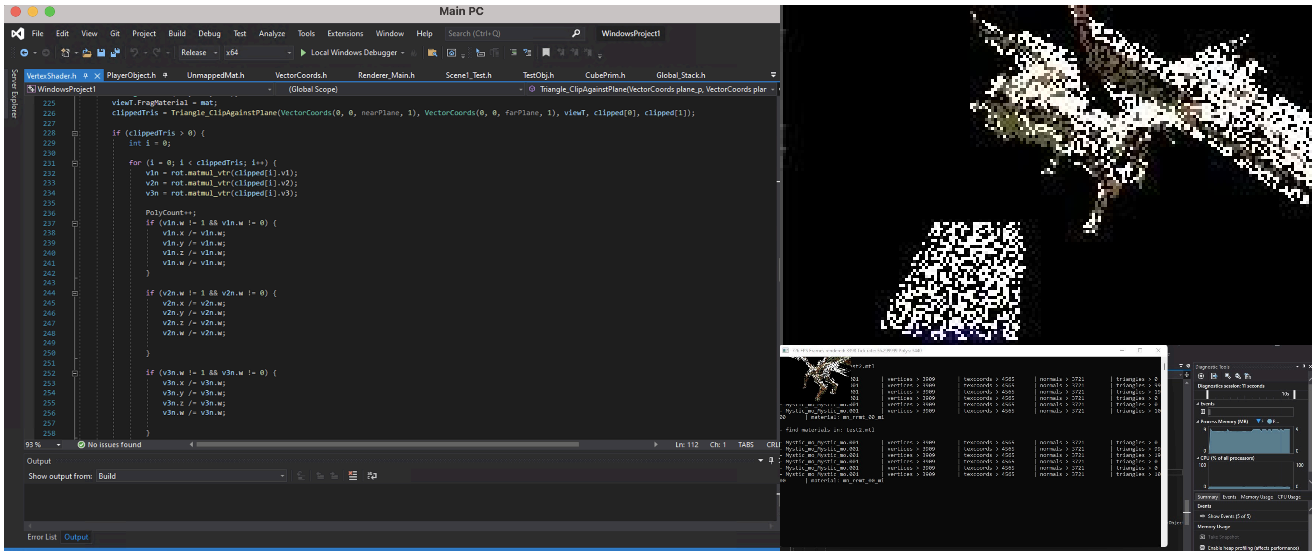
3DRenderer_Cpp

Software+Hardware 3D Renderer/Rasterizer in C++!
https://github.com/kennyzhang620/3DRenderer_Cpp


Main PC
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
WindowsProject1
Release
x64
Local Windows Debugger
VertexShader.h PlayerObject.h UnmappedMat.h VectorCoords.h Renderer_Main.h Scene1_Test.h TestObj.h CubePrim.h Global_Stack.h
WindowsProject1
(Global Scope)
Triangle_ClipAgainstPlane(VectorCoords plane_p, VectorCoords plar
93 %
No issues found
Ln: 112 Ch: 1 TABS CRLF
Output
Show output from: Build
Error List Output
726 FPS Frames rendered: 3398 Tick rate: 36.299999 Polys: 3440

```
viewT.FragMaterial = mat;
clippedTris = Triangle_ClipAgainstPlane(VectorCoords(0, 0, nearPlane, 1), VectorCoords(0, 0, farPlane, 1), viewT, clipped[0], clipped[1]);

if (clippedTris > 0) {
    int i = 0;

    for (i = 0; i < clippedTris; i++) {
        v1n = rot.matmul_vtr(clipped[i].v1);
        v2n = rot.matmul_vtr(clipped[i].v2);
        v3n = rot.matmul_vtr(clipped[i].v3);

        PolyCount++;
        if (v1n.w != 1 && v1n.w != 0) {
            v1n.x /= v1n.w;
            v1n.y /= v1n.w;
            v1n.z /= v1n.w;
            v1n.w /= v1n.w;
        }

        if (v2n.w != 1 && v2n.w != 0) {
            v2n.x /= v2n.w;
            v2n.y /= v2n.w;
            v2n.z /= v2n.w;
            v2n.w /= v2n.w;

        }

        if (v3n.w != 1 && v3n.w != 0) {
            v3n.x /= v3n.w;
            v3n.y /= v3n.w;
            v3n.z /= v3n.w;
            v3n.w /= v3n.w;

        }
```

```
                  est2.mtl

                 001        | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 0
                 001        | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 99
                 001        | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 19
                 001        | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 0
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 0
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 10
00      | material: mn_rrmt_00_mi

- find materials in: test2.mtl

- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 0
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 99
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 19
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 0
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 0
- Mystic_mo_Mystic_mo.001   | vertices > 3909      | texcoords > 4565      | normals > 3721       | triangles > 10
00      | material: mn_rrmt_00_mi
```

Introduction

This game engine is created using C++ with all of the code required for basic 3D rendering and game logic written from scratch. Environments are composed of GameObjects which form a basic structure that is used to build with and allow for encapsulation of the core rendering components which allows for easy deployment of scenes.

Use of OOP (Object Oriented Programming)

In previous iterations of the project, rendering is done through function calls. They set up the model, pass arguments through the linear algebra functions to set up the view and projection matrices and transform them to view space to be displayed on screen. The only concern is that for many objects in scene, we have to do that for every single object, so we use OOP to streamline and modularize the project for larger scenes, inspired by Unity's GameObject and Scene system.

The GameObject

GameObjects form a fundamental unit for all objects in the engine. They contains code for essential components such as the Camera, MeshRenderer, Object Transforms and other relevant code. At its core, it consists of three basic components: An ObjectTransform which represents its position, rotation and scale in three-dimensional euclidean space. Virtual methods for Start() which is called once and Update() for each frame rendered. It can be expanded easily to support more functions such as collision detection through inheritance. Additional GameObject components that can be inherited.

Rendering

Objects are rendered with a five stage graphic pipeline that can be expandable. Objects are broken up into triangles. Objects are transformed from Model Space (The span of points usually between (-1,-1,-1) to (1,1,1) normalized, pre-generated by a 3D modelling application such as Blender or Maya.) to a position in world space specified in the GameObject's Transform through transformation matrices. Those points ae then transformed into the camera's point of view using a view matrix. Finally those points are transformed into normalized screen-space (0,1) using a Projection Matrix and plotted as triangle points to a fragment shader that renders each triangle into frame using barycentric interpolation.

Problems Encountered

One of the central issues encountered near the beginning was getting a functional camera that supported rotation in an easy to use manner. Due to originally using orthogonal projection and transforming objects near the origin (0,0,0), we were able to get objects to move horizontally and vertically, but rotations were unable to be completed. Through breakpoint and step by step debugging, it revealed that we were missing a camera transformation matrix. While I transform all the individual objects from their model space (-1,1,1) -> (1,1,1), I need to move all of those objects near the camera. Crude attempts include transforming all world space vertices by an offset based on the user's input. Unfortunately, while it solved the translating problem, rotation is unresolved. Therefore, from consulting with papers and learning about how camera matrices work, I implemented a camera matrix which allows for proper camera movement. Next, I needed to implement perspective projection to get a realistic 3D render similar to how objects shrink the farther they are from the camera. Performing a z-divide would not work as it would set all z values to 1 and does not consider elements such as camera frustum and the users' field of view. This also makes triangle drawing impossible as depth information is lost which is required to maintain the illusion of proper depth. Hence, a proper projection matrix is needed. Using a 4x4 projection matrix resolved these problems. We would multiply the transformed points in camera space by this matrix to get a normalised set of coordinate points and the z-divide is stored in the w component which allowed dividing by the w value to perform the perspective divide whilst retaining the z component needed for the z-buffer. In the end, proper camera control using the WASD IJKL keys are implemented and perspective projection of 3D scenes is accomplished.