Computer Graphics

Programming I

Contents in Programming Courses

- Releasing programming assignments
- Clarifying requirements of the programming assignments
- Giving hints for programming assignments

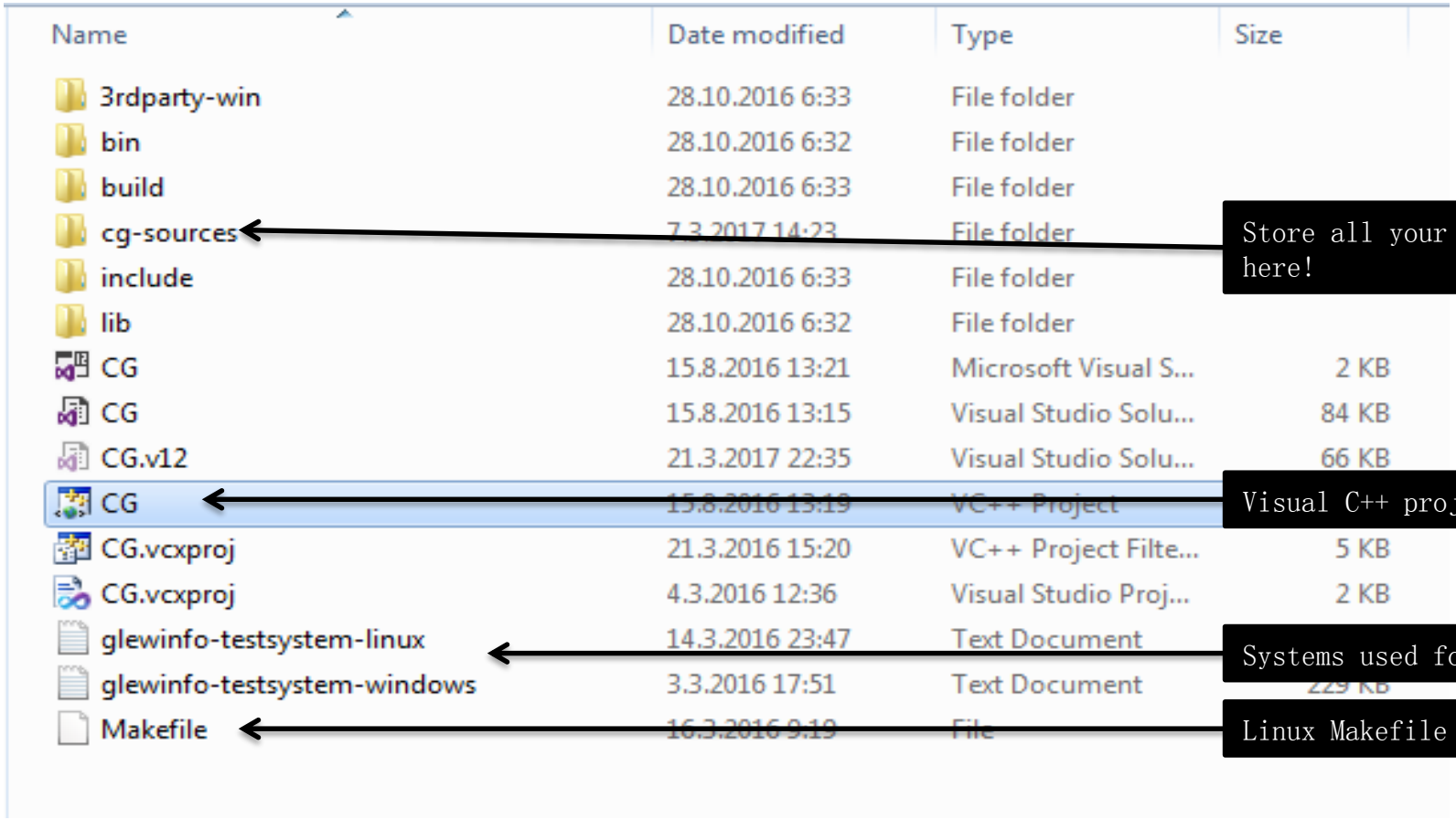
Programming Assignment 1

- Grading
  - Total points as 8, and 5 to pass the assignment.
  - Passing the programming assignment is necessary for passing the course
  - Ask for extension in advance if needed
- Contents
  - Basics, object rendering and selection
- Deadline 2019-04-09 24:00

How programming assignments are organized

- Example package is given
  - Download zip file from *Assignments* page in Noppa
  - Multiple examples in a simple framework
- Get example codes running first and base your own work on them
  - You are allowed to reuse that code in your own work!
  - Quick way to get started

Sample package contents

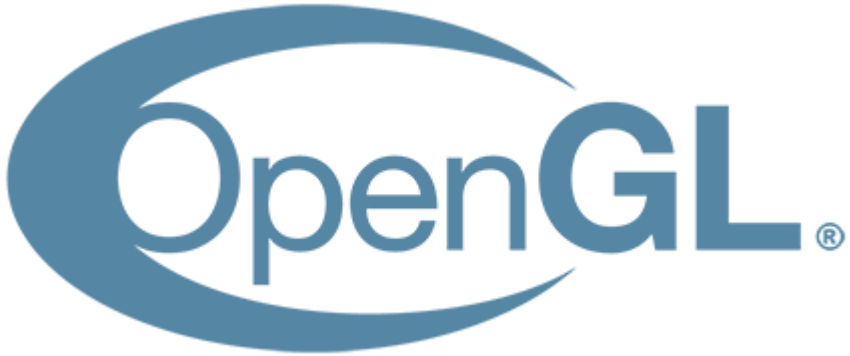


Modern OpenGL VS. Old OpenGL

- We use **modern** OpenGL (3.3+) in programming assignments
- Many tutorials on the net teach old OpenGL
  - They are **NOT compatible** with what we teach in this year
    - They assume a fixed function pipeline, no shaders
  - **Avoid** tutorials and examples that contain function calls like **glBegin(), glEnd(), glVertex3f()** etc.
- Modern tutorials have been collected in ”Common additional material” section in Noppa
  - Find more with search engines
  - Use official OpenGL and GLSL references
  - OpenGL.org has very good information in wiki pages

Dependent Libraries

- Graphics Library: **OpenGL 3.3** or **higher**
- OpenGL Context: SDL2
- OpenGL extensions: GLEW
- Matrix operations: GLM



Requirements

- Get included examples to work
  - Download, build, and run example code package.
- Render a simple cube
  - Modify the provided code framework to render the cube composed of 12 triangles.
- Change the cube color by mouse clicking
  - Handle mouse clicking event to update the color value of the cube
  - Modify the fragment shader to set the color property

How to do it?

- Modify and submit these files:
  - **assignment1.cpp** and **assignment1.h**: main files for rendering
  - **data/assignment1.fs** and **data/assignment1.vs**: for your own color rendering.

How to do it?

- Define your cube here:

```
void assignment1::createCube()
{
// Define our object. Note that after it has been stored in GPU memory,
source buffers could be freed as long as we remember how many
// vertices we want to to render from it

// for example, the first triangle on front face v1->v3->v4
cube.push_back(Vertex(-1.0f, -1.0f, 1.0f, 1.0f, 0.0f, 0.0f));
cube.push_back(Vertex(1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f));
cube.push_back(Vertex(-1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f));

}
```

- The cube data will be copied to the OpenGL buffer for rendering later

```
bool assignment1::init()
{
...

createCube();

...
glBufferData(GL_ARRAY_BUFFER, cube.size() * sizeof (struct Vertex), &cube[0],
GL_STATIC_DRAW);
```

How to do it?

- Define your mouse event handler here:

```
bool assignment1::handleEvent(const SDL_Event &e)
{
        // Put any event handling code here.
        // Window-resizing is handled in event loop already.

// Return false if you want to stop the program
return true;
}
```

How to do it?

- Render and update the color of your cube here:

```
// Render view
void assignment1::render()
{
// Clear background
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Calculate model transformation
modelMat = glm::mat4(1.0f);
modelMat = glm::translate(modelMat, glm::vec3(1.0, 0.0, 0.0)); // Translate object +1 on x-
axis after rotation
modelMat = glm::rotate(modelMat, rotation, glm::vec3(0.0, 1.0, 0.0)); // Rotate object
around y-axis
modelMat = glm::scale(modelMat, glm::vec3(0.5f, 0.5f, 0.5f));

// Precalculate transformation matrix for the shader and use it
mvpMat = projectionMat * viewMat * modelMat;
glUniformMatrix4fv(glGetUniformLocation(shaderProgram.getShaderProgram(), "mvpmatrix"), 1,
GL_FALSE, glm::value_ptr(mvpMat));

// Draw individual triangles when we have correct VBO in use
// We are not using cube vector data here at all. We need to just know the number of
vertices to draw!
glDrawArrays(GL_TRIANGLES, 0, static_cast<GLsizei>(cube.size()));
}
```