作业1

环境搭建

作业1需要opencv库，按照下面的教程笔者搭建好了opencv库以及完善了相应的依赖:

【GAMES101闫令琪图形学】作业1（配置opencv4.5.4，并解决常见错误）_hans774882968的博客-CSDN博客

宿主机：Windows10；虚拟机：Ubuntu20.04。 作者：hans774882968 opencv4.5.4：https://opencv.org/releases/ opencv_contrib4.5.4：https://github.com/opencv/opencv_contrib/releases/tag/4.5.4 unzip opencv-4.5.4.zip unzip opencv_contrib-4.5.4.zip 注意，两个文件夹要放在同一个文件夹（我命名为opencv）下，目录结构： opencv build opencv-

https://blog.csdn.net/hans774882968/article/details/122885417

在跟着教程进行环境配置的时候，出现了python-dev指定错误的情况，此时复制出错信息上网找解决办法即可

基础部分

按照实验指导书要求完善旋转矩阵和透视投影矩阵部分，相关代码如下：

```
Eigen::Matrix4f get_model_matrix(float rotation_angle)
{
    Eigen::Matrix4f model = Eigen::Matrix4f::Identity();

    // TODO: Implement this function
    // Create the model matrix for rotating the triangle around the Z axis.
    // Then return it.

    Eigen::Matrix4f rotationMatrix = Eigen::Matrix4f::Identity();;
    double angle = rotation_angle / 180.0 * MY_PI;
    rotationMatrix << cos(angle),-sin(angle),0,0,
                    sin(angle),cos(angle),0,0,
                    0,0,1,0,
                    0,0,0,1;
    model = rotationMatrix * model;

    return model;
}
```

```
Eigen::Matrix4f get_projection_matrix(float eye_fov, float aspect_ratio,
                                      float zNear, float zFar)
{
    // Students will implement this function

    Eigen::Matrix4f projection = Eigen::Matrix4f::Identity();

    // TODO: Implement this function
    // Create the projection matrix for the given parameters.
    // Then return it.
    double r,l,t,b;
    double fov_angle = eye_fov / 180.0 *MY_PI;
    t = zNear * tan(fov_angle / 2);
    b = -t;
    r = t * aspect_ratio;
    l = -r;
    Eigen::Matrix4f scale = Eigen::Matrix4f::Identity();
    Eigen::Matrix4f trans = Eigen::Matrix4f::Identity();
    Eigen::Matrix4f persp = Eigen::Matrix4f::Identity();
    scale << 2/(r-l),0,0,0,
            0,2/(t-b),0,0,
            0,0,2/(zFar-zNear),0,
            0,0,0,1;
    trans << 1,0,0,-(r+l)/2,
            0,1,0,-(t+b)/2,
            0,0,1,-(-zNear-zFar)/2,
            0,0,0,1;
    persp << -zNear,0,0,0,
            0,-zNear,0,0,
            0,0,-zNear-zFar,-zNear*zFar,
            0,0,1,0;
    projection = scale * trans * persp;
    return projection;
}
```

只要按照老师在课堂上讲解的公式进行相关矩阵运算即可，不过需要注意的是：

- **顶部top与底部bottom、左部left与右部right相互对称**
  在透视投影矩阵相关代码部分，实验默认的是模型的顶部(t, t > 0)和底部(b)、左部(l)和右部(r, r > 0)是对称的，所以`b==-t, l = -r`，其实如果为了简化计算，还可以将矩阵里的算式进行约分，但笔者基于代码以及公式的可读性，就完全按照原公式进行计算
- **zNear和zFar的正负问题**
  get_projection_matrix()函数的参数zNear和zFar传值为正，但是公式内的near和far则为负值，这里需要注意符号问题，否则实验初始化时将会看到一个倒转的三角形
- **projection矩阵分为三个矩阵的积**
  一个是缩放矩阵，将模型长宽高限制在2以内，一个是平移矩阵，将模型平移至原点，这两个矩阵的作用综合为将模型限制在[-1, 1]区间（也就是正交投影）；最后一个矩阵则是将透视投影转换为正交投影，至于原理请复习Games101课程

提高部分

提高部分要求得到绕任意过原点的轴的旋转变换矩阵，也就是得到罗德里格旋转公式

$$\mathbf{R}(\mathbf{n},\alpha) = \cos(\alpha)\mathbf{I} + (1-\cos(\alpha))\mathbf{n}\mathbf{n}^{\mathbf{T}} + \sin(\alpha)\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

相关代码如下：

```
Eigen::Matrix4f get_rotation(Vector3f axis, float angle)
{
    Eigen::Matrix4f Rodri4f = Eigen::Matrix4f::Identity();
    Eigen::Matrix3f Rodri3f = Eigen::Matrix3f::Identity();
    Eigen::Matrix3f I = Eigen::Matrix3f::Identity();
    Eigen::Matrix3f N = Eigen::Matrix3f::Identity();

    N << 0,-axis.z(),axis.y(),
            axis.z(),0,-axis.x(),
            -axis.y(),axis.x(),0;

    double rotate_angle = angle/180.0*MY_PI;
    Rodri3f = cos(rotate_angle)*I+(1-cos(rotate_angle)) * axis * axis.transpose() + sin(rotate_angle)*N;
    Rodri4f.block(0,0,3,3) = Rodri3f;
    return Rodri4f;
}
```

- `Rodri(3,3) = 1`
  按照公式，本应该进行计算的矩阵为三维矩阵，但要得到的矩阵为四维矩阵，对此有两个解决办法，一个为直接进行四维矩阵的运算，但会出现一个问题，根据公式，矩阵的最后一个元素也就是Rodri(3, 3)应该为1，但是在实验中出现了旋转的三角形大小变化的情况，这是因为对于浮点数运算，可能会出现约等于0而不是0的情况，所以笔者建议使用第二种方法：先进行三维矩阵的运算，再将得到的罗德里格旋三维转矩阵赋值给一个四维单位矩阵即可，笔者采用的是第二种方法
- **剩余修改部分（为了验证结果）**

```
void rst::rasterizer::set_Rodrigues(const Eigen::Matrix4f& r)
{
    Rodrigues = r;
}
```

```
float f1 = (100 - 0.1) / 2.0;
    float f2 = (100 + 0.1) / 2.0;

    Eigen::Matrix4f mvp = projection * view * model * Rodrigues;
```

```
int main(int argc, const char** argv)
{
    float angle = 0;
    bool command_line = false;  //定义命令行开关标志，默认为关
    std::string filename = "output.png";

    Eigen::Vector3f rotated_axis(0,0,1);
    float rangle = 0, ra;
```

```
    if (argc >= 3) {  //接收到的参数大于三个，即检测到通过命令行传入参数时
        command_line = true;
        angle = std::stof(argv[2]); // -r by default
        if (argc == 4) {  //接收到的参数为四个，那么说明命令行输入了文件名参数
            filename = std::string(argv[3]);
        }
    }

    rst::rasterizer r(700, 700);  //设定700*700像素的光栅器视口

    Eigen::Vector3f eye_pos = {0, 0, 5};  //设定相机位置

    std::vector<Eigen::Vector3f> pos{{2, 0, -2}, {0, 2, -2}, {-2, 0, -2}};

    std::vector<Eigen::Vector3i> ind{{0, 1, 2}};  //设定三顶点序号,用于画图时确定需要处理几个顶点，这里表示的是三个顶点

    auto pos_id = r.load_positions(pos);
    auto ind_id = r.load_indices(ind);  //保存多个图形的顶点和序号，本次作业只涉及一个图形

    int key = 0;  //键盘输入
    int frame_count = 0;  //帧序号

    if (command_line) { //如果命令行开关标志为开（这一段代码是为了应用命令行传入的参数，比如初始角度和文件名）
        r.clear(rst::Buffers::Color | rst::Buffers::Depth);  //初始化帧缓存和深度缓存（本次作业本次作业只涉及一个图形，所以不涉及深度）

        r.set_model(get_model_matrix(angle));
        r.set_view(get_view_matrix(eye_pos));
        r.set_projection(get_projection_matrix(45, 1, 0.1, 50));
        r.set_Rodrigues(get_rotation(rotated_axis, rangle));

        r.draw(pos_id, ind_id, rst::Primitive::Triangle);
        cv::Mat image(700, 700, CV_32FC3, r.frame_buffer().data());
        image.convertTo(image, CV_8UC3, 1.0f);
        cv::imwrite(filename, image);

        return 0;
    }

    bool rFlag = false;

    std::cout << "please enter the axis and the angle: "<< std::endl;
    std::cin >> rotated_axis.x() >> rotated_axis.y() >> rotated_axis.z() >> ra;  //定义罗德里格斯旋转轴和角

    while (key != 27) {  //只要没有检测到按下ESC就循环(ESC的ASCII码是27)
        r.clear(rst::Buffers::Color | rst::Buffers::Depth);
        r.set_model(get_model_matrix(angle));
        r.set_view(get_view_matrix(eye_pos));
        r.set_projection(get_projection_matrix(45, 1, 0.1, 50));

        if(rFlag)
            r.set_Rodrigues(get_rotation(rotated_axis,rangle));
        else
            r.set_Rodrigues(get_rotation({0,0,1},0));

        r.draw(pos_id, ind_id, rst::Primitive::Triangle);

        cv::Mat image(700, 700, CV_32FC3, r.frame_buffer().data());
        image.convertTo(image, CV_8UC3, 1.0f);
        cv::imshow("image", image);  //显示图像
        key = cv::waitKey(10);

        std::cout << "frame count: " << frame_count++ << '\n';  //显示当前是第几帧画面

        if (key == 'a') {  //按下a，逆时针旋转10°
            angle += 10;
        }
        else if (key == 'd') {  //按下d，顺时针旋转10°
            angle -= 10;
        }
        else if (key == 'e')  //按下e，绕给定旋转轴旋转ra°
        {
            rFlag = true;
            rangle += ra;
        }
        else if (key == 'q')  //按下q，绕给定旋转轴反向旋转ra°
        {
            rFlag = true;
            rangle -= ra;
        }
    }

    return 0;
}
```

至此，作业1完成！