FACULTAD DE INFORMÁTICA

UNIVERSIDAD POLITÉCNICA DE VALENCIA

RECONSTRUCCIÓN 3D DE OBJETOS POLIÉDRICOS A PARTIR DE SU REPRESENTACIÓN AXONOMÉTRICA OBLICUA 2D

PROYECTO FIN DE CARRERA.

PRESENTADO POR JOSÉ GARCÍA RESTA.

DIRIGIDO POR DR. D. JOSÉ MARÍA GOMIS MARTÍ.

VALENCIA, ABRIL 1997.

A mis padres

RECONSTRUCCIÓN 3D DE OBJETOS POLIÉDRICOS A PARTIR DE SU REPRESENTACIÓN AXONOMÉTRICA OBLICUA 2D

ÍNDICE

1. PRELIMINARES..... 2

1.1 Introducción. 2

1.2 Representación de objetos sólidos. 4

1.3 El diseño gráfico.....5

1.4 Descripción del proyecto. 7

2. REVISIÓN HISTÓRICA. 8

2.1 Introducción. 8

2.2 Clasificación de los métodos.....12
2.2.1 Métodos de vistas múltiples. 12
2.2.1.1 Métodos Brep.....12
2.2.1.2 Métodos CSG.....12
2.2.2 Métodos de vista única.....13
2.2.2.1 Método Perceptual. 13
2.2.2.2 Método Perceptual: Conclusiones.....15

2.3 Breve Resumen.....16

3. OBJETIVOS. 17

4. DESCRIPCIÓN DE LAS AXONOMETRÍAS..... 18

4.1 Proyecciones. 18
4.1.1 Introducción. 18
4.1.2 Proyección perspectiva. 19
4.1.3 Proyección paralela. 19

4.2 Sistemas de representación gráfica.....20
4.2.1 Introducción. 20
4.2.2 Sistema axonométrico. 20

5. HERRAMIENTAS Y ENTORNO DE DESARROLLO. 23

5.1 El lenguaje de programación c. 23

5.2 El entorno windows 95. 24

5.3 El paquete gráfico AutoCAD 12. 25
5.3.1 ADS y la programación en C. 27
5.3.1.1 Estructura de una aplicación ADS. 27
5.3.1.2 ADS para Windows. 29
5.3.1.3 Estructura de una aplicación ADS para Windows. 29

6. DESCRIPCIÓN DEL ALGORITMO. 31

6.1 Preliminares. 31
6.1.1 Introducción. 31
6.1.2 Fases del algoritmo. 31
6.1.3 Esquema del algoritmo. 34
6.1.4 Estructura de datos. 37
6.1.4.1 Vértices. 37
6.1.4.2 Aristas. 37
6.1.4.3 Caras. 38
6.1.4.4 Estructuras auxiliares. 38

6.2 Grafos. Generalidades. 40
6.2.1 Introducción. 40
6.2.2 Definiciones. 40
6.2.3 Caminos y conexión. 41

6.3 Teorema de euler 42
6.3.1 Superficie poliédrica. Poliedro convexo. 42
6.3.2 Propiedades de los poliedros convexos. 42
6.3.3 Superficies poliédricas y poliedros no convexos. 43
6.3.4 Teorema de Euler. 44
6.3.5 Consecuencias del teorema de Euler. 44
6.3.6 Poliedros Eulerianos. 44

6.4 Identificación de las caras. 46
6.4.1 Obtención de los vértices y aristas. 46
6.4.1.1 Obtención de los vértices. 46
6.4.1.2 Obtención de las aristas. 47
6.4.2 Restricciones en la identificación de las caras. 47
6.4.3 Técnica de búsqueda en profundidad. 48
6.4.4 Algoritmo general de identificación de las caras en 2D. 48
6.4.5 Algoritmo de búsqueda de bucles o ciclos. 50
6.4.5.1 Descripción del algoritmo. 50
6.4.5.2 Esquema del algoritmo. 51
6.4.5.3 Ejemplo. 54
6.4.6 Análisis de la valencia de un vértice. 56
6.4.7 Tratamiento de las posibles caras de más de cinco lados. 57

6.5 Obtención de la figura en 3D. Obtención de vértices y aristas del modelo. 60
6.5.1 Introducción. 60
6.5.2 Introducción de los ejes. 60
6.5.3 Transformación 3D de las aristas. 61
6.5.3.1 Transformación de las aristas coincidentes con los ejes. 62
6.5.3.2 Transformación de las aristas con un vértice ya transformado a 3D y paralelas a uno de los ejes de coordenadas. 62
6.5.3.3 Obtención del resto de aristas. 63
6.5.3.4 Transformación del resto de aristas. 69
6.5.3.5 Ejemplo. 69

6.6 Construcción de las caras en 3d.....71
6.6.1 Introducción.....71
6.6.2 Orden 3DCARA.....71
6.6.3 Caras cóncavas y convexas.....72
6.6.4 Algoritmo para la construcción de las caras en 3D.....74
6.6.4.1 Forma de calcular la concavidad, convexidad de una arista.....75
6.6.4.2 Caras convexas.....77

6.7 Coste de los algoritmos.....79
6.7.1 Eficiencia de los algoritmos.....79
6.7.2 Medición del tiempo de ejecución de un programa.....79
6.7.3 Cálculo del tiempo de ejecución de un programa.....79
6.7.4 Análisis de programas recursivos.....80
6.7.5 Análisis de los algoritmos.....80
6.7.5.1 Módulo ID.....81
6.7.5.2 Módulo OMA.....86
6.7.5.3 Módulo OMS.....87
6.7.5.4 Programa principal.....90

7. EJEMPLOS.....92

7.1 Ejemplo 1.....92

7.2 Ejemplo 2.....94

7.3 Ejemplo 3.....96

7.4 Ejemplo 4.....98

7.5 Ejemplo 5.....100

8. DESARROLLOS FUTUROS.....102

8.1 Desarrollos a corto plazo.....102

8.2 Desarrollos a medio y largo plazo.....103

9. CONCLUSIONES.....104

10. BIBLIOGRAFÍA.....105

ÍNDICE FIGURAS

Figura 1: (a) Dibujo lineal con etiquetado. (b) un etiquetado sin sentido. 9
Figura 2: Axonometrías isométrica y oblicua 14
Figura 3: Proyección....18
Figura 4: Sistema axonométrico....21
Figura 5: Representación de un punto en el sistema axonométrico....22
Figura 6: Ejemplo de una cadena....41
Figura 7: Poliedro no convexo....43
Figura 8: Superficie noNo simplemente conexa 45
Figura 9: Poliedro no euleriano....45
Figura 10: Obtención de vértices 46
Figura 11: Ejemplo de obtención de las posibles caras de una figura....54
Figura 12: Posible cara verdadera....55
Figura 13: Posible cara falsa....55
Figura 14: Retroceso....56
Figura 15: Posible cara falsa....56
Figura 16: Posible cara verdadera....56
Figura 17: Valencia de un vértice 57
Figura 18: Ejempplo 58
Figura 19: Algunas de las posibles caras encontradas por el algoritmo de búsqueda de ciclos 58
Figura 20 Caras verdaderas....59
Figura 21: Obtención de vértices y aristas 65
Figura 22: Teorema de Thales 66
Figura 23: Determinación de puntos de una recta por su razón de distancias a dos de ellas....66
Figura 24: Aplicación del teorema de Thales....67
Figura 25: Aplicación de la orden 3DCARA 71
Figura 26: Cara convexa 72
Figura 27: Cara cóncava 72
Figura 28: Tipos de aristas (1) 73
Figura 29: Tipos de aristas (2) 73
Figura 30: Cálculo de la pendiente 76
Figura 31: División de una cara cóncava 77
Figura 32: División de una cara convexa en caras triangulares....78
Figura 33: División de una cara cóncava en caras triangulares....78

ÍNDICE ESQUEMAS

Esquema 1: Fases del algoritmo.....32
Esquema 2: Identificación de las caras en 2D.....34
Esquema 3: Obtención del modelo alámbrico.....35
Esquema 4: Obtención del modelo de superficies.....36
Esquema 5: Procedimiento para la búsqueda de ciclos.....53
Esquema 6: Módulo para la construcción de las caras en 3D 75

RECONSTRUCCIÓN 3D DE OBJETOS POLIÉDRICOS A PARTIR DE SU REPRESENTACIÓN AXONOMÉTRICA OBLICUA 2D

MEMORIA

1. PRELIMINARES

1.1 Introducción.

Las proyecciones bidimensionales de objetos tridimensionales se han estudiado desde hace tiempo en geometría descriptiva y proyectiva, en donde el énfasis se ha centrado en cómo describir un objeto en un plano bidimensional. El problema contrario de cómo reconstruir automáticamente la estructura de un objeto tridimensional (estructura geométrica y topológica) a partir de su proyección, empezó a atraer la atención sólo a finales de los 60, con el desarrollo de los ordenadores digitales.

La cuestión de la reconstrucción se considera un área de investigación importante hoy en día. Se han desarrollado distintos modelos formales que le permiten al ordenador adquirir el tipo de conocimiento que los humanos usan para entender las proyecciones; de manera que el ordenador puede, en cierto modo, comprender estas proyecciones como lo hacen los humanos. El problema subyacente de pasar de dos dimensiones a tres dimensiones es tanto teórico como matemático. Preguntas tales como ¿es posible pasar de 2D a 3D? ¿Cuantas soluciones obtendremos?, ¿hay una solución general?, etc., ya son interesantes en si mismas

La importancia de este problema estriba no solo en su valor teórico sino también en sus aplicaciones prácticas.

Áreas tales como la comunicación hombre-máquina y visión artificial para robots de uso industrial, defensa militar, tratamiento médico e investigación geológica, por ejemplo, todas requieren alguna forma de reconstrucción de 2D a 3D. En muchos casos, el reconocimiento del objeto ofrece solo una solución parcial y el problema de la reconstrucción es el que lo valida. En los sistemas de visión artificial se puede aplicar la reconstrucción de objetos, después de reconocer los objetos, para descubrir sus estructuras 3D.

Otro ejemplo proviene de las áreas de ingeniería en las que los dibujos de ingeniería proceden de proyecciones ortográficas de objetos 3D[1]. Estos dibujos/proyecciones 2D son el mecanismo estándar de comunicación entre los departamentos de ingeniería y fabricación. La reconstrucción de modelos 3D a partir de los dibujos lineales de ingeniería es crítica para el mantenimiento y desarrollo continuado de los objetos representados. La automatización de esta actividad permitiría tomar los objetos existentes e incorporarlos a paquetes de modelado de sólidos, donde se podrían modificar gradualmente para fabricar objetos más nuevos y modernos.

Los primeros esfuerzos de reconstrucción se concentraron en objetos poliédricos, y mas recientemente en objetos analíticos no poliédricos. En este estudio revisaremos solo la reconstrucción a partir de las proyecciones 2D de dibujos lineales puros. Por “puro” entendemos que no se consideran la textura, la gama, el sombreado y otros parámetros adicionales. Estos otros parámetros se están usando en el reconocimiento de objetos, pero no se han aplicado aun a la reconstrucción de objetos. Por la misma razón, no revisaremos la investigación sobre trabajos de reconstrucción a partir de imágenes de

[1] M.A. Wesley and G. Markowsky, "Generation of solid models from two-diniensional and three-dimensional data," *Solid Modeling by, Computer: from Theory.to Application,* pp. 23-51, (1986).

cámara, ya sean individuales o como secuencia de imágenes. Cabría señalar que, aunque la mayor parte del trabajo que revisamos aquí se limita a ambientes artificiales (dibujos lineales puros), parte del trabajo se ha ampliado para poder tratar dibujos incompletos, o sea, dibujos lineales mas acordes con la realidad. Es cierto que en el estadio actual de la investigación, la mayoría de los modelos no son adecuados aun para su aplicación directa a la vida real. Sin embargo, creemos que cada vez serán mas útiles como estadio de post-procesamiento de la visión de la realidad con el ordenador, sin olvidar las aplicaciones que puede tener en el campo de la ingeniería.

1.2 Representación de objetos sólidos.

Actualmente existen cuatro modelos principales usados en la representación de objetos sólidos 3D. Estos son la Geometría Constructiva de Sólidos (**CSG**), la Representación de Fronteras (**BRep**), la Representación de Barrido y la Enumeración Espacial. Las representaciones más comúnmente utilizadas en el problema de reconstrucción son la CSG y la BRep.

La CSG fue introducida por primera vez por Requicha y es un método efectivo y directo para la reconstrucción de un objeto sólido. Cada objeto se presenta por una colección de primitivas, y por un conjunto de transformaciones y operaciones booleanas. Las primitivas son sólidos simples concretos, tales como cubos, pirámides, cilindros, conos y esferas. Las transformaciones incluyen la traslación, rotación, y escala, y se usan para definir la posición, orientación y tamaño arbitrario de las primitivas. Las operaciones booleanas consisten en un conjunto de operaciones de unión, intersección y diferencia (regularizados para producir objetos con interiores no vacíos) y se usan como operadores de composición para generar objetos mas complejos. La composición se puede representar por una estructura jerárquica de datos, denominada “árbol CSG”, cuyas ramas representan las primitivas y cuyos nodos internos quedan etiquetados con las operaciones booleanas regularizadas, con sus respectivas transformaciones. La raíz de este árbol CSG se identifica como el objeto deseado y recorrer el árbol proporciona un mecanismo de representación o análisis del objeto. Este método esta conceptualmente próximo a las técnicas que actualmente se usan en ingeniería para el diseño de piezas mecánicas y, a menudo, da una representación compacta del objeto.

La BRep ofrece una representación del objeto mas general pero no necesariamente mas compacta. Un sólido se modela por un número finito de superficies delimitadas, cada una de las cuales se representa por un conjunto de aristas orientadas que la limitan. Cada arista se representa por dos vértices, definidos en algún sistema de coordenadas. Las superficies que se usan normalmente para describir los objetos sólidos incluyen superficies planas (por ejemplo polígonos), superficies cuadráticas (por ejemplo cilíndricas, cónicas y esféricas) y *splines*. Las operaciones de construcción en BRep aseguran que el contorno esta bien formado; esto incluye requisitos tales como que “*el contorno sea cerrado*”.

En algunos casos, más de un objeto puede generar la misma, o mismas, vistas al proyectar. Un método se llama de “solución múltiple” si encuentra todos los objetos que puedan corresponderse con la(s) vista(s) 2D, y de “solución única” si se para después de encontrar un objeto.

1.3 El diseño gráfico.

El diseño gráfico por ordenador ha experimentado en los últimos años un gran auge, favorecido, sobre todo, por la integración de los ordenadores en todos los ámbitos de la sociedad. Y dentro de ella, ha adquirido una notable relevancia en los campos profesionales técnicos y artísticos, fomentados por el uso intensivo de estas poderosas herramientas En particular, disciplinas como la *arquitectura* y el *diseño industrial* utilizan las aplicaciones de diseño de objetos (piezas mecánicas, estructuras arquitectónicas, componentes electrónicos, etc.) como herramienta básica de diseño.

Por otra parte, la gran aceptación que estas herramientas han tenido frente a los métodos habituales han sido motivadas por una serie de ventajas, como son la facilidad de manejo de la información referente al diseño, la sustitución de las reglas y lápices por asépticos trazadores gráficos, precisión del dibujo, ahorro considerable de tiempo, posibilidad de obtener varios dibujos (distintas vistas) almacenando la información en una estructura única y compacta, copia de dibujos, etc. Las temidas reticencias al cambio del método de trabajo se han visto obviadas frente a las grandes virtudes que ofrecen estas nuevas herramientas.

El diseño gráfico por ordenador es un campo amplio y muy diverso, que incluye el estudio de numerosos temas como son los métodos matemáticos para el cálculo de los objetos, modelos de representación y almacenamiento de los objetos, las transformaciones que se pueden realizar al objeto, la interfaz con el usuario, métodos de edición de objetos, etc.

En cualquier caso todos los campos en los que se puede dividir el estudio del diseño tridimensional se combinan entre sí para constituir herramientas prácticas de dibujo y en ocasiones no son totalmente independientes, de forma que, por ejemplo, un método de edición de objetos necesita un interfaz específico.

Dentro del diseño de objetos por ordenador podemos clasificarlo según las dimensiones que utilicemos: bidimensional (2D) o el diseño tridimensional (3D). Por ejemplo una placa de circuitos integrados electrónicos puede ser definido perfectamente en 2D usando representaciones simbólicas o bien en 3D utilizando una imagen sintética de la placa completa.

El diseño en 2D es más sencillo, pues la representación del objeto en la pantalla es directa. Es por ello que existen interfaces entre el programa y el usuario que permiten a éste realizar con mucha sencillez y de manera intuitiva el diseño.

El diseño tridimensional supone una mayor complejidad, no solo por la multiplicación de cálculos a realizar, sino por la complejidad añadida de tener que representar y construir un objeto tridimensional en una pantalla de ordenador; que es bidimensional.

Existen varios procesos para la edición de figuras tridimensionales entre ellos cabe destacar el consistente en girar el punto de observación, de manera que nos permita ir construyendo las entidades geométricas pertenecientes a los distintos planos de la figura, a partir de los cuales obtener el objeto tridimensional. Otro método se fundamenta en obtener el objeto tridimensional a partir de sus proyecciones ortográficas.

Los programas de diseño de objetos tridimensionales deben ser lo suficientemente potentes como para crear **modelos** del objeto que lo describan de forma completa y única.

Los **modelos** pueden ser distintos según las necesidades. Así un *modelo alámbrico* es la forma más sencilla de representar un objeto, lo que puede resultar suficiente para obtener una idea general de un objeto; pero resultaría insuficiente si lo que se desea es realizar manipulaciones del objeto como puedan ser la eliminación de caras ocultas, aplicación de modelos de iluminación, sombreado, etc. Para ello es necesario disponer de un modelo que tenga definidas, además de vértices y aristas, las caras del objeto.

En el proceso de diseño y fabricación asistidos por ordenador (**CAD-CAM**) la reconstrucción del modelo sólido con ordenador es un paso vital.

Tres aproximaciones son utilizadas comúnmente para la creación de modelos sólidos:

1) Modelado mediante primitivas.
2) Barrido traslacional y/o rotacional.
3) Geometría Constructiva de Sólidos (**CSG**).

Estos métodos son los más utilizados en los programas comerciales de diseño gráfico, incluido AutoCAD; que es el paquete gráfico utilizado como herramienta de edición gráfica en la realización del presente proyecto. Como alternativa a estos métodos, cabe considerar la *reconstrucción de modelos 3D* a partir de sus proyecciones. Es una aproximación orientada a dibujo en dos dimensiones y está basada en el substrato clásico de los dibujos de ingeniería. Genera modelos en tres dimensiones partiendo de un conjunto de proyecciones planas, recuperando información de una dimensión superior partiendo de la información existente en una dimensión inferior.

1.4 Descripción del proyecto.

La finalidad del proyecto es conseguir un algoritmo para la reconstrucción del modelo sólido tridimensional a partir de su representación 2D axonométrica oblicua.

El algoritmo debe resolver casos especiales que presentan ambigüedad en su interpretación e identificar las posibles soluciones correctas. Se utilizan técnicas eficientes para la construcción del modelo alámbrico, manejo de información incompleta y redundante; tratamiento de líneas incompletas y vértices que, en principio, no existen como entidad propia, obtención de las caras del objeto y los bucles que describen cada una de ellas y reconstrucción definitiva del modelo sólido. Además, se demuestra la completitud del algoritmo y su complejidad espacial y temporal.

La generación de figuras tridimensionales a partir de su representación 2D axonométrica oblicua atraviesa cuatro fases, en las cuales la información obtenida en una fase se utiliza para la consecución de la siguiente fase siguiendo el método deductivo y cognitivo humano. Lógicamente, la primera fase se alimentará de la información de partida: el objeto representado en 2D.

Las **cuatro** fases en la generación de las figuras tridimensionales consisten en:

1) Obtención de la topología del objeto, vértices y aristas y las propiedades de los mismos, a partir de la axonometría oblicua bidimensional.

2) Identificación de las caras del objeto en el modelo bidimensional.

3) Obtención del modelo alámbrico tridimensional.

4) Transformación del modelo alámbrico a un modelo de superficies.

La fase 1) utiliza la información del objeto representado. Se obtienen los vértices y aristas del objeto, sus coordenadas bidimensionales, el grado o valencia de cada vértice y el paralelismo entre las aristas.

La fase 2) utiliza la información topológica ya obtenida en la fase anterior para , por medio de recorridos y controlando que se cumplan unas reglas determinadas tales como el teorema de Euler, que una arista no puede pertenecer a más de dos caras, etc. se obtienen las distintas caras que forman el objeto, confeccionando así un modelo alámbrico bidimensional.

La fase 3) manipula los datos obtenidos tanto en la fase 1) como en la 2) además de otros solicitados al usuario como son cuatro vértices de la figura que formen los tres ejes de coordenadas, así como las escalas de los mismos, para construir el modelo alámbrico tridimensional del objeto.

La fase 4) es un paso necesario para poder aplicar los algoritmos de renderización, eliminación de aristas ocultas, aplicación de modelos de iluminación y sombreados.

El **modelo** así obtenido es un modelo completo que nos sirve para poder representar sin pérdida de información, de manera inequívoca y completa la estructura topológica y la forma geométrica de un objeto tridimensional.

2. REVISIÓN HISTÓRICA.[2]

2.1 Introducción.

La historia de tratar de comprender el dibujo lineal data de los años 60. El primer intento de interpretación del dibujo lineal fue el de Roberts, quien limitó los objetos a un conjunto de modelos predefinidos. Su enfoque intentaba identificar una transformación que (cuando se aplica a uno de los modelos) da una proyección que encaja con el dibujo lineal, en el sentido de mínimos cuadrados. Claramente, este método basado en modelos es muy limitado.

Intentos posteriores eliminaron esta restricción. Guzman desarrolló un programa llamado SEE para analizar dibujos lineales de poliedros sin usar modelos explícitos de objetos. Su programa incluso identifica objetos parcialmente ocultos. Basándose en algunos conceptos heurísticos sencillos sobre las uniones de los dibujos lineales, SEE examina cada unión y agrupa regiones alrededor de la unión si estas regiones están asignadas al mismo cuerpo. Además SEE fusiona los gráficos según otros supuestos heurísticos y crea nuevos nodos. Como muchas reglas heurísticas no tienen base teórica, SEE puede cometer errores. Sin embargo, ha influido en gran medida en los trabajos de investigación posteriores.

Si se limita el problema a la interpretación de dibujos de vistas múltiples, tales como los dibujos de ingeniería compuestos de las vistas en alzado, planta y perfil, el problema es mas fácil de solucionar. Idesawa trató matemáticamente los dibujos de ingeniería de tres vistas. Wesley y Markowsky dieron una solución perfecta al problema de reconstrucción a partir de vistas múltiples (más de una, pero no necesariamente tres) de un poliedro. Sakurai amplió el algoritmo de Wesley-Maarkowsky para incluir algunos objetos con simetría rotacional, como cilindros o conos. Gu y otros ampliaron posteriormente el algoritmo de Sakurai para suavizar algunas de las restricciones en la orientación de objetos curvos. Otros autores ampliaron gradualmente los algoritmos previos para poder tratar un mundo de objetos más general, y así presentar un enfoque ligeramente diferente, o incluso para intentar obtener información adicional de los textos de los dibujos. Aquí los problemas principales son dos: (1) como establecer una correspondencia entre diferentes vistas y (2) como encontrar una realización 3D usando esa correspondencia y juntando piezas individuales. Aldefeld y Ho reconstruyen la representación CSG para objetos. Pero sus sistemas se apoyan mucho en la interacción con el usuario.

Lafue y Liardet han desarrollado sistemas interactivos para interpretar dibujos lineales de vista única, en donde el problema es cómo realizar interacciones suaves y flexibles entre los usuarios y los ordenadores. Lamb y Bandopadhay presentaron un sistema para determinar estructuras 3D a partir de un dibujo lineal impreciso. Su sistema usa reglas heurísticas, más algunas reglas perceptuales, para intentar ajustar automáticamente el dibujo y reconstruir su estructura 3D. Cuando no lo logra, depende de la interacción del usuario para conseguir que el proceso siga.

[2] Weidong Wang, D.Sc., Georges G. Grinstein, Ph.D., "A Survey of 3D Solid Reconstruction from 2D Projection Line Drawings" *Computer Graphics Forum, Volume 12,* pp. 137-158, (1993).

El problema se hace más desafiante y difícil cuando se pide la interpretación automática o reconstrucción a partir de una sola vista. Aquí se parte del supuesto de punto de vista general para eliminar posibles casos de degeneración. En el campo de la visión artificial se han considerado una serie de esquemas de etiquetado, con el fin de interpretar escenas 3D. Huffman y Clowes exploraron formalmente la configuración de las líneas en las uniones. Descubrieron que las configuraciones posibles de líneas etiquetadas en las uniones forman un subconjunto muy pequeño de entre todas las combinaciones de asignaciones de etiquetas para las líneas en las uniones. La Figura 1(a) muestra un dibujo lineal sencillo etiquetado. Su trabajo, y el de Mackworth y Sugihara, resolvieron formalmente el problema de etiquetado de dibujos que contenían sólo objetos poliédricos. Los análisis y los algortimos resultantes son matemáticamente rigurosos. Es más, la validez de los esquemas de etiquetado se ha verificado en otros tipos adicionales de dibujos lineales, tales como dibujos en los que las líneas ocultas se representan mediante líneas de trazos, dibujos con objetos curvos delimitados por superficies cuadráticas, y dibujos de superficies teóricas sin espesor (u objetos de papel "Origami"). Sin embargo, los esquemas para etiquetar dibujos lineales de escenas que contengan objetos curvos eran heurísticos, incompletos y carecían de justificación matemática adecuada. Malik presentó un esquema matemáticamente riguroso para el etiquetado de dibujos lineales de escenas formadas por objetos sólidos regulares y opacos. Los objetos pueden estar delimitados por superficies lisas en forma de piezas sin marcas ni textura.

Cabría observar que el esquema de etiquetado de Huffman-Clowes, así como otros esquemas de etiquetado publicados, se basan en condiciones necesarias pero no suficientes para que un dibujo lineal represente un objeto 3D. Un dibujo lineal correcto siempre admite una asignación consistente de etiquetas, pero la existencia de un etiquetado consistente no implica que el dibujo lineal sea correcto. La Figura 1(b) muestra un dibujo absurdo etiquetado de forma consistente.

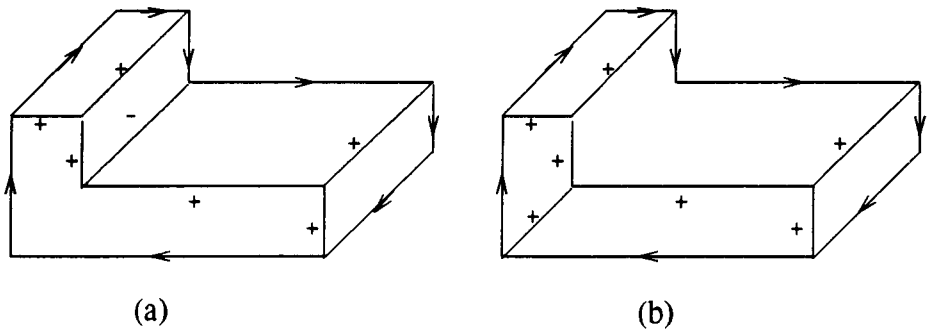


Figura 1: (a) Dibujo lineal con etiquetado. (b) un etiquetado sin sentido.

Debido a que de los esquemas de etiquetado no eran completos, se propusieron varias ideas nuevas. Se han dado varios tipos de condiciones que debería satisfacer un dibujo lineal correcto y se han usado para comprobar la corrección de los dibujos lineales etiquetados. Una de las ideas más llamativas de Mackworth es el uso de figuras reciprocas en un espacio gradiente/dual. Para cada dibujo lineal que representa un poliedro, podemos definir una figura reciproca en el espacio dual: en ella los vértices, aristas y caras se corresponden con las caras, aristas y vértices respectivamente del dibujo lineal original. Así, un dibujo lineal etiquetado se puede juzgar como incorrecto

si no puede admitir su figura reciproca. Sin embargo, son necesarias determinadas condiciones, y por tanto, no se puede discriminar con precisión entre interpretaciones correctas e incorrectas.

Wei amplió el concepto de espacio gradiente de Mackworth a su espacio dual ampliado, que también es un espacio tridimensional. Un plano en el espacio se representará como un punto en su espacio dual. En su espacio dual ampliado, estableció cuatro restricciones que existen en la relación entre un poliedro y su proyección general. Demostró que estas cuatro restricciones son una condición necesaria y suficiente para que exista un poliedro en el espacio 3D. Basándose en estas cuatro restricciones, dio tres reglas de búsqueda de solución para hallar de forma recursiva las superficies planas del poliedro, y una condición para elegir los valores iniciales de la recursión. Dio la solución general del problema, y posteriormente trató la cuestión de la existencia de posibles errores en las posiciones de los vértices y cómo hacer que sus reglas funcionen en tal situación.

Otro método en el problema de la reconstrucción es el uso de la programación lineal para el análisis de los dibujos lineales. Ya se sabía que un poliedro representado por un dibujo lineal debe satisfacer ciertas ecuaciones lineales. Aunque este método parece natural, no ha sido tan aceptado como el método de figuras reciprocas. Bastante recientemente, Sugihara consiguió dar una condición necesaria y suficiente que permitía que un dibujo lineal representase un objeto poliédrico en términos del problema de programación lineal. Así, el problema de discriminar entre dibujos lineales correctos e incorrectos se puede resolver usando su método de programación lineal.

Sin embargo, el método de Sugihara no es práctico. La condición es tan precisa matemáticamente que algunos dibujos son rechazados simplemente porque los vértices se desvían ligeramente de las posiciones correctas. La dificultad estriba en el hecho de que hay ecuaciones redundantes en el sistema. Sugihara soluciona esta dificultad identificando las ecuaciones redundantes y permitiendo extraer un subconjunto de ecuaciones según la latitud de la posición. Así el problema de discriminación de dibujos lineales de poliedros correctos e incorrectos se puede resolver no sólo desde el punto de vista teórico sino también práctico.

En el campo de la reconstrucción de una representación CSG de un sólido, se ha avanzado poco hasta fechas recientes. Wang y Grinstein presentaron un algoritmo para extraer automáticamente los bloques primitivos de un poliedro. Se supone que el poliedro es rectangular, es decir, los tres planos que se unen para formar un vértice deben ser perpendiculares entre sí. Los autores proponen examinar el dibujo lineal etiquetado (con cualquier esquema de etiquetado válido) y extraer información de los bloques a partir de ciertos tipos de uniones. Wang amplió el algoritmo para abarcar objetos semi-rectilineos y algunos objetos curvilineos. Un objeto semi-rectilineo tiene una dirección axial y cualquier superficie cuya normal no sea perpendicular al eje, es ella misma perpendicular al eje. Wang también introdujo los cilindros como primitivas. La clave del algoritmo es el etiquetado de las uniones. Se analizan las uniones curvas identificando y eliminando previamente los cilindros facetados. Una vez se han examinado todas las uniones curvas, se obtiene un poliedro semi-rectilineo. El algoritmo continúa identificando el prisma convexo a partir del poliedro, reduciendo la complejidad del dibujo lineal al mismo tiempo.

En su disertación, Wang dio una solución completa para encontrar la representación CSG de un dibujo lineal de un poliedro general con el tetraedro como nueva forma primitiva. También amplió los algoritmos semi-rectilíneos previos para poder abarcar un mundo del objeto más amplio que incluiría a los objetos curvos. Para cada algoritmo, se presentaban pruebas que muestran que si el dibujo representa un objeto válido el algoritmo termina y genera una representación CSG válida. Se daba una condición general y suficiente, basada en estos algoritmos, para que un dibujo lineal pueda representar un sólido válido. Sus algoritmos también podían manejar algunos dibujos incorrectos, que los esquemas de etiquetado existentes no podían distinguir. El primero, Roberts quien restringió los objetos a un conjunto finito fijo de modelos predefinidos. Su aproximación intenta identificar una transformación que cuando es aplicado a uno de los modelos, divide el objeto en los cuadrados mínimos. Claramente el modelo basado en esta proximidad es muy restringido.

2.2 Clasificación de los métodos.

Los diferentes métodos existentes se dividen en dos categorías: métodos de vistas múltiples y métodos de vista única. En la primera categoría, los métodos se subclasifican según la representación interna usada. En la segunda categoría, el criterio de clasificación es el propio método. La siguiente tabla muestra la organización de estos métodos.

Métodos de vistas múltiples	Métodos de vista única
Métodos Brep Idesawa 1973 Wesley-Markowsky 1980, 1981 Sakurai 1983 y Gu y otros 1985 Preiss 1984 **Métodos CSG** Aldefeld 1983, 1984 Ho 1986	**Métodos de etiquetado** Huffman 1971 y Clowes 1971 Lees y otros 1985 Malik 1987 **Métodos del espacio gradiente** Mackworth 1973 Wei 1987 **Métodos de programación lineal** Sugihara 1984, 1986 **Método perceptual** Lamb y otros 1990 **Método de identificación de primitivas** Wang y otros 1989, 1991, 1992

Tabla 1: *Organización del análisis de los métodos.*

2.2.1 Métodos de vistas múltiples.

Estos se clasifican según sus representaciones.

2.2.1.1 Métodos Brep.

Los pasos principales en todos estos métodos son:

1. La transformación de vértices 2D a vértices 3D
2. La generación de segmentos lineales 3D a partir de vértices 3D
3. La construcción de caras a partir de segmentos lineales 3D
4. La formación de objetos 3D a partir de las caras

Este es un método que va de abajo hacia arriba. Las principales diferencias entre los distintos métodos BRep residen en las técnicas que se usan para resolver estos cuatro pasos. Debería señalarse que puesto que estas vistas múltiples (no necesariamente tres) son las vistas conocidas en ingeniería, la correspondencia entre vistas está predefinida. Encontrar la correspondencia entre dos imágenes arbitrarias es en si mismo un problema muy difícil en el campo de visión artificial.

2.2.1.2 Métodos CSG.

Todos los métodos dentro de esta categoría asumen que cada objeto sólido 3D se puede construir a partir de ciertas formas primitivas de manera jerárquica. Más concretamente, un árbol CSG describe la síntesis recursiva de objetos compuestos de sub-objetos en términos de su posición relativa (expresada como transformaciones de coordenadas), y el conjunto de operaciones regularizadas de unión, intersección, y diferencia. Un sub-objeto es un caso de un miembro de un pequeño subconjunto

parametrizado de primitivas predefinidas; como una esfera con los parámetros de radio y centro, o un bloque con los parámetros de altura, longitud, anchura y las correspondientes matrices de transformación.

2.2.2 Métodos de vista única.

Obviamente es mucho más fácil reconstruir objetos 3D a partir de proyecciones de vistas múltiples que a partir de proyecciones de vista única. La reconstrucción de una única vista presenta más ambigüedades y da más soluciones. Describiremos algunos de los métodos típicos que se usan en el tema de la reconstrucción a partir de una vista única. Estos métodos se clasifican según sus modelos.

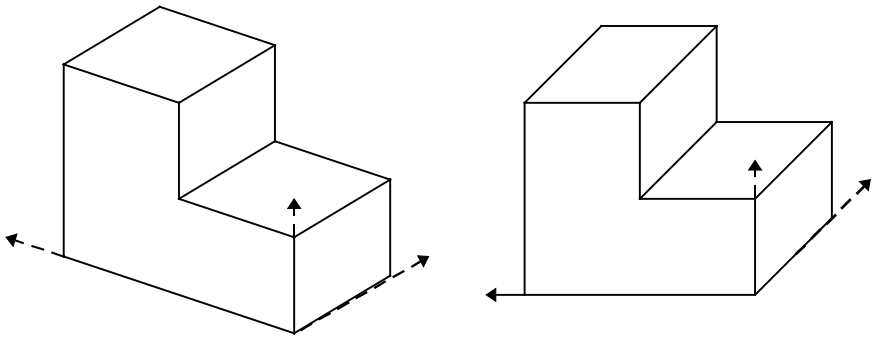
Dos de los cuatro métodos, el método del etiquetado y el del espacio *gradiente*, son métodos de interpretación más que de reconstrucción. Sirven para la interpretación de dibujos lineales 2D y generan información 3D que se puede usar posteriormente en el trabajo de reconstrucción. Pero por si mismos no hacen el trabajo de reconstrucción: no se obtiene ninguna representación de objetos 3D con estos métodos.

Entre estos métodos vamos a destacar el **Método perceptual** por ser el que más se asemeja al objetivo del proyecto.

2.2.2.1 Método Perceptual.

Lamb y Bandopadhy presentaron un sistema para extraer la estructura 3D, a partir de un dibujo lineal impreciso. La idea es dar a los diseñadores que utilizan CAD la capacidad de visualizar interactivamente la interpretación más probable de los dibujos disponibles en cada momento, los cuales pueden ser imprecisos, para permitir los cambios pertinentes cuando la interpretación de los mismos sea incorrecta.

El algoritmo toma como dato un dibujo axonométrico que puede ser isométrico (Figura 2 (a)), u oblicuo (Figura 2 (b)). En estos dibujos, las líneas paralelas aparecen paralelas, y las aristas paralelas a los ejes principales se dibujan con longitudes proporcionales a las dimensiones reales. El algoritmo pre-procesa el dibujo generando un gráfico de adyacencia (un mapa de vértices y aristas) y después etiqueta el dibujo usando el método de Waltz. Se le asignan al dibujo los ejes principales, los cuales, a su vez, forman los planos principales. También se selecciona el origen. Aquí una región se define como cualquier cara extraída mediante la información de etiquetado.



(a) Axonometría isométrica (b) Axonometría oblicua

Figura 2: Axonometrías isométrica y oblicua

El objetivo del algoritmo es asignar coordenadas 3D a todos los vértices del grafo. Esto se hace mediante los siguientes pasos:

1. Seleccionar la “mejor” región para la asignación de las coordenadas. La “mejor” región se determina mediante la evaluación de la información existente.

2. Seleccionar un punto de referencia para la región. Toda las coordenadas de esta región se asignarán en relación a este punto de referencia. Si esta es la primera región seleccionada, tiene sentido seleccionar la del origen. Si no, se espera que la “mejor” región contenga algún(os) vértice(s) ya asignados.

3. Asignar las coordenadas a todos los vértices de la región. Si la región es paralela a un plano principal, entonces es fácil asignar las coordenadas ya que la longitud de una arista refleja la distancia real. Si la región es paralela a un plano identificado, la información sobre el plano conocido y la longitud se pueden usar para determinar las coordenadas. Si la región no es paralela a ningún plano conocido, pero se pude aplicar una regla de simetría, entonces se usa la información dada por la simetría, así como otras reglas heurísticas (un paralelogramo visto como un rectángulo inclinado, por ejemplo) para determinar las coordenadas. Si no se cumple nada de lo anterior, el algoritmo necesita que el usuario especifique las coordenadas.

4. Repetir los 3 pasos anteriores hasta que se les hayan asignados las coordenadas a todos los vértices visibles.

Dependiendo de la precisión del dibujo y el orden en el que se procesen las regiones, es posible que a dos vértices se les asignen coordenadas que sean incompatibles con la ecuación del plano. Cuando esto ocurre, el algoritmo retrocede y modifica uno de los valores de las coordenadas.

2.2.2.2 Método Perceptual: Conclusiones.

Este método difiere de los otros en que no usa métodos numéricos, que son susceptibles de imprecisiones en los datos, aunque algunos de dichos métodos proclaman que tienen extensiones para tratar estas imprecisiones.

Este modelo ofrece un buen interfaz del usuario para que los diseñadores que utilizan CAD puedan ver objetos 3D. Pero tiene limitaciones en el uso de las reglas de percepción heurísticas. Estas son ciertas en muchos casos, pero no siempre. Por ejemplo, este método corregirá líneas con una ligera inclinación y las hará horizontales o verticales, incluso aunque esta inclinación sea intencionada. Otro problema surge de la determinación precisa de si dos planos son paralelos, o si dos objetos son simétricos. A menos que exista algún supuesto sobre el mundo del objeto (como la rectilinearidad), estas propiedades no se pueden determinar antes de que se asignen las coordenadas. Así pues, este modelo no ofrece una reconstrucción exacta del objeto 3D.

2.3 Breve Resumen.

La siguiente tabla da un breve resumen de los algoritmos existentes recalcando el tipo de objetos que componen la escena, el número de vistas necesarias, la representación usada, el número de soluciones y si se requiere la interacción con el usuario.

Métodos	**Tipos de superficies permitidas**	**Número de vistas necesarias**	**Representación usada**	**Número de soluciones**	**Interacción con usuario requerida**
Huffman 71 Clowes 71	Plana	Una	Etiquetado	Múltiple/única	Ninguna
Idesawa 73	Plana	Múltiple	de Contorno	Única	Ninguna
Wesley 80,81	Plana	Múltiple	de Contorno	Múltiple	Ninguna
Sakurai 83	Plana/ cilíndrica	Múltiple	de Contorno	Múltiple	Ninguna
Gu 85	cónica/ esférica/ tórica				
Preiss 84	Plana/ cilíndrica	Múltiple	de Contorno	Única	Ninguna
Aldefeld 83	Plana/ cilíndrica	Múltiple	de Contorno	Única	Ninguna
Aldefeld 84	Plana/ cilíndrica	Múltiple	CSG*	Única	Mucha
Ho 86	Plana/ esférica cilíndrica	Múltiple	CSG	Única	Mucha
Malik 85	Plana/ cilíndrica cónica/ esférica tórica	Una	Etiquetado	Múltiple/única	Ninguna
Mackworth 73	Plana	Una	de Contorno	Única	Ninguna
Sugihara 86	Plana	Múltiple	de Contorno	General	Ninguna
Sugihara 86	Plana	Una	de Contorno	General	Alguna
Wei 87	Plana	Una	de Contorno	General	Ninguna
Lamb 90	Plana	Una	de Contorno	Única	Alguna
Wang 92	Plana/ cilíndrica	Una	CSG	Única/ múltiple	Ninguna

3. OBJETIVOS.

Dentro del diseño gráfico por ordenador se pretende construir una herramienta que ayude al diseñador a automatizar el proceso de obtención de objetos tridimensionales a partir de su axonometría oblicua.

Para ello se pretenden cumplir los siguientes objetivos:

- Analizar el proceso deductivo y cognitivo humano en la obtención de la topología del objeto y representación visual de las caras. Consolidar un mecanismo que automatice este proceso, genere los vértices tridimensionales, la topología del modelo alámbrico y las caras del modelo completo.
- Obtener un método de eliminación de información redundante en la edición del dibujo[3] y en el proceso de extracción de información de entidades no definidas explícitamente por el usuario.
- Dotar de un interfaz amigable y uniforme con el estilo utilizado en AutoCAD. Este deberá en determinados momentos interactuar con el usuario para obtener determinadas confirmaciones y datos necesarios para la ejecución del programa.
- Devolver la información precisa, completa e inequívoca del modelo a AutoCAD, de forma que a través de los distintos sistemas de representación en formato gráfico: **DWG**, **DXF** o **IGES,** sea factible, mediante exportación, la manipulación y edición en otros sistemas y paquetes de diseño asistido por ordenador
- Profundizar en el estudio de la representación gráfica por ordenador, especialmente la representación de objetos tridimensionales. Utilizando para ello los conceptos y métodos matemáticos pertinentes.
- Ampliar el objetivo inicial de obtención del modelo alámbrico con la obtención de las caras, para así, una vez obtenido el modelo, poder aplicar las técnicas de renderización, eliminación de aristas ocultas, iluminaciones y sombreados.

[3] En concreto, este procedimiento, se puede generalizar como una herramienta útil para poder eliminar información redundante, facilitando la claridad, el almacenamiento de la información gráfica y el coste temporal en el ploteado de dibujos.

4. DESCRIPCIÓN DE LAS AXONOMETRÍAS.

4.1 Proyecciones.

4.1.1 Introducción.

La proyección es un operación fundamental en la representación gráfica. Lo que se plantea es el paso de un modelo de 3D a un plano, que en nuestro caso será la pantalla del monitor. Este paso se realizará mediante el mecanismo de las proyecciones.

En general, en matemáticas, una proyección se define como un paso de un espacio n-dimensional a otro espacio (n-1)dimensional:

$$\mathrm{IR}^{n} \longrightarrow IR^{n-1}$$

Solamente nos interesan las proyecciones del espacio 3D al 2D, de forma que al plano 2D dónde vamos a representar los objetos 3D se le conoce como **PLANO DE PROYECCIÓN** (**PP**):

$$\mathrm{IR}^{3} \longrightarrow IR^{2}$$

En este caso $IR^2 \equiv$ PP.

Supongamos un punto P tridimensional y un plano PP además de otro punto del espacio al que llamaremos **CENTRO DE PROYECCIÓN** (**CP**) o **PUNTO DE VISTA** (**PV**).

Si trazamos una recta que parta del CP, pase por P y corte al PP, dicha recta se llama **VISUAL** y a la intersección de la visual con el plano de proyección se le conoce como ***proyección del punto P***.

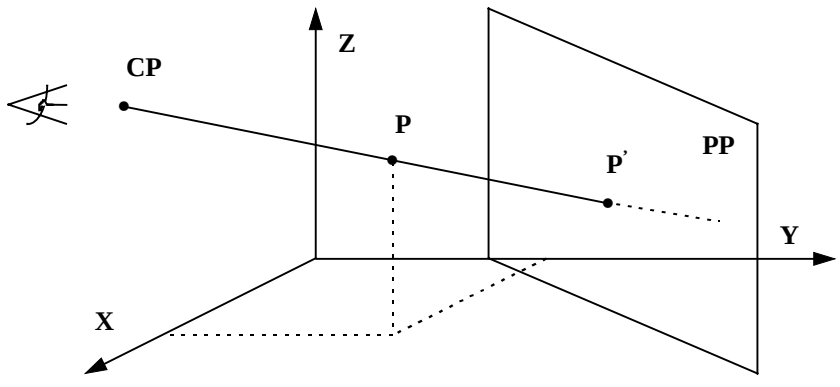


Figura 3: Proyección

Podemos establecer una clasificación, clásica en dibujo técnico, en dos grandes grupos en función de la posición del CP: **proyección perspectiva** y **proyección paralela**.

4.1.2 Proyección perspectiva.

El centro de proyección ocupa un punto real del espacio llamado **propio**. Las visuales que se tracen a partir del centro de proyecciones no serán paralelas. A esta proyección también se le conoce como ***proyección central***.

Esta clase de proyección guarda gran silimitud con la visión humana y la fotografía. En la primera el centro de proyección es el centro óptico del ojo y todas las visuales procedentes de un objeto pasan por dicho punto y desde él se proyectan sobre la superficie (esférica) de la retina. En la segunda el centro de proyección es el foco del objetivo de la cámara y los rayos visuales del objeto, tras converger en él se proyectan sobre la superficie (plana) de la película fotográfica.

4.1.3 Proyección paralela.

El centro de proyección no es un punto real (**impropio**) que se considera situado en el infinito. Todas las visuales son paralelas. Como en este tipo de proyecciones no tiene mucho sentido hablar de CP lo que se da es una **dirección de proyección** (**DP**), que es un vector que nos indica cual es la dirección de las visuales. A esta proyección también se le llama *proyección cilíndrica.*

Las proyecciones paralelas se clasifican a su vez, según la dirección de proyección, en dos tipos:

- **PROYECCIÓN OBLICUA**: La dirección de proyección forma con el plano de proyección un ángulo menor de 90° y mayor de 0°.
- **PROYECCIÓN ORTOGONAL**: También llamada proyección ortográfica. En estas proyecciones la dirección de proyección es perpendicular al plano de proyección.

4.2 Sistemas de representación gráfica.

4.2.1 Introducción.

La misión fundamental de los sistemas de representación gráfica consiste en permitir representar sobre una superficie plana –papel–, formas tridimensionales previamente creadas por el técnico, y viceversa, permitir la reconstrucción tridimensional –***restitución***– de un objeto a partir de su representación gráfica (en un plano).

Como consecuencia de dicha misión, para que un sistema de representación sea válido, será condición esencial la **BIUNIVOCIDAD** de la RELACIÓN espacio-plano, que en él se establezca. Es decir, que a cada punto del espacio corresponda una única representación en el plano (del dibujo) y que a cada representación de un punto en el plano (dibujo) le corresponda un único punto del espacio.

En definitiva, la validez de un sistema de representación, conllevará la existencia de una relación BIUNÍVOCA entre el espacio a representar (3D) y el plano de representación (2D).

Los sistemas de representación más importantes son:

I. **SISTEMA CÓNICO**: Compuesto por el plano de proyección π llamado ***plano del cuadro*** y un centro de proyección (CP) propio.

II. **SISTEMA DE PLANOS ACOTADOS**: En este sistema la forma de conseguir la representación de un punto del espacio consiste en obtener de él una proyección cilíndrica ortogonal sobre un plano de proyección π. La proyección de un cuerpo en este sistema es la misma que la proyección horizontal del sistema diédrico, solamente varía en que cada punto tiene que llevar indicada numéricamente su cota (altura del punto sobre el plano de comparación).

III. **SISTEMA DIÉDRICO**: Llamado también ***Sistema de Monge***, dispone de un conjunto formado por dos planos de proyección ortogonales entre sí, que se colocan generalmente: el primero horizontalmente (*plano horizontal de proyección H*), y el otro verticalmente (*plano vertical de proyección V*). La representación de un punto del espacio se consigue proyectando ortogonalmente el punto sobre el plano H, dando lugar a su *proyección horizontal*, y proyectando el punto también ortogonalmente sobre el plano V, obteniendo así la *proyección vertical*.

IV. **SISTEMA AXONOMÉTRICO**: Las figuras a representar se proyectan en proyección paralela sobre un plano del cuadro referidas a un sistema de ejes, sobre cuyos planos coordenados tienen las mismas sus correspondientes proyecciones ortogonales. La proyección en el cuadro de estas proyecciones sobre los planos coordenados acaba de determinar los elementos representados.

4.2.2 Sistema axonométrico.

Si representamos un objeto en sistema diédrico y procuramos que sus planos principales sean paralelos a los de proyección, obtenemos imágenes fáciles de dibujar y medibles sin dificultad. No obstante, éstas no siempre son suficientemente intuitivas.

Imágenes más claras e intuitivas se obtienen representando el objeto en el sistema axonométrico.

De acuerdo con lo expuesto anteriormente, la representación en axonométrico la obtendremos refiriendo el objeto a un sistema de tres ejes coordenados en el espacio, procurando que las direcciones principales del mismo coincidan con las de los ejes, y proyectando cilíndricamente a continuación el objeto, sus proyecciones ortogonales sobre los planos coordenados (proyecciones previas) y los ejes sobre el plano del

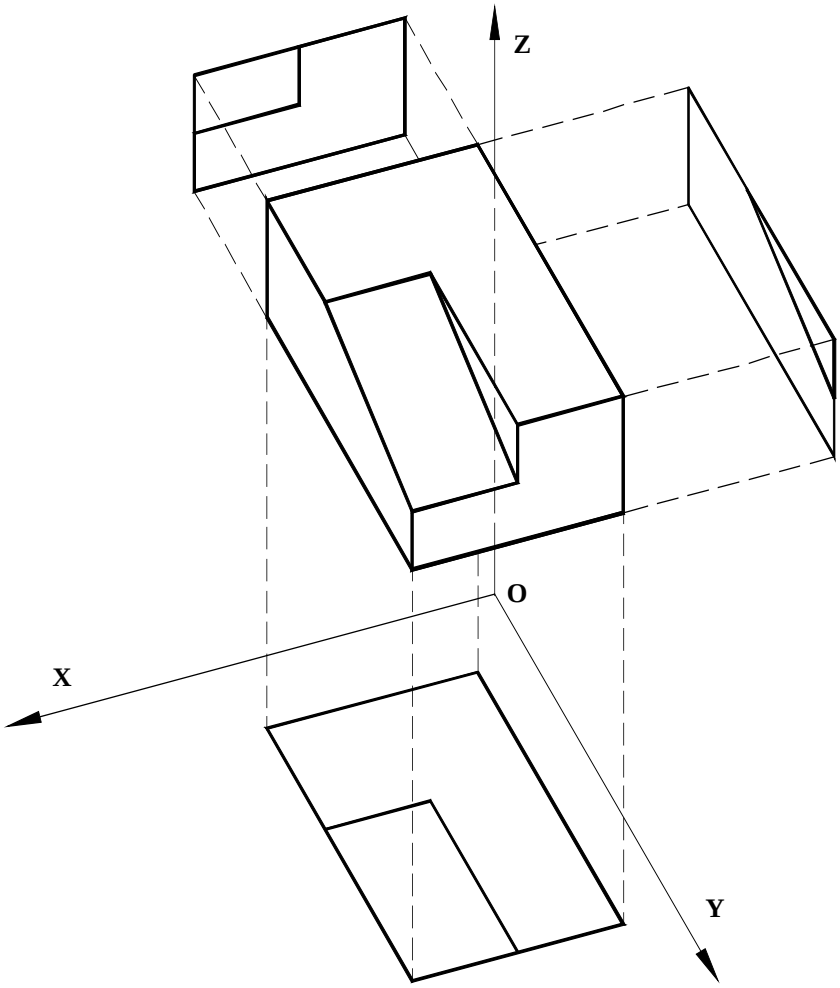


Figura 4: Sistema axonométrico

cuadro.

El triedro de referencia OXYZ lo supondremos trirrectángulo y orientado positivamente. Es decir, de manera que situados en el semieje positivo OZ, el eje OX gire en sentido antihorario sobre el eje OY (Norma DIN 5).

En dicho sistema un punto lo podemos situar por medio de sus coordenadas fácilmente, llevándolas paralelamente a las proyecciones de los ejes aprovechando la condición de invariante del paralelismo en proyeción cilíndrica

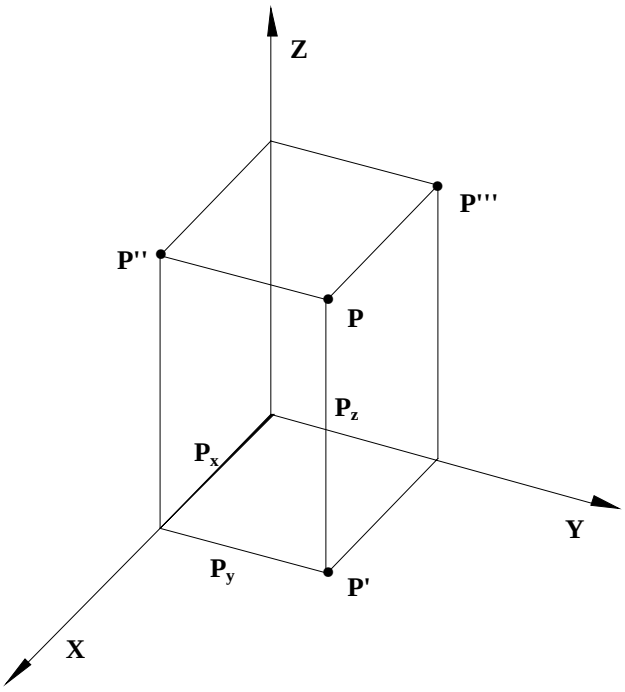


Figura 5: Representación de un punto en el sistema axonométrico

A la hora de llevar los segmentos correspondientes a cada coordenada se ha de tener presente, no obstante, que éstos se proyectan con valores proporcionales a las relaciones que existen entre segmentos cualesquiera tomados sobre los ejes y sus proyecciones. Dichas relaciones de proporcionalidad reciben el nombre de ***coeficientes de relación*** o ***deformación*** e_x, e_y, e_z.

De manera que $(P_x) = e_x \cdot P_x$, $(P_y) = e_y \cdot P_y$, $(P_z) = e_z \cdot P_z$.

Los distintos valores que pueden tomar los valores e_x, e_y, e_z, así como las diferentes posiciones relativas que pueden adoptar las proyecciones de los ejes son función, por un lado, de la orientación del triedro de referencia respecto del plano del cuadro y, por otro, de la relación que existe entre la dirección de proyección y el plano del cuadro.

En particular, cuando la dirección de proyección es ortogonal al plano del cuadro diremos que la Axonometría es **ORTOGONAL**, en el resto de los casos diremos que la Axonometría es **OBLICUA**.

Si la axonometría es ortogonal, existe una relación que liga a los coeficientes e_x, e_y, e_z con las proyecciones de los ejes. De manera que, dadas las proyecciones de los ejes del sistema, los coeficientes e_i son únicos y viceversa. Dados los coeficientes e_i las proyecciones de los ejes son únicas. Ambas posibilidades dan lugar a los denominados **Problemas Directo** e **Inverso de la Axonometría Ortogonal**.

5. HERRAMIENTAS Y ENTORNO DE DESARROLLO.

5.1 El lenguaje de programación c.

El C es un lenguaje de programación ampliamente difundido. Está encuadrado dentro de los lenguajes de nivel intermedio a camino entre los lenguajes de alto nivel y los de bajo nivel (ensambladores) incluyendo las ventajas de ambos. Por una parte contiene elementos propios de lenguajes de alto nivel, entre ellos, un conjunto reducido de sentencias de control y manipulación de datos, que pueden utilizarse para definir construcciones de alto nivel. Pero por otra parte, a diferencia de un lenguaje de alto nivel, donde se ha diseñado y se ha incorporado previamente todo aquello que posiblemente va a querer el programador, el C permite definir rutinas para llevar a cabo comandos de alto nivel. Estas rutinas se llaman ***funciones*** y son muy importantes en el lenguaje C. Pueden diseñarse una librería de funciones C personalizadas, a medida del usuario, para adaptarse a las tareas específicas de su programa.

El C manipula bits, bytes y puede operar directamente sobre los caracteres. En C estos procedimientos se llevan a cabo por funciones que no son propiamente parte del lenguaje, sino que se proporcionan como parte de una librería estándar. Estas funciones son rutinas especialmente escritas en el propio C y que llevan a cabo esas operaciones. El código en C es muy **portable**, lo que facilita la tarea de poder volver a ser utilizado en un nuevo ordenador con un procesador diferente. La mayoría de programas de aplicación tendrán que ser escritos una sola vez y compilados en el compilador C escrito para el nuevo procesador, lo que se traduce en un considerable ahorro de tiempo y dinero.

Los programas generados por compiladores C suelen ejecutarse tan rapidamente como los escritos en lenguaje ensamblador, consiguiendo una ventaja adicional; velocidad del ensamblador con la comodidad y legibilidad del C.

Otra ventaja adicional, es que al utilizar el paquete gráfico Autocad 12 podemos utilizar el lenguaje C frente a Autolisp, que es la otra opción de programación en Autocad. El C funciona como un lenguaje compilado y no interpretado, mientras que Autolisp es un intérprete, es decir utiliza un mecanismo de lectura del código fuente del programa, línea a línea, llevando a cabo cada vez las instrucciones específicas contenidas en la línea. En cambio el compilador lee el programa entero y lo convierte directamente en código objeto que es la traducción del código fuente del programa. El código objeto también se conoce como código binario o código máquina y es directamente ejecutable por el ordenador. Cualquier mención a una variable, instrucción o línea del código fuente carece de sentido como tal en la ejecución del programa. Consecuencia directa de esto es que los programas compilados se ejecutan mucho más rápidamente que los interpretados, aunque la compilación, en si misma, consume más tiempo, pero es rápidamente amortizable a medida que se utiliza el programa. Además los compiladores protegen el código fuente de posibles plagios o copias ilegales, así como de indebidas alteraciones, siendo, la alternativa utilizada casi de forma exclusiva por las empresas de software para realizar sus programas.

5.2 El entorno windows 95.

Windows 95 es un sistema operativo y, como todo sistema operativo, una de sus principales funciones es facilitar la comunicación entre el usuario y el ordenador, simplificando la utilización de los programas informáticos. En toda comunicación entre dos partes, tiene que haber métodos para que cada una de esas partes pueda emitir la información.

Microsoft Windows es un entorno gráfico de ventanas para ordenadores personales. La **multitarea** es una de sus características destacables que le capacita para ejecutar más de una aplicación al mismo tiempo. Puesto que múltiples aplicaciones pueden estar abiertas a la vez, no se necesita salir de una para pasar a la otra, pudiendo conmutar entre ellas fácilmente. Esto se traduce en que, por ejemplo, se puede dejar al sistema ordenando una base de datos en una ventana mientras se escribe una carta en otra. La multitarea sólo es posible en máquinas equipadas con procesadores 80386 o superiores aunque son recomendables los 80486 o bien los procesadores de última generación Pentium®.

Las ventanas junto con la multitarea son probablemente las características más importantes de Windows. Una **ventana** es el mecanismo que implementa Windows para enviar al usuario información de cualquier tipo, ya sea texto o gráfico. A grandes rasgos una ventana simplemente es una caja o recuadro que tiene un determinado contenido. Pueden haber varias ventanas abiertas a la vez. Pero la posibilidad de transferencia de datos también potencia su valor: permite transferir un bloque de texto, una tabla de números o una imagen desde un documento en una ventana a otro (con **COPY** y **PASTE**). Esto significa que se puede fácilmente insertar una imagen gráfica en una carta o una hoja de cálculo en un informe.

La transferencia de información entre programas es posible, porque todas las aplicaciones escritas para el entorno Windows soportan las mismas capacidades de transferencia de datos. Se puede incluso CORTAR y PEGAR entre algunas aplicaciones no escritas específicamente para este entorno.

Otra característica muy avanzada es la que le permite manejar la memoria más allá de la barrera de los 640 Kbytes. Además, una presentación buena y un interfaz agradable, común a todas las aplicaciones Windows, le convierten en la plataforma adecuada para toda aplicación gráfica.

5.3 El paquete gráfico AutoCAD 12.

El paquete de diseño AutoCAD es un potente instrumento de ayuda al dibujo. En función de las órdenes recibidas ejecuta rápidamente los dibujos deseados. Permite corregir los errores de dibujo en la edición e incluso efectuar revisiones importantes sin necesidad de rehacer todo el dibujo. El producto final es un dibujo nítido y preciso.

AutoCAD trabaja según las instrucciones del usuario, sin añadir por su cuenta detalles no especificados. Un dibujo creado por AutoCAD es virtualmente idéntico al realizado cuidadosamente a mano (virtualmente; porque AutoCAD es capaz de reproducir con excelente fidelidad el dibujo si se dispone de un equipo apropiado). El resultado es un dibujo que corresponde exactamnete a las instrucciones dadas; cada elemento queda ubicado en el lugar preciso designado por el usuario.

AutoCAD pone a disposición del usuario una serie de ***entidades*** que sirven para construir el dibujo. Una entidad es un elemento de dibujo tal como una línea, círculo, texto, etc... Para indicarle al programa cuales son las entidades que debe dibujar, el usuario introduce órdenes en forma de **comandos**, tecleándolos o designándolos desde un menú. Una vez escogida la orden, se introducen los parámetros solicitados por los mensajes en la pantalla, como puedan ser el punto correspondiente al lugar del dibujo en el que se desea ubicar la entidad, un tamaño, un ángulo de rotación, etc... Cuando se hallan reunidos los elementos de información necesarios, la entidad se dibuja en la pantalla. Después, se puede introducir otra orden para dibujar una entidad diferente o pasar a otra función de AutoCAD. El efecto de cada modificación especificada se visualiza inmediatamente en la pantalla.

Otras características de AutoCAD permiten editar el dibujo de muchas maneras: las entidades se pueden borrar, desplazar o copiar para crear estructuras con elementos repetidos. Se pueden visualizar distintas partes del dibujo en la pantalla o hacer que se lea en pantalla información acerca del dibujo. AutoCAD proporciona ayudas para emplazar las entidades con exactitud. Se puede obtener copia impresa del dibujo mediante salida a un trazador (plotter) o una impresora gráfica.

El formato sencillo de las órdenes de AutoCAD facilita el acceso a las funciones del programa. Las órdenes se pueden realizar tecleándolas o designándolas desde un menú con un dispositivo señalador, ya sea un ratón, tableta digitalizadora o similar.

Quizás una de las características más destacables de AutoCAD sea su enfoque de ***arquitectura abierta***. Lo que le faculta para que su sistema de dibujo sea utilizable en todo tipo de aplicaciones, permitiendo a cada usuario adaptar el programa a sus necesidades. Dentro de esta filosofía, el usuario puede:

- Definir sus propios menús de pantalla, menús desplegables, menús de símbolos, menús de tablero y menús de pulsadores a fin de automatizar las operaciones más frecuentes.
- Programar sus propios letreros de dialogo.
- Crear funciones de guión para automatizar largas secuencias de órdenes.
- Definir sus propios tipos de letra, de línea (de puntos y rayas), patrones de sombreado o de relleno *PostScript*.

- Crear sus propios símbolos y bibliotecas de componentes.
- Crear dibujos prototipo con valores implícitos propios.
- Personalizar el fichero de ayuda y crear otros adicionales.
- Usar ficheros **DXF** o **IGES™** para pasar datos del dibujo a otros programas (de análisis de elementos finitos, de fabricación asistida por ordenador o **Computer Aided Manufacturing** (**CAM**)**,** crear dibujos a partir de datos generados por otros programas.
- Crear ficheros de fotos (**SLIDES**) o *PostScript* a partir de los dibujos, a fin de incluirlos en documentos producidos por programas de autoedición.
- Ejecutar programas externos en medio de la edición de un dibujo.
- Usar AutoLisp o ADS para efectuar cálculos, automatizar tareas repetitivas, crear nuevas órdenes de AutoCAD o redefinir las existentes.
- Usar la característica de ***Datos de Entidad Extendida*** o ***Extended Entity Data*** (**EED**), para que el usuario pueda añadir sus propios datos a entidades de AutoCAD a través de los programas AutoLisp o ADS.
- Crear equivalencias o abreviaturas para órdenes.
- Escribir módulos de gestión para equipos particulares.

La versión 12 de AutoCAD para Windows está específicamente diseñada para sistemas 80386 y 80486™ que utilizan DOS 3.3 o posteriores y Windows 3.1. Esta versión ofrece varias características únicas del entorno Windows siendo la más importante la **Interfaz Gráfica de Usuario** (**GUI**) que facilita el uso de AutoCAD, pudiendo, gracias a la característica de arquitectura abierta de este producto, facilitar la creación de nuevas aplicaciones Windows que funcionan con AutoCAD.

AutoCAD para Windows funciona como una aplicación Windows en el modo mejorado de Windows 386. Tanto la ventana gráfica como la de texto poseen botones que al seleccionarse ocupan toda la pantalla con la ventana o la reducen a un icono. Cuando el programa está representado por un icono, continua activo sin interrumpir la operación en curso ni terminar la sesión de edición.

Las características enumeradas a continuación aprovechan el entorno Windows:

- Sesiones múltiples de AutoCAD.
- Ventanas gráficas y de texto móviles y redimensionables.
- Barra de herramientas personalizada y caja de herramientas flotante.
- Reconfiguración dinámica de la pantalla desde AutoCAD.
- Ayuda sensible al contexto.
- Digitalizador de AutoCAD usado como dispositivo señalador del sistema Windows.
- Órdenes en forma de iconos o formatos de texto y gráficos del *Portapapeles*.

- Ventana de vista de área.
- Enlace o inserción de objetos (**OLE**).
- Intercambio dinámico de datos o ***Dynamic Data Exchange*** (**DDE**).
- Soporte del sistema de desarrollo o ***Autodesk Development System*** (**ADS**).

5.3.1 ADS y la programación en C.

El Sistema de desarrollo de AutoCAD (AutoCAD Development System, ADS) es una interface de programación que permite escribir aplicaciones de AutoCAD en C.

De igual modo que las bibliotecas estándar de C, el entorno ADS está definido por bibliotecas y ficheros de encabezamiento.

Aunque las aplicaciones base de ADS están escritas en C, para AutoCAD son idénticas a las funciones escritas en AutoLISP. Una aplicación ADS no se escribe para ser un programa independiente, sino que se escribe para ser un conjunto de funciones externas para ser cargadas, y llamadas, por el intérprete de AutoLISP. Como se puede observar las aplicaciones ADS son básicamente servidoras de AutoLISP y permanecen inactivas hasta que AutoLISP realiza una petición para que una función externa sea ejecutada. Por otra parte mientras la aplicación ADS está respondiendo a AutoLISP, ambas permanecen esperando las respuestas de las funciones de biblioteca de ADS, y no pueden responder a ninguna entrada de usuario.

5.3.1.1 Estructura de una aplicación ADS.

Las aplicaciones ADS deben implementar la interface con AutoLISP. Esta interface requiere que la aplicación llame a ciertas funciones de la biblioteca de ADS, con unos valores definidos y en una secuencia predeterminada.

AutoLISP accede a una aplicación ADS de acuerdo a la siguiente secuencia de eventos:

1. AutoLISP carga la aplicación ADS al inicializarse AutoCAD o cuando (xload) o ads_xload() se invocan. También es posible cargar una aplicación ADS utilizando la aplicación de AutoLISP appload, que se carga automáticamente al llamarla. Ésta aplicación establece una ventana, desde la cual podemos cargar tanto aplicaciones ADS como AutoLISP, y descargar aplicaciones ADS cuando ya no las vayamos a utilizar.
2. La aplicación ADS inicializa la comunicación con AutoLISP llamando a la función ads_init().
3. La aplicación ADS indica que está lista para procesar una petición de AutoLISP llamando a la función ads_link() con el código RSRSLT (application result code).
4. AutoLISP retorna de la llamada ads_link() el código RQXLOAD (request code).
5. La aplicación define sus funciones externas llamando a la función ads_defun() para cada función que se vaya a definir como externa.

6. La aplicación llama a la función ads_link() de nuevo con el código RSRSLT (a menos que se hubiese detectado un error, en cuyo caso lo haría con el código RSERR).
7. AutoLISP devuelve de la llamada ads_link() el código RQSUBR (request code) cuando el usuario o una función AutoLISP evalúa una de las funciones externas de la aplicación.
8. Después de evaluar la función externa, la aplicación llama a la función ads_link() con el código RSRSLT (result code).

Como se puede observar ads_link() es llamada repetidamente, por lo tanto la manera más adecuada de mantener la secuencia es llamándola desde el principio, dentro de en un bucle infinito. Además, dentro de este, incluiremos todos los casos para el manejo de las distintas peticiones de AutoLISP.

Así pues, el prototipo de una aplicación ADS será el siguiente:

```
#include <stdio.h>
#include "adslib.h"
#define ELEMENTS(array) (sizeof(array)/sizeof((array)[0]))
struct func_entry
{       char *func_name;
        int (*func) _((struct resbuf *));
};
void main _((int,char**));
int load_func _((void));
int do_func _((void));
int name_func _((struct resbuf *rb));
static struct func_entry func_table[] = {{"c:name_lsp", name_func},…};

void main(argc, argv)
int argc;
char *argv[];
{       int stat;
        short scode = RSRSLT;
        ads_init(argc, argv);
        for ( ;; )
        {       if ((stat = ads_link(scode)) < 0)
                {       printf("Estado incorrecto de ads_link() = %d\n", stat);
                        fflush(stdout);
                        exit(1);
                }
                scode = RSRSLT;
                switch (stat)
                {       case RQXLOAD:           scode= load_func() ? RSRSLT: RSERR;
                                                break;
                        case RQSUBR:    scode= do_func()==RTNORM ? RSRSLT:RSERR;
                                        break;
                        case RQXUNLD:           break;
                        case RQSAVE:    break;
                        case RQQUIT:    break;
                        case RQEND:     break;
                        default:                break;
                }
        }
}
static int load_func()
```

```
{       int i;
        for (i = 0; i < ELEMENTS(func_table); i++)
        {       if (!ads_defun(func_table[i].func_name, i))
                 return RTERROR;
        }
        return RTNORM;
}
static int do_func()
{       struct resbuf *rb;
        int val;
        if ((val = ads_getfuncode()) < 0 || val >= ELEMENTS(func_table))
        {       ads_fail("Recibido código de función inexistente.");
                 return RTERROR;
        }
        rb = ads_getargs();
        val = (*func_table[val].func)(rb);
        ads_relrb(rb);
        return val;
}
int name_func()
{       /* implementación de la función externa */
        return RSRSLT;
}
```

5.3.1.2 ADS para Windows.

Las aplicaciones ADS en el entorno Windows pueden escribirse independientemente de AutoCAD y comportarse de un modo más conocido para los usuarios de Windows. La aplicación funciona en su propia ventana con su propia interface de usuario. Puede aceptar la entrada de datos desde el teclado o el ratón, usar letreros de diálogo de Windows y emplear todas las herramientas gráficas de Windows.

La biblioteca ADS para Windows incluye un módulo que soporta la interface de la biblioteca al sistema operativo Windows. El nombre de este módulo varía dependiendo del compilador que se utilice. En general se llama *winads.obj* del cual AutoCAD para Windows también provee la fuente en *winads.c*. Este módulo para el compilador de C con el que se está realizando la implementación, Borland C++, es *winadsbc.obj*.

5.3.1.3 Estructura de una aplicación ADS para Windows.

La estructura de una aplicación ADS para Windows es casi idéntica a la de una aplicación ADS convencional (descrita anteriormente), salvo que la versión para Windows contiene algunas secuencias y llamadas de función adicionales.

En primer lugar debemos tener en cuenta que las aplicaciones para Windows deben incluir dos nuevos ficheros de encabezamiento, que contienen unas secuencias especificas de Windows.

```
#include  <windows.h>
#include  "windde.h"
```

Por otra parte podemos incluir código especifico de Windows para:

- Registro y envío de mensajes,
- Presentación de la aplicación como icono,

- Asegurar que la aplicación funcione bien,
- Letreros de diálogo,
- ...

6. DESCRIPCIÓN DEL ALGORITMO.

6.1 Preliminares.

6.1.1 Introducción.

Como indica su propia definición, un ***algoritmo*** es un conjunto de reglas básicas que permiten resolver de forma metódica un problema. También se puede definir como una lista de instrucciones que especifican una secuencia de operaciones ***deterministas*** que llevadas a cabo por un agente ejecutor (***modelo computacional***) da una solución a cualquier instancia de un problema determinado en un *tiempo finito*.

Seguidamente vamos a ver las fases del algoritmo creado para resolver el problema de la restitución 3D de objetos a partir de sus representaciones axonométricas oblicuas. Posteriormente veremos un esquema general de dicho algoritmo y por último la estructura de datos utilizada. En posteriores capítulos se desarrollarán las diferentes fases del algoritmo.

6.1.2 Fases del algoritmo.

El proyecto a realizar consta de tres grandes fases relacionadas entre sí, de tal forma que los datos de salida de una de ellas corresponden a los datos de entrada de la siguiente. Estas fases son:

1. Identificación de las caras en 2D,
2. Obtención del modelo alámbrico en 3D y
3. Construcción de las caras en 3D.

La primera fase, **Identificación de las caras en 2D**, tiene como datos de entrada los del dibujo 2D, correspondiente a una axonometría oblicua, en la base de datos del sistema CAD utilizado. De ella obtendremos los datos necesarios para poder definir los vértices y las aristas. A partir de estos se pueden deducir propiedades necesarias para las futuras fases como son paralelismo de las aristas, longitudes, valencia o grado de los vértices, etc.

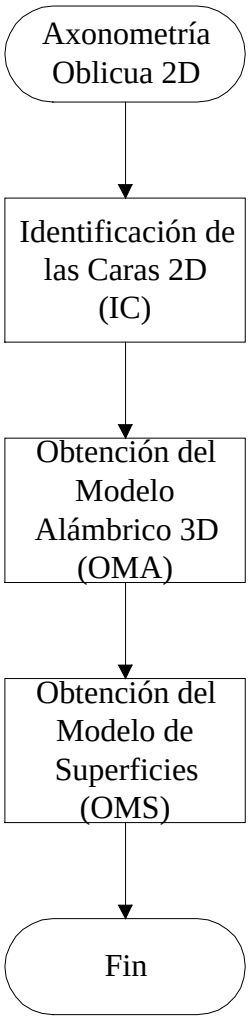
Una vez obtenidos estos datos se utilizará un "algoritmo de búsqueda de bucles de caras" para poder detectar las caras, que estarán formadas por bucles o ciclos de aristas y vértices donde el primer y último vértice coincidirán y no habrá ningún otro vértice que se repita dentro del ciclo.

En primer lugar buscaremos caras de tres lados, después de cuatro lados y si todavía quedan caras a detectar, que lo comprobaremos aplicando el teorema de Euler, entonces se buscarán el resto de caras. Para ello, en primer lugar comprobaremos el grado o valencia[4] de los vértices, y si estos son superiores a cuatro entonces dividiremos el vértice en tantos de valencia tres o cuatro como sean necesarios, creando así mismo, tantas aristas de longitud cero, **aristas auxiliares**, como sean necesarias uniendo estos vértices. (Figura 11. Pág. 54)

[4] El grado o valencia de un vértice indicará el número de aristas que confluyen en él.

En segundo lugar aplicaremos el algoritmo de búsqueda de ciclos limitando la profundidad del árbol de búsqueda al máximo de vértices existentes. Si éste es muy grande entonces se solicitará al usuario que ponga un número máximo de aristas por cara. De esta forma detectaremos un conjunto de ciclos susceptibles de convertirse en caras, **posibles caras**.

Por último, y de modo interactivo, se solicitará al usuario que de esas posibles caras indique cuales son "verdaderas" caras eliminándose las restantes. Para ello se modificará el atributo cromático del perímetro de la cara en proceso y el usuario decidirá si lo es o no.



Esquema 1: Fases del algoritmo.

Para simplificar los algoritmos, tanto el de búsqueda de bucles de caras como el de eliminación de caras no reales, solo se buscarán bucles con aquellas aristas que no estén saturadas[5]. Dado que estas aristas ya están contenidas en dos caras ya identificadas, y por tanto, no pueden estar en más caras.

Al final de esta fase habremos identificado los vértices, las aristas y las caras del objeto a partir de su representación en 2D. El paso siguiente será convertir las estructuras 2D de estos datos en estructuras 3D.

Para la realización de ésta fase el algoritmo utilizará el módulo **IC** (Identificación de las Caras 2D).

En la segunda fase, **Obtención de la figura en 3D**, se partirá de las estructuras de vértices, aristas y caras creadas en la fase anterior, y de las coordenadas de los ejes dados por el usuario. Con estos datos y utilizando regularidades geométricas basada en los invariantes de paralelismo y de proporcionalidad de las aristas existentes entre la proyección (2D) y el modelo (3D), se obtienen las coordenadas 3D de los vértices y las aristas.

La obtención del modelo 3D dependerá, en gran medida, de los puntos elegidos por el usuario para el centro de coordenadas y los ejes así como de las escalas dadas para los mismos.

La obtención del modelo seguirá los siguientes pasos que se desarrollarán en puntos posteriores del proyecto:

1. Selección por el usuario de las aristas coincidentes con los ejes del sistema de referencia.

[5] Una arista estará saturada cuando las dos caras a las que pertenezca hayan sido identificadas.

2. Obtención de las aristas 3D paralelas a los ejes de las que conocemos las coordenadas 3D de uno de sus vértices. En realidad el apartado 1 se puede considerar como un subapartado de este, como veremos posteriormente.

3. Obtención del resto de aristas 3D. Para este caso utilizaremos dos algoritmos:

    3.1. Obtención de aquellas aristas 3D, uno de cuyos vértices 3D sea obtenible en una de las caras que la contiene.

    3.2. Obtención de aquellas aristas no paralelas a ningún eje, pero que al menos en una de las caras a la que pertenece se conocen dos aristas no contiguas en 3D.

Para aquellos casos no contenidos en los anteriores, el algoritmo no podrá calcular las coordenadas 3D del vértice y/o arista en cuestión, avisando de ello.

Para la realización de ésta fase el algoritmo utilizará el módulo **OMA** (Obtención del Modelo Alámbrico).

Al finalizar esta fase habremos obtenido el conjunto de **vértices** 3D relacionados entre sí por medio de las **aristas** 3D que forman un objeto 3D, pudiendo representar éste en modelo alámbrico. Sin embargo, las caras del objeto, todavía no están incluidas en la estructura de datos 3D. Sabemos qué bucle de aristas forma cada cara, pero al no tener definidas las caras 3D del objeto en la base de datos del sistema CAD, no podremos hacer renderizados, ni ocultación de líneas, etc, a partir de la representación 3D de dicho objetosean necesarias.

En la tercera fase que se ha denominado **Construcción de caras en 3D**, procederemos a la definición automática de las caras en 3D del objeto.

Para construir las caras en 3D utilizaremos como datos de entrada las caras detectadas en 2D y toda la información que tenemos de los vértices y de las aristas, tanto en 2D como en 3D. En primer lugar construiremos las caras de tres lados, pasando posteriormente a construir el resto de las caras. Esto lo haremos así porque las caras de tres lados pasarán directamente por medio de la orden 3DCARA a ser caras 3D, pudiendo existir, en cambio, problemas en el resto de las caras con concavidades, como veremos posteriormente.

Para construir las caras de más de tres lados los pasos que se seguirán van a ser los siguientes:

1. Estudio de la concavidad/convexidad de las caras[6].

    1.1. Si la cara es cóncava entonces dicha cara se subdividirá en dos caras a partir de la arista que genera la concavidad. A continuación se pasa a estudiar la concavidad/convexidad de estas dos nuevas caras, repitiendo el proceso hasta encontrar únicamente caras convexas.

    1.2. Si la cara es convexa entonces la triangularizaremos y aplicaremos la orden 3DCARA a cada una de las caras triangulares obtenidas.

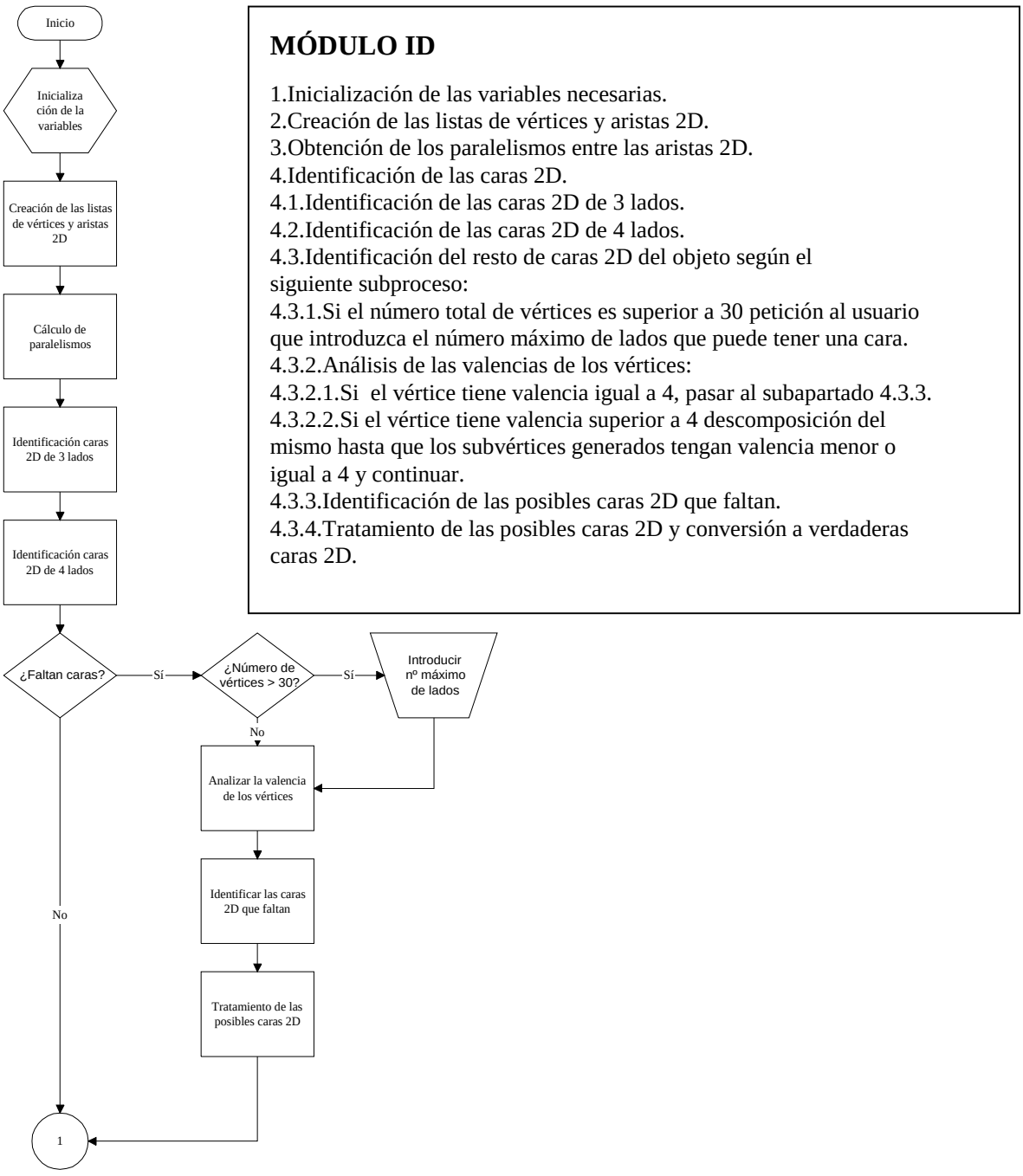
[6] Ver apartado concavidad/convexidad de una cara

De esta forma obtendremos un conjunto de **caras triangulares 3D** que definirán perfectamente la figura y con las que podremos hacer todo aquello que los programas de diseño nos permita (Renderizado, iluminaciones, ocultación de líneas, etc).

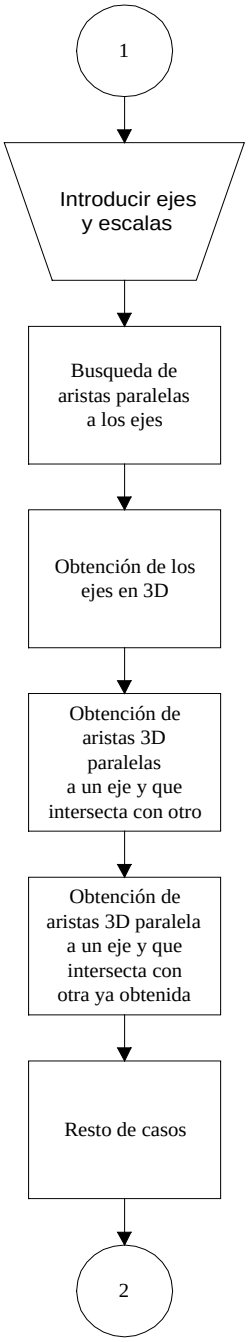
Para la realización de ésta fase el algoritmo utilizará el módulo **OMS** (Obtención del modelo de superficies).

6.1.3 Esquema del algoritmo.

El esquema general del algoritmo para la reconstrucción 3D de objetos poliédricos a partir de sus representaciones axonométricas oblicuas es el siguiente:



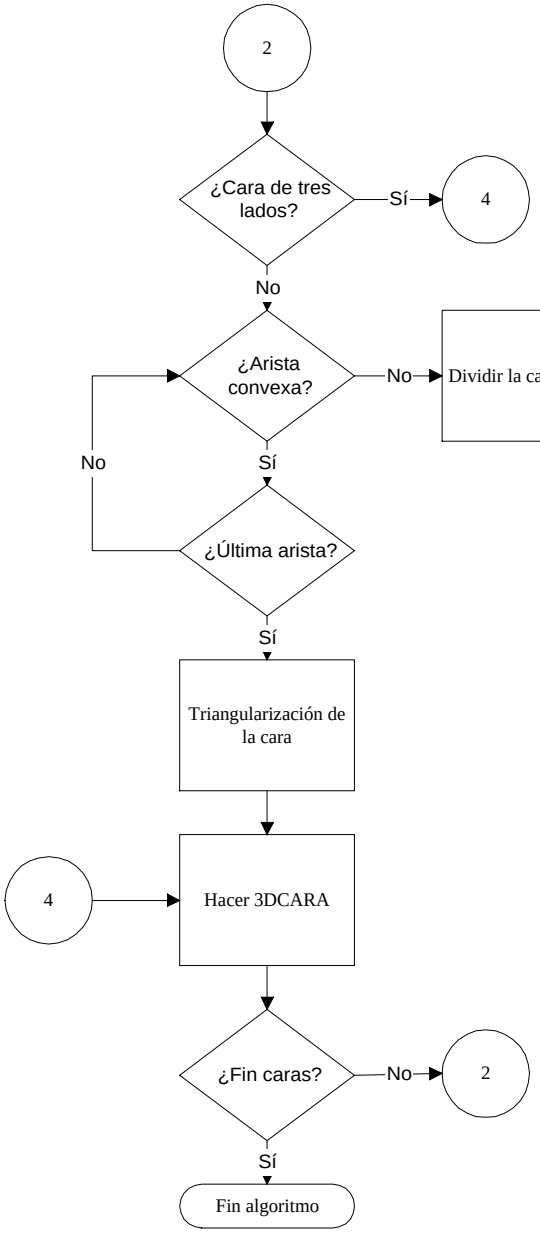
Esquema 2: Identificación de las caras en 2D



Esquema 3: Obtención del modelo alámbrico

MÓDULO OMA

1.Obtención de los vértices y aristas 3D.
1.1.Petición al usuario que introduzca los puntos que definen el origen de coordenadas y cada uno de los tres ejes (X, Y, Z), así como las escalas de estos.
1.2.Búsqueda de las aristas paralelas a los ejes.
1.3.Obtención de los ejes en 3D.
1.4.Obtención de las aristas 3D con uno de sus extremos coincidentes con un punto de uno de los ejes coordenados y paralelas a otro eje.
1.5.Obtención de aristas 3D paralelas a un eje y que intersectan con otra arista 3D ya obtenida.
1.6.Obtención del resto de aristas.
1.6.1.Recorrer la lista de vértices y si no está ya en 3D, analizarlo de la siguiente forma:
1.6.1.1.Obtención de aristas con alguno de sus dos vértices en 3D y el otro obtenible en alguna de las caras que la contienen.
1.6.1.2.Obtención de aquellas aristas que al menos una de las caras a la que pertenece tiene dos aristas, no alineadas entre sí, en 3D.
1.6.1.3.Aviso mediante letrero de diálogo, al usuario en caso de que no se pueda obtener algún vértice en 3D.



MÓDULO OMS

1.Construcción de las caras 3D. Recorrer la lista caras y:
1.1.Si la cara es de tres lados hacer 3DCARA,
1.2.Recorrer las aristas de la cara.
1.2.1.Si la arista es convexa en esa cara entonces pasar a la siguiente.
1.2.2.Si es cóncava entonces dividir la cara en dos nuevas caras por esta arista, que está generando una concavidad en la cara.
A continuación se pasa a estudiar la concavidad/convexidad de estas dos nuevas caras, repitiendo el proceso desde 1.1.
1.2.3.Repetir el subapartado 1.2. hasta completar la lista de aristas de la cara.
1.3.Triangularización de la cara (dividir la cara en un número mínimo de caras triangulares).
1.4.Para cada cara triangular obtenida le aplicaremos la orden 3DCARA.
1.5.Repetir el proceso hasta completar la lista de caras.
2.Fin del algoritmo.

Esquema 4: Obtención del modelo de superficies

6.1.4 Estructura de datos.

El acceso a los datos para realizar los cálculos pertinentes es de gran importancia para la ***eficiencia*** del algoritmo. Con una buena distribución de los datos podemos, además, controlar la seguridad de los mismos y que estos no interactúen con otros elementos del programa.

Definiremos tres estructuras principales diferentes relacionadas entre sí y que representaran a cada uno de los objetos vértice, arista y cara. Además de estas estructuras se van a definir otras dos auxiliares para los ejes y para las posibles caras respectivamente. Veamos estas estructuras:

6.1.4.1 Vértices.

La información almacenada para los vértices es la siguiente:

- Identificador del vértice.
- Valencia o grado. Número de aristas inciden en él.
- Listado de aristas que inciden en el vértice.
- Coordenadas del vértice en 2D.
- Coordenadas del vértice en 3D.
- Variable que indicará si el vértice ya está identificado en 3D.

Con estos datos el objeto vértice queda perfectamente definido y se facilita la búsqueda de las aristas que inciden en el mismo así como de los vértices adyacentes.

6.1.4.2 Aristas.

Esta estructura va a ser la principal, puesto que a partir de ella se podrá definir el objeto completamente. Es en esta estructura donde se consigue enlazar las caras con los vértices y con las propias aristas. Los datos que almacenaremos en ella son:

- Identificador de la arista.
- Longitud. Distancia entre sus dos vértices extremos.
- Enlace al vértice inicial y al vértice final de la arista. Enlace a cada uno de los vértices extremos de la arista.
- Enlaces a cada una de las dos caras a las que pertenece la arista.
- Lista de aristas que son paralelas a ésta.
- Y una variable que indicará si la arista ya está asignada a una cara, a las dos (y por lo tanto ya estará saturada) o a ninguna. Esta **variable de saturación** es fundamental a la hora de la búsqueda de las caras, puesto que gracias a ella el algoritmo de identificación de las caras se simplifica bastante.

6.1.4.3 Caras.

En la estructura caras almacenamos la siguiente información:

- Identificador de la cara.
- Número de aristas reales que la definen.
- Número de aristas tanto reales como ficticias que la definen.
- Listado de aristas contenidos en la cara.
- Y listado de vértices contenidos en la cara.

Como se puede observar, con estas tres estructuras definimos perfectamente un objeto. Aunque es sólo en la estructura de vértices donde se guardará información del objeto en 3D. Esta información consistirá en las coordenadas del vértice en 3D y si el vértice ya ha sido identificado en 3D o no. Esto es así gracias a que por un lado las estructuras están vinculadas entre sí, y por otro, a que la topología del objeto es idéntica en 2D y en 3D. Una arista tendrá los mismos vértices extremos en 2D y en 3D, una cara contendrá al mismo conjunto de vértices y aristas en 2D y en 3D, etc. Si bien, naturalmente las coordenadas de un vértice serán distintas en 2D y en 3D.

6.1.4.4 Estructuras auxiliares.

Además de estas estructuras, se han definido dos estructuras auxiliares. Una de *posibles caras*, donde se almacenarán los datos de un "ente" susceptible de ser cara. Estos datos son:

- Número de aristas contando las ficticias.
- Número de aristas reales.
- Cual es el primer vértice.
- Lado utilizado para encontrar la cara. Este concepto se verá en capítulos posteriores, cuando se vea el tratamiento de las posibles caras.
- Valencia del segundo vértice.
- Listado de aristas que forman la posible cara.
- Y listado ordenado de vértices que están contenidos en la posible cara.

Esta es una estructura temporal, los datos desaparecerán una vez definida la verdadera cara.

La otra estructura que tendremos es la que nos almacena la información referente a los *ejes 3D*. Un eje lo consideraremos como una arista especial, y solo almacenaremos la siguiente información:

- Eje (X,Y ó Z)
- Coordenadas 2D de los vértices que lo definen.
- Longitud. Distancia entre sus extremos.
- Lista de aristas paralelas al mismo.

- Y escala del eje.

Partiendo de la información de esta estructura y de las tres principales, Vértices, aristas y caras, identificaremos el objeto en 3D.

6.2 Grafos. Generalidades.[7]

6.2.1 Introducción.

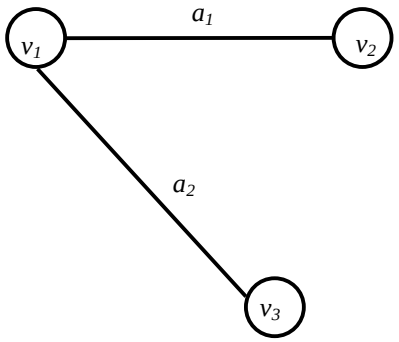
En los problemas originados en ciencias de la computación, matemáticas, ingenierías y otras muchas disciplinas, a menudo es necesario representar relaciones arbitrarias entre objetos. Los grafos se emplean en distintas disciplinas para modelar relaciones de objetos. En diseño, un objeto representado mediante un modelo alámbrico está compuesto por vértices conectados por una arista si están relacionados entre sí.

En este capítulo introduciremos la definición de grafo y otros conceptos básicos relacionados necesarios para el desarrollo de este proyecto. Después se introducirán las restricciones impuestas para que la identificación de las caras se haga de forma rápida y sin que sea una búsqueda exhaustiva. Por último se describirá el algoritmo utilizado para la identificación de las caras.

6.2.2 Definiciones.

Llamamos **GRAFO**[8], G=(V(G),A(G)), a toda estructura formada por un conjunto no vacío V(G), en adelante **vértices**, y un conjunto A(G), en adelante **aristas**, de pares no ordenados de puntos de V.

Sea a=(*u*,*v*) una arista, decimos que se trata de una arista de **extremos *u* y *v*** y que es **incidente** con *u* y *v* y recíprocamente. Dos aristas incidentes en un mismo vértice se llaman **adyacentes**.



Según las definiciones anteriores, por ejemplo a_1 y a_2 son adyacentes y a_1 es incidente con los extremos v_1 y v_2 y los vértices v_1 y v_2 son adyacentes.

Podemos afirmar que:

1. Pueden existir aristas (*u*,*v*) con *u*=*v*, es decir, aristas cuyos dos extremos son el mismo vértice. A estas aristas las llamamos **bucles**.

[7] Apuntes asignatura Complementos matemáticas. 4º Facultad de Informática, Universidad Politécnica de Valencia. Cristina Jordán Lluch.

[8] Cuando habremos de grafos nos referiremos a grafos no dirigidos. Se diferencia de un grafo dirigido en que cada arista en A es un par no ordenado de vértices.

2. No pueden existir varias aristas entre dos vértices *u* y *v*. Si en la definición hubiéramos considerado válida esta condición tendríamos un **pseudografo** o **multigrafo**, dependiendo de que se aceptara la existencia de bucles o no.

Diremos que un **grafo** es **simple** si no tiene bucles ni varias aristas que unan un mismo par de vértices, en adelante grafo.

Llamamos **grado** o **valencia** de un vértice *v* al número de aristas incidentes con *v*. Para el caso que nos atañe, objetos o figuras tridimensionales, el mínimo grado de un vértice será 3.

6.2.3 Caminos y conexión.

Una de las propiedades más interesantes que puede poseer un grafo es la conexión.

Una **cadena** es una sucesión finita alternada de vértices y aristas

$$v_0\text{-}a_1\text{-}v_1\text{-}a_2\text{-}v_2\text{-}\ldots\ldots\text{-}a_k\text{-}v_k$$

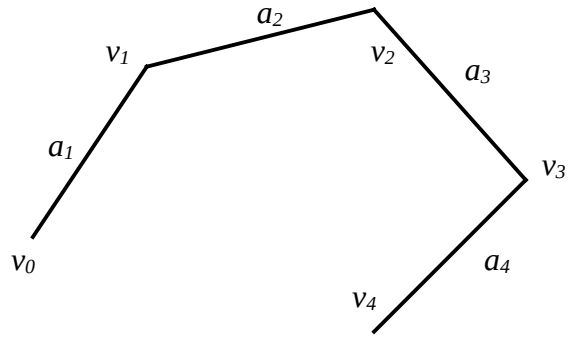


Figura 6: Ejemplo de una cadena

donde v_i pertenece al conjunto de vértices y a_i al conjunto de aristas y en la que cada arista a_i es incidente con los dos vértices inmediatamente anterior y posterior.

A v_o y v_k les llamamos **vértices inicial** y **final** de la cadena, siendo los restantes vértices los llamados **vértices internos**.

Una cadena decimos que es **simple** si las aristas son distintas dos a dos, y decimos que es un **camino** si son los vértices los que son distintos dos a dos. Todo camino será una cadena simple, al revés no es cierto.

Llamamos distancia entre dos vértices *u* y *v* o longitud de la cadena a la longitud del camino más corto de *u* a *v*.

Una cadena es **cerrada** si los vértices inicial y final coinciden, y llamamos **ciclo** a toda cadena cerrada donde los vértices interiores son distintos dos a dos, es decir, solo coinciden el vértice inicial y final siendo el resto distintos.

Una cara de un objeto será un ciclo.

Dado G un grafo, dos vértices están **conectados** si existe una cadena de longitud mayor o igual que 0 que los une. Para que el algoritmo de identificación de las caras funcione es condición imprescindible que **todos** los vértices de la figura estén conectados.

6.3 Teorema de euler.

6.3.1 Superficie poliédrica. Poliedro convexo.[9]

Llamaremos **superficie poliédrica** al conjunto de un número finito de polígonos, llamados **caras** de la superficie, que cumplan las condiciones siguientes:

1. Cada lado de una cara pertenece también a otra y sólo a otra. Ambas caras se llaman **contiguas**.

2. Dos caras contiguas están en distinto plano.

La superficie poliédrica se llama **convexa** si además se cumple:

3. El plano de cada cara deja en un mismo semiespacio las demás.

Llamaremos entonces **poliedro convexo** al conjunto de los puntos comunes a todos estos semiespacios. Los vértices y lados de las caras se llaman **vértices** y **aristas** del poliedro.

Los semiplanos de dos caras contiguas limitados por la arista común definen un diedro en cuyo interior se halla el poliedro.

Puesto que en cada arista concurren dos caras, en cada vértice de todo poliedro convexo concurrirán al menos tres aristas.

La definición implica, por otra parte, que los polígonos de las caras sean convexos. Su conjunto forma la superficie del poliedro. Los puntos del poliedro no situados en la superficie se llaman **interiores**; los demás del espacio se llaman **exteriores** al poliedro.

6.3.2 Propiedades de los poliedros convexos.

De la definición de poliedros convexos dada anteriormente, resulta:

Los segmentos que unen un vértice con un punto cualquiera de una cara que no le contiene, pertenecen al poliedro. En general: El segmento determinado por dos puntos del poliedro tiene todos sus puntos en él.

Las secciones planas de un poliedro convexo son polígonos convexos.

Establecido lo anterior, podemos demostrar para los poliedros convexos propiedades análogas a las de los polígonos convexos en particular (generalización del ***teorema de Jordan***[10] a poliedros convexos):

I. Toda semirrecta r con origen en un punto 0 interior a un poliedro convexo corta a la superficie en un punto. De donde: Toda recta que pasa por un punto interior la corta en dos puntos. Si una recta atraviesa en un punto la superficie, la corta en otro y sólo en otro.

[9] Superficie poliédrica. Poliedro convexo. Curso de geometría métrica. Tomo I fundamentos. Euler editorial s.a. Autor Puig Adam. Madrid 1986. Lección 37, página 235.

[10] Teorema de Jordan. Curso de geometría métrica. Tomo I fundamentos. Euler editorial s.a. Autor Puig Adam. Madrid 1986. Lección 2, página 14.

II. Existen rectas cuyos puntos son todos exteriores a la superficie.

III. Lo mismo el interior que el exterior son regiones conexas, es decir, se puede unir sus pares de puntos por quebradas que no cortan a la superficie.

IV. Toda quebrada que une un punto interior con otro exterior corta a la superficie.

V. El número de caras que tienen número impar de lados es par, y el número de ángulos sólidos o de vértices que tienen número impar de caras es par.

VI. No existe ningún poliedro convexo que tenga todas sus caras con más de cinco lados, ni todos sus ángulos sólidos con más de cinco aristas.

6.3.3 Superficies poliédricas y poliedros no convexos.

Existen superficies poliédricas no convexas; para construir una de ellas basta, por ejemplo, sustituir una cara ABCD de la superficie de un poliedro convexo por triángulos que resultan de proyectar sus lados desde el punto 0 interior al poliedro.

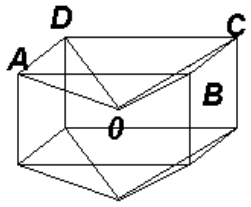


Figura 7: Poliedro no convexo

Las caras de la nueva superficie obtenida siguen verificando las condiciones 1 y 2 del punto 1.1, pero no la 3.

Entenderemos por **poliedro no convexo** limitado por la superficie del ejemplo anterior al conjunto de puntos resultante de suprimir del poliedro convexo de partida los del poliedro 0ABCD (menos los puntos de las caras triangulares que forman parte de la nueva superficie). Llamaremos puntos **interiores** a dicho poliedro los pertenecientes a él y no pertenecientes a su superficie y puntos **exteriores** a los restantes del espacio.

La Figura 7 pertenece a una clase de superficies poliédricas más general que las convexas, y que dividen al espacio en dos regiones conexas: es la de las superficies llamadas simples, definidas por las condiciones siguientes:

1. Las caras son polígonos simples, cada uno de cuyos lados pertenecen a dos caras y solo a dos llamadas **contiguas**.

2. Dos caras contiguas no son coplanarias. Modernamente se tiende a prescindir de esta condición y a manejar *redes triangulares* descomponiendo las caras en triángulos.

3. Dos caras no contiguas pueden unirse por una sucesión de caras contiguas, cada una con la anterior y la siguiente.

4. Dos caras no contiguas no pueden tener más punto en común que un vértice de ambas, y si le tienen deben pertenecer a un mismo anguloide de la superficie (es decir, a un mismo conjunto ordenado de caras concurrentes, cada una contigua con la anterior y la siguiente y la última con la primera, conjunto llamado también **ciclo**).

Se llama cuerpo poliédrico o poliedro simple al conjunto de puntos interiores a una superficie simple más los de la superficie. Según se considere esta superficie unida a una u otra región interior o exterior, diremos que define una u otra cara de dicha superficie, y en este sentido diremos que la superficie es de **dos caras**.

6.3.4 Teorema de Euler.

En todo poliedro convexo la suma del número ***c*** de caras más el número ***v*** de vértices excede en dos unidades al número **a** de aristas.

$$\boxed{c + v = a + 2}$$

6.3.5 Consecuencias del teorema de Euler.

Como cada cara contiene tres o más aristas, el número de ésta será $\geq 3c/2$ porque cada arista está en dos caras. Es decir, $3c \leq 2a$.

Análogamente, como en cada vértice concurren tres o más aristas y cada una de ellas une dos vértices, se tendrá $3v \leq 2a$.

Combinando las dos desigualdades con la igualdad de Euler $a + 2 = c + v$ resulta

$$3a + 6 = 3c + 3v \leq 4a \text{ de donde } \boxed{a \geq 6}$$

Combinando una sola de ellas se obtiene $3a + 6 = 3c + 3v \leq 2a + 3v$, de donde $a + 6 \leq 3v \leq 2a$ y análogamente $a + 6 \leq 3c \leq 2a$. Combinadas con $a \geq 6$ dan:

$$\boxed{c \geq 4, v \geq 4}$$

6.3.6 Poliedros Eulerianos.

Se llama **poliedro euleriano**[11] a todo poliedro cuya superficie es bilátera y simplenente conexa, y cuyas caras son simplemente conexas.

Se llama **superficie bilátera** aquélla en la que, partiendo de un punto de una de sus dos caras, y siguiendo la superficie, no se puede llegar a ese mismo punto, pero de la otra cara, sin atravesar la superficie. En cambio, una **superficie unilátera** es aquella que permite pasar de una cara a otra sin pasar por el borde.

[11] Poliedros eulerianos. Poliedros. Escuela Técnica Superior de Caminos, Canales y Puertos de Madrid. Cátedra de dibujo técnico y sistemas de representación. 1988. Páginas 9 y 10.

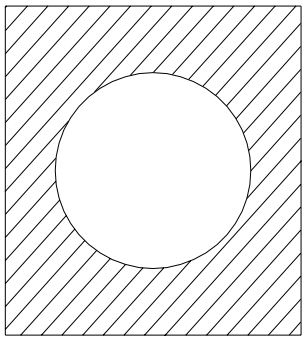


Figura 8: Superficie noNo simplemente conexa

Una superficie se llama **conexa**, cuando puede trazarse una línea continua de un punto a otro cualquiera sin que la línea de unión se salga de la superficie; por ejemplo, una superficie esférica es una superficie conexa y, en cambio, el conjunto de dos superficies esféricas que no se cortan, no es conexa. Una superficie conexa se dice que es **simplemente conexa**, cuando toda línea cerrada situada sobre la superficie puede reducirse por deformación a un punto sin que atraviese los bordes de la superficie; un ejemplo de estas superficies es un cuadrado y, en cambio, en la Figura 8 se indica una superficie que no es simplemente conexa.

Todos los poliedros eulerianos cumplen la relación de Euler, pero el recíproco no es cierto, pues existen poliedros no eulerianos (Figura 9) que cumplen la relación de Euler.

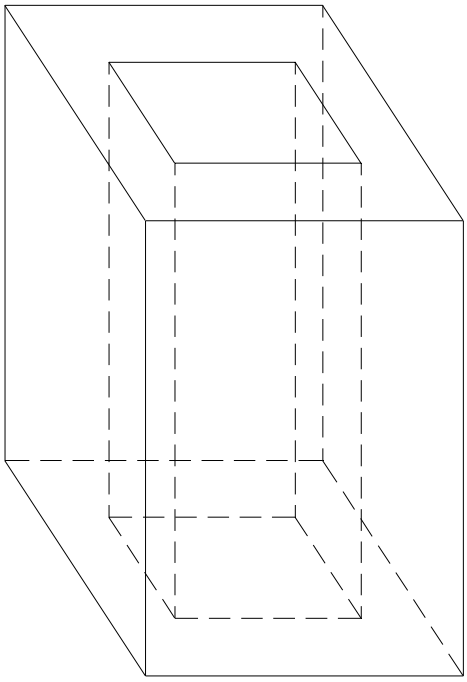


Figura 9: Poliedro no euleriano

6.4 Identificación de las caras.

6.4.1 Obtención de los vértices y aristas.

Para poder obtener la reconstrucción 3D de un objeto a partir de su representación en axonometría oblicua es condición indispensable que los datos de partida sean precisos y completos. Para ello se requiere que la información de **todos** los vértices y aristas del objeto estén definidos en la base de datos de AutoCAD. Una vez interrogada la misma, estos datos junto con otros que se deducirán a partir de ellos pasarán a formar parte de la *estructura de datos* diseñada al efecto.

La información contenida en la base de datos de AutoCAD la dividiremos en tres bloques de los cuales solamente utilizaremos dos, desechando la información del tercer bloque. Estos dos bloques son información concerniente a los vértices (sus coordenadas) e información concerniente a las aristas (vértices que las delimitan).

Dado que AutoCAD guarda la información agrupada en entidades, y que manejaremos solo modelos alámbricos, entonces los datos almacenados en la base de datos estará agrupada en entidades línea. Consecuencia de esto es que la información de las aristas y de los vértices se extraerán de forma paralela.

6.4.1.1 Obtención de los vértices.

La información que se recogerá de la base de datos del sistema CAD que estamos utilizando, referente a los vértices son las coordenadas X, Y de los mismos. Estos datos los obtendremos de manera rápida y sencilla, simplemente recorriendo dicha base de datos.

Un pequeño inconveniente que nos surge es que puesto qué la información de la base de datos de AutoCAD está agrupada en entidades línea, entonces las coordenadas de los vértices estarán repetidas en tantas aristas como sean acotadas por dicho vértice. Por lo tanto, y para no tener información redundante, a la vez que vayamos obteniendo los vértices, se comprobará que este todavía no esta incluido en nuestra estructura y en caso contrario incrementaremos su **grado o valencia**[12].

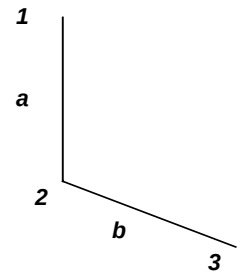


Figura 10: Obtención de vértices

Ejemplo:

Si observamos la

Figura *10*, cuando recojamos la información de la arista ***a***, a su vez introduciremos los datos de los vértices *1* y *2*.

Al introducir los datos de la arista ***b***, intentaremos introducir, también, los datos de los vértices *2* y *3*. Si introdujésemos de nuevo el vértice *2* entonces en nuestra estructura tendríamos información repetida. Lo que haremos será buscarlo en la lista de vértices y

[12] Grado o Valencia de un vértice es el número de aristas que confluyen en él. Puesto que estamos trabajando con figuras tridimensionales, la valencia de un vértice debe ser **tres o superior**.

si está, como en este caso, aumentarle su grado o valencia. De esta forma obtendremos una lista sin vértices repetidos.

Un vértice estará contenido en tantas caras como en aristas y este número vendrá indicado por su valencia.

Además de esta información almacenaremos una lista de aristas a las que pertenece el vértice y otra de caras.

6.4.1.2 Obtención de las aristas.

Una arista será una entidad línea de AutoCAD y estará delimitada por dos vértices y pertenecerá solo a dos caras.

La obtención de las aristas es tan fácil como recorrer la base de datos de AutoCAD. De aquí obtendremos los vértices que la acotan. A partir de estos datos obtendremos otros interesantes como su longitud y una lista de aristas paralelas a esta.

Una característica muy importante y que posteriormente se usará para la construcción de las caras, va a ser el **grado de saturación** de la arista. Esta característica, que inicialmente será cero, nos indicará posteriormente si la arista ya ha sido asignada a una cara, a dos o a ninguna. Por lo que nos será de gran utilidad para la construcción de las caras, como veremos en puntos posteriores.

Una vez obtenidas todas las aristas se comprobará cuales son paralelas entre sí. Esta información va ser importante por que la transformación a 3D la haremos por paralelismo entre las aristas.

6.4.2 Restricciones en la identificación de las caras.

En este apartado se exponen las restricciones que utilizaremos a la hora de buscar las caras, para que, de esta manera, no se tengan que recorrer todos los vértices y la búsqueda no sea exhaustiva.

Entre estas restricciones, la primera y fundamental es el **teorema de Euler** incluido en el apartado 0. La figura debe cumplir que el número de aristas mas dos sea igual al número de caras mas el de vértices.

Una forma de simplificar la búsqueda es limitándola al máximo número de aristas que pueda tener las caras. De esta forma cuando se haga la búsqueda de la cara el algoritmo retrocederá y buscará por otro lado en el momento en que se sobrepase ese máximo. Este máximo indicará la profundidad del árbol de búsqueda.

Otra restricción viene dada por la propia definición que hemos hecho de cara. Una cara la hemos definido como un ciclo, por tanto solo coincidirán el primer y último vértice, siendo el resto de vértices distintos dos a dos. De forma que si durante la búsqueda de la cara encontramos un vértice que se repite dentro de la lista de vértices entonces retrocederemos un nivel y buscaremos por otra arista.

Por último, pero no por ello menos importante, otra restricción viene dada por una de las propiedades de las aristas, una arista solo pertenece a dos caras. A esta propiedad la llamamos **grado de saturación**. Una arista esta **saturada** si las dos caras a las que pertenece ya están definidas. De esta forma se simplificará mucho el algoritmo

de búsqueda de caras puesto que si durante la búsqueda encontramos una arista saturada entonces pasaremos a la siguiente arista, dado que por ésta las caras ya están definidas.

Como se puede deducir el algoritmo de búsqueda de caras va a seguir la técnica de búsqueda en profundidad.

6.4.3 Técnica de búsqueda en profundidad.[13]

Para resolver con eficiencia muchos problemas relacionados con grafos, es necesario visitar los vértices y aristas de manera sistemática. La búsqueda en profundidad es una técnica importante para lograrlo.

Supóngase que se tiene un grafo G en el cual todos los vértices están marcados en principio como no visitados. La búsqueda en profundidad trabaja seleccionando un vértice v de G como vértice de partida; v se marca como visitado. Después se recorre cada vértice adyacente a V no visitado, usando recursivamente la búsqueda en profundidad. Una vez que se han visitado todos los vértices que se pueden alcanzar desde v, la búsqueda de v está completa.

Esta técnica se conoce como búsqueda en profundidad porque continúa buscando hacia adelante (en profundidad) mientras sea posible. Por ejemplo, supóngase que x es el vértice visitado más recientemente. La búsqueda en profundidad selecciona alguna arista no explorada $x \rightarrow y$ que parta de *x*. Si se ha visitado *y*, el procedimiento intenta continuar por otra arista que no se haya explorado y que parta de *x*. Si *y* no se ha visitado, entonces el procedimiento marca *y* como visitado e inicia una nueva búsqueda a partir de *y*. Después de completar la búsqueda de todos los caminos que parten de *y*, la búsqueda regresa a *x*, el vértice desde el cual se visitó *y* por primera vez. Se continúa el proceso de selección de aristas sin explorar que parte de *x* hasta que todas hayan sido exploradas.

6.4.4 Algoritmo general de identificación de las caras en 2D.

Una vez obtenidos los vértices y las aristas (apartado 6.4.1), se utilizará un "algoritmo de búsqueda de ciclos" para poder detectar las caras, que estarán formadas por bucles o ciclos de aristas y vértices donde el primer y último vértice coincidirán y no habrá ningún otro vértice que se repita dentro del ciclo.

En primer lugar buscaremos caras de tres lados, para ello llamaremos al algoritmo de identificación de caras limitando el número máximo de aristas que puede contener una cara a tres. Buscamos solo caras triangulares. A las aristas que formen estas caras les aumentaremos el contador de saturación de tal forma que si dicho contador llegase a dos, querría decir que las dos caras a las que pertenece la arista ya han sido encontradas y, por tanto, en los siguientes recorridos para buscar otras caras, si se llega a esa arista se desechará y se pasará a hacer la búsqueda por otro lado (PODA).

En segundo lugar repetimos la operación anterior con caras de cuatro lados.

Con estas dos fases habremos obtenido "*todas*" las caras de tres y de cuatro lados y también, habremos eliminado muchos de los posibles recorridos necesarios para

[13] Búsqueda en profundidad. Estructura de datos y algoritmos. Autores Aho, Hopcroft, Ullman. Editorial Addison-Wesley Iberoamericana. Temas 6 y 7.

identificar el resto de las caras, puesto que habrán muchas aristas totalmente saturadas, con lo que en el tercer apartado de búsqueda de caras no pasaremos por ellas, y por tanto el coste del algoritmo será menor.

Las caras así encontradas van a tener como característica especial el que el segundo vértice del bucle que la forma no tendrá valencia superior a cuatro. Esto es debido a que para hacer menos costoso el algoritmo el grado de este vértice nos indicará el número máximo de recorridos iniciales (ramas del árbol de búsqueda, en el segundo nivel: una vez elegida la primera arista de la cara, por cuantos caminos podemos ir para encontrar el resto de las aristas de la cara) que pueden existir, y está limitado a tres, uno menos que el grado del vértice. Por este motivo en la tercera fase del algoritmo los vértices con valencia superior a cuatro se dividirán en tantos vértices de valencia tres o cuatro como sean necesarios. Así aunque tengamos que recorrer todos los vértices para estudiar su grado o valencia, el algoritmo de búsqueda de ciclos resultará mas sencillo y mucho menos costoso.

Por último, y si todavía quedan caras por detectar, es decir todavía no se cumple el Teorema de Euler, comprobaremos la valencia de cada vértice de tal forma que si existe algún vértice con un grado o valencia superior a cuatro se dividirá en tantos vértices auxiliares como sean necesarios, como se explica en el punto 6.4.6. Si el número de vértices existentes es superior a 30, se supone que ya es un número lo suficientemente grande para el número de vértices de las caras de la mayoría de las figuras susceptibles de ser procesadas por el algoritmo, entonces se pedirá al usuario que introduzca el número máximo de aristas que puede tener una cara. Hay que tener en cuenta que este número va a indicar la "*profundidad del árbol de búsqueda"* y que, por tanto, cuanto mayor sea, más tardará el algoritmo en ejecutarse, más costoso será. Se repetirá el algoritmo de identificación de caras para el máximo número de vértices que tenga la figura o para el máximo número de lados que introduzca el usuario, según sea el caso. En esta tercera fase no se encontrarán caras sino *posibles caras*, es decir, ciclos o bucles de vértices y aristas que son susceptibles de ser caras. De estas posibles caras de más de cinco aristas, solo unas pocas pasarán a convertirse en verdaderas caras, eliminándose el resto. Será el usuario el que al ver coloreado el perímetro de la posible cara en la pantalla indicará si es una verdadera cara o no. Esto lo estudiaremos en el apartado 6.5.3.4.

El esquema general del algoritmo de identificación de caras 2D es:

1.1 Identificación de las caras 2D de 3 lados.

1.2 Identificación de las caras 2D de 4 lados.

1.3 Si todavía faltan caras por identificar.

- 1.3.1 Si el número de vértices es superior a 30 pedir al usuario que introduzca el número máximo de lados que puede tener una cara.
- 1.3.2 Análisis de las valencias de los vértices.
  - 1.3.2.1 Si el vértice tiene valencia superior a 4 disgregarlo.
- 1.3.3 Identificación de las posibles caras 2D que faltan.
- 1.3.4 Tratamiento de las posibles caras 2D y conversión a caras 2D.

6.4.5 Algoritmo de búsqueda de bucles o ciclos.

6.4.5.1 Descripción del algoritmo.

La parte principal para la identificación de las caras es el algoritmo de búsqueda de ciclos. Como datos de entrada del mismo tendremos tanto la información almacenada de los vértices como de las aristas, además del número máximo de lados que puede tener la cara.

Se utiliza la técnica de búsqueda en profundidad. Se parte de un vértice y de una de las aristas que inciden en él. A partir de estos se van recorriendo las aristas hasta llegar a ese mismo vértice, en cuyo caso obtendremos el bucle de una cara. Este proceso se repetirá con cada una de las aristas que inciden en dicho vértice.

Dado que la búsqueda de caras se hará recorriendo todos los vértices y aristas, deberemos de encontrar una forma de restringir esta búsqueda, puesto que es de un coste temporal muy elevado. El mayor porcentaje de tiempo de ejecución del programa es debido a esta búsqueda. Las restricciones implantadas en la búsqueda son las siguientes:

- Una vez encontradas las caras de 3 y 4 lados, y siempre que falten caras por identificar, el programa pasará a hacer la búsqueda más compleja de las caras de más de cuatro lados.

- Si cuando estamos recorriendo las aristas nos encontramos con una que ya está saturada entonces retrocederemos, dado que esa arista ya está asignada a dos caras.

- El bucle o ciclo se convertirá en posible cara solo en el caso de que el último vértice del mismo coincida con el primero.

- Si el vértice que estamos analizando en este momento coincide con alguno de los ya analizados en ese ciclo y no es con el primero, entonces no será posible definir un bucle de cara; se retrocederá. En caso de que coincida con el primero, el ciclo se habrá completado y por tanto será una posible cara. Una cara estará formada por un conjunto de vértices unidos por unas aristas, donde el único vértice que debe de coincidir será el primero con el último.

Ejemplo: La secuencia de vértices *1-2-3-4-1* forma una posible cara, pero la secuencia *1-2-3-4-2* no puede formar una posible cara.

Posteriormente, cuando analicemos las caras que surjan desde el vértice *2* entonces nos aparecerá el bucle *2-3-4-2* que sí que formará una posible cara.

- Para ahorrarnos recorridos ya hechos, el numero identificador del primer vértice del recorrido siempre será menor que el segundo. Cuando cargamos los vértices en sus estructuras los enumeramos, este número en el primer vértice debe de ser siempre menor que en el segundo. De esta forma nos ahorramos recorridos como el *i*.

Ejemplo: Si el bucle *1-2-3-4-1* es una posible cara

El bucle *i*: *2-1-4-3-2* identifica la misma posible cara que antes, pero no es tenida en cuenta puesto que el primer vértice es mayor que el segundo.

6.4.5.2 Esquema del algoritmo.

El esquema del algoritmo de búsqueda de los bucles o ciclos es el siguiente:

1 Escogemos el primer vértice, v_0.

2 Buscamos que aristas inciden en el vértice v_0 y escogemos la primera, a_0. Para cada una de ellas hacemos:

2.1 Comprobamos si la arista está saturada, si lo está entonces escogemos otra arista asociada a'_0, y repetimos el paso.

2.2 Comprobamos que el segundo vértice no sea inferior al primero, en cuyo caso se desecharía la arista y se escogería la siguiente y repetiríamos el paso 2. Seleccionamos el segundo vértice. Añadimos el vértice a la lista de vértices de la posible cara de forma ordenada.

3 Buscamos la segunda arista del recorrido y la añadimos a la lista de aristas siempre que no esté saturada, en cuyo caso retrocederemos en el árbol de búsqueda, escogeremos otra arista .

4 Mientras que el número máximo de aristas no exceda del límite de profundidad del árbol.

4.1 Si la arista no está saturada:

4.1.1 Seleccionamos el siguiente vértice.

4.1.2 Si no está en la lista de vértices lo incluimos y buscamos la siguiente arista. Repetiremos el algoritmo desde el paso 4, pero con este vértice v_i.

4.1.3 Si el vértice ya está incluido en la lista de vértices entonces:

4.1.3.1 Si coincide con el primer vértice entonces **posible cara**. Insertamos el ciclo o bucle en la lista de posibles caras y continuamos con el vértice "hermano"[14]. Repetiremos el algoritmo desde el paso 4 con este vértice.

4.2 Si coincide con otro vértice distinto del primero, entonces hemos encontrado un bucle o ciclo interior al recorrido que estamos analizando, que hemos de descartar. Pasamos al vértice anterior y buscamos otra arista incidente. Repetir desde el paso 4.

5 Una vez analizadas todas las aristas del vértice v_0, habremos encontrado un conjunto de posibles caras que estarán clasificadas por el siguiente orden:

- Recorrido al que pertenece. El número de recorrido vendrá indicado por el número de aristas que incidan en el segundo

[14] Vértice hermano es aquél que está en el mismo nivel en el árbol de búsqueda de ciclos

vértice, sin contar la primera arista que será incidente con el vértice v_0.

- Número de lados. Número de aristas que forman el bucle. De menor a mayor, primero estarán las posibles caras de tres lados, luego las de cuatro y así sucesivamente.

5.1 Comparamos las primeras posibles caras de los distintos recorridos. Si alguna tiene tres o cuatro lados (contando las aristas imaginarias), entonces la insertaremos en la lista de caras como una verdadera cara.

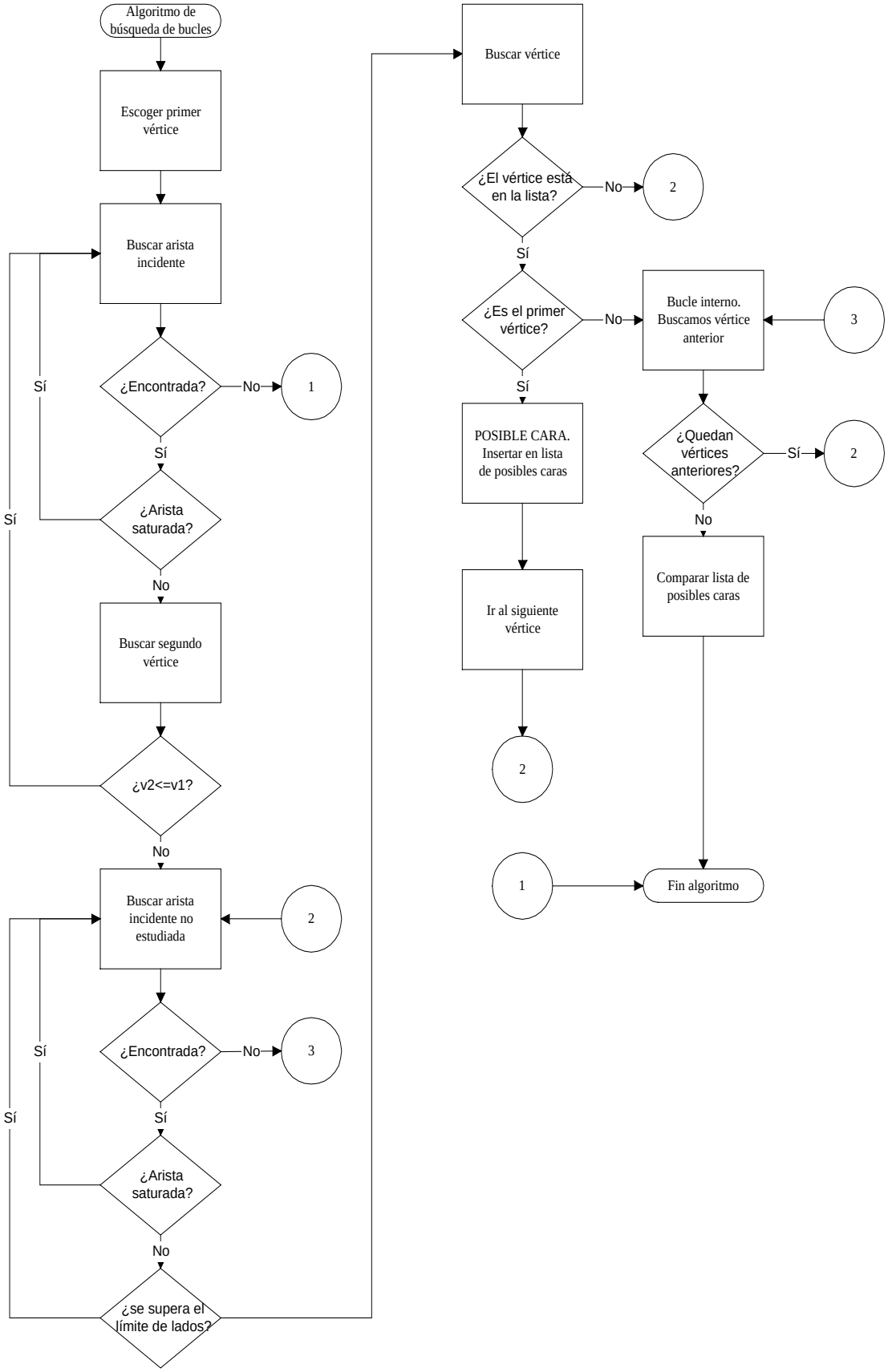
- Si encontramos dos caras de 3 ó 4 lados, cada una de un recorrido distinto, que cumple esta condición entonces, ya habremos encontrado las dos caras a las que pertenece la primera arista del recorrido a_0 por lo que el resto de posibles caras que contengan a dicha arista se eliminaran

- Si encontramos solo una cara de 3 ó 4 lados, entonces el resto de posibles caras del recorrido al que pertenece la cara encontrada se eliminaran y las de los restantes posibles recorridos pasarán a formar parte de una lista de posibles caras de más de cinco aristas, que se analizará con posterioridad.

- Si no encontramos ninguna cara entonces todas pasaran a formar parte de la lista de posibles caras de más de cinco aristas.

En definitiva, lo que estamos haciendo es una búsqueda en profundidad en el árbol de todos los posibles recorridos desde ese primer vértice y un posterior análisis de esos ciclos o bucles.

Un esquema representativo de este algoritmo sería:



Esquema 5: Procedimiento para la búsqueda de ciclos

6.4.5.3 Ejemplo.

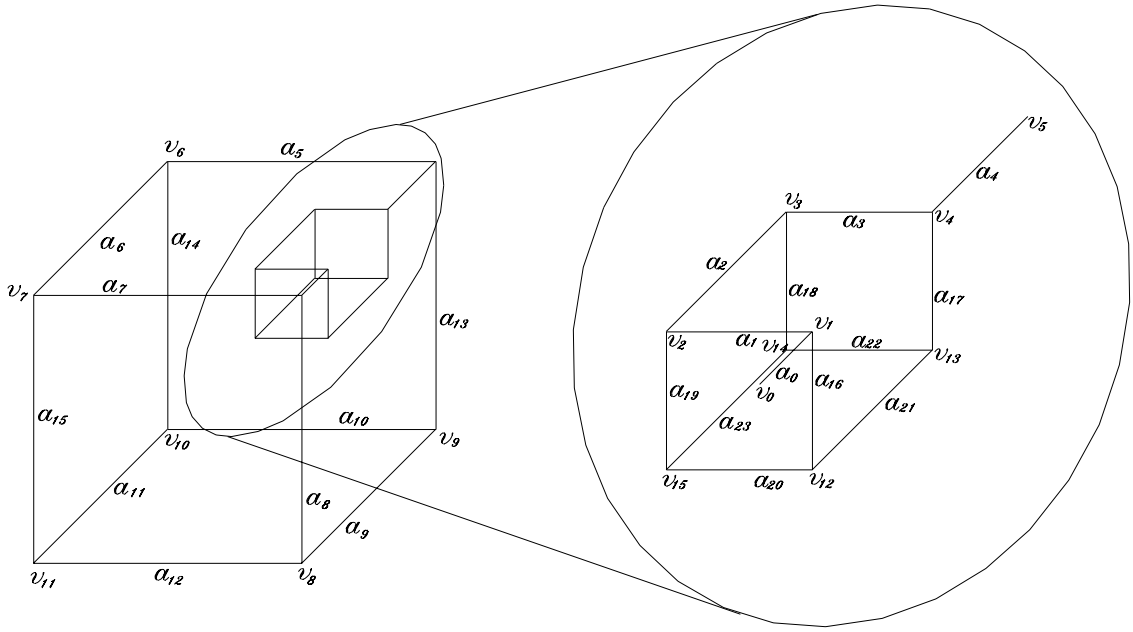


Figura 11: Ejemplo de obtención de las posibles caras de una figura

Sea la Figura 11, supongamos que las caras de cuatro lados ya han sido obtenidas, vamos a ver como se obtienen las caras de ocho lados.

Aristas ya saturadas: $a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a15$

Profundidad del árbol de búsqueda de ciclos = 15, número máximo de aristas.

1	Vértice inicial $\boldsymbol{v_0}$	
2	Arista inicial $\boldsymbol{a_0}$ (nº de caras asignadas=0)	v_0-a_0
3	2º vértice $\boldsymbol{v_1}$ (v_1>v_0)	v_0-a_0-v_1
4	Valencia del 2º vértice: 3	2 recorridos
5	Mientras que falten recorridos por hacer	
5.1	2ª arista $\boldsymbol{a_1}$ (nº de caras asignadas=1)	v_0-a_0-v_1-a_1
5.2	3er vértice $\boldsymbol{v_2}$	v_0-a_0-v_1-a_1-v_2
5.3	Mientras que no se exceda el límite de la profundidad del árbol de búsqueda, ir recorriendo todas las aristas	
5.3.1	Siguiente arista $\boldsymbol{a_2}$ (nº de caras asignadas=1)	v_0-a_0-v_1-a_1-v_2-a_2
5.3.2	Siguiente vértice $\boldsymbol{v_2}$	v_0-a_0-v_1-a_1-v_2-a_2-v_3

Si repetimos el paso 5.3 llegamos al ciclo

v_0-a_0-v_1-a_1-v_2-a_2-v_3-a_3-v_4-a_4-v_5-a_5-v_6-a_6-v_7-a_7-v_0.

Encontramos la POSIBLE CARA de la Figura 12

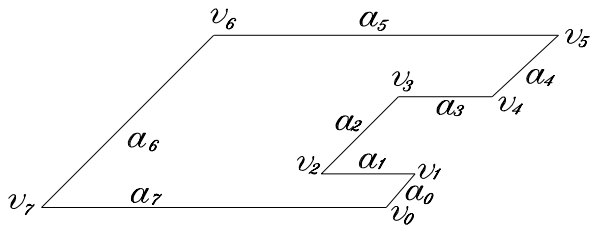


Figura 12: Posible cara verdadera

Otra POSIBLE CARA que encontramos es la Figura 13:

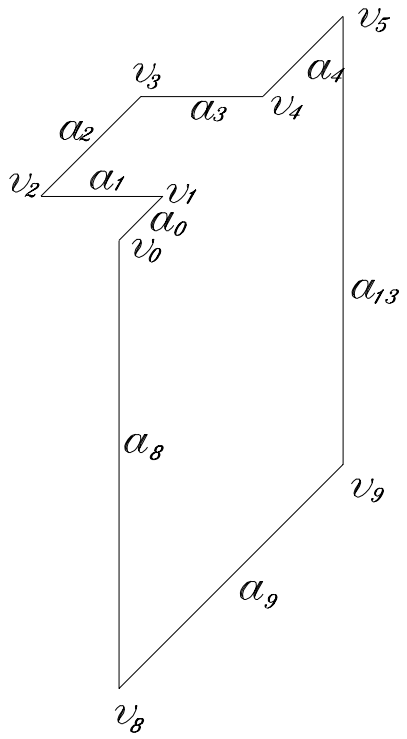


Figura 13: Posible cara falsa

Segundo recorrido:

5.1 2ª arista a_{16} (nº de caras asignadas=1) v_0-a_0-v_1-a_{13}

5.2 3er vértice v_{12} v_0-a_0-v_1-a_{13}-v_{12}

5.3 Mientras que no se exceda el límite de la profundidad del árbol de búsqueda, ir recorriendo todas las arista.

5.3.1 Siguiente arista a_{20} (nº de caras asignadas=2, ya saturada).

5.3.1 Siguiente arista a_{21} (nº de caras asignadas=1) v_0-a_0-v_1-a_{13}-v_{12}-a_{21}

5.3.2 Siguiente vértice v_{13} v_0-a_0-v_1-a_{13}-v_{12}-a_{21}-v_{13}

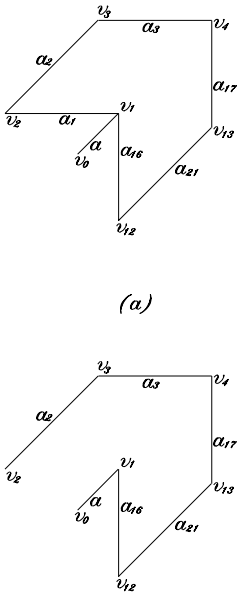


Figura 14: Retroceso

Se pueden dar casos como los de la Figura 14 *(a)*, donde hemos encontrado un bucle pero el vértice que se repite es el v_1, que no es el vértice inicial, por lo que se retrocederá y se pasará a la Figura 14 *(b)*. El algoritmo seguirá retrocediendo hasta la Figura 14 *(c)* donde volverá a avanzar hasta encontrar una POSIBLE CARA como la de la Figura 15.

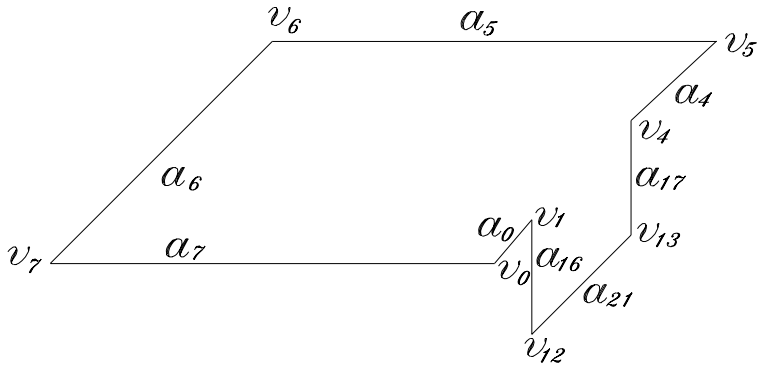


Figura 15: Posible cara falsa

Otra POSIBLE CARA que se obtiene con este 2º recorrido es la Figura 16.

Una vez obtenidas todas las posibles caras se pasarán a analizar como se indica en el punto 6.4.7. (se solicitará al usuario que las vaya identificando)

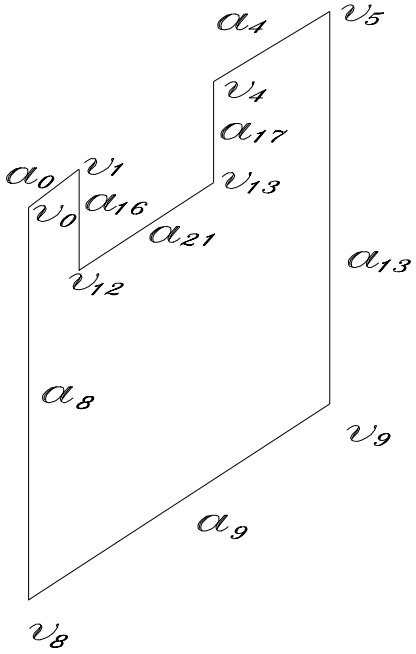


Figura 16: Posible cara verdadera

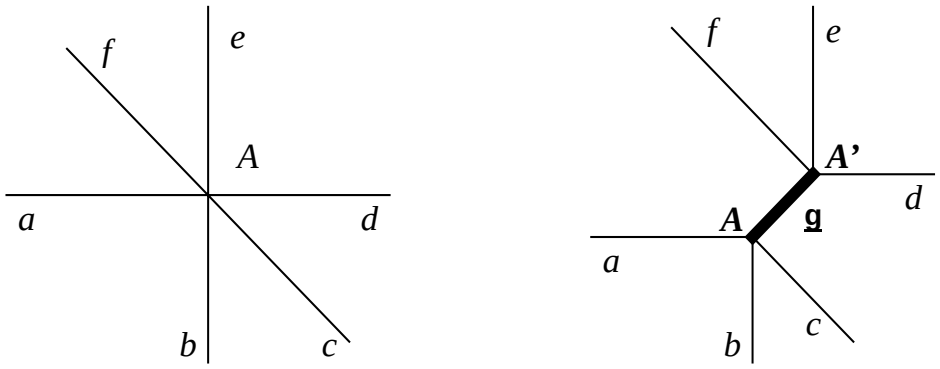
6.4.6 Análisis de la valencia de un vértice.

Llamamos **grado** o **valencia** de un vértice *v* al número de aristas incidentes en *v*. Dado que estamos trabajando con figuras tridimensionales el grado o valencia de un vértice debe de ser tres, cuatro ó superior.

En el caso de tener una valencia de tres ó cuatro, el algoritmo lo tratará directamente. Pero, si la valencia es superior a cuatro entonces el vértice se tratará de una forma especial.

Una vez averiguados todos los vértices y aristas pasaremos a tratar aquellos vértices con una valencia superior a cuatro. Encontrado un vértice de valencia superior a 4 lo dividiremos en tantos vértices ficticios como sean necesarios para que todo el conjunto de dichos vértices tenga valencia 4 ó inferior. Todos estos vértices auxiliares estarán unidos por unas aristas también auxiliares de longitud 0.0.

Esta operación se realizará en el momento en que estemos construyendo las caras. En primer lugar intentaremos averiguar las caras de tres y cuatro lados, si todavía quedasen caras para construir entonces pasaremos a desglosar los vértices de valencia mayor a cuatro y posteriormente a averiguar las caras de más de cuatro lados.



Antes de analizar *Después de analizar*

Figura 17: Valencia de un vértice

Como se puede observar en la Figura 17, al vértice ***A*** confluyen seis aristas, a saber: *a, b, c, d, e* y *f*. Es decir, tiene valencia 6. Este vértice se transformará en dos ***A*** y ***A'*** con valencia 4 respectivamente y apareciendo una arista auxiliar *g* de longitud 0.0.

6.4.7 Tratamiento de las posibles caras de más de cinco lados.

Una vez estudiadas todas las aristas y vértices y hechos todos los recorridos para la búsqueda de posibles caras, y habiendo identificado ya las caras de tres y cuatro lados incluidos los ficticios, el programa pasa a, con ayuda del usuario, detectar el resto de posibles caras.

Hay que recordar que este punto lo hace solo en el caso de que todavía falten caras por identificarse, y que esto lo sabremos aplicando el teorema de Euler.

Para identificar estas caras, de la lista de posibles caras de más de cuatro lados o aristas , se escogerá una y se modificará el atributo cromático de su perímetro, de manera que será el usuario tras la observación de este hecho en la pantalla, el que decida si es una verdadera cara o no.

Si el usuario considera que el ciclo o bucle que define el perímetro de la posible cara es una verdadera cara entonces, se insertará en la lista de caras y se eliminará de la lista de posibles caras de más de cuatro lados. Además se saturarán aquellas aristas que ya estén asociadas a dos caras y se recorrerá la lista de posibles caras de más de cuatro lados eliminando aquellas que se hayan quedado con alguna arista saturada.

Si este ciclo no es una verdadera cara, simplemente lo eliminaremos de la lista de posibles cara de más de cinco lados.

Este proceso lo repetiremos mientras falten caras por identificar. Una vez identificadas todas las caras entonces se eliminará la lista de posibles caras de más de cuatro lados.

Este proceso se hace interactivamente (con ayuda del usuario) y puede constituir el punto de partida de otros posibles trabajos para mejorar las prestaciones de este algoritmo.

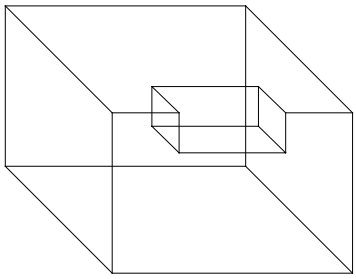


Figura 18: Ejempplo

Veamos un ejemplo. En la Figura 18, y una vez identificadas las caras de tres aristas (no hay ninguna) y de cuatro aristas (hay ocho), quedan por identificar las dos caras de ocho aristas existentes. La máquina no sabe cual de todos los posibles casos existentes constituyen las dos caras verdaderas (cara 1 y cara 2). Es por ello que le pide ayuda al usuario.

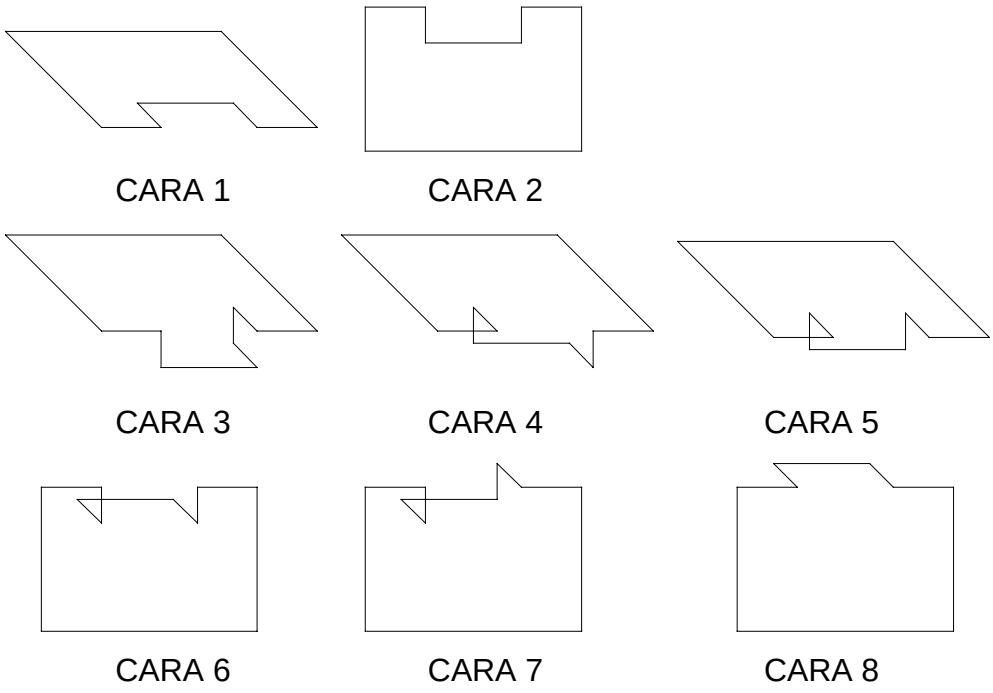


Figura 19: Algunas de las posibles caras encontradas por el algoritmo de búsqueda de ciclos

Siguiendo el ejemplo tratado en el apartado 6.4.5.3. Tenemos cuatro posibles caras identificadas. Se dibujará la primera (Figura 12), que sí que es una VERDADERA CARA, por lo que se procederá a aumentar el grado de saturación de sus aristas. (En este caso las aristas a_1, a_2, a_3, a_5, a_6 y a_7 quedarán asignadas a dos caras por lo que ya estarán saturadas.

Recorreremos la lista de POSIBLES CARAS y toda aquella posible cara donde exista una arista saturada la eliminaremos. (Se eliminarán las posibles caras representadas en la Figura 13 y Figura 14, por lo que nos quedarán solo como VERDADERAS CARAS las representadas en la Figura 20**.**

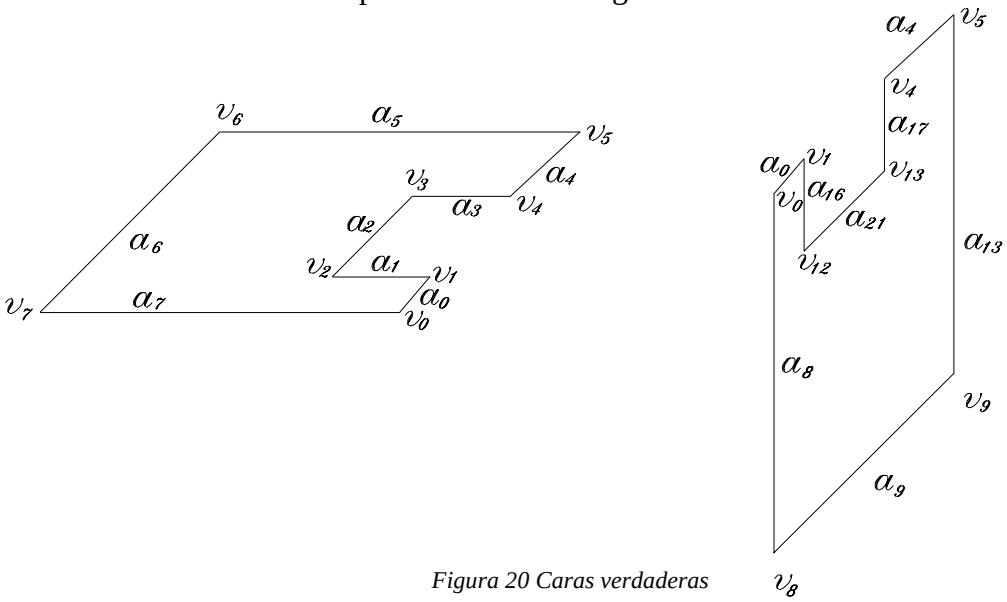


Figura 20 Caras verdaderas

Una posible mejora de prestaciones del algoritmo, en la identificación de este tipo de caras, discurriría por la vía de comenzar la oferta de posibles caras para la validación interactiva, por aquellas con mayor nivel de paralelismo entre las aristas que la forman, en relación al resto de aristas que conforman el objeto.

6.5 Obtención de la figura en 3D. Obtención de vértices y aristas del modelo.

6.5.1 Introducción.

La reconstrucción del modelo 3D dependerá, en gran medida, de los puntos elegidos por el usuario para el centro de coordenadas y los ejes así como de las escalas dadas para los mismos.

La obtención del modelo seguirá los siguientes pasos que se desarrollarán en puntos posteriores:

1 Introducción de los ejes de coordenadas y transformación de los mismos.

2 Obtención de las aristas 3D contenidas en los ejes coordenados introducidos por el usuario.

3 Obtención de las aristas 3D paralelas a los ejes y que tienen uno de los vértices ya con coordenadas 3D. En realidad el punto 1 se puede considerar como un subapartado de este, como veremos posteriormente.

4 Obtención del resto de aristas 3D. Para este caso utilizaremos dos algoritmos:

- 4.1 Obtención de aquellas aristas 3D con alguno de sus dos vértices en 3D y el otro obtenible en alguna de las caras que la contienen.
- 4.2 Obtención de aquellas aristas 3D no paralelas a ningún eje coordenado, pero que al menos una de las caras a la que pertenece tenga dos aristas no paralelas entre sí en 3D.

Para aquellos casos no contenidos en los anteriores, el algoritmo no podrá calcular las coordenadas 3D del vértice y/o arista en cuestión, avisando de ello.

6.5.2 Introducción de los ejes.

Una vez identificadas las caras de la figura, el algoritmo pasa a obtener las coordenadas 3D de los vértices y por tanto, a "transformar" la figura 2D a 3D. Para ello, el algoritmo solicita al usuario, en primer lugar la relación de las aristas 2D que van a permitir los ejes coordenados, así como las escalas de éstos. Dado que es el usuario el que elige la escala de los ejes, la figura puede llegar a transformarse de tal forma que la imagen 2D inicial puede llegar a ser una representación más o menos distorsionada del modelo. En cualquier caso, este hecho confiere al usuario unos generosos grados de libertad para prefijar los parámetros de la representación axonométrica.

Para determinar el sistema de referencia se introducen los siguientes puntos en 2D:

- Centro del eje de coordenadas. (c_0)
- Punto perteneciente al eje X. (x_0)
- Punto perteneciente al eje Y. (y_0)
- Punto perteneciente al eje Z. (z_0)

Estos puntos pueden estar en cualquier parte del plano del dibujo 2D, pero lo más recomendable es que pertenezcan a la figura que deseamos transformar, dado que de esta forma la obtención de la figura en 3D será mucho más sencilla. Para evitar problemas, es muy recomendable que al elegir cada punto de un eje estos estén unidos al escogido como centro del eje de coordenadas por una arista. De esta forma, dicha arista se transformará de forma directa a 3D.

Una vez introducidos estos puntos, se solicitarán las escalas con respecto a cada eje. Si el usuario no cambia las escalas existentes por defecto (1 para cada eje) entonces el programa utilizará las longitudes de las aristas para poder obtener los vértices y aristas en 3D. Si las escalas se cambian entonces las longitudes de las aristas se multiplicarán por el valor de las escalas en cada una de las coordenadas.

Estos punto se transformarán a 3D de la siguiente forma:

- El centro de coordenadas será el punto (0,0,0)
- El resto de los puntos pasarán a tener como coordenada la distancia del punto al centro de coordenadas, es decir
    - $x_0 \Rightarrow (escaX \times dist(c_0, x_0),0,0)$
    - $y_0 \Rightarrow (0, escaY \times dist(c_0, y_0),0)$
    - $z_0 \Rightarrow (0,0, escaZ \times dist(c_0, z_0))$

Si el usuario ha hecho coincidir estos puntos con algunos vértices entonces los vértices serán transformados de forma directa.

6.5.3 Transformación 3D de las aristas.

Obtenidos los ejes, se pasará a obtener las aristas 3D. Para ello seguiremos el siguiente esquema:

1. Transformación de las aristas coincidentes con los ejes. Como se ha dicho en el apartado anterior, si los ejes coinciden con aristas del objeto entonces éstas se habrán transformado de forma directa al haber obtenido los ejes de coordenadas en 3D.
2. Transformación de las aristas con un vértice ya transformado a 3D y paralelas a uno de los ejes de coordenadas. Aquí distinguiremos en función de que:
    - El vértice ya transformado se encuentra en el eje de coordenadas
    - El vértice ya transformado no se encuentra en el eje de coordenadas
3. Transformación de otras aristas con al menos un vértice ya transformado
4. Transformación del resto de aristas.

La longitud de las aristas y las relaciones de paralelismo entre éstas y los ejes coordenados van a ser los factores que faciliten la determinación de los vértices 3D. Por lo que antes de pasar a la transformación en sí, comprobaremos qué aristas son paralelas a los ejes.

6.5.3.1 Transformación de las aristas coincidentes con los ejes.

Una vez obtenidos los ejes de coordenadas 3D pasaremos a buscar si existe alguna arista que coincida con alguno de ellos, en cuyo caso la transformación de estas aristas va a ser directa. El vértice que coincida con el centro de coordenadas pasará a tener como nuevas coordenadas (0,0,0). El vértice que coincida con el punto perteneciente a alguno de los ejes pasará a tener de coordenadas la longitud por la escala del eje en dicho eje y cero en el resto de los ejes.

Esta operación se repetirá para cada uno de los ejes. Y en definitiva, las coordenadas 3D de los vértices situados en los ejes serán:

- $(escaX \times dist(c_0, x_0), 0, 0)$ para los vértices contenidos en el eje *X*
- $(0, escaY \times dist(c_0, y_0), 0)$ para los vértices contenidos en el eje *Y*
- $(0, 0, escaZ \times dist(c_0, z_0))$ para los vértices contenidos en el eje *Z*

6.5.3.2 Transformación de las aristas con un vértice ya transformado a 3D y paralelas a uno de los ejes de coordenadas.

Para el resto de aristas paralelas a los ejes coordenados, iremos recorriendo para cada eje aquellas aristas que son paralelas a éstos y las transformaremos de forma semejante a como hemos transformado las que coinciden con los ejes.

Tendremos un bucle que se repetirá hasta que en alguna de las pasadas **NO** se transforme ninguna arista, o no queden más aristas que transformar. En este caso cada pasada estará formada por un recorrido a las aristas paralelas a cada uno de los ejes.

En este caso, como en los anteriores, estamos tratando de obtener aristas que son paralelas a uno de los ejes y por tanto son perpendiculares al resto de los ejes, de los que solo consideraremos uno. Para lograr este objetivo basta obtener las coordenadas de uno de los vértices que delimitan estas aristas, ya que el otro tiene que estar transformado. Con estas condiciones las ecuaciones de transformación se pueden obtener como sigue.

Supongamos una arista a_i cuyos vértices a los que incide son v_1 (x_1,y_1) y v_2 (x_2, y_2). Así mismo supongamos que el vértice que ya está transformado es el v_1 y sus coordenadas son $(x_1^{3D}, y_1^{3D}, z_1^{3D})$. Las ecuaciones que permiten obtener las coordenadas 3D del vértice v_2 son:

1 Arista a_i paralela al eje *X*.

1.1 $x_2^{3D} = x_1^{3D} + escaX \times longa_1$

1.2 $y_2^{3D} = y_1^{3D}$

1.3 $z_2^{3D} = z_1^{3D}$

2 Arista a_i paralela al eje *Y*

2.1 $x_2^{3D} = x_1^{3D}$

2.2 $y_2^{3D} = y_1^{3D} + escaY \times longa_1$

2.3 $z_2^{3D} = z_1^{3D}$

3 Arista a_i paralela al eje *Z*

3.1 $x_2^{3D} = x_1^{3D}$

3.2 $y_2^{3D} = y_1^{3D}$

3.3 $z_2^{3D} = z_1^{3D} + escaZ \times longa_1$

Como se puede observar en las fórmulas de obtención de los vértices, solo variará aquella coordenada que sea la del eje al que es paralela la arista. El resto de coordenadas serán iguales en ambos vértices de la arista.

6.5.3.3 Obtención del resto de aristas.

Por último, pasaremos a transformar aquellas aristas que no son paralelas a ninguno de los ejes. Estas aristas sólo se podrán transformar si se cumple uno de los siguientes requisitos.

1. La arista posee uno de sus vértices 3D y el otro es obtenible en una de las caras que contienen a la arista.
2. La arista que deseamos obtener pertenece a alguna cara que tiene dos aristas no alineadas entre sí ya obtenidas en 3D.

6.5.3.3.1 La arista tiene uno de sus vértices ya obtenido en 3D y el otro es obtenible en una de las caras que contienen a la arista.

Supongamos que deseamos obtener en 3D el vértice $v_3(x_3,y_3)$ de la Figura 21. Para ello:

1. Escogemos una cara donde pertenezca el vértice. Cara obtenida por los vértices $v_1(x_1,y_1)$, $v_2(x_2,y_2)$ y $v_3(x_3,y_3)$ en la figura.
2. Buscamos dos aristas paralelas a dos ejes de coordenadas distintos dentro de la cara escogida y que ya estén transformados a 3D. Si no las encontramos entonces volvemos al apartado 1. En este caso las aristas son la *b* (paralela al eje *Z*) y la *c* (paralela al eje *X*).
3. Buscamos un punto auxiliar ($v_4(x_4,y_4)$) en una de las aristas obtenidas en el apartado 2 (arista *c*), de manera que unido junto con el vértice que queremos transformar (v_3), forme una arista paralela a la otra arista (arista *b*). La arista así obtenida la nombraremos como arista *a*.

Para obtener v_4 nos basamos en las ecuaciones de las rectas en 2D y en sus pendientes. Para el caso de la Figura 21

$$y_4 - y_3 = m_1 \times (x_4 - x_3) \qquad (1)$$

$$y_4 - y_1 = m_2 \times (x_4 - x_1) \qquad (2)$$

donde

$$m_1 = \frac{E_{yf} - E_{yi}}{E_{xf} - E_{xi}} \qquad (4)$$

$$m_2 = \frac{y_1 - y_2}{x_1 - x_2} \quad (3)$$

donde E_x, E_y, E_z indicarán el eje al que es paralelo la arista *b*. E_{xi}, E_{yi}, E_{zi} indicará la coordenada inicial y E_{xf}, E_{yf}, E_{zf} la coordenada final del eje (vértice inicial y final respectivamente). m_1 y m_2 son las pendientes de las rectas (1) y (2) respectivamente.

No se puede cumplir que $x_1 - x_2 = 0$ y $E_{xf} - E_{xi} = 0$ puesto que entonces $m_1 = \infty$ y $m_2 = \infty$ y por tanto, tienen la misma pendiente, entonces las aristas *b* y *c* serían paralelas. Por tanto no se cumple la condición de que ambas aristas (*b* y *c*) no pueden ser paralelas (apartado 2).

Si $x_1 - x_2 = 0$ entonces $x_4 = x_1$

$$y_4 = m_2 \times (x_4 - x_3) + y_3 \qquad (5)$$

Si $E_{xf} - E_{xi} = 0$ entonces $x_4 = x_3$

$$y_4 = m_1 \times (x_4 - x_1) + y_1 \qquad (6)$$

En otro caso $$x_4 = \frac{m_2 \times x_3 - y_3 - m_1 \times x_1 + y_1}{m_2 - m_1}$$

y_4 puede ser (5) ó (6)

4. Obtenemos en 3D el punto v_4 por paralelismo. Para ello aplicamos el apartado 6.5.3.2
5. Transformamos el vértice deseado por paralelismo aplicando el apartado 6.5.3.2
6. Pasamos al siguiente vértice sin transformar y repetimos los pasos anteriores.

Obtención de la figura en 3D. Obtención de vértices y aristas del modelo.

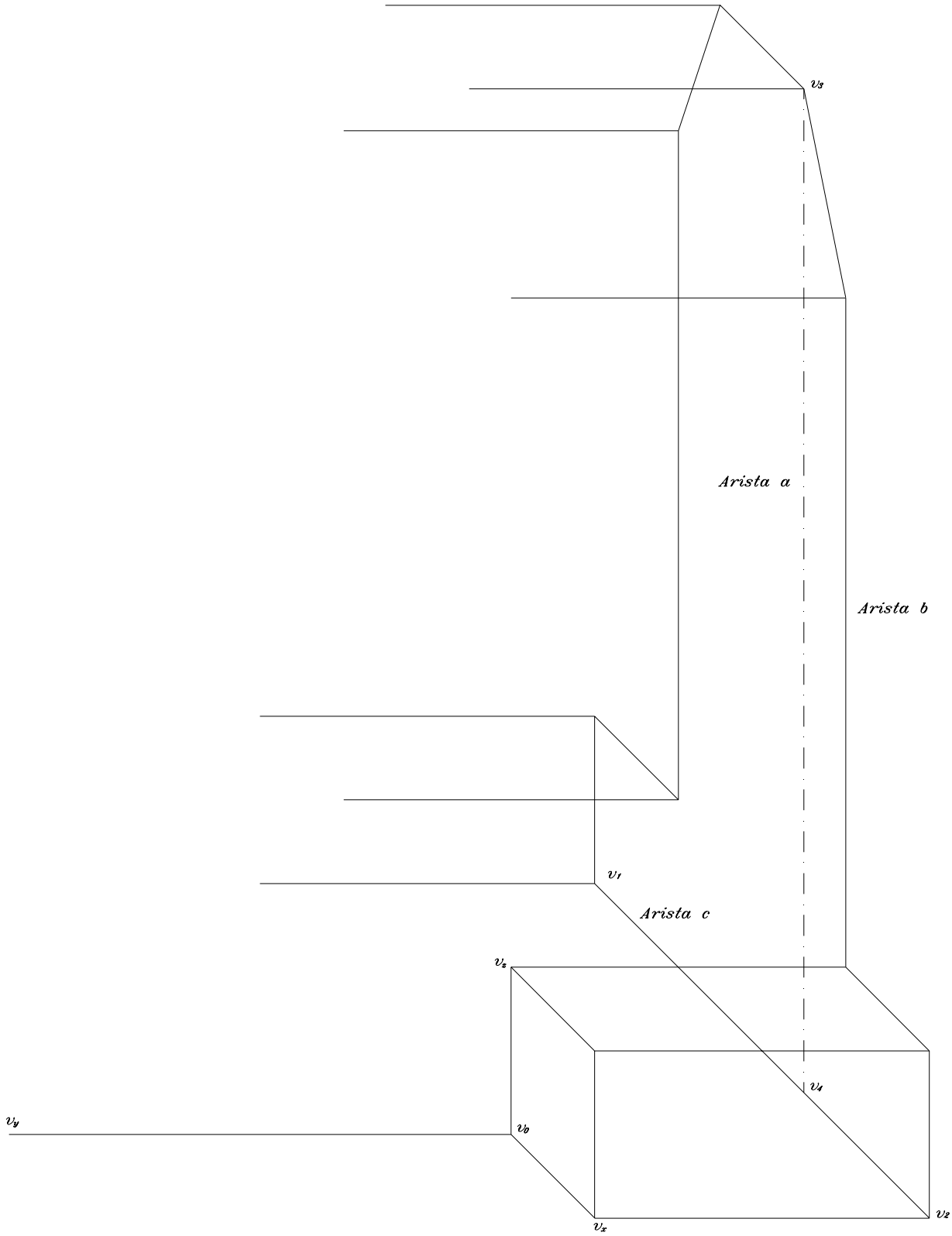


Figura 21: Obtención de vértices y aristas

6.5.3.3.2 La arista que deseamos obtener pertenece a alguna cara que tiene dos aristas no alineadas entre sí ya obtenidas en 3D.

Teorema de Thales

Si dos rectas *r* y *r'* se cortan por un sistema de paralelas, los segmentos determinados por los puntos de intersección sobre una de ellas son proporcionales a los determinados por los puntos correspondientes en la otra.

División de un segmento en partes proporcionales a segmentos dados

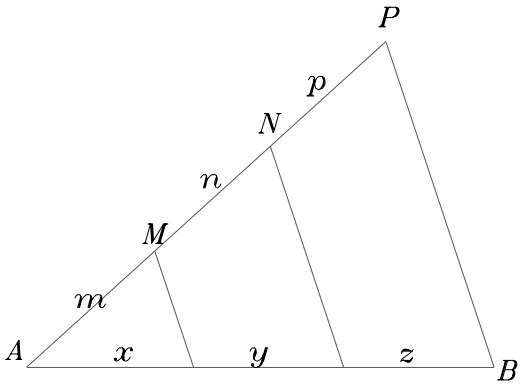


Figura 22: Teorema de Thales

El Teorema de Thales nos permite dividir con gran sencillez un segmento dado *AB* en partes proporcionales a otros *m*, *n*, *p*. Llevando estos segmentos consecutivamente sobre una semirrecta concurrente con el segmento dado *AB* en uno de sus extremos *A*, y uniendo el extremo *P* de la suma *m+n+p*, así construida, con el extremo *B* del segmento. Las paralelas a *PB* por los puntos de división *M* y *N* determinarán en el segmento dado los segmentos *x*, *y*, *z* proporcionales a *m*, *n*, *p* y cuya suma es *AB*.

Determinación de puntos de una recta por su razón de distancias a dos de ellas.

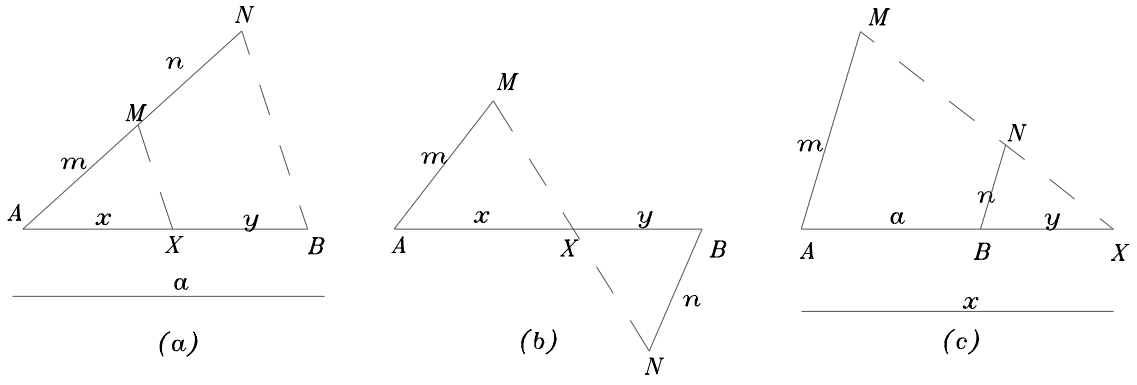


Figura 23: Determinación de puntos de una recta por su razón de distancias a dos de ellas

Cuando las partes proporcionales en que se divide un segmento *AB* son dos *m*, *n*, el problema anterior es equivalente al de hallar un punto *X* del segmento *AB*, cuya razón de distancias a *A*, *B* sea igual a *m:n*. Dados *A* y *B* en un orden, no hay mas que un punto interior *X* que cumpla esta condición, puesto que el segmento *AX=x* es el único que verifica la proporción *(m+n):m=a:x*, que se desprende de la construcción. Este punto puede también obtenerse como indica la Figura 23 (b), o sea llevando los segmentos *m* y *n* a partir de *A* y *B* sobre dos semirrectas paralelas situadas en semiplanos distintos

respecto de *AB*, el punto *X* de intersección de la recta *MN* que une los extremos con *AB* es el punto pedido, puesto que se verifica *x:y=m:n*.

Si llevamos *m* y *n*, Figura 23 (c) sobre semirrectas paralelas situadas en el mismo semiplano obtendremos (si $m \neq n$) un punto *X'* exterior cuya razón de distancias a *A* y *B* es asimismo X'A:X'B=m:n, y es el único punto exterior que la cumple, puesto que tiene que verificar asimismo en valor absoluto y signo:

$$\frac{AX'}{AX'-BX'} = \frac{m}{m-n} \quad \text{, es decir,} \qquad \frac{AX'}{AB} = \frac{m}{m-n}$$

X' está pues en la semirrecta *AB* si el segundo miembro es positivo, es decir si *m>n*, y en la semirrecta opuesta en caso contrario.

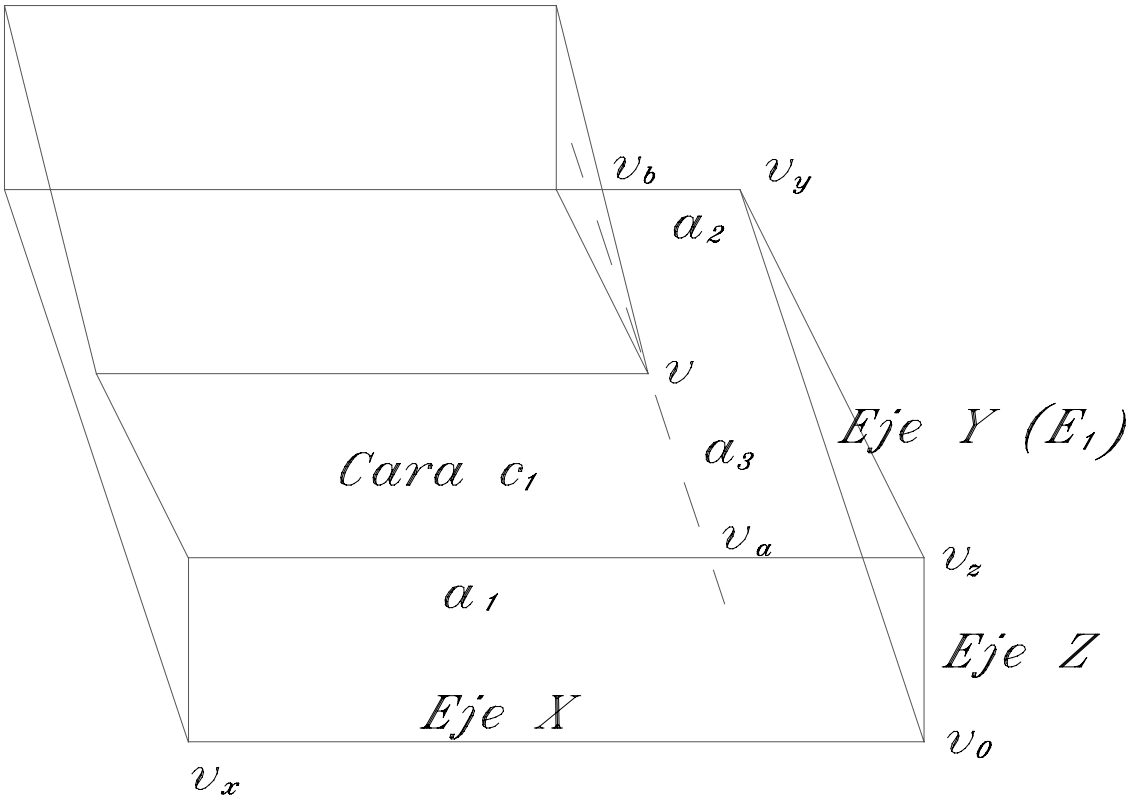


Figura 24: Aplicación del teorema de Thales

Aplicación a la transformación de aristas 3D

Si todavía quedan vértices que obtener entonces se utilizará un método totalmente diferente al utilizado hasta ahora para poder obtenerlo. El método consiste en buscar dos puntos que en dos aristas no alineadas que ya estén transformadas y por medio de proporciones (aplicando el teorema de Thales) obtener el vértice que deseamos. En este caso los pasos a seguir son (Figura 24):

1. Buscar una cara c_1 donde se encuentre dicho vértice *v*.
2. Buscar dos aristas en dicha cara c_1 que cumplan que no están alineadas y que estén ya transformadas a *3D* (sus vértices ya están en *3D*). Las denominaremos a_1 y a_2 respectivamente.

3. Seleccionamos un eje de coordenadas al que ninguna de las dos aristas anteriormente encontradas es paralela. Lo llamaremos E_1. Así podremos crear una arista a_3 paralela a dicho eje y que pase por el vértice v que deseamos obtener. (Aplicamos el Teorema de Thales)

4. Buscamos un punto auxiliar (v_a) en una de las aristas obtenidas en el apartado 2 (arista a_1), que unido junto con el vértice que queremos transformar (v), forme una arista paralela al eje E_1.

5. Repetimos el apartado 4 obteniendo otro punto auxiliar (v_b) distinto de (v_a).

6. Dado que las distancias (proporciones) en 2D se mantienen en 3D entonces

    - Obtenemos el vértice v_a en 3D por proporciones,
    - Obtenemos el vértice v_b en 3D por proporciones, y
    - Obtenemos el vértice v en 3D por proporciones.

Para el cálculo de los puntos auxiliares v_a y v_b se utilizarán las ecuaciones descritas en el punto 6.5.3.3.1 (punto anterior).

Para el cálculo de los vértices por proporciones o distancias utilizaremos las siguientes ecuaciones:

Supongamos tres puntos p_1, p_2 y p_3. Los puntos p_1 y p_2 ya están obtenidos en 3D y queremos obtener el punto p_3 en 3D.

- Calculamos la razón[15] r de distancias de p_3 a p_1 y p_2 en la recta (en 2D).

Si $p_{1x} \neq p_{3x}$ y $p_{2x} \neq p_{3x}$ entonces $r = \dfrac{p_{3x} - p_{1x}}{p_{2x} - p_{3x}}$

sino si $p_{2y} \neq p_{3y}$ entonces $r = \dfrac{p_{3y} - p_{1y}}{p_{2y} - p_{3y}}$

en otro caso no se podrá calcular.

- Una vez obtenida la razón r y como ésta se mantiene en 3D entonces pasamos a calcular las coordenadas en 3D del punto p_3 directamente[16]:

$$p_{3x}^{3D} = \frac{p_{1x}^{3D} + r \times p_{2x}^{3D}}{1+r}$$

$$p_{3y}^{3D} = \frac{p_{1y}^{3D} + r \times p_{2y}^{3D}}{1+r}$$

$$p_{3z}^{3D} = \frac{p_{1z}^{3D} + r \times p_{2z}^{3D}}{1+r}$$

[15] Razón = Proporción

[16] Para una demostración de estas ecuaciones dirigirse al libro “Geometría analítica”, autor Charles H. Lehmann. Editorial Limusa. México. Pág.324 teorema 2

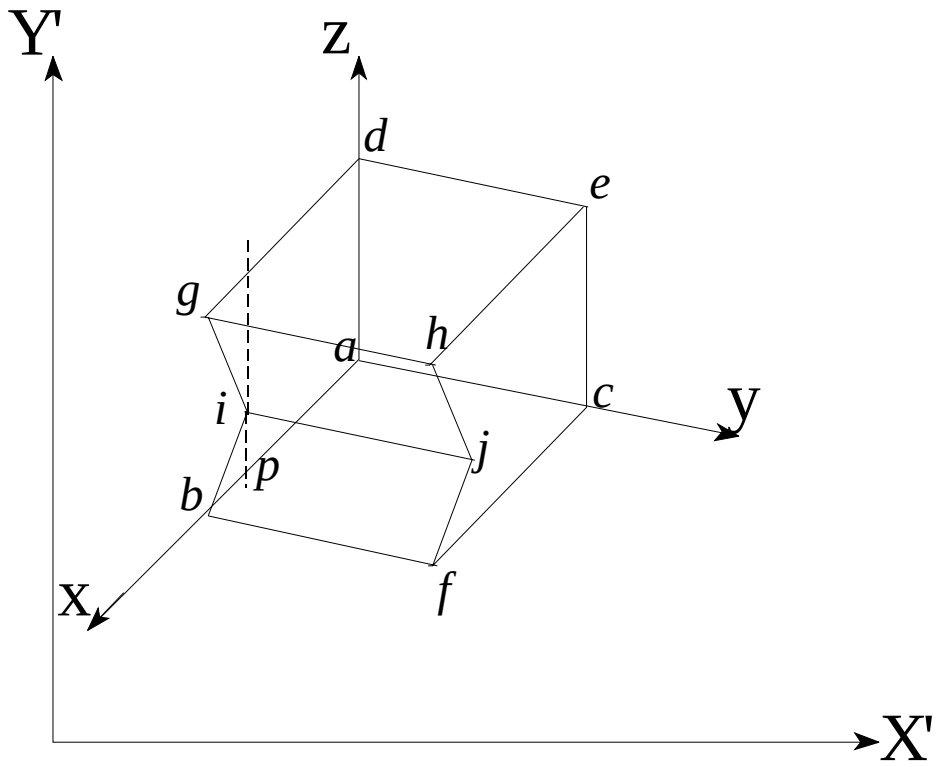
6.5.3.4 Transformación del resto de aristas.

Para las aristas que no hayan sido transformadas aplicando los apartados anteriores, el algoritmo diseñado no dispone de recursos para transformarlas de forma automática.

La casuística que genera la aparición de aristas no previstas para su transformación 3D en ninguno de los apartados anteriores, constituye un campo para futuros trabajos en la línea de la reconstrucción de poliedros 3D a partir de una única imagen 2D del mismo.

6.5.3.5 Ejemplo.

Pasemos a ver un ejemplo sencillo de obtención de los vértices y aristas de una figura:



Veamos como transformar la figura anterior de 2D a 3D.

En primer lugar escogeremos los ejes. Para ello elegimos como centro de coordenadas el punto *a*; como punto en el eje X el *b*; como punto perteneciente al eje Y el *c* y como punto perteneciente al eje Z el *d*.

Una vez obtenidos los ejes pasamos a transformar las aristas:

- Primero las que coinciden con los ejes. Estas aristas se transformarán de forma directa y serían:
    - *ab*, que coincide con el eje X
    - *ac*, que coincide con el eje Y, y

- *ad*, que coincide con el eje Z.

- Transformación de las aristas con un vértice ya transformado y paralelas a uno de los ejes:
    - *cf* paralela al eje X, transformado el vértice *c*.
    - *gd* paralela al eje X, transformado el vértice *d*.
    - *bf* paralela al eje Y, transformado el vértice *b* o el vértice *f*. transformada directamente puesto que ambos vértices lo están.
    - *de* paralela al eje Y, transformado el vértice *d*.
    - *ce* paralela al eje Y, transformado el vértice *c*.
    - *eh* paralela al eje X, transformado el vértice *e*.
    - *gh* paralela al eje X, transformado el vértice *g* o el vértice *h*. Transformada directamente, puesto que ambos vértices lo están.
- Transformación de otras aristas con al menos un vértice ya transformado:
    - Arista *gi*. Queremos transformar el vértice *i*. Esta arista no es paralela a ninguno de los ejes. Seguiremos los siguientes pasos:
        1. Escogemos la cara: *abigd*
        2. Buscamos dos aristas paralelas a dos ejes distintos:
            - *ab* paralela al eje X. Es el propio eje.
            - *ad* paralela al eje Z. Es el propio eje.
        3. Buscamos un punto auxiliar *p* el una de estas aristas (por ejemplo en *ab*) que unido con el vértice *i* forma una arista paralela a la arista *ad*.
        4. Transformamos a 3D el punto *p*. La arista *ap* es paralela al eje X y tiene transformado el vértice *a*.
        5. Transformamos a 3D el punto *i*. La arista *ip* es paralela al eje Z y tiene transformado el vértice *p*.
    - Arista *hj*. Se siguen unos pasos equivalentes al caso anterior.
    - Resto de aristas: No habrían más aristas por transformar.

6.6 Construcción de las caras en 3d.

6.6.1 Introducción.

Hasta este momento hemos identificado las caras de la figura a reconstruir en 2D y hemos obtenido sus vértices y aristas en 3D. Tenemos una nube de puntos en 3D unidos por un conjunto de aristas que forman el modelo alámbrico de la figura, pero no hemos obtenido las caras de la misma en 3D. Es decir si a la figura le aplicamos un renderizado o unas luces y sombras, el sistema CAD que utilicemos para ello nos advertirá que no puede hacerlo puesto que no tiene definida ninguna entidad superficial. Este va a ser el último paso que va ha realizar el algoritmo, Partiremos de esa nube de puntos y ese conjunto de aristas en 3D, así como de la topología de las caras definidas en 2D, y obtendremos las caras 3D del objeto.

Al aplicar esta orden de AutoCAD nos pueden surgir problemas en algunas caras que inciden en la mala formación de las mismas, debido a la forma de actuar de la propia orden 3DCARA (Figura 25). Más concretamente, debido al proceso de triangularización de la superficie de la cara, implementado en la orden 3DCARA.

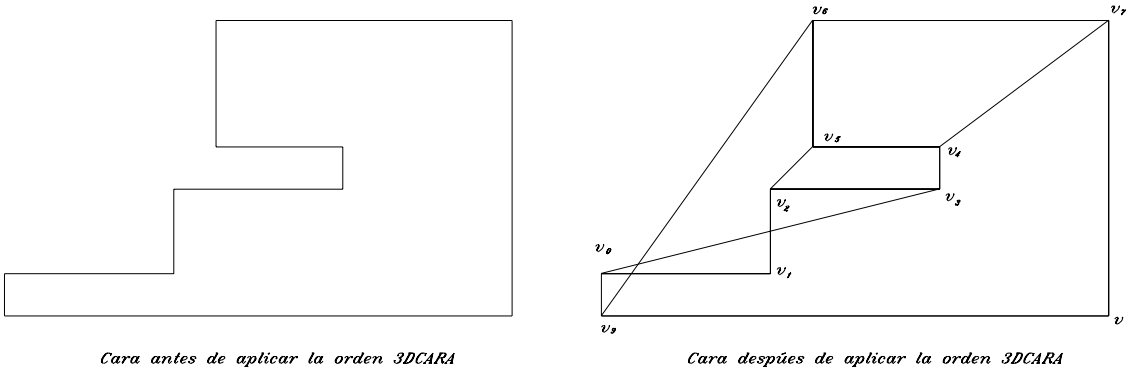


Figura 25: Aplicación de la orden 3DCARA

Seguidamente vamos a ver como actúa la orden 3DCARA y la forma de solucionar los problemas.

6.6.2 Orden 3DCARA.

La orden 3DCARA permite crear una cara tridimensional. El formato de la orden pide la introducción de cuatro puntos. Debe hacerse en **sentido circular** (horario o antihorario) y no cruzado como en la orden **SOLIDO**. Los últimos puntos introducidos pueden utilizarse como los dos primeros de una nueva cara 3D, con lo que bastará con introducir un tercer y cuarto puntos en el orden correcto para obtener una nueva cara unida a la anterior por una arista común. Esto se puede hacer sucesivamente cuantas veces se desee, hasta pulsar RETURN o barra espaciadora.

Su formato es: 3DCARA

Primer punto: (punto 3D)

Segundo punto: (punto 3D)

Tercer punto: (punto 3D)

Cuarto punto: (punto 3D, o RETURN para sección triangular)

Tercer punto: (punto 3D, o RETURN para terminar la orden)

Obsérvese que, para dibujar una cara con cuatro puntos, el orden de introducción de los puntos tiene que ser en sentido trigonométrico o inverso, si no se desea obtener una figura en forma de "mariposa". En este aspecto, la orden **3DCARA** funciona al revés de la orden **SOLIDO**.

Para hacer que un lado de una cara 3D resulte invisible, escribir "i" antes del primer punto de ese lado. Hay que indicar "Invisible" antes de especificar cualquier modo de referencia a objetos, filtros XYZ o coordenadas para ese lado.

El relleno sólido no se aplica a las caras 3D. La orden OCULTA considera las caras como opacas si son planares; en caso contrario, OCULTA dibuja una representación "alámbrica".

Como se puede observar por la definición de la propia orden 3DCARA, en caras como la representada en la Figura 25 se aplicará mal puesto que habrán punto que se encuentren en sentido horario y otros en sentido antihorario, por lo que para poder obtener las caras es necesario un estudio más profundo de las caras en 2D.

6.6.3 Caras cóncavas y convexas.

Una **cara** la consideraremos **convexa** si **todas** sus aristas son convexas, en caso contrario la consideraremos **cóncava**.

La cara representada en la Figura 27 es cóncava. En cambio la de la Figura 26 es convexa.

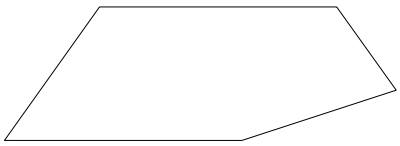


Figura 26: Cara convexa

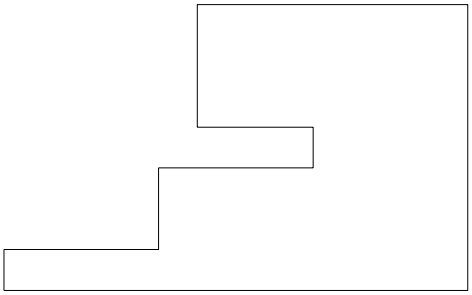


Figura 27: Cara cóncava

También podríamos considerar la cara **convexa** como aquella que al ser atravesada por una línea en cualquier ángulo, como máximo corta dos aristas de la cara siendo **cóncava** cuando se cortan más de dos aristas.

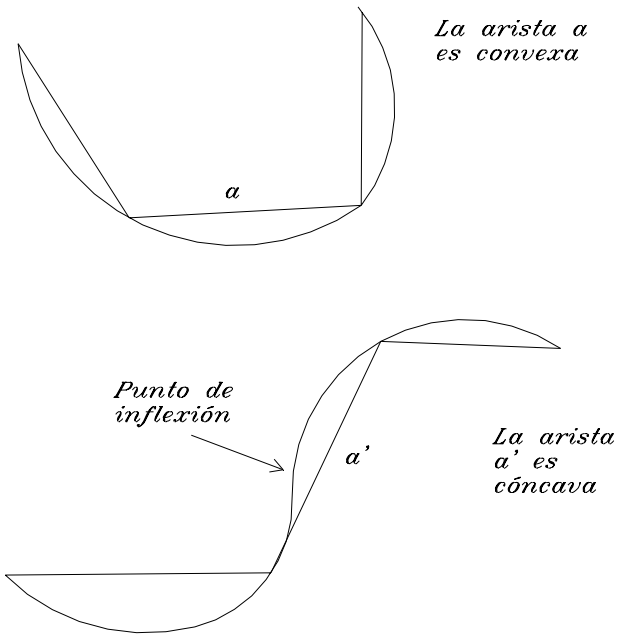


Figura 28: Tipos de aristas (1)

Una **arista** será **convexa** en una cara si tanto la arista anterior como la posterior están en el mismo lado en relación a ella. Es decir será convexa si las aristas contiguas quedan en el mismo semiplano respecto de la recta, teórica que la contiene. Figura 29 (b).

Una **arista** será **cóncava** en una cara si la arista anterior y la posterior quedan en lados diferentes en relación a ella. Es decir, una arista será cóncava si sus aristas contiguas quedan en semiplanos distintos respecto de la recta teórica que la contiene. Figura 29 (a).

En este caso, si trazásemos una curva que pasase por cada uno de los vértices comprobaríamos que en el arco que comprende la arista cóncava, dicha curva posee un punto de inflexión.

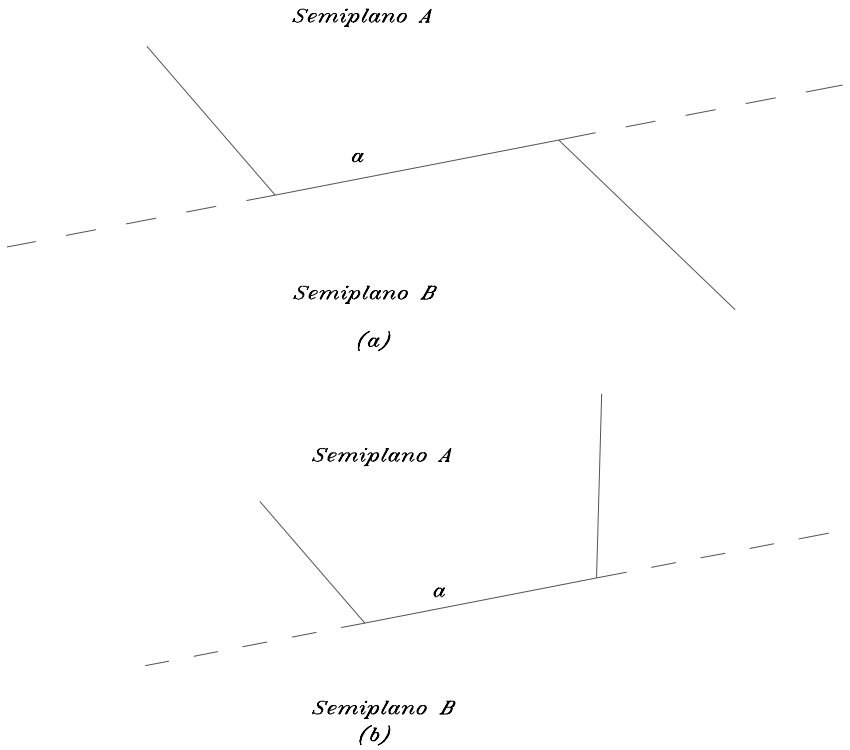
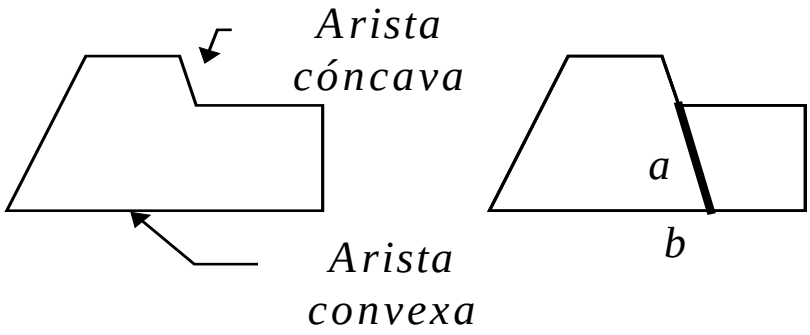


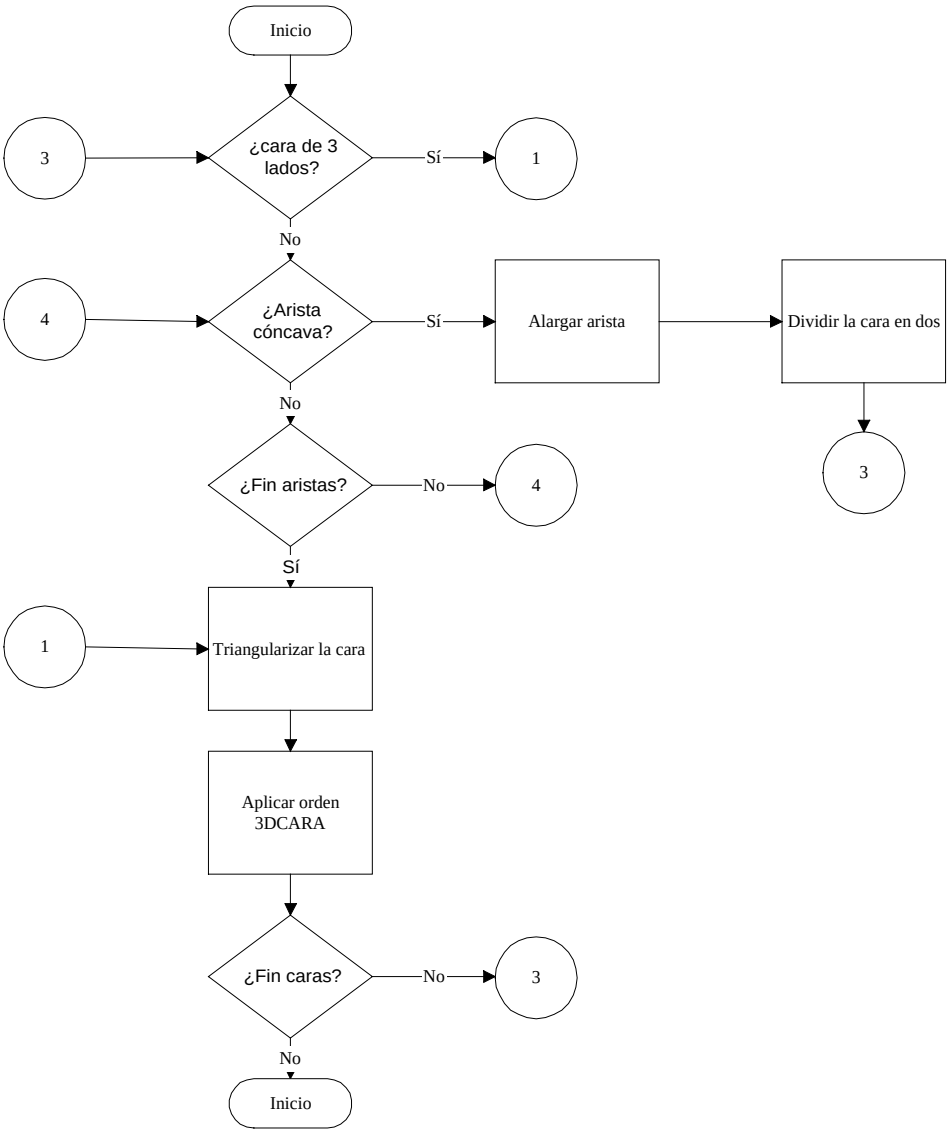
Figura 29: Tipos de aristas (2)

6.6.4 Algoritmo para la construcción de las caras en 3D.

El algoritmo para la construcción de las caras se estructura según el siguiente esquema:

- Si la cara tiene tres lados directamente se transforma con la orden 3DCARA.
- Si la cara tiene más de tres lados:
    - Ir recorriéndose todas las aristas de la cara para comprobar si son cóncavas o convexas.
    - Si todas las aristas son convexas entonces se transforma con la orden 3DCARA.
    - Si hemos encontrado una arista *a* cóncava:
        - Alargar (arista *a*) la arista hasta que intersecte con otra de la misma cara (arista *b*)
        - Dividir la arista *b* en dos, por el punto donde intersecta con la arista *a*.
        - Dividir la cara en dos, una a la izquierda de la arista *a* y otra a la derecha.
        - Repetir el proceso para estas dos caras nuevas teniendo en cuenta que la arista *a* será siempre convexa.





Esquema 6: Módulo para la construcción de las caras en 3D

6.6.4.1 Forma de calcular la concavidad, convexidad de una arista.

Para averiguar si una arista es cóncava o convexa lo haremos en 2D puesto que nos resultará más sencillo calcular las pendientes y hacer todas las operaciones necesarias.

Fácilmente, puede demostrarse que si una cara 3D posee una arista cóncava, al proyectar dicha cara sobre un plano, obtenemos un polígono en el que el lado proyección de la arista cóncava es también cóncavo. Ocurrirá otro tanto con cualquier arista convexa de una cara 3D, en cuyo caso, al proyectar sobre un plano, el lado proyección de dicha arista, es un lado convexo del polígono proyección de la cara.

Calcularemos la pendiente m_1 (ángulo de la arista actual). Si este ángulo es mayor de 180º entonces le restaremos 180 para así poder trabajar en el primer o segundo cuadrante.

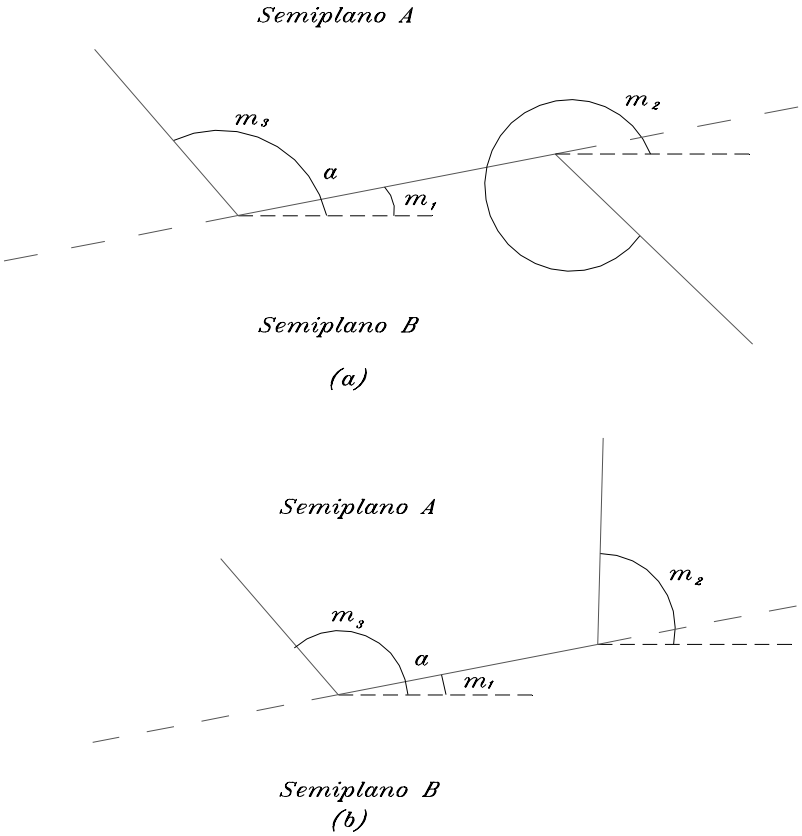


Figura 30: Cálculo de la pendiente

Calculamos las pendientes (los ángulos) de las aristas contiguas m_2 y m_3. Si estos están en el mismo semiplano (ver Figura 30) respecto de la arista actual entonces la arista será convexa sino será cóncava y procederemos a dividir la cara en dos semicaras separadas por una arista imaginaria desde el vértice que posee el punto de inflexión y un punto situado en otra arista a_3. La arista a_3 la dividiremos en dos $a_{3'}$ y $a_{3''}$ por un nuevo vértice v_p y pasaremos a estudiar la concavidad-convexidad de las dos nuevas caras.

El nuevo vértice v_p surge de calcular la intersección entre las dos aristas a_0 y a_3

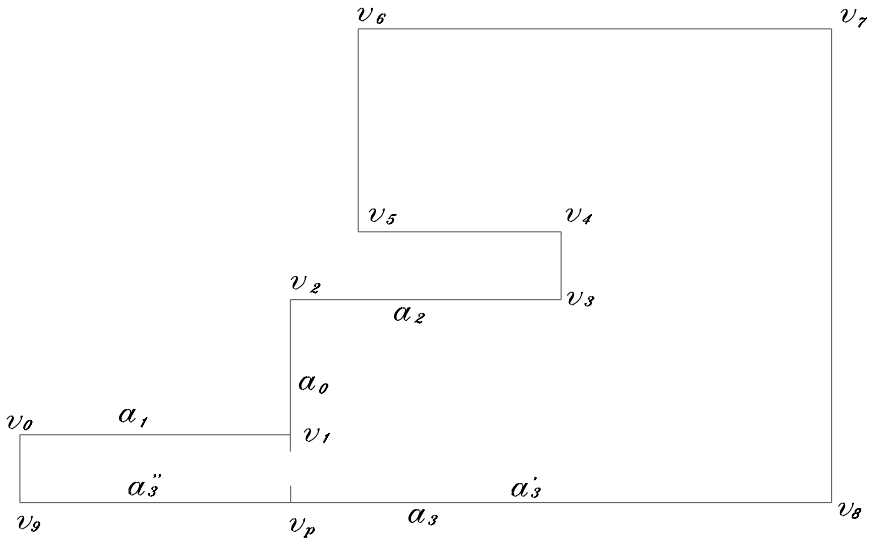


Figura 31: División de una cara cóncava

6.6.4.2 Caras convexas.

Una vez que se encuentra la cara convexa debemos convertirla por medio de la orden de AutoCAD 3DCARA. Para ello la dividiremos en tantas caras triangulares como sean necesarias y aplicaremos la orden 3DCARA a esas caras triangulares. Por tanto la cara inicial ha pasado a convertirse en un conjunto de caras triangulares. (Figura 33)

El algoritmo de triangulización de las caras convexas consiste en:

1. Escogemos un vértice v_0.
2. Escogemos una arista que incida en él. (a_0)
3. Escogemos la siguiente arista a a_0. (a_1)
4. Unimos el segundo extremo de a_1 (v_2) con v_0. Creamos la arista a_{p0}.
5. Escogemos la siguiente arista a a_1 (a_2)
6. Repetimos el paso 4 con la arista actual
7. Repetimos desde el paso 5 hasta que lleguemos de nuevo al vértice v_0. Para ser más exactos, a la otra arista que incide en v_0.

Ver Figura 32.

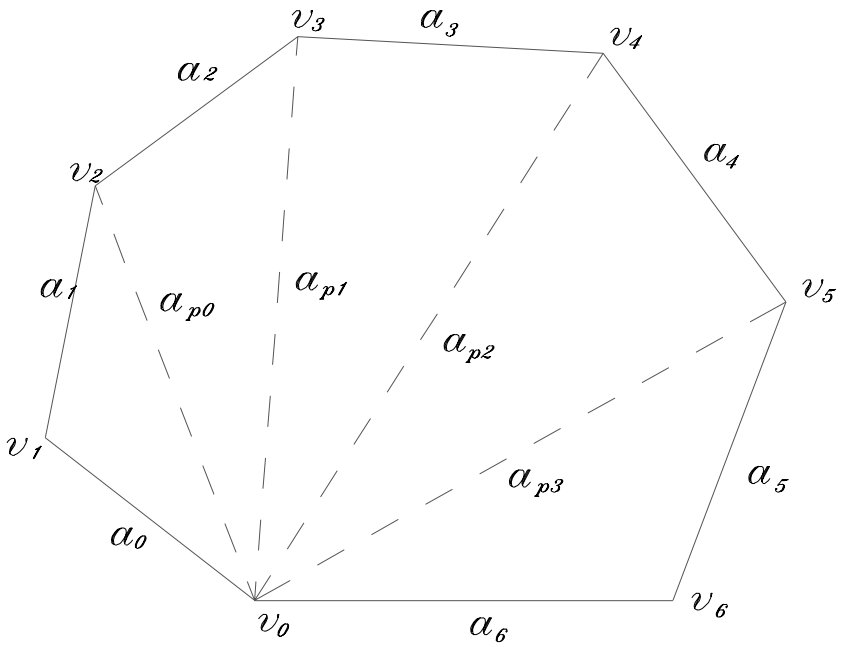


Figura 32: División de una cara convexa en caras triangulares

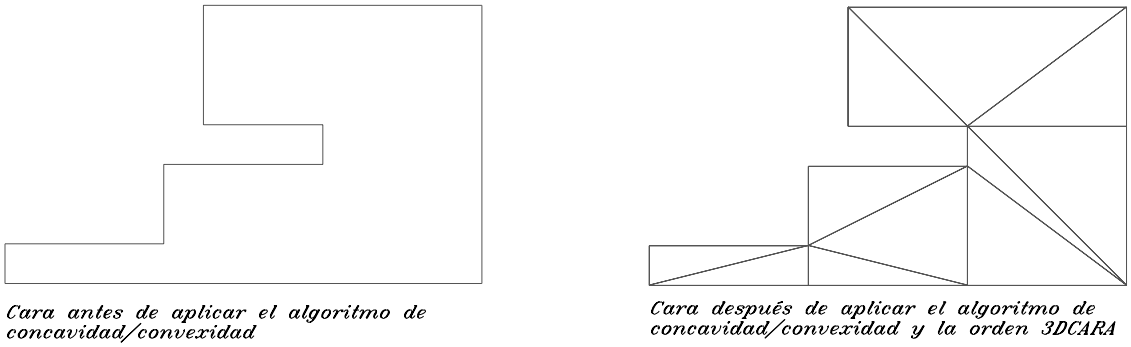


Figura 33: División de una cara cóncava en caras triangulares

6.7 Coste de los algoritmos.

6.7.1 Eficiencia de los algoritmos.

Una forma de determinar la eficiencia del tiempo de ejecución de un algoritmo es programarlo y medir el tiempo que lleva la versión en particular, en un computador específico, para un conjunto seleccionado de entradas. Aunque es popular y útil, este enfoque tiene algunos problemas inherentes. Los tiempos de ejecución dependen no sólo del algoritmo base, sino también del conjunto de instrucciones del ordenador, de la calidad del compilador y de la destreza del programador. Para superar esos inconvenientes, se han adoptado la complejidad de tiempo asintótica como una medida fundamental del rendimiento de un algoritmo. El término *eficiencia* se referirá a esta medida y, en especial, a la complejidad de tiempo en el peor caso.

La eficiencia o complejidad en el peor caso de un algoritmo es *O(f(n))* si *O(f(n))* es la función de *n* que da el máximo, en todas las entradas de longitud *n*, del número de pasos dados por el algoritmo en esas entradas. En otras palabras, existe una constante *c* tal que para una *n* suficientemente grande, *cf(n)* es una cota superior del número de pasos dados por el algoritmo con cualquier entrada de longitud *n*.

6.7.2 Medición del tiempo de ejecución de un programa.

El tiempo de ejecución de un programa depende de factores como:

- Los datos de entrada del programa,
- la calidad del código generado por el compilador utilizado para crear el programa objeto,
- la naturaleza y rapidez de las instrucciones de máquina empleadas en la ejecución del programa, y
- la complejidad de tiempo del algoritmo base del programa.

El hecho de que el tiempo de ejecución dependa de la entrada, indica que el tiempo de ejecución de un programa debe definirse como una función de la entrada. Con frecuencia, el tiempo de ejecución no depende de la entrada exacta, sino sólo de su "*tamaño*". Para muchos programas, el tiempo de ejecución es en realidad una función de la entrada específica, y no sólo del tamaño de ella.

6.7.3 Cálculo del tiempo de ejecución de un programa.

Calcular el tiempo de ejecución de un programa arbitrario, aunque sólo sea una aproximación a un factor constante, puede ser un problema matemático complejo. Sin embargo, en la práctica esto suele ser más sencillo; basta con aplicar unos cuantos principios básicos. Estos principios son:

- El tiempo de ejecución de cada proposición de asignación (lectura y escritura), por lo común puede tomarse como *O(1)*.
- El tiempo de ejecución de una secuencia de proposiciones se determina por la regla de la suma. Esto es, el tiempo de ejecución de una secuencia es, dentro

de un factor constante, el máximo tiempo de ejecución de una proposición de la secuencia.

- El tiempo de ejecución de una proposición condicional **if** es el coste de las proposiciones que se ejecutan condicionalmente, más el tiempo para evaluar la condición. El tiempo para evaluar la condición, por lo general es *O(1)*. El tiempo para la construcción **if-then-else** es la suma del tiempo requerido para evaluar la condición más el mayor entre los tiempos necesarios para ejecutar las proposiciones cuando la condición es verdadera y el tiempo de ejecución de las proposiciones cuando la condición es falsa.

- El tiempo para ejecutar un ciclo es la suma, sobre todas las iteraciones del ciclo, del tiempo de ejecución del cuerpo y del empleado para evaluar la condición de terminación (este último suele ser *O(1)*). A menudo este tiempo es el producto del número de iteraciones del ciclo y el mayor tiempo posible para una ejecución del cuerpo.

6.7.4 Análisis de programas recursivos.

El análisis de un programa recursivo suele implicar la solución de una ecuación de diferencias. Las técnicas para la solución de ecuaciones de diferencias algunas veces son sutiles, y tienen una considerable semejanza con los métodos de solución de ecuaciones diferenciales, alguna de cuya terminología se utiliza.

Existen tres enfoques distintos para resolver una ecuación de recurrencia.

1. Conjeturar una solución *f(n)* y usar la recurrencia para mostrar que *T(n)<=f(n)*. Algunas veces sólo se conjetura la forma de *f(n)*, dejando algunos parámetros sin especificar (por ejemplo, supóngase $f(n)=an^2$ para alguna a) y deduciendo valores adecuados para los parámetros al intentar demostrar que *T(n)<=f(n)* para toda *n*.

2. Utilizar la recurrencia misma para sustituir *m<n* por cualquier *T(m)* en la derecha, hasta que todos los términos *T(m)* para *m>1* se hayan reemplazando por fórmulas que impliquen sólo *T(1)*. Como *T(1)* siempre es constante, se tiene una fórmula para *T(n)* en función de *n* y de algunas constantes. A esta fórmula se ha denominado “forma cerrada” para *T(n)*.

3. Emplear la solución general para ciertas ecuaciones de recurrencia de tipos comunes de esta sección o de otra.

6.7.5 Análisis de los algoritmos.

Como se indica en el capítulo anterior el proyecto a realizar consta de tres grandes fases relacionadas entre sí. Estas fases son:

1. Identificación de las caras en 2D, (Módulo ID)

2. Obtención del modelo alámbrico en 3D (Módulo OMA)

3. Construcción de las caras en 3D (Módulo .OMS).

Supondremos *m* el cardinal de la lista aristas, es decir, el máximo número de aristas; *n* el cardinal de la lista vértices, máximo número de vértices; *p* la valencia de los

vértices; *q* el máximo número de aristas que puede formar una cara, es decir, la profundidad del árbol de búsqueda; *r* el cardinal de la lista de posibles caras y *s* el cardinal de la lista caras.

Puesto que se debe de cumplir la Ley de Euler entonces tenemos que siempre $m>n$ y $m>s$.

Dados estos valores pasemos a analizar el coste o eficiencia de cada uno de los algoritmos utilizados para el desarrollo de estas fases.

6.7.5.1 Módulo ID.

En este módulo destaca por su importancia el algoritmo de búsqueda de ciclos (BC) que pasaremos a analizar en primer lugar pasando a analizar posteriormente el módulo ID. Dentro del algoritmo de búsqueda de bucles analizaremos la función recorrido puesto que es recursiva y la que mayor coste posee.

El algoritmo de búsqueda de bucles o ciclos es:

```
1   Función Crearcara (v0, maxlados)
2   Inicio
3       Lista_arista := NULL
4       Buscar_arista_incidente (v0, a0, Lista _arista)
5       Mientras a0 != NULL hacer //mientras existan aristas incidentes
6           Si a0 saturada entonces Buscar_arista_incidente (v0, a0, Lista_arista)
7           Sino
8               v1 := segundo_vértice (a0, v0)
9               Si v1 < v0 Buscar_arista_incidente (v0, a0, Lista_arista)
10              Sino
11                  Añadir_a_lista (a0, Lista_arista)
12                  Añadir_a_lista_ordenada (v1, Lista_vértices)
13                  Buscar_arista_incidente (v1, a1, Lista_arista)
14                  Mientras a1 != NULL hacer
15                      Si a1 saturada
16                      Entonces Buscar_arista_incidente (v1, a1, Lista_arista)
17                      Sino
18                          v1 := segundo_vértice (a1, v1)
19                          Añadir_a_lista (a1, Lista_arista)
20                          Añadir_a_lista_ordenada (v1, Lista_vértices)
21                          profundidad = 3
22                          terminado = 0
23                          Recorrido (v1, v, terminado, valencia v1, limprof)
24                      Fin Si
25                  Fin Mientras
26                  Tratarposiblecara(valencia v1, v)
27              Fin Si
28          Fin Si
29      Fin Mientras
30  Fin Función
```

Líneas	Número de pasos
3	O(1)
4, 6, 9, 13, 16	La función Buscar_arista_incidente tendrá un coste de O(m) puesto que se escogerá una arista incidente y se buscará en la lista de aristas.
8, 18	El coste de la función Segundo_vértice es O(1), puesto que solo escoge el segundo vértice de la arista que estamos analizando.
11, 19	La función Añadir_a_lista tendrá un coste de O(1) puesto que se añade al principio de la lista.
12, 20	La función Añadir_a_lista_ordenada tendrá un coste de como máximo O(n) dado que n es el máximo de vértices.
15	O(1)
21, 22	O(1)
23	Coste de la función recursiva Recorrido: O(p*r*q) como se verá mas adelante.
17-23	O(1)+ O(1)+ O(n)+ O(1)+ O(1)+O(p*r*q)= O(p*r*q)
15-24	Condición: máximo de O(1)+ O(m) ó O(1)+ O(p*r*q). En este caso es O(p*r*q) puesto que r>m
14-25	Bucle mientras: O(m*p*r*q) El bucle se repetirá tantas veces como aristas incidentes existan. En un caso hipotético se repetirá mientras existan aristas.
26	Coste de la función Tratarposiblecara, como máximo será del O(r) puesto que la función recorrerá la lista de posibles caras e irá mostrándolas de forma interactiva al usuario para que éste decida si es una verdadera cara o no.
10-27	O(1)+O(n)+O(m)+O(m*p*r*q)+ O(r)= O(m*p*r*q)
9-27	Condición: máximo de O(1)+ O(m) ó O(1)+ O(m*p*r*q). En este caso es O(m*p*r*q)
7-28	O(1)+ O(m*p*r*q) = O(m*p*r*q)
6-28	Condición: máximo de O(1)+ O(m) ó O(1)+ O(m*p*r*q). En este caso es O(m*p*r*q)
5-29	Bucle mientras: $O(m^2*p*r*q)$ El bucle se repetirá tantas veces como aristas incidentes existan. En un caso hipotético se repetirá mientras existan aristas.
Coste total	$O(m^2*p*r*q)$ Teniendo en cuenta que p y q se pueden considerar constantes comparándolos con m y r entonces podemos considerar que el coste de la

función Crearcara es $O(m^2*r)$

El coste de esta función va a depender del coste de la función recursiva Recorrido que pasamos a analizar seguidamente:

```
1   Función Recorrido(v1, v, terminado, valencia v1, limprof)
2   Inicio
3       Si (terminado == 1)
4       Entonces a1 = NULL;
5       Sino Buscar_arista_incidente (v1, a1, Lista_arista)
6       Mientras (a1 != NULL)
7           Si a1 saturada
8           Entonces Buscar_arista_incidente (v1, a1, Lista_arista)
9           Sino
10              vant = v1
11              v1 := segundo_vértice (a1, v1)
12              Si v1 != primervertice
13              Entonces Si Buscar_vértice(v1 Lista_vértices) == NULL)
14                  Entonces  (* El vértice no está *)
15                            Añadir_a_lista (a1, Lista_arista)
16                            Añadir_a_lista_ordenada (v1, Lista_vértices)
17                            recorrido(v1, v, terminado, valencia v1, limprof-1);
18                            Si (terminado == 1)
19                            Entonces a1 = NULL;
20                            Sino Buscar_arista_incidente (v1, a1, Lista_arista)
21                            Fin Si
22                  Sino    (* El vértice sí que está y no es el primero *)
23                          v1 = vant;
24                          Buscar_arista_incidente (v1, a1, Lista_arista)
25                  Fin Si
26              Sino (* El vértice coincide con el primero *)
27                  Añadir_a_lista (a1, Lista_arista)
28                  insertar posiblecara( Lista_aristas, Lista_vértices)
29                  Si ((Aristas_reales==3) || ((Aristas_reales==4) && (valencia==3)))
30                  Entonces  a1 = NULL
31                            terminado = 1
32                  Sino Buscar_arista_incidente (v1, a1, Lista_arista)
33                  Fin Si
34              Fin Si
35          Sino Buscar_arista_incidente (v1, a1, Lista_arista)
36          Fin Si
37      Fin Mientras
38      Si (a1==NULL)
39      Entonces  Eliminar_vértice(v1, Lista_vértices)
40                Eliminar_arista(a1, Lista_aristas)
41      Fin Si
42  Fin Función
```

El coste vendrá calculado de la siguiente forma:

Líneas	Número de pasos
3, 4, 7, 10, 12, 18, 19,23 29, 30, 31, 38	O(1)
5, 8, 20, 24,	La función Buscar_arista_incidente tendrá un coste de O(m) puesto que

32, 35	se escogerá una arista incidente y se buscará en la lista de aristas.
3-5	Condición Si; coste de la condición más el coste mayor de una de las proposiciones: O(1)+O(m)=O(m)
11	El coste de la función Segundo_vértice es O(1), puesto que solo escoge el segundo vértice de la arista que estamos analizando.
13	La función Buscar-vértice tiene un coste de O(n) puesto que hace un recorrido por la lista de vértices.
15, 27	La función Añadir_a_lista tendrá un coste de 1 puesto que se añade al principio de la lista.
16	La función Añadir_a_lista_ordenada tendrá un coste de como máximo O(n) dado que n es el máximo de vértices.
15-20	Coste de proposición entonces (El vértice no está). El coste será el máximo tiempo de una de las proposiciones que lo forman: O(1)+O(n)+O(m)=O(m) puesto que m es mayor que n
23-24	O(1)+O(m)=O(m)
13-25	O(n)+O(m)=O(m)
28	La función Insertar-posiblecara tiene un coste de O(r) puesto que r representa el cardinal de la lista de posibles caras.
29-33	O(1)+O(m)=O(m)
26-34	O(1)+O(r)+O(m)=O(r). Dado que posibles caras pueden llegar a haber “infinitas”.
12-34	O(1)+O(r) =O(r)
7-36	O(1)+O(r) =O(r)
6-37	Bucle mientras. El número de iteraciones va a depender de la valencia del vértice. Por lo general, ésta será de 3 ó 4 pero vamos a considerar que es de un valor indeterminado p que será la máxima valencia existente. Por lo tanto el coste del bucle será: p*O(r)=O(p*r)
39	La función Eliminar-vértice tiene un coste de como máximo O(n), puesto que como máximo se tendrá que recorrer toda la lista de vértices para poder eliminar el deseado.
40	La función Eliminar-arista tiene un coste de como máximo O(m), puesto que como máximo se tendrá que recorrer toda la lista de aristas para poder eliminar el deseado.
38-41	Máximo de O(n) y O(m). Como el número de aristas es mayor que el de vértices entonces será: O(1)+O(m)=O(m)

Coste de la función Recorrido en el caso de q=1	Puesto que el número de posibles caras puede ser infinitamente mayor que el número de aristas entonces, el coste de la función Recorrido será: O(m)+O(p*r)+O(m)=O(p*r)
Coste total	$O(q) = \begin{cases} p*r + O(q-1) & \text{si q > 1} \\ p*r & \text{si q = 1} \end{cases}$ por tanto el coste total será O(p*r*q)

Por último pasamos a analizar la función ID:

```
1   Función ID
2   Inicio
3       Inicialización
4       Crear_vértices_y_aristas
5       Calcular_Paralelismos
6       v0 = primervertice;
7       Mientras v0 != NULL
8           Crearcara (v0, 3)
9           v0 = siguiente vértice
10      Fin Mientras
11      v0 = primervertice;
12      Mientras v0 != NULL
13          Crearcara (v0, 4)
14          v0 = siguiente vértice
15      Fin Mientras
16      Si faltan caras
17      Entonces
18              Analizar_valencia_vértices
19              v0 = primervertice;
20              Mientras v0 != NULL
21                  Crearcara (v0, 4)
22                  v0 = siguiente vértice
23              Fin Mientras
24              Convertir a caras
25      Fin Si
26  Fin Algoritmo
```

Líneas	Número de pasos
3	Inicialización: coste constante. O(1)
4	Función Crear_vértices_y_aristas: irá “cogiendo” aristas del dibujo e insertándolas en la lista de aristas. A su vez se recorrerá la lista de vértices para insertar aquellos que todavía no lo están. Su coste es O(m*n)
5	Función Calcular_paralelismos: Va recorriéndose la lista de aristas y comparando dos a dos si son paralelas o no. Coste $O(m^2)$
6, 9, 11, 14, 19, 22	O(1)

19, 22	
8, 13, 21	$O(m^2*r)$
7-10, 12-15, 20-23	Bucle mientras. Recorre la lista de vértices y llama a la función crearcara para cada vértice. Su coste es $O(n*m^2*r)$
18	Función Analizar_valencia_vértices: recorre la lista de vértices y va comprobando su valencia. Su coste es O(n)
24	Función Convertir_a_caras: función que va recorriendose la lista de posibles caras y comprobando si son verdaderas caras o no. En el mejor de los casos tendrán un coste del cardinal de la lista de caras, es decir, el número de caras. En el peor de los casos su coste se elevará al cardinal de las posibles caras. O(r)
17-25	$O(n)+ O(1)+ O(n*m^2*r)+ O(r)= O(n*m^2*r)$
16-25	Bucle si: $O(1)+ O(n*m^2*r)= O(n*m^2*r)$
Coste total	$O(1)+O(m*n)+O(m^2)+O(1)+O(n*m^2*r)+O(1)+O(n*m^2*r)+O(n*m^2*r)=$ $=O(n*m^2*r)$

Este coste no es tan grande como parece puesto que en realidad el número de aristas y de vértices es pequeño, por lo que *n* y *m* son pequeños.

6.7.5.2 Módulo OMA.

Como se ha comentado en puntos anteriores, este módulo se encarga de obtener el modelo alámbrico 3D. Una descripción general del algoritmo será:

```
1   Función OMA
2   Inicio
3       Obtención_Ejes(Ejes)
4       Paralelismo_Arista_Ejes(Lista_aristas, Ejes)
5       Trans_Aristas_paralelas_Ejes(Lista_aristas, Ejes, Contador_vértices)
6       Trans_Aristas_Intersectan_con_Transformadas(Lista_aristas, Ejes, Contador_vértices)
7       Si Quedan_vértices_por_transformar
8       Entonces Transformar_resto_puntos
9       Fin Si
10  Fin Algoritmo
```

Líneas	Número de pasos
3	O(1)
4	La función Paralelismo_Arista_Ejes tendrá un coste de O(m) puesto que se recorrerá la lista de aristas y se irá comprobando si es paralela a alguno de los ejes. Recordemos que m indica el número máximo de aristas.
5, 6	El coste de estas funciones serán como máximo de O(m) puesto que como máximo se transformarán todas las aristas.

7	Esta condición solo comprueba si un contador es menor que n en cuyo caso se ejecutará la función Trasnformar_resto_puntos. El coste de esta comparación es constante e igual a O(1). Recordemos que n es el número máximo de vértices existentes.
8	Como máximo habrá que recorrerse todos los vértices por lo que el coste de esta función dependerá del número de vértices. Como habrá que recorrerse también la lista de aristas para poder hacer cálculos y así poder transformar estos vértices, entonces el coste también dependerá de la lista de aristas. Por tanto el coste de la función Transformar_resto_puntos será O(n*m) dado que n es el máximo de vértices y m el máximo de aristas.
7-9	Función si: O(1)+ O(n*m)=O(n*m)
Coste total	O(1)+ O(m)+ O(m)+ O(m)+O(n*m)=O(n*m)

6.7.5.3 Módulo OMS.

Para analizar este módulo, previamente vamos a estudiar el coste de la función recursiva Renderizar, pasando, posteriormente a analizar el propio módulo OMS.

```
1   Función renderizar (c, n)
2   Inicio
3       Si n<=0
4       Entonces 3dcara(c);
5       Sino
6           Si número_vértices_de_la_cara>3
7           Entonces
8               ea1= Primera_arista_de_la_cara;
9               Mientras Existan_aristas_en_la_cara
10                  Si Esconvexa(ea1, Lista_aristas_de_la_cara)
11                  Entonces
12                      Dividir_cara(ea1, Lista_aristas_de_la_cara, subcara1, subcara2);
13                      Si Existen(subcara1,subcara2)
14                      Entonces
15                          renderizar(subcara1, n-1);
16                          renderizar(subcara2, n-1);
17                          return;
18                      Sino Siguiente_arista
19                      Fin si
20                  Sino Siguiente_arista
21                  Fin si
22              Fin mientras
23          Fin si
24          Si No_Esconvexa y Número_vértices>3
25          Entonces
26              Primer_vértice_de_la_cara(m)
27              Buscar_siguiente_arista(ea1)
28              Si Arista_encontrada
29              Entonces
30                  Buscar_siguiente_arista(ea2);
```

```
31                  n=segundo_vértice (ea1, m)
32                  Mientras n!= Primer_vértice
33                      crearcarapara3dcara(m, n);
34                      3dcara(aux1c);
35                      m=n;
36                      Buscar_siguiente_arista(ea2);
37                      n=segundo_vértice (ea1, m)
38                  Fin Mientras
39              Sino
40                  Buscar_siguiente_vértice(m)
41              Fin si
42          Sino
43              Si Número_vértices=3
44              Entonces 3dcara(c);
45              Fin si
46          Fin si
47      Fin si
48  Fin algoritmo
```

Líneas	Número de pasos
3	O(1)
4, 34, 44	La función 3dcara tendrá un coste de O(m) puesto que recorrerá la lista de aristas pertenecientes a la cara. Consideramos el peor de los casos, que sería cuando sólo existiese una cara, caso que nunca va a ocurrir por lo que en verdad el coste va a ser menor.
6	O(1)
8	La función Primera_arista_de_la_cara tiene coste O(1) puesto que solo mira en el conjunto de aristas que forman la cara y escoge la primera.
9	La función Existan_aristas_en_la_cara es una simple comprobación de que no se ha llegado al final de la lista de aristas que forman dicha cara. Su coste es, por tanto O(1).
10	La función Esconvexa compara la arista actual con sus aristas colindantes para averiguar si es cóncava o convexa respecto a ellas. Su coste es O(1)
12	La función Dividir_cara divide la cara actual en dos a partir de la arista actual. Su coste va a depender del número de aristas que tiene la cara. En el peor de los casos este coste es O(m)
13	La función Existen es una simple comparación por lo que su coste es O(1)
14-17	Para el caso de la última iteración del algoritmo recursivo, no se entrará en estas líneas por lo que no las consideraremos. Para el resto de los casos el coste será el de la función recursiva.

18, 20	La función Siguiente_arista tendrá un coste de O(1), puesto que solo se desplaza a la siguiente arista de la lista de arsitas que forman la cara.
13-19	Para la última iteración del algoritmo recursivo el coste de la condición es O(1).
11-19	O(m)+O(1)= O(m)
10-21	En este caso el coste de la condición es O(m)
9-22	Bucle mientras: En el peor de los casos el bucle se repetirá m veces por lo que su coste será $O(m^2)$
7-22	$O(1)+ O(m^2)= O(m^2)$
6-23	Condición: $O(m^2)$
24	O(1)
26	La función Primer_vértice_de_la_cara se encarga de escoger el primer vértice existente en la lista de vértices pertenecientes a la cara. Su coste es O(1)
27, 30, 36	La función Buscar_siguiente_arista recorre la lista de aristas hasta encontrar la siguiente que pertenece a la cara actual. Tiene un coste de O(m)
28	La función Arista_encontrada comprueba que no hayamos recorrido ya todas las aristas. su coste es O(1)
31, 37	El coste de la función Segundo_vértice es O(1), puesto que solo escoge el segundo vértice de la arista que estamos analizando.
32	O(1)
33	La función Crearcarapara3dcara prepara la caraa actual para que se le pueda aplicar la orden 3dcara. Su coste va a depender del número de aristas de la cara: O(m)
35	O(1)
33-37	O(m)+O(1)+O(m)+O(1)= O(m)
32-38	Bucle mientras. Se repite el número de vértices que tiene la cara actual. En el peor de los casos n veces. Por lo tanto su coste es O(n*m)
29-38	O(m)+ O(1)+ O(1)+ O(n*m)= O(n*m)
40	La función Buscar_siguiente_vértice recorre la lista de vértices hasta encontrar el siguiente que pertenece a la cara actual. Tiene un coste de O(n)

28-41	Condición: su coste será O(n*m)
25-41	O(1)+ O(m)+ O(n*m)= O(n*m)
43	O(1)
43-45	Condición: su coste será O(m)
24-46	Condición: su coste será O(n*m)
5-46	$O(m^2)$+ O(n*m)= $O(m^2)$ puesto que el número de aristas será como poco igual al de vértices.
3-47	Condición: su coste será $O(m^2)$
Coste de la función Renderizar en el caso de n=1	$O(m^2)$
Coste total	$O(n) = \begin{cases} m^2 + O(n-1) & \text{si } n > 1 \\ m^2 & \text{si } n = 1 \end{cases}$ por tanto el coste total será $O(n*m^2)$

Veamos por último el coste de la función OMS

```
1   Función OMS
2   Inicio
3       c0=ppio_lista_caras
4       Mientras c0 != NULL Hacer
5           renderizar(c0, vértices c0);
6           c0=Siguiente_cara;
7       Fin Mientras
8   Fin Algoritmo
```

Líneas	Número de pasos
3, 6	O(1)
4	O(1)
5	Función Renderizar. $O(n*m^2)$
4-7	Bucle mientras. Este bucle se ejecutará mientras existan caras. El número de iteraciones es s. Por lo tanto su coste será $O(s*n*m^2)$
Coste total	O(1)+ $O(s*n*m^2)$= $O(s*n*m^2)$

6.7.5.4 Programa principal.

El coste general vendrá dado por el máximo coste de los distintos módulos:

Módulo	Número de pasos
ID	$O(n*m^2*r)$
OMA	$O(n*m)$
OMS	$O(s*n*m^2)$
Coste total	$O(n*m^2*r)$ puesto que *r* representa el cardinal de las posibles caras, en cambio *s* representa el cardinal de las caras. $r>s$.

Como se puede observar, el coste del algoritmo parece elevado, pero hay que tener presente que está preparado para figuras sencillas donde el número de aristas, vértices y caras no es muy elevado por lo que el coste no es tan grande.

7. EJEMPLOS.

A continuación vamos a ver algunos ejemplos de menor a mayor complejidad.

7.1 Ejemplo 1.

7.2 Ejemplo 2.

7.3 Ejemplo 3.

7.4 Ejemplo 4.

7.5 Ejemplo 5.

8. DESARROLLOS FUTUROS.

8.1 Desarrollos a corto plazo.

El objetivo inmediato es reducir la participación del usuario en el proceso de resolución. Esta reducción se debería concentrar en conseguir que el sistema detecte automáticamente las direcciones principales, la idea de utilizar los "gráficos de distribución angular" lanzada por Lipson y Shpitalni[17] parece prometedora.

Para evitar que el usuario tenga que decidir sobre "caras candidatas", parece interesante la vía de obtener las múltiples soluciones que cada problema pueda tener. De esta forma el usuario sólo tiene que hacer una elección final. Y, al mismo tiempo, puede visualizar alternativas que le hubieran quedado "enmascaradas" en las dos dimensiones de la perspectiva de partida.

Capacitar al método para reconstruir algunos poliedros no eulerianos, como son los típicos poliedros provistos de agujeros prismáticos.

Adaptar el método a distintos paquetes CAD de reciente aparición tales como MicroStation 95 y AutoCad 13.

Ampliar la aplicación para mostrar la información que se obtiene de vértices, aristas y caras.

[17] Weidong Wang, D.Sc., Georges G. Grinstein, Ph.D., "A Survey of 3D Solid Reconstruction from 2D Projection Line Drawings" *Computer Graphics Forum, Volume 12,* pp. 137-158, (1993).

8.2 Desarrollos a medio y largo plazo.

Capacitar al método para reconstruir algunos poliedros no eulerianos más complejos, como son conos, cilindros, esferas y sus derivados.

Incorporar la información dimensional contenida en la acotación, y la información de fabricación contenida en todo tipo de símbolos normalizados (tolerancias, acabados superficiales, procedimientos de fabricación, etc.)

Revisar la normalización de las representaciones gráficas actualmente en uso, para eliminar las ambigüedades, redundancias e indefiniciones, que un técnico entrenado es capaz de detectar y corregir, pero que resultaría mucho más complicado conseguir que un sistema de reconstrucción adquiriese dicho grado de "experiencia" y capacidad de decisión. Tal línea de trabajo redundaría en un simplificación de la tarea de reconstrucción, y, lo que es más importante, en una mejora del lenguaje gráfico normalizado. Se obtendría una "normalización de la normalización".

Conseguir una herramienta que permita al diseñador, una vez haya plasmado sus ideas en bocetos "virtuales" (trabajando directamente sobre el sistema CAD), pedir una "solidificación" de los mismos. Ello permitiría comprobar la "bondad geométrica" y/o otro tipo de requisitos funcionales del diseño (por ejemplo cualquier tipo de análisis resistente, térmico, etc.). De este modo, el sistema permitiría actuar sobre el boceto y/o sobre el modelo 3D, en refinamientos posteriores, hasta fijar el diseño definitivo.

Integrar todo el proceso de diseño en un mismo entorno, permitiendo así un tratamiento simultáneo de todos los aspectos del proceso iterativo del diseño. Y eliminando cualquier tipo de "exportación" de la información generada durante dicho proceso, ya que dicha información estaría totalmente integrada en la base de datos del sistema.

9. CONCLUSIONES.

Con este proyecto se ha conseguido implementar una herramienta de ayuda al dibujo. Se ha estudiado la problemática de la proyección axonométrica oblicua, la obtención del modelo alámbrico asociado a esa axonometría y la generación de las caras implícitas en ese objeto alámbrico. Para ello se ha utilizado un software CAD comercial, potente, sencillo de manejar y de amplia difusión.

Todos los aspectos se han cubierto sobradamente e incluso se han ampliado y mejorado alguno de ellos que no habían sido mencionados en la bibliografía existente. Prueba de ello son los ejemplos que se adjuntan al proyecto, en los que se puede advertir su complejidad.

Por otra parte han existido problemas añadidos al proceso descrito, no previstos en un principio, tales como el estudio de la concavidad-convexidad de las caras y el estudio de la valencia de los vértices, que han requerido un estudio en profundidad de la casuística derivada. Gracias a esta profundización se ha conseguido ***acotar*** los problemas resolubles con la técnica utilizada.

Tras la profundización en el paquete de diseño AutoCAD se han obtenido unas herramientas muy útiles, no sólo para nuestro objetivo, sino para cualquier diseñador que utilice las entidades de dibujo y quiera optimizar tanto el proceso de edición como el de impresión.

Se ha ampliado el objetivo inicial de obtención de la perspectiva axonométrica oblicua en 3D con la obtención de las caras, consiguiendo el modelo de superficies, y así poder aplicar las técnicas de eliminación de aristas ocultas, iluminaciones y sombreados.

Se ha descrito un algoritmo de reconstrucción 3D de modelos BRep para objetos poliédricos eulerianos, a partir de una sola perspectiva axonométrica. El algoritmo ha sido implementado y varios ejemplos han sido reconstruidos con éxito. Se ha comprobado que es tolerante a representaciones que no cumplen el supuesto de punto de vista general.

En comparación con otros métodos referenciados en la introducción histórica, en los que se utilizan algoritmos de búsqueda de caras capaces de trabajar con poliedros eulerianos pero con un mayor coste temporal y con la posibilidad de fallos y de aparición de soluciones múltiples, el algoritmo descrito supone un método alternativo cuya principal ventaja radica en la simplificación de la definición topológica del objeto.

El modelo de superficies obtenido mediante el algoritmo anterior, resuelve completamente los inconvenientes derivados de la presencia de caras cóncavas en los objetos poliédricos procesados.

10. BIBLIOGRAFÍA.

?? *"AutoCAD Development System™ . ADS Programmer✍ Reference"* Manual. Autodesk Development B.V., Neuchatel (Switzerland). 1992.
? ISBN: 2-88447-000-X

?? *"Manual de personalización de AutoCAD versión 12"*. Autodesk Development B.V., Neuchatel (Switzerland). 1992. ISBN: 2-88447-045-X

?? *"Manual del usuario de AutoCAD para windows"*. Autodesk Development B.V., Neuchatel (Switzerland). 1993. ISBN: 2-88447-169-3

?? **AHO**, Alfred V., **HOPCROFT**, John E., **ULLMAN**, Jeffrey D.: *"Estructuras de datos y algoritmos"*. Ed. Addison-Wesley Iberoamericana. México 1988.
? ISBN: 0-201-64024-4

?? **ALSINA**, C., **TRILLAS**, E. *"Lecciones de álgebra y geometría. Curso para estudiantes de arquitectura."* Ed. Gustavo Gili S.A. Barcelona 1984.
? ISBN: 84-252-1187-5.

?? **ATKINSON**, Lee, **ATKINSON**, Mark. *"Programaci✍n en Borland C++ 3.x"*. Ed Anaya Multimedia S.A., Madrid 1993. ISBN: 84-7614-472-5

?? **BESANT**, C.B., **LUI**, C.W.K. *"Computer-Aided Design and Manufacture"*. Ed. Ellid Horwood Limited, 1986. ISBN: 0-85312-952-5

?? **FERNÁNDEZ**, Fernando y otros: *"Poliedros"*. Ed. E.T.S.I. Caminos, Canales y Puertos de Madrid. Cátedra de Dibujo Técnico y Sistemas de Representación. Madrid 1988.

?? **FOLEY**, James D., **DAM**, Andries van :*"Fundamnentals of interactive computer graphics"*. Ed. Addison Wesley Iberoamericana. Delaware, E.U.A. 1982.

?? **FOLEY**, James D., **DAM**, Andries van, **FEINER**, Steven K., **HUGHES**, John F.: *"Computer graphics: principles and practice"*. Ed. Addison Wesley Iberoamericana. Delaware, E.U.A. 1990.

?? **FOLEY**, James D., **DAM**, Andries van, **FEINER**, Steven K., **HUGHES**, John F., **PHILLIPS**, Richard L.: *"Introducción a la graficación por computador"* Addison Wesley Iberoamericana. Delaware, E.U.A. 1996. ISBN: 0-201-62599-7

?? **GOMIS**, José M.: *"Dibujo técnico I"*. Ed. Servicio de publicaciones U.P.V. Valencia 1990. ISBN: 84-7721-117-5

?? **JORDÁN**, Cristina *"Apuntes Complementos de matemáticas"* Servicio Reprografía U.P.V. Valencia 1993.

?? **LEHMANN**, Charles H.: *"Geometría analítica"*. Ed. Limusa. México 1982. ISBN: 968-18-1176-3

?? **LÓPEZ**, Javier, **TAJADURA**, José A.: *"AutoCAD avanzado Versión 12"*. Ed. McGraw-Hill. Madrid 1993. ISBN: 84-481-0081-6

?

?? **MIRA**, José R., **GOMIS**, José M.: *"Ejercicios de dibujo técnico «Sistemas de representación»"*. Ed. Servicio de Publicaciones U.P.V. Valencia 1993.
? ISBN: 84-7721-204-X

?? **MORTENSON**, M.E. *"Geometric Modeling"*. Ed. John Wiley & Sons, 1985. ISBN: 0-471-88279-8

?? **PUIG**, Pedro: *"Curso de Geometría Métrica"*. Ed. Euler Editorial S.A. Tomo I Fundamentos. Madrid 1986. ISBN: 84-85731-05-0

?? **SANTOS,** Carlos: *"Generación de los polígonos de una figura espacial a partir de sus proyecciones ortográficas"*. Proyecto Final de Carrera. Facultad de Informática, U.P.V. Valencia 1994.

?? **SCHILDT**, Herbert. *"C: Manual de referencia."* Ed. Osborne/McGraw-Hill. Madrid 1992. ISBN: 84-7615-535-2

?? **SELLARÉS**, J.A.: *"Fundamentos de los gráficos por ordenador"*. Ed. EDUNSA. Barcelona 1988. ISBN: 84-7747-012-X

?? **THOMAS**, Robert M.: *"AutoCAD 12 para profesionales"*. Ed. McGraw-Hill. 1993.

?? **VEJARANO**, Javier. *"Generación automática de axonometrías oblicuas y modelos 3D a partir de vistas ortográficas de objetos"*. Proyecto Final de Carrera. Facultad de Informática, U.P.V. Valencia 1996.

?? **VIVÓ**, Roberto, **LLUCH**, Javier, **MOLLÁ**, Ramón. *"Fundamentos de informática gráfica 2D"*. Ed. Servicio de Publicaciones U.P.V. Valencia 1995.

?? **WAITE**, Mitchell, **PRATA**, Stephen, **MARTIN**, Donald. *"Programación en C: Introducción y conceptos avanzados"*. Ed. Anaya Multimedia S.A., Madrid 1990 ISBN: 84-7614-030-4

?? **WEIDONG** Wang, D.Sc., **GEORGES** G. **GRINSTEIN**, Ph.D., *"A Survey of 3D Solid Reconstruction from 2D Projection Line Drawings"*. Computer Graphics Forum, Volume 12. 1993.

?? **WESLEY**, M.A., **MARKOWSKY**, G. *"Generation of solid models from two-diniensional and three-dimensional data,"* Solid Modeling by, Computer: from Theory.to Application. 1986.

?? **YAN**, Qing-Wen, **CHEN**, C.L. Philip, **TANG**, Zesheng: *"Efficient algorithm for the reconstruction of 3D objects from ortographics projections"*. Computer-Aided Design Volume 26 Number 9 September 1994.