Examen Coga

Pregunta 1

- **¿Cada cuánto se ejecuta el vertex shader y el fragment shader? Una vez por vértice, una vez por segmento, una vez por ciclo, una vez por pixel. Razona la respuesta**

El vertex shader ya que es el encargado de desplazar cada vértice desde las posiciones local a las del world space se ejecutará una vez por vértice y así situar todos correctamente.

El fragment shader en cambio lo que hará será ejecutarse una vez por cada fragmento que le llegue y deducir así el color del píxel resultante. Puede ser que haya fragmentos que por tener objetos delante no se vean y es por eso que hay más fragmentos que píxeles y no podemos afirmar que se ejecuta una vez por píxel.

- **¿Después de tener los fragmentos que cálculos hay que realizar?**

Quedan por realizar los cálculos respecto a la iluminación, el texturizado, la transparencia (blending test), el alpha test, qué objetos serán visibles del frustrum (depth-test) y de evitar hacer calculos de objetos no visibles como es el z-test.

- **¿Qué es el z-buffer? ¿Y cómo se calcula?**

El z-buffer se utiliza para determinar si un objeto o parte de un objeto es visible en una escena. Tiene el mismo ancho y alto que el búfer de color. Cada píxel tiene una coordenada XYZ de profundidad o distancia de la cámara (16, **24** o 32 bits). Compara el valor de cada fragmento con el anterior y en función del resultado se sustituye o no.
Se renderiza un objeto (escritura frame buffer y z buffer) → Se va a renderizar el siguiente, si el valor del z-buffer es inferior se dibuja y actualiza.
Las comparaciones se basan en la ecuación :

$$F_{depth} = \frac{1/z - 1/near}{1/far - 1/near}$$

- **Explicar Flat shading, Gouraud shading y Phong Shading**

Flat shading es un modelo de iluminación local que se basa en el **cálculo de la normal de cada cara** y, a partir de esta normal, calcular la iluminación de toda la cara. Es la más eficiente pero el resultado no es el mejor.
Gouraud Shading es un modelo de iluminación local que se basa en el **cálculo de la normal de cada vértice y, a partir de la normal de cada vértice, de cada cara**. Se hace una **interpolación bilineal** para que la cara tenga una iluminación variable. Este método es menos eficiente que el anterior pero el resultado obtenido es mejor.
Phong Shading es un modelo de iluminación local que se basa en el **cálculo de la normal de cada punto a partir de la interpolación bilineal**. Es muy costoso pero ofrece mejores resultados que las técnicas anteriores.

Pregunta 2

Detalla paso a paso dos secuencias diferentes de transformaciones (traslaciones, rotaciones o escalados) que conviertan el segmento [(2,1); (4,1)] en el segmento [(2,1); (2,5)]. Explica los pasos a realizar y describe el código OpenGL correspondiente.

Primera opción: trasladar al origen, rotar, escalar y trasladar a la posición deseada:

glTranslatef(-2,-1,0); -> 0,0-2,0

glRotatef(90,0,0,1) -> 0,0-0,2

glScalef(1,4/2,1) -> 0.0-0.4

glTranslatef(2,1,0) -> 2,1-2,5

A ALGUIEN SE LE OCURRE OTRA?

Valdría así? Es una gitanada que no tiene pinta de que valga

(A mi me mola) I I

glTranslatef(-4,-1,0); -> -2,0-0,0

glScalef(4/2,1,1) -> -4.0-0.0

glRotatef(-90,0,0,1) -> 0,0-0,4

glTranslatef(2,1,0) -> 2,1-2,5

Pregunta 3

- Supongamos **gluPerspective** (90, 1, 1, 11);
- Calcula la matriz de proyección.
- Calcula el punto de proyección de (0,0,-1), (0,11,-11), (0,4,-6); Razona la respuesta

$$\begin{bmatrix} \frac{1}{Aspt\tan(fov/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(fov/2)} & 0 & 0 \\ 0 & 0 & \frac{N+F}{N-F} & \frac{2.F.N}{N-F} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Punto	P. Coordenadas Homogéneas	Resultado	Resultado Normalizado
(0,0,-1),			
(0,11,-11),			
(0,4,-6);			

Calculamos la matriz de proyección:

1	0	0	0
0	1	0	0
0	0	-1.2	-2.2
0	0	-1	0

Ahora calculamos las coordenadas resultantes del punto:

0,0,-1	-> 0,0,-1,1	-> 0,0,-1,1	-> 0,0,-1,1
0,11,-11	-> 0,11,-11,1	-> 0,11,11,11	-> 0,1,1,1
0,4,-6	-> 0,4,-6,1	-> 0,4,5,6	-> 0,4/6,%,1

Estas serán las nuevas coordenadas de estos puntos normalizadas ya, al verse deformadas por la perspectiva que da un realismo a la imagen pero lo paga con las deformaciones hacia el punto de fuga.

Pregunta 4

Explicar que son el vertex shader y el fragment shader. Parámetros entrada-salida y cómo se comunican

En el contexto de la programación gráfica, el vertex shader y el fragment shader son etapas fundamentales en el proceso de renderizado de gráficos en una tarjeta gráfica. Ambos son programas pequeños y altamente especializados que se ejecutan en la GPU (Unidad de Procesamiento Gráfico) y tienen diferentes roles en el procesamiento de los vértices y fragmentos de una malla o primitiva gráfica.

El vertex shader se encarga de transformar las coordenadas de los vértices de un objeto en el espacio tridimensional al espacio bidimensional de la pantalla. Esto implica aplicar las transformaciones necesarias, como rotaciones, escalado y traslación, para posicionar correctamente los vértices en relación con la cámara y la proyección. Además, el vertex shader también puede calcular información adicional, como las coordenadas de textura o los vectores normales utilizados en la iluminación.

El fragment shader, por otro lado, trabaja a nivel de fragmentos o píxeles. Su función principal es calcular el color de cada fragmento generado durante el proceso de rasterización. El fragment shader puede realizar una variedad de tareas, como aplicar efectos de sombreado, texturizado, iluminación y otros cálculos para determinar el color final de cada píxel en la pantalla.

En cuanto a los parámetros de entrada y salida, tanto el vertex shader como el fragment shader pueden recibir y devolver información.

El vertex shader recibe como parámetros de entrada los atributos de los vértices, que pueden incluir su posición en el espacio tridimensional, coordenadas de textura, vectores

normales, colores, entre otros. Estos atributos son proporcionados por la CPU (Unidad Central de Procesamiento) antes de enviar los datos a la GPU. Además de los atributos de los vértices, el vertex shader también puede recibir matrices de transformación y otros parámetros necesarios para realizar los cálculos requeridos.

A medida que el vertex shader procesa los vértices, puede generar valores de salida, como las coordenadas de los vértices transformados, los atributos interpolados para cada fragmento y cualquier otra información necesaria para el fragment shader.

El fragment shader recibe como parámetros de entrada los atributos interpolados generados por el vertex shader para el fragmento actual, como las coordenadas de textura interpoladas, vectores normales interpolados, colores interpolados, etc. Además de estos atributos interpolados, el fragment shader también puede recibir otras entradas, como texturas, luces, sombras, etc., necesarias para realizar los cálculos de sombreado y determinar el color final del fragmento.

Como salida, el fragment shader devuelve el color calculado para el fragmento actual. Esta salida puede ser utilizada para colorear directamente el píxel en la pantalla o puede ser procesada posteriormente por etapas adicionales, como el blending (mezclado), para producir el resultado final en el framebuffer.

La comunicación entre el vertex shader y el fragment shader se realiza a través de los atributos interpolados. Durante el proceso de rasterización, se generan fragmentos en función de los vértices de las primitivas y se interpolan los atributos correspondientes entre los vértices adyacentes. Estos atributos interpolados se pasan como entrada al fragment shader para calcular el color del fragmento. La interpolación se realiza automáticamente por la GPU, lo que permite que los atributos se suavicen de manera adecuada a lo largo de la superficie de la primitiva, lo que resulta en una representación visualmente más realista del objeto.

Pregunta 5

Describe matricialmente cómo se pasa de Local Space a Screen Space

En un comienzo nos situamos en el Local space que no es otra cosa que las coordenadas de los objetos teniendo como origen el centro de la figura. De ahí a través de la matriz del modelo hacemos las transformaciones geométricas necesarias para situar los objetos en el world space, es decir, en la escena que queremos construir.

A continuación a través del view matrix y de nuevo, mediante transformaciones geométricas, colocamos la cámara, es decir, el lugar desde el que observaremos el world space. Disponemos los objetos de tal manera que obtengamos la imagen que deseamos.

Por último, a través de la projection matrix, creamos el clip space, es decir, ya conocemos que objetos estarán dentro del frustum y cuales no. Los que estén fuera se descartarán para así ganar rendimiento, ya que no serán visibles; y los que si entren dentro del campo de visión sí que se transformarán. Además pasamos ahora de una imagen 3D a una 2D.

Finalmente, a través del viewport pasamos a las coordenadas de la pantalla y calculamos sus fragmentos. Puede haber varios viewports.