Shadows

CS425: Computer Graphics I

Fabio Miranda

https://fmiranda.me

UIC COMPUTER SCIENCE

Overview

- Shadow projection
- Shadow Mapping
- Shadow Volume
- Shadow Accumulation

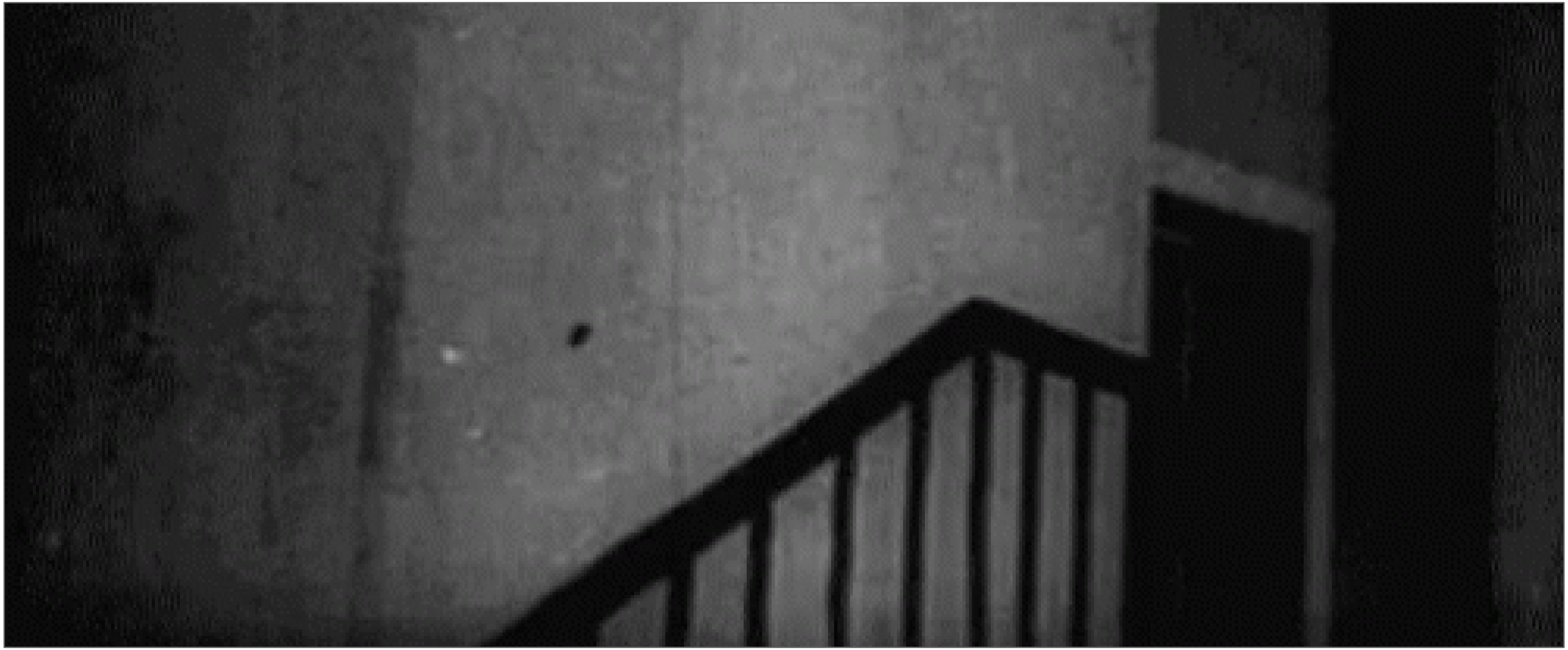
Shadows

Shadows

Shadows

Shadows

Shadows

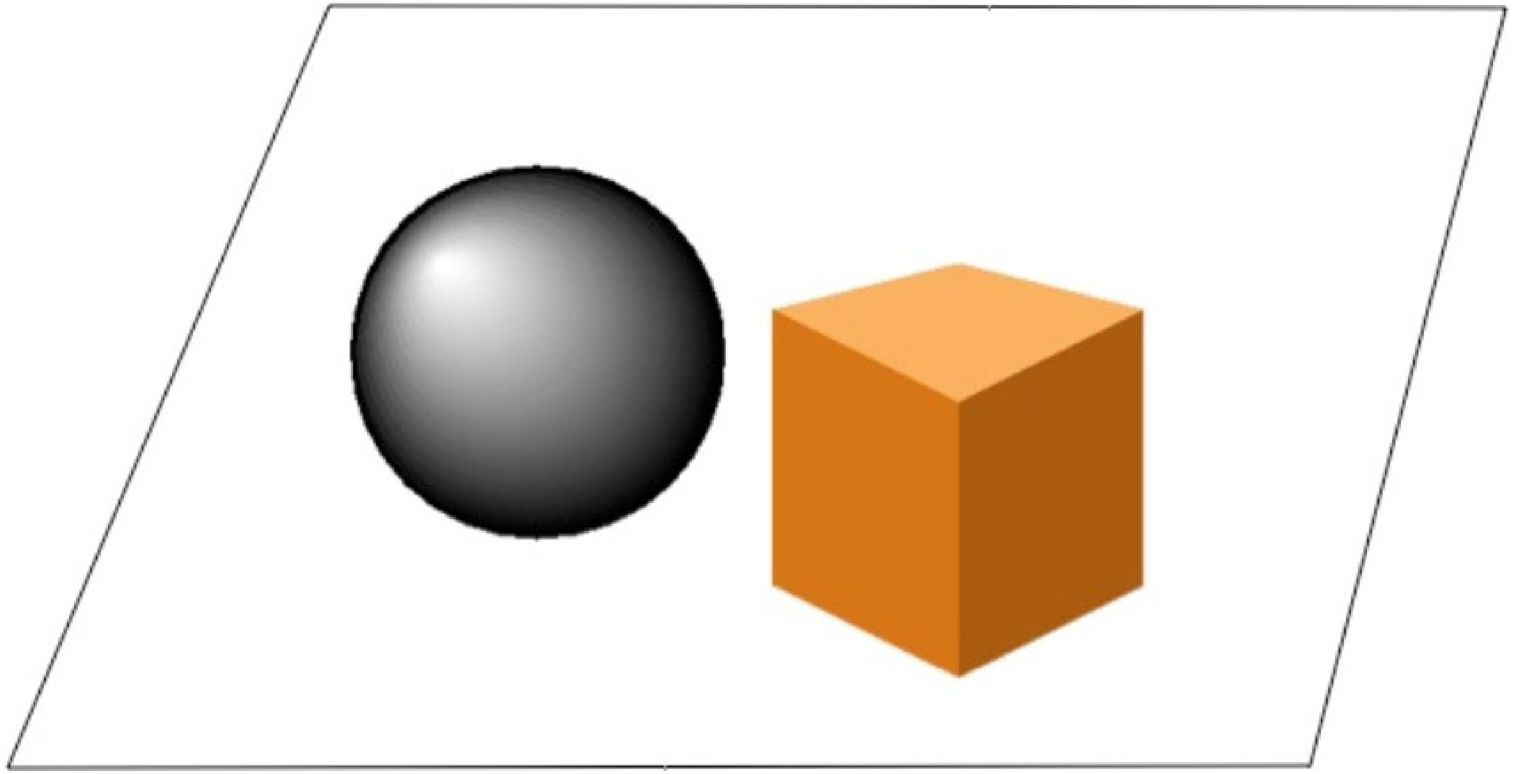


Shadows

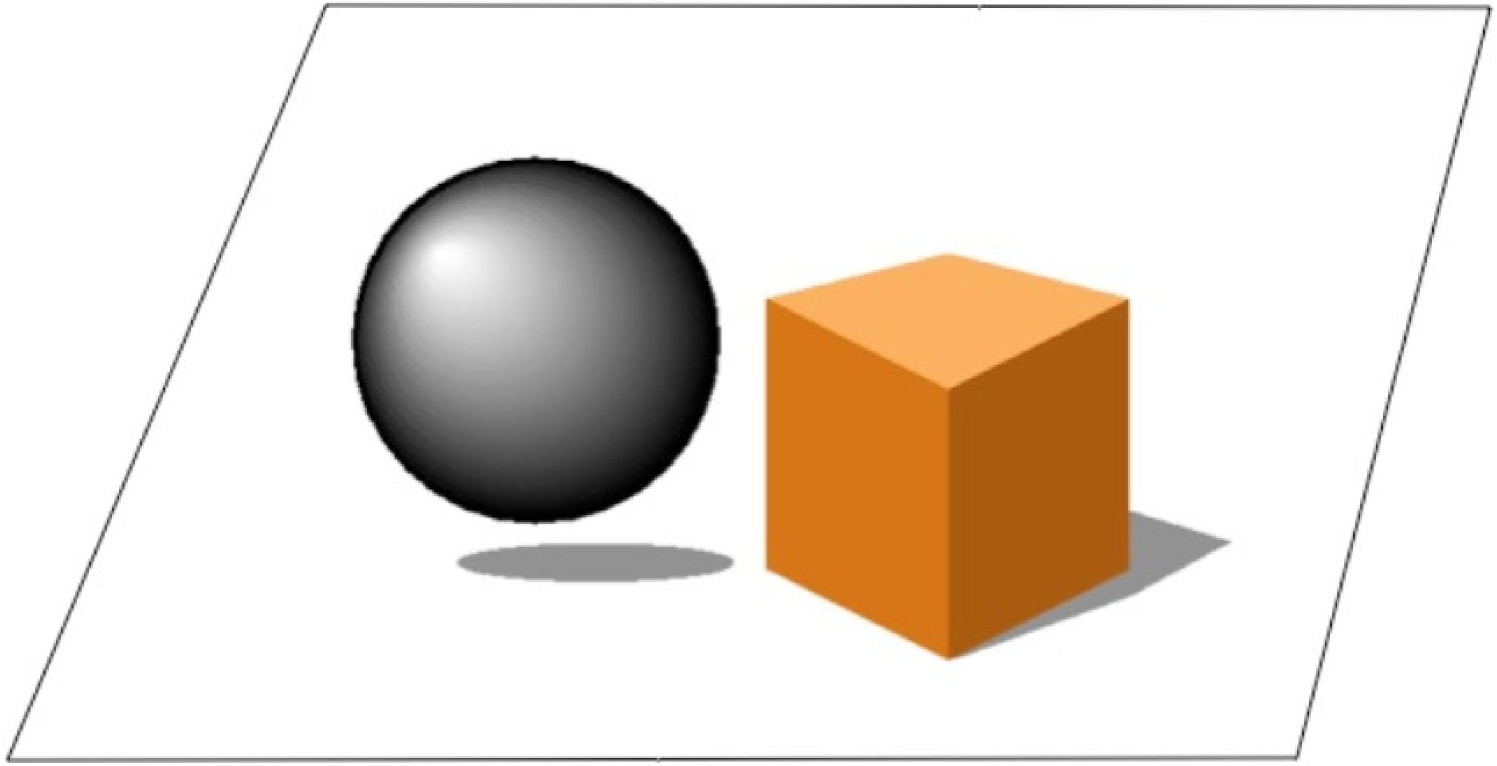
Importance of shadows

- Add realism to the scene.
- Shape, volume of the object.
- Position of light source.
- Depth perception.

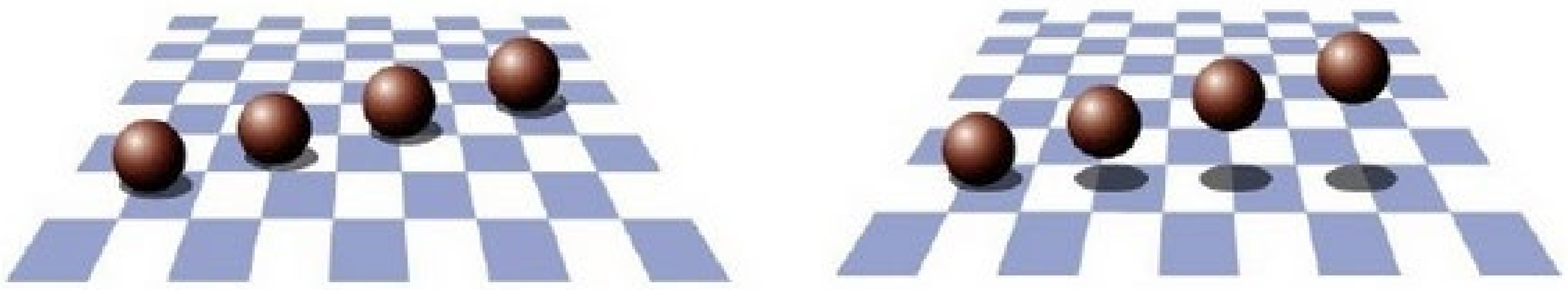
Importance of shadows



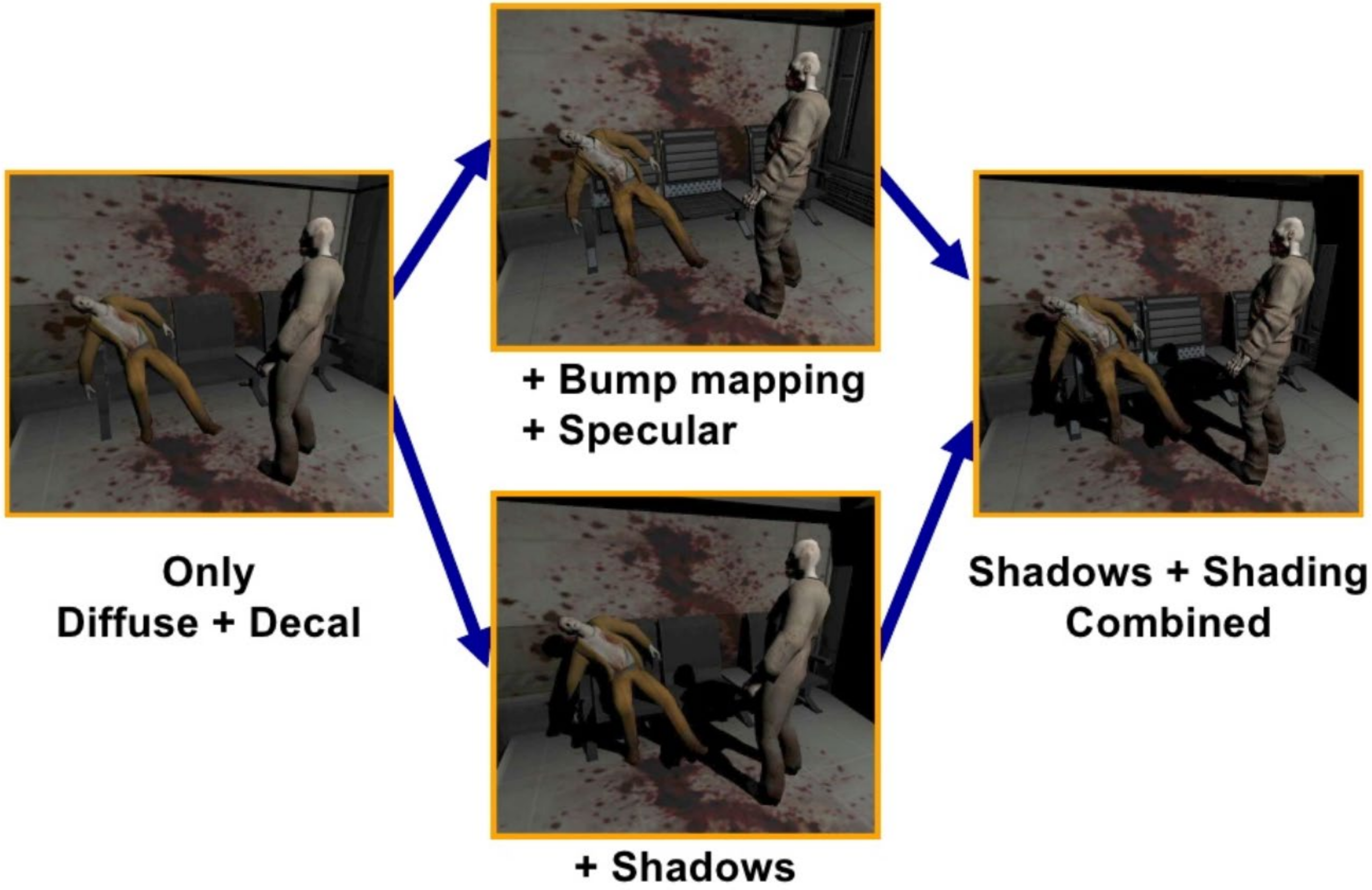
Importance of shadows



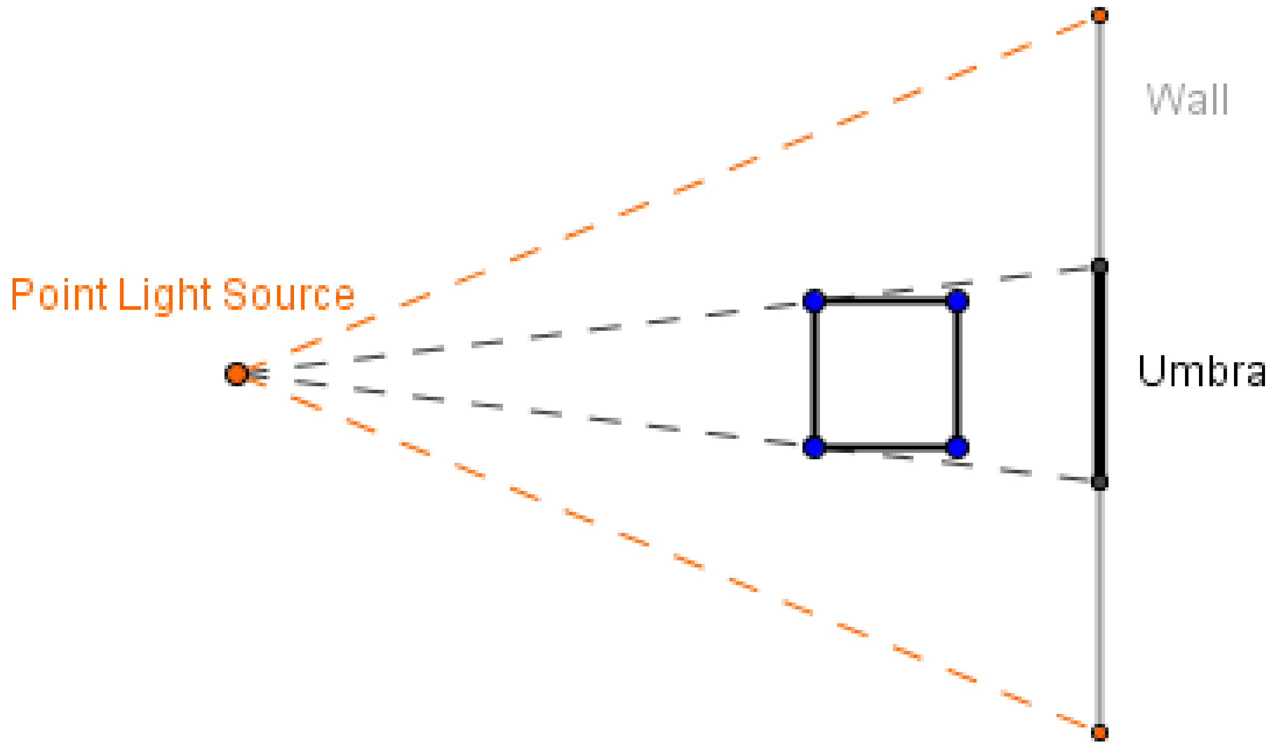
Importance of shadows



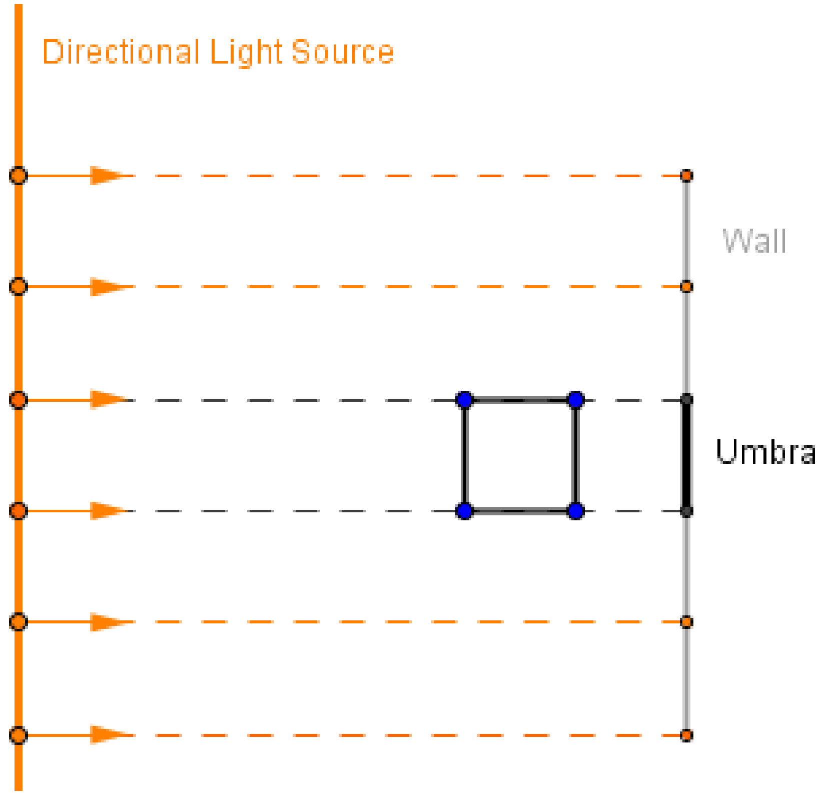
Shading and shadows



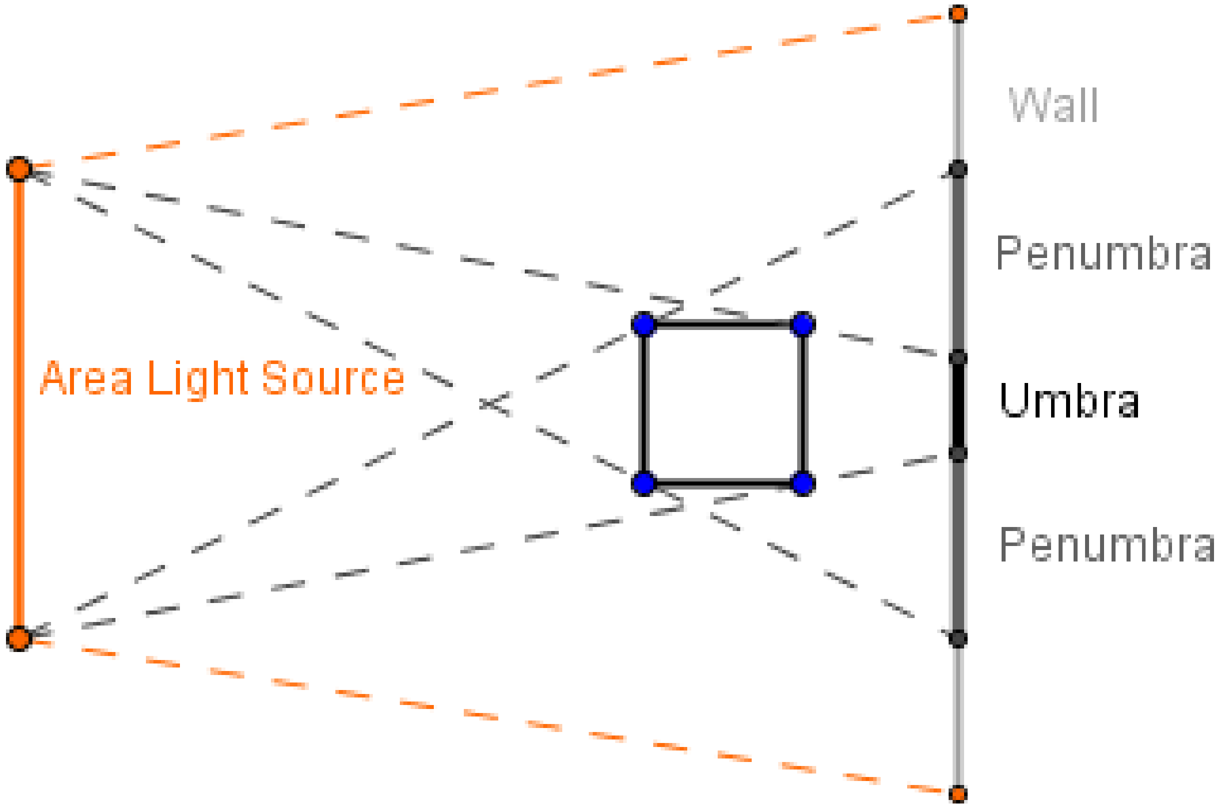
Shadow components



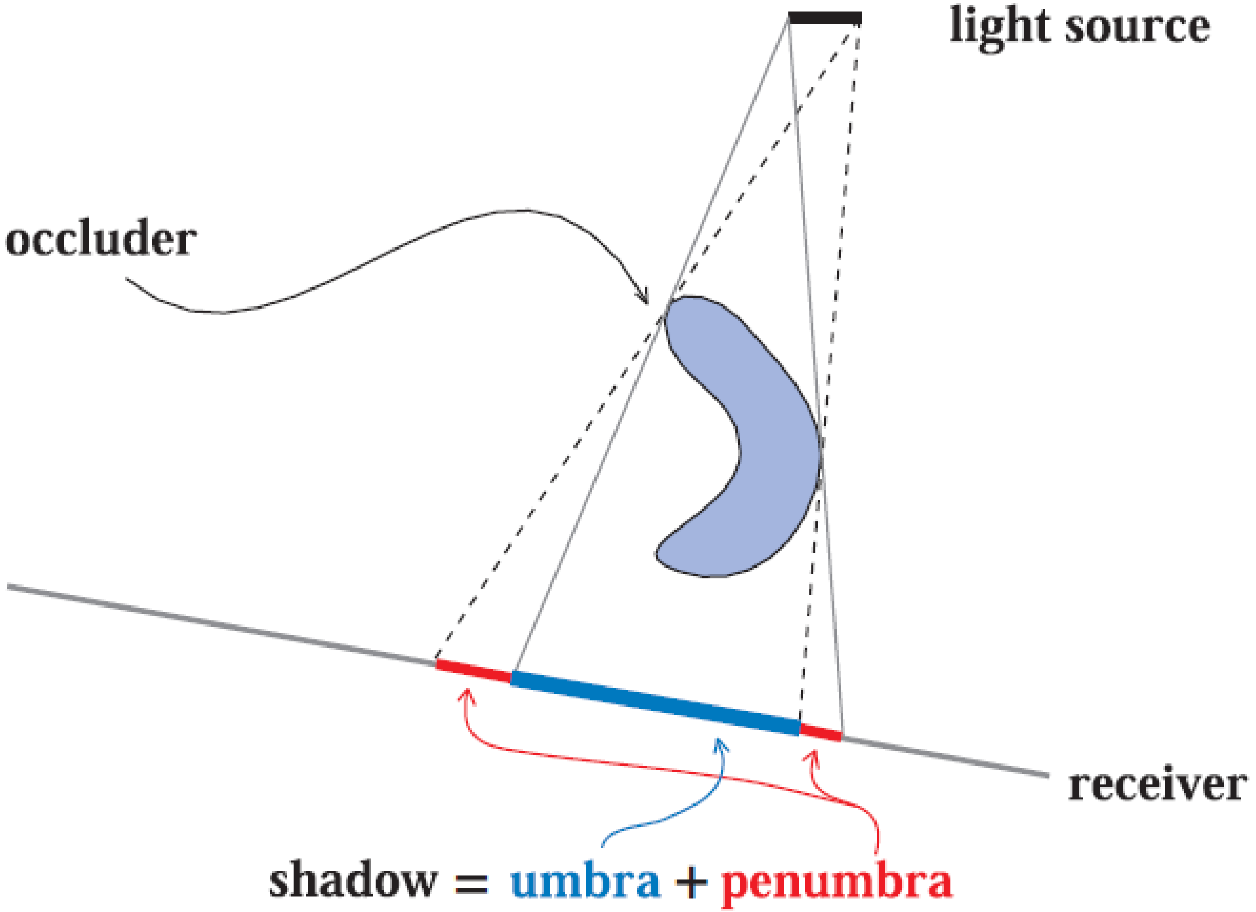
Shadow components



Shadow components

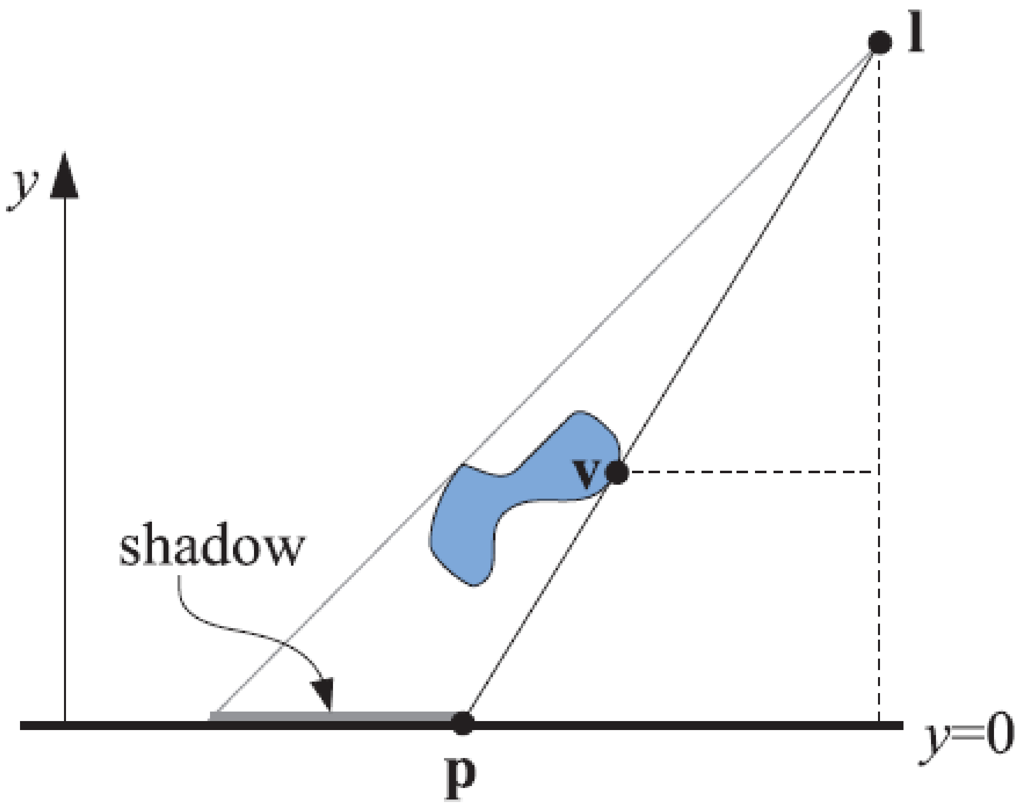


Goal



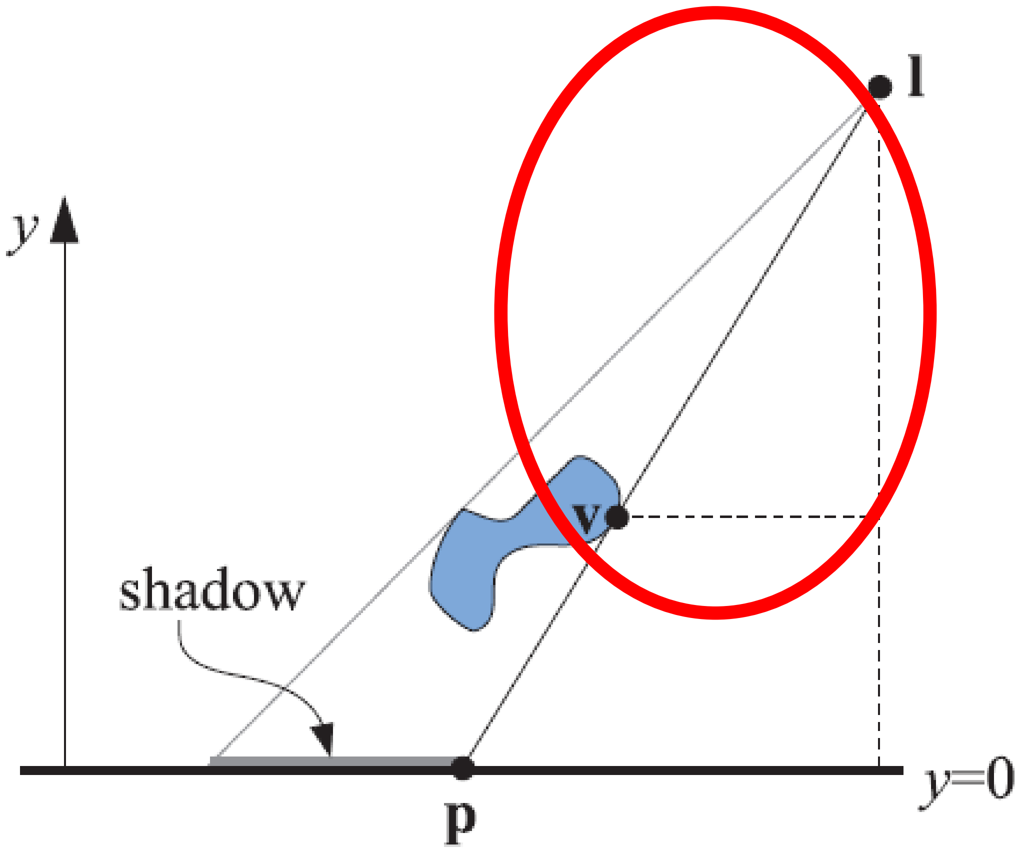
UIC COMPUTER SCIENCE

Planar shadows

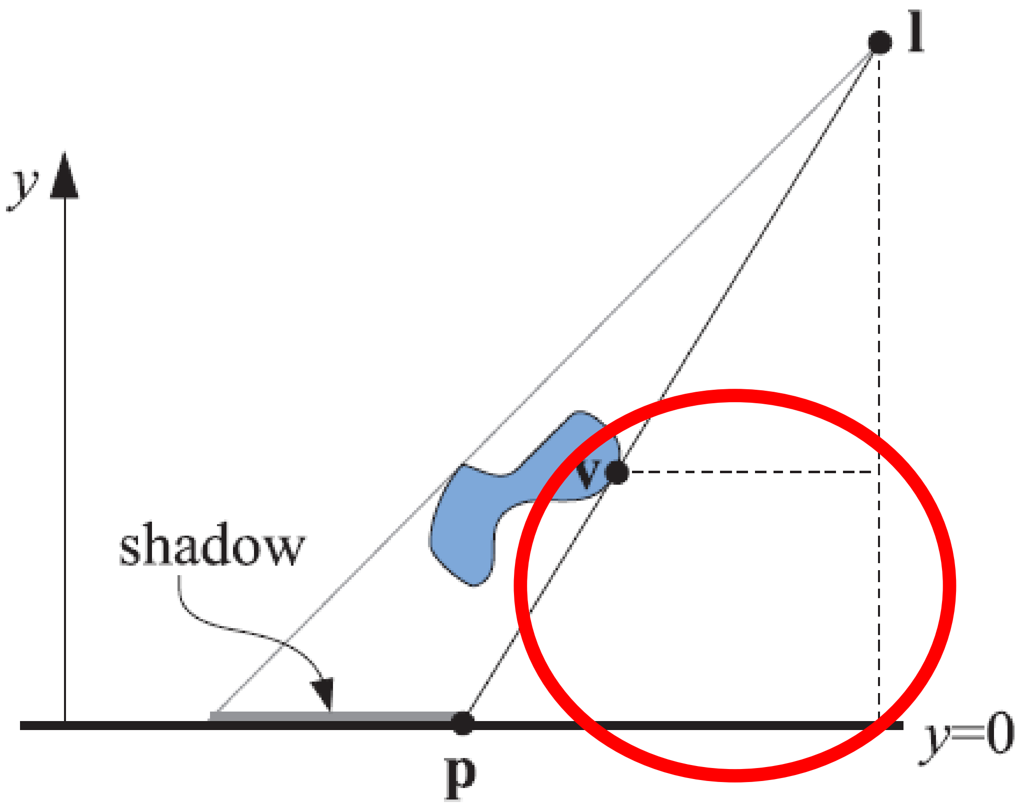


p?

Planar shadows

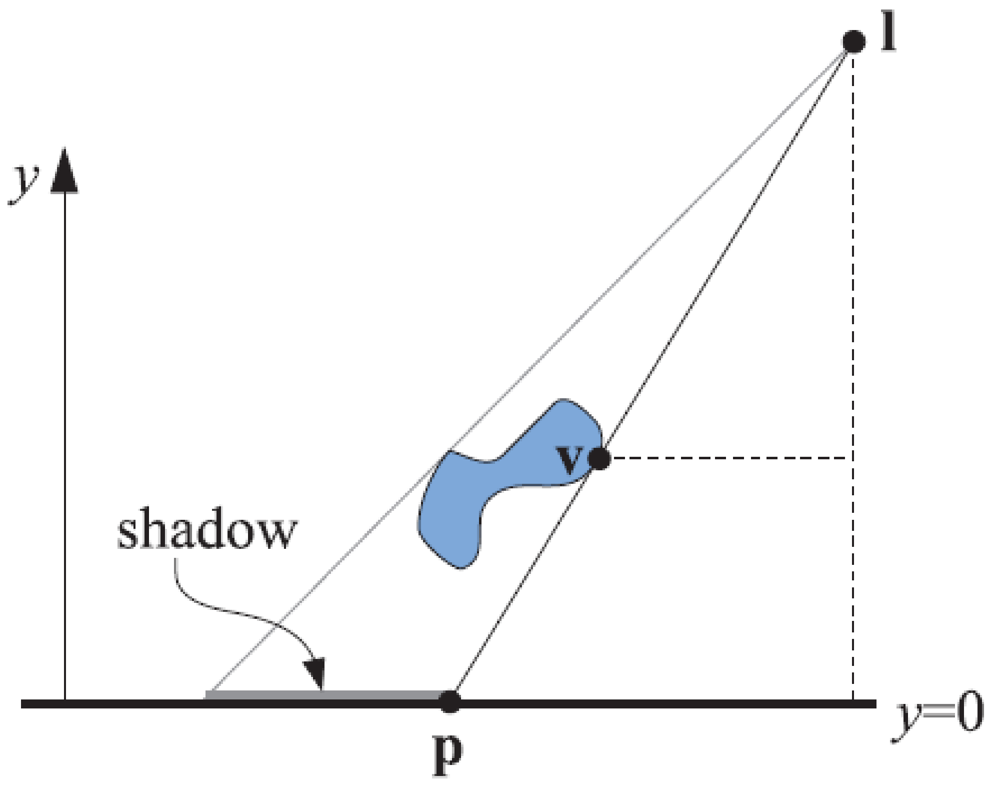


Planar shadows



UIC COMPUTER SCIENCE

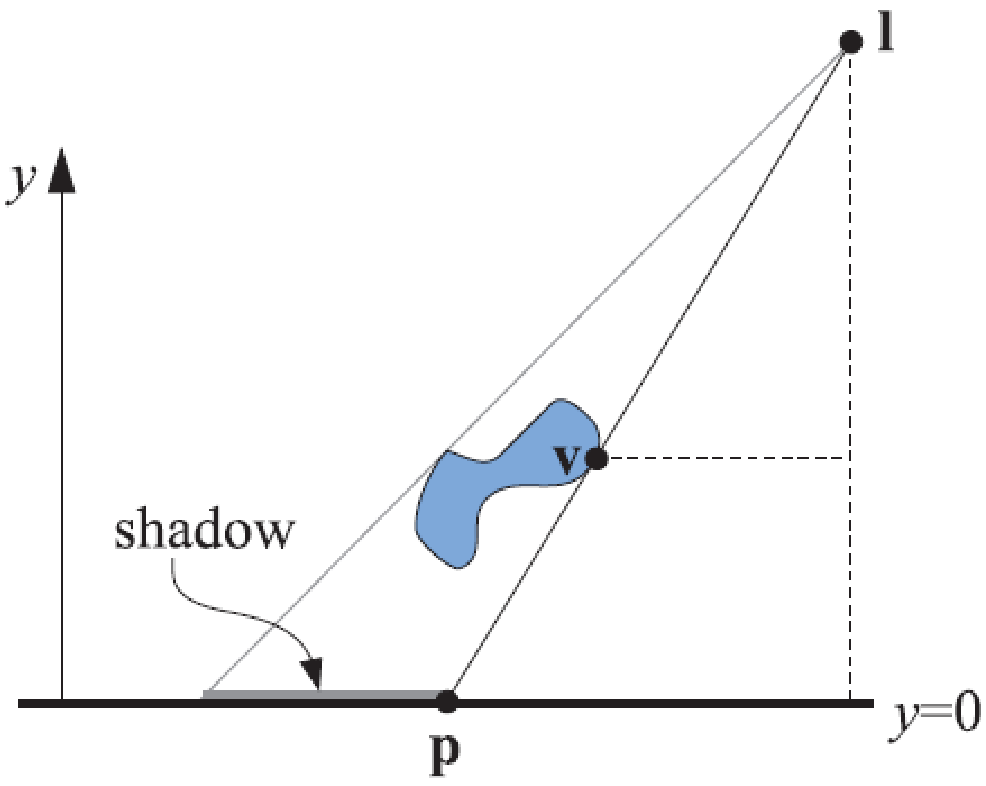
Planar shadows



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

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

Planar shadows



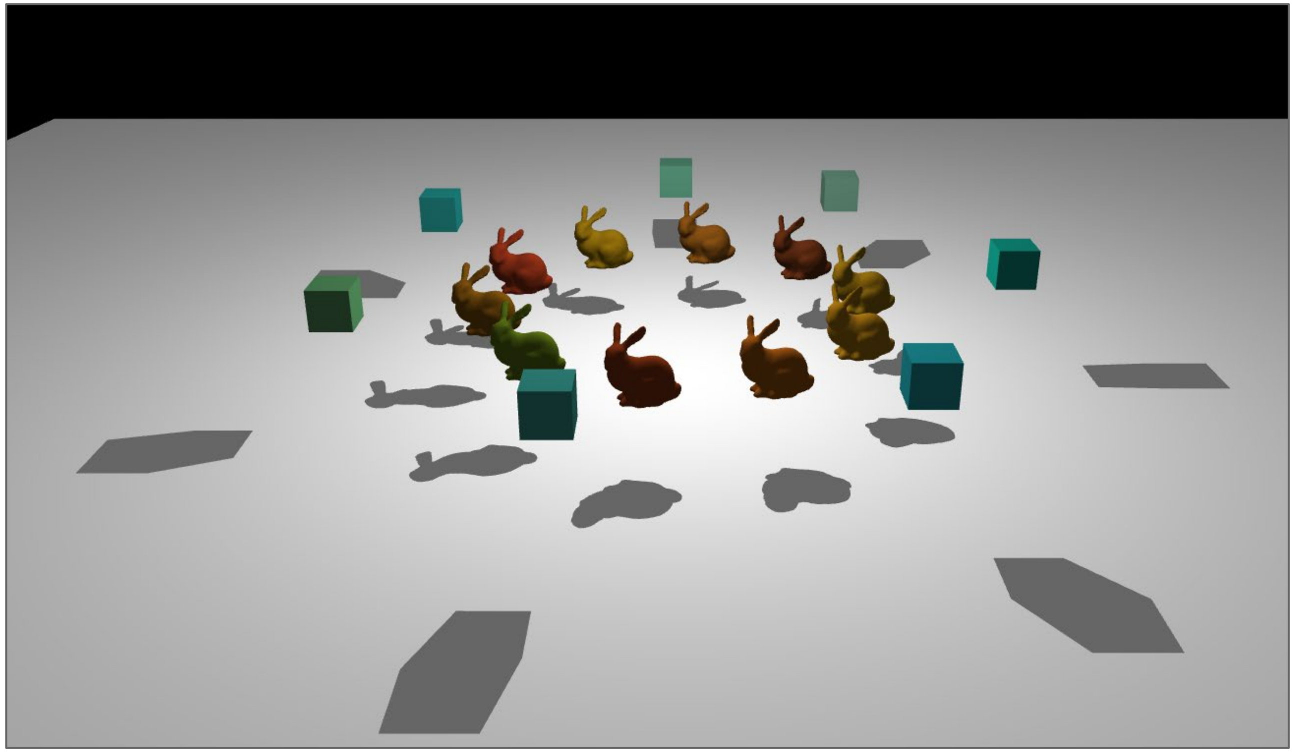
$$\mathbf{M}\mathbf{v} = \mathbf{p}$$

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

Planar shadow: steps

- Render receiving plane
- Render occluder, projecting with matrix M
- Render occluder

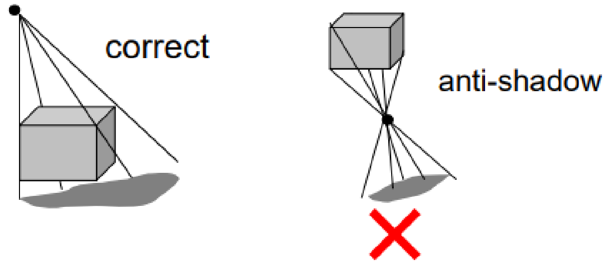
Planar shadows: example



https://erkaman.github.io/planar_proj_shadows/planar_proj_shadows.html

Planar Shadows: shortcomings

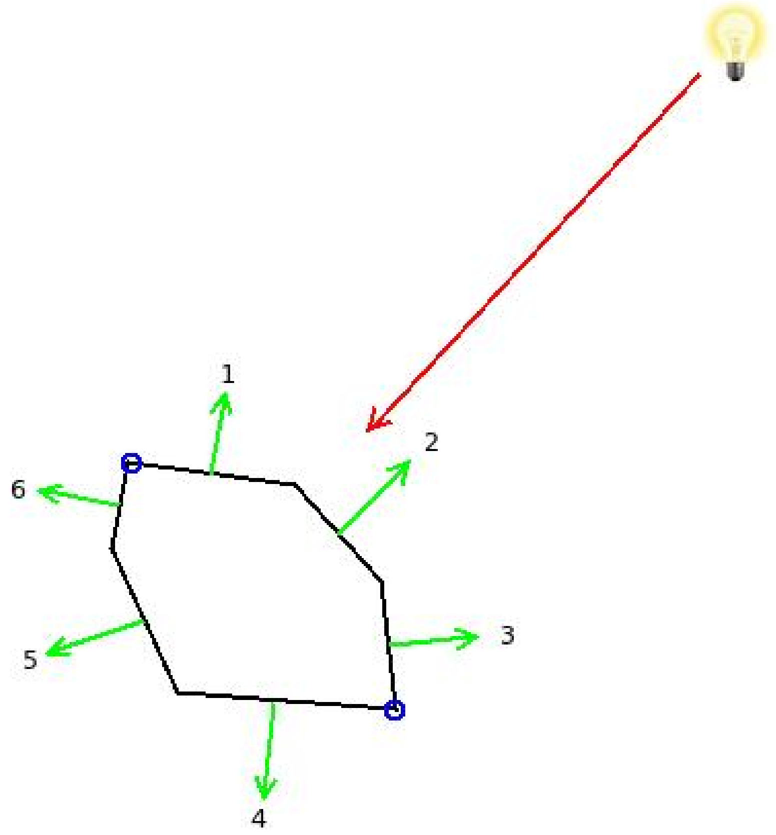
- Z-fighting:
  - Precision problem between receiving plane and occlude.
  - Use polygon offset (glPolygonOffset).
- Restricted to planar objects.
- Anti-shadows.



Shadow volume

UIC COMPUTER SCIENCE

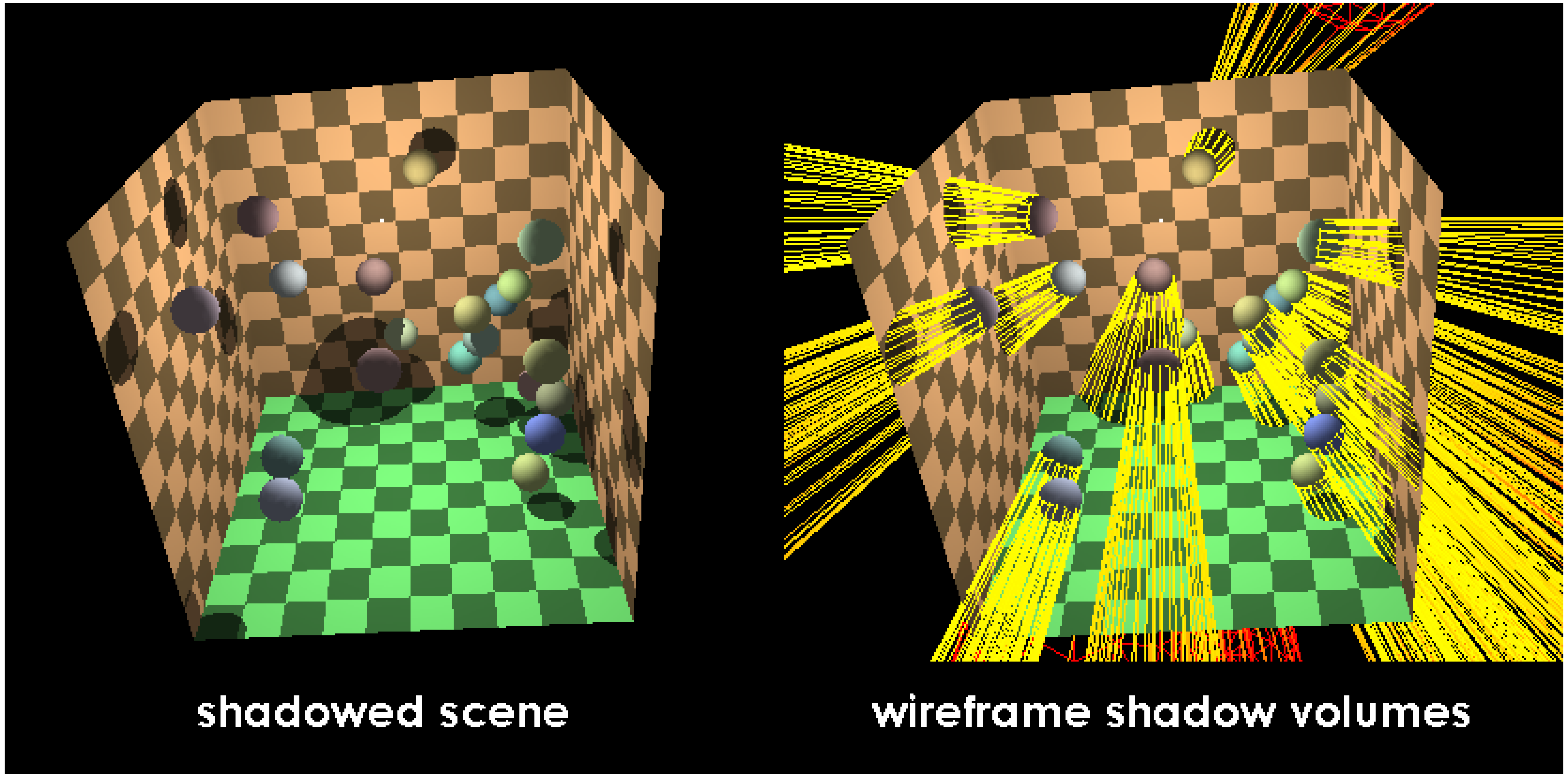
Silhouette detection



Silhouette:

- If one neighboring edge is facing the light, the other not.

Silhouette detection



Silhouette detection

How to find the edges?

Geometry shader

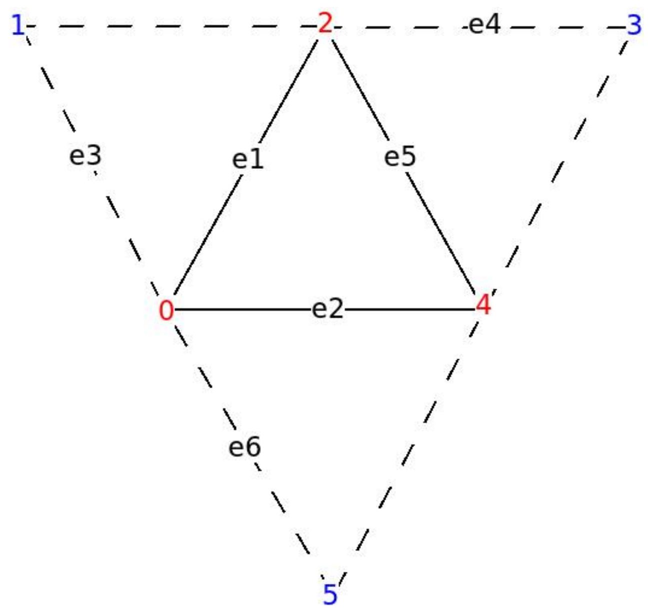
```
#version 330

layout (location = 0) in vec3 Position;
layout (location = 1) in vec2 TexCoord;
layout (location = 2) in vec3 Normal;

out vec3 WorldPos0;

uniform mat4 gWVP;
uniform mat4 gWorld;

void main()
{
    vec4 PosL = vec4(Position, 1.0);
    gl_Position = gWVP * PosL;
    WorldPos0 = (gWorld * PosL).xyz;
}
```

Vertex Shader

Silhouette detection

How to find the edges?

Geometry shader




```
#version 330

layout (triangles_adjacency) in;
layout (line_strip, max_vertices = 6) out;

in vec3 WorldPos0[];

uniform vec3 gLightPos;

void main()
{
    vec3 e1 = WorldPos0[2] - WorldPos0[0];
    vec3 e2 = WorldPos0[4] - WorldPos0[0];
    vec3 e3 = WorldPos0[1] - WorldPos0[0];
    vec3 e4 = WorldPos0[3] - WorldPos0[2];
    vec3 e5 = WorldPos0[4] - WorldPos0[2];
    vec3 e6 = WorldPos0[5] - WorldPos0[0];

    vec3 Normal = cross(e1,e2);
    vec3 LightDir = gLightPos - WorldPos0[0];

    if (dot(Normal, LightDir) > 0.00001) {

        Normal = cross(e3,e1);

        if (dot(Normal, LightDir) <= 0) {
            // Silhouette!!!
        }

        Normal = cross(e4,e5);
        LightDir = gLightPos - WorldPos0[2];

        if (dot(Normal, LightDir) <=0) {
            // Silhouette!!!
        }

        Normal = cross(e2,e6);
        LightDir = gLightPos - WorldPos0[4];

        if (dot(Normal, LightDir) <= 0) {
            // Silhouette!!!
        }
    }
}
```


UIC COMPUTER SCIENCE

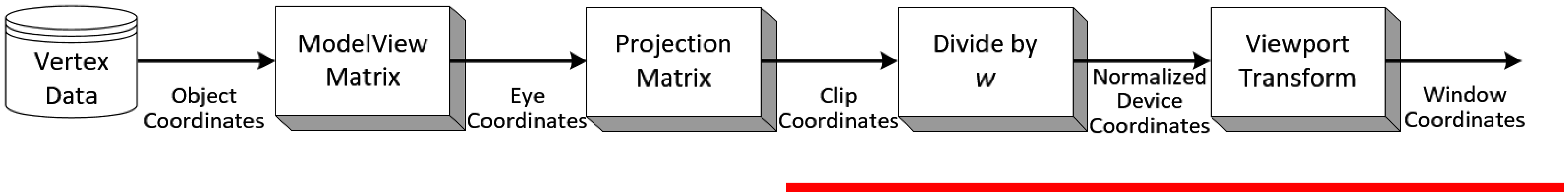
Projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 4 \end{bmatrix}$$

$$\frac{(2,3,4)}{4} = (0.5, 0.75, 1)$$

$$\frac{(2,3,4)}{0} = (\infty, \infty, \infty)$$

Projection

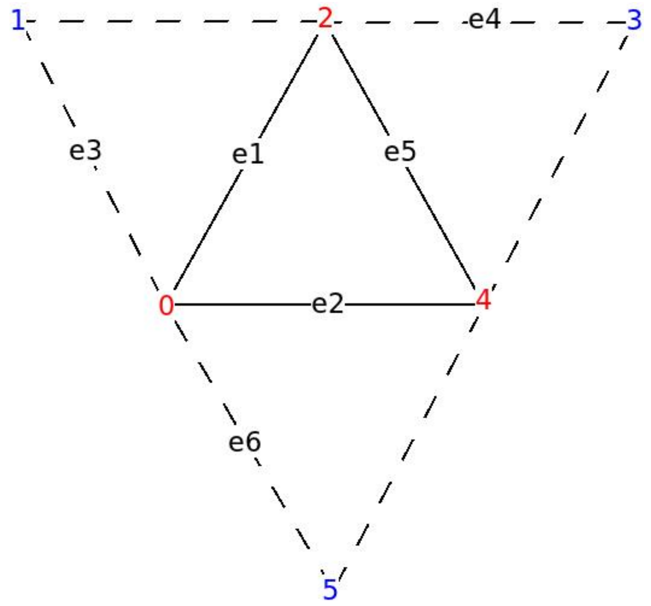


Vertex
Data
Object
Coordinates
ModelView
Matrix
Eye
Coordinates
Projection
Matrix
Clip
Coordinates
Divide by
w
Normalized
Device
Coordinates
Viewport
Transform
Window
Coordinates

Silhouette detection

How to create volume?

Geometry shader




```
// Emit a quad using a triangle strip
void EmitQuad(vec3 StartVertex, vec3 EndVertex)
{
    // Vertex #1: the starting vertex (just a tiny bit below the original edge)
    vec3 LightDir = normalize(StartVertex - gLightPos);
    gl_Position = gWVP * vec4((StartVertex + LightDir * EPSILON), 1.0);
    EmitVertex();

    // Vertex #2: the starting vertex projected to infinity
    gl_Position = gWVP * vec4(LightDir, 0.0);
    EmitVertex();

    // Vertex #3: the ending vertex (just a tiny bit below the original edge)
    LightDir = normalize(EndVertex - gLightPos);
    gl_Position = gWVP * vec4((EndVertex + LightDir * EPSILON), 1.0);
    EmitVertex();

    // Vertex #4: the ending vertex projected to infinity
    gl_Position = gWVP * vec4(LightDir , 0.0);
    EmitVertex();

    EndPrimitive();
}
```

Projection

Far plane at infinity

$$P = \begin{pmatrix} \frac{1}{aspectRatio \cdot tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{-2 \cdot far \cdot near}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Projection

Far plane at infinity

$$\lim_{far \to \infty} -\frac{far + near}{far - near} = \lim_{far \to \infty} -\frac{\frac{far}{far} + \frac{near}{far}}{\frac{far}{far} - \frac{near}{far}} = -\frac{1+0}{1-0} = -1$$

$$P = \begin{pmatrix} \frac{1}{aspectRatio \cdot tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{-2 \cdot far \cdot near}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Projection

Far plane at infinity

$$\lim_{far \to \infty} -\frac{far + near}{far - near} = \lim_{far \to \infty} -\frac{\frac{far}{far} + \frac{near}{far}}{\frac{far}{far} - \frac{near}{far}} = -\frac{1+0}{1-0} = -1$$

$$P = \begin{pmatrix} \frac{1}{aspectRatio \cdot tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{-2 \cdot far \cdot near}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

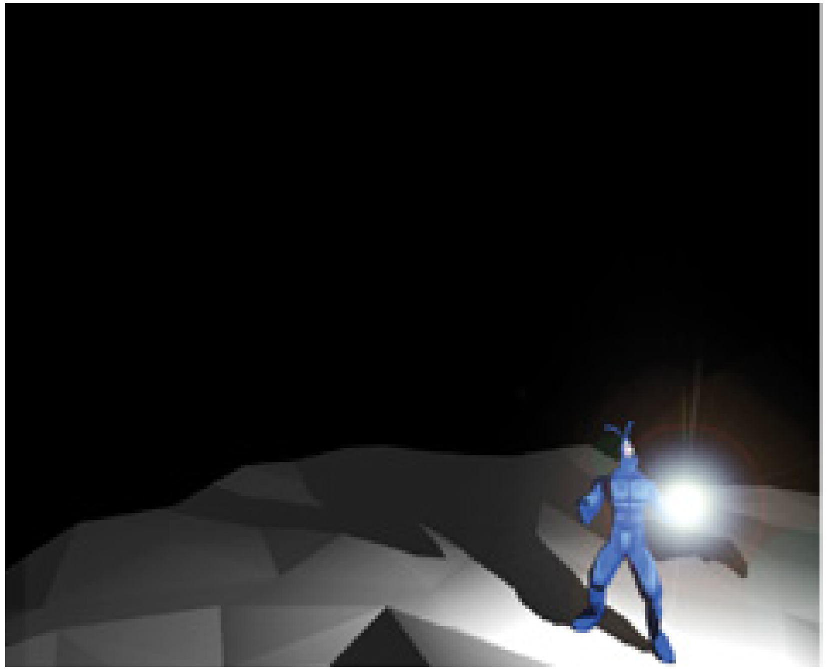
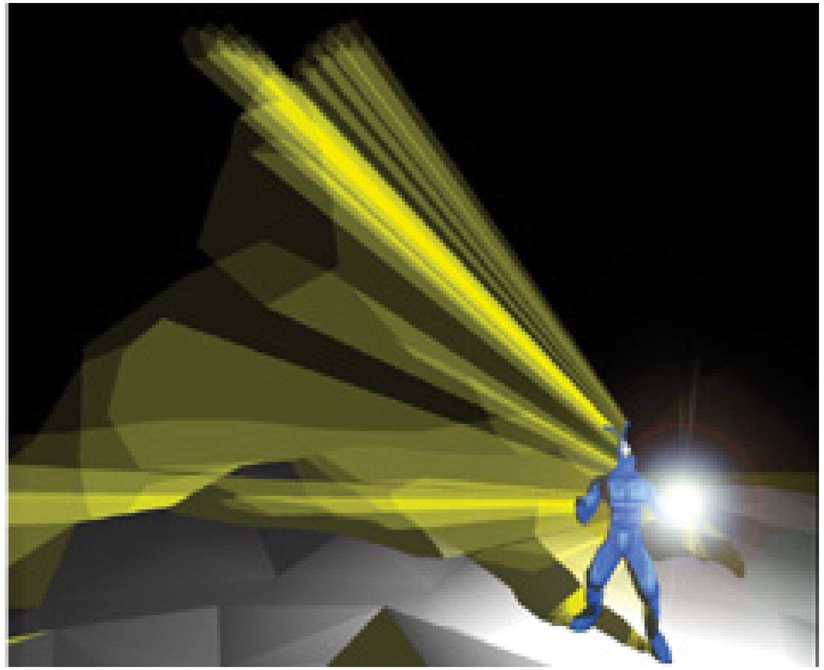
$$\lim_{far \to \infty} -\frac{2 \cdot far \cdot near}{far - near} = \lim_{far \to \infty} -\frac{\frac{2 \cdot far \cdot near}{far}}{\frac{far}{far} - \frac{near}{far}} = -\frac{2 \cdot near}{1-0} = -2 \cdot near$$

Projection

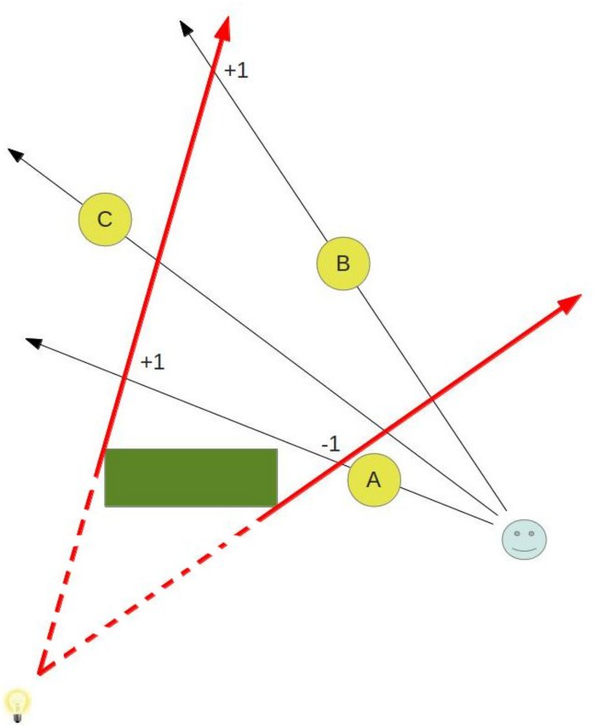
Far plane at infinity

$$P_{inf} = \begin{pmatrix} \frac{1}{aspectRatio \cdot tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & -1 & -2 \cdot near \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Shadow volume

Shadow volume

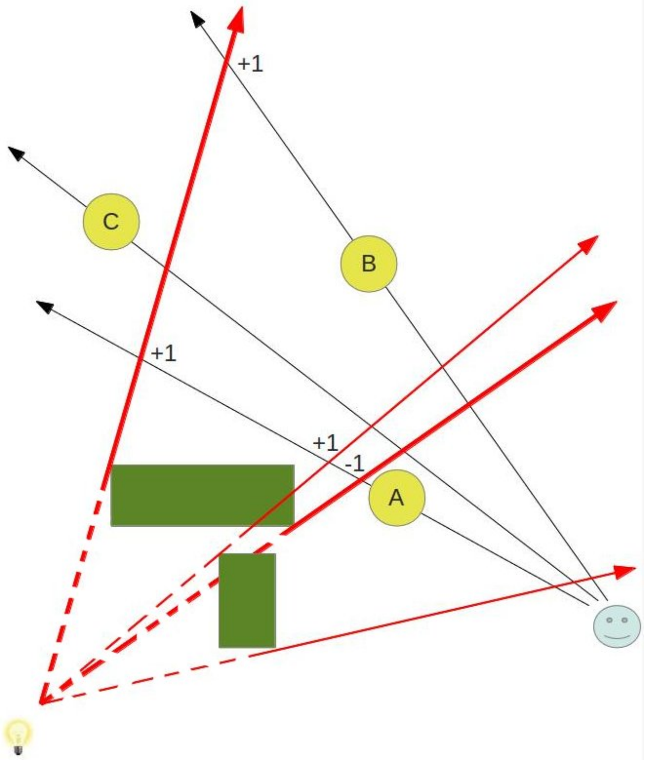


ogldev.org

- Trace rays
  - Ray enters volume
    - Increase counter
  - Ray exists volume
    - Decreases counter
  - Shadow: counter different than zero

Point *P* is in shadow if and only if there were more entering intersections than exiting intersections along a ray to infinity.

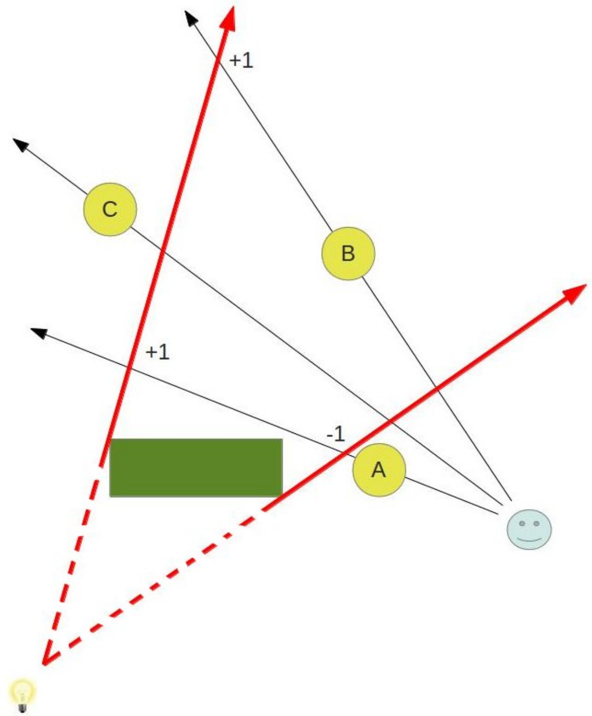
Shadow volume



- Trace rays
  - Ray enters volume
    - Increase counter
  - Ray exists volume
    - Decreases counter
  - Shadow: counter different than zero

Point *P* is in shadow if and only if there were more entering intersections than exiting intersections along a ray to infinity.

Shadow volume



- For each object, determine its shadow volume
- Render back facing polygons of volumes into stencil buffer
  - Depth test fail: increment
- Render front facing polygons of volume into stencil buffer
  - Depth test fail: decrement

Point *P* is in shadow if and only if there were more entering intersections than exiting intersections along a ray to infinity.

Shadow volume

- Advantages
  - Self-shadowing
  - Everything can shadow everything, including self
- Disadvantages
  - Silhouette computation required
  - Slow on scenes with polygons with large number of triangles

Shadow maps

- Image-space shadow determination.
- Leverages GPU hardware:
  - Depth buffering + texture mapping
- Two steps algorithm:
  - First, render scene from light's point of view.
  - Second, render scene from eye's point of view.

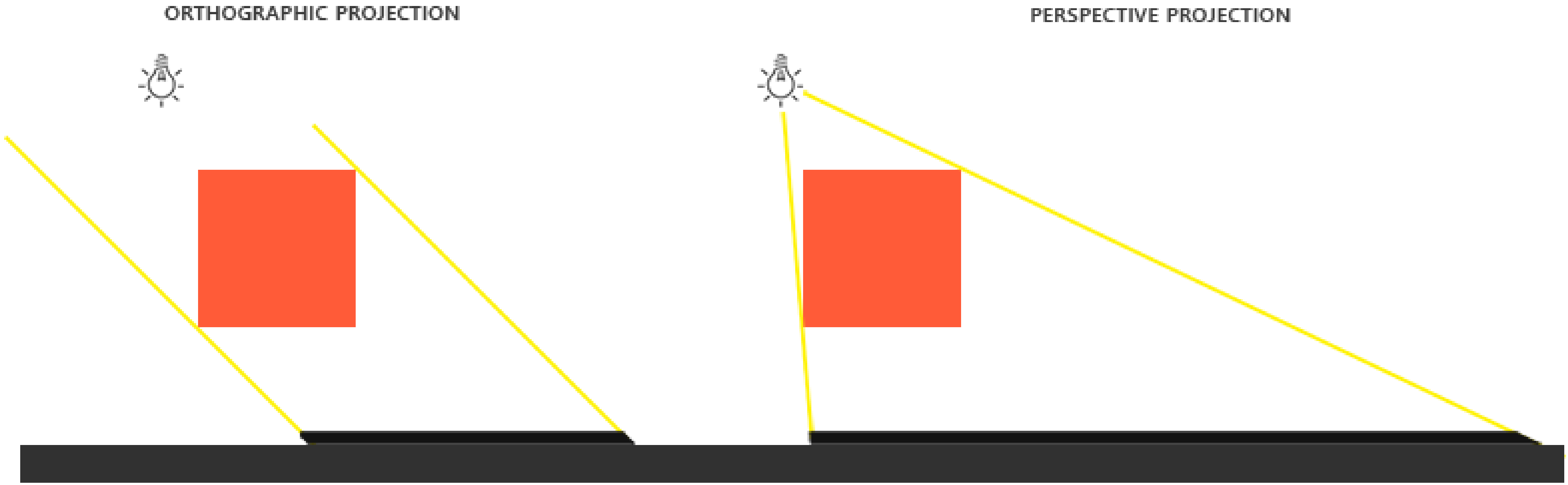
Shadow maps

- First step:
  - Render from light’s point of view:
    - Result stored in a depth buffer, as a shadow map.
    - A 2D function indicating the depth of the closest pixel to the light.
  - Shadow map is used in the second step.

Shadow maps

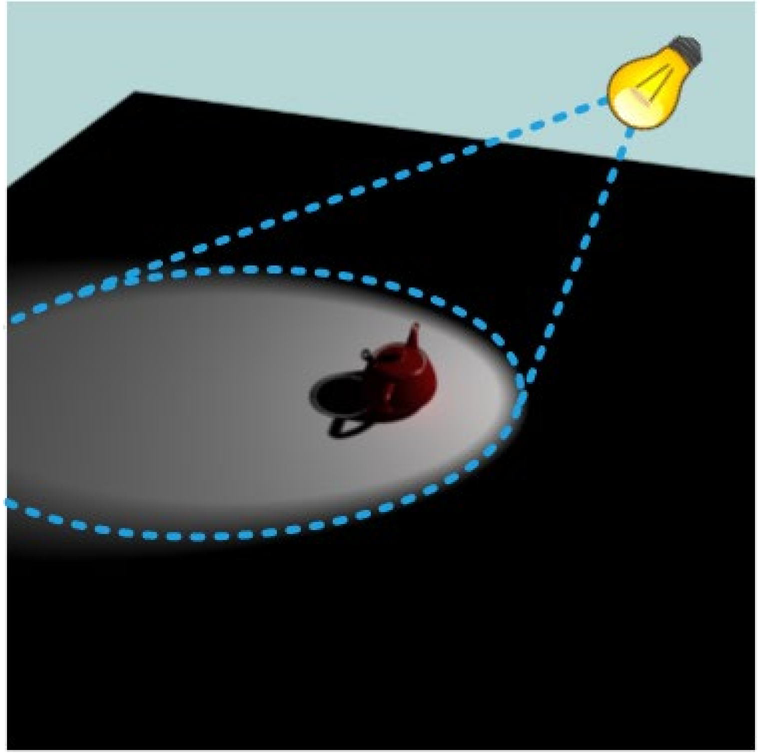
- Second step:
  - Render from eye's point of view:
    - For each fragment, determine its position in the light space.
    - Compare depth value at light position with the depth value from shadow map.
  - Two values:
    - A: z value of fragment in light space.
    - B: z value of fragment in shadow map.
    - B>A: shadow
    - A>B: no shadow

Shadow maps

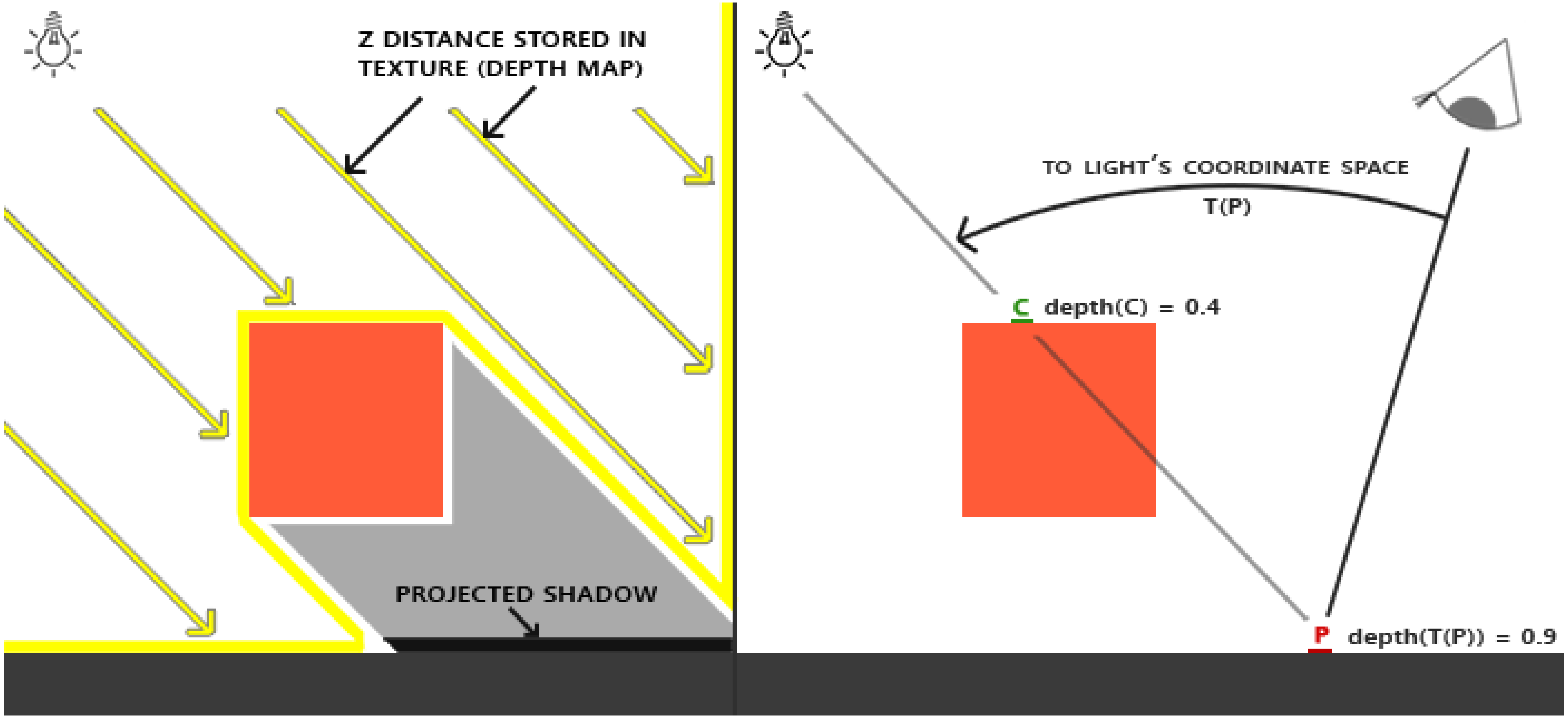


UIC COMPUTER SCIENCE

Shadow test



Shadow test



Z DISTANCE STORED IN
TEXTURE (DEPTH MAP)
PROJECTED SHADOW
TO LIGHT'S COORDINATE SPACE
T(P)
C depth(C) = 0.4
P depth(T(P)) = 0.9

Shadow maps + GL

1. Creating shadow map framebuffer for rendering shadow map:

```
unsigned int depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

2. Creating shadow map texture with size 1024

```
const unsigned int SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;

unsigned int depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
             SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Snippets from: https://learnopengl.com/

Shadow maps + GL

3. Attach texture to framebuffer

```
unsigned int depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

```
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_
2D, depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Shadow maps + GL

4. Render scene twice

```
// 1. first render to depth map
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glClear(GL_DEPTH_BUFFER_BIT);
    ConfigureShaderAndMatrices();
    RenderScene();
glBindFramebuffer(GL_FRAMEBUFFER, 0);
// 2. then render scene as normal with shadow mapping (using depth map)
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
ConfigureShaderAndMatrices();
glBindTexture(GL_TEXTURE_2D, depthMap);
RenderScene();
```

Shadow maps + GL

4. Shadow test: vertex shader

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out VS_OUT {
    vec3 FragPos;
    vec3 Normal;
    vec2 TexCoords;
    vec4 FragPosLightSpace;
} vs_out;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;
uniform mat4 lightSpaceMatrix;

void main()
{
    vs_out.FragPos = vec3(model * vec4(aPos, 1.0));
    vs_out.Normal = transpose(inverse(mat3(model))) * aNormal;
    vs_out.TexCoords = aTexCoords;
    vs_out.FragPosLightSpace = lightSpaceMatrix * vec4(vs_out.FragPos, 1.0);
    gl_Position = projection * view * vec4(vs_out.FragPos, 1.0);
}
```

Shadow maps + GL

4. Shadow test: fragment shader

```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // perform perspective divide
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    [...]
}
```

```
projCoords = projCoords * 0.5 + 0.5;
```

```
float closestDepth = texture(shadowMap, projCoords.xy).r;
```

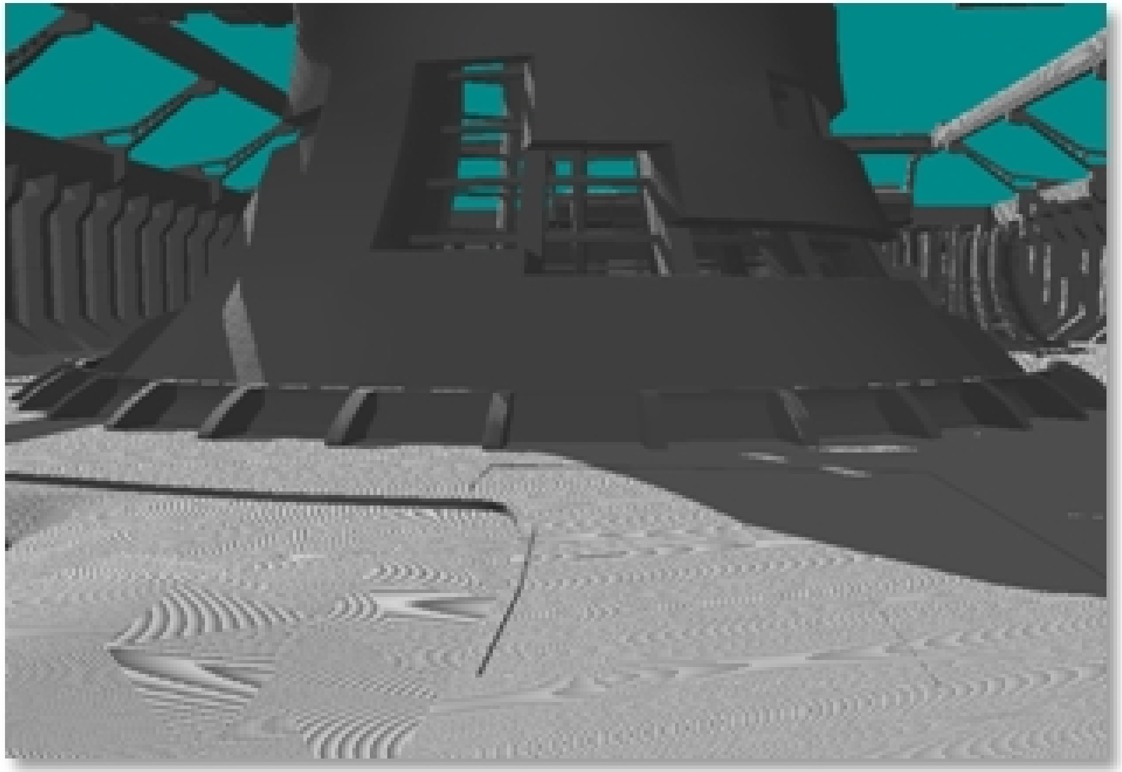
```
float currentDepth = projCoords.z;
float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;
```

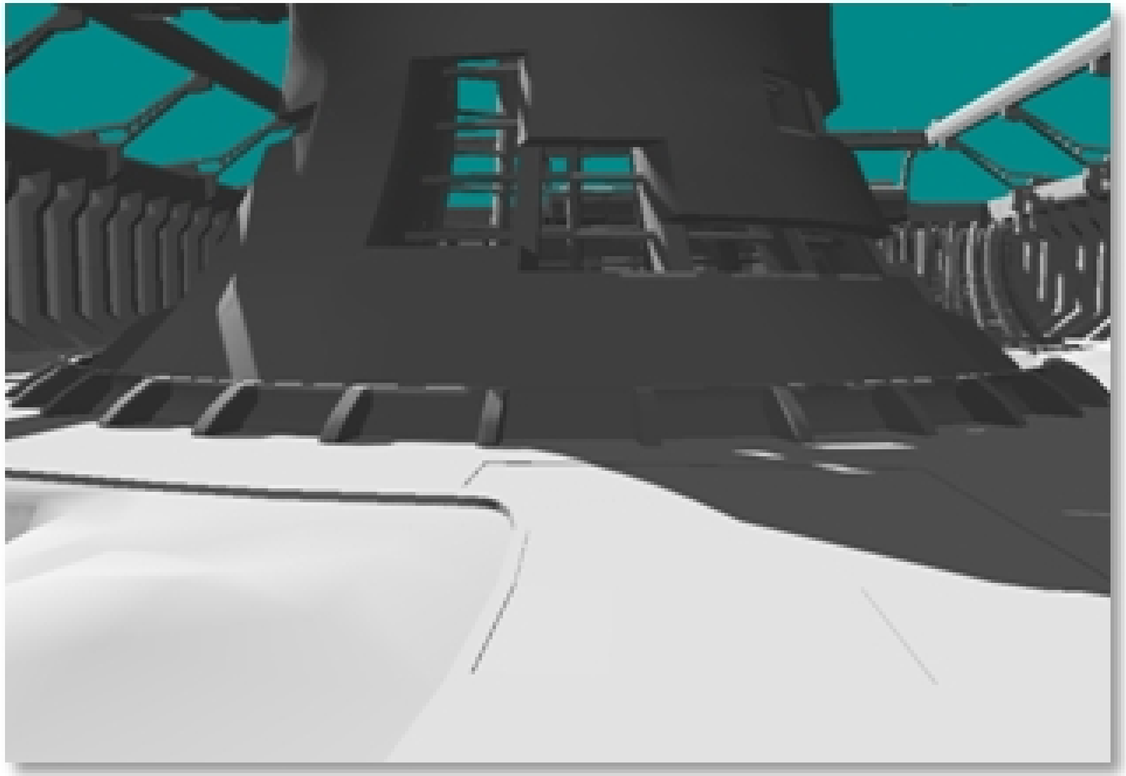
Shadow map result

Shadow maps

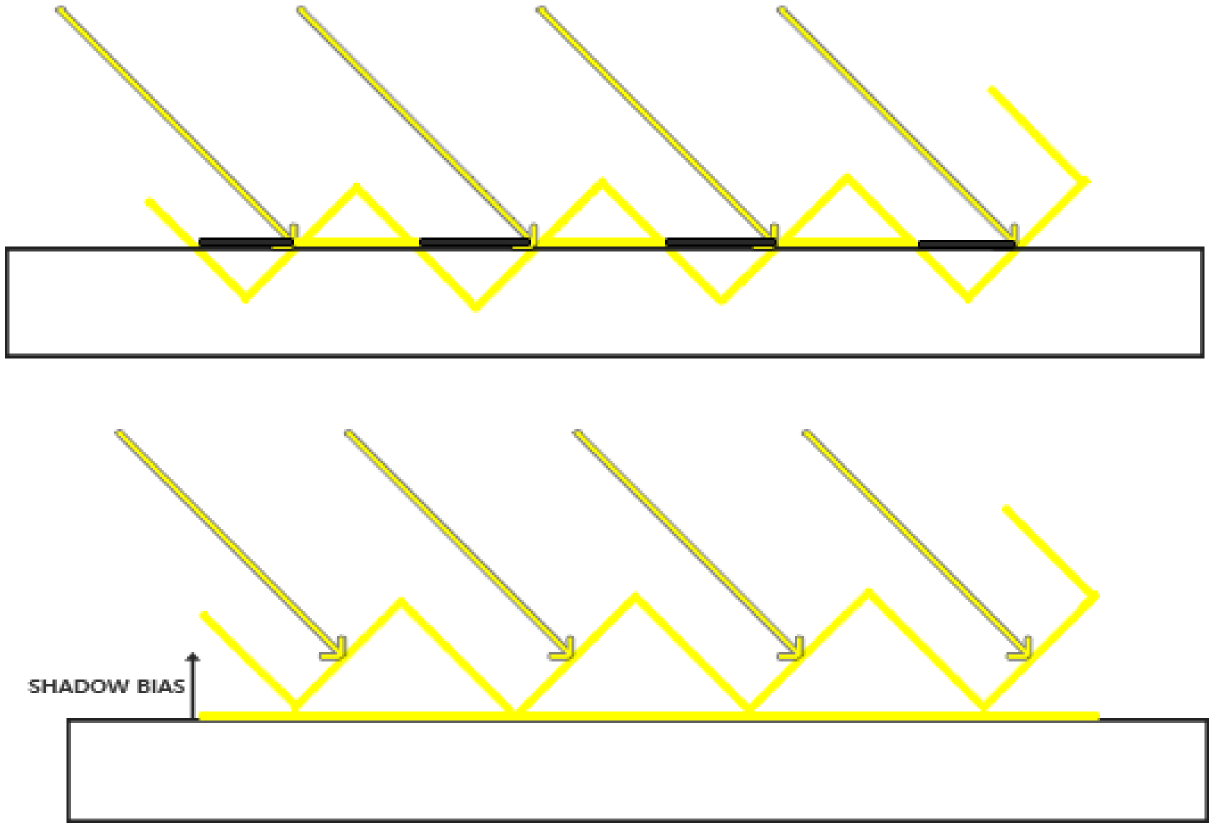
- Advantages:
  - Fast
  - Simple depth map comparison

Shadow acne



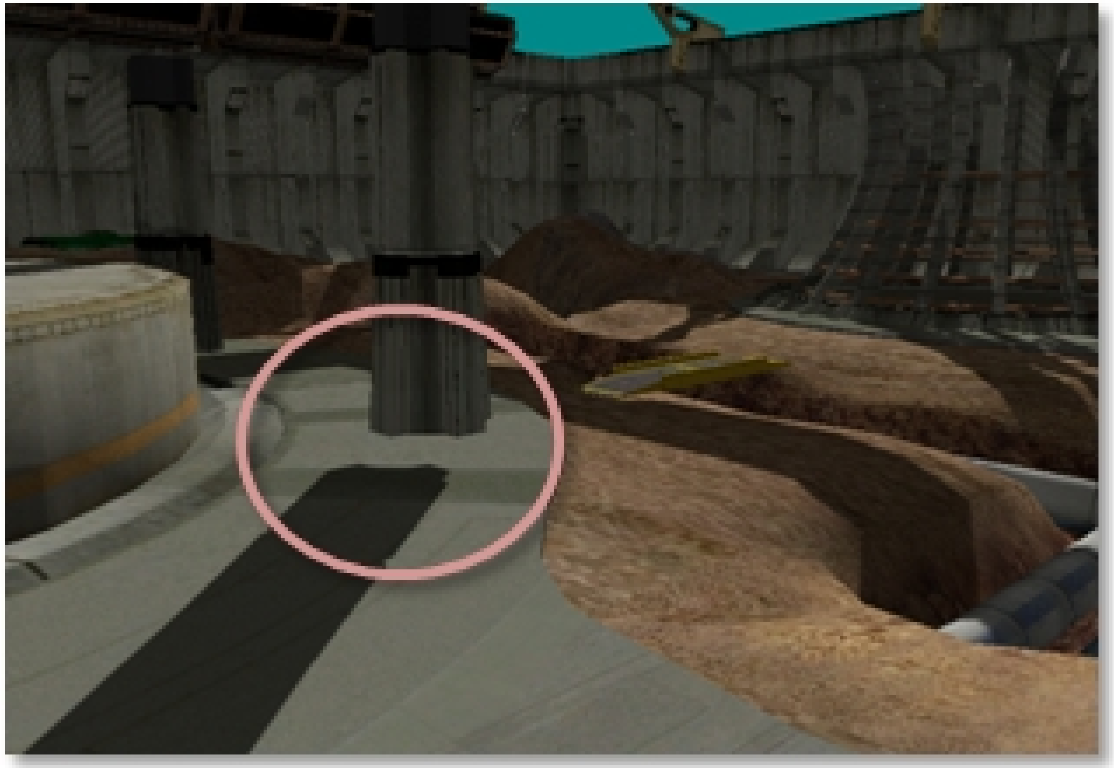


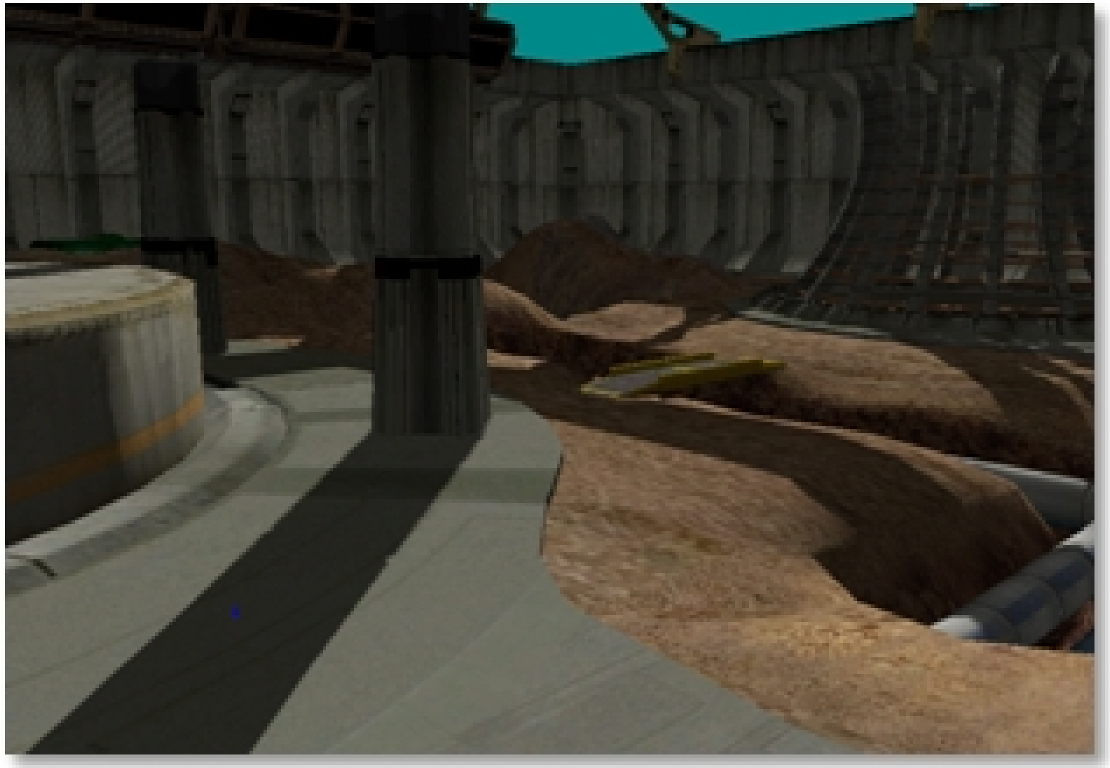
Shadow acne



UIC COMPUTER SCIENCE

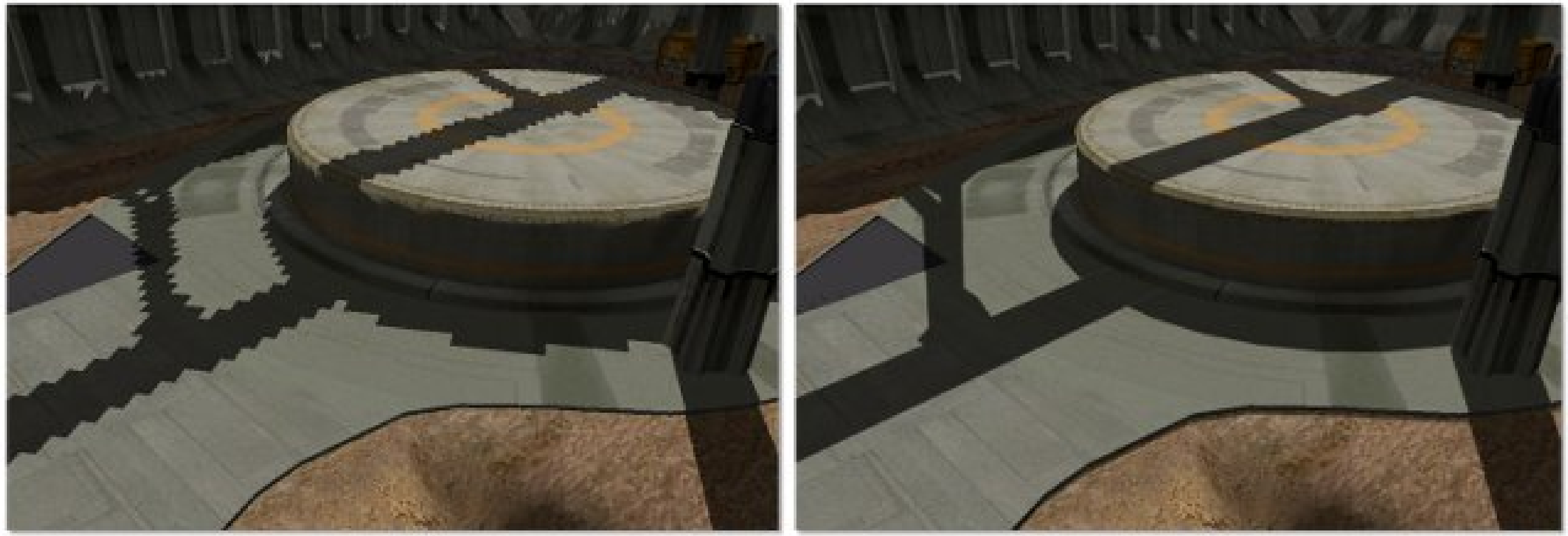
Peter panning





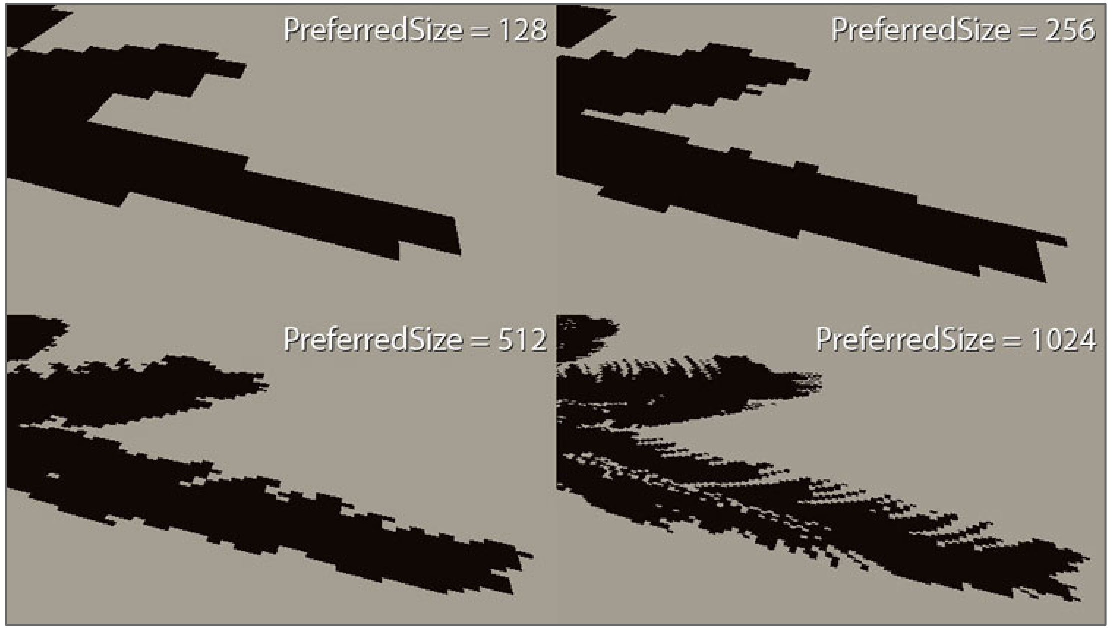
Shadow aliasing

- Finite shadow map resolution: pixelized shadows.
- Large scenes require high shadow map resolution, and a tight projection.



Shadow aliasing

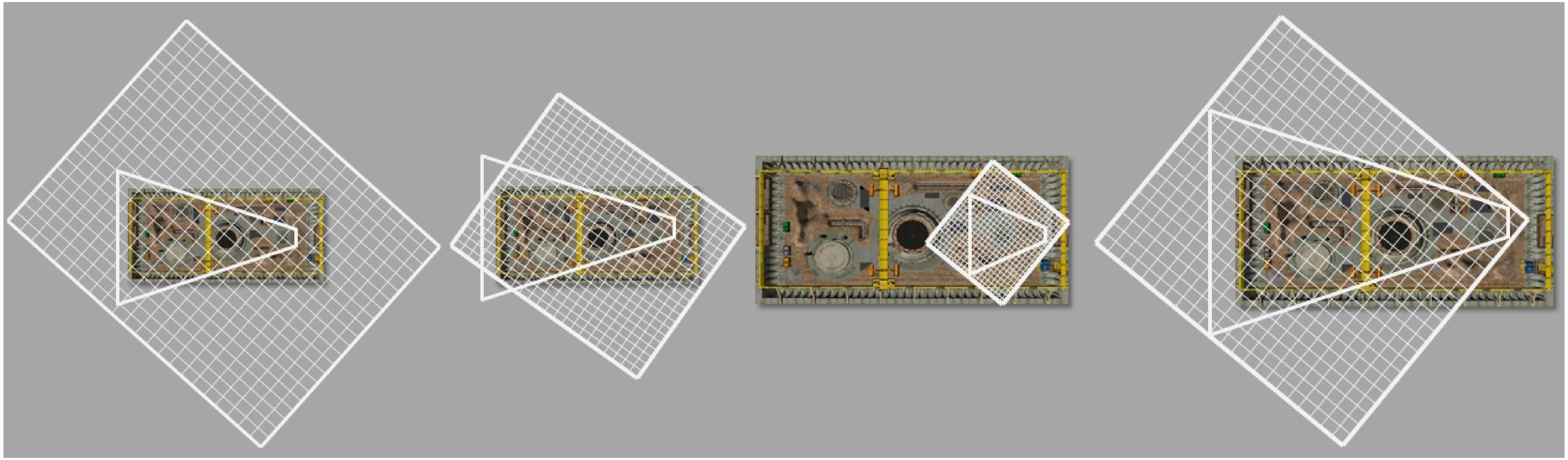
- Finite shadow map resolution: pixelized shadows.
- Large scenes require high shadow map resolution, and a tight projection.



DigitalRune

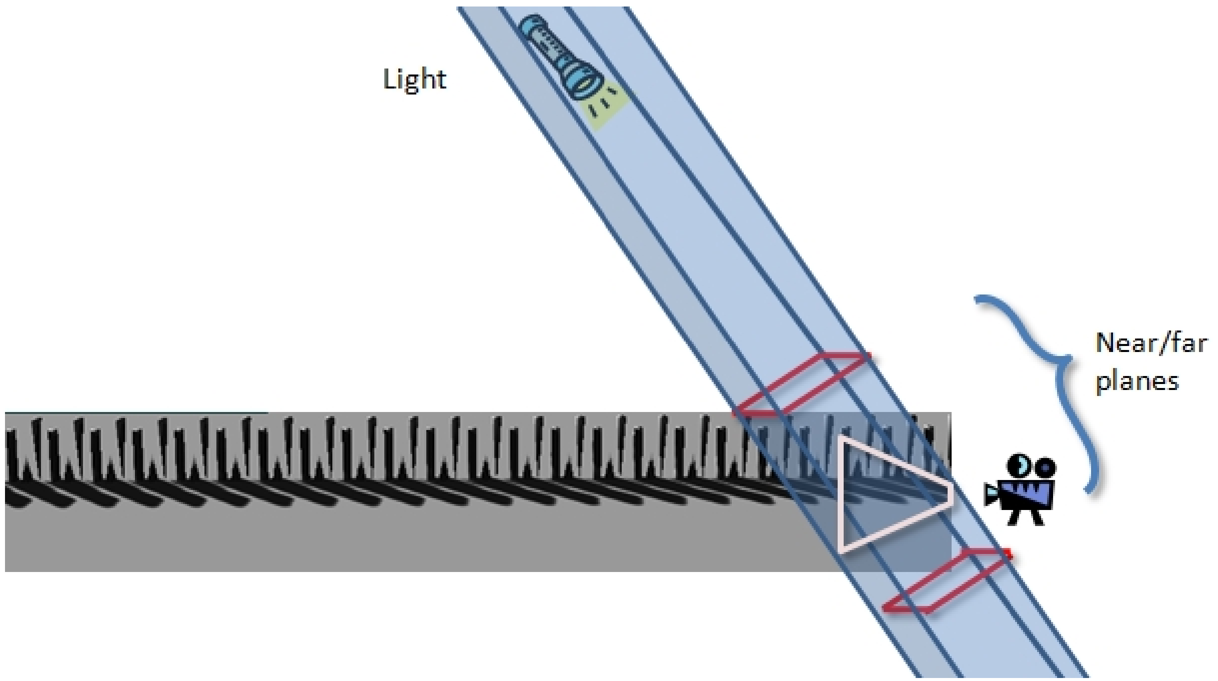
Shadow aliasing

- Calculate a tight projection. How?



Computing optimal projection

- Calculate 8 corners of view frustum in light space.
- 6 planes: 4 sides, near, far.
- Clip scene's bounds against 4 side planes.
- Smallest and largest z-values of clipped boundaries represent the near and far plan, respectively.



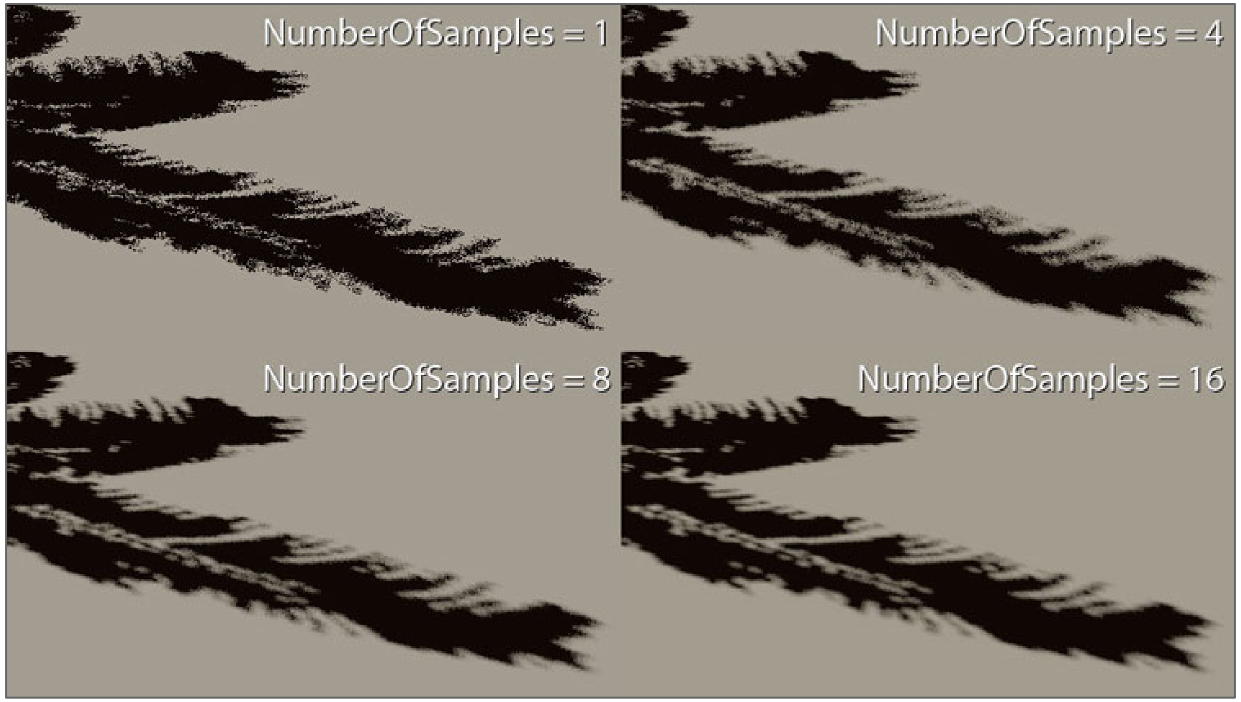
Shadow aliasing

- Solution: sample from more than once from shadow map

```
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
for(int x = -1; x <= 1; ++x)
{
    for(int y = -1; y <= 1; ++y)
    {
        float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y)
* texelSize).r;
        shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
    }
}
shadow /= 9.0;
```

Shadow aliasing

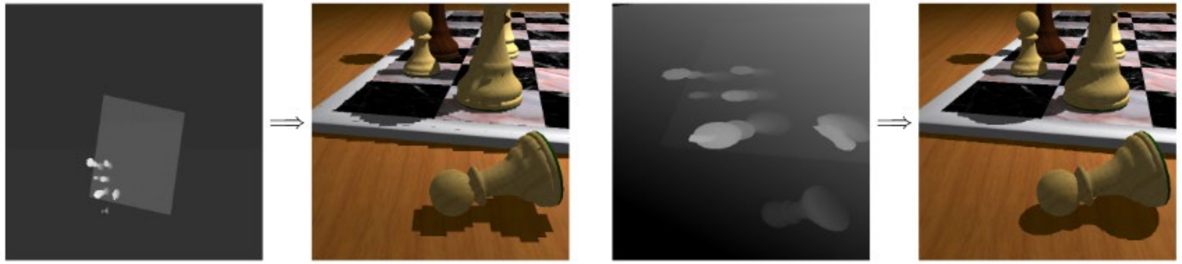
- Solution: sample from more than once from shadow map.



DigitalRune

Shadow maps

Improving Shadow maps



Perspective Shadow Maps
[Stamminger and Drettakis, 2002]

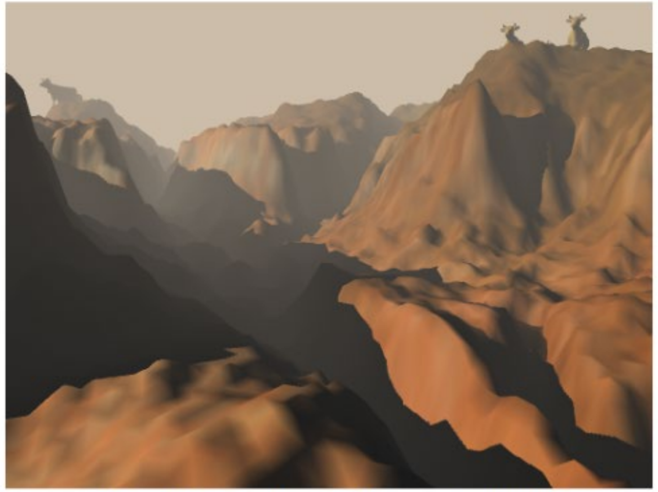


Figure 3-1. Large scale terrain rendering with 4-splits CSM

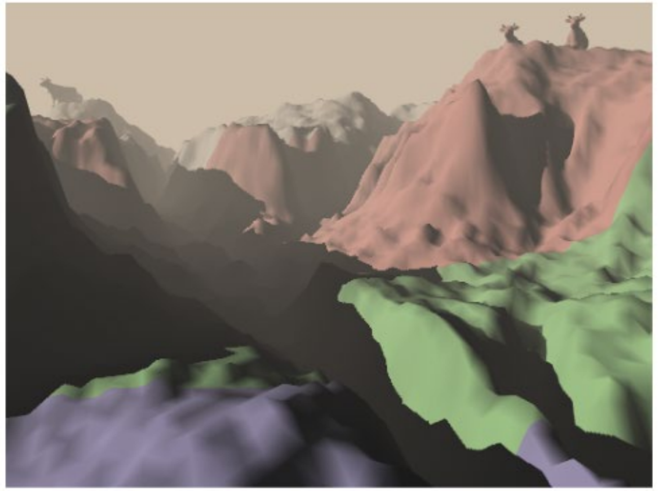


Figure 3-2. Texture look ups from different shadow maps are highlighted

Cascaded Shadow Maps [Dimitrov, 2007]

Comparison

- Shadow Volumes:
  - Pros: accurate hard shadows
  - Slower, rasterization heavy
- Shadow Maps:
  - Pros: Fast, supports soft shadows
  - Cons: high memory usage, aliasing