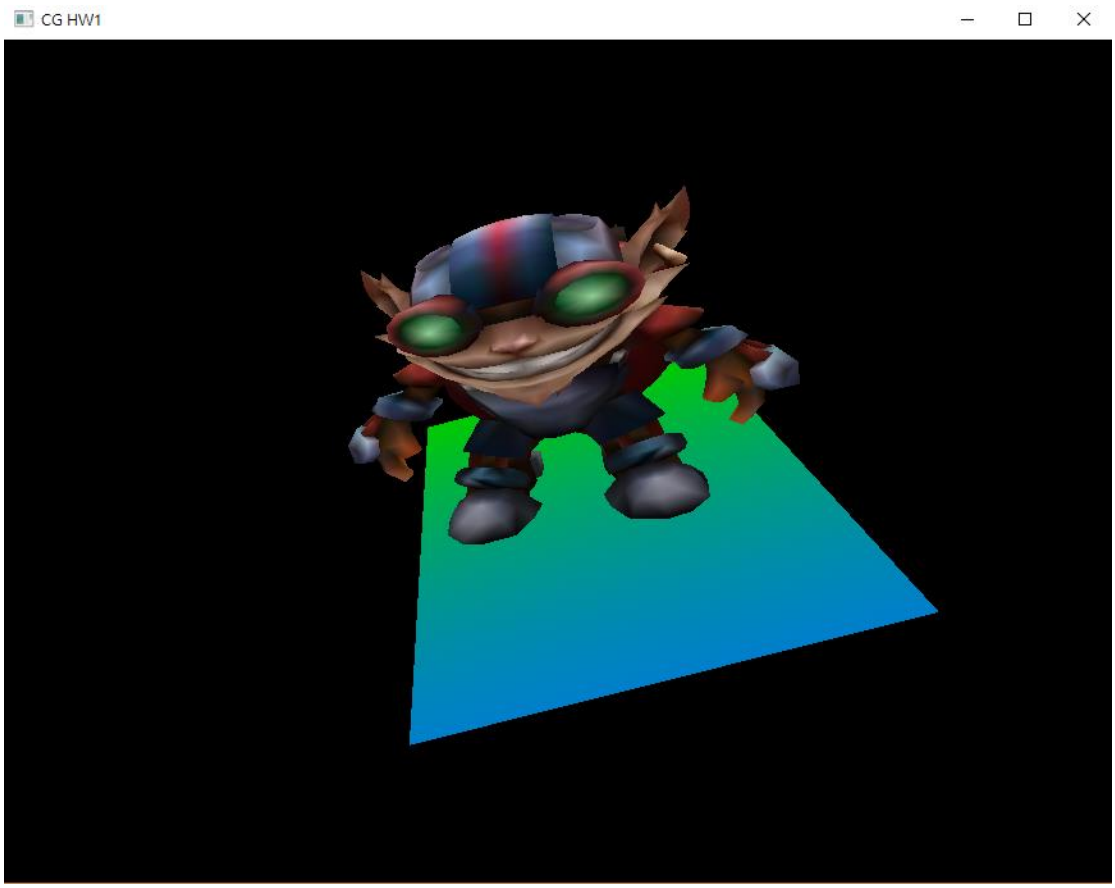
Computer Graphic Hw1

- Introduction
  - CS19 104062202 吳柏鋙
  - Implement the Geometrical transformation(translation, scaling, rotation), Viewing transformation(eye, center), Projection(orthogonal, perspective)
  - All works are followed by TODO in code and lecture slide chapter.5
  - Work display by screenshot



- Explanation of works about transformation, code and its reference
  - Translate

```
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;
}
```

Translation in Homogeneous Rep.

◆ 4x4 matrix form

$$\begin{aligned} x' &= x + d_x \\ y' &= y + d_y \\ z' &= z + f_z \end{aligned} \longrightarrow \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} \longrightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = T(d_x, d_y, d_z) \cdot P, \qquad T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

39

- Scaling

```
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;
}
```

Scaling in Homogeneous Rep.

◆ 4x4 matrix form

$$\begin{aligned} x' &= s_x \cdot x \\ y' &= s_y \cdot y \\ z' &= s_z \cdot z \end{aligned} \longrightarrow \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad P' = S(s_x, s_y, s_z) \cdot P$$

40

- Rotate in X-axis

```
// [Done] given a float value then o
Matrix4 rotateX(GLfloat val)
{
    Matrix4 mat;
    mat = Matrix4(
        1, 0, 0, 0,
        0, cosf(val), -sinf(val), 0,
        0, sinf(val), cosf(val), 0,
        0, 0, 0, 1
    );
    return mat;
}
```

Rotation in Homogeneous Rep.

◆ 4x4 matrix form (Rotate in x-axis)

$$\begin{aligned} x' &= x \\ y' &= y \cdot \cos\theta - z \cdot \sin\theta \\ z' &= y \cdot \sin\theta + z \cdot \cos\theta \end{aligned} \longrightarrow \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = R_x(\theta) \cdot P$$

41

- Rotate at Y-axis

```
Matrix4 rotateY(GLfloat val)
{
    Matrix4 mat;
    mat = Matrix4(
        cosf(val), 0, sinf(val), 0,
        0, 1, 0, 0,
        -sinf(val), 0, cosf(val), 0,
        0, 0, 0, 1
    );
    return mat;
}
```

Rotation in Homogeneous Rep.

◆ 4x4 matrix form (Rotate in y-axis)

$$\begin{aligned} x' &= x \cdot \cos\theta + z \cdot \sin\theta \\ y' &= y \\ z' &= -x \cdot \sin\theta + z \cdot \cos\theta \end{aligned} \longrightarrow \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

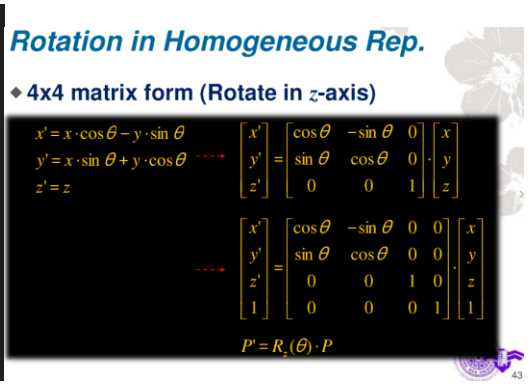
$$\longrightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = R_y(\theta) \cdot P$$

42

- Rotate at Z-axis

```
Matrix4 rotateZ(GLfloat val)
{
    Matrix4 mat;
    mat = Matrix4(
        cosf(val), -sinf(val), 0, 0,
        sinf(val), cosf(val), 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1
    );
    return mat;
}
```



- Viewing matrix

```
// [Done] Compute viewing matrix according 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 = Rz.cross(Rx);

    view_matrix = Matrix4(
        Rx.x, Rx.y, Rx.z, 0,
        Ry.x, Ry.y, Ry.z, 0,
        Rz.x, Rz.y, Rz.z, 0,
        0, 0, 0, 1
    )*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
    );
}
```

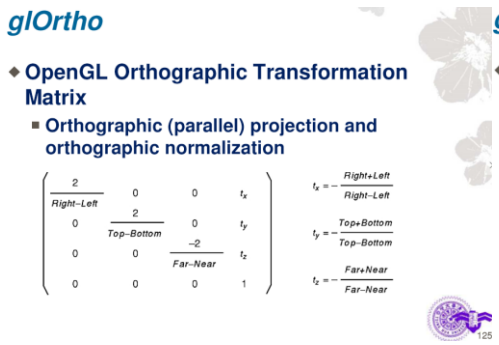
Fast Viewing Matrix Derivation

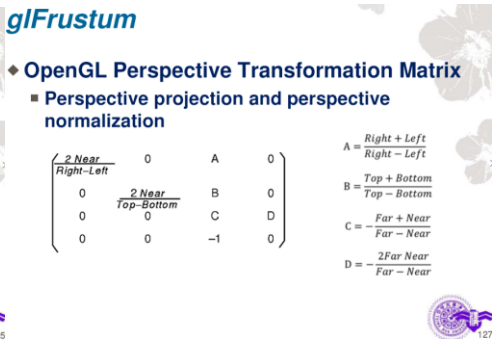
- **Final Viewing Matrix is defined as**

$$M_{view} = R \bullet T = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Orthogonal projection matrix (glOrtho at slide.125)

```
// [Done] compute orthogonal projection matrix
void setOrthogonal()
{
    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
    );
}
```





- Perspective projection matrix (glFrustum at slide.127)
  - I first try the one with gluPerspective in slide.130 but it seems not work well, so I use the other one and it works.

```
// [Done] compute persepective(perspective) projection matrix
void setPerspective()
{
    /*
    GLfloat f = cos(proj.fovy / 2) / sin(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*proj.farClip*proj.nearClip/(proj.nearClip - proj.farClip),
        0, 0, -1, 0
    );*/
    project_matrix = Matrix4(
        2*proj.nearClip/(proj.right-proj.left), 0, (proj.right+proj.left)/(proj.right-proj.left), 0,
        0, 2*proj.nearClip/(proj.top-proj.bottom), (proj.top+proj.bottom)/(proj.top-proj.bottom), 0,
        0, 0, -((proj.farClip+proj.nearClip)/(proj.farClip-proj.nearClip)), -(2*proj.farClip*proj.nearClip/(proj.farClip-proj.nearClip)),
        0, 0, -1, 0
    );
}
```

- MVP
  - Followed by hint in comment

```
void drawModel(model* model)
{
    Matrix4 T, R, S;
    T = translate(model->position);
    R = rotate(model->rotation);
    S = scaling(model->scale);

    // [Done] Assign MVP correct value
    // [HINT] MVP = projection_matrix * view_matrix * ??? * ??? * ???
    Matrix4 MVP;
    MVP = project_matrix * view_matrix * T * R * S;
    GLfloat mvp[16];
```

- Input setting
  - Use a global variable (mode) to record which type of transformation is current used and another global Boolean variable (project_otho) to record which type of projection is now.
  - Define

```
69      bool project_otho;
70      int mode;    // 1:translation 2:rotation 3:scaling 4:center 5:eye
```

  - Initialize in function initParameter()

```
742         project_otho = true; // paul
743         mode = 1; // translation
```

  - Change value in function onKeyboard()

```
case 'o':
    if (!project_otho)
        setOrthogonal();
    project_otho = true;
    break;
case 'p':
    if (project_otho)
        setPerspective();
    project_otho = false;
    break;
```

```
case 'e':
    mode = 5;
    printf("set to viewing transformation : eye.\n");
    break;
case 'c':
    mode = 4;
    printf("set to viewing transformation : center.\n");
    break;
```

```
683         case 't':
684             mode = 1;
685             printf("set to Geometrical transformation : translation.\n");
686             break;
687         case 's':
688             mode = 3;
689             printf("set to Geometrical transformation : scaling.\n");
690             break;
691         case 'r':
692             mode = 2;
693             printf("set to Geometrical transformation : rotation.\n");
694             break;
```

- Used in mouse input to decide the desired control
  - In function onMouse for wheel up

```
case GLUT_WHEEL_UP:
    printf("wheel up      \n");
    // [TODO] assign corresponding operation
    if (mode == 5){
        main_camera.position.z -= 0.025;
        setViewingMatrix();
        printf("Camera Position = ( %f , %f , %f )\n", main_camera.position.x, main_camera.position.y, main_camera.position.z);
    }
    else if (mode == 4){
        main_camera.center.z += 0.1;
        setViewingMatrix();
        printf("Camera Viewing Direction = ( %f , %f , %f )\n", main_camera.center.x, main_camera.center.y, main_camera.center.z);
    }
    else if (mode == 6) // won't be used in HW1
    {
        main_camera.up_vector.z += 0.33;
        setViewingMatrix();
        printf("Camera Up Vector = ( %f , %f , %f )\n", main_camera.up_vector.x, main_camera.up_vector.y, main_camera.up_vector.z);
    }
    else if (mode == 1){
        models[cur_idx].position.z += 0.1;
    }
    else if (mode == 3){
        models[cur_idx].scale.z += 0.025;
    }
    else if (mode == 2){
        models[cur_idx].rotation.z += (PI/180.0) * 5;
    }
    break;
```

  - In function onMouse for wheel down

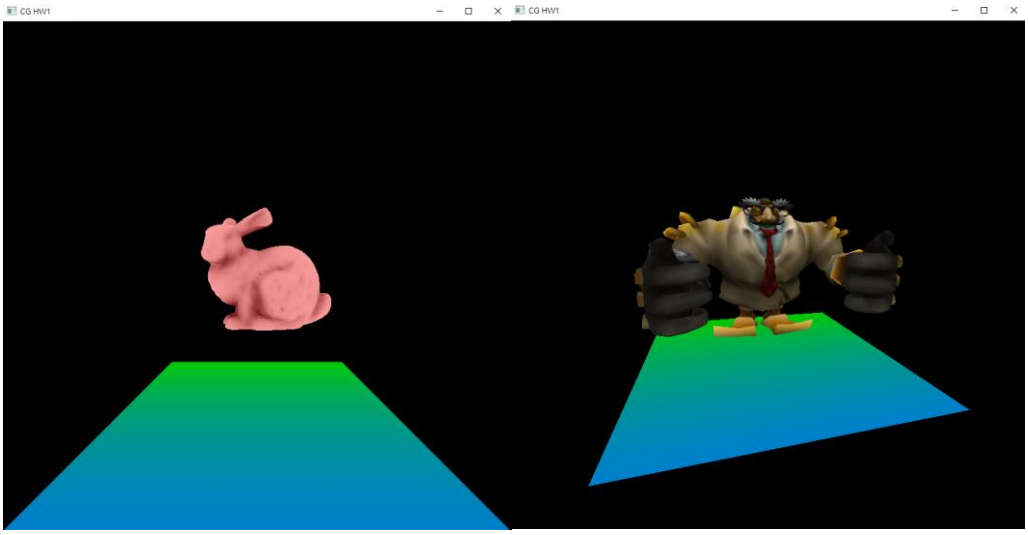
```
case GLUT_WHEEL_DOWN:
    printf("wheel down    \n");
    // [TODO] assign corresponding operation
    if (mode == 5){
        main_camera.position.z += 0.025;
        setViewingMatrix();
        printf("Camera Position = ( %f , %f , %f )\n", main_camera.position.x, main_camera.position.y, main_camera.position.z);
    }
    else if (mode == 4){
        main_camera.center.z -= 0.33;
        setViewingMatrix();
        printf("Camera Viewing Direction = ( %f , %f , %f )\n", main_camera.center.x, main_camera.center.y, main_camera.center.z);
    }
    else if (mode == 6) // won't be used in HW1
    {
        main_camera.up_vector.z -= 0.33;
        setViewingMatrix();
        printf("Camera Up Vector = ( %f , %f , %f )\n", main_camera.up_vector.x, main_camera.up_vector.y, main_camera.up_vector.z);
    }
    else if (mode == 1){
        models[cur_idx].position.z -= 0.33;
    }
    else if (mode == 3){
        models[cur_idx].scale.z -= 0.025;
    }
    else if (mode == 2){
        models[cur_idx].rotation.z -= (PI / 180.0) * 5;
    }
    break;
```

  - In function onMouseMotion() for dragging

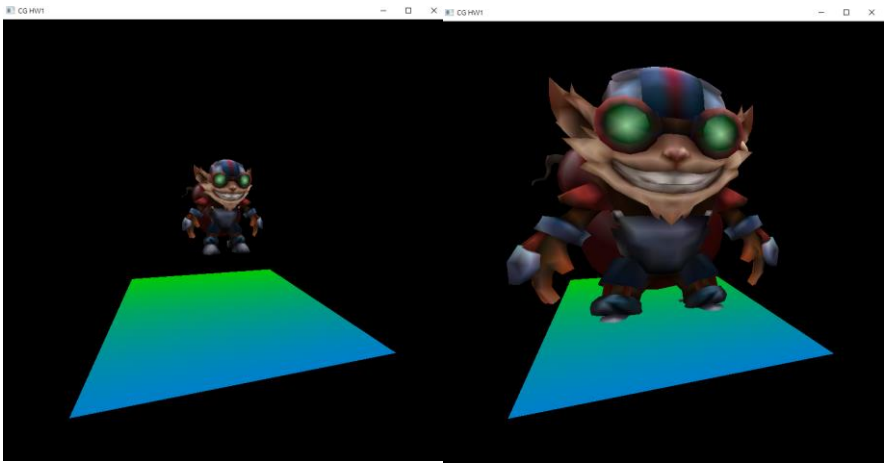
```
// [TODO] assign corresponding operation
if (mode == 5){
    main_camera.position.x += diff_x*(1.0 / 400.0);
    main_camera.position.y += diff_y*(1.0 / 400.0);
    setViewingMatrix();
    printf("Camera Position = ( %f , %f , %f )\n", main_camera.position.x, main_camera.position.y, main_camera.position.z);
}
else if (mode == 4){
    main_camera.center.x += diff_x*(1.0 / 400.0);
    main_camera.center.y += diff_y*(1.0 / 400.0);
    setViewingMatrix();
    printf("Camera Viewing Direction = ( %f , %f , %f )\n", main_camera.center.x, main_camera.center.y, main_camera.center.z);
}
else if (mode == 6) // won't be used in HW1
{
    main_camera.up_vector.x += diff_x*0.1;
    main_camera.up_vector.y += diff_y*0.1;
    setViewingMatrix();
    printf("Camera Up Vector = ( %f , %f , %f )\n", main_camera.up_vector.x, main_camera.up_vector.y, main_camera.up_vector.z);
}
else if (mode == 1){
    models[cur_idx].position.x += diff_x*(1.0 / 400.0);
    models[cur_idx].position.y -= diff_y*(1.0 / 400.0);
}
else if (mode == 3){
    models[cur_idx].scale.x += diff_x*0.025;
    models[cur_idx].scale.y += diff_y*0.025;
}
else if (mode == 2){
    models[cur_idx].rotation.x += PI / 180.0*diff_y*(45.0 / 400.0);
    models[cur_idx].rotation.y += PI / 180.0*diff_x*(45.0 / 400.0);
}
printf("%18s(): (%d, %d) mouse move\n", __FUNCTION__, x, y);
```

- Works Display
  - Orthogonal projection

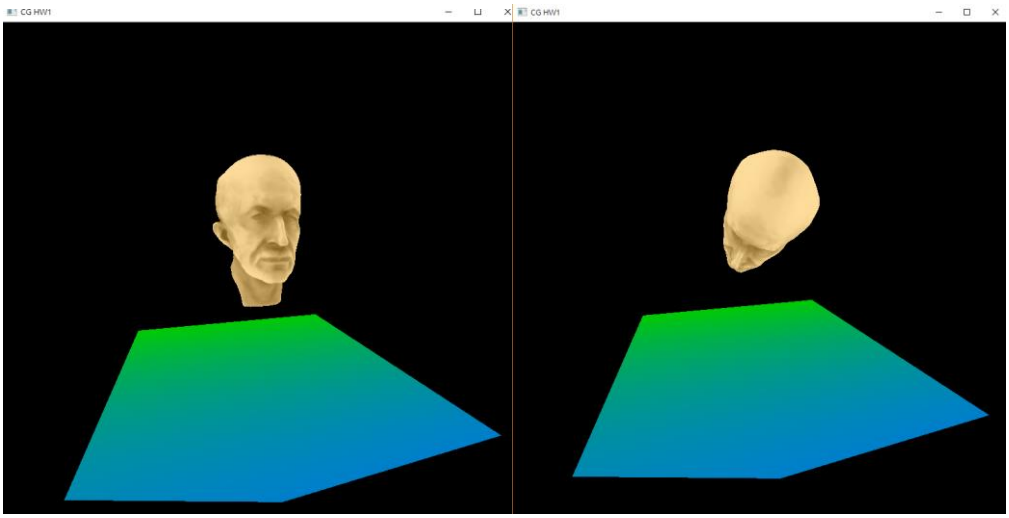
  - Perspective projection



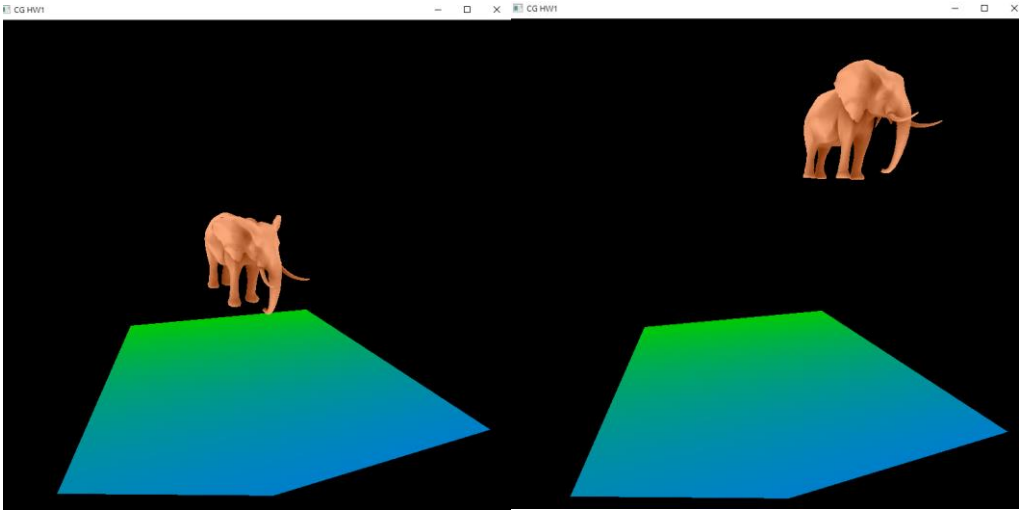
  - Scaling



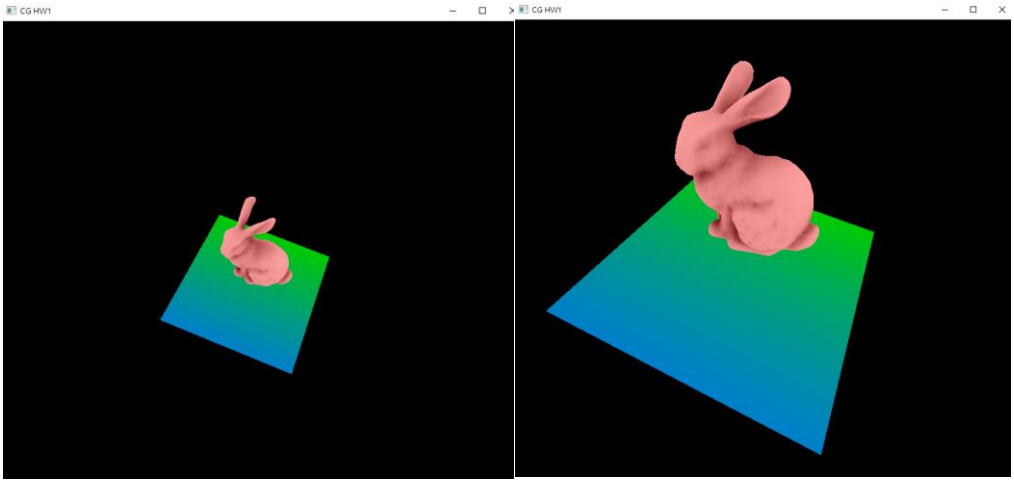
■ Rotate



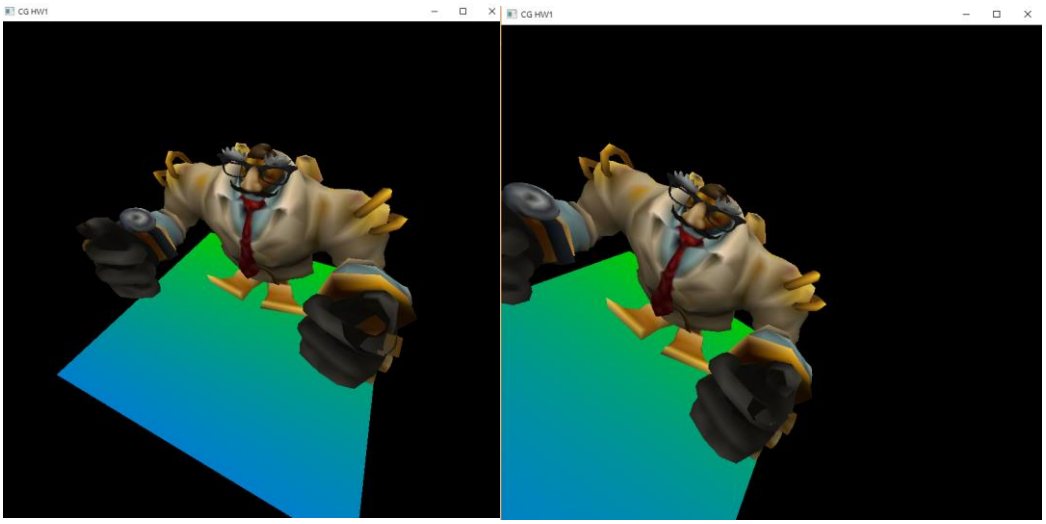
■ Translate



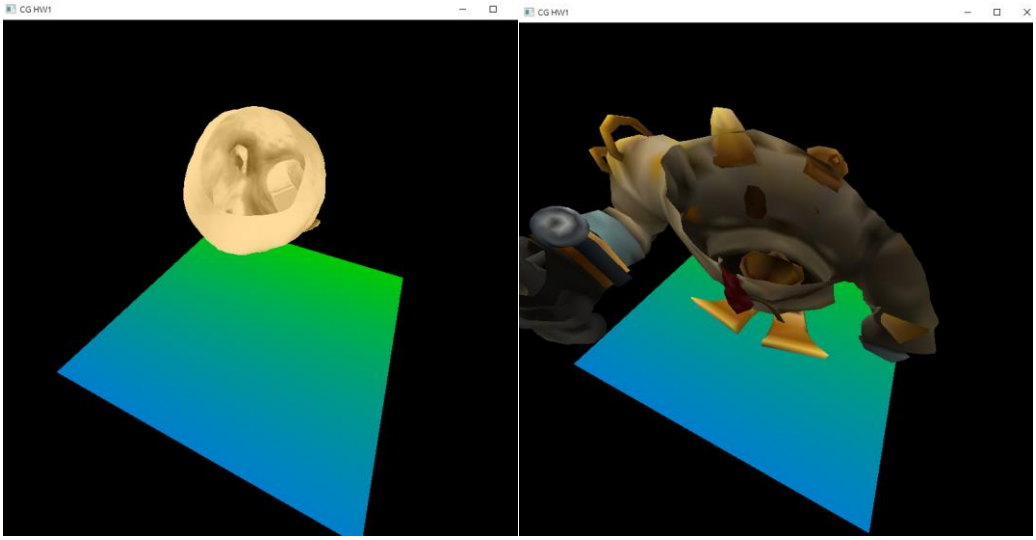
■ Eye change



- Center change



- Projection on the model to see inside



➢ Difficulty and thought

- Thanks to lots help and clear TODO from teaching assistant, it is not hard to discover how to do. The most difficult part for me is trace the code and understand what they are doing. It is amazing that such kinds of change in matrix could lead to different showing projection and view.