sus puntos, y a cada uno de ellos, les asigna una ecuación estandarizada. Un plano se representa por la ecuación $ax + by + cz - d = 0$, computacionalmente, por la cuádrupla (a, b, c, d), sin embargo, la cuádrupla (ßa, ßb, ßc, ßd) representa al mismo plano. Este problema se puede solucionar de distintas formas, la más natural es usar la cuádrupla $(w, x, y, z) = (1 / [(a, b, c, d)])\ (a, b, c, d)$, es decir, el vector direccional de (a, b, c, d) en R^4; sin embargo, el cálculo de la magnitud puede substituirse por uno más sencillo al representar al plano por la tupla (x, y, z, ∂), donde $\partial \varepsilon \{x, y, z\}$, y $f(\partial)$ está en función de las variables restantes.

Una vez que se ha generado un plano, este debe de ser comparado con los planos existentes y agregado a la lista en caso de que no hubiese existido. Si ya existía, únicamente agregamos las entidades que también pertenecen a él.

Cuando se agrega un plano a los ya existentes, se le asigna una base ortonormal, y un desplazamiento. De esta forma, será posible tratar las coordenadas como parte de un subespacio lineal de dos dimensiones desplazado, o sea que todas las operaciones entre las coordenadas del plano, como búsqueda de cubierta convexa, o determinar si un ciclo pertenece a otro, serán realizadas en coordenadas de dos dimensiones.

Al agregar cualquier entidad a las listas de las que conforman el plano, se guardan como atributos sus coordenadas relativas. (proyección ortogonal).

Una vez que tenemos todos los planos que pueden generar todas las líneas, verificamos una a una las entidades circulares. para agregarlas en una forma similar.

La diferencia estriba en que la información que recibimos de una figura circular en la lista de entidades -centro, radio, angulo de inicio y incremento angular-, no es suficiente para definir un plano. En la lista de entidades, viene además un entero que identifica a la vista en donde está definido la figura circular. La vista está definida por una base ortonormal, y representa a una transformación lineal isomorfa de cambio de coordenadas. La figura circular es paralela al plano xy del sistema de coordenadas de la vista, con una profundidad z.

Entonces se obtiene la ecuación del plano xy de la vista, y se sigue el mismo procedimiento. En el caso de que este plano no existiera, la base ortonormal estará definida por los vectores x, y de la vista.

2.3.3. Creación de ciclos.

Una vez que agrupamos a las entidades por planos -cada entidad lineal pertenecerá a lo más a dos grupos, y las entidades circulares únicamente a uno-, es necesario reconocer los ciclos que forman estas entidades.

Mediante la comparación ordenada de entidades, agrupamos a las entidades por las coincidencias entre sus puntos finales. Si la figura no tiene error, se debe formar al menos un ciclo -cadena cerrada.

Un ciclo debe contar entre sus atributos, con elementos que permitan identificar su orientación, en el modelo propuesto, la lista lineal de entidades pertenecientes a un plano, se reordena en forma de una o varias listas circulares, donde cada una de estas serán un atributo de cada ciclo. Para definir la ordenación, se crea un campo en cada nodo indicando el signo de su orientación; este signo será positivo si la orientación del ciclo es la misma que la de la figura en la lista de entidades, y negativo en caso contrario.

Topológicamente, un ciclo define a un conjunto cerrado -concavo o convexo-, y es hasta que lo hemos formado que podemos determinar la dirección del interior del ciclo. Para determinar esta dirección, se calculan los ángulos entre las entidades del ciclo conservando su dirección, si esta suma es 2π rad., el interior del ciclo se encuentra a la derecha de cada entidad -en sentido de las manecillas del reloj-; si la suma es -2π rad. el interior del ciclo se encuentra a la izquierda de cada entidad. -Siempre que se consideren los ángulos positivos en dirección de las manecillas del reloj, y el ángulo se obtenga en el rango $(-\pi, \pi)$-.

El sentido que ha tomado un ciclo respecto a su interior, es otro de sus atributos, y se señala como un signo en su estructura.

Consideramos además que un círculo es por sí mismo un ciclo, y como los arcos tienen un punto inicial y uno final, no existe diferencia en su tratamiento en este algoritmo respecto de las líneas.

2.3.4. Algoritmo de inclusión de ciclos.

Si estuvieramos modelando un politopo, cada plano tendría exactamente un ciclo, y sería la estructura más sencilla. Al generalizar el sólido, existe el concepto de poltrusiones e intrusiones, y en un modelo de armazón generan un ciclo interno a otro. Cuando se tiene un ranurado, se parte la superficie en dos ciclos, y ninguno contiene al otro. Estos son los conceptos de ciclo interno, y ciclo hermano, y se detectan mediante un algoritmo de inclusión de ciclos.

Se diseño un algoritmo en base a un algoritmo existente de proyección horizontal, que funciona únicamente para variedades lineales en dos dimensiones. El primer algoritmo parte del concepto de inclusión de un punto en una variedad lineal -polígono-, mediante la formación de una "caja contenedora" con las coordenadas planares máximas y mínimas de todos los vértices del polígono. Si el punto esta fuera de la caja, el punto está fuera del polígono, si el punto está dentro de la caja, puede o no puede estar dentro del polígono y esto se deduce mediante el número de intersecciones que tenga cualquier recta que incluya al punto, con las aristas del polígono. Un segmento de recta está determinado por:

$$f(t) = a + bt, \quad \text{para } t \in [0,1].$$

Si $t \in R$ definimos completamente la recta. La intersección entre la recta $a + bt$ y la recta $c + ds$, es la solución a un sistema de ecuaciones lineales.

Al incluir arcos en un círculo, este algoritmo se debe expandir para localizar la intersección entre una recta y un arco; para ello consideramos la ecuación paramétrica de segmento de recta, y la ecuación paramétrica de un arco. Un círculo se define así:

$$g(s) = ((c_x, c_y) + r((\cos \phi_1 + s(\phi_2 - \phi_1)), (\sin \phi_1 + s(\phi_2 - \phi_1)))'$$

donde c_x, c_y son las coordenadas del centro, r su radio, ϕ_1 su ángulo inicial y ϕ_2 su incremento angular en radianes - el apóstrofe significa traspuesto.

A partir de la igualdad $f(t) = g(s)$, podemos despejar a t en función de s, y encontrar la intersección mediante la solución numérica de :

$$h(s) = b_y r \cos \beta - b_x r \sin \beta - (a_x b_y + b_x (c_y - a_y) - c_x b_y) = 0.$$

donde $\beta = (\phi_1 + s(\phi_2 - \phi_1))$, y han sido indicados los componentes de a y de b.

Esta ecuación es senoidal, y tendrá 0, 1 ó 2 soluciones en el intervalo [0,1], que indica el número de intersecciones de una recta (infinita) con un arco.

El algoritmo recibe como datos de entrada a los ciclos que conforman una superficie, registrados como ciclos hermanos, y su finalidad es acomodarlos de acuerdo a su inclusión entre ellos. Como informaicón de salida, arroja la esctructura descrita en la figura 3.7. Se dice que un ciclo tiene profundidad 0 si no está dentro de ningún otro ciclo, tiene profundidad 1 si está contenido en solo un ciclo de profundidad 0, y así sucesivamente, generando una estructura recursiva.

El algoritmo recorre la lista de superficies, y se aplica a los ciclos de cada una de ellas, formando todos los conujuntos posibles de dos ciclos (a,b), verificando que se cumpla solo una de las siguientes reglas:

1. Si a está dentro de b, entonces a es ciclo interno de b.
2. Si b está dentro de a, entonces b es ciclo interno de a.
3. Si a no esta dentro de b, ni b dentro de a, entonces son ciclos hermanos.

En la implantación del sistema, para verificar si el ciclo a esta contenido en el ciclo b, únicamente comprobamos si un punto de a pertenece al ciclo b, ya que no es permitida la inclusión parcial de ciclos ni la intersección de entidades,situaciones que generan errores gráficos fácilmente detectables. Si se quisiera verificar estos posibles errores, se tiene que verificar además de la inclusión de todos los puntos de a en b, la no intersección de entidades de a con entidades de b. -se supone que se debe verificar con anterioridad la no inclusión entre entidades de cada ciclo-.
Para saber si un punto p pertenece a un ciclo c, nos basamos en el algoritmo propuesto por Mortenson[24], que plantea la construcción de la caja mínima que contenga al polígono. Si el punto no pertenece a esta caja, entonces no pertenece al polígono, simplificando los cálculos. Cuando se incluyen entidades curvas, debemos asegurar que la caja contenga al polígono. Las entidades curvas que estamos tratando son círculos y arcos; en el caso de un círculo, la verificación de la inclusión se reduce a la sol,ución de una desigualdad cuadrática, pero esto no es tan trivial en el caso de un arco.
Un arco está conectado a otras entidades para formar un ciclo, y puede representar a una saliente curva o su contraparte convexa. Esta información la obtenemos al verificar el signo del ciclo, y el signo del arco -originalmente un arco siempre viene declarado en sentido de las manecillas del reloj-, si los signos son iguales, el arco representa una saliente, si son diferentes representa su contraparte. La caja solo se afecta en el primer caso, podemos calcular los incrementos sobre los ejes que representa el arco, dado su radio interno, y la posición de su centro.

Si el punto está dentro de la caja, entonces es posible que lo este también del ciclo. Para saber esto, el algoritmo plantea el trazo de un segmento de recta paralelo al eje x que pase por el punto p y el conteo de las intersecciones que tenga con las aristas del polígono a. Si se intersecta un número par de veces ntes o después del punto, está afuera, y si es un número impar está dentro. Al generalizar este algoritmo, debemos tomar en cuenta las intersecciones entre una recta y un arco, descritas con anterioridad. En la implantación estas ecuaciones se simplifican, ya que la componente en y del vector de dirección de la recta (b_x) es cero por ser una línea horizontal.
La solución a la ecuación de intersección es numérica, y se implantó con el método de la secante, se escogió un método cerrado por la naturaleza senoidal de la función, y porque sabemos que solo puede tener 0, 1 ó 2 raíces en el intervalo [0,1]. Si existe un cambio de signo entre h(0) y h(1) sabemos que existe una sola raíz, y aplicamos el método; si no existe dicho cambio, entonces existen cero o dos raíces, y evaluamos f(0.5), si es de signo contrario existen dos raíces, una en cada mitad, y evaluamos dos veces el método, una en [0, 0.5] y otra en [0.5,0].

Como se usa una estructura recursiva para el acomodo de ciclos, su formación debe ser de la misma naturaleza, y para un ciclo verifica si sus ciclos posteriores estan incluidos en el, en cuyo caso, desencadena el nodo y lo inserta como ciclo interno, un proceso inverso sucede si este ciclo está incluido en otros nodos. Si no sucede ninguna de estas opciones, no se realiza alguna acción, ya que en inicio los ciclos son declarados como hermanos. Se realiza de nuevo el proceso con su ciclo hermano, y con su ciclo interno.

3.2.2.5. Cálculo de ángulos entre superficies.

La información que se ha generado hasta el momento es vital para el cálculo entre las superficies del sólido. El algoritmo a partir de las estructuras logradas hasta ahora, deberá de calcular los ángulos entre todas las superficies, y almacenar esta información en la lista de entidades. Algunos autores, como Chang [6] sugiere la formación de una matriz binaria, o ternaria indicando si hay relación entre las superficies

y si es cóncava o convexa según el caso, pero la creación de una nueva estructura matricial dinámica es innecesaria.
El algoritmo se basa en el conocimiento que se tiene de la dirección del interior de cada ciclo, y lo usa como base para definir la dirección normal de cada plano.
El algoritmo recorre todas las superficies, y dentro de las superficies todos los ciclos. Cada entidad de cada ciclo, está relacionando a otra superficie con la actual. Si la profundidad del ciclo es par, entonces el interior del sólido esta señalado por el interior del ciclo, contrariamente a si la profundidad es impar. Esta información determina una dirección -y sentido- normal a la superficie.

En el caso de superficies especiales, aun que no existe una dirección normal a un túnel, si es posible con el mismo método definir si esta representando un hueco, o un sólido.

3.2.2.6. Reconocimiento de superficies especiales.

Se han detectado cuatro posibles superficies no planas, todas con un número fijo de entidades involucradas. Su reconocimiento se hace simulando el que haría una persona.
Cuando se ha detectado un arco, se sabe que será parte de un ciclo, dentro de una superficie, pero también debe ser parte de un túnel -recto o arqueado-. El algoritmo busca para cada arco, una línea u otro arco que coincida con el (en un modelo gráfico válido, las coincidencias deben de ser únicamente en puntos terminales, y que pertenezca a otra superficie, excepto en el caso del círculo). Cuando se detecta esta entidad, digamos una línea, se buscará ahora un arco que termine en la otra punta de la línea, si no existe, indicaremos un error, en el otro caso, seguiremos la trayectoria buscando ahora una línea que cierre esta cadena.

Las superficies involucradas con círculos se componen de tres entidades: dos círculos, y una línea o arco que los una. La búsqueda es similar, detectamos un círculo, buscamos una línea/arco tal que una de sus puntas pertenezca al perímetro del círculo, y después buscamos un círculo tal que su perímetro incluya a esta línea/arco.

§ 3. Propuesta de Síntesis Geométrica.

El representar un sólido mediante un modelo -el que se considere más apropiado- es la base para transformar esa representación en un concepto. La computadora debe de "comprender" a la pieza, cuando menos a las características topológicas que sirven para tomar decisiones respecto a las macrooperaciones. Para lograr un reconocimiento geométrico es importante poder representar la comprensión de la pieza.

A un ingeniero, en muchos de los casos, le basta con tener tres vistas y una proyección isométrica para conceptualizar un sólido, comprender sus superficies principales, sus depresiones y poltrusiones. Cuando nosotros observamos las figuras tridimensionales en un texto, o incluso un modelo de representación visual de un sistema de diseño, lo comprendemos de igual forma.

La capacidad de razonamiento espacial humana, es un área que aún permanece obscura, no sabemos cuál es la información que utilizamos de una escena visual para comprenderla, y esto se denomina visión en el campo de la inteligencia artificial.

Si le presentamos un dibujo a un experto de procesos, lo primero que hace es conceptualizar la naturaleza general de la pieza, si es prismática, si es un sólido de revolución, o si es una combinación de ambas, y visualiza un sólido más sencillo que contenga al inicial, al cual le aplicará operaciones de maquinado -destructivas- hasta lograr el objetivo.

El planeador asigna entonces un orden de obtención de superficies -no necesariamente único-, existirán superficies de referencia para manufacturar otras superficies. Siguiendo este orden, el planeador asignará a cada superficie una macrooperación.

Cada macrooperación tiene alternativas de máquina-herramienta, y de herramienta para realizarse, pero no todas serán factibles, dadas las limitaciones tecnológicas de la máquina-herramienta, de la herramienta, de los dispositivos de sujección, y su disponibilidad. El planeador debe de conocer información adicional: características del material del que se pretende hacer la pieza, requerimientos de

superficie, y las limitaciones propias del equipo disponible.

El planeador, evaluará y seleccionará entonces, basado en su experiencia, máquinas, herramientas y dispositivos a cada macrooperación, generando una secuencia ordenada de maquinado.

La finalidad del reconocimiento geométrico -para CAPP- es diseñar un algoritmo, o un sistema experto, capaz de extraer, y procesar la información necesaria para la asignación de una secuencia ordenada de maquinado a un diseño por computadora. En pocas palabras, un módulo de reconocimiento geométrico, será el que obtenga la información semántica de un diseño.

El esquema que se propone en esta tésis para el reconocimiento de características de maquinado que posteriormente nos llevarán a macrooperaciones, es una orientación a vértices, y está basada en el siguiente esqema de grafos atribuidos.

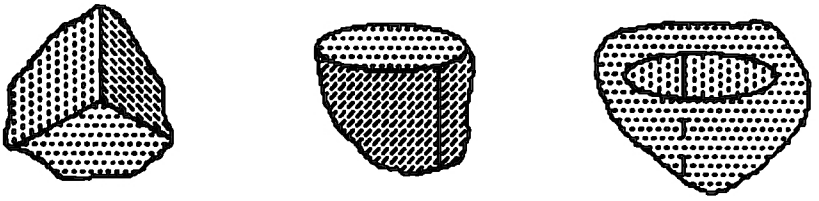
Sea V el conjunto de vértices de un sólido, cada uno con atributos resaltando para el reconocimiento la asignación de un tipo de vértice en base a la relación angular que existe entre las superficies que conecta, y sea E el conjunto de entidades que relacionará a los elementos de V con atributos definidos -dimensión-. Las relaciones entre V y E definen contornos de superficies, pero esta información ya la posee el modelo, lo que nos interesa es el tipo de vértices y como se conectan.

Decimos que un vértice v es convexo, si todas las relaciones entre las superficies que conecta son convexas, en otro caso será parcial o totalmente cóncavo. Si un sólido únicamente tiene vértices convexos, se trata de un sólido convexo -si es prismático-.

Si consideramos a la materia prima como a un sólido convexo, entonces la presencia de vértices con algún grado de concavidad es evidencia de que ha sido removido material, creando concavidades con alguna forma en particular.

Inicialmente distinguimos a seis tipos de vértices (Fig 4.1), pero la información que proporciona un tipo de vértice no es suficiente, la información de las características se forma mediante la relación entre los vértices concavos, hasta que estos encuentran a vértices totalmente convexos. El Dr. Sánchez [4.17] menciona algunas características básicas, todas ellas representables con este esquema de gráficos, como muestra la figura 4.2.

a) Totalmente convexo b) Dos relaciones concavas y una convexa c)Dos relaciones concavas y una convexa



d) Totalmente concavo e) Relación convexa y suave convexa f) Relación convexa y suave concava

Figura 4.1. Diferentes tipos de vértices.

Al analizar nuestro modelo sólido, buscaremos vértices con algún grado de concavidad, y cuando los encontremos seguiremos sus enlaces, hasta encontrar sólo vertices convexos. Después reconoceremos este grafo como una característica, o como la unión de varias. Con los atributos de las aristas (del grafo) podemos obtener las dimensiones de la característica, y su posición espacial.

A cada característica, le asignaremos una macrooperación, su realización dependerá de las dimensiones y de las características tecnológicas que hasta este momento no se han contemplado, y el orden de estas macrooperaciones, estará determinado por un sorteo topológico.

Inicialmente los seis tipos de vértices son definidos con ángulos rectos, porque aún que una relación sea cóncava, el proceso de manufactura puede variar si se trata de un ángulo agudo o de uno obtuso, se pueden definir tantos tipos de características como sea necesario.

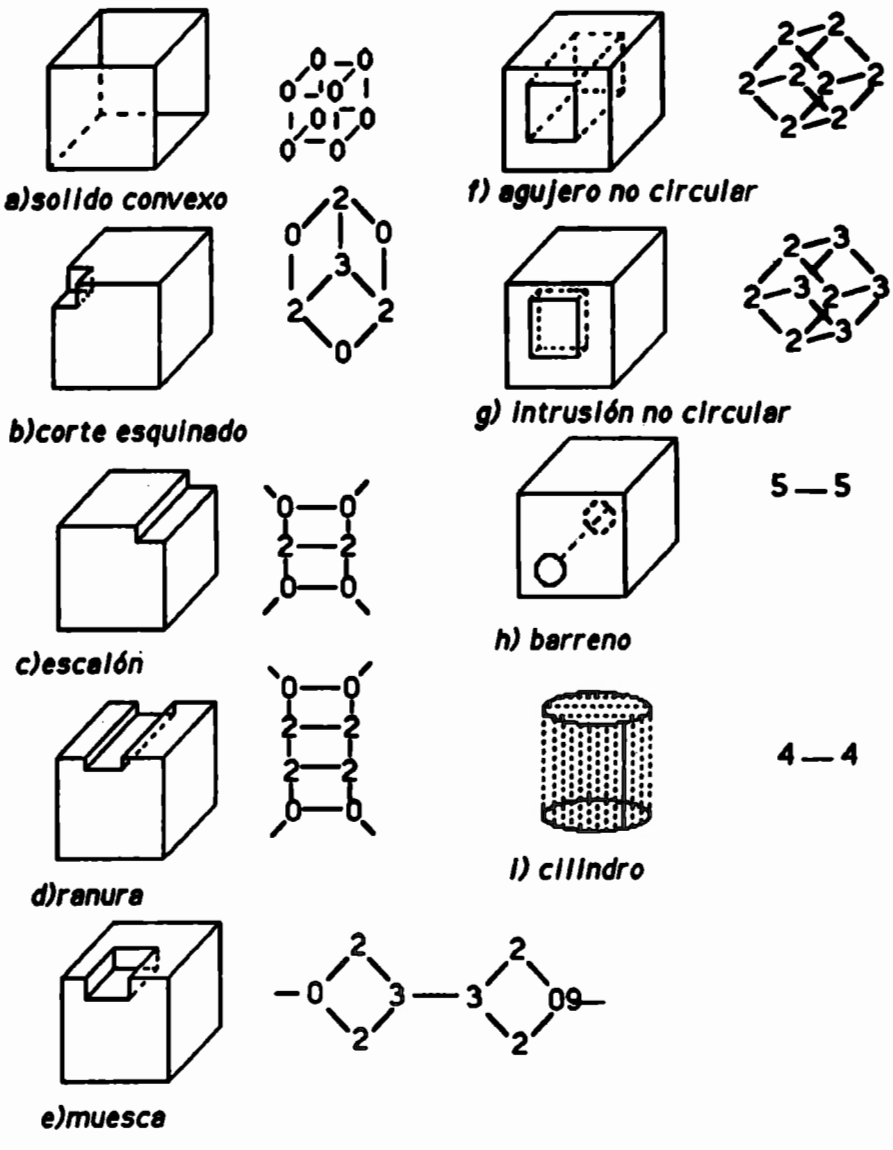


Figura 4.2. Características básicas de maquinado y su representación orientada a vértices.

3.2. Bases de conocimiento e información.

La propuesta de modelación considera una base de información y dos de conocimiento, descritas como bases de datos relacionales, para no limitar su crecimiento.

a) Base de archivos de maquinado.
Este es un archivo que contiene los nombres de los archivos donde se encuentran explicaciones detalladas de cada característica de maquinado. En un futuro se puede desarrollar una interfaz iterativa con el usuario o un sistema experto para ampliar libremente esta información. La estructura se muestra en la figura 4.3

b) Base de conocimiento de tipos de vértices.
Esta base de conocimiento nos permite relacionar un vértice a un tipo predefinido, o definible mediante algún sistema de adquisición de conocimiento. La información necesaria de cada vértice está resumida en siete campos, como lo muestra la figura 4.4. Se asumen intervalos cerrados, y en el caso de un vértice suave -el que cierra un cilindro-, consideramos su angulo como $\pi/2$ si el interior del sólido está hacia adentro del círculo o como $-\pi/2$ en caso contrario.

c) Base de conocimiento de características para macrooperaciones.
Las macrooperaciones también están registradas en una base de conocimientos, como una lista de nodos con tres enlaces. Cada nodo tendrá un identificador y su tipo de vértice. Existe un archivo para cada característica de maquinado y la figura 4.5. muestra el ejemplo para un corte esquinado.

id.	nombre	archivo
0	convexo	XXX.dat
1	conv2-conc1	XXC.dat
2	conv1-conc2	XCC.dat
3	concavo	CCC.dat
4	concav, suav conv	CSx.dat
5	convexo, suav conc	CSc.dat

Figura 4.3. Base de datos para archivos de características de maquinado.

Id	relación 1 min.	max	relación 2 min.	max	relación 3 min.	max
0	π/2	π/2	π/2	π/2	π/2	π/2
1	π/2	π/2	π/2	π/2	-π/2	-π/2
2	π/2	π/2	-π/2	-π/2	-π/2	-π/2
3	-π/2	-π/2	-π /2	-π/2	-π/2	-π/2
4	π/2	π/2	π/2	π/2	-	-
5	π/2	π/2	-π/2	-π/2	-	-

Figura 4.4. Base de conocimientos para tipos de vértices.

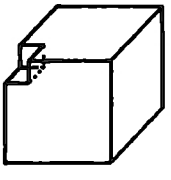
3.3.Explicación de los algoritmos utilizados.

Para el funcionamiento de cualquier algoritmo, se asume que las bases de conocimiento e información ya existen, ya que su información aunada al modelo del sólido determinará la síntesis deseada.

3.3.1. Algoritmo de reconocimiento de características de maquinado.

Su función es identificar las diferentes características de maquinado que aparezcan en el sólido. Tiene como entrada el modelo del sólido, detallado en el capítulo anterior, y da como salida una lista de características identificadas y relacionadas a ciertos vértices.
El algoritmo hace un recorrido exhaustivo y por profundidad, es decir, primero buscará algún grado de concavidad en la lista de vértices, si no lo encuentra se trata de un sólido convexo, en caso contrario tratara de hacer coincidir la estructura que rodea a ese vértice con la primer característica de maquinado, si no lo logra con la segunda y así sucesivamente.

nodo		liga 1		liga 2		liga 3	
tipo	id	tipo	id	tipo	id	tipo	id
3	1	2	3	2	4	2	2
2	2	0	7	3	1	0	5
2	3	0	7	3	1	0	6
2	4	3	1	0	5	0	6
0	5	2	2	2	4		
0	6	2	3	2	4		
0	7	2	3	2	2		



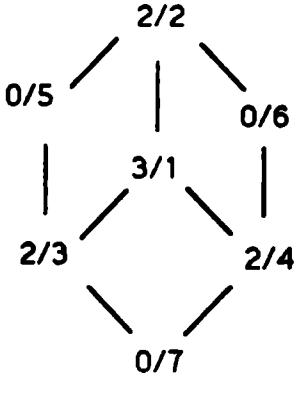


Figura 4.5. Ejemplo de base de conocimientos para características de macrooperaciones (corte esquinado)

Como es un enlace por tipos, son posibles varios acomodos de los grafos, esto se evalúa por profundidad, ya que por amplitud crecería demasiado la información y se complica por la presencia de ciclos en el grafo. Se va formando la estructura como una réplica de alguna posible formación de la que existe en el archivo hasta que se agotan las posibilidades y se intenta con la siguiente característica o hasta que coincide y se deduce la presencia de esta característica.

Para hacer la búsqueda por profundidad, el algoritmo parte con la primer característica a reconocer, y busca en el sólido un vértice con el máximo grado de concavidad con que cuente la característica -v.g. el máximo grado de concavidad de un corte esquinado es 3, y el máximo de un escalón 2-. Cuando localiza un tipo igual, busca los enlaces del sólido y los compara con el renglón de la tabla; ya que no es necesario que se conserve el orden, el algoritmo hará el seguimiento para el primer enlace del sólido con la primer liga de la característica hasta que no coincidan los tipos, entonces lo intenta coincidir con el segundo vértice del sólido y si fracasa con el tercero. Si fracasan tres vértices, buscará otro vértice del mismo tipo en el sólido, hasta que ya no existan vértices de ese tipo y no reconozca la característica de maquinado. Se utilizan banderas para evitar la ciclicidad.

Cuando se logra el éxito para el primer vértice, se repite el algoritmo para el segundo y si se logra éxito de nuevo, se repite para el tercero. Si el resultado es exitoso, la característica ha sido reconocida, y se crea un nodo en una lista de características de la figura. Existe una lista para cada característica y cada vértice del sólido que se ha detectado que pertenece a una caracterísitca es señalado con una bandera.

El algoritmo entonces cambia de característica a reconocer, y hace el mismo procedimiento hasta que no haya características a reconocer.
Finalmente el algún vértice no ha sido señalado se genera un mensaje que indica que la existencia de una característica no conocida.

3.3.2. Posición espacial.

Cada caracteristica reconocida tendrá asociada una posición espacial consistente en sus coordenadas mínimas y máximas en cada uno de sus ejes. Si el volumen que envuelve a una superficie se encuentra dentro del de otra, entonces decimos que la segunda superficie cubre a la primera, que debe de ser maquinada primero y le asignamos un valor de verdadero en una matriz binaria de posiciones.

La secuenciación, se refiere a un ordenamiento topológico en base a la matriz de posiciones espaciales. No contamos con la información suficiente para establecer un orden único de operación, sin embargo, existen operaciones que han de ser realizadas antes que otras, por lo que hablaremos de niveles.

Una macrooperación estará en primer nivel si ninguna otra le cubre, en caso contrario, estará un nivel más abajo que la operación que le ha cubierto, de esta forma, se presenta como resultado las operaciones que pueden ser candidatas a realizarse desde la materia prima, luego se presentan operaciones que deben de ser realizadas después de otras, etc.

§ 4. Resultados y Discusión.

La metodología propuesta en los capítulos anteriores, tomó forma mediante la construcción de un programa computacional en lenguaje C, cuyo listado se encuentra en los anexos de esta tesis. A continuación se muestran dos ejemplos en los que fue aplicado este programa. Se trata de figuras muy sencillas dada la cantidad de información generada para cada una de ellas.

La primera consiste en un cubo con un barreno. Como se puede observar, en la definición de la pieza existe un arreglo rectangular de quince renglones y nueve columnas. El primer dígito de cada renglón es un número entero que indica el tipo de entidad que se trata 2 si es una línea, 3 si es un círculo. El segundo es el identificador de identidad que CADKEY asignó conforme se fué haciendo la construcción. El tercero es el número de vista en el que fue declarada la entidad; esta información no tiene sentido en el caso de una línea porque la información siguiente la define de manera única, no es así en el caso de los círculos.
Como se puede ver, el diseño consiste en trece líneas y dos círculos, y se muestra en la figura 4.1.

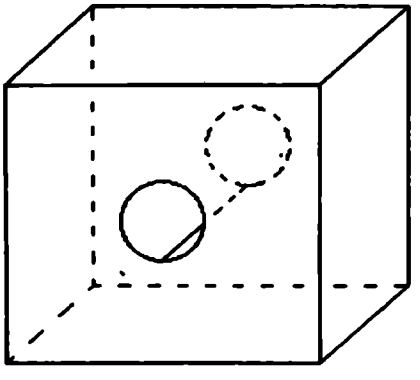


figura 4.1. Cubo con barreno

Al ejecutar el programa, observamos que lo inicial es obtener todos los planos posibles entre todas las entidades, obsérvese como las entidades 22 y 23 no forman plano con ninguna otra, esto se debe a que son entidades circulares y definen por sí mismas un plano.

Una vez que el sistema ha verificado los planos, detecta los planos diferentes, que en este caso son seis, y los lista con sus parámetros, para asignarles una base ortonormal y un desplazamiento -recordemos que un plano es espacio vectorial sólamente en el caso de que pase por el origen-. Además de la base ortonormal, le asigna a cada una de las entidades su pertenencia a un ciclo que va acomodando en orden de padre-hijo.

Con esta información fue posible completar el archivo de entidades, dándoles ahora la referencia a dos superficies y a uno o dos vértices dependiendo el caso. Se puede observar que se utiliza un enfoque híbrido orientdo a vértices y orientado a entidades. Una vez que se tiene esta información, interesa conocer los planos en los que caen las superficies, algunos de estos serán planos de corte, que es la siguiente información generada.

El reconocimiento de características de maquinado es la siguiente tarea, para finalizar con un ordenamiento, en donde se va a indicar el nivel de cada operación. Si es factible realizar todas las tareas al mismo tiempo todas tendrán nivel 1, si una tarea se tiene que realizar después de otra, entonces la segunda tendrá nivel más bajo que la primera.

A continuación se muestra el funcionamiento del sistema para el ejemplo de la figura 4.1 y para el ejemplo de la figura 4.2

EJEMPLO 1. FIGURA CUBO CON BARRENO.

```
2 10 0 0.00000000 0.00000000 0.00000000 1.00000000 0.00000000 0.00000000
2 11 0 1.00000000 0.00000000 0.00000000 1.00000000 1.00000000 0.00000000
2 12 0 1.00000000 1.00000000 0.00000000 0.00000000 1.00000000 0.00000000
2 13 0 0.00000000 1.00000000 0.00000000 0.00000000 0.00000000 0.00000000
2 14 0 0.00000000 0.00000000 1.00000000 1.00000000 0.00000000 1.00000000
2 15 0 1.00000000 0.00000000 1.00000000 1.00000000 1.00000000 1.00000000
2 16 0 1.00000000 1.00000000 1.00000000 0.00000000 1.00000000 1.00000000
2 17 0 0.00000000 1.00000000 1.00000000 0.00000000 0.00000000 1.00000000
2 18 0 0.00000000 1.00000000 1.00000000 0.00000000 1.00000000 0.00000000
2 19 0 0.00000000 0.00000000 1.00000000 0.00000000 0.00000000 0.00000000
2 20 0 1.00000000 1.00000000 0.00000000 1.00000000 1.00000000 1.00000000
2 21 0 1.00000000 0.00000000 0.00000000 1.00000000 0.00000000 1.00000000
3 22 1 0.50000000 0.50000000 1.00000000 0.25000000 0.00000000 6.28318531
3 23 1 0.50000000 0.50000000 0.00000000 0.25000000 0.00000000 6.28318531
2 24 0 0.50000000 0.75000000 0.00000000 0.50000000 0.75000000 1.00000000
```

INFORMACION DE PROCESO PARA FIGURA 1

Haciendo plano entre entidades 10 y 11
Haciendo plano entre entidades 10 y 13
Haciendo plano entre entidades 10 y 19
Haciendo plano entre entidades 10 y 21
Haciendo plano entre entidades 11 y 12
Haciendo plano entre entidades 11 y 20
Haciendo plano entre entidades 11 y 21
Haciendo plano entre entidades 12 y 13
Haciendo plano entre entidades 12 y 18
Haciendo plano entre entidades 12 y 20
Haciendo plano entre entidades 13 y 18
Haciendo plano entre entidades 13 y 19
Haciendo plano entre entidades 14 y 15
Haciendo plano entre entidades 14 y 17
Haciendo plano entre entidades 14 y 19
Haciendo plano entre entidades 14 y 21
Haciendo plano entre entidades 15 y 16
Haciendo plano entre entidades 15 y 20
Haciendo plano entre entidades 15 y 21
Haciendo plano entre entidades 16 y 17
Haciendo plano entre entidades 16 y 18
Haciendo plano entre entidades 16 y 20
Haciendo plano entre entidades 17 y 18
Haciendo plano entre entidades 17 y 19
Haciendo plano entre entidades 22 y 0
Haciendo plano entre entidades 23 y 0
sup. 1, parametros: z = 0.0000,0.0000,0.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)
ENTIDADES :
ent 10, coords = (1.0000, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
ent 11, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
ent 13, coords = (1.0000, 1.0000) (1.0000, 0.0000) (0.0000, 0.0000)
ent 12, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 23, coords = (0.2500, 0.5000) (0.2500, 0.5000) (0.5000, 0.5000)

sup. 2, parametros: y = 0.0000,0.0000,-0.0000
base = {(1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}

desplazamiento : (0.0000, 0.0000, 0.0000)
ENTIDADES :
  ent 10, coords = (0.0000, 0.0000) (1.0000, 0.0000) (0.0000, 0.0000)
  ent 19, coords = (0.0000, 1.0000) (0.0000, 0.0000) (0.0000, 0.0000)
  ent 21, coords = (1.0000, 0.0000) (1.0000, 1.0000) (0.0000, 0.0000)
  ent 14, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)

sup. 3, parametros: x = 0.0000,0.0000,1.0000
 base = {(0.0000, -1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 1.0000, 0.0000)
ENTIDADES :
  ent 11, coords = (1.0000, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
  ent 20, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
  ent 21, coords = (1.0000, 0.0000) (1.0000, 1.0000) (0.0000, 0.0000)
  ent 15, coords = (1.0000, 1.0000) (0.0000, 1.0000) (0.0000, 0.0000)

sup. 4, parametros: y = 0.0000,0.0000,1.0000
 base = {(1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 1.0000, 0.0000)
ENTIDADES :
  ent 12, coords = (1.0000, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
  ent 18, coords = (0.0000, 1.0000) (0.0000, 0.0000) (0.0000, 0.0000)
  ent 20, coords = (1.0000, 0.0000) (1.0000, 1.0000) (0.0000, 0.0000)
  ent 16, coords = (1.0000, 1.0000) (0.0000, 1.0000) (0.0000, 0.0000)

sup. 5, parametros: x = 0.0000,0.0000,-0.0000
 base = {(0.0000, -1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 1.0000, 0.0000)
ENTIDADES :
  ent 13, coords = (0.0000, 0.0000) (1.0000, 0.0000) (0.0000, 0.0000)
  ent 18, coords = (0.0000, 1.0000) (0.0000, 0.0000) (0.0000, 0.0000)
  ent 19, coords = (1.0000, 1.0000) (1.0000, 0.0000) (0.0000, 0.0000)
  ent 17, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)

sup. 6, parametros: z = 0.0000,0.0000,1.0000
 base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 1.0000)
ENTIDADES :
  ent 14, coords = (1.0000, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
  ent 15, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
  ent 17, coords = (1.0000, 1.0000) (1.0000, 0.0000) (0.0000, 0.0000)

ent 16, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 22, coords = (0.2500, 0.5000) (0.2500, 0.5000) (0.5000, 0.5000)

sup. 1, parametros: z = 0.0000,0.0000,0.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)
CICLOS :

ciclo 1 (-), ENTS: 10 (+), 11 (+), 12 (+), 13 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

ciclo 2 (+), ENTS: 23 (+),
xmin : 0.2500, xmax : 0.7500, ymin : 0.2500, ymax : 0.7500

sup. 2, parametros: y = 0.0000,0.0000,-0.0000
base = {(1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 0.0000, 0.0000)
CICLOS :

ciclo 1 (+), ENTS: 10 (+), 21 (+), 14 (-), 19 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

sup. 3, parametros: x = 0.0000,0.0000,1.0000
base = {(0.0000, -1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 1.0000, 0.0000)
CICLOS :

ciclo 1 (-), ENTS: 11 (+), 20 (+), 15 (-), 21 (-),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

sup. 4, parametros: y = 0.0000,0.0000,1.0000
base = {(1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 1.0000, 0.0000)
CICLOS :

ciclo 1 (-), ENTS: 12 (+), 18 (-), 16 (-), 20 (-),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

sup. 5, parametros: x = 0.0000,0.0000,-0.0000
base = {(0.0000, -1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 1.0000, 0.0000)
CICLOS :

ciclo 1 (+), ENTS: 13 (+), 19 (-), 17 (-), 18 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

sup. 6, parametros: z = 0.0000,0.0000,1.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 1.0000)
CICLOS :

ciclo 1 (-), ENTS: 14 (+), 15 (+), 16 (+), 17 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

ciclo 2 (+), ENTS: 22 (+),
xmin : 0.2500, xmax : 0.7500, ymin : 0.2500, ymax : 0.7500

Entidades :
Entidad (10 2 0) 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
en superf. (1 2) Angulo (1.5708), Vertices (1 2)
Entidad (11 2 0) 1.0000 0.0000 0.0000 1.0000 1.0000 0.0000
en superf. (1 3) Angulo (1.5708), Vertices (2 3)
Entidad (12 2 0) 1.0000 1.0000 0.0000 0.0000 1.0000 0.0000
en superf. (1 4) Angulo (1.5708), Vertices (3 4)
Entidad (13 2 0) 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
en superf. (1 5) Angulo (1.5708), Vertices (4 1)
Entidad (14 2 0) 0.0000 0.0000 1.0000 1.0000 0.0000 1.0000
en superf. (6 2) Angulo (1.5708), Vertices (5 6)
Entidad (15 2 0) 1.0000 0.0000 1.0000 1.0000 1.0000 1.0000
en superf. (6 3) Angulo (1.5708), Vertices (6 7)
Entidad (16 2 0) 1.0000 1.0000 1.0000 0.0000 1.0000 1.0000
en superf. (6 4) Angulo (1.5708), Vertices (7 8)
Entidad (17 2 0) 0.0000 1.0000 1.0000 0.0000 0.0000 1.0000
en superf. (6 5) Angulo (1.5708), Vertices (8 5)
Entidad (18 2 0) 0.0000 1.0000 1.0000 0.0000 1.0000 0.0000
en superf. (4 5) Angulo (1.5708), Vertices (8 4)
Entidad (19 2 0) 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
en superf. (2 5) Angulo (1.5708), Vertices (5 1)
Entidad (20 2 0) 1.0000 1.0000 0.0000 1.0000 1.0000 1.0000
en superf. (3 4) Angulo (1.5708), Vertices (3 7)
Entidad (21 2 0) 1.0000 0.0000 0.0000 1.0000 0.0000 1.0000
en superf. (2 3) Angulo (1.5708), Vertices (2 6)
Entidad (22 30 1) 0.5000 0.5000 1.0000 0.2500 0.0000 6.2832
en superf. (6 -515) Angulo (-1.5708), Vertices (10 0)

Entidad (23 30 1) 0.5000 0.5000 0.0000 0.2500 0.0000 6.2832
en superf. (1 -515) Angulo (-1.5708), Vertices (9 0)
Entidad (24 2 0) 0.5000 0.7500 0.0000 0.5000 0.7500 1.0000
en superf. (-515 0) Angulo (-1.5708), Vertices (9 10)
vista 1:
{ (1.0000 0.0000 0.0000), (0.0000 1.0000 0.0000), (0.0000, 0.0000, 1.0000) }
vista 2:
{ (1.0000 0.0000 0.0000), (0.0000 0.0000 1.0000), (0.0000, -1.0000, 0.0000) }
vista 3:
{ (-1.0000 0.0000 0.0000), (0.0000 0.0000 1.0000), (0.0000, 1.0000, 0.0000) }
vista 4:
{ (-1.0000 0.0000 0.0000), (0.0000 1.0000 0.0000), (0.0000, 0.0000, -1.0000) }
vista 5:
{ (0.0000 1.0000 0.0000), (0.0000 0.0000 1.0000), (1.0000, 0.0000, 0.0000) }
vista 6:
{ (0.0000 -1.0000 0.0000), (0.0000 0.0000 1.0000), (-1.0000, 0.0000, 0.0000) }
vista 7:
{ (0.7071 0.7071 0.0000), (-0.4082 0.4082 0.8165), (0.5774, -0.5774, 0.5773) }
vista 8:
{ (0.5000 -0.8336 0.1464), (0.5000 0.1464 -0.8536), (0.7071, 0.5000, 0.5000) }

vertice 1, (0.0000, 0.0000, 0.0000)
Entidades (id, side, tipo): (10, 1, 2) (13, 2, 2) (19, 2, 2)
vertice 2, (1.0000, 0.0000, 0.0000)
Entidades (id, side, tipo): (10, 2, 2) (11, 1, 2) (21, 1, 2)
vertice 3, (1.0000, 1.0000, 0.0000)
Entidades (id, side, tipo): (11, 2, 2) (12, 1, 2) (20, 1, 2)
vertice 4, (0.0000, 1.0000, 0.0000)
Entidades (id, side, tipo): (12, 2, 2) (13, 1, 2) (18, 2, 2)
vertice 5, (0.0000, 0.0000, 1.0000)
Entidades (id, side, tipo): (14, 1, 2) (17, 2, 2) (19, 1, 2)
vertice 6, (1.0000, 0.0000, 1.0000)
Entidades (id, side, tipo): (14, 2, 2) (15, 1, 2) (21, 2, 2)
vertice 7, (1.0000, 1.0000, 1.0000)
Entidades (id, side, tipo): (15, 2, 2) (16, 1, 2) (20, 2, 2)
vertice 8, (0.0000, 1.0000, 1.0000)
Entidades (id, side, tipo): (16, 2, 2) (17, 1, 2) (18, 1, 2)
vertice 9, (0.5000, 0.7500, 0.0000)
Entidades (id, side, tipo): (24, 1, 2) (22, 0, 30)
vertice 10, (0.5000, 0.7500, 1.0000)
Entidades (id, side, tipo): (24, 2, 2) (22, 0, 30)
Orden de planos :
Plano 1 (-0.0000, -0.0000 1.0000)
Id: 1, Prof.: 0
Id: 2, Prof.: 1

Plano 2 (-0.0000, 1.0000 -0.0000)
Id: 1, Prof.: 0
Plano 3 (-1.0000, 0.0000 0.0000)
Id: 1, Prof.: 0
Plano 4 (0.0000, -1.0000 0.0000)
Id: 1, Prof.: 0
Plano 5 (1.0000, -0.0000 -0.0000)
Id: 1, Prof.: 0
Plano 6 (-0.0000, -0.0000 1.0000)
Id: 1, Prof.: 0
Id: 2, Prof.: 1

TUNELES

TUNELES

Cilindro -515, tipo 2 Ents c1 22, c2 23, l1 24 signol -, Vertices 10 & 9

VERTICES:
vertice 1, (0.0000, 0.0000, 0.0000) tipo 0 carac 0
vertice 2, (1.0000, 0.0000, 0.0000) tipo 0 carac 0
vertice 3, (1.0000, 1.0000, 0.0000) tipo 0 carac 0
vertice 4, (0.0000, 1.0000, 0.0000) tipo 0 carac 0
vertice 5, (0.0000, 0.0000, 1.0000) tipo 0 carac 0
vertice 6, (1.0000, 0.0000, 1.0000) tipo 0 carac 0
vertice 7, (1.0000, 1.0000, 1.0000) tipo 0 carac 0
vertice 8, (0.0000, 1.0000, 1.0000) tipo 0 carac 0
vertice 9, (0.5000, 0.7500, 0.0000) tipo 5 carac 1
vertice 10, (0.5000, 0.7500, 1.0000) tipo 5 carac 1

CARACTERISTICAS:
1 cilindro 0.2500 0.5000 0.2500 0.5000 0.0000 1.0000

ORDEN:
1 cilindro 1

EJEMPLO 2 CUBO CON DOS TUNELES.

2 10 0 0.00000000 0.00000000 0.00000000 1.00000000 0.00000000 0.00000000
2 11 0 1.00000000 0.00000000 0.00000000 1.00000000 0.25000000 0.00000000
2 12 0 1.00000000 0.25000000 0.00000000 0.00000000 0.25000000 0.00000000
2 13 0 0.00000000 0.25000000 0.00000000 0.00000000 0.00000000 0.00000000
2 14 0 0.00000000 0.75000000 0.00000000 1.00000000 0.75000000 0.00000000
2 15 0 1.00000000 0.75000000 0.00000000 1.00000000 1.00000000 0.00000000
2 16 0 1.00000000 1.00000000 0.00000000 0.00000000 1.00000000 0.00000000
2 17 0 0.00000000 1.00000000 0.00000000 0.00000000 0.75000000 0.00000000
2 18 0 1.00000000 0.00000000 0.00000000 1.00000000 0.00000000 1.00000000
2 19 0 1.00000000 0.00000000 1.00000000 0.75000000 0.00000000 1.00000000
2 20 0 0.00000000 0.00000000 1.00000000 0.00000000 0.00000000 0.00000000
2 21 0 1.00000000 1.00000000 0.00000000 1.00000000 1.00000000 1.00000000
2 22 0 1.00000000 1.00000000 1.00000000 0.75000000 1.00000000 1.00000000
2 23 0 0.00000000 1.00000000 1.00000000 0.00000000 1.00000000 0.00000000
2 24 0 1.00000000 0.00000000 1.00000000 1.00000000 1.00000000 1.00000000
2 25 0 0.00000000 0.00000000 1.00000000 0.00000000 1.00000000 1.00000000
3 26 6 -0.5000000 0.00000000 -1.0000000 0.25000000 0.00000000 3.14159265
3 27 5 0.50000000 0.00000000 0.00000000 0.25000000 0.00000000 3.14159265
2 28 0 0.25000000 1.00000000 1.00000000 0.00000000 1.00000000 1.00000000
2 29 0 0.00000000 0.00000000 1.00000000 0.25000000 0.00000000 1.00000000
2 30 0 0.25000000 0.00000000 1.00000000 0.25000000 1.00000000 1.00000000
2 31 0 0.75000000 0.00000000 1.00000000 0.75000000 1.00000000 1.00000000
3 32 2 0.50000000 1.00000000 0.00000000 0.25000000 0.00000000 3.14159265
3 33 3 -0.5000000 1.00000000 1.00000000 0.25000000 0.00000000 3.14159265

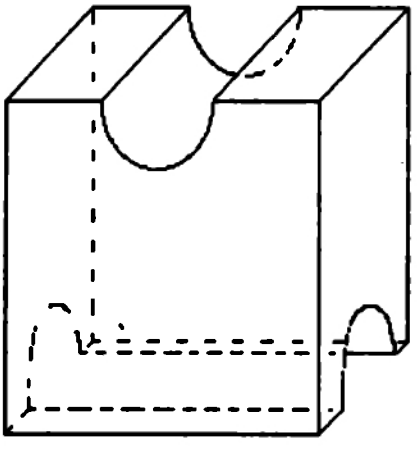


Figura 4.2. Cubo con dos túneles.

INFORMACION PRODUCIDA PARA EL EJEMPLO 2

Haciendo plano entre entidades 10 y 11
Haciendo plano entre entidades 10 y 13
Haciendo plano entre entidades 10 y 18
Haciendo plano entre entidades 10 y 20
Haciendo plano entre entidades 11 y 12
Haciendo plano entre entidades 11 y 18
Haciendo plano entre entidades 12 y 13
Haciendo plano entre entidades 13 y 20
Haciendo plano entre entidades 14 y 15
Haciendo plano entre entidades 14 y 17
Haciendo plano entre entidades 15 y 16
Haciendo plano entre entidades 15 y 21
Haciendo plano entre entidades 16 y 17
Haciendo plano entre entidades 16 y 21
Haciendo plano entre entidades 16 y 23
Haciendo plano entre entidades 17 y 23
Haciendo plano entre entidades 18 y 19
Haciendo plano entre entidades 18 y 24
Haciendo plano entre entidades 19 y 24
Haciendo plano entre entidades 19 y 31
Haciendo plano entre entidades 20 y 25
Haciendo plano entre entidades 20 y 29
Haciendo plano entre entidades 21 y 22
Haciendo plano entre entidades 21 y 24
Haciendo plano entre entidades 22 y 24
Haciendo plano entre entidades 22 y 31
Haciendo plano entre entidades 23 y 25
Haciendo plano entre entidades 23 y 28
Haciendo plano entre entidades 25 y 28
Haciendo plano entre entidades 25 y 29
Haciendo plano entre entidades 28 y 30
Haciendo plano entre entidades 29 y 30
Haciendo plano entre entidades 26 y 0
Haciendo plano entre entidades 27 y 0
Haciendo plano entre entidades 32 y 0
Haciendo plano entre entidades 33 y 0
sup. 1, parametros: z = 0.0000,0.0000,0.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}

desplazamiento : (1.0000, 0.0000, 0.0000)
ENTIDADES :
ent 10, coords = (1.0000, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
ent 11, coords = (0.0000, 0.0000) (0.0000, 0.2500) (0.0000, 0.0000)
ent 13, coords = (1.0000, 0.2500) (1.0000, 0.0000) (0.0000, 0.0000)
ent 12, coords = (0.0000, 0.2500) (1.0000, 0.2500) (0.0000, 0.0000)
ent 14, coords = (1.0000, 0.7500) (0.0000, 0.7500) (0.0000, 0.0000)
ent 15, coords = (0.0000, 0.7500) (0.0000, 1.0000) (0.0000, 0.0000)
ent 17, coords = (1.0000, 1.0000) (1.0000, 0.7500) (0.0000, 0.0000)
ent 16, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)

sup. 2, parametros: y = -0.0000,0.0000,-0.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)
ENTIDADES :
ent 10, coords = (1.0000, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
ent 18, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
ent 20, coords = (1.0000, 1.0000) (1.0000, 0.0000) (0.0000, 0.0000)
ent 19, coords = (0.0000, 1.0000) (0.2500, 1.0000) (0.0000, 0.0000)
ent 29, coords = (1.0000, 1.0000) (0.7500, 1.0000) (0.0000, 0.0000)
ent 32, coords = (0.2500, 1.0000) (0.7500, 1.0000) (0.5000, 1.0000)

sup. 3, parametros: x = -0.0000,-0.0000,1.0000
base = {(0.0000, 1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)
ENTIDADES :
ent 11, coords = (0.0000, 0.0000) (0.2500, 0.0000) (0.0000, 0.0000)
ent 18, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
ent 15, coords = (0.7500, 0.0000) (1.0000, 0.0000) (0.0000, 0.0000)
ent 21, coords = (1.0000, 0.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 24, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 26, coords = (0.2500, -0.0000) (0.7500, 0.0000) (0.5000, -0.0000)

sup. 4, parametros: x = -0.0000,-0.0000,0.0000
base = {(0.0000, 1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 0.0000, 0.0000)
ENTIDADES :
ent 13, coords = (0.2500, 0.0000) (0.0000, 0.0000) (0.0000, 0.0000)
ent 20, coords = (0.0000, 1.0000) (0.0000, 0.0000) (0.0000, 0.0000)
ent 17, coords = (1.0000, 0.0000) (0.7500, 0.0000) (0.0000, 0.0000)
ent 23, coords = (1.0000, 1.0000) (1.0000, 0.0000) (0.0000, 0.0000)
ent 25, coords = (0.0000, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)

ent 27, coords = (0.7500, 0.0000) (0.2500, 0.0000) (0.5000, 0.0000)

sup. 5, parametros: y = -0.0000,0.0000,1.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 1.0000, 0.0000)
ENTIDADES :
ent 16, coords = (0.0000, 0.0000) (1.0000, 0.0000) (0.0000, 0.0000)
ent 21, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
ent 23, coords = (1.0000, 1.0000) (1.0000, 0.0000) (0.0000, 0.0000)
ent 22, coords = (0.0000, 1.0000) (0.2500, 1.0000) (0.0000, 0.0000)
ent 28, coords = (0.7500, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 33, coords = (0.7500, 1.0000) (0.2500, 1.0000) (0.5000, 1.0000)

sup. 6, parametros: z = 0.0000,0.0000,1.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 1.0000)
ENTIDADES :
ent 19, coords = (0.0000, 0.0000) (0.2500, 0.0000) (0.0000, 0.0000)
ent 24, coords = (0.0000, 0.0000) (0.0000, 1.0000) (0.0000, 0.0000)
ent 31, coords = (0.2500, 0.0000) (0.2500, 1.0000) (0.0000, 0.0000)
ent 22, coords = (0.0000, 1.0000) (0.2500, 1.0000) (0.0000, 0.0000)
ent 25, coords = (1.0000, 0.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 28, coords = (0.7500, 1.0000) (1.0000, 1.0000) (0.0000, 0.0000)
ent 29, coords = (1.0000, 0.0000) (0.7500, 0.0000) (0.0000, 0.0000)
ent 30, coords = (0.7500, 0.0000) (0.7500, 1.0000) (0.0000, 0.0000)

sup. 1, parametros: z = 0.0000,0.0000,0.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)
CICLOS :

ciclo 1 (-), ENTS: 10 (+), 11 (+), 12 (+), 13 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 0.2500

ciclo 2 (-), ENTS: 14 (+), 15 (+), 16 (+), 17 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.7500, ymax : 1.0000

sup. 2, parametros: y = -0.0000,0.0000,-0.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)

CICLOS :

ciclo 1 (-), ENTS: 10 (+), 18 (+), 19 (+), 32 (+), 29 (-), 20 (+),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.2500

sup. 3, parametros: x = -0.0000,-0.0000,1.0000
base = {(0.0000, 1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 0.0000, 0.0000)
CICLOS :

ciclo 1 (+), ENTS: 11 (+), 26 (+), 15 (+), 21 (+), 24 (-), 18 (-),
xmin : 0.0000, xmax : 1.0000, ymin : -0.2500, ymax : 1.0000

sup. 4, parametros: x = -0.0000,-0.0000,0.0000
base = {(0.0000, 1.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (0.0000, 0.0000, 0.0000)
CICLOS :

ciclo 1 (-), ENTS: 13 (+), 20 (-), 25 (+), 23 (+), 17 (+), 27 (+),
xmin : 0.0000, xmax : 1.0000, ymin : -0.2500, ymax : 1.0000

sup. 5, parametros: y = -0.0000,0.0000,1.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 0.0000, 1.0000)}
desplazamiento : (1.0000, 1.0000, 0.0000)
CICLOS :

ciclo 1 (+), ENTS: 16 (+), 23 (-), 28 (-), 33 (+), 22 (-), 21 (-),
xmin : 0.0000, xmax : 1.0000, ymin : 0.0000, ymax : 1.2500

sup. 6, parametros: z = 0.0000,0.0000,1.0000
base = {(-1.0000, 0.0000, 0.0000) , (0.0000, 1.0000, 0.0000)}
desplazamiento : (1.0000, 0.0000, 1.0000)
CICLOS :

ciclo 1 (+), ENTS: 19 (+), 31 (+), 22 (-), 24 (-),
xmin : 0.0000, xmax : 0.2500, ymin : 0.0000, ymax : 1.0000

ciclo 2 (+), ENTS: 25 (+), 28 (-), 30 (-), 29 (-),
xmin : 0.7500, xmax : 1.0000, ymin : 0.0000, ymax : 1.0000

Entidades :
Entidad (10 2 0) 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000

en superf. (1 2) Angulo (1.5708), Vertices (1 2)
Entidad (11 2 0) 1.0000 0.0000 0.0000 1.0000 0.2500 0.0000
en superf. (1 3) Angulo (1.5708), Vertices (2 3)
Entidad (12 2 0) 1.0000 0.2500 0.0000 0.0000 0.2500 0.0000
en superf. (1 -1) Angulo (1.5708), Vertices (3 4)
Entidad (13 2 0) 0.0000 0.2500 0.0000 0.0000 0.0000 0.0000
en superf. (1 4) Angulo (1.5708), Vertices (4 1)
Entidad (14 2 0) 0.0000 0.7500 0.0000 1.0000 0.7500 0.0000
en superf. (1 -1) Angulo (1.5708), Vertices (5 6)
Entidad (15 2 0) 1.0000 0.7500 0.0000 1.0000 1.0000 0.0000
en superf. (1 3) Angulo (1.5708), Vertices (6 7)
Entidad (16 2 0) 1.0000 1.0000 0.0000 0.0000 1.0000 0.0000
en superf. (1 5) Angulo (1.5708), Vertices (7 8)
Entidad (17 2 0) 0.0000 1.0000 0.0000 0.0000 0.7500 0.0000
en superf. (1 4) Angulo (1.5708), Vertices (8 5)
Entidad (18 2 0) 1.0000 0.0000 0.0000 1.0000 0.0000 1.0000
en superf. (2 3) Angulo (1.5708), Vertices (2 9)
Entidad (19 2 0) 1.0000 0.0000 1.0000 0.7500 0.0000 1.0000
en superf. (2 6) Angulo (1.5708), Vertices (9 10)
Entidad (20 2 0) 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
en superf. (2 4) Angulo (1.5708), Vertices (11 1)
Entidad (21 2 0) 1.0000 1.0000 0.0000 1.0000 1.0000 1.0000
en superf. (3 5) Angulo (1.5708), Vertices (7 12)
Entidad (22 2 0) 1.0000 1.0000 1.0000 0.7500 1.0000 1.0000
en superf. (5 6) Angulo (1.5708), Vertices (12 13)
Entidad (23 2 0) 0.0000 1.0000 1.0000 0.0000 1.0000 0.0000
en superf. (5 4) Angulo (1.5708), Vertices (14 8)
Entidad (24 2 0) 1.0000 0.0000 1.0000 1.0000 1.0000 1.0000
en superf. (3 6) Angulo (1.5708), Vertices (9 12)
Entidad (25 2 0) 0.0000 0.0000 1.0000 0.0000 1.0000 1.0000
en superf. (4 6) Angulo (1.5708), Vertices (11 14)
Entidad (26 3 6) -0.5000 0.0000 -1.0000 0.2500 0.0000 3.1416
en superf. (3 -1) Angulo (1.5708), Vertices (3 6)
Entidad (27 3 5) 0.5000 0.0000 0.0000 0.2500 0.0000 3.1416
en superf. (4 -1) Angulo (1.5708), Vertices (5 4)
Entidad (28 2 0) 0.2500 1.0000 1.0000 0.0000 1.0000 1.0000
en superf. (5 6) Angulo (1.5708), Vertices (15 14)
Entidad (29 2 0) 0.0000 0.0000 1.0000 0.2500 0.0000 1.0000
en superf. (2 6) Angulo (1.5708), Vertices (11 16)
Entidad (30 2 0) 0.2500 0.0000 1.0000 0.2500 1.0000 1.0000
en superf. (6 -2) Angulo (0.6435), Vertices (16 15)
Entidad (31 2 0) 0.7500 0.0000 1.0000 0.7500 1.0000 1.0000
en superf. (6 -2) Angulo (0.0000), Vertices (10 13)
Entidad (32 3 2) 0.5000 1.0000 0.0000 0.2500 0.0000 3.1416
en superf. (2 -2) Angulo (1.5708), Vertices (10 16)

Entidad (33 3 3) -0.5000 1.0000 1.0000 0.2500 0.0000 3.1416
en superf. (5 -2) Angulo (1.5708), Vertices (15 13)
vista 1:
{ (1.0000 0.0000 0.0000), (0.0000 1.0000 0.0000), (0.0000, 0.0000, 1.0000) }
vista 2:
{ (1.0000 0.0000 0.0000), (0.0000 0.0000 1.0000), (0.0000, -1.0000, 0.0000) }
vista 3:
{ (-1.0000 0.0000 0.0000), (0.0000 0.0000 1.0000), (0.0000, 1.0000, 0.0000) }
vista 4:
{ (-1.0000 0.0000 0.0000), (0.0000 1.0000 0.0000), (0.0000, 0.0000, -1.0000) }
vista 5:
{ (0.0000 1.0000 0.0000), (0.0000 0.0000 1.0000), (1.0000, 0.0000, 0.0000) }
vista 6:
{ (0.0000 -1.0000 0.0000), (0.0000 0.0000 1.0000), (-1.0000, 0.0000, 0.0000) }
vista 7:
{ (0.7071 0.7071 0.0000), (-0.4082 0.4082 0.8165), (0.5774, -0.5774, 0.5773) }
vista 8:
{ (0.5000 -0.8336 0.1464), (0.5000 0.1464 -0.8536), (0.7071, 0.5000, 0.5000) }

vertice 1, (0.0000, 0.0000, 0.0000)
Entidades (id, side, tipo): (10, 1, 2) (13, 2, 2) (20, 2, 2)
vertice 2, (1.0000, 0.0000, 0.0000)
Entidades (id, side, tipo): (10, 2, 2) (11, 1, 2) (18, 1, 2)
vertice 3, (1.0000, 0.2500, 0.0000)
Entidades (id, side, tipo): (11, 2, 2) (12, 1, 2) (26, 1, 3)
vertice 4, (0.0000, 0.2500, 0.0000)
Entidades (id, side, tipo): (12, 2, 2) (13, 1, 2) (27, 2, 3)
vertice 5, (0.0000, 0.7500, 0.0000)
Entidades (id, side, tipo): (14, 1, 2) (17, 2, 2) (27, 1, 3)
vertice 6, (1.0000, 0.7500, 0.0000)
Entidades (id, side, tipo): (14, 2, 2) (15, 1, 2) (26, 2, 3)
vertice 7, (1.0000, 1.0000, 0.0000)
Entidades (id, side, tipo): (15, 2, 2) (16, 1, 2) (21, 1, 2)
vertice 8, (0.0000, 1.0000, 0.0000)
Entidades (id, side, tipo): (16, 2, 2) (17, 1, 2) (23, 2, 2)
vertice 9, (1.0000, 0.0000, 1.0000)
Entidades (id, side, tipo): (18, 2, 2) (19, 1, 2) (24, 1, 2)
vertice 10, (0.7500, 0.0000, 1.0000)
Entidades (id, side, tipo): (19, 2, 2) (31, 1, 2) (32, 1, 3)
vertice 11, (0.0000, 0.0000, 1.0000)
Entidades (id, side, tipo): (20, 1, 2) (25, 1, 2) (29, 1, 2)
vertice 12, (1.0000, 1.0000, 1.0000)
Entidades (id, side, tipo): (21, 2, 2) (22, 1, 2) (24, 2, 2)
vertice 13, (0.7500, 1.0000, 1.0000)
Entidades (id, side, tipo): (22, 2, 2) (31, 2, 2) (33, 2, 3)

vertice 14, (0.0000, 1.0000, 1.0000)
Entidades (id, side, tipo): (23, 1, 2) (25, 2, 2) (28, 2, 2)
vertice 15, (0.2500, 1.0000, 1.0000)
Entidades (id, side, tipo): (28, 1, 2) (30, 2, 2) (33, 1, 3)
vertice 16, (0.2500, 0.0000, 1.0000)
Entidades (id, side, tipo): (29, 2, 2) (30, 1, 2) (32, 2, 3)
Orden de planos :
Plano 1 (-0.0000, -0.0000 1.0000)
Id: 1, Prof.: 0
Id: 2, Prof.: 0
Plano 2 (-0.0000, -1.0000 0.0000)
Id: 1, Prof.: 0
Plano 3 (1.0000, 0.0000 0.0000)
Id: 1, Prof.: 0
Plano 4 (1.0000, 0.0000 0.0000)
Id: 1, Prof.: 0
Plano 5 (0.0000, 1.0000 -0.0000)
Id: 1, Prof.: 0
Plano 6 (-0.0000, -0.0000 1.0000)
Id: 1, Prof.: 0
Id: 2, Prof.: 0

TUNELES
Tunel -1, tipo 2 Ents a1 26, a2 27, l1 12, l2 14 signos -, -, +, +. Vertices 6 3 4 5
Tunel -2, tipo 2 Ents a1 32, a2 33, l1 31, l2 30 signos -, -, +, -. Vertices 16 10 13 15

TUNELES

VERTICES:
vertice 1, (0.0000, 0.0000, 0.0000) tipo 0 carac 0
vertice 2, (1.0000, 0.0000, 0.0000) tipo 0 carac 0
vertice 3, (1.0000, 0.2500, 0.0000) tipo 6 carac 1
vertice 4, (0.0000, 0.2500, 0.0000) tipo 6 carac 1
vertice 5, (0.0000, 0.7500, 0.0000) tipo 6 carac 1
vertice 6, (1.0000, 0.7500, 0.0000) tipo 6 carac 1
vertice 7, (1.0000, 1.0000, 0.0000) tipo 0 carac 0
vertice 8, (0.0000, 1.0000, 0.0000) tipo 0 carac 0
vertice 9, (1.0000, 0.0000, 1.0000) tipo 0 carac 0
vertice 10, (0.7500, 0.0000, 1.0000) tipo 0 carac 0
vertice 11, (0.0000, 0.0000, 1.0000) tipo 0 carac 0
vertice 12, (1.0000, 1.0000, 1.0000) tipo 0 carac 0
vertice 13, (0.7500, 1.0000, 1.0000) tipo 0 carac 0
vertice 14, (0.0000, 1.0000, 1.0000) tipo 0 carac 0
vertice 15, (0.2500, 1.0000, 1.0000) tipo 0 carac 0
vertice 16, (0.2500, 0.0000, 1.0000) tipo 0 carac 0

CARACTERISTICAS:
1 tunel 0.0000 1.0000 0.2500 0.7500 0.0000 0.2500
2 tunel 0.2500 0.7500 0.0000 1.0000 0.7500 1.0000

ORDEN:
1 tunel 2
2 tunel 1

§ 5 Conclusiones.

La tendencia de los sistemas computarizados para manufactura, es integrase en un solo concepto: CIM, que algunos autores categorizan más como filosofía que como sistema computacional.

En los últimos años se han visto cambios que se pensaban eran historias de ciencia ficción, cambios tecnológicos, políticos y sociales. México no ha sido la exepción y se ve involucrado en una economía globalizada, luego de haber vivido en un esquema paternalista. Las empresas tienen que modernizarse y utilizar los esquemas de producción globales, que primero fueran la particularización de la producción. Este tipo de producción tiene las siguientes características:

a) Variedad de productos y procesos.
b) Complejidad de capacidad productiva -demanda dinámica-
c) Planeación incierta.
d) Dificultad para planear la producción y calendarizarla.
f) Control de producción dinámico.

En otras palabras, estamos hablando de un esquema dinámico de producción, para conservar su adaptividad los sistemas de manufactura dependen cada vez más de su capacidad de respuesta.

Para incrementar la velocidad de respuesta, es necesario combinar el estudio de técnicas que respondan al cambio, con el poder de sistemas de apoyo automáticos y asertivos.

Entre las técnicas encontramos el Diseño para la Manufacturabilidad, que sistemiza el concepto de diseño en fabricación. Esto es deseable pero difícil de lograr humanamente. Si Houtzeel reporta una déficit de expertos de producción, imaginemos, la de expertos de producción que dominen técnicas de diseño, además la generación de diferencias en el diseño se convertiría en un nuevo problema. Sin embargo, un diseñador puede observar de inmediato los resultados de su idea -ingeniería concurrente-, podrá mejorar su diseño sin salir de su área de trabajo.

El diseño para la manufactura, es una necesidad para lograr competitividad, y para esto se necesita de la automatización de la generación de planes de procesos.

En la realización de este trabajo, involucramos técnicas para la obtención de características topológicas a partir de un modelo geométrico. Se propusieron algoritmos, algunas ampliaciones de otros existentes, otros basándose en técnicas heurísticas y otros en técnicas analíticas, explicados con detalle en los capítulos 2 y 3.

La obtención de el conocimiento topológico de una pieza a partir de un modelo de armazón, es posible aunque no se tiene conocimiento de una técnica perfecta. Es una labor de relacionar exhaustivamente las características de las entidades para inferir superficies, y su posición espacial relativa.

Como en un modelo de armazón no se tiene conocimiento de superficies, no es posible asignarles atributos como el terminado requerido, la obtención de esta información se hace en medios interactivos por medio de un diálogo, o en forma de lote por medio de lenguajes de descripción, lo que representa una limitante para la reducción en la intervención humana entre el CAD, la definición del método de fabricación y el CAPP.

Las desventajas que observa este sistema es una explosión geométrica de los tipos de superficies entre más entidades consideremos, y no se plantea un esquema para manejar superficies complejas. Sin embargo, no es posible pasar inadvertida la tendencia al diseño en tres dimensiones. Si se habla de Diseño para Manufactura, la tendencia es que se tenga un modelo compatible tanto para diseño como para manufactura, es decir un modelo que sea manejable tanto gráfica como simbólicamente. Los enfoques primitivos para diseño en tres dimensiones son técnicas de CSG, y de hecho se encuentran en paquetes comerciales, pero requieren de tantos recursos matemáticos y gráficos, que hoy día se presentan mas bien como sistemas para procesar sólidos creados con técnicas bidimensionales, aunque no está lejano el día en que sean comerciales.

En el apéndice A se encuentran los listados de las rutinas utilizadas en lenguaje C, y en el apèndice B se encuentra informaciòn ampliada sobre tòpicos concernientes a este trabajo.

APENDICE A. Listado de los algoritmos implantados en lenguaje C.

El programa de implantación, fué diseñado en varios módulos, cada uno de ellos está en un archivo diferente, por esto se declaran externas las funciones que se van a utilizar de otros módulos. Lo que está escrito en negritas no es parte del programa, son comentarios que facilitan su entendimiento y no se encontrarán en el archivo de texto.

Encabezados.

Estructuras.

Este módulo define las construcciones en memoria necesarias para definir entidades, planos, superficies, vértices, las estructuras de datos y bases de conocimientos que requiere el sistema.

```
                    /* definicion de estructuras */

typedef struct r2           /* estructura para un vector en dos dimensiones */
{
  double x,y;
} r2;

typedef struct r3           /* estructura para un vector en tres dimensiones */
{
  double x,y,z;
} r3;

typedef struct entinver     /* estructura para entidades en lista de vertices */
{
  int id, side, tipo;
  struct entinver *sigentinver;
} entinver;

typedef struct vertex       /* estructura para lista de vertices */
{
  int id;
  r3 punto;
  entinver *content;
  struct vertex *sig_ver;
} vertex;
```

```
typedef struct vista              /* estructura para las vistas de cadkey */
{
  int id_vis;
  r3 v1,v2,v3;
  struct vista *sig_vis;
} vista;

typedef struct entity        /* estructura de la lista global de entidades  */
{                        /* contiene toda la informacion */
  int tipo_ent;
  int id_ent;
  int vis_ent;
  r3 p1,p2;
  double ang;                    /* angulo que genera entre sup1 y sup2 */
  int sup1, sup2,
     ver1, ver2;
  struct entity *sig;
}entity;


typedef struct simplent       /* Estructura simplificada de entidades que se */
{                        /* enlazara a la estructura plano, para indicar */
  int id, tipo;            /* las entidades que lo conforman  */
  r2 coord1,coord2,coord3;
  char sign;
  struct simplent *sig_ent;
}simplent;

typedef struct ciclo          /* estructura de un ciclo con sus entidades */
{
  int id;                  /* y enlazado al plano a que pertenece     */
  int prof;                /* tiene ciclos del mismo nivel y de menor  */
  char sign;                     /* positivo si interior a la izq.de entidad */
  double xmin,xmax,
         ymin,ymax;
  struct ciclo *sig_ciclo;    /* es decir hermanos e hijos, tambien      */
  struct ciclo *ciclo_int;    /* tiene las entidades que conforman al    */
  simplent *sig_ent;          /* ciclo esta lista de entidades es circular*/
}ciclo;
```

```
typedef struct plano          /* estructura que contiene la lista total de */
{
  int id;
  double a,b,c;             /* superficies o planos de la figura       */
  char despeje;            /* variable sobre la cual se normaliza x,y,z */
  r3 dir;                 /* tiene dos apuntadores uno a la siguiente  */
  struct plano *sig_sup;     /* superficie y otro a la lista de entidades */
  simplent *sig_ent;        /* que tiene ese plano                     */
  ciclo *sig_ciclo;        /* apuntador al ciclo                      */
  r3  v1,v2,delta;              /* Base Ortonormal & desplazamiento          */
}plano;

typedef struct tunel             /* estructura que contiene lista tuneles */
{
  int tipo,
     id,                  /* Id del tunel, y de arco1,arco2, linea1 y 2*/
     idarc1,
     idarc2,
     idlin1,
     idlin2,
     interior;           /* bandera 0 cuando el interior esta hacia el centro*/
  char sgna1, sgna2,             /* signos de direccion */
       sgnl1, sgnl2;
  int v1, v2, v3, v4;        /* p1 & p2 conectan arc1 con las lineas */
  struct tunel *sig;             /* p3 & p4 conectan arc2 con las lineas */
}tunel;

typedef struct cilindro          /* estructura que contiene lista cilindros */
{
  int tipo, id,           /* Id del cilindro y de circ1,circ2, linea1 */
     idcir1,
     idcir2,
     idlin1,
     interior;           /* 0 si el interior esta hacia dentro, 2 haca fuera*/
  char sgnl1;                    /* signos de direccion */
  int v1, v2;                    /* centros y puntos de contacto */
  struct cilindro *sig;
}cilindro;
```

```
typedef struct ttver            /* tabla de tipo de vertices */
{
  int id,flag,tipo12,tipo23,tipo31;  /*tipos de entidades que unen a las sups*/
  r2  ang1, ang2, ang3;              /* x <- coord min, y <- coord max */
  struct ttver *sig;
}ttver;

typedef struct tcmaq            /* tabla de caract. de maquinado */
{
  int   id, tipo, liga1, liga2, liga3;
  struct tcmaq *sig;
}tcmaq;

typedef struct trcmaq  /* tabla relacional de caract. de maquinado */
{
  int   id;
  tcmaq *carac;
  struct trcmaq *sig;
}trcmaq;

typedef struct listafin  /*tabla con las características reconocidas */
{
  int   id, tipo;
  double xmin, xmax, ymin, ymax;
  struct listafin *sig;
}listafin;
```

Encabezado que define rutinas de álgebra lineal.

```
                    /* DEFINICION DE RUTINAS DE ALGEBRA LINEAL */

#define sqr(x) ((x) * (x))
extern int exor();              /* exor entre dos expresiones */

extern double dotprod2();           /* producto punto en r2 */

extern double dotprod3();           /* producto punto en r3 */

extern double magn2();              /* magnitud de un vector r2 */

extern double magn3();              /* magnitud de un vector r2 */
```

```
extern r2 add2();                    /* suma de dos vectores en  r2 */

extern r3 add3();                    /* suma de dos vectores en  r3 */

extern r2 sub2();                    /* resta de dos vectores en  r2 */

extern r3 sub3();                    /* resta de dos vectores en  r3 */

extern r2 escprod2();          /* producto escalar en dos dimensiones */

extern r3 escprod3();          /* producto escalar en tres dimensiones */

extern r3 crossprod();         /* producto cruz */

extern double det3();          /* determinante de una matriz de 3 vectores*/

extern double det2();

extern int equal1() ;

extern int equal2() ;

extern int equal3 ();          /* Compara si dos vectores tridimensionales */

extern void asigna2 ();              /* Asigna valores a vector bidimensional */

extern void asigna3 ();        /* Asigna valor a un vector tridimensional */

extern r3 plrv_w();            /* convierte coord. polares de vista a coord. mundo*/

extern r3 vis_wrd();           /* convierte un vector en coord vista a mundo */

extern r2 proj_ort();          /* Proyecta ortogonalmente un vector sobre otros dos*/
                               /* luego de restarle un desplazamiento */
```

Encabrzado que define las tolerancias permitidas, y a π

```
#define EPS_COP 0.000001    /* tolerancia para coplanaridad */
#define EPS_PT3 0.000001    /* tolerancia puntual */
#define EPS_PT2 0.000001
#define EPS_PT1 0.000001

#define PI 3.14159265358787
```

Módulo Principal.

Este módulo enlaca la información que proviene de los demás, y la conserva con la cabeza de cada lista.

```
#include <c:\pitufo\estruc.hdr>
#include <stdio.h>
#include <stdlib.h>
```

Funciones que utiliza de otros módulos.

```
extern entity   *crealent();
extern vista    *crealvis();
extern vertex   *crealver();
extern plano    *creaplanos();
extern void     agrega_ciclo();
extern void     ordenaciclos();
extern tunel    *creatunel();
extern cilindro *hazcils();
extern void     assnormal();
extern void     comp_ver();
extern trcmaq   *hazlista();
extern ttver    *knowver();
extern void     reconoce();
extern recmaq *reconoce();
extern void     ordena();

extern void muestra();
extern void muestrav();
extern void muestravtx();
extern void muestraplano();
extern void muestra2plano();
extern void muestra3plano();
extern void muestratunel();
extern void muestracil();
```

```
main()
{
```

Declaración de listas de encabezado para entidades, vistas, vértices, superficies planas, túneles rectos, túneles arqueados, cilindros, bujes, características de maquinado

```
  entity   *headent;
  vista    *headvis;
  vertex   *headver;
  plano    *headplano;
  tunel    *headtunel,
           *headtarq;
  cilindro *headcil,
           *headbuje;
  trcmaq   *headcars;
  ttver    *headttver;
  recmaq   *headrec;

  FILE     *fs;
```

Apertura de archivo de salida, abajo de archivo de entrada y de archivo de vistas.

```
  fs       = fopen("c:\\pitufo\\sal3.dat","w");
                if (fs == NULL)
                {
                  printf("Error, no se puede abrir archivo de salida");
                  exit(0);
                }

  headent   = crealent("c:\\pitufo\\fig3.dat");
  headvis   = crealvis("c:\\pitufo\\vistas.dat");
  headver   = crealver(headent, headvis);
  headplano = creaplanos(headent, headvis,fs);
                muestraplano(headplano,fs);
                agrega_ciclo(headent, headplano);
                muestra2plano(headplano,fs);
                ordenaciclos(headplano, headent);
  headtunel = creatunel(headent, headver, 2);
  headtarq  = creatunel(headent, headver, 3);
  headcil   = hazcils(headent, headver, headplano, 2);
  headbuje  = hazcils(headent, headver, headplano, 3);
                comp_ver(headver, headcil);
                comp_ver(headver, headbuje);
```

```
                assnormal(headplano, headent, headtunel, headtarq, headcil, headbuje,
headver);
  headcars  = hazlista("c:\\pitufo\\cars.dat");
  headttver = knowver("c:\\pitufo\\knowver.dat");
                asignavertex(headver, headttver, headent, fs);
  headrec = reconoce(headver, headcars, headent, fs);
            ordena (headrec, headver,headent,headplano,headtunel,headtarq,headcil,hea

  muestra(headent,fs);
  muestrav(headvis,fs);
  muestravtx(headver,fs);
  muestra3plano(headplano,fs);
  muestratunel(headtunel,fs);
  muestratunel(headtarq,fs);
  muestracil(headcil,fs);
  muestracil(headbuje,fs);
  reporte(headrec, fs);
  fclose(fs);
  return 1;
}
```

Módulo que realiza las rutinas de álgebra lineal definidas en un encabezado anterior.

```
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\defines.hdr>
#include <math.h>

        /* RUTINAS DE ALGEBRA LINEAL */

#define sqr(x) (x) * (x)


int exor(a, b)                  /* exor entre a, b */
int a, b;
{
  return (a && !b) || (!a && b);
}
```

```
double dotprod2(v,w)          /* producto punto en r2 */
r2 v, w;
{
  return (v.x * w.x + v.y * w.y);
}

double dotprod3(v,w)          /* producto punto en r3 */
r3 v, w;
{
  return (v.x * w.x + v.y * w.y + v.z * w.z);
}

double magn2(r2 v)            /* magnitud de un vector r2 */
{
  return(sqrt(sqr(v.x) + sqr(v.y)));
}

double magn3(r3 v)            /* magnitud de un vector r2 */
{
  return(sqrt(sqr(v.x) + sqr(v.y) + sqr(v.z)));
}

r2 add2(v, w)                 /* suma de dos vectores en  r2 */
r2 v, w;
{
  r2 aux;

  aux.x = v.x + w.x;
  aux.y = v.y + w.y;
  return aux;
}

r3 add3(v, w)                 /* suma de dos vectores en  r3 */
r3 v, w;
{
  r3 aux;

  aux.x = v.x + w.x;
  aux.y = v.y + w.y;
  aux.z = v.z + w.z;
  return aux;
}
```

```
r2 sub2(v, w)                    /* resta de dos vectores en  r2 */
r2 v, w;
{
  r2 aux;

  aux.x = v.x - w.x;
  aux.y = v.y - w.y;
  return aux;
}

r3 sub3(v, w)                    /* resta de dos vectores en  r3 */
r3 v, w;
{
  r3 aux;

  aux.x = v.x - w.x;
  aux.y = v.y - w.y;
  aux.z = v.z - w.z;
  return aux;
}

r2 escprod2(a, v)                /* producto escalar en dos dimensiones */
double a;
r2 v;
{
  r2 aux;

  aux.x = a * v.x;
  aux.y = a * v.y;
  return aux;
}

r3 escprod3(a, v)                /* producto escalar en tres dimensiones */
double a;
r3 v;
{
  r3 aux;

  aux.x = a * v.x;
  aux.y = a * v.y;
  aux.z = a * v.z;
  return aux;
}
```

```
r3 crossprod(v, w)              /* producto punto en tres dimensiones */
r3 v, w;
{
  r3 z;

  z.x = v.y * w.z - w.y * v.z;
  z.y = w.x * v.z - v.x * w.z;
  z.z = v.x * w.y - w.x * v.y;

  return z;
}

double det3(v, w, z)            /* determinante de una matriz de 3 vectores */
r3 v, w, z;
{
  double r;

  r = (v.x)*((w.y)*(z.z) - (w.z)*(z.y));
  r += (v.y)*((w.z)*(z.x) - (w.x)*(z.z)) + (v.z)*((w.x)*(z.y) - (w.y)*(z.x));

  return r;
}

double det2(v,w)
r2 v, w;
{
  return (v.x * w.y) - (v.y * w.x);
}

int equal1(v,w)
double v,w;
{
  return (sqr(v-w) < sqr(EPS_PT1));
}

int equal2(v,w)
r2 v,w;
{
  return((sqr(v.x - w.x) + sqr(v.y - w.y)) <= sqr(EPS_PT2));
}
```

```
int equal3 (v,w)                /* Compara si dos vectores tridimensionales */
r3 v,w;                         /* son iguales o no                         */
{
  return ((sqr(v.x - w.x) + sqr(v.y - w.y) + sqr(v.z - w.z)) <= sqr(EPS_PT3));
}

void asigna2 (v,a,b)            /* Asigna valores a vector bidimensional */
r2 *v;
double a,b;
{
   v->x = a;
   v->y = b;
}

void asigna3 (v, a, b, c)       /* Asigna valor a un vector tridimensional */
r3 *v;
double a,b,c;
{
   v->x = a;
   v->y = b;
   v->z = c;
}

r2 proj_ort(v1, v2, vect, delta) /* Proyecta vect - delta sobre v1, v2 */
r3 v1, v2, vect, delta;
{
  r2 res;
  res.x = dotprod3(sub3(vect, delta), v1);
  res.y = dotprod3(sub3(vect, delta), v2);
  return res;
}

r3 plrv_w(v1, v2, v3, r, t, c)  /* Convierte coord polares de vista a de mundo*/
r3 v1,v2,v3,c;                  /* Vectores base de vista y centro */
double r,t;                     /* radio y angulo */

{
  return (add3 (add3 (escprod3 (r * cos(t) + c.x, v1),
                      escprod3 (r * sin(t) + c.y, v2)),
                escprod3 (c.z, v3)));
}
```

Módulo para la lectura de entidades en un archivo de texto para la formación de listas de entidades, de vistas y con esta información generar una de vértices.

```
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <c:\pitufo\defines.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern error();

entity *crealent(noment)        /* crea lista de entidades */
char noment[];
{
  entity *crea_nodo();          /*crea e inicializa nodo de lista de entidades */
  void mete_lista();            /* agrega un nodo a la lista de entidades */
  void sepcirdear();          /* separa circulos de arcos */

  int tipo,id,vis;
  double f1,f2,f3,f4,f5,f6;
  entity *headent, *actent;
  FILE *fileptr;

  fileptr = fopen(noment,"r");
  if (fileptr == NULL)
    error("El archivo de entidades no se pudo abrir");
  headent = crea_nodo(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
  if (headent == NULL)
    error("Memoria insuficiente creacion de lista entidades");
  do                        /* rutina que lee linea por linea */
  {
 fscanf(fileptr,"%d%d%d%lf%lf%lf%lf%lf%lf",&tipo,&id,&vis,&f1,&f2,&f3,&f4,&f5,&f6);
    actent = crea_nodo(tipo,id,vis,f1,f2,f3,f4,f5,f6);
    if (actent == NULL)
        error("Memoria insuficiente creacion de lista entidades");
    mete_lista(headent,actent);
  } while (!feof(fileptr));
  fclose(fileptr);
  sepcirdear(headent);
  return headent;
}
```

```
vista *crealvis(nomvis)          /* crea lista de vistas */
char nomvis[];
{
  vista *crea_nvis();            /*crea e inicializa nodo de lista de vistas */
  void mete_lvis();              /* agrega un nodo a la lista de vistas */

  int id;
  double f1,f2,f3,f4,f5,f6,f7,f8,f9;
  vista *head, *act;
  FILE *fileptr;

  fileptr = fopen(nomvis,"r");
  if (fileptr == NULL)
    error("El archivo de vistas no se pudo abrir");
  head = crea_nvis(0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
  if (head == NULL)
    error("Memoria insuficiente creacion de lista vistas");
  do                             /* rutina que lee linea por linea */
  {
 fscanf(fileptr,"%d%lf%lf%lf%lf%lf%lf%lf%lf%lf",&id,&f1,&f2,&f3,&f4,&f5,&f6,&f7,&f8,&f9);
    act = crea_nvis(id,f1,f2,f3,f4,f5,f6,f7,f8,f9);
    if (act == NULL)
        error("Memoria insuficiente creacion de lista vistas");
    mete_lvis(head,act);
  } while (!feof(fileptr));
  fclose(fileptr);
  return head;
}

vertex *crealver(headent, headvis)    /* Crea lista de vertices */
entity *headent;
vista *headvis;

{
  vista *busvista();    /* devuelve la vista dado su id */
  r3 plrv_w();                  /* convierte de polar en vista a mundo */
  void insentinv();             /* inserta entidad en vertice */
  vertex *add_ver();            /* verifica los vertices existentes y agrega */

  vertex *headver;
  entity *axent;
  vista *axvis;
  r3 parc;
```

```
  headver = (vertex *)malloc(sizeof(vertex));
  if (headver == NULL)
    error("Falta de memoria en creacion de cabeza de vertices");
  headver->content = NULL;
  headver->sig_ver = NULL;
  asigna3(&headver->punto,0.0,0.0,0.0);
  headver->id = 0;

  for (axent = headent->sig; axent != NULL; axent = axent->sig)
  {
    if(axent->tipo_ent == 3)
    {           /* Convirtiendo e insertando punto inic. arco */
        axvis = busvista(headvis, axent->vis_ent);
        parc = plrv_w(axvis->v1, axvis->v2, axvis->v3,
                    axent->p2.x, axent->p2.y, axent->p1);
        insentinv(add_ver(headver, parc), axent, 1);
                            /* Conv. e ins. punto final arco */
        parc = plrv_w(axvis->v1, axvis->v2, axvis->v3,
                    axent->p2.x, axent->p2.y + axent->p2.z, axent->p1);
        insentinv(add_ver(headver, parc), axent, 2);
    }
    else if(axent->tipo_ent == 2)
          {                     /* Insertando punto inic. y fin. de linea */
            insentinv(add_ver(headver, axent->p1), axent, 1);
            insentinv(add_ver(headver, axent->p2), axent, 2);
          }
  }
  return headver;
}
              /****************/

entity *crea_nodo(t, id, vis, x1, y1, z1, x2, y2, z2)   /*crea nodo entidades */
int t,id;
double x1,y1,z1,x2,y2,z2;
{
  entity *nueva_ent;

  nueva_ent = (entity *) malloc(sizeof(entity));
  if (nueva_ent == NULL)
```

```
    return NULL;
  nueva_ent->tipo_ent = t;
  nueva_ent->id_ent = id;
  nueva_ent->vis_ent = vis;
  asigna3(&nueva_ent->p1, x1, y1, z1);
  asigna3(&nueva_ent->p2, x2, y2, z2);
  nueva_ent->sup1 = 0;
  nueva_ent->sup2 = 0;
  nueva_ent->ang = 0;
  nueva_ent->ver1 = 0;
  nueva_ent->ver2 = 0;
  nueva_ent->sig = NULL;
  return nueva_ent;
}

void mete_lista(entity *lista, entity *nuevo_elemento)
{
  entity *aux;

  for(aux = lista; aux ->sig != NULL; aux = aux->sig);
  aux->sig = nuevo_elemento;
}

void sepcirdear(entity *head)          /* Separa circulos de arco, dando un id de 30 */
{
  entity *aux;

  for (aux = head; aux != NULL; aux = aux->sig)
    if (aux->tipo_ent == 3 && equal1(aux->p2.z, 2 * PI))
        aux->tipo_ent = 30;
}


              /*******************/

vista *crea_nvis(id, x1, x2, x3, y1, y2, y3, z1, z2, z3)   /*crea nodo entidades */
int id;
double x1, y1, z1, x2, y2, z2, x3, y3, z3;
{
  vista *nueva_vis;

  nueva_vis = (vista *) malloc(sizeof(vista));
  if (nueva_vis == NULL)
    return NULL;
  nueva_vis->id_vis = id;
```

```
  asigna3(&nueva_vis->v1, x1, y1, z1);
  asigna3(&nueva_vis->v2, x2, y2, z2);
  asigna3(&nueva_vis->v3, x3, y3, z3);
  nueva_vis->sig_vis = NULL;
  return nueva_vis;
}

void mete_lvis(vista *head, vista *nuevo_elemento)
{
  vista *aux;

  for(aux = head; aux->sig_vis != NULL; aux = aux->sig_vis);
  aux->sig_vis = nuevo_elemento;
}

                /***************/


vista *busvista(vista *headvis, int id)
{
  vista *aux_vis;
  for (aux_vis = headvis;
        aux_vis != NULL && aux_vis->id_vis != id;
        aux_vis = aux_vis->sig_vis);
  if (aux_vis == NULL)
  {
    printf(" Error!, vista %d no esta en archivo de vistas ",id);
    exit(0);
  }
  return aux_vis;
}

vertex *add_ver(headver, punto)      /* devuelve el apuntador al vertice, y lo crea */
vertex *headver;
r3 punto;
{                                    /* si no existe */
  vertex *nuevtx, *aux_ver;
  int i;

  for(i=1, aux_ver=headver; aux_ver->sig_ver != NULL;
                                    i++, aux_ver = aux_ver->sig_ver)
    if (equal3(aux_ver->sig_ver->punto, punto))
        break;
```

```
  if (aux_ver->sig_ver == NULL)
  {
    nuevtx = (vertex *) malloc (sizeof(vertex));
    if (nuevtx == NULL)
        error("Error !, falta de memoria en creacion lista de vertices ");
    nuevtx->content = NULL;
    nuevtx->sig_ver = NULL;
    nuevtx->punto = punto;
    nuevtx->id = i;
    aux_ver->sig_ver = nuevtx;
  }
  aux_ver = aux_ver->sig_ver;
  return aux_ver;
}


void insentinv(vert, ent, side)          /* Inserta un nodo de entidad en la lista de vert.*/
vertex *vert;
entity *ent;
int side;                          /* de que lado esta el vertice */
{
  entinver *ultimaent, *axsupinv;

  for(ultimaent = vert->content; ultimaent != NULL; ultimaent = ultimaent-
>sigentinver)
    if(ultimaent->id == ent->id_ent && ultimaent->side == side)
        break;
  if (!(ultimaent->id == ent->id_ent && ultimaent->side == side) || ultimaent == NULL)
  {
    if (vert->content != NULL)
        for(ultimaent = vert->content; ultimaent->sigentinver != NULL;
                                        ultimaent = ultimaent->sigentinver);
    axsupinv = (entinver *) malloc (sizeof(entinver));
    if (axsupinv == NULL)
        error("Error en creacion de nodo de entidades en lista de vertices");
    axsupinv->id = ent->id_ent;
    axsupinv->side = side;
    axsupinv->tipo = ent->tipo_ent;
```

```
    axsupinv->sigentinver = NULL;
    if (vert->content == NULL)
        vert->content = axsupinv;
    else
        ultimaent->sigentinver = axsupinv;
    if(side == 1)
        ent->ver1 = vert->id;
    else
        ent->ver2 = vert->id;
  }
}
```

Módulo para la creación de superficies planas.

```
                /* MODULO QUE CREA PLANOS DADAS ENTIDADES */
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <c:\pitufo\defines.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern void error();

plano *creaplanos(headent, headvis,fs)
entity *headent;
vista *headvis;
FILE *fs;
{
  plano haz_plano();        /* a partir de dos lineas genera z = ax + by + c */
  void agrega_plano();                  /* lo agrega en lista superficies */


  entity *aux_fijo,         /* apuntan la primera y segunda entidad a comp */
         *aux_mov;

  plano *headplano;

  printf("\n\n");
  fprintf(fs,"\n\n");
  headplano = (plano *)malloc(sizeof(plano));
  if (headplano == NULL)
    error("Error en generacion de cabeza de planos");
```

```
  headplano->a = 0;
  headplano->b = 0;
  headplano->c = 0;
  headplano->despeje = '\0';
  headplano->id = 0;
  asigna3(&headplano->dir, 0.0, 0.0, 0.0);
  asigna3(&headplano->v1, 0.0, 0.0, 0.0);
  asigna3(&headplano->v2, 0.0, 0.0, 0.0);
  asigna3(&headplano->delta, 0.0, 0.0, 0.0);
  headplano->sig_sup = NULL;
  headplano->sig_ent = NULL;
  headplano->sig_ciclo = NULL;
  aux_mov = NULL;
                                /* Comparacion entre lineas para gen. plano */
  for (aux_fijo = headent->sig; aux_fijo != NULL; aux_fijo = aux_fijo->sig)
    for (aux_mov = aux_fijo->sig; aux_mov != NULL; aux_mov = aux_mov->sig)
    {
        if(aux_fijo->tipo_ent == 2 && aux_mov->tipo_ent == 2)        /* linea con
linea */
          if (equal3 (aux_fijo->p1, aux_mov->p1))
            agrega_plano(headvis, headplano, haz_plano(aux_fijo, aux_mov, 0,
headvis,fs), aux_fijo, aux_mov);
          else
            if (equal3 (aux_fijo->p1, aux_mov->p2))
                agrega_plano(headvis, headplano, haz_plano(aux_fijo, aux_mov, 1,
headvis,fs), aux_fijo, aux_mov);
            else
                if (equal3 (aux_fijo->p2, aux_mov->p1))
                  agrega_plano(headvis, headplano, haz_plano(aux_fijo, aux_mov, 2,
headvis,fs), aux_fijo, aux_mov);
                else
                  if (equal3 (aux_fijo->p2, aux_mov->p2))
                      agrega_plano(headvis, headplano, haz_plano(aux_fijo, aux_mov,
3, headvis,fs), aux_fijo, aux_mov);
    }
                                                    /* planos de circulos y arcos */
  for (aux_fijo = headent->sig; aux_fijo != NULL; aux_fijo = aux_fijo->sig)
    if (aux_fijo->tipo_ent == 3 || aux_fijo->tipo_ent == 30)
        agrega_plano(headvis, headplano, haz_plano(aux_fijo, NULL, 4, headvis,fs),
aux_fijo, NULL);
  return headplano;
}
              /*********************/

plano haz_plano(ent_a, ent_b, flag, headvis,fs)
```

```
entity *ent_a, *ent_b;          /* primera y segunda lineas */
vista *headvis;
FILE *fs;
int flag;                /* bandera que indica en que punto coinciden */
{
  void crea_base();
  vista *busvista();

  plano reg;                    /* temporal para regreso funcion */
  vista *axvis;          /* temporal para regreso de vista en circulo */

  r3 v1,v2;                     /* vectores sobre los que esta el plano */
  double a,b,c,d;               /* coeficientes del plano sin normalizar */

  printf("Haciendo plano entre entidades %d y %d\n",ent_a->id_ent, ent_b->id_ent);
  fprintf(fs,"Haciendo plano entre entidades %d y %d\n",ent_a->id_ent, ent_b->id_ent);

  switch (flag)          /* la bandera es para ver en que punto coinciden */
  {
    case 0:v1 = sub3(ent_a->p2, ent_a->p1);
           v2 = sub3(ent_b->p2, ent_a->p1);
           reg.delta = ent_a->p1;
           break;
    case 1:v1 = sub3(ent_a->p2, ent_a->p1);
           v2 = sub3(ent_b->p1, ent_a->p1);
           reg.delta = ent_a->p1;
           break;
    case 2:v1 = sub3(ent_a->p1, ent_a->p2);
           v2 = sub3(ent_b->p2, ent_a->p2);
           reg.delta = ent_a->p2;
           break;
    case 3:v1 = sub3(ent_a->p1, ent_a->p2);
           v2 = sub3(ent_b->p1, ent_a->p2);
           reg.delta = ent_a->p2;
           break;
    case 4:axvis = busvista(headvis, ent_a->vis_ent);
           v1 = axvis->v1;
           v2 = axvis->v2;
           reg.delta = vis_wrd(axvis->v1, axvis->v2, axvis->v3, ent_a->p1);
  }

                                /* calculando los coeficientes sin normalizar */
  a = v1.y * v2.z - v2.y * v1.z;
  b = v2.x * v1.z - v1.x * v2.z;
  c = v1.x * v2.y - v2.x * v1.y;
```

```
  d = a * reg.delta.x + b * reg.delta.y + c * reg.delta.z;

  if (c)                    /* normalizacion de plano, despeje a z,y ¢ x */
  {
    reg.despeje = 'z';
    reg.a = -a / c;
    reg.b = -b / c;
    reg.c =  d / c;
  }
  else
    if (b)
    {
        reg.despeje = 'y';
        reg.a = -a / b;
        reg.b = -c / b;
        reg.c =  d / b;
    }
    else
        if (a)
        {
          reg.despeje = 'x';
          reg.a = -b / a;
          reg.b = -c / a;
          reg.c =  d / a;
        }
        else
        {
          printf("Error, plano generado sobre puntos colineales");
          exit(0);
        }
  reg.id = 0;
  reg.sig_sup = NULL;
  reg.sig_ent = NULL;
  reg.v1 = v1;
  reg.v2 = v2;
  crea_base(&reg);
  return(reg);
}


void crea_base(p)           /* ortonormaliza la base con las entidades */
plano *p;

{
  r3 v3;                    /* tercer vector para verificar sentido */
```

```
  p->v1 = escprod3( (1/magn3(p->v1)) , p->v1);
  p->v2 = sub3(p->v2, escprod3( dotprod3( p->v1, p->v2), p->v1));
  p->v2 = escprod3( (1/magn3(p->v2)) , p->v2);
                              /* verifica que sea derecha -det. positivo- */
  v3 = crossprod(p->v1, p->v2);
  if (det3(p->v1, p->v2, v3) < 0)  /* la base esta invertida */
  {
    v3 = p->v1;
    p->v1 = p->v2;
    p->v2 = v3;
  }
}
                    /************************/

void agrega_plano(headvis, list_plano, new_plano, a, b)  /* agrega un nuevo plano si
hace falta */
plano *list_plano, new_plano;          /* estruc. con plano nuevo */
vista *headvis;
entity *a, *b;                /* ents/ para agregar, b = NULL si solo es una */

{
  int comp_plano();
  void add_simplent();

  plano *temp, *aux_plano;   /* temporal por si se crea nuevo nodo */
  int i;                     /* contador para identificador */

  for(i = 1, aux_plano = list_plano; aux_plano->sig_sup != NULL;
                                        i++, aux_plano = aux_plano->sig_sup)
                     /* ya existia el plano */
    if (comp_plano(new_plano, aux_plano->sig_sup))
    {
        if (b != NULL)
        {
          add_simplent(headvis, aux_plano->sig_sup, a, 0);
          add_simplent(headvis, aux_plano->sig_sup, b, 0);
        }
        else
          add_simplent(headvis, aux_plano->sig_sup, a, 1);
        break;
    }
                     /* no existia el plano */
  if(aux_plano->sig_sup == NULL)
  {
```

```
    temp = (plano *) malloc(sizeof(plano));
    if(temp == NULL)
    {
        printf("Error !, falta de memoria en creacion de planos");
        exit(0);
    }
    *temp = new_plano;
    temp->id = i;
    aux_plano->sig_sup = temp;
    aux_plano = temp;
    if (b!= NULL)
    {
        add_simplent(headvis, aux_plano, a, 0);
        add_simplent(headvis, aux_plano, b, 0);
    }
    else

        add_simplent(headvis, aux_plano, a, 1);
  }
}


int comp_plano(p1,p2)          /* compara dos planos con tolerancia predef.*/
plano p1, *p2;
{
  if (p1.despeje == p2->despeje)
    return((sqr(p1.a - p2->a) + sqr(p1.b - p2->b) + sqr(p1.c - p2->c)) <=
sqr(EPS_COP));
  else
   return (0);
}


void add_simplent(headvis, sup, a,flag)        /* agrega nueva entidad si hace falta */
vista *headvis;
plano *sup;
entity *a;
int flag;                  /* indica si es linea o arco */

{
  vista *busvista();

  vista *axvis;
  simplent *actse, *auxse;     /* actual y auxiliar creac. de simplent */
  r3 axp1, axp2, axp3;       /* auxiliares para puntos de circulo */
```

```
  actse = sup->sig_ent;
  if (actse != NULL)
    while (actse != NULL)
        if (actse->id == a->id_ent)   /* ya existia entidad */
          break;
        else
          actse = actse->sig_ent;
  if (actse == NULL)
  {                                   /* no existia entidad */
    auxse = (simplent *) malloc(sizeof(simplent));
    if (auxse == NULL)
    {
        printf("Error en generacion de lista de entidades de superficie %d",sup->id);
        exit(0);
    }
    auxse->id = a->id_ent;

    if (! a->sup1)                    /* asignacion a entity de la superficie donde esta */
        a->sup1 = sup->id;
    else if (! a->sup2)
        a->sup2 = sup->id;
    else
    {
        printf("Error ! entidad %d une tres planos: %d, %d, %d\n",a->id_ent, a->sup1,
a->sup2, sup->id);
        exit(0);
    }

                                      /* asignacion a simplent de las coordenadas rels. al plano
*/
    if (!flag)
    {                   /* punto inicial y final de la linea */
        auxse->coord1 = proj_ort(sup->v1, sup->v2, a->p1, sup->delta);
        auxse->coord2 = proj_ort(sup->v1, sup->v2, a->p2, sup->delta);
        asigna2(&auxse->coord3, 0.0, 0.0);
    }
    else
    {                     /* punto inicial, final y centro del arco */
                                      /* aqui se convierten de coord. vista a coord. mundo */
        axvis = busvista(headvis, a->vis_ent);
        axp1 = plrv_w(axvis->v1, axvis->v2, axvis->v3, a->p2.x, a->p2.y, a->p1);
        axp2 = plrv_w(axvis->v1, axvis->v2, axvis->v3, a->p2.x, a->p2.y + a->p2.z, a-
>p1);
        axp3 = vis_wrd(axvis->v1, axvis->v2, axvis->v3, a->p1);
```

```
                                /* aqui de coord mundo (menos desplaz) a coord
superficie */

        auxse->coord1 = proj_ort(sup->v1, sup->v2, axp1, sup->delta);
        auxse->coord2 = proj_ort(sup->v1, sup->v2, axp2, sup->delta);
        auxse->coord3 = proj_ort(sup->v1, sup->v2, axp3, sup->delta);
    }
    auxse->sign    = '+';
    auxse->tipo = a->tipo_ent;
                                /* Insercion al final de la lista */
    if ((actse = sup->sig_ent) != NULL)
    {
        while (actse->sig_ent != NULL)
          actse = actse->sig_ent;
        actse->sig_ent = auxse;
    }
    else
        sup->sig_ent = auxse;
    auxse->sig_ent = NULL;
  }
}
```

Módulo para la creación de ciclos de entidades.

```
/* d:\guillerm\ciclos.c

  Modulo que forma ciclos de entidades para cada superficie, y las ordena */

#include <math.h>
#include <alloc.h>
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <c:\pitufo\defines.hdr>

extern void error();

double ang_ciclo(ent1,ent2)  /* devuelve el angulo entre 2 ents. direccionadas */
simplent *ent1, *ent2;       /* en un rango entre -pi y +pi */
{
  r2 v1,v2;                  /* vectores con la direccion del ciclo */
  double t1,t2, reg;
                             /* caso de angulo entre lineas */
  if(ent1->tipo == 2 && ent2->tipo == 2)
  {
```

```
  v1 = sub2(ent1->coord2, ent1->coord1);
  v2 = sub2(ent2->coord2, ent2->coord1);
}
else
{
  if (ent1->tipo == 2)
  {                 /* asigna al arco la direccion normal al radio*/
      if (ent1->coord2.x - ent1->coord3.x)
      {
        v1.x = (ent1->coord3.y - ent1->coord2.y)/(ent1->coord2.x - ent1->coord3.x);
        v1.y = 1;
      }
      else          /* el vector de diferencia es vertical */
      {
        v1.x = 1;
        v1.y = 0;
      }                         /* comprueba la orientacion del vertice normal */
      if (exor(ent1->sign == '+',
            det2(sub2(ent1->coord2, ent1->coord3), v1) > 0))
        v1 = escprod2( -1.0, v1);
  }
  if (ent2->tipo == 2)
  {                 /* asigna al arco la direccion normal al radio*/
      if (ent2->coord2.x - ent2->coord3.x)
      {
        v2.x = (ent2->coord3.y - ent2->coord2.y)/(ent2->coord2.x - ent2->coord3.x);
        v2.y = 1;
      }
      else          /* el vector de diferencia es vertical */
      {
        v2.x = 1;
        v2.y = 0;
      }                         /* comprueba la orientacion del vertice normal */
      if (exor(ent2->sign == '+',
            det2(sub2(ent2->coord2, ent2->coord3), v2) > 0))
        v2 = escprod2( -1.0, v2);
  }
}
    /* angulo es diferencia. entre angulos polares */
if (v1.x)
  t1 = atan(v1.y/v1.x);
else
  t1 = PI / 2;
if (t1 >= 0)
  t1 = (v1.x >= 0 && v1.y >= 0)? t1: t1 + PI;   /* distincion entre 1 y 3 cuadr. */
```

```
  else
    t1 = (v1.x >= 0 && v1.y >= 0)? 2 * PI + t1: t1 + PI;   /* distincion entre 2 y 4 cuadr.
*/

  if (v2.x)              /* similar para vector 2 */
    t2 = atan(v2.y/v2.x);
  else
    t2 = PI / 2;
  if (t2 >= 0)
    t2 = (v2.x >= 0 && v2.y >= 0)? t2: t2 + PI;
  else
    t2 = (v2.x >= 0 && v2.y >= 0)? 2 * PI + t2: t2 + PI;

  reg = t2 - t1;              /* ubicacion en (-pi, pi] */
  reg = reg >= 0? (reg>PI? reg - 2*PI: reg): (reg< -PI? 2*PI + reg: reg);
  return reg;
}


void swap (ent)          /* invierte la coordenada inicial y final si no */
simplent *ent;           /* llevan el mismo sentido que la primera entidad */
                         /* tambien pone un signo menos en la entidad */
{
 r2 temporal;            /* variable para poder hacer el swap de coorden */

 temporal.x   = ent->coord1.x;
 temporal.y   = ent->coord1.y;
 ent->coord1.x = ent->coord2.x;
 ent->coord1.y = ent->coord2.y;
 ent->coord2.x = temporal.x;
 ent->coord2.y = temporal.y;
 ent->sign   = '-';

}

int comp_simplent (simp1,simp2)      /* compara si la coordenada 2 de la primera */
simplent *simp1,*simp2;              /* entidad es igual que la coordenada de la */
                                     /* segunda. Si es asi, el ciclo continua. Sino */
                                     /* quiere decir que hay que hacer otro ciclo  */
{
  return (equal2(simp1->coord2, simp2->coord1) || equal2(simp1->coord2, simp2-
>coord2));
}

ciclo *haz_ciclo(headent, superf)  /* A partir de un plano toma las entidades que  */
```

```
entity *headent;
plano *superf;          /* se encuentran enlazadas a el y forma una lista */
{                               /* de ciclos. Si encuentra que hay lineas que no  */
                                /* tienen puntos en comun, forma el ciclo hermano */

  extern entity *buscaent();
  ciclo    *cicleptr, *tempptr,     /* apuntadores para la lista de ciclos   */
          *initptr;
  simplent  *entptr, *auxptr,       /* sirven para recorrer lista de simplent */
            *currptr,*temp1,*temp2,       /* encadenada a la superficie          */
            *temp3,*initpoint;
  int       bandera,flag,idnum;
  double    angulo,xmax,xmin,ymax,ymin, radio;
  r2        coord1h, coord1l,coord2h,coord2l;         /* auxiliares en la creacion de caja
*/

  flag   = 1;                       /* bandera que indica si quedan simplent */
  idnum  = 0;
  tempptr = NULL;
  angulo  = 0;
  entptr  = superf->sig_ent;
  auxptr  = entptr;
  while (auxptr != NULL)
   {
    temp1   = auxptr;
    currptr = auxptr->sig_ent;
    while ( currptr != NULL )
     {
      temp2 = currptr->sig_ent;
      if (comp_simplent (auxptr,currptr))
          {
           if (equal2(auxptr->coord2, currptr->coord2))
               swap (currptr);
          temp3 = auxptr->sig_ent;
        if (auxptr->sig_ent != currptr)
              currptr->sig_ent = temp3;
          auxptr->sig_ent  = currptr;
          if (temp1 != auxptr)
              temp1 ->sig_ent  = temp2;
          }
         else
          temp1 = currptr;
         currptr = temp2;
     }
    auxptr = auxptr->sig_ent;
```

```
 }
auxptr = entptr;
superf->sig_ent = NULL;
while (flag)
 {
  angulo = 0;
  idnum++;
  cicleptr = (ciclo *) malloc(sizeof(ciclo));
  if (cicleptr == NULL)
       error("Error en generacion de lista de ciclos de superficie %d");
  cicleptr->ciclo_int = NULL;
  cicleptr->sig_ciclo = NULL;
  cicleptr->prof = 0;
  cicleptr->sig_ent  = auxptr;
                              /* inicializa max y min como los primeros */
  xmax = xmin = auxptr->coord1.x;
  ymax = ymin = auxptr->coord1.y;

  if (idnum==1)
      {
       tempptr = cicleptr;
       initptr = cicleptr;
      }
  else
      tempptr->sig_ciclo = cicleptr;
  cicleptr->id = idnum;
  bandera = 1;
  initpoint = auxptr;
  if (auxptr->tipo == 30)
      {
       radio = buscaent(headent, auxptr->id)->p2.x;
       auxptr->sign = '+';
       xmax = auxptr->coord3.x + radio;
       xmin = auxptr->coord3.x - radio;
       ymax = auxptr->coord3.y + radio;
       ymin = auxptr->coord3.y - radio;
       currptr = auxptr->sig_ent;
       auxptr->sig_ent = auxptr;
       cicleptr->sign = '+';
       cicleptr->sig_ent = auxptr;
       auxptr = currptr;
       bandera = 0;
      }
  while ((auxptr != NULL) && (bandera) && (auxptr->tipo != 30))
      {          /* coordenadas normales por linea */
```

```
        coord1h = coord1l = auxptr->coord1;
        coord2h = coord2l = auxptr->coord2;
        if (auxptr->tipo == 3)
        {       /* coordenadas extendidas por arco */
          radio = buscaent(headent, auxptr->id)->p2.x;
          coord1h.x += radio;
          coord1h.y += radio;
          coord1l.x -= radio;
          coord1l.y -= radio;
          coord2h.x += radio;
          coord2h.y += radio;
          coord2l.x -= radio;
          coord2l.y -= radio;
        }

        if (coord1l.y < ymin) ymin = coord1l.y;
        if (coord2l.y < ymin) ymin = coord2l.y;
        if (coord1h.y > ymax) ymax = coord1h.y;
        if (coord2h.y > ymax) ymax = coord2h.y;
        if (coord1l.x < xmin) xmin = coord1l.x;
        if (coord2l.x < xmin) xmin = coord2l.x;
        if (coord1h.x > xmax) xmax = coord1h.x;
        if (coord2h.x > xmax) xmax = coord2h.x;

        currptr = auxptr->sig_ent;
        if ((currptr != NULL) && (comp_simplent(auxptr,currptr)))
          angulo += ang_ciclo (auxptr,currptr);
        else
          if (comp_simplent (auxptr,initpoint))
          {
            angulo += ang_ciclo (auxptr,initpoint);
            if (angulo > 0)
                cicleptr->sign = '+';       /* si da como resultado 2 pi es positivo*/
            else                            /* en caso de - 2 pi es negativo */
                cicleptr->sign = '-';
            auxptr->sig_ent = cicleptr->sig_ent;
            bandera = 0;
          }
        auxptr = currptr;
      }
    if (auxptr==NULL)
      flag = 0;
    cicleptr->xmin = xmin;
    cicleptr->xmax = xmax;
    cicleptr->ymin = ymin;
```

```
    cicleptr->ymax = ymax;
    tempptr = cicleptr;
   }
 return initptr;
}

void agrega_ciclo (headent, headptr)      /* Se le pasa como parametro el apuntador */
entity *headent;
plano *headptr;                           /* del plano inicial para asi poder crear */
                                          /* en cada una de las superficies la lista */
                                          /* de ciclos que le corresponde          */
{
 plano *planoptr;

 planoptr = headptr->sig_sup;
 while (planoptr != NULL)
  {
   planoptr->sig_ciclo = haz_ciclo (headent, planoptr);
   planoptr = planoptr->sig_sup;
  }
}
```

Módulo para el ordenamiento de los ciclos, de acuerdo a sus inclusiones entre sí.

```
/* cicloin.c

 Modulo que ordena los ciclos espacialmente */

#include <math.h>
#include <alloc.h>
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>

void ordenaciclos(headplano, headent)
plano *headplano;
entity *headent;
{
 void acomoda();
 plano *supact;


 for (supact = headplano->sig_sup; supact != NULL; supact = supact->sig_sup)
   acomoda(supact, supact->sig_ciclo,0, headent);
}
```

```
void acomoda(sup, c,prof,headent)   /* procedimiento recursivo que ordena los ciclos */
plano *sup;
entity *headent;
ciclo *c;
int prof;
{
  void colcic();          /* coloca un ciclo abajo de otro */
  int puntoenciclo();  /* dice si un punto pertenece a un ciclo */

  ciclo *cm, *aux, aux2, *aux3;              /* ciclo en movimiento */


  if (c != NULL)
  {
    for (cm = c->sig_ciclo; cm != NULL;)
        if (puntoenciclo(cm->sig_ent->coord1, c,headent))
        {
          aux = cm->sig_ciclo;
          colcic(sup, cm, c, prof+1);
          cm = aux;
        }
        else
        {
          if (puntoenciclo(c->sig_ent->coord1, cm,headent))
          {
            aux = c;
            aux2 = *c;              /* swap entre c, cm */
            *c = *cm;
            *cm = aux2;              /* swap de ap. de cadena */
            aux3 = c->sig_ciclo;
            c->sig_ciclo = cm->sig_ciclo;
            cm->sig_ciclo = aux3;
            colcic(sup, cm, c, prof+1);
            cm = aux;
          }
          cm = cm->sig_ciclo;
        }
    acomoda(sup, c->sig_ciclo, prof,headent);
    acomoda(sup, c->ciclo_int, prof + 1,headent);
  }
}

void colcic(sup, c1, c2, prof)  /* coloca c1 abajo de c2 */
```

```
plano *sup;                /* c2 siempre es anterior a c1*/
ciclo *c1, *c2;
int prof;
{
  ciclo *cmov;

  if (sup->sig_ciclo != c1)
  {
    for (cmov = c2; cmov->sig_ciclo != c1; cmov = cmov->sig_ciclo);
    cmov->sig_ciclo = c1->sig_ciclo;
  }
  else
    sup->sig_ciclo = c1->sig_ciclo;
  c1->sig_ciclo = c2->ciclo_int;
  c2->ciclo_int = c1;
  c1->prof = prof;
  c1 = c2;
}


 /***********************************/

int puntoenciclo(punto, cicloptr,headent)  /* funcion que indica si un punto esta en
ciclo*/
r2 punto;
ciclo *cicloptr;
entity *headent;
{
  extern entity *buscaent();
  int estaencaja();
  int esposible();
  r3 interlin(),
    interarc();

  double rad;
  simplent *sentptr;
  int regreso;
  int antes = 0,
     despues = 0;
  r3 resulta;
  r2 res2;

  if (estaencaja(punto, cicloptr))
  {
    sentptr = cicloptr->sig_ent;
```

```
    if (sentptr->tipo == 30)
    {
        rad = buscaent(headent, sentptr->id)->p2.x;
        regreso = ( sqr(punto.x - sentptr->coord3.x)
                  + sqr(punto.y - sentptr->coord3.y) <= rad);
    }
    else
    {
        do
        {
          if (sentptr->tipo == 2 && esposible(punto, sentptr))
          {
            resulta = interlin(cicloptr->xmin, cicloptr->xmax, punto, sentptr);
            if (resulta.z)   /* Si no es paralela horizontal y sentptr*/
                if (resulta.y >= 0 && resulta.y <= 1) /*Si la inter. esta en sentptr*/
                                        /* posibiliadad de intersecciones */
                                         /* busca interseccion al final */
                  if (equal1(resulta.y, 1.0))
                  {         /* determina si es cruce o punto contacto */
                      if ( (  sentptr->coord1.y < punto.y
                           && sentptr->sig_ent->coord2.y > punto.y)
                         ||(  sentptr->coord1.y > punto.y
                           && sentptr->sig_ent->coord2.y < punto.y))
                        if (resulta.z <0) antes++; /* choco antes la linea */
                         else despues++;
                  }
                  else
                      if (resulta.z <0) antes++;
                        else despues++;
          }
          else
            if (sentptr->tipo == 3)
            {
                resulta = interarc(&res2, cicloptr->xmin, cicloptr->xmax, punto, sentptr,
headent);
                antes += res2.x;
                despues += res2.y;
            }
        } while((sentptr = sentptr->sig_ent) != cicloptr->sig_ent);
        regreso = antes % 2;    /* si es impar esta adentro */
    }
  }
  else
    regreso = 0;  /* si no esta en la caja, no esta */
  return regreso;
```

```
int estaencaja(p, c)        /* Indica si p esta en la caja que encierra a p */
r2 p;
ciclo *c;
{
  return (  (p.x <= c->xmax && p.x >= c->xmin)
          && (p.y <= c->ymax && p.y >= c->ymin));
}

int esposible(p, e)         /* Ind. si se puede int. x= p.x & e */
r2 p;
simplent *e;
{
  return (  (p.y <= e->coord2.y && p.y >= e->coord1.y)
          ||(p.y >= e->coord2.y && p.y <= e->coord1.y));
}


r3 interlin(xl, xh, p, lin)  /* checa interseccion entre y = yc entre xl,xh y lin*/
double xl, xh;       /* res.x param y=yc, res.y param lin */
r2 p;
simplent *lin;               /* res.z <0 int.izq; res.z>0 int.der; res.z = 0 no int.*/
{
  r2 a, b, c, d, aux;       /* parametros matriciales */
  double pto,det;           /* solucion a parametros */
  r3 regreso;

  a.x = xl;
  a.y = p.y;
  b.x = xh - xl;
  b.y = 0;
  c = lin->coord1;
  d = sub2(lin->coord2, lin->coord1);
                        /* Solucion ecuacion matrix 2 * 2, con b.y = 0 */
  if ((det = -(b.x * d.y)) != 0)
  {
    aux = sub2(c, a);
    regreso.x = (d.x * aux.y - d.y * aux.x) / det; /*param. de horiz al pto*/
    regreso.y = b.x * aux.y / det;              /*param. de linea*/
                          /* de que lado intersecto */
    pto = (p.x - xl) / (xh - xl);
    if (regreso.x < pto) regreso.z = -1;  /*izquierdo*/
    else regreso.z = 1; /* derecho */
```

```
  else
  {
    regreso.x = 0;
    regreso.y = 0;
    regreso.z = 0;
  }
  return regreso;
}
  /*************************/


r3 interarc(res2,xl,xh,p,arc,headent)  /* Interseccion entre lin. horiz y arco */
double xl, xh;          /* limites de la caja */
r2 *res2, p;            /* r2->x intersecc. antes, r2->y desp. punto a evaluar */
simplent *arc;          /* arco */
entity *headent;
{
  extern entity *buscaent(); /*falta informacion de arco */
  double rfalsa();
  double f();
  double tdados();              /* obtencion de t dado s en arco */

  r2 a,b,c;             /* despl. y dir. de recta, y centro arco */
  double r,f1,f2,    /* radio, angulo inicial e increm. y parametro */
         razon;   /* auxiliares para determinacion del lado interseccion */
  r3 regreso;
  entity *cir;

                        /* asignacion de los parametros de las ecucaciones */
  a.x = xl;
  a.y = p.y;
  b.x = xh - xl;
  b.y = 0.0;

  cir = buscaent(headent, arc->id);
  r = cir->p2.x;          /* radio del arco */
  f1 = cir->p2.y;     /* angulos inicial e incremental */
  f2 = cir->p2.z;
  c = arc->coord3;    /* centro en coord del plano */
  razon = (p.x - xl) / (xh - xl); /* altura del punto en la caja */
  asigna2 (res2, 0.0, 0.0);
                                          /* cero o dos soluciones */
  if (exor((f(0.0,a,b,c,r,f1,f2) > 0.0), (f(1.0,a,b,c,r,f1,f2) < 0.0)))
                                          /* dos soluciones */
```

```
    if (exor((f(0.0,a,b,c,r,f1,f2) > 0.0), (f(0.5,a,b,c,r,f1,f2) > 0.0)))
    {
        regreso.z = 2.0;
        regreso.x = rfalsa(0, 0.5);
        regreso.y = rfalsa(0.5, 1.0);
        if (tdados(regreso.x,a,b,c,r,f1,f2) < razon) res2->x++; else res2->y++;
        if (tdados(regreso.y,a,b,c,r,f1,f2) < razon) res2->x++; else res2->y++;
    }
    else
        asigna3(&regreso, 0.0, 0.0, 0.0);
  else
  {
    regreso.z = 1.0;
    regreso.x = rfalsa(0, 1.0);
    regreso.y = 0.0;
    if (tdados(regreso.x,a,b,c,r,f1,f2) < razon) res2->x++; else res2->y++;
  }
  return regreso;
}



double rfalsa(x,y,a,b,c,r,f1,f2) /* Encuentra una raiz si cambia el signo entre x, y*/
r2 a,b,c;       /* parametros de la funcion */
double x,y,r,f1,f2; /* lim. inf y sup. de regla falsa, y param. de funcion */
{
  double f();       /* Evaluador de funciones */

  double fx, fy, z, fz, regreso;

  fx = f(x,a,b,c,r,f1,f2);
  fy = f(y,a,b,c,r,f1,f2);
  z = (x * fy - y * fx)/(fy - fx);
  fz = f(z,a,b,c,r,f1,f2);

  if (!equal1(fz, 0.0))
  {
    if (exor(fx > 0, fz > 0))
        regreso = rfalsa(x,z,a,b,c,r,f1,f2);
    else
        regreso = rfalsa(z,y,a,b,c,r,f1,f2);
  }
  else
    regreso = z;
  return regreso;
```

```
}

double f(s,a,b,c,r,f1,f2)        /* Evaluador de funcion de interseccion */
r2 a,b,c;        /* a:desplaz., b:dir. recta; c: centro del arco */
double s,r,f1,f2;  /* s:parametro, r:radio arco, f1,f2:angulo inicial e increm*/
{
  double ang;  /*angulo logrado con un avance de s*/

  ang = f1 + s * (f2 - f1);

  return ( - b.x*r*sin(ang) - b.x*(c.y-a.y));
}

double tdados(s,a,b,c,r,f1,f2) /* parametro de la recta dado un valor del parm. arc */
double s,r,f1,f2;
r2 a,b,c;
{
  return ((c.x + r*cos(f1 + s*(f2 - f1)) - a.x)/b.x);
}
```

Módulo para la creación de túneles y túneles arqueados (dependiendo el parámetro)

```
/* d:\guillerm\creasep.c, MODULO QUE RECONOCE TUNELES Y TUNELES
ARQUEADOS */


#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>

extern void error();
                                 /* Crea tuneles si flag = 2, y tunarq. si flag = 3 */
tunel *creatunel(headent, headver, flag)
entity *headent;
vertex *headver;
int flag;
{
  entity *buscaentdver();
  vertex *buscaverid();
  tunel creantunel();
  void addtunel();
```

```
int estaensup();

tunel *headtunel,axnodot;
vertex *axver1, *axver2, *axver3;
entinver *axeinv1, *axeinv2, *axeinv3, *axeinv4;
entity *axa1, *axa2, *axl1, *axl2;
int f1, f2, f3, f4;
      /* Creacion de cabeza de tuneles */
headtunel = (tunel *)malloc(sizeof(tunel));
if (headtunel == NULL)
  error("Falta de memoria creacion de cabeza de tuneles ");
headtunel->id = flag == 2? 0: -256;
headtunel->tipo = flag;
headtunel->idarc1 = 0;
headtunel->idarc2 = 0;
headtunel->idlin1 = 0;
headtunel->idlin2 = 0;
headtunel->sgna1 = '+';
headtunel->sgna2 = '+';
headtunel->sgnl1 = '+';
headtunel->sgnl2 = '+';
headtunel->v1 = 0;
headtunel->v2 = 0;
headtunel->v3 = 0;
headtunel->v4 = 0;
headtunel->sig = NULL;
                              /* Busca los arcos en las entidades que hay en
                              cada vertice, vertice por vertice */

for(axver1 = headver->sig_ver, axeinv1 = axver1->content; axver1 != NULL;
            axeinv1 = (axver1 = axver1->sig_ver)->content)
{
  axa1 = buscaentdver(headent, &axeinv1, 3);
  if (axa1 != NULL)
  {
      f1 = 2 - axeinv1->side;

/*                              busca una linea/arco en el mismo vertice */
/*                      que no este en la misma superficie */
      for(axeinv2 = axver1->content;
            (axl1 = buscaentdver(headent, &axeinv2, flag)) != NULL;
            axeinv2 = axeinv2->sigentinver)
        if (axl1 != NULL)
          if (!estaensup(axl1, axa1->sup1))
          {
```

```
                f2 = axeinv2->side - 1;
                if (f2)
                  axver2 = buscaverid(headver, axl1->ver1);
                else
                  axver2 = buscaverid(headver, axl1->ver2);

/*                          con el nuevo vertice que dio la linea/arco*/
/*                  busca arcos */
                for(axeinv3 = axver2->content;
                      (axa2 = buscaentdver(headent, &axeinv3, 3)) != NULL;
                      axeinv3 = axeinv3 ->sigentinver)
                  if (axa2 != NULL)
                  {
                      f3 = axeinv3->side - 1;
                      if (f3)
                        axver3 = buscaverid(headver, axa2->ver1);
                      else
                        axver3 = buscaverid(headver, axa2->ver2);

/*                          con el nuevo vertice del arco, buscara una*/
/*                    nueva linea/arco */
                      for(axeinv4 = axver3->content;
                            (axl2 = buscaentdver(headent, &axeinv4, flag)) != NULL;
                            axeinv4 = axeinv4->sigentinver)
                        if (axl2 != NULL)
                        {
                          if(!estaensup(axl2, axa2->sup1))
                          {
                              f4 = axeinv4->side - 1;
                              if((f4 && (axl2->ver1 == axa1->ver1 || axl2->ver1 ==
axa1->ver2)) ||
                              (!f4 && (axl2->ver2 == axa1->ver1 || axl2->ver2 ==
axa1->ver2)))
    /* exito */
                              axnodot = creantunel(axa1, axa2, axl1, axl2, f1, f2, f3, f4,
flag);
                              addtunel(headtunel, axnodot, axa1, axa2, axl1, axl2, flag);
                          }
                        }
                        else
                        {
                          printf("\nError falta linea/arco para cerrar tunel entre ");
                          printf("arcos %d, %d, y linea %d", axa1->id_ent, axa2-
>id_ent, axl1->id_ent);
                          exit(0);
```

```
                    }
                }
                else
                {
                    error ("\nError, tunel esperado entre arco %d, y linea/arco %d",
                            axa1->id_ent, axl1->id_ent);
                    exit(0);
                }
            }
        }
    }
    return headtunel;
}

                                /*busca entidad dado su tipo en ents en vertice */
entity *buscaentdver(headent,auxeinv, tipo)
entity *headent;
entinver **auxeinv;
int tipo;
{
    entity *buscaent();

    entity *regreso;

    for( ;*auxeinv != NULL; *auxeinv = (*auxeinv)->sigentinver)
        if((*auxeinv)->tipo == tipo)
        {
            regreso = buscaent(headent, (*auxeinv)->id);
            break;
        }
    if (*auxeinv == NULL)
        regreso = NULL;
    return regreso;
}

entity *buscaent(headent, id)   /* regresa entidad dado su id */
entity *headent;
int id;
{
    entity *auxent;
    for(auxent = headent->sig; auxent != NULL && auxent->id_ent != id;
                                    auxent = auxent->sig);
    return auxent;
}
```

```
vertex *buscaverid(headver, id)        /* regresa vertice dado su id */
vertex *headver;
int id;
{
  vertex *auxver;
  for(auxver = headver->sig_ver; auxver != NULL && auxver->id != id;
                                    auxver = auxver->sig_ver);
  return auxver;
}

tunel creantunel(axa1, axa2, axl1, axl2, f1, f2, f3, f4, flag)
entity *axa1, *axa2, *axl1, *axl2;
int f1, f2, f3, f4, flag;
{       /* Crea un nodo de tunel dados parametros */
  tunel axtun;

  axtun.tipo = flag;
  axtun.idarc1 = axa1->id_ent;
  axtun.idarc2 = axa2->id_ent;
  axtun.idlin1 = axl1->id_ent;
  axtun.idlin2 = axl2->id_ent;
  axtun.sgna1 = f1? '-': '+';
  axtun.sgnl1 = f2? '-': '+';
  axtun.sgna2 = f3? '-': '+';
  axtun.sgnl2 = f4? '-': '+';
  axtun.v1 = f1? axa1->ver2: axa1->ver1;
  axtun.v2 = f2? axl1->ver2: axl1->ver1;
  axtun.v3 = f3? axa2->ver2: axa2->ver1;
  axtun.v4 = f4? axl2->ver2: axl2->ver1;
  axtun.sig = NULL;

  return axtun;
}


void addtunel(headtunel, newtunel, a, b, c, d, flag)  /* agrega un tunel nuevo si hace falta
*/
tunel *headtunel, newtunel;   /* cabeza y tunel nuevo */
entity *a, *b, *c, *d;        /* ents/ para modificar */
int flag;

{
  int comp_tunel();
  void asignastun();
```

```
  tunel *axtunel, *temp;
  int i;                    /* contador para identificador */

  for(i = (flag == 2? -1: -257), axtunel = headtunel; axtunel->sig != NULL;
                                                  i—, axtunel = axtunel->sig)
                            /* ya existia el tunel */
    if (comp_tunel(&newtunel, axtunel->sig))
        break;
              /* no existia el tunel */
  if(axtunel->sig == NULL)
  {
    temp = (tunel *) malloc(sizeof(tunel));
    if(temp == NULL)
        error("Error !, falta de memoria en creacion de planos");
    *temp = newtunel;
    temp->id = i;
    axtunel->sig = temp;
    axtunel = temp;             /* asignacion a las sups del tunel */
    asignastun(a, i);
    asignastun(b, i);
    asignastun(c, i);
    asignastun(d, i);
  }
}

int comp_tunel(a, b)            /* compara 2 tuneles para ver si son el mismo*/
tunel *a, *b;
{
   return(  (  a->idarc1 == b->idarc1 && a->idarc2 == b->idarc2
             || a->idarc1 == b->idarc2 && a->idarc2 == b->idarc1)
          && (  a->idlin1 == b->idlin1 && a->idlin2 == b->idlin2
             || a->idlin1 == b->idlin2 && a->idlin2 == b->idlin1));
}


void asignastun(a, i)           /* asigna a la superficie el tunel donde esta */
entity *a;
int i;
{
  if (!a->sup1)
    a->sup1 = i;
  else if (!a->sup2)
          a->sup2 = i;
        else
        {
```

```
            printf("Error!, entidad %d une 3 superficies : %d, %d, %d\n",a-> id_ent, a-
>sup1, a->sup2, i);
            exit(0);
          }
}

int estaensup (a,b)      /* Es verdadera si la entidad a esta en sup b */
entity *a;
int b;
{
  return (a->sup1 == b || a->sup2 == b);
}
```

Módulo que forma cilindros y bujes..

```
/* d:\guillermo\creacil.c, MODULO QUE RECONOCE CILINDROS Y BUJES */

#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>

extern void error();

cilindro *hazcils(headent, headver, headplano, flag)
entity *headent;
vertex *headver;
int flag;
{
  extern entity *buscaent();  /* Regresa entidad dado id, esta en creasep.c*/
  extern entity *buscaentdver(); /*Busca una entidad dado un tipo y un vertice*/
  extern vertex *buscaverid();  /* Busca vertice dado id, tambien esta en creasep.c*/

  int verencir();              /* Dice si un vertice esta en un circulo */
  cilindro creancil();         /* Crea nodo  de cilindro */
  void addcil();               /* Agrega nodo de cilindro */


  cilindro *headcil, axnodocil;
  entity *axcir, *axcir2, *axlin;
  vertex *axver, *axver2;
  entinver *axeinv;
```

```
  int f,exito = 0;                    /* f es para el lado de la linea, y exito para
desanidar*/
                                      /*Creacion de cabeza de cilindros */
  headcil = (cilindro *) malloc(sizeof(cilindro));
  if (headcil == NULL) error("ERROR ! Falta de memoria creacion cabeza de
cilindros");
  headcil->tipo = flag;
  headcil->id = flag == 2? -514: -1024;
  headcil->idcir1 = headcil->idcir2 = headcil->idlin1 = 0;
  headcil->sgn11 = '+';
  headcil->v1 = headcil->v2 = 0;
  headcil->sig = NULL;

                                      /*Busca en entidades y considera los circulos */
  for(axcir = headent->sig; axcir != NULL; axcir = axcir->sig)
  {
    if(axcir->tipo_ent == 30)
                                      /*Busca en todos los vertices cual pertenece al circ*/
        for(axver = headver->sig_ver; axver != NULL; axver = axver->sig_ver)
      {
          if (verencir(headplano, axcir, axver))
                                      /* Va a buscar la linea/arco de enlace */
                for(axeinv = axver->content; axeinv != NULL; axeinv = axeinv-
>sigentinver)
                {
                  axlin = buscaentdver(headent, &axeinv, flag);
                  if (axlin != NULL)
                  {
                    f = axeinv->side - 1;
                    axver2 = f? buscaverid(headver, axlin->ver1):
                                buscaverid(headver, axlin->ver2);
                                      /* Va a buscar algun circulo que quede en axver2*/
                    for(axcir2 = headent->sig; axcir2 != NULL; axcir2 = axcir2->sig)
                        if(axcir2->tipo_ent == 30)
                        {
                          if (verencir(headplano, axcir2, axver2))
/* Exito */
                          {
                            axnodocil = creancil(axcir, axlin, axcir2, axver->id, axver2-
>id, f);
                            addcil(headcil, axnodocil, axcir, axlin, axcir2, flag);
                            exito = 1;
                            break;
                          }
                        }
```

```
                    if (axcir2 == NULL)
                    {
                        printf("Error, falta cerrar tunel entre circulo %d y linea/arco %d",
                                axcir->id_ent, axlin->id_ent);
                        exit(0);
                    }
                  }
                  if (exito || axeinv == NULL) break;
                }
        if (exito) break;
         }
    exito = 0;
  }
  return headcil;
}
        /*******************************************************/

int verencir(headplano, circulo, vertice)
plano *headplano;
entity *circulo;
vertex *vertice;
{
  plano *buscaplano();       /* busca plano donde esta el circulo */
  int verinplan();           /* dice si el vertice pertenece al plano */

  r2 axver, axcc;            /* auxiliares vertice y centro cir. en coord. plano*/
  plano *supcir;             /* superficie donde esta el circulo */
  int regreso;
                             /* busca si el vertice pertenece al mismo plano */
                             /* que el circulo, si no regresa cero */
  supcir = buscaplano(headplano, circulo);
  if (verinplan(vertice->punto, supcir))
  {
    axver = proj_ort(supcir->v1, supcir->v2, vertice->punto, supcir->delta);
    axcc = proj_ort(supcir->v1, supcir->v2, circulo->p1, supcir->delta);
    regreso = equal1((sqr(axver.x - axcc.x) + sqr(axver.y - axcc.y)), sqr(circulo->p2.x));
  }
  else
    regreso = 0;
  return regreso;
}

plano *buscaplano(headplano, circulo)  /* devuelve el plano donde esta el circulo*/
plano *headplano;
entity *circulo;
```

```
{
  plano *axplano;

  for(axplano = headplano->sig_sup;
                (axplano->id != circulo->sup1) && (axplano != NULL);
                 axplano = axplano->sig_sup);
  return axplano;
}

int verinplan(punto, sup)       /* dice si el punto esta en plano */
r3 punto;
plano *sup;
{
  int regreso;

  if (sup->despeje == 'z')
    regreso = equal1(punto.z, sup->a*punto.x + sup->b*punto.y + sup->c);
  else if (sup->despeje == 'y')
        regreso = equal1(punto.y, sup->a*punto.x + sup->b*punto.z + sup->c);
    else if (sup->despeje == 'x')
          regreso = equal1(punto.x, sup->a*punto.y + sup->b*punto.z + sup->c);
        else
          regreso = 0;
  return regreso;
}
        /*****************************/

cilindro creancil(axcir, axlin, axcir2, ver1, ver2, f)
entity *axcir, *axlin, *axcir2;
int ver1, ver2;
int f;
{
  cilindro ret;

  ret.idcir1 = axcir->id_ent;
  ret.idcir2 = axcir2->id_ent;
  ret.idlin1 = axlin->id_ent;
  ret.sgn11 = f? '-': '+';
  ret.v1 = ver1;
  ret.v2 = ver2;
  ret.sig = NULL;

  return ret;
}
```

```
void addcil(headcil, newcil, axcir, axlin, axcir2, flag)
cilindro *headcil, newcil;
entity *axcir, *axlin, *axcir2;
int flag;
{
  extern void asignastun();    /* asigna a la entidad la superficie (creasep) */
  int compcil();

  cilindro *axcil, *temp;
  int i;

  for(axcil = headcil, i = (flag == 2? -515: -1025); axcil->sig != NULL;
                i—, axcil = axcil->sig)
                                    /* ya existia el cilindro */
    if (compcil(axcil->sig, &newcil))
        break;
                                    /* no existia el cilindro */
  if (axcil->sig == NULL)
  {
    temp = (cilindro *) malloc(sizeof(cilindro));
    if (temp == NULL) error("Error!, falta de memoria en creacion de tuneles");
    *temp = newcil;
    temp->id = i;
    temp->tipo = flag;
    axcil->sig = temp;
    axcil = temp;
    asignastun(axcir, i);
    asignastun(axcir2, i);
    asignastun(axlin, i);
                                /* asigna al circulo el vertice */
    axcir->ver1 = temp->v1;
    axcir2->ver1 = temp->v2;
  }
}

int compcil(a, b)
cilindro *a, *b;
{
  return (  (a->idlin1 == b->idlin1)
          && ( (a->idcir1 == b->idcir1 && a->idcir2 == b->idcir2)
            ||(a->idcir1 == b->idcir2 && a->idcir2 == b->idcir1)));
}
```

Módulo que con la información anterior completa la lista de vértices.

```
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>

extern error();
/* modulo que completa la lista de vertices */


void comp_ver(headver, headcil)
vertex   *headver;
cilindro *headcil;
{
  vertex   *buscaver();
  void     addver();

  cilindro  *auxcil;

  for (auxcil = headcil->sig; auxcil != NULL; auxcil = auxcil->sig)
  {
    addver (buscaver(auxcil->v1, headver), auxcil);
    addver (buscaver(auxcil->v2, headver), auxcil);
  }
}

   /*****************************/

vertex *buscaver(id, headver)
int   id;
vertex *headver;
{
  vertex *auxver;

  for(auxver = headver->sig_ver; auxver->id != id && auxver != NULL;
                                  auxver = auxver->sig_ver);

  return auxver;
}

void addver(vertice, cil)
```

```
vertex  *vertice;
cilindro *cil;
{
  entinver *crea_eiv();

  entinver *axeiv, *neweiv;

  if (vertice->content != NULL)
  {
    for(axeiv = vertice->content; axeiv->sigentinver != NULL;
                                    axeiv = axeiv->sigentinver);
    neweiv = crea_eiv(cil->idcir1, 0, 30);
    axeiv->sigentinver = neweiv;
  }
  else
  {
    neweiv = crea_eiv(cil->idcir1, 0, 30);
    vertice->content = neweiv;
  }
}


entinver *crea_eiv(id, side, tipo)
int id,side,tipo;
{
  entinver *neweiv;

  neweiv = (entinver *) malloc(sizeof(entinver));
  if (neweiv == NULL)
    error("Falta de memoria en correccion de lista de vertices ");
  neweiv->id = id;
  neweiv->side = side;
  neweiv->tipo = tipo;
  neweiv->sigentinver = NULL;

  return neweiv;
}
```

Módulo que asigna dirección normal a cada superficie, en el caso de túneles y arcos indica con una bandera en su estructura si la dirección del sólido es hacia el centro del arco o el caso contrario..

```
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <c:\pitufo\defines.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


                        /* Asigna direccion normal a todas las sups */
                    /* asigna direccion normal a cada sup.*/
void assnormal(headplano, headent, headtunel, headtarq, headcil, headbuje, headvertex)
plano   *headplano;
entity  *headent;
tunel   *headtunel, *headtarq;
cilindro *headcil, *headbuje;
vertex  *headvertex;
{
  void alosciclos();            /* auxiliar recursivo para ciclos */
  plano *supact;
                                /* @ superficie */
  for(supact = headplano->sig_sup; supact != NULL; supact = supact->sig_sup)
    alosciclos(supact->sig_ciclo, supact, headent, headplano, headtunel, headtarq,
headcil, headbuje, headvertex);
}

void alosciclos(cicloact, supact, headent, headplano, headtunel, headtarq, headcil,
headbuje, headvertex)
                                /*auxiliar para recorrer todos los ciclos*/
ciclo   *cicloact;
plano   *supact, *headplano;
entity  *headent;
tunel   *headtunel, *headtarq;
cilindro *headcil, *headbuje;
vertex  *headvertex;

{
  extern entity *buscaent();
```

```
void        asigna();
void        angentsup();

simplent *sent;
entity  *ent;
int     superf;   /* la otra superficie a la que pertenece */
char    signo;

if (cicloact != NULL)
{
  sent = cicloact->sig_ent;
  do
  {          /* a cada entidad de ciclo */
      ent = buscaent(headent, sent->id);
      superf = (ent->sup1 == supact->id)? ent->sup2: ent->sup1;
                                  /* EL SIGNO INDICA EL LADO DEL SOLIDO
*/
                                  /* PARA CADA ENTIDAD */
      if (cicloact->prof % 2)      /* si es profundidad impar */
        signo = exor((cicloact->sign == '+'), (sent->sign == '+'))? '+': '-';
      else                        /* si es profundidad par */
        signo = exor((cicloact->sign == '+'), (sent->sign == '+'))? '-': '+';

                              /* asigna direccion normal */
      asigna(superf, supact, sent, signo, headplano, headtunel, headtarq, headcil,
headbuje);

                              /* asigna angulo entre vertices */
      angentsup(ent, sent, signo, headvertex, headent);


      sent = sent->sig_ent;
  } while (sent != cicloact->sig_ent);

  alosciclos(cicloact->sig_ciclo, supact, headent, headplano, headtunel, headtarq,
headcil, headbuje,headvertex);
  alosciclos(cicloact->ciclo_int, supact, headent, headplano, headtunel, headtarq,
headcil, headbuje,headvertex);
 }
}

void asigna(superf, supact, sent, signo, headplano, headtunel, headtarq, headcil,
headbuje)
                              /*asigna direccion normal dado signo */
plano   *headplano, *supact;
```

```
tunel   *headtunel, *headtarq;
cilindro *headcil,  *headbuje;
simplent *sent;
int     superf;
{
  extern double ang_ciclo();
  plano   *buscasup();
  tunel   *buscatun();
  cilindro *buscacil();

  plano   *sup;                 /* auxiliares para referencias de tipos superficies */
  tunel   *tun;
  cilindro *cil;
  r3      auxdir,
            cero3 = {0.0, 0.0, 0.0};
  simplent prjaxdr;             /* proyeccion de direccion normal */
  simplent sentaux;


  if (superf > 0)       /* si la superficie es un plano */
  {
    sup = buscasup(headplano, superf);
                        /* asigna a auxdir la direccion normal negativa*/
    if (sup->despeje == 'z') asigna3(&auxdir, sup->a, sup->b, -1.0);
        else if (sup->despeje == 'y') asigna3(&auxdir, sup->a, -1.0, sup->b);
          else asigna3(&auxdir, -1.0, sup->a, sup->b);

                         /* se proyecta ortnrml la normal al plano ref.*/
    prjaxdr.tipo = 2;
    asigna2(&prjaxdr.coord1, 0.0, 0.0);
    prjaxdr.coord2 = proj_ort(supact->v1, supact->v2, auxdir, cero3);
                         /* se direcciona el vector de referencia */
    sentaux = *sent;
    if (signo == '-')
    {
        sentaux.coord2 = sent->coord1;
        sentaux.coord1 = sent->coord2;
    }
                    /* si el angulo es negativo lo invierte */
    if (ang_ciclo(&sentaux, &prjaxdr) < 0)  auxdir = escprod3(-1.0, auxdir);

    sup->dir = auxdir;
  }
  else if (superf < -1024)      /* un buje */
    {
```

```
        cil = buscacil(headbuje, superf);
        cil->interior = (signo == '+')? 0: 1;
    }
    else if (superf < -514)     /* un cilindro */
        {
          cil = buscacil(headcil, superf);
          cil->interior = (signo == '+')? 0: 1;

        }
        else if (superf < -256)  /* un tunel arqueado */
          {
            tun = buscatun(headtarq, superf);
            tun->interior = (signo == '+')? 0: 1;
          }
          else              /* un tunel */
          {
            tun = buscatun(headtunel, superf);
            tun->interior = (signo == '+')? 0: 1;
          }
}



plano *buscasup(headplano, superf)  /* devuelve superficie dado su id */
plano *headplano;
int superf;
{
  plano *movplano;

  for (movplano = headplano->sig_sup;
        movplano != NULL && movplano->id != superf;
        movplano = movplano->sig_sup);
  return movplano;
}


tunel *buscatun(headtunel, superf)  /* devuelve tunel dado su id*/
tunel *headtunel;
int superf;
{
  tunel *movtunel;

  for(movtunel = headtunel->sig;
     movtunel != NULL && movtunel->id != superf;
     movtunel = movtunel->sig);
```

```
  return movtunel;
}

cilindro *buscacil(headcil, superf)  /* devuelve cilindro dado su id*/
cilindro *headcil;
int superf;
{
  cilindro *movcil;

  for(movcil = headcil->sig;
     movcil != NULL && movcil->id != superf;
     movcil = movcil->sig);
  return movcil;
}


    /*****************************/

                        /* asigna angulos entre superficies */
void angentsup(ent, sent, signo, headvertex, headent)
entity  *ent;
simplent *sent;
char    signo;
vertex  *headvertex;
entity  *headent;
{
  extern  vertex *buscaverid();
  extern  double ang_ciclo();

  vertex  *axver;
  simplent *axsent;     /* sig. ent. de sent para calculo de angulo */
  entinver *eiv;
  double   angulo;
  char     signtot;


  axsent = sent->sig_ent;
  if (axsent == sent)    /* una sola entidad en el ciclo */
  {
    if(sent->tipo == 30)
    {              /* un circulo tiene solo un vertice */
        axver = buscaverid(headvertex, ent->ver1);
        for(eiv = axver->content; eiv != NULL; eiv = eiv->sigentinver)
          if(eiv->id != ent->id_ent)
              break;
```

```
        if (eiv != NULL) /* si el solido es hacia dentro del cil o hacia fuera*/
        {
          buscaent(headent, eiv->id)->ang = signo == '+'? PI / 2: -PI /2;
          buscaent(headent, sent->id)->ang = signo == '+'? PI / 2: -PI/2;
        }
    }
  }
  else              /* Este signo considera el enlace por la cola, no por punta*/
  { signtot = exor(signo == '+', sent->sign == '+')? '-': '+';
    angulo = ang_ciclo(sent, axsent); /*depend. de orien. busca vertice*/
    axver = (sent->sign == '+')? buscaverid(headvertex, ent->ver2):
                                 buscaverid(headvertex, ent->ver1);
    for(eiv = axver->content; eiv != NULL; eiv = eiv->sigentinver)
        if(eiv->id != axsent->id && eiv->id != sent->id)
          break;            /* tercera entidad en el vertice */
    if (eiv != NULL)
        buscaent(headent, eiv->id)->ang = (signtot == '+')? angulo: -angulo;
  }
}
```

Módulo que recibe la información archivos de conocimiento.

```
        /* Modulo que lee de archivo las bases de conocimiento */
#include <c:\pitufo\estruc.hdr>
#include <c:\pitufo\alg_lin.hdr>
#include <c:\pitufo\defines.hdr>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern error();

ttver *knowver(nomarch)
char *nomarch;
{
  void  addttv();

  ttver *headttver;
  FILE  *fileptr;
  int   id,t12,t23,t31;
  double a1x,a1y,a2x,a2y,a3x,a3y;

  headttver = (ttver *) malloc (sizeof(ttver));
```

```
  if (headttver == NULL)
    error("Error, falta de memoria en creacion cabeza de tipos vertices");
  headttver->id = 0;
  headttver->flag = 0;
  asigna2(&(headttver->ang1), 0.0, 0.0);
  asigna2(&(headttver->ang2), 0.0, 0.0);
  asigna2(&(headttver->ang3), 0.0, 0.0);
  headttver->tipo12 = 0;
  headttver->tipo23 = 0;
  headttver->tipo31 = 0;
  headttver->sig = NULL;


  fileptr = fopen(nomarch,"r");
  if (fileptr == NULL)
    error("Error!, no se puede abrir el archivo de tipos de vertices");
  printf("Procesando tipos de vertices en %s",nomarch);
  while (!feof(fileptr))
  {
    fscanf(fileptr, "%d %d %d %d %lf %lf %lf %lf %lf %lf\n",&id,
                    &t12,&t23,&t31,&a1x,&a1y,&a2x,&a2y,&a3x,&a3y);
    addttv(headttver,id,t12,t23,t31,a1x,a1y,a2x,a2y,a3x,a3y);
  }
  return headttver;
}


void addttv(headttver,id,t12,t23,t31,a1x,a1y,a2x,a2y,a3x,a3y)
ttver   *headttver;
int     id,t12,t23,t31;
double  a1x,a1y,a2x,a2y,a3x,a3y;
{
  ttver *currttv, *newttv;
                                /*Nodo nuevo*/
  newttv = (ttver *)malloc(sizeof(ttver));
  if (newttv == NULL)
    error("Error, falta de memoria en creacion de cabeza tabla vertices");
  newttv->id = id;
  newttv->flag = 0;
  newttv->tipo12 = t12;
  newttv->tipo23 = t23;
  newttv->tipo31 = t31;
  asigna2(&(newttv->ang1),a1x,a1y);
  asigna2(&(newttv->ang2),a2x,a2y);
  asigna2(&(newttv->ang3),a3x,a3y);
```

```
  newttv->sig = NULL;

  for(currttv = headttver; currttv->sig != NULL; currttv = currttv->sig);
  currttv->sig = newttv;
}



        /*********************************/

trcmaq *hazlista(nomarch)
char nomarch[];
{
  tcmaq *leecar();          /* lee una caracteristica */
  void  agrenodolis();          /* agrega un nodo a la tabla de caracter */

  tcmaq  *caracptr;
  trcmaq *headcars;
  FILE   *fileptr1;
  int id;
  char desc[10], arch[20];
                                /* nodo inicial headcars */
  headcars = (trcmaq *)malloc(sizeof(trcmaq));
  if (headcars == NULL)
    error("Falta memoria en creacion cabeza de lista caracteristicas ");
  headcars->id = 0;
  headcars->carac = NULL;
  headcars->sig = NULL;

  fileptr1 = fopen(nomarch,"r");
  if (fileptr1 == NULL)
    error ("Error !, no se pudo abrir relacion de archivos ");
  printf("\nprocesando %s\n",nomarch);
  while (!feof(fileptr1))
  {
    fscanf(fileptr1,"%d %s %s",&id,&desc,&arch);
    printf("%d %s %s",id, desc, arch);
    caracptr = leecar(arch);
    agrenodolis(headcars, id, caracptr);
  }
  return headcars;
}

void agrenodolis(headcars, id, caracptr)
trcmaq *headcars;
```

```
int   id;
tcmaq  *caracptr;
{
  trcmaq *caract, *newcar;

  for (caract = headcars; caract->sig != NULL; caract = caract->sig);
  newcar = (trcmaq *) malloc (sizeof(trcmaq));
  if (newcar == NULL)
    error("falta de memoria en creacion de caracteristicas maquinado");
  newcar->id = id;
  newcar->carac = caracptr;
  newcar->sig = NULL;
  caract->sig = newcar;
}
  /************************/

tcmaq *leecar(arch)
char arch[];
{
  void addci();

  tcmaq *headci;
  FILE *fileptr2;
  int id, tipo, liga1, liga2, liga3;

  headci = (tcmaq *) malloc(sizeof(tcmaq));
  if(headci == NULL)
    error(" falta de memoria en creacion de cabeza caract. individual");
  headci->id = 0;
  headci->tipo = 0;
  headci->liga1 = 0;
  headci->liga2 = 0;
  headci->liga3 = 0;
  headci->sig = NULL;

  fileptr2 = fopen(arch,"r");
  if (fileptr2 == NULL)
    error("Error en apertura de archivos individuales caracteristicas");
  printf("\nProcesando %s\n",arch);
  while(!feof(fileptr2))
  {
    fscanf(fileptr2, "%d %d %d %d %d\n",&id,&tipo,&liga1,&liga2,&liga3);
    printf("%d %d %d %d %d\n",id, tipo, liga1, liga2, liga3);
    addci(headci,id, tipo, liga1, liga2, liga3);
  }
```

```
  return headci;
}

void addci(headci, id, tipo, liga1, liga2, liga3)
tcmaq *headci;
int  id, tipo, liga1, liga2, liga3;
{
  tcmaq *currci, *newci;

  for(currci = headci->sig; currci != NULL; currci = currci->sig);
  newci = (tcmaq *) malloc (sizeof(tcmaq));
  if (newci == NULL)
    error("falta de memoria en creacion de nodo en car. ind. maq");
  newci->id = id;
  newci->tipo = tipo;
  newci->liga1 = liga1;
  newci->liga2 = liga2;
  newci->liga3 = liga3;
  newci->sig = NULL;
  currci->sig = newci;
}
```

Módulo que escribe en un archivo de salida la información que va generando el programa.

```
                    /* MODULO QUE MUESTRA LAS ESTRUCTURAS */
#include <stdio.h>
#include <c:\pitufo\estruc.hdr>


void muestra(entity *head, FILE *fs)
{
  entity *aux;
  printf("\nEntidades :\n");
  fprintf(fs,"\nEntidades :\n");
  for (aux = head->sig; aux != NULL; aux = aux->sig)
  {
    printf("Entidad (%d %d %d) %3.4f %3.4f %3.4f %3.4f %3.4f %3.4f\n",
        aux->id_ent,aux->tipo_ent,aux->vis_ent,aux->p1.x,
        aux->p1.y,aux->p1.z,aux->p2.x,aux->p2.y,aux->p2.z);
    fprintf(fs,"Entidad (%d %d %d) %3.4f %3.4f %3.4f %3.4f %3.4f %3.4f\n",
        aux->id_ent,aux->tipo_ent,aux->vis_ent,aux->p1.x,
        aux->p1.y,aux->p1.z,aux->p2.x,aux->p2.y,aux->p2.z);
```

```
    printf(" en superf. (%d %d) Angulo (%2.4f), Vertices (%d %d)\n",
        aux->sup1, aux->sup2, aux->ang, aux->ver1, aux->ver2);
    fprintf(fs," en superf. (%d %d) Angulo (%2.4f), Vertices (%d %d)\n",
        aux->sup1, aux->sup2, aux->ang, aux->ver1, aux->ver2);
  }
}

void muestrav(vista *head, FILE *fs)
{
  vista *aux;
  for(aux = head->sig_vis; aux != NULL; aux = aux->sig_vis)
  {
    printf("vista %d:\n{ (%3.4f %3.4f %3.4f), (%3.4f %3.4f %3.4f), (%3.4f, %3.4f,
%3.4f) }\n"
        ,aux->id_vis,aux->v1.x,aux->v1.y,aux->v1.z,aux->v2.x,aux->v2.y,aux->v2.z
        ,aux->v3.x, aux->v3.y, aux->v3.z);
    fprintf(fs,"vista %d:\n{ (%3.4f %3.4f %3.4f), (%3.4f %3.4f %3.4f), (%3.4f, %3.4f,
%3.4f) }\n"
        ,aux->id_vis,aux->v1.x,aux->v1.y,aux->v1.z,aux->v2.x,aux->v2.y,aux->v2.z
        ,aux->v3.x, aux->v3.y, aux->v3.z);
  }
}

void muestravtx(vertex *lista, FILE *fs)
{
  vertex *aux;
  entinver *axeiv;

  printf("\n");
  fprintf(fs,"\n");
  for(aux = lista->sig_ver; aux != NULL; aux = aux->sig_ver)
  {
    printf("vertice %d, (%3.4f, %3.4f, %3.4f) \n Entidades (id, side, tipo): ",
        aux->id, aux->punto.x, aux->punto.y, aux->punto.z);
    fprintf(fs,"vertice %d, (%3.4f, %3.4f, %3.4f) \n Entidades (id, side, tipo): ",
        aux->id, aux->punto.x, aux->punto.y, aux->punto.z);

    for (axeiv = aux->content; axeiv != NULL; axeiv = axeiv->sigentinver)
    {
      printf(" (%d, %d, %d) ",axeiv->id, axeiv->side, axeiv->tipo);
      fprintf(fs," (%d, %d, %d) ",axeiv->id, axeiv->side, axeiv->tipo);
    }
    printf("\n");
    fprintf(fs,"\n");
  }
```

```
void muestraplano(plano *head, FILE *fs)   /* muestra la estructura de planos */
{
  plano *aux;
  simplent *entptr;

  printf("\n\n");
  for(aux = head->sig_sup; aux != NULL; aux = aux->sig_sup)
  {
    printf("sup. %d, parametros: %c = %3.4lf,%3.4lf,%3.4lf\n base = {(%3.4lf, %3.4lf,
%3.4lf) , (%3.4lf, %3.4lf, %3.4lf)}\n",
                    aux->id, aux->despeje, aux->a, aux->b, aux->c,aux->v1.x,
                    aux->v1.y, aux->v1.z, aux->v2.x, aux->v2.y, aux->v2.z);
    printf("desplazamiento : (%3.4lf, %3.4lf, %3.4lf)\n",aux->delta.x, aux->delta.y, aux-
>delta.z);
    printf("ENTIDADES :\n");
    fprintf(fs,"sup. %d, parametros: %c = %3.4lf,%3.4lf,%3.4lf\n base = {(%3.4lf,
%3.4lf, %3.4lf) , (%3.4lf, %3.4lf, %3.4lf)}\n",
                    aux->id, aux->despeje, aux->a, aux->b, aux->c,aux->v1.x,
                    aux->v1.y, aux->v1.z, aux->v2.x, aux->v2.y, aux->v2.z);
    fprintf(fs,"desplazamiento : (%3.4lf, %3.4lf, %3.4lf)\n",aux->delta.x, aux->delta.y,
aux->delta.z);
    fprintf(fs,"ENTIDADES :\n");

    for(entptr = aux->sig_ent; entptr != NULL; entptr = entptr->sig_ent)
    {
        printf("  ent %d, coords = (%3.4lf, %3.4lf) (%3.4lf, %3.4lf) (%3.4lf,
%3.4lf)\n",entptr->id,
            entptr->coord1.x, entptr->coord1.y, entptr->coord2.x,
            entptr->coord2.y, entptr->coord3.x, entptr->coord3.y);
        fprintf(fs,"  ent %d, coords = (%3.4lf, %3.4lf) (%3.4lf, %3.4lf) (%3.4lf,
%3.4lf)\n",entptr->id,
            entptr->coord1.x, entptr->coord1.y, entptr->coord2.x,
            entptr->coord2.y, entptr->coord3.x, entptr->coord3.y);
    }
   printf("\n\n");
   fprintf(fs,"\n\n");
  }
}


void muestra2plano(plano *head, FILE *fs) /* muestra la estructura de planos */
{
```

```
  plano *aux;
  simplent *axsent;
  ciclo *axciclo;

  printf("\n\n");
  fprintf(fs,"\n\n");

  for(aux = head->sig_sup; aux != NULL; aux = aux->sig_sup)
  {
    printf("sup. %d, parametros: %c = %3.4lf,%3.4lf,%3.4lf\n base = {(%3.4lf, %3.4lf,
%3.4lf) , (%3.4lf, %3.4lf, %3.4lf)}\n",
                  aux->id, aux->despeje, aux->a, aux->b, aux->c,aux->v1.x,
                  aux->v1.y, aux->v1.z, aux->v2.x, aux->v2.y, aux->v2.z);
    printf("desplazamiento : (%3.4lf, %3.4lf, %3.4lf)\n",aux->delta.x, aux->delta.y, aux-
>delta.z);
    printf("CICLOS :\n");
    fprintf(fs,"sup. %d, parametros: %c = %3.4lf,%3.4lf,%3.4lf\n base = {(%3.4lf,
%3.4lf, %3.4lf) , (%3.4lf, %3.4lf, %3.4lf)}\n",
                  aux->id, aux->despeje, aux->a, aux->b, aux->c,aux->v1.x,
                  aux->v1.y, aux->v1.z, aux->v2.x, aux->v2.y, aux->v2.z);
    fprintf(fs,"desplazamiento : (%3.4lf, %3.4lf, %3.4lf)\n",aux->delta.x, aux->delta.y,
aux->delta.z);
    fprintf(fs,"CICLOS :\n");

    for(axciclo = aux->sig_ciclo; axciclo != NULL; axciclo = axciclo->sig_ciclo)
    {
        printf("\n  ciclo %d (%c), ENTS:",axciclo->id,axciclo->sign);
        fprintf(fs,"\n  ciclo %d (%c), ENTS:",axciclo->id,axciclo->sign);
        axsent = axciclo->sig_ent;
        do
        {
          printf(" %d (%c), ",axsent->id, axsent->sign);
          fprintf(fs," %d (%c), ",axsent->id, axsent->sign);
          axsent = axsent->sig_ent;
        }
        while (axsent != axciclo->sig_ent);
        printf("\n xmin : %3.4lf, xmax : %3.4lf, ymin : %3.4lf, ymax : %3.4lf \n",
              axciclo->xmin, axciclo->xmax, axciclo->ymin, axciclo->ymax);
        fprintf(fs,"\n xmin : %3.4lf, xmax : %3.4lf, ymin : %3.4lf, ymax : %3.4lf \n",
              axciclo->xmin, axciclo->xmax, axciclo->ymin, axciclo->ymax);

    }
    printf("\n");
    fprintf(fs,"\n");
  }
```

```
  printf("\n\n");
  fprintf(fs,"\n\n");
}


void muestra3plano(headplano, fs)
plano *headplano;
FILE *fs;
{
  void m3pl_rec();

  plano *supact;

  printf("Orden de planos :\n");
  fprintf(fs,"Orden de planos :\n");
  for (supact = headplano->sig_sup; supact != NULL; supact = supact->sig_sup)
  {
    printf("Plano %d (%3.4f, %3.4f %3.4f)\n", supact->id, supact->dir.x,
                                              supact->dir.y, supact->dir.z);
    fprintf(fs,"Plano %d (%3.4f, %3.4f %3.4f)\n", supact->id, supact->dir.x,
                                              supact->dir.y, supact->dir.z);
    m3pl_rec(supact->sig_ciclo,fs);
  }
  printf("\n");
  fprintf(fs,"\n");
}

void m3pl_rec(cyclo,fs)
ciclo *cyclo;
FILE *fs;
{
  int i;

  if (cyclo != NULL)
  {
    for(i = 0; i <= cyclo->prof; i++)
        printf("  ");
    printf("Id:  %d, Prof.:  %d\n", cyclo->id, cyclo->prof);
    fprintf(fs,"Id:  %d, Prof.:  %d\n", cyclo->id, cyclo->prof);
    m3pl_rec(cyclo->ciclo_int);
    m3pl_rec(cyclo->sig_ciclo);
  }
}

void muestratunel(headtunel,fs)
```

```
tunel *headtunel;
FILE *fs;
{
  tunel *a;

  printf("TUNELES\n");
  fprintf(fs,"TUNELES\n");
  for(a = headtunel->sig; a != NULL; a = a->sig)
  {
    printf("Tunel %d, tipo %d Ents a1 %d, a2 %d, l1 %d, l2 %d",
          a->id, a->tipo, a->idarc1, a->idarc2, a->idlin1, a->idlin2);
    printf(" signos %c, %c, %c, %c. Vertices %d %d %d %d\n",
          a->sgna1, a->sgna2, a->sgnl1, a->sgnl2, a->v1, a->v2, a->v3, a->v4);
    fprintf(fs,"Tunel %d, tipo %d Ents a1 %d, a2 %d, l1 %d, l2 %d",
          a->id, a->tipo, a->idarc1, a->idarc2, a->idlin1, a->idlin2);
    fprintf(fs," signos %c, %c, %c, %c. Vertices %d %d %d %d\n",
          a->sgna1, a->sgna2, a->sgnl1, a->sgnl2, a->v1, a->v2, a->v3, a->v4);
  }
  printf("\n");
  fprintf(fs,"\n");
}


void muestracil(headcil, fs)
cilindro *headcil;
FILE *fs;
{
  cilindro *a;

  printf("CILINDROS\n");
  for(a = headcil->sig; a != NULL; a = a->sig)
  {
    printf("Cilindro %d, tipo %d Ents c1 %d, c2 %d, l1 %d",
          a->id, a->tipo, a->idcir1, a->idcir2, a->idlin1);
    printf(" signol %c, Vertices %d & %d\n",
          a->sgnl1, a->v1, a->v2);
    fprintf(fs,"Cilindro %d, tipo %d Ents c1 %d, c2 %d, l1 %d",
          a->id, a->tipo, a->idcir1, a->idcir2, a->idlin1);
    fprintf(fs," signol %c, Vertices %d & %d\n",
          a->sgnl1, a->v1, a->v2);
  }
  printf("\n");
  fprintf(fs,"\n");
}
```

APENDICE B Tópicos computacionales en CAPP.

B.1. Tipos de modelos.

Modelar significa seleccionar un conjunto de características relevantes de un producto o proceso, representarlas en una forma que facilite su análisis como unidad y su síntesis con un medio. Modelamos un proceso de inventario para encontrar la política óptima de requisición de materiales, un sistema de líneas de espera para encontrar el mejor número de servidores, un producto para simular su comportamiento físico, conocer sus demandas de factores de producción, evaluar el diseño actual, para mejorarlo.

Se considera que un diseño es mejorable cuando puede aproximar a la empresa a su objetivo (máxima utilidad a los socios, a la sociedad y al consumidor). Se dice que es óptimo cuando ya no es mejorable, la causa de la no mejorabilidad indudablemente es un freno tecnológico. Como la tecnología evoluciona contínuamente, un diseño siempre es mejorable y debe evaluarse periódicamente.

No existe un modelo perfecto. Al momento de representar un objeto de una forma más comprensible o manejable para nostoros, lo estamos privando de ciertas características y estamos resaltando otras. El objetivo del diseño de solidos es que a partir del diseño, podamos llegar a una interpretación visual o conceptual y tomar las acciones convenientes. La figura B.1. nos muestra una clasificación de los modelos de sólidos, donde las flechas nos indican el camino habitual de la información.

B.1.1. Modelos gráficos.

Un modelo gráfico, tiene mayor interés en el dibujo que en el objeto, este representa un sólido en función de líneas, arcos y características gráficas. Manualmente un diseñador imagina un

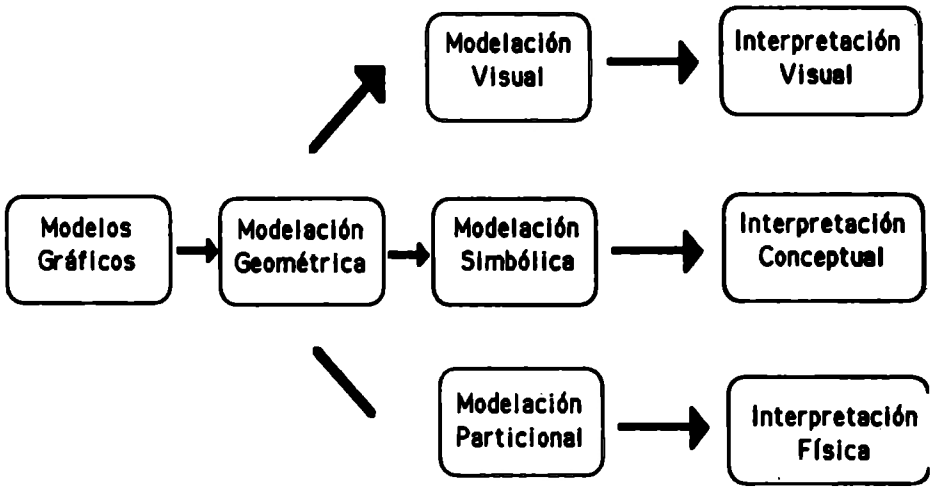


Figura B.1. Diferentes Tipos de Sólidos

producto, y para dar forma a su idea, la dibuja en un plano. Es usual que se un diseño computacional parta también de estas características, aún cuando lo puede hacer de características geométricas o descriptivas.

La fase del diseño, es la fase más humana y artística del proceso, es la parte donde se explota una de las más grandes cualidades humanas: la capacidad de crear. La computadora es una gran herramienta, nos da el poder de la edición y la simulación en el diseño, pero no puede substituir al ser en este arte.

La mayoría de los sistemas de diseño asistido por computadora, se basan en su antecesor manual: crear un diseño en un plano. Esto es diseñar un cuerpo de 3 dimensiones partiendo de la construcción en 2 dimensiones, y agregando información sobre la tercera. Esta información se agrega mediante la proyección de un plano hacia otro, la rotación de la figura, o su duplicación a otra posición en el espacio.

B.1.1.1. Modelo de armazón (wire frame).

Llamaremos entidad a una unidad básica de dibujo en dos dimensiones, una línea, un punto, un arco, un círculo, un polígono, líneas especiales (que definan funciones creadas por el diseñador o por una interpolación). Cada entidad tendrá atributos, que pueden ser espaciales o conceptuales (color, grueso y tipo de linea, identificador), que la definen de manera única.

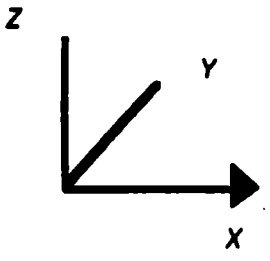
El diseñador parte ante un monitor en blanco -de dos dimensiones-, el monitor representa una proyección ortogonal del espacio sobre un plano (vista), y debe de plasmar en este monitor su idea del diseño.

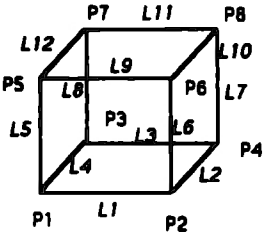
Es usual que parta de una vista paralela a algún plano coordenado y mediante opciones de menú, irá generando entidades, hasta lograr una figura plana (una base o un corte lateral). El efecto tridimensional, se logra al proyectar una selección de entidades una o varias veces de un plano a otro, uniendo las partes respectivas mediante líneas o arcos obteniendo la imagen de las aristas (cóncavas, convexas y suavizadas) de un sólido. A esta representación se le conoce como representación de armazón.

B.1.1.2. Estructura de datos de un modelo gráfico.

La estructura de datos de un modelo gráfico se concentrará en manejar la información proveniente de las entidades. La tabla B.2. nos muestra los atributos para las entidades de CADKEY.

Mantyla[23] sugiere una estructura donde la única información espacial la contiene un tipo llamado punto. Existe una





Punto	X	Y	Z
P1	0	0	0
P2	1	0	0
P3	0	1	0
P4	1	1	0
P5	0	0	1
P6	1	0	1
P7	0	1	1
P8	1	1	1

Identidad	Pto Ini	Pto Fin
L1	P1	P2
L2	P2	P4
L3	P4	P3
L4	P3	P1
L5	P1	P5
L6	P2	P6
L7	P4	P8
L8	P3	P7
L9	P5	P6
L10	P6	P8
L11	P8	P7
L12	P7	P5

Figura B.2. Estructura de Datos de un módelo Geométrico

La representación gráfica es la forma más natural de comenzar un diseño, y en ocasiones requiere de información adicional. Esta representación puede ser ambigua , o no válida. La ambiguedad se puede eliminar restringiendo los objetos representables (limitando el número de aristas que un vertice puede unir a 3) o mediante información adicional. Existen algoritmos para obtener todos los sólidos representables por un modelo de armazón [23].

El sistema computacional desarrollado, utiliza como información de entrada, la información espacial de entidades de CADKEY, como el sistema no persigue fines de visualización ignora los demás atributos.

tipo de entidad	atributos geométricos
PUNTO	Posición, Identificador
LINEA	Posición inicio. Posición final. Identificador
ARCO o CIRCULO	Posición centro. Radio. Angulo de inicio. Incremento angular. Identificador.
POLIGONO	Número de Vertices. Posición de cada vertice (ordenado).

Tabla B.2. Atributos geomètricos de algunas entidades de CADKEY.

Cadkey representa internamente a sus entidades mediante la estructura mostrada en la tabla B.2, donde los atributos espaciales de cada entidad son absolutos, esto es, no son referencia a otra entidad. Esta información se puede accesar mediante la generación de un archivo de salida en un lenguaje propio (CADL), pero como es obvio suponer genera toda la información sobre el diseño y resultaría impráctico diseñar un interprete de esta información.

En la fase de programación de esta tesis, se pensó en una iteración directa con la base de datos de CADKEY, mediante un módulo comercial llamado CDE (CADKEY Dynamic Extensions) , que permite al lenguaje C (en un compilador comercial Metaware) una iteración directa con CADL. El lenguaje de programación interno de CADKEY: CADL, es un lenguaje iterativo, jóven y de bajo poder; sus principales limitaciónes es el no contar con almacenamiento dinámico de memoria, ni ser un lenguaje estructurado, ni es ejecutable por pasos (debugger), por lo que no se consideró para desarrollar un sistema. Esta iteración mediante CDE no fué posible, por no contar con un compilador Metaware de C, ni un módulo CDE, en sustitución, se programó una rutina en CDE, que escribiera en un archivo en código ASCII la información adelante detallada.

B.1.2. Modelación Geométrica.

En un modelo geométrico, la información relevante son las características geométricas y topológicas del sólido.
Comenzaremos por definir un sólido como un conjunto de puntos en el espacio (R^3), topológicamente cerrado y acotado.

B.1.2.1. Modelos basados en semiespacios.

Un semiespacio (en R^3), es el conjunto de puntos en el espacio que son limitados por una restricción. Una restricción divide

el espacio en conjuntos perfectamente definidos. Es usual que una restricción sea de la forma f(X) >= 0 (topológicamente cerrada), entonces su complemento es f(X) < 0, X e R^3. Si la función es lineal, se dice que es un semiespacio lineal.

Son de especial interés, los semiespacios cuya función de restricción sea fácil de representar, como una ecuación cuadrática y en particular cuando se trata de un plano.

Los semiespacios lineales se pueden representar en forma vectorial, y son conjuntos convexos y no acotados -no tienen aristas-. Para representar un sólido -que tiene aristas, y es acotado-, se puede utilizar la unión de la intersección de varios semiespacios -combinación booleana-.

Cuando se puede representar un sólido únicamente por la intersección de varios semiespacios lineales, se dice que es un **polítopo**, y es una variedad lineal convexa, representable mediante un sistema de desigualdades. Es fácil observar que para todo sólido existe una infinidad de polítopos que lo contengan; al polítopo de menor volumen se le conoce como **envoltura convexa** (convex hull).

Este método de representación tiene varias ventajas, entre ellas están el no ser una representación ambigua, y el ser relativamente fácil de manejar computacionalmente (si consideramos únicamente restriciones lineales y cuadráticas), sin embargo, es difícil de conceptualizar un diseño en términos de semiespacios para un diseñador.
La figura B.8. nos muestra un sólido contruido como una combinación booleana de semiespacios.

B.1.2.2. Geometría Sólida Constructiva (CSG).

Otro enfoque en la modelación de sólidos, es la geometría sólida constructiva, y parte de la idea de crear un sólido a partir de

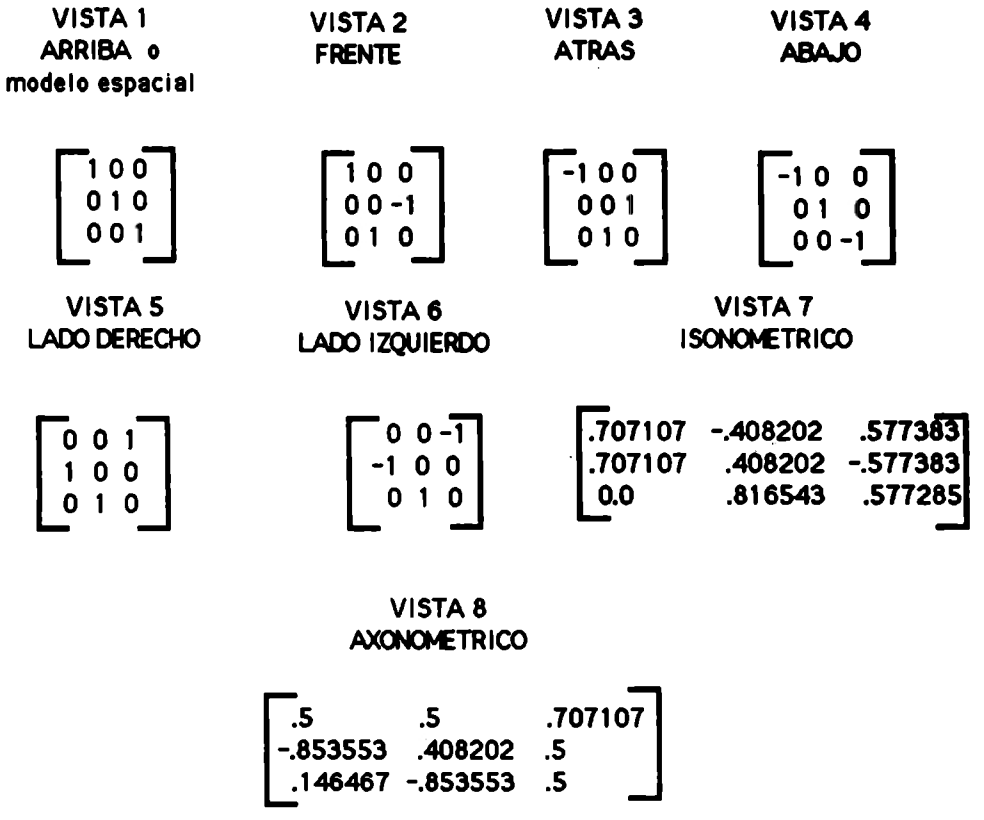


Figura B.3. Vistas Predefinidas por CADKey

solidos básicos o primitivos utilizando sobre ellos operaciones básicas de conjuntos -union, intersección, diferencia-.

Un modelo CSG, se puede representar como un árbol binario, donde las hojas serán los sólidos primitivos, los nodos intermedios las operaciones booleanas, y la raíz el solido modelado. Los sólidos primitivos son figuras básicas como bloques, conos, cilindros, esferas, toroides o cuñas. La figura B.4. nos muestra los sólidos primitivos que soporta el módulo SOLIDS, de CADKEY.

Algunas características de esta representación son:

Las operaciones son cerradas, si los primitivos son sólidos válidos, cualquier árbol lo será.
Un árbol representa de forma única a un sólido (aunque este no es único).
El cálculo de las intersecciones entre los primitivos es computacionalmente difícil, y puede ser explosivo si el número de primitivos crece.

Esta técnica también se complica al momento de que el diseñador quiere plasmar su idea en la computadora. En microcomputadora, una interfase gráfica para CSG tiende a ser lenta, pero aún que fuera rápida, exiten problemas de visualizacón naturales al manipular conceptos de tres dimensiones en dispositivos de dos.

Para manufactura se requiere información sobre las superficies del sólido para asignar parámetros de maquinado, pero es complejo obtener cualquier información sobre los límites de la figura, pues representa el recorrido del árbol binario.

B.1.2.3. Modelado exterior. (Boundary representation)

Ya que un sólido es un conjunto cerrado y acotado, podemos pensar que si definimos su cerradura topológica, definiremos completamente al sólido.
Es más, podemos pensar en un sólido como como en un conjunto de superficies, que son unidas por sus límites (aristas del sólido), hasta que no queda superficie que no ha sido completamente relacionada. Cada superficie debe indicar la dirección del interior del sólido.

Es común que esta representación se utilice sólamente si la vecindad de cualquier punto en la superficie es topológicamente equivalente a un disco abierto de R^2, esto es, que se pueda establece una relación uno a uno entre estos dos espacios. en cuyo caso se dice que la representación es un 2-manifold.

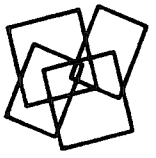
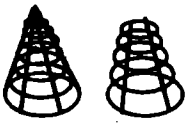
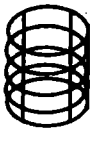
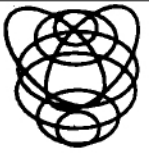
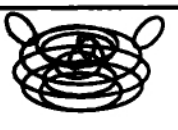
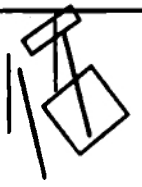
Este concepto sirve para establecer continuidad en el exterior del sólido, pese a la presencia de una arista. De esta forma es posible hacer un recorrido por las superficies del modelo sin ninguna ambiguedad, ya que una superficie se relaciona exactamente con otra en cada arista. También es importante notar que las superficies no se intersectan si no es en las aristas del sólido.

Esta modelación es una consecuencia natural del modelo gráfico de armazón, la transformación de un modelo a otro, es el "rellenado" del armazón.

Se requiere de una estructura de datos que relacione los vértices y las líneas existentes. Como las superficies serán conjuntos de líneas relacionaldos entre sí por sus puntos terminales, es necesario introducir el concepto de ciclo. Un ciclo es la forma de representar una superficie mediante las entidades que forman su perímetro; al definir una superficie, involucra el concepto de pertenencia: Una entidad pertenece a un ciclo si está contenida en su totalidad en la superficie definida por este (si está contenida parcialmente, se trata de un error), un ciclo pertenece a otro ciclo si todas las entidades que lo conforman pertenecen a el (de nuevo una pertenencia parcial implica error).

Definimos un ciclo como un conjunto ordenado de vértices, o como un conjunto ordenado de aristas, dependiendo el enfoque de nuestro modelo. El orden o dirección del ciclo, puede decir si se trata de una intrusión o de una extrusión.

Figura 3.4. Sólidos Primitivos de SOLIDS de CADKey

Figura	Parámetros
 Bloque	Plano de su base Tamaño (Incremento en x, y, z) Posición de la esquina inferior izquierda tras.
 Cono	Longitud Radio de la base Radio menor Eje direccionado.
 Cilindro	Longitud Radio Plano base
 Esfera	Radio Posición del Centro
 Toroide	Radio Radio interno Plano base
 Cuña	Plano base Tamaño -incrementos en x, y, uno para z. Posición esquina inf.izq.tras.

Se necesita una estructura de datos que relacione las caras de la figura mediante aristas y vértices. Es común la estructura del tipo "Winged-Edge", donde cada arista es relacionada por excactamente dos caras, y por dos vértices. (Figura B.5).

B.1.3. Modelación particional.

La modelación particional visualiza al sólido como la yuxtaposición de un grupo de celdas. Una celda es un sólido topológicamente equivalente a una esfera, y se busca que sean de fácil modelación. Las celdas pueden tener en común puntos de su cerradura, pero no de su interior. Al descomponer un sólido en celdas, resulta más claro el análisis de sus propiedades volumétricas, por lo que este tipo de modelos se utilizan para una interpretación de las propiedades físicas (CAE).

B.1.4. Modelación visual.

La visualización, es una trasnformación de un cuerpo en el espacio, sobre un plano. Esta transformación no tiene inversión algebráica por la reducción de una dimensión, y es un área de estudio de inteligencia artificial desarrollar algoritmos que aproximen lo mejor posible a la transformación inversa. Este tipo de modelos, se trata por lo general, de un conjunto de puntos que no tienen relación entre sí, pero que sus atributos de desplegado son determinados por la aplicación de algoritmos que resalten las características visuales a un modelo geométrico (contrastes, sombras, puntos de iluminación, perspectivas, tonalidades).

B.1.4. Modelación simbólica.

La representación simbólica de un sólido, es la representación de este mediante conceptos parametrizados. Intenta simular

la descripción que una persona haría del sólido. "Es una especie prismática, un paralelepípedo, sus ángulos interiores miden 30, 90, 45 y 45 grados (por parejas contraesquinadas), está hecho de acero ... , etc. ". Es difícil utilizar una representación simbólica para propósitos de diseño, pero para fines de reconocimiento de sólidos se utiliza una modelación simbólica orientada a características -barrenos,

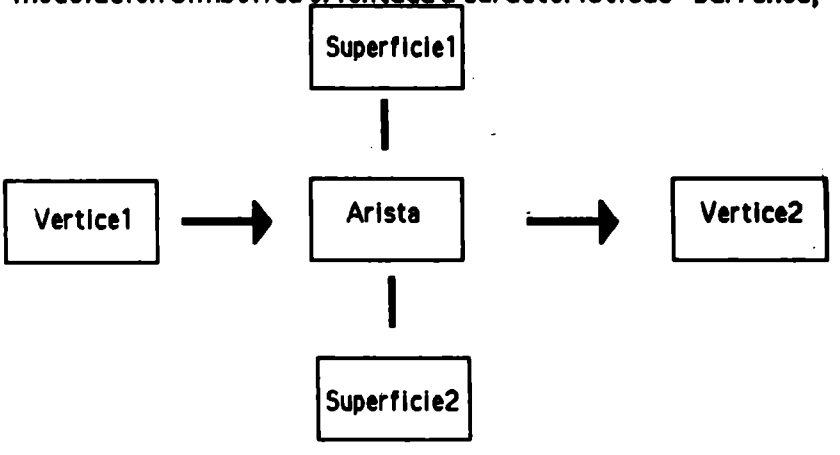


Figura B.5 Estructura Tipo " Winged-Edge"

canales, etc-. La modelación simbólica se puede construir a partir de un diseño iterativo con el usuario, o reconociendo el sólido representado por alguna otra modelación.

B.2. Esquemas de reconocimiento geométrico.

Podemos dividir los esquemas de reconocimiento lógico en los que son de partición geométrica, los lógicos y los analíticos.

B.2.1. Partición geométrica.

Da Vinci, afirmaba que para esculpir un elefante, sólamente había que conseguir una pieza de mármol, y quitarle todo lo que no se pareciera a un elefante. Estos modelos, particionarán un modelo geométrico en volúmenes que puedan ser removidos mediante técnicas de manufactura.

En una definición pobre, el maquinado es la remoción de material, mediante la intervención de una herramienta y un movimiento aplicados a una pieza en bruto (materia prima). Esto infiere la existencia anterior de un solido que cubre completamente al objetivo de manufactura, y de una resta geométrica. El resultado de esta resta, es lo que obendríamos si uniéramos toda la viruta desprendida en su posición original.

La partición geométrica busca identificar los volúmenes faltantes para otro sólido llamado materia prima, ya sea como características solas o como una combinación de estas. La identificación de estos sólidos negativos puede realizarse por diversas técnicas, una de ellas es la descomposición en pequeñas celdas del sólido, y su faltante será ya superposición de celdas.

B.2.2. Partición lógica.

Estas son propuestas de esquemas de razonamiento sobre un sólido. El objetivo es reconocer una característica de maquinado, dada la existencia de varias condiciones.

Existe un barreno, si una superficie tiene un círculo en su interior, y este círculo está conectado gráficamente por una línea -en algunos modelos por una malla- con otro círculo de igual tamaño o una proyección de este en otro plano. Este esquema es propio de las propuestas de uso de Inteligencia artificial, y es la continuación de el modelo lógico de un sólido, desgraciadamente el problema se traslada a la construcción de un modelo lógico..

B.2.3. Esquemas analíticos.

Estos esquemas, tratan de utilizar herramientas matemáticas discretas para la comprensión de la semántica del sólido.
Entre estas herramintas se propone el uso de técnicas como:

+Reconocimiento sintáctico. Este es una técnica en teoría de lenguajes. Su aplicación en geometría funciona mediante una conversión de trazos geométricos primitivos dos dimensiones en un conjunto de símbolos llamados terminales, y el diseño de un conjunto de reglas que relacionen estos símbolos para determinar si pertencen o no a un lenguaje. La pertenencia a este lenguaje será el reconocimiento de una característica geométrica de dos dimensiones, como el contorno de un sólido de revolución o de un ranurado. Su uso se limita a dos dimensiones, porque la teoría de lenguajes estudia el significado de una secuencia de símbolos terminales, en tres dimensiones es difícil visualizar, y más aún estandarizar a un sólido como una secuencia de superficies. Un esquema de reconocimiento sintáctico se presenta en la figura 4.1

+Autómatas. Se definió un autómata en el capítulo 2. Un esquema de reconocimiento geométrico por autómatas, basa su transición entre estados, por relaciones geométricas entre superficies. Kakino (clasifica esas relaciones como adyacencia cóncava o convexa. Este esquema conserva la definición de una secuencia como requisito para su uso.

+ Teoría de grafos. Un grafo es un conjunto de elementos llamados nodos o vértices, relacionados por otro conjunto de aristas. Esta relación puede llevar una dirección -en cuyo caso se llamará digrafo- o ser siempre bidireccional. Joshi desarrolla el concepto de grafos de atributos de adyacencia , donde asigna un elemento de un nuevo conjunto de atributos a cada arista del grafo. Chang [4.4] propone un modelo de reconocimiento de características desarrollando un tipo de grafos que nombra AAG (Attributed Adjacency Graph), donde le proporciona un atributo a cada arista (del grafo). Los nodos del grafo representan las caras del sólido, y el atributo representa la relación entre caras -cóncava o convexa-.

BIBLIOGRAFIA

[1]
Abellanas,M. Lodares, D.
Matemática Discreta
Macrobit ra-ma, 1991

[2]
Bazaraa, Mokhtar. Jarvis, John J.
Programación Lineal y Flujo en Redes
Editorial Limusa, 1981

[3]
Bazaraa, Mokhtar. Shetty, C.M.
Nonlinear Programming, Theory and Algorithms
John Wiley & Sons, 1979

[4]
Burch, John. Strater, Felix.
Information Sistems: Theory and Practice
Hamilton Publishing Company, 1974

[5]
Chang, Tien-Chien.
Expert Process Planning for Manufacturing
Addison-Wesley Publishing, 1990

[6]
Chang , Tien-Chien. Joshi, Sanjay.
Computer Aided Process Planning,
Planning and Design Issues in the Automated Factory
D.I. Cleland and B. Bidanda, TAB Books INC. , 1989.

[7]
Chang, Tien-Chien. Wysk, Richard.
An Introduction to Automated Process Planning Systems.
Integration CAD and CAM through automated process planning.
Prentice Hall
1985, Vol. 22 No. 5 877-894

[8]
Corme, Thomas H. Leiserson, Charles. Rivest, Ronald L.
Introduction to Algorithms
McGraw Hill Book Company, 1992

[9]
Faux. Pratt.
Computational Geometry for Design and Manufacture
Ellis Horwood Limited Publishers,1981

[10]
Foley, James D. ; van Dam Andries , Feiner Steven K. , Hughes John F.
Computer Graphics: Principles and Practice
Addison Wesley Publishing Company,1990

[11]
Gaylord, John
Factory Information Systems.
Marcel Dekker, Inc, 1987

[12]
Grasa S., Pedro
Ingeniería de Manufactura
ITESM Campus Estado de México, 1992

[13]
Graves, Gerald. Yelamanchili, Balaji. Parks, Charles M.
An Interface Architecture for CAD/CAPP Integration Using
Knowledge Based Systems and Feature Recognition Algorithms.
International Journal of Computer Integrated Manufacturing.
1988, Vol. 1, No. 2, 89-100 pp.

[14]
Grimaldi, Ralph P.
Matemática discreta y combinatoria
Addison Wesley Iberoamericana, 1985

[15]
Groppetti, R. Semeraro,Q.
CAPP using relational databases
ISATA
1986, October pp 6-10

[16]
Hitomi, Katsundo.
Manufacturing Systems Engineering
Taylor & Francis LTD., 1979

[17]
Houtzeel, Alexander.
Computer Assisted Process Planning
-a first step towards integration-
Proceedings of Autofact West, Society of Manufacuring Engineers.
1980, Nov. pp 17-20

[18]
Klishinami,T. Kanai, S. Saito, K.
Robotics & Computer-Integrated Manufacturing.
An Integrated Approach To CAD/CAPP/CAM
Based on Cell-Constructed-Geeometric Model (CCM).
1987, Vol.3, No.2. ,215-220 pp.

[19]
Lang, Serge
Introducción al Algebra Lineal
Addison-Wesley Iberoamericana, 1990

[20]
Liu, Richard. Srinivasan,Ramesh.
Compueters in Mechanical Engineering.
Generative Process Planning Using Syntatic Pattern Recognition.
1984, Marzo. 63-66 pp.

[21]
Luenberger, David E.
Programación lineal y no lineal
Addison Wesley Iberoamericana, 1989

[22]
Machover, Carl
The C4 Handbook CAD, CAM, CAE, CIM
TAB professional and reference books 1989

[23]
Mäntylä, Martti
An Introduction to Solid Modeling
Computer Science Press, Inc.,1988

[24]
Morteson, M.E.
Computer Graphics: An introduction to the Mathematics and
Geometry

[25]
Nolen, James
Computer-Automated Process Planning for world class manufacturing
Marcel Dekker Inc., 1989

[26]
Phillips, R.H. Arunthavanathan, v. Zhou, X.
PED, Computer Aided Intelligent Process Planning
Symbolic Representation of CAD Data for
Artificial Intelligence Based Process Planning.
ASME WAM, 1985, Vol. 19, Nov 17-22, pp 31-42

[27]
Sánchez, José M.
Desarrollo Integral de Productos,
Un Enfoque para la Manufactura de Clase Mundial
Centro de Inteligencia Artificial, ITESM Campus Monterrey

[28]
Sedgewick, Robert
Algorithms in C
Addison-Wesley Publishing, 1990.

[29]
Trusky, H.
American Machinist.
Automated Planning Reduces Costs.
1984, April.