Junio-2018-Resuelto.pdf

CarlosGarSil98

Realidad Virtual

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
Universidad de Huelva

Universidad de Huelva

Departamento de Tecnologías de la Información

Área de Ciencias de la Computación e Inteligencia Artificial

Realidad Virtual
Examen de junio

EJERCICIO 1 (2 puntos)

Describa brevemente las etapas del proceso de renderizado utilizado en OpenGL. (Un párrafo para cada etapa)

VertexShader: Se ejecuta sobre cada vértice de forma independiente. Utiliza atributos de entrada asociados a cada vértice y genera como salida la posición del vértice en coordenadas clip.

Ensamblado de primitivas: Es una etapa no programable. Recibe la indicación de las primitivas a dibujar y los atributos de los vértices que las forman, generados por el VertexShader. Además, detecta si están ocultos o fuera del área de dibujo.

Teselado: Recibe los datos de un parche y genera nuevas primitivas. Se divide en tres fases:

- Control de Teselado (TCS): Recibe los vértices del patch y genera los valores de los vértices de las primitivas que se crearán.
- Generación de Primitivas (TPG): Genera las primitivas en un espacio de coordenadas llamado espacio de parámetros sus coordenadas se denominan u y v, que van desde 0 a 1.
- Evaluación de Teselado (TES): Transforman las coordenadas del espacio de parámetros a coordenadas de posición de los vértices (clip).

GeometryShader: Es una etapa opcional. Recibe las primitivas ya ensambladas y permite realizar modificaciones sobre ellas.

Rasterización e Interpolación: Es una etapa no programable. Estudia qué vértices quedan dentro del clipping volume, desechando aquellos que estén fuera, y adaptando las primitivas que estén parcialmente fuera a través de nuevos vértices que se crearan en los bordes del clipping volumen. Después se identifican qué pixeles forman parte de la primitiva a dibujar según la ubicación del centro del píxel.

FragmentShader: Se ejecuta por cada píxel dentro de la primitiva, recibiendo los valores interpolados para el píxel. Genera el color asociado a cada píxel.

Operaciones de raster: Combina los diferentes fragmentos para generar la versión final de la imagen.

El proceso de renderizado genera al final el contenido de la imagen en una estructura que conoceremos como ColorBuffer.

EJERCICIO 2 (1.5 puntos)

¿Qué es un Vertex Buffer Object? ¿Y un Vertex Array Object? Describa la forma de crearlos y configurarlos.

Un Vertex Buffer Object (VBO) es una estructura de datos que puede almacenar en la memoria de la tarjeta gráfica (GPU) diferentes tipos de información, según qué tipo de buffer se le asigna al activarlo como, por ejemplo: almacenar atributos de los vértices o almacenar una lista de índices a vértices.

Se crean, desde el código para la CPU, a través de la función glGenBuffers(), donde se le pasa el número de buffers a crear y los identificadores de los buffers. Ya creado un VBO, para poder utilizarlo es necesario activarlo con la función glBindBuffer(), donde se le indica qué tipo de buffer será. Por último, para almacenar los datos, se utiliza la función glBufferData(), indicándole el tipo de buffer a llenar y dónde están los datos que va a guardar.

Los Vertex Array Object (VAO) son creados después de los VBOs y una vez estén almacenados los datos de los vértices. Son estructuras de datos que definen el mapeo entre los atributos y los nombres de las entradas de los shaders.
A través de la función glVertexArrays(), podemos crear un cierto número de VAOs y obtener sus identificadores. Para usar un VAO, se activa primero con la función glBindVertexArray().
Para definir un atributo perteneciente a un VAO se usa la función glVertexAttribPointer(), indicándole el índice, cuántos datos habrá por atributo y el tipo que será.
Para asignar la posición de una variable de entrada, se utiliza la función glBindAttribLocation(), indicándole el shader, la posición del atributo y el nombre de la variable.

EJERCICIO 3 (1.5 puntos)

Describa el modelo de iluminación de Phong. ¿Cuántos tipos de luz utiliza? ¿Como se calcula el efecto de cada tipo de luz?

El color con el que vemos los objetos se debe a que la luz choca contra ellos y es en parte absorbida y en parte dispersada. Por ello en el modelo de iluminación más se considera la iluminación como la suma de tres tipos de luz diferentes: luz ambiental, luz difusa y luz especular.
El modelo de Phong consiste en interpolar las propiedades de los vértices para cada pixel y calcular el color directamente en el FragmentShader. Aunque resulta más costoso, se consiguen mejores resultados de coloreado.
Como hemos dicho el color final de cada píxel (Cf) se calcula con la suma de cada color para cada tipo de luz, es decir: Cf = Ca + Cd + Cs.

- Color de Luz Ambiental: Se calcula como el producto de la luz ambiental (La) y el color del material ante la luz ambiental (Ma): Ca = La * Ma.
- Color de Luz Difusa: Se calcula como el producto de la luz difusa (Ld) y el color material ante la luz difusa (Md), multiplicados por un factor de incidencia de la luz. Ese factor se calcula como el producto entre el vector de dirección de la luz (l) y el vector normal a la superficie (N). Cd = Ld * Md * Fl.
- Color de Luz Especular: Se calcula como el producto de la luz especular (Ls) y el color del material ante la luz especular (Ms), multiplicados por el factor especular. Dicho factor se calcula como el producto entre el vector de posición del observador (V) y el vector de reflexión de la luz (R), elevado una cierta potencia. Cs = Ls * Ms * FS.

Las propiedades de un material frente a la luz ambiental se describen con los componentes RGBA. Por ejemplo, un objeto de color azul puro se definiría como (0.0,0.0,1.0,1.0).
Si la luz ambiental fuera un blanco puro (1.0,1.0,1.0,1.0) el color del objeto el azul puro seria (0.0,0.0,1.0,1.0). Pero si la luz ambiental fuera rojo puro (1.0,0.0.0.0,1.0), el objeto absorbería toda esa luz y se vería negro (0.0,0.0,0.0,0.0).
De la misma forma se puede definir el efecto de la luz difusa y de la luz especular sobre el objeto.

EJERCICIO 4 (1.5 puntos)

¿Qué es una textura? ¿Qué tipos de texturas existen? ¿Como se crean las texturas? ¿Como se asigna su contenido? ¿Como se aplican las texturas?

Se denomina *textura* a la aplicación de una imagen a una primitiva geométrica. Los puntos de las texturas se denominan *texels*.
Según cómo esté estructurada la textura, existen varios tipos:

CIUDAD JARDÍN
AUTOESCUELA

10 Años contigo Sevilla

Más de 10 años de experiencia formando a nuevos conductores

Todos los permisos

Elige la oferta que mejor se adapte a ti

Desde **79€**

SCAN ME

www.autoescuelaciudadjardin.es

INFÓRMATE AHORA

- Texturas 1D: es un vector de texels en una dimensión.
- Texturas 2D: matriz del texels.
- Texturas 3D: matriz tridimensional de texels. (colección de imágenes).
- Cubemaps: es un conjunto de 6 suoerficiaes cuadradas, distribuidas como las caras de un cubo.

Para trabajar con texturas se definen *objetos textura* que definen buffers de datos que se almacenan en la GPU. Cada *objeto textura* se identifica mediante un número. Se reservan/crean los identificadores de texturas mediante la función *glGenTextures()*, indicándole el número de identificadores que se deseen y un array donde los almacene. Después, se define el tipo de textura que tendrá un identificador a través de la función *glBindTexture()*.

Por último, para cargar una textura a un identificador, se utiliza la función *glTexImage_D()* según el tipo de textura.
También se puede asignar las texturas copiando los valores del color buffer con *glCopyTexImage_D()*, indicando desde qué posición tiene que copiar. Otra forma puede ser cargar sólo un trozo de una textura sobre la ya cargada con *glSubTexImage_D()*, indicando desde qué posición empieza a sobrescribir.

EJERCICIO 5 (1.5 puntos)

Describa el proceso de generación automática de primitivas por medio de las etapas de teselado. ¿Qué funciones realiza cada etapa del proceso de teselado? ¿Qué entradas y salidas utiliza cada etapa? ¿Cómo se configura el nivel de teselado?

Es una etapa opcional del proceso de renderizado, pero si un programa contiene esta etapa, solo se pueden renderizar primitivas de tipo parche. Esta etapa se divide en tres fases:

- Control de Teselado (TCS): Recibe los vértices del parche y con su información genera los valores de los vértices de las primitivas que se crearán.
- Generación de Primitivas (TPG): Recibe una lista de primitivas que genera, las sitúa en un espacio de coordenadas llamado espacio de parámetros, donde sus coordenadas se denominan u y v, que van de 0 a 1.
- Evaluación de Teselado (TES): Recibe las coordenadas de cade vértice (tessCoord) en el espacio de parámetros y los atributos de los parches (gl_in) y los transforma a coordenadas de posición de vértices (clipping volumen).

El nivel de teselado se puede configurar como variables de entrada uniforma del TPG que indiquen el numero de puntos de control que tendrá una superficie. Pero también puede calcularse internamente según la profundidad por ejemplo (indicando en la entrada los niveles máximo y mínimo de profundidad).

EJERCICIO 6 (1 puntos)

¿Qué es un Frame Buffer Object? Describa la forma de crearlo y utilizarlo.

EL proceso de renderizado genera el contenido de la imagen en el ColorBuffer. Si además se encuentra activada la opción de “test de profundidad”, se almacena la información sobre la coordenada *Z* (profundidad) en una estructura paralela llamada DepthBuffer. Existe otro buffer, el StencilBuffer, que se utiliza como buffer auxiliar para otro tipo de información configurable.

Un FrameBuffer Object (FBO) es una estructura de datos que describe la salida del proceso de renderizado y contiene enlaces al ColorBuffer, DepthBuffer y StencilBuffer. Por defecto, OpenGL crea un FBO con identificador 0 que asocia a la pantalla.

Para crear un FBO se utiliza la función glGenFramebuffers(), indicándole el numero de FBOs a crear y el array done están los identificadores, Para activar uno de ellos, se utiliza la función glBindFramebuffer().

Para dirigir el proceso de renderizado a memoria, se crea un FBO y los buffers que estén vinculados a él (Color y Depth). Para vincular estos buffers al FBO, se utiliza glFramebufferTextura2D() si el DepthBuffer está cargado como textura, si no es así, es necesario cargarlo mediante un RenderBuffer. El tamaño de los buffers debe ser el mismo. Una vez vinculados, se activa el FBO. Por último, se indica que el FBO será el buffer de dibujo mediante la función glDrawBuffers(). Después hay que devolver la salida del renderizado a la pantalla.

EJERCICIO 7 (1 PUNTOS)

¿Qué es un Transform Feedback Object? Describa la forma de crearlo y utilizarlo.

Una de las formas para generar escenas con figuras no rígidas es el Transform FeedBack. Consiste en modificar los atributos de los vértices en función de los valores generados en un renderizado anterior, lo que requiere almacenar los datos generados en la memoria de la GPU y ser capaz de leerlos en un futuro, renderizados.

La técnica utilizada se conoce como *buffer ping-pong* porque realiza el renderizado en dos pasadas. En la primera se ejecuta solo el VertexShader y en la segunda se realiza el proceso completo, tomando como entrada el buffer generado en la primera pasada.

Se definen dos Transform FeedBack Object (TFO) que describen los buffers a utilizar como salida y entrada del VertexShader. Estos se crean con el glGenTransformFeedbacks(), indicando el numero de buffers y el array donde están los indicadores, en este caso serian dos. Se activan con la función glBindTransformFeedback() y se enlazan tantos buffers como se quieran tener ubicados, con glTransformFeedbackVaryings().

Durante el renderizado hay que iniciar el proceso de transform feedback con la ufncion glBeginTransformFeedback().

Cuando se lancen las primitivas de dibujo, las salidas del VertexShader se almacenarán en los buffers indicador del TFO. Para finalizar el proceso de transform feedback se utiliza el comando glEndTransformFeedback().