

# Shadow Mapping

A Computer Graphics Presentation by Aditya

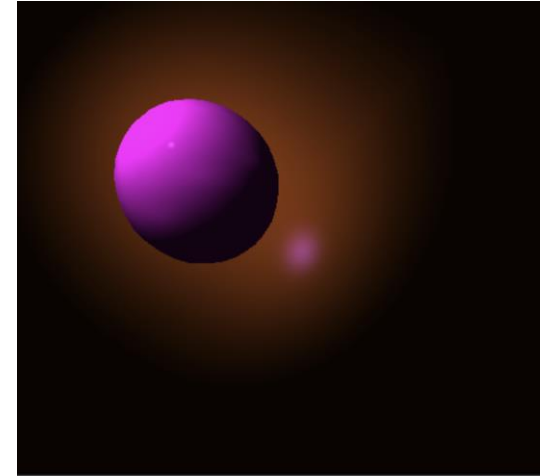
# What is shadow mapping?

A way to implement shadows into a computer graphics rendering pipeline

# What is shadow mapping?

A way to implement shadows into a computer graphics rendering pipeline

Without Shadow

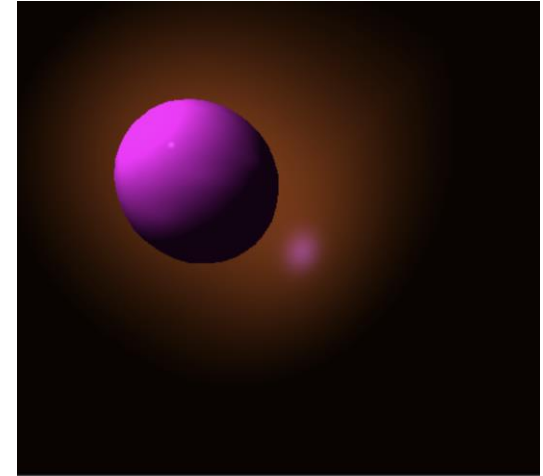


# What is shadow mapping?

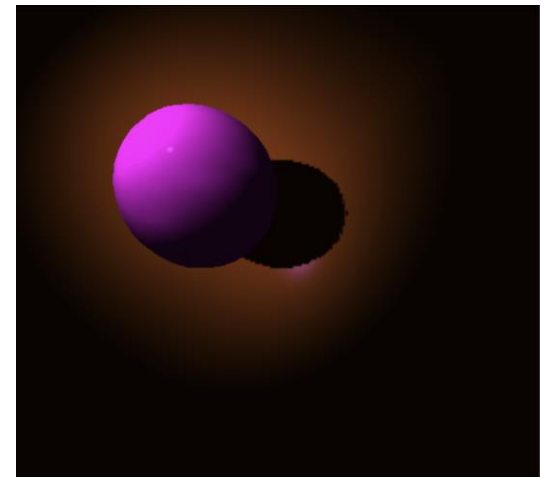
A way to implement shadows into a computer graphics rendering pipeline

Checking if a pixel is visible from the light source, and lighting the pixel based on that

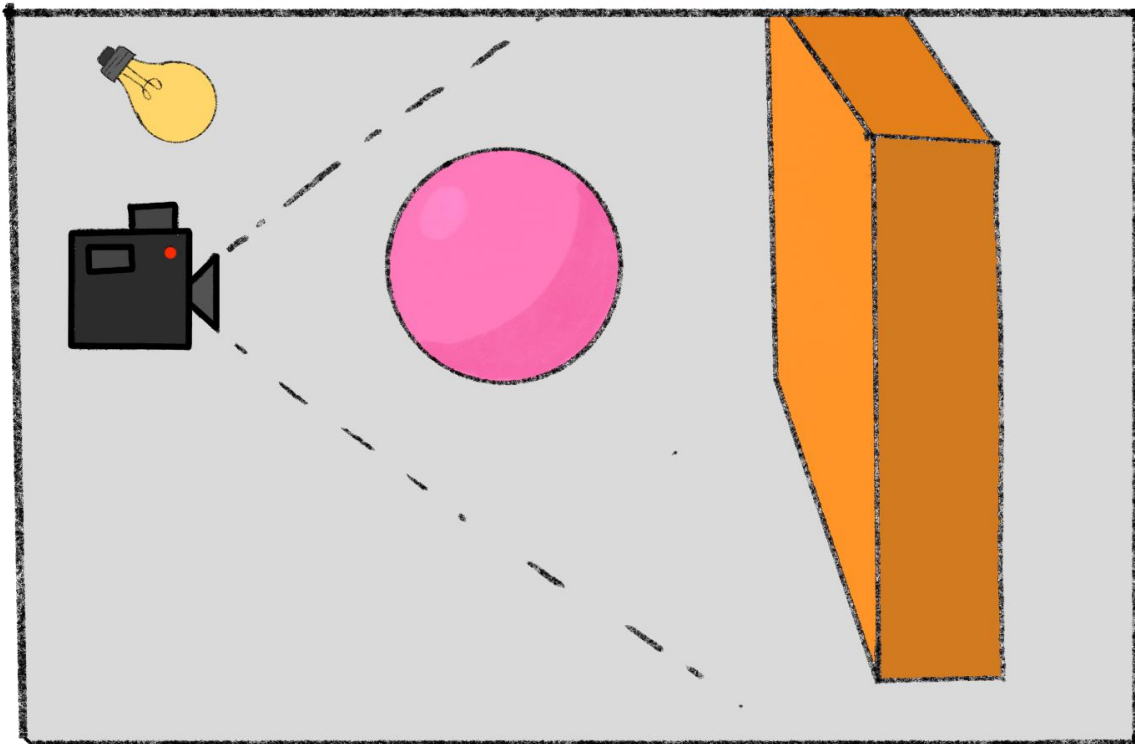
Without Shadow



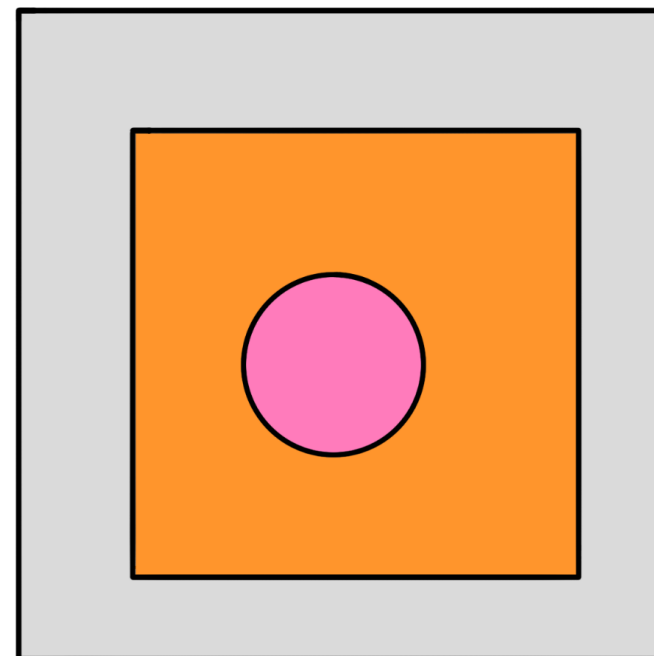
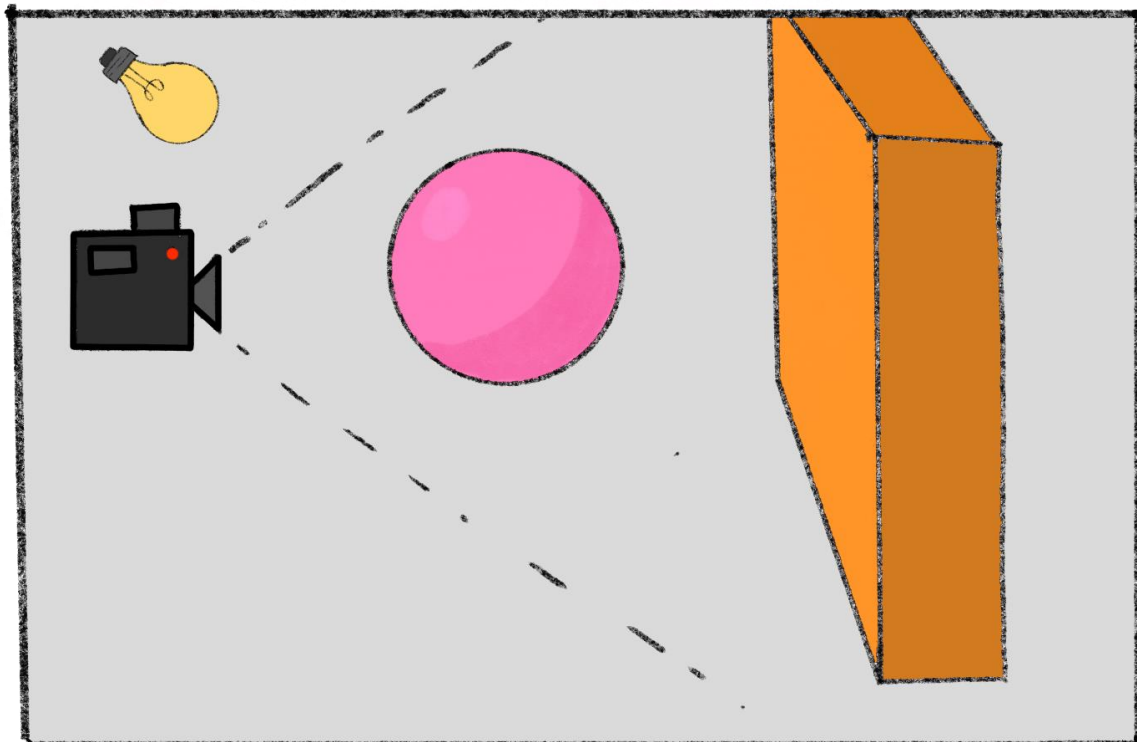
With Shadow



# How It Works

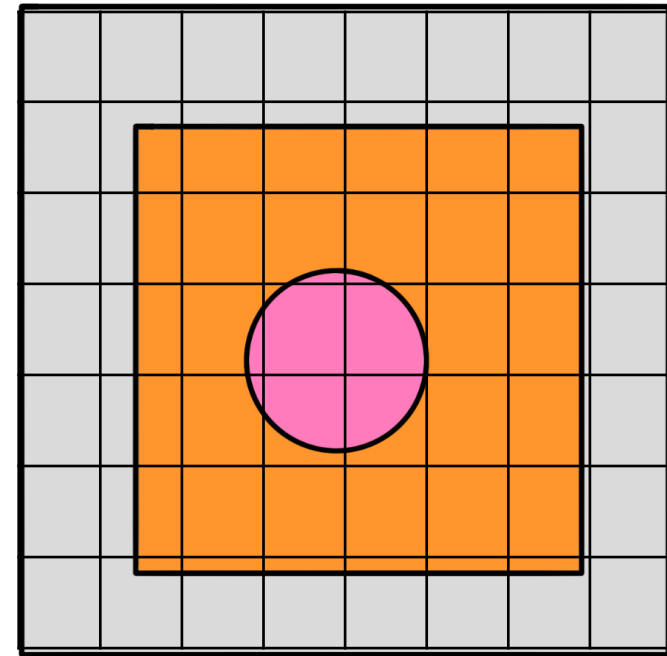
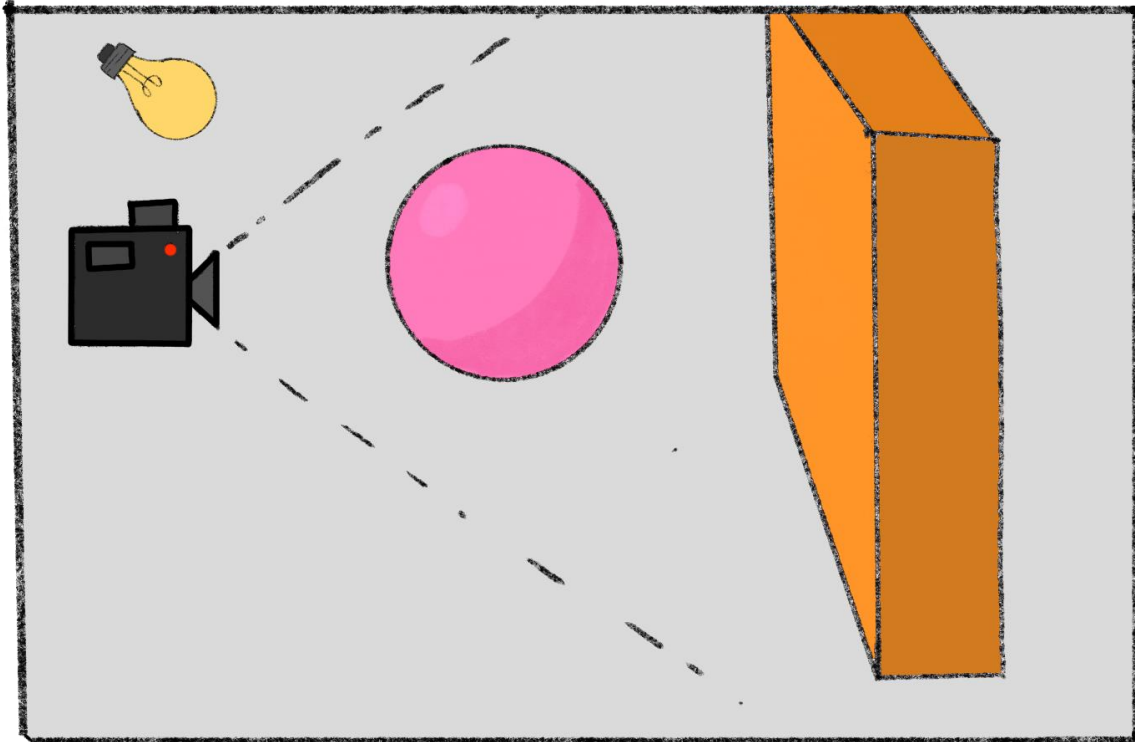


# How It Works



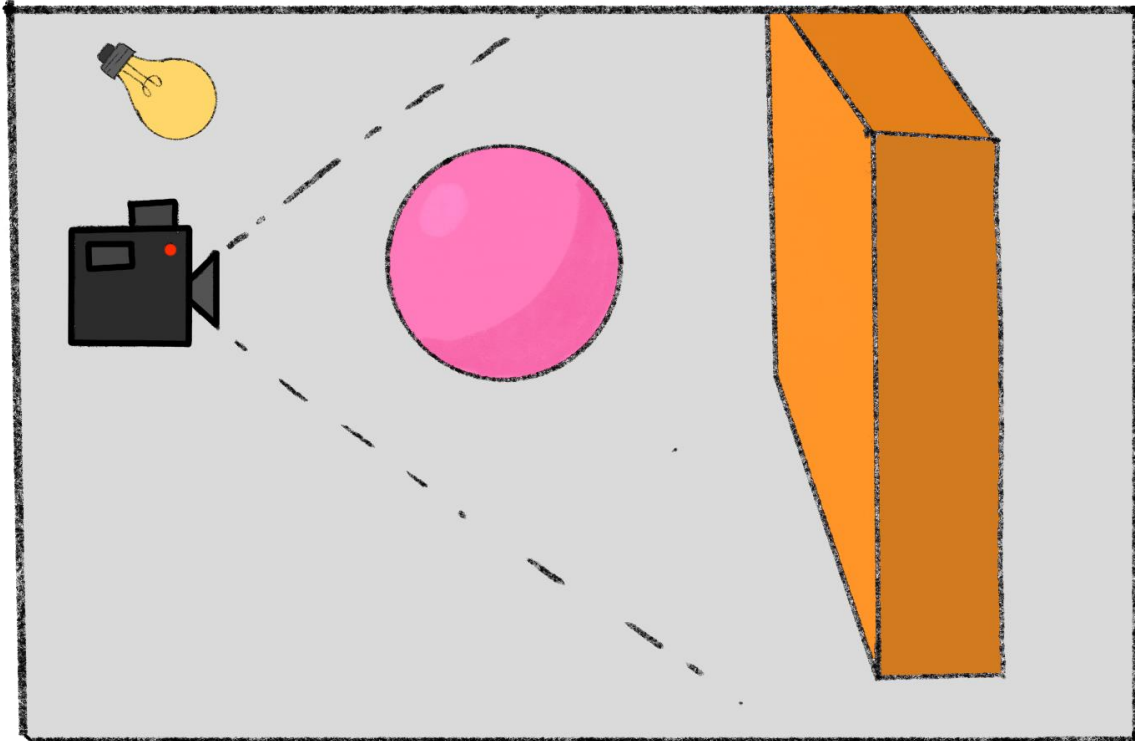
Light POV

# How It Works

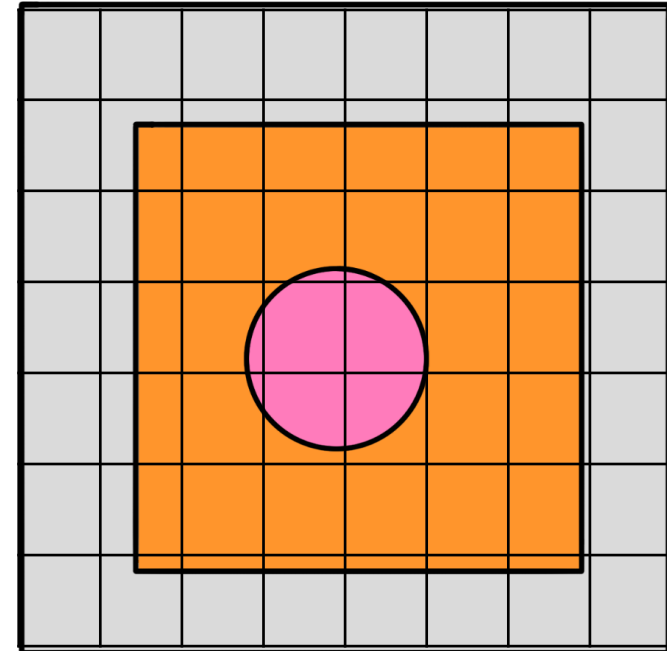


Light POV

# How It Works



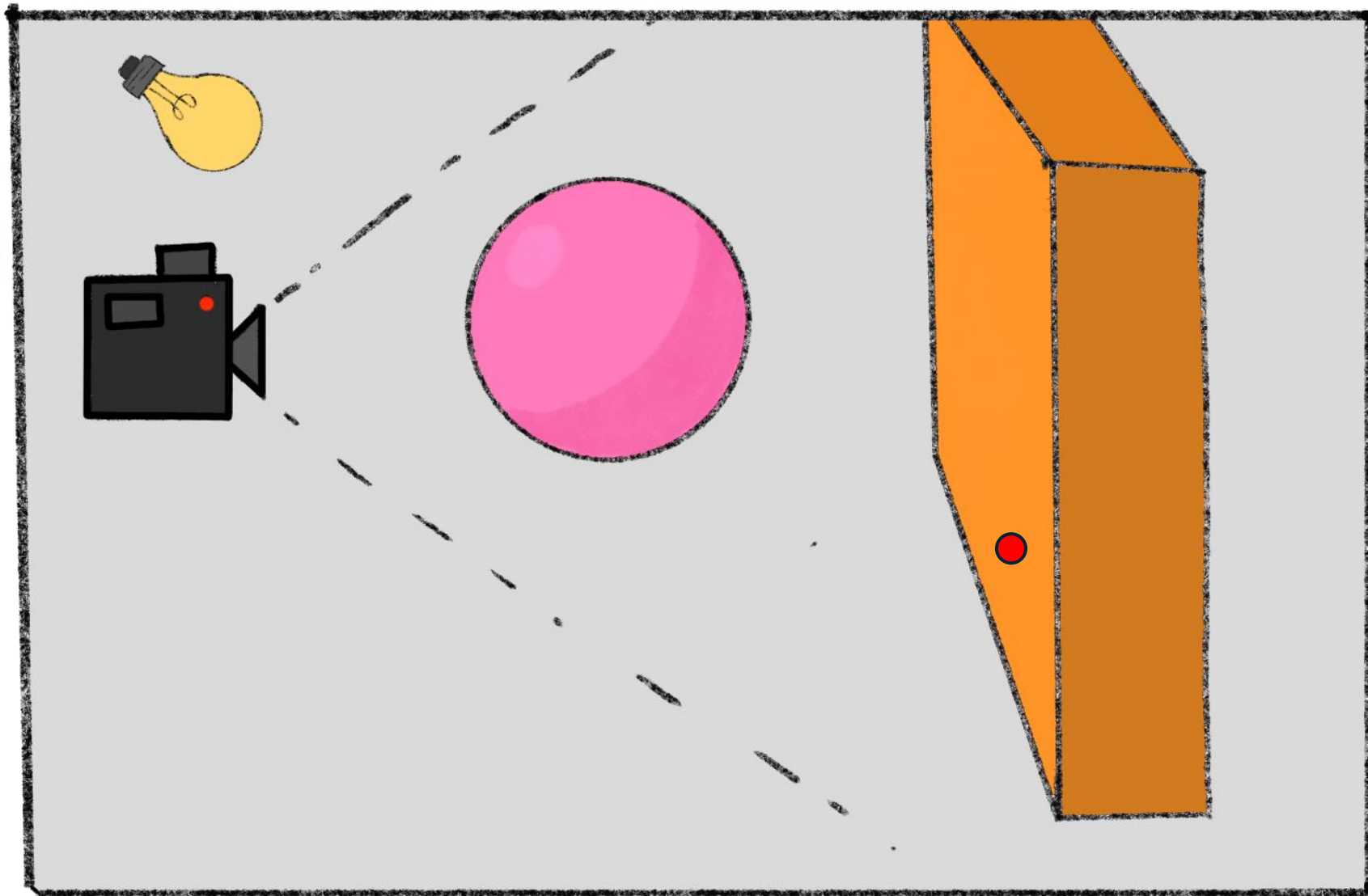
Z/ Depth Value for every  
pixel stored in "Shadow  
Buffer"



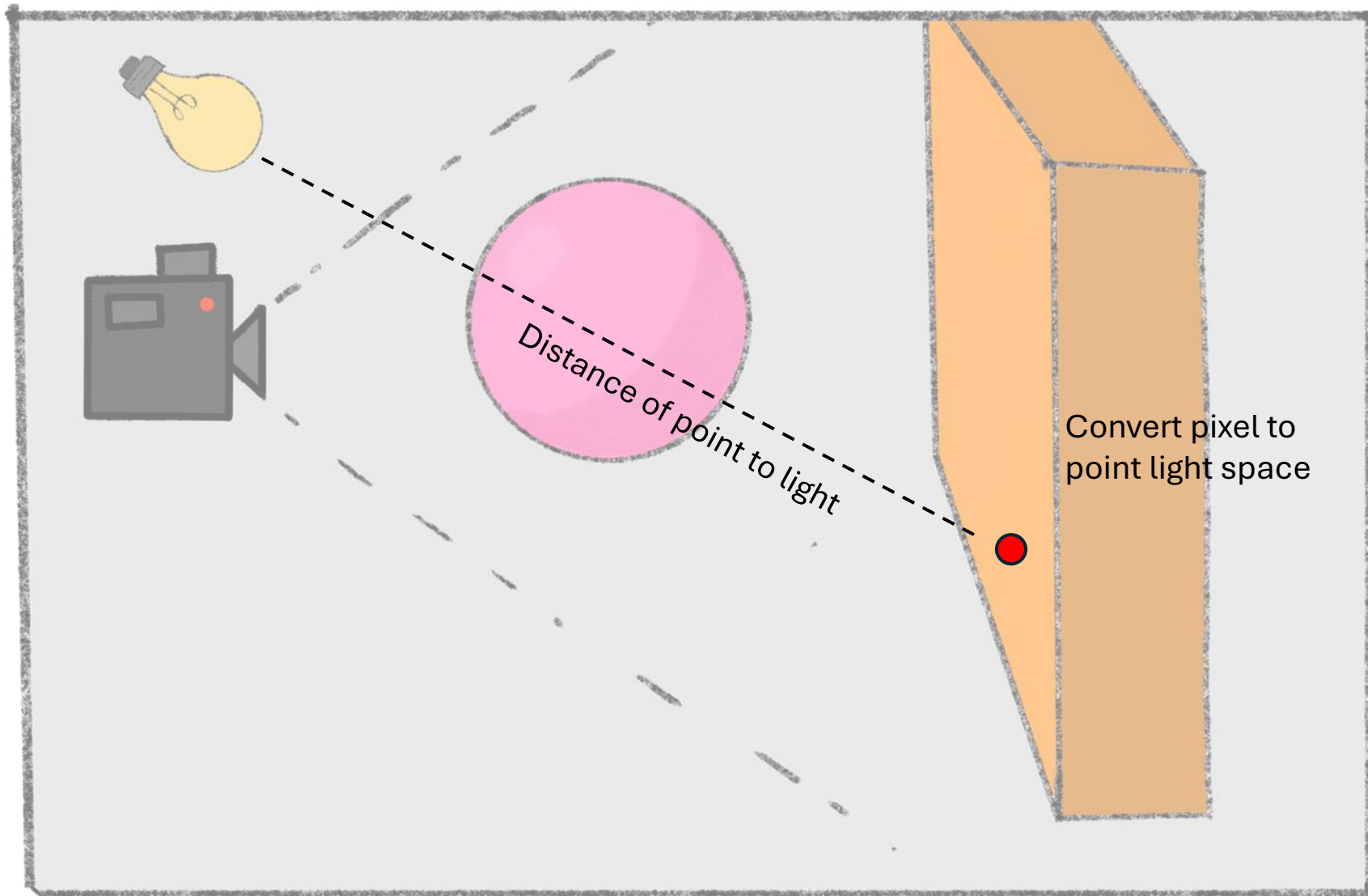
Light POV



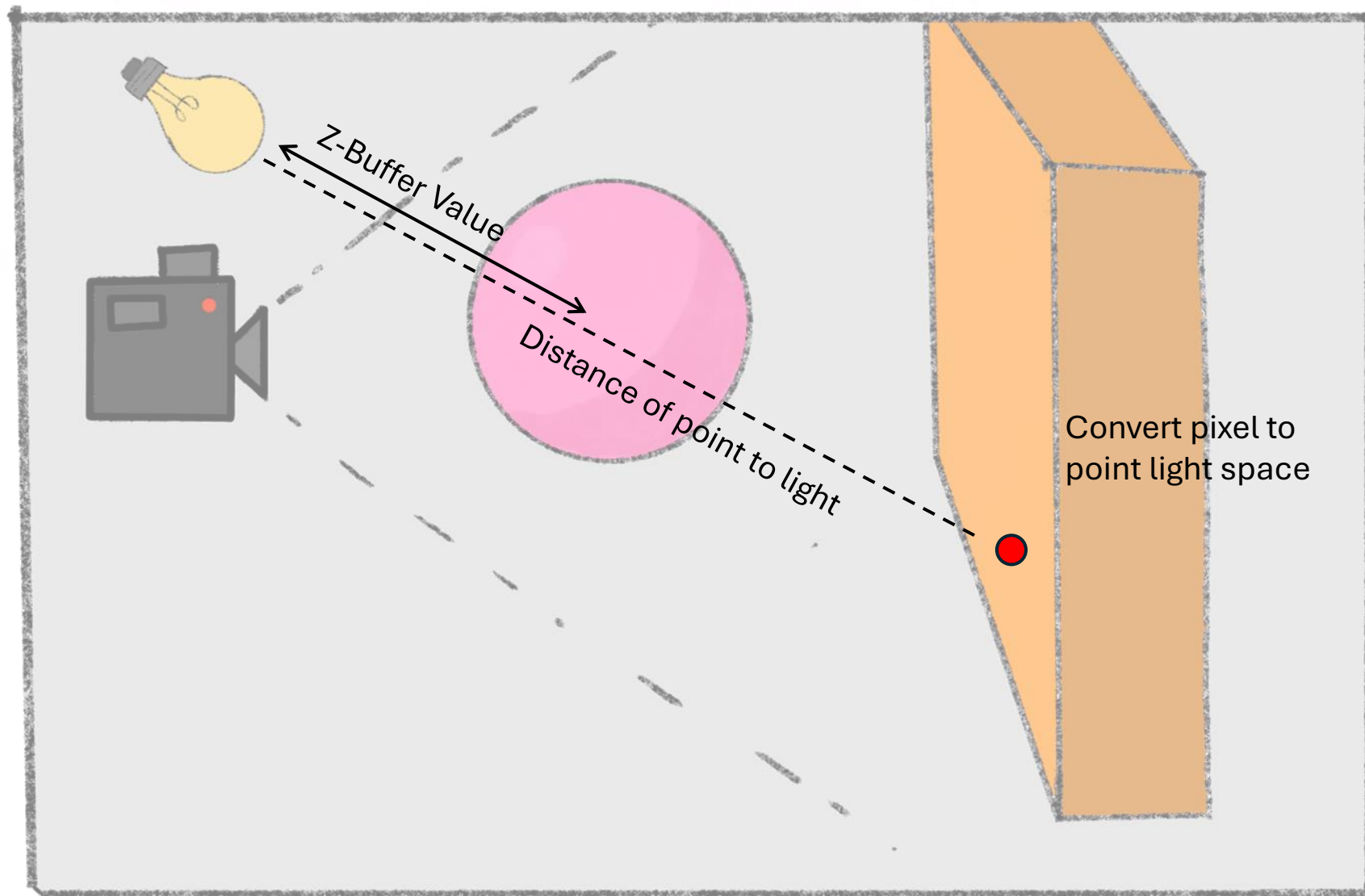
# How It Works



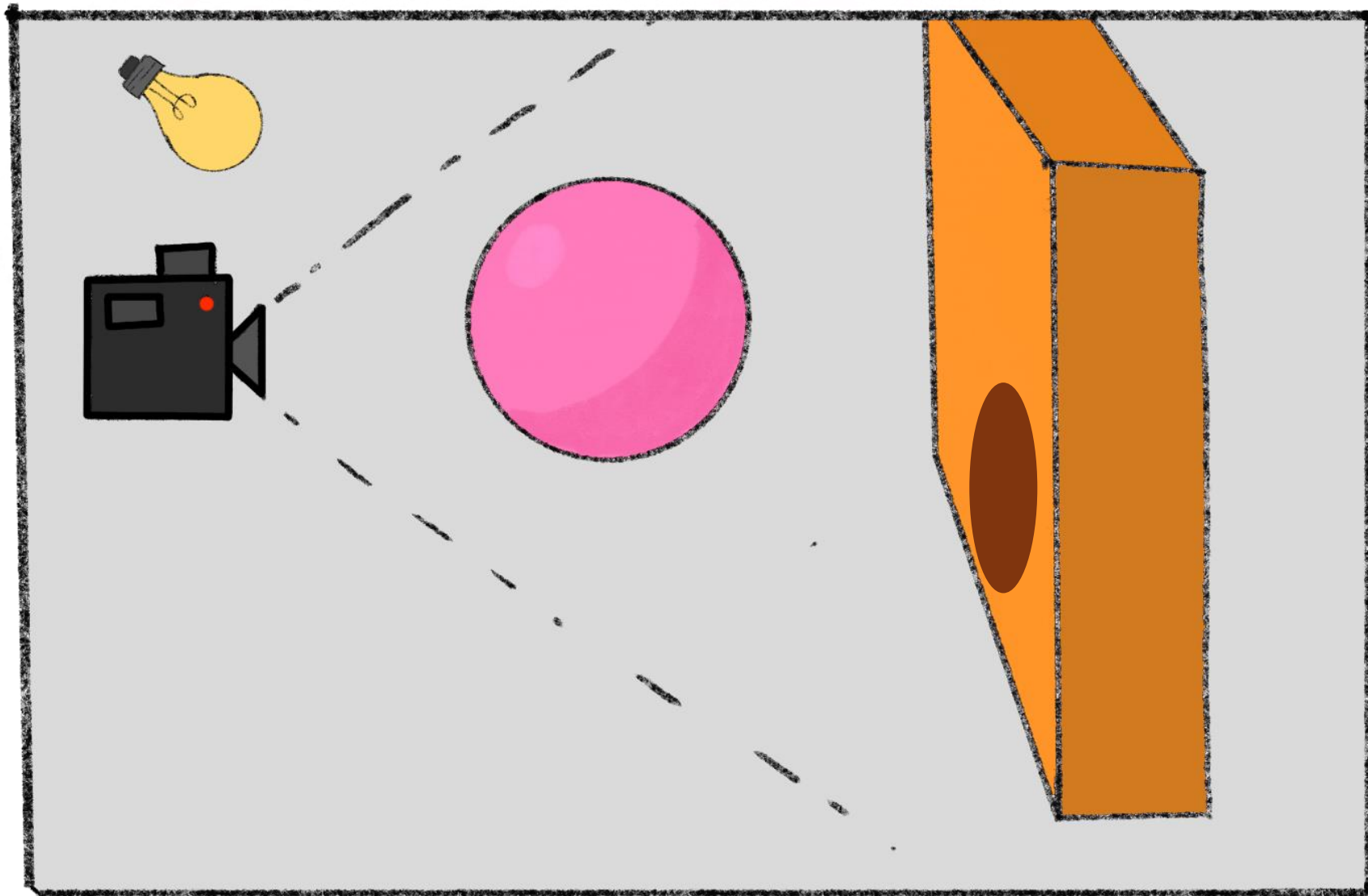
# How It Works



# How It Works



# How It Works



tada

# First Pass: Shadow Map

```
def render_shadow_map(self):
    """First pass: Generate shadow map from light's perspective"""
    # Save current camera
    original_camera = self.camera

    # Create camera from light's perspective
    light_camera = OrthoCamera(-10, 10, -10, 10, -10, 10)
    light_camera.transform = self.light.transform
    self.camera = light_camera

    # Clear shadow map
    self.z_buffer.fill(np.inf)

    # Render depth only from light's perspective
    for mesh in self.meshes:
        for face_idx, face in enumerate(mesh.faces):
            v0 = mesh.transform.apply_to_point(mesh.verts[face[0]])
            v1 = mesh.transform.apply_to_point(mesh.verts[face[1]])
            v2 = mesh.transform.apply_to_point(mesh.verts[face[2]])

            p0 = light_camera.project_point(v0)
            p1 = light_camera.project_point(v1)
            p2 = light_camera.project_point(v2)

            screen_p0 = np.array(self.screen.device_to_screen(p0[:2]))
            screen_p1 = np.array(self.screen.device_to_screen(p1[:2]))
            screen_p2 = np.array(self.screen.device_to_screen(p2[:2]))

            self._draw_triangle_depth_reversed(screen_p0, screen_p1, screen_p2,
                                              p0[2], p1[2], p2[2])

    # Save shadow map and restore camera
    self.shadow_map = self.z_buffer.copy()
    self.camera = original_camera
    self.z_buffer.fill(-np.inf)
```

Create Camera from  
Light's Perspective

Fill the Z buffer the way you  
did for depth rendering

# Second Pass: Rendering w/ Shadow Test

```
def _check_shadow(self, point):  
    """Helper function to check if a point is in shadow"""  
    light_camera = OrthoCamera(-10, 10, -10, 10, -10, 10)  
    light_camera.transform = self.light.transform  
  
    # Transform point to light space  
    point_homogeneous = np.append(point, 1)  
    point_light_space = np.dot(light_camera.transform.inverse_matrix(), point_homogeneous)[:3]  
    light_space_point = np.dot(light_camera.ortho_transform, np.append(point_light_space, 1))[:3]  
  
    # Get shadow map coordinates  
    shadow_coords = self.screen.device_to_screen(light_space_point[:2])  
    x, y = int(shadow_coords[0]), int(shadow_coords[1])  
  
    if (0 <= x < self.screen.width and 0 <= y < self.screen.height):  
        shadow_depth = self.shadow_map[y, x]  
        point_depth = light_space_point[2]  
        return point_depth > shadow_depth - self.shadow_bias  
  
    return False
```

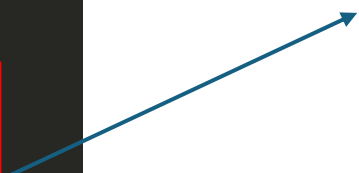
Transform point to  
light space

Compare light space point  
depth to shadow buffer value

# Second Pass: Rendering w/ Shadow Test

```
def _check_shadow(self, point):  
    """Helper function to check if a point is in shadow"""  
    light_camera = OrthoCamera(-10, 10, -10, 10, -10, 10)  
    light_camera.transform = self.light.transform  
  
    # Transform point to light space  
    point_homogeneous = np.append(point, 1)  
    point_light_space = np.dot(light_camera.transform.inverse_matrix(), point_homogeneous)[:3]  
    light_space_point = np.dot(light_camera.ortho_transform, np.append(point_light_space, 1))[:3]  
  
    # Get shadow map coordinates  
    shadow_coords = self.screen.device_to_screen(light_space_point[:2])  
    x, y = int(shadow_coords[0]), int(shadow_coords[1])  
  
    if (0 <= x < self.screen.width and 0 <= y < self.screen.height):  
        shadow_depth = self.shadow_map[y, x]  
        point_depth = light_space_point[2]  
        return point_depth > shadow_depth - self.shadow_bias  
  
    return False
```

Transform point to  
light space



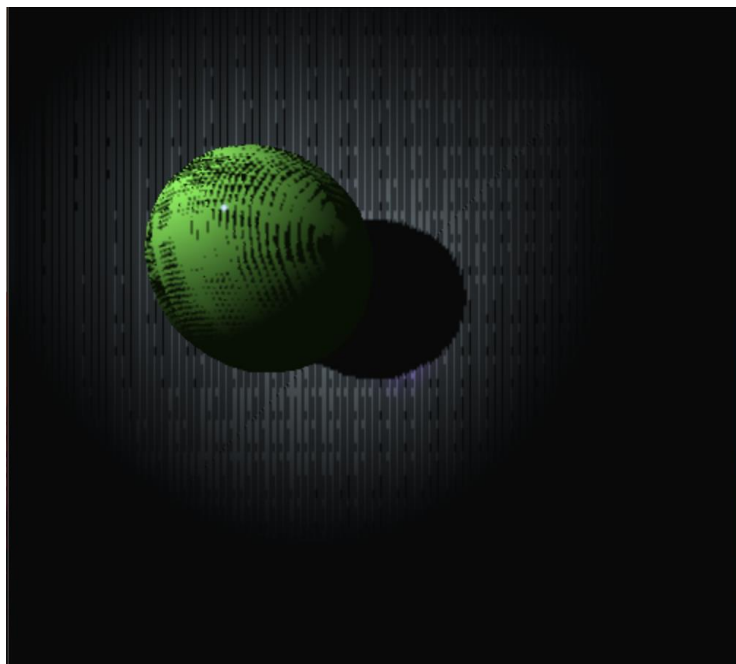
Compare light space point  
depth to shadow buffer value



```
def _draw_triangle_phong(self, p0, p1, p2, z0, z1, z2,  
    # Check if point is in shadow  
    in_shadow = self._check_shadow(world_pos)  
    if in_shadow:  
        # Only compute ambient lighting if in shadow  
        color = mesh.diffuse_color * mesh.ka * ambient_light  
    else:  
        # regular lighting and shading
```

# Challenges

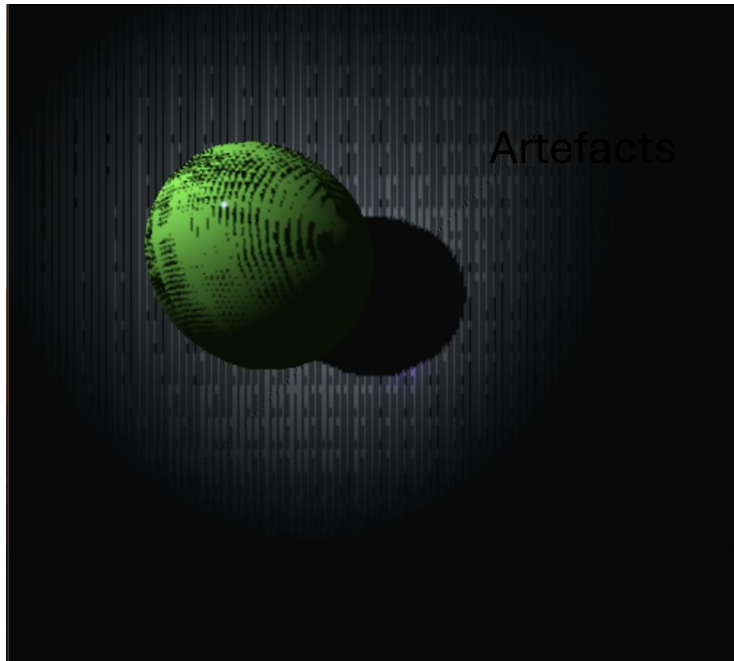
Artefacts in Render





# Challenges

## Artefacts in Render



Solution: PCF  
(Percent Closest Filtering)

```
for dx, dy in sample_offsets:
    x = int(x_base + dx)
    y = int(y_base + dy)

    if 0 <= x < self.screen.width and 0 <= y < self.screen.height:
        weight = gaussian_weight(dx, dy)
        shadow_depth = self.shadow_map[x, y]

        diff = point_depth - (shadow_depth + self.shadow_bias)
        if diff > 0:
            shadow_amount = min(1.0, diff * 10)
            shadow_factor += weight * shadow_amount

        total_weight += weight

# Normalize shadow factor
if total_weight > 0:
    shadow_factor /= total_weight

return shadow_factor > 0.5
```

# References

- <https://www.youtube.com/watch?v=LUjXAoP5GG0>
- <https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping#:~:text=The%20idea%20behind%20shadow%20mapping,itself%20and%20a%20light%20source.>
- <https://www.opengldev.org/www/tutorial42/tutorial42.html>