Tema 1: Introducción a OpenGL

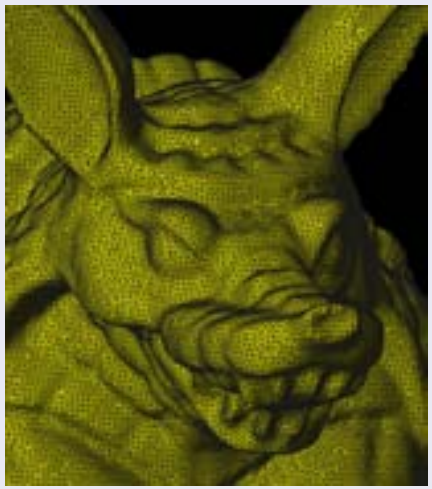
J. Ribelles

SIE020: Síntesis de Imagen y Animación
Institute of New Imaging Technologies, Universitat Jaume I

Contenido

¿Qué es OpenGL?

- OpenGL es una interfaz de programación de aplicaciones (API) para generar imágenes por ordenador.
- OpenGL es independiente tanto del sistema operativo como del sistema gráfico de ventanas.
- Este capítulo introduce la programación con OpenGL y presenta el lenguaje GLSL para la programación de *Shaders*.



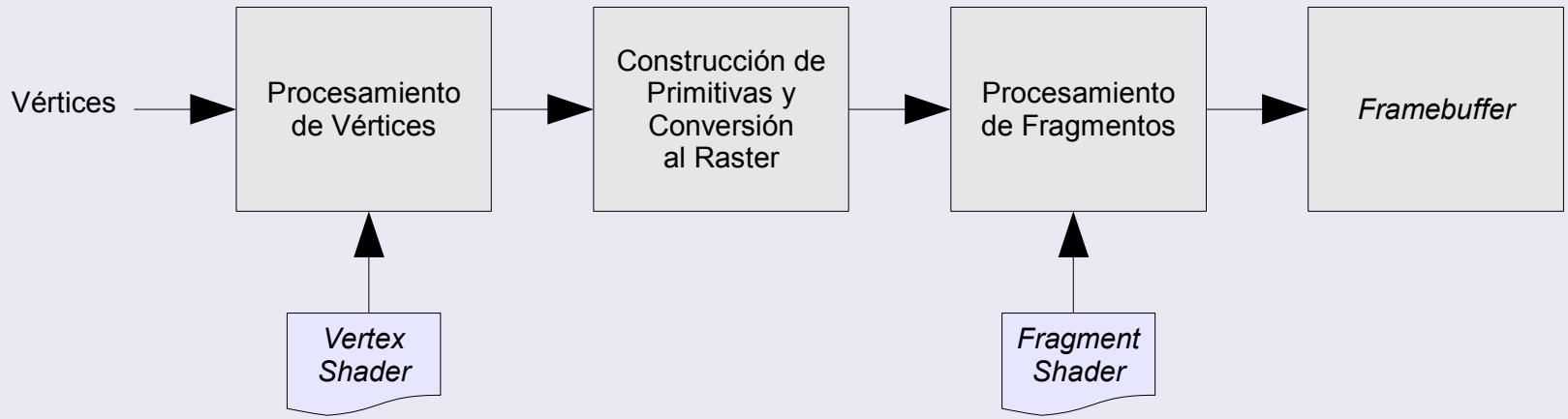
Antecedentes y evolución

- Se presentó en 1992, la OpenGL Architecture Review Board (ARB) conduce la evolución.
- En sus orígenes, OpenGL se basó en un *pipeline* configurable de funcionamiento fijo.
- Con el paso del tiempo, se creo un mecanismo para definir extensiones que permitía a los fabricantes proporcionar hardware gráfico con mayores posibilidades.
- En el 2004 aparece OpenGL 2.0, con GLSL 1.1, iba a permitir escribir código que fuese ejecutado por la GPU, *Shaders*.

Antecedentes y evolución

- Dos años más tarde, ARB pasó a ser parte del grupo Khronos (http://www.khronos.org/). Entre sus miembros activos: AMD (ATI), Apple, Nvidia, S3 Graphics, Intel, IBM, ARM, Sun, Nokia, etc.
- Es en el año 2008 cuando OpenGL, con la aparición de OpenGL 3.0 y GLSL 1.3, adopta el modelo de obsolescencia.
- Sin embargo, en el año 2009, con las veriones de OpenGL 3.1 y GLSL 1.4 es cuando probablemente se realiza el cambio más significativo, el *pipeline* de funcionalidad fija y sus funciones asociadas son eliminadas.

Secuencia básica de operaciones del *pipeline* de OpenGL



FreeGLUT, http://freeglut.sourceforge.net/

Permite crear ventanas que contienen contextos OpenGL en cualquiera de los sistemas operativos más populares como Microsoft Windows, Apple's Mac OS X y Linux. Para crear interfaces más amigables:

- GLUI, http://glui.sourceforge.net/
- FLTK, http://www.fltk.org/

GLEW, http://glew.sourceforge.net/

Determina en tiempo de ejecución qué extensiones OpenGL están soportadas.

GLM, http://glm.g-truc.net/

Librería matemática basada en la especificación del lenguaje GLSL. Está hecha en C++ y consta únicamente de ficheros de cabecera.

El primer programa con OpenGL

El flujo general

1. Crear una ventana.
2. Inicializar los estados de OpenGL.
3. Procesar los eventos generados por el usuario.
4. Dibujar la escena (y volver al paso 3).

Ejemplo básico

```
void init (void) {
  glClearColor (1.0, 1.0, 0.0, 1.0);
}

void reshape (int ancho, int alto) {
  glViewport (0, 0, ancho, alto);
}

void display (void) {
  glClear (GL_COLOR_BUFFER_BIT);
  glutSwapBuffers ();
}

int main (int argc, char **argv) {
  glutInit               (&argc, argv);
  glutInitDisplayMode    (GLUT_DOUBLE | GLUT_RGBA);
  glutInitWindowSize     (1024, 768);
  glutInitWindowPosition (100, 100);
  glutCreateWindow       (argv[0]);

  init ();

  glutDisplayFunc (display);
  glutReshapeFunc (reshape);
  glutMainLoop ();
  return 0;
}
```

El lenguaje GLSL

- Permite escribir código para ejecutar en la GPU. En la actualidad hay cinco procesadores: vértices, control de teselación, evaluación de teselación, geometría y fragmentos.
- GLSL es un lenguaje de alto nivel, parecido al C, aunque también toma prestadas cosas del C++. Su sintaxis se basa en el ANSI C. Constantes, identificadores, operadores, expresiones y sentencias son básicamente las mismas que en C. El control de flujo con bucles, la sentencias condicionales if-then-else y las llamadas a funciones son idénticas al C.
- Y también hay características del C no soportadas en OpenGL como es el uso de punteros, los tipos: byte, char, short, long int; y la conversión implícita de tipos está muy limitada.
- Del C++, GLSL copia la sobrecarga, el concepto de constructor y el que las variables se pueden declarar en el momento de ser utilizadas.

El lenguaje GLSL

- Pero GLSL también añade características no disponibles en C, entre otras se destacan las siguientes:
  - Tipos vector: vec2, vec3, vec4
  - Tipos matriz: mat2, mat3, mat4
  - Tipos *sampler* para el acceso a texturas: sampler1D, sampler2D, sampler3D, samplerCube
  - Tipos para comunicarse entre *Shaders* y aplicación: uniform, in, out
  - Acceso a componentes de un vector mediante: .xyzw .rgba .stpq
  - Operaciones vector-matriz, por ejemplo: *vec*4 $a = b * c$, siendo b de tipo *vec*4 y c de tipo *mat*4
  - Variables predefinidas que almacenan estados de OpenGL
- GLSL también dispone de funciones propias como, por ejemplo, trigonométricas (sin, cos, tan, etc.), exponenciales (pow, exp, sqrt, etc.), comunes (abs, floor, mod, etc.), geométricas (length, cross, normalize, etc.), matriciales (transpose, inverse, etc.) y más.

Ejemplo básico

```
const char *vertexShaderSource =
{
  "#version 140\n"
  ""
  "in vec4 posicion;"
  ""
  "void main ()"
  "{"
  "    gl_Position = posicion;"
  "}"
};

const char *fragmentShaderSource =
{
  "#version 140\n"
  ""
  "out vec4 color;"
  ""
  "void main (void)"
  "{"
  "    color = vec4 (1.0,0.0,0.0,1.0);"
  "}"
};
```

Trabajando con Shaders

El compilador de GLSL

Está integrado en el propio *driver* de OpenGL. Esto implica que la aplicación en tiempo de ejecución será quien envíe el código fuente del *Shader* al *driver* para que sea compilado y enlazado.

Pasos

1. Crear y compilar los objetos *Shader*.
2. Crear un programa y añadirle los objetos compilados.
3. Enlazar el programa creando un ejecutable.

Ejemplo básico

```
void initShader (void)
{
  GLuint vertexShader = glCreateShader (GL_VERTEX_SHADER);
  glShaderSource  (vertexShader, 1, &vertexShaderSource, NULL);
  glCompileShader (vertexShader);
  chequeaCompilacion (vertexShader);

  GLuint fragmentShader = glCreateShader (GL_FRAGMENT_SHADER);
  glShaderSource  (fragmentShader, 1, &fragmentShaderSource,NULL);
  glCompileShader (fragmentShader);
  chequeaCompilacion (fragmentShader);

  program = glCreateProgram ();

  glAttachShader  (program, vertexShader);
  glAttachShader  (program, fragmentShader);

  glLinkProgram   (program);
  chequeaEnlazado (program);

  coordenadas = glGetAttribLocation (program, "posicion");
}
```