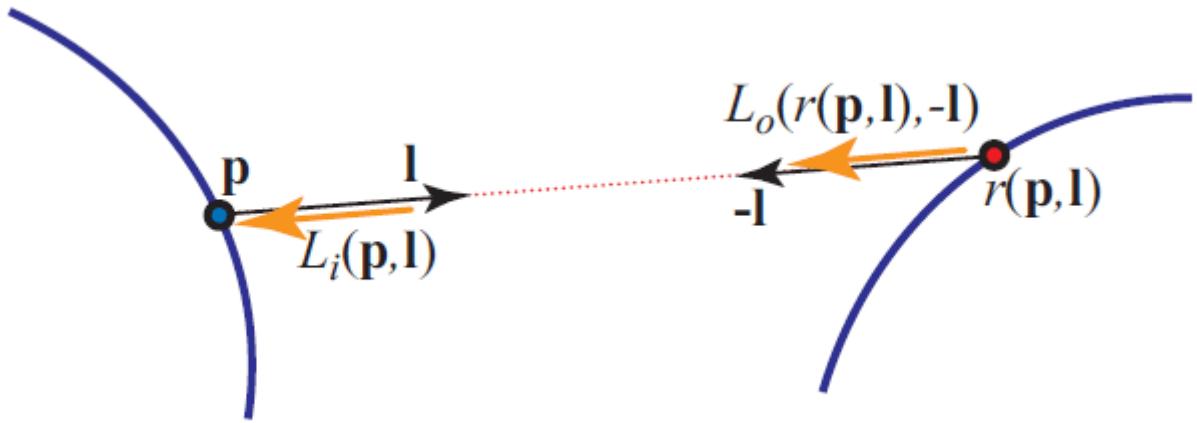
Shadow Rendering

CS7GV3 – Real-time Rendering

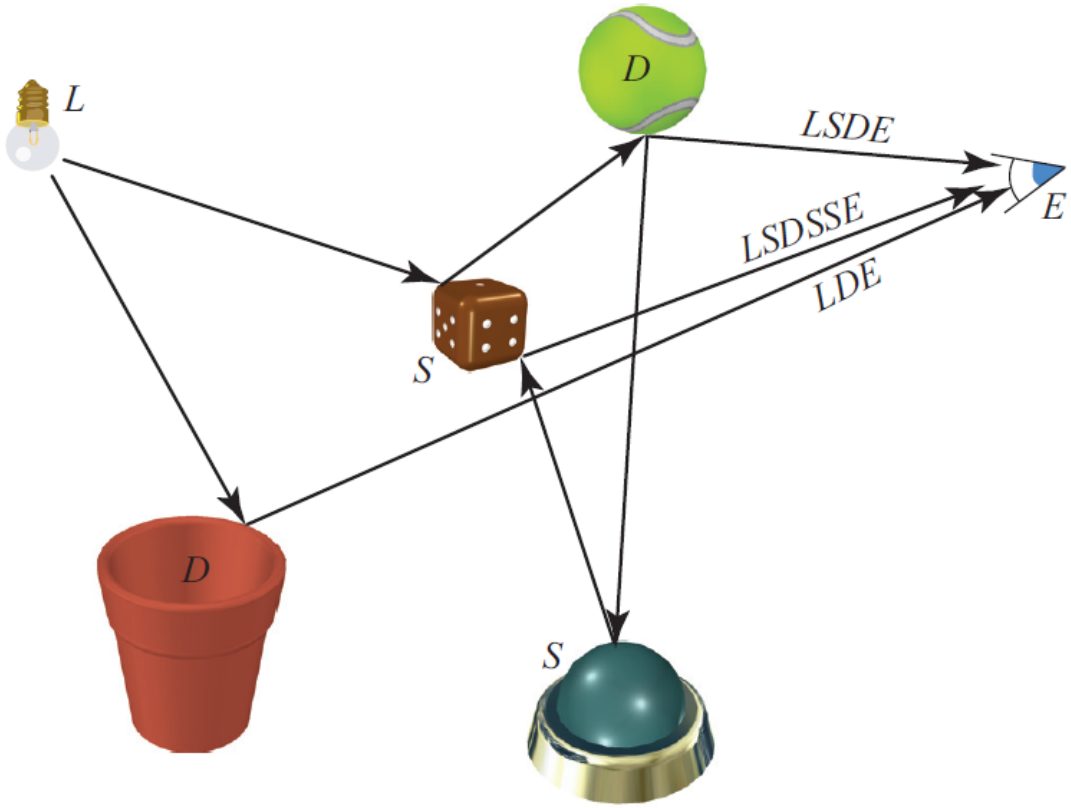
Global Illumination



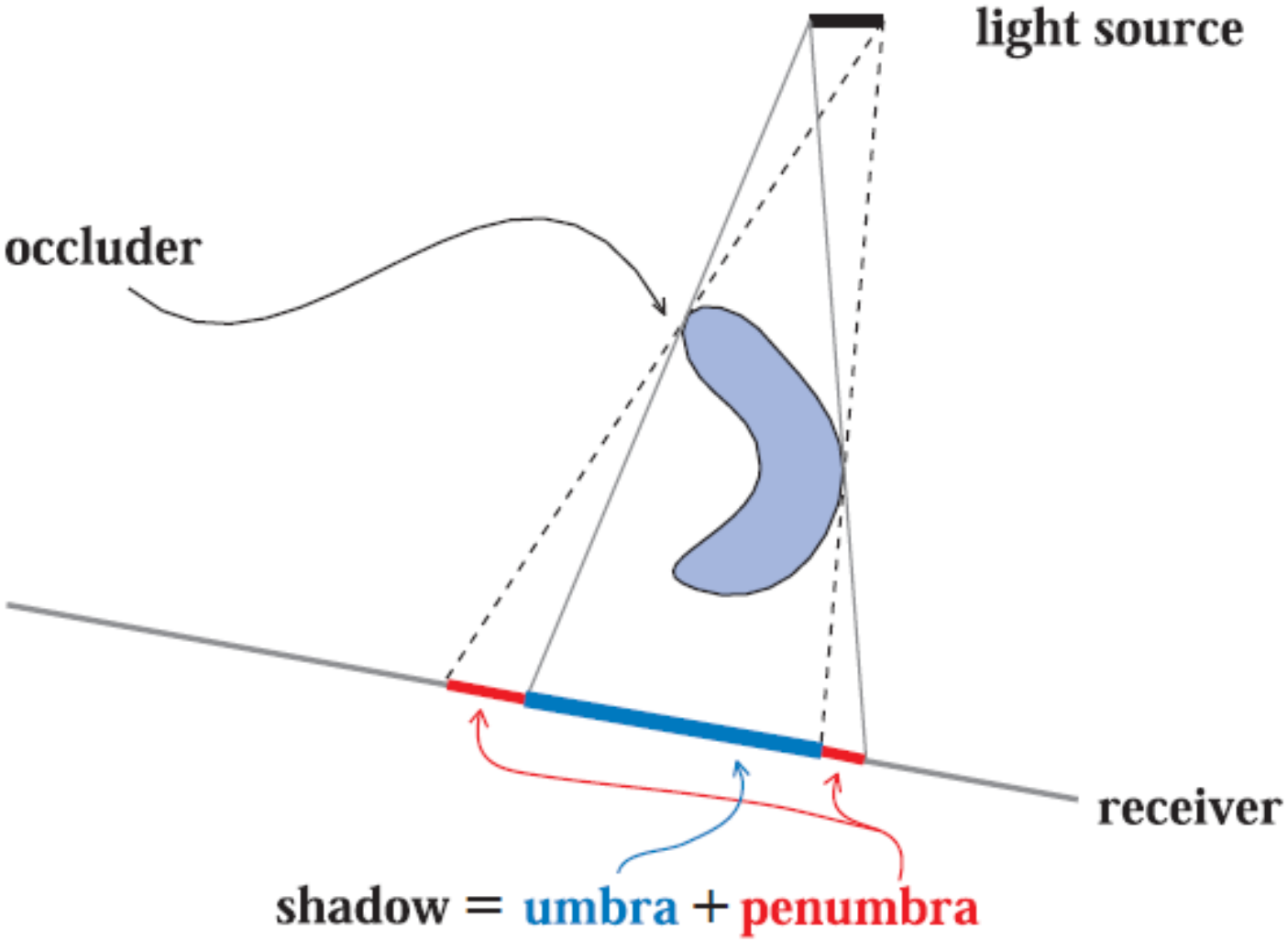
- The incoming radiance $L_i(\mathbf{p},\mathbf{l})$ at some point **p** is the outgoing $L_o(r(\mathbf{p},\mathbf{l})-\mathbf{l})$ from another point

A Recursive Term

- r(r(**p**,**l**), **l**’)
- r(r(r(**p**,**l**), **l**’),**l**’’)
- ad infinitum



Shadows



Shadows and Perception

Position Cues

Shadows and Perception

Position Cues

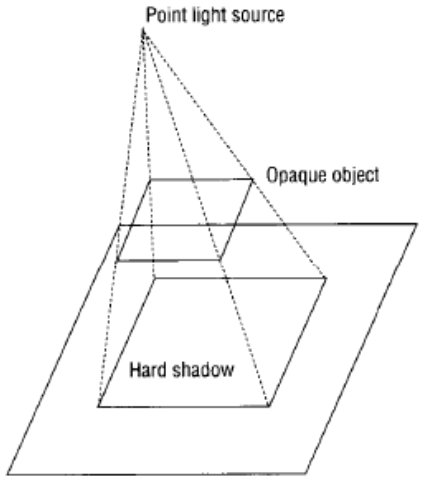
Shadows and Perception

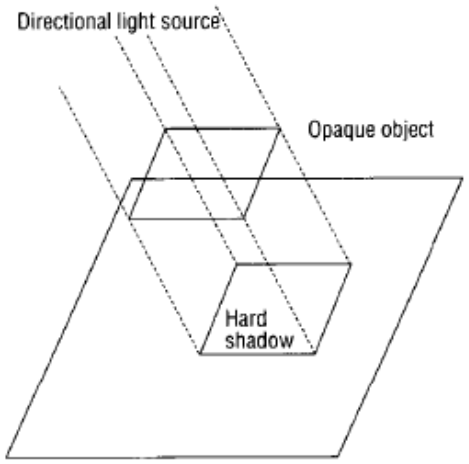
Geometric Information

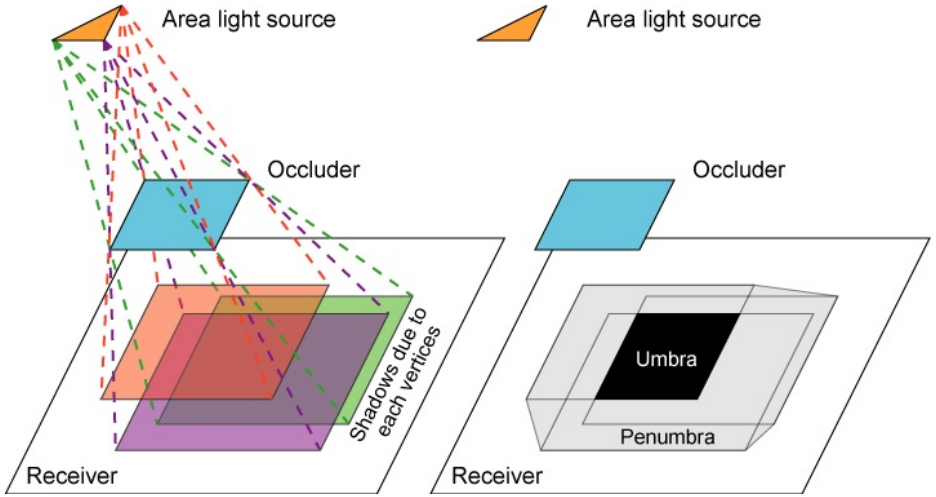
Shadows and Perception

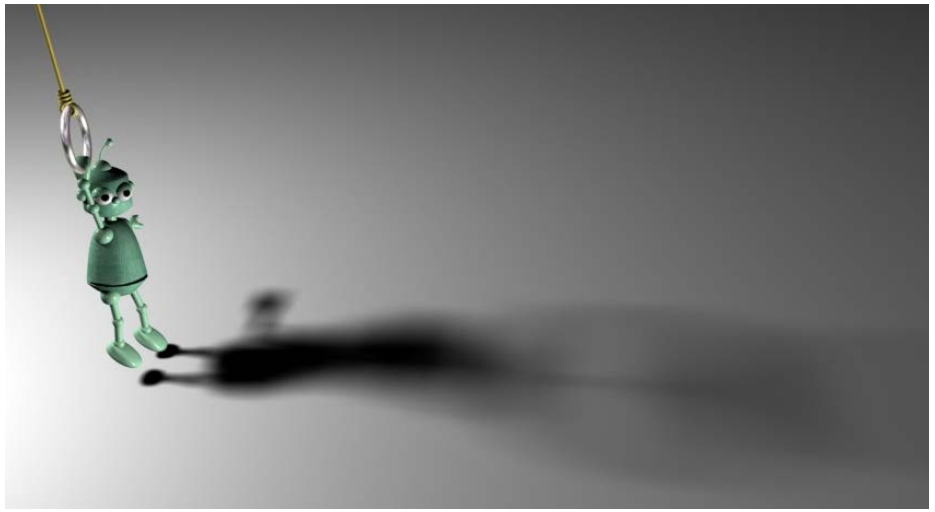
Geometric Information

Shadows and Perception

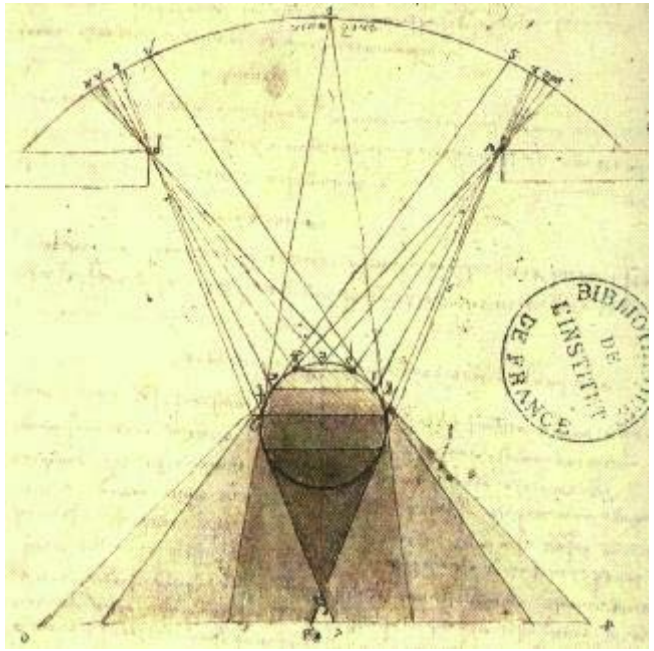




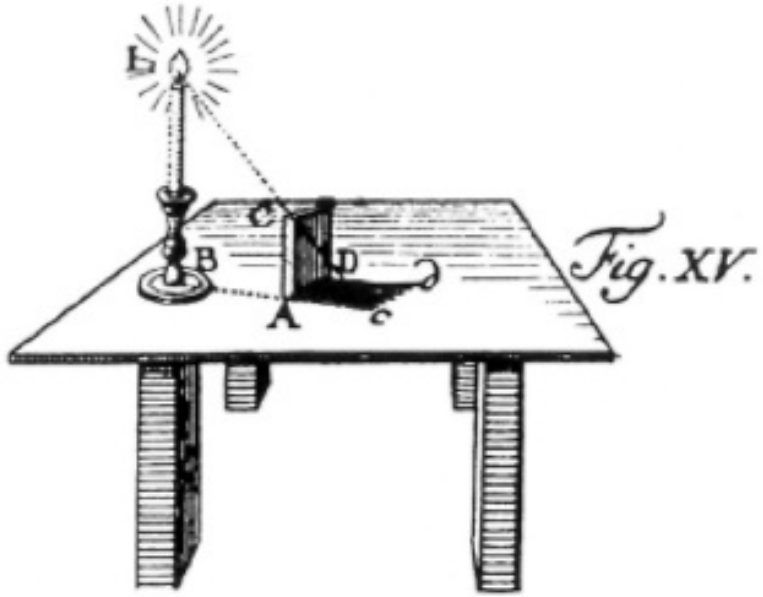




Shadow Mapping History



Leonardo Da Vinci.
Codex Urbinas. 1490.



Johann Heinrich Lambert.
Die freye Perspektive. 1759.

Seminal Papers

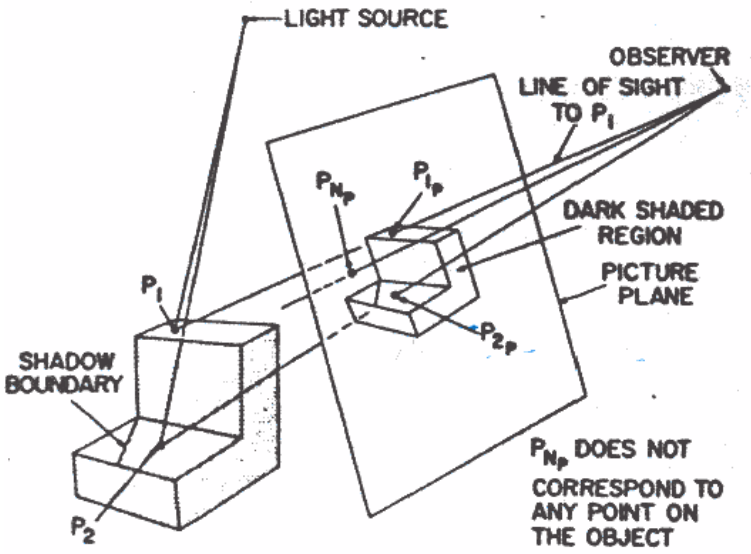
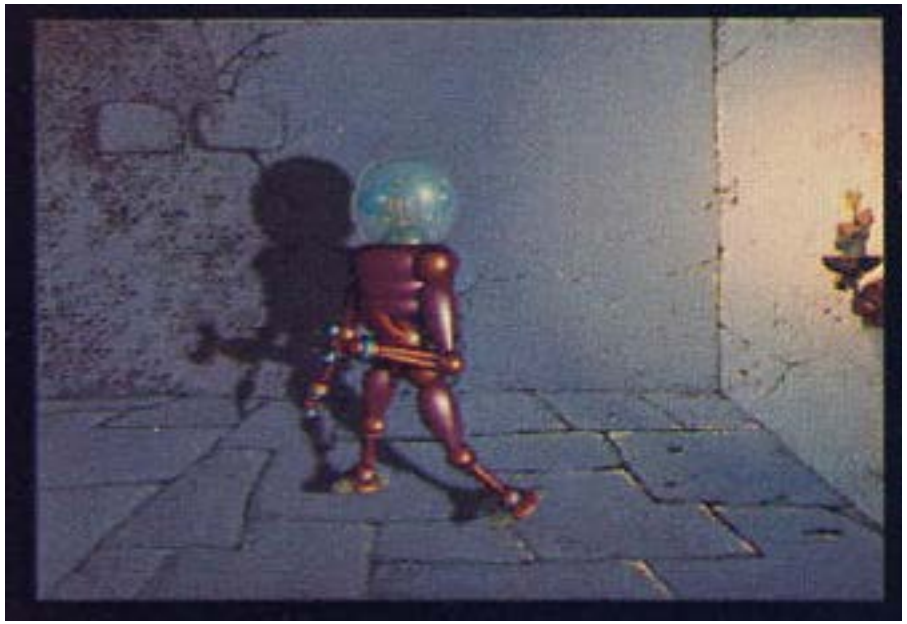


Figure 6 – Point by point shading

Shadow Ray-casting Appel 1966

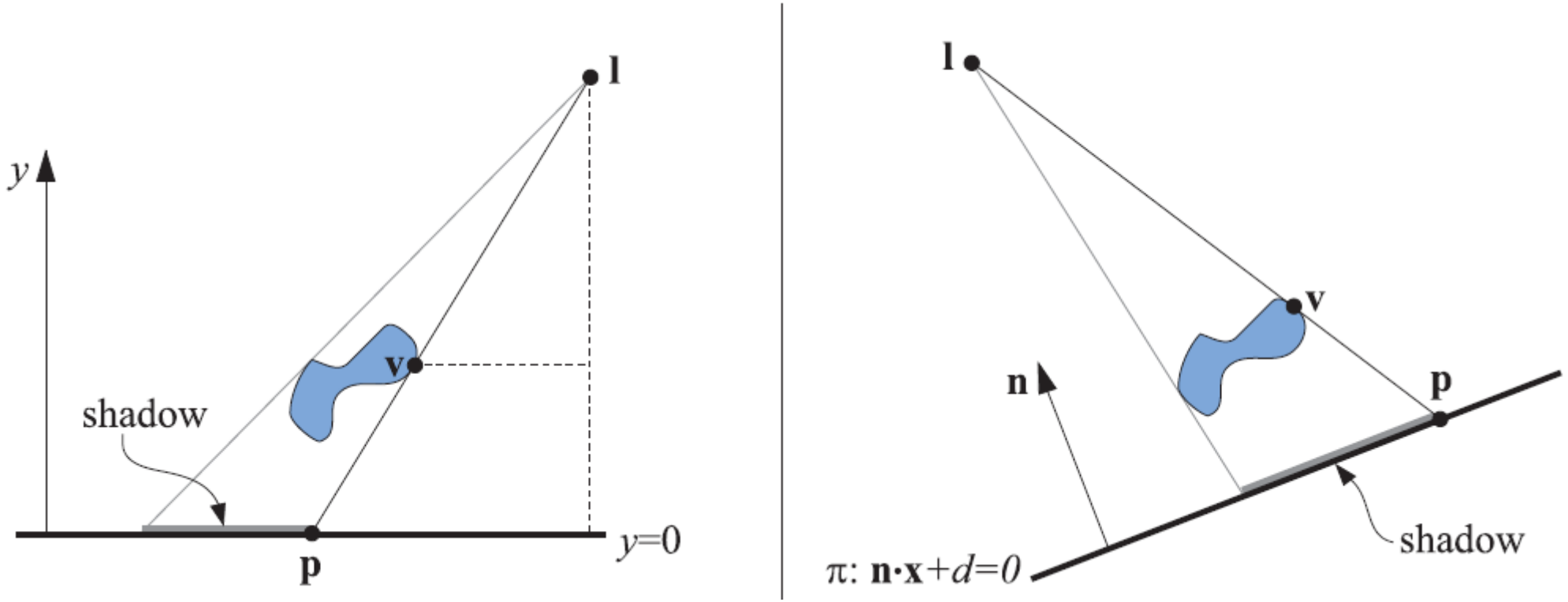


Williams 1978 – zbuffer shadows

Real-time Shadows

- Projective Shadows
  - Trivial for planar surface: project all vertices of blocker onto plane of receiver, draw in black
- Shadow Volumes
  - Test points to see if inside space occluded by blocker
- Shadow Maps
  - We look at these in more detail
- Ray Traced Shadows
- PRT (lightmaps)
- Pre-compute the lighting and bake into textures

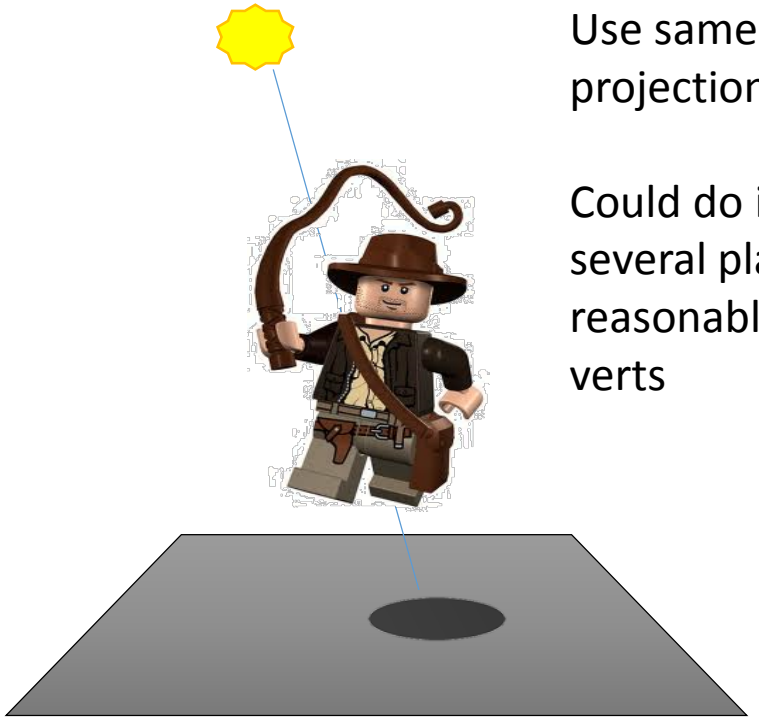
Projective Shadows



We project all vertices onto the receiver and draw them in black

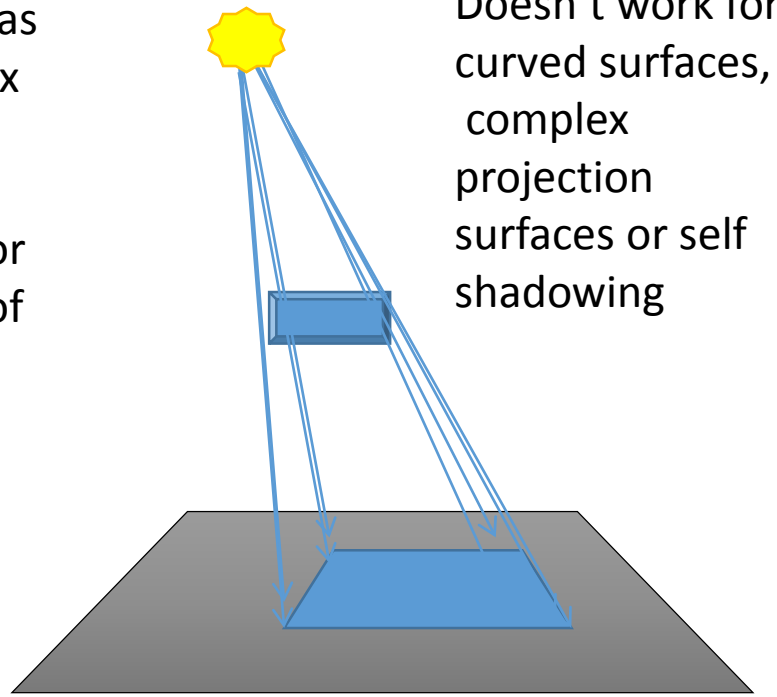
Projective Shadows

- Even highly inaccurate shadow provide useful cues
  - Extend lines from point light through centre of object, draw ellipse where this hits ground plane
  - Alternatively extend lines through from light each vertex



Use same math as projection matrix

Could do it for several planes for reasonable no. of verts



Doesn't work for curved surfaces, complex projection surfaces or self shadowing

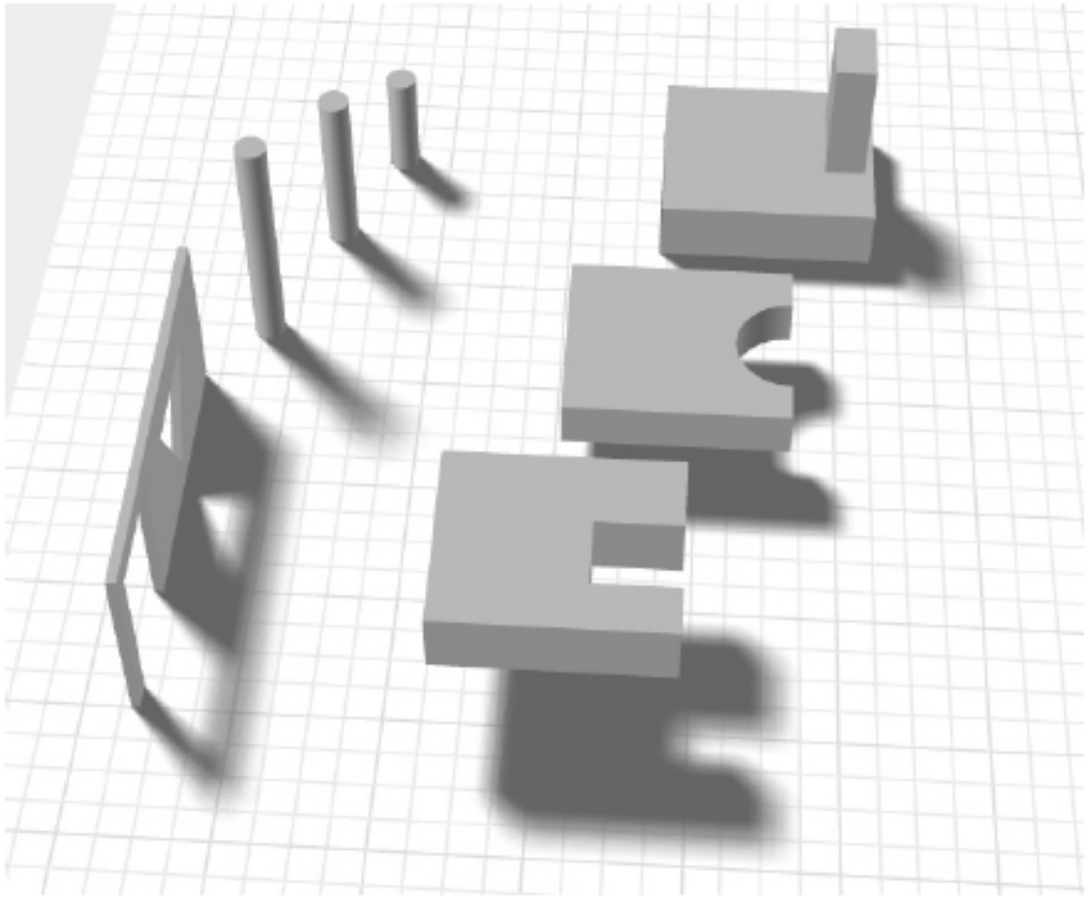
Shadow Mapping

- Introduced by Williams in 1978
- Casting Curved Shadows onto Curved Surfaces
  - Image space shadowing technique
- Used in Renderman (Toy Story)
- Scales to an arbitrary number of objects
- Can be extended to soft shadows
- Handles self-shadowing

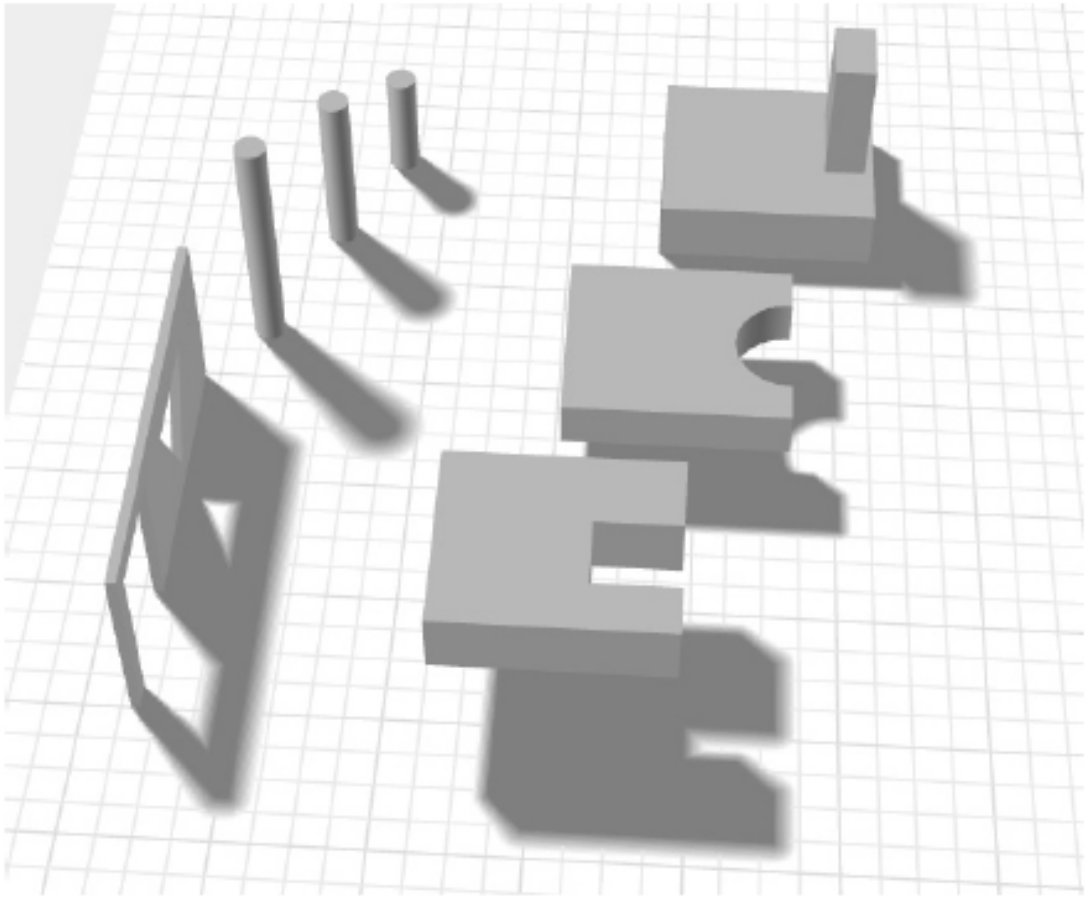
No Self-shadowing

Self-shadowing

Soft Shadows



Heckberts and Herf’s methode (256 passes)

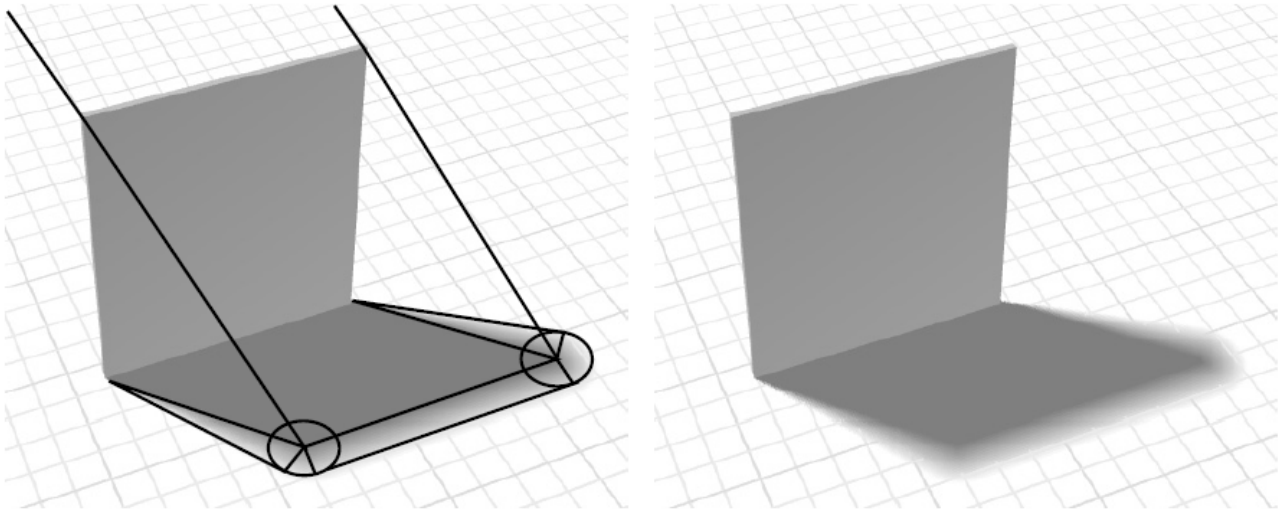


Haines’ methode (one pass)

Soft Shadows

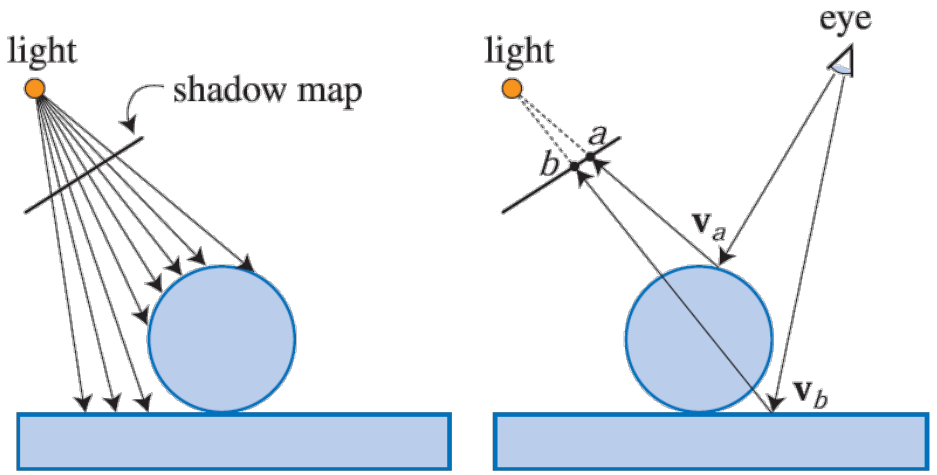
- Heckberts and Herf’s methode (256 passes)

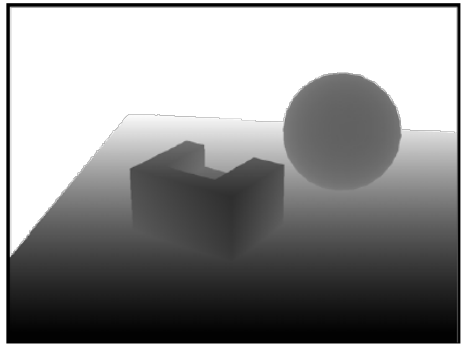
- Haines’ methode (one pass)



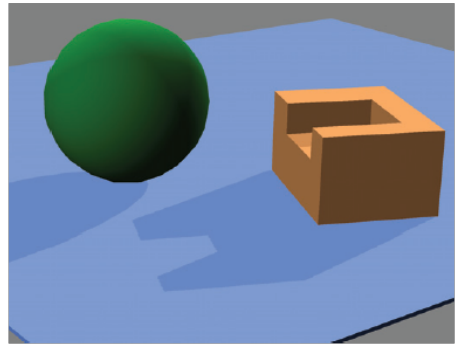
Shadow Mapping

- Exploits image space shadow buffer
  - Render scene from point of view of light
  - For each visible point (pixel in image plane)
    - Project into *light space, determine distance to light*
    - Compare with stored value in depth map
    - If depth > stored depth then in shadow





Depth Image from Light Viewpoint

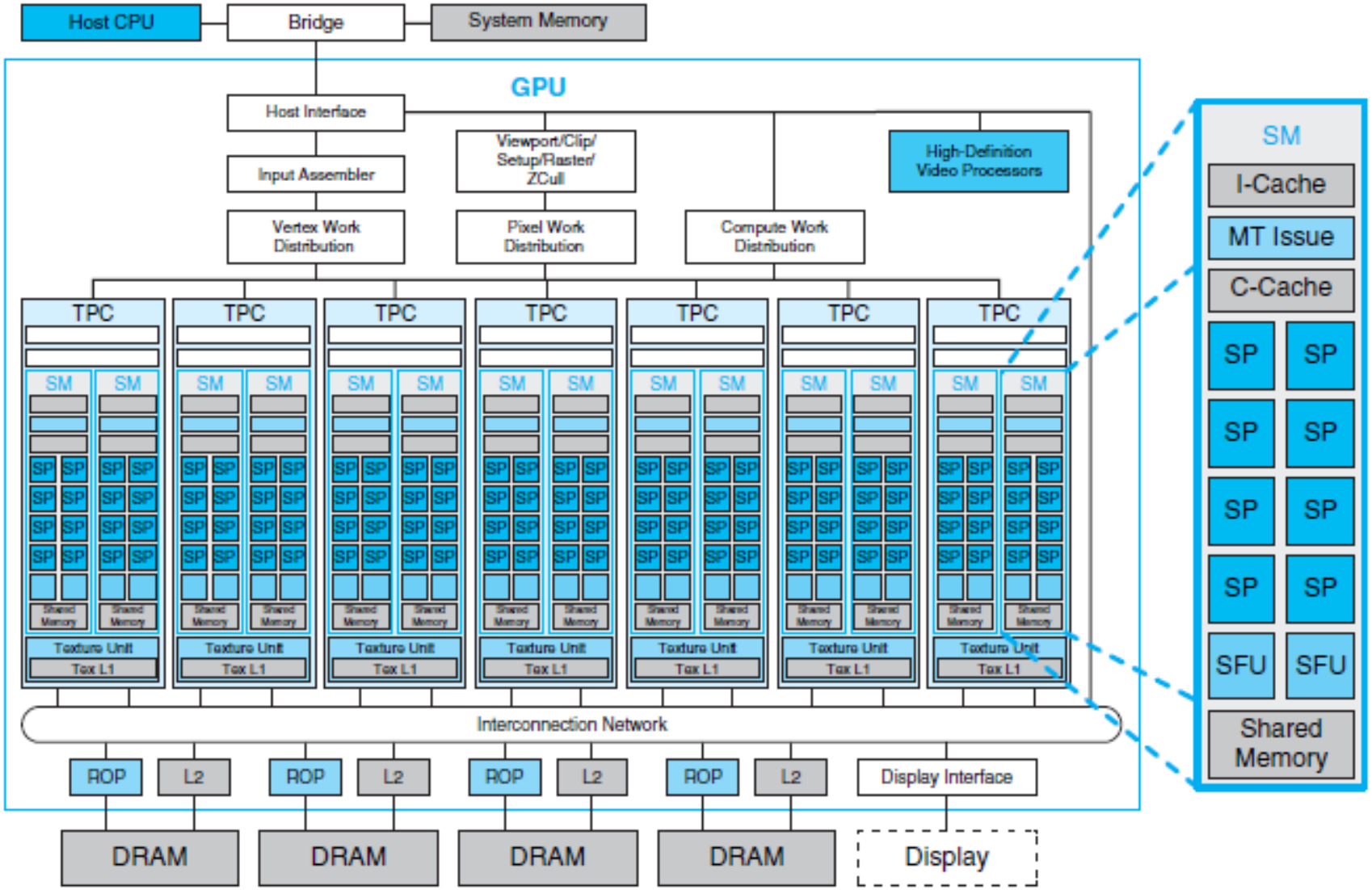


Colour Image from Eye Viewpoint

Creating a Shadow Map

- Multipass technique
- By rendering from the light position we see all point visiable from that position
- There are no shadows
- Points not seen from the light will not be rendered if the depht test fails
- In the 2nd pass we project the surface coordinates into the light’s refernec frame
  - Then compair there depth

GeForce 8800 (2006) but didn't change much



1st Step

- We need to create a depth texture
- Attach to a framebuffer object
- Important:
  - The texture comparison mode allows us to compair between a reference value and a value stored in the texture
  - This is performed by the texture hardware and not the shader

Creating a Framebuffer Object with a Depth Attached

```
// Create a depth texture
   glGenTextures(1, &depth_texture);
   glBindTexture(GL_TEXTURE_2D, depth_texture);
// Allocate storage for the texture data
   glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32,
                DEPTH_TEXTURE_SIZE, DEPTH_TEXTURE_SIZE,
                0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
// Set the default filtering modes
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// Set up deph comparison mode
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
                   GL_COMPARE_REF_TO_TEXTURE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
```

Creating a Framebuffer Object with a Depth Attached

```
//Set up wrapping modes
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
   glBindTexture(GL_TEXTURE_2D, 0);
// Create FBO to render depth into
   glGenFramebuffers(1, &depth_fbo);
   glBindFramebuffer(GL_FRAMEBUFFER, depth_fbo);
// Attach the depth texture to it
   glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT,
                        depth_texture, 0);
// Disable color rendering as there are color attachments
   glDrawBuffer(GL_NONE);
```

2nd Step

- We need to setting up the Matrices for Shadow Map Generation

Setting up the Matrices for Shadow Map Generation

```
// Time varying light position
   vec3 light_position = vec3(sinf(t * 6.0f * 3.141592f) * 300.0f, 200.0f,
                                    cosf(t * 4.0f * 3.141592f) * 100.0f + 250.0f);
// Matrices for rendering the scene
   mat4 scene_model_matrix = rotate(t * 720.0f, Y);
// Matrices used when rendering from the light's position
   mat4 light_view_matrix = lookat(light_position, vec3(0.0f), Y);
   mat4 light_projection_matrix(frustum(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, FRUSTUM_DEPTH));
// Now we render from the light's position into the depth buffer.
// Select the appropriate program
    glUseProgram(render_light_prog);
    glUniformMatrix4fv(render_light_uniforms.model_view_projection_matrix,
                       1, GL_FALSE, light_projection_matrix * light_view_matrix *
                       scene_model_matrix);
```

3rd Step

- Setting up shaders that generate the depth buffer from the ligth’s position

Simple Vertex Shader for Shadow Map Generation

```
#version 330

uniform mat4 model_view_projection_matrix;

layout (location = 0) in vec4 position;

void main(void)
{
    gl_Position = model_view_projection_matrix * position;
}
```

Simple Fragment Shader for Shadow Map Generation

```
#version 330

layout (location = 0) out vec4 color;

void main(void)
{
    color = vec4(1.0);
}
```

4th Step

- Now we render the scene from the light's view
- Notice:
  - Polygon offset is used to prevent *depth fighting*

Rendering the Scene from the light’s point of view

```
// Bind the 'depth only' FBO and set the viewport to the size of the depth texture
    glBindFramebuffer(GL_FRAMEBUFFER, depth_fbo);
    glViewport(0, 0, DEPTH_TEXTURE_SIZE, DEPTH_TEXTURE_SIZE);

// Clear
    glClearDepth(1.0f);
    glClear(GL_DEPTH_BUFFER_BIT);

// Enable polygon offset to resolve depth-fighting isuses
    glEnable(GL_POLYGON_OFFSET_FILL);
    glPolygonOffset(2.0f, 4.0f);

// Draw from the light's point of view
    DrawScene(true);
    glDisable(GL_POLYGON_OFFSET_FILL);
```

5th Step

- After rendering the scene from the light's point of view
  - In order to calcuate the depth
- We render the scene with regular shaders
- Important:
  - Shadow matrix
  - Transforms world coordinates into the light's projective space

Matrix Calculation for Shadow Map Rendering

```
// Matrices for rendering the scene
   mat4 scene_model_matrix = rotate(t * 720.0f, Y);
   mat4 scene_view_matrix = translate(0.0f, 0.0f, -300.0f);
   mat4 scene_projection_matrix = frustum(-1.0f, 1.0f, -aspect, aspect,
                                          1.0f, FRUSTUM_DEPTH);
   const mat4 scale_bias_matrix = mat4(vec4(0.5f, 0.0f, 0.0f, 0.0f),
                                       vec4(0.0f, 0.5f, 0.0f, 0.0f),
                                       vec4(0.0f, 0.0f, 0.5f, 0.0f),
                                       vec4(0.5f, 0.5f, 0.5f, 1.0f));
 mat4 shadow_matrix = scale_bias_matrix *
                      light_projection _matrix *
                      Light_view_matrix;
```

6th Step

- In this step the vertex shader will apply these matrices to incoming vertices

Vertex Shader for Rendering from Shadow Maps

```
#version 330

uniform mat4 model_matrix;
uniform mat4 view  matrix;
uniform mat4 projection  matrix;
uniform mat4 shadow_matrix;

layout (location = 0) in vec4 position;
layout (location = 1) in vec3 normal;

out VS_FS_INTERFACE

{
   vec4 shadow_coord;
   vec3 world_coord;
    vec3 eye  coord;
    vec3 normal;
} vertex;
```

Vertex Shader for Rendering from Shadow Maps

```
void main(void)
{
   vec4 world_pos = model_matrix * position;
   vec4 eye_pos = view_matrix * world_pos;
   vec4 clip_pos = projection_matrix * eye_pos;

   vertex.world_coord = world_pos.xyz;
   vertex.eye_coord = eye_pos.xyz;
   vertex.shadow_coord = shadow_matrix * world_pos;
   vertex.normal = mat3(view_matrix * model_matrix) * normal;

   gl_Position = clip_pos;
}
```

7th Step

- The Fragment Shader will perform the lighting calculation
- Important:
  - uniform sampler2DShadow depth_texture;
  - float f = textureProj(depth_texture, fragment.shadow_coord);
  - This is where the comparison takes place

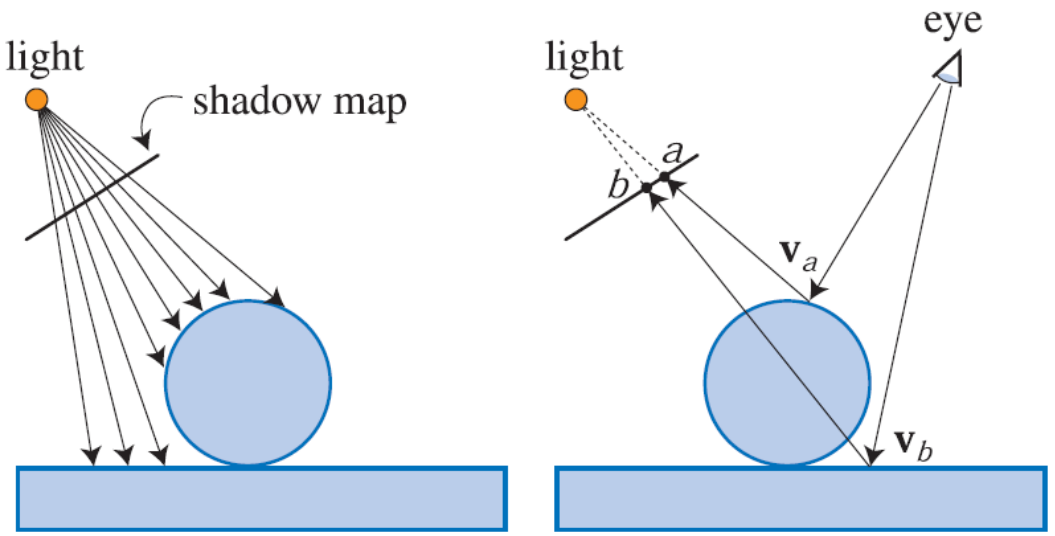
Fragment Shader for Rendering from Shadow Maps

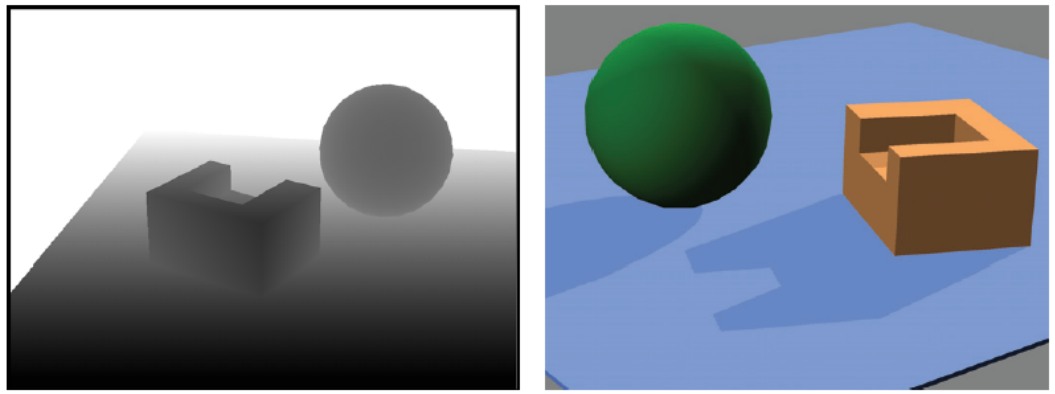
```
#version 330

uniform sampler2DShadow depth_texture;
uniform vec3 light_position;

uniform vec3 material_ambient;
uniform vec3 material_diffuse;
uniform vec3 material_specular;
uniform float material_specular_power;

layout (location = 0) out vec4 color;

in VS_FS_INTERFACE
{
    vec4 shadow_coord;
    vec3 world_coord;
    vec3 eye_coord;
    vec3 normal;
} fragment;
```

Fragment Shader for Rendering from Shadow Maps

```
void main(void)
{
   vec3 N = fragment.normal;
   vec3 L = normalize(light_position - fragment.world_coord);
   float LdotN = dot(N, L);
   vec3 R = reflect(-L, N);

   float diffuse = max(LdotN, 0.0);
   float specular = max(pow(dot(normalize(-fragment.eye_coord), R),
                            material_specular_power), 0.0);

   float f = textureProj(depth_texture, fragment.shadow_coord);

   color = vec4(material_ambient + f * (material_diffuse * diffuse +
                                        material_specular * specular), 1.0);
}
```

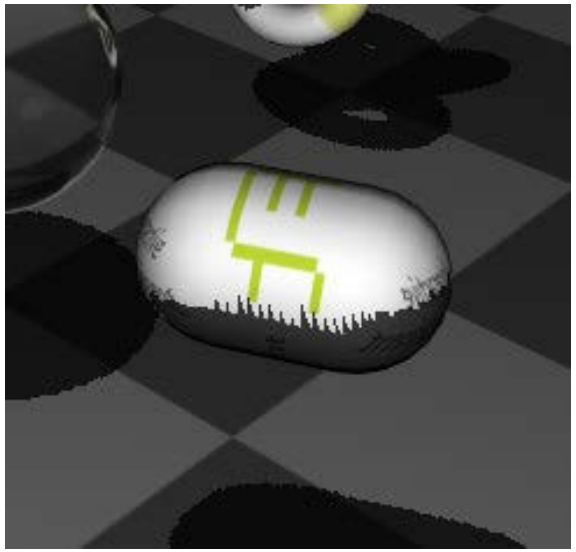
Shadow Map

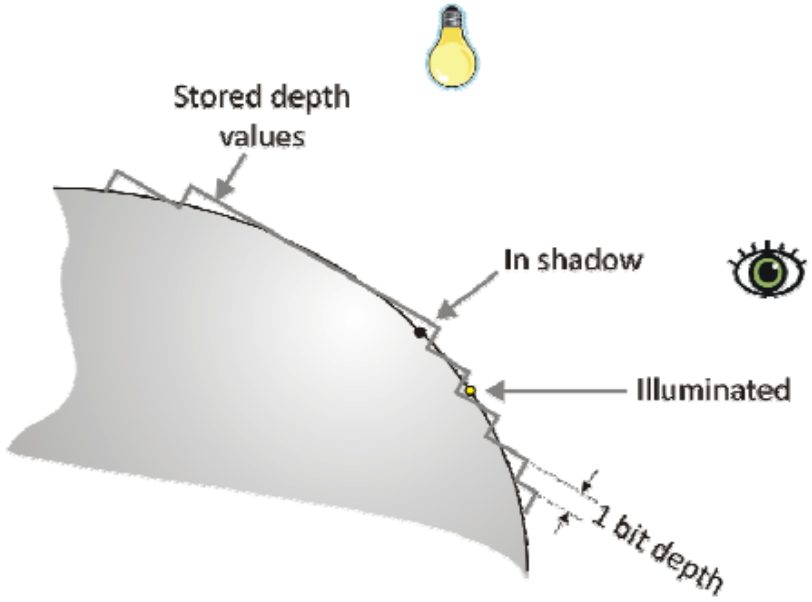




Shadow Map Precision

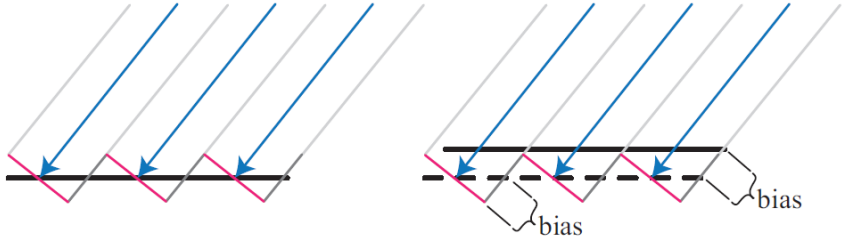
- Shadow map has fixed precision = bit depth
  - Limits accuracy of depth test
  - Leads to problems e.g. Surface acne

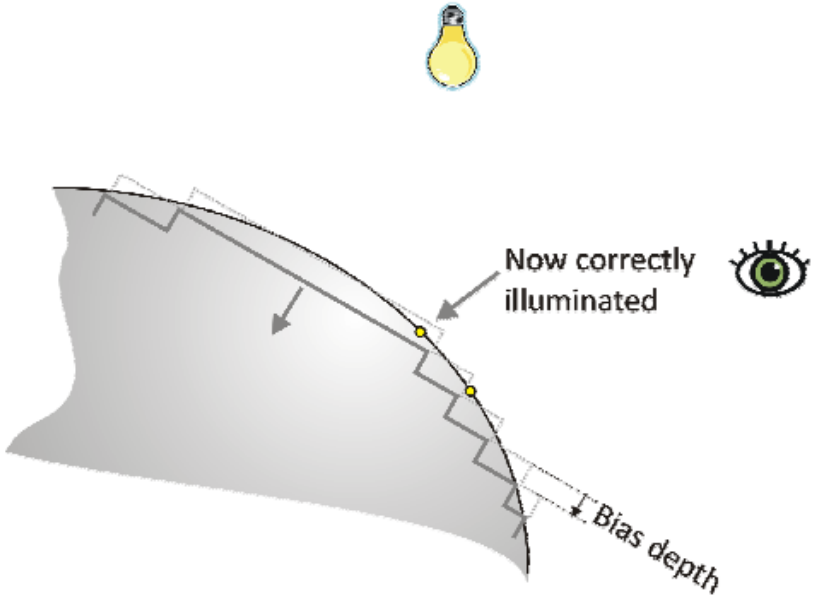




Shadow Map Bias

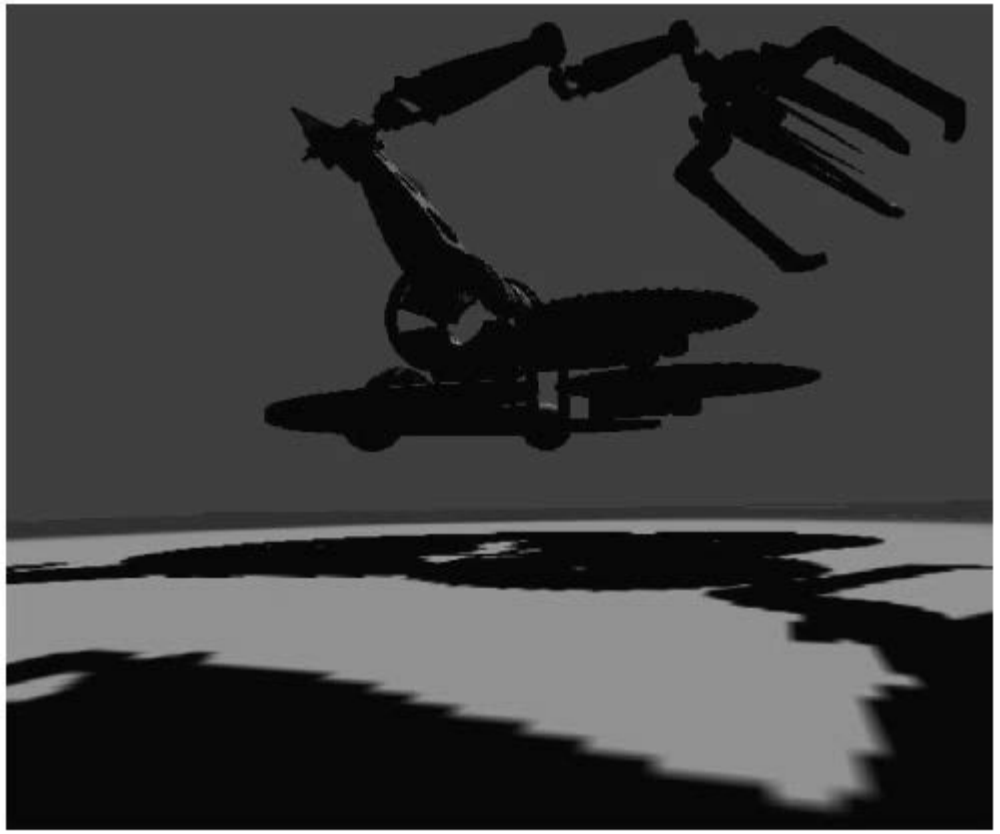
- One “hack” is to bias the depth



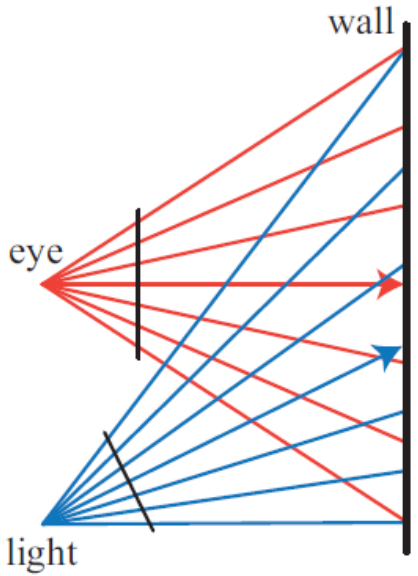


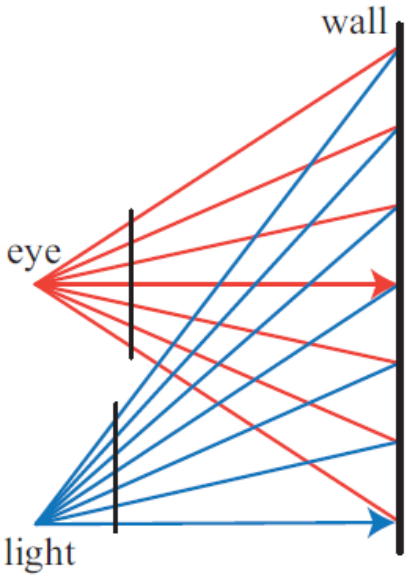
  - Shift shadow map depth a little bit
  - Just moves the problem but is still useful
  - We did this in the previous example

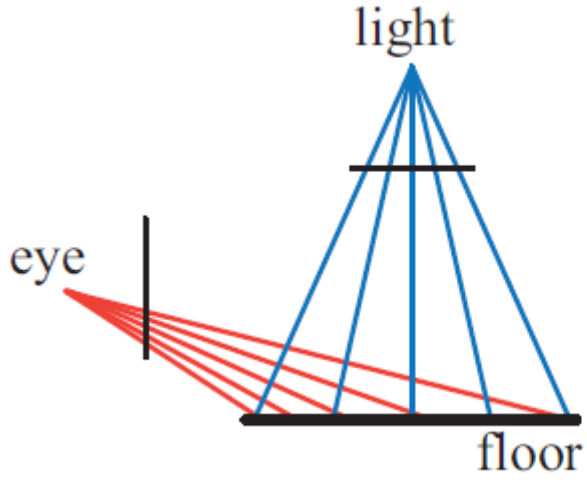
Perspective Aliasing Stairstepping Artifacts

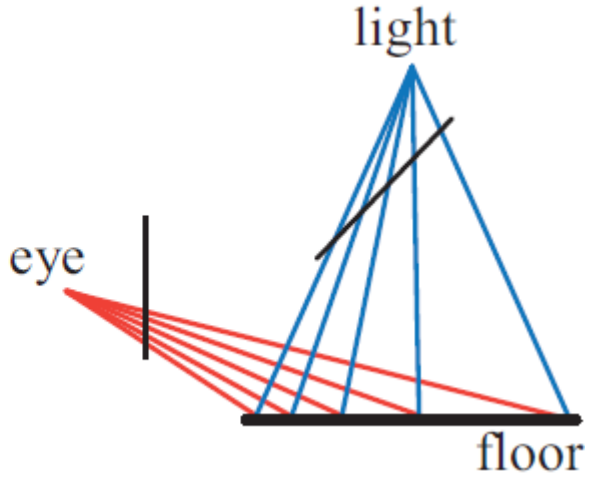


Perspective Aliasing Stairstepping Artifacts



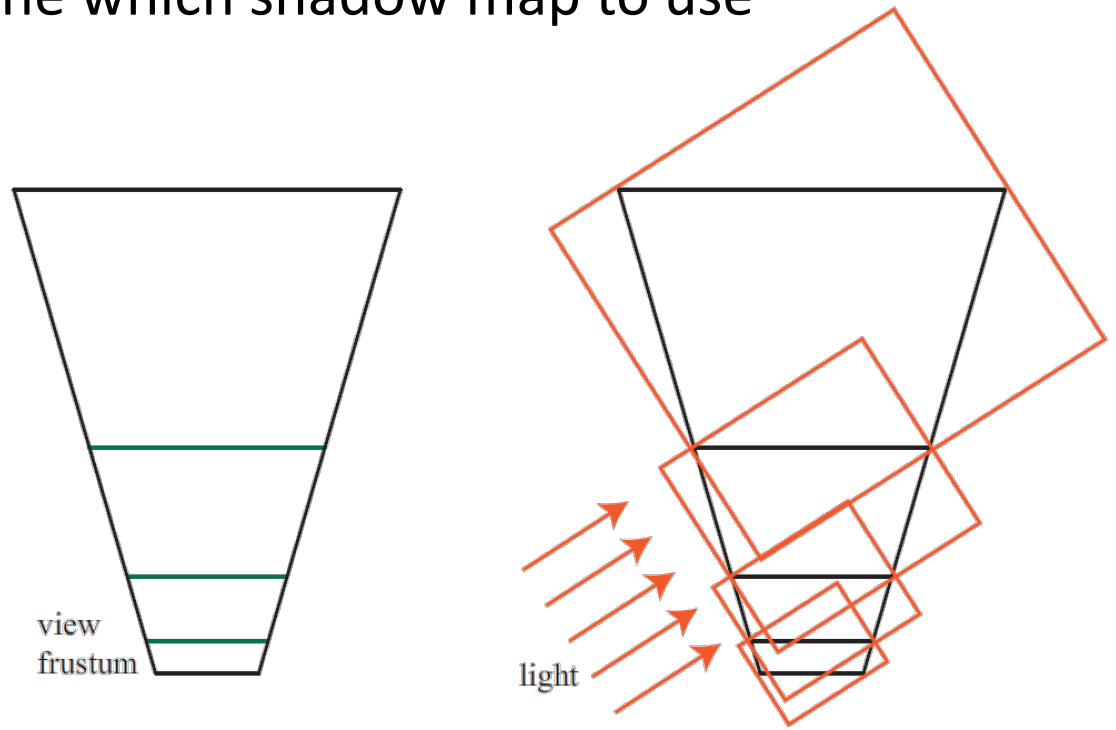






Hierarchical Shadow Maps

- Use variable resolution Shadow maps across the scene
  - Split view frustum into separate volumes
  - Create bounding box for each volume
  - Use this to determine which shadow map to use



Percentage Closer Filtering

- Sample depth map at multiple locations around a point
  - Assume area light source: causing variable exposure of projection surface
  - Instead of sampling the light over an area, use point-light model and sample around the projection point instead

Also creates soft shadows