CSE 167

Discussion 04 ft. Timothy
10/23/2017

Announcements

- Midterm next Tuesday(10/31)
  - Sample midterms are up
- Project 2 grading this Friday

Contents

- Common errors
- Lighting
  - Some clarifications
  - Phong shading model
  - Implementation tips

Common errors

- Parsing errors
  - If you are getting more than actual number of faces
- Trackball mapping errors
  - If you (think you) implemented everything correctly but the movement is little off when you try to move it around
- Centering and scaling errors
  - If you did make translation and scale matrices but nothing changes after multiplying them to toWorld
- Shader errors
  - When everything is (in your opinion) correct but nothing or something weird is showing up

Common errors: parsing

- If you are getting more than actual number of faces
  - You probably did not take care of empty lines or the lines that start with ‘#’
  - Make sure that the number of indices that you parsed is 3 * num_faces

Common errors: trackball mapping

- If you (think you) implemented everything correctly but the movement is little off
  - Make sure you are updating old_cursor_position once you are done rotating/translating
  - You should be remembering old_cursor_position as 2D coordinate not trackball-mapped 3D coordinate!!!
  - You have to multiply a small number to the amount of rotation/translation to make it follow your cursor smoothly

Common errors: centering and scaling

- If you did make translation and scale matrices but nothing changes after multiplying them to toWorld
  - Whenever you click on F1, F2, or F3, all of the models need to be centered and scaled
  - So remember the centering/translation matrix AND scaling matrix
  - Each time you press F1, F2, or F3, just set the toWorld = T * S * glm::mat4(1.0f)

Common errors: shader

- When everything is (in your opinion) correct but nothing or something weird is showing up
  - Case I : if you implemented vertices and normals as **vector<glm::vec3>** and indices as **vector<GLuint>**

```
Cube::Cube()
{
	toWorld = glm::mat4(1.0f);

	glGenVertexArrays(1, &VAO);
	glGenBuffers(1, &VBO);
	glGenBuffers(1, &EBO);

	glBindVertexArray(VAO);

	glBindBuffer(GL_ARRAY_BUFFER, VBO);

	glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * vertices.size(), vertices.data(),
GL_STATIC_DRAW);
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE,3 * sizeof(GLfloat),(GLvoid*)0);

	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(GLuint) * indices.size(), indices.data() or
&indices[0], GL_STATIC_DRAW);

	glBindBuffer(GL_ARRAY_BUFFER, 0);
	glBindVertexArray(0);
}
```

```
void Cube::draw(GLuint shaderProgram)
{
	glm::mat4 modelview = Window::V * toWorld;

	uProjection = glGetUniformLocation(shaderProgram, "projection");
	uModelview = glGetUniformLocation(shaderProgram, "modelview");

	glUniformMatrix4fv(uProjection, 1, GL_FALSE, &Window::P[0][0]);
	glUniformMatrix4fv(uModelview, 1, GL_FALSE, &modelview[0][0]);

	glBindVertexArray(VAO);

	glDrawElements(GL_TRIANGLES,??, GL_UNSIGNED_INT, 0);

	glBindVertexArray(0);
}
```

Common errors: shader

- When everything is (in your opinion) correct but nothing or something weird is showing up
  - Case II : if you implemented vertices and normals as **vector<float>** and indices as **vector<GLuint>**

```
Cube::Cube()
{
	toWorld = glm::mat4(1.0f);

	glGenVertexArrays(1, &VAO);
	glGenBuffers(1, &VBO);
	glGenBuffers(1, &EBO);

	glBindVertexArray(VAO);

	glBindBuffer(GL_ARRAY_BUFFER, VBO);

	glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * vertices.size(), vertices.data() or
&vertices[0], GL_STATIC_DRAW);
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE,3 * sizeof(GLfloat),(GLvoid*)0);

	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(GLuint) * indices.size(), indices.data ()
or &indices[0], GL_STATIC_DRAW);

	glBindBuffer(GL_ARRAY_BUFFER, 0);
	glBindVertexArray(0);
}
```

```
void Cube::draw(GLuint shaderProgram)
{
	glm::mat4 modelview = Window::V * toWorld;

	uProjection = glGetUniformLocation(shaderProgram, "projection");
	uModelview = glGetUniformLocation(shaderProgram, "modelview");

	glUniformMatrix4fv(uProjection, 1, GL_FALSE, &Window::P[0][0]);
	glUniformMatrix4fv(uModelview, 1, GL_FALSE, &modelview[0][0]);

	glBindVertexArray(VAO);

	glDrawElements(GL_TRIANGLES,??, GL_UNSIGNED_INT, 0);

	glBindVertexArray(0);
}
```

Lighting: some clarifications

- Three different materials: one for each model
  - Read the write-up carefully, each material has different requirement!!!
- Lighting is per-pixel lighting, so you should do it **in fragment shader**
- Light models(cone, sphere) need not be huge: make them look cute
- For simplicity, we changed the write-up such that you only need to define only one color for a light(no diffuse/specular/ambient breakdown)

A lot of chalkboard time...

Lighting: Phong shading model, light

- How to determine c
- Directional light
  - Color
  - Direction
- Point light
  - Color
  - Position
  - Attenuation(linear)
- Spotlight
  - Color
  - Position
  - Direction
  - Cutoff, exponent
  - Attenuation(quadratic)

```
struct Light{
        int light_mode;

        vec3 light_color;
        vec3 light_pos;
        vec3 light_dir;

        float cons_att;
        float linear_att;
        float quad_att;

        float cutoff_angle;
        float exponent;
};
```

Lighting: Phong shading model, light

- How to determine c
- Directional light
  - Color
  - Direction
- Point light
  - Color
  - Position
  - Attenuation(linear)
- Spotlight
  - Color
  - Position
  - Direction
  - Cutoff, exponent
  - Attenuation(quadratic)

```
struct Light{
        int light_mode;

        vec3 light_color;
        vec3 light_pos;
        vec3 light_dir;

        float cons_att;
        float linear_att;
        float quad_att;

        float cutoff_angle;
        float exponent;
};
```

Lighting: Phong shading model, material

- The color at a point is: $c_d + c_s + c_a$
- Diffuse
  - cd = cl · kd(n · L)
- Specular
  - cs = cl · ks(R · L)^p
- Ambient
  - ca = cl · ka

```
struct Material{
        int object_mode;

        vec3 color_diff;
        vec3 color_spec;
        vec3 color_ambi;
};
```

- What is the difference between c and L?
- What is the datatype of each constant and term?
- What should be the datatype of color that is the output of fragment shader?
- How do we determine k's and c? Reference link

Lighting: Phong shading model, light+model

(1) Calculate c_l
(2) Calculate c_d, c_s, c_a
(3) Multiply attenuation to each term(for point and spot lights only)
(4) Add the three terms
(5) Pass as glm::vec4!!!

Lighting: implementation tips

- Make light and material classes for cleaner coding
  - Red is in Light class, blue is in fragment shader, green is in Window.cpp

```
class Light{
        int mode;
        …

        void draw();
        ...

};
```

```
void Light::draw()
{
        …
        light_mode = glGetUniformLocation(program,
        “Light.light_mode”);

        glUniform1i(light_mode, mode);
        ...
}
```

```
struct Light{
        int light_mode;

        vec3 light_color;
        vec3 light_pos;
        vec3 light_dir;

        float cons_att;
        float linear_att;
        float quad_att;

        float cutoff_angle;
        float exponent;
};
```

```
...
Light* light_ptr;
Light dir_light(0);
Light point_light(1);
...
```

```
// How are you going to pass the shader program to
// be used?

light_ptr->draw();
```

Lighting: implementation tips

- Transform lights just like you transformed your objects
  - For example, when you want to rotate directional light, you just need to rotate the directional light's direction and assign the new direction to that light.
- Pass lights' information to the shader just like you passed your objects' information
  - Treat lights as objects
  - Since the lights don't need to be "lighted" and will not be lighted because of the normals, you can set the ambient color for the light models to be something high

Lighting: implementation tips

- Vertex and gl_Position
  - Vertex = vec3(model * vec4(position, 1.0f))
  - gl_Position = projection * view * model * vec4(position, 1.0f);
  - What is the difference between Vertex and gl_Position?
  - Which variable should you pass to fragment shader to calculate the color?
- Normal
  - What is the problem if you just pass normal values as they are?
  - How do you pass “correct” normal values to fragment shader?
  - Solution: Normal = mat3(transpose(inverse(model))) * normal;
  - Why? Chalkboard time! reference