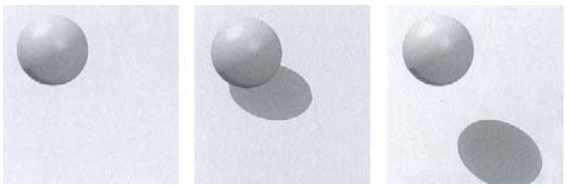
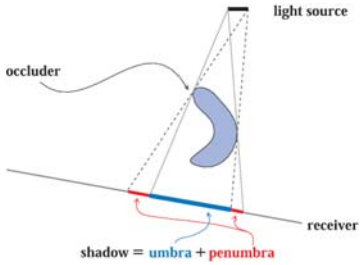
Shadows

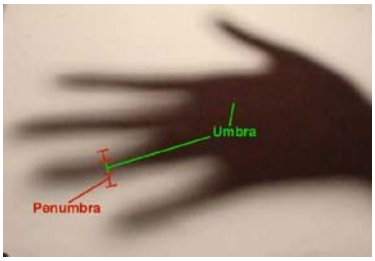
Niels Jørgen Christensen
IMM . DTU

Effect of Shadows

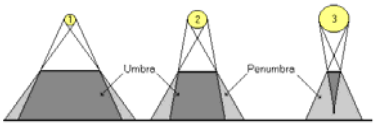


Soft Shadows





Light source - Size



Anything like a shadow

Projection

$$\mathbf{P}_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_{oblique} = \begin{bmatrix} 1 & 0 & \cot\theta & 0 \\ 0 & 1 & \cot\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}_{central} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

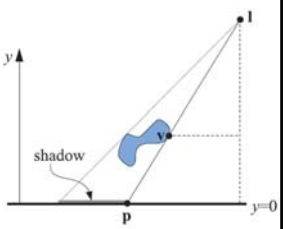
Projection Shadows - Möller

$$\frac{p_x - l_x}{v_x - l_x} = \frac{l_y}{l_y - v_y} \Rightarrow p_x = \frac{l_y v_x - l_x v_y}{l_y - v_y}$$

$$p_y = 0$$

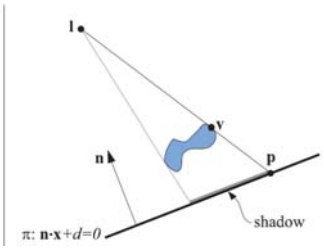
$$p_z = \frac{l_y v_z - l_z v_y}{l_y - v_y}$$

$$M = \begin{bmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{bmatrix} \qquad p = Mv$$



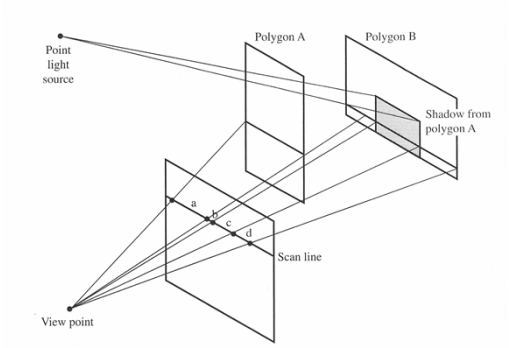
$$\pi :: y = 0 \rightarrow \pi .. : nx + d = 0$$

$$p = l - \frac{d + n \cdot l}{n \cdot (v - l)}(v - l)$$

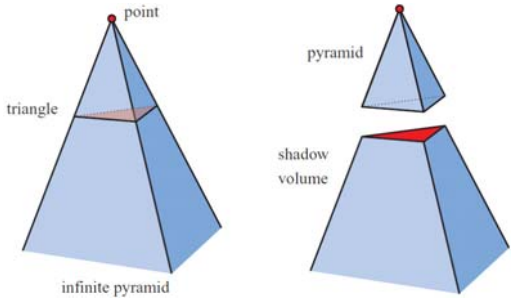


$$M = \begin{bmatrix} n \cdot l + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & n \cdot l + d - l_y n_y & -l_y n_z & -l_x d \\ -l_z n_x & -l_z n_y & n \cdot l + d - l_z n_z & -l_x d \\ -n_x & -n_y & -n_z & n \cdot l \end{bmatrix}$$

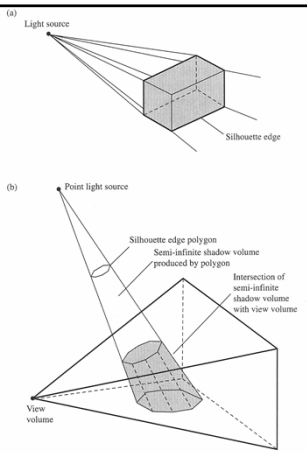
Scanline

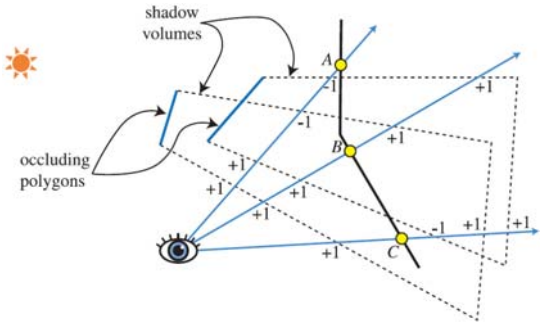


Shadow Volume

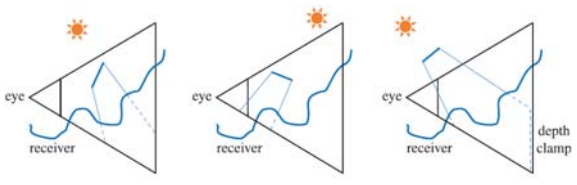


Shadow Volum





Shadow Volume - Problems

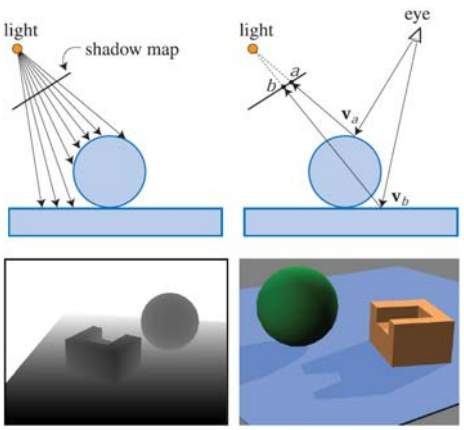


Contours

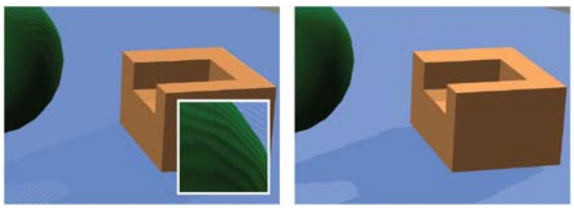
Shadow Volume - Properties

- General purpose HW
- No sampling problems – not image based
- Fill rate limited – SV polygons big
  - Rasterizing bottleneck
- Stencil buffer needed
  - or modified method
- Soft shadows from blending multiple SV
  - Expensive
- No HW shadows from curved surfaces

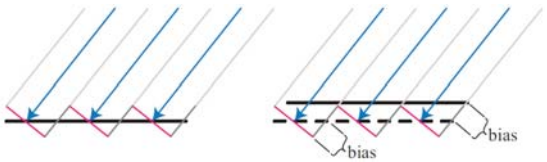
Shadow Map



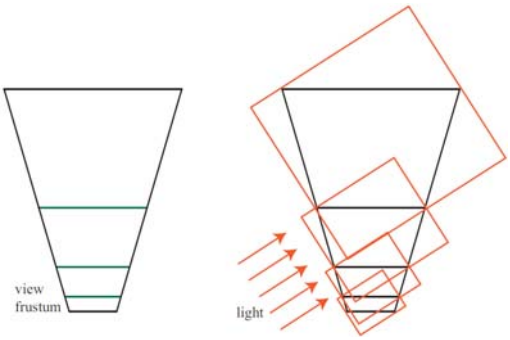
Shadow Map - problems



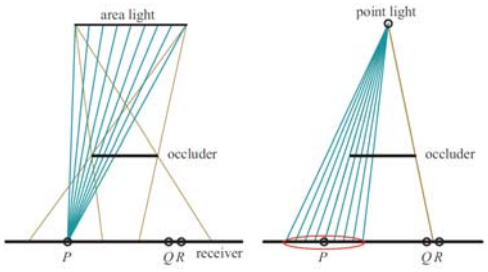
Shadow Map - Bias



Shadow Map - Problem

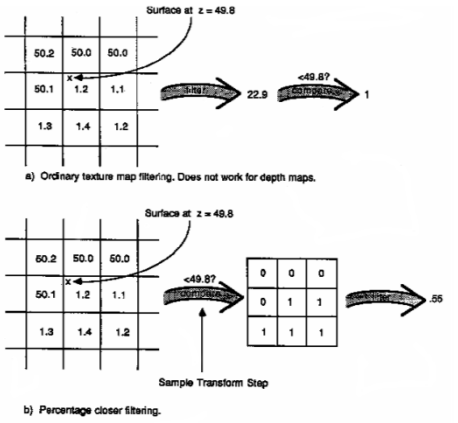


Percentage Closest Filtering - PCF

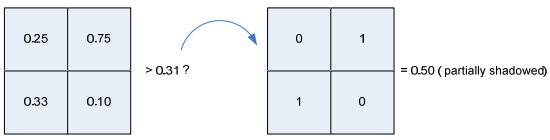


Percentage Closest Filtering

0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1

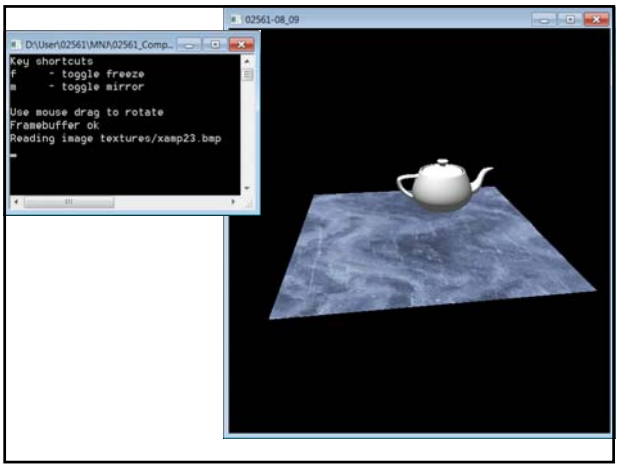


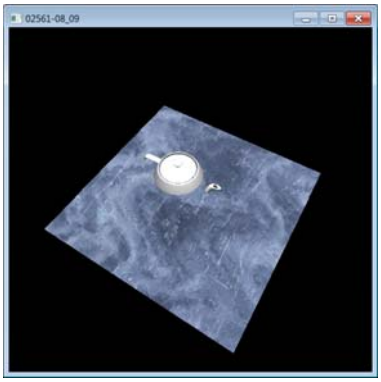
Percentage-Closer Filter

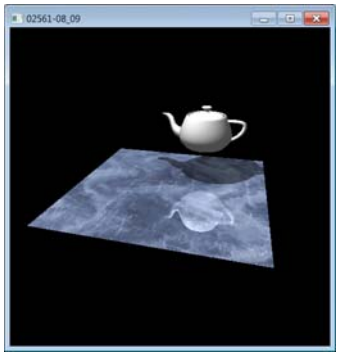


Shadow Map - Exercise

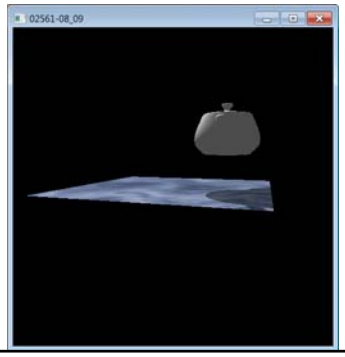
- Shadow maps - week 8
- Reflections – week 9 (continuation)

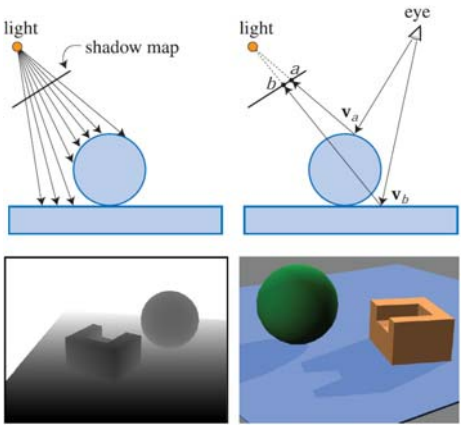


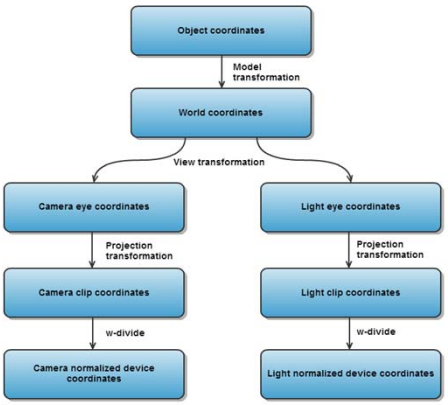




To do



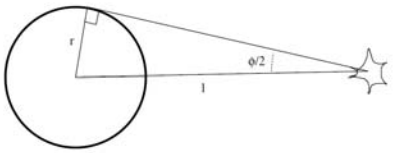





```
getLightProjection() {
// todo implement
return mat4();
}

mat4 getLightView() {mat4
// todo implement
return mat4();
}

mat4 getLightViewProjection(){
return getLightProjection() * getLightView();
}

void updateProjShadowTexture() {
// todo bind framebuffer, set viewport to shadowmap and clear to white

mat4 projection = getLightProjection();
mat4 view = getLightView();
mat4 model = Translate(teapotPosition);

// todo render black teapot

// todo release framebuffer, setviewport to window
```

P_{light}

```
mat4 getLightProjection() {
// todo: implement Perspective from light
// Use bounding volume around teapot
return mat4();
                                    }
```



V_{light}

```
mat4 getLightView() {
// todo : use  LookAt
return mat4();
}
```

$P_{light} * V_{light}$

```
mat4 getLightViewProjection(){
return getLightProjection() * getLightView();
}
```

updateProjShadowTexture

```
void updateProjShadowTexture() {
// todo bind framebuffer, set viewport to shadowmap, clear to white

mat4 projection = getLightProjection();
mat4 view = getLightView();
mat4 model = Translate(teapotPosition);

// todo render black teapot

// todo release framebuffer, setviewport to window
}
```

LookAt - V

```
mat4 LookAt( const vec4& eye, const vec4& at, const vec4& up )
{
vec4 n = normalize(eye - at);
vec4 u = vec4(normalize(cross(up,n)),0.0);
vec4 v = vec4(normalize(cross(n,u)),0.0);
vec4 t = vec4(0.0, 0.0, 0.0, 1.0);
mat4 c = mat4(u, v, n, t);
return c * Translate( -eye );
}
```

In Angel/mat.h

Perspective P_{per}

```
mat4 Perspective( const GLfloat fovy, const GLfloat aspect,
  const GLfloat zNear, const GLfloat zFar) {
    GLfloat top   = tan(fovy*DegreesToRadians/2) * zNear;
    GLfloat right = top * aspect;
    mat4 c;
    c[0][0] = zNear/right;
    c[1][1] = zNear/top;
    c[2][2] = -(zFar + zNear)/(zFar - zNear);
    c[2][3] = -2.0*zFar*zNear/(zFar - zNear);
    c[3][2] = -1.0;
    c[3][3] = 0.0;
    return c;
} //In Angel/mat.h
```

updateProjShadowTexture

```
void updateProjShadowTexture() {
// todo bind framebuffer, set viewport to shadowmap,clear to white

mat4 projection = getLightProjection();
mat4 view = getLightView();
mat4 model = Translate(teapotPosition);

// todo render black teapot

// todo release framebuffer, setviewport to window
}
```

```
GLuint buildFrameBufferObject(int width, int height, GLuint textureId)
{
GLuint framebufferObjectId, renderBufferId;
glGenFramebuffers(1, &framebufferObjectId);
glGenRenderbuffers(1, &renderBufferId);
glBindRenderbuffer(GL_RENDERBUFFER, renderBufferId);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, width,
height);
glBindFramebuffer(GL_FRAMEBUFFER, framebufferObjectId);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
GL_TEXTURE_2D, textureId, 0);
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_RENDERBUFFER, renderBufferId);
GLenum frameBufferRes = glCheckFramebufferStatus(GL_DRAW_FRAMEBUFFER);
cout << getFrameBufferStatusString(frameBufferRes)<<endl;
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
return framebufferObjectId;
}
```