

# Rasterizer

Jason Haciepiri, Maria Marinova

**R**asterization is a way of computing a 2D image representation of a 3D scene. It is faster than raytracing, and hence typically used for real-time visualisation. However, it is not as accurate as raytracing.

## 1. Compiling and running the code

The project includes a Makefile, which compiles the code. It contains the `-O3` flag in `CC_Opts` so that the code is compiled in the most optimised way. The `-lX11` flag is added to the linker options `LL_Opts` to allow communication with the X11 Windows Manager. The flag `-fopenmp` is added to the `CC` flags. The last two flags allow for parallelism in the code, which will be elaborated on in Section 3.

The project is compiled and run using the command:

```
make && .\Build\skeleton
```

## 2. Parts 1 and 2

The first aspect implemented was projecting points from 3D to 2D in perspective. As the pinhole camera model is used, the relation is derived using similar triangles; the width and height of the screen are also taken into account.

The camera can be moved via the keyboard – translation and rotation are implemented in the `Update()` function.

The vertices of the triangles which compose the 3D scene are therefore projected to the 2D image. Interpolation is used to draw the edges between them, and hence produce the respective triangles in the 2D image.

A depth buffer is implemented to ensure the 2D representation reflects the position of the objects in the 3D scene.

The illumination can be computed per vertex or per pixel – the former is faster but less accurate. It is computed as a combination of direct and indirect light, and is also affected by the objects reflectance. While in the base case the reflectance is simply the colour of the objects, it can be extended to hold the reflectance properties of the materials.

In order to simulate illumination, a light source which can be moved via the keyboard is added.

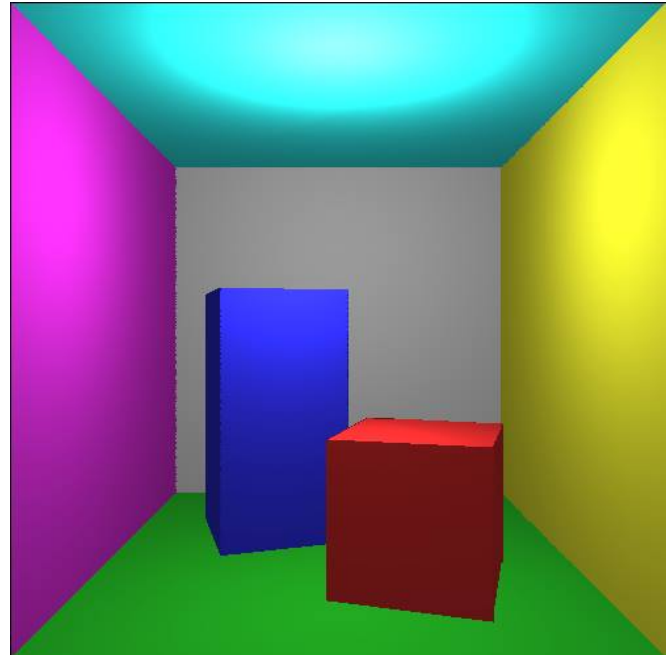


Figure 1: *Rasterizer base*

## 3. Extensions

### Parallelisation via OpenMP

Similarly to the raytracer, the render time was improved by parallelising loops with no data dependencies. This was done with means of OpenMP – `pragma omp parallel for` was used above the main for loops in the `Draw()` function. Until that point, `vec3 reflectance` was a global variable, however, to make the loop parallelisable, it was passed as an argument – `vec3 currentReflectance` – to all relevant functions.

Additionally, the for loop in the `DrawDepthField()` function was also parallelised once completed.

As mentioned above, the `-lX11` is necessary to allow communication with the X11 Windows Manager. The flag `-fopenmp` and the header `#include <X11/Xlib.h>` were also needed to allow usage of OpenMP.