Informática Gráfica

Juan Carlos Torres

Curso 2024/25

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Disclaimer
You can edit this page to suit your needs. For instance, here we have a no copyright statement, a colophon and some other information. This page is based on the corresponding page of Ken Arroyo Ohori's thesis, with minimal changes.

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.

Visualización 9

Los sistemas gráficos suelen generar como resultado final una imagen en algún dispositivo físico de visualización, normalmente en un monitor raster. Aunque hay otros tipos de dispositivos, como las pantallas estereoscópicas o los cascos de realidad virtual.

Para generar la imagen en un monitor se deben transformar las coordenadas de la escena al sistema de coordenadas del dispositivo, que es bidimensional y discreto.

Cuando el modelo es 2D esta transformación se puede especificar dando el rectángulo de la escena a visualizar(llamado *ventana*) y el del monitor en el que se va a generar la imagen (denominada *viewport*), como se muestra en la figura 9.1.

Cuando la escena es 3D el proceso es mas complejo. Conceptualmente la imagen se genera como si tuviésemos una cámara virtual en el mundo 3D, que se especifica dado su posición, orientación y focal. La imagen que se genera es la proyección del mundo en un rectángulo (el plano de proyección de la cámara virtual, que ene este caso sería la *ventana*). Esta proyección se transforma al *viewport* del dispositivo de salida (figura 9.2).

Las operaciones que hay que realizar para generar la imagen de una escena son:

- Transformación de visualización: Transformar coordenadas 3D de los objetos a coordenadas de pantalla.
- Recortado: Eliminación de partes de polígonos fuera de la zona visible.
- Rasterización y eliminación de partes ocultas: Determinar los pixeles cubiertos por cada primitiva.
- Iluminación y texturización: Calcular el color de cada pixel.

Estos pasos pueden realizarse en diferente orden.

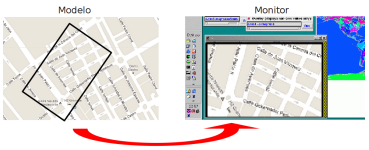

Figura 9.1: La transformación de visualización en 2D transforma la ventana en el viewport.

Figura 9.2: Cámara virtual en la visualización de un escenario 3D.

9.1. Cauce gráfico

El término cauce gráfico (*graphics pipeline*) hace referencia al conjunto de pasos que se realizan para visualizar polígonos. El procesamiento se realiza en tres fases (figura 9.3):

- Operaciones con vértices.
- Recortado y rasterización.
- Operaciones con fragmentos.

La primera y tercera fases son programables, mientras que la intermedia tiene una funcionalidad fija (figura 9.3).

En las fases programables (operaciones con vértices y con fragmentos) se realizan las transformaciones, la iluminación y la texturización. Ya vimos como se realiza el cálculo de iluminación (Capítulo 5), y que se

puede realizar a nivel de polígono, de vértice o de pixel. Cuando se realiza por polígono o vértice se calcula después de la transformación de visualización como último paso de las operaciones con vértices. Si se realiza a nivel de pixel se calcula en las operaciones con fragmentos. En cualquier caso, la aplicación de textura se debe realizar al final del pipeline en las operaciones con fragmentos.

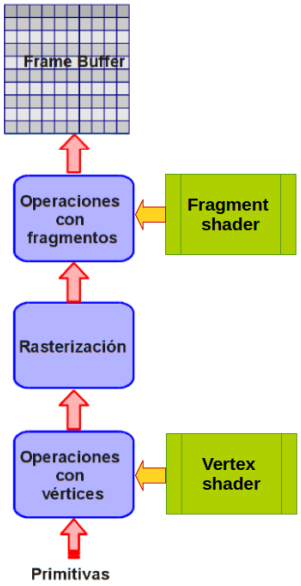

Figura 9.3: Cauce gráfico en GPU programable.

9.1.1. Transformación de visualización

Al visualizar la escena las coordenadas de los objetos sufren diferentes transformaciones geométricas. En primer lugar se transforman para ubicarlos en la escena. A continuación se deben transformar para calcular como se ven desde la cámara. Finalmente se transformar a coordenadas de dispositivo (figura 9.4).

A lo largo de este proceso, existen diferentes sistemas de coordenadas que organizan y definen la representación de los objetos en la escena[1]:

- **Coordenadas de objeto (Object Space)** Este es el sistema de coordenadas local de cada objeto. En este espacio, las coordenadas de los vértices se definen en relación con el origen del propio objeto, que suele ser el centro, un vértice u otro punto relevante del objeto (por ejemplo el centro de la base).
- **Coordenadas del mundo (World Space)** Las coordenadas del mundo son un sistema de referencia global en el que se posicionan todos los objetos de la escena. Cada objeto tiene su ubicación, escala y rotación definidas en este espacio. La primera transformación que se realiza es la **transformación de modelado**, que convierte las coordenadas de objeto en coordenadas del mundo.
- **Coordenadas de cámara (View Space)** En el espacio de la cámara ésta está en el origen de las coordenadas, las coordenadas de los objetos están expresadas en función de su posición respecto de la cámara. Para determinar cómo se ve la escena desde un punto de vista concreto, las coordenadas del mundo se transforman mediante la **transformación de vista** en coordenadas de cámara. En el sistema de coordenadas de la cámara el eje Z suele estar invertido para que la coordenada Z coincida con la profundidad.
- **Coordenadas normalizadas (Normalized Coordinates)** Sistema de coordenadas 3D en el que el espacio visualizado se representa dentro de un cubo normalizado. La conversión de coordenadas de cámara a coordenadas normalizadas se realiza mediante la **transformación de proyección**.
- **Coordenadas de dispositivo (Device Space)** Sistema de coordenadas 2D del dispositivo de viusalización. La conversión de coordenadas normalizadas a coordenadas de dispositivo se realiza mediante la **transformación de viewport**.

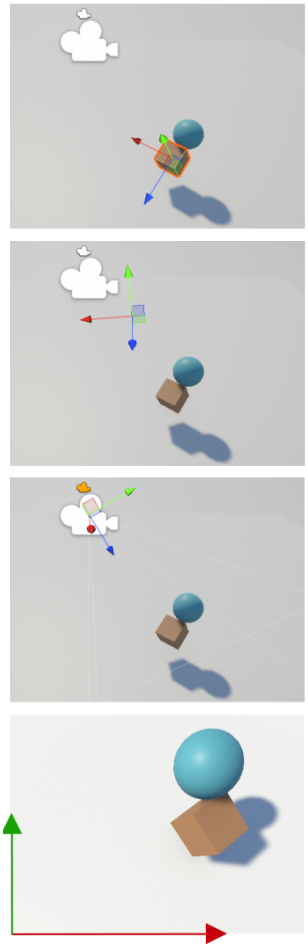

Figura 9.4: Principales sistemas de coordenadas usados en un sistema gráfico. De arriba a abajo: coordenadas de objeto, Coordenadas del mundo, Coordenadas de cámara y Coordenadas de dispositivo.

Las transformaciones de modelado y vista realizan escalados, rotaciones y traslaciones. Las transformaciones que se debe aplicar para realizar la transformación de vista se pueden calcular a partir de la posición y orientación de la cámara en el sistema de coordenadas del mundo, que se suele dar con los siguientes parámetros:

PRP (Projection reference point) Punto del espacio foco de la proyección, donde esta situada la cámara.

VPN (View plane normal) Vector perpendicular al plano de proyección.
VRP (View reference point) Indica el punto de interés, o target, que es el punto que se proyecta en el centro de la imagen. Podemos pensar en él como el punto al que se mira ("look at point").
VUP (Vector vertical) Dirección vertical de la cámara.

La transformación de vista transforma los vértices de manera que PRP queda en el origen, VPN queda alineado con el eje Z y el vector VUP es vertical (9.5).

Un sistema de coordenadas está definido por tres vectores ortogonales, como quiera que los vectores introducidos para definir la cámara pueden no ser ortogonales, es necesario generar tres vectores ortogonales: $\mathbf{c_x}$,$\mathbf{c_y}$ y $\mathbf{c_z}$ que, junto con la posición de la cámara, forman el **sistema de referencia de la cámara**:

$$\mathbf{c_z} = \frac{\mathbf{VPN}}{\|\mathbf{VPN}\|} \quad \text{(eje Z paralelo a VPN)}$$

$$\mathbf{c_x} = \frac{\mathbf{VPN} \times \mathbf{VUP}}{\|\mathbf{VPN} \times \mathbf{VUP}\|} \quad \text{(eje X perpendicular a VPN y VUP)}$$

$$\mathbf{c_y} = \mathbf{c_z} \times \mathbf{c_x} \quad \text{(eje Y perpendicular a los otros dos)}$$

La transformación de vista se puede construir a partir de estos tres vectores como composición de las siguientes transformaciones (figura 9.6):

- Traslación de $-PRP$, haciendo que el origen de coordenadas este en la cámara.
- Rotar respecto al eje X para llevar el vector VPN al plano XZ.
- Rotar respecto al eje Y para llevar VPN al eje $-Z$.
- Rotación respecto al eje Z para llevar a c_y al eje Y.

9.1.2. Proyección

La proyección se puede realizar en perspectiva o paralela (u ortográfica). La proyección en perspectiva es similar al proceso que ocurre en las cámaras fotográficas cuando la escena se proyecta en el CCD. En una proyección en perspectiva el tamaño de los objetos en la imagen disminuye con la distancia a la cámara.

En una proyección ortográfica las líneas de proyección son paralelas al plano de proyección, por lo que el tamaño de los objetos en la imagen es independiente de la profundidad.

El volumen de visión (espacio de la escena que se muestra en la imagen) es un paralelepípedo en el caso de la proyección paralela y un tronco de pirámide de base rectangular para la proyección en perspectiva (figura 9.7).

La figure 9.8 muestra un esquema 2D del cálculo de la transformación de perspectiva[2]. Es fácil comprobar, por semejanza de triángulos que

$$y'/near = y/z$$

1: Internamente se definen algunos sistemas de coordenadas adicionales para facilitar la realización del recortado y la eliminación de partes ocultas.

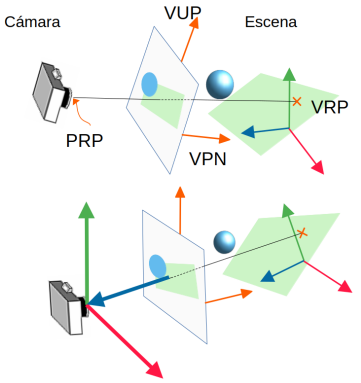

Figura 9.5: Transformación de vista. Arriba parámetros de la cámara en sistema de coordenadas del mundo. Abajo sistema de coordenadas de cámara.

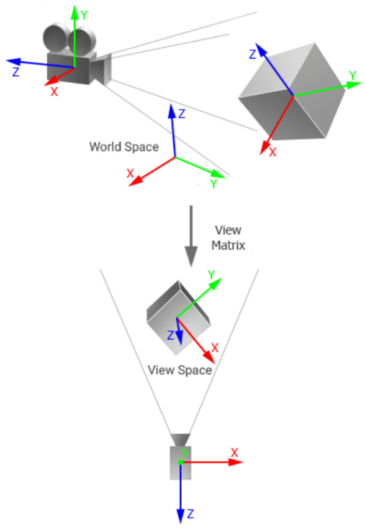

Figura 9.6: Sistemas de coordenadas del mundo y de vista.

2: https://learnwebgl.brown37.net/08_projections/projections_perspective.html

por tanto

$$y' = y \cdot near/z$$

La transformación se representa como una transformación en coordenadas homogéneas por una matriz con valores no nulos en la última fila, que puede generar un valor de w' distinto de 1.

9.1.3. Recortado

En el recortado se elimina la geometría que no es visible teniendo en cuenta los parámetros de la cámara. Para ello se descartan los polígonos que está totalmente fuera del volumen de visión (*view frustrum culling* figura 9.9) y se recortan los que son parcialmente visibles (figura9.10).Esto es, se calcula y triangula la parte visible de los que cruzan el volumen de visión, descartando el resto. El recortado simplifica los algoritmos de las fases siguiente y reduce el volumen de información que es necesario procesar en ellas.

9.1.4. Rasterización

La rasterización es el proceso de discretización de las primitivas en pixels. La discretización se realiza por hardware procesando los triángulos por líneas de barrido. Un pixel está cubierto por un triángulo si su centro se encuentra dentro del triángulo.

La rasterización puede producir bordes dentados (pixelados) cuando hay un gradiente grande en la imagen. Para evitarlo se pueden usar técnicas de antialising.

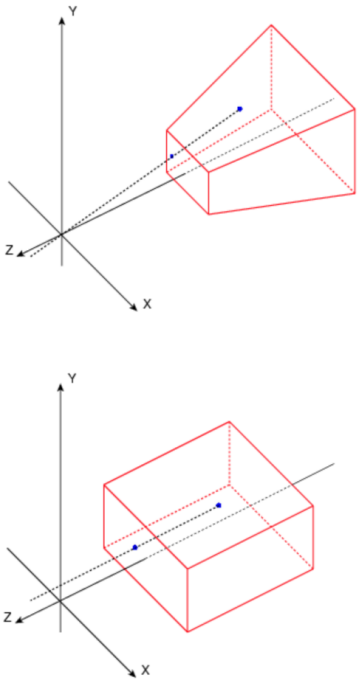

Figura 9.7: Volumen de visión: proyección perspectiva arriba y ortográfica abajo.

9.1.5. Transformación de viewport

El término *viewport* hace referencia a la zona rectangular de la ventana donde se dibuja la escena. La transformación de viewport es lineal y consta simplemente de escalados y traslaciones.

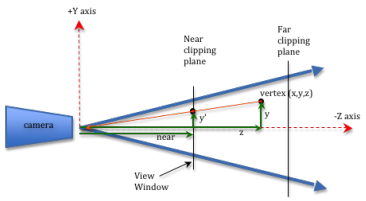

Figura 9.8: Esquema de la transformación de perspectiva.

9.2. Visualización en OpenGL

OpenGL almacena en su estado dos pilas de matrices de transformación. En una almacena la concatenación de las matrices de modelado y vista, en la otra la transformación de proyección. Se puede seleccionar a cual de las dos matrices se le concatenan las transformaciones que se den con la orden *glMatrixModel*:

```
glMatrixMode(GL_MODELVIEW);
glMatrixMode(GL_PROJECTION);
```

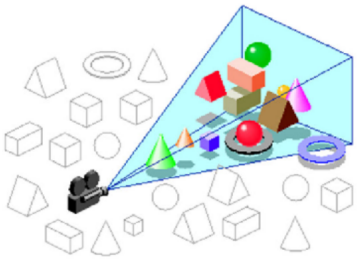

Figura 9.9: View frustrun culling. Los objetos que están totalmente fuera del volumen de visión son descartados.

9.2.1. Proyección

La región en forma de pirámide truncada contendrá los objetos de la escena visibles en la imagen, se llama *view-frustum*, y está determinada por estos 6 valores reales (figura 9.11):

n,f = (*near* y *far*). Coordenadas Z (negadas) de la cara delantera y trasera del frustum, determinan su extensión en Z. Se cumple $0 < n < f$. Si z está en el frustum, $-f \leq z \leq -n$ (los objetos visibles están en Z-).

l,r = (*left* y *right*). Coordenadas x de la arista vertical izquierda y derecha respectivamente, de la cara delantera del frustum (la más cercana al observador).

b,t = (*bottom* y *top*). Coordenadas y de la arista inferior y superior, respectivamente, de la cara delantera del frustum.

Para usar una transformación de perspectiva se utiliza la función *glFrustum*:

```
glFrustum( GLdouble l, GLdouble r, GLdouble b, GLdouble t,
           GLdouble n, GLdouble f ) ;
```

Si queremos una proyección ortográfica, debemos usar *glOrtho* con los mismos parámetros:

```
glOrtho( GLdouble l, GLdouble r, GLdouble b, GLdouble t,
         GLdouble n, GLdouble f ) ;
```

Estas llamadas generan la matriz de transformación y la componen con la matriz de proyección existente.

Para proyecciones en perspectiva se puede usar la función *gluPerspective* como alternativa (figura 9.12):

```
gluPerspective(GLdouble alfa, GLdouble a, GLdouble n, GLdouble f);
```

esta función equivale a un *glFrustum* con $r = -l$ y $t = -b$, con:

$alfa$ es la apertura vertical, en grados, del campo de visión. Con valores entre 0 y 180.0).

a es la relación de aspecto de la imagen (ancho dividido por alto).

9.2.2. Viewport

En cualquier momento es posible cambiar la configuración del viewport que OpenGL almacena como parte de su estado llamando a la función *glViewport*:

```
glViewport( GLint xl, GLint yb, GLsizei w, GLsizei h);
```

donde (xl, yb) es la posición (columna y fila) del pixel que ocupa, en la ventana del dispositivo de salida, la esquina inferior izquierda del viewport. Los valores w y h indican la anchura y altura en pixel del viewport.

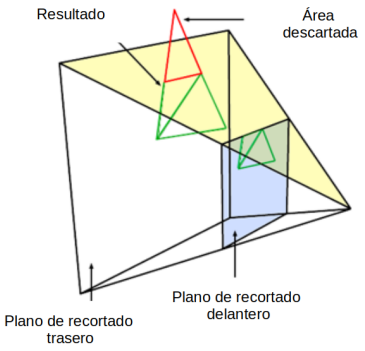

Figura 9.10: Recortado. Se calcula la parte del polígono que está dentro del volumen de visión, que es triángulada (triángulos verdes).

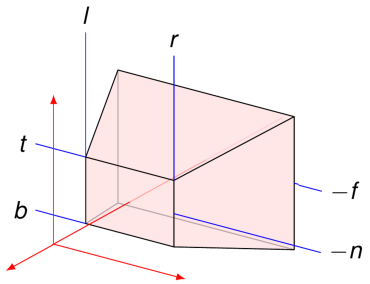

Figura 9.11: Parámetros de definición del frustrum.

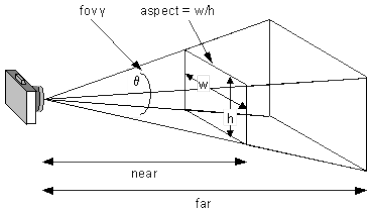

Figura 9.12: Parámetros de definición del frustrum con gluPerspective.

9.2.3. Transformaciones de modelado y vista

La matriz *modelview* se puede especificar en OpenGL mediante estos pasos:

1. Hacer la llamada *glMatrixMode(GL_MODELVIEW)*, para indicar que las siguientes operaciones actuan sobre la matriz *modelview*.
2. Usar *glLoadidentity* para hacer qu la matriz *Modelview* sea la identidad.
3. Indicar la matriz de vista.
4. Usar una (o varias) llamadas a las funciones *glScale*, *glRotate*, o *glTranslate* para componer la matriz de modelado.

OpenGL construye *modelview* componiendo las matrices que se le proporcionan en los pasos 3 y 4.

La matriz de vista se puede dar indicando explícitamente la transformación del sistema de coordenadas del mundo al de la cámara usando rotaciones y traslaciones.

Alternativamente se puede usar la función *gluLookAt* (de la librería GLU) que permite componer una matriz de vista de forma muy cómoda, ya que acepta directamente los valores de *PRP* (posición de la cámara), *VRP* (target) y *VUP* (vertical de la cámara) como parámetros. Está declarada como sigue:

```
void gluLookAt( GLdouble PRPx, GLdouble PRPy, GLdouble PRPz,
                GLdouble VRPx, GLdouble VRPy, GLdouble VRPz,
                GLdouble VUPx, GLdouble VUPy, GLdouble VUPz);
```

El vector *VPN* Se calcula como $VRP - PRP$[3].

3: Esto hace que no se puedan definir proyecciones oblicuas.

9.2.4. Cámara orbital

Cámara orbital en OpenGL se puede definir usando ángulos de Euler:

```
void dibujoEscena() {
  ....
  glClear( GL_COLOR_BUFFER_BIT  | GL_DEPTH_BUFFER_BIT );
  glTranslatef( 0, 0,-D);
  glRotatef( view_rotx, 1.0, 0.0, 0.0 ); // Rotaciones de la
    camara
  glRotatef( view_roty, 0.0, 1.0, 0.0 );
  glRotatef( view_rotz, 0.0, 0.0, 1.0 );
  glTranslatef(PRPx,PRPy,PRPz);
  ...
```

donde PRP es el centro de atención, $view\_rot$ son los ángulos de Euler y D es la distancia de la cámara al centro de atención.

La interacción se puede realizar con teclado:

```
static void especial(int k, int x, int y) {
  switch (k) {
  case GLUT_KEY_UP:
    view_rotx += 5.0;
    break;
  case GLUT_KEY_DOWN:
    view_rotx -= 5.0;
```

```
    break;
  case GLUT_KEY_LEFT:
    view_roty += 5.0;
    break;
  case GLUT_KEY_RIGHT:
    view_roty -= 5.0;
    break;
  default:
    return;
  }
  glutPostRedisplay();
}
```

o con ratón:

```
void clickRaton( int boton, int estado, int x, int y )
{
  if(boton== GLUT_MIDDLE_BUTTON && estado == GLUT_DOWN) {
    Xref=x;        // Almacena posicion (en coor. de pantalla)
    Yref=y;
    RatonPulsado=GL_TRUE;
    }
  else RatonPulsado=GL_FALSE;
    ..
}

void RatonMovido( int x, int y )
{
  if(RatonPulsado)  {      // Si el raton esta pulsado
    view_roty -= (Yref -y) / 100.0;    // Cambia offset de imagen
    view_rotx += (Xref -x) / 100.0;
    Xref=x;                // Actualiza ultima posicion de raton
    Yref=y;
    glutPostRedisplay();
    }
  ...
}
```

Alternativamente usando $glutLookAt$ se puede crear como.

```
void dibujoEscena() {
  ....
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
  gluLookAt(PRPx, PRPy, PRPz, VRPx, VRPy, VRPz, VUPx, VUPy, VUPz);
  ...
```

donde VRP es el centro de atención, VUP el vector hacía arriba y PRP la posición de la cámara.

9.2.5. Cámara primera persona

La cámara está en la posición de un avatar que se mueve en el escenario. El usuario puede controlar la dirección en la que mira la cámara.

La dirección se puede especificar como dos giros en direcciones ortogonales a la dirección de avance del personaje (con las mismas formulas de la posición del caso anterior pero ahora usadas para calcular VPN).

$$VPN = (sin(\phi)\, sin(\theta), cos(\phi), sin(\phi)\, cos(\theta))$$

VUP se debe calcular de forma que no sea perpendicular al plano proyección, por ejemplo haciendo:

$$VUP = ((0,1,0) \times VPN) \times VPN)$$

Además es conveniente hacer que la orientación de la cámara se ajuste a la dirección de movimiento del personaje. Para ello debemos componer dos giros de la cámara: alinear con la dirección de movimiento del personaje y el indicado por el usuario.

9.3. Cámara en Unity

La cámara es un objeto del grafo de escena. Modificamos sus parámetros en el inspector. Para que la cámara siga a un componente de la escena basta con incluirla en el grafo de escena dependiendo de ese componente (figura 9.13).

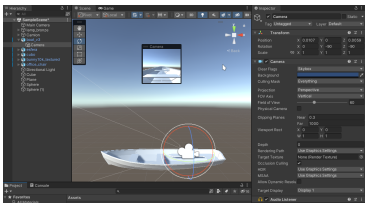

Figura 9.13: Cámara en primera persona en Unity.

Una cámara en tercera persona sigue la posición de un avatar que se mueve en el escenario a un distancia fija de él. El usuario puede controlar la dirección en la que mira la cámara. Ahora la posición de la cámara ($\vec{P_c}$) será la del personaje menos una cantidad (d) por su dirección de movimiento $\vec{D_p}$

$$\vec{P_c} = d\ \vec{D_p}$$

d indica la distancia de la cámara al personaje.

9.4. Ejercicios

1. Redacta un algoritmo para hacer view frustrum culling en un sistema con proyección ortogonal.
2. Dado un cubo unidad situado con un vértice en el origen del sistema de coordenadas y otro en el punto (1,1,1), indicar los parámetros y transformaciones a usar para visualizarlo desde las siguientes posiciones:
   - Desde el eje Y, a una distancia de 5 unidades mirando hacia el origen.
   - Desde la bisectriz del octante con X negativo e Y,Z positivos, mirando hacia el centro del cubo.
3. Describir la imagen que se generaría en cada uno de los casos anteriores.
4. Redactar procedimientos que efectúen los siguientes recorridos de cámara:
   - Recorrido circular de radio 5 unidades mirando hacia el origen sobre el eje y sobre el plano y=0.

   - Recorrido en forma de escalera de caracol, de radio 5 unidades, partiendo de una altura de y=5 hasta y=-5, mirando hacia el origen.
   - Alejamiento de la cámara, mirando desde la bisectriz del primer octante al origen de coordenadas (comenzando en una distancia de 5 y finalizando a una distancia de 20).

5. Implementar un algoritmo para realizar un efecto zoom en OpenGL.
6. Describir un algoritmo que simule una caída en barrena vertical en OpenGL. El observador siempre mira al origen de coordenadas, partiendo de una altura y=20 hasta y=1, haciendo que la cámara gire mientras desciende.
7. Para definir una vista se puede utilizar el siguiente conjunto de parámetros:

   - PCamara: Posición del observador.
   - PAtencion: Posición a la que mira.
   - Apertura: Angulo de apertura del cono de visión.
   - Spin: Angulo de giro de cámara respecto a la vertical.

   Indicar como se fijarían los parámetros de vista de OpenGL a partir de estos.
8. ¿Se puede realizar la transición de una proyección en perspectiva a una paralela de forma continua?
9. Programa una cámara en tercera persona que siga a un objeto en movimiento permitiendo que el usuario ajuste la distancia de la cámara al objeto y la orientación de la cámara.
10. Crea un efecto de oscilación de cámara como si el observador caminara. La cámara debe moverse ligeramente hacia arriba y abajo mientras se traslada hacia adelante.
11. Implementa una cámara que simule una cámara de seguridad, que gire automáticamente para seguir un objeto del escenario.
12. Diseña un sistema que permita indicar interactivamente posiciones en el escenario. Cuando se introduzca una posición la cámara debe avanzar de forma continua hasta ese punto a una velocidad prefijada.
13. Crea una visualización con dos viewport de forma que se simule la visualización estereoscópica para gafas de realidad virtual. La posición de la cámara en las vistas deben estar ligeramente desplazada.