Open Graphics Library (OpenGL)

Images were taken from the book: “Interactive Computer Graphics” by Angel and Shreiner
OpenGL

• Application programming interface (API) for rendering 2D\3D graphics

• The API interacts with the graphics processing unit (GPU)

• Created in 1991 by Silicon Graphics. Now it is managed by the Khronos consortium
OpenGL – Design

• A library of functions (e.g., `glDrawArrays`) and constants (e.g., `GL_TRIANGLES`)

• Language-independent: has many bindings, e.g., [WebGL](https://www.w3.org/TR/webgl).

• Platform-independent: works on Win/Linux/..
OpenGL – Documentation

The manuals are known by their cover’s color:

• The Red Book: OpenGL Programming Guide
• The Orange Book: OpenGL Shading Language
• The Blue Book: OpenGL Reference manual
• The Green Book: OpenGL & X Window System
• The Alpha Book: OpenGL & Windows
OpenGL – Libraries

• OpenGL doesn’t support input or windowing

• GLUT (or freeglut) is used to create a context

• Platform-dependent extensions can be provided by vendors. Identified by ARB, EXT, ..

• GLEW wraps the extraction of functionality
OpenGL – History

• OpenGL 1 (1992): Fixed pipeline
• OpenGL 2 (2004): Programmable shaders
• OpenGL 3: Deprecated the fixed functionality
• **OpenGL 3.1** (2009): Removed completely the fixed pipeline
• ..
• OpenGL 4.5 (2014):
OpenGL – History

- OpenGL 1
- OpenGL 2
- OpenGL 3

- OpenGL 3.1
- ..
- OpenGL 4.5

State Machine

Data flow model
OpenGL – Suggested Readings

• The internet is full of tutorials..

• Our course book has many examples
  – Check chapters 2, 3 and appendices A, D.

• Also, see here:
  – http://www.opengl-tutorial.org/
OpenGL – Coding

• The API is obtained by including freeglut:

```c
#include <GL/freeglut.h>
```

• Extensions are accessed through GLEW:

```c
#include <GL/glew.h>

glewInit();
```
OpenGL – Coding

• Data flow model: send data to the GPU

• Many functions in the form of

```c
glSomeFunction*();
```

• $* = nt[v]$, where $n \in \{2,3,4\}$ and $t \in \{i,d,f\}$, and $v$ means we pass a pointer
OpenGL – Coding

• Examples:

```glUniform1f(..);```

```glUniform3iv(..);```

• Later, we will see how to transfer matrices
OpenGL – Coding

• OpenGL contains functions that compute transformation matrices
  – glOrth
  – glFrustum
  – glRotatelf (‘f’ for ‘float’ type)
  – ...

• These functions **update** a transformation matrix, read about them before you use them
OpenGL – Coding

• Do not use OpenGL functions (except for the calls in the skeleton) in the HW assignments until you are explicitly required to do so!

• The skeleton uses OpenGL to display your buffer on the screen, you will write your own OpenGL code only in HW #3
• Rendering our data can be done simply by

```c
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glDrawArrays(GL_TRIANGLES, 0, sizeof(points));
    glFlush();
}
```
OpenGL – Coding

• GL_TRIANGLES is one of many primitives OpenGL supports:
  – GL_POINTS
  – GL_LINES
  – GL_POLYGON
  – …
OpenGL – Coding

• Send the array points to the GPU

• Name and create a vertex array object (VAO)

```c
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexBufferArray(vao);
```
OpenGL – Coding

• Next, we create a **vertex buffer object (VBO)**

• This is how data is actually **stored**

```c
GLuint buffer;
glGenBuffers(1, &buffer);
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);
```
OpenGL – Graphics Pipeline

• Similar to our software renderer

• Your program will consist of shaders
  – No default shaders
  – You must at least supply vertex and fragment shaders
OpenGL – Shaders

• **Vertex shader:**
  – Input: vertex location, possibly in object frame
  – Output: *vertex location* (clip space) to the rasterizer

• **Fragment shader:**
  – Input: fragment (pixel) inside the clipping volume
  – Output: a *color* of the fragment

Why “vertex” and not “vertices”?
OpenGL – Basic Shaders

- The **vertex** and **fragment** shaders run on the GPU

- **Vertex shaders** process each of the vertices
- **Fragment shaders** process each pixel in the relevant range
  - For example, if we draw a line between two vertices, the **vertex shader** will process the two vertices and the **fragment shader** will process the pixels on the line
**GL Shading Language (GLSL)***

- C-like language

- Need to compile and link in *runtime*

- Create a **program object** with **shader objects**

- Connect shader’s entities to our program
OpenGL – Basic Shaders

A minimal vertex shader assigns a color to each fragment

```glsl
#version 150
in vec4 vPosition;
void main() {
    gl_Position = vPosition;
}
```

A minimal fragment shader assigns a color to each fragment

```glsl
#version 150
out vec4 fColor;
void main() {
    fColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```
OpenGL – Coding

• Create a **program object with shader objects**

```c
GLuint program;
program = InitShader("vsoure.glsl", "fsource.glsl");
```

• Connect shader’s entities to our program

```c
GLuint loc;
loc = glGetUniformLocation(program, "vPosition");
 glEnableVertexAttribArray(loc);
 glVertexAttribPointer(loc, 2, GL_FLOAT, GL_FALSE, 0, 0);
```

Do not normalize data  Contiguous data  First index

An array of 2 floats per vertex
void init(void) {
    ...
    points = ...;
    ...
    GLuint program = InitShader( "vshader.glsl",
                                "fshader.glsl" );
    glUseProgram( program );

    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );
OpenGL – Example

```c
GLuint buffer;
glGenBuffers( 1, &buffer );
glBindBuffer( GL_ARRAY_BUFFER, buffer );
glBufferData( GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW );

GLuint loc = glGetAttribLocation( program,
                                   "vPosition" );

glEnableVertexAttribArray( loc );
glVertexAttribPointer( loc, 2, GL_FLOAT, GL_FALSE, 0, 0);

glClearColor( 1.0, 1.0, 1.0, 1.0 );
}
```
OpenGL – Example

```c
void display(void) {
    glClear( GL_COLOR_BUFFER_BIT );
    glDrawArrays( GL_POINTS, 0, sizeof(points) );
    glFlush( void );
}
```
Suggested Readings

• Interactive Computer Graphics, Chapter 2.8-2.10