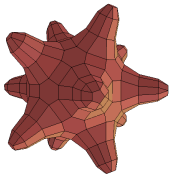

CS430 Computer Graphics

Project 2 – Basic Raycaster

In this project you will write code to raycast mathematical primitives based on a scene input file into a pixel buffer. You will then write the pixel buffer to a PPM formatted file using the code you wrote in Project 1 (P3 or P6 format).

Your program should be resistant to errors and should not segfault or produce undefined behavior. If an error occurs, it should print a message to stderr with "Error:" prefixed to a descriptive error message before returning a non-zero error code. I have a test suite designed to test the robustness of your program.

Your program (raycast) should have this usage pattern:

raycast width height input.json output.ppm

The JSON data file should consist of an array of objects where each object is an object in the scene. Object fields include "type" (with possible values "sphere" and "plane"), "color", "position", "radius", "camera", "width", "height", and "normal". Vectors should be given as an array of three numbers. Consider this example scene with a sphere and a plane:

```
[
 {
  "type": "camera",
  "width": 0.5,
  "height": 0.5
 },
 {
  "type": "sphere",
  "color": [1.0, 0, 0],
  "position": [0, 2, 5],
  "radius": 2
 },
 {
  "type": "plane",
  "color": [0, 0, 1.0],
  "position": [0, 0, 0],
  "normal": [0, 1, 0]
 }
```

```
]
```

You should assume the camera is at position (0, 0, 0) and looking in the positive z direction with y being up. The view plane is forward of the camera by one unit.

The JSON format is described in more detail at http://www.json.org/. But to make parsing simpler we will make these assumptions:

- Our format is always a list of objects; any other input should result in an error.
- The number of objects in the list will not exceed 128; you may allow more at your option.
- The first field in an object will be "type"; you may consider arbitrary order for this field at your option.
- Other fields may appear in any order.
- Vectors will always be given as a list of three numbers.
- You may assume that strings do not have non-ascii characters (e.g. Unicode) or escaped characters.
- You may assume the file itself is ASCII and not Unicode.

Technical Objectives

Technical objectives describe the organizational or code-related features that are a required part of your application and will be evaluated in the technical objective rubric for this project. In grading technical objectives, we will ask the question "How well does this project provide evidence of the objective?" For example, a single Git commit probably does **not** represent outstanding "use of git".

- Ability to read JSON scene files
- Ability to raycast plane primitives
- Ability to raycast sphere primitives
- Ability to write to image data output
- Use of C programming language
- Use of consistent coding style and commenting
- Use of Git

Creative Objectives

Creative objectives mirror the technical objectives but involve subjective creative features of your project. In this project you should create a demonstration scene. For this project the creative objective has very little weight but I still encourage you to consider these objectives:

- Uses a range of colors
- Makes the correct orientation of the scene obvious (which way is up, left, right, etc.)
- Visually interesting

What do I turn in?

In BBLearn you should turn in a report (entered with the BBLearn editor) with the following format:

Project # and Title
Your Name
Your NAU User ID

Link to Github Repository

Your Github repository should contain all the code for your repository. It should also include a README.md file that describes your application, usage, and communicates any special notes to the grader. A Makefile which can be used to build your project. Your program should compile with gcc without any special libraries (libc and libm are ok).

The grading rubric is posted in BBLearn.

Graduate Student Extension (CS599)

If you are a graduate student, you should also implement quadric intersections. Your JSON file should include a type for "quadric" – the parameters of which are given by A, B, C, D, E, F, G, H, I, and J in the equation:

$$F(x, y, z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0$$

The intersection test for quadrics is described here.

You should include example scenes for a cylinders, cone, and ellipsoid in your repository.