Graphics I - Lab 6

Objective

Today’s lab will work on the assumption you have completed Lab 5. We will now introduce the concept of correcting our texturing issue from the previous lab. Since we did not do perspective correctness on our texture coordinates, our texture was affine (parallel) to the triangle it was drawn on. In order to fix this we need to adjust for the perspective on our texture coordinates.

In order to do camera movement, we should use GetAsyncKeyState(..) to grab information from the keyboard. We should also travel based off time, so take that into account when you translate your camera. Make sure to translate **before** you inverse the view matrix, not doing so will produce incorrect values. Note that if you did not account for clipping, if you get too close to the cube or the grid, you might crash your program.

Next will be implementing the Bi-Linear filtering for our textures. As discussed in lecture you should be interpolating 3 times. Keep in mind which pixels to interpolate with.

After implementing Bi-Linear filtering, involve the mip-chain. We will use the mip levels as which “texture” to gather information from. Based on the linear depth of the pixel, we choose a proper mip level, and use this information to filter our texture.

Alongside the previously mentioned you will implement 3D and 2D clipping. Once you have camera movement and try to render triangles behind the near plane, your project will crash. In order to fix the crashing, we must clip the lines and triangles so they reach the near plane before rasterization.

Example of 3D clipping lines will have you calculate a new start / end point for the line. Once you run the vertex shader on the vertex, adjust the line to the near plane. You should make some helper function that guides you on what happened within said function. Was the line completely behind the near plane? Do not draw it. Was the line straddling the near plane into the scene? Adjust the line so the vertex is at the near plane and rasterize that line. Was the line in front of the near plane? Draw line like regular. Before you rasterize the line though you must assure to homogenize the vertex and clip 2 dimensionally (x and y coordinates in NDC only). This means that we will remove homogenization of the vertex from the vertex shader and do the homogenization after 3D clipping. Using linear interpolation is a big clue on how to adjust the line to the near plane.

As for triangles, certain scenarios arise. Were 2 verticies behind the near plane? Just 1? How would be go about adjusting vertices to match the near plane here? For a single triangle, possible outcomes for clipping become:

* Triangle completely in front of the near plane? Draw like normal
* Triangle completely behind the near plane? Do not draw the triangle
* Triangle straddling the near plane and into the scene?
  + 1 vertex behind the near plane? Create 2 triangles and adjust vertices accordingly. Render 2 triangles.
  + 2 vertices behind the near plane? Adjust both vertices and draw the triangle like normal.

The routine for clipping triangles should go:

Run the vertex shader (without homogenization) and clip the triangle along the near plane. You should make a function that indicates which of the 4 outcomes happened when clipping this triangle. (You could turn 3 vertices into 4 vertices here). In this function after you’ve adjusted/created your vertices, homogenize at the very end before sending to your fill triangle routine.

Grade Breakdown

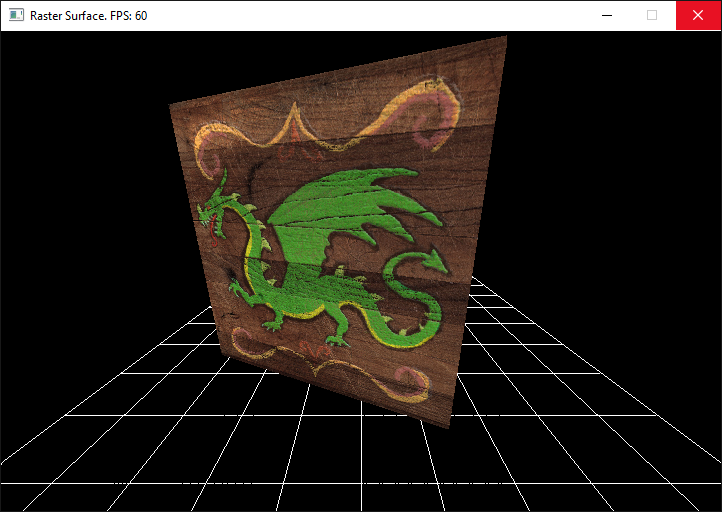
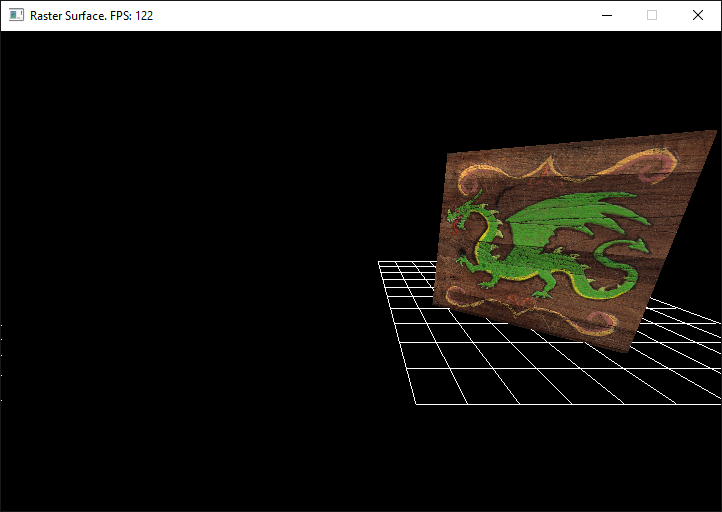
The grading breakdown for this lab will be split up into 4 portions.

* 25% - Camera Movement
  + Keys to move forward, backward, left, right, up, and down
* 50% - Account for perspective distortion on the texture
* 75% - Bilinear Filtering & Mip-mapping with distance check
  + Image shows with mip level 4 to show you what should happen
* 100% - Clipping in 3D and 2D (2D simple version is OK)
  + Wireframe drawing not needed but helps to show what should happen with the triangles

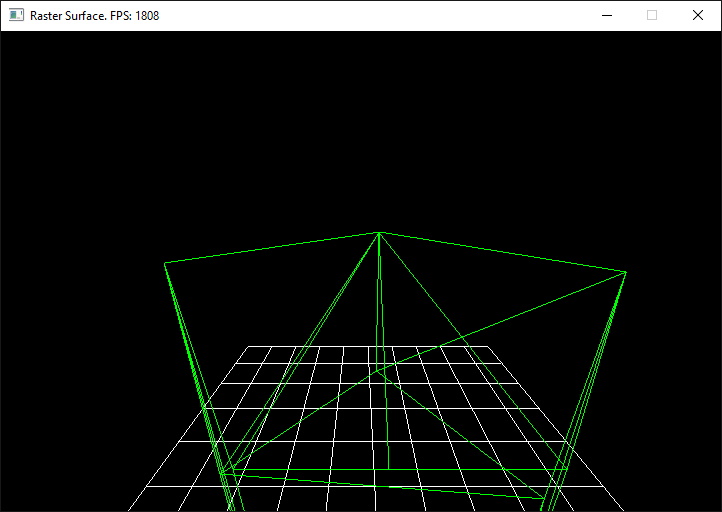
Example Images

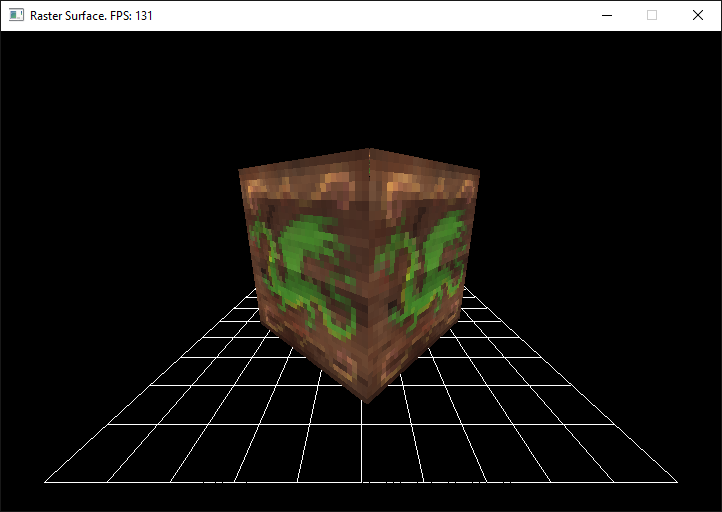
These are example images of what you should be seeing as you progress through today’s lab.

**25% 50%**



**75% 100%**

****



Submission

Labs will be turned in to … in the format of a zip file. The naming convention for the file will be ***Lastname.Firstname.Lab6.zip***. The contents of your .zip file should ***only*** be your main.cpp file. We would like this submission to be as small as possible in order to accommodate space for everyone’s submission. If you fail to submit properly *you will lose points* on your grade.

Prickly Shrubbery

1. Adjust your mip-mapping algorithm to adjust for transitions on the Z axis. This is called a “Trilinear” texturing filter and was discussed in class. Use the space bar to visually toggle between this and the standard “Bilinear” mip-map filter.
2. Wind the vertices for all of the triangles in your cube to be created in a clockwise fashion. Use the cross product to compute a vector perpendicular to the face of the triangle. If the vector is facing toward you draw the triangle. If it isn't, the triangle is facing away from you and you can safely ignore it. This optimization is known as "Back-face Culling".

Frequently Asked Questions

Q.

A.