Implementing a Software Rasterizer in C

Thomas Dizon

October 2, 2025

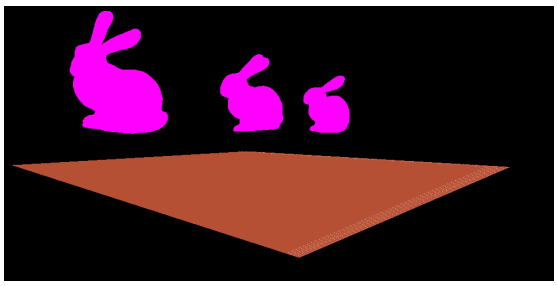

Figure 1:

Contents

1 Introduction

1.1 Motivation

* Learn about the Graphics Pipeline * Help me develop good, optimized, unique graphics in games / game assets * Learn how the GPU works!

1.1.1 What is a *Software Rasterizer*?

* The 'rasterization' step in the graphics render pipline is a specific part which involves converting Primitives into Fragments * A primitive, in my case, is a Triangle and a Fragment is an (x,y) value on the screen with an additional z value to track its depth * In the real world, rasterization is done purely through specialized hardware on the GPU (which is why it is so fast) * To conceptually understand how it works, instead of building a GPU from scratch (hard), I'll use my good old CPU to do the rasterizing (still hard but easier)

1.1.2 Why *C*?

* I first touched C during my Systems Programming course at University. It was extrememly difficult for me at the time but very benefiical for my programming skills. I felt that I haven't extracted all that I can from the language yet. * Also, I wanted to get as low level as possible for this project to ensure I had minimal dependencies. To truly understand the Graphics Pipeline, I wanted to build it truly (well, mostly) from scratch

1.1.3 What's the difference between a Software Rasterizer and a *Real* Rasterizer?

* bleeh blah

1.1.4 What does the project depend on?

* Right now, it depends on the C standard library, std_image.h (reference) for dealing with .png files and SDL2 (reference) for handling all the OS level stuff like inputs and writing to the screen. Though you coud easily replace SDL with any other kind of 2D array.

2 Background

2.1 Math Concepts

2.1.1 Vectors

* In this document, we will use typical Vector operations and notation including but not limited to the following * A vector is **v** is a collection of real numbers denoted by * The *dot product* is defined by * The *cross product* is defined by * A unit vector is denoted by the hat * A plane –

2.1.2 Matrices

2.1.3 Quaternions

* A quaternion is a type of number system blah blah

2.1.4 Transformations

2.1.5 Coordinate Systems

*

2.2 Algorithms

2.2.1 Rasterization

* Takes a Primitive (Triangle made of Vertices) after is has been transformed by MVP and VP. It computes Bounding Box and using BaryCentric Coordinates with respect to the Triangle vertex positions decides which pixels to remove based on whether they are inside or outside of the triangle

2.2.2 Clipping

2.2.3 Anti-Aliasing

3 System Architecture

3.1 Overview

3.1.1 System Diagram

3.1.2 Project Structure

3.1.3 Conventions

3.2 Data Design

3.2.1 Primitives: Triangles and Vertices

3.2.2 Textures and Materials

3.2.3 Framebuffer and Zbuffer

3.2.4 Scene, GameObject, Camera, LightSource

3.3 Software Design

3.3.1 Procedural Programming vs. Object Oriented Programming

* I decided to learn towards a Procedural Programming approach in this project because of the language of choice (C) * This ended up being a limitation because when I wanted to make the system more complex, it was difficult to refactor and extend