Tech document

The main parts I edited are MazeWindow::draw() and Maze::Draw_View(). The below will elaborate how I completed this project. In this project, I used glm to help me do only nothing but the matrix calculation.

1. Project between spaces

First part is how I built matrix to project from model space to screen. This part is done in function MazeWindow::draw().

1-a. construct model-view matrix

This matrix helps us translate coordinate from model space to view space.

```
// model to view
float eye[3] = {
maze->viewer_posn[Maze::Y],
0.0f,
maze->viewer_posn[Maze::X]
};

float center[3] = {
    eye[Maze::X] + sin(Maze::To_Radians(maze->viewer_dir)),
    eye[Maze::Y],
    eye[Maze::Z] + cos(Maze::To_Radians(maze->viewer_dir))
};

glm::vec3 up(0.0, 1.0, 0.0);

float length = sqrt(pow((eye[Maze::X] - center[Maze::X]), 2) + pow

glm::vec3 w(
    eye[Maze::X] - center[Maze::X] / length,
    eye[Maze::Y] - center[Maze::Y] / length,
    eye[Maze::Z] - center[Maze::Z] / length);

glm::vec3 u = glm::cross(up, w);
glm::vec3 v = glm::cross(w, u);
```

Vector eye is the origin of camera, vector center is made to derive the gaze direction, and vector up is the vector always vertical point to the top. In this case vector up refers to y axis.

Vector w, normal to the view plane, is the normalized viewing direction.

Vector u, right vector to the view plane, can be derived from the cross product of up and w.

Vector v, up vector to the view plane, can be derived from the cross product of w and u.

W, u, and v will form the coordinate system of view space.

```
    float rotation_matrix[16] = {
        u[Maze::X], v[Maze::X], w[Maze::X], 0.0,
        u[Maze::Y], v[Maze::Y], w[Maze::Y], 0.0,
        u[Maze::Z], v[Maze::Z], w[Maze::Z], 0.0,
        0.0, 0.0, 0.0, 1.0
    };

    float translation_matrix[16] = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        -eye[Maze::X], -eye[Maze::Y], -eye[Maze::Z], 1.0
    };

    // matrix to glm
    glm::mat4x4 rotation = glm::make_mat4(rotation_matrix);
    glm::mat4x4 translation = glm::make_mat4(translation_matrix);
    glm::mat4x4 view = rotation * translation;
```

Model to view matrix is a multiplication of a rotation and translation. I constructed the rotation matrix and translation matrix same with the power point of the class but with column-major style.

1-b. constructing view to clip space matrix (perspective projection matrix)

This matrix translate object from view space to clip space with perspective projection.

```
    // perspective projection
    float aspect = (float)w() / h();
    float t = maze->n * tan(Maze::To_Radians(maze->viewer_fov) * 0.5);            //top
    float r = maze->n * tan(Maze::To_Radians(maze->viewer_fov * aspect) * 0.5);     //right

    float perspective_matrix[16] = {
        maze->n / r, 0.0, 0.0, 0.0,
        0.0, maze->n / t, 0.0, 0.0,
        0.0, 0.0, (maze->n + maze->f) / (maze->n - maze->f), -1.0,
        0.0, 0.0, 2 * maze->n * maze->f / (maze->n - maze->f), 0.0
    };
```

Aspect is the ratio of window width and window height.

T, the top of view frustum, was derived from the field of view (y axis).

R, the right boundary of view frustum, was from the field of view (x axis).

FOV of the x axis was generated by the aspect (ratio of window width and window height).

Still, I constructed the perspective projection matrix same as the power point of the class but in column-major style.

1-c. go to draw

After the matrix needed were all constructed, I passed them to Maze::Draw_View() to draw the maze.

```
    // draw
    maze->Draw_View(focal_length, view, perspective);
```

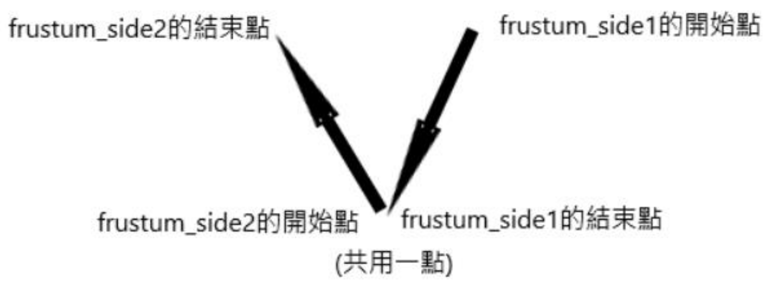
2. How to draw the maze

2-a. Maze::Draw_View()

```cpp
void Maze::
Draw_View(const float focal_dist, glm::mat4x4 view, glm::mat4x4 perspective)
//======================================================================
{
    frame_num++;
    glDisable(GL_DEPTH_TEST);

    for (int i = 0; i < this->num_cells; i++) {
        this->cells[i]->footprint = false;
    }

    LineSeg left_frustrum(this->n * tan(To_Radians(this->viewer_fov * 0.5f)), -this->n, this->f * tan(To_Radians(thi
    LineSeg right_frustrum(-this->f * tan(To_Radians(this->viewer_fov * 0.5f)), -this->f, -this->n * tan(To_Radians(

    Draw_Cell(this->view_cell, left_frustrum, right_frustrum, view, perspective);
}
```

First, I reset every cell to be undrawn.

Secondly, I defined the left and right boundaries of view frustum with the way the figure shows.



The reason that I defined the start point of left boundary and the end point of right boundary is that it is easier to determine whether the point is in the view frustum. (Point at right hand side of the boundary always is the view frustum, and vise versa)

2-b. Clipping

Clipping helps us determine whether to draw the wall and correct end points to be drawn. Returning true means the wall needs to be drawn.

There are three condition of the walls. First, the whole wall is in the view frustum. If so, there is no need to find new end point, and we can simply return true. Second, the wall is completely outside the view frustum. If so, the wall does not need to be drawn, and we simply return false. Last, a part of the wall is in the view frustum, and the rest of the wall is not. In this case, we need to find the new end point and then return true.

```
bool Maze::
clipping(LineSeg frustum_edge, glm::vec4& start, glm::vec4& end) {

    if (frustum_edge.Point_Side(start[0], start[2]) == Edge::RIGHT) {
        if (frustum_edge.Point_Side(end[0], end[2]) == Edge::LEFT) {
            // end point is outside the frustum
            // need to get the new end point
            LineSeg wall(start[0], start[2], end[0], end[2]);
            float percent = wall.Cross_Param(frustum_edge);
            float newX = start[0] + (end[0] - start[0]) * percent;
            float newZ = start[2] + (end[2] - start[2]) * percent;

            end[0] = newX;
            end[2] = newZ;
        }
    }
    else if (frustum_edge.Point_Side(end[0], end[2]) == Edge::RIGHT) {
        // start point is outside the frustum
        // need to get the new start point
        LineSeg wall(start[0], start[2], end[0], end[2]);
        float percent = wall.Cross_Param(frustum_edge);
        float newX = start[0] + (end[0] - start[0]) * percent;
        float newZ = start[2] + (end[2] - start[2]) * percent;

        start[0] = newX;
        start[2] = newZ;

    }
    else {
        // the wall is outside the frustum
        // no need to draw the wall
        return false;
    }

    return true;
}
```

The first two if else if statement show the third condition, a part of the wall in view frustum. To find the new end point, I first find the percentage of the line segment made of start point and the point of intersection accounting for the whole wall, and then I can calculate the point of intersection, which is the new end point.

2-c. recursively draw cell

In function Maze::Draw_Cell(), I denoted the current cell I tried to draw as drawn (footprint = true). Then, I walked through the edges of the cell.

```
glm::vec4 start(
    current_cell->edges[i]->endpoints[Edge::START]->posn[Y],
    1.0,
    current_cell->edges[i]->endpoints[Edge::START]->posn[X],
    1.0
);

glm::vec4 end(
    current_cell->edges[i]->endpoints[Edge::END]->posn[Y],
    1.0,
    current_cell->edges[i]->endpoints[Edge::END]->posn[X],
    1.0
);

// translate to view space
start = view * start;
end = view * end;

// clip the wall
if (!clipping(left_frustrum, start, end) || !clipping(right_frustrum, start, end))
    continue;
```

First step is to translate all the end point of the wall into view space and clip the wall. If the wall is outside the view frustum, we go for next wall.

```
if (current_cell->edges[i]->opaque) {
    // opaque wall

    if (!clipping(front, start, end))
        continue;

    // translate to canonical view volume
    start = perspective * start;
    end = perspective * end;

    if (start[Z] > this->n || end[Z] > this->n) {

        // if in the front, draw wall
        glBegin(GL_POLYGON);
        glColor3fv(current_cell->edges[i]->color);
        glVertex2f(start[X] / start[Z], start[Y] / start[Z]);
        glVertex2f(end[X] / end[Z], end[Y] / end[Z]);
        glVertex2f(end[X] / end[Z], -end[Y] / end[Z]);
        glVertex2f(start[X] / start[Z], -start[Y] / start[Z]);
        glEnd();
    }

}
```

If the wall is opaque and needs to be drawn, we translate it with the perspective projection matrix to the screen. The x and y are divided by z because z is our depth, and we can make the wall in correct order in this way.

```
else {
    // transparent wall

    if (current_cell->edges[i]->Neighbor(current_cell) != NULL) {
        // recursive when there is another cell behind

        float Lx = 0.0, Ly = 0.0, Rx = 0.0, Ry = 0.0;
        LineSeg center_frustum(0.0, 0.0, (start[X] + end[X]) / 2, (start[Z] + end[Z]) / 2);

        // whole wall in the frustum, in 2 condition
        if (center_frustum.Point_Side(start[X], start[Z]) == Edge::RIGHT && center_frustum.Point_Side(end[X], end[Z]) == Edge::LEFT) {
            Lx = end[X];
            Ly = end[Z];
            Rx = start[X];
            Ry = start[Z];
        }
        else if (center_frustum.Point_Side(start[X], start[Z]) == Edge::LEFT && center_frustum.Point_Side(end[X], end[Z]) == Edge::RIGHT) {
            Lx = start[X];
            Ly = start[Z];
            Rx = end[X];
            Ry = end[Z];
        }

        // new frustum
        LineSeg newL(Lx, Ly, Lx / Ly * -this->f, -this->f);
        LineSeg newR(Rx / Ry * -this->f, -this->f, Rx, Ry);

        if (!current_cell->edges[i]->Neighbor(current_cell)->footprint) {
            Draw_Cell(current_cell->edges[i]->Neighbor(current_cell), newL, newR, view, perspective);
        }
    }
}
```

If the wall is transparent, we redefine our view frustum boundaries to recursively draw cell that is behind the transparent wall and has not been drawn.
When redefining the frustum, we adjust our reference points of view frustum boundary if the whole transparent wall is in the view frustum.

Reference

FLTK Project Maze
https://medium.com/maochinn/fltk-project-maze-338c2109989d

電腦圖學 01-Transformation
https://medium.com/maochinn/%E9%9B%BB%E8%85%A6%E5%9C%96%E5%AD%B8
01-transformation-%E6%96%BD%E5%B7%A5%E4%B8%AD-ea46dedf01f9

台科電腦圖學導論 Project 2: Maze Visibility and Rendering Graphics
https://hackmd.io/@frakw/ByBRenFdP

OpenGL 投影矩阵(Projection Matrix)构造方法
https://zhuanlan.zhihu.com/p/73034007?fbclid=IwAR09yQxkWjLRTi6hgKX3LLuY_lZJI
blK-nccYtJu2QsRaj1Syq6stWd9hHo