课程名称：计算机图形学　　**指导教师**：王振武

班级：计科 19-2　　**姓名**：王凌峰　　**学号**：1910630221

实验项目名称：

7. 纹理映射

实验目的及要求：

完成纹理映射

实验内容（方法和步骤）：

把某种墙的纹理映射到球面上。

实际上 OpenGL 的纹理映射是在 rasterization 过程中发生的，rasterization 阶段产生的 fragments 被送入 fragment shader 处理，像素的颜色也在此阶段决定。因此和实验 1 ~ 实验 3 一样，这里也只是纹理映射的模拟。

目录结构：

```
texture/
|-- image.cc
|-- main.cc
|-- makefile
|-- point.cc
|-- point.hh
|-- rgb.cc
|-- rgb.hh
|-- shader.cc
|-- shader.hh
|-- texture.cc
|-- texture.hh
|-- trivial.frag
|-- trivial.vert
|-- wall.jpg
```

为了获得与每个点关联的颜色，fragment shader 中声明 `in vec3 texColor`，颜色直接从 `texColor` 中取出：

```
#version 330 core
in vec3 texColor;
out vec4 color;
void main()
{
    color = vec4(texColor,1.0f);
}
```

code 1: `trivial.frag`

fragment shader 的变量 `texColor` 由 vertex shader 传入：

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;

out vec3 texColor;

uniform mat4 M;
uniform mat4 M_cam;

const int windowWidth = 1920;
const int windowHeight = 1028;
const float depth = 2000;
const float l = - (windowWidth - 1) / 2.0f;
const float r = (windowWidth - 1) / 2.0f;
const float b = - (windowHeight - 1) / 2.0f;
const float t = (windowHeight - 1) / 2.0f;
const float n = 0;
const float f = -depth;
const mat4 M_ortho_proj = mat4(
    vec4(2.0f/(r - l),0.0f,0.0f,0.0f),
    vec4(0.0f,2.0f/(t - b),0.0f,0.0f),
    vec4(0.0f,0.0f,2.0f/(n - f),0.0f),
    vec4((r + l)/(l - r),(t + b)/(b - t),(n + f)/(f - n),1.0f)
);

```

```
26 void main()
27 {
28     gl_Position = M_ortho_proj * M_cam * vec4(aPos,1.0f);
29     texColor = aColor;
30 }
```

code 2: trivial.vert

图片 wall.jpg 的读取用 stb 库。该库是 header-only library，其要求存在一个 .cc 文件包含 stb 库相应头文件的实际实现：

```
1 #define STB_IMAGE_IMPLEMENTATION
2 #include <stb/stb_image.h>
```

code 3: image.cc

这个实验使用了 point 和 rgb 类，其中 point 类与之前实验的类似，只是扩展到了三维：

```
1  #ifndef POINT_HH
2  #define POINT_HH
3
4  #include <glbinding/gl/gl.h>
5  #include <glbinding/glbinding.h>
6
7  class point
8  {
9      public:
10         gl::GLfloat x;
11         gl::GLfloat y;
12         gl::GLfloat z;
13         point(float xx,float yy,float zz):x(xx),y(yy),z(zz) {}
14 };
15 bool lessf(const point &p1,const point &p2);
16
17 #endif //POINT_HH
```

code 4: point.hh

```cpp
#include "point.hh"
bool lessf(const point &p1,const point &p2) {
    if (p1.x < p2.x) {
        return true;
    }
    else if (p1.x == p2.x) {
        if (p1.y < p2.y) {
            return true;
        }
        else if (p1.y == p2.y) {
            return p1.z < p2.z;
        }
        return false;
    }
    return false;
}
```

code 5: point.cc

rgb 类只是颜色分量的集合：

```cpp
#ifndef RGB_HH
#define RGB_HH

#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

class rgb
{
    public:
        gl::GLfloat r;
        gl::GLfloat g;
        gl::GLfloat b;
        rgb(float rr,float gg,float bb) : r(rr),g(gg),b(bb) {}
        rgb() = default;
};

#endif //RGB_HH
```

code 6: `rgb.hh`

```
#include "rgb.hh"
```

code 7: `rgb.cc`

`rasterize_with_texture` 将球心坐标 $\vec{o}$，半径 300 的球体 rasterize，并根据每个像素的位置 (φ, θ) 映射到纹理图片，取出相应颜色与该像素绑定，返回所有像素点及与其对应的颜色。

`texture.cc` 中用的是粗糙的 rasterization 的方法。对 $\varphi \in [0, 2\pi]$ 及 $\theta \in [0, \pi]$ 分别以某个步长遍历，得到整个球面上采样点的坐标。步长越小，球面就显示得越精细。这个实验中 $\Delta\theta$ 和 $\Delta\varphi$ 都为 $0.2°$，效果可以接受。

图片上点的自由度为 2, 球面上点的自由度也是 2, 可以方便地建立映射。为了更方便，使用 $row = A \cdot \theta + C_1, col = B \cdot \varphi + C_2$ 的形式。

令 $C_1 = C_2 = 0, A \cdot \theta_{max} = height, B \cdot \frac{1}{2} \cdot \varphi_{max} = width$，其中 $width$ 和 $height$ 分别为纹理图片以像素为单位的宽度和高度。则每半个球面分布一张纹理图片。

```
#ifndef TEXTURE_HH
#define TEXTURE_HH

#include "rgb.hh"
#include "point.hh"

#include <cmath>
#include <map>

//glm
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

gl::GLfloat *rasterize_with_texture(const unsigned char *image,int width,int
↪  height,size_t &cnt);
rgb map_to_tex(const unsigned char *image,int width,int height,float
↪  theta,float phi);
```

```

#endif //TEXTURE_HH
```

code 8: texture.hh

```
#include "texture.hh"

using namespace gl;
using namespace std;

GLfloat *rasterize_with_texture(const unsigned char *image,int width,int height,size_t &cnt)
{
    //rasterize a sphere with radius 300 and center ō
    const float r(300);
    //control the grainularities in 2 directions
    const float delta_phi(glm::radians(0.2f));
    const float delta_theta(glm::radians(0.2f));

    map<point, rgb, decltype(lessf)*> points(lessf);
    for (float phi = 0; phi < glm::radians(360.0f); phi += delta_phi) {
        for (float theta = 0; theta < glm::radians(180.0f); theta += delta_theta) {
            const point p(
                r * sin(theta) * cos(phi),
                r * sin(theta) * sin(phi),
                r * cos(theta)
            );
            //map to texture coordinate from spherical coord.
            points[p] = map_to_tex(image,width,height,theta,phi);
        }
    }

    cnt = points.size();
    auto retval = new GLfloat[6 * cnt];
    size_t i(0);

#define ADD(M) retval[i++]=M
#define ADD_POINT(X,Y,Z,R,G,B) ADD(X);ADD(Y);ADD(Z);ADD(R);ADD(G);ADD(B);
```

```
33
34      for (const auto &p : points) {
35          ADD_POINT(p.first.x, p.first.y, p.first.z, p.second.r, p.second.g,
            ↪  p.second.b);
36      }
37
38      return retval;
39  }
40  rgb map_to_tex(const unsigned char *image,int width,int height,float
    ↪  theta,float phi)
41  {
42      //A · θ_max = height
43      const int A(height / glm::radians(180.0f));
44      //B · φ_max/2 = width
45      const int B(width / glm::radians(180.0f));
46
47      const int row = static_cast<int>(A * theta) % height;
48      const int col = static_cast<int>(B * phi) % width;
49      const int pos = 3 * (row * width + col);
50      return rgb(image[pos]/255.0,image[pos+1]/255.0,image[pos+2]/255.0);
51  }
```

code 9: texture.cc

camera transformation 与三维图形变换实验中用到的类似，以 $\vec{o}$ 为圆心在 xOz 平面上以半径 400 绕 y 轴旋转。

```
1  #include "shader.hh"
2  #include "point.hh"
3  #include "rgb.hh"
4  #include "texture.hh"
5
6  #include <iostream>
7  //disable inclusion of the development environment header
8  #define GLFW_INCLUDE_NONE
9  #include <GLFW/glfw3.h>
10 //glbinding
11 #include <glbinding/gl/gl.h>
12 #include <glbinding/glbinding.h>
```

```cpp
//image loading
#include <stb/stb_image.h>

using namespace std;
using namespace gl;

GLFWwindow *initWindow();
glm::mat4 gen_M_cam(const glm::vec3 &center);

const unsigned windowWidth(1920);
const unsigned windowHeight(1028);

int main()
{
    auto wd = initWindow();
    glbinding::initialize(glfwGetProcAddress);
    glEnable(GL_DEPTH_TEST);

    shader prog("trivial.vert","trivial.frag");

    //load image data
    int width,height,channel;
    const auto image = stbi_load("wall.jpg", &width, &height, &channel, 0);

    //rasterize a sphere with texture
    size_t cnt;
    const auto vert = rasterize_with_texture(image,width,height,cnt);
    stbi_image_free(image);

    GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    GLuint vbo;
    glGenBuffers(1,&vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT)*cnt*6, vert,
    ↪  GL_STATIC_DRAW);
    delete [] vert;
```

```
51
52      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GL_FLOAT),(void
        ↪  *)0);
53      glEnableVertexAttribArray(0);
54      //every 3D point comes with RGB color
55      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GL_FLOAT),(void
        ↪  *)(3*sizeof(GL_FLOAT)));
56      glEnableVertexAttribArray(1);
57
58      //unbind
59      glBindBuffer(GL_ARRAY_BUFFER, 0);
60      glBindVertexArray(0);
61
62      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
63      while(!glfwWindowShouldClose(wd))
64      {
65          if (glfwGetKey(wd, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
66              glfwSetWindowShouldClose(wd, true);
67          }
68
69          glClearColor(1,1,1,1);
70          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
71
72          //get new camera transformation, center at ō
73          const auto M_cam = gen_M_cam(glm::vec3(0,0,0));
74
75          prog.use();
76          prog.setMat4("M_cam", M_cam);
77
78          glBindVertexArray(vao);
79          //draw as points as to simulate the process of texturing
80          glDrawArrays(GL_POINTS, 0, cnt);
81
82          glfwSwapBuffers(wd);
83          glfwPollEvents();
84      }
85
86      glDeleteVertexArrays(1, &vao);
87      glDeleteBuffers(1, &vbo);
```

```
88
89      glfwTerminate();
90  }
91  glm::mat4 gen_M_cam(const glm::vec3 &center)
92  {
93      //eye position
94      const float radius = 400;
95      const auto time = glfwGetTime();
96      //eye rotates along a circle whose plane is perpendicular to y axis
97      const auto e = glm::vec3(
98          center.x + radius*sin(time),
99          center.y,
100         center.z + radius*cos(time)
101     );
102     //gaze direction
103     const auto target = center;
104     const auto g = target - e;    //always gaze at the center of the cuboid
105     //view-up vector
106     const auto t = glm::vec3(0,1,0);
107     //construct (u,v,w) basis
108     const auto w = - glm::normalize(g);
109     const auto u = glm::normalize(glm::cross(t, w));
110     const auto v = glm::cross(w, u);
111     //matrix to align (u,v,w) to (x,y,z)
112     glm::mat4 M_cam_rot(
113         glm::vec4(u,0),    //first col
114         glm::vec4(v,0),    //second col
115         glm::vec4(w,0),    //third col
116         glm::vec4(0,0,0,1)    //translation col
117     );
118     M_cam_rot = glm::transpose(M_cam_rot);
119     //matrix to translate e to o
120     const glm::mat4 M_cam_transl(
121         glm::vec4(1,0,0,0),
122         glm::vec4(0,1,0,0),
123         glm::vec4(0,0,1,0),
124         glm::vec4(-e,1)
125     );
126     //construct M_cam
```

```
127     return M_cam_rot * M_cam_transl;
128 }
129 GLFWwindow *initWindow()
130 {
131     if (!glfwInit()) {
132         std::cerr << "init failed." << std::endl;
133     }
134     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
135     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
136     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
137 
138     auto w = glfwCreateWindow(windowWidth, windowHeight, "tAsK", nullptr,
        ↪  nullptr);
139     if (!w) {
140         std::cerr << "window creation failed" << std::endl;
141         exit(-1);
142     }
143     glfwMakeContextCurrent(w);
144 
145     return w;
146 }
```

code 10: `main.cc`

实验结果与分析:

图 1: 某种墙的纹理

图 2: 效果

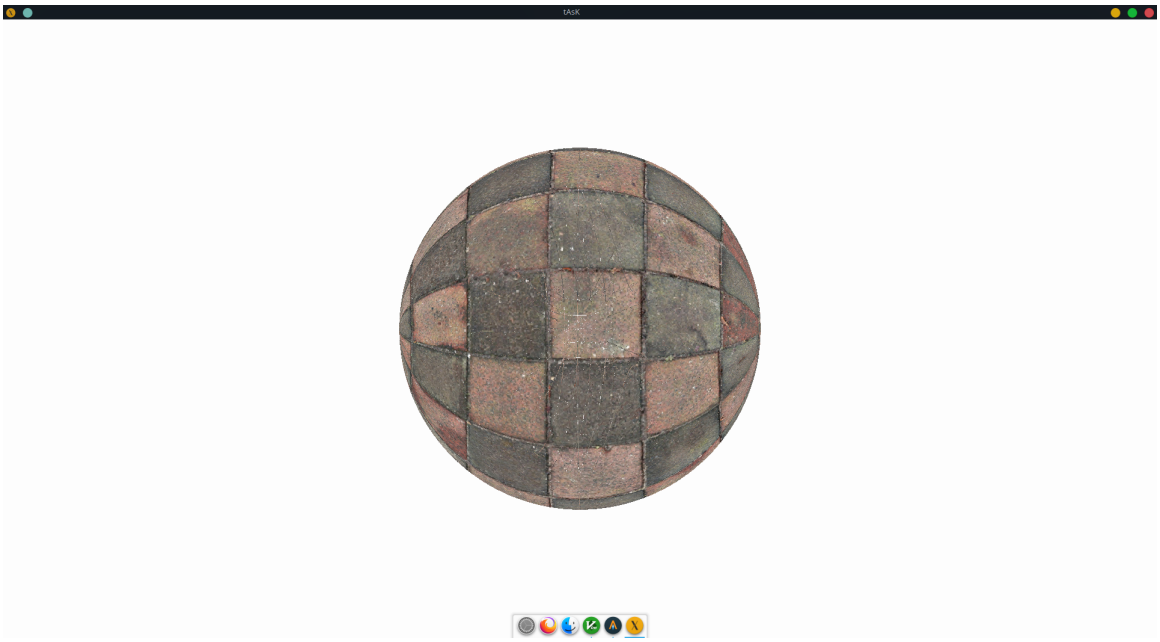

图 3: 另一个角度的效果

成绩: **批阅教师签名**: **年 月 日**