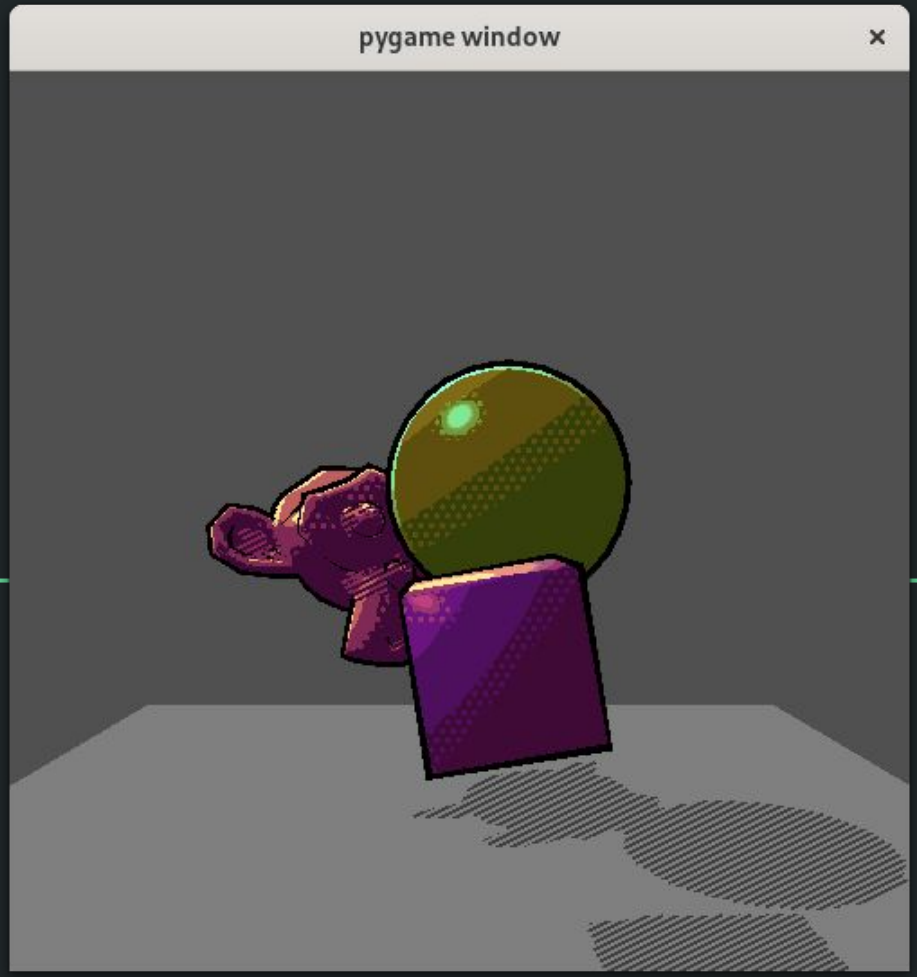
Stylized Rendering and Shadows

By Robin and Zach



Shadow Maps

- Directional Light Implementation
- Bias: Add constant value to depth of pixel we are checking
- Challenge: Perspective correction converting pixel back to world space

Generate shadow depth map

↓

Screen pixel

↓

World space point

↓

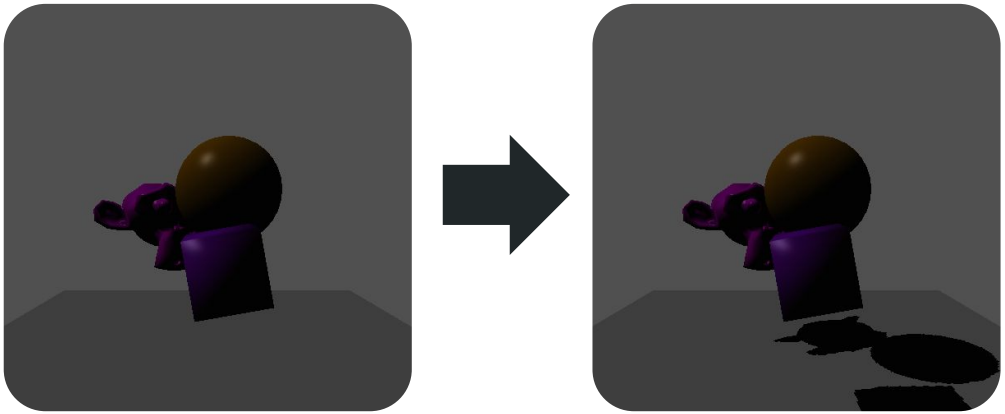
Project point to shadow map camera

↓

Use device_to_screen to get x, y

↓

Compare with shadow depth map[x][y]



Shadow Map Performance

Average Run Time Three Trials

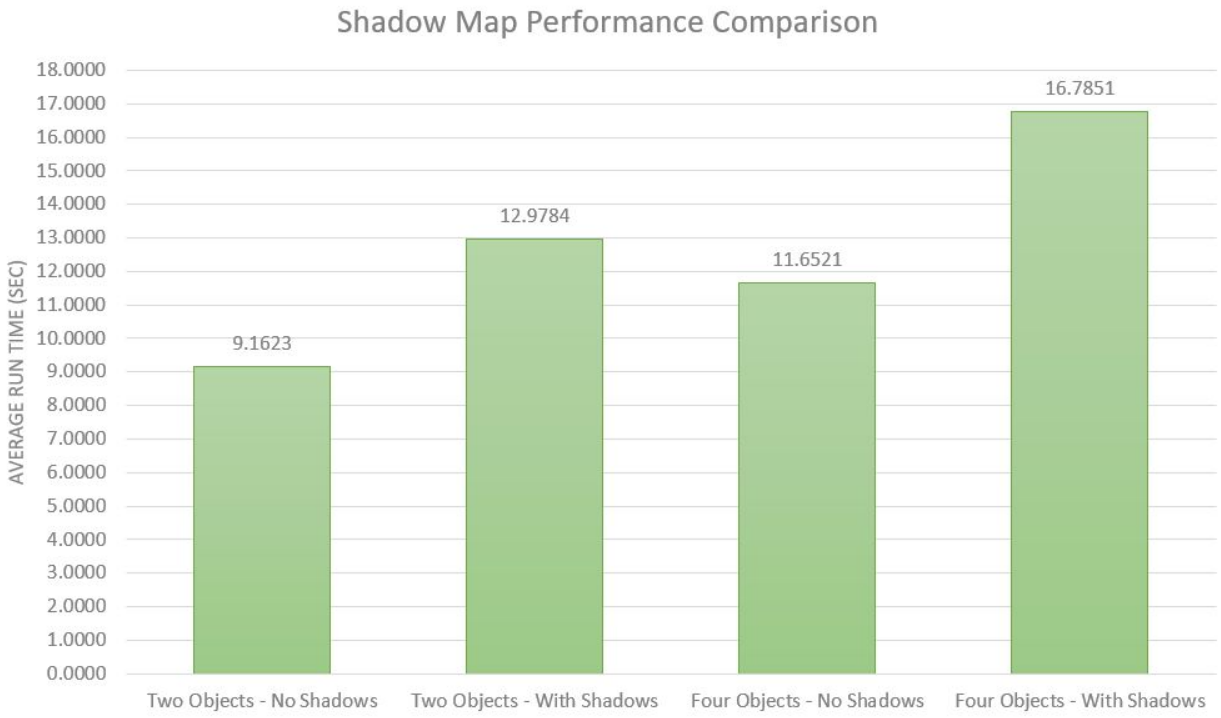
Shading Style: Blinn-Phong

Two Objects

- Suzanne, Plane
- 42% increase run time
- 3.8161 sec run time diff

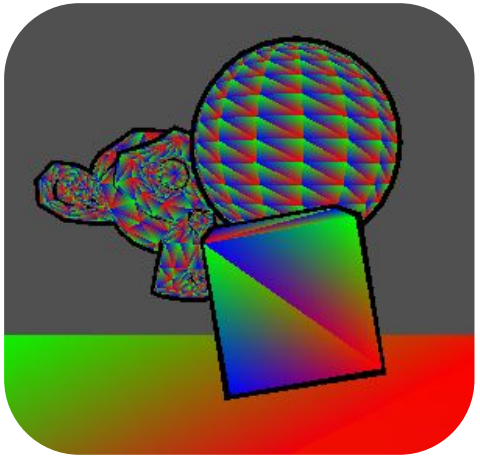
Four Objects

- Suzanne, Cube, Sphere, Plane
- 44% increase run time
- 5.1330 sec run time diff



Outlines

- Inverted Hull Method
  - Duplicate the mesh, add vertex normal multiplied by outline size to vertex positions, and flip the normals
  - Only works on manifold meshes



Outline Performance

Average Run Time Three Trials

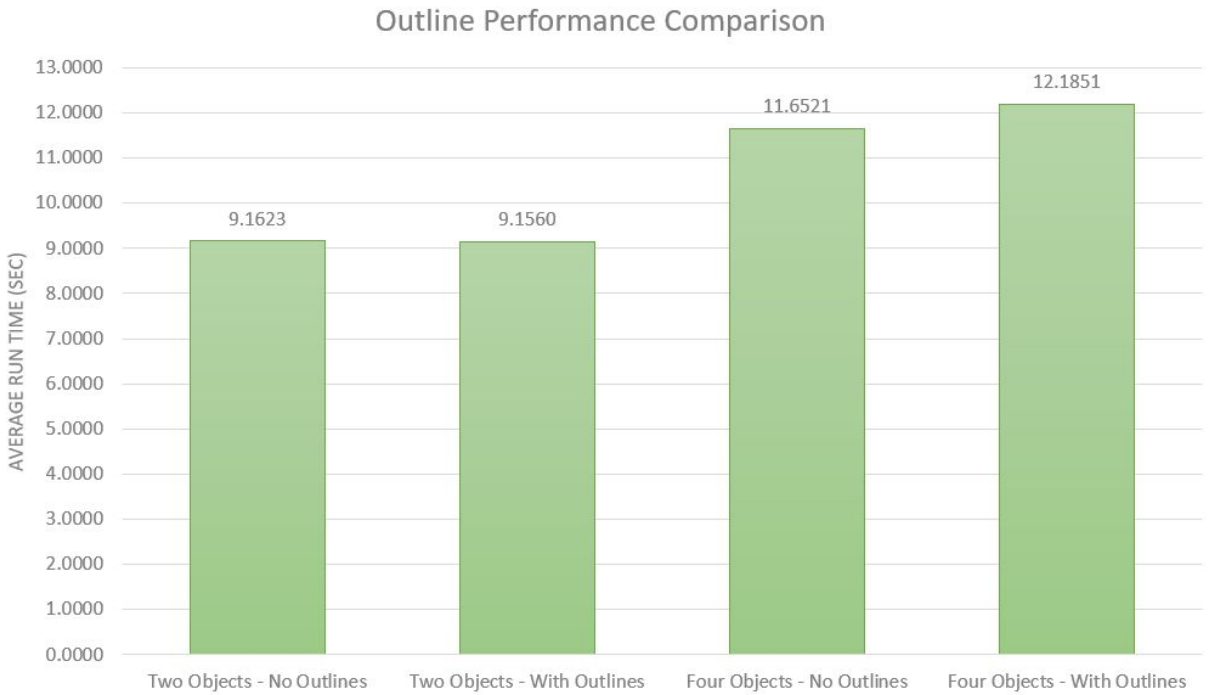
Shading Style: Blinn-Phong

Two Objects

- Suzanne, Plane
- 0.06% decrease run time?
- 0.0063 sec run time diff

Four Objects

- Suzanne, Cube, Sphere, Plane
- 5% increase run time
- 0.5330 sec run time diff



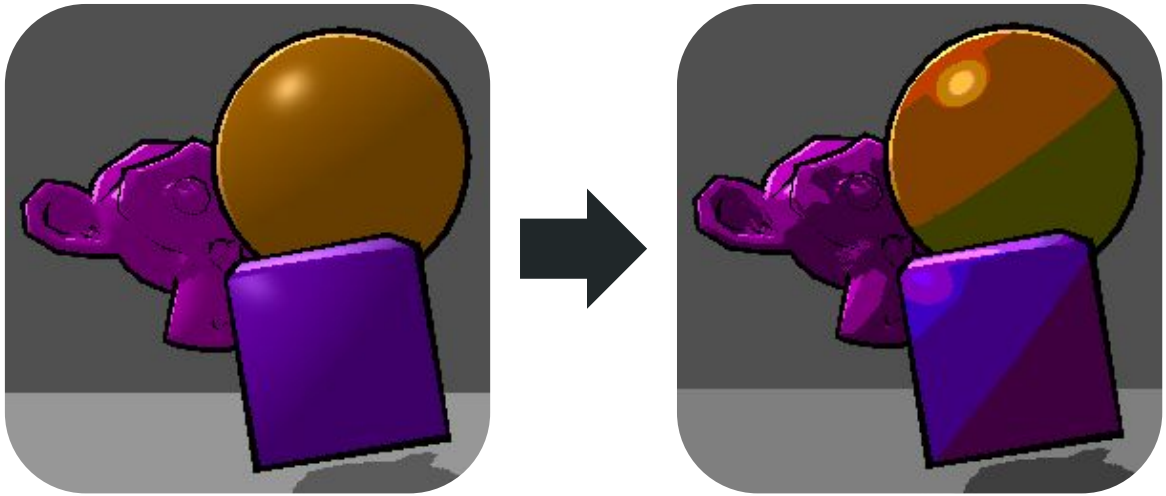
Render Pass System

- Allows multiple shading styles in one scene, including stylized, blinn-phong, outlines, etc.

```
# Multiple render pass example
blinn_phong_pass = Pass([mesh_1], “phong-blinn”)
barycentric_pass = Pass([mesh_2], “barycentric”)
depth_pass = Pass([mesh_3, mesh_4], “depth”)
Outline_pass = OutlinePass([mesh_1, mesh_3], [[1, 0.5, 0.4], [0, 0.5, 0]],
[0.07, 0.02])
renderer.render([blinn_phong_pass, barycentric_pass, depth_pass, outline_pass],
[80, 80, 80], [0.2, 0.2, 0.2])
```

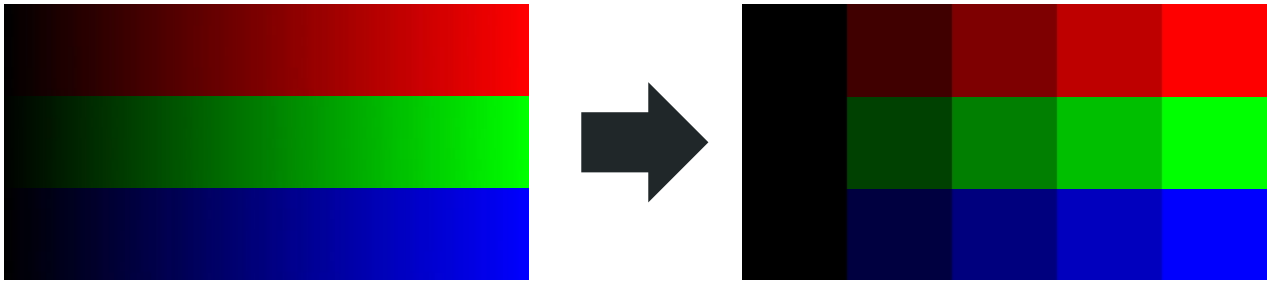
Cel shading

- Key idea: Quantize values to a discrete set of values with a step function



Cel shading

```
# Quantize a color to a more limited palette
def quantize_color(target: np.ndarray):
    bucket_count = 5
    return 255 / (bucket_count - 1)
        * (target * bucket_count // 256)
```



125 colors
(what is this, 1985?)

Cel shading, but with flavor

- Color palettes aren't always a linear scale of one shade
- Distort the colors as they get brighter?

```
# The brighter a color, the more it will be distorted.
def skew_color(target: np.ndarray, amount):
    s = (target[0] + target[1] + target[2]) / (255 * 3) * amount
    return np.array([target[2] * s + target[0] * (1 - s), target[0] * s +
target[1] * (1 - s), target[1] * s + target[2] * (1 - s)])
```

Stippling!

- Shade in-between areas with dots, lines, etc.
- First idea: Change the step function to return two closest colors

Stippling!

- Second idea: Start removing components when they’re below a threshold
- Creates stippled areas where each component fades out

Stippling!

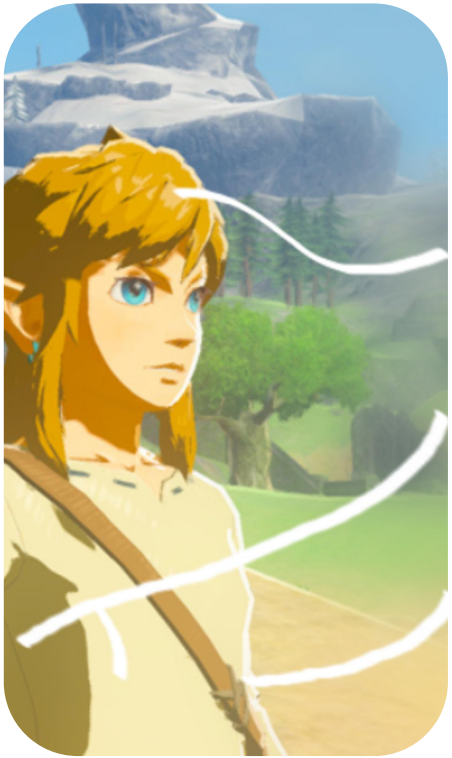
- Procedural textures in NDC

```
def checker(x, y, size):
    return ((x // size) % 2 == 0) ^ ((y // size) % 2 == 0)

def _dots(x, y, size):
    period_x = size * 2
    period_y = period_x * math.sqrt(3)
    cell_x = x % period_x
    cell_y = y % period_y
    radius = size / math.sqrt(math.pi)
    return (cell_x - size) ** 2 + (cell_y - size) ** 2 <=
radius ** 2

def dots(x, y, size):
    return _dots(x, y, size) or _dots(x + size, y + size *
math.sqrt(3), size)

def lines(x, y, size):
    return (x - 2 * y) % (size * 2) >= size
```

Rim lighting

- Kind of magic
- If sufficiently close to the rim, add some specular light

```
rim = 1 - max(Vector3.dot(v, n), 0)
rim = 0.5 if rim > 0.8 else 0
r = Vector3.mul(i_s, rim * lit * unoccluded)
```



Final result

LED Support

- Supports Neopixel LED matrix running on Raspberry Pi
- Challenges
  - Many, many import errors
  - Odd rows left to right, even rows right to left

```
# Account for led wiring
row = i // self.height
column = (self.width - 1 - (i % self.width)) if
(i // self.width) % 2 == 1 else (i % self.width)
```