PREGUNTAS-FRECUENTES-EXAMENES-IN...

BrokenQuagga

Informática Gráfica

3º Grado en Ingeniería Informática

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

PREGUNTAS FRECUENTES EXAMENES INFORMÁTICA GRÁFICA

Explique, lo mas detallado posible, las distintas formas de realizar un pick.

Hay 3 formas de realizarlo:

- Identificación por color: A cada objeto que se quiere identificar se le asigna un identificador, un numero 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 pixel seleccionado. Se lee el pixel del buffer en la posición x e y. Se convierte al color seleccionado.
  En el caso de representar el color con el modelo RGB y 24 bits, se tiene la posibilidad de identificar 2^{24}-1 objetos diferentes. El blanco indica que no se selecciona 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

- Lanzado de un rayo: La idea básica es que cuando pulso un 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 del dispositivo y las convertimos a coordenadas de vista. Hacemos pasar una línea recta 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 ID del objeto y la profundidad. Por último, ordenamos en profundidad y devolvemos el ID de la intersección más cercana.

- Por ventana: Este método consiste en aprovechar la etapa de discretización del cauce visual. Esto es, cuando pasamos de fórmulas a píxeles. La idea consiste en que cuando se maraca la posición con el ratón, se obtiene una posición x e y. Alrededor de dicha posición se crea una pequeña ventana. 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 quedamos con el identificador del más cercano.

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.

1. Transformación del modelo: Situarlo en escena, cambiarlo de tamaño y crear modelos compuestos de otros más simples.

2. Transformación de vista: Poner al observador en la posición deseada.

3. Transformación de perspectiva: Pasar de un mundo 3D a una imagen 2D.

4. Rasterización: Calcular para cada pixel su color, teniendo en cuenta la primitiva que se muestra, su color, material, texturas, luces, etc.

5. Transformación del dispositivo: Adaptar la imagen 2D a la zona de dibujado.

Describa las formas de interacción de la luz como partícula según la superficie de los objetos y el proceso que sigue para realizar un suavizado de Gouraud.

Si la luz interactúa con una superficie pulida opaca, es una reflexión especular o regular. Si la luz interactúa con una superficie rugosa opaca, es una reflexión difusa.

El suavizado de Gouraud se realiza mediante las normales de los vértices para caluclar la intensidad de la luz. Luego se interpolan esas intensidad para encontrar los valores en los pixeles

en los que proyecta el polígono en pantalla.

¿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.).

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.

¿Qué matriz de OpenGL almacena la transformación de vista?
GL_MODELVIEW

WUOLAH + BBVA

1/6	BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.	

Te regalamos

15€

1

Abre tu Cuenta Online **sin comisiones ni condiciones**

2

Haz una compra igual o superior a 15€ **con tu nueva tarjeta**

3

BBVA te devuelve un **máximo de 15€**

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);
```

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.

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.

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 predeterminada.
- 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.

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

a) La estructura de datos para guardar las coordenadas de textura
Sería un vector para vértices en dos dimensiones. Su tamaño sería igual al número de vértices:
Vector<_vertex2d> Vertices_texcoordinates;
Vertices_texcoordinates.resize(vertices.size());

b) La función que dibujaría el objeto texturado

```
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();
```

c) 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.
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);
```

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 pixel 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 pixeles se guardan en este buffer. Si utilizamos dichos pixeles en pantalla para dibujar otro objeto, se producirá la comparación de las profundidades de dichos pixeles. Si la profundidad del último píxel es mayor que la nueva (está más lejos) el pixel nuevo no se pinta, mientras que si está más cerca (la profundidad es menor), se dibuja el pixel y se guarda la nueva profundidad en el zbuffer.

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.

¿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.

¿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.

$$\frac{n_v}{\|N_v\|}$$

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.

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.

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.

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)