Visual Computing Exercise 6: Introduction to OpenGL

Leonhard Helminger
leonhard.helminger@disneyresearch.com
## Schedule

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Exercise</th>
<th>Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 07/09</td>
<td>Exercise 6: OpenGL Rendering</td>
<td>Leonhard</td>
</tr>
<tr>
<td>Nov 14/16</td>
<td>Exercise 7: Shaders in OpenGL</td>
<td>Leonhard</td>
</tr>
<tr>
<td>Nov 21/23</td>
<td>Exercise 8: Theory: Light and Colors</td>
<td>Riccardo</td>
</tr>
<tr>
<td>Nov 28/30</td>
<td>Exercise 9: Matrices and Quaternions</td>
<td>James</td>
</tr>
<tr>
<td>Dec 05/07</td>
<td>Exercise 10: Lighting and Shading</td>
<td>Riccardo</td>
</tr>
<tr>
<td>Dec 12/14</td>
<td>Exercise 11: Rigid body dynamics</td>
<td>James</td>
</tr>
<tr>
<td>Dec 19/21</td>
<td>Q&amp;A</td>
<td>All</td>
</tr>
</tbody>
</table>
Overview

• Introduction to OpenGL
  – What is OpenGL?
  – Shaders
  – OpenGL-related libraries

• Exercise 6
What is OpenGL?

• Software interface to graphics hardware
• API for creating 2D and 3D computer graphics applications
• Hardware independent – implemented on many different platforms
Documentation

• OpenGL:  
  The red book

• OpenGL shading language:  
  The orange book

• http://www.opengl.org/registry
Rendering with OpenGL (1)

- Construct shapes from geometric primitives
Rendering with OpenGL (2)

• Arrange objects in 3D space
• Specify viewpoint
Rendering with OpenGL (3)

• Calculate colors of objects
  – Textures, materials, lighting
• Colors explicitly controlled with shaders
OpenGL is a state machine

- OpenGL can be put into various states or modes
- Settings remain in effect until changed again
- Examples: drawing color, characteristics of lights, viewing parameters
OpenGL is a state machine

```c
struct object_name {
    float option1;
    int    option2;
    char[] name;
};

struct OpenGL_Context {
    ...
    object* object_Window_Target;
    ...
};
```

// create object
unsigned int objectId = 0;
glGenObject(1, &objectId);
// bind object to context
glBindObject(GL_WINDOW_TARGET, objectId);
// set options of object currently bound to GL_WINDOW_TARGET
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_HEIGHT, 600);
// set context target back to default
glBindObject(GL_WINDOW_TARGET, 0);

Exemplary
Overview

• Introduction to OpenGL
  – What is OpenGL?
  – Shaders
  – OpenGL-related libraries

• Exercise 6
Graphics Pipeline

{ vertices } → Vertex shader → Shape assembly → Geometry shader

Tests and Blending → Fragment shader → Rasterization
Graphics Pipeline

• Programmable pipeline
  – Before OpenGL 3.0, it was called fixed function pipeline

• OpenGL shading language (GLSL)
  – Part of OpenGL > 2.0
  – C-based
Vertex and fragment shaders

Attributes given per vertex

Vertex shader computes varying

Interpolation of varying values

Fragment shader computes pixel color
Shader input/output

- **Uniforms (vertex/fragment shader)**
  - Global constants (for every vertex)
  - Examples: Light position, texture map

- **Attributes (vertex shader)**
  - Vertex-specific values
  - Examples: vertex position, normal

- **Varyings (vertex/fragment shader)**
  - Values passed from vertex to fragment shader
  - Interpolated across primitive
  - Example: fragment color

```glsl
uniform vec3 lightPos;
in vec4 position;
out vec3 color_out;
```
Vertex shader example

```cpp
#version 150

uniform vec4 lightPos;
uniform mat4 ProjectView_mat;

in vec4 position;
in vec4 color_in;
in vec3 normal;

out vec4 color_out;

void main(void) {

    // Lighting
    vec3 vecToLight = normalize(lightPos.xyz - position.xyz);
    float diffuseIntensity = dot(normal, vecToLight);
    diffuseIntensity = clamp(diffuseIntensity, 0.0, 1.0);
    color_out = color_in * diffuseIntensity;

    // Project vertex coordinates to screen
    gl_Position = ProjectView_mat * position;
}
```

In: Global constants
In: Per-vertex attribs
Out: Vertex color
Out: Vertex pos.
Fragment shader example

```glsl
#version 150
uniform vec4 I_am_not_used;
in vec4 color_out;
out vec4 color;

void main(void) {
    // Final color
    color = color_out;
}
```

(In: Global constants)

(In: Interp. pixel color)

(Trivial shader)
Overview

• Introduction to OpenGL
  – What is OpenGL?
  – Shaders
  – OpenGL-related libraries

• Exercise 6
API Overview

• OpenGL
  – Core functionality

• GLUT (OpenGL Utility Toolkit) / GLFW
  – Portable windowing API
  – Platform independent
  – Not officially part of OpenGL
• Commands for:
  – Window management: opening and configuring a window
  – Obtaining user input: mouse, keyboard, …
GLEW

- cross-platform C/C++ extension loading library
- it provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
Using GLFW

```c
int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 2);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

    GLFWwindow* window = glfwCreateWindow(800, 600, "OpenGL", nullptr, nullptr);
    glfwMakeContextCurrent(window);

    glewExperimental = GL_TRUE;
    glewInit();

    // set callback function for key-inputs
    glfwSetKeyCallback(window, key_callback);
    glfwSetErrorCallback(error_callback);
    ...
}
```
Using GLFW

... 
while(!glfwWindowShouldClose(window))
{
    glfwPollEvents();
    displayFunc();
    glfwSwapBuffers(window);
}

glfwTerminate();
return 0;
}
Using GLFW

• Example callback functions

```cpp
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, GLFW_TRUE);
    }
}

void error_callback(int, const char* err_str)
{
    std::cout << "GLFW Error: " << err_str << std::endl;
}
```
Overview

• Introduction to OpenGL
  – What is OpenGL?
  – Shaders
  – OpenGL-related libraries

• Exercise 6
Compiling

• Makefile generation with CMake
  – Readme and scripts for Windows, Linux, OSX

• Backup solution: Use files from 2013
  – CGL homepage → Teaching → Former Courses
1) Mesh setup and initialization

- Setup vertex buffer and index buffer
- Pass data to vertex shader
Mesh representations

• Focus on triangle meshes
• 3D mesh consists of:
  – vertices
  – faces
• Information stored:
  – vertex: position, color, normal, ...
  – face: links to vertices, surface normal, ...
Mesh representations

- Indexed triangle list
  - Stores vertices only once
  - Define triangles by indexing

<table>
<thead>
<tr>
<th>Vertex list</th>
<th>Index list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: (x1, y1, z1)</td>
<td>1</td>
</tr>
<tr>
<td>2: (x2, y2, z2)</td>
<td>2</td>
</tr>
<tr>
<td>3: (x3, y3, z3)</td>
<td>3</td>
</tr>
<tr>
<td>4: (x4, y4, z4)</td>
<td>4</td>
</tr>
</tbody>
</table>
Vertex data structure

• Store vertex data in array

```c
struct Vertex {
    GLfloat pos[4]; //homogeneous coordinates
    GLfloat color[4];
    GLfloat normal[3];
};

Vertex *pVertexArray = new Vertex[numVertices]

//now: fill buffer!
```
Vertex buffers

• Procedure
  – Generate buffer
  – Bind buffer
  – Load data to buffer (vertex array can be destroyed afterwards)

```c
// pVertexArray has data
GLuint handleVBO = 0;
glGenBuffers(1, &handleVBO);
glBindBuffer(GL_ARRAY_BUFFER, handleVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertex)*numVertices,
pVertexArray, GL_STATIC_DRAW);
```

handles need to be stored in global variable in the exercise
Index buffers

• Define faces
  – Here: indexed triangles
  – Same procedure as for vertex buffers
  – Use GLshort array to store indices
  – Use GL_ELEMENT_ARRAY_BUFFER as target
Binding shader inputs

• Binding attributes

```c
location = glGetAttribLocation(shader, "attrib name");
glVertexAttribPointer(location, dimension, GL_FLOAT, GL_FALSE, numBytes, offset);
glEnableVertexAttribArray(location);
```

e.g. 3 for vec3

– Hint: use `offsetof` macro for offset

– Pointers point to the currently bound buffer
Binding shader inputs

- Binding uniforms

```c
location = glGetUniformLocation(shader, "uniform name");
glUniform4fv(location, count, data);
```

dimension (2,3,4) 1 for single constant
Rendering

• Render indexed triangles with vertex and index buffer

```c
void displayFunc(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, handleIndexBuffer);
  glDrawElements(GL_TRIANGLES, idxBufferSize, GL_UNSIGNED_SHORT, 0);
}
```
2) Normals for lighting

- Calculate face and vertex normals
• Face normals
  – Normalized cross product of a and b
Normals

• Vertex normal
  – Average of surrounding face normals
  – Actually better to weight according to angles (optional)
3) Coloring the mesh

- Color vertices depending on height
Color spaces

- **RGB (red, green, blue)**
  - normalized to $[0, 1]$
- **HSV**
  - hue (Farbton)
  - saturation (Sättigung)
  - value (Helligkeit)
- **Transformation HSV to RGB provided**
4) Adding color effects

- Modify shader to highlight point on mesh

interesting point
Questions

leonhard.helminger@disneyresearch.com
References

- https://learnopengl.com/
- https://open.gl/
- http://www.opengl.org/registry