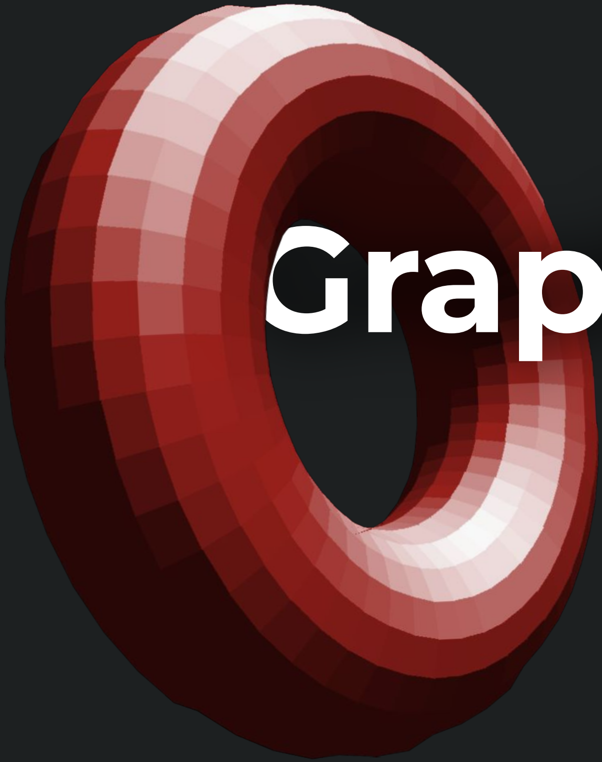
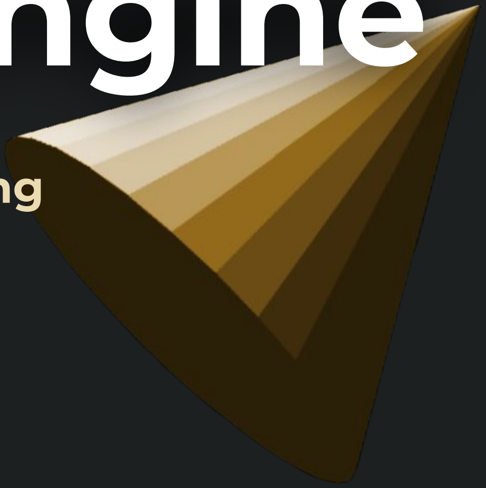


Graphics Engine

For 3D OBJ rendering



Why did I build a graphics engine?

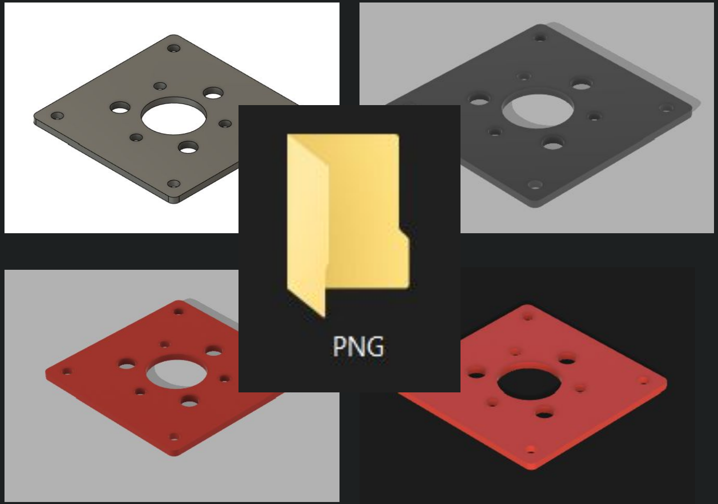
Printables Website

Printables is a community based 3D model sharing platform

I share most of my designs to potentially save others time

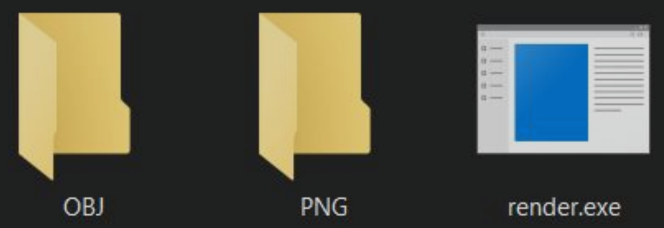
But presenting models in a consistent, good quality took a long time

The Old Process...



- Open OBJ in Fusion 360
- Switch to render mode
- Assign a material
- Adjust the environment
- Adjust the lighting
- Adjust the camera properties
- Wait for render (or spend credits)
- Save image to the right folder

The New Process...

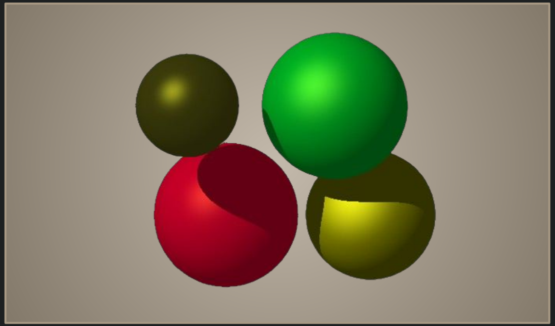


- Drop script into a folder
- Run it

How does it work?

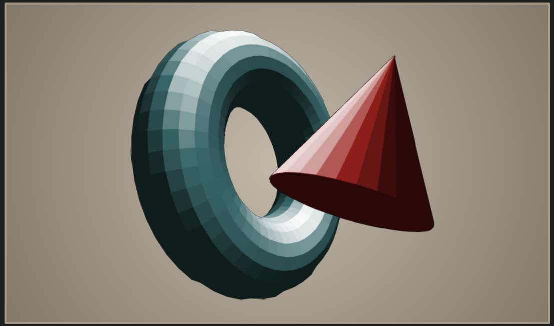
Rendering Method...

I researched multiple rendering algorithms but it came down to the following:



Option 1: Raytracing

Simulates the path of light rays from a camera through a scene for each pixel



Option 2: Rasterization

Plots points on a 2D screen and draws triangles based on the light in the scene

OBJ Files

This is the raw mesh data used to render the object stored as text

Vertices are positions:

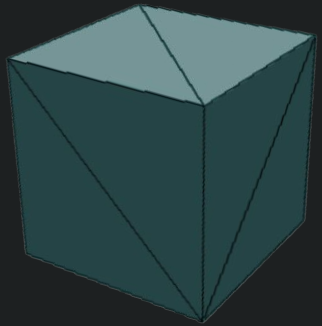
v x_pos y_pos z_pos

```
v -0.5 -0.5 -0.5
v -0.5 0.5 -0.5
v 0.5 0.5 -0.5
v 0.5 -0.5 -0.5
v -0.5 -0.5 0.5
v -0.5 0.5 0.5
v 0.5 0.5 0.5
v 0.5 -0.5 0.5
```

Normals are vectors:

vn x_dir y_dir z_dir

```
vn  1.0  0.0  0.0
vn -1.0  0.0  0.0
vn  0.0  1.0  0.0
vn  0.0 -1.0  0.0
vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
```



Faces are indices:

f vertex / texture / normal

```
f 3//1 7//1 8//1
f 3//1 8//1 4//1
f 1//2 5//2 6//2
f 1//2 6//2 2//2
f 7//3 3//3 2//3
f 7//3 2//3 6//3
f 4//4 8//4 5//4
f 4//4 5//4 1//4
f 8//5 7//5 6//5
f 8//5 6//5 5//5
f 3//6 4//6 1//6
f 3//6 1//6 2//6
```

Transformation matrices

Once the points are defined, transformations can be used to position the object

$$Rxyz = \begin{bmatrix} \cos(Ry)\cos(Rz) & \cos(Ry)\sin(Rz) & -\sin(Ry) & 0 \\ -\cos(Rx)\sin(Rz) + \sin(Rx)\sin(Ry)\cos(Rz) & \cos(Rx)\cos(Rz) + \sin(Rx)\sin(Ry)\sin(Rz) & \sin(Rx)\cos(Ry) & 0 \\ \sin(Rx)\sin(Rz) + \cos(Rx)\sin(Ry)\cos(Rz) & -\sin(Rx)\cos(Rz) + \cos(Rx)\sin(Ry)\sin(Rz) & \cos(Rx)\cos(Ry) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix} \qquad S = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Bavesh Budhkar - Explanation of 3D Transformations

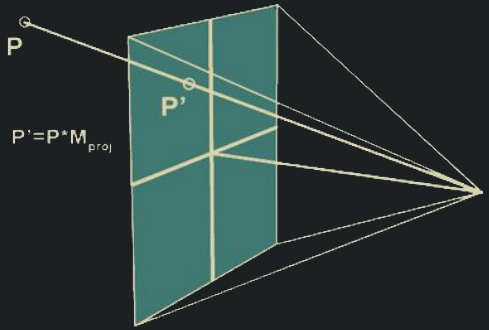
```
/// @brief returns a translation matrix
Matrix *get_translation(double x, double y, double z);

/// @brief returns a rotation matrix
Matrix *get_rotation(double x, double y, double z);

/// @brief returns a scale matrix
Matrix *get_scale(double x, double y, double z);
```

Projection Matrix

Once the object is in the right place, it needs to be projected to 2D screen space



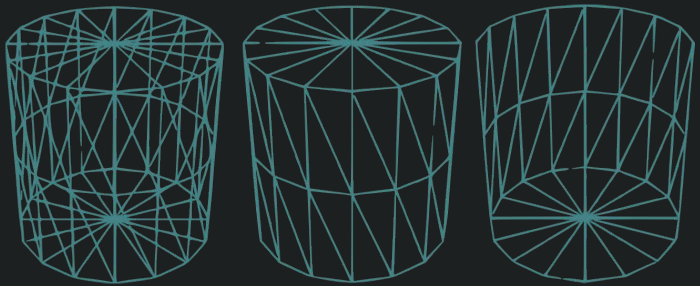
$$\begin{bmatrix} \tan^{-1}(\frac{FOVx}{2}) & 0 & 0 & 0 \\ 0 & \tan^{-1}(\frac{FOVy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{Z_{far} + Z_{near}}{Z_{far} - Z_{near}} & -\frac{2\,(Z_{near}\,Z_{far})}{Z_{far} - Z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

[The Perspective and Orthographic Projection Matrix]

```
/// @brief returns a perspective projection matrix
/// @param width screen width
/// @param height screen height
/// @param fov field of view
/// @param z_near near clipping plane
/// @param z_far far clipping plane
Matrix *get_projection(int width, int height, int fov, int z_near, int z_far);
```

Back-Face Culling

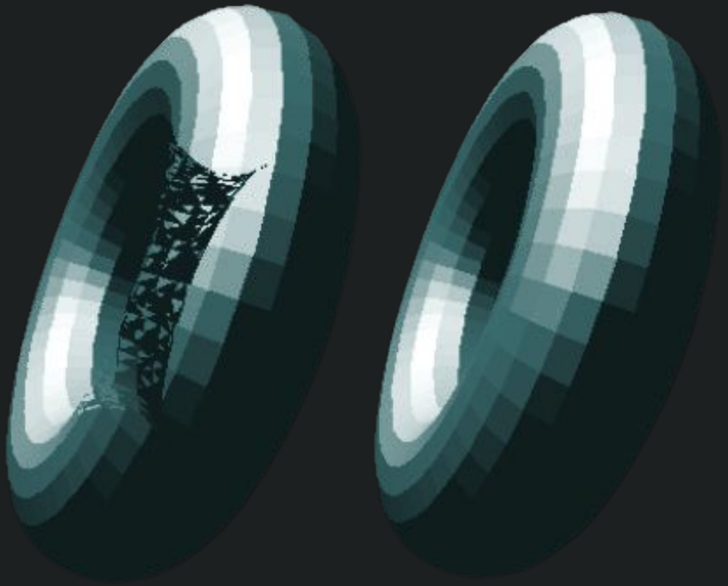
Some triangles will be invisible to the camera, normals are used to check visibility



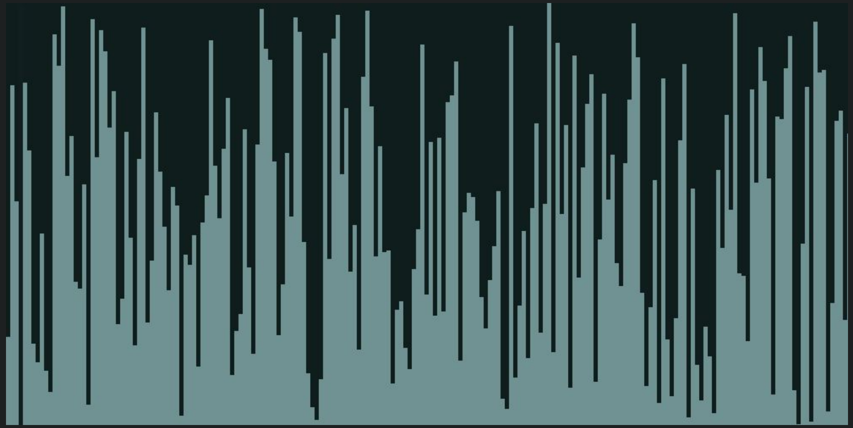

```
/// @brief culls backfaces from a mesh
/// @param m mesh to cull
/// @note apply before projection
void cull_backfaces(Mesh *m);
```

Z-Ordering

Some triangles will be behind others, so they are sorted to render back to front



Before   After

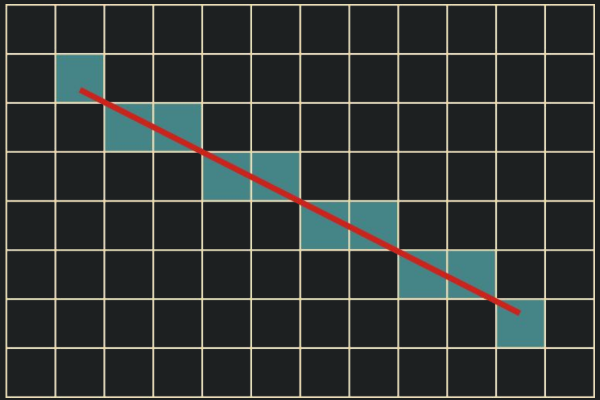


[The Bubble Sort Algorithm]

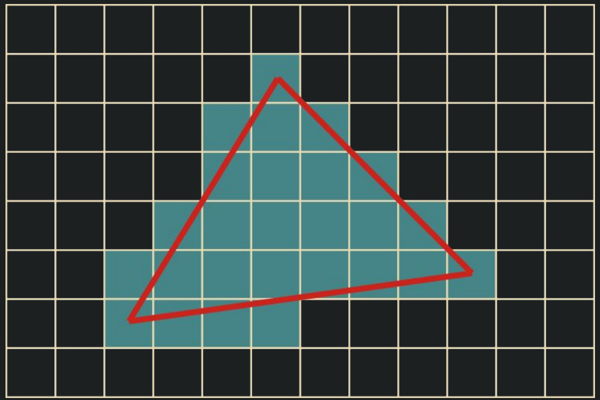
```
/// @brief sorts faces in a mesh by z value (bubble sort)
/// @param m mesh to order
void z_order_tris(Mesh *m);
```

Bresenham's Algorithm + Scanline Rendering

With all of the points in a 2D space, the faces are drawn with these algorithms:



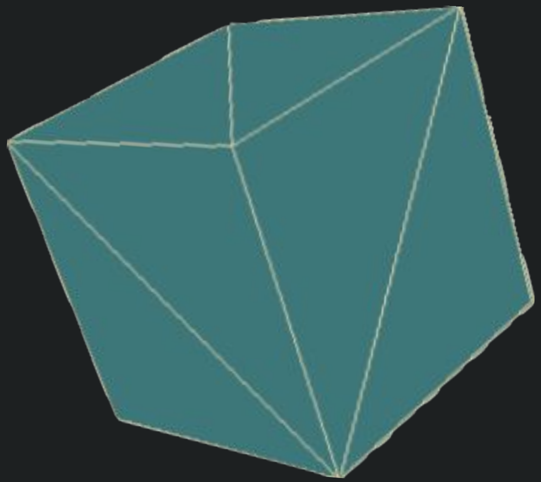
Bresenham’s algorithm draws lines



Scanline algorithm fills between lines

```
/// @brief draws a line on the screen at the specified coordinates with the specified intensity
void draw_line(Matrix *screen, int x1, int y1, int x2, int y2, double intensity);

/// @brief draws a triangle on the screen at the specified coordinates with the specified intensity
void draw_triangle(Matrix *screen, int x1, int y1, int x2, int y2, int x3, int y3, double line_colour, double fill_colour);
```



After all of that, a cube!

Directional Shading

One last thing to add is lighting, which illuminates the faces of the object

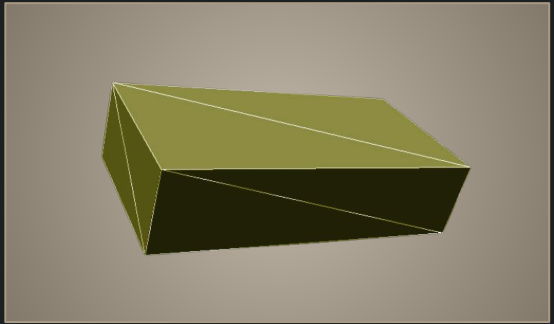
Comparing **normals** to a directional light source simulates lighting

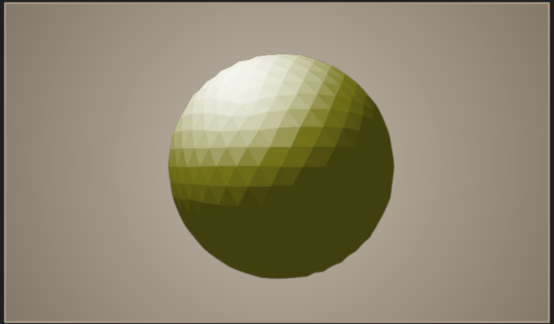
```
/// @brief get the light intensity of a face
/// @param m mesh containing the face
/// @param face_idx index of the face
double get_light_intensity(Mesh *m, int face_idx);
```

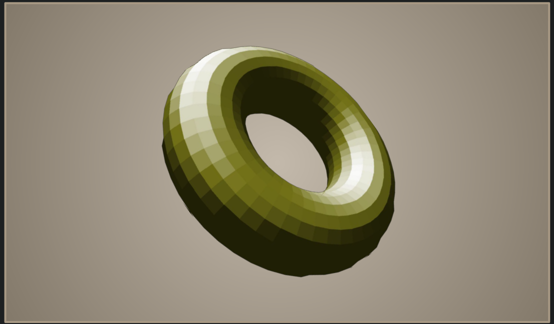
Project Evaluation

Testing Functionality

I tested all the basic features of the graphics engine rigorously







Transformation test:

Can I move, rotate and scale the object?

Passed

Lighting test:

Can I render with directional lighting?

Passed

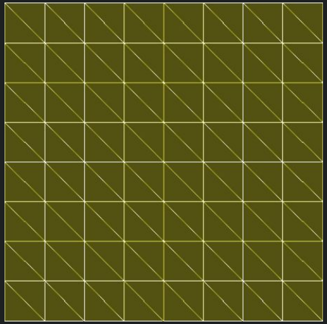
High poly test:

Can the code render a large geometry?

Passed

Testing Performance

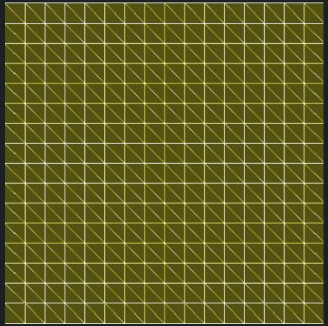
Testing against Fusion 360 showed a significantly quicker time



128 triangles
1000 x 1000 px

Fusion: 7s + setup

Engine: 2.94s



512 triangles
1000 x 1000 px

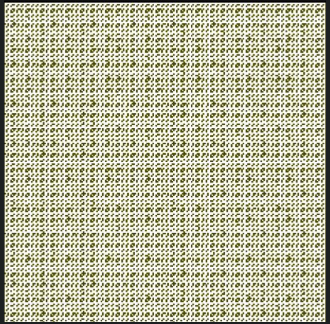
Fusion: 7s + setup

Engine: 2.96s

2048 triangles
1000 x 1000 px

Fusion: 7s + setup

Engine: 2.99



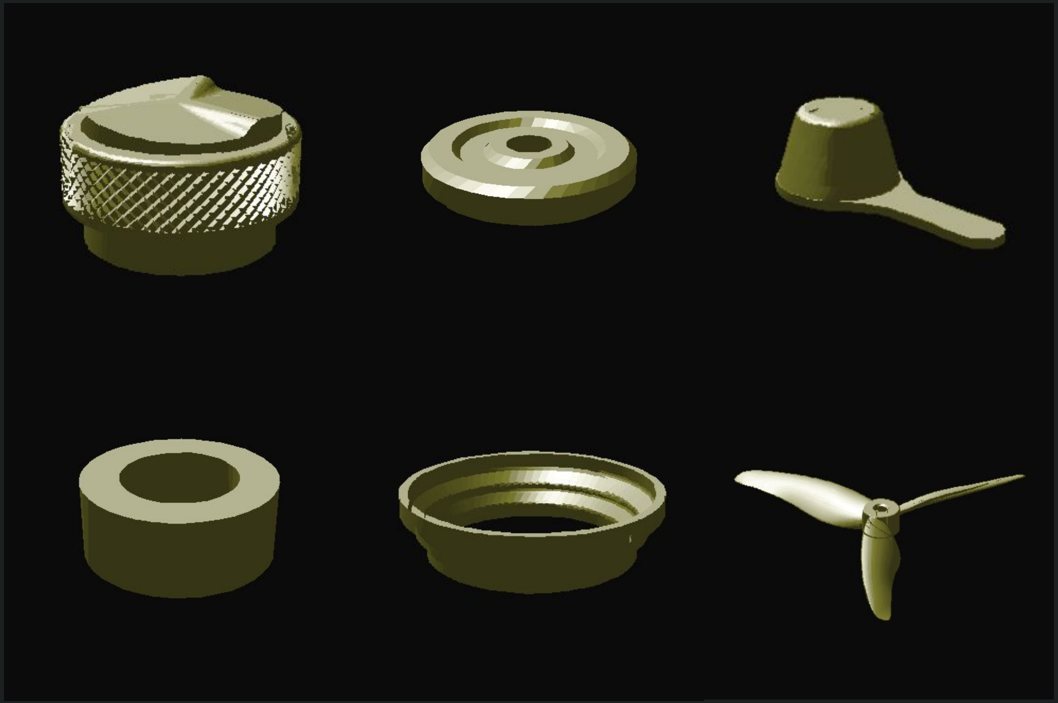
8192 triangles
1000 x 1000 px

Fusion: 7s + setup

Engine: 3.08s

My Printables

I have used this script to render all of my printables in my custom style



Fast rendering and can handle large triangle counts

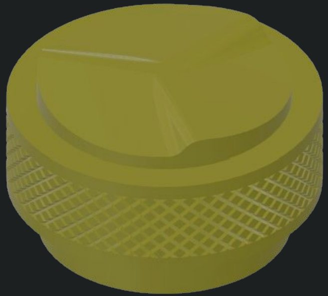
Some artefacts due to rasterization limitations

No reflections or shadows

Not efficient for large meshes

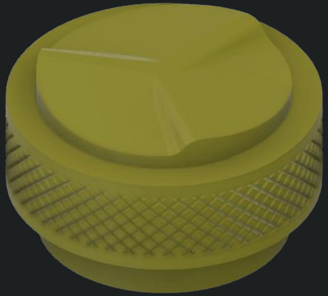
Potential Improvements

There are many improvements I would make on this project, such as:



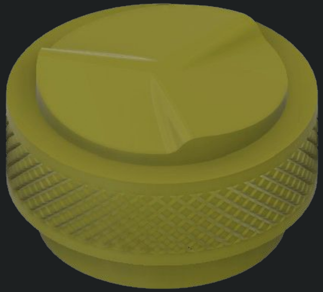
Anti-Aliasing

Blurs edges to reduce pixelation



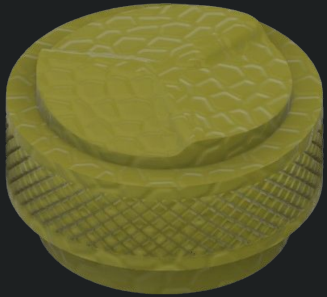
Object Shadows

Let the object cast shadows on itself



Ambient Occlusion

Create shadows in corners and crevices



Texturing

Applying a pattern to the mesh

Open Source Methodology

This project along with most other coding projects are open source on my GitHub

My 3D printing models are all free to download on my Printables

```
/*
 * Graphics Engine
 *
 * File:     render_pgm.c
 * Author:   James Bray
 * Repo:     https://github.com/James-Bray19/Graphics-Engine
 *
 * A simple 3D graphics engine that reads in one or more obj files
 * and renders them as a pgm image.
 *
 * LML repo:        https://github.com/jamesbray03/Lightweight-Matrix-Library
 * 4x4 matrices:    http://www.codinglabs.net/article_world_view_projection_matrix.aspx
 * Bresenham's:     https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
 * Scanline:        https://en.wikipedia.org/wiki/Scanline_rendering
 */
```

Thank you!