作业4 比较不同的小球绘制方式

18340012 陈晨

1 实验题目

- 使用 VBO 对作业 3 小球进行绘制，使用足够的细分产生充足的顶点和三角面片，便于计算绘制时间。
- 讨论是否使用 VBO 的效率区别。
- 讨论是否使用 index array 的效率区别。

2 实验过程

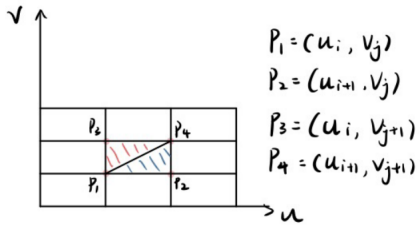
根据题目要求，设计三种绘制小球的方式：不采用VBO，采用VBO，采用EBO。

- 绘制小球

  绘制小球的思路与作业3相同：

  根据球面的数学模型，可以得到角度组(u,v)对应的球面坐标。

  遍历不同的（u,v），得到对应球面坐标，通过画三角形近似球面。易知下面四个点共面。



  在一次采样中，依次画出（P1，P4，P3）和（P1，P3，P2）两个三角形。

  根据(u,v)获取球面坐标的函数如下：

```
GLfloat* get_point_on_ball(GLfloat u, GLfloat v, GLfloat r) {
    GLfloat* points = new GLfloat[3];
    points[0] = r * sin(M_PI * u) * cos(2 * M_PI * v);//x
    points[1] = r * sin(M_PI * u) * sin(2 * M_PI * v);//y
    points[2] = r * cos(M_PI * u);//z
    return points;
}
```

- 绘图函数的相同部分

  每个绘图函数内部，计时的方式如下所示：

```
LARGE_INTEGER  freq, start, end;

QueryPerformanceCounter(&start);
...//绘图
QueryPerformanceCounter(&end);
```

  在初始化的时候，获得计时频率：

```
void MyGLWidget::initializeGL()
{
    initializeOpenGLFunctions();
    glViewport(0, 0, width(), height());
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    QueryPerformanceFrequency(&freq);
}
```

主绘图函数paintGL由以下两个部分组成。

```
void MyGLWidget::paintGL()
{
    ...//属性设置，各种矩阵的初始化等

    ...//计算与打印计时
}
```

第一个部分的实现如下：

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
glEnable(GL_CULL_FACE);
glFrontFace(GL_CW);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f, (GLfloat)width() / (GLfloat)height(), 0.1f,
200.0f);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -5.0f);

glColor3f(1.0f, 1.0f, 0.0f);
```

第二个部分的实现如下。为了比较性能，这边进行多次更新，再去计时的平均值。由于最开始几次更新的数值很不稳定，因此这里从第十次调用update开始计算。

```
draw();//当前测试的绘制小球函数

static int cnt = 0;
static double total_time = 0.0;
double current_time;
cnt += 1;
if (cnt > 10){
    current_time = (__int64)(end.QuadPart - start.QuadPart) * 1000 /
(double)freq.QuadPart;
    total_time += current_time;
    printf(" %lfms\n\n", current_time);
    if (cnt % 10 == 0) {
        printf(" cnt = %d, avg_time = %lfms.\n\n", (cnt - 10),
total_time / (cnt - 10));
    }
}
```

这一部分对于三种绘制小球方式都是一样的。

- 不采用VBO

  不采用VBO的情况下，三角形通过以下方式画出：

```
void MyGLWidget::draw1() {
    GLfloat* ball = new GLfloat[6 * 3 * u_num * v_num];
    create_ball(ball);
    QueryPerformanceCounter(&start);
    glBegin(GL_TRIANGLES);
    for (int i = 0; i < 6 * 3 * u_num * v_num; i += 3) {
        glVertex3f(ball[i], ball[i + 1], ball[i + 2]);
    }
    glEnd();
    QueryPerformanceCounter(&end);
    delete[] ball;
}
```

其中球面坐标数组ball是对于球面采样u_num * v_num次，每次得到6个点（对应两个三角形），每个点包含三维坐标。因此，计算球面坐标数组的函数如下所示：

```
void MyGLWidget::create_ball(GLfloat* ball) {
    GLfloat u_step = 1.0f / u_num;
    GLfloat v_step = 1.0f / v_num;
    GLuint offset = 0;
    for (int u = 0; u < u_num; u++) {
        for (int v = 0; v < v_num; v++) {
            GLfloat* point_1 = get_point_on_ball(u * u_step, v * v_step,
r);
            GLfloat* point_2 = get_point_on_ball((u + 1) * u_step, v *
v_step, r);
            GLfloat* point_3 = get_point_on_ball((u + 1) * u_step, (v +
1) * v_step, r);
            GLfloat* point_4 = get_point_on_ball(u * u_step, (v + 1) *
v_step, r);

            GLfloat* points[6] = { point_1, point_4, point_3, point_1,
point_3, point_2 };
            for (int i = 0; i < 6; i++) {
                memcpy(ball + offset, points[i], sizeof(GLfloat) * 3);
                offset += 3;
            }
            delete[] point_1, point_2, point_3, point_4;
        }
    }
}
```

- 采用VBO

  采用VBO的情况下，需要初始化VBO和VAO，并且使用glDrawArrays进行绘制。

```
void MyGLWidget::draw2() {
    //创建球的数组
    GLfloat* ball = new GLfloat[6 * 3 * u_num * v_num];
    create_ball(ball);
    init_vbo(ball);
    QueryPerformanceCounter(&start);
    glDrawArrays(GL_TRIANGLES, 0, 6 * u_num * v_num);
    QueryPerformanceCounter(&end);
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    delete[] ball;
}
```

初始化VBO，VAO的方式如下。顶点数据（三维坐标和法向量）存储在VBO中。通过glGenBuffers函数生成VBO的id并通过glBindBuffer绑定，便创建了VBO。然后调用glBufferData函数，把通过create_ball函数得到的数据复制到当前绑定的缓冲中。VAO存储了顶点属性相关的信息。通过glGenVertexArrays函数生成VAO的id，通过glBindVertexArray进行绑定，以生成VAO。

最后链接顶点属性。由于顶点数据为三维坐标，所以一个顶点包含了3个GLfloat数据。通过glVertexAttribPointer和glEnableVertexAttribArray指定坐标属性的位置。

```
void MyGLWidget::init_vbo(GLfloat* ball) {
    //创建VBO，VAO
    glGenBuffers(1, &VBO);
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * 6 * 3 * u_num *
v_num, ball, GL_STATIC_DRAW);

    //链接顶点属性
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat),
(void*)0);
    glEnableVertexAttribArray(0);
}
```

由于采用VBO的情况下，也是三个点画一个三角形，一次取样取六个点，因此计算球面坐标数组ball的函数与不采用VBO的相同。

- 采用EBO

采用EBO的情况下，需要初始化VBO，VAO和EBO，并且使用glDrawElements进行绘制。由于每次采样其实只有4个不重复的点，因此这里对于ball所申请分配的内存会比前面两种要小。需要额外创建一个索引数组，其中的索引值对应于ball数组的下标，从0开始。

```
void MyGLWidget::draw3() {
    //创建球的数组
    GLfloat* ball = new GLfloat[4 * 3 * u_num * v_num];
    create_ball_with_indices(ball);
    //创建索引数组
    GLint* indices = new GLint[6 * 3 * u_num * v_num];
    for (GLint i = 0, j = 0; i < 6 * 3 * u_num * v_num; i += 6, j += 4)
{
        GLint index[6] = { j, j + 3, j + 2, j, j + 2, j + 1 };
        memcpy(indices + i, index, sizeof(GLint) * 6);
    }
    init_vbo_with_indices(ball, indices);
```

```
    QueryPerformanceCounter(&start);
    glDrawElements(GL_TRIANGLES, 6 * u_num * v_num, GL_UNSIGNED_INT, 0);
    QueryPerformanceCounter(&end);
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);
    delete[] ball;
    delete[] indices;
}
```

初始化VBO，VAO和EBO的方式如下。

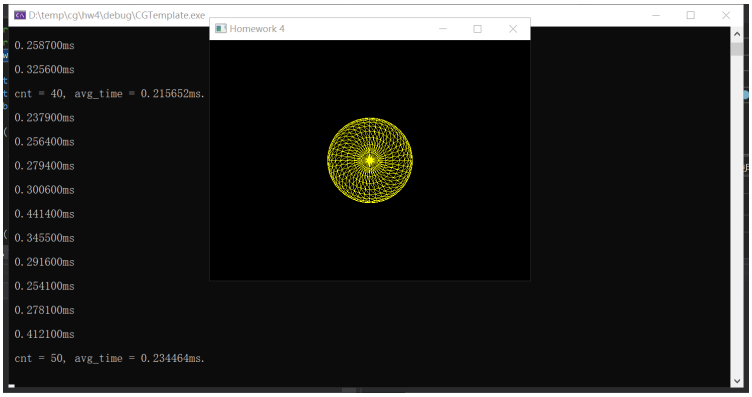
```
void MyGLWidget::init_vbo_with_indices(GLfloat* ball, GLint* indices) {
    //创建VBO，VAO，EBO
    glGenBuffers(1, &VBO);
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &EBO);

    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * 4 * 3 * u_num * v_num, ball, GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(GLint) * 6 * 3 * u_num * v_num, indices, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(0);
}

```

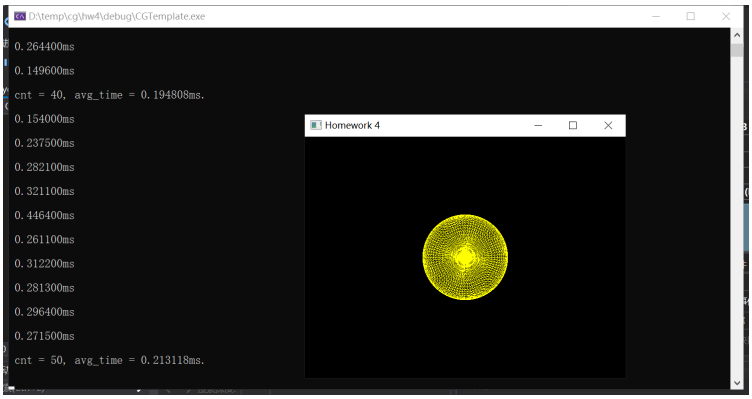
由于每次采样只需保存4个不重复的点，因此计算球面坐标数组的函数修改至如下所示：

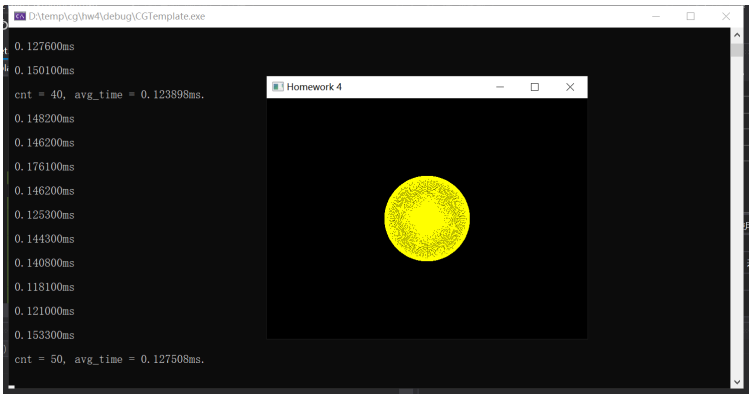
```
void MyGLWidget::create_ball_with_indices(GLfloat* ball) {
    ...
    for (int u = 0; u < u_num; u++) {
        for (int v = 0; v < v_num; v++) {
            GLfloat* point_1 = get_point_on_ball(u * u_step, v * v_step, r);
            ...//point_2,3,4
            GLfloat* points[4] = { point_1, point_2, point_3, point_4};
            for (int i = 0; i < 4; i++) {
                memcpy(ball + offset, points[i], sizeof(GLfloat) * 3);
                offset += 3;
            }
            delete[] point_1, point_2, point_3, point_4;
        }
    }
}
```

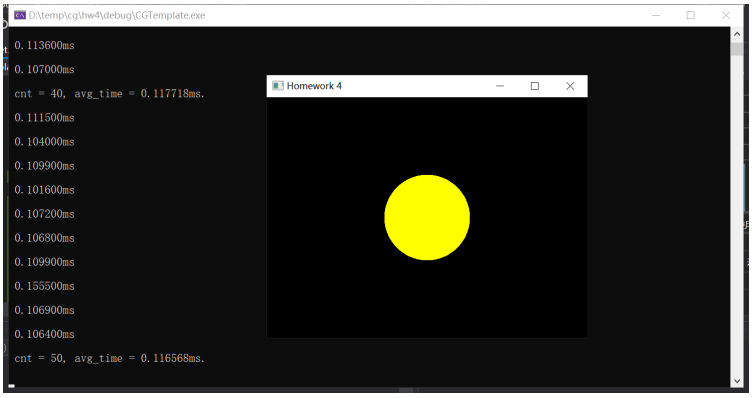
3 实验结果

分别对于u_num = 20, 40, 80, 160, 320, 640, v_num = 40, 80, 160, 320, 640, 1280进行测试。为证明实现的正确性，放出部分截图如下：









最终测试结果如下（单位：毫秒）：

方式\(u_num, v_num)	(20, 40)	(40, 80)	(80, 160)	(160, 320)	(320, 640)	(640, 1280)
不采用VBO	0.235962	0.788936	1.533062	3.917474	14.814882	56.463320
采用VBO	0.198812	0.202966	0.130746	0.104528	0.095112	0.098380
采用EBO	0.234464	0.213118	0.127608	0.116568	0.118038	0.135280

可以观察到，在细分程度较小时，三种方法的差别不大。而在细分程度较大时，采用VBO和采用EBO的耗时并未明显上升，甚至相较于细分程度最小的情况还有所下降，而不采用VBO的方法耗时则上升了很多。采用VBO和采用EBO的方法，基本上后者耗时略长一点，但是差距并不明显。

4 实验感想

本次实验主要是比较三种绘制小球的方式。在上一次作业中，我是先实现了不采用VBO的方式画球，再改为了采用VBO的方式，但是并没有注意到二者可能存在的性能差别。在本次实验中，我对于三种方式的性能差异有了新的认识。然而，由于再大的细分方式会导致进程内存过大而导致VS报错，因此没能再测试细分程度更大的结果。除此之外，细分程度更大时，在几次测试中都得到了采用VBO和EBO的耗时比细分程度小时更小的情况，并没能想出其原因。