On CLI-based Renderers:
In which we investigate the utility of rendering teapots in a command line

Michael Sandler

University of Illinois at Urbana-Champaign

March 16, 2019

Abstract

We investigate the feasibility of producing a renderer within a CLI environment, and the usability of thereof. We also examine the effects that a blissful disregard for good coding practice and performance can have on such a project. A proof of concept is completed in the form of a cube, and extended in order to produce a teapot.

1 Introduction and Motivation

Rendering can often be highly technical and sophisticated, and cutting-edge technologies are often created to speed it up. We propose an alternative, in which we use rendering to actually *set technology back* a few years. In pursuit of this, we wrote a renderer that produces nothing except pictures of teapots. While there are technically controls, the refresh rate is so slow that the rendering could hardly be called realtime, even if we used more sophisticated techniques. For this reason, we also do not pursue fast rendering speeds.

The reason for teapots is simple: It is common knowledge that the presence of teapots has a positive impact on both the mental and academic well-being of students. It is less known, and a worthy research topic, as to the effectiveness of *virtual* teapots. However, as virtual teapots produced in industry-standard renderers are utterly indistinguishable from real teapots, this experiment would not reveal anything interesting. So, in order to investigate this, we wrote a 3d renderer in python that renders a teapot.

2 Implementation

Although the base code was a barebones implementation of a raycaster, we nevertheless encountered a few pitfalls in in implementaton.

1. This project was started less than twelve hours before my analysis final for which I had not studied. This was not a good idea, and I do not recommend that people follow my example.
2. It was shockingly hard to convince people that 2 was a photo of a pixellated cube, and not an amorphous blob.
3. The code was written in python, a language known for its high speed and efficiency.
4. Refusing to use external libraries (potentially with C bindings) may also have contributed to the utter lack of rendering speed.

That said, the technical details of the implementation were as follows:

- Renders were produced under diffuse lighting from a directional source.
- The rendering algorithm was a basic raycasting algorithm, casting rays per-pixel and checking for triangle collision.
- No spatial splitting algorithm was used, as the authors felt incompetent and lazy.

3 Results

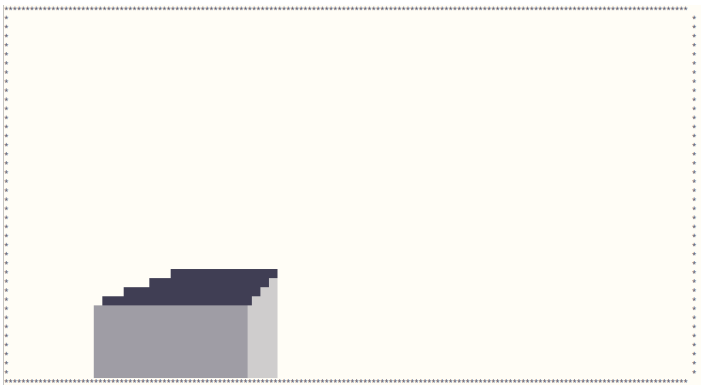

Figure 1: The initial render of a cube.

Surprisingly, figure 2 only took a few milliseconds to render. However, the ncurses overhead alone meant (surprisingly enough) that the cube still had non-negligible refresh rates. Although there were realtime capabilities for camera movement, our guinea pigs remained unconvinced as to the fact that a cube was indeed being rendered, and not some bizarre-looking square contraption.

Frustrated, we plugged in an STL model of the teapot. Due to the issues detailed earlier, this render took half an hour; however, it was far more successful in convincing cynical spectators that our renderer was indeed effective and creating pictures of unphysically lit triangles.

Figure 2: The best-looking teapot render.

A cursor is visible due to a slow and ineffective rendering algorithm.

4 Conclusion

This project led the authors to a single conclusion: Renderers, CLI or not, should not be written in python. A preliminary port to C++ yielded speed improvements of three thousand percent. However, the idea of a working CLI renderer is not wholly ridiculous, and can lead to effective (and amusing) results.

As a direction of further investigation, we intend to bring near-realtime rendering to the command line, and allow command line, modal (in the sense of Vi) editing of models. We hope that a keyboard driven editor with no mouse support will convince others as to the complete uselesness of our work.