

Aprobador-Examen-Teoria-y-examen...



DevGenius



Informática Gráfica



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada



MÁSTER EN

Inteligencia Artificial & Data Management

MADRID

Formamos
talento para un futuro
Sostenible

saber más



Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH



TEMA 1

Z-BUFFER

CÓDIGO:

```
color canvas[resx,resy]=color de fondo
int zbuffer[resx,resy]=infinito
for each primitive
    for each fragment
        if zbuffer[x,y] > fragment.depth
            zbuffer[x,y] = fragment.depth
            canvas[x,y] = fragment.color
```

EXPLICACIÓN:

Para cada píxel de la pantalla, mantenemos un registro de la profundidad de un objeto dentro del píxel que se encuentra más cerca del observador. Además de la profundidad, también registramos la intensidad que se debe mostrar para mostrar el objeto. El búfer de profundidad es una extensión del búfer de cuadro. El algoritmo de búfer de profundidad requiere 2 matrices, intensidad y profundidad, cada una de las cuales está indexada por coordenadas de píxeles (x, y).

TEMA 2

MALLA

Existen varias estructuras de datos para representar una malla, como la estructura de caras-aristas-vértices, aristas aladas o las semiaristas aladas. Pero en este caso vamos a hablar de la estructura utilizada en prácticas, que se compone de dos **vectores uno con caras y otro con vértices**, donde cada posición del vector de caras posee una estructura cara donde se encuentran los índices de los tres vértices que forman esa cara los cuales podemos encontrar en el vector de vértices donde cada posición de este posee una estructura vértice con las coordenadas de dicho vértice.

Los vértices se deben dar de forma consistente, normalmente se suelen dar en **sentido antihorario** (CCW).

Para dibujar la malla, se tiene que dibujar cada triángulo que la compone.

Problemas:

- Información redundante: **Podemos evitarlo con indexación de vértices (añadir un vector de vértices).**
- Puede haber errores en la representación **Podemos evitar esto usando estructuras concretas de las que obtener los datos (formato PLY).**

WUOLAH

- No se puede usar para representar sólidos. Podemos representar cilindros mediante mallas de revolución (a partir de una lista de vértices iniciales vamos girando estos y generando nuevos vértices y con estos caras que cada instancia con la anterior y la posterior a ella generando así el sólido).
- No se pueden realizar operaciones de edición salvo a nivel de triángulo.

TEMA 3

LUCES

Para hacer que el objeto sea aún más real tenemos 3 aproximaciones al realismo que van de menos a más:

- Sombreado plano: Una vez por cada polígono que forma el modelo, tras esto asigna el color a cada pixel del polígono.
- Sombreado de Gouraud (de vértices o smooth shading): una vez por vértice, tras esto se interpola con los píxeles de cada polígono.
- Sombreado de Phong (de pixel): Una vez por pixel del polígono, se interpolan los atributos de los vértices.

La iluminación en OpenGL está formada por la suma de 3 componentes:

- Componente difusa: esta componente depende de la inclinación que tenga nuestra fuente de luz, cuanto más inclinada esté, más se reflejarán los rayos de luz. La forma que tenemos de averiguar esta inclinación es con las normales.

Para calcular la reflexión difusa de una cara realizamos el producto escalar de su vector normal y el vector de la fuente de luz, que tiene la siguiente expresión: $N \cdot L = |N| \cdot |L| \cdot \cos(\alpha)$. Si tenemos los vectores normalizados, los módulos de los vectores valdrán 1, por lo que se podría simplificar la expresión de la siguiente forma: $N \cdot L = 1 \cdot 1 \cdot \cos(\alpha) = \cos(\alpha)$. La reflexión difusa de una cara (teniendo su normal calculada), tenderá a 0 cuando la normal sea perpendicular al vector de dirección de la luz y será máximo cuando los vectores sean paralelos (debido al valor del coseno de α). ¿Es mas acertado usar la normal de los vértices que la de las caras para esta componente?.

- Componente especular: cuando un rayo se refleja, produce un rayo de salida con el mismo ángulo que el rayo inicial. Esto hace que la posición del observador sea clave, ya que dependiendo de ella se podrá ver el rayo o no. Podemos obtenerla usando la transformación de vista (inversa de la transformación de la cámara)

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	 PLAN TURBO	 PLAN PRO	 PLAN PRO+
 Descargas sin publi al mes	10 	40 	80 
 Elimina el video entre descargas			
 Descarga carpetas			
 Descarga archivos grandes			
 Visualiza apuntes online sin publi			
 Elimina toda la publi web			
 Precios Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo,
¿Qué nota vas a sacar?



WUOLAH

Informática Gráfica



Comparte estos flyers en tu clase y consigue más dinero y recompensas



Banco de apuntes de la

WUOLAH

1 Imprime esta hoja

2 Recorta por la mitad

3 Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes

4 Llévate dinero por cada descarga de los documentos descargados a través de tu QR



Para calcular esta componente tendremos en cuenta el ángulo (β) que forma el rayo reflejado (R) con la posición del observador (O). La expresión sería la siguiente: $R \cdot O = \cos^n(\beta)$. Podemos ver que según la expresión, la máxima reflexión se obtendrá cuando el ángulo sea 0 y mínima cuando el ángulo sea 90 grados.

- **Componente ambiental:** La componente ambiental podemos modelar una especie de flujo de luz constante que viene de todas direcciones. Este componente se forma a partir de varias reflexiones de los rayos de luz de los diferentes objetos de la escena. Conforme los rayos se reflejan, el objeto irá adquiriendo la tonalidad del material que tenga aplicado.

Podemos hacer los cálculos del modelo de iluminación en cualquier sistema de coordenadas del modelo, aunque tendríamos que aplicar a cada uno sus ajustes correspondientes.

TEXTURAS

A efectos prácticos, una textura es como un envoltorio que ponemos sobre un modelo. Los pasos a seguir para aplicar una textura son los siguientes:

- Pasamos la imagen (que está en forma matricial), a un sistema de coordenadas normalizado, con coordenadas u y v ($0 \leq v \leq 1$ y $0 \leq u \leq 1$). Con esto conseguimos independizar el tamaño real de la imagen de su aplicación en el modelo.
- Asignamos a cada punto del modelo las coordenadas de textura que le correspondan. Podemos realizar esta asignación de dos formas:
 1. Asignación explícita: las coordenadas son un dato de entrada en forma de vector o tabla de coordenadas de vértices. Puede hacerse manualmente (como en el dado de prácticas) o de forma aislada, haciendo uso de algoritmos de parametrización (blender).
 2. Asignación procedural: implementamos una función auxiliar que nos devuelva/calculé los valores de u y v para un punto determinado.

PROYECCIONES

La idea es mostrar objetos que se definen en 3 dimensiones, en dispositivos que solo permite mostrarlo de forma bidimensional. Con lo cual necesitamos la proyección, que se encarga de transformar objetos en 3 dimensiones a 2 dimensiones.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

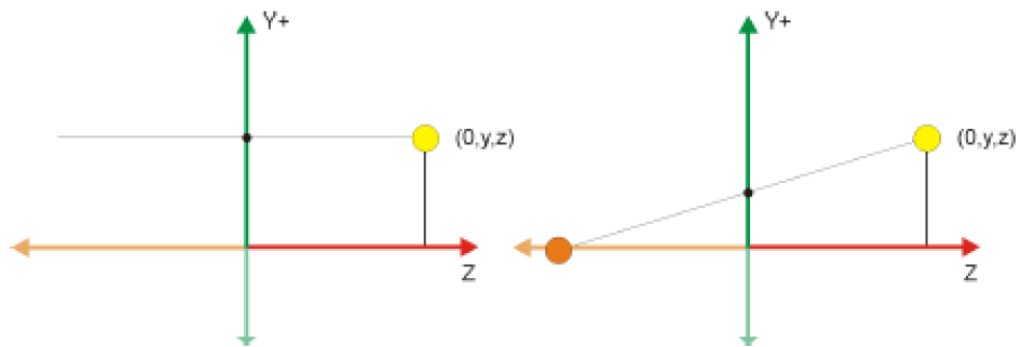
Hay 2 tipos de proyección:

- Proyección **perspectiva**: Es la que muestra de **forma más natural** los objetos, como el funcionamiento del ojo humano. Una de sus principales características es el acortamiento perspectivo, es decir **los objetos que se ven a lo lejos se ven más pequeños que los objetos que están más cerca**. En este caso los proyectores son líneas rectas que unen cada vértice con el centro de proyección (CP), dicho de otro modo todos los vértices convergen en el centro de proyección (CP) en un punto finito. Dado un plano de proyección, la proyección es la intersección de los proyectores con el plano de proyección. (frustum: necesita los valores de la ventana, la distancia con respecto al origen del plano delantero y la distancias con respecto al origen del plano trasero

```
glFrustum(X_MIN,X_MAX,Y_MIN*relacion_esp,Y_MAX*relacion_esp,FRONT_PLANE_PERSPECTIVE,BACK_PLANE_PERSPECTIVE);
```

- La proyección **paralela**: en esta **por muy lejos que esté el objeto siempre se ve del mismo tamaño**, ya que el centro de proyección converge en el infinito. (ortho:

```
glOrtho(X_MIN*mult,X_MAX*mult,Y_MIN*mult*relacion_esp,Y_MAX*mult*relacion_esp,FRONT_PLANE_PERSPECTIVE,BACK_PLANE_PERSPECTIVE);
```



Proyección paralela

Proyección perspectiva

En realidad la proyección paralela no existe, pero se usa para el dibujo técnico y otro tipo de reproducciones en los que no queremos que se cambien las medidas con respecto a la distancia.

Los planos de corte, como su propio nombre indica, se encargan de recortar el objeto que está por delante del panel delantero y/o detrás del panel trasero, esto nos evita uno de los principales problemas, la distorsión topológica y evitar cargar un objeto complejo en unos pocos píxeles cuando el objeto esté demasiado lejos.

WUOLAH

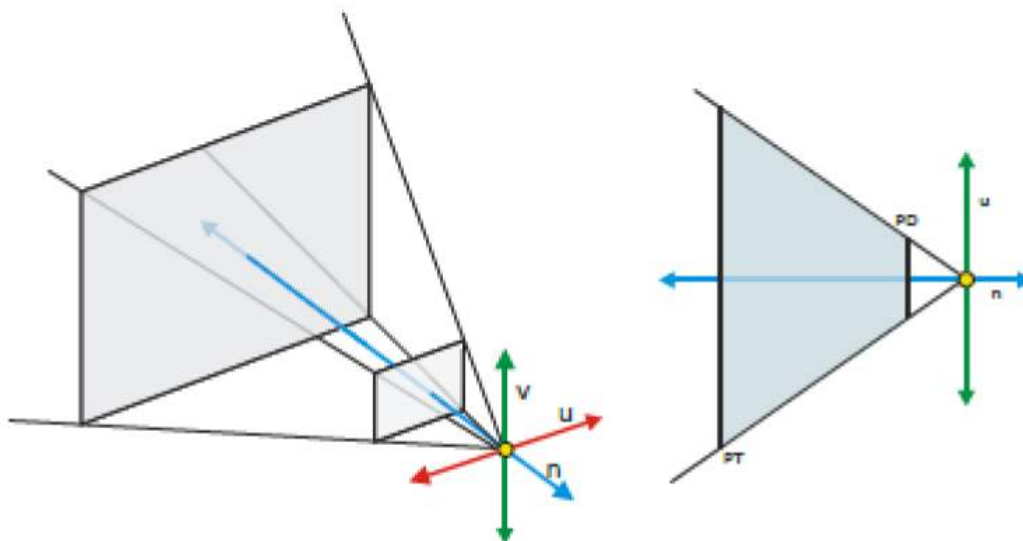


Figura 6.2: Frustum

En OpenGL hay que definir el volumen de visión, es decir todo el espacio tridimensional que podemos ver. En OpenGL se representa con una ventana rectangular, a parte hay que indicar el panel frontal y el panel trasero. Para definir la ventana necesitamos 4 valores, es decir las coordenadas (x,y) de las esquinas inferior izquierda y superior derecho del panel delantero. Y los paneles delantero y trasero son 2 valores más (Z), en total 6 valores para definir el volumen de visión, esto sirve tanto para la proyección perspectiva como la paralela.

TEMA 4

PICK (Identificación por color)

Una selección es dado el conjunto de elementos de la escena, queremos poder seleccionar individualmente cada uno de ellos, para aplicarle algún tipo de proceso (borrarlo, moverlo, etc). Para ello tenemos que asociar un **identificador a cada triángulo** de cada figura en la escena, para ello convertimos ese identificador en un color y también a la inversa de un color a un identificador. Con lo cual solo tenemos que pintar el objeto con el color correspondiente de cada identificador de cada triángulo en la escena.

Cuando seleccione el ratón un triángulo de una figura obtenga el color de la misma y lo convierta en un identificador para que le podamos aplicar el cambio de color. Al usar el modelo **RGB** tenemos que tener en cuenta que cada componente tiene 1 byte, es decir 256 tonalidades. De tal forma que para el azul van las posiciones del 0 al 255, para el verde del 256 hasta 65535 y el rojo desde 65536 hasta el 16777215, con lo cual solo nos falta saber que valor corresponde a cada componente RGB, para ello dividimos el número entre 65536, siendo el cociente de la componente roja, el resto de esa división se divide entre 256, siendo el cociente para la componente verde y el resto para la componente azul. Solo nos falta saber qué identificador tiene cada triángulo de cada figura, es tan simple como obtener la posición de cada triángulo del vector.

Pero si cada triángulo de la figura tiene un color diferente, la figura a mostrar se verá con múltiples colores, para que la figura se vea de un solo color, pero pueda identificar cada triángulo con un color distinta usamos el **framebuffer**, es una zona de memoria donde se almacena la imagen final, no solo para visualizarla, sino también para realizar la selección, con esto no vamos a representar de forma visual el objeto, sino la codificación por posición (cada triángulo de un color). Al usar el framebuffer vamos a ver en la pantalla la visualización de la codificación solo un instante y después la visualización normal. La solución para evitar esto es utilizar otro framebuffer.

Con lo cual habrá que tener un framebuffer principal que será el que visualiza el objeto en pantalla y el resto de buffers que se puedan crear no se visualizan por pantalla, a estos buffers se les llama off-screen.

Para ello tenemos que crear un framebuffer que tendrá asociado una zona de memoria para dibujar y otra zona de memoria para usar el z-buffer.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

perdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

4. Explique los diferentes métodos que se pueden usar para realizar la selección o pick (2ptos).

Hay tres formas de realizar la selección:

a) Identificación por color

A cada objeto que se quiere identificar se le asigna un identificador, un número natural. Este número es convertido a un color. Se activa la eliminación de partes ocultas. Cuando se dibuja el objeto se usa el color que tiene asociado. Al mover el cursor y pulsar para realizar la selección se guardan las coordenadas x e y del píxel seleccionado. Se lee el píxel del buffer en la posición x e y. Se convierte el color al identificador.

En el caso de representar el color con el modelo RGB y 24 bits, se tienen la posibilidad de identificar $2^{24}-1$ objetos diferentes. El color blanco implica que no se ha seleccionado nada.

Para pasar del identificador al color se usan máscaras de bits para obtener cada parte. Para pasar de color a ID se hacen los pasos inversos.

b) Lanzando un rayo

La idea básica es que cuando pulso el botón del ratón voy a seleccionar el objeto más cercano que está en la posición del cursor. Para ello, obtenemos la posición x e y del cursor en coordenadas de dispositivo y las convertimos a coordenadas de vista. Hacemos pasar una línea recta (o rayo) por el centro de proyección y la nueva posición. Esto es, obtenemos la ecuación de una recta. Calculamos la intersección con los objetos. Si hay intersección se añade a la lista, guardando el identificador del objeto y la profundidad. Por último, ordenamos en profundidad y devolvemos el identificador de la intersección más cercana.

c) Por ventana

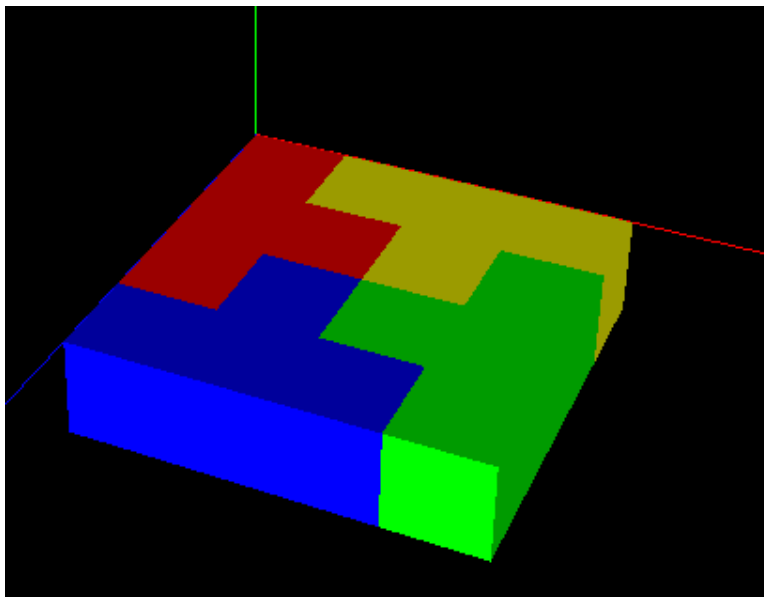
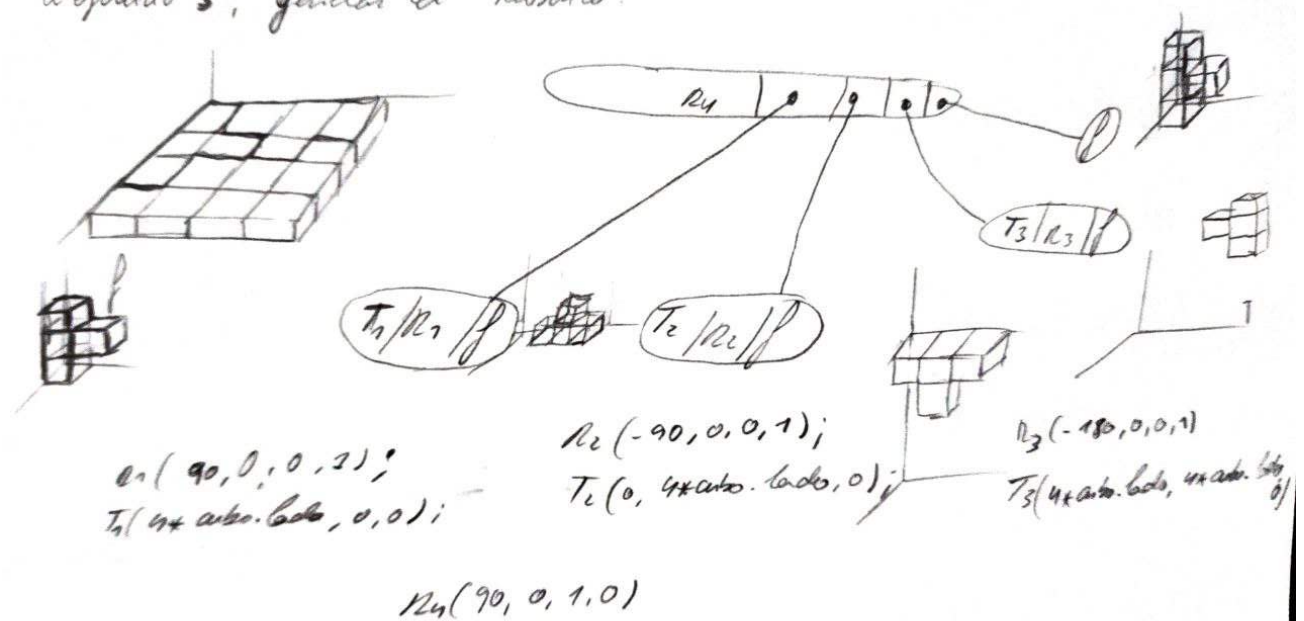
Este método consiste en aprovechar la etapa de discretización del cauce visual. Esto es, cuando pasamos de fórmulas (líneas, triángulos, etc) a píxeles. La idea consiste en que cuando se marca la posición con el ratón, se obtiene una posición x e y. Alrededor de dicha posición se crea una pequeña ventana (normalmente unos pocos píxeles). Una vez identificados los píxeles que conforman la ventana, lo único que hay que hacer es dibujar cada objeto, al cual se le asigna un identificador. Si al convertir el objeto en píxeles, coincide con alguno o varios de la ventana, entonces hay selección. Se guarda el identificador del objeto y la profundidad. Finalmente podemos hacer una ordenación por profundidad y quedarnos con el identificador del más cercano.

EJERCICIOS VARIOS

7- Generar el modelo necesario para obtener un mosaico de la figura 5, partiendo de la figura generada en el ejercicio 3.

WUOLAH

⑦ Partiendo de la figura generada en el ejercicio 3, generar el mosaico.



```
// ejercicio 7
glRotatef(90,1,0,0);
```

```
glPushMatrix();
cubo.color[0] = 0.0;
cubo.color[1] = 1.0;
cubo.color[2] = 0.0;
cubo.color[3] = 0.0;
glTranslatef(4*cubo.lado, 4* cubo.lado, 0);
glRotatef(-180, 0, 0,1);
ejercicio3_ejercicios();
glPopMatrix();
```

```
glPushMatrix();
cubo.color[0] = 0.0;
cubo.color[1] = 0.0;
cubo.color[2] = 1.0;
cubo.color[3] = 0.0;
glTranslatef(0, 4* cubo.lado, 0);
glRotatef(-90, 0, 0,1);
ejercicio3_ejercicios();
glPopMatrix();
```

```
glPushMatrix();
cubo.color[0] = 1.0;
cubo.color[1] = 1.0;
cubo.color[2] = 0.0;
cubo.color[3] = 0.0;
glTranslatef(4 * cubo.lado, 0, 0);
glRotatef(90, 0, 0,1);
ejercicio3_ejercicios();
glPopMatrix();
```

```
cubo.color[0] = 1.0;
cubo.color[1] = 0.0;
cubo.color[2] = 0.0;
cubo.color[3] = 0.0;
ejercicio3_ejercicios();
```


Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio

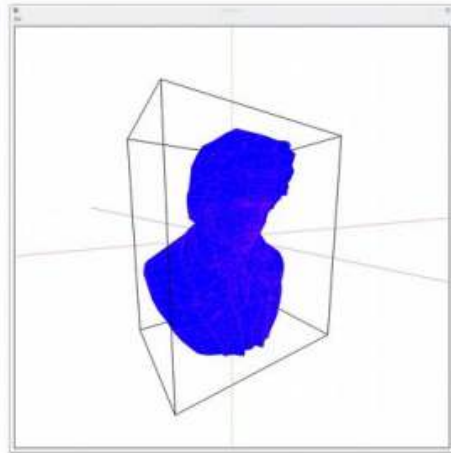
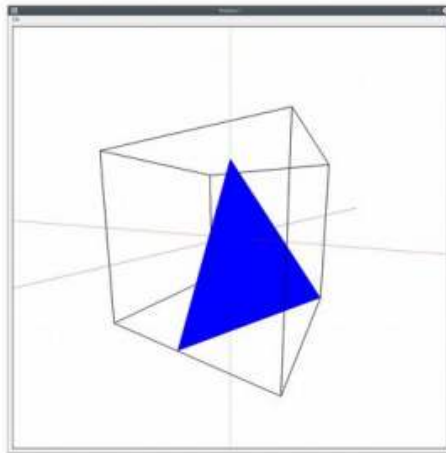


Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

2. Dado un objeto 3D que se define con definido por sus vértices, `vector<_vertex3f> Vertices`, y triángulos, `vector<_vertex3ui> Triangles`, implemente mediante pseudocódigo o C++ el programa que calcularía la caja frontera (la caja frontera es el menor paralelepípedo que incluye a la figura; basta con salvar el vértice con los coordenadas menores y el vértice con las mayores coordenadas) (2ptos).



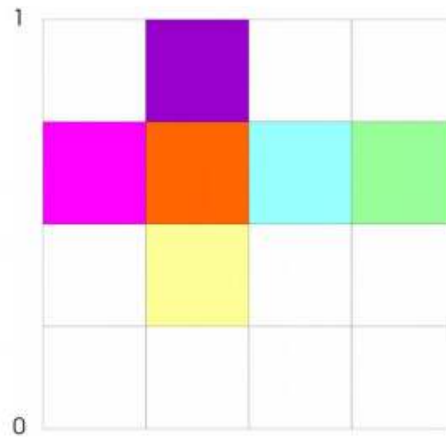
```
Pos_min=_vertex3f(1e8,1e8,1e8);
Pos_max=_vertex3f(-1e8,-1e8,-1e8);

for (unsigned int i=0;i<Vertices.size();i++)
{
    If (Vertices[i].x<Pos_min.x)
        Pos_min.x=Vertices[i].x;
    If (Vertices[i].y<Pos_min.y)
        Pos_min.y=Vertices[i].y;
    If (Vertices[i].z<Pos_min.z)
        Pos_min.z=Vertices[i].z;

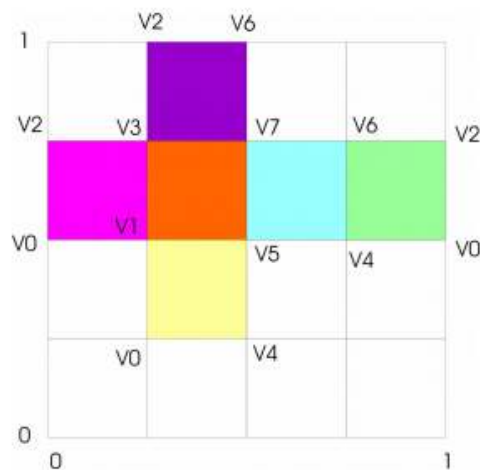
    If (Vertices[i].x>Pos_max.x)
        Pos_max.x=Vertices[i].x;
    If (Vertices[i].y>Pos_max.y)
        Pos_max.y=Vertices[i].y;
    if (Vertices[i].z>Pos_max.z)
        Pos_max.z=Vertices[i].z;
}
```

WUOLAH

3. Tenemos un cubo definido por sus 8 vértices y 12 triángulos y queremos aplicarle la textura de esta manera (ver imagen). Indicar si se puede hacer o no. En caso de que no se pueda, exponer cómo se resolvería. Indicar los valores de las coordenadas de textura para cada vértice (3ptos).



Si nombramos cada uno de los vértices tal que:



Como se puede ver en la imagen, el problema es que un mismo vértice, y por tanto sus coordenadas de textura, aparecen repetidos. Por ejemplo, el V0 se repite 3 veces. Eso implica que la posición del vértice es la misma, pero para cada cara tiene unas coordenadas de textura.

Por tanto, la solución pasa por repetir los vértices y tener una descripción para los triángulos que usa esos vértices nuevos. Para cada uno de estos nuevos vértices debe haber un elemento que guarde las coordenadas de textura correspondientes.

Por ejemplo, tendríamos

- V0a \rightarrow (0,0.5)
- V0b \rightarrow (0.25,0.25)
- V0c \rightarrow (1,0.5)

De la misma forma habría que poner las coordenadas para cada vértice del cubo.

5. ¿Qué es la transformación de vista?

La transformación de vista es una transformación que permite cambiar de sistema de coordenadas. Esta transformación permite simular el posicionamiento de la cámara en cualquier posición y orientación aplicando transformaciones geométricas (traslaciones, rotaciones, etc.).

6. Indique qué parámetros de la cámara están implicados en la transformación de vista y cómo se usan para obtener la transformación de la vista.

VRP: Posición donde está la cámara.

VPN: Hacia donde mira la cámara.

VUP: Indica la orientación hacia arriba.

7. ¿Qué matriz de OpenGL almacena la transformación de vista?

GL_MODELVIEW

8. Cree un ejemplo de transformación de vista incluyendo las llamadas de OpenGL.

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(0,0,-10);  
glRotatef(37,1,0,0);  
glRotatef(45,0,1,10);
```

9. Enumere y explique las propiedades de la transformación de perspectiva.

Acortamiento perspectivo: objetos más lejanos producen una proyección más pequeña.

Puntos de fuga: cualquier par de líneas paralelas convergen en un punto llamado punto de fuga.

Inversión de vista: los puntos que están detrás del centro proyección se proyectan invertidos.

Distorsión topológica: cualquier elemento geométrico que tenga una parte delante y otra detrás del centro proyección produce dos proyecciones semiinfinitas.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

10. Queremos realizar acercarnos a un objeto para ver sus detalles. Explicar cómo se podría hacer en una proyección de perspectiva.

La solución más sencilla consiste en acercarse al objeto. El problema está en que si no se cambia el plano delantero habrá un momento en el que se alcanza el objeto y lo recortará. Si colocamos la cámara en una posición donde los planos de corte no recorten el objeto, se puede hacer un zoom simplemente cambiando el tamaño de la ventana de proyección.

WUOLAH

12. Indica los pasos que hay que realizar en OpenGL y los elementos que intervienen y por tanto han de estar definidos para conseguir que una escena se vea iluminada.

Definir los vectores normales de cada cara: Es necesario definir un vector normal (perpendicular a la superficie apuntando hacia fuera de la parte visible) por cada uno de los vértices de nuestra representación.

Situar las luces: Para iluminar una escena será necesario situar las luces. OpenGL maneja dos tipos de iluminación:

- Luz ambiental: ilumina toda la escena por igual, ya que esta no proviene de una dirección determinada.
- Luz difusa: viene de una dirección específica, y depende de su ángulo de incidencia para iluminar una superficie en mayor o menor medida.

Definiendo materiales: OpenGL permite controlar la forma en que la luz se refleja sobre nuestros objetos, que es lo que se conoce como definición de materiales.

13. Explica el funcionamiento Z-Buffer.

El algoritmo del Z-buffer es del tipo espacio-imagen. Cada vez que se va a renderizar un píxel, comprueba que no se haya dibujado antes en esa posición un píxel que esté más cerca respecto a la cámara (posición en eje Z). Este algoritmo funciona bien para cualquier tipo de objetos: cóncavos, convexos, abiertos y cerrados. Cuando dibujamos un objeto, la profundidad de sus píxeles se guardan en este buffer. Si utilizamos dichos píxeles en pantalla para dibujar otro objeto, se producirá la comparación de las profundidades de dichos píxeles. Si la profundidad del último píxel es mayor que la nueva (está más lejos) el píxel nuevo no se pinta, mientras que si está más cerca (la profundidad es menor), se dibuja el píxel y se guarda la nueva profundidad en el z-buffer. Para activarlo, hay que hacer una llamada a

`glEnable(GL_DEPTH_TEST)`

Esta llamada le dice a OpenGL que active el test de profundidad. Además, cada vez que se redibuje la escena, aparte de borrar el buffer de color, hay que borrar el buffer de profundidad. Esto se hace con la llamada

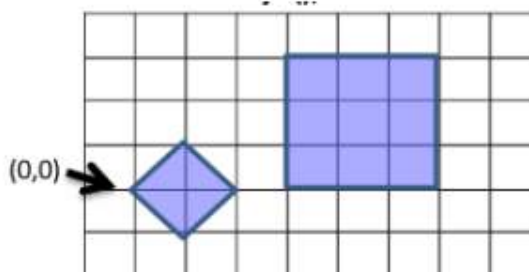
`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT).`

Por último, pero no menos importante, al inicializar OpenGL se le tiene que definir el buffer de profundidad en el modo de visualización.

14. ¿Qué es una partícula en Informática Gráfica? Describe su ciclo de vida. ¿Puedes poner algún ejemplo?

Las partículas son elementos lógicos a las que se debe otorgar propiedades gráficas para que sean visibles. Por ejemplo, cada partícula se puede sustituir por un objeto geométrico, gracias a lo cual es posible mostrar cualquier forma o dibujo. Con la adición del material correspondiente se pueden utilizar sistemas de partículas para mostrar humo, niebla o fuego.

15. Dada la figura de la izquierda indique qué transformaciones son necesarias para obtener la figura de la derecha. La figura se dibuja con la llamada a la función `cuadrado.dibujar()`.



```
glPushMatrix();
glTranslatef(3, 0, 0);
glScalef(3/12, 3/12, 1);
glRotatef(45, 0, 0, 1);
cuadrado.dibujar();
glPopMatrix();
```

16. ¿Cómo se calculan las normales a un vértice?

Podemos definir el vector normal de un vértice como el vector normal de un plano tangente al objeto en dicho vértice, pero esto es muy difícil de calcular. Por ello, realizamos un aproximación muy fiable: la suma de las normales de los triángulos adyacentes a dicho vértice. Para ello, calculamos las normales de todos los triángulos como el producto vectorial de dos de sus aristas, teniendo en cuenta que las normales tienen dirección hacia el exterior del objeto. Las normalizamos con la siguiente fórmula:

$$\frac{n_C}{\|N_C\|}$$

A continuación, sumamos para cada cara, el valor de su normal a la normal de cada uno de sus vértices. Una vez acabado, normalizamos las normales de vértices.

$$\frac{n_V}{\|N_V\|}$$

17. Describa las transformaciones de vista que se pueden aplicar a una cámara.

Las transformaciones de vista son `glFrustum` y `glOrtho`. La declaración de estas transformaciones es:

- `glFrustum (left, right, bottom, top, near, far);`
- `glOrtho (left, right, bottom, top, near, far);`

La función `glFrustum` describe una matriz de perspectiva que produce una proyección en perspectiva.

La función `glOrtho` describe una matriz de perspectiva que produce una proyección paralela.

En ambas, la matriz actual se multiplica por esta matriz y el resultado reemplaza a la matriz actual.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

perdo
espacio



18. Describe brevemente para que sirven, en un programa OpenGL/glut, cada una de estas cuatro funciones:

- glutDisplayFunc(funcion); Esta función se llamará cada vez que se dibuje la ventana.
- glutReshapeFunc(funcion); Función de control del cambio de tamaño de la ventana de visualización.
- glutKeyboardFunc(funcion); Función de control de eventos con el teclado.
- glutSpecialFunc(funcion); Función de control de eventos con el teclado para cuando se ha pulsado una tecla especial.

19. Explique el sistema de colores que se usa en OpenGL.

El sistema de color empleado en OpenGL es el sistema RGB. RGB significa los colores rojo, verde y azul: los colores primarios aditivos. A cada uno de estos colores se le asigna un valor, en OpenGL generalmente un valor entre 0 y 1. El valor 1 significa la mayor cantidad posible de ese color, y 0 significa ninguna cantidad de ese color. Podemos mezclar estos tres colores para obtener una gama completa de colores.

20. Sobre las proyecciones. Indicar si es verdadero V o falso F.

- La proyección de perspectiva acorta los objetos más lejanos (V).
- Dos líneas paralelas en el modelo sólo fugan si no son paralelas al plano de proyección (F).
- El vector Z del sistema cartesiano del observador (punto de mira punto del observador) y el vector de inclinación pueden tener cualquier orientación (V).
- El vector de inclinación siempre coincide con el eje Y del observador (F)
- La ventana debe estar centrada para que se pueda realizar la proyección (F)
- En algunos casos es obligatorio poner el plano delantero detrás del plano trasero (F)
- Si un objeto tiene todos sus vértices detrás del centro de proyección, no se podrá proyectar correctamente (V)
- En una proyección paralela el zoom se puede implementar moviendo los planos de corte (V)

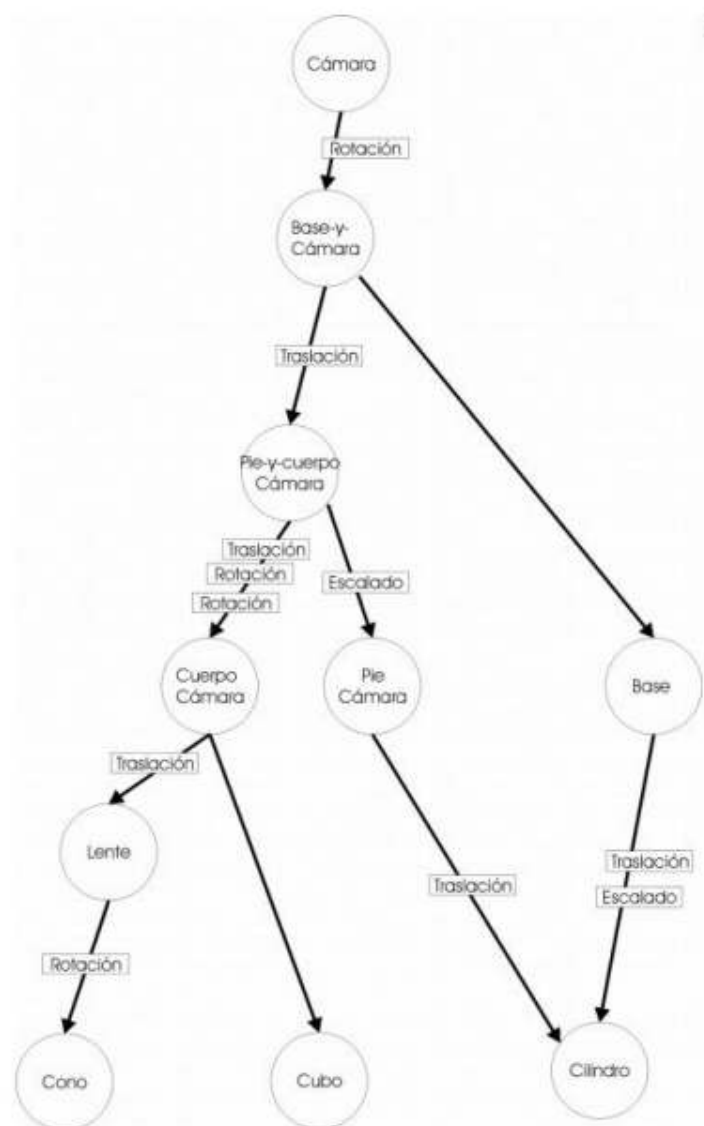
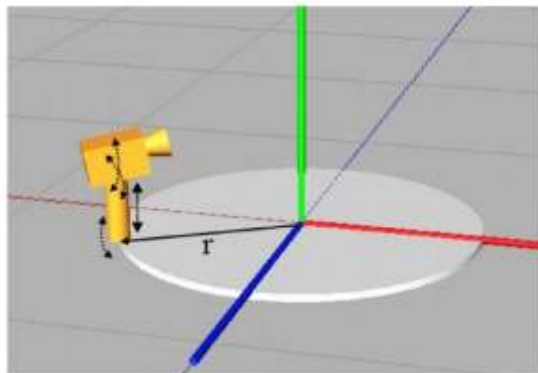
21. Generar el grafo de escena incluyendo las transformaciones, tal que, partiendo de un cubo, un cilindro y un cono unidad, se pueda realizar el modelo de una cámara de TV situada en el borde de una plataforma de radio r. La plataforma (cilindro) rota con respecto al eje Y, la cámara rota toda ella con respecto a su base (cilindro), y permite subir y bajar, y el cuerpo de la cámara (cubo) y el objetivo (cono) rotan arriba abajo e izquierda derecha. Las medidas de las partes del modelo se dejan a discreción. Implementar el modelo en C++ y OpenGL. (nota: se han dibujado los ejes cartesianos como cilindros y no forman parte del modelo) (2.5 pt).

Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

WUOLAH



EXÁMENES RESUELTOS CON CHATGPT

ENERO 2022

1. Explique, lo más detallado que pueda, las distintas formas de realizar un pick en OpenGL (2 puntos).

En OpenGL existen varios métodos para realizar la selección o pick de un objeto en una escena 3D. Algunos de los métodos más comunes son:

1. Selección por color: Este método consiste en asignar un color único a cada objeto en la escena y luego leer el color del píxel en la posición del cursor del mouse. El objeto que se encuentra en esa posición se selecciona.
2. Selección por nombre: Este método consiste en asignar un nombre único a cada objeto en la escena y luego utilizar la función `glLoadName()` para seleccionar el objeto en la posición del cursor del mouse.
3. Selección por Z-Buffer: Este método consiste en utilizar el Z-Buffer de OpenGL para identificar el objeto que se encuentra en la posición del cursor del mouse. El Z-Buffer almacena la profundidad de cada píxel en la escena, y el objeto con el valor más cercano al plano de la cámara se selecciona.
4. Selección por ray-casting: Este método consiste en lanzar un rayo desde la posición del cursor del mouse hacia la escena 3D y determinar si el rayo intersecta con algún objeto. El objeto con la intersección más cercana se selecciona.

Cada uno de estos métodos tiene sus ventajas y desventajas y pueden ser más adecuados según las necesidades específicas del proyecto. Es importante también considerar el rendimiento y la complejidad de cada método al elegir cuál utilizar.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

2. Describa el flujo de transformaciones que se realizan en OpenGL desde que proporcionamos las coordenadas 3D de un modelo hasta que tenemos una imagen en pantalla. Indique el propósito de cada etapa y el resultado obtenido tras cada una de las transformaciones (2 puntos).

En OpenGL, el flujo de transformaciones que se realizan desde que proporcionamos las coordenadas 3D de un modelo hasta que tenemos una imagen en pantalla es el siguiente:

1. **Transformación modelo:** En esta etapa, se transforman las coordenadas 3D del modelo para situarlo en el espacio de la escena. Se utilizan las funciones `glTranslatef()`, `glRotatef()` y `glScalef()` para realizar esta transformación. El objetivo de esta etapa es situar el modelo en el espacio de la escena para que pueda ser visto desde la perspectiva de la cámara.
2. **Transformación de vista:** En esta etapa, se transforman las coordenadas del modelo para situarlas en la perspectiva de la cámara. Se utiliza la función `gluLookAt()` para realizar esta transformación. El objetivo de esta etapa es simular la posición y orientación de la cámara en la escena.
3. **Transformación de proyección:** En esta etapa, se transforman las coordenadas del modelo para proyectarlas en el plano de la pantalla. Se utiliza la función `glFrustum()` o `glOrtho()` para realizar esta transformación. El objetivo de esta etapa es simular la perspectiva de una cámara real y asegurar que los objetos lejanos se vean más pequeños que los objetos cercanos.
4. **Transformación de vista-pantalla:** Última etapa del proceso de transformación que se realiza antes de mostrar una escena en pantalla. La salida de esta transformación son las coordenadas 2D que son utilizadas para dibujar los objetos en la pantalla, así como también la profundidad de cada punto, que es almacenada en el Z-Buffer.

WUOLAH

3. Describe la estructura de aristas aladas, y explica (apoyando la explicación con pseudocódigo si lo ves conveniente) el proceso que se debe seguir para construir una estructura de aristas aladas a partir de un archivo PLY (2 puntos).

Una estructura de aristas aladas es una representación de un modelo 3D que se compone de un conjunto de triángulos y aristas. Esta estructura se utiliza para facilitar la manipulación de un modelo 3D, ya que permite acceder fácilmente a las caras y vértices de un objeto.

La estructura de una arista alada se compone de los siguientes elementos:

- **Vértices:** Son los puntos que definen las posiciones de los objetos en el espacio.
- **Aristas:** Son las líneas que conectan los vértices entre sí.
- **Caras:** Son los triángulos que se forman entre los vértices conectados por las aristas.

4. GRAFO (4 puntos).

ENERO 2018

2. Una técnica para reducir el número de caras de un modelo es el colapso de aristas. Dado un objeto descrito con vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos), escribir en pseudocódigo el programa que lo realizaría (2 puntos).

1. Inicializar una estructura de datos para almacenar las nuevas caras y vértices del objeto
2. Recorrer cada cara del objeto
3. Para cada cara, recorrer cada arista de la cara
4. Si la arista ya ha sido procesada, omitir el paso 5
5. Si la arista no ha sido procesada, calcular el vértice medio de la arista
6. Agregar el vértice medio al conjunto de vértices
7. Reemplazar la arista original en la cara actual con el vértice medio y agregar la nueva cara formada por los vértices adyacentes al vértice medio
8. Marcar la arista como procesada
9. Repetir los pasos 2 a 8 hasta que no hayan más caras que procesar
10. Reemplazar el conjunto de vértices y caras del objeto original con los nuevos conjuntos de vértices y caras

3. Las lentes de las cámaras con zoom permiten cambiar la zona visible, desde ángulos más grandes (gran angular) a más pequeños (tele). ¿Cómo se podría conseguir el mismo efecto con los parámetros del `glFrustum`? Explicarlo y poner ejemplos de valores (2 puntos).

El `glFrustum` es una función de OpenGL que se utiliza para definir la proyección de una escena 3D en el plano de la pantalla. Los parámetros que se utilizan en `glFrustum` son el `left`, `right`, `bottom`, `top`, `near` y `far`, que indican los límites de la zona visible en el plano de la pantalla.

Para conseguir el mismo efecto que el zoom de una cámara, se puede modificar los valores de los parámetros `left`, `right`, `bottom` y `top`. Si se quiere aumentar el zoom (ver una zona más pequeña), se deben reducir los valores de los parámetros `left`, `right`, `bottom` y `top`, lo que hará que la zona visible sea más pequeña. Por otro lado, si se quiere disminuir el zoom (ver una zona más grande), se deben aumentar los valores de los parámetros `left`, `right`, `bottom` y `top`, lo que hará que la zona visible sea más grande.

Un ejemplo de valores para aumentar el zoom sería:

```
glFrustum(-1.0, 1.0, -0.5, 0.5, 1.0, 100.0);
```

Por otro lado, un ejemplo de valores para disminuir el zoom sería:

```
glFrustum(-2.0, 2.0, -1.0, 1.0, 1.0, 100.0);
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH



4. Explique los diferentes métodos que se pueden usar para realizar la selección o pick (1 punto).

Ya explicado en enero 2022.

5. Explica el funcionamiento del Z-buffer (1 punto).

El Z-buffer, también conocido como buffer de profundidad, es una técnica utilizada en la computación gráfica para determinar qué píxeles deben ser dibujados en una escena 3D. El Z-buffer es una estructura de datos que almacena la profundidad de cada píxel en la escena. Cada vez que se dibuja un objeto en la escena, se comprueba el valor de profundidad del píxel en el Z-buffer antes de dibujar el nuevo píxel.

El funcionamiento del Z-buffer se basa en el hecho de que los objetos más cercanos a la cámara deben ser dibujados sobre los objetos más lejanos. Por lo tanto, cuando se dibuja un objeto, se comprueba el valor de profundidad del píxel en el Z-buffer y se dibuja el nuevo píxel solo si es más cercano a la cámara que el píxel existente.

El Z-buffer es muy eficiente en resolver los problemas de superposición de objetos y se utiliza ampliamente en la computación gráfica para asegurar que los objetos se dibujen correctamente. Sin embargo, tiene un costo en términos de memoria ya que requiere almacenar la profundidad de cada píxel en la escena.

WUOLAH

6. Dado el siguiente modelo de iluminación local describe cada uno de sus elementos y cómo influyen sus parámetros en la iluminación de una superficie dada. (1 punto)

$$I_{\lambda} = I_{a\lambda} + I_{L\lambda} (k_{d\lambda} (\vec{N} \cdot \vec{L}) + k_s (\vec{R} \cdot \vec{V})^n)$$

Este modelo de iluminación local se conoce como modelo de Phong, y se compone de los siguientes elementos:

- $I_{a\lambda}$: Es la iluminación ambiental, que es la luz que proviene de todas las direcciones y es reflejada de manera uniforme en todas las superficies. Este parámetro afecta el color base de una superficie.
- $I_{L\lambda}$: Es la iluminación de una fuente de luz puntual, que es la luz que proviene de una dirección específica. Este parámetro afecta la iluminación directa en una superficie.
- $k_{d\lambda}$: Es el coeficiente de difusión, que indica qué porcentaje de la luz incidente es reflejado de manera difusa por una superficie. Este parámetro afecta la iluminación difusa en una superficie.
- $\vec{N} \cdot \vec{L}$: Es el producto escalar entre el vector normal de una superficie y el vector de luz, que indica la dirección de la luz incidente en un punto específico. Este parámetro afecta la iluminación difusa en una superficie.
- k_s : Es el coeficiente de especularidad, que indica qué porcentaje de la luz incidente es reflejado de manera especular por una superficie. Este parámetro afecta la iluminación especular en una superficie.
- $(\vec{R} \cdot \vec{V})^n$: Es el producto escalar entre el vector de reflexión de la luz incidente y el vector de vista, elevado a la potencia n, que indica la dirección de la luz reflejada en un punto específico. Este parámetro afecta la iluminación especular en una superficie.

En resumen, este modelo de iluminación local combina los efectos de la iluminación ambiental, la iluminación directa y la iluminación difusa y especular en una superficie. Los parámetros del modelo controlan la cantidad de luz reflejada por una superficie y cómo se distribuye la luz reflejada en función de la dirección de la luz incidente y la dirección de la vista.

ENERO 2021

1. ¿Qué es la transformación de vista?

La transformación de vista en gráficos 3D es un proceso mediante el cual se mapea la escena 3D en el espacio de la cámara en el espacio 2D de la pantalla. Es una de las etapas fundamentales en el pipeline de renderizado en OpenGL.

La transformación de vista se utiliza para cambiar la posición y orientación de la cámara en relación con la escena, permitiendo crear diferentes perspectivas y puntos de vista. También se utiliza para definir el ángulo de visión de la cámara y la distancia de la cámara al plano de proyección.

¿Qué parámetros de la cámara están implicados?

- **Posición de la cámara:** indica la posición en el espacio 3D donde se encuentra la cámara.
- **Dirección de la cámara:** indica la dirección hacia la cual está apuntando la cámara.
- **Vector de vista:** indica la dirección hacia la cual está orientado el plano de proyección de la cámara.
- **Ángulo de visión (o campo de visión):** indica el ángulo que abarca la cámara en el plano de proyección.
- **Relación de aspecto:** indica la relación entre el ancho y la altura del plano de proyección de la cámara.
- **Planos de recorte:** indican los planos que delimitan la zona visible de la escena desde la cámara.

¿Qué matriz de OpenGL almacena dicha transformación?

La matriz de OpenGL que almacena la transformación de vista es la matriz de modelview. La matriz de modelview es una combinación de la matriz model y la matriz view, se encarga de combinar la información de la geometría del modelo y la posición de la cámara en el espacio.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

Cree un ejemplo incluyendo las llamadas de OpenGL.

```
// Inicializar la matriz de modelview
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// Definir la posición de la cámara
gluLookAt(0, 0, 2, // Posición de la cámara
          0, 0, 0, // Punto hacia el que apunta la cámara
          0, 1, 0); // Vector de vista

// Aplicar una traslación a la escena
glTranslatef(1, 0, 0);

// Dibujar el modelo
drawModel();
```

2. Enumere y explique las propiedades de la transformación de perspectiva.

La transformación de perspectiva es una técnica que se utiliza para simular la perspectiva realista en una imagen generada por ordenador. Esta transformación tiene varias propiedades importantes que son las siguientes:

1. Punto de fuga: es el punto en el espacio 3D hacia el cual converge todo el contenido de la escena. Todo lo que se encuentra en el plano de la cámara se proyecta en este punto en el plano de proyección.
2. Plano de proyección: es el plano en el que se proyectan los vértices de la escena después de pasar por la transformación de perspectiva. Este plano suele situarse en una distancia determinada de la cámara, llamada distancia de enfoque.
3. Distancia de enfoque: es la distancia desde la cámara hasta el plano de proyección. A medida que se aumenta esta distancia, el ángulo de visión se reduce y los objetos se ven más pequeños y lejanos.
4. Ángulo de visión: es el ángulo que abarca la cámara en el plano de proyección. Este ángulo se mide en grados y suele ser configurable. A medida que se aumenta el ángulo de visión, los objetos se ven más grandes y cercanos.
5. Relación de aspecto: es la relación entre el ancho y la altura del plano de proyección. Esta relación se suele ajustar para evitar distorsiones en la imagen final.
6. Planos de recorte: son los planos que delimitan la zona visible de la escena desde la cámara. Cualquier objeto que se encuentre fuera de estos planos no será visible en la imagen final.

WUOLAH

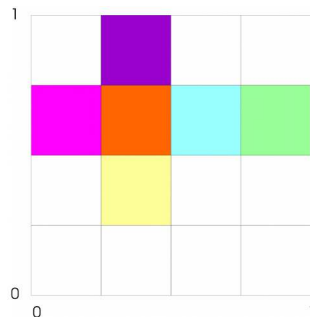
3. Queremos realizar acercarnos a un objeto para ver sus detalles. Explicar cómo se podría hacer en una proyección de perspectiva, ventajas e inconvenientes.

Para acercarnos a un objeto en una proyección de perspectiva, existen varias técnicas que se pueden utilizar, a continuación explicaré algunas de ellas:

1. **Aumentar la distancia de enfoque:** Una de las formas más comunes de acercarnos a un objeto es aumentando la distancia de enfoque, es decir, acercando el plano de proyección al objeto. Al hacer esto, los objetos se ven más grandes y cercanos en la imagen final. Sin embargo, esto también reduce el ángulo de visión de la cámara, lo que puede hacer que los objetos fuera del plano de proyección se vean muy pequeños y distorsionados.
2. **Reducir el ángulo de visión:** Otra forma de acercarnos a un objeto es reduciendo el ángulo de visión de la cámara. Esto hace que los objetos se vean más grandes y cercanos en la imagen final, pero también reduce la cantidad de escena que se ve.
3. **Aumentar la escala del objeto:** Una forma de acercarnos a un objeto es aumentando su escala en el sistema de coordenadas. Esto hará que el objeto se vea más grande en la imagen final, pero también afectará a las proporciones de los objetos cercanos y puede generar distorsiones en la escena.
4. **Trasladar la cámara hacia el objeto:** Otra forma de acercarnos a un objeto es trasladando

4. Dado un cubo definido por sus vértices, `vector<_vertex3f> Vertices`, y triángulos, `vector<_vertex3ui> Triangles`, indique lo siguiente si queremos mostrar el cubo texturado:

- 1) La estructura de datos para guardar las coordenadas de textura
- 2) La función que dibujaría el objeto texturado
- 3) Un ejemplo de coordenadas de textura para cada vértice, si la textura se aplica a todas las caras sin repetirla (esto es, cada cuadrado NO muestra la textura completa)



- 1) Sería un vector para dos flotantes. Su tamaño sería igual al número de vértices
`Vector<_vertex2d> Vertices_texcoordinates;`

2)

```
glBegin(GL_TRIANGLES);  
  
for(unsigned int i=0; i<Triangles.size();i++){  
  
    glTexCoord2fv((GLfloat *) &Vertices_texcoordinates[Triangles[i]._0]);  
    glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);  
  
    glTexCoord2fv((GLfloat *) &Vertices_texcoordinates[Triangles[i]._1]);  
    glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);  
  
    glTexCoord2fv((GLfloat *) &Vertices_texcoordinates[Triangles[i]._2]);  
    glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);  
  
}  
  
glEnd();
```

3)

Si no tenemos en cuenta el problema de los puntos repetidos, bastaría con desplegar el cubo sobre la textura y asignar los valores de las coordenadas de textura correspondientes. Por ejemplo:

`Vertices_texcoordinates[0]=_vertex2f(0,0.5);`

`Vertices_texcoordinates[1]=_vertex2f(0.25,0.5);`

`Vertices_texcoordinates[2]=_vertex2f(0,0.75);`

`Vertices_texcoordinates[3]=_vertex2f(0.25,0.75);`

`Vertices_texcoordinates[4]=_vertex2f(0.75,0.5);`

`Vertices_texcoordinates[5]=_vertex2f(0.75,0.5);`

`Vertices_texcoordinates[6]=_vertex2f(0.5,0.75);`

`Vertices_texcoordinates[7]=_vertex2f(0.5,0.5);`

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

FEBRERO 2021

1. Explique el modelo de iluminación que se usa en OpenGL (2 puntos).

OpenGL utiliza un modelo de iluminación basado en la ecuación de Phong. Este modelo se basa en tres componentes básicos:

1. Iluminación ambiental: Es la luz que se refleja en todas las direcciones y que proviene de todas las fuentes de luz en la escena.
2. Iluminación difusa: Es la luz que se refleja de una superficie de manera difusa, esparciéndose en todas las direcciones.
3. Iluminación especular: Es la luz que se refleja de una superficie de manera especular, produciendo un reflejo brillante en un punto específico.

La ecuación de Phong combina estos tres componentes de iluminación para calcular el color de cada píxel en una superficie. Los parámetros utilizados en la ecuación de Phong son:

- I_a : Intensidad ambiental de la luz
- I_l : Intensidad de la luz
- k_d : Coeficiente de difusión de la superficie
- k_s : Coeficiente especular de la superficie
- n : Índice de brillo de la superficie
- L : Vector de dirección de la luz
- N : Vector normal de la superficie
- V : Vector hacia la cámara.

OpenGL proporciona funciones para establecer los parámetros de iluminación y materiales como la posición de la luz, la intensidad de la luz, los coeficientes de reflexión, entre otros, y se encarga de aplicar esta ecuación en tiempo de ejecución para calcular el color de cada píxel en la escena.

WUOLAH

2. Dado un objeto 3D que se define con definido por sus vértices, `vector<_vertex3f> Vertices`, y triángulos, `vector<_vertex3ui> Triangles`, implemente mediante pseudocódigo o C++ el programa que calcularía la caja frontera (la caja frontera es el menor paralelepípedo que incluye a la figura; basta con salvar el vértice con los coordenadas menores y el vértice con las mayores coordenadas) (2 puntos).

```
//Función para calcular la caja frontera
procedure calcularCajaFrontera(Vertices, Triangles, min, max)

    //Iniciando las variables min y max con los valores del primer vértice
    min = Vertices[0]
    max = Vertices[0]

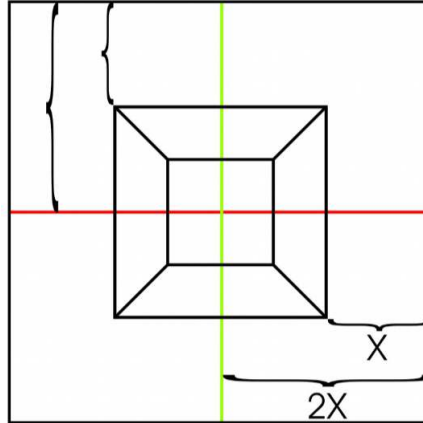
    //Recorriendo los vértices
    for i = 0 to Vertices.size()-1
        //Actualizando los valores de min y max
        if Vertices[i].x < min.x then min.x = Vertices[i].x
        if Vertices[i].y < min.y then min.y = Vertices[i].y
        if Vertices[i].z < min.z then min.z = Vertices[i].z

        if Vertices[i].x > max.x then max.x = Vertices[i].x
        if Vertices[i].y > max.y then max.y = Vertices[i].y
        if Vertices[i].z > max.z then max.z = Vertices[i].z

    end for

end procedure
```

3. Tenemos un cubo unitario centrado en el origen. La transformación de vista consiste en una traslación, $\text{glTranslate}(0,0,-2)$. Dado que el plano delantero está a una distancia 1, el plano trasero a una distancia 3, calcular los valores de la ventana ($x_{\min}, y_{\min}, x_{\max}, y_{\max}$), de tal manera que la proyección ocupe la mitad de la ventana (ver figura) (2 puntos).



Para calcular los valores de la ventana, necesitamos conocer el tamaño del cubo unitario y la distancia del plano de proyección.

Dado que el cubo es unitario, su tamaño es 1 en cada dimensión (x, y, z). La traslación es $(0, 0, -2)$, lo que significa que el plano trasero del cubo está a una distancia de 3 unidades de la cámara.

Para calcular la ventana, primero debemos calcular el tamaño del cubo en el plano de proyección. El tamaño del cubo en el plano de proyección se calcula dividiendo su tamaño en la dimensión correspondiente entre la distancia del plano de proyección y la cámara.

En este caso, el tamaño del cubo en el plano de proyección en el eje x es $1 / (3 - 0) = 0.5$. El tamaño del cubo en el plano de proyección en el eje y es también 0.5.

Dado que queremos que la proyección ocupe la mitad de la ventana, los valores de la ventana son:

- $x_{\min} = -0.25$
- $y_{\min} = -0.25$
- $x_{\max} = 0.25$
- $y_{\max} = 0.25$

Es importante tener en cuenta que estos son valores arbitrarios y dependen de la implementación y configuración de la ventana en particular.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio

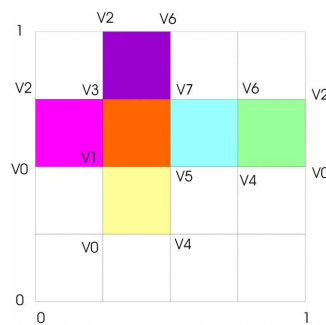


Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

4. Tenemos un cubo definido por sus 8 vértices y 12 triángulos y queremos aplicarle la textura de esta manera (ver imagen). Indicar si se puede hacer o no. En caso de que no se pueda, exponer cómo se resolvería. Indicar los valores de las coordenadas de textura para cada vértice (3 puntos).



Aplicar una textura a un cubo de esta manera se puede hacer ya que el cubo es un polígono con caras planas. Cada cara del cubo se puede asignar una textura diferente.

Para asignar una textura a un cubo, se debe especificar las coordenadas de textura para cada vértice del cubo. Las coordenadas de textura son valores que indican dónde se encuentra cada vértice en la textura.

En el caso de la imagen que proporcionas, se puede asignar las siguientes coordenadas de textura a cada vértice del cubo:

- Vértice 1: (1,1)
- Vértice 2: (0,1)
- Vértice 3: (0,0)
- Vértice 4: (1,0)
- Vértice 5: (0,1)
- Vértice 6: (1,1)
- Vértice 7: (1,0)
- Vértice 8: (0,0)

Dependiendo de la implementación y la librería que se esté utilizando, se pueden utilizar diferentes métodos para aplicar las coordenadas de textura a cada vértice del cubo.

Para aplicar esta textura se debería de utilizar los triángulos y las coordenadas de textura, esto se hace mediante un proceso conocido como "texturizado" que

WUOLAH

consiste en asociar coordenadas de textura a los vértices del modelo y luego enlazarlos con las caras del modelo.

En general, esta tarea puede ser compleja y requiere un conocimiento específico de la librería de gráficos que se esté utilizando, pero en general se puede hacer.

5. Explique los diferentes métodos que se pueden usar para realizar la selección o pick (2 puntos).

Ya explicado en enero 2022.