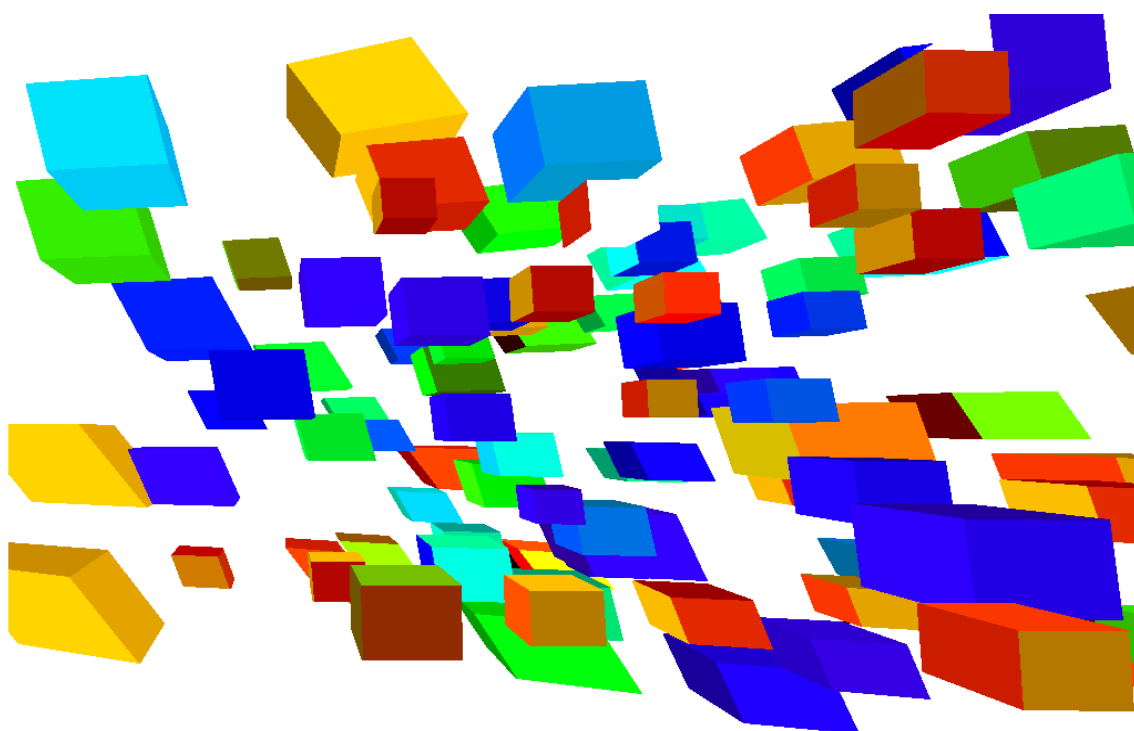

6

Computación Gráfica en Tres Dimensiones



6.1 Objetivos de la Computación Gráfica Tridimensional

Como mencionáramos en la Introducción, la Computación Gráfica tridimensional representó el primer cambio de paradigma en la disciplina. Se pasó del objetivo de la obtención de “gráficos” al objetivo de representar un “mundo virtual” como si fuese visto desde una ventana determinada por el monitor de la computadora. Por lo tanto, la idea esencial en este Capítulo es comenzar el estudio de las técnicas que permiten representar gráficamente las imágenes con un grado de realismo que permita la percepción o visualización de objetos tridimensionales.

La búsqueda del realismo tridimensional produjo durante la década del ‘70 el desarrollo de un conjunto de técnicas que permiten, gradualmente, una representación más adecuada de entidades gráficas en un mundo tridimensional virtual en el que ocurren efectos de iluminación, atmósfera, interacciones, etc. (consultar en [2] un panorama de dichas técnicas y las referencias bibliográficas a los trabajos originales). Podemos citar, por ejemplo:

Proyección perspectiva: Es una técnica geométrica que representa la tercera dimensión (profundidad) como un factor de escala que afecta a las otras dos dimensiones, en función de la posición de un observador o “cámara”. Se basa en un modelo simplificado del comportamiento geométrico de la luz en el ojo. Fue desarrollada en el Renacimiento por pintores como Durero, que buscaban salir del modelo artístico medieval.

Uso del color: También es una técnica desarrollada por pintores. Es otra de las maneras de representar profundidad, alterando el color de los objetos alejados en función de la misma (por ejemplo, haciendo que se vean más grises y oscuros).

Eliminación de las líneas y caras ocultas: Es una técnica indispensable, dado que las partes “no visibles” de las entidades gráficas molestan y confunden si son graficadas. Sin embargo, dada la enorme dificultad para encontrar algoritmos eficientes de cara oculta, los sistemas gráficos tridimensionales se valen usualmente de técnicas *ad hoc* y simplificaciones.

Modelos de iluminación y sombreado: Aquí la palabra “sombreado” se refiere al inglés *shading* que representa la alteración de los tonos cromáticos para dar idea de mayor o menor iluminación, y no a *shadowing* que representa la proyección de sombras de un objeto sobre otro. Los modelos de iluminación plantean la existencia de fuentes de luz ubicadas en el espacio, y de un modelo de reflexión por parte de los objetos. Esto determina que el color de los mismos varía en función de su posición y de la ubicación de la cámara.

Visión estereoscópica: Se producen dos perspectivas, levemente desplazadas, de modo que los objetos sean percibidos por cada ojo como si estuviesen “detrás” de la pantalla. En la mayoría de las personas ésto produce que el sistema visual reconstruya la percepción de una situación tridimensional (como también sucede con los estereogramas de punto aleatorio).

Modelos cinéticos y animación: Es la culminación del modelo paradigmático de la Computación Gráfica 3D con realismo. Se programan características cinéticas para los objetos de modo que se vean con la ilusión de un movimiento. Esto también ayuda a la interpretación y comprensión adecuada de la escena.

6.2 Transformaciones y Coordenadas Homogeneas

Generalizando lo visto en 2D, un punto en el espacio 3D homogéneo será un vector columna

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Las transformaciones en 3D son muy similares a las transformaciones en 2D. Esencialmente hay que agregar una fila y una columna a las matrices para representar la nueva coordenada de nuestras entidades. De esa manera tenemos que una matriz de escalamiento es

$$p' = E \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 & 0 \\ 0 & e_y & 0 & 0 \\ 0 & 0 & e_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Ahora debemos tener en cuenta que nuestro espacio puede rotar alrededor de tres ejes (la rotación en 2D es implícitamente rotar el espacio bidimensional del plano xy alrededor del eje z). La rotación un ángulo θ alrededor del eje x se representa como

$$p' = R_x \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

La rotación un ángulo θ alrededor del eje y es:

$$p' = R_y \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Por último, la rotación un ángulo θ alrededor del eje z es:

$$p' = R_z \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Al igual que en 2D, es posible representar la traslación $x' = t_x + x, y' = t_y + y, z' = t_z + z$ con una matriz de transformación:

$$p' = T \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Toda otra transformación afín puede representarse en términos de una composición de estas transformaciones básicas. Por ejemplo la transformación T que rota por θ alrededor de un eje paralelo al eje x se representa por una matriz que se obtiene con los siguientes pasos (ver Figura 6.1):

1. Se trasladan los objetos con una matriz de traslación T_t de modo que el eje de rotación sea el eje $x = 0$.
2. Se rota un ángulo θ con respecto al nuevo eje x con una matriz T_r .
3. Por último se trasladan los objetos a su posición original con $T_t^{-1} = T_{-t}$.
4. La transformación buscada es $T = T_{-t} \cdot T_r \cdot T_t$.

Si debe hacerse una rotación alrededor por un ángulo α alrededor de un eje arbitrario, entonces la secuencia es un poco más compleja:

1. Se trasladan los objetos con una matriz de traslación T_t de modo que el eje de rotación pase por el nuevo origen de coordenadas.

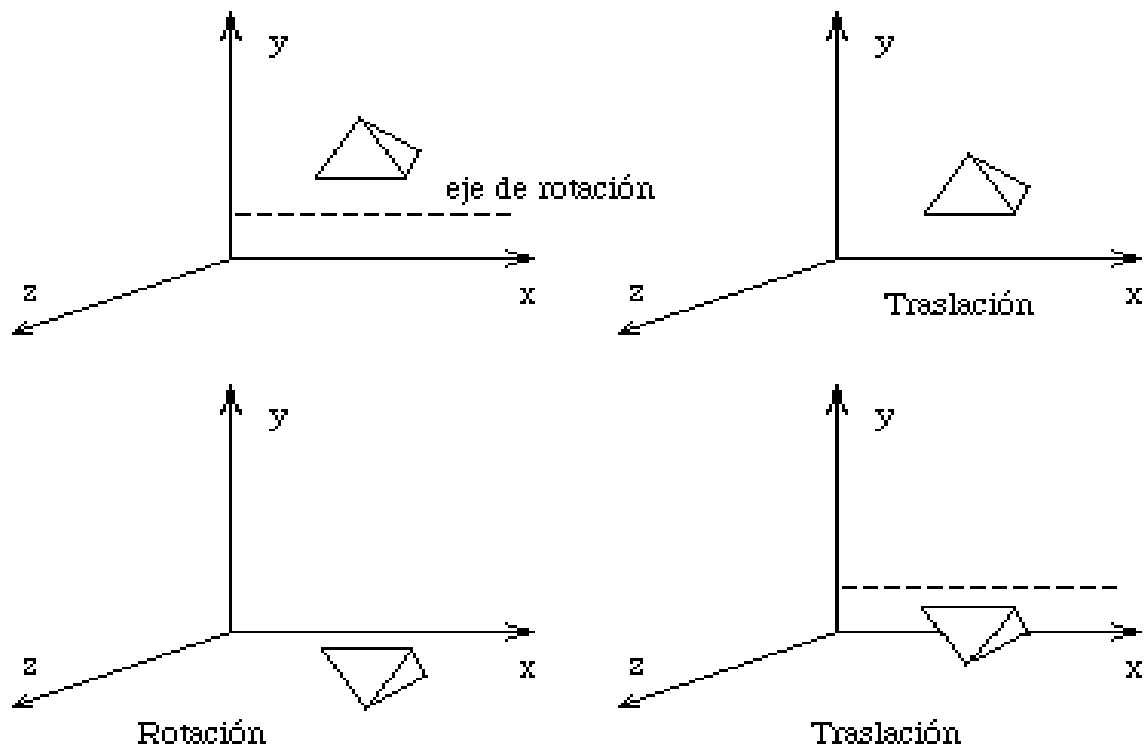


Figura 6.1 Rotación alrededor de un eje paralelo al eje x .

2. Se rotan los objetos un ángulo θ_x con respecto al nuevo eje x con una matriz T_{rx} de modo que el eje de rotación pertenezca al nuevo plano xz .
3. Se rota un ángulo θ_y con respecto al nuevo eje y con una matriz T_{ry} de modo que el eje de rotación se transforme al eje z .
4. Rotar un ángulo α alrededor del nuevo eje z (que, como vimos, coincide con el eje de rotación) con una matriz T_{rz}
5. Se rota un ángulo θ_{-y} con respecto al nuevo eje y con una matriz $T_{ry}^{-1} = T_{-ry}$.
6. Se rota un ángulo θ_{-x} con respecto al nuevo eje x con una matriz $T_{rx}^{-1} = T_{-rx}$.
7. Por último se trasladan los objetos a su posición original con $T_t^{-1} = T_{-t}$.
8. La transformación buscada es $T = T_{-t} \cdot T_{-rx} \cdot T_{-ry} \cdot T_{rz} \cdot T_{ry} \cdot T_{rx} \cdot T_t$.

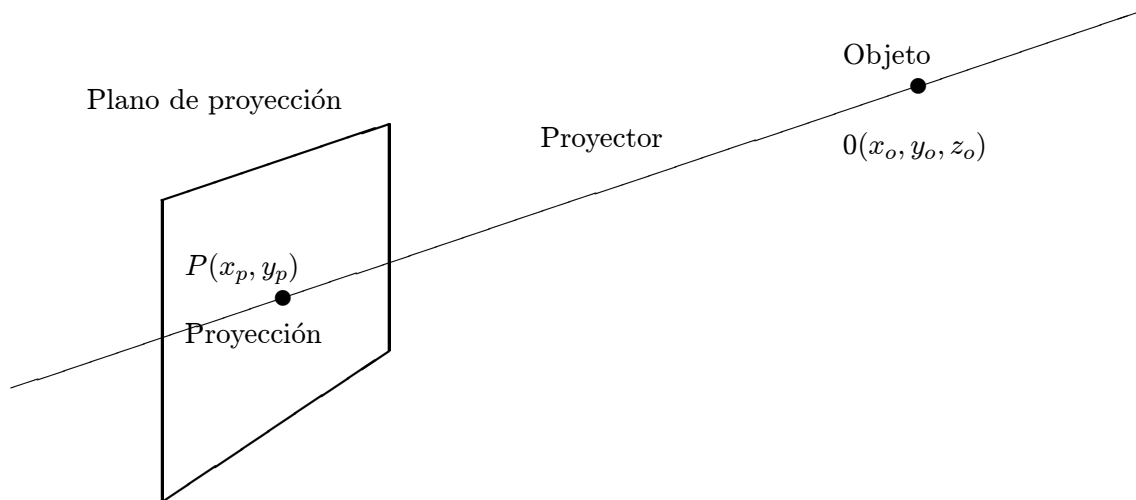


Figura 6.2 Esquema de una proyección geométrica planar.

6.3 Proyecciones y Perspectiva

Una *proyección* abstractamente es una operación en la cual se reduce la cantidad de dimensiones de una entidad. En Computación Gráfica se utilizan normalmente las *proyecciones geométricas* para representar en 2D los objetos cuyo modelo es de 3D. Una proyección geométrica consiste en hacer pasar *ejes proyectores* por el objeto a proyectar, encontrando su intersección con una entidad de proyección, normalmente una variedad diferenciable o superficie (ver Figura 6.2).

Si la proyección se realiza sobre un plano, entonces la proyección geométrica se denomina *planar*. Si además los ejes proyectores son paralelos entre sí, entonces la proyección se denomina *paralela* (ortogonal u oblicua). En cambio, si los proyectores convergen sobre un foco, la proyección se denomina *perspectiva* (ver Figura 6.3).

Las proyecciones ortográficas son muy utilizadas en el dibujo técnico, dado que representan la tercera dimensión conservando las distancias. Esto hace que los diagramas mantengan una escala uniforme en todas sus dimensiones y que por lo tanto sean más fáciles de interpretar y utilizar (por ejemplo en planos). El sistema multivista consiste en hacer pasar ejes paralelos a los ejes del sistema de coordenadas entre las entidades a proyectar y planos paralelos a los planos del sistema de coordenadas.

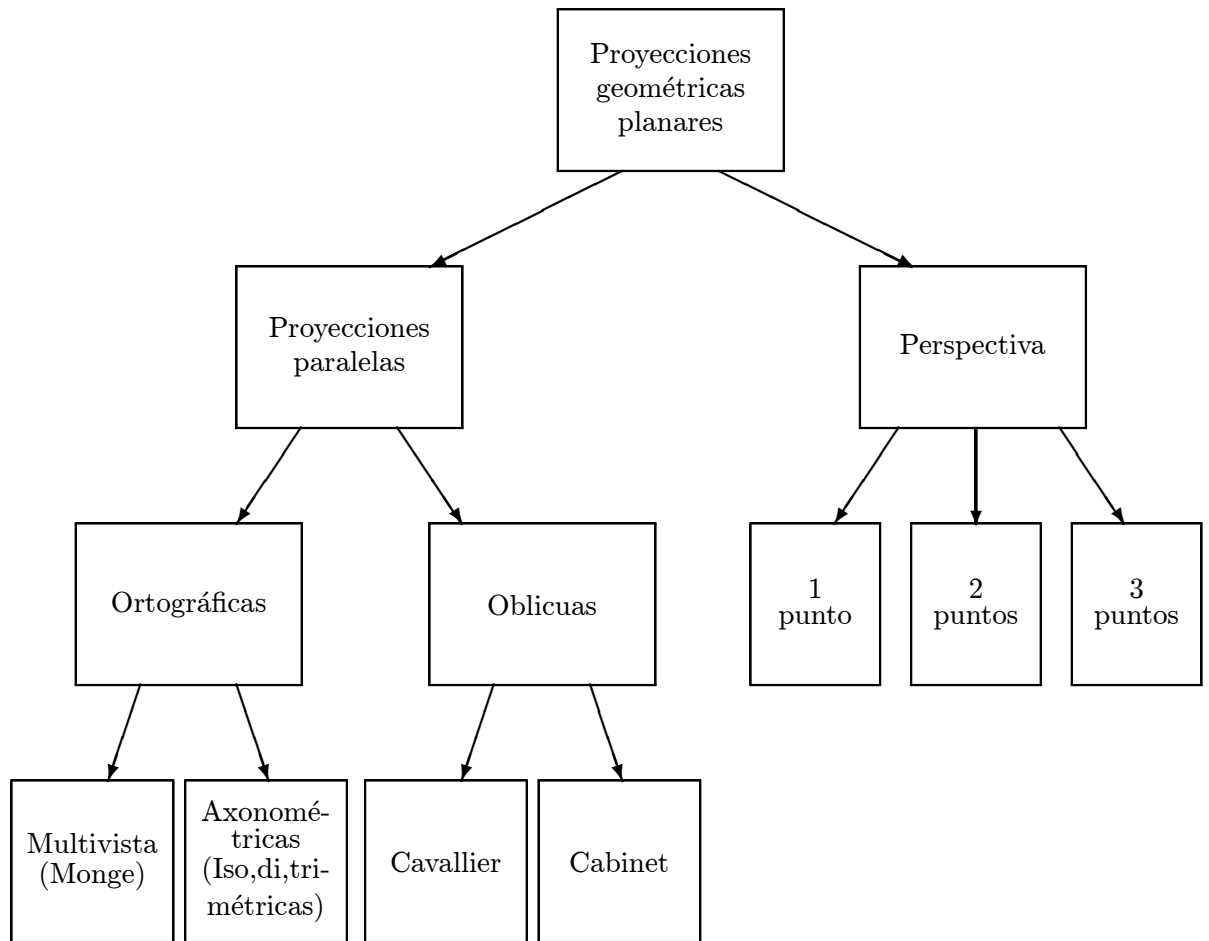


Figura 6.3 Distintas clases de proyecciones geométricas planares.

Como veremos, esto equivale a *ignorar* la coordenada respectiva. Por ejemplo, hacer pasar ejes paralelos al eje z por un plano paralelo al plano xy , conserva las componentes en x e y de las entidades proyectadas, pero descarta la componente z (ver Figura 6.4). Es decir,

$$x_p = x_o, \quad y_p = y_o,$$

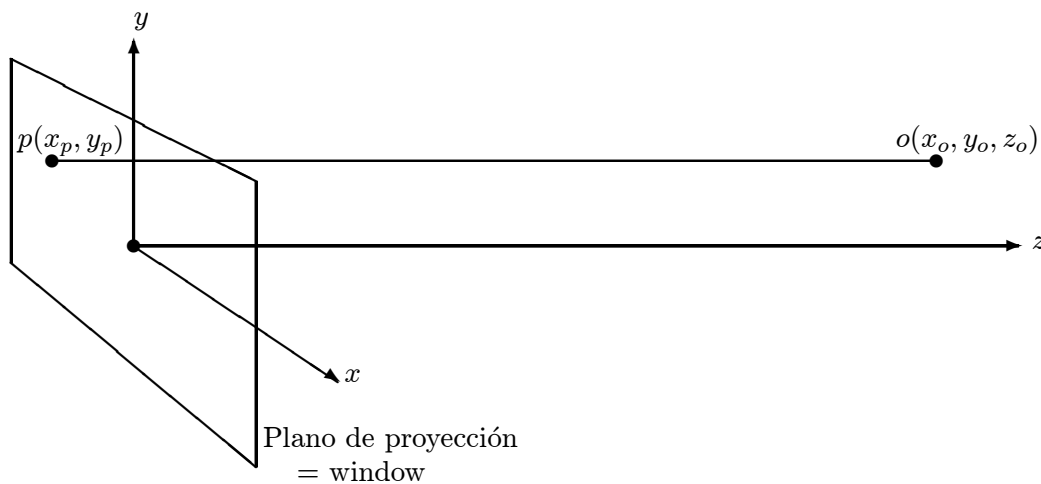


Figura 6.4 Proyección ortográfica.

y por lo tanto la operación de proyección se puede representar por medio de una matriz (denominada matriz de proyección):

$$p = p_r \cdot o, \text{ es decir, } \begin{bmatrix} x_p \\ y_p \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}.$$

Normalmente se adopta la convención de que la proyección se efectúa sobre el plano del window (probablemente en NCS), por lo que es necesario pasar de las coordenadas homogéneas a las coordenadas del window:

$$x = \frac{x_p}{h}, \quad y = \frac{y_p}{h},$$

y por último al viewport por medio de la transformación de windowing adecuada.

Las proyecciones axonométricas (isométrica, dimétrica y trimétrica), son similares a las ortográficas, pero posicionan el plano de proyección en forma alabeada a los planos del sistema de coordenadas. Como los ejes proyectores son perpendiculares al plano de proyección, entonces los ejes dejan de ser paralelos a los ejes del sistema de coordenadas.

De esa manera, ubicando un plano de proyección en un lugar adecuado, es posible obtener en una única proyección toda la información necesaria, algo que con las proyecciones ortográficas es difícil y requiere varias vistas. Las proyecciones oblicuas, en cambio, hacen pasar ejes proyectores en forma oblicua por el plano de proyección, el cual sigue siendo paralelo a los planos del sistema de coordenadas. En ambos casos,

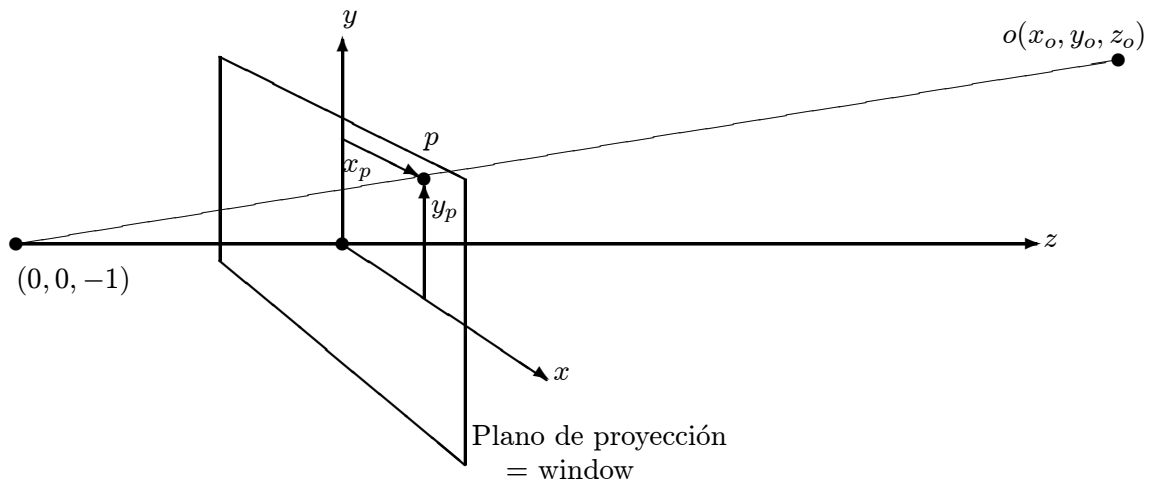


Figura 6.5 Elección particular del plano de proyección y del foco para la proyección perspectiva.

se respeta la uniformidad de distancias en los tres ejes, aunque cada uno de ellos tiene una escala diferente.

Para la Computación Gráfica, la proyección perspectiva es de mayor utilidad porque, si bien de una manera limitada, produce un efecto geométrico similar al efecto visual en el ojo humano. La perspectiva consiste en hacer pasar los ejes proyectores entre el objeto a proyectar y un punto distinguido, denominado *foco* en el dibujo técnico, y *observador*, *punto de vista* o *cámara* en Computación Gráfica.

En la literatura es común encontrar planteos muy diversos (e innecesariamente complejos) para describir la perspectiva en términos de una transformación. Esto es así dado que es posible ubicar el plano de proyección y el foco en diversos lugares del espacio del mundo. Nosotros aquí emplearemos una técnica que simplifica enormemente la matriz resultante y los cálculos involucrados. Suponiendo que el plano de proyección es el plano xy , y que el observador está en $(0, 0, -1)$ mirando hacia la dirección positiva del eje z (ver Figura 6.5), entonces es posible plantear la siguiente semejanza de triángulos:

$$x_p = \frac{x_o}{z_o + 1}.$$

Lo mismo se efectúa para la coordenada y :

$$y_p = \frac{y_o}{z_o + 1}.$$

Una forma de combinar la división en la perspectiva con el pasaje de coordenadas homogéneas a comunes es utilizar la siguiente matriz de perspectiva:

$$p = p_r \cdot o, \text{ es decir, } \begin{bmatrix} x_p \\ y_p \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix},$$

y luego se pasa de las coordenadas homogéneas a las coordenadas del window:

$$x = \frac{x_p}{h} = \frac{x_o}{z_o + 1}, \quad y = \frac{y_p}{h} = \frac{y_o}{z_o + 1}.$$

En el uso de la perspectiva es necesario realizar algunas consideraciones prácticas, dado que luego deben ejecutarse los algoritmos de cara oculta. Normalmente se realiza primero la transformación perspectiva sin proyección (es decir, sin eliminar la coordenada z) con una matriz

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Esta matriz puede premultiplicarse a todas las demás matrices de transformación, lográndose el pasaje del espacio del mundo (en el cual el área visible es un sólido con forma de pirámide truncada denominado “*frustum*”) a un espacio en el cual el área visible tiene forma cúbica (ver Figura 6.12).

Como veremos en la Sección 6.5, luego de la perspectiva, el *clipping* es más sencillo porque los planos de *clipping* del nuevo espacio son paralelos a los ejes. Es necesario efectuar *clipping* con el plano $z = 0$ a fin de eliminar las partes que quedan “por delante” del plano de proyección, lo cual produciría un efecto inadecuado.

Al retener la coordenada z de los puntos transformados, es posible utilizar esta información para los algoritmos de cara oculta. Por ejemplo, los algoritmos basados en prioridades utilizan el valor de dicha coordenada para ordenar las caras y de esa manera obtener el resultado deseado. Una vez efectuadas las manipulaciones necesarias, en el momento en el que se debe graficar una entidad, se efectúa una proyección paralela (es decir, se descarta la coordenada z).

La premultiplicación de todas las matrices involucradas en la tubería de procesos gráficos en 3D (ver Figura 6.13) tiene la siguiente forma:

$$\begin{bmatrix} e_x & r & r & t_x \\ r & e_y & r & t_y \\ r & r & e_z & t_z \\ p_x & p_y & p_z & e_h \end{bmatrix},$$

donde los factores e_x , e_y y e_z son factores de escala para las respectivas coordenadas, mientras que e_h es un factor de escala global que altera la coordenada homogénea. Todos los factores r producen rotación o inclinación de algún tipo. Los factores t_x , t_y y t_z producen traslación según las coordenadas respectivas, y los factores p_x , p_y y p_z , por último, producen una perspectiva (sin proyección) en la cual la escala es afectada inversamente por las coordenadas respectivas. Dependiendo de cuántos de estos factores sean no nulos, es el tipo de perspectiva (1 punto de fuga, o 2 puntos, o 3, ver Figura 6.3).

6.4 Primitivas y Estructuras Gráficas

La manera de estructurar las entidades gráficas en 3D sigue los mismos lineamientos generales que en 2D. Es posible realizar modelos de objetos relativamente complejos (mobiliarios, edificios, etc.) utilizando solamente instancias de cubos. La única diferencia, de gran importancia, es que es necesario observar una metodología más disciplinada para describir los objetos primitivos.

Como ya mencionáramos, la eliminación de líneas y caras ocultas es indispensable para lograr el realismo, y según veremos al final de este capítulo, los algoritmos que realizan estas tareas requieren una rápida y correcta identificación de determinadas características geométricas de las entidades gráficas. Lo mismo sucede con los modelos de iluminación y sombreado. De esa manera, en el momento de graficar una entidad, puede ser necesario ubicar el normal de sus caras, las aristas y vértices que concurren a las mismas, etc. Si todas estas determinaciones debieran realizarse por búsqueda, el tiempo de ejecución sería prohibitivo.

Por lo tanto, se recurre a describir las primitivas según un orden arbitrario pero estricto. Por ejemplo, dado el cubo de la Figura 6.6, numeramos sus vértices de modo que su ubicación en su propio espacio intrínseco esté relacionado con el índice del vértice. Luego buscamos que cada cara contenga la secuencia de vértices (ordenada por ejemplo en sentido antihorario si es vista desde afuera). Esto permite encontrar rápidamente la secuencia de aristas para dibujar la cara, y también permite encontrar el vector normal,

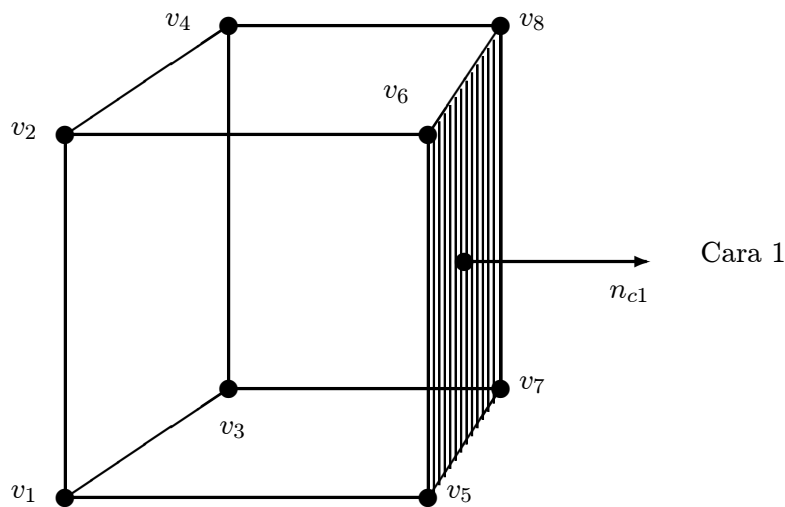


Figura 6.6 Descripción ordenada de un cubo.

dado que el mismo es el producto vectorial de dos aristas cualesquiera. Por ejemplo, el normal a la cara 1 es $(v7 - v5) \times (v8 - v7)$. Otra posibilidad es precalcular los normales y utilizar las matrices de transformación corriente para transformar los normales. Por último, puede ser de gran utilidad poder determinar cuáles son las caras que concurren a un vértice sin tener que recorrer la estructura.

En la Figura 6.7 observamos las declaraciones de tipos para una representación como la mencionada. Al igual que en el Capítulo 3, las entidades estructuradas se representan por medio de un registro variante, que puede apuntar tanto a un cubo o, recursivamente, a un (sub)objeto estructurado.

Dado un objeto estructurado en 3D, el algoritmo para graficarlo se basa en una recursión, cuyo caso base ocurre cuando el objeto es un cubo. En dicho caso, se pre-multiplica la matriz de instancia del cubo (dónde debe aparecer, con qué tamaño, rotación, etc.) por la matriz de transformación “corriente”, recibida por parámetro. Esta última transformación actúa como pila, según veremos. Con la matriz producto obtenida, se transforman los ocho vértices del cubo, y se grafican las doce aristas del mismo, siguiendo el orden correspondiente a su definición (por ejemplo, la mostrada en la Figura 6.6).

```

type
  cubo = array [1..8] of punto;
  transf = array [0..3] of array [0..3] of real;
  entidades = (cubs, defis);
  defi = array[0..9] of ^objeto;
  objeto = record
    t: ^transf;
    case tipo: entidades of
      defis: (d: ^defi);
      cubs: (c: ^cubo);
    end;

```

Figura 6.7 Declaraciones de tipos para representar objetos estructurados en 3D.

Si el objeto no es un cubo, entonces está estructurado a partir de subobjetos. En dicho caso, la parte recursiva del algoritmo aplica la matriz de instancia del objeto a cada subobjeto, y se llama recursivamente para cada subobjeto con la nueva matriz, la cual hace de matriz “corriente” de todos ellos. El procedimiento debe necesariamente terminar en el graficado de cubos (ver Figura 6.8).

Para inicializar una estructura como la mostrada, debemos primero inicializar el cubo, dándole coordenadas a cada uno de sus vértices. De acuerdo a la Figura 6.6 utilizamos la asignación

v1 = (-1, -1, -1, 1)	v2 = (-1, 1, -1, 1)
v3 = (-1, -1, 1, 1)	v4 = (-1, 1, 1, 1)
v5 = (1, -1, -1, 1)	v6 = (1, 1, -1, 1)
v7 = (1, -1, 1, 1)	v8 = (1, 1, 1, 1).

De esa manera, el algoritmo que grafica el cubo debe graficar las seis caras del mismo. Para graficar la cara delantera debe dibujar las aristas que van del vértice v1 al v2, del v2 al v6, del v6 al v5, y del vértice v5 al v1. También es necesario inicializar la matriz “corriente” con la transformación “del mundo”, la cual puede ser en principio la matriz identidad. Una vez establecidos estos datos, debemos comenzar a asociar a cada objeto con su matriz y definición correspondiente, la cual, como en 2D, es una lista de punteros a objetos, cada uno con su transformación y definición, y así recursivamente hasta llegar a los cubos. Una vez inicializada la estructura, se llama al algoritmo de

```

procedure graf_cub(o:objeto;at:transf);
var t,t1:transf;
    cu,c:cubo;
    i:integer;
begin
    t:=o.t^;
    cu:=o.c^;
    matprod(t,at,t1);                                {preproducto por matriz corriente}
    for i:=1 to 8 do begin
        c[i].x:=cu[i].x*t1[0][0]+cu[i].y*t1[0][1]+      {transforma un vertice}
            cu[i].z*t1[0][2]+cu[i].w*t1[0][3];
        ...                                            {idem con los otros 7}
    end;
    linea(c[1],c[2]);    linea(c[2],c[3]);            {grafica las 12 aristas}
    ...
end;

procedure graf_obj(o:objeto;at:transf);
var t,t1:transf;
    d:defi;
    i:integer;
begin
    case o.tipo of
        cubs : graf_cub(o,at);    {si es cubo lo grafica con matriz corriente}
        defis: begin
            t:=o.t^;                {si es estructura}
            d:=o.d^;
            i:=0;
            while d[i]<>nil do begin    {para cada subobjeto}
                matprod(t,at,t1);    {nueva matriz corriente}
                graf_obj(d[i]^,t1);  {llama recursivamente}
                i:=i+1;
            end;
        end;
    end;
end;
end;

```

Figura 6.8 Algoritmo base para graficar un cubo, y recursivo para graficar un objeto estructurado en 3D.

```

procedure graficar;
var
    cu : cubo;
    at,t1 ... t8 : transf;
    o1 ... o8 : objeto;
    mesa,escena : defi;
begin
    {matriz corriente = transformacion global}
    at[0][0]:=1;  at[0][1]:=0;  at[0][2]:=0;  at[0][3]:=0;
    ... {otros 12 coeficientes}

    cu[1].x:=-1; cu[1].y:=-1; cu[1].z:=-1; cu[1].w:=1; {primer vertice}
    ... {otros 7 vertices}

    t1[0][0]:=10;  t1[0][1]:=0;  t1[0][2]:=0;  t1[0][3]:=0;
    ...
    o1.t:=@t1;      o1.c:=@cu;

    {similar a lo visto en 2D}

    graf_obj(o8,at);      {llama al algoritmo recursivo}
end;

```

Figura 6.9 Inicialización de estructuras.

graficación pasándole la referencia al objeto “tope” de la estructura, y con la matriz “corriente” (ver Figura 6.9).

Podemos ver en la Figura 6.10 el resultado de definir una escena como objeto estructurado. Primero se definió la mesa como cinco instancias de cubos, cada uno con su transformación correspondiente. Luego se definió la escena como cuatro instancias de mesa, cada una con su transformación.

6.5 Clipping y Transformación de Viewing

El clipping en 3D es más mandatorio que en 2D, dado que debemos eliminar lo que está detrás del plano de proyección para que el modelo de “camara” tenga sentido. Al

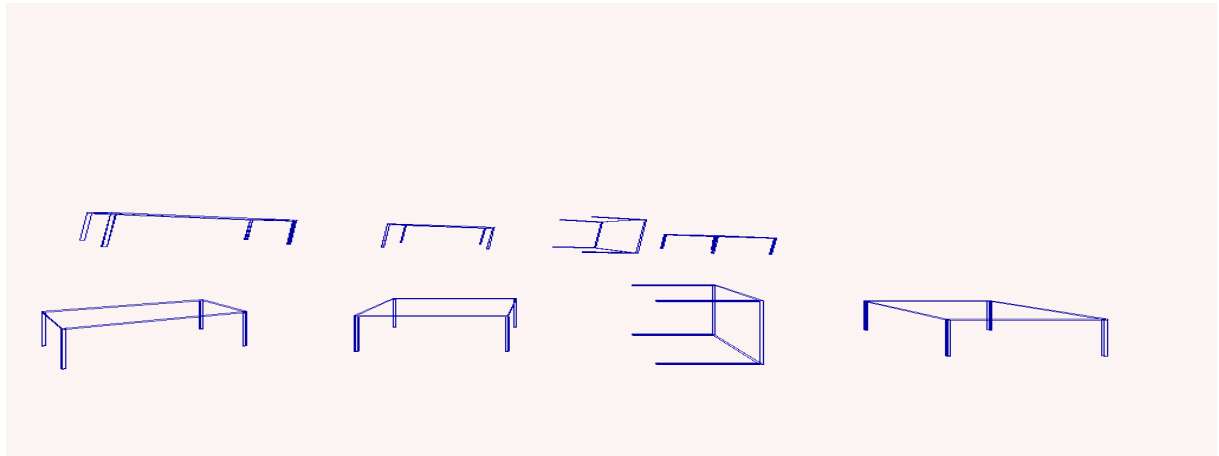


Figura 6.10 Una escena 3D.

mismo tiempo, suele agregarse un plano adicional con un z muy lejano, para no perder tiempo graficando objetos que no influyen en el resultado final de la escena. Como vimos, el primer paso antes del clipping es aplicar la transformación perspectiva (sin proyectar), para transformar el frustum o área visible (ver Figura 6.11) en una “caja” rectangular delimitada por planos paralelos y ortogonales.

Entonces se introduce un nuevo espacio “del frustum”, el cual está definido como el espacio rectangular que se obtiene luego de aplicar al mismo la transformación perspectiva (sin proyectar). La característica más importante de este nuevo espacio es que los planos de clipping dejan de ser oblicuos y pasan a ser paralelos (ver Figura 6.12).

Un algoritmo de clipping en dicha área procede de forma similar al Cohen-Sutherland en 2D, salvo que tenemos 27 áreas en el espacio, de las cuales solo una es visible, y las mismas se etiquetan con una séxtupla de booleanos. Las ventajas de proceder de esta manera son varias (pero es necesario recorrer algún camino para poder asimilar su importancia, por lo que recomendamos al lector que relea esta Sección luego de haber leído la Sección siguiente):

1. El pasaje de coordenadas homogéneas a comunes y la perspectiva se “enganchan” en una única operación que involucra solamente dos divisiones por cada punto procesado.

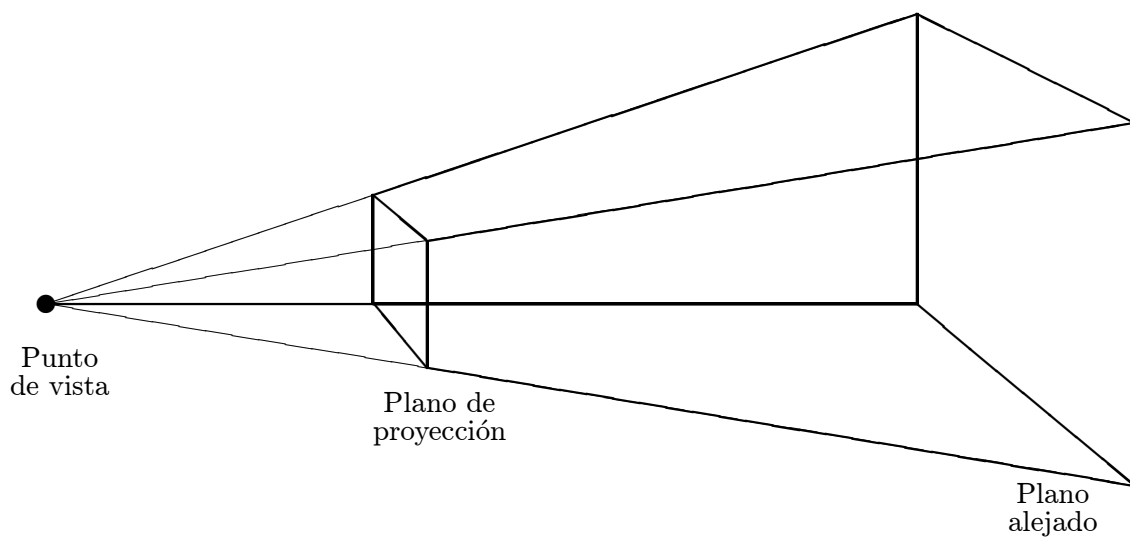


Figura 6.11 El "frustum" o área visible.

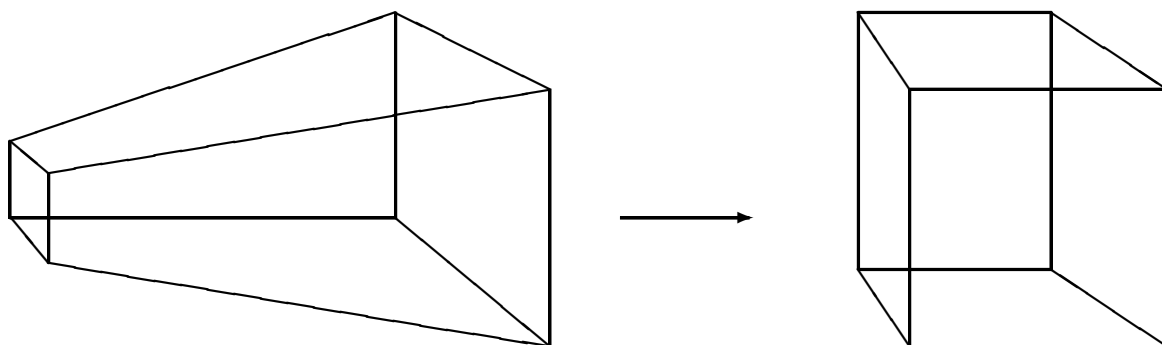


Figura 6.12 El espacio del "frustum", luego de la transformación perspectiva.

2. La matriz de transformación perspectiva puede premultiplicarse a todas las demás transformaciones corrientes en una única matriz de 4×4 que procesa todos los puntos del modelo de la escena de una manera eficiente.
3. Luego de la perspectiva, el clipping es mucho más sencillo porque solo hay que modificar levemente el algoritmo de Cohen-Sutherland ya implementado.
4. La eliminación de caras ocultas también es más sencilla porque si hay dos o más partes de la escena que “caen” en el mismo lugar de la pantalla, esto se puede descubrir porque tienen el mismo valor de x e y luego de la perspectiva.
5. Luego del clipping y de la cara oculta, la proyección es ahora una proyección paralela, la cual en definitiva es descartar la coordenada z .

Podemos en este punto intercalar una nueva transformación y un nuevo espacio, los cuales serán redundantes pero simplifican enormemente la descripción de determinados efectos, en particular al realizar animaciones. Cuando definimos la posición del observador o “camara” y su plano de proyección asociado, utilizamos un caso particular en el cual se simplificaban los aspectos matemáticos de la transformación perspectiva. Particularmente, se eligió en la Sección anterior que el observador esté en $(0, 0, -1)$, y que el plano de proyección sea el plano $z = 0$.

Sin embargo, un observador puede moverse en el espacio, y/o modificar la dirección hacia la cual está observando la escena. Por lo tanto, podemos considerar la existencia de un espacio del observador, en el cual el mismo está siempre en $(0, 0, -1)$ y el plano de proyección es el plano $z = 0$. Los cambios en la posición del observador se representarían con una transformación de viewing que modifica el espacio del mundo de modo tal que el punto $(0, 0, -1)$ del mundo pase a ser la posición del observador, y que el plano $z = 0$ pase a ser el plano de proyección.

Esto se podría representar *moviendo la escena*, por lo cual estrictamente no es necesaria una nueva transformación y un nuevo espacio. Sin embargo, mover la escena para acomodarse al movimiento del observador es artificial, y destruye la información de la *verdadera* posición de los objetos y el observador dentro de la escena. Por otra parte, modificar la escena y modificar la posición del observador son operaciones conceptualmente distintas y es un error incorporarlas en una única matriz. Y si tuviésemos *dos* transformaciones del mundo, es fácil ver que la segunda sería *la inversa* de la transformación de viewing (mover el observador a la derecha es equivalente a mover la escena hacia la izquierda, rotar su plano de proyección un ángulo α alrededor de un eje es equivalente a rotar la escena $-\alpha$ alrededor del mismo eje, etc).

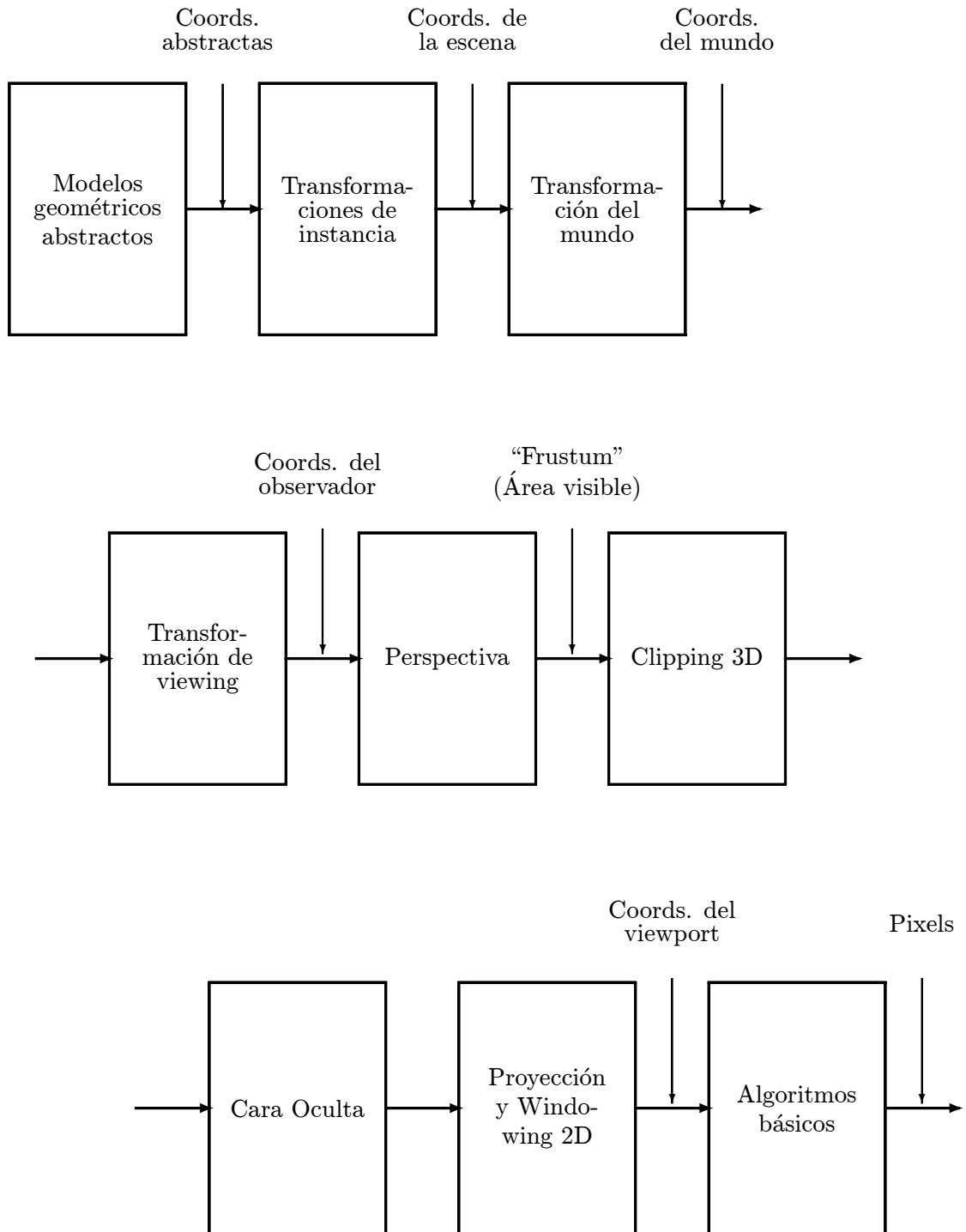


Figura 6.13 La "tubería" de procesos gráficos en 3D.

Podemos ver en la Figura 6.13 la integración de todos los procesos gráficos 3D en una única tubería. Los elementos restantes, correspondientes a los modelos de iluminación y sombreado que se verán en el Capítulo 7, se integran en la misma a la altura de los algoritmos básicos.

6.6 Cara Oculta

La eliminación de las partes de una escena que no deben verse es una tarea esencial en un sistema gráfico tridimensional con pretensiones de realismo. Pero por otra parte no existen soluciones generales que funcionen con costos tolerables. Por ello existe una gran diversidad de métodos y técnicas *ad hoc* que se utilizan solos o en combinación y para casos particulares determinados. Entre las técnicas relativamente sencillas podemos mencionar las siguientes:

Orientación de caras: Es una de las primeras en utilizarse, y suele acompañar a otros métodos para reducir la complejidad. Cada cara tiene una orientación que determina cuál es la parte “exterior” de un objeto. Esto queda determinado por la dirección del normal de la cara (recordar el cuidado que tuvimos para definir los normales del cubo en la Figura 6.6) Por lo tanto, si una cara es mirada “desde atrás” es porque no debe ser visible. Este test puede realizarse de manera sencilla y rápida, dado que es el resultado del signo del producto escalar entre el normal a la cara y un vector que va de un vértice de la misma al observador. Si el signo es positivo, el normal apunta “hacia el mismo lado” que el observador, y la cara es visible. Este test se denomina “backface culling” (filtrado de las caras posteriores). Estrictamente, el método es correcto solo si la escena está compuesta por un único objeto cóncavo y de caras planas, aunque en la práctica se aplica *antes* de utilizar otros métodos más complejos, para reducir el cómputo en un factor de dos o tres.

Algoritmo del pintor: Debe su nombre a que se basa en una técnica utilizada por los pintores (en este punto debemos reconocer que la Computación Gráfica le debe varias cosas a los pintores renacentistas). Por ejemplo, Leonardo no puede tener a Mona sentada frente a un paisaje durante el tiempo necesario. Entonces va hacia el paisaje y lo pinta con toda tranquilidad, y luego va a la casa de Mona y la retrata *encima* del lienzo pintado con el paisaje. En términos de lo estudiado aquí, lo que Leonardo ha hecho es ordenar las entidades por su coordenada z y luego pintarlas de atrás hacia adelante. Este método tiene varias desventajas. Primero, es de complejidad $O(n^2)$, donde n es el número de caras. Segundo, no siempre es posible decidir cuál cara está delante. Tercero, es muy ineficiente porque se pintan pixels varias veces.

z-buffer: Es un algoritmo de gran popularidad por ser muy sencillo e implementable en hardware. Consiste en tener un buffer donde se almacena la coordenada z de cada pixel. Si durante la graficación se accede a un pixel con una profundidad menor que la almacenada, entonces se pinta el pixel y se actualiza el buffer (ver Figura 6.14).

```

procedure zbuffer;
...
zbuffer:=infinito;
for cada-poligono do
  for cada-pixel do
    if pixel.z < zbuffer[pixel.x][pixel.y] then begin
      putpixel(pixel.x,pixel.y,poligono.color);
      zbuffer[pixel.x][pixel.y]:=pixel.z
    end;
  end;
end;

```

Figura 6.14 Esquema del algoritmo de *z-buffer*.

Las desventajas del método es que utiliza una gran cantidad de memoria (usualmente cuatro veces más que el buffer de pantalla) y que es muy ineficiente porque un pixel puede ser pintado varias veces.

Ray tracing: Para cada pixel se envía un “rayo” desde el observador hacia la escena. Si el rayo intersecta más de un objeto, se debe pintar con el color del objeto más cercano. Si no intersecta ningún objeto, entonces se pinta con un color “de fondo” (ver Figura 6.15). Estos algoritmos son sencillos y eficientes, y pueden aplicarse recursivamente para computar un modelo de iluminación híbrido pero de buenos resultados visuales, aunque las entidades gráficas representables deben ser de una geometría limitada [41]. Por dicha razón los consideraremos con cierto detalle en la Sección 7.5.

6.6.1 Eliminación de líneas ocultas en superficies funcionales

Este algoritmo fue desarrollado por Wright [86] y es muy importante no solo por su aplicabilidad, sino porque es el primero en proponer un orden *front to back* como solución al problema de las entidades ocultas. El método está pensado primariamente para graficar funciones de dos variables, pero puede utilizarse también para representar

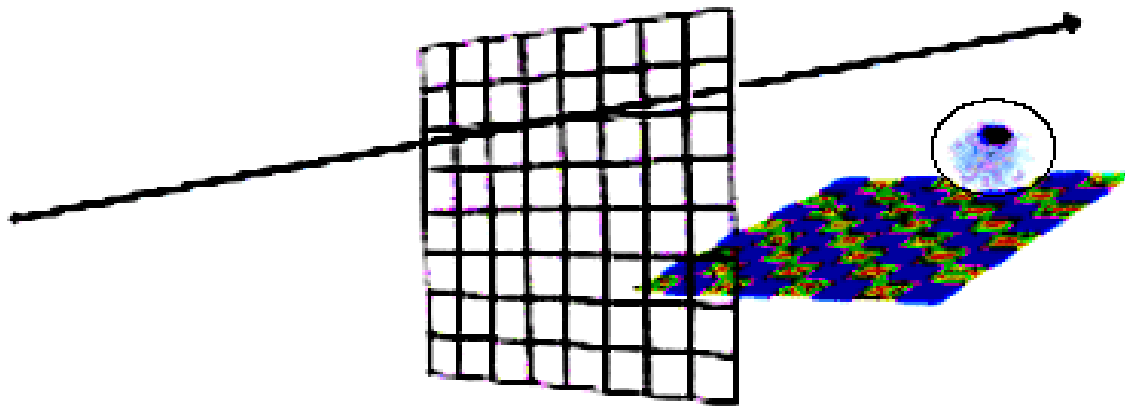


Figura 6.15 Ray tracing.

arreglos bidimensionales de valores, lo cual es de gran utilidad en Computación Gráfica. Cada fila de dicha matriz representa el valor de $y = f(x, z)$ a un valor constante de z .

Es posible determinar cuál de los cuatro vértices de la matriz es el más cercano al observador, y se recorren las filas en orden, comenzando por dicho vértice. Al tiempo que se grafican las filas, se actualizan dos arreglos donde se almacena, para cada x de pantalla, el máximo y mínimo valor de y . Los valores de estos arreglos determinan la “silueta” de lo ya dibujado, la cual es un área dentro de la cual no debe dibujarse (ver Figura 6.16).

Es muy importante observar que para que el algoritmo funcione correctamente, las filas deben dibujarse de adelante hacia atrás. La implementación de este algoritmo es sencilla, dado que solamente debe modificarse el algoritmo de discretización de segmentos de recta para incluir la condición de no invadir la silueta de lo ya dibujado. Para mejorar el aspecto de lo graficado, normalmente se dibujan también las líneas a x constante (es decir, la poligonal que une los valores de las columnas). Una estrategia posible para hacerlo es “marcar” los valores que fueron visibles en la recorrida por filas y luego unirlos. Esto, sin embargo, tiene algunos problemas, dado que entre dos valores visibles puede haber una porción no visible. Otra solución es ejecutar el algoritmo por filas y por columnas utilizando siluetas independientes.

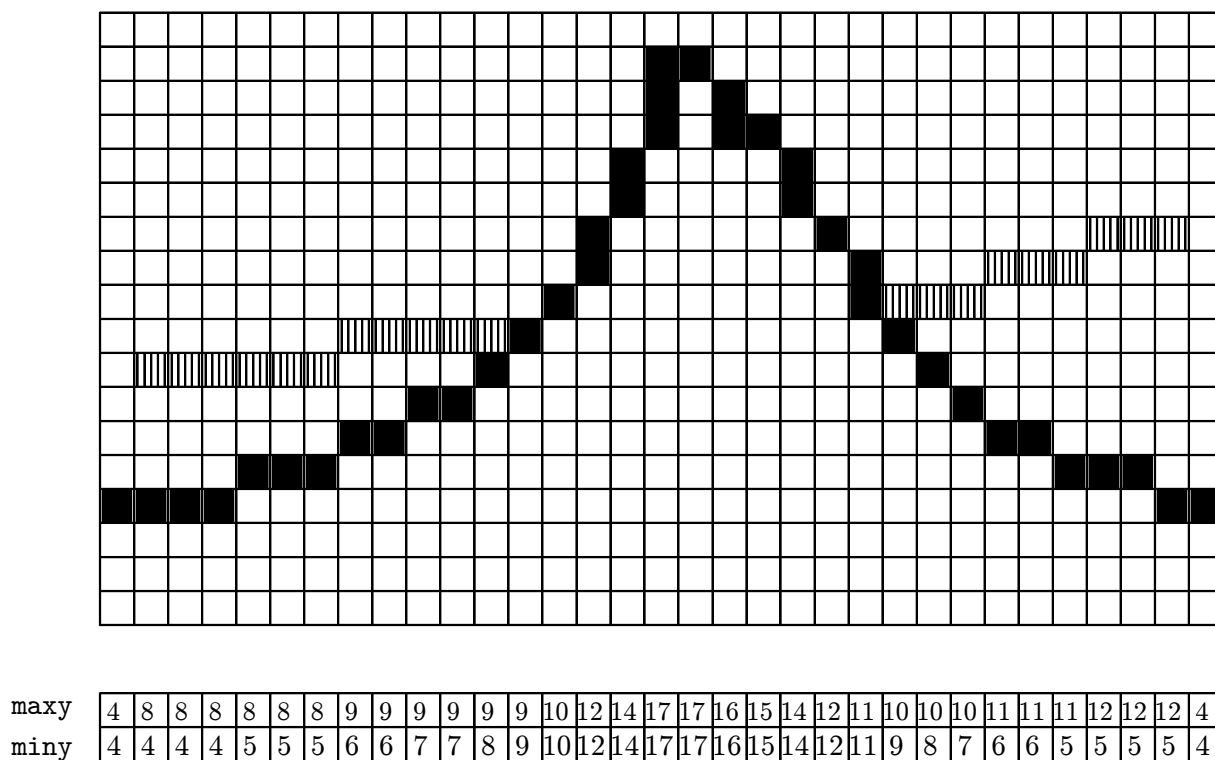


Figura 6.16 Estado intermedio durante el cómputo del algoritmo de Wright para superficies funcionales.

6.6.2 Clasificación de los métodos generales

Las consideraciones de espacio impiden referirnos aquí a los métodos importantes de cara oculta con la profundidad necesaria. Por lo tanto intentaremos dar una breve reseña de los mismos, clasificándolos por la naturaleza de los algoritmos utilizados. Los lectores interesados en profundizar los conceptos pueden consultar la bibliografía recomendada.

Métodos en 3D: Los denominamos de esta manera porque efectúan el cómputo en el espacio del mundo, es decir, en tres dimensiones y con precisión *real*. Por lo tanto, el resultado es independiente del dispositivo, y se pueden utilizar para dispositivos de vectores. Tienen, sin embargo, la desventaja del elevado costo computacional. Además, estrictamente computan líneas ocultas, no pudiendo pintarse las caras.

Roberts: Utiliza una representación biparamétrica de cada arista. Uno de los parámetros permite recorrer un punto de vértice a vértice, y el otro parámetro va de dicho punto al observador. Cada objeto de la escena se representa con una colección convexa de caras orientadas. De esa manera, es posible utilizar las técnicas de programación lineal para resolver si la formulación biparamétrica tiene una solución *dentro* de alguno de los objetos convexos o no. Si así sucede, entonces parte de la arista es no visible.

Appel: Representa cada cara como una colección orientada de aristas. Mientras se recorre cada arista, se verifica que la proyección de la misma no intersecte la proyección de otra arista. Si ocurre una intersección con una arista orientada positivamente y que está por delante, entonces se incrementa en uno un contador de “visibilidad cualitativa”. Si la misma está orientada negativamente, entonces se decrementa. La recorrida de la arista se realiza pintando mientras la visibilidad cualitativa sea cero.

Métodos en 2.5D: También denominados métodos de prioridades, dado que trabajan en el espacio del mundo en x e y , pero establecen una jerarquía cualitativa en z . Todo método en el que se ordenen los objetos por profundidad antes de graficarlos cae en esta categoría. Por lo tanto, tienen un costo $O(n^2)$.

Newell et. al.: Es un caso particular del algoritmo del pintor, en el cual se realiza un esquema de ordenación similar al bubble-sort o burbujeo. El algoritmo recorre la lista de caras comparándolas de a pares, llevando hacia abajo en la lista a la que aparezca detrás. Una vez que ubicó la más alejada del observador, la grafica e itera con las caras restantes. Esta comparación de la profundidad entre caras es bastante problemática. En algunos casos se puede determinar (si el z de todos los vértices de una está por delante o por detrás del z de todos los vértices de la otra), pero en general es bastante difícil, y en el caso del “solapamiento cíclico” de caras no hay una solución automática.

Schumacker: Irónicamente, fue el desarrollador de muchos métodos que actualmente se utilizan en los video juegos, simuladores de carreras, por ejemplo. La idea principal es que las escenas pueden descomponerse en “clusters” o racimos de caras, de modo tal que las caras de un cluster que está por delante de otro pueden dibujarse sin efectuar la comparación por profundidad. De esa manera se puede encontrar una asignación estática de prioridades a los objetos. Por ejemplo, en un simulador de carreras, el cielo y el paisaje de fondo están siempre detrás de la pista, la pista está siempre detrás de los demás automóviles, los cuales están siempre detrás del volante del conductor. También puede encontrarse una prioridad dinámica precomputada, en función de la posición del observador. Por último, es posible encontrar una asignación de prioridades entre las caras de cada cluster. Este tipo de métodos produce resultados aptos para la animación en tiempo real, pero la asignación de prioridades debe hacerse en forma estática y compilarse dentro del programa.

Wolfenstein: Comenzó a utilizarse en las implementaciones de video-juegos de la serie *Doom*, *Hexxen* y similares. Por lo tanto es muy reciente y no está debidamente documentado. Las escenas en estos juegos suelen ser polígonos con texturas, las cuales tienen un significado indispensable para el juego. El algoritmo consiste en trazar un rayo desde el observador a un punto dado de cada polígono, para determinar el punto de entrada al mapa de texturas. De esa manera, si bien se comete un error geométrico (más importante cuanto más próximo es el polígono al observador), es posible determinar rápidamente el orden de graficación, y acelerar el uso de mapas de texturas.

Métodos en 2D: Trabajan en el espacio de la imagen, recorriendo por línea de barrido. Por dicha razón se los denomina métodos scan-line. Son los más eficientes en tiempo, aunque la precisión depende del dispositivo gráfico de salida. Además trabajan bien en relación con los algoritmos de iluminación y sombreado. Podemos citar trabajos de varios autores (Wilye, Romney, Watkins), pero todos coinciden en procesar la escena por líneas a y constante. En cada línea es posible determinar las caras *activas*, es decir, aquellas que tienen por lo menos un pixel con dicha coordenada y . Para todas las caras activas se realiza la conversión scan (como la vista en la Sección 2.6 pero para solo una línea, denominada *span*). Durante dicha conversión es posible computar el z de cada uno de los pixel del comienzo y final del span (ver Figura 6.17). Por lo tanto, es posible determinar de una manera sencilla cuándo un span está por delante de otro, simplemente teniendo en cuenta que son segmentos de recta y que su intersección es fácil de calcular (ver Figura 6.18).

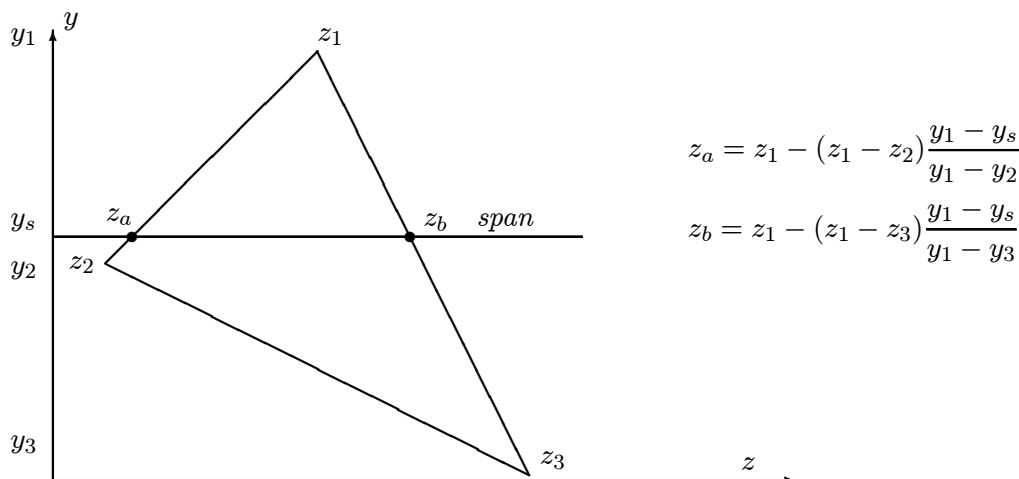


Figura 6.17 Determinación del valor de z de cada pixel en un span.

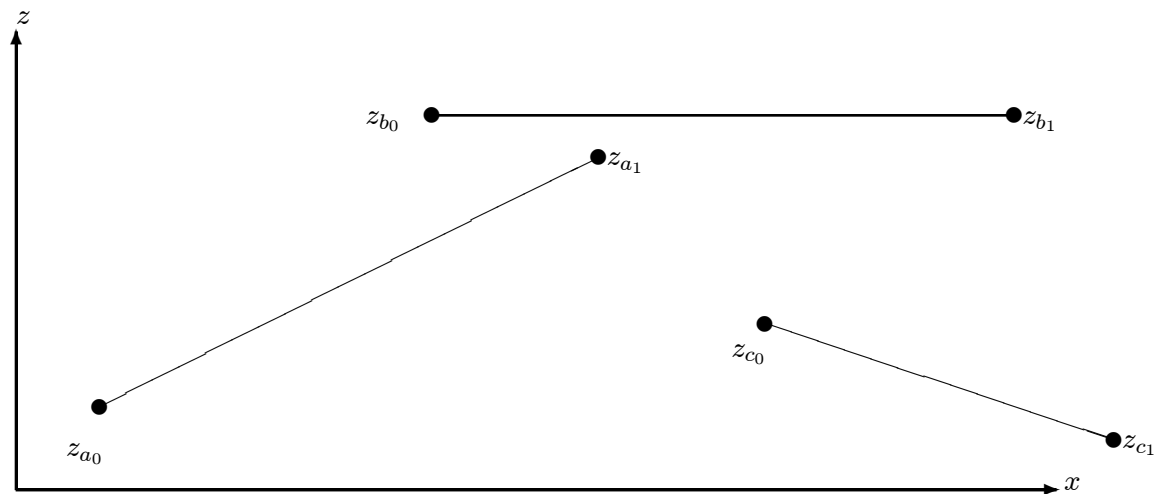


Figura 6.18 Decidiendo cuándo un span está delante de otro.

Métodos de subdivisión recursiva: La esencia de algunos métodos sencillos y eficientes como el clipping de Cohen-Sutherland está en poder descomponer recursivamente el problema en casos triviales. La idea de recursión se puede extrapolar a la solución del problema de la cara oculta. De esa manera, existen algoritmos que se basan en una subdivisión recursiva del espacio del mundo, o del espacio de la imagen. Entre los primeros se cuentan el algoritmo BSP (Binary Space Partition) que ordena un conjunto de caras por subdivisión binaria. Se toma una cara del conjunto, y las restantes pasan a estar orientadas positiva o negativamente. Cada uno de dichos subconjuntos es luego procesado recursivamente, de una manera similar al *quicksort*. Otra técnica de subdivisión recursiva del espacio del mundo es utilizar *octoárboles*, los cuales permiten subdividir recursivamente la escena hasta un nivel de resolución (“*voxels*”) manejable.

Entre los algoritmos de subdivisión recursiva del espacio de la imagen podemos mencionar al de Warnock, que interroga por el contenido del viewport. Si todo el viewport contiene un único objeto, o ningún objeto, entonces se lo pinta con el color del objeto o con el color de fondo, respectivamente. En caso contrario, el viewport se subdivide en cuatro, y a cada uno de los sub-viewports se lo procesa recursivamente con el mismo algoritmo.

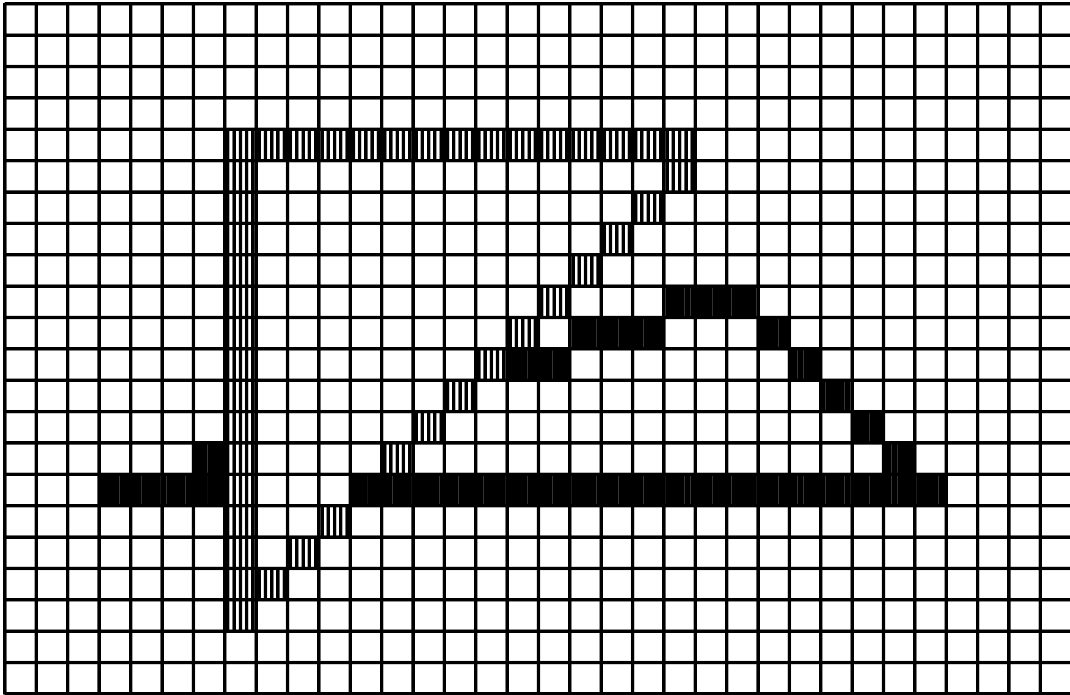


Figura 6.19 Resultado de graficar dos triángulos con el algoritmo SPIII de cara oculta.

6.6.3 El algoritmo SPIII

El algoritmo que describiremos ahora se basa en una combinación de las técnicas de prioridades, áreas de invisibilidad, y scan-line. Sin embargo, es de implementación relativamente sencilla (con respecto a los otros métodos), de ejecución eficiente, y posee pocos casos particulares en los que fracasa. Su descripción puede hacerse en dos líneas conceptuales:

1. Ordenar los polígonos por profundidad.
2. Dibujarlos de adelante hacia atrás, manteniendo en una estructura las áreas de invisibilidad.

Podemos ver en la Figura 6.19 el resultado de dibujar dos triángulos con este método.

Una técnica práctica para efectuar el ordenamiento entre caras es utilizar una variante del *radix-sort* [57] en función de las profundidades de las caras. Sea n la cantidad de caras a ordenar, entonces cada cara tiene un z mínimo, es decir, la profundidad del vértice más cercano al plano de proyección. Es posible dimensionar un arreglo de n lugares para distribuir las caras en el mismo. Cada lugar del arreglo está en correspondencia con la profundidad z del vértice más cercano de una cara según la ecuación

$$I = \frac{n(z - \min z)}{\max z - \min z},$$

donde $\min z$ y $\max z$ son las profundidades del vértice más cercano de la cara más cercana y de la más lejana, respectivamente (ver Figura 6.20).

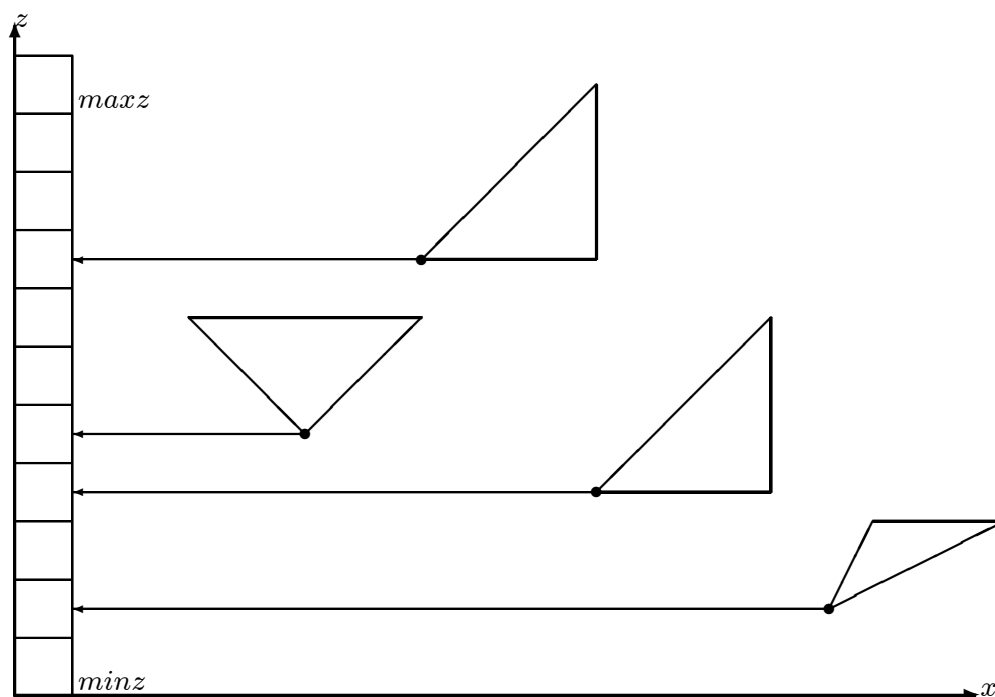


Figura 6.20 Ordenamiento de caras en el algoritmo SPIII.

Idealmente, si las caras están uniformemente distribuidas en profundidad, es posible que corresponda una sola cara a cada lugar del arreglo. En la práctica pueden existir lugares vacíos y lugares con varias caras. Lo importante es que las mismas pueden procesarse por orden, y en el caso de tener varias caras en una misma profundidad, de acuerdo al tipo de escena y de los objetos que se dibujan (superficies funcionales como las que veremos en el Capítulo 8, por ejemplo), normalmente no hay que ordenarlas entre sí. De todas maneras, en caso de tener que ordenarlas, se trataría siempre de

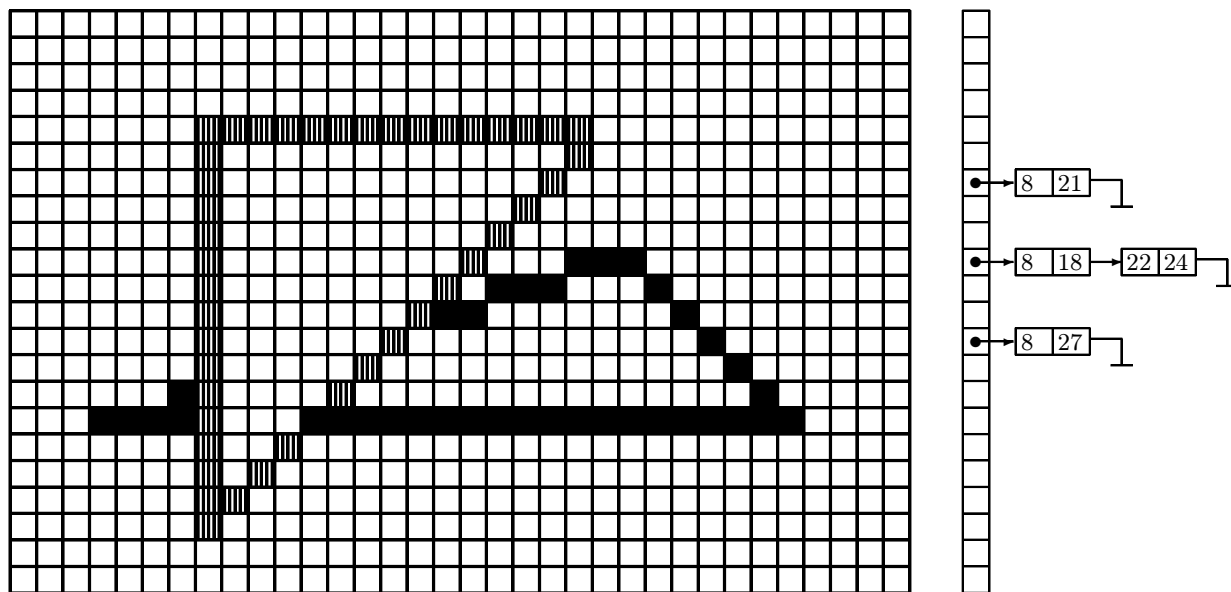


Figura 6.21 Mantenimiento de las áreas de invisibilidad (se muestran solo algunas).

un subconjunto muy pequeño del conjunto total de caras, y la complejidad del ordenamiento no se alejaría de $O(n)$ [27].

Con respecto al dibujado de las caras con áreas de invisibilidad, la idea se basa en el algoritmo de Wright para superficies funcionales, pero acomodada para trabajar en forma similar a los algoritmos scan-line. Esto se hace así dado que es posible utilizar el mismo cómputo para implementar los algoritmos de iluminación y sombreado que estudiaremos en el Capítulo 7. Esta es una de las razones principales de la practicidad y eficacia del algoritmo SPIII, dado que los demás algoritmos de cara oculta no están pensados para facilitar el sombreado de las caras.

Las áreas de invisibilidad se computan manejando una lista de pares para cada línea de barrido. Dichos pares contienen el menor y mayor x de cada cara dibujada en una línea y dada (ver Figura 6.21). Durante la conversión scan de cada polígono, entonces, es necesario chequear que para un dado y , el x del pixel a pintar no esté comprendido dentro del área de invisibilidad determinado por cada uno de los pares existentes en dicho y , específicamente, que no sea menor que el mayor x y mayor que el menor x de alguno de los pares.

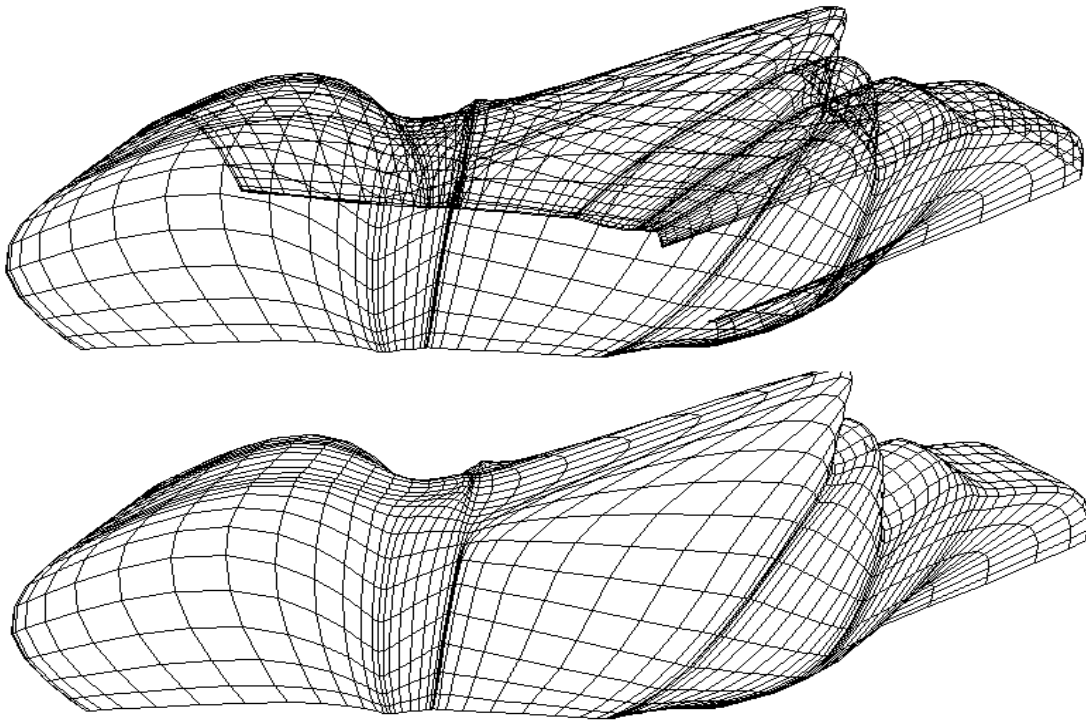


Figura 6.22 Algoritmo SPIII aplicado a una estructura de caras.

Como puede verse en la Figura 6.21, cuando las áreas de invisibilidad de dos polígonos se solapan, es posible utilizar un único par. Esta técnica produce una disminución notable en el tiempo de ejecución, dado que la cantidad de tests de invisibilidad se reduce considerablemente [27]. Podemos ver en la Figura 6.22 el resultado de aplicar este algoritmo a una compleja estructura de caras.

6.7 Ejercicios

1. Implementar un objeto estructurado en 3D y graficarlo con proyección paralela para varias transformaciones y posiciones del objeto.
2. Repetir el ejercicio anterior pero con proyección perspectiva.
3. Dado un objeto en 3D, comparar el resultado de graficarlo con distintas escalas, graficarlo acercándolo al plano de proyección, y graficarlo con el observador alejándose del plano de proyección.
4. Repetir el ejercicio anterior utilizando backface culling. Recordar aplicar primero la perspectiva, luego la eliminación de caras ocultas y por último la proyección.
5. Repetir el ejercicio 3 pero utilizando el algoritmo del pintor y efectuando la conversión scan de las caras.
6. Implementar el algoritmo de Wright para superficies funcionales.
7. Implementar el algoritmo SPIII.

6.8 Bibliografía recomendada

El manejo de las transformaciones en 3D y las coordenadas homogéneas puede consultarse en el Capítulo 5 del libro de Foley et. al. [33] y el Capítulo 22 del libro de Newmann y Sproull [66]. La descripción de los modelos estructurados de entidades gráficas puede consultarse en el capítulo 22 del Newmann-Sproull [66], y algunos aspectos avanzados en el Capítulo 3 del libro de Giloi [40] y el Capítulo 7 del Foley.

El tema de las proyecciones y la transformación de viewing está tratada de una forma compleja y algo confusa en el Capítulo 23 del Newman-Sproull. Un tratamiento más amplio figura en el Capítulo 6 del Foley, aunque tampoco se llega a una formulación sencilla. La descripción definitiva puede verse en [10]. Aquellos interesados en conocer en detalle los aspectos matemáticos de las proyecciones planares pueden consultar [19].

La interrelación entre las coordenadas homogéneas, la perspectiva y el clipping son bastante más profundas de lo expuesto en este Capítulo. Una discusión adecuada, junto con la explicación del costo computacional y los problemas en la representación numérica de estas operaciones puede leerse en [14].

La descripción y caracterización de algoritmos de línea y cara oculta se basa en el trabajo de Sutherland et. al. [78], el cual pese a su antigüedad sigue vigente y es de lectura muy recomendable. La descripción de las técnicas más primitivas y algunos métodos más modernos puede consultarse en el Capítulo 15 del Foley. También puede encontrarse en el Capítulo 5 del Giloi la descripción de algunos algoritmos competitivos y su comparación con los tradicionales. El funcionamiento del algoritmo de Wolfenstein me fue comunicado por Gustavo Patow. Los detalles de implementación, resultados, y técnicas asociadas al algoritmo SPIII pueden consultarse en [27].