Computer Graphics HW1

111061553 謝霖泳

Implementation details

1. Translation 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;
}
```

照定義打出平移矩陣。

2. Scaling 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;
}
```

照定義打出伸縮矩陣。

3. Rotation matrix

```
Matrix4 rotateX(GLfloat val)
{
    Matrix4 mat;
    GLfloat theta = val * PI / 180.0;
    mat = Matrix4(
        1, 0, 0, 0,
        0, cos(theta), -sin(theta), 0,
        0, sin(theta), cos(theta), 0,
        0, 0, 0, 1
    );
    return mat;
}
```

先將角度轉弳度，並照定義打出x方向平移矩陣，y方向與z方向同理不再贅述。

4. Viewing matrix

```
void setViewingMatrix()
{
    Vector3 P1P2, P1P3;
    GLfloat Rx[3], Ry[3], Rz[3];
    Matrix4 R, T;
    P1P2 = main_camera.center - main_camera.position;
    P1P3 = main_camera.up_vector - main_camera.position;
    GLfloat p12[3] = {P1P2.x, P1P2.y, P1P2.z}, p13[3] = {P1P3.x,
P1P3.y, P1P3.z};

    for (int i = 0; i < 3; ++i)
        Rz[i] = -p12[i];
    Cross(p12, p13, Rx);
    Cross(Rz, Rx, Ry);
    Normalize(Rx);
    Normalize(Ry);
    Normalize(Rz);

    R = 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
    );
    T = translate(-main_camera.position);
    view_matrix = R * T;
}
```

先由`p1p2`得到`Rz`向量，再將`p1p2`與`p2p3`做外積得到Rx向量，再將`Rz`與`Rx`做外積得到`Ry`向量，有了上述三者normalize後即得R矩陣，並根據定義由`main_camera.position`得到`T`向量，將`R`、`T`兩矩陣相乘及為所求。

5. Orthogonal projection matrix

```
void setOrthogonal()
{
    cur_proj_mode = Orthogonal;
    GLfloat xmax = proj.right, xmin = proj.left;
    GLfloat ymax = proj.top, ymin = proj.bottom;
    GLfloat zfar = proj.farClip, znear = proj.nearClip;

    project_matrix = Matrix4(
        2.0 / (xmax - xmin), 0, 0, -((xmax + xmin) / (xmax - xmin)),
        0, 2.0 / (ymax - ymin), 0, -((ymax + ymin) / (ymax - ymin)),
        0, 0, -2.0 / (zfar - znear), -((zfar + znear) / (zfar -
znear)),
        0, 0, 0, 1.0
    );
}
```

照定義打出orthogonal projection matrix。

6. Persepective projection matrix

```
void setPerspective()
{
    cur_proj_mode = Perspective;
    GLfloat f = 1.0 / tan((proj.fovy * PI / 180) / 2);
    GLfloat far = proj.farClip, near = proj.nearClip;
    project_matrix = Matrix4(
        f / proj.aspect, 0, 0, 0,
        0, f, 0, 0,
        0, 0, (far + near) / (near - far), (2.0 * far * near) / (near
- far),
        0, 0, -1, 0
    );
}
```

照定義打出persepective projection matrix。

7. Window reshape

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

利用新的寬高計算出新的aspect ratio，並呼叫`setViewingMatrix()`重新設定viewing matrix，再依據現在的投影模式(orthogonal或perspective)設定相應的projection matrix。

8. Draw plane

```
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 = project_matrix * view_matrix;
    GLfloat mvp[16];
    // 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];

    // glBindVertexArray(quad.vao);
    glUniformMatrix4fv(iLocMVP, 1, GL_FALSE, mvp);

    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);

    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
0);
    glEnableVertexAttribArray(0);
    glGenBuffers(1, &VBO_COLOR);
    glBindBuffer(GL_ARRAY_BUFFER, VBO_COLOR);
    glBufferData(GL_ARRAY_BUFFER, sizeof(colors), colors,
GL_STATIC_DRAW);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
0);
    glEnableVertexAttribArray(1);

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glDrawArrays(GL_TRIANGLES, 0, 6);
}
```

先將MVP由row-major轉為column-major的形式，再呼叫OpenGL相關的API來設定相關的buffer。

9. Render scene

```
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
    MVP = project_matrix * view_matrix * T * R * S;

    // [TODO] 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];

    if (display_mode == SOLID_MODE)
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    else
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    // use uniform to send mvp to vertex shader
    glUniformMatrix4fv(iLocMVP, 1, GL_FALSE, mvp);
    glBindVertexArray(m_shape_list[cur_idx].vao);
    glDrawArrays(GL_TRIANGLES, 0, m_shape_list[cur_idx].vertex_count);
    drawPlane();
}
```

首先，利用現在model的`position`、`rotation`及`scale`計算出對應的矩陣`T`、`R`、`S`。之後在第二個todo中，將`project_matrix`、`view_matrix`、`T`、`R`、`S`全部乘起來可得`MVP`，並在第三個todo中將之轉為column-major的形式。

10. Key callback

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    // [TODO] Call back function for keyboard
    if (action == GLFW_PRESS) {
        switch (key) {
            case GLFW_KEY_W:
                display_mode = -display_mode;
                break;

            case GLFW_KEY_Z:
                cur_idx--;
                if (cur_idx < 0)
                    cur_idx = 4;
                break;

            case GLFW_KEY_X:
                cur_idx++;
                cur_idx = cur_idx % 5;
                break;

            case GLFW_KEY_O:
                setOrthogonal();
                break;

            case GLFW_KEY_P:
                setPerspective();
                break;

            case GLFW_KEY_T:
                cur_trans_mode = GeoTranslation;
                break;

            case GLFW_KEY_S:
                cur_trans_mode = GeoScaling;
                break;

            case GLFW_KEY_R:
                cur_trans_mode = GeoRotation;
                break;

            case GLFW_KEY_E:
                cur_trans_mode = ViewEye;
                break;

            case GLFW_KEY_C:
                cur_trans_mode = ViewCenter;
                break;

            case GLFW_KEY_U:
                cur_trans_mode = ViewUp;
                break;

            case GLFW_KEY_I:
                cout << "Information:\n";
                cout << "(1) Translation Matrix:\n" << translate(models[cur_idx].position) << "\n";
                cout << "(2) Rotation Matrix\n" << rotate(models[cur_idx].rotation) << "\n";
                cout << "(3) Scaling Matrix\n" << scaling(models[cur_idx].scale) << "\n";
                cout << "(4) Viewing Matrix\n" << view_matrix << "\n";
                cout << "(5) Projection Matrix\n" << project_matrix << "\n";
                break;

            default:
                break;
        }
    }
}
```

判斷現在鍵盤按下的案件執行對應的設定。其中，按下`Z`時，我會讓model的`cur_idx`減少1，按下`X`時則反之。

11. Scroll callback

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    // [TODO] scroll up positive, otherwise it would be negtive
    GLfloat scaling_factor = 100.0;
    GLfloat diff = yoffset / scaling_factor;
    switch (cur_trans_mode) {
        case GeoTranslation:
            models[cur_idx].position.z += diff;
            break;

        case GeoRotation:
            models[cur_idx].rotation.z += yoffset;
            break;

        case GeoScaling:
            models[cur_idx].scale.z += diff;
            break;

        case ViewEye:
            main_camera.position.z += diff;
            setViewingMatrix();
            break;

        case ViewCenter:
            main_camera.center.z += diff;
            setViewingMatrix();
            break;

        case ViewUp:
            main_camera.up_vector.z += diff;
            setViewingMatrix();
            break;

        default:
            break;
    }
}
```

利用滾輪滾動量配合`cur_trans_mode`對model z方向的相應變量做出改變，為了不讓改變過於劇烈影響觀察，我設了一個scaling factor減低變化量以方便觀察。

12. Mouse button callback

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

在滑鼠左鍵或右鍵被按下時，將`mouse_pressed`設為`true`，反之則設為`false`。

13. Cursor position callback

```
static void cursor_pos_callback(GLFWwindow* window, double xpos,
double ypos)
{
    // [TODO] cursor position callback function
    prevx = curx;
    prevy = cury;
    curx = xpos;
    cury = ypos;
    GLfloat dx = curx - prevx, dy = cury - prevy;
    GLfloat scaling_factor = 60.0;

    if (starting_press_x < 0 && starting_press_y < 0) {
        starting_press_x = xpos;
        starting_press_y = ypos;
    }
    if (!mouse_pressed)
        return;
    if (first_time) {
        first_time = false;
        starting_press_x = xpos;
        starting_press_y = ypos;
        return;
    }

    switch (cur_trans_mode) {
        case GeoTranslation:
            models[cur_idx].position.x += dx / scaling_factor;
            models[cur_idx].position.y += dy / scaling_factor;
            break;

        case GeoScaling:
            models[cur_idx].scale.x += dx / scaling_factor;
            models[cur_idx].scale.y += dy / scaling_factor;
            break;

        case GeoRotation:
            models[cur_idx].rotation.x += dx * 0.5;
            models[cur_idx].rotation.y += dx * 0.5;
            break;

        case ViewEye:
            main_camera.position.x += dx / scaling_factor;
            main_camera.position.y += dy / scaling_factor;
            setViewingMatrix();
            break;

        case ViewCenter:
            main_camera.center.x += dx / scaling_factor;
            main_camera.center.y += dy / scaling_factor;
            setViewingMatrix();
            break;

        case ViewUp:
            main_camera.up_vector.x += dx / scaling_factor;
            main_camera.up_vector.y += dy / scaling_factor;
            setViewingMatrix();
            break;

        default:
            break;
    }
    if (mouse_pressed) {
        starting_press_x = xpos;
        starting_press_y = ypos;
    }
}
```

更新滑鼠的位置，並與先前滑鼠位置計算水平與鉛直方向位移量`dx`、`dy`。根據兩方向的位移量配合`cur_trans_mode`改變model x y方向的相應變數。

14. Load models

```
void setupRC()
{
    // setup shaders
    setShaders();
    initParameter();

    // OpenGL States and Values
    glClearColor(0.2, 0.2, 0.2, 1.0);
    vector<string> model_list{ "../ColorModels/bunny5KC.obj",
"../ColorModels/dragon10KC.obj", "../ColorModels/lucy25KC.obj",
"../ColorModels/teapot4KC.obj", "../ColorModels/dolphinC.obj"};
    // [TODO] Load five model at here
    for (int i = 0; i < model_list.size(); ++i) {
        cout << "Loading model #" << i << "\n";
        LoadModels(model_list[i]);
    }
}
```

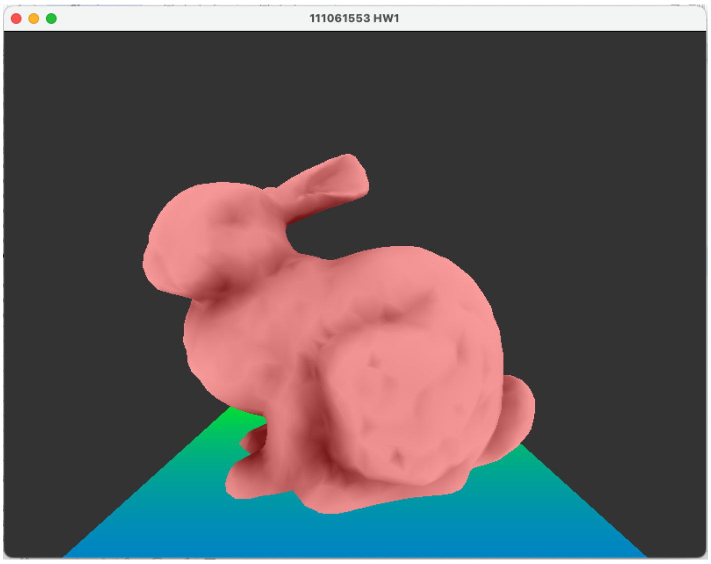
將五個model load進來。

15. Vertex shader

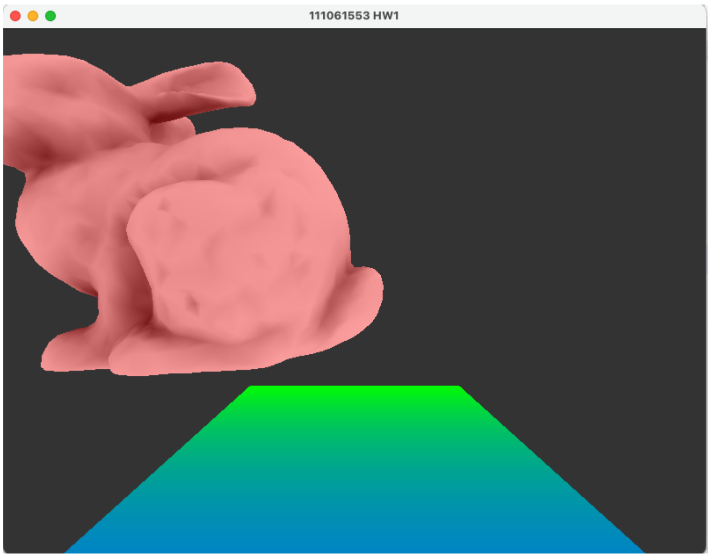
```
void main()
{
    // [TODO]
    gl_Position = mvp * vec4(aPos.x, aPos.y, aPos.z, 1.0);
    vertex_color = aColor;
}
```

Some screen shots

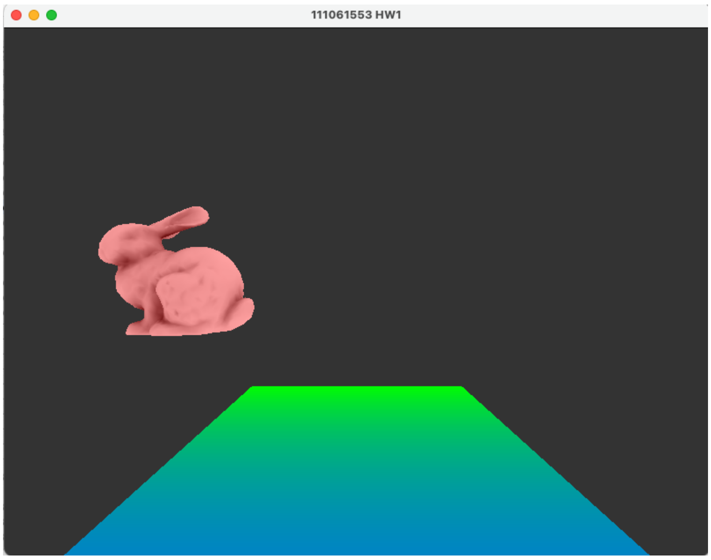
- 初始畫面



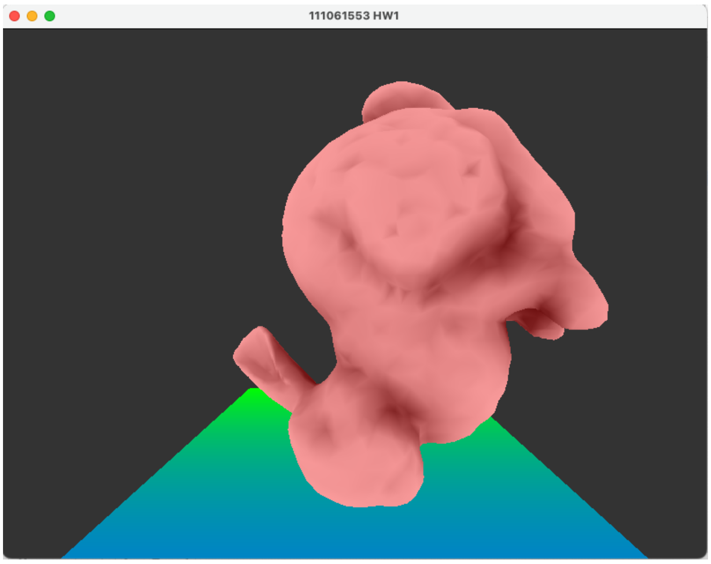
- 滑鼠拖曳



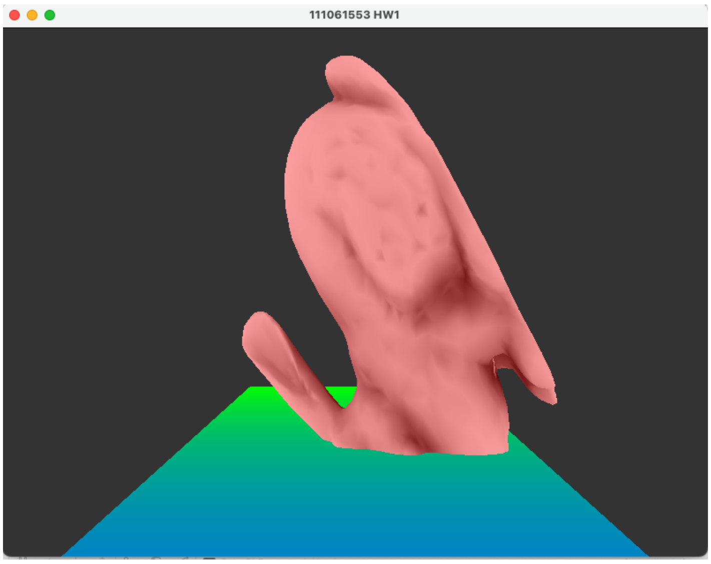
- 滾輪控制



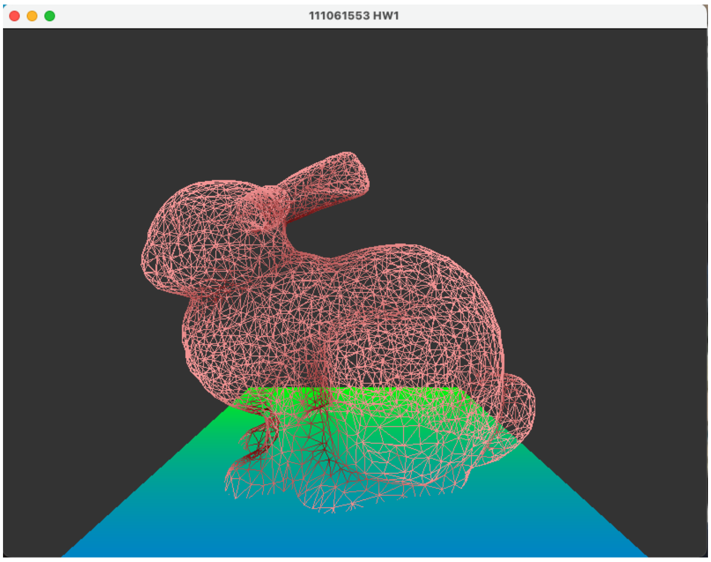
- Rotation



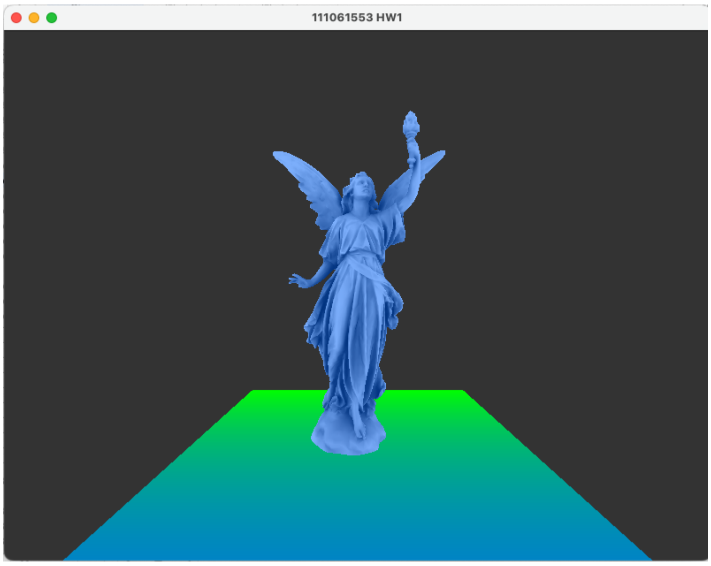
- Scaling

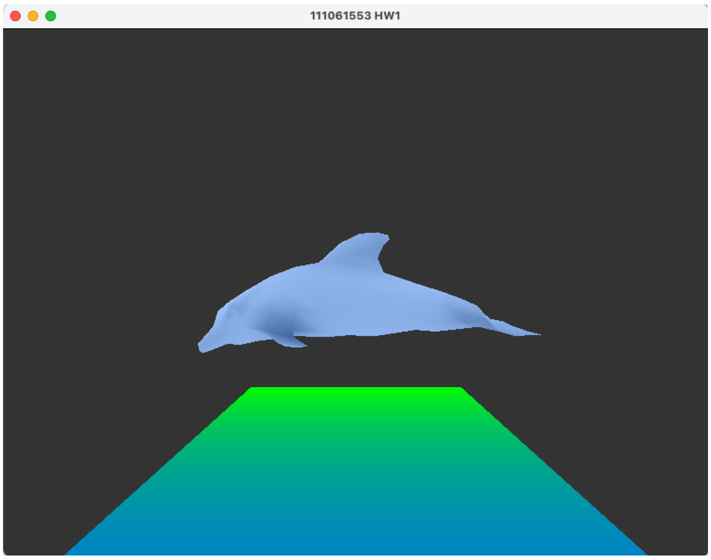


- wireframe mode

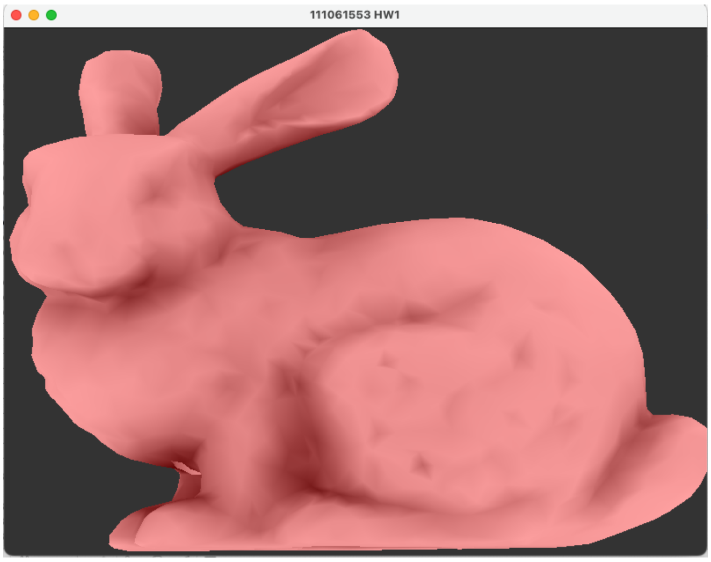


- Switch between models

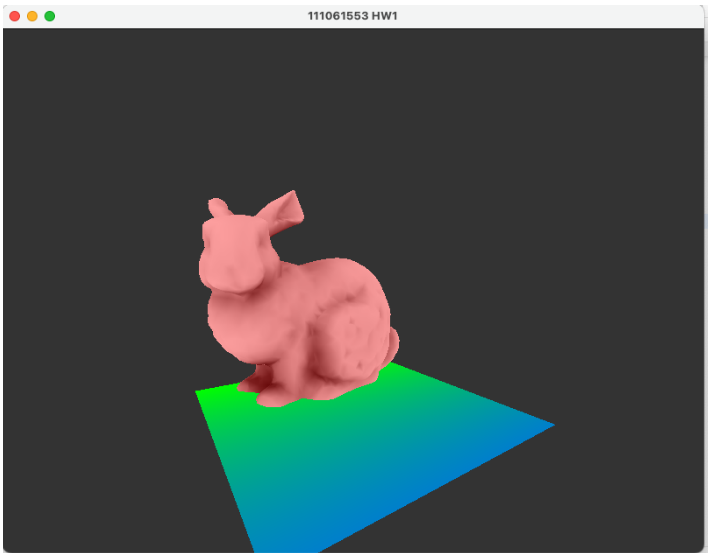




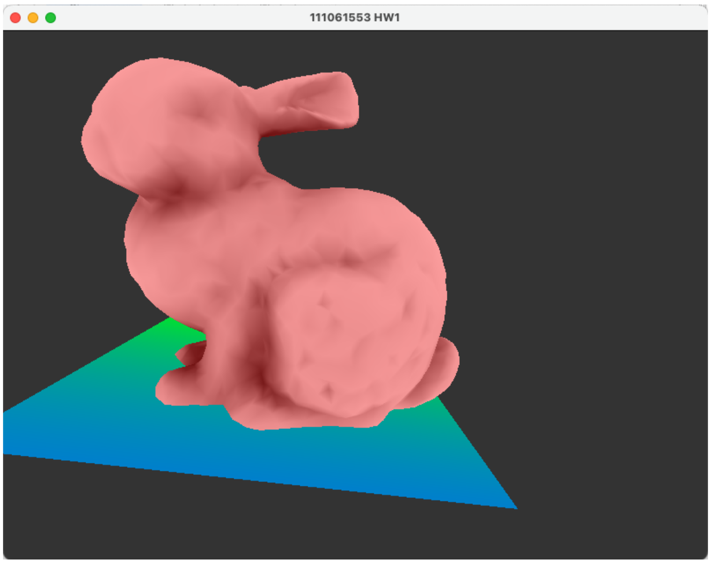
- orthogonal projection



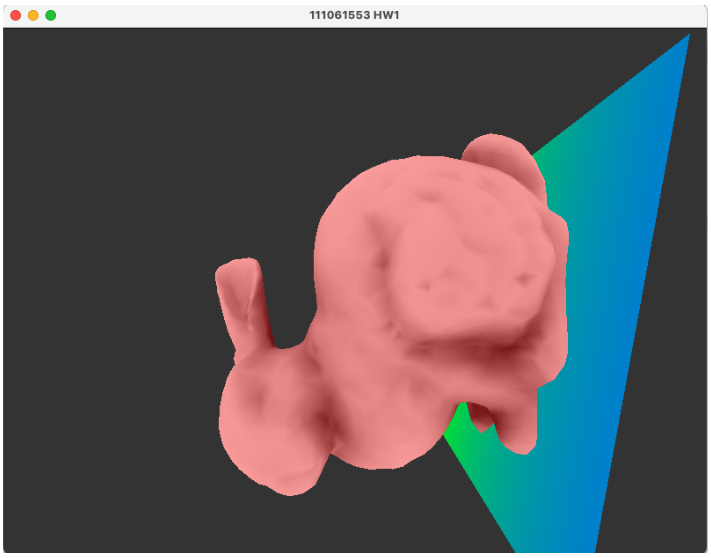
- translate eye position mode



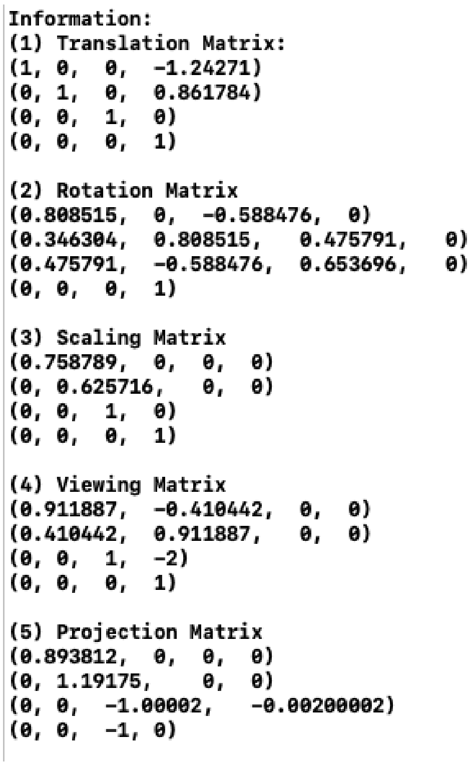
- translate viewing center position mode



- translate camera up vector position mode



- print information



```
Information:
(1) Translation Matrix:
(1, 0,  0,  -1.24271)
(0, 1,  0,  0.861784)
(0, 0,  1,  0)
(0, 0,  0,  1)

(2) Rotation Matrix
(0.808515,  0,  -0.588476,  0)
(0.346304,  0.808515,   0.475791,   0)
(0.475791,  -0.588476,  0.653696,   0)
(0, 0,  0,  1)

(3) Scaling Matrix
(0.758789,  0,  0,  0)
(0, 0.625716,   0,  0)
(0, 0,  1,  0)
(0, 0,  0,  1)

(4) Viewing Matrix
(0.911887,  -0.410442,  0,  0)
(0.410442,  0.911887,   0,  0)
(0, 0,  1,  -2)
(0, 0,  0,  1)

(5) Projection Matrix
(0.893812,  0,  0,  0)
(0, 1.19175,    0,  0)
(0, 0,  -1.00002,   -0.00200002)
(0, 0,  -1, 0)
```