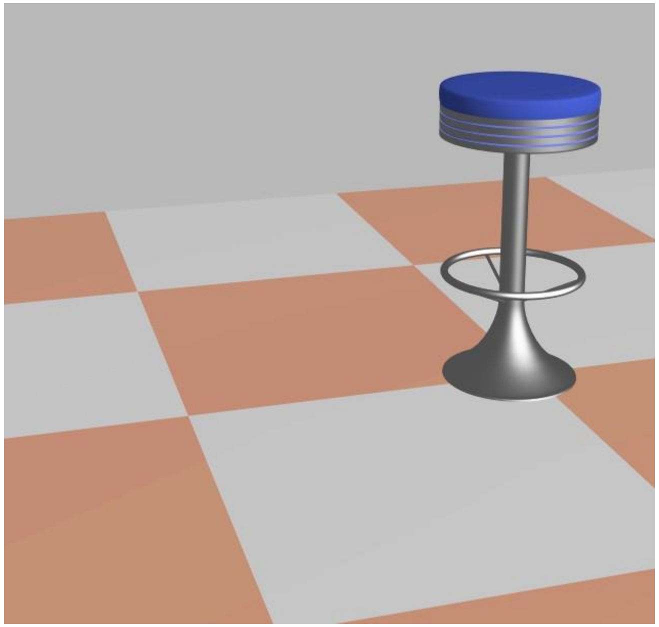
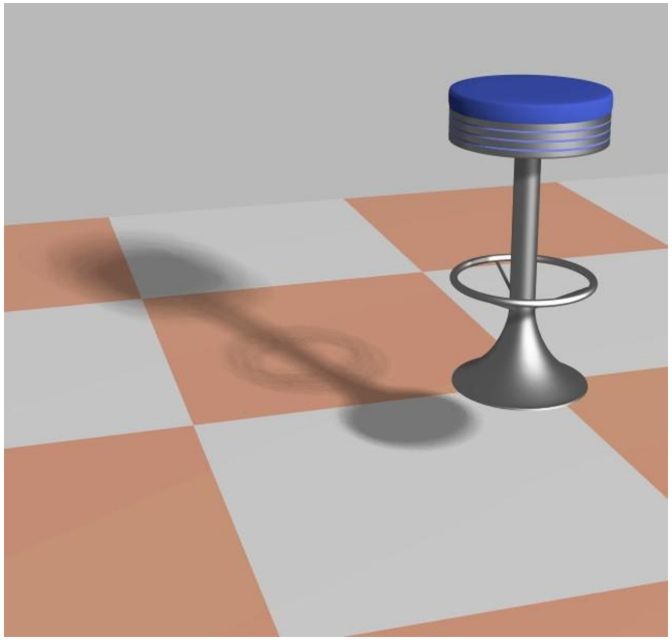
Real-time Graphics

5. Shadows

Martin Samuelčík

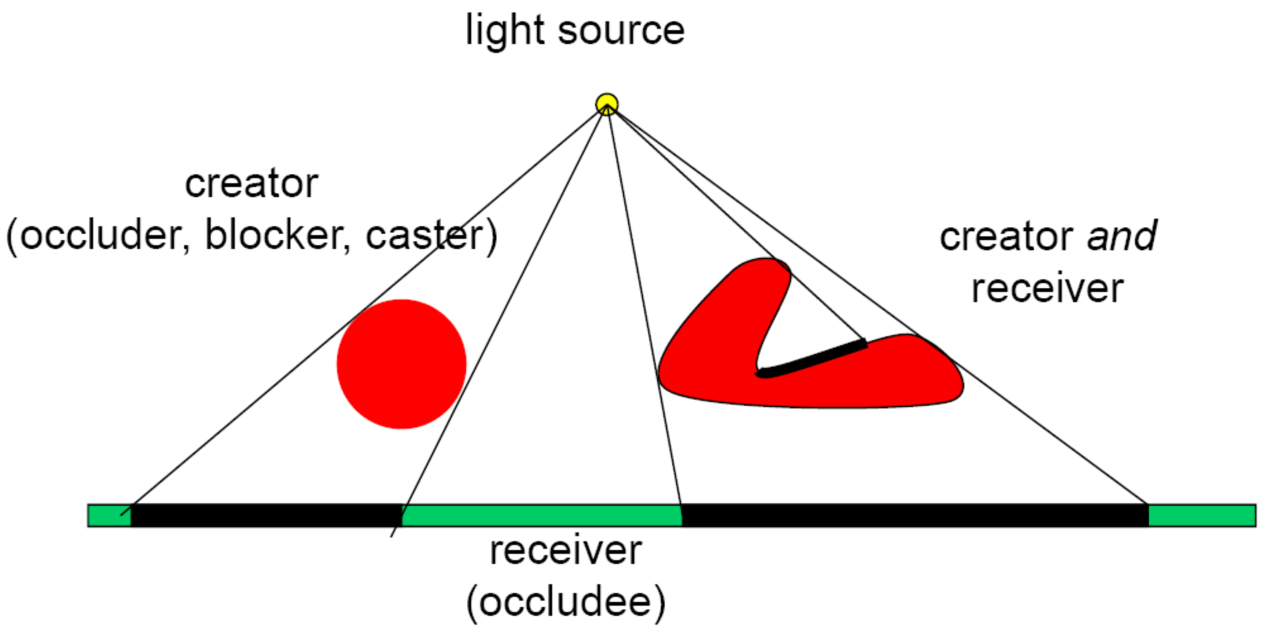
Shadows

- Realism, atmosphere
- Spatial relationships, object orientation, surface
- Light position, light properties

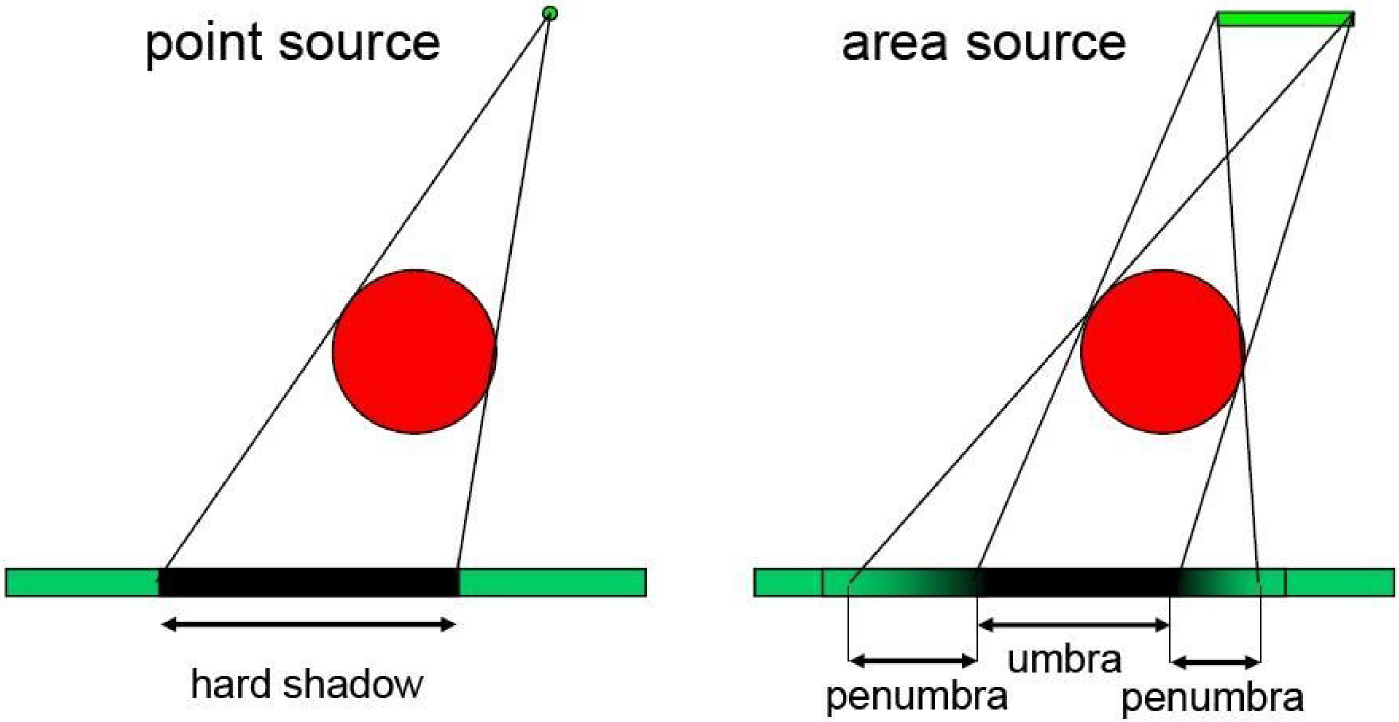
Shadows

- Part of global illumination
- For each point, determine if there is some occluder between point and light



Hard vs. Soft shadows

- For each point, determine how much of the light is visible from that point



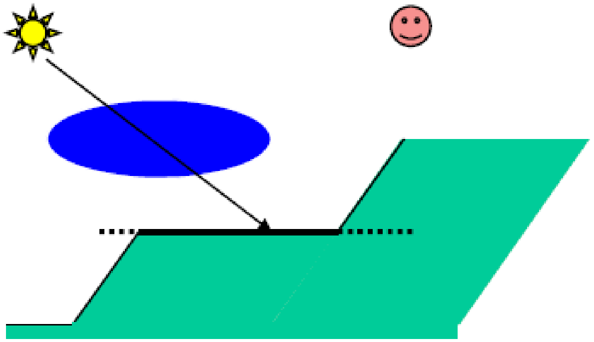
Shadows generation

- Offline generation + lights maps for static objects and lights
- Approximate shadows using simple polygons
- Cast shadows on planar polygons by transformation of objects into plane
- Compute intersection of shadow volumes with scene
- Determine occlusion from light's distance given by shadow map

Light maps

Approximate shadows

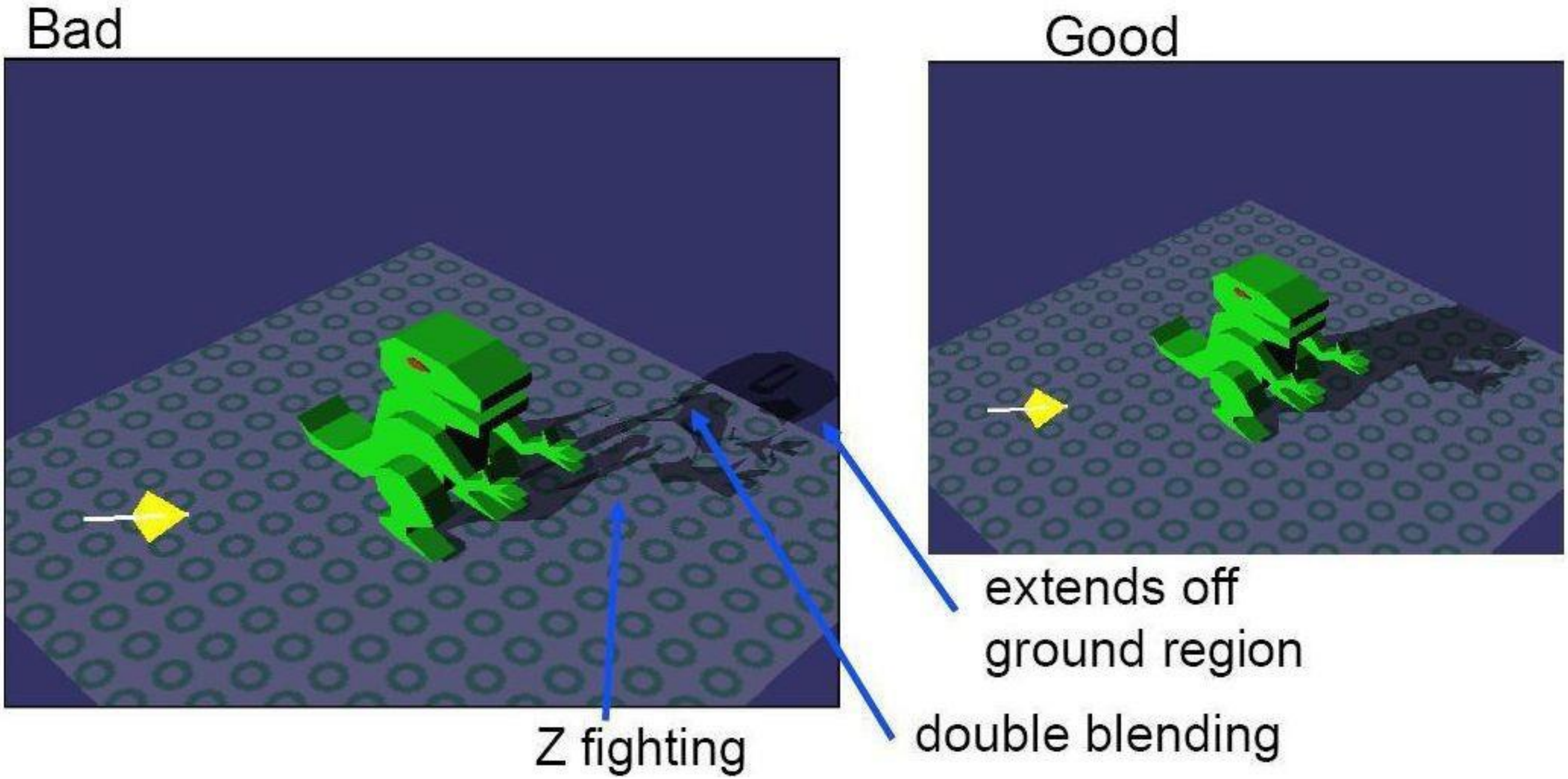
- Replacing shadow with simple shape
- Cast ray from light through feature object
- Blend simple shape with framebuffer



Planar shadows

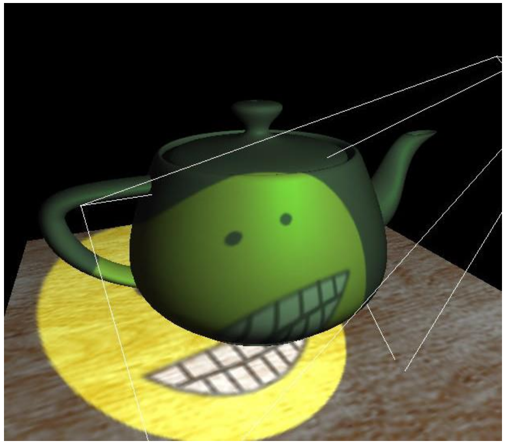
- For planar receivers
- Render occluder as deformed object transformed into receiver plane
- Blend receiver with transformed object
- Planar soft shadows - sample light source
- Several objects in one plane -> Z-fighting -> solve with *glPolygonOffset*
- Parts of transformed occluder outside receiver & double blending -> solve with stencil buffer

Planar shadows



Projective shadows

- Separate objects into occluders and receivers
- Render black occluders from light position into texture with white background
- Project rendered map onto receivers
- No self shadowing, artifacts

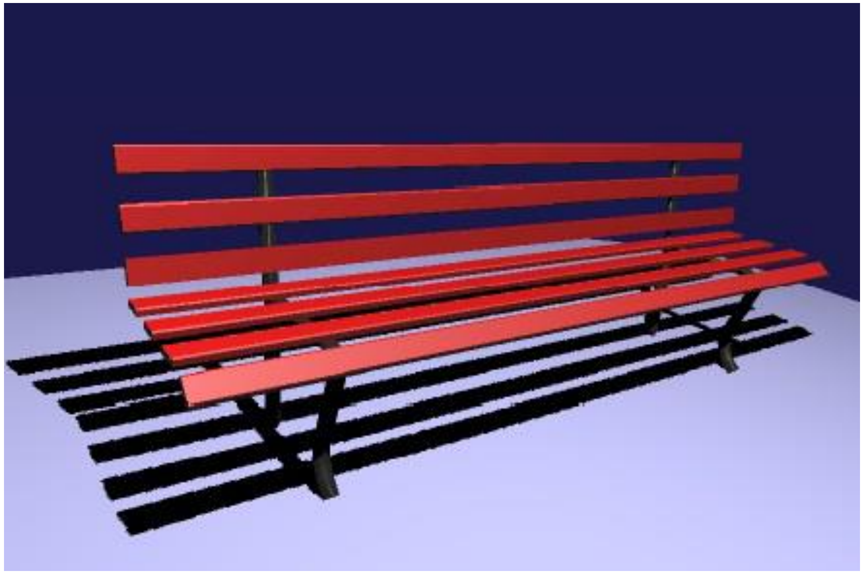


Shadow Texture

Result

Shadow mapping

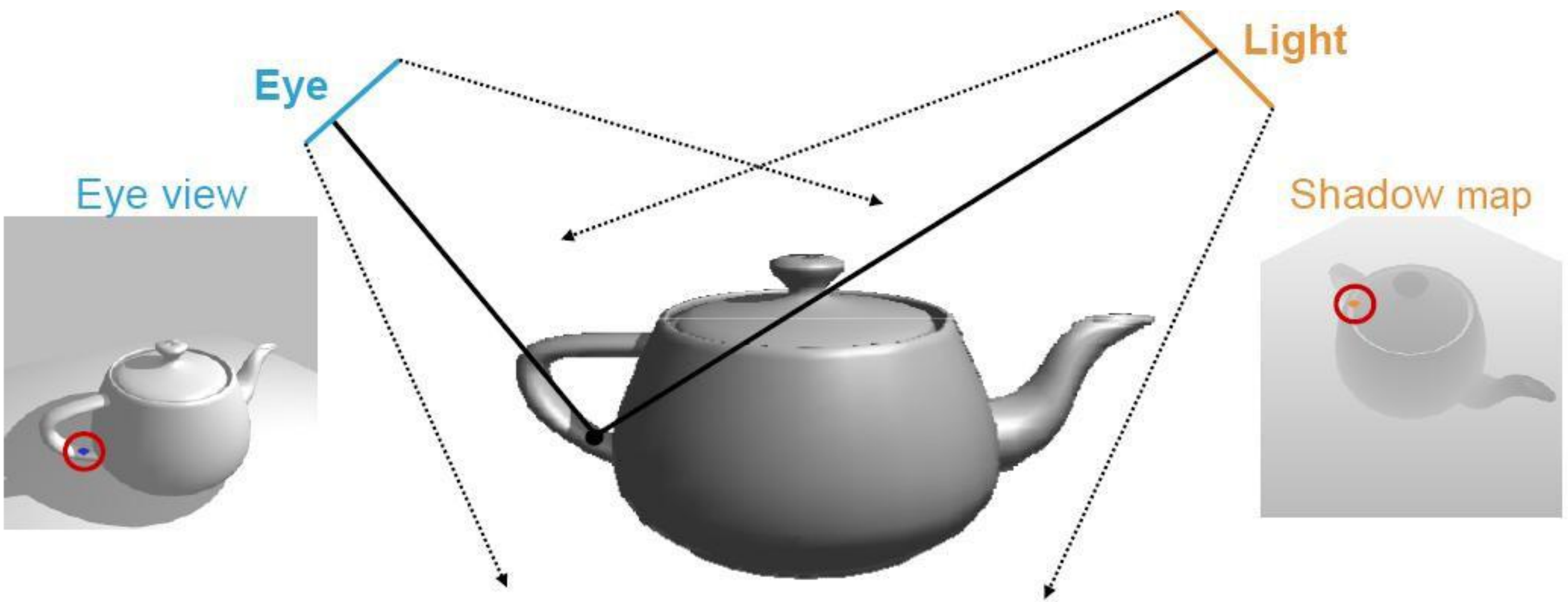
- Using map storing distances from light
- Image space – 2 pass algorithm
- HW supported, major shadow algorithm



Shadow mapping

- 1.pass - Render scene from light position
  - Render only depth buffer to texture
  - Depth buffer → shadow map
  - Shadow map holds distance D of objects to light
- 2. pass - Render scene normally
  - For each fragment, calculate distance S to the light
  - Transform fragments to light space – compute texture coordinates for shadow map lookup
  - Get distance D from shadow map (light space)
  - If S>D → fragment is in shadow

Shadow mapping



Light space

- 1.pass - Rendering scene from light's point of view
- Setting modelview (MV_L) and projection (P_L) transformation when rendering from light, so that light frustrum fit scene tightly
- Shadow map texture coordinates are computed for each vertex by reconstructing light transformation pipeline
- Third coordinate of shadow map coordinates is normalized sitance of vertex from camera

$[shadow\_clip] = P_L.MV_L.[x_o,y_o,z_o]^T$

$[shadow\_norm] = shadow\_clip / shadow\_clip.w$

$[shadow\_coords] = N. [shadow\_norm]$

$$N = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Light space

- MV_L consits of V_L (light view matrix) and M_L (model matrix)
- 2.pass – rendering from camera, with P_C (camera projection matrix), MV_C (camera modelview matrix) consists of V_C and M_C
- $M_L = M_C$
- We want to reconstruct shadow texture coordinates, need P_L and MV_L, but MV_L can be different for each object
- Solution 1: s_clip = $P_L.V_L.V_C^{-1}.MV_C.[x_o,y_o,z_o]T$
- Solution 2: s_clip = $P_L.V_L.M_C.[x_o,y_o,z_o]T$

Rendering depth to texture

- Using FBO
- Internal format for texture
  - GL_DEPTH_COMPONENT
  - GL_DEPTH_COMPONENT24
  - GL_DEPTH_COMPONENT32
- For perspective projection
- Depth precision
  - $s = 2^d - 1$, d is bit precision of buffer, <0,s> is then range of values in depth buffer
  - z_e, w_e - coordinates of vertex in eye space
  - z_w integer depth value of vertex in depth buffer

$$z_w = s * \left(\frac{w_e}{z_e} * \frac{f * n}{f - n} + 0.5\frac{f + n}{f - n} + 0.5\right)$$

$$\frac{z_e}{w_e} = \frac{\frac{f * n}{f - n}}{\frac{z_w}{s} - 0.5\frac{f + n}{f - n} + 0.5}$$

$$= \frac{f * n}{\frac{z_w}{s}(f - n) - 0.5(f + n) - 0.5(f - n)}$$

$$= \frac{f * n}{\frac{z_w}{s}(f - n) - f}$$

Depth precision

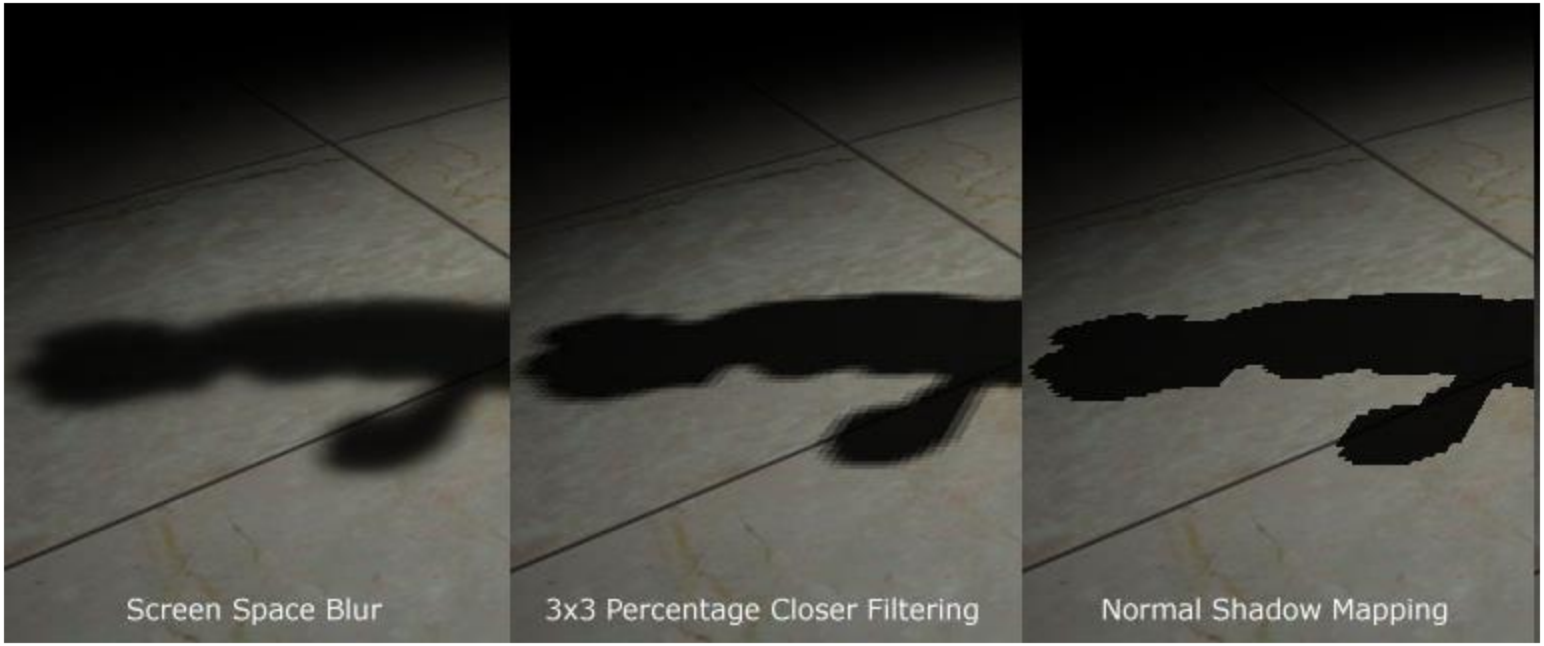
- https://www.opengl.org/wiki/Depth_Buffer_Precision
- Example 1:
  - 16 bit depth buffer, d = 16, s = 65535, n = 0.01, f = 1000, $w_e = 1$
  - $z_w = 0 \rightarrow z_e = -0.01 = -n$
  - $z_w = 1 \rightarrow z_e = -0.01000015$
  - $z_w = s-1 = 65534 \rightarrow z_e = -395.9$
  - $z_w = s = 65535 \rightarrow z_e = -1000 = -f$
  - All vertices with distance from camera between 0.01 and 0.01000015 are mapped into two values 0, 1 in depth buffer – very good precision
  - All vertices with distance from camera between 395.9 and 1000 are mapped only into two values 65534, 65535 in depth buffer – very poor precision

Depth precision

- Example 2:
  - 16 bit depth buffer, d = 16, s = 65535, n = 0.1, f = 100, w_e = 1
  - z_w = 0 -> z_e = -0.1 = -n
  - z_w = 1 -> z_e = -0.01000015
  - z_w = s-1 = 65534 -> z_e = -98.499
  - z_w = s = 65535 -> z_e = -100 = -f
  - Vertices with distance from camera between 0.01 and 0.01000015 are mapped into two values 0, 1 in depth buffer – very good precision
  - Vertices with distance from camera between 98.499 and 100 are mapped into two values 65534, 65535 in depth buffer – relatively good precision
- As the ratio (f/n) increases, less precision is available near the back of the depth buffer and more precision is available close to the front of the depth buffer
- For same precision in whole range, use linear depth – normalized z_e values

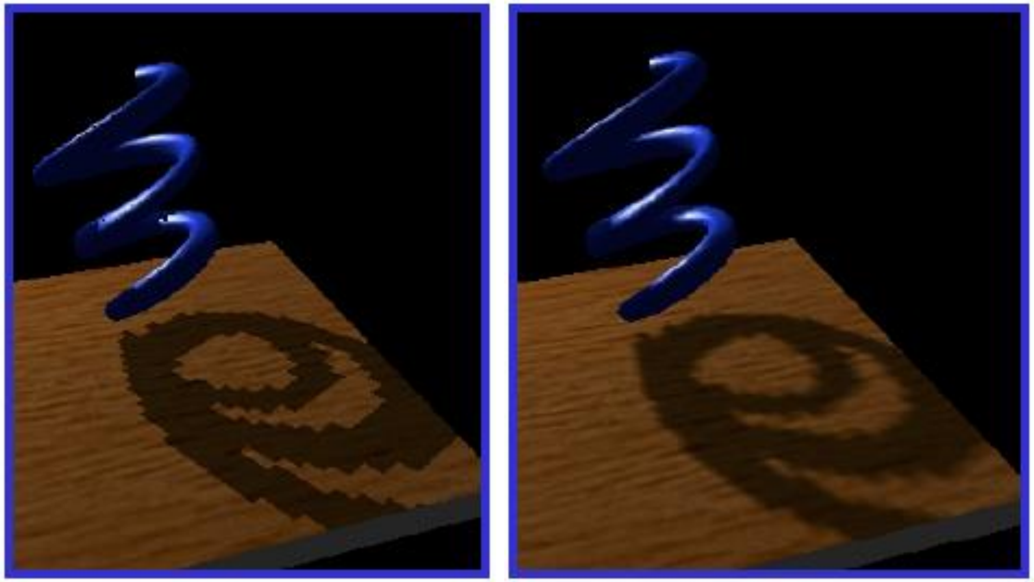
SM resolution problem

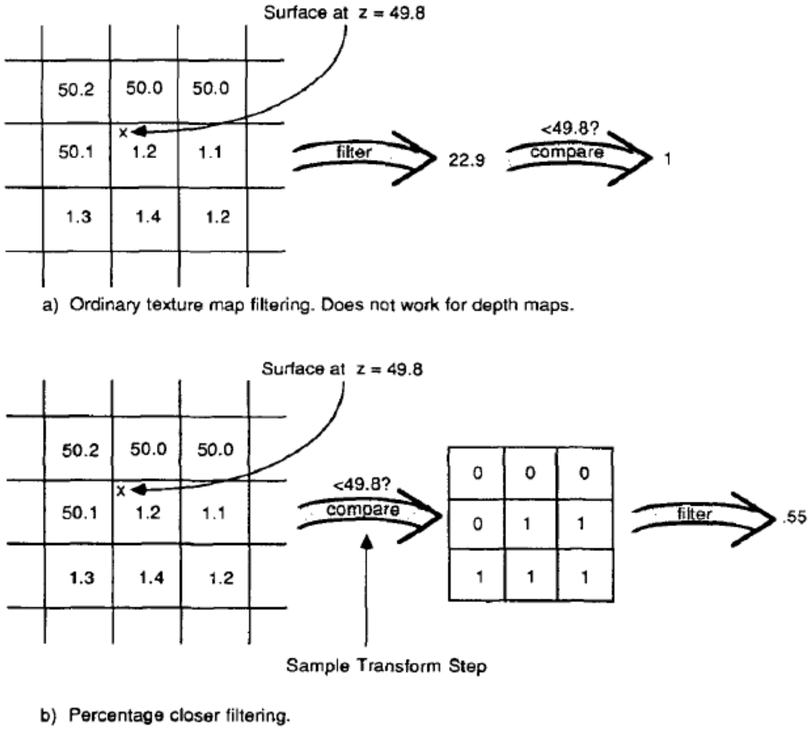
- Map resolution – jagged edges of shadows
- Solution – use higher resolution, blur shadow map (percentage close filtering), use more maps, ...



Percentage close filtering

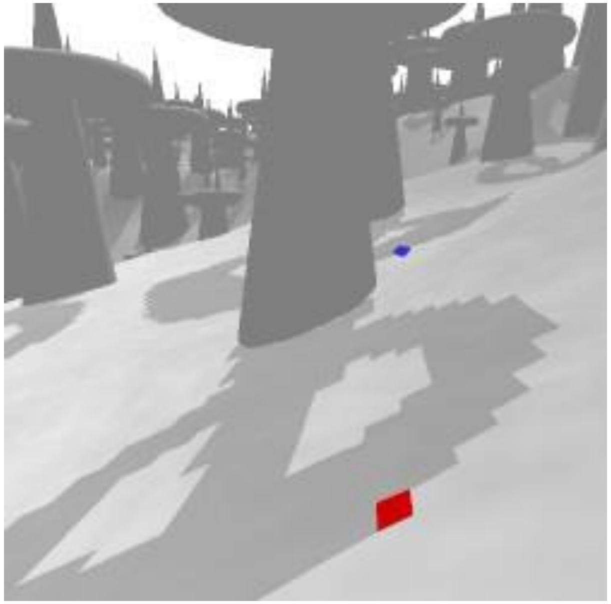
- Classical filter – filtering of depth values
- PCF – filtering depth comparison values
- HW supported

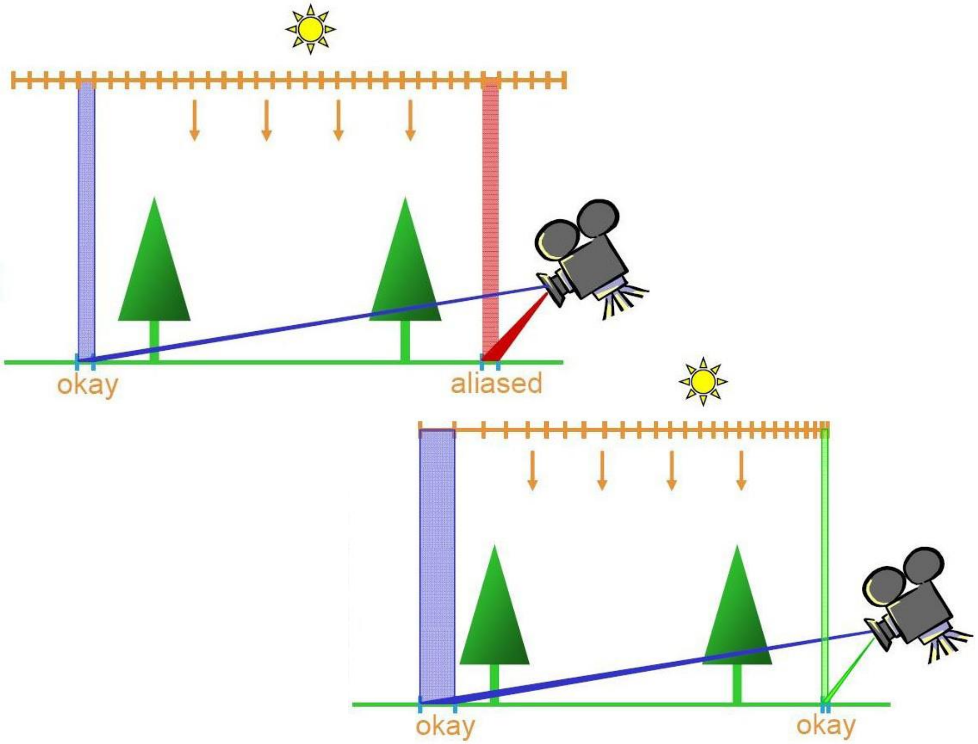




SM problems

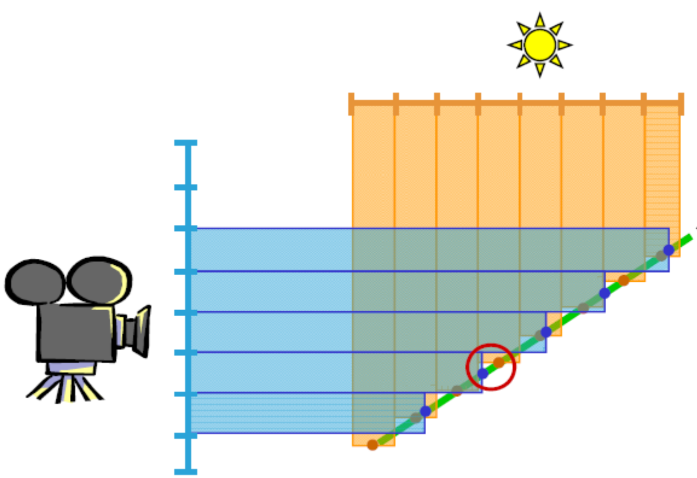
- Perspective aliasing – same shadow map resolution for near and far objects
- Solution – redistribute values in shadow map

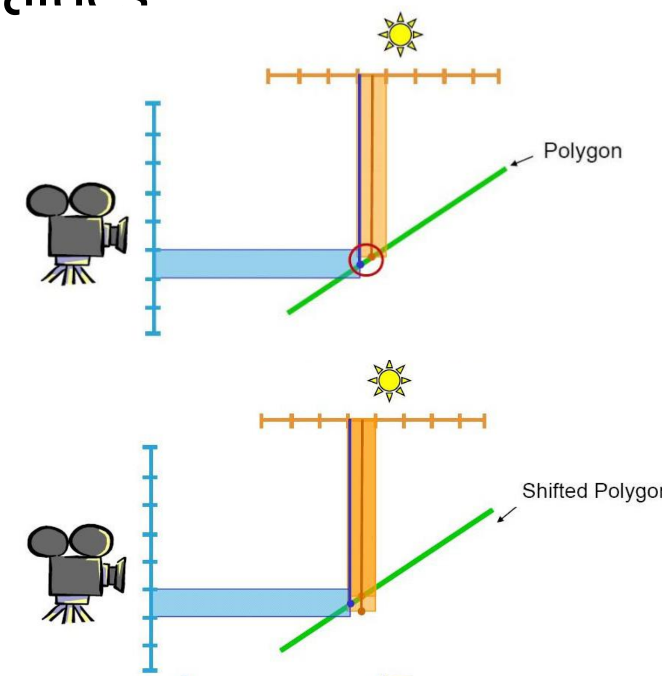




SM problems

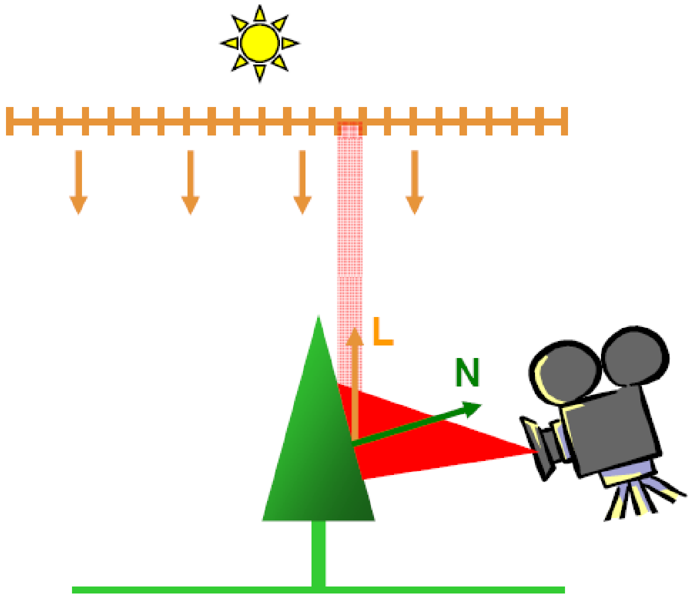
- Incorrect self-shadowing – caused by similar values of S (normalized z eye coordinate of fragment) and D (depth from shadow map) on lit surface and SM precision
- Solution – add bias to shadow map values





SM problems

- Projective aliasing – for planes almost parallel to light direction
- Solution – similar to perspective aliasing



OpenGL SM support

- Not using shaders
  - Computation of light space in texture transformation
  - *GL_ARB_shadow* – testing, depth comparision
- Shaders
  - Test in fragment shader
  - Uniforms for accessing depth textures
  - sampler2D – fetch from texture gets depth values, , possibly filtered
  - sampler2Dshadow – fetch from texture returns result of comparison, possibly filtered
  - Comparision function for shadow sampler setting
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_REF_TO_TEXTURE)
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL)
  - Support for float samplers and sampler objects in newer OpenGL

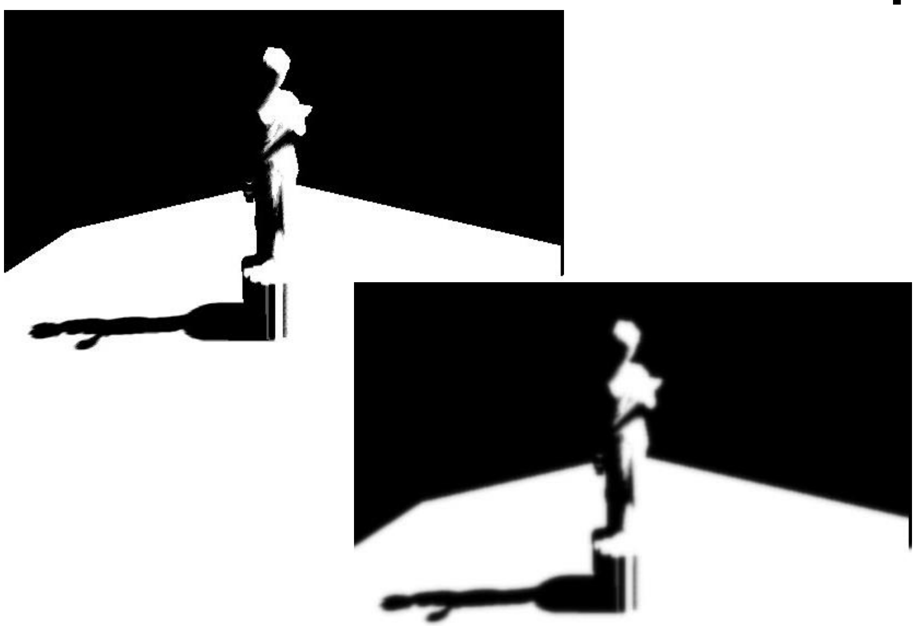
Shadow mapping - GLSL

```
// SHADOW MAPPING VERTEX SHADER - 2.pass
varying vec4 L_eye;
varying vec4 N_eye;
varying vec4 diffTexCoords;
varying vec4 depthTexCoords;

uniform mat4 lightProj;
uniform mat4 lightView;
uniform mat4 cameraViewInverse;

void main(void)
{
  // compute vectors for light calculation
  vec4 V_eye = gl_ModelViewMatrix * gl_Vertex;
  L_eye = normalize(gl_LightSource[0].position - V_eye);
  N_eye = vec4(gl_NormalMatrix * gl_Normal, 0.0);
  gl_Position = gl_ProjectionMatrix * V_eye;

  diffTexCoords = gl_MultiTexCoord0;

  // compute normalized light space coordinates
  depthTexCoords = lightProj * lightView *  cameraViewInverse * V_eye;
  depthTexCoords = 0.5 * (depthTexCoords / depthTexCoords.w) + 0.5;
  V_eye = -V_eye;
}
```

```
// SHADOW MAPPING FRAGMENT SHADER - 2.pass
varying vec3 L_eye;
varying vec3 N_eye;
varying vec4 diffTexCoords;
varying vec4 depthTexCoords;
uniform sampler2D diffuseTexture;
uniform sampler2DShadow depthLightTexture;

void main(void)
{
  vec3 L = normalize(vec3(L_eye));
  vec3 N = normalize(vec3(N_eye));

  difColor = texture(diffuseTexture, vec2(diffTexCoords));
  float diffuse = max(dot(L, N), 0);

  // add some epsilon to depth to prevent bias
  depthTexCoords.z += 0.0001;

  // get result of comaprision between depth from shadow map and third
     coordinate of depthTexCoords
  float depth_test = texture(depthLightTexture, vec3(depthTexCoords));
  diffuse = depth_test * diffuse;

  gl_FragColor = 0.5f * gl_LightSource[0].ambient * diffColor +
                 diffuse * gl_LightSource[0].diffuse * diffColor;
}
```

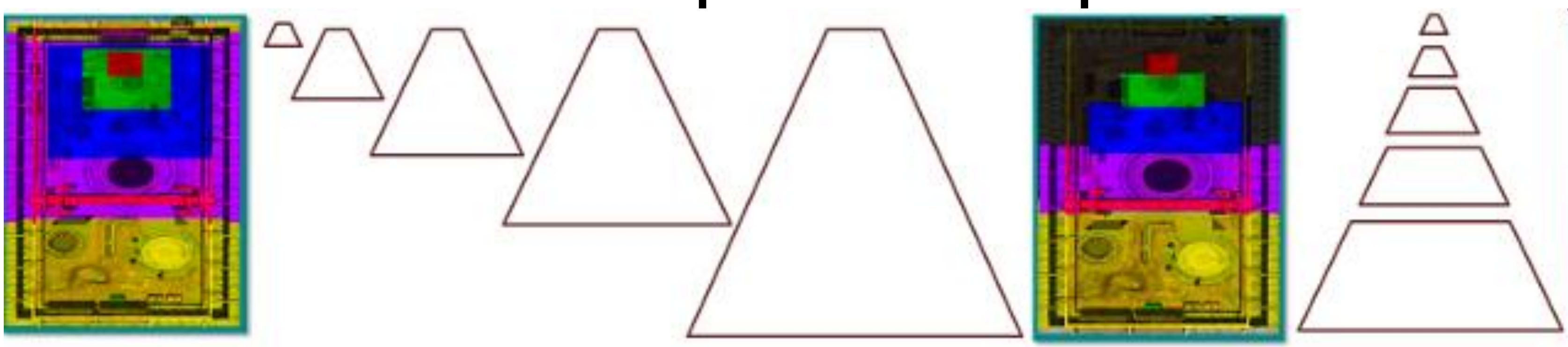
Soft shadows

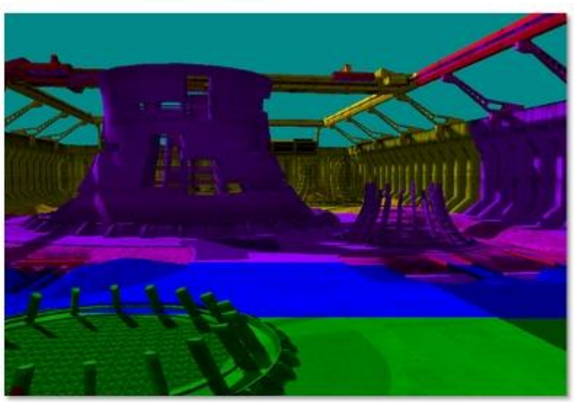
- PCF generates blurred, soft shadows
- Can be blurred also in image space
- PCF kernel size adaptive to distance

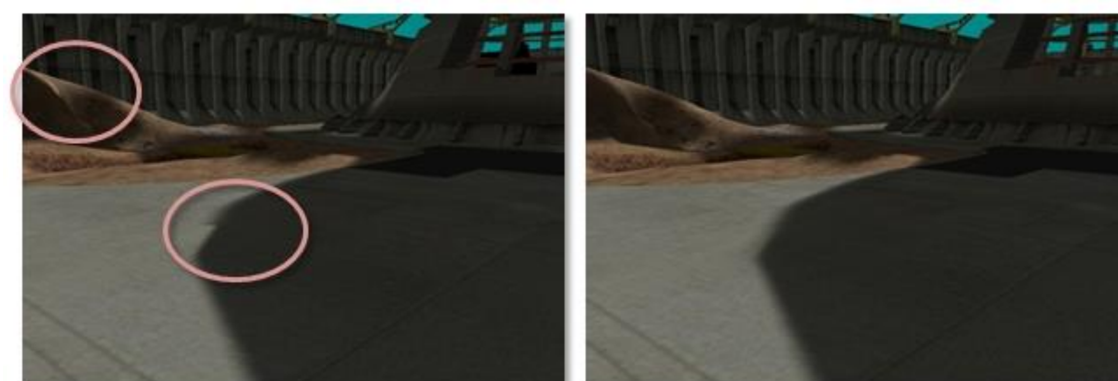


Cascaded SM

- Split view frustum to several parts and create shadow map for each part



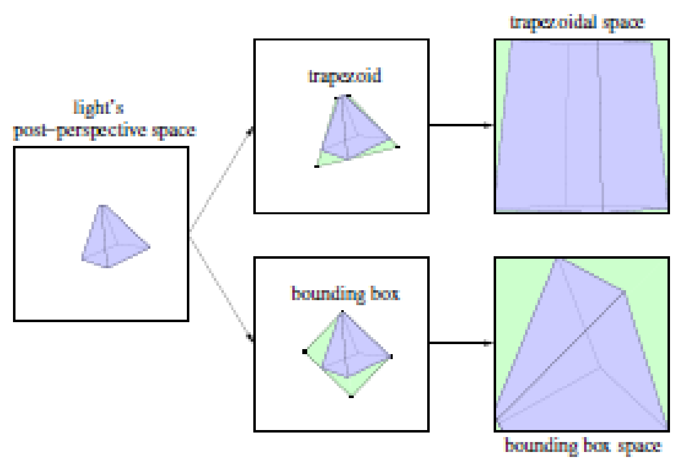




Microsoft

Trapezoidal SM

- http://www.comp.nus.edu.sg/~tants/tsm.html
- In shadow map texture space, find 2D trapezoid that contains whole projected
- Create transformation TR, that maps classic shadow map texture space to trapezoid texture space
- Store only interior of trapezoid in final shadow map
- In 2. pass, add TR when transforming from object space to light's normalized space



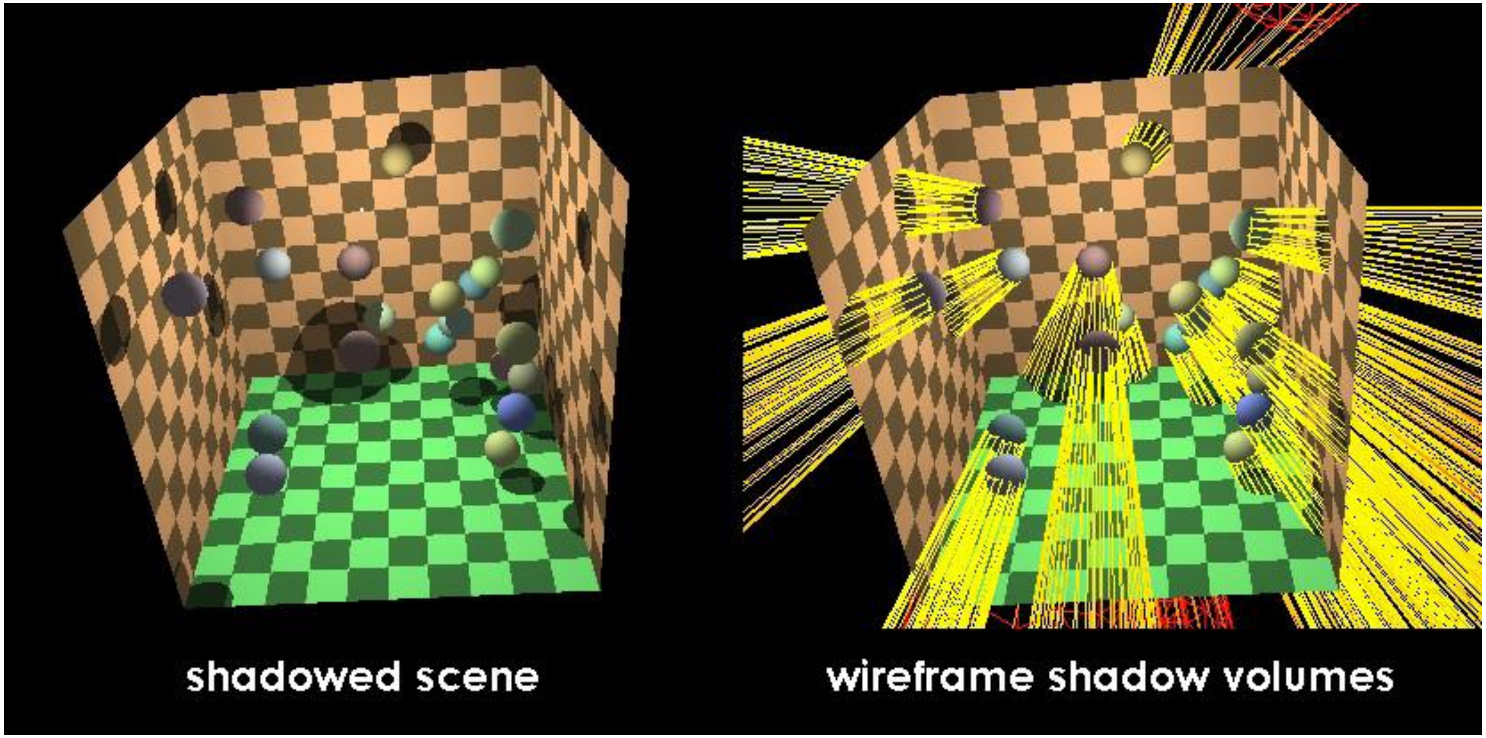
SM optimalization

- Simple
  - SSM "Simple"
- Splitting
  - PSSM "Parallel Split"
  - CSM "Cascaded"
- Warping
  - LiSPSM "Light Space Perspective"
  - TSM "Trapezoid"
  - PSM "Perspective"
- Smoothing
  - PCF "Percentage Closer Filtering"
- Filtering
  - ESM "Exponential"
  - CSM "Convolution"
  - VSM "Variance"
  - SAVSM "Summed Area Variance"
- Soft Shadows
  - PCSS "Percentage Closer"
- Assorted
  - ASM "Adaptive"
  - AVSM "Adaptive Volumetric"
  - CSSM "Camera Space"
  - DASM "Deep Adaptive"
  - DPSM "Dual Paraboloid"
  - DSM "Deep"
  - FSM "Forward"
  - LPSM "Logarithmic"
  - MDSM "Multiple Depth"
  - RMSM "Resolution Matched"
  - SDSM "Sample Distribution"
  - SPPSM "Separating Plane Perspective"

wikipedia.org

Shadow volumes

- Shadow volume – set of rays from light through each vertex of occluder, rays begin at vertex



Shadow volumes

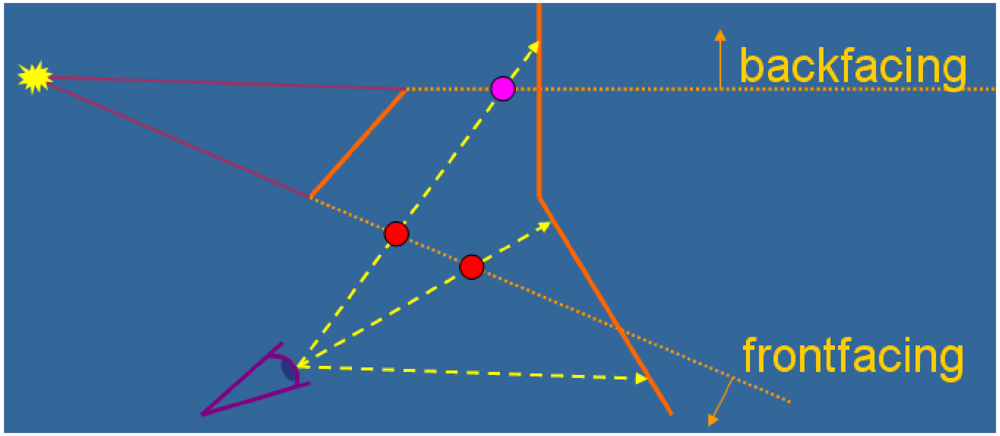
- For polygonal models
- Compute silhouette edges of shadow casting object with respect to the light source
  - Get orientation of face from face vertices
  - Silhouette edge – between front-facing and back-facing faces
  - Remove interior edges
- Compute volume quad by extruding silhouette edge till the end of scene in the light direction
- Add caps at the ends of volume

Shadows with SV

- Points in shadow are inside some light shadow volume
- For each fragment, check if fragment is in interior of shadow volume
- Stencil buffer implementations – masking scene
- Shadow volume algorithm
  - 1. Render the scene as if it were completely in shadow (ambient light)
  - 2. For each light source:
    - 1. Construct a mask in the stencil buffer that has holes only where the visible surface is not in shadow.
    - 2. Render the scene again with diffuse and specular light only in lit areas based on the stencil buffer mask. Use additive blending to add this render to the scene.

Shadows with SV

- C – position of camera
- Fragment F inside shadow volume
  - If C is outside, then segment CF has odd number of intersections with SV planes (depth pass)
  - Ray from F and direction (F-C) has odd number of intersections with SV planes (depth fail)



Carmack's Reverse
Creative Labs patent ☺

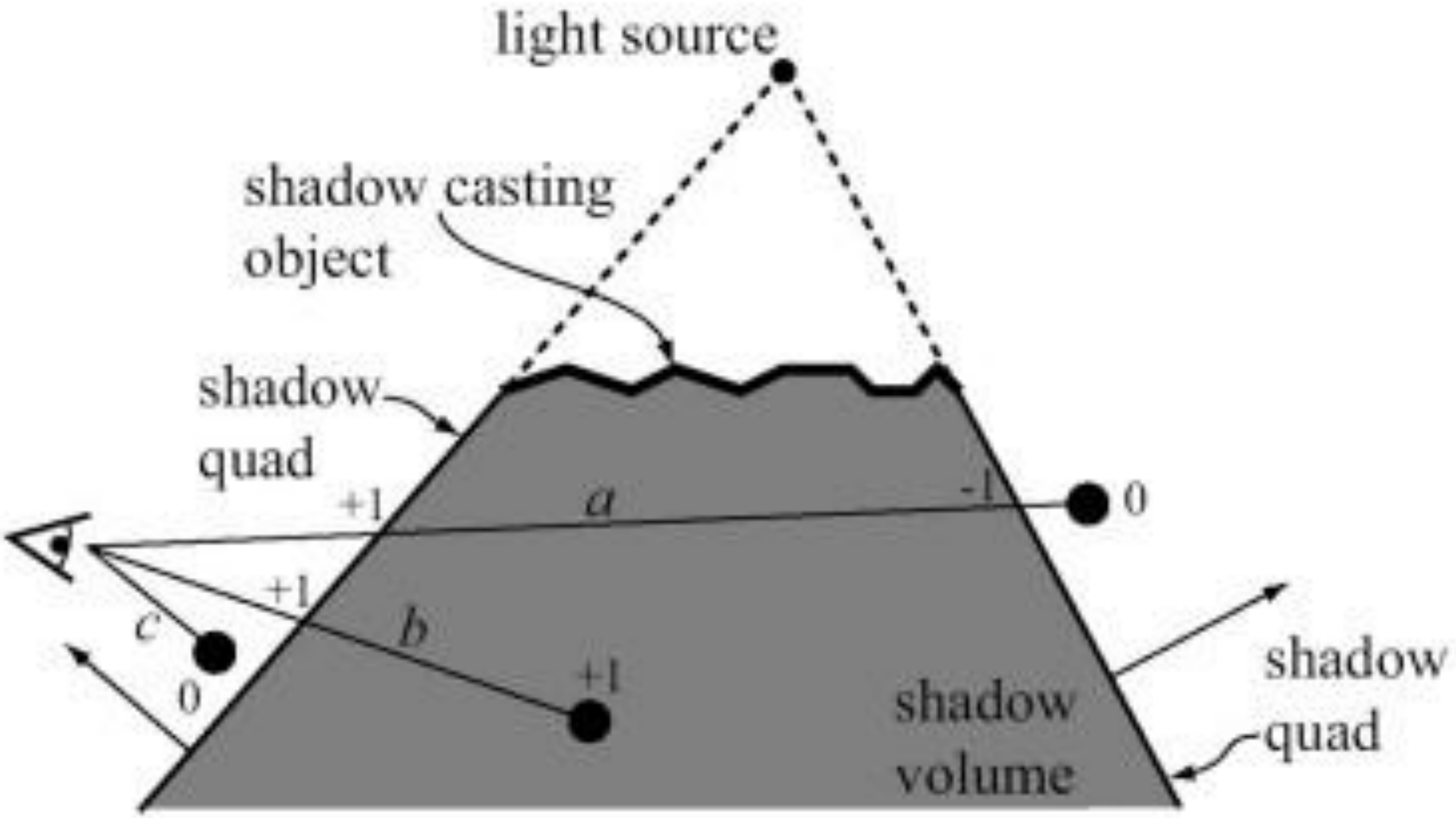
SV - depth pass

- Clear stencil and color buffer
- Render scene with ambient light
- Disable writes to the depth and color buffers.
- Use back-face culling.
- Set the stencil operation to increment on depth pass (only count shadow volume planes in front of the object).
- Render the shadow volumes (only front faces are rendered).
- Use front-face culling.
- Set the stencil operation to decrement on depth pass.
- Render the shadow volumes (only back faces are rendered).
- Enable writes to the depth and color buffers.
- Render scene with diffuse & specular light, update fragments only where stencil = 0

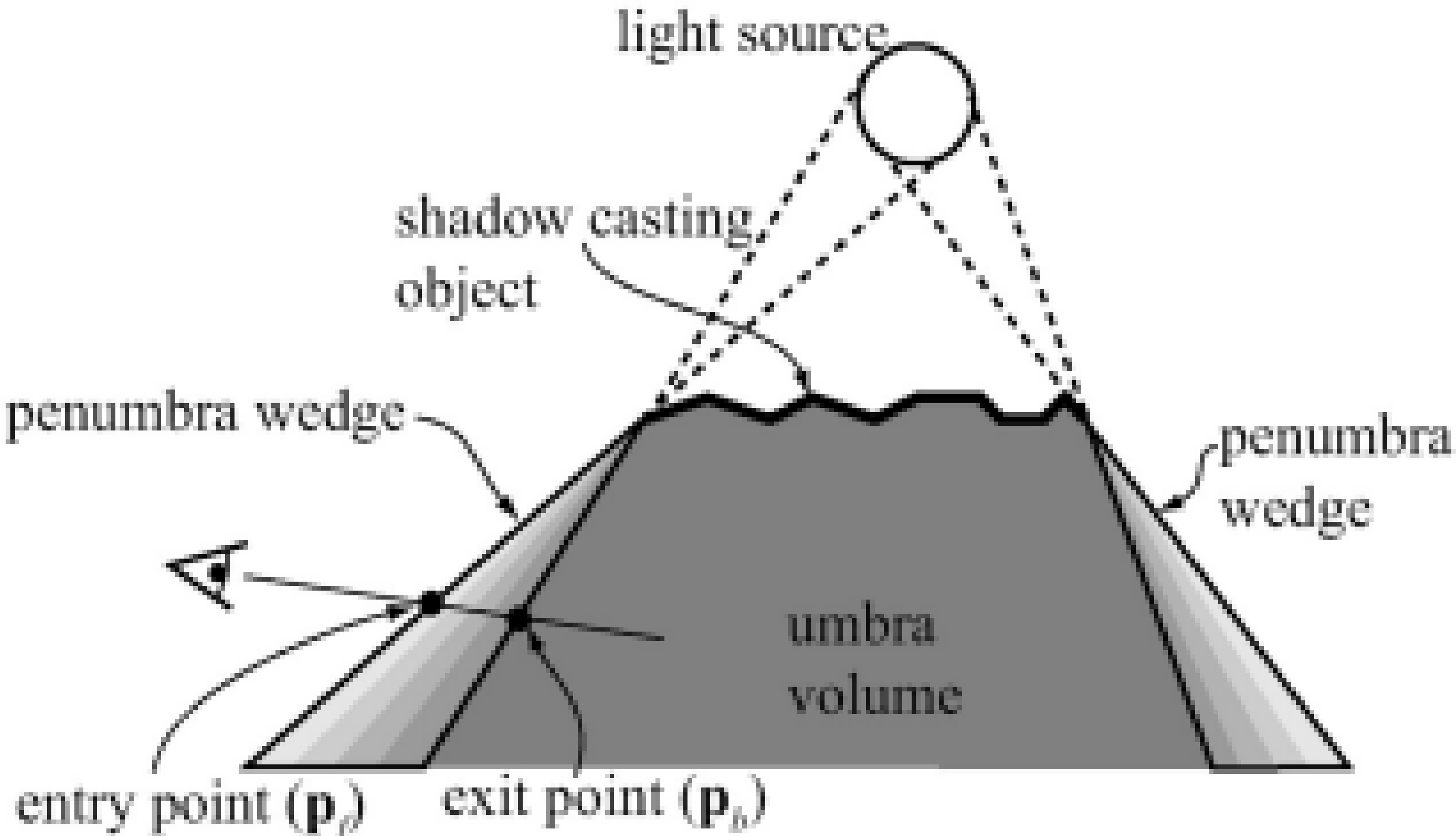
SV - depth fail

- Clear stencil and color buffer
- Render scene with ambient light
- Disable writes to the depth and color buffers.
- Use front-face culling.
- Set the stencil operation to increment on depth fail (only count shadow volume planes behind the object).
- Render the shadow volumes with caps.
- Use back-face culling.
- Set the stencil operation to decrement on depth fail.
- Render the shadow volumes with caps.
- Enable writes to the depth and color buffers.
- Render scene with diffuse & specular light, update fragments only where stencil = 0

Shadow volumes



Shadow volumes



SV optimalization

- Reduction of SV rasterization using scissor test
- For camera inside SV, use depth fail, and depth pass otherwise
- Problem with far plane -> homogenous coordinates
- Use *EXT_stencil_two_side* to reduce number of SV rendering passes
- Use *EXT_depth_bounds_test* to remove shadow volumes that do not affect the visible scene
- Approximation of silhouette
- Using simple bounding volumes, using BSP trees

Shadow volumes

- Only hard shadows, geometry limited
- Robust, self-shadowing, GPU

id Software

Bioware

Shadow sources

- http://www.nealen.net/projects/ibr/shadows.pdf
- http://graphics.pixar.com/library/
- http://developer.nvidia.com/object/hwshadowmap_paper.html
- http://www.ia.hiof.no/~borres/cgraph/explain/shadow/p-shadow.html
- http://en.wikipedia.org/wiki/Shadow_mapping
- http://en.wikipedia.org/wiki/Shadow_volume
- http://www.cg.tuwien.ac.at/courses/Realtime/slides/2008/07Shadows.pdf
- http://developer.nvidia.com/object/fast_shadow_volumes.html
- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch10.html
- http://msdn.microsoft.com/en-us/library/ee416307%28v=vs.85%29.aspx
- http://developer.download.nvidia.com/shaderlibrary/docs/shadow_PCSS.pdf

Questions?