Primer Coloquio – Computación Gráfica

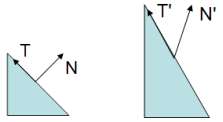
1.A. Para efectuar una rotación alrededor de un punto arbitrario ¿Qué secuencias de transformaciones debe efectuarse?

Traslación al origen, rotación, traslación al punto sobre el que deseo rotar.

Las transformaciones acumuladas serían $T_{acumulada} = Traslación_P * Rotación * Traslación_{origen}$.

1.B. Se da una matriz de transformación lineal T1 y un objeto 2D. Este tiene las normales definidas para cada uno de sus lados. Se transforma el objeto 2D aplicando T1. ¿Las normales se transforman con la misma matriz? Explique con un ejemplo.

Las normales no deben ser transformadas por la misma matriz de transformación, ya que las nuevas normales serán incorrectas. Dada una transformación T aplicada a los puntos, las normales deben ser transformadas por $(T^{-1})^T$, la inversa traspuesta de la matriz en cuestión (la traspuesta es la matriz invertida es su diagonal). La deformación de las normales no sucede si T1 es una rotación y/o una traslación, pero de ser un escalado diferencial éstas se deforman.



1.C. ¿Cuál es la razón de usar coordenadas homogéneas para representar las matrices de la transformación?

Se utilizan coordenadas homogéneas para representar las matrices de transformación dado que permite agrupar todas las transformaciones afines en una única matriz (rotación, escalado, etc). Se obtiene así mayor eficiencia tanto de cálculo cómo de utilización de memoria. Con las coordenadas homogéneas, no es necesario sumar todas las transformaciones para obtener una transformación final, basta con ir realizando el producto entre la transformación nueva y la matriz de transformaciones acumuladas.

1.D. ¿Por qué el espacio de pantalla es 3D?

El espacio de pantalla es 3D pues almacena información de profundidad en el componente z. Esto es necesario, pues al momento de dibujar en la pantalla es necesario saber cuáles objetos están adelante y cuáles están atrás de otros.

1.E. Para definir el modelo de una cámara se deben especificar varios parámetros. Defina conceptualmente cuáles son los parámetros de posición y el vector look at. ¿Es necesario especificar también la orientación? Justifique. ¿Cuál es la razón de especificar los planos near y far?

Posición: ubicación de la cámara en el mundo. P=(x, y, z).

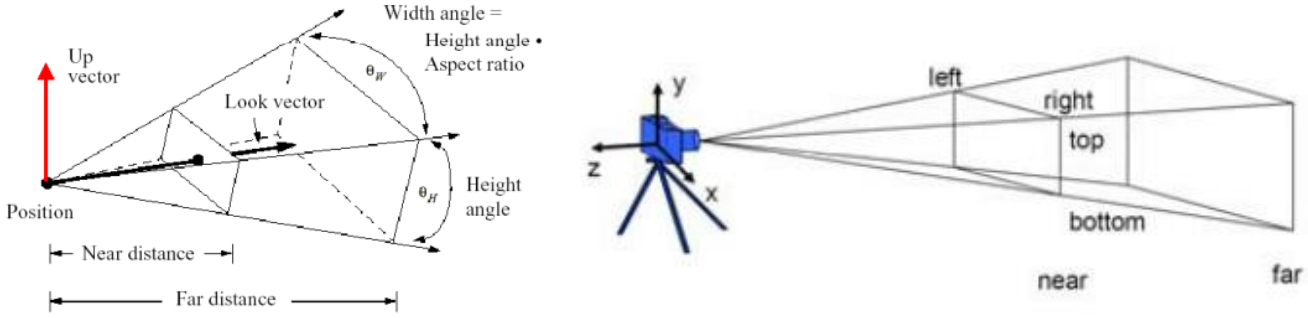
Look-At: vector que indica hacia donde apunta la cámara.

Es necesario definir la orientación para poder especificar cuando hay una rotación sobre su vector look-At para poder definir donde esta “arriba”, esto se hace mediante el vector Up.

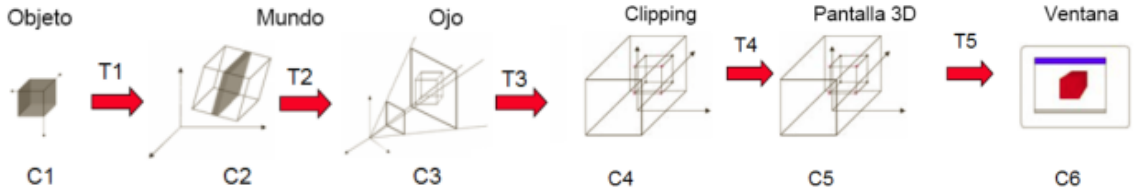
Plano Near: los objetos que están demasiado cerca de la cámara probablemente no deben dibujarse ya que bloquean al resto y seguramente están muy distorsionados. Tampoco se quieren dibujar los objetos detrás de la cámara ya que uno no espera verlos. Si uno decidiera dibujar los objetos detrás de la cámara seguramente aparecerían distorsionados por la perspectiva.

Plano Far: los objetos demasiado alejados de la cámara podrían no querer dibujarse ya que aparecerían muy pequeños para ser visualmente significativos y requerían mucho tiempo para ser renderizados. Descartando estos objetos, entonces, perdemos poco en lo que se refiere a detalle, y ganamos en lo que respecta al tiempo de renderizado. Por otro lado, puede ser que la escena este llena de objetos significativos y queramos despejar la escena de modo tal que se rendericen los objetos cercanos y se descarte el resto.

1.F. Especifique en que puntos se toma el valor a los planos near, far, top, bottom, left y right cuando se quiere utilizar glFructum de OpenGL.



2. Durante la ejecución de un pipe 3D un objeto se ve sometido a diversas transformaciones.



2.A. Especifique para cada Ci a qué espacio de coordenadas corresponde, mencionando brevemente cuales son la tares que se realizan en cada espacio.

T0 – Transformación de Modelado: traslación, rotación, escalado.

C1 – Coordenadas del Objeto: aquí se generan distintas instancias de un objeto.

T1 – Transformación de Ubicación en el Mundo: traslación, rotación.

C2 – Coordenadas del Mundo: se ubican los objetos dentro del mundo y se genera así la escena.

T2 – Transformación de Vista: traslación, rotación.

C3 – Coordenadas del Ojo/Cámara: se mapean puntos del espacio del mundo al espacio del ojo mediante las transformaciones T2. Se ubica la escena según la cámara y se especifica la cámara.

T3 – Transformación de Proyección: traslación, escalado, proyección (transformación perspectiva).

C4 – Coordenadas de Clipping: se efectúa el clipping mediante T3. (x, y, z, w)

T4 – Transformación de Perspectiva: División Perspectiva (no lineal), división por w.

C5 – Coordenadas Normalizadas del Dispositivo: se normalizan las coordenadas mediante w. (x_n, y_n, z_n, 1)

T5 – Transformación del Viewport: traslación, escalado.

C6 – Coordenadas de Ventana: se mapea el cubo de 1x1x1 que corresponde a las CND a coordenadas de la Pantalla mediante T5 para así renderizar la imagen. (x_p, y_p)

2.B. Especifique para cada Ti a qué tipo de transformaciones se refiere, mencionando cuales son las posibles transformaciones de cada etapa y si son lineales o no.

T0 – Transformación de Modelado: sirven para generar modelar el objeto. Las traslaciones, rotaciones y escalado se dan en el espacio del Objeto.

T1 – Transformación de Modelado: especifica la transformación de las coordenadas del Objeto a coordenadas comunes del Mundo. Incluye traslación y rotación del objeto dentro del mundo.

T2 – Transformación de Vista: corresponde a las especificaciones y orientación de la cámara u ojo. Transforma las coordenadas del Mundo a las coordenadas del Ojo. Incluye traslación y rotación del mundo.

T3 – Transformación de Proyección: determina el tipo de proyección (perspectiva u ortogonal) y transforma las coordenadas del Ojo a coordenadas de Clipping. Incluye las transformaciones de traslación, escalado y, obviamente, proyección.

T4 – Transformación de Perspectiva: División Perspectiva (no lineal), división por w.

T5 – Transformación del Viewport: en las CND el volumen de vista está contenido en un cubo centrado en el origen entre -1 y 1. Esta transformación mapea el cubo a coordenadas de la Pantalla. Incluye transformaciones como traslación y escalado.

2.C. ¿En qué espacio de coordenadas se lleva a cabo el clipping? ¿Por qué?

El clipping tridimensional identifica y guarda todo lo que se encuentra dentro del volumen de vista para mostrarlo posteriormente. Se descartan todos los objetos y/o partes de los mismos que estén fuera de dicho volumen. En lugar de clippear contra las rectas que constituyen los bordes de la ventana, clippearemos los objetos contra los planos limitantes del volumen de vista. El clipping en 3D puede llevarse a cabo usando extensiones del clipping en 2D.

Si hiciéramos el clipping antes de la transformación en perspectiva, es decir en el espacio del Ojo, entonces deberíamos clippear contra el fructum de vista para luego realizar la transformación en perspectiva, la proyección y dibujar en la ventana. Esto no es eficiente ya que las transformaciones que se realizan luego del clipping tienen un costo mayor, ya que tranquilamente se pueden acumular dentro de una matriz y realizarse antes evitando así varios cálculos.

Si lo hacemos luego de la transformación de perspectiva, hay dos opciones:

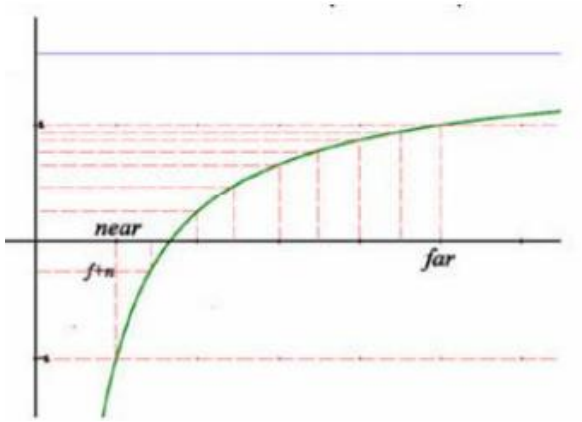
- en coordenadas 3D (CND), donde (-1, -1, -1)<= (x, y, z)<= (1, 1, 1).
- en coordenadas homogéneas (de Clipping), donde (-w, -w, -w)<= (x, y, z)<= (w, w, w).

El clipping se realiza en coordenadas homogéneas, es decir, antes de dividir por w y después de realizar la transformación de proyección perspectiva. Esto se debe a que el clipping no resulta bueno después de la normalización, por un lado porque normalizamos puntos que van a ser clipeados y por otro hay ambigüedad generada por la división de w.

2.D. ¿Alguna de las Ti es una transformación no lineal? ¿Cuál? ¿Cuál es su objetivo?

T4, la división por w es una transformación no lineal. El objetivo de esta transformación es pasar del espacio de clipping homogéneo a las coordenadas normalizadas del dispositivo.

(-w, -w, -w)<= (x, y, z)<= (w, w, w) → (-1, -1, -1)<= (x, y, z)<= (1, 1, 1)



2.E. En el pipeline programable de OpenGL ¿cuáles son las transformaciones de Ti que deben efectuarse en el programa de vértices?

En el programa de vértices se deben llevar a cabo las transformaciones de Modelado (T0), las de Ubicación en el Mundo (T1), las de Vista (T2) y las de Proyección (T3), de ahí las matrices ModelView y Proyección.

2.F. ¿Qué datos de entrada y salida debe tener un programa de vértices? ¿Cuál es el espacio en el que están los vértices que son la entrada de un programa de vértices? ¿Cuál es el espacio en el que deben estar los vértices en la salida de un programa de vértices? ¿Qué datos de entrada y salida debe tener un programa de fragmentos?

Programa de vértices: entramos en coordenadas del objeto y salimos en coordenadas de clipping.
DE: vértices en coordenadas del objeto, matriz modelView y matriz proyección.
DS: vértices en coordenadas de clipping.

Programa de fragmentos:
DE: fragmentos rasterizados.
DS: color del fragmento.

2.G. La transformación T5 involucra la transformación del viewport. ¿Qué transformaciones básicas involucra? En el espacio de la pantalla 3D ¿Cuáles son los valores límites para cada una de las coordenadas? ¿Y en el espacio de la ventana?

En las CND el volumen de vista está contenido en un cubo centrado en el origen entre -1 y 1. Esta transformación mapea el cubo a coordenadas de la Pantalla. Incluye transformaciones como traslación y escalado.

En el espacio de la pantalla 3D los valores límites son -1<=x<=1; -1<=y<=1 y -1<=z<=1

En el espacio de la Ventana los valores límites son:

- 0<=x<=Width en pixeles
- 0<=y<=Height en pixeles
- 0<=dz<=1

2.H. Indique en qué etapa del pipeline se realiza la rasterización y en cuál la cara oculta,

La rasterización se lleva a cabo luego de la transformación del Viewport y antes del procesamiento por del programa de fragmentos. Básicamente determina qué pixeles están dentro de una primitiva que se ha

especificado mediante un conjunto de vértices, produce así los fragmentos, sus características de color y su ubicación en la pantalla.

La cara oculta se realiza durante la rasterización para el caso del Z-Buffer y el Scan-Line; en coordenadas 3D de Pantalla (CND) para el caso del Depth Sort; y en coordenadas del ojo para los de Culling (Back Face Culling).

2.I. En OpenGL se generan las matrices modelView y de Proyección. ¿Cuáles son las transformaciones Ti del pipeline que contiene cada una de ellas?

ModelView; contiene las transformaciones de Modelado, Ubicación en el Mundo y Vista; T0, T1 y T2 respectivamente.

Proyección: contiene las de perspectiva T3.

2.J. Si pudiera separar la modelView en una matriz de modelado y una de vista. ¿Cuáles son las transformaciones Ti que deberían incluir cada una de ellas? Explique conceptualmente que realiza cada transformación.

La transformación de modelado es una secuencia de transformaciones de modelado, acumuladas en una sola matriz, que ubican al objeto en el mundo y modelan el objeto. Model=T1*T0

La transformación de vista es una secuencia de transformaciones, acumuladas en una sola matriz, que orientan y posicionan la cámara en el espacio. View=T2

Así se ve que ModelView=T2*T1*T0.

2.K. ¿Qué etapas del pipeline de la figura reemplaza un programa de vértices? Explique conceptualmente.

El programa de vértices toma como entrada vértices en coordenadas del Objeto y da como salida vértices en el espacio de clipping, cubre los espacios C2 y C3 y realiza las transformaciones correspondientes a T0, T1, T2 y T3. Realiza las transformaciones de vértices y normales, genera y transforma coordenadas de textura, iluminación y selección de los materiales.

2.L. ¿Qué etapas del pipeline de la figura reemplaza un programa de fragmentos? Explique conceptualmente.

El programa de fragmentos corresponde a T5: realiza operaciones sobre valores interpolados, iluminación por fragmentos, acceso a texturas, niebla y suma de colores.

3.A. Suponga que va a realizar el Clipping de rectas con el algoritmo visto en clase (Cohen-Sutherland). Se observó específicamente que el clipping se realice en la región de los w>0. Detalle los datos de entrada y salida del algoritmo.

DE: los dos puntos que definen el segmento de recta y las coordenadas de la ventana de clipping.

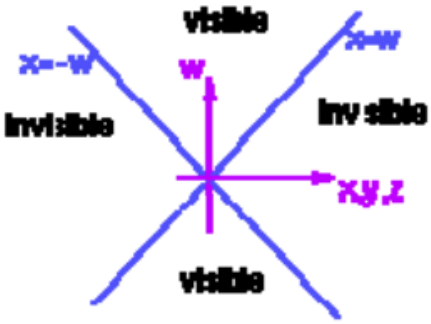
DS: los puntos que definen el segmento de recta a dibujar, y un booleano que determina si se dibuja.

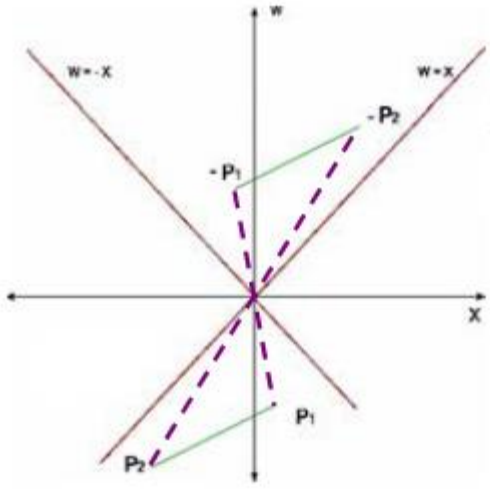
3.B. Suponga que la primitiva es clipping(…) y la recta a clippear está determinada por los puntos P1 y P2. Escriba el segmento de código utilizando dicha primitiva, clipee correctamente la recta aumentando no sólo el clipping en la región de los w>0, sino también en los w<0.

Si P1 y P2 son con w>0 clipping(P1, P2, limites)

Si P1 y P2 son con w<0 clipping(-P1, -P2, limites)

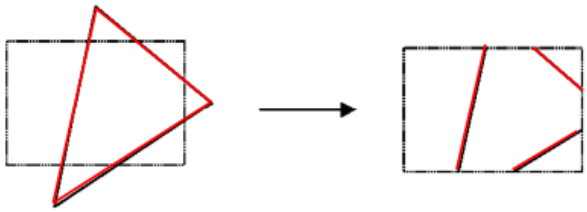
Si P1 y P2 son con w>0 Y w<0 –uno de cada uno- clipping(P1, P2, limites) y clipping(-P1, -P2, limites)





3.C. El diseño de clipping de polígonos ¿podría detallarse como una consecuencia de llamadas al algoritmo de clipping de rectas visto en clase? Explique porqué.

El clipping de polígonos no puede hacerse con los algoritmos de clipping de líneas ya que no se obtiene un área cerrada. Podemos, sin embargo, hacer lo siguiente: por cada pixel en el polígono, decidir si está dentro del volumen de vista. Esto funciona pero es demasiado ineficiente ya que hace la conversión scan de todo el polígono aunque sólo dibuja las partes visibles.

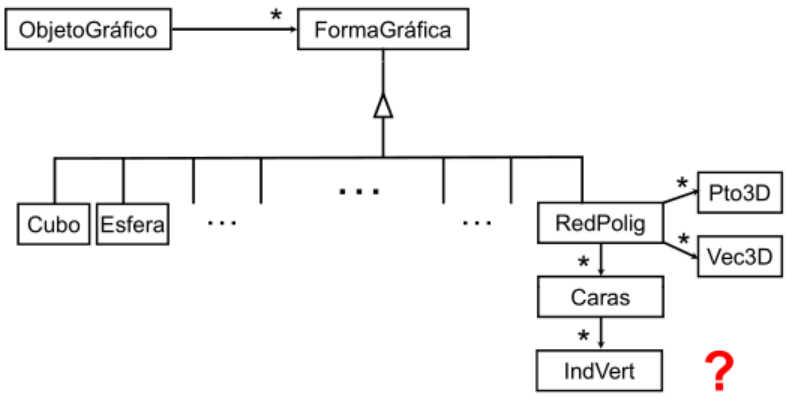


Un algoritmo es el Bounding Box en el cuál se utiliza el menor rectángulo, alineado con la ventana, que contiene al polígono. Se puede evitar así el clipping de aquellos polígonos cuyo Bounding Box está contenido dentro o fuera de la ventana de clipping. Si está parcialmente dentro, sin embargo, se requerirá un procesamiento más detallado.

Otra opción es el algoritmo de Southerland-Hodgeman que procesa cada polígono, en orden, con cada uno de los bordes del área de clipping. Sin embargo, funciona sólo para polígonos convexos. Para cada uno de los bordes de la ventana se procesan todos los vértices del polígono y se genera un conjunto de vértices que se utilizará como nuevo borde. En cada paso de procesamiento pueden presentarse 4 casos: paso de adentro a adentro, de adentro a afuera, de afuera a adentro o de afuera a afuera.

Si generáramos el algoritmo de Southerland-Hodgeman en función del de clipping de rectas tal vez resultaría posible hacerlo.

3.D. Una manera de controlar la suavidad de un objeto 3D sería tener en cuenta el ángulo entre las normales. Suponiendo que esto se adiciona a la clase red poligonal: especifique la clase objetoGráfico, teniendo en cuenta la incorporación de tita. Incorpore jerarquías y clases necesarias para considerar objetos 3D generados por Sweep. Escriba un algoritmo para suavizar un objeto 3D representado por una red poligonal.




```
Por cada cara de la red poligonal
    Recorrer sus caras vecinas
        Si el ángulo entre las normales de la cara actual y la vecina es <90°
        Entonces (Suavizar)
            Promediar Normales
            Asignar a los vértices compartidos la nueva normal promediada.
```

*Necesito información de la topología, las caras vecinas por ejemplo.

Clipping

Si la primitiva es un punto, alcanza con comparar sus coordenadas con la ventana de clipping para definir si está dentro o fuera. Si la primitiva es una línea se puede dar los casos que esté completamente dentro o fuera o se intersecte con la ventana. El algoritmo de clipping debe identificar y dibujar las líneas que están completamente dentro de la ventana y encontrar las intersecciones de las líneas que están parcialmente dentro de la ventana y dibujar las secciones correspondientes.

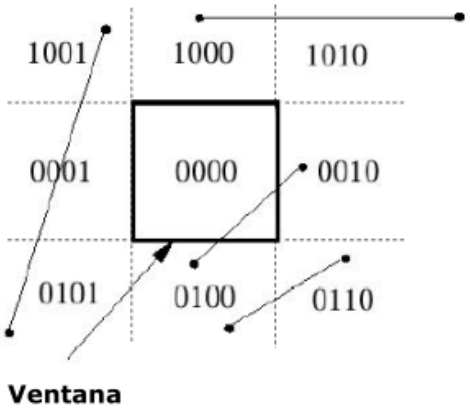
A fuerza bruta podríamos hacer el clipping con la ecuación aproximada de cada recta y los bordes de la ventana, pero así la fórmula sólo maneja líneas infinitas y no maneja líneas verticales. Usaremos entonces la ecuación paramétrica de la línea ya que es adecuada para un segmento:

$$p(t) = p_0 + t(p_1 - p_0) \qquad 0 < t < 1$$

Algoritmo de Clipping de Cohen-Sutherland de Rectas

Para identificar rápidamente las líneas que están dentro y fuera de la ventana, les asigna un código de región de 4 bits a cada uno de los puntos extremos del segmento de recta. Los bits son puestos en 1 de acuerdo a:

- Bit 1: puntos por arriba de la ventana.
- Bit 2: puntos por debajo de la ventana.
- Bit 3: puntos a la izquierda de la ventana.
- Bit 4: puntos a la derecha de la ventana.



Se ve así que hay 3 casos:

- Una línea es trivialmente aceptada si los códigos de ambos puntos son 0000.

- Una línea es trivialmente rechazada si la operación AND bit a bit de los códigos de los puntos NO es 0000.
- De otro modo se requiere más procesamiento ya que la línea no puede ser trivialmente aceptada o rechazada.

Aplicamos así el algoritmo de clipping sobre un segmento de la recta. Si éste no puede aceptarse o rechazarse trivialmente, debemos dividirlo en dos segmentos y aplicar el algoritmo sobre cada segmento generado. Se debe convenir un orden para chequear cada uno de los lados de la ventana de clipping pues para realizar la subdivisión hay que calcular el punto de intersección con la ventana. Así, el lado contra el que se clippea fija o bien y o x, y puede sustituirse en la ecuación de la recta.

```
Algoritmo Código
DE: p (coord. x,y)
    Ventana (coord. xmin,ymin,xmax,ymax)
DS: codigo

codigo ← centro
si x<xmin
entonces
  codigo ← codigo or izquierda
sino
  si x>xmax
  entonces
    codigo ← codigo or derecha

si y<ymin
entonces
  codigo ← codigo or abajo
sino
  si y>ymax
  entonces
    codigo ← codigo or arriba
```

```
Algoritmo Descartar Segmento                Centro ← 0000
DE: codigo                                  Izq    ← 0001
    p0(coord x0,y0), p1(coord x1,y1)        Der    ← 0010
    Ventana (coord. xmin,ymin,xmax,ymax)    Abajo  ← 0100
DS: p (coord x,y)                           Arriba ← 1000

si (codigo and arriba)<>0
entonces
  x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0); y = ymax;
sino
  si (codigo and abajo)<>0
  entonces
    x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0); y = ymin;
  sino
    si (codigo and derecha)<>0
    entonces
      y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0); x = xmax;
    sino
      si (codigo and izquierda)<>0}
      entonces
        y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0); x = xmin;
```

```
Algoritmo Clipping
DE: p0(coord. x0,y0), p1(coord. x1,y1)
    Ventana (coordenadas xmin,ymin,xmax,ymax)
DS: p0(coordenadas x0,y0), p1(coordenadas x1,y1)
    Se Dibuja (booleano)

Código(p0,Ventana,codigo0); Código(p1,Ventana,codigo1);
si (codigo0 = centro) and (codigo1 = centro)
entonces
  Se Dibuja ← verdadero
sino
  si (codigo0 and codigo1 )<>centro
  entonces
    Se Dibuja ← falso
  sino
    si codigo0 <> centro
    entonces
      Descartar Segmento (codigo0,p0,p1,p)
      Clipping(p,p1,Ventana,Se Dibuja); p0 ← p
    sino
      Descartar Segmento (codigo1,p1,p0,p)
      Clipping(p0,p,Ventana,Se Dibuja); p1 ← p
```

El clipping en 3D utiliza el algoritmo de Cohen-Sutherland y divide el espacio en 27 regiones usando un código de 6 bits para chequear si los puntos están dentro del volumen de vista –w<=x,y,z<=w. Se clippea, sin embargo, para w>0 y para w<0 se hace la inversa. (izquierda, derecha, arriba, abajo, frente, detras)