Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

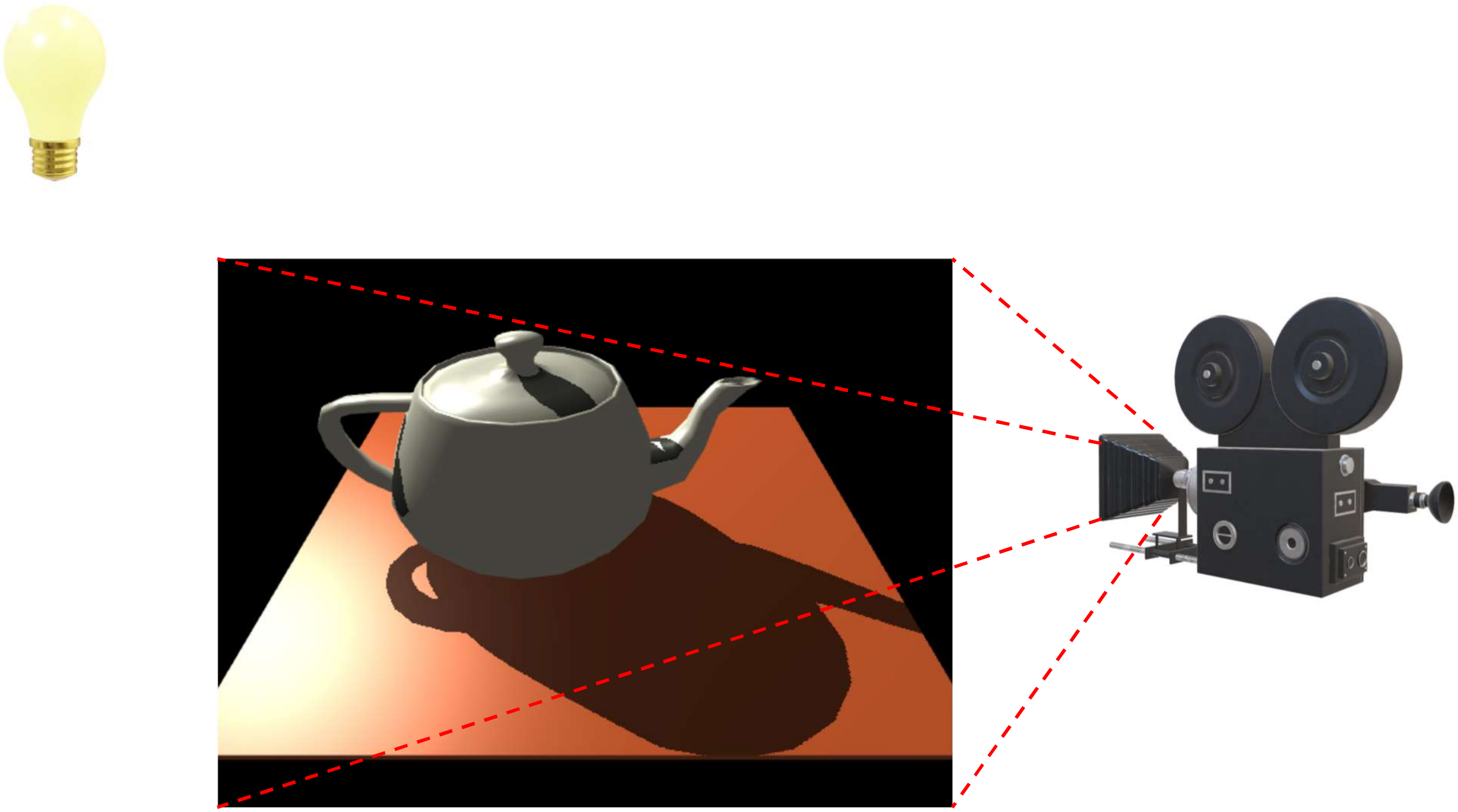
IITK CS360A

Acknowledgements

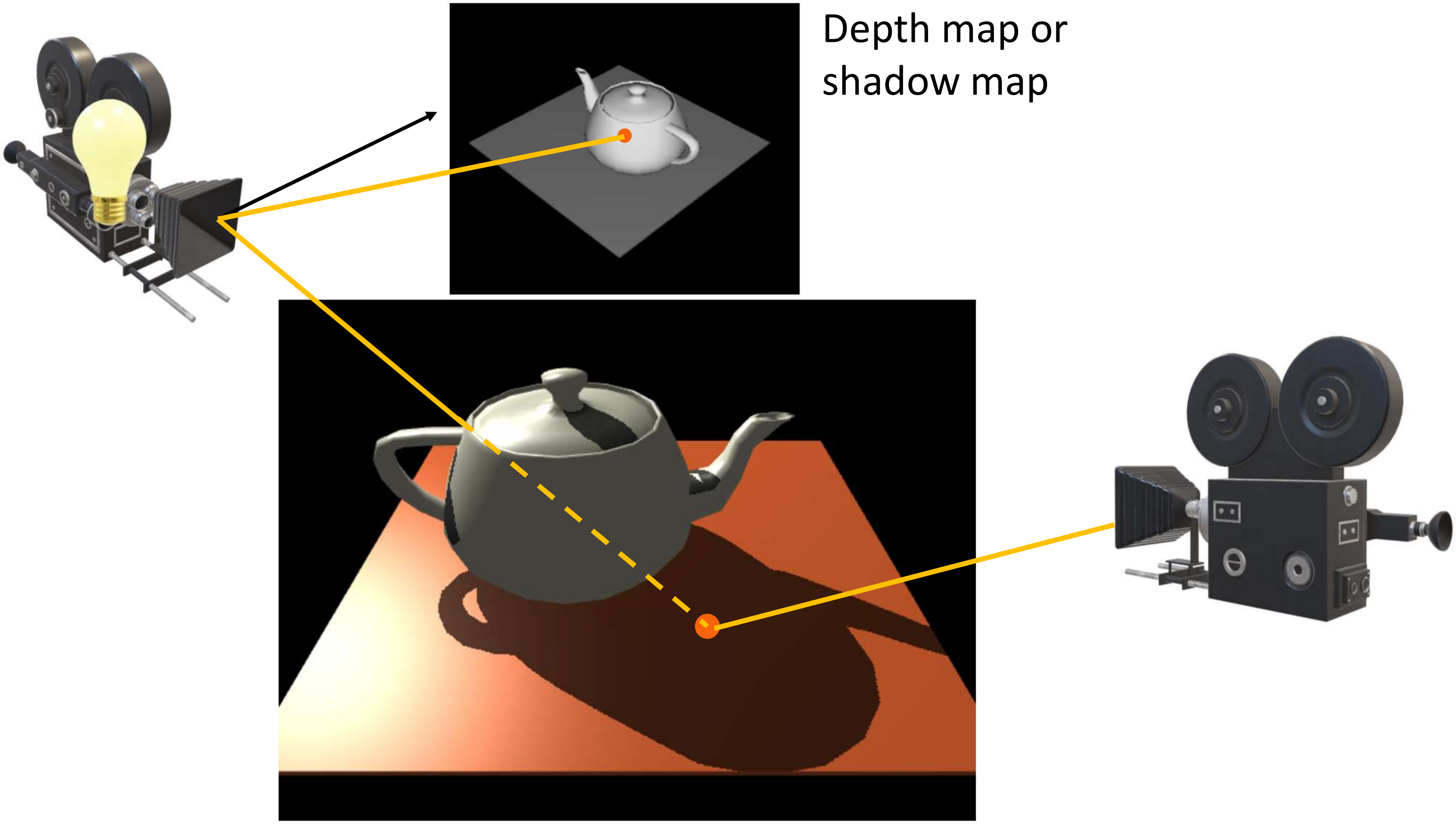
- A subset of the slides that I will present throughout the course are adapted/inspired by excellent courses on Computer Graphics offered by Prof. Han-Wei Shen, Prof. Wojciech Matusik, Prof. Frédo Durand, Prof. Abe Davis**,** and Prof. Cem Yuksel

Shadow

Shadow Map Idea



Shadow Map Idea

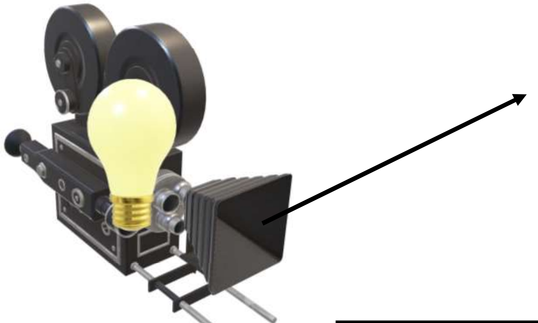


Depth map or shadow map

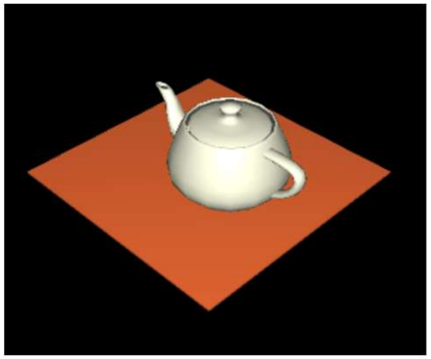
Two Pass Algorithm

- First pass
  - Render the scene with respect to light’s perspective, i.e., put your camera at the location of light
  - Record the depth values from each point
  - Store the depth values into an FBO as depth map (shadow map)
- Second pass
  - Render the scene as usual
  - Use the depth map to compare the depth value of current location to the depth from the light’s perspective
  - If current depth > shadow map depth, then the point is in shadow

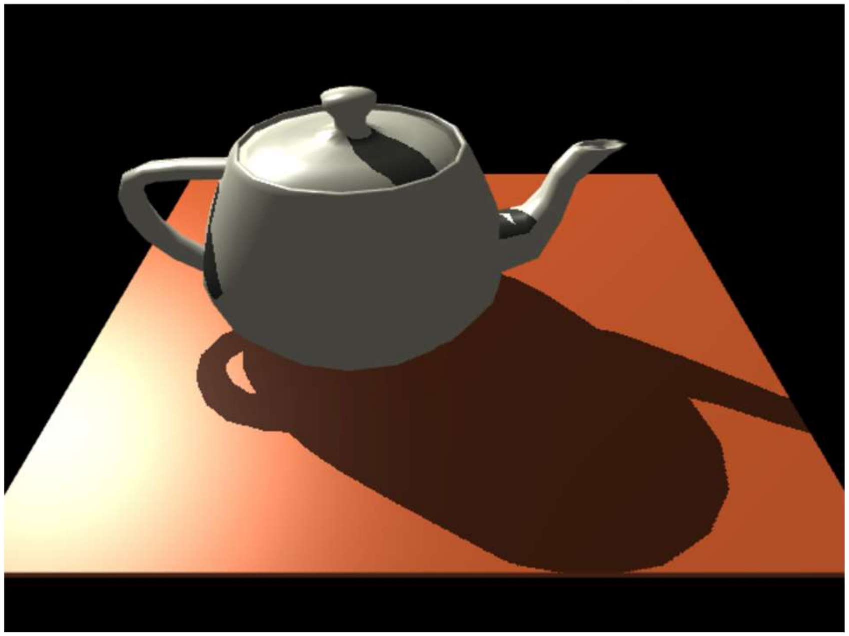
Shadow Map Idea



Place camera at light position

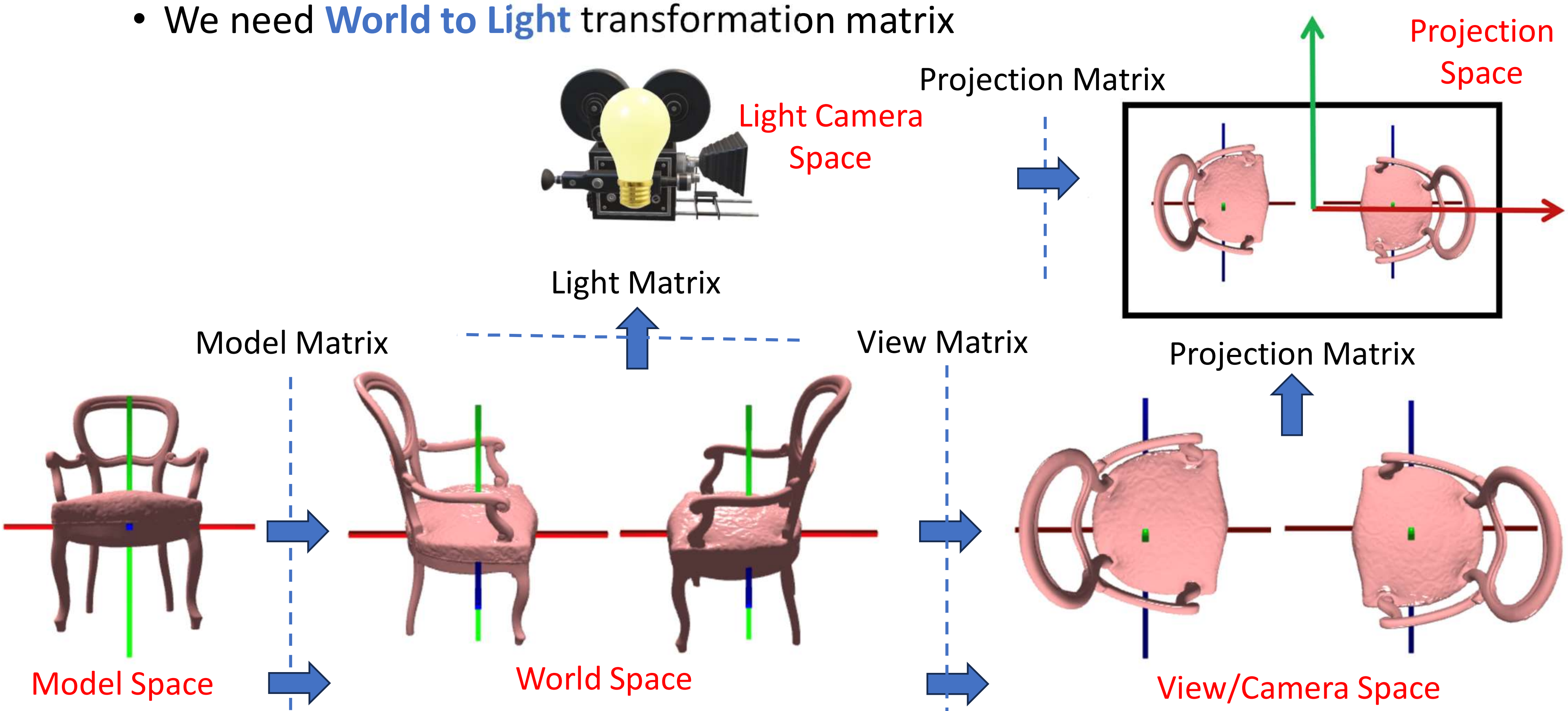


Scene from light’s POV



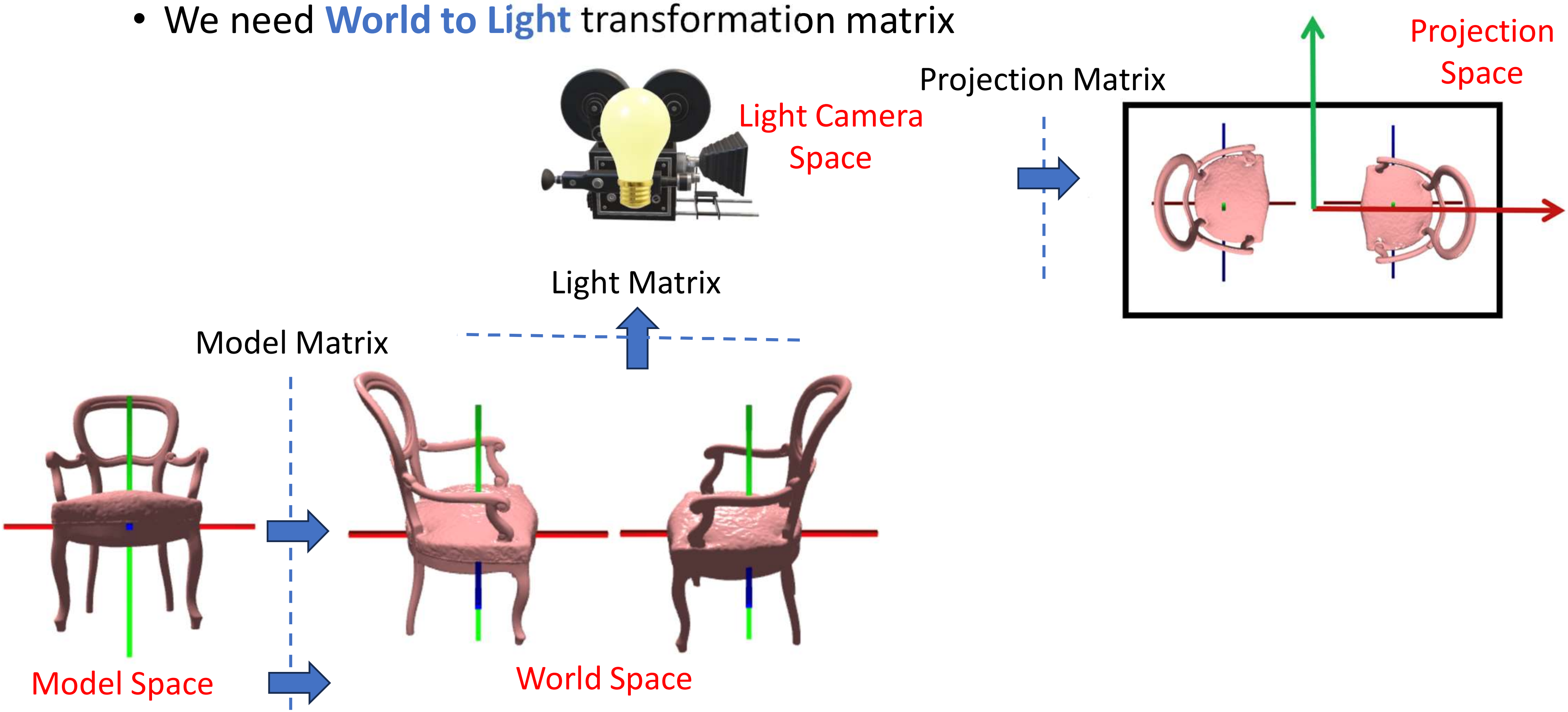
Transformations for Shadow Map

- Need to render the <u>depth map</u> from light's perspective
  - We need **World to Light** transformation matrix

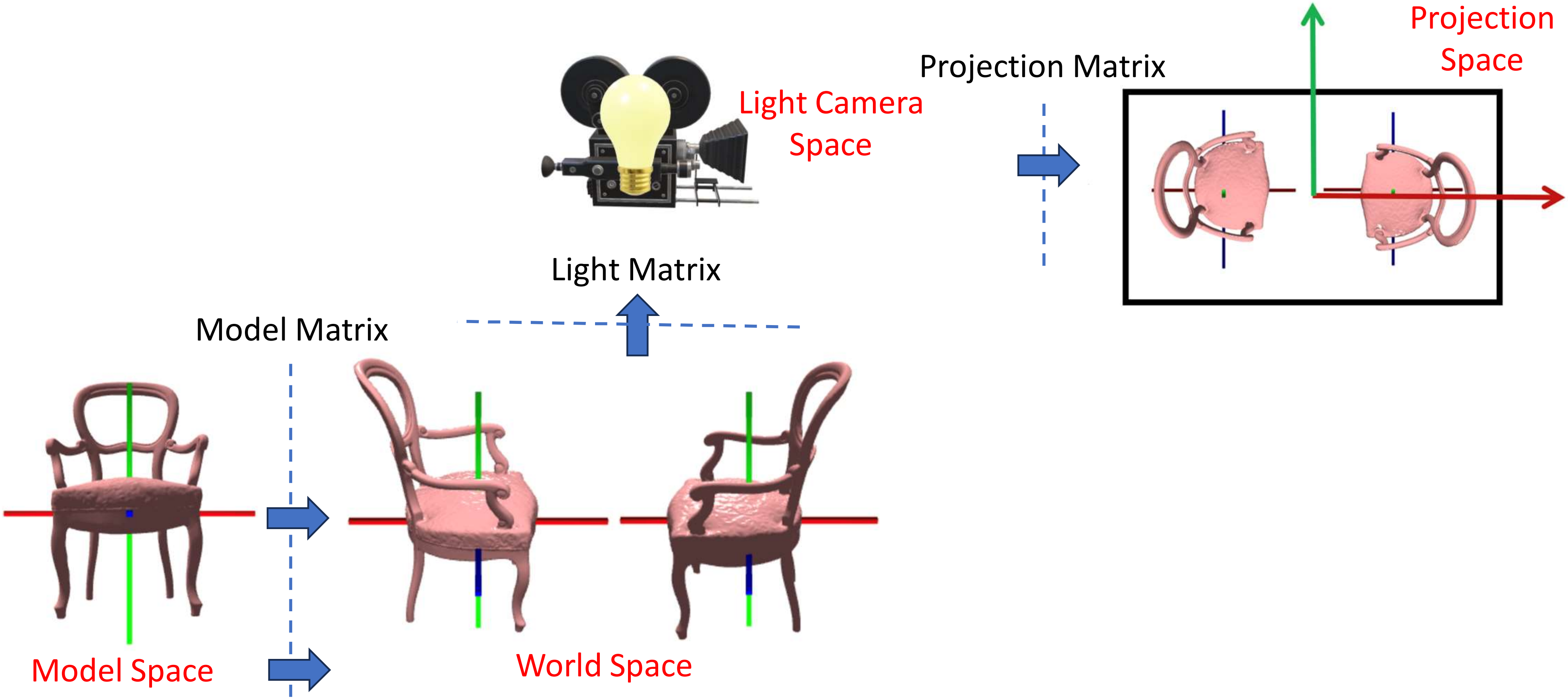


Transformations for Shadow Map

- Need to render the depth map from light's perspective
    - We need **World to Light** transformation matrix



Shadow Mapping Pass 1



Model Matrix

Light Matrix

Light Camera Space

Projection Matrix

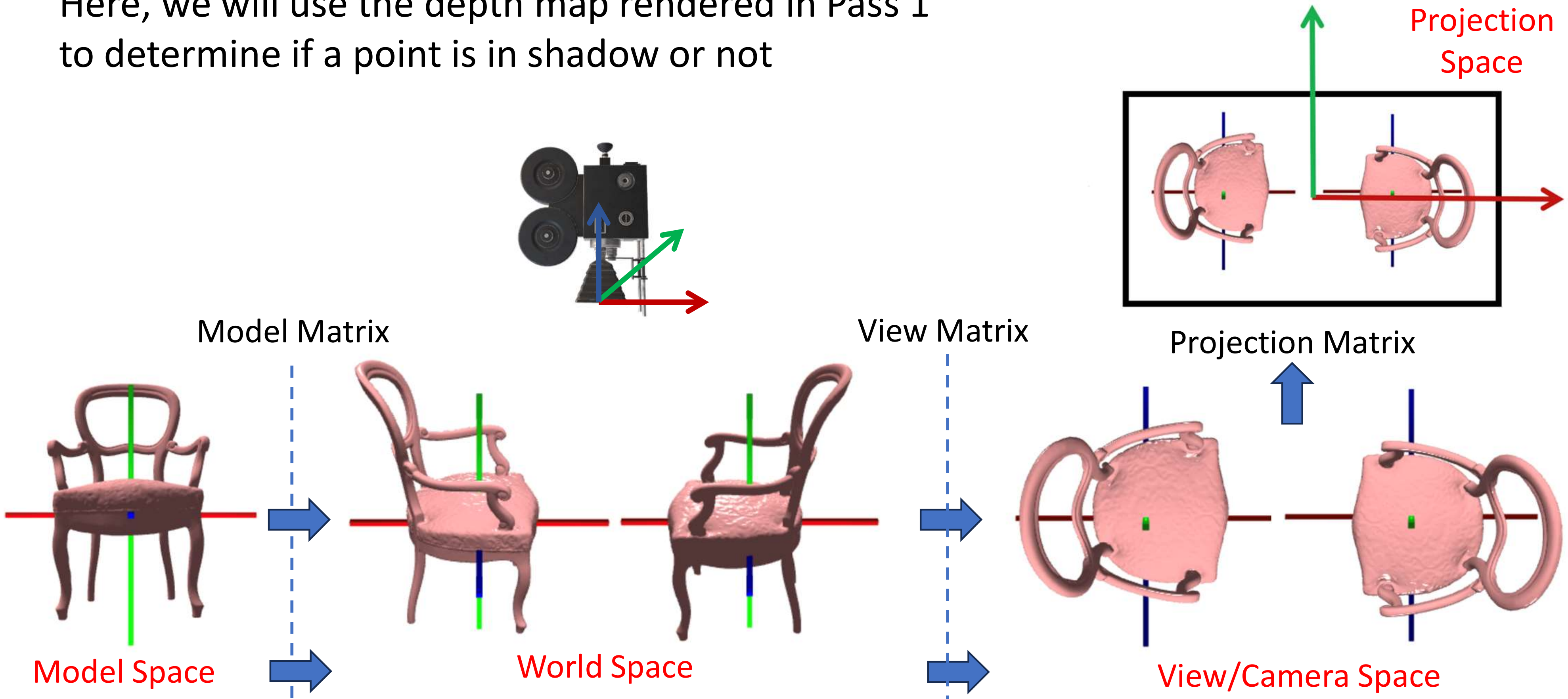
Projection Space

Model Space

World Space

Shadow Mapping Pass 2

Here, we will use the depth map rendered in Pass 1 to determine if a point is in shadow or not



Projection Space

Model Matrix

View Matrix

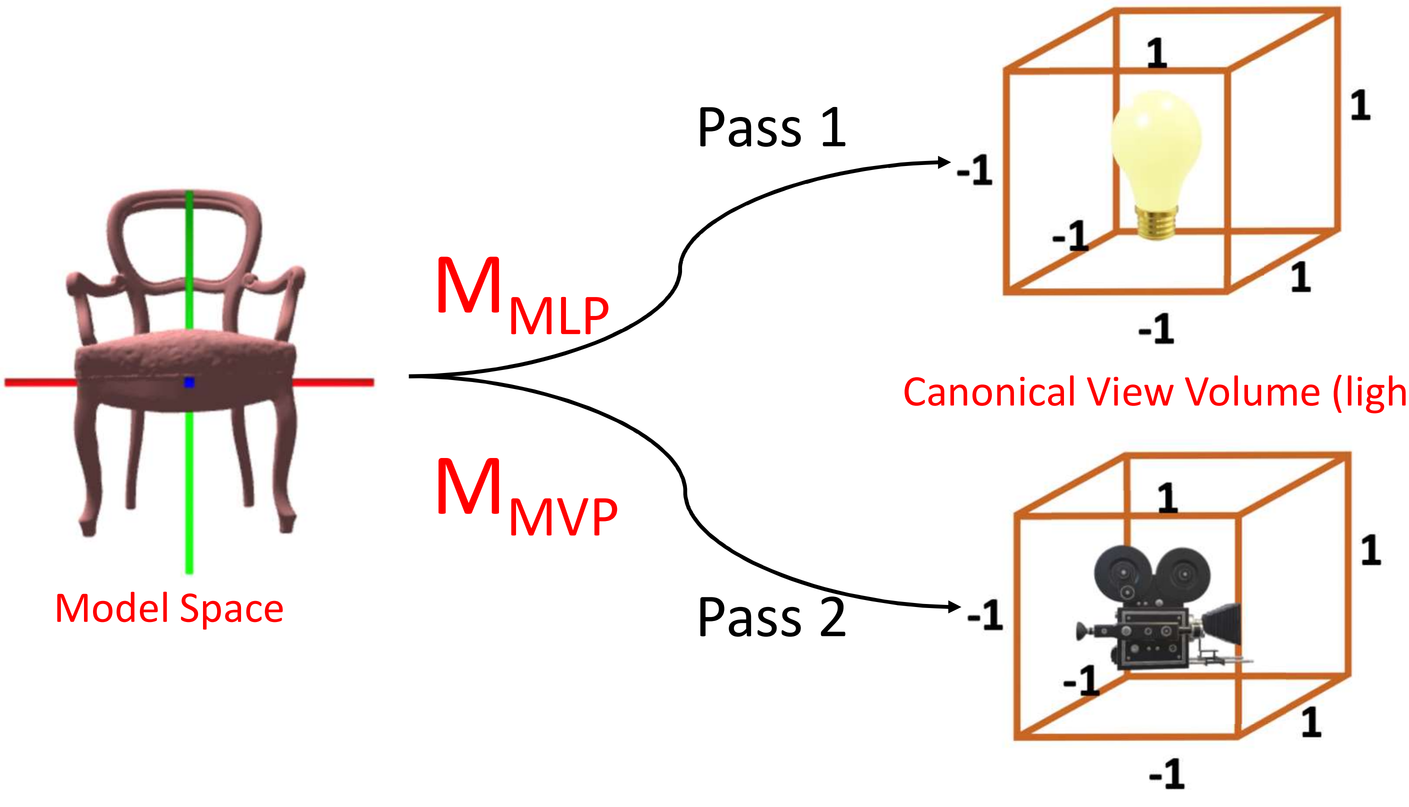
Projection Matrix

Model Space

World Space

View/Camera Space

Shadow Transformations



Model Space

M_{MLP}

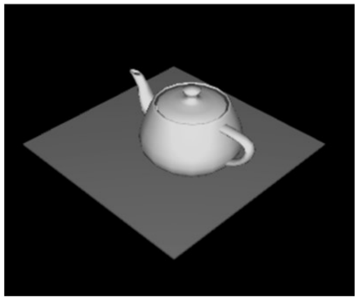
Pass 1

M_{MVP}

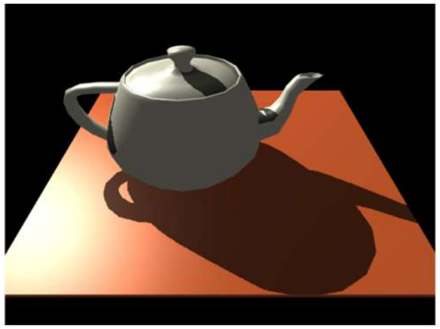
Pass 2

Canonical View Volume (light)

Canonical View Volume (standard)



Depth Map

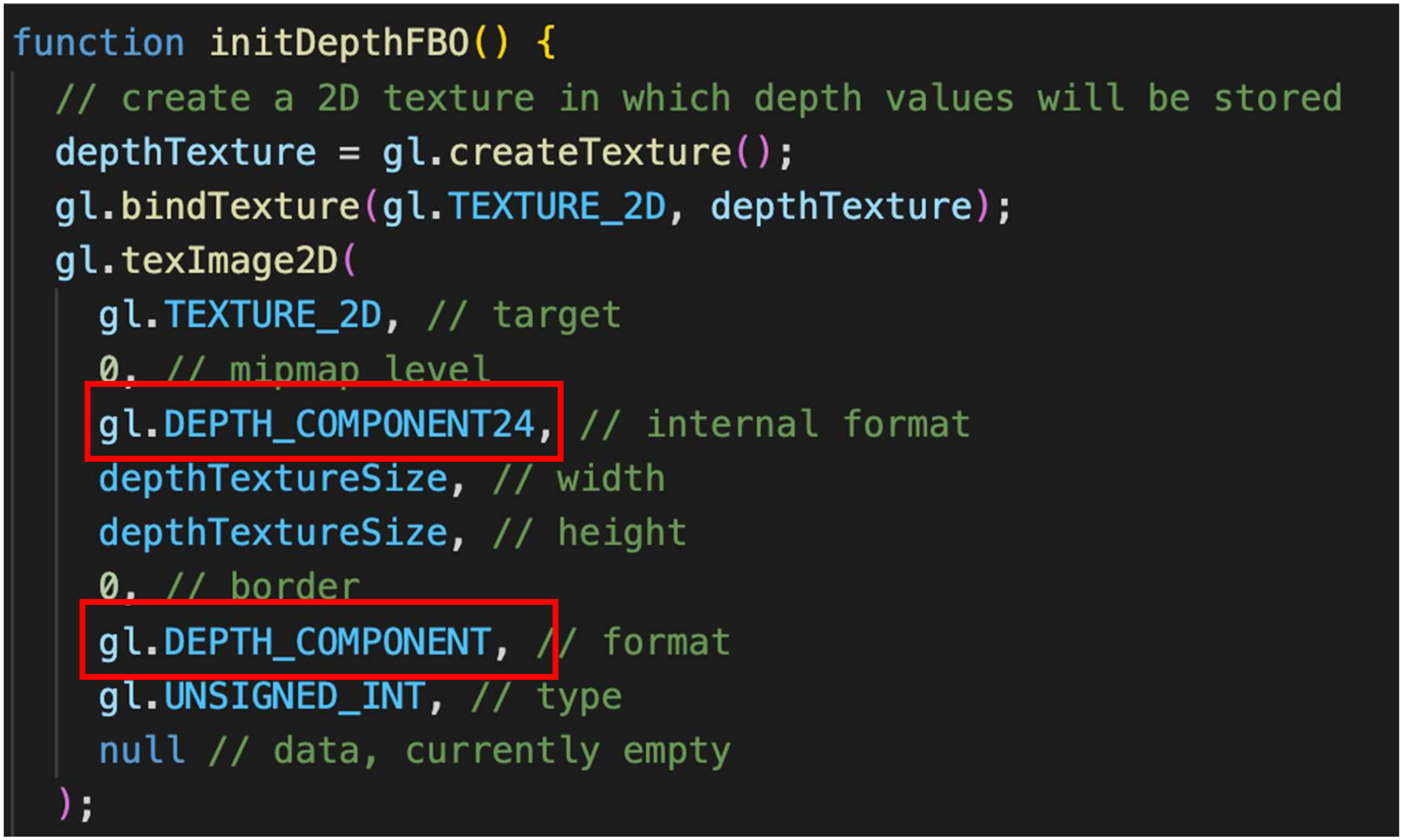


Final Scene

Implementation Details

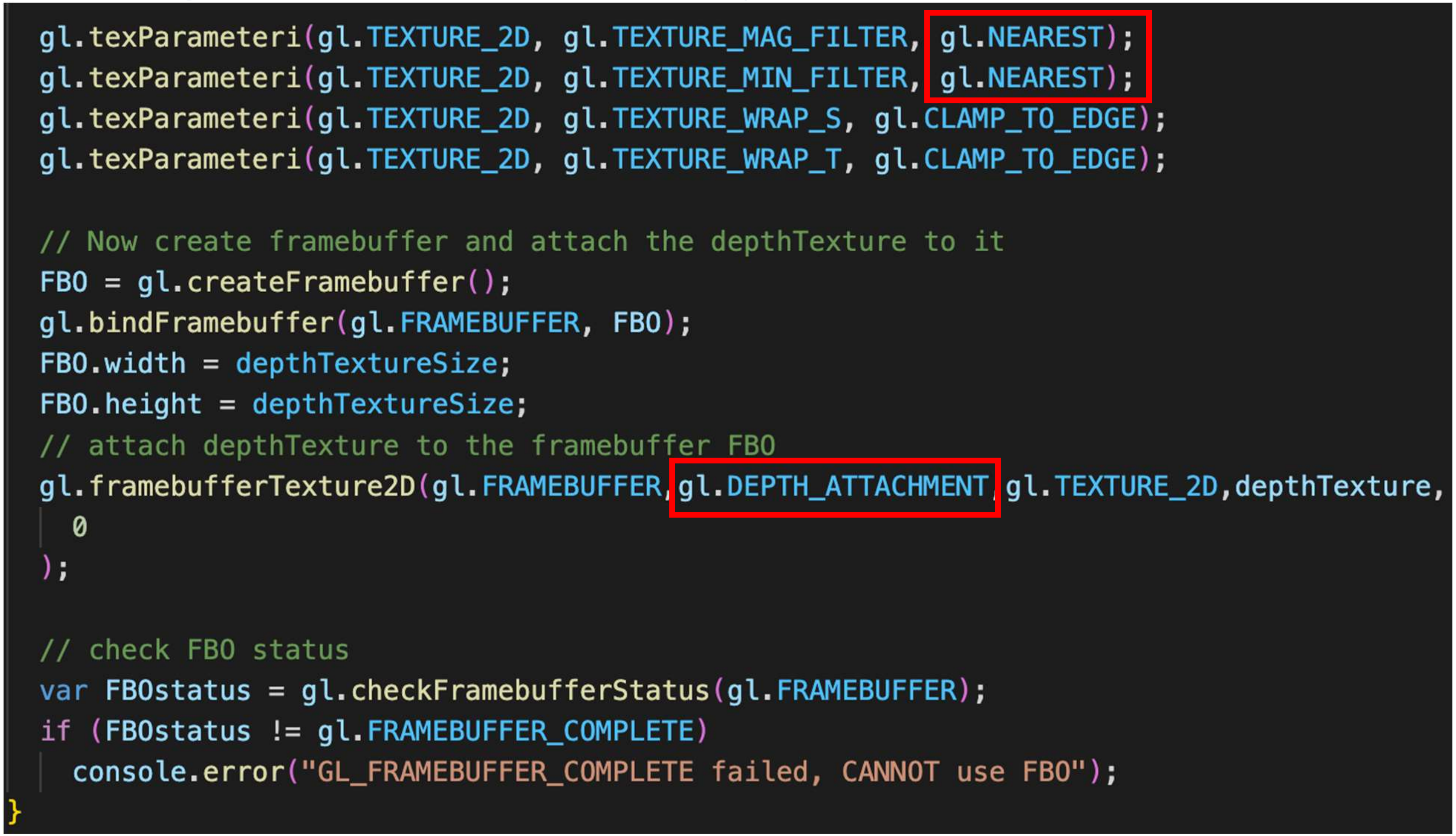
- Two pass rendering algorithm
- Two shaders for two passes
- Pass 1: Render the scene to depth texture from light’s view using FBO
- Pass 2: Render the scene with depth comparison from depth texture to determine if a fragment is in shadow or not

Configure FBO with Depth Texture



```
function initDepthFBO() {
  // create a 2D texture in which depth values will be stored
  depthTexture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, depthTexture);
  gl.texImage2D(
    gl.TEXTURE_2D, // target
    0, // mipmap level
    gl.DEPTH_COMPONENT24, // internal format
    depthTextureSize, // width
    depthTextureSize, // height
    0, // border
    gl.DEPTH_COMPONENT, // format
    gl.UNSIGNED_INT, // type
    null // data, currently empty
  );
```

Configure FBO with Depth Texture



```
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);

  // Now create framebuffer and attach the depthTexture to it
  FBO = gl.createFramebuffer();
  gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);
  FBO.width = depthTextureSize;
  FBO.height = depthTextureSize;
  // attach depthTexture to the framebuffer FBO
  gl.framebufferTexture2D(gl.FRAMEBUFFER,gl.DEPTH_ATTACHMENT,gl.TEXTURE_2D,depthTexture,
    0
  );

  // check FBO status
  var FBOstatus = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
  if (FBOstatus != gl.FRAMEBUFFER_COMPLETE)
    console.error("GL_FRAMEBUFFER_COMPLETE failed, CANNOT use FBO");
}
```

DrawScene Function: Pass 1

- function drawScene() {

```
//Draw into framebuffer
gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);

shaderProgram = shadowPassShaderProgram;
gl.useProgram(shaderProgram);

// set up the light view matrix
mat4.identity(vMatrix);
vMatrix = mat4.lookAt(lightPos, [xCam, yCam, zCam], [0, 1, 0], vMatrix);
```

DrawScene Function: Pass 2

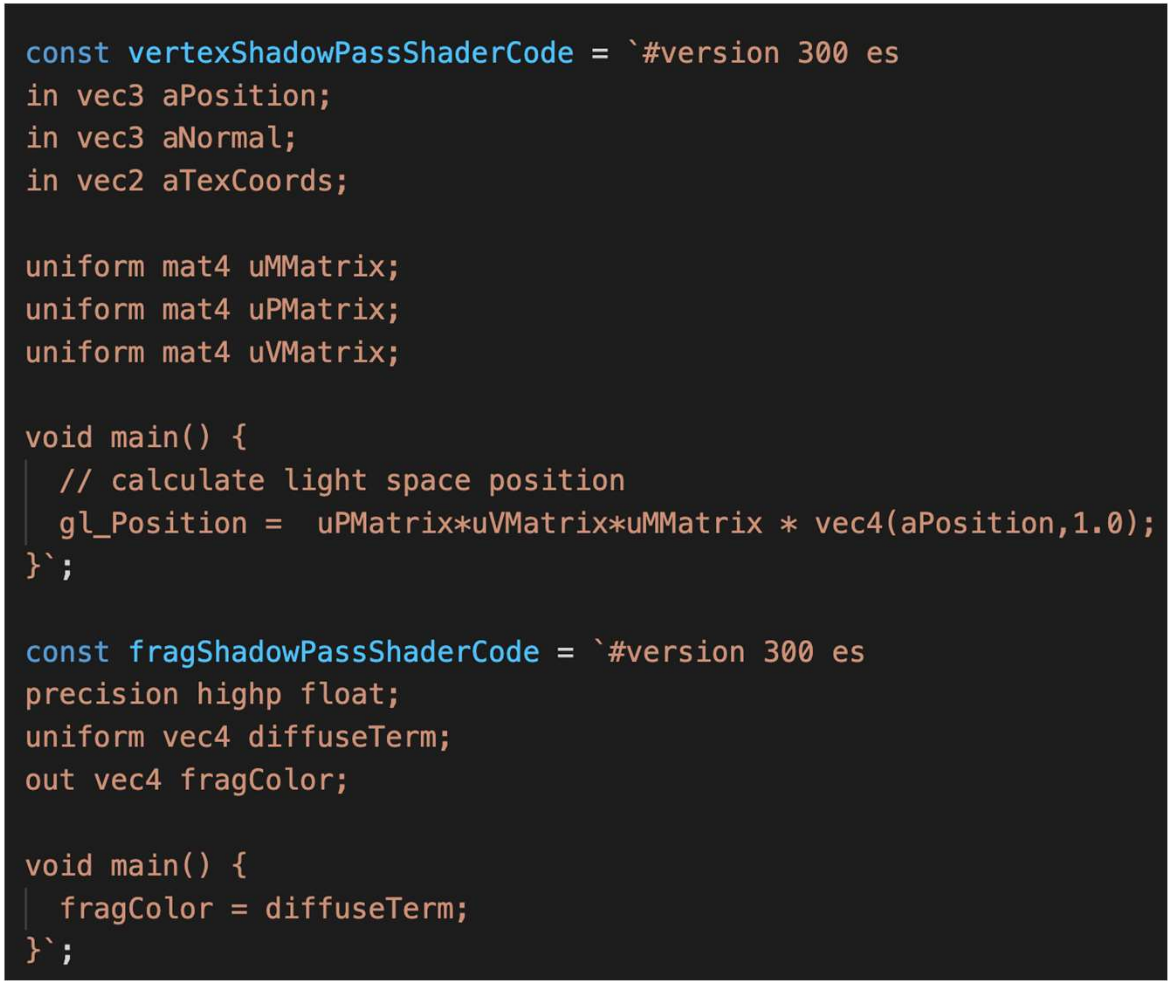
- function drawScene() {

```
//Draw into screen
gl.bindFramebuffer(gl.FRAMEBUFFER, null);

shaderProgram = renderPassShaderProgram;
gl.useProgram(shaderProgram);

// setup light view matrix
mat4.identity(lvMatrix);
lightViewMat = mat4.lookAt(lightPos, [xCam, yCam, zCam], [0, 1, 0], lightViewMat);
lvMatrix = mat4.multiply(lvMatrix, pMatrix);
lvMatrix = mat4.multiply(lvMatrix, lightViewMat);
```

DrawScene Function: Pass 2

```
//for texture binding from FBO
gl.activeTexture(gl.TEXTURE0); // set texture unit 1 to use
gl.bindTexture(gl.TEXTURE_2D, depthTexture); // bind the texture object to the texture unit
gl.uniform1i(uShadowLocation, 0); // pass the texture unit to the shader
}
```

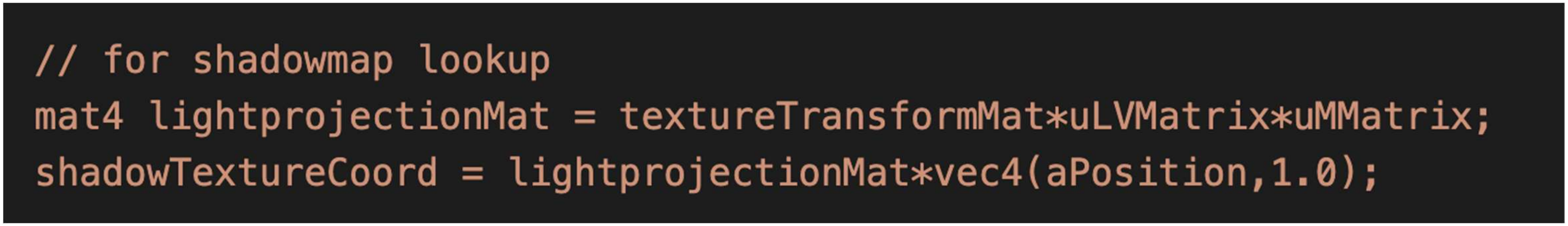
Pass 1 Shaders: Shadow Pass



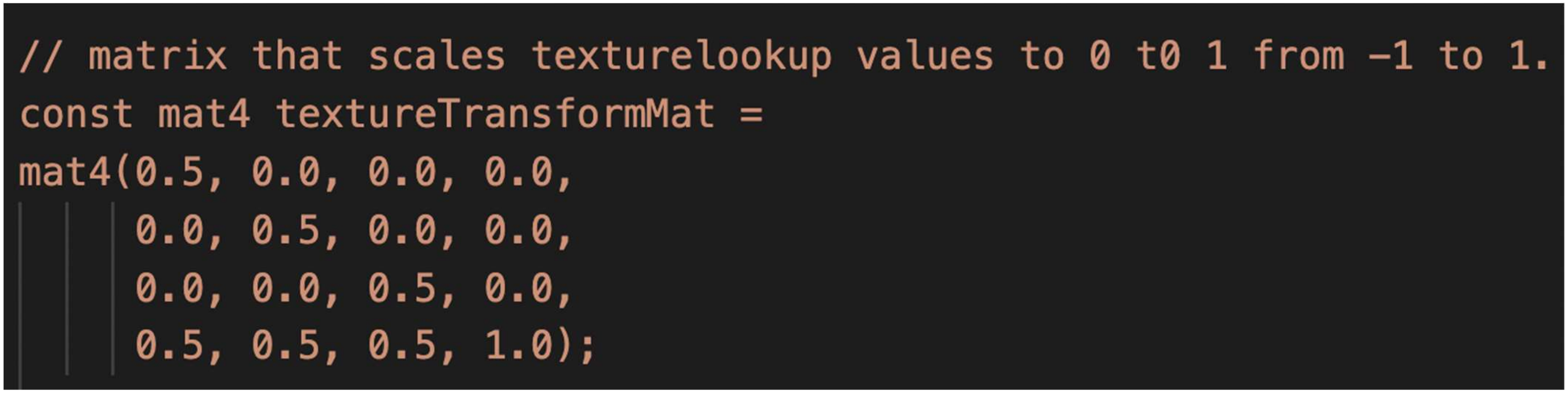
```
const vertexShadowPassShaderCode = `#version 300 es
in vec3 aPosition;
in vec3 aNormal;
in vec2 aTexCoords;

uniform mat4 uMMatrix;
uniform mat4 uPMatrix;
uniform mat4 uVMatrix;

void main() {
  // calculate light space position
  gl_Position =  uPMatrix*uVMatrix*uMMatrix * vec4(aPosition,1.0);
}`;

const fragShadowPassShaderCode = `#version 300 es
precision highp float;
uniform vec4 diffuseTerm;
out vec4 fragColor;

void main() {
  fragColor = diffuseTerm;
}`;
```

Pass 2 Shaders: Render Pass

Vertex Shader Excerpt



```
// for shadowmap lookup
mat4 lightprojectionMat = textureTransformMat*uLVMatrix*uMMatrix;
shadowTextureCoord = lightprojectionMat*vec4(aPosition,1.0);
```



```
// matrix that scales texturelookup values to 0 t0 1 from -1 to 1.
const mat4 textureTransformMat =
mat4(0.5, 0.0, 0.0, 0.0,
     0.0, 0.5, 0.0, 0.0,
     0.0, 0.0, 0.5, 0.0,
     0.5, 0.5, 0.5, 1.0);
```

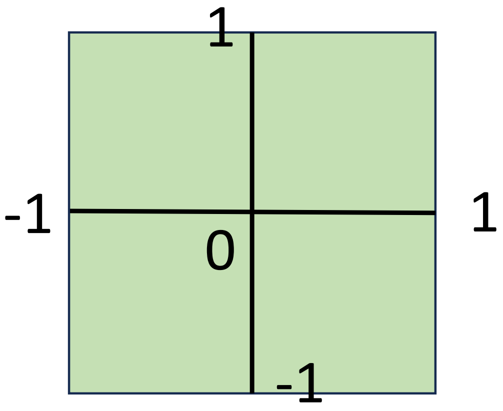
TextureTransformMat

- Texture coordinates are between 0-1
- Positions in Canonical view volume after MVP transformation scales to -1 to 1
- To do correct texture look up using values at canonical view volume, we need to scale them to 0-1 range
- **textureTransformMat** helps to achieve that

**[0.5, 0.0, 0.0, 0.0,
0.0, 0.5, 0.0, 0.0,
0.0, 0.0, 0.5, 0.0,
0.5, 0.5, 0.5, 1.0]**

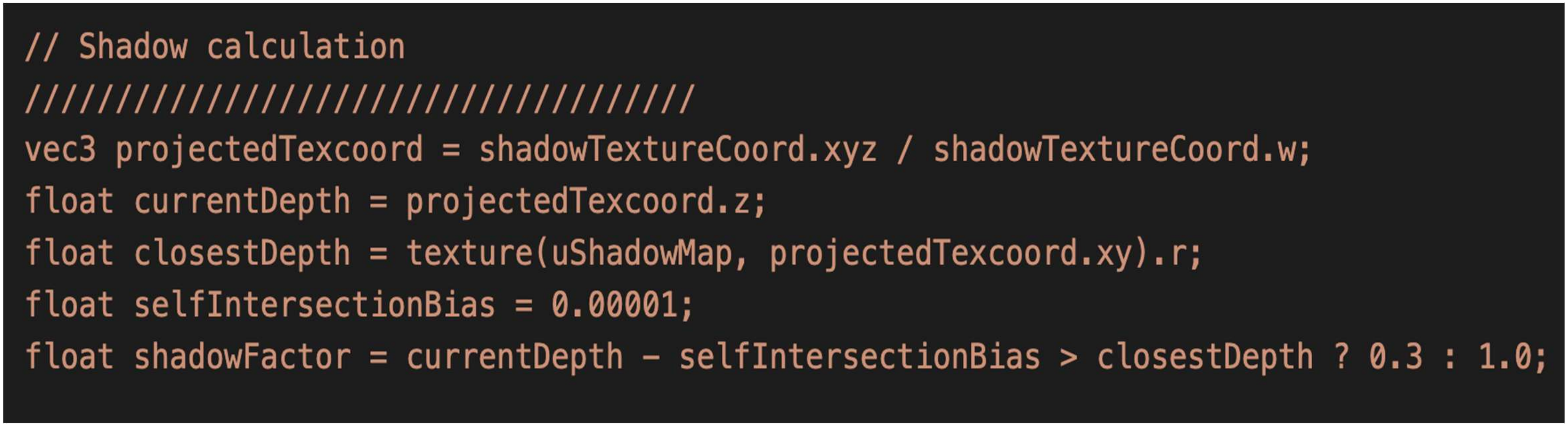
textureTransformMat

Note the Row vs Column major representation



Pass 2 Shaders: Render Pass

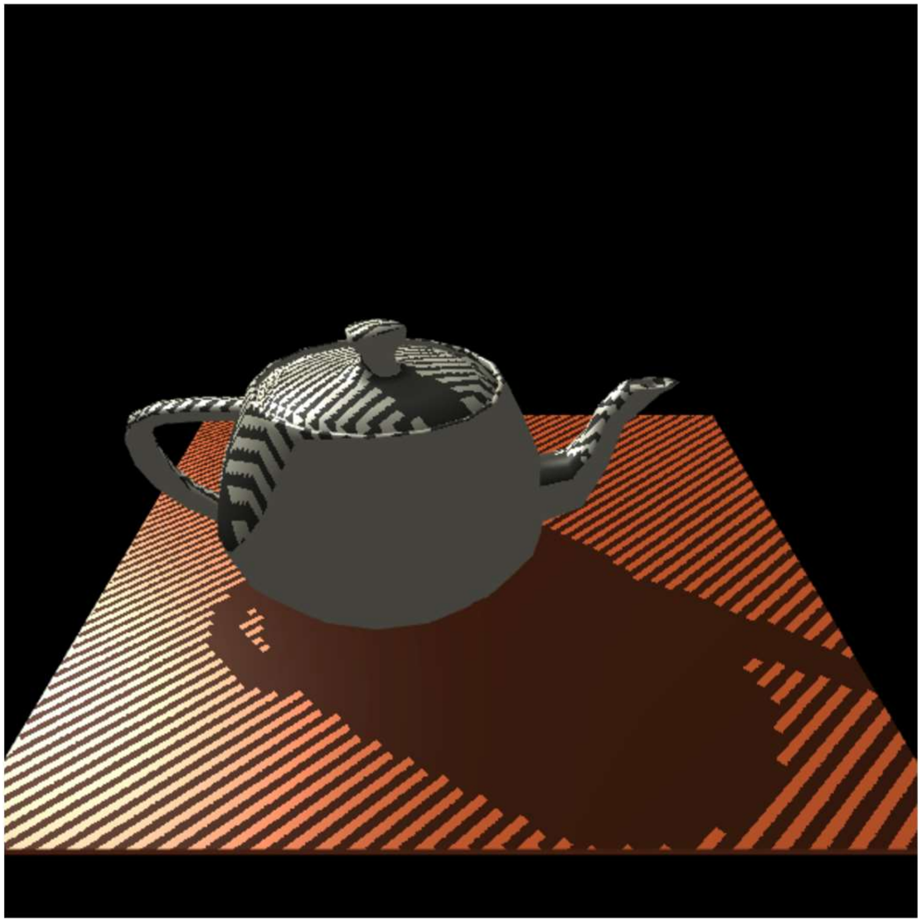
Fragment Shader Excerpt



```
// Shadow calculation
/////////////////////////////////////
vec3 projectedTexcoord = shadowTextureCoord.xyz / shadowTextureCoord.w;
float currentDepth = projectedTexcoord.z;
float closestDepth = texture(uShadowMap, projectedTexcoord.xy).r;
float selfIntersectionBias = 0.00001;
float shadowFactor = currentDepth - selfIntersectionBias > closestDepth ? 0.3 : 1.0;
```

```
fragColor = shadowFactor*phongColor;
```

Shadow

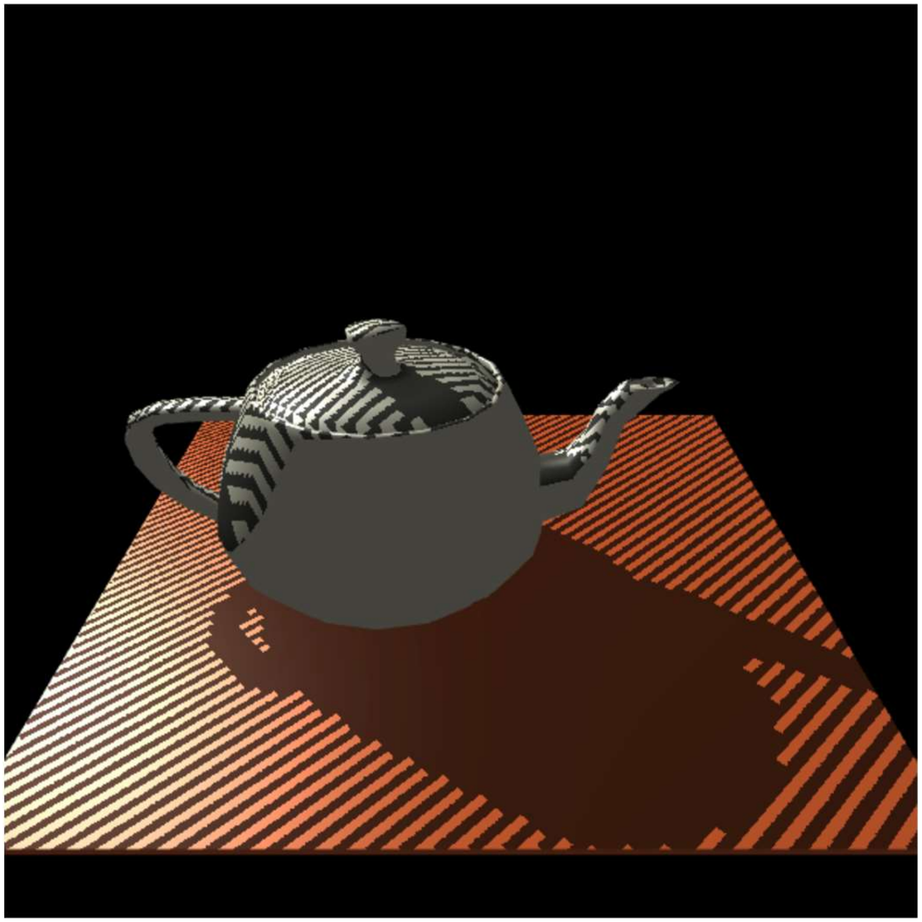


Two Problems:

- Shadow Acne
- “jaggy” Shadow

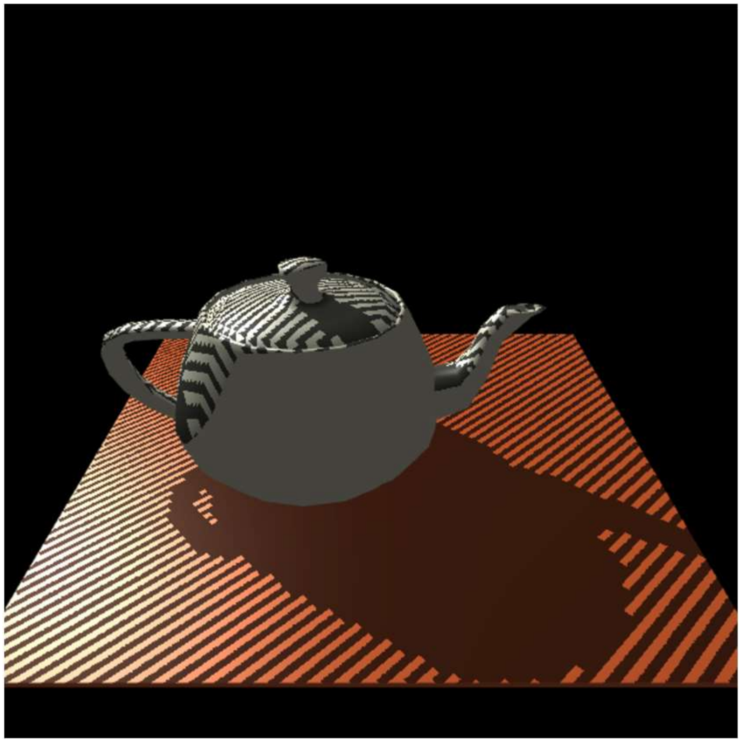
Shadow: Source of Problems

IITK CS360A

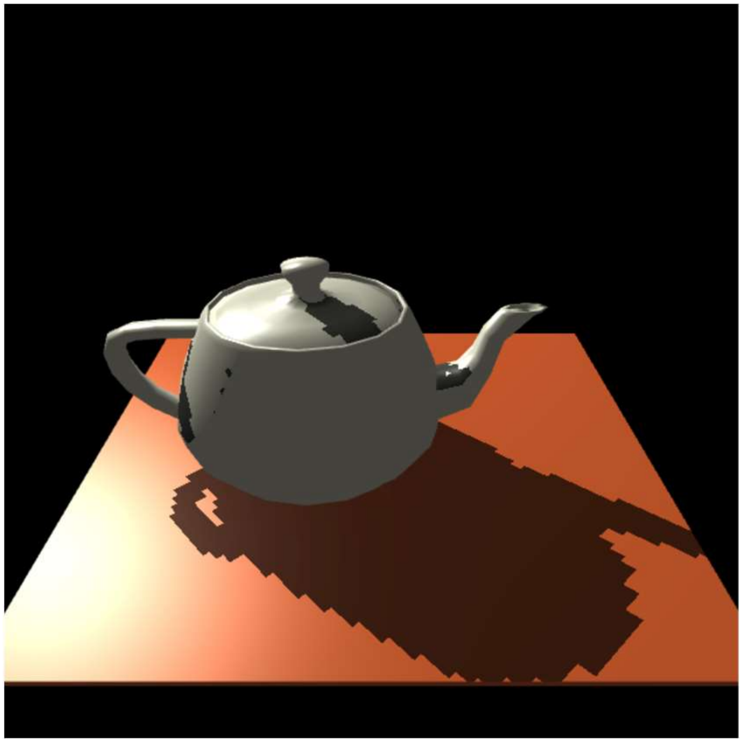


- Limited shadow map resolution and precision (as depth is stored using a 24bits per pixel)
- Values that are stored in the depth buffer will be quantized to the nearest precision
- One pixel of the shadow map will cover many pixels on the view

Shadow: Self Intersection: Shadow Acne

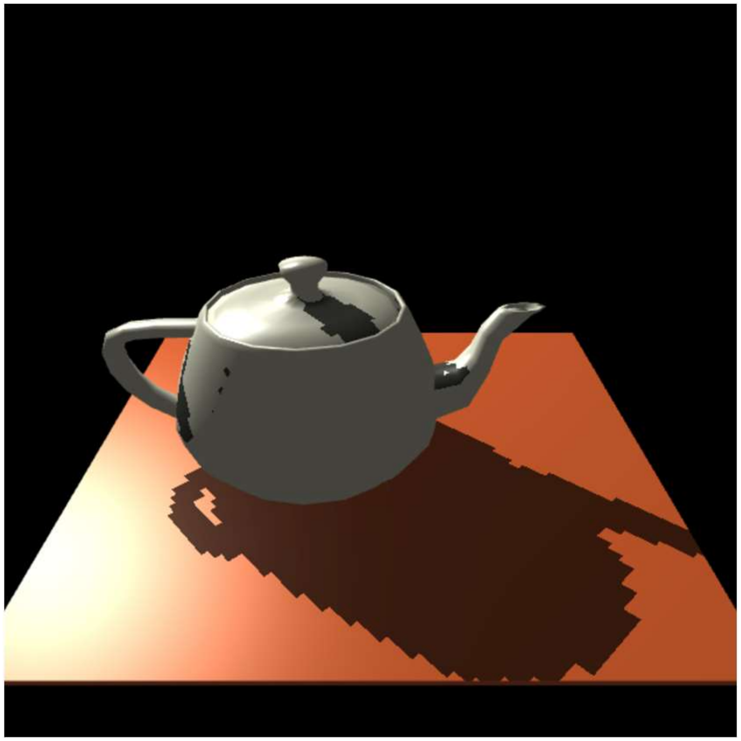


Add a small bias

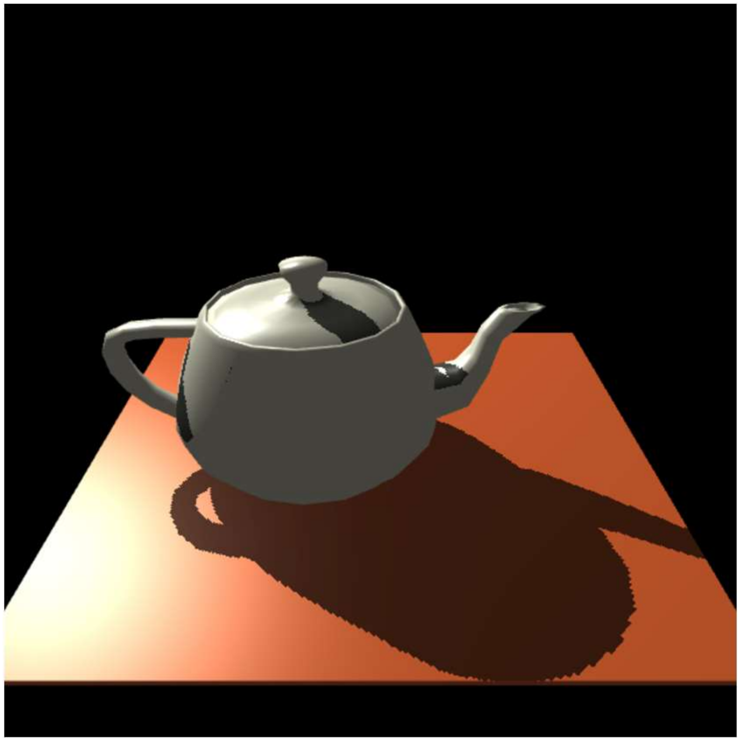


shadowFactor = currentDepth - selfIntersectionBias > closestDepth ? 0.3 : 1.0;

Jaggy Shadow: Resolution of Shadow Map

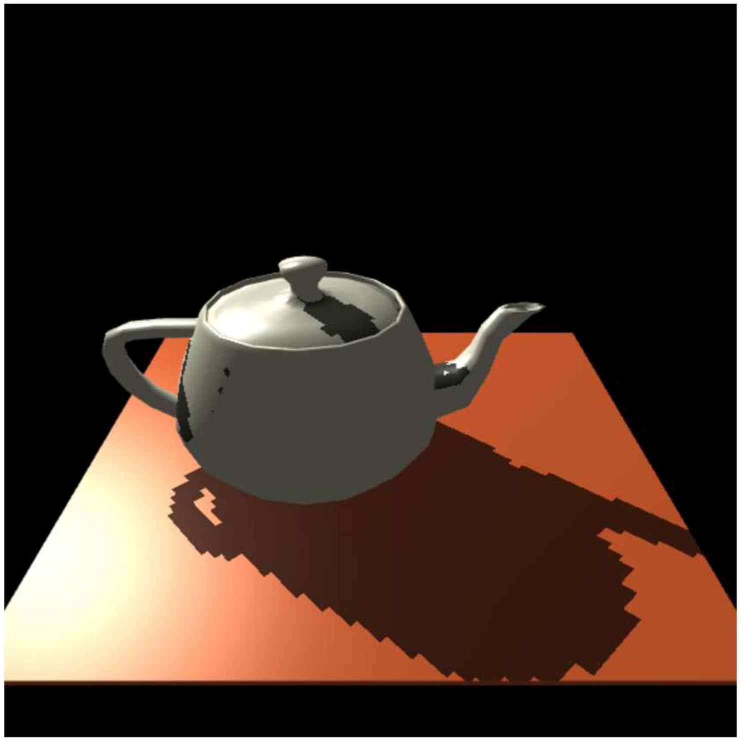


Depth texture resolution: 256x256

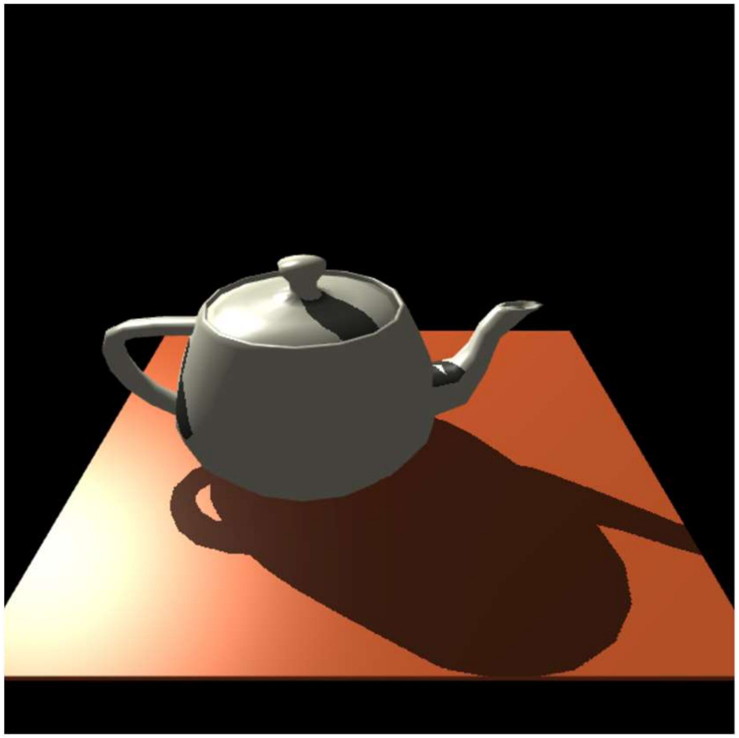


Depth texture resolution: 1024x1024

Jaggy Shadow: Resolution of Shadow Map

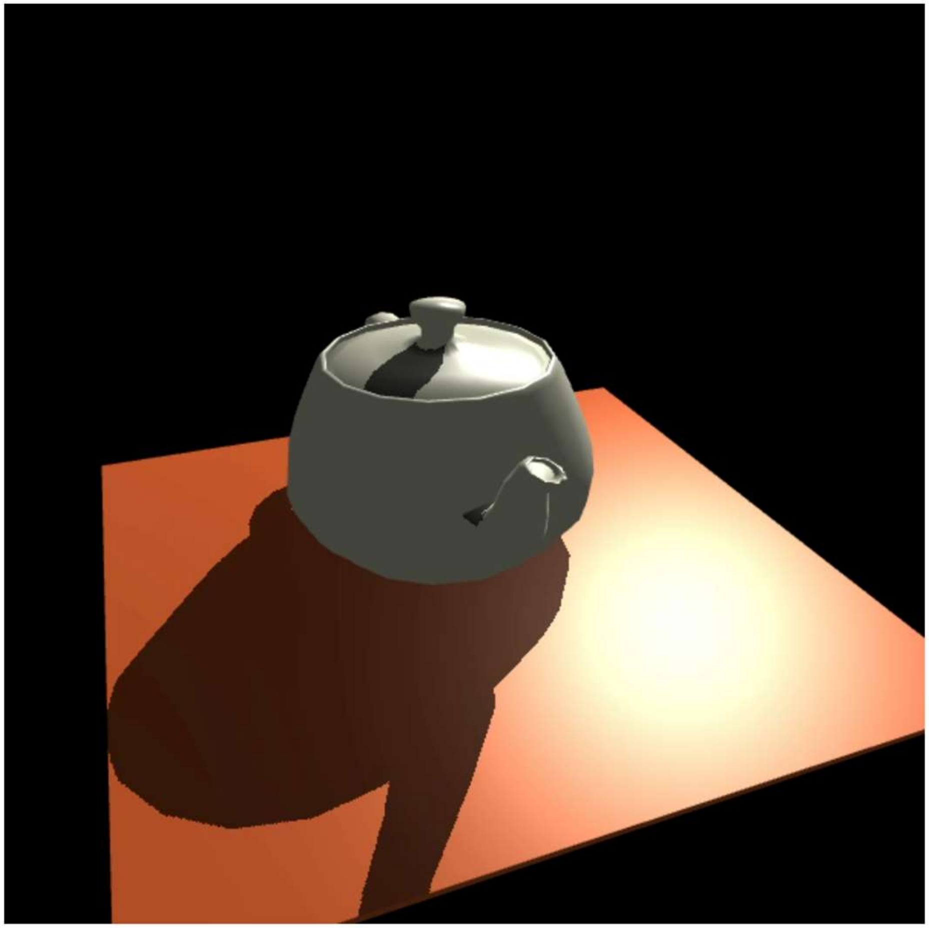


Depth texture resolution: 256x256



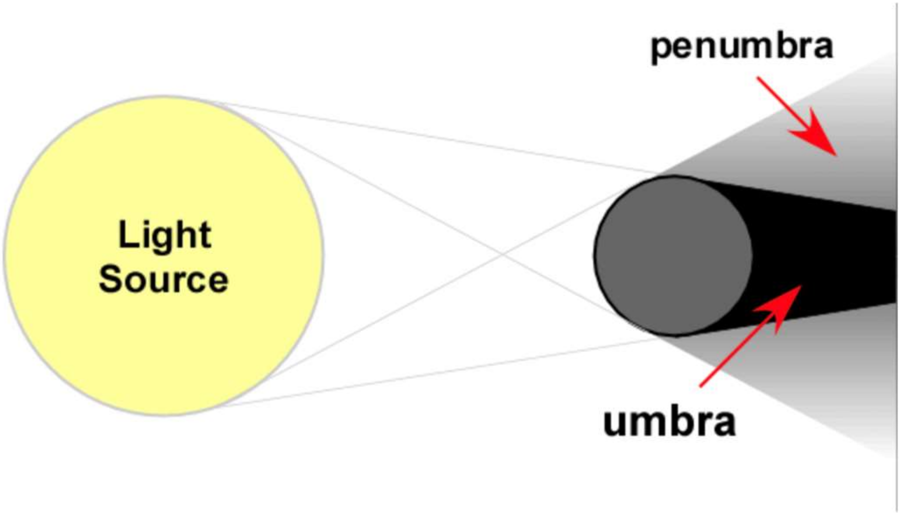
Depth texture resolution: 4096x4096

Shadow Animation



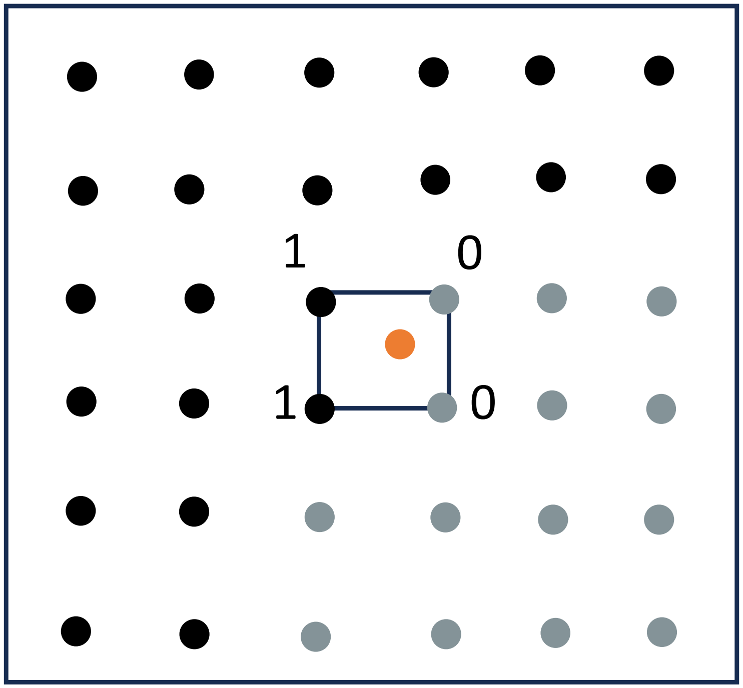
Soft Shadow: Percentage-Closer Filter (PCF)

- Instead of hard shadow classification, we want soft shadow to reduce the aliasing artifacts
- Instead of taking one sample to determine if a fragment is in the shadow or not, we take "several" and average them
- The result is a penumbra and umbra zone

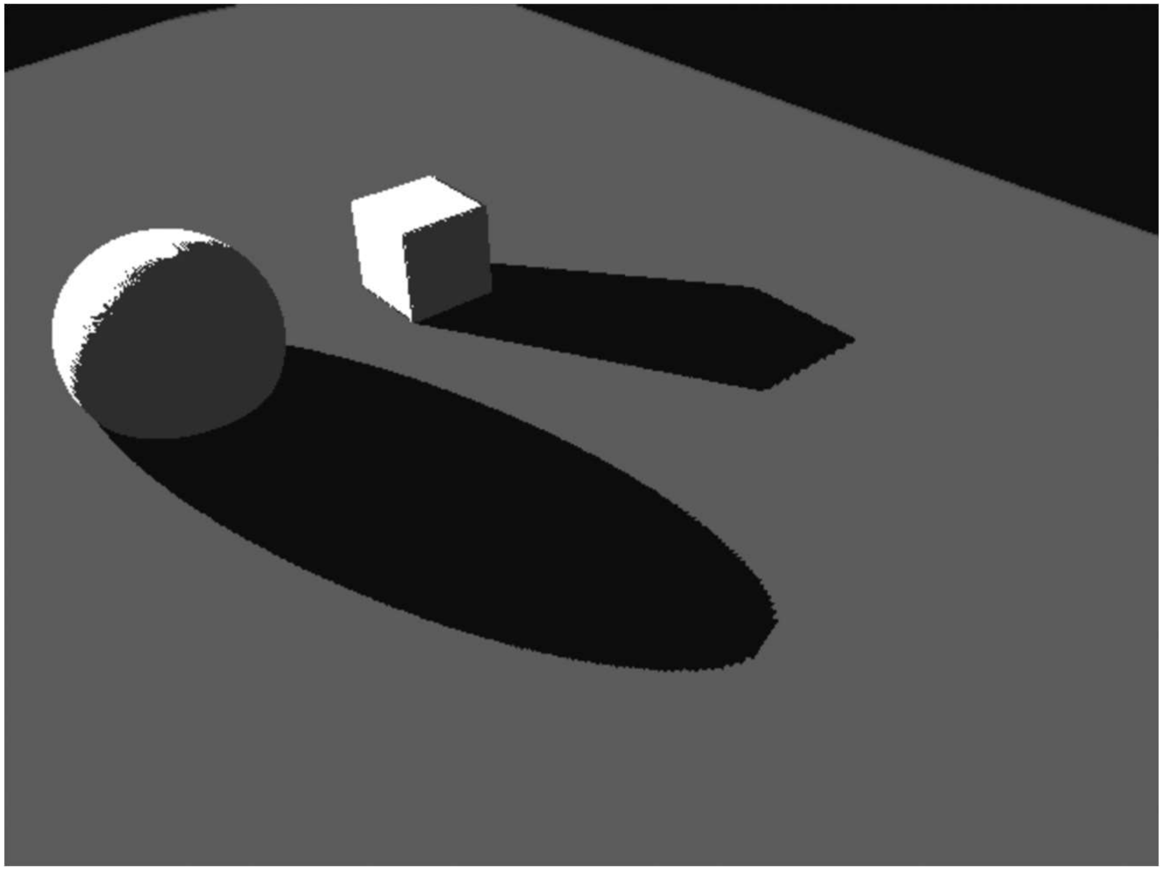


https://developer.nvidia.com

Soft Shadow: Percentage-Closer Filter (PCF)

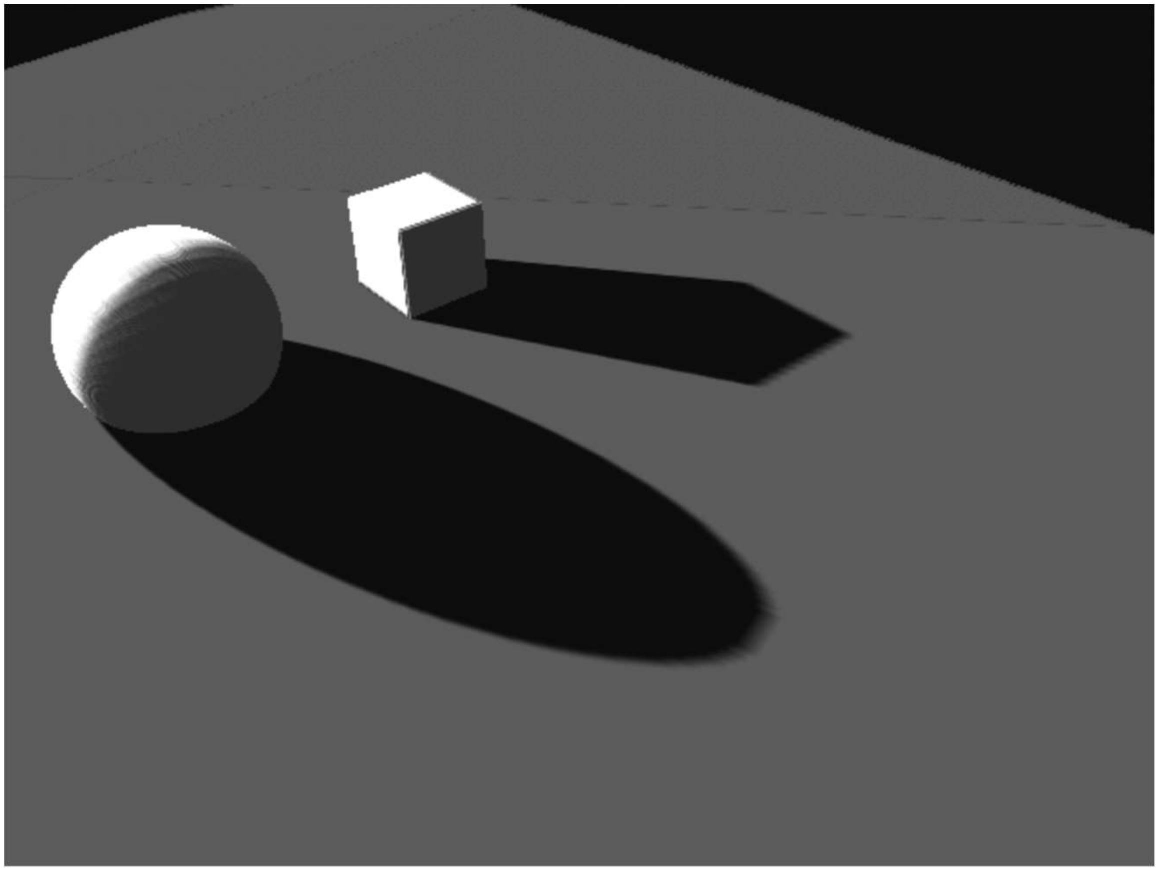


Soft Shadow: Percentage-Closer Filter (PCF)



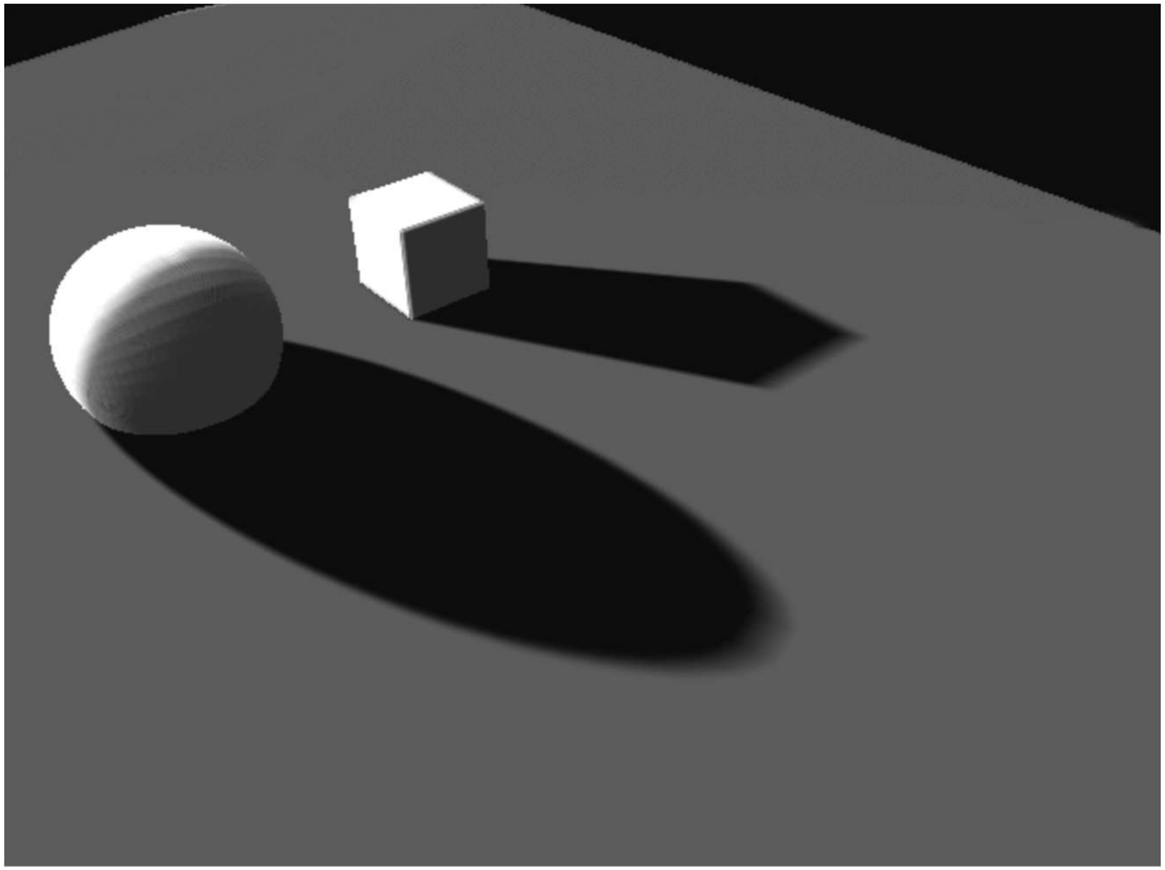
www.fabiensanglard.net

Soft Shadow: Percentage-Closer Filter (PCF)



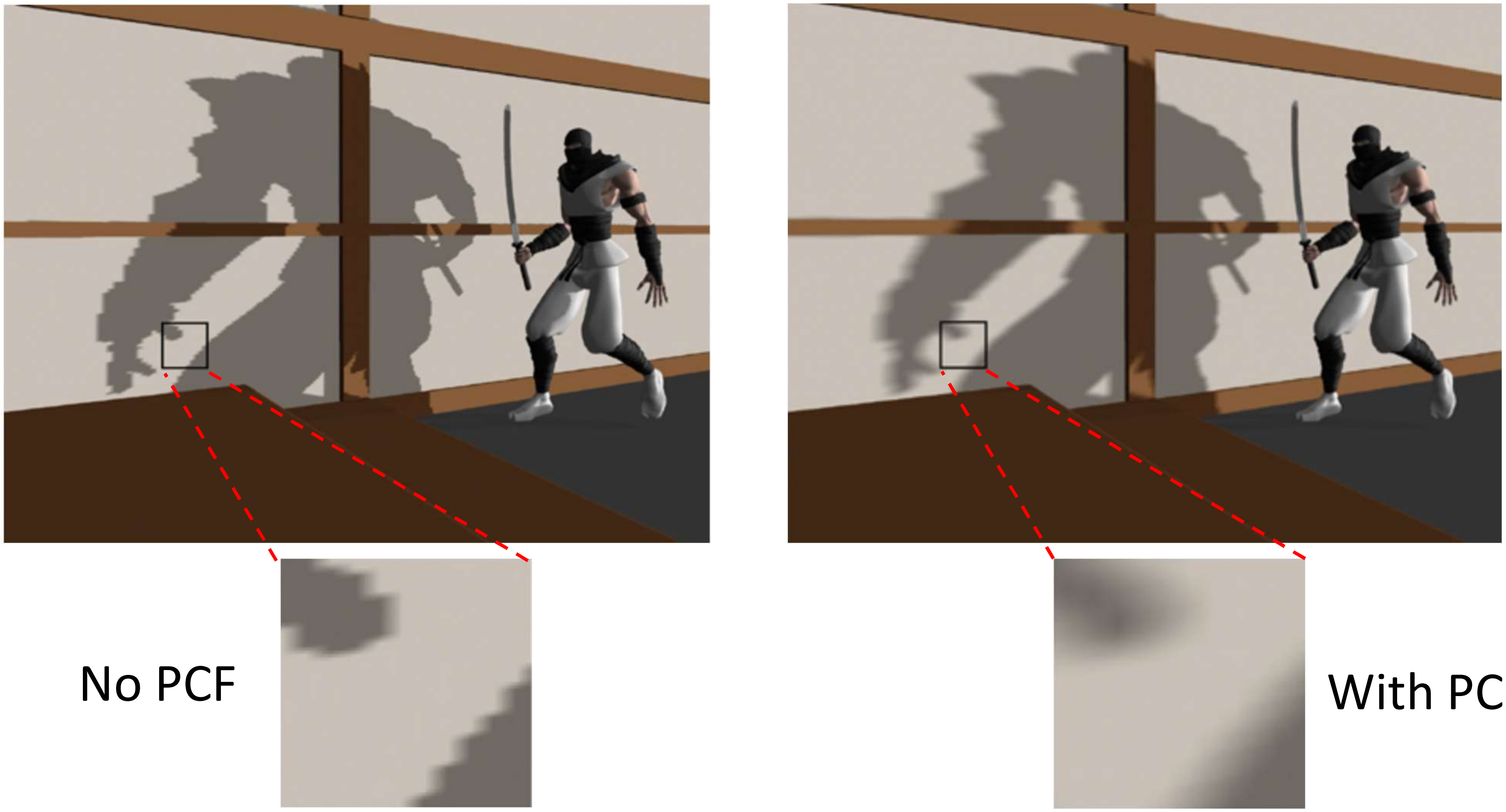
www.fabiensanglard.net

Soft Shadow: Percentage-Closer Filter (PCF)



www.fabiensanglard.net

Soft Shadow: Percentage-Closer Filter (PCF)



No PCF

With PCF

https://developer.nvidia.com