Assignment 4: Axis Angle & Orthogonal Matrix & Depth Buffering

Topics

- Build the world to camera Build the model to world matrix for the airplane using the airplane's three axis
- Build the world to camera matrix using the orthogonal method (three types of cameras from assignment 3 need to work)
- Calculate the depth buffer value
- Create the depth buffer array
- Clear the depth buffer array every loop to the far value
- Update the triangle rasterizer so that depth buffer is now supported

Description

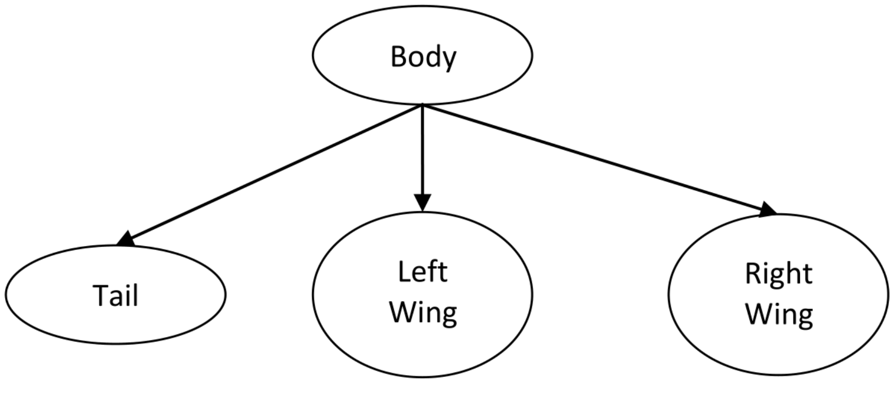
Given an input file "input.txt" containing the following information:

- The camera/projection window left, right, top, bottom, focal, near plane and far plane values (ignore the near and far values for this assignment)
- A vertex list defining a cube centered at the origin and of size 1 (defined in model space)
- A face list holding the indexes to the triangles in the cube
- A face color list holding the colors of the given faces
- A texture coordinates list holding the texture coordinates of all vertexes in the cube (number of texture coordinates is equal to number of faces * 3)

Using the cube vertexes database we will build an airplane having the following body parts:

- Body
- Tail (attached to the body)
- Left and Right Wings (both attached to the body)

The following diagram will show the airplane hierarchy:



The assignment is about building a transformation based on the keyboard input that transforms the airplane from the model space to the world space using the orthogonal. Every time the user pitches, yaws or rolls the airplane, update the airplane's vectors accordingly. Once the vertexes are in the world space, build and use the camera matrix using the inverse orthogonal matrix to transform the vertexes to camera space. Make sure to update the depth buffer z value for every vertex. Next, you need to transform the vertexes to the projection space, then NDC then viewport space and rasterize.

The application should:

1. Read the input file
2. Accept from the keyboard the user input in order to navigate an object (the Tank)
3. Compute the camera position and target.
4. Build the camera transformation according to the user input
5. Allow to swap between cameras
6. Populate the world with a few cubes
7. Compute depth buffer z value
8. Cull in camera space using the naïve culling method (compare z-values farther than near)
9. Project the vertexes
10. Map the vertexes from the projection plane to the view-port
11. Rasterize on the screen using depth buffer

Airplane information

- Body dimensions or scale values: Sx=15,Sy=12.5, Sz=40
- Body Initial Position: Anywhere along the Negative z-axis

- Tail dimensions or scale values: Sx=5, Sy=7.5, Sz=10
- Tail Initial position: (0, 10, -15)

- Left Wing dimensions or scale values: Sx=20, Sy=5, Sz=10
- Left Wing Initial position: (-17.5, 0, 0)

- Right Wing dimensions or scale values: Sx=20, Sy=5, Sz=10
- Right Wing Initial position: (17.5, 0, 0)

Camera

Camera properties should also be extracted from the file. Left/Right give information about the width of the window and same with Top/Bottom for the height. For this assignment *far* and *near* data should be added to the perspective matrix.

When creating the view matrix, make sure that the view vector gets aligned with the **negative Z axis** and that your perspective matrix also takes that into account.

Scene

Along with the airplane keep the grid from Assignment 3. The grid needs to have cubes of scale (10, 10, 10) throughout the XZ-plane. Add a 5x5 grid of cubes from positions (-100, 0,-100) to (100, 0, 100).

Input

- 1: wireframe mode
- 2: solid mode
- 3: first person mode
- 4: third person mode
- 5: rooted camera mode
- a: Rolls airplane body / Rotate around forward (cw)
- d: Rolls airplane body / Rotate around forward (ccw)
- q: Yaws airplane body / Rotate around up (ccw)
- e: Yaws airplane body / Rotate around up (cw)
- w: Pitches airplane body / Rotate around right (cw)
- s: Pitches airplane body / Rotate around right (ccw)
- space: Move airplane body forward
- z: Decrease camera distance
- x: Increase camera distance
- h: Decrease camera height
- y: Increase camera height

Grade Breakdown

Feature	Grade %
Updating the right, up, and forward vectors	30%
Orthogonal view matrix	30%
Depth buffer	20%
Depth interpolating rasterizer	10%
Code quality	10%