HW1 Report

1. **Switch between solid and wireframe mode**

   先設定一個型別為 GLenum 變數 mode 做為預先設定圖形化出來的 polygon mode，這裡預先設定為 solid frame mode。

   可以透過按鍵 w 切換 solid frame and wireframe，並在模型 draw array 之前改變模型的 polygon mode。

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    // [DONE] Call back function for keyboard
    if (key == GLFW_KEY_W && action == GLFW_PRESS) {/* switch between solid and wireframe mode*/
        if (mode == GL_FILL) {
            mode = GL_LINE;
        }
        else {
            mode = GL_FILL;
        }
        //change_mode = true;
```

```
glPolygonMode(GL_FRONT_AND_BACK, mode);
glDrawArrays(GL_TRIANGLES, 0, m_shape_list[cur_idx].vertex_count);
```

2. **Switch the model**

   先把 ColorModels 資料夾裡的五個模型用 for 迴圈以 LoadModles()讀出來。

```
vector<string> model_list{ "../ColorModels/bunny5
// [DONE] Load five model at here
for (int i = 0; i <= 4; i++) {
    LoadModels(model_list[i]);
}
```

   已知共有五個模型，依照模型讀進來的順序，藉由 Z 切換至前一個模型，由 X 切換至後一個模型。

```
else if (key == GLFW_KEY_Z && action == GLFW_PRESS) {/* switch pre model */
    cur_idx -= 1;
    if (cur_idx < 0) {
        cur_idx = 4;
    }
}
else if (key == GLFW_KEY_X && action == GLFW_PRESS) {/* switch post model */
    cur_idx += 1;
    if (cur_idx > 4) {
        cur_idx = 0;
    }
}
```

3. **Orthogonal projection**

```
void setOrthogonal()
{
    cur_proj_mode = Orthogonal;
    GLfloat t_x, t_y, t_z;

    t_x = -(proj.right + proj.left) / (proj.right - proj.left);
    t_y = -(proj.top + proj.bottom) / (proj.top - proj.bottom);
    t_z = -(proj.farClip + proj.nearClip) / (proj.farClip - proj.nearClip);

    if (proj.aspect <= 1) {
        project_matrix = Matrix4(
            (2 / (proj.right - proj.left)), 0, 0, t_x,
            0, (2 / (proj.top - proj.bottom)*proj.aspect), 0, t_y,
            0, 0, -2 / (proj.farClip - proj.nearClip), t_z,
            0, 0, 0, 1
        );
    }
    else {
        project_matrix = Matrix4(
            (2 / (proj.right - proj.left)/proj.aspect), 0, 0, t_x,
            0, (2 / (proj.top  - proj.bottom )), 0, t_y,
            0, 0, -2 / (proj.farClip - proj.nearClip), t_z,
            0, 0, 0, 1
        );
    }
    //printf("change to orthogonal mode");
}
```

利用按鍵 O 切換至 orthogonal 投影模式。

```
else if (key == GLFW_KEY_O && action == GLFW_PRESS) {/* orthogonal projection */
    //cur_proj_mode = Orthogonal;
    setOrthogonal();
}
```

4. NDC Perspective Projection

```
void setPerspective()
{
    cur_proj_mode = Perspective;
    GLfloat f = 1 / tan(((proj.fovy) / 2) * PI / 180.0f);
    if (proj.aspect <= 1) {
        f = f * proj.aspect;
        project_matrix = Matrix4(
            (f / proj.aspect), 0, 0, 0,
            0, f, 0, 0,
            0, 0, (proj.farClip + proj.nearClip) / (proj.nearClip - proj.farClip), (2 * proj.farClip*proj.nearClip) / (proj.nearClip - proj.farClip),
            0, 0, -1, 0
        );
    }
    else {
        project_matrix = Matrix4(
            (f / proj.aspect), 0, 0, 0,
            0, f, 0, 0,
            0, 0, (proj.farClip + proj.nearClip) / (proj.nearClip - proj.farClip), (2 * proj.farClip*proj.nearClip) / (proj.nearClip - proj.farClip),
            0, 0, -1, 0
        );
    }
    //printf("change to perspecive mode");
}
```

利用按鍵 P 切換至 perspective 投影模式。

```
else if (key == GLFW_KEY_P && action == GLFW_PRESS) {/* NDC perspective projection */
    //cur_proj_mode = Perspective;
    setPerspective();
}
```

5. **Translate Matrix**

```
Matrix4 translate(Vector3 vec)
{
    Matrix4 mat;

    mat = Matrix4(
        1, 0, 0, vec.x,
        0, 1, 0, vec.y,
        0, 0, 1, vec.z,
        0, 0, 0, 1
    );

    return mat;
}
```

利用按鍵 T 切換至 translation 的操作模式。

```
else if (key == GLFW_KEY_T && action == GLFW_PRESS) {/* translation mode */
    cur_trans_mode = GeoTranslation;
}
```

6. **Scale Matrix**

```
Matrix4 scaling(Vector3 vec)
{
    Matrix4 mat;

    mat = Matrix4(
        vec.x, 0, 0, 0,
        0, vec.y, 0, 0,
        0, 0, vec.z, 0,
        0, 0, 0, 1
    );

    return mat;
}
```

利用按鍵 S 切換至 scaling 的操作模式。

```
else if (key == GLFW_KEY_S && action == GLFW_PRESS) {/* scale mode */
    cur_trans_mode = GeoScaling;
}
```

7. **Rotation Matrix**

先分別計算對於 x, y, z 軸的 rotation matrix。

```
Matrix4 rotateX(GLfloat val)
{
    Matrix4 mat;

    GLfloat c_v = cos(val*PI / 180.0f); //cos value
    GLfloat s_v = sin(val*PI / 180.0f); //sin value
    mat = Matrix4(
        1,0,0,0,
        0,c_v,-s_v,0,
        0,s_v,c_v,0,
        0,0,0,1
    );

    return mat;
}
```

```
Matrix4 rotateY(GLfloat val)
{
    Matrix4 mat;

    GLfloat c_v = cos(val*PI / 180.0f); //cos value
    GLfloat s_v = sin(val*PI / 180.0f); //sin value
    mat = Matrix4(
        c_v, 0, s_v, 0,
        0, 1, 0, 0,
        -s_v, 0, c_v, 0,
        0, 0, 0, 1
    );

    return mat;
}
```

```
Matrix4 rotateZ(GLfloat val)
{
    Matrix4 mat;

    GLfloat c_v = cos(val*PI / 180.0f); //cos value
    GLfloat s_v = sin(val*PI / 180.0f); //sin value
    mat = Matrix4(
        c_v, -s_v, 0, 0,
        s_v, c_v, 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1
    );

    return mat;
}
```

再將分別對三軸的矩陣進行相乘得出最後的旋轉矩陣。

```
Matrix4 rotate(Vector3 vec)
{
    return rotateX(vec.x)*rotateY(vec.y)*rotateZ(vec.z);
}
```

利用按鍵 R 切換至 rotation 的操作模式。

```
else if (key == GLFW_KEY_R && action == GLFW_PRESS) {/* rotation mode */
    cur_trans_mode = GeoRotation;
}
```

針對相機的視角操作，計算相對應的 viewing matrix。

```
void setViewingMatrix()
{
    Vector3 r_x, r_y, r_z;
    Vector3 p_12, p_13;
    p_12 = main_camera.center - main_camera.position; //f
    p_13 = main_camera.up_vector.normalize(); //u'
    r_z = p_12.normalize(); //f'
    r_x = r_z.cross(p_13);  //s
    r_y = r_x.cross(p_12);  // u''


    Matrix4 R, T;

    R = Matrix4(
        r_x.x,r_x.y,r_x.z,0,
        r_y.x,r_y.y,r_y.z,0,
        (-r_z.x),(-r_z.y),(-r_z.z), 0,
        0,0,0,1
    );
    //cout << R << endl;

    T = Matrix4(
        1,0,0,-main_camera.position.x,
        0,1,0,-main_camera.position.y,
        0,0,1,-main_camera.position.z,
        0,0,0,1
    );
    //cout << T << endl;

    view_matrix = R * T;
    //cout << view_matrix << endl;

}
```

在計算完所有模式的矩陣後，將這些矩陣相乘為 MVP 矩陣，將此矩陣傳給 vertex shader(shader.vs)並以此更新模型的狀態。

```
void RenderScene(void) {
    // clear canvas
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);

    Matrix4 T, R, S;

    // [DONE] update translation, rotation and scaling
    T = translate(models[cur_idx].position);
    R = rotate(models[cur_idx].rotation);
    S = scaling(models[cur_idx].scale);

    Matrix4 MVP;
    GLfloat mvp[16];

    // [DONE] multiply all the matrix
    //cout << "view_matrix" << view_matrix << endl;
    MVP = project_matrix * view_matrix * T * R * S;
    //cout << "mvp" << MVP << endl;
    // [DONE] row-major ---> column-major


    mvp[0] = MVP[0];  mvp[4] = MVP[1];   mvp[8] = MVP[2];    mvp[12] = MVP[3];
    mvp[1] = MVP[4];  mvp[5] = MVP[5];   mvp[9] = MVP[6];    mvp[13] = MVP[7];
    mvp[2] = MVP[8];  mvp[6] = MVP[9];   mvp[10] = MVP[10];   mvp[14] = MVP[11];
    mvp[3] = MVP[12]; mvp[7] = MVP[13];  mvp[11] = MVP[14];   mvp[15] = MVP[15];


    // use uniform to send mvp to vertex shader
    glUniformMatrix4fv(iLocMVP, 1, GL_FALSE, mvp);
    glBindVertexArray(m_shape_list[cur_idx].vao);

    glPolygonMode(GL_FRONT_AND_BACK, mode);
    glDrawArrays(GL_TRIANGLES, 0, m_shape_list[cur_idx].vertex_count);

    drawPlane();
```

```
void main()
{
    // [DONE]
    gl_Position = mvp*vec4(aPos.x, aPos.y, aPos.z, 1.0);
    vertex_color = aColor;
}
```

8. **Eye position mode**

利用按鍵 E 切換至 translate eye position mode。

```
else if (key == GLFW_KEY_E && action == GLFW_PRESS) {/* translate eye position mode */
    cur_trans_mode = ViewEye;
}
```

9. **Viewing center position mode**

利用按鍵 C 切換至 viewing center position mode。

```
else if (key == GLFW_KEY_C && action == GLFW_PRESS) {/* translate viewing center position mode*/
    cur_trans_mode = ViewCenter;
}
```

10. Camera up vector position mode

利用按鍵 U 切換至 camera up vector position mode。

```
else if (key == GLFW_KEY_U && action == GLFW_PRESS) {/* trnaslate camera up vector position mode */
    cur_trans_mode = ViewUp;
}
```

11. Print information

根據作業規範，印出模型當下對應的各矩陣。

```
else if (key == GLFW_KEY_I && action == GLFW_PRESS) {/* print information */
    printf("Matrix Value:\n");

    printf("Viewing Matrix:\n");
    cout << view_matrix << endl;

    printf("Projectiong Matrix:\n");
    cout << project_matrix << endl;

    Matrix4 T, R, S;

    T = translate(models[cur_idx].position);
    R = rotate(models[cur_idx].rotation);
    S = scaling(models[cur_idx].scale);

    printf("Translation Matrix:\n");
    cout << T << endl;

    printf("Rotation Matrix:\n");
    cout << R << endl;

    printf("Scaling Matrix:\n");
    cout << S << endl;

}
```

12. Render quad

平面一樣會受到投影及相機視角的影響，因此計算其矩陣(僅包含投影和 viewing matrix 相乘)並將此訊息傳至 vertex shader，其中因為模型在切換 render 出來的 polygon mode 時平面不會跟著切換，所以在 draw array 之前設定其 polygon mode 的畫法為 GL_FILL。

```
void drawPlane()
{
    // [TODO] draw the plane with above vertices and color
    Matrix4 MVP;
    GLfloat mvp[16];

    MVP = project_matrix * view_matrix;

    mvp[0] = MVP[0];  mvp[4] = MVP[1];   mvp[8] = MVP[2];    mvp[12] = MVP[3];
    mvp[1] = MVP[4];  mvp[5] = MVP[5];   mvp[9] = MVP[6];    mvp[13] = MVP[7];
    mvp[2] = MVP[8];  mvp[6] = MVP[9];   mvp[10] = MVP[10];   mvp[14] = MVP[11];
    mvp[3] = MVP[12]; mvp[7] = MVP[13];  mvp[11] = MVP[14];   mvp[15] = MVP[15];

    glBindVertexArray(quad.vao);
    glUniformMatrix4fv(iLocMVP, 1, GL_FALSE, mvp);
    //GL.begin();
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glDrawArrays(GL_TRIANGLES, 0, 6);

}
```

在 SetupRC()的部分，設定給定的頂點及顏色資訊，分別將頂點資訊及顏色資訊畫至 window 中。

```
//setup quad vertices
GLfloat vertices[18]{ 1.0, -0.9, -1.0,
    1.0, -0.9,  1.0,
    -1.0, -0.9, -1.0,
    1.0, -0.9,  1.0,
    -1.0, -0.9,  1.0,
    -1.0, -0.9, -1.0 };

//setup quad colors
GLfloat colors[18]{ 0.0,1.0,0.0,
    0.0,0.5,0.8,
    0.0,1.0,0.0,
    0.0,0.5,0.8,
    0.0,0.5,0.8,
    0.0,1.0,0.0 };

glGenVertexArrays(1, &quad.vao);
glGenBuffers(1, &quad.vbo);

glBindVertexArray(quad.vao);

glBindBuffer(GL_ARRAY_BUFFER, quad.vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

glGenBuffers(1, &quad.p_color);
glBindBuffer(GL_ARRAY_BUFFER, quad.p_color);

glBufferData(GL_ARRAY_BUFFER, sizeof(colors), colors, GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(1);
```

13. Change size

```
void ChangeSize(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
    // [TODO] change your aspect ratio
    proj.aspect = (float)width / (float)height;
    if (cur_proj_mode == Perspective) {
        setPerspective();
    }
    else if (cur_proj_mode == Orthogonal) {
        setOrthogonal();
    }

}
```

接著分別依照投影模式的不同，進行投影矩陣的調整，並在 aspect ratio 小於等於 1 時進行不同的處理。

14. Key mapping

利用鍵盤切換模式的部分已在上述報告中講解，以下的部分將針對滑鼠操作進行說明。

根據作業規範，滑鼠滾輪的部分僅會影響 Z 軸的操作及視角，往上為正其餘為負，針對一般的滑鼠滾輪操作僅會影響 y offset，因此針對不同的操作模式，依據滑鼠滾輪 y offset 的值改變對應模式的 z 軸值。

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    // [DONE] scroll up positive, otherwise it would be negtive
    //A normal mouse wheel, being vertical, provides offsets along the Y - axis.

    if (cur_trans_mode == GeoTranslation) {
        models[cur_idx].position.z += 0.5*yoffset;
    }
    else if (cur_trans_mode == GeoRotation) {
        models[cur_idx].rotation.z += 0.5*yoffset;
    }
    else if (cur_trans_mode == GeoScaling) {
        models[cur_idx].scale.z += 0.5*yoffset;
    }
    else if (cur_trans_mode == ViewCenter) {
        main_camera.center.z += 0.5*yoffset;
    }
    else if (cur_trans_mode == ViewEye) {
        main_camera.position.z += 0.5*yoffset;
    }
    else if (cur_trans_mode == ViewUp) {
        main_camera.up_vector.z += 0.5*yoffset;
    }


}
```

在滑鼠按鍵及拖動的部分，使用到專案中已存在的三個變數紀錄滑鼠的狀態。

```
bool mouse_pressed = false;
int starting_press_x = -1;
int starting_press_y = -1;
```

利用 mouse_button_callback()函數判定滑鼠左鍵是否按著。

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    // [DONE] mouse press callback function
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
        mouse_pressed = true;
    }
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_RELEASE) {
        mouse_pressed = false;
    }
}
```

若 mouse_pressed 變數為真，才會進行游標移動的操作。

先計算 x 軸及 y 軸的 offset 計算，再將其值根據不同的操作模式進行 x 軸和 y 軸的改變。其中一個比較需要注意的地方是 rotation 的部分，滑鼠進行 x 軸方向的移動代表的是對 y 軸進行旋轉，滑鼠進行 y 軸方向的移動則代表對

x 軸進行旋轉。

```
// [DONE] cursor position callback function
float xoffset, yoffset;
xoffset = xpos - starting_press_x;
yoffset = starting_press_y - ypos;
starting_press_x = xpos;
starting_press_y = ypos;

if (mouse_pressed) {
    if (cur_trans_mode == GeoTranslation) {
        models[cur_idx].position.x += 0.01*xoffset;
        models[cur_idx].position.y += 0.01*yoffset;
    }
    else if (cur_trans_mode == GeoRotation) {
        models[cur_idx].rotation.y += 5 * PI / 180 * xoffset;
        models[cur_idx].rotation.x += 5 * PI / 180 * yoffset;
    }
    else if (cur_trans_mode == GeoScaling) {
        models[cur_idx].scale.x += 0.01*xoffset;
        models[cur_idx].scale.y += 0.01*yoffset;
    }
    else if (cur_trans_mode == ViewCenter) {
        main_camera.center.x += 0.01*xoffset;
        main_camera.center.y += 0.01*yoffset;
        setViewingMatrix();
    }
    else if (cur_trans_mode == ViewEye) {
        main_camera.position.x += 0.01*xoffset;
        main_camera.position.y += 0.01*yoffset;
        setViewingMatrix();
    }
    else if (cur_trans_mode == ViewUp) {
        main_camera.up_vector.x += 0.01*xoffset;
        main_camera.up_vector.y += 0.01*yoffset;
        setViewingMatrix();
    }
```