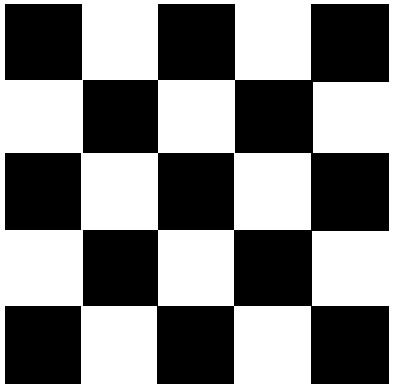
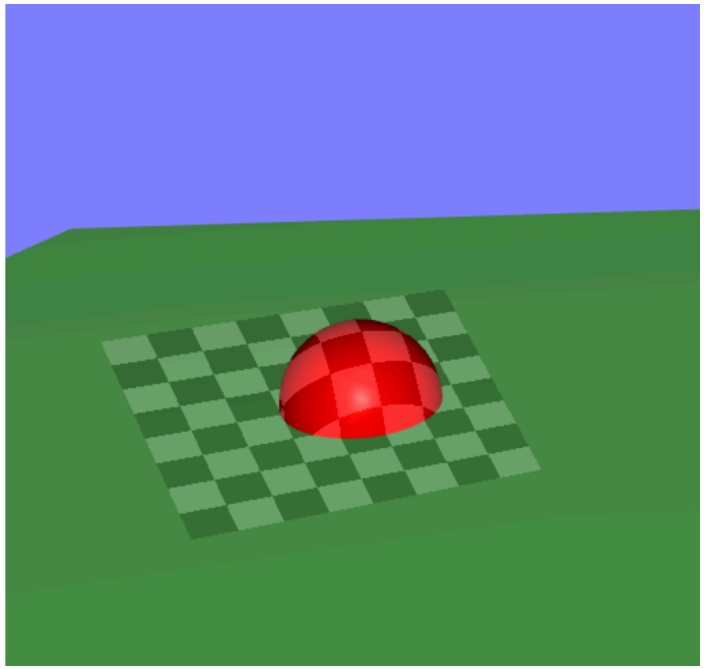
# Projective textures

*A nice example of a complex light source*

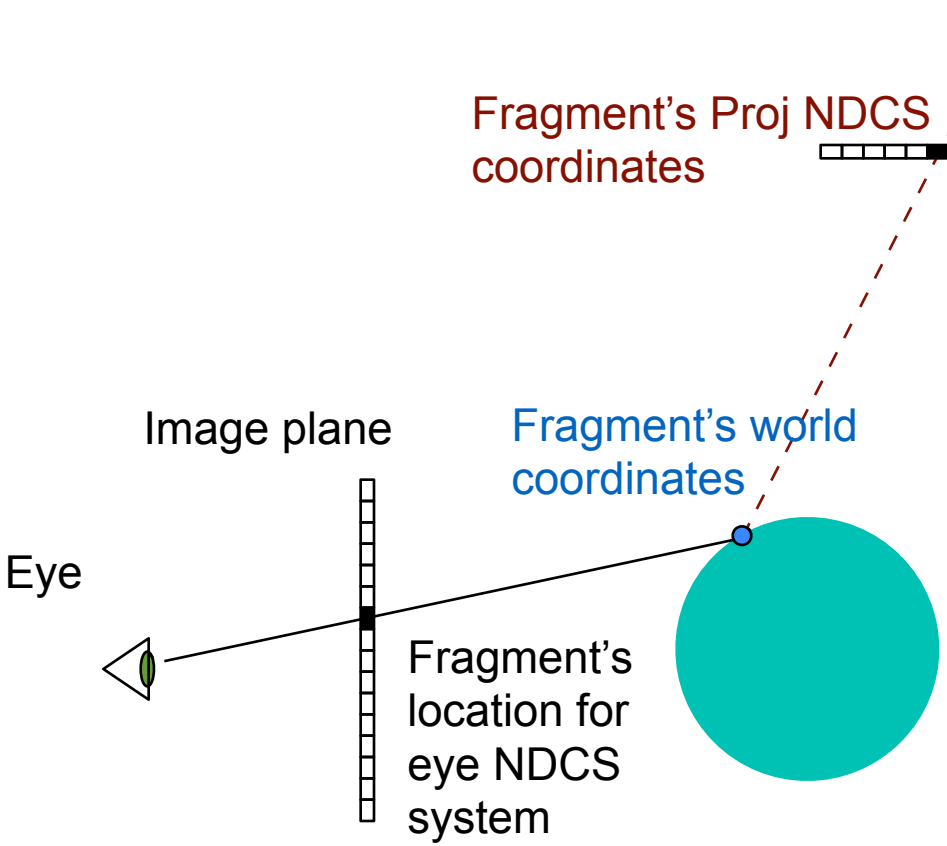




# Projective Textures

- Define a camera system
- Map a texture to its image plane
- Project the image plane onto the objects
- In some sense we are following the light's trajectory in the opposite direction
- So a pixel on this camera's image plane is now a texel
- Note: we will use the NDCS coordinates of this camera as textures, so they have to be rescaled to [0,1] through a bias matrix

# Visually the two pipelines



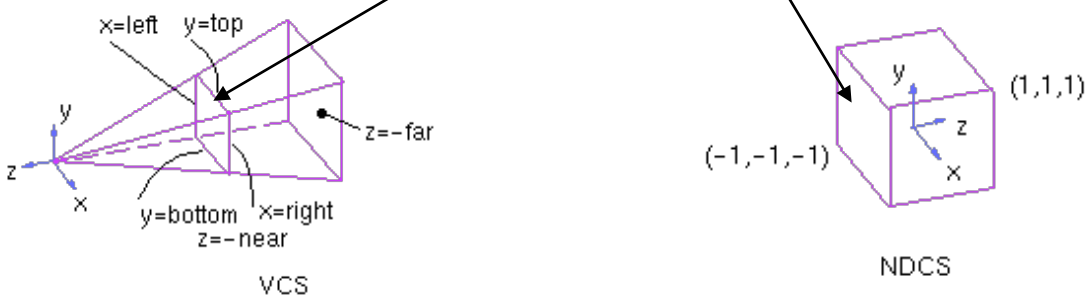

NDCS is [-1,1] so bias to [0,1] texture coordinates

When rendering the scene:
From the word coordinates of the point (fragment) compute the corresponding NDCS wrt projector and bias to texture coordinates.

# The projector

Top, right pixel (texel)
View CS: (right,top,-near)
NDCS: (1,1,1)
Texture CS: (1,1)

Bottom left pixel (texel)
View CS: (left,bottom.-near)
NDCS: (-1,-1,-1)
Texture CS: (0,0)



# Projective textures

***Normal texture mapping:***

- Texture coordinates assigned by designer

***Projective texture***

- Texture considered mapped on the near plane covering the visible viewport (in NDCS [-1,1]) of the projector
- Texture coordinates computed for each vertex(fragment)  as the NDCS coordinates of the vertex (fragment) of the projector's pipeline, appropriately scaled to be in [0,1]

# Projective Textures

- Define a camera system for the projector: View Matrix V, Projection Matrix P (transforms the volume into NDCS)
- Add a bias matrix B, why?

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

- Combined matrix  M = BPV
- What does this matrix do?

# Projective Textures

- Define a camera system: Modelview Matrix V, Projection Matrix P (transforms the volume into NDCS)
- Add a bias matrix B, why?

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

- Combined projector matrix  M = BPV
- Pproj = M Pworld  Takes world coordinates into projector's Clipping Coordinates

# Example Code

***In Javascript host program:***

- Load texture to unit 0 and bind it
- Set the filters

# Example code

## *In Javascript host program*

- Set the projector matrix and assign it to a uniform

```
var projPos = vec3(2.0,5.0,5.0);
var projAt = vec3(0.0,0.0,1.0);
var projUp = vec3(0.0,1.0,0.0);
var projView = lookAt(projPos, projAt, projUp);
var projProj = perspective(30.0,1.0, 0.2, 1000.0);
var projScaleTrans = mult(translate([0.5,0.5,0.5]),
scale(0.5,0.5,0.5)) ;
var ProjectorMatrix = mult(projScaleTrans, mult(projProj, projView));

var projectorMatrixLoc = gl.getUniformLocation(program,
"ProjectorMatrix") ;

gl.uniformMatrix4fv(projectorMatrixLoc, false,
flatten(ProjectorMatrix) );
// Is the projector fixed in world coordinates or not?? If not what
do you need to do to fix it?
```

# Vertex Shader

```
#version 300 es

in vec4 VertexPosition;
in vec3 VertexNormal;

out vec4 ProjTexCoord;
out vec4 fColor;

uniform mat4 modelViewMatrix;
uniform mat4 modelMatrix;
uniform mat4 normalMatrix;
uniform mat4 projectionMatrix;


uniform mat4 ProjectorMatrix;
```

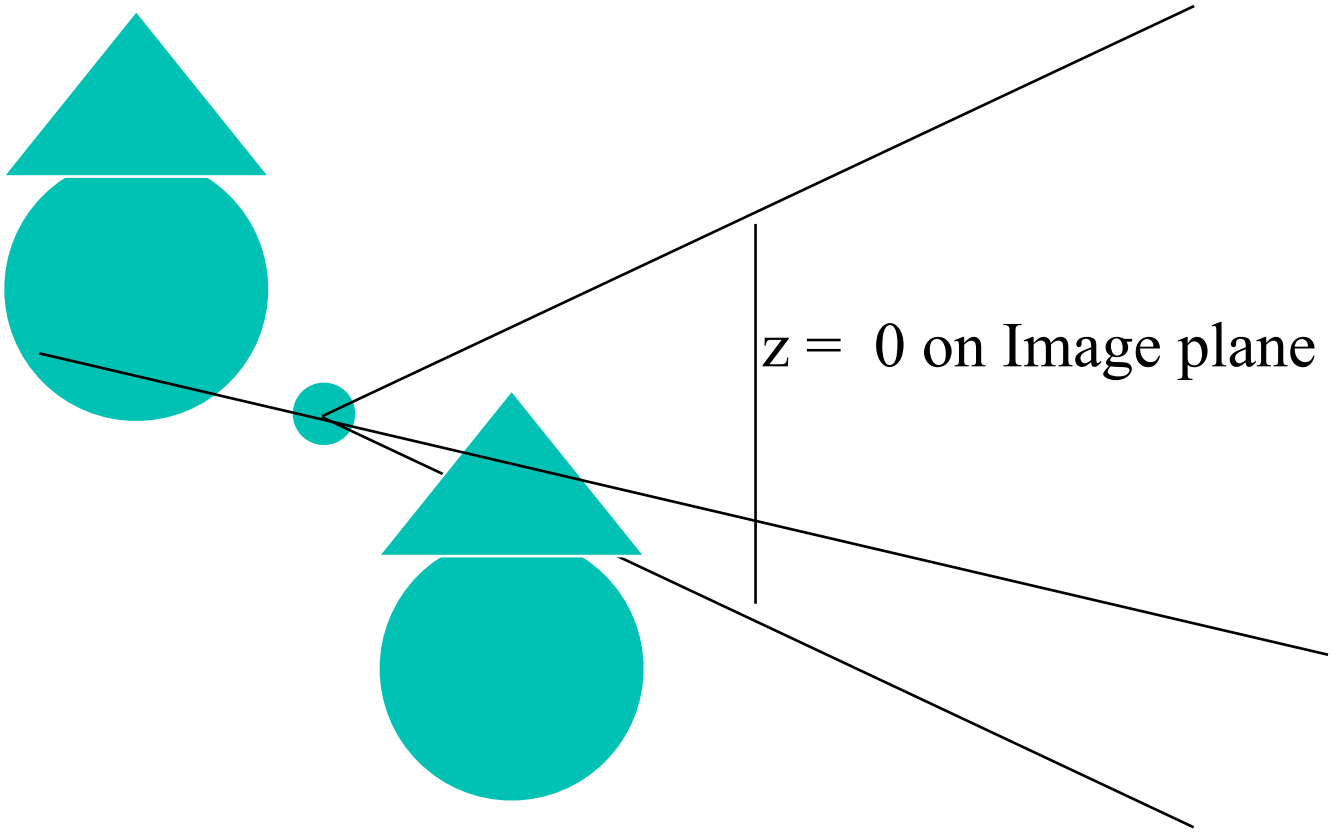
# Vertex Shader

```
…

out vec4 ProjTexCoord;

void main()
{
  gl_Position = projectionMatrix * modelViewMatrix *
  vPosition;

  fColor = ads(…);
  fColor.a = 1.0;
  ProjTexCoord = ProjectorMatrix * (modelMatrix *
                 vPosition);
}
```

# Fragment Shader

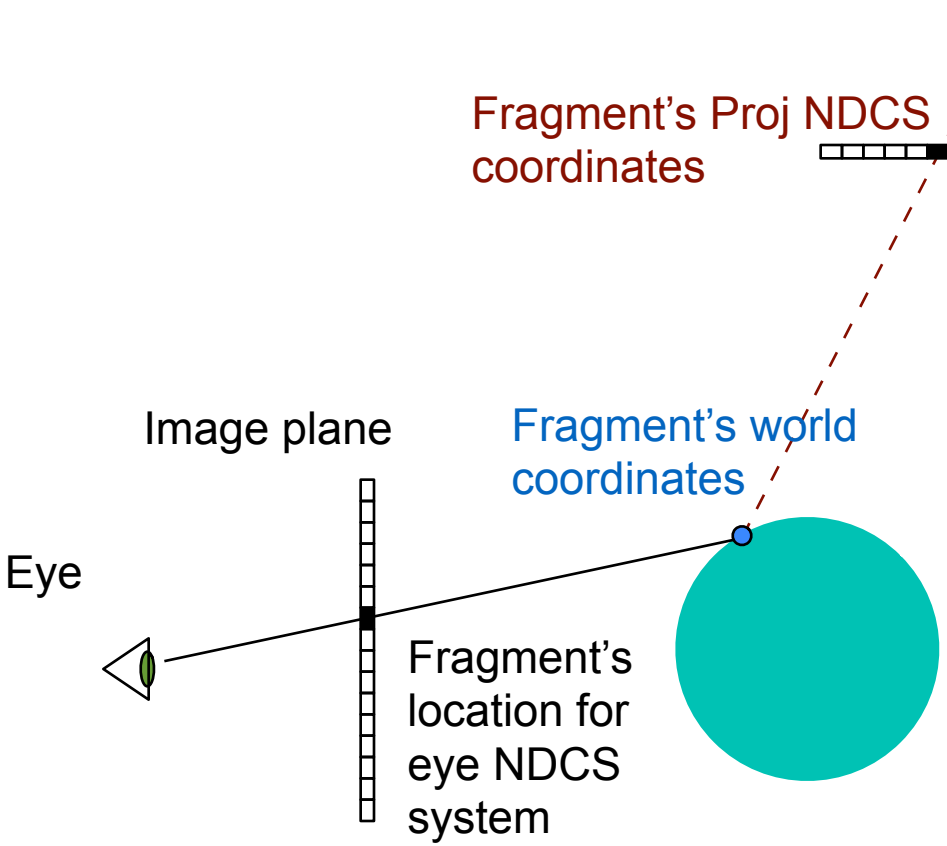
```
…
in vec4 fColor ;
in vec4 ProjTexCoord;

uniform sampler2D texture1;
layout( location = 0 ) out vec4 FragColor;

void main() {
vec4 c1,c2,c3 ;

    fragColor = fColor;
    if( ProjTexCoord.z > 0.0) {
       float x = ProjTexCoord.x / ProjTexCoord.w;
       float y = ProjTexCoord.y / ProjTexCoord.w;
       if( ( x < 0.0) ||  (x > 1.0) || (y < 0.0) || (y > 1.0) )
           c1 = vec4(0.f,0.f,0.f,1.0f ) ;
       else
           c1 = texture( texture1, vec2(x,y));
    }
    fragColor = fColor + c1*0.7;
    fragColor.a = 1.0 ;
}
```

# Cases where z < 0




We are essentially doing clipping in terms of z

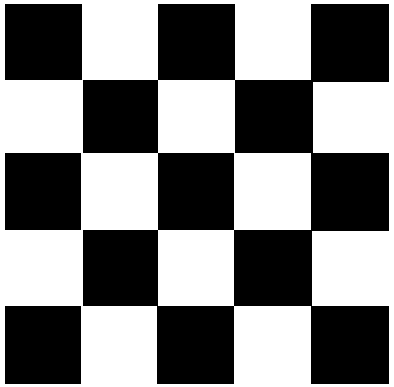
# Visually the two pipelines

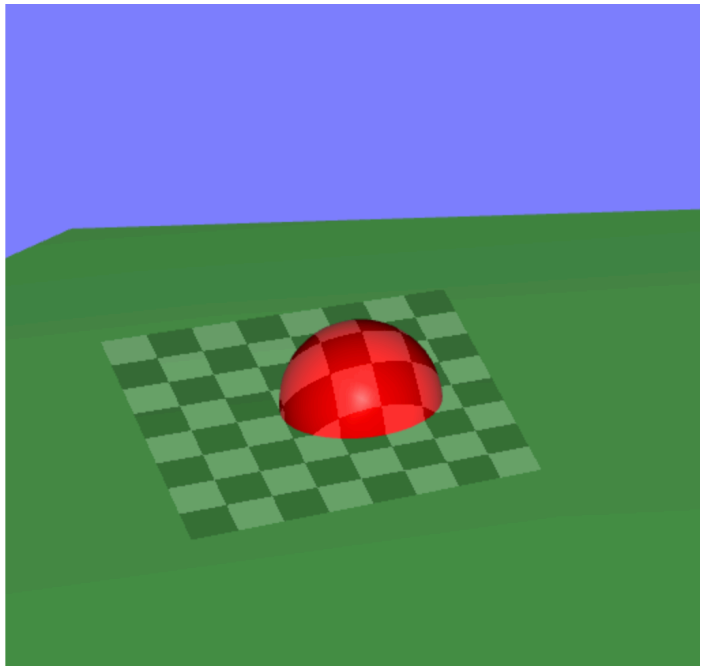



NDCS is [-1,1] so bias to [0,1] texture coordinates

When rendering the scene:
From the word coordinates of the point (fragment) compute the corresponding NDCS wrt projector and bias to texture coordinates.

# Result





# Issues

***No occlusions***

- Texture will project on all objects in its path