The Obra-Engine

Benjamin Ryan

CS 4204 Final Project

What is the Obra-Engine?

Expanded on the graphics engine we created in class by adding:

1. Shadow Mapping
2. Dithering
3. '1-bit' Monochrome Graphics

Inspired by 'Return of the Obra Dinn' by Lucas Pope

Code Additions

`Ditherer` Class

```
@staticmethod
rgb_to_grayscale()

@staticmethod
ordered_dithering()
```

`ShadowMapper` Class

```
self.screen
self.light
self.meshes

render_shadow_map()

barycentric()

debug_shadow_buffer(
)
```

`Image` Class

```
@staticmethod
save_as_bmp

@staticmethod
save_as_gif
```

Code Changes

`Renderer` Class

```
__init__(self,screen,camera,meshes,light,shadow_mapper=None)

render(self, shading, bg_color, ambient_light, dither=None)
```

`PointLight` Class

```
self.camera
```

Shadow Mapping

How I did it

`ShadowMapper` class has its own render loop which returns a buffer to the `Renderer` before it enters its own render loop.

`render_shadow_map` basically uses the extra credit assignment with the "depth" shading

Per pixel, determine if it is `in_shadow` by interpolating the pixel from camera-space back to world-space to the light-space and check the value of `shadow_buffer[x, y]`

Challenges I ran into

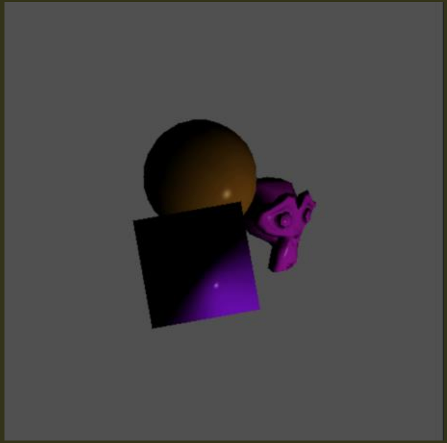
Shadow Acne is nothing to joke about! It is just like acne; annoying and hard to get rid of. Fixes:
1. Instead of culling the back triangles, cull the front triangles
2. Add bias

(Both improved getting rid of most of the shadow acne but normal culling resulted in strange artifacts sometimes)

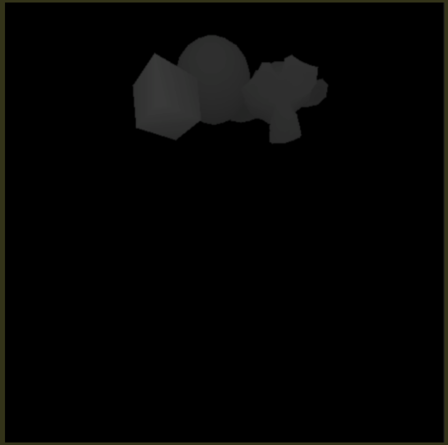
Shadows were clipped by near/far planes due to depth
Fix: I would not include the meshes that would have shadows cast onto them in the list of `meshes` rendered by the `ShadowMapper`

We used Point Lights which do not have direction. In actual shadow mapping with point lights you use a cubemap composed of 6 shadow maps. Doing this or creating a new Light class was out of scope for this project. Fix: Assign a camera to PointLight

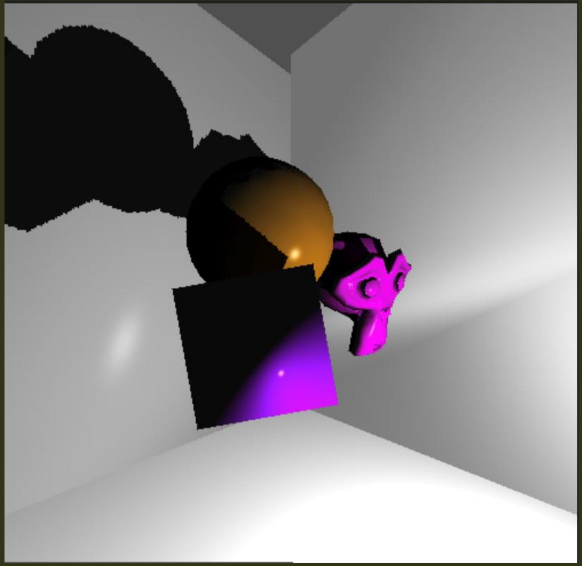
Shadow Mapping



+

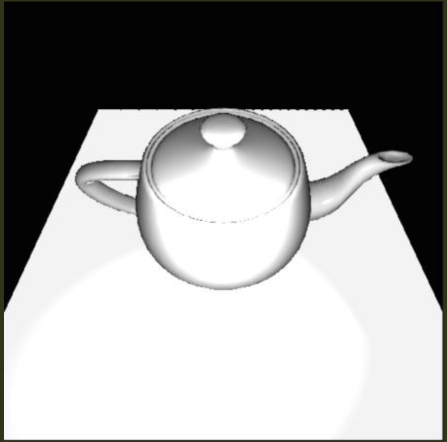


=

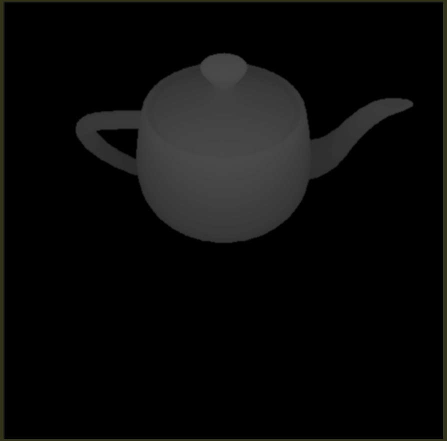


**Note: In order to fully appreciate shadows, I had to create 'background' meshes in Blender to cast onto*

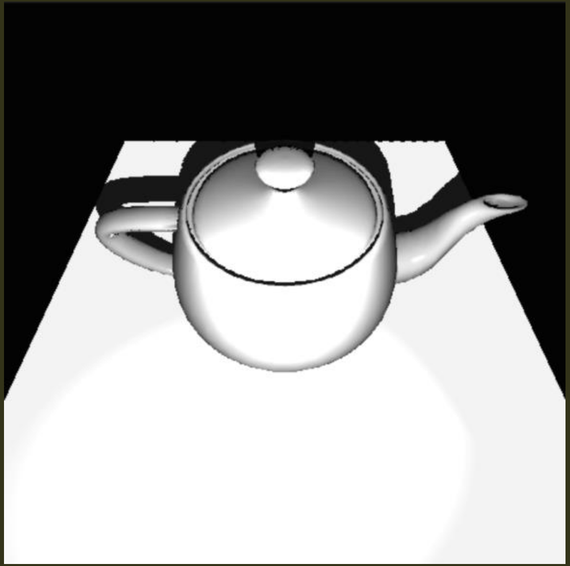
Shadow Mapping



+



=



**Note: In order to fully appreciate shadows, I had to create 'background' meshes in Blender to cast onto*

Dithering

Dithering is a practice where you intentionally apply noise to images
There are many different dithering methods. I was between implementing Blue Noise or Ordered dithering since both are used in ‘Return of the Obra Dinn’.

Blue Noise - random noise with higher frequencies having higher intensities

Ordered Dithering - a set structured threshold map of values

I opted for the latter method and used **Bayer Matrices** since it would be simple to implement in our pipeline since it is just another matrix multiplication to do.

Ordered Dithering

The most common methods of Ordered Dithering is using a 'dither map' or a Bayer Matrix.

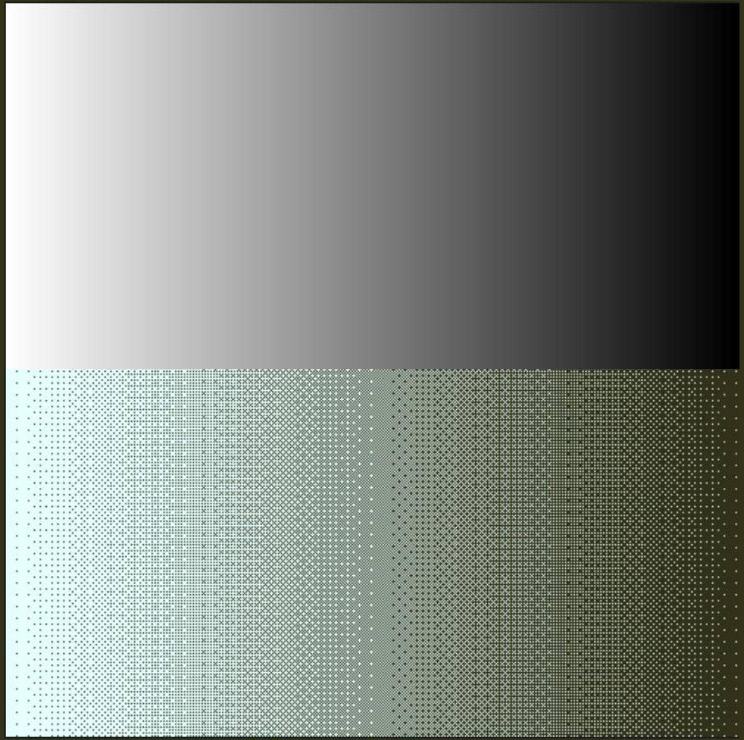
Algorithm:

Convert rgb buffer into grayscale of their lumininance

```
grayscale buffer = rgb_buffer * [0.299, 0.587, 0.114]
```

Determine if the luminance/grayscale value is above or below Bayer Matrix threshold

```
bayer_matrix = np.array([
   [0,  32, 8,  40, 2,  34, 10, 42],
   [48, 16, 56, 24, 50, 18, 58, 26],
   [12, 44, 4,  36, 14, 46, 6,  38],
   [60, 28, 52, 20, 62, 30, 54, 22],
   [3,  35, 11, 43, 1,  33, 9,  41],
   [51, 19, 59, 27, 49, 17, 57, 25],
   [15, 47, 7,  39, 13, 45, 5,  37],
   [63, 31, 55, 23, 61, 29, 53, 21]
]) / 64.0
```



Ordered Dithering

The most common methods of Ordered Dithering is using a ‘dither map’ or a Bayer Matrix.

Algorithm:

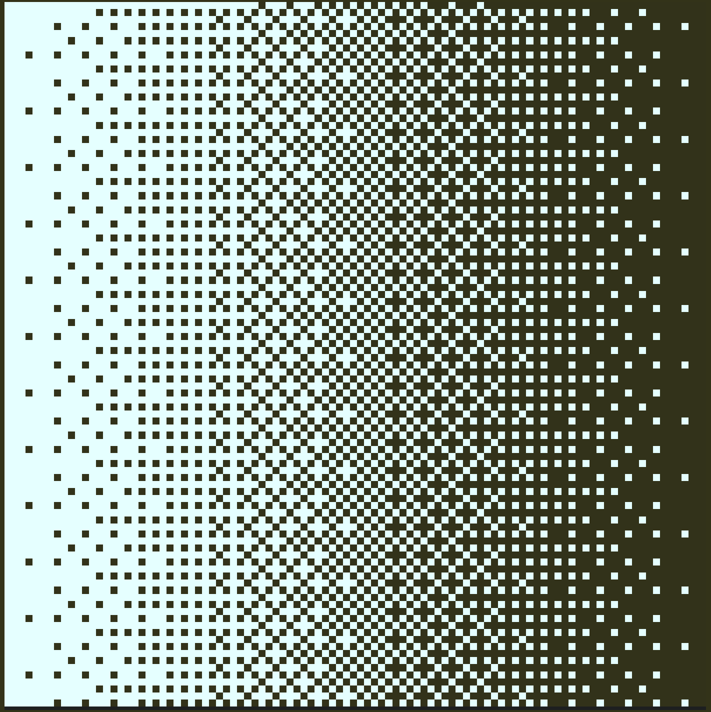
Convert rgb buffer into grayscale of their lumininance

```
grayscale buffer = rgb_buffer * [0.299, 0.587, 0.114]
```

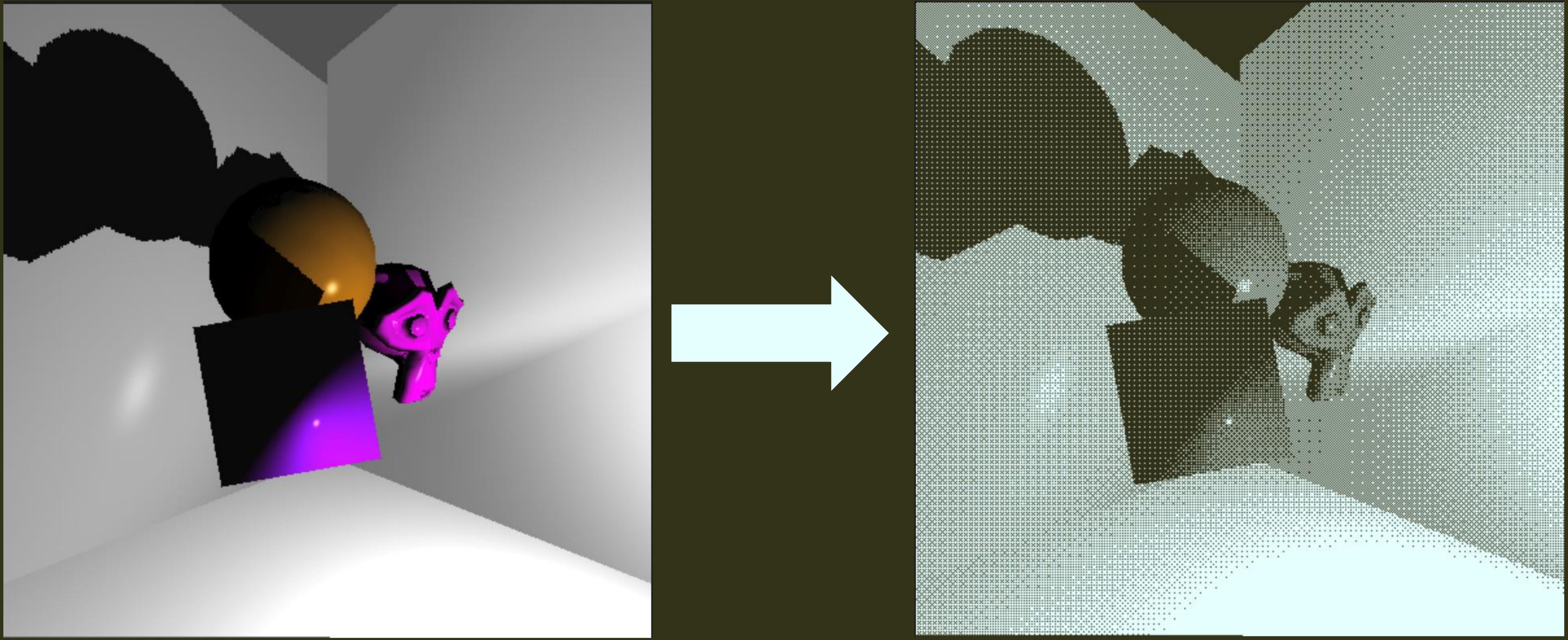
Determine if the luminance/grayscale value is above or below Bayer Matrix threshold

```
bayer_matrix = np.array([
    [0,  32, 8,  40, 2,  34, 10, 42],
    [48, 16, 56, 24, 50, 18, 58, 26],
    [12, 44, 4,  36, 14, 46, 6,  38],
    [60, 28, 52, 20, 62, 30, 54, 22],
    [3,  35, 11, 43, 1,  33, 9,  41],
    [51, 19, 59, 27, 49, 17, 57, 25],
    [15, 47, 7,  39, 13, 45, 5,  37],
    [63, 31, 55, 23, 61, 29, 53, 21]
]) / 64.0
```

It really is 1-bit!



Ordered Dithering



Ordered Dithering

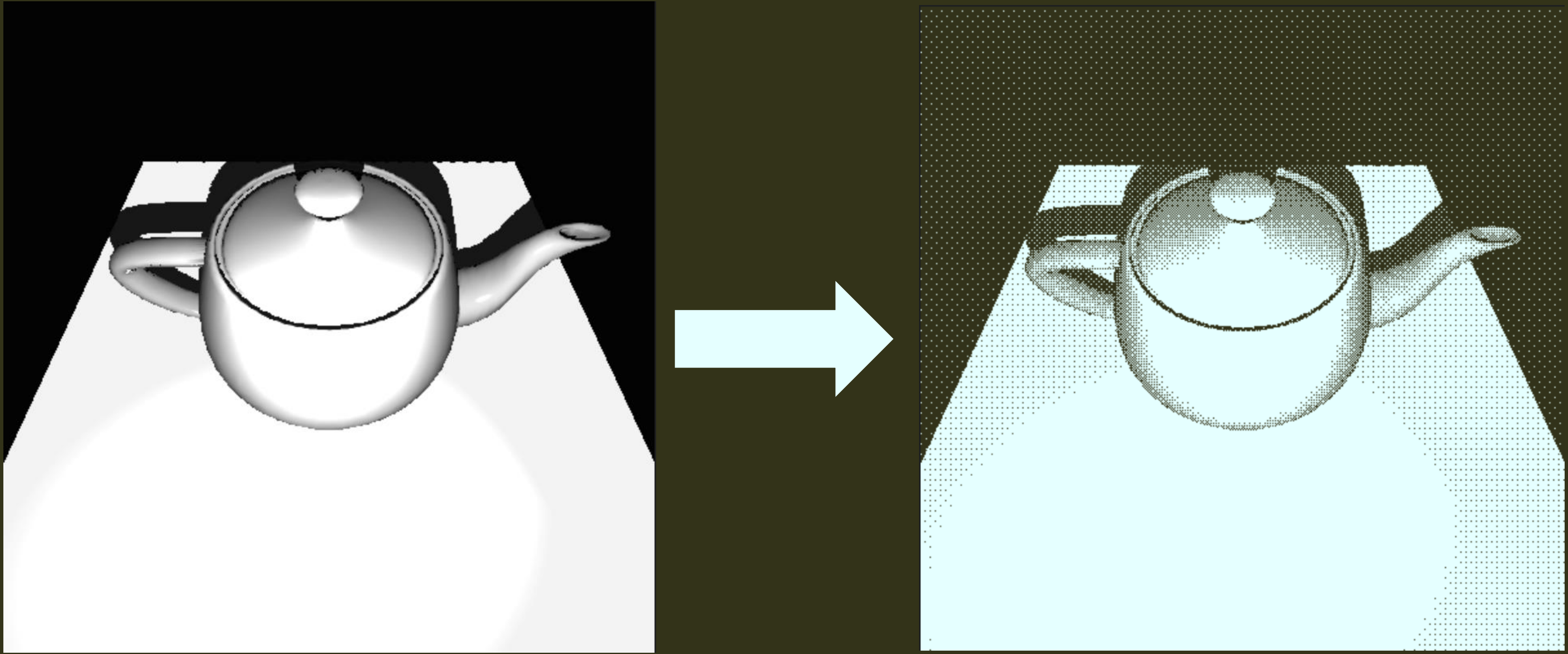


Image Processing

I wrote an `Image` class which transforms a numpy array into a bitmap (.bmp) file

```
def save_as_bmp(filename, array)
```

because I experienced a strange glitch where Pygame would “flicker” with the binary images and the saved photos would not actually be monochromatic.

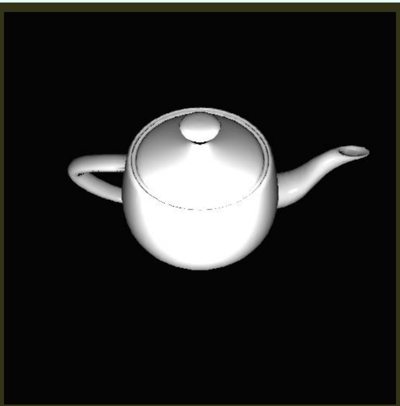
I also wrote a function which transforms a list of bitmap file names into a GIF (.gif) file

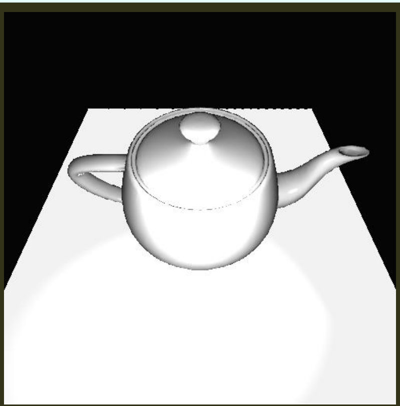
```
def save_as_gif(bmp_files, filename, duration)
```

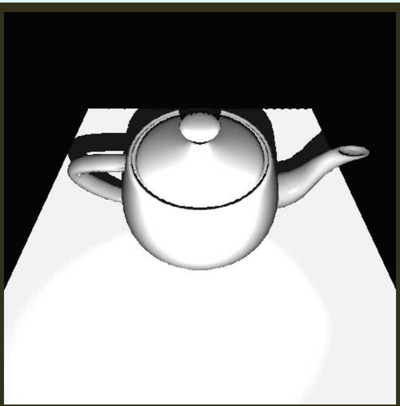
This allows me to make super cool “animations” showcasing the full extent of the dithering and shadow mapping in the Obra Engine.

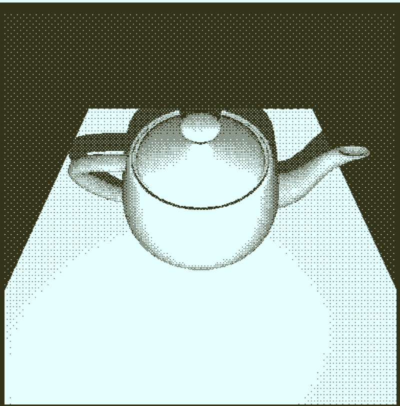
Performance

Teapot

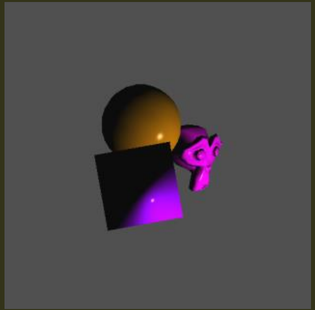

5.090437 seconds

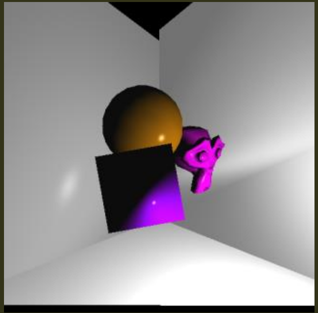

13.480632 seconds

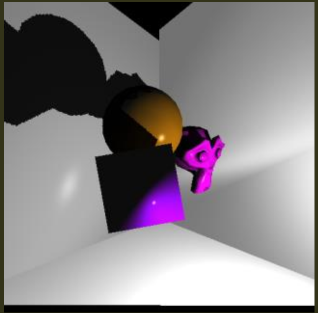

21.306721 seconds

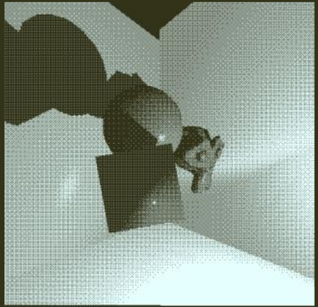

21.427394 seconds

Depth


2.205572 seconds


19.510748 seconds


28.480489 seconds


28.612639 seconds

Grand Reveal