KAIST

CS 380

Introduction to Computer Graphics

LAB (1) : OpenGL Tutorial

2015.03.04

Reference : Foundations of 3D Computer Graphics, Steven J. Gortler

VISUAL COMPUTING Lab

Goals

KAIST

- Understand OpenGL pipeline
- Practice basic OpenGL programming
- Set up a project for OpenGL programs

VISUAL COMPUTING Lab

OpenGL with GLSL

KAIST

- OpenGL
  - Open Graphics Library
  - Cross-language and multi-platform API for rendering vector graphics
- GLSL
  - OpenGL Shading Language
  - Allow application programmers to express the processing that occurs at those programmable points of the OpenGL pipeline

VISUAL COMPUTING Lab

GLUT, GLEW

KAIST

- GLUT : The OpenGL Utility Toolkit
  - To open windows and to allow our program to respond to mouse and keyboard events
- GLEW : The OpenGL Extension Wrangler Library
  - To gain access to the latest features of OpenGL in Windows

VISUAL COMPUTING Lab

OpenGL Installation

KAIST

- Recommended Setup
  - Environment
    - Windows 32 bits or 64 bits
    - Visual Studio (c++)
  - OpenGL, GLUT and GLEW

    (we provide them as *asst0.zip* file)

VISUAL COMPUTING Lab

OpenGL Installation

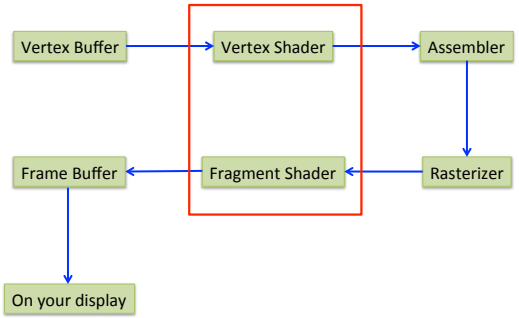
KAIST

- Download 'asst0.zip' and extract it
- After compilation of the project, the debug or release folder would be created on the project folder
- Two files (glut32.dll, glew32.dll) in dll directory should be copied into directory which the exe file is in (debug or release folder)

VISUAL COMPUTING Lab

OpenGL Pipeline

KAIST



VISUAL COMPUTING Lab

Basic OpenGL

KAIST

- APIs

```
#include <GL/glew.h>
#ifdef __MAC__
#   include <GLUT/glut.h>
#else
#   include <GL/glut.h>
#endif
```


VISUAL COMPUTING Lab

Basic OpenGL

KAIST

- Main Program

```
int main(int argc, char **argv) {
  try {
    initGlutState(argc,argv);
    glewInit(); // load the OpenGL extensions

    cout << (g_Gl2Compatible ? "Will use OpenGL 2.x / GLS
    if ((!g_Gl2Compatible) && !GLEW_VERSION_3_0)
      throw runtime_error("Error: card/driver does not su
    else if (g_Gl2Compatible && !GLEW_VERSION_2_0)
      throw runtime_error("Error: card/driver does not su

    initGLState();
    initShaders();
    initGeometry();
    glutMainLoop();
    return 0;
  }
  catch (const runtime_error& e) {
    cout << "Exception caught: " << e.what() << endl;
    return -1;
  }
}
```


VISUAL COMPUTING Lab

Basic OpenGL

KAIST

- Main Program

```
int main(int argc, char **argv) {
  try {
    initGlutState(argc,argv);
    glewInit(); // load the OpenGL extensions

    cout << (g_Gl2Compatible ? "Will use OpenGL 2.x / GLS
    if ((!g_Gl2Compatible) && !GLEW_VERSION_3_0)
      throw runtime_error("Error: card/driver does not su
    else if (g_Gl2Compatible && !GLEW_VERSION_2_0)
      throw runtime_error("Error: card/driver does not su

    initGLState();
    initShaders();
    initGeometry();
    glutMainLoop();
    return 0;
  }
  catch (const runtime_error& e) {
    cout << "Exception caught: " << e.what() << endl;
    return -1;
  }
}
```


VISUAL COMPUTING Lab

Basic OpenGL

KAIST

- initGlutState()

```
static void initGlutState(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);
  glutInitWindowSize(g_width, g_height);          // create a window
  glutCreateWindow("CS 380 : Hello World!!");     // title the window

  // callback functions
  glutDisplayFunc(display);                       // display rendering callback
  glutReshapeFunc(reshape);                       // window reshape callback
  glutMouseFunc(mouse);                           // mouse click callback
  glutKeyboardFunc(keyboard);                     // keyboard callback
}
```


VISUAL COMPUTING Lab

Callback Functions

KAIST

- Whenever some events occur, callback functions catch them and do some actions already defined

```
static void initGlutState(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);
  glutInitWindowSize(g_width, g_height);          // create a window
  glutCreateWindow("CS 380 : Hello World!!");     // title the window

  // callback functions
  glutDisplayFunc(display);                       // display rendering callback
  glutReshapeFunc(reshape);                       // window reshape callback
  glutMouseFunc(mouse);                           // mouse click callback
  glutKeyboardFunc(keyboard);                     // keyboard callback
}
```


VISUAL COMPUTING Lab

Callback Functions

KAIST

- reshape();

```
static void reshape(int w, int h){
    g_width = w;
    g_height = h;
    glViewport(0, 0, w, h);
    glutPostRedisplay();
}
```


VISUAL COMPUTING Lab

Vertex Buffer

KAIST

```
static GLfloat sqPos[6 * 2] = {
    -0.5,  0.5,
    0.2,  0.2,
    0.3,  0.3
};

static GLfloat sqCol[6 * 3] =  {
    1, 1, 1,
    1, 0, 0,
    1, 1, 0,
};

glBindBuffer(GL_ARRAY_BUFFER, posVbo);
glBufferData(
  GL_ARRAY_BUFFER,
  12*sizeof(GLfloat),
  sqPos,
  GL_STATIC_DRAW);
checkGlErrors();


glBindBuffer(GL_ARRAY_BUFFER, colVbo);
glBufferData(
  GL_ARRAY_BUFFER,
  18*sizeof(GLfloat),
  sqCol,
  GL_STATIC_DRAW);
checkGlErrors();
```

Determine the initial coordinates and colors of vertices

Pass the program data to the OpenGL buffer

VISUAL COMPUTING Lab

Callback Functions

KAIST

- Whenever some events occur, callback functions catch them and do some actions already defined

```
static void initGlutState(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);
  glutInitWindowSize(g_width, g_height);          // create a window
  glutCreateWindow("CS 380 : Hello World!!");     // title the window

  // callback functions
  glutDisplayFunc(display);                       // display rendering callback
  glutReshapeFunc(reshape);                       // window reshape callback
  glutMouseFunc(mouse);                           // mouse click callback
  glutKeyboardFunc(keyboard);                     // keyboard callback
}
```


VISUAL COMPUTING Lab

Display

KAIST

```
static void display(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  const ShaderState& curSS = *g_shaderStates[0];
  glUseProgram(curSS.program);

  safe_glUniform1f(curSS.h_uVertexTemp, g_vertexTemp);
  safe_glUniform1f(curSS.h_uColorTemp, g_colorTemp);
  g_square->draw(curSS);

  glutSwapBuffers();

  // check for errors
  checkGlErrors();
}
```

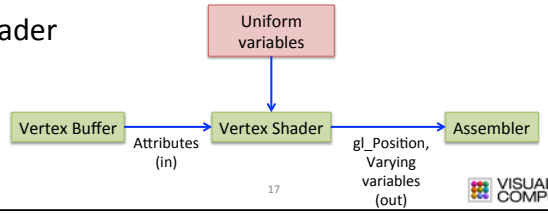
```
void draw(const ShaderState& curSS) {
  int numverts=6;
  safe_glEnableVertexAttribArray(curSS.h_aPosition);
  safe_glEnableVertexAttribArray(curSS.h_aColor);

  glBindBuffer(GL_ARRAY_BUFFER, posVbo);
  safe_glVertexAttribPointer(curSS.h_aPosition,
                             2, GL_FLOAT, GL_FALSE, 0, 0);

  glBindBuffer(GL_ARRAY_BUFFER, colVbo);
  safe_glVertexAttribPointer(curSS.h_aColor,
                             3, GL_FLOAT, GL_FALSE, 0, 0);

  glDrawArrays(GL_TRIANGLES,0,numverts);

  safe_glDisableVertexAttribArray(curSS.h_aPosition);
  safe_glDisableVertexAttribArray(curSS.h_aColor);
}
```


VISUAL COMPUTING Lab

Vertex Shader

KAIST

- Usually final positions of vertices on the screen are determined in the vertex shader
- When *draw* function is called, each vertex data in the vertex buffer passes through the vertex shader

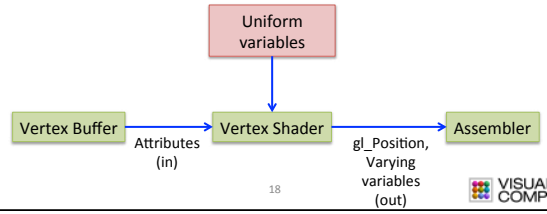


VISUAL COMPUTING Lab

Vertex Shader

KAIST

```
uniform float uVertexTemp;

attribute vec2 aPosition;
attribute vec3 aColor;

varying vec3 vColor;

void main() {
  gl_Position = vec4(aPosition.x, aPosition.y, 0,1);
  vColor = aColor;
}
```

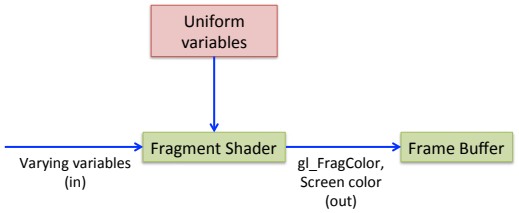


VISUAL COMPUTING Lab

Fragment Shader

KAIST

- Final pixel colors on the screen are determined in the fragment shader

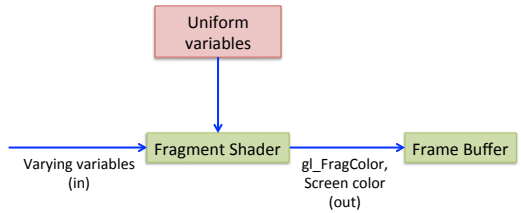


VISUAL COMPUTING Lab

Fragment Shader

KAIST

```
uniform float uColorTemp;

varying vec3 vColor;

void main(void) {
  vec4 color = vec4(vColor.x, vColor.y, vColor.z, 1);
  gl_FragColor = color;
}
```



VISUAL COMPUTING Lab

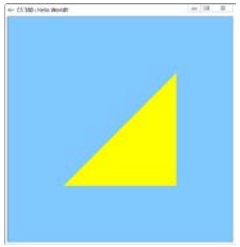
Exercise: Draw the First Polygon

- Vertices are stored in a vertex buffer
- Positions, color and other attributes

```
struct SquareGeometry {
      static GLfloat sqPos[6 * 2] = {
            -0.5,  -0.5,
            0.5,  0.5,
            0.5, -0.5
        };

      static GLfloat sqCol[6 * 3] =  {
            1, 1, 0,
            1, 1, 0,
            1, 1, 0
      };
      ..
}
```

Vertex positions (x, y) of the triangle

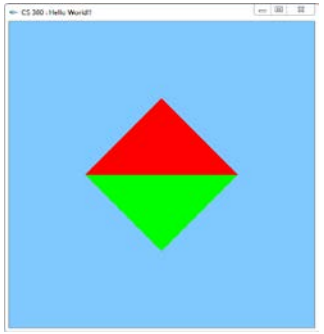
Colors (R,G,B) of the triangle



VISUAL COMPUTING Lab

Exercise: Draw Your Own Polygon

e.g.)



VISUAL COMPUTING Lab

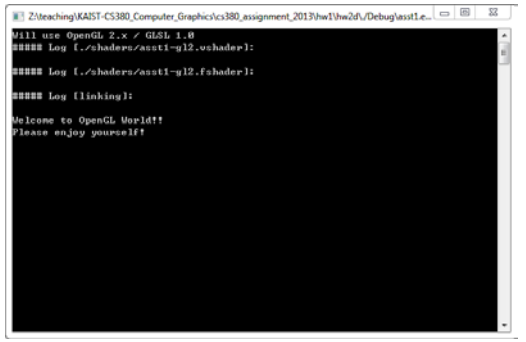
Exercise: Keyboard Input

- Callback functions are provided
- Add the following code into the 'keyboard' function

```
void keyboard(unsigned char key, int x, int y) {
  switch (key) {
  case 'h':
    cout << "Welcome to OpenGL World!\n " << " Please enjoy your self!\n
\n ";
   break;
  }
  glutPostRedisplay();
}
```


VISUAL COMPUTING Lab

Keyboard Input



VISUAL COMPUTING Lab

Mouse Input

KAIST

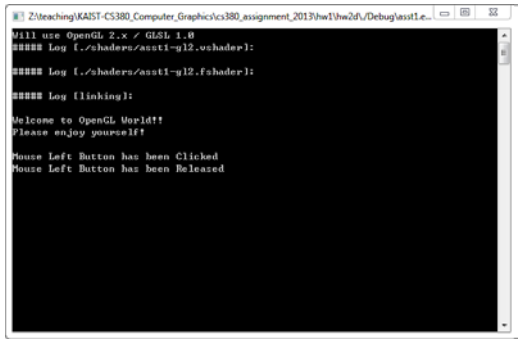
- Callback function are provided
- Add the code into the 'mouse' function

```
static void mouse(int button, int state, int x, int y) {
 if (button == GLUT_LEFT_BUTTON) {
  if (state == GLUT_DOWN) {
     cout << "Mouse Left Button has been Clicked\n";
  }
  else {
     cout << "Mouse Left Button has been Released\n";
  }
 }
glutPostRedisplay();
}
```


VISUAL COMPUTING Lab

Mouse Input

KAIST



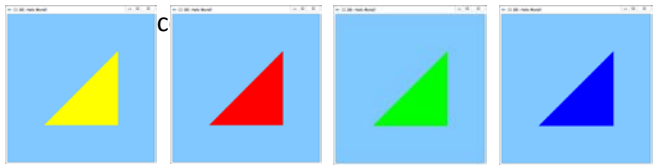
VISUAL COMPUTING Lab

Exercise

KAIST

YOU SHOULD MODIFY THE SHADER FILE

- If you press 'r', 'g' or 'b', the color of the polygon should be changed to red, green or blue.
- Then, if you click 'p', the color should be returned to



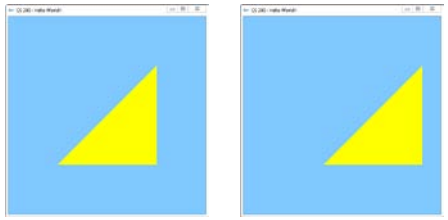
original r g b

VISUAL COMPUTING Lab

Exercise

KAIST

- If you click the 'mouse left or right button', then a polygon should move to left or right.



original right

VISUAL COMPUTING Lab

Hint

KAIST

- In vertex shader, we can determine the final positions of vertices
- In fragment shader, we can determine the final colors of vertices
- You should modify the uniform variables provided in the shader file

```
static float g_vertexTemp  = 0.0;      // for vertex moving
static float g_colorTemp   = 0.0;      // for changing color
```


VISUAL COMPUTING Lab