Universidad Rey Juan Carlos

ETSII

Práctica 1: Introducción a GLSL

Informática Gráfica

Grado en Diseño y Desarrollo de Videojuegos

Miguel Ángel Otaduy

Marcos García Lorenzo

Tabla de contenido

Introducción

El objetivo de esta práctica es que el alumno entienda los fundamentos de programación de "*shaders*" explicados en el parte teórica de la asignatura. Cada grupo de prácticas deberá implementar la funcionalidad básica detallada en este guión de prácticas.

Normativa

Esta práctica se divide en tres bloques:

- Bloque guiado: este bloque se realizará de forma guiada en el aula de prácticas. La **asistencia es obligatoria**. El bloque guiado no es re-evaluable por lo que la no asistencia acarreará el suspenso de la asignatura.
- Bloque obligatorio: los grupos de prácticas deberán realizar este bloque de forma supervisada (pero no guiada) en las horas de laboratorio que se habiliten para tal efecto. El trabajo que no pueda realizarse dentro de las horas de laboratorio podrá realizarse fuera del horario de la asignatura. El trabajo realizado, tanto en el bloque guiado como en el bloque obligatorio, se evaluará mediante un examen de prácticas que podrá aprobarse tanto en la convocatoria de diciembre como en la de junio. El código de ambos bloques se entregará junto, a través de campus virtual en el plazo que indique en la página web de la asignatura. Deberá adjuntarse una pequeña memoria explicativa.
- Bloque opcional: las partes opcionales se realizarán por el alumno fuera del horario de la asignatura. Las tareas de este bloque no son obligatorias pero la nota final dependerá en gran medida de las partes opcionales realizadas. El bloque opcional se entregará junto con una memoria explicativa a través de la herramienta habilitada en campus virtual, en el plazo que se indique.

Las prácticas podrán realizarse tanto de forma individual como en grupos de dos personas.

Parte guiada

Paso 1: Implementación de un *Shader* básico

1. El profesor explicará el entorno compuesto de:
    a. Un proyecto de Visual Studio que implementa la funcionalidad del cliente.
    b. La librería IGlib que esconde la funcionalidad de OpenGL utilizada.
    c. El modelo 3D de un cubo.
    d. El esqueleto de un *shader* de vértices (shader.v0.vert) y de un *shader* de fragmentos (shader.v0.frag), sobre los que se implementará la funcionalidad requerida.
2. Copia los ficheros *shader.v0.vert* y *shader.v0.frag* a *shader.v1.vert* y *shader.v1.frag*.
3. Modifica el fichero main.cpp para que utilicen los nuevos *shaders* de vértices y fragmentos.
4. Modifica el *shader* de fragmentos de forma que todos los fragmentos se pinten del mismo color.

5. Diseña un *shader* de vértices de forma que los vértices del modelo se proyecten utilizando perspectiva simétrica con una apertura de 60º. Sitúa la cámara de forma que sea capaz de visualizar toda la escena.

   Recuerda que la matriz de proyección en OpenGL es:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Paso 2: Cliente/Servidor

En este paso el cliente suministrará las transformaciones al *shader* de vértices:

1. Copia los ficheros *shader.v1.vert* y *shader.v1.frag* a *shader.v2.vert* y *shader.v2.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica el fichero *main.cpp* de forma que la cámara se situe en el punto (0, 0, 6), mirando en la dirección de las Z's negativas y que la matriz de proyección defina una vista simétrica con una apertura de 60º.
4. Modifica el *shader* de vértices para que utilice las nuevas matrices de vista y proyección.
5. Ejecuta la aplicación.
6. Utiliza la función Idle para rotar el cubo sobre su eje.

Paso 3: Mejorando la eficiencia

La librería IGL calcula los productos de las matrices *model, view y projection*. Esto nos permite acelerar el procesado de los vértices.

1. Copia los ficheros *shader.v2.vert* y *shader.v2.frag* a *shader.v3.vert* y *shader.v3.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Mejora la eficiencia del *shader* de vértices.

Paso 4: Depurando primitivas

1. Copia los ficheros *shader.v3.vert* y *shader.v3.frag* a *shader.v4.vert* y *shader.v4.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica la funcionalidad del *shader* de fragmentos para que pinte cada primitiva de un color distinto.

Paso 5: Depurando vértices

1. Copia los ficheros *shader.v4.vert* y *shader.v4.frag* a *shader.v5.vert* y *shader.v5.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica la funcionalidad del *shader* de vértices y del *shader* de fragmentos para que pinte cada primitiva de un color distinto.

Paso 6: Color por vértice

1. Copia los ficheros *shader.v5.vert* y *shader.v5.frag* a *shader.v6.vert* y *shader.v6.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica la funcionalidad del *shader* de vértices de forma que el color del fragmento venga determinado por el color de los vértices de la primitiva a la que pertenece.

Paso 7: Acceso a texturas

1. Copia los ficheros *shader.v6.vert* y *shader.v6.frag* a *shader.v7.vert* y *shader.v7.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica el fichero *main.cpp* para que asigne una textura al objeto (IGlib::addColorTex).
4. Modifica la funcionalidad del *shader* de vértices y del *shader* de fragmentos de forma que el color final del fragmento venga determinado por una textura.

Paso 8: Depurando normales (I)

1. Copia los ficheros *shader.v7.vert* y *shader.v7.frag* a *shader.v8.vert* y *shader.v8.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica la funcionalidad del *shader* de vértices y del *shader* de fragmentos de forma que el color final del fragmento venga por su normal.

Paso 9: Depurando normales (II)

1. Copia los ficheros *shader.v8.vert* y *shader.v8.frag* a *shader.v9.vert* y *shader.v9.frag*.
2. Modifica el fichero *main.cpp* para que utilice los nuevos *shaders* de vértices y fragmentos.
3. Modifica la funcionalidad del *shader* de vértices y del *shader* de fragmentos de forma que el color final del fragmento venga por su normal en coordenadas de la cámara.

Parte obligatoria

En este bloque deberá implementarse la siguiente funcionalidad:

1. **Define una matriz de proyección que conserve el *aspect ratio*** cuando cambiamos el tamaño de la ventana.
2. **Añade un nuevo cubo a la escena**. El segundo cubo deberá orbitar alrededor del primero describiendo una circunferencia a la vez que rota sobre su eje Y.
3. **Control de la cámara con el teclado**. Controles mínimos que deberán incluirse: movimiento hacia adelante, retroceso, movimientos laterales (izquierda y derecha) y giros (izquierda y derecha).

Parte opcional

En este apartado se describen las partes que podrán realizarse de forma opcional.

1. Controla el giro de la cámara utilizando el ratón.
2. Crea un tercer cubo y hazlo orbitar alrededor del primero. Define su movimiento utilizando curvas de *Bézier*, *splines* cúbicos o polinomios de interpolación de *Catmull-Rom.*
3. Crea un nuevo modelo y añádelo a la escena.
    a. Opción 1: Define sus vértices manualmente.
    b. Opción 2: Utiliza *3D studio* o una herramienta similar, exporta sus vértices y crea un fichero "*.h*" del estilo de "*Box.h*".
    c. Opción 3: Carga un fichero de un formato conocido. Puedes utilizar librerías auxiliares como ASSIMP (http://assimp.sourceforge.net/).
4. Puedes sugerir nuevas partes opcionales a los profesores de prácticas.