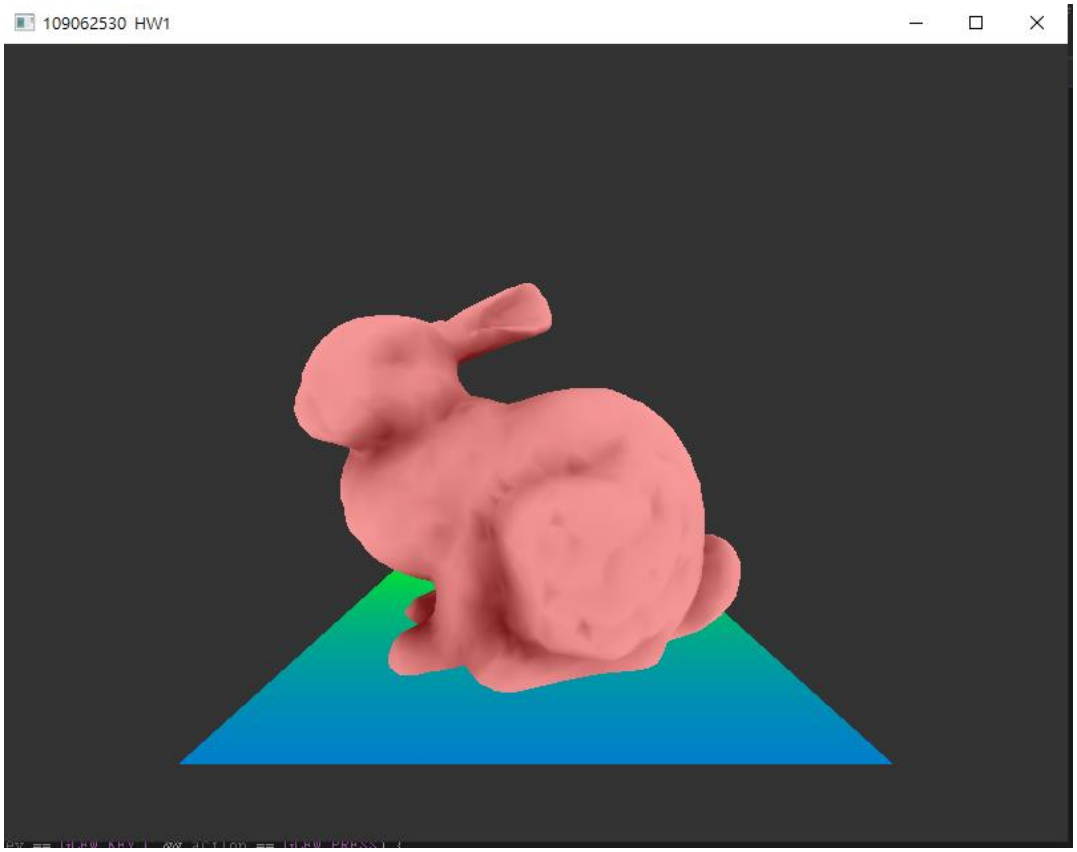
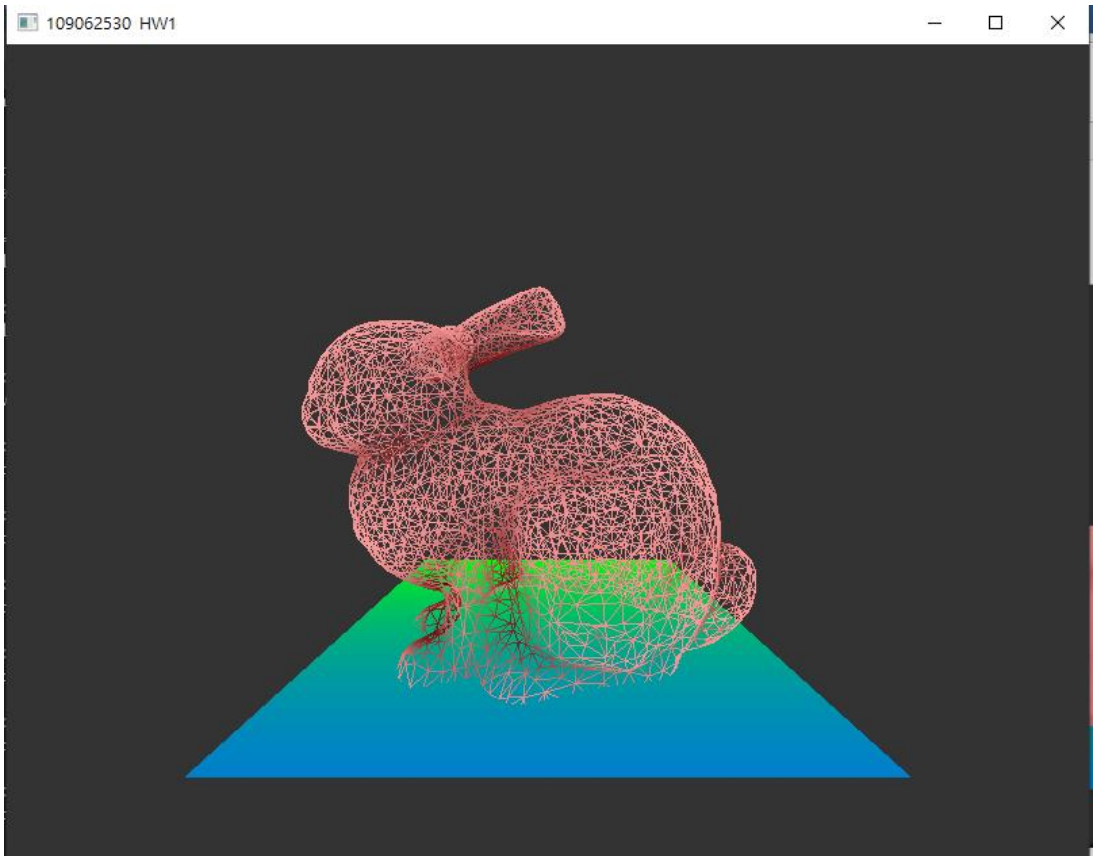
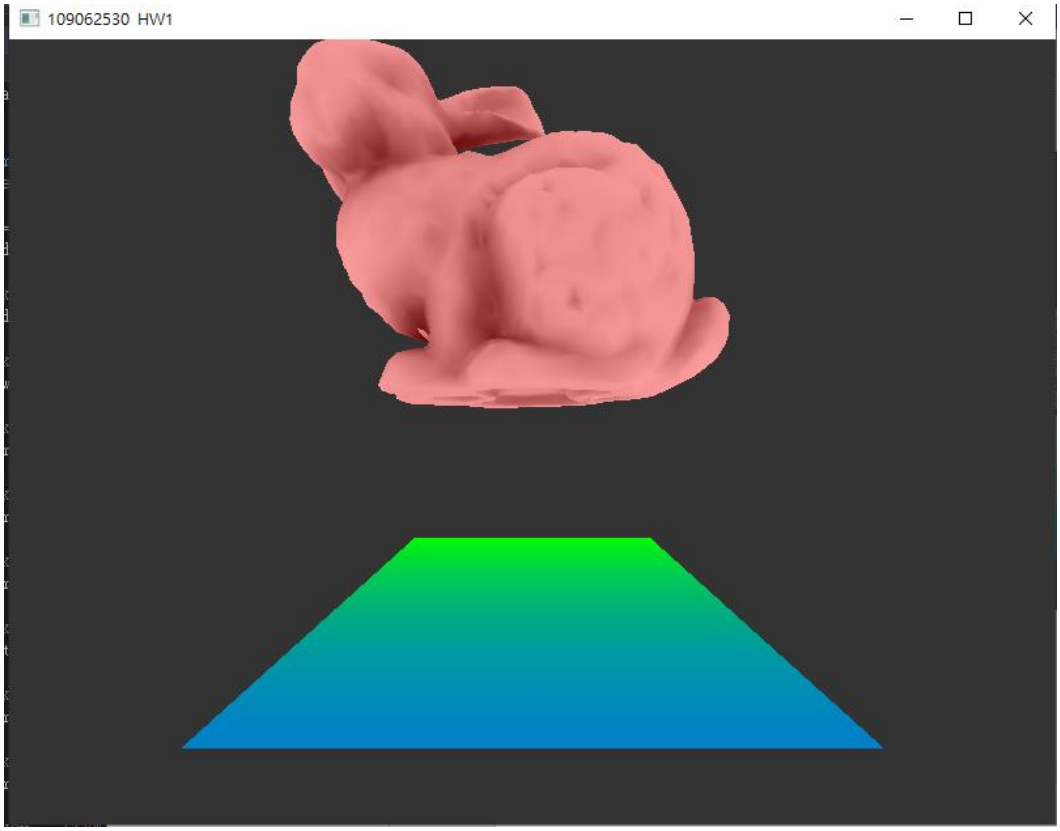
基本畫面:



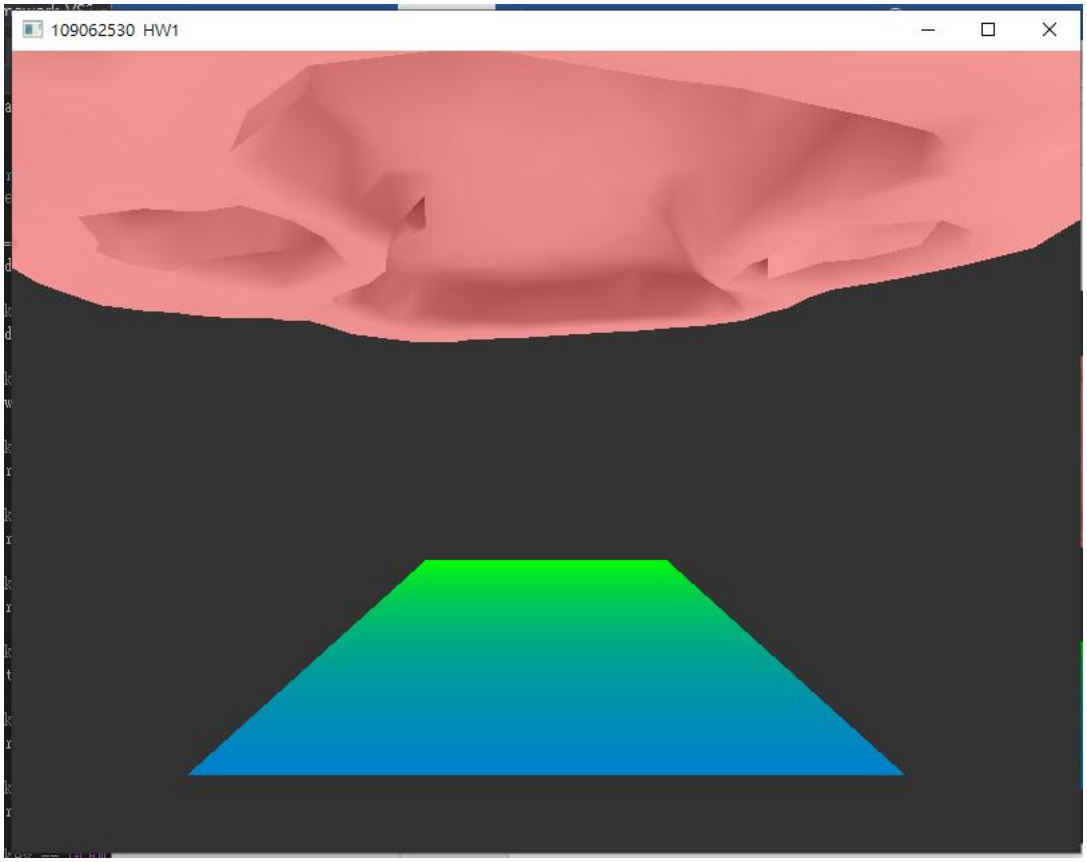
線圖:



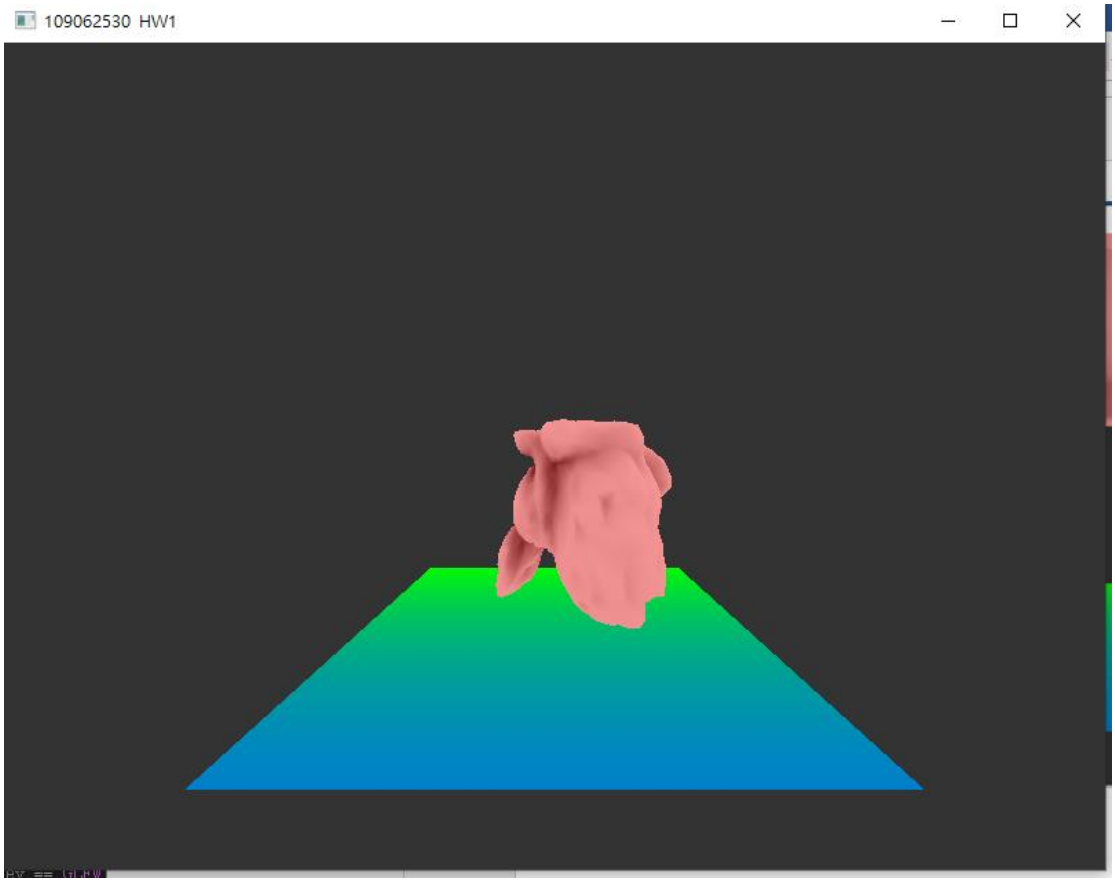
模型平移向上:



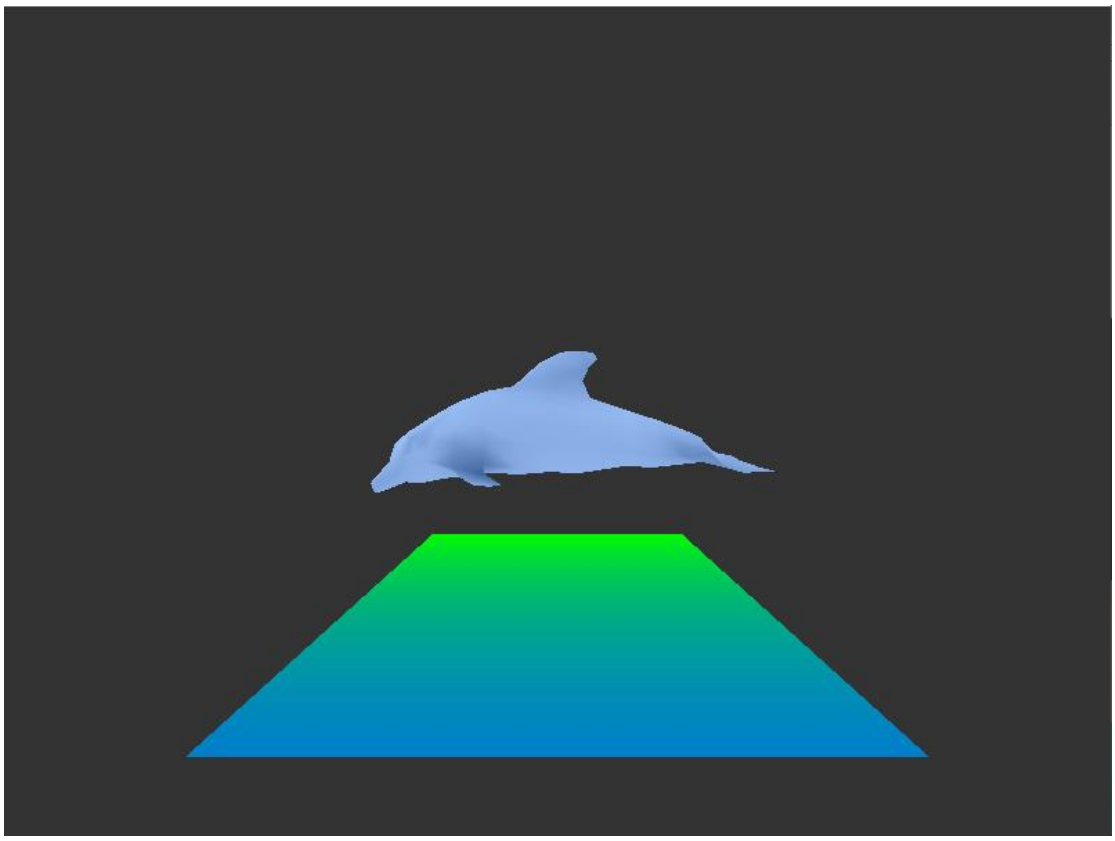
模型放大:



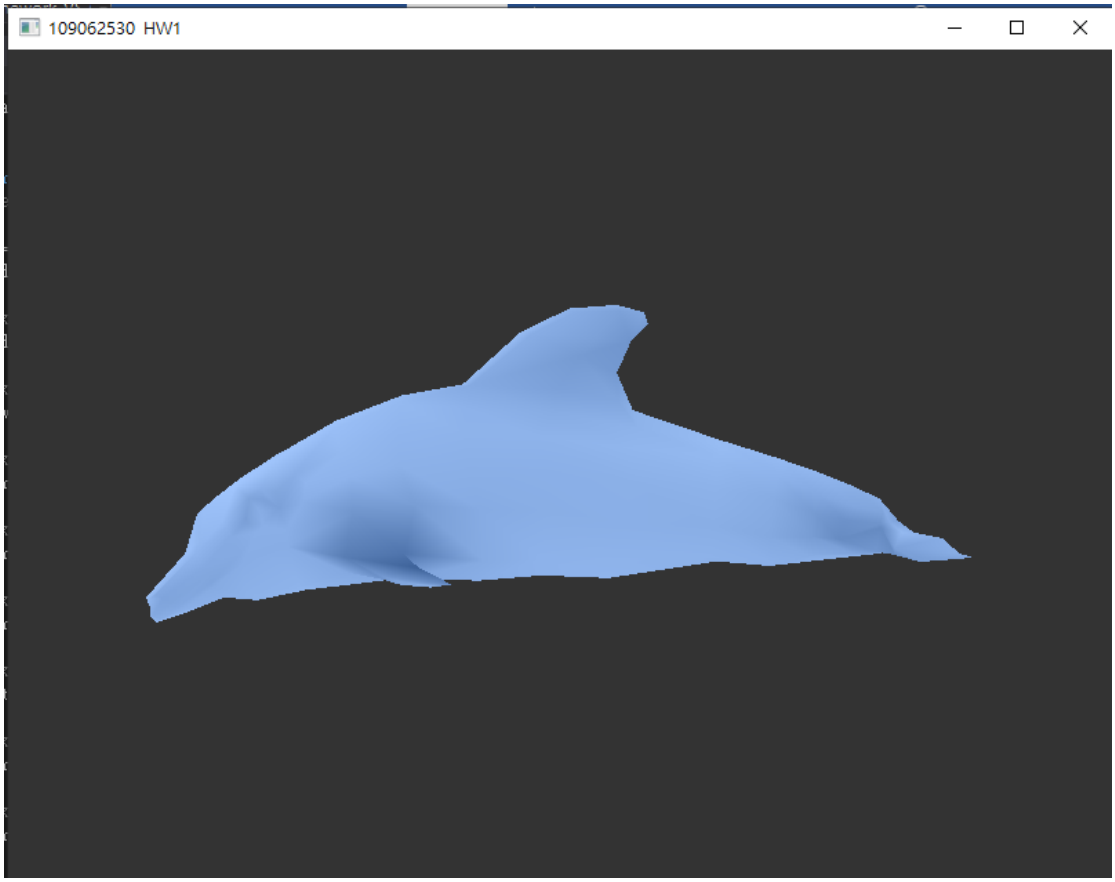
模型縮小、旋轉:



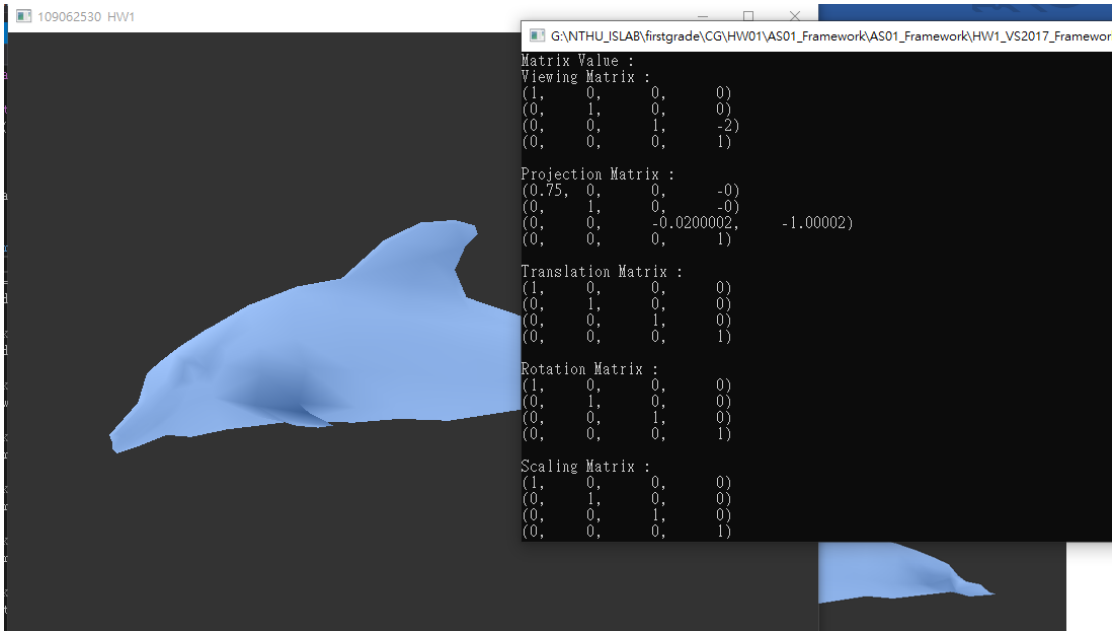
換模型:



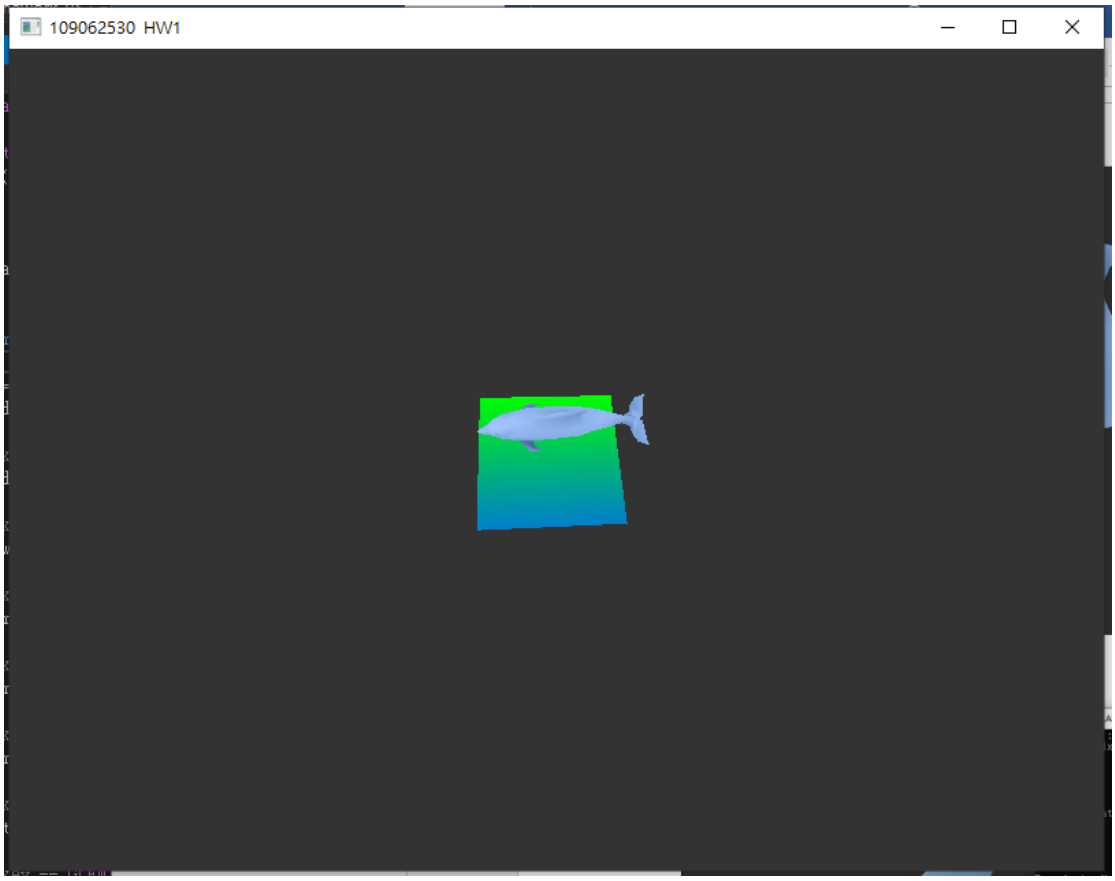
Orthogonal:



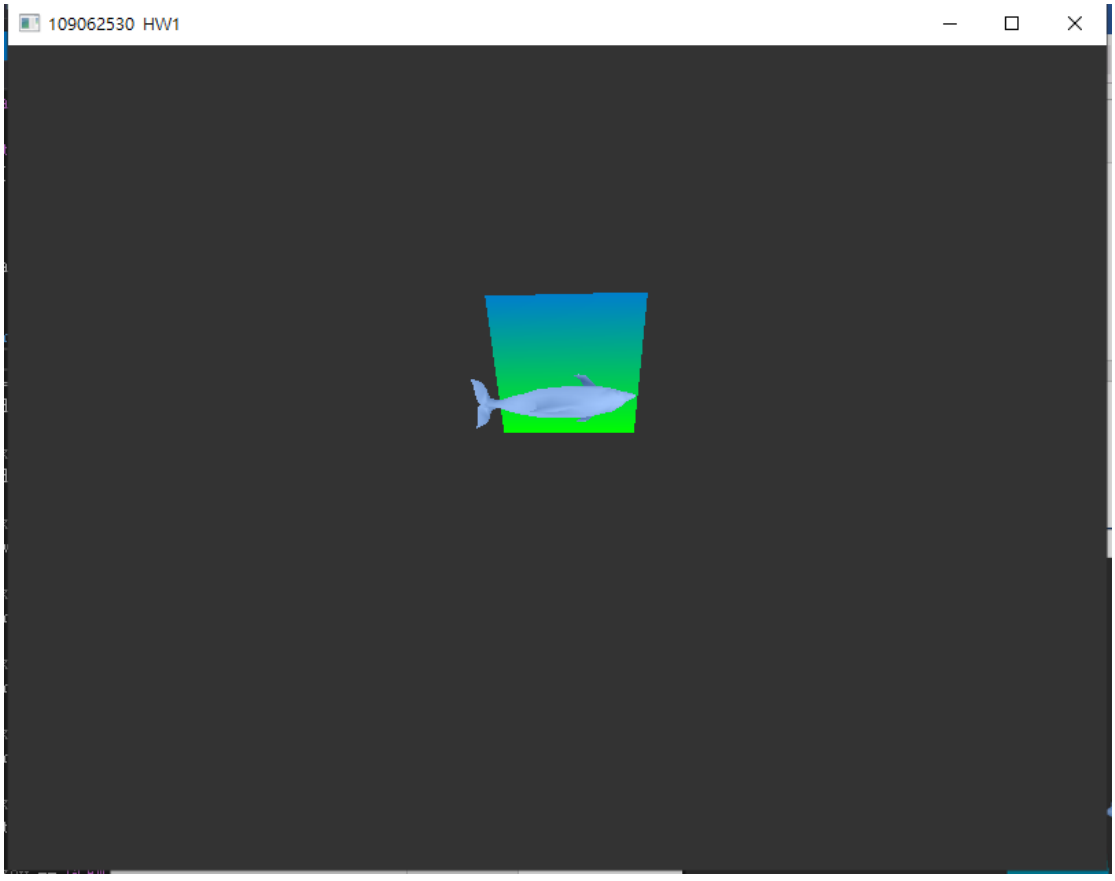
輸入 I 得到資訊



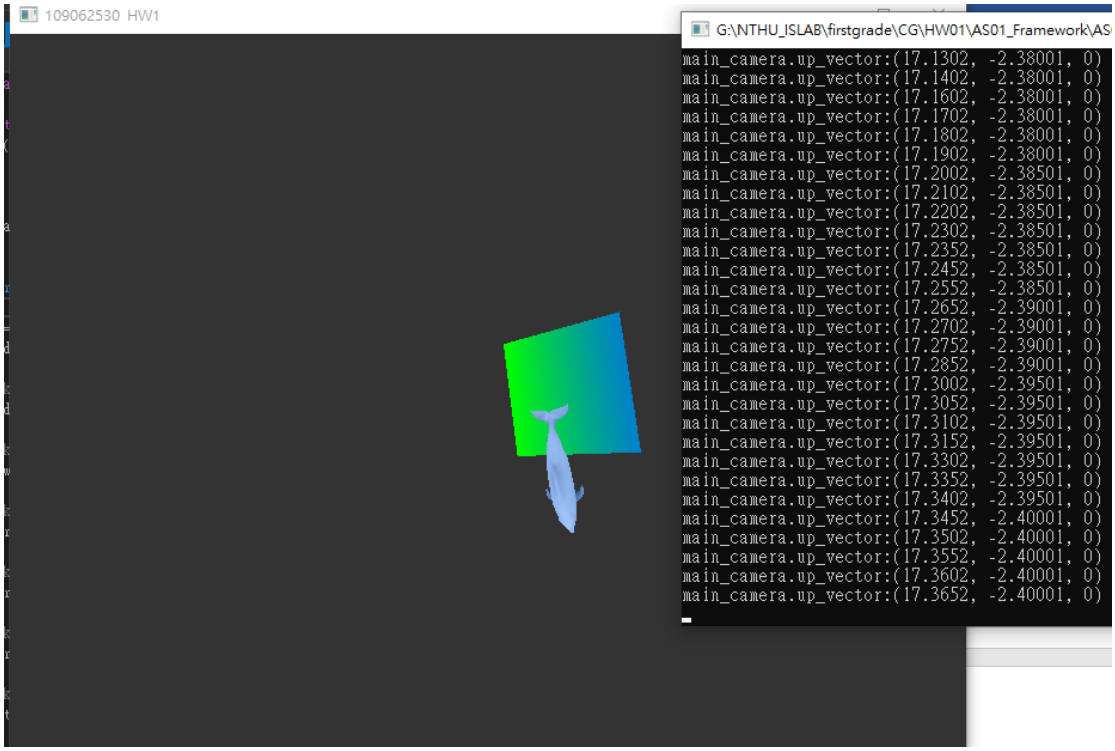
translate eye position 向上移:



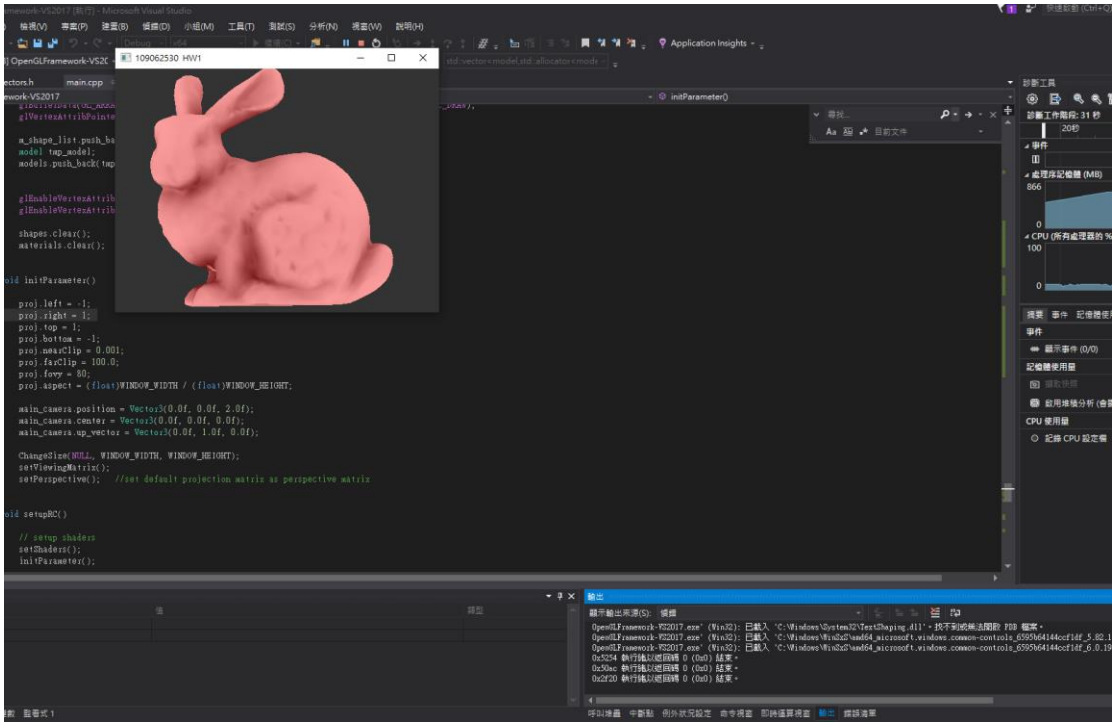
translate viewing center position:

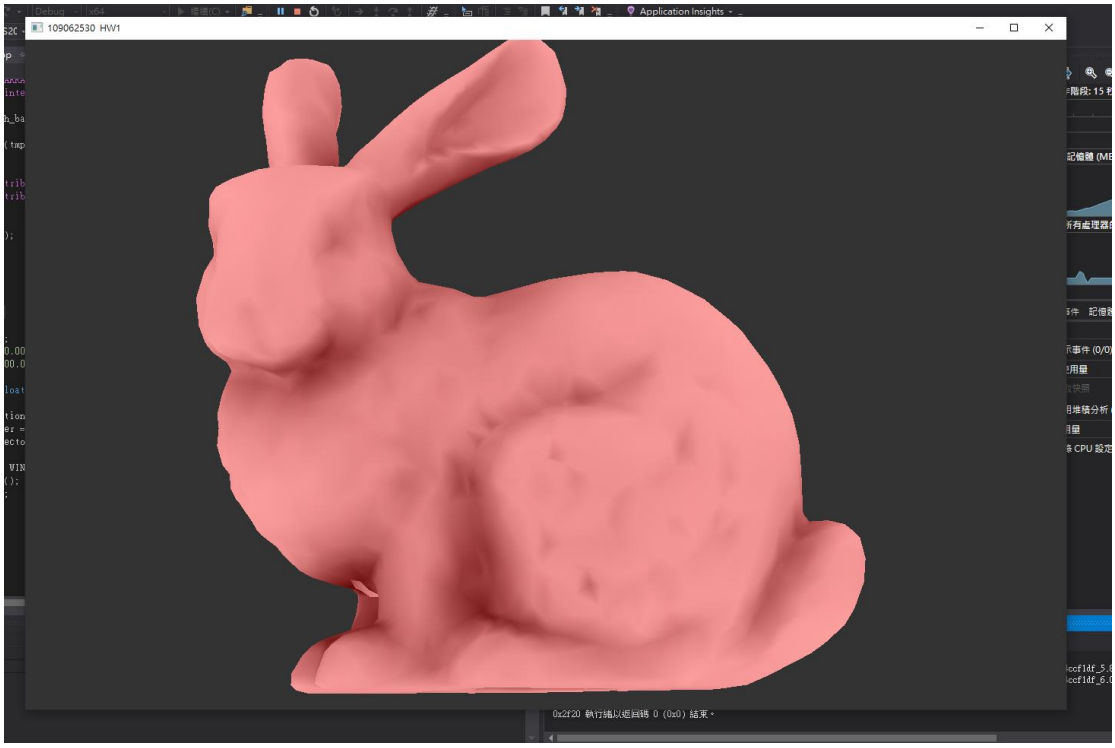


translate camera up vector position:



重繪視窗考量長寬比例:





程式控制指令與截圖:

X/Z: 更換模型
W :改成線、實體
R:改成模型旋轉模式
S:改成模型放大縮小模式
T:改成模型位移模式
O:正交投影
P:透視投影
E: eye position
C: view center
U: up vector
I:輸出資料

```
key = char(key);
// [TODO] Call back function for keyboard
if ( key == GLFW_KEY_X && action == GLFW_PRESS) {
    cur_idx == 0 ? cur_idx = 4 : cur_idx= cur_idx-1 ;
}
else if( key == GLFW_KEY_Z && action == GLFW_PRESS) {
    cur_idx == 4 ? cur_idx = 0 : cur_idx = cur_idx + 1;
}
else if (key == GLFW_KEY_W && action == GLFW_PRESS) {
    isDrawWireframe == false ? isDrawWireframe = true : isDrawWireframe = false ;
}
else if (key == GLFW_KEY_R && action == GLFW_PRESS) {
    cur_trans_mode = GeoRotation;
}
else if (key == GLFW_KEY_S && action == GLFW_PRESS) {
    cur_trans_mode = GeoScaling;
}
else if (key == GLFW_KEY_T && action == GLFW_PRESS) {
    cur_trans_mode = GeoTranslation;
}
else if (key == GLFW_KEY_O && action == GLFW_PRESS) {
    setOrthogonal();
}
else if (key == GLFW_KEY_P && action == GLFW_PRESS) {
    setPerspective();
}
else if (key == GLFW_KEY_E && action == GLFW_PRESS) {
    cur_trans_mode = ViewEye;
}
else if (key == GLFW_KEY_C && action == GLFW_PRESS) {
    cur_trans_mode = ViewCenter;
}
else if (key == GLFW_KEY_U && action == GLFW_PRESS) {
    cur_trans_mode = ViewUp;
}
else if (key == GLFW_KEY_I && action == GLFW_PRESS) {
    cout << "Matrix Value :" << endl;
    cout << "Viewing Matrix :" << endl;
    cout << view_matrix << endl;
```

ALL TODO:

Load model:

```
// [TODO] Load five model at here
for (int i = 0; i < 5; i++)
{
    LoadModels(model_list[i]);
}
```

Translate , scaling, rotate matrix 參考講義公式:

```
// [TODO] given a translation vector then output a Matrix4 (Translation Matrix)
Matrix4 translate(Vector3 vec)
{
    Matrix4 mat;

    mat = Matrix4(
            1, 0, 0, vec[0],
            0, 1, 0, vec[1],
            0, 0, 1, vec[2],
            0, 0, 0, 1
        );

    return mat;
}
```

```
// [TODO] given a scaling vector then output a Matrix4 (Scaling Matrix)
Matrix4 scaling(Vector3 vec)
{
    Matrix4 mat;


    mat = Matrix4(
            vec[0], 0, 0, 0,
            0, vec[1], 0, 0,
            0, 0, vec[2], 0,
            0, 0, 0, 1
        );


    return mat;
}
```

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

    mat = Matrix4(
            1, 0, 0, 0,
            0, cos(val), -sin(val), 0,
            0, sin(val), cos(val), 0,
            0, 0, 0, 1
        );

    return mat;
}

// [TODO] given a float value then ouput a rotation matrix alone axis-Y (rotate alone axis-Y)
Matrix4 rotateY(GLfloat val)
{
    Matrix4 mat;

    mat = Matrix4(
            cos(val), 0, sin(val), 0,
            0, 1, 0, 0,
            -sin(val), 0, cos(val), 0,
            0, 0, 0, 1
        );

    return mat;
}

// [TODO] given a float value then ouput a rotation matrix alone axis-Z (rotate alone axis-Z)
Matrix4 rotateZ(GLfloat val)
{
    Matrix4 mat;

    mat = Matrix4(
            cos(val), -sin(val), 0, 0,
            sin(val), cos(val), 0, 0,
            0, 0, 1, 0,
            0, 0, 0, 1
        );
```

ViewingMatrix 參考講義公式:

$$f = c - e$$

$$f' = \frac{f}{|f|}$$ ← -z axis

$$u' = \frac{u}{|u|}$$

$$s = f' \times u'$$ ← x axis

$$u'' = s \times f$$ ← y axis

$$M = \begin{pmatrix} s_x & s_y & s_z & 0 \\ u''_x & u''_y & u''_z & 0 \\ -f'_x & -f'_y & -f'_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
// [TODO] Compute viewing matrix accroding to the setting of main_camera
void setViewingMatrix()
{
    Vector3 Rz = (main_camera.center - main_camera.position).normalize();
    Vector3 Rx = (main_camera.center - main_camera.position).cross(main_camera.up_vector - main_camera.position).normalize();
    Vector3 Ry = Rx.cross(Rz);

    view_matrix = Matrix4(
        Rx[0], Rx[1], Rx[2], 0,
        Ry[0], Ry[1], Ry[2], 0,
        -Rz[0], -Rz[1], -Rz[2], 0,
        0, 0, 0, 1
    );

    Matrix4 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
    );
    view_matrix = view_matrix * T;
}

// [TODO] Compute orthogonal projection matrix
void setOrthogonal()
{
    cur_proj_mode = Orthogonal;
    // project_matrix [...] = ...
    project_matrix = Matrix4(
        2 / (proj.right - proj.left), 0, 0, -(proj.right + proj.left) / (proj.right - proj.left),
        0, 2/(proj.top - proj.bottom),0, -(proj.top + proj.bottom) / (proj.top - proj.bottom),
        0, 0, -2 / (proj.farClip - proj.nearClip), -(proj.farClip + proj.nearClip) / (proj.farClip - proj.nearClip),
        0, 0, 0, 1
    );
}
```

setOrthogonal 參考講義公式:

glOrtho

- **OpenGL Orthographic Transformation Matrix**
  - **Orthographic (parallel) projection and orthographic normalization**

$$\begin{pmatrix} \frac{2}{Right-Left} & 0 & 0 & t_x \\ 0 & \frac{2}{Top-Bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{Far-Near} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$t_x = -\frac{Right+Left}{Right-Left}$$

$$t_y = -\frac{Top+Bottom}{Top-Bottom}$$

$$t_z = -\frac{Far+Near}{Far-Near}$$

```
// [TODO] Compute orthogonal projection matrix
void setOrthogonal()
{
    cur_proj_mode = Orthogonal;
    // project_matrix [...] = ...
    project_matrix = Matrix4(
        2 / (proj.right - proj.left), 0, 0, -(proj.right + proj.left) / (proj.right - proj.left),
        0, 2/(proj.top - proj.bottom),0, -(proj.top + proj.bottom) / (proj.top - proj.bottom),
        0, 0, -2 / (proj.farClip - proj.nearClip), -(proj.farClip + proj.nearClip) / (proj.farClip - proj.nearClip),
        0, 0, 0, 1
    );
}
```

setPerspective 參考講義公式

gluPerspective

- **OpenGL Perspective Transformation Matrix**
  - **Perspective projection and perspective normalization**

$$\begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{far + near}{near - far} & \frac{2 \cdot far \cdot near}{near - far} \\ 0 & 0 & -1 & 0 \end{pmatrix} \qquad f = \cot\left(\frac{fovy}{2}\right)$$

```
double cot(double x)
{
    return 1 / tan(x);
}

// [TODO] Compute persepective projection matrix
void setPerspective()
{
    cur_proj_mode = Perspective;

    /*  proj.left = -1;
    proj.right = 1;
    proj.top = 1;
    proj.bottom = -1;
    proj.nearClip = 0.001;
    proj.farClip = 100.0;
    proj.fovy = 80;
    proj.aspect = (float)WINDOW_WIDTH / (float)WINDOW_HEIGHT;
    */

    GLfloat f = -cot(proj.fovy / 2);

    project_matrix = Matrix4(

        f/proj.aspect, 0 , 0 , 0 ,
        0, f, 0, 0,
        0, 0, (proj.farClip + proj.nearClip) / (proj.nearClip- proj.farClip), (2.0*proj.farClip*proj.nearClip) / (proj.nearClip - proj.farClip),
        0, 0, -1, 0
    );
}
```

模型繪製 參考註解說明，並另外加上 glPolygonMode 繪製 wireframe 與 solid:

```
// Render function for display rendering
void RenderScene(void) {
    // clear canvas
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);

    Matrix4 T, R, S;
    // [TODO] 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];

    // [TODO] multiply all the matrix
    // [TODO] row-major ---> column-major
    MVP = project_matrix * view_matrix * (S * R * T);

    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
    // [TODO] draw 3D model in solid or in wireframe mode here, and draw plane
    glUniformMatrix4fv(iLocMVP, 1, GL_FALSE, mvp);
    glBindVertexArray(m_shape_list[cur_idx].vao);
    if (isDrawWireframe == false) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    }
    else {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    }
    glDrawArrays(GL_TRIANGLES, 0, m_shape_list[cur_idx].vertex_count);

    glBindVertexArray(0);
    drawPlane();
}
```

平面繪製:
參考註解說明完成

```
void drawPlane()
{
    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 };

    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 };
    // [TODO] draw the plane with above vertices and color
    Matrix4 MVP;
    GLfloat mvp[16];
    // [TODO] multiply all the matrix
    // [TODO] row-major ---> column-major
    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];

    //please refer to LoadModels function
    //glGenVertexArrays..., glBindVertexArray...
    //glGenBuffers..., glBindBuffer..., glBufferData...
    glGenVertexArrays(1, &quad.vao);
    glBindVertexArray(quad.vao);

    glGenBuffers(1, &quad.vbo);
    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(GLfloat), (void*)0);

    quad.vertex_count = sizeof(vertices) / sizeof(GLfloat) / 3;
    glGenBuffers(1, &quad.p_color);
    glBindBuffer(GL_ARRAY_BUFFER, quad.p_color);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices) , colors, GL_STATIC_DRAW);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (void*)0);

    glEnableVertexAttribArray(0);
    glEnableVertexAttribArray(1);
```

```
    glUniformMatrix4fv(iLocMVP, 1, GL_FALSE, mvp);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    glDrawArrays(GL_TRIANGLES, 0, quad.vertex_count);
    glBindVertexArray(0);

}
```

滑鼠滾輪 CALL BACK:

根據作業說明與當前 MODE 調整相對應的 Z 軸

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    // [TODO] scroll up positive, otherwise it would be negtive
    switch (cur_trans_mode) {
        case GeoTranslation:
            models[cur_idx].position = models[cur_idx].position +  Vector3(0, 0, 0.01 * yoffset);
            break;

        case GeoScaling:
            models[cur_idx].scale = models[cur_idx].scale + Vector3(0, 0, 0.01* yoffset);
            break;

        case GeoRotation:
            models[cur_idx].rotation = models[cur_idx].rotation + Vector3(0, 0, 0.01 * yoffset);
            break;

        case ViewEye:
            main_camera.position = main_camera.position - Vector3(0, 0, 0.01 * yoffset);
            cout << "main_camera.position:" << main_camera.position << endl;

            setViewingMatrix();
            break;
        case ViewCenter:
            main_camera.center = main_camera.center - Vector3(0, 0, 0.01 * yoffset);
            cout << "main_camera.center:" << main_camera.center << endl;

            setViewingMatrix();
            break;
        case ViewUp:
            main_camera.up_vector = main_camera.up_vector - Vector3(0, 0, 0.01 * yoffset);
            cout << "main_camera.up_vector:" << main_camera.up_vector << endl;

            setViewingMatrix();
            break;
    }

}
```

滑鼠按壓偵測:

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    // [TODO] Call back function for mouse
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
        mouse_pressed = true;
    }
    else {
        mouse_pressed = false;
    }

}
```

鼠標移動 CALL BACK:

根據作業說明改動對應的 X 與 Y

視點有關的要進行重算矩陣

```
static void cursor_pos_callback(GLFWwindow* window, double xpos, double ypos)
{
    // [TODO] Call back function for cursor position


    if(mouse_pressed==true){
        switch (cur_trans_mode) {
            case GeoTranslation:
                models[cur_idx].position = models[cur_idx].position+ Vector3(0.005 * (xpos - starting_press_x), -0.005 * (ypos - starting_press_y), 0);
                break;
            case GeoScaling:
                models[cur_idx].scale = models[cur_idx].scale+ Vector3(0.005 * (xpos - starting_press_x), 0.005 * (ypos - starting_press_y), 0);
                break;
            case GeoRotation:
                models[cur_idx].rotation = models[cur_idx].rotation + Vector3(0.005 * (ypos - starting_press_y), 0.005 * (xpos - starting_press_x), 0);
                break;
            case ViewEye:
                main_camera.position = main_camera.position - Vector3(0.005 * (xpos - starting_press_x), 0.005 * (ypos - starting_press_y), 0);
                cout << "main_camera.position:" << main_camera.position << endl;
                setViewingMatrix();
                break;
            case ViewCenter:
                main_camera.center = main_camera.center - Vector3(0.005 * (xpos - starting_press_x), 0.005 * (ypos - starting_press_y), 0);
                cout << "main_camera.center:" << main_camera.center << endl;
                setViewingMatrix();
                break;
            case ViewUp:
                main_camera.up_vector = main_camera.up_vector - Vector3(0.005 * (xpos - starting_press_x), 0.005 * (ypos - starting_press_y), 0);
                cout << "main_camera.up_vector:" << main_camera.up_vector << endl;

                setViewingMatrix();
                break;
        }
    }
    starting_press_x = xpos;
    starting_press_y = ypos;
}
```

重繪:

考慮到邊長比例，有不同的範圍比

```
// Call back function for window reshape
void ChangeSize(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
    // [TODO] change your aspect ratio in both perspective and orthogonal view
    proj.aspect = (float)width / height;

    if ((float)width/height>1) {
        proj.left = -(float)width / height;
        proj.right = (float)width / height;
        proj.top = 1.0;
        proj.bottom = -1.0;
    }
    else {
        proj.left = -1;
        proj.right = 1;
        proj.top = (float)height/ width;
        proj.bottom = -(float)height / width;
    }

    if (cur_proj_mode == Perspective) {
        setPerspective();
    }
    else if (cur_proj_mode == Orthogonal) {
        setOrthogonal();
    }
    setViewingMatrix();


}
```

額外小改善:

```
void initParameter()
{
    proj.left = -1;
    proj.right = 1;
    proj.top = 1;
    proj.bottom = -1;
    proj.nearClip = 0.001;
    proj.farClip = 100.0;
    proj.fovy = 80;
    proj.aspect = (float)WINDOW_WIDTH / (float)WINDOW_HEIGHT;

    main_camera.position = Vector3(0.0f, 0.0f, 2.0f);
    main_camera.center = Vector3(0.0f, 0.0f, 0.0f);
    main_camera.up_vector = Vector3(0.0f, 1.0f, 0.0f);

    ChangeSize(NULL, WINDOW_WIDTH, WINDOW_HEIGHT);
    setViewingMatrix();
    setPerspective();   //set default projection matrix as perspective matrix
}
```

初始化時主動執行更新一次視窗大小，使正交投影不變型。