Abertay University

Shadow maps (and depth)

CMP301 Graphics Programming with Shaders

This week

- Depth
- Shadowing techniques
- Shadow mapping overview
- Shadow mapping implementation

Depth buffer

- A little bit of a reminder
- Stores the distance each fragment is from the camera
  - Used in depth test
  - Make sure objects are rendered in the correct order
  - Is a texture
- Unlike last year we will be interacting with the depth data

Depth buffer

- Depth value
  - Between 0.0f and 1.0f
  - Represents the position of a fragment between the near and far plane
  - Not linear
    - Limited size and thus limited precision
    - More precision on fragments closer to the camera
    - Less precision further away
    - Possibility of z-fighting in the distance

Depth buffer

- 90/10 rule
  - 90% of the floating point value occurs in the first 10% of the depth buffer
    - At the near plane
  - The remaining 10% take up the remaining 90% of the buffer
    - 0.9f to 1.0f

Depth buffer

- Purpose
  - Knowing the depth we can control other aspects
  - For example
    - Do bump mapping on close objects and just diffuse lighting on far objects
    - Variable post processing, more blurring at a great distance
    - Shadows

Alternatives

- We could interact directly with the depth buffer linked to the pipeline
- Or we can calculate and store depth values ourselves
  - Depends what you are doing
  - For many of the techniques mentioned we want to store it ourselves
    - Means multi pass rendering

Process

- Render scene to render-to-texture storing depth values as colour
  - No textures or lighting
- Render scene again, pass the depth texture for use within shaders
  - Required for shadow mapping
  - And many other techniques

Getting depth

- When we transform vertices by world-view-projection matrices
  - We calculate the depth of the geometry and the fragment
  - Depth buffer stores the depth of each pixel as long as it is closer than the previously stored values
- We will capture and store that data as colour

Example

- Render a simple plane
- Instead of outputting texture and lighting
  - Output a colour based on range of depth value
- Need to update the near & far planes
  - Depth is proportional to the near far distance
  - New near
    - 1.0f
  - New far
    - 100.0f

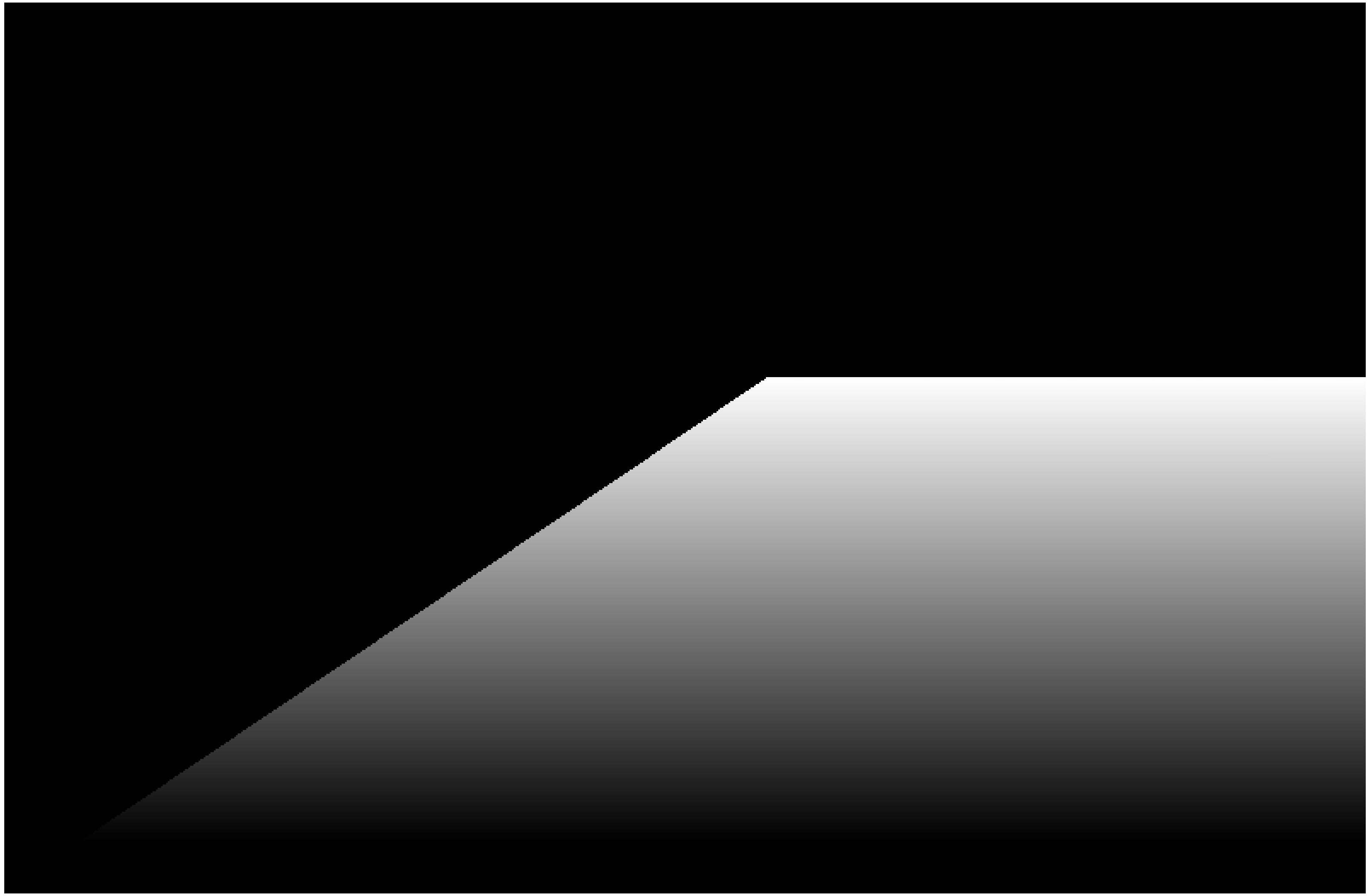
Depth_vs

```
OutputType main(InputType input)
{
    OutputType output;

    // Change the position vector to be 4 units for proper matrix calculations.
    input.position.w = 1.0f;

    // Calculate the position of the vertex against the world, view, and projection matrices.
    output.position = mul(input.position, worldMatrix);
    output.position = mul(output.position, viewMatrix);
    output.position = mul(output.position, projectionMatrix);

    // Store the position value in a second input value for depth value calculations.
    output.depthPosition = output.position;

    return output;
}
```

Depth_ps

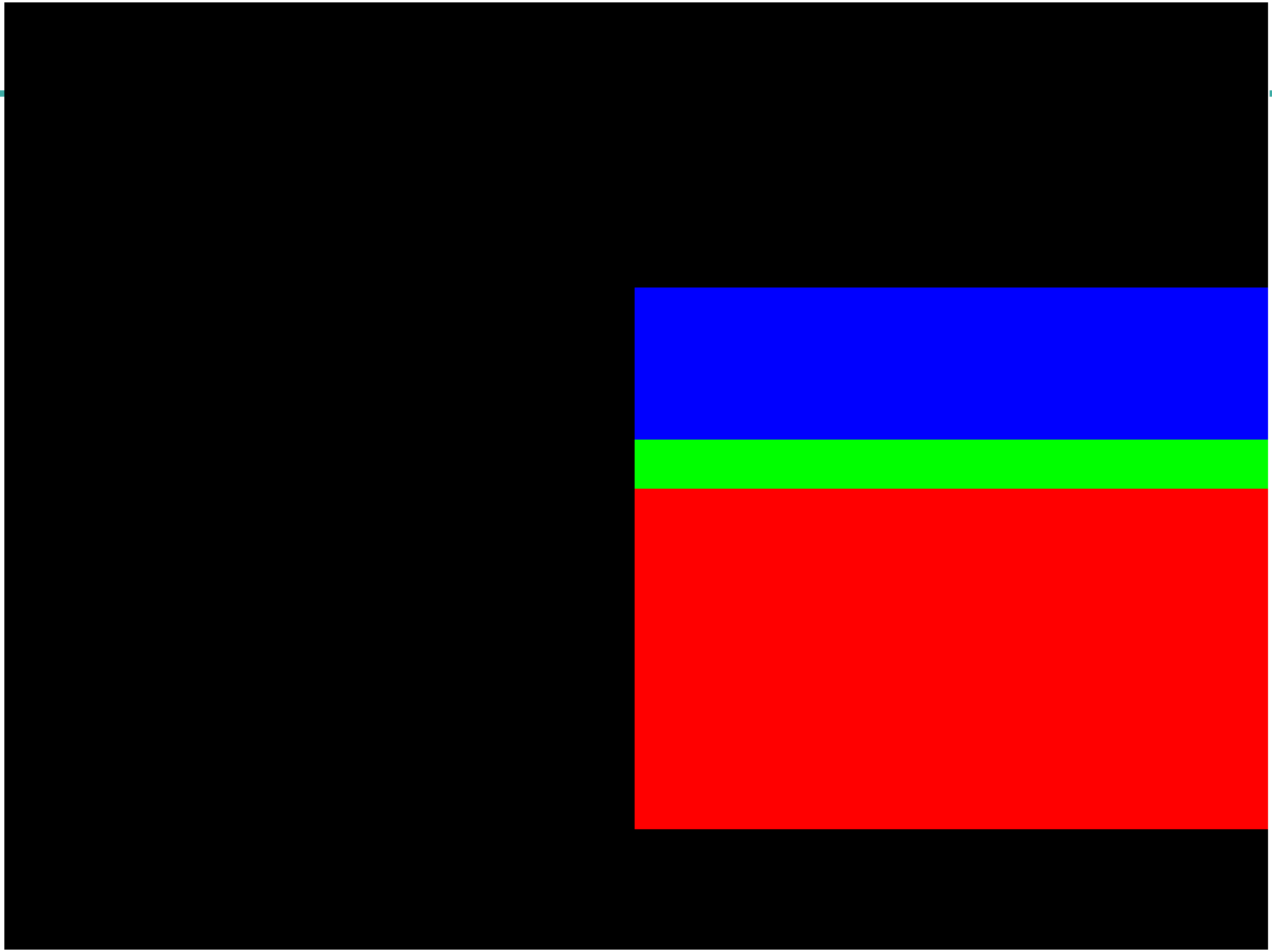
```
struct InputType
{
    float4 position : SV_POSITION;
    float4 depthPosition : TEXCOORD0;
};

float4 main(InputType input) : SV_TARGET
{
    float depthValue;
    float4 colour;

    // Get the depth value of the pixel by dividing the Z pixel depth by the
    homogeneous W coordinate.
    depthValue = input.depthPosition.z / input.depthPosition.w;

    colour = float4(depthValue, depthValue, depthValue, 1.0f);

    return colour;
}
```

Results



Depth

- To demonstrate the depth buffer is non-linear
- Based on value out a different colour
  - If < 0.9
    - Red
  - If > 0.9
    - Green
  - If > 0.925
    - Blue



Importance of shadows

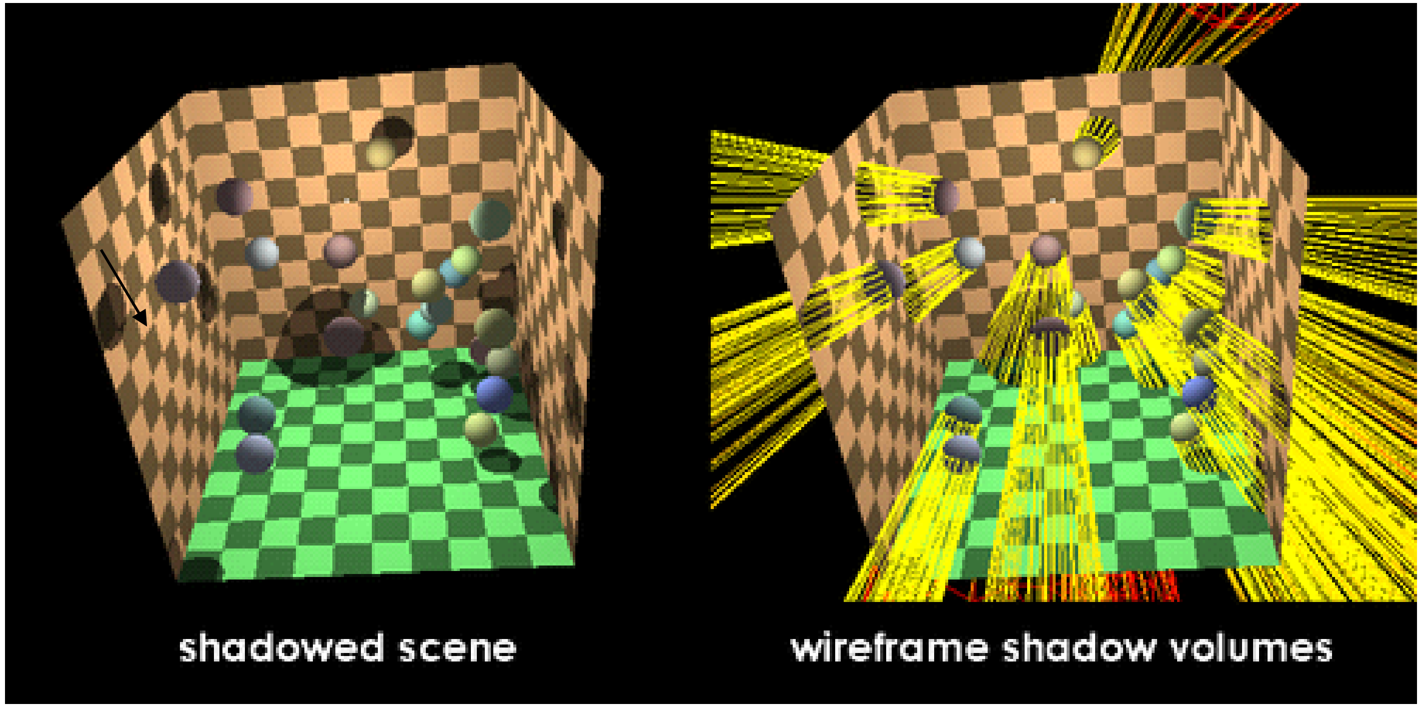
- Why
  - We need shadows to convey the location of objects in the scene
- Many methods
  - Imposters
  - Volumes
  - Maps
  - GI

Shadows imposters

- Just draw a blob on the ground
- Looks OK
- We can do a lot better

Shadow Volumes

- Casting a ray from the light through each vertex of the model
- Really Accurate
- Can support point light shadows by default
- They reveal rough geometry
- Not as fast as shadow mapping



Shadow Mapping

- Most games use it or a variant
- Directional light shadows
- Easiest to make

Shadow Mapping

- Multiple lights?
- More difficult

Shadow Mapping

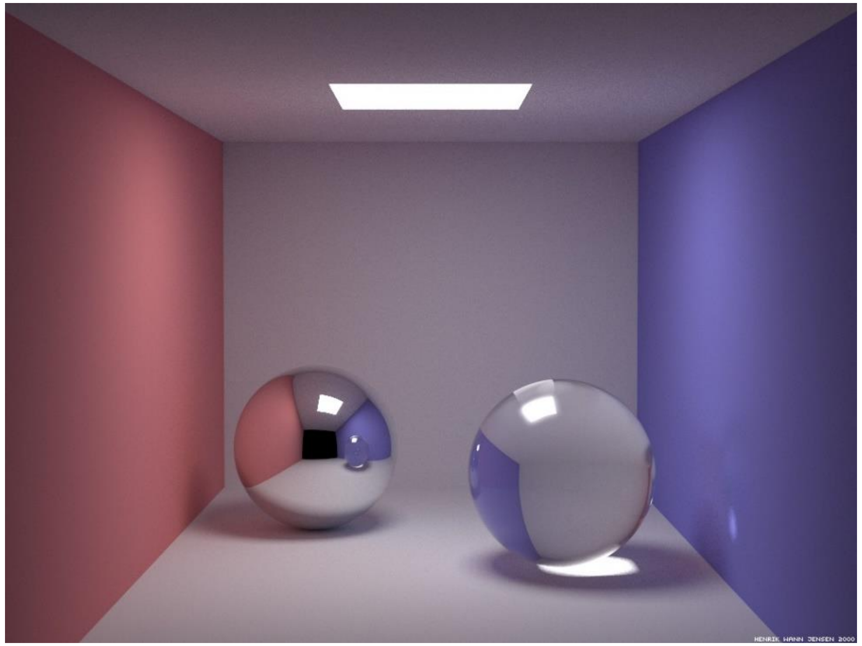
- Point Lights – An extension on spot lights

Shadow Mapping

- Hard Shadows
- Soft Shadows
  - Need to be able to make hard shadows first


Global Illumination

- Abandons the current lighting model
- Light amount is calculated by casting rays from the light source and bouncing them off objects
- Completely outside the scope of this module
- https://www.youtube.com/watch?v=n0vHdMmp2_c



Shadow Mapping - Overview

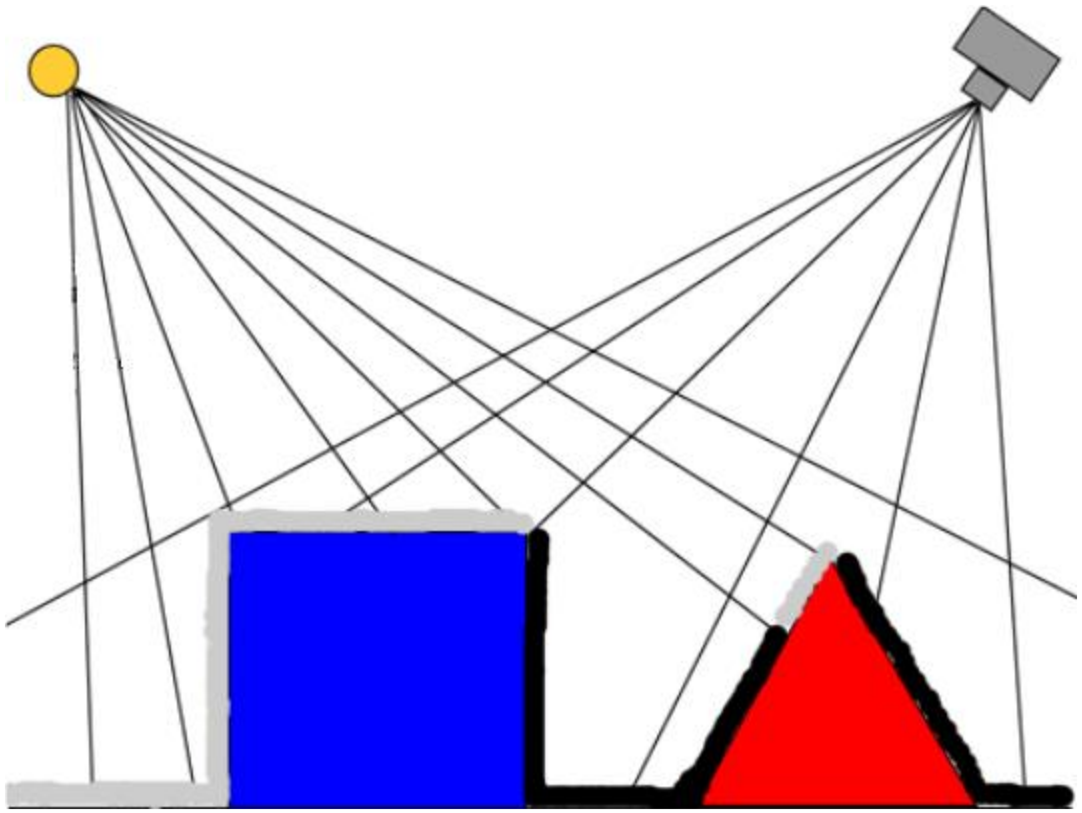
- How
- Challenges
- Solutions

Shadow Mapping - Overview

- To Create a shadow map
- Render the scene from the lights position
- With a separate view and projection matrix for the light
  - This acts as the camera for the light
- Saving the depth into the render target
- Shadow map now holds the depth of the points closest to the light
- Anything in the shadow map cannot be in shadow

Shadow Mapping - Overview

- To use the shadow map
  - Render the scene as normal
  - When performing lighting calculations add a shadow calculation
  - We can compare the depth in the shadow map with a depth we calculate in the shader
  - If the depth is less than the shadow map value then light the pixel



Shadow Mapping - Overview

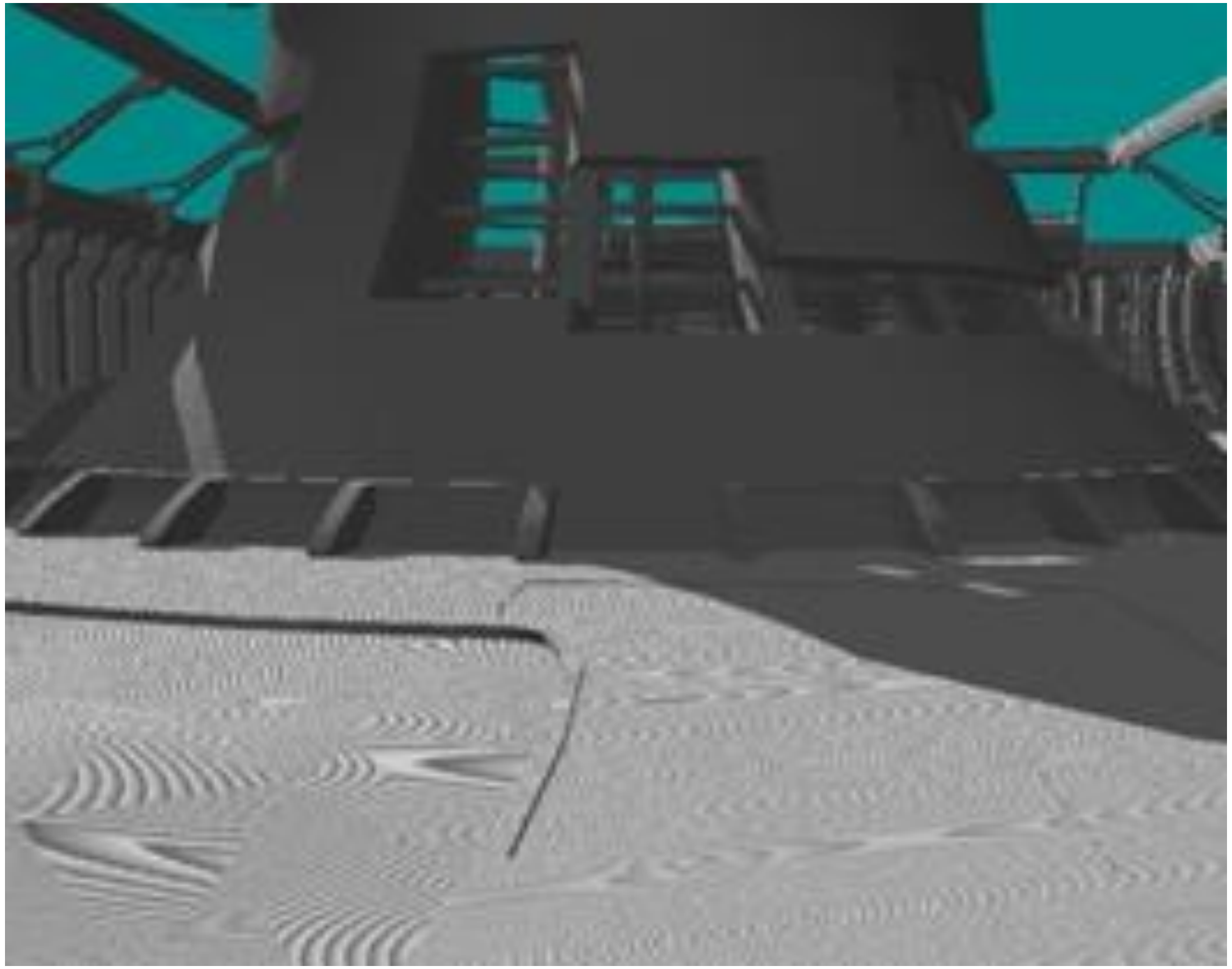
- Requires multi pass rendering
  - First pass get depth
  - Second pass render scene as normal
- Depth passes are significantly faster than colour + depth passes
- GFX cards are highly optimised for depth calculations

Challenges

- Two major challenges with shadow mapping
  - Artefacts
  - Shadow separation

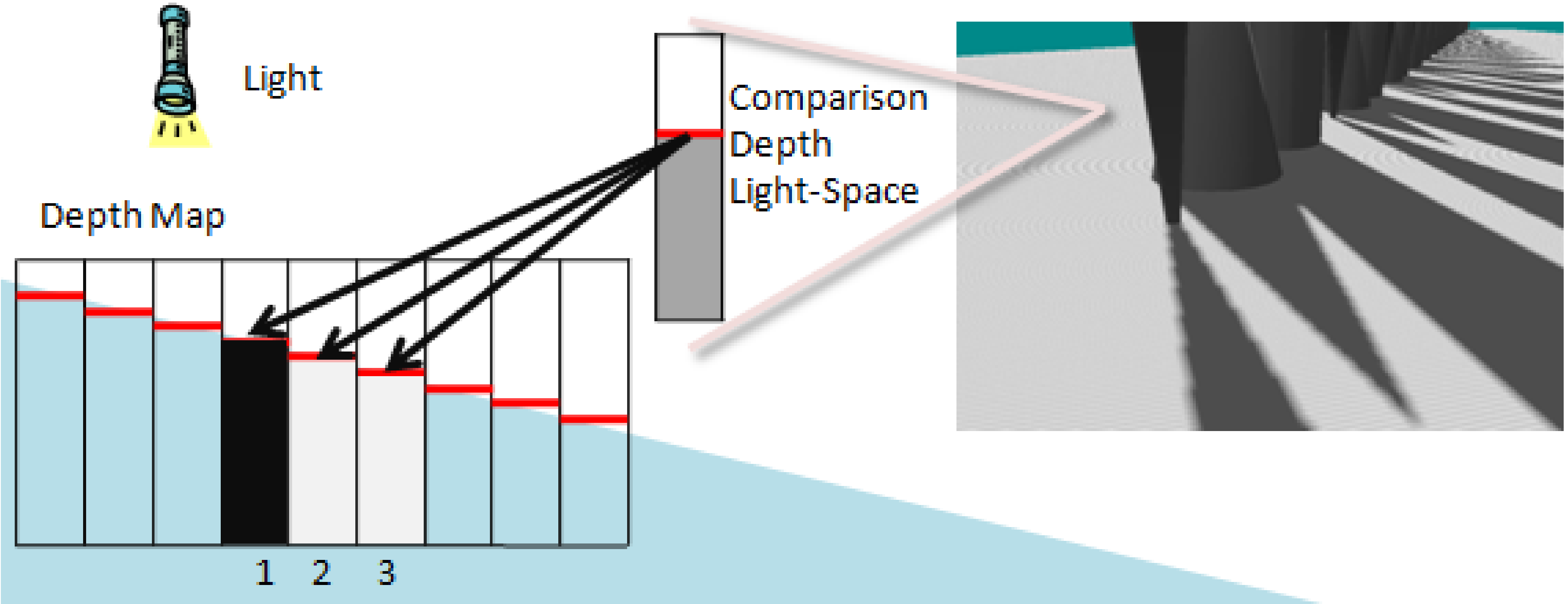
Artefacts

- Shadow Acne
- http://msdn.microsoft.com/en-us/library/windows/desktop/ee416324(v=vs.85).aspx



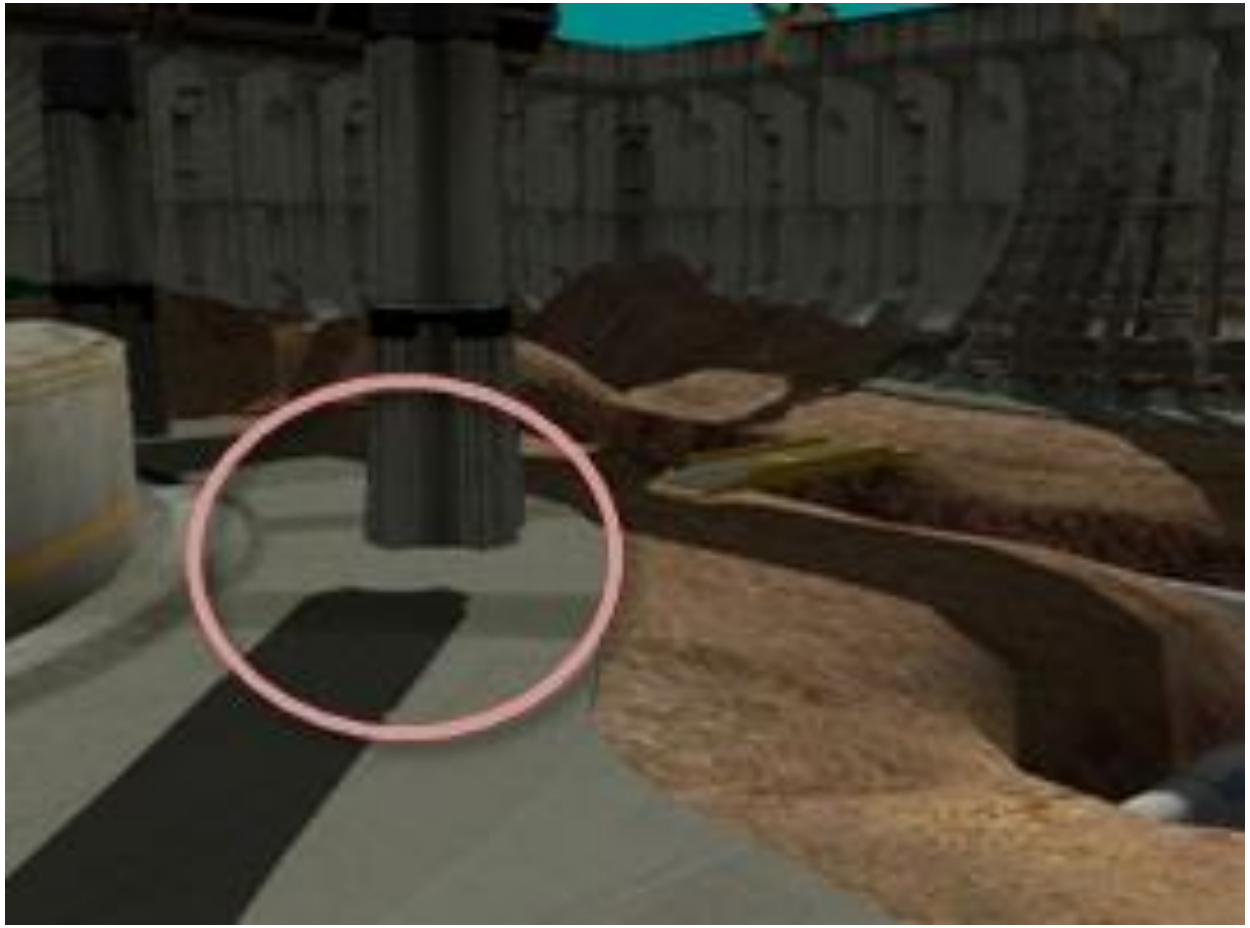
Depth Bias

- Too little will give you shadow acne



Bias?

- Too much will give you shadow separation
  - Peter Panning



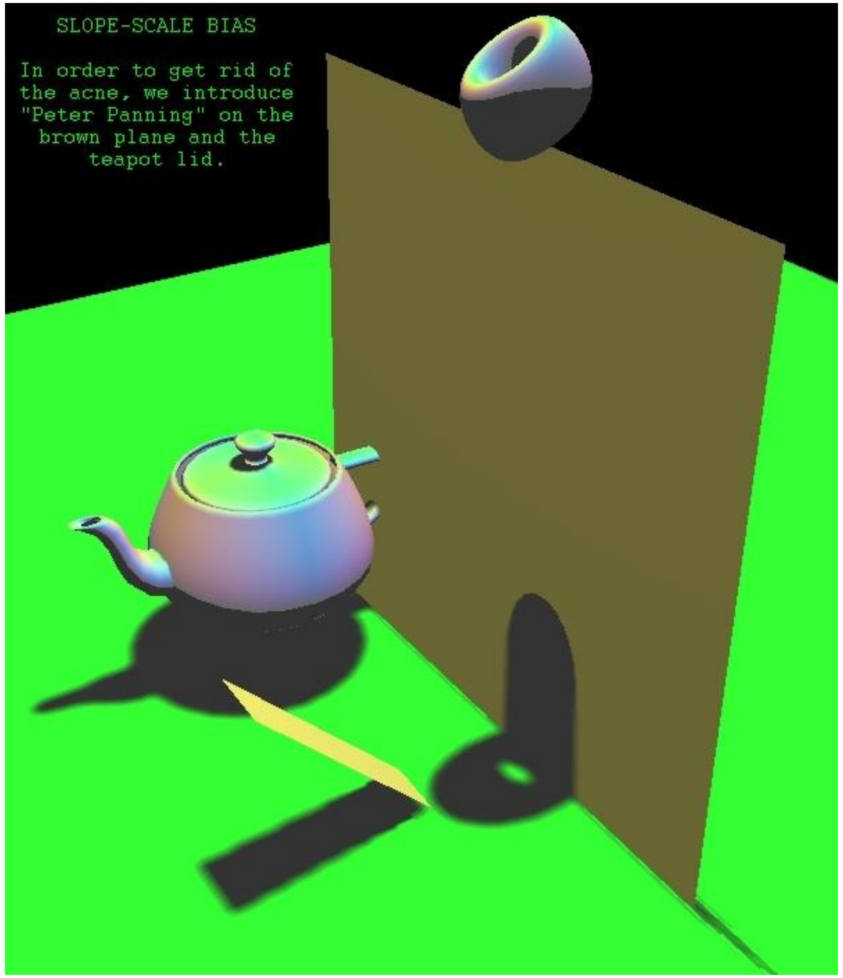
Constant Bias

- Were just going to use a constant bias
- But you can use variations of this
- Too high will give you peter panning
- Too low will give you more acne

- Alternatives
  - Slope scale bias
  - Normal offset bias

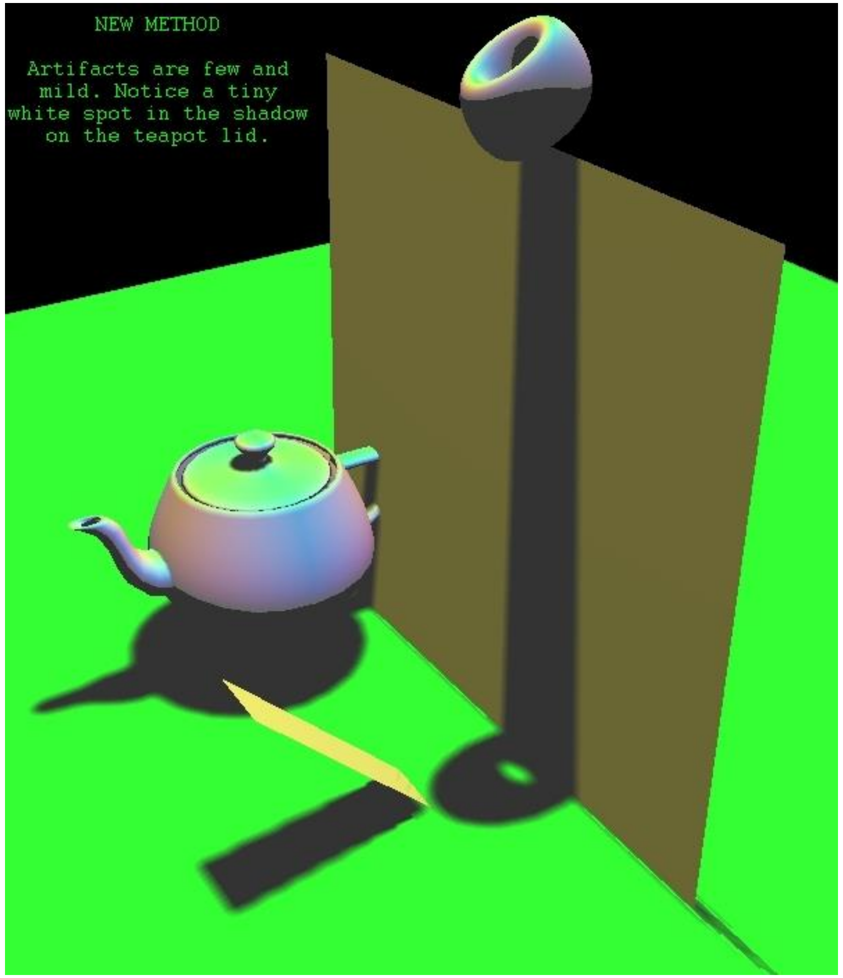
Slope scale bias

- Adjust bias based on gradient
- Not great for really steep surfaces



Normal Offset bias

- Push the point out along its normal
- Can have issues with normal mapping
- http://www.dissidentlogic.com/old/#Normal Offset Shadows



Example

- We will render a couple of objects
  - Floor
  - Teapot
- Using the depth shaders mentioned earlier
- Adding new shadow shaders

Render pseudo-code

- Render scene to texture
  - All objects
  - Using depth shaders
  - From the light view point
- Render scene (to back buffer)
  - All objects
  - Using shadow shaders
    - Still requires light view
    - Calculate objects distance from light and depth from shadow map
    - Apply diffuse lighting to pixels <= to depth value

Render to shadow map()

- Render to texture
  - To store depth data
- Generate a view for the light (can be done once if light is static)
- Get the view and projection matrices
- Render scene using the depth shader
  - Using the lights view and projection matrices
  - Instead of camera view or projection matrices
- Set the render targets back again
- Reset viewport

Generate a view matrix for the light

- The light class can generate its view and projection matrices
- Once the light is positioned or every frame

```
void generateViewMatrix();
void generateProjectionMatrix(float near, float far);
```

Shadow map

- Render the scene from the lights prospective

```
// get the world, view, and projection matrices from the camera and d3d objects.
light->generateViewMatrix();
lightViewMatrix = light->getViewMatrix();
lightProjectionMatrix = light->getProjectionMatrix();
worldMatrix = renderer->getWorldMatrix();

// Render floor
mesh->sendData(renderer->getDeviceContext());
depthShader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, lightViewMatrix, lightProjectionMatrix);
depthShader->render(renderer->getDeviceContext(), mesh->getIndexCount());
```

Shadow shaders

- Major differences for the shadow shaders
  - Increased number of matrices required
  - Two samplers

```
struct MatrixBufferType
{
    XMMATRIX world;
    XMMATRIX view;
    XMMATRIX projection;
    XMMATRIX lightView;
    XMMATRIX lightProjection;
};
```

Render scene()

- Additional matrices added during SetShaderParams()

```
worldMatrix = renderer->getWorldMatrix();
viewMatrix = camera->getViewMatrix();
projectionMatrix = renderer->getProjectionMatrix();


// Render floor
mesh->sendData(renderer->getDeviceContext());
shadowShader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, viewMatrix, projectionMatrix,
    textureMgr->getTexture("brick"), renderTexture->getShaderResourceView(), light);
shadowShader->render(renderer->getDeviceContext(), mesh->getIndexCount());
```

Shadow_vs.hlsl

```
cbuffer MatrixBuffer : register(cb0)
{
    matrix worldMatrix;
    matrix viewMatrix;
    matrix projectionMatrix;
    matrix lightViewMatrix;
    matrix lightProjectionMatrix;
};

cbuffer LightBuffer2 : register(cb1)
{
    float3 lightPosition;
    float padding;
};
struct InputType
{
    float4 position : POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
};
```

Shadow_vs.hlsl

```
struct OutputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
    float4 lightViewPosition : TEXCOORD1;
    float3 lightPos : TEXCOORD2;
};


OutputType main(InputType input)
{
    OutputType output;
    float4 worldPosition;

    // Change the position vector to be 4 units for proper matrix calculations.
    input.position.w = 1.0f;

    // Calculate the position of the vertex against the world, view, and projection matrices.
    output.position = mul(input.position, worldMatrix);
    output.position = mul(output.position, viewMatrix);
    output.position = mul(output.position, projectionMatrix);
```

Shadow_vs.hlsl

```
     // Calculate the position of the vertice as viewed by the light source.
    output.lightViewPosition = mul(input.position, worldMatrix);
    output.lightViewPosition = mul(output.lightViewPosition, lightViewMatrix);
    output.lightViewPosition = mul(output.lightViewPosition, lightProjectionMatrix);

     // Store the texture coordinates for the pixel shader.
    output.tex = input.tex;

    // Calculate the normal vector against the world matrix only.
    output.normal = mul(input.normal, (float3x3)worldMatrix);

    // Normalize the normal vector.
    output.normal = normalize(output.normal);

    // Calculate the position of the vertex in the world.
    worldPosition = mul(input.position, worldMatrix);

    // Determine the light direction based on the position of the light and the position of the vertex in the world.
    output.lightPos = lightPosition.xyz - worldPosition.xyz;

    // Normalize the light direction vector.
    output.lightPos = normalize(output.lightPos);

    return output;
}
```

Shadow_ps.hlsl

```
Texture2D shaderTexture : register(t0);
Texture2D depthMapTexture : register(t1);

SamplerState SampleTypeWrap  : register(s0);
SamplerState SampleTypeClamp : register(s1);

cbuffer LightBuffer : register(cb0)
{
    float4 ambientColor;
    float4 diffuseColor;
};

struct InputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
    float4 lightViewPosition : TEXCOORD1;
    float3 lightPos : TEXCOORD2;
};
```

Shadow_ps.hlsl

```
float4 main(InputType input) : SV_TARGET
{
    float bias;
    float4 color;
    float2 projectTexCoord;
    float depthValue;
    float lightDepthValue;
    float lightIntensity;
    float4 textureColor;

     // Set the bias value for fixing the floating point precision issues.
     bias = 0.0001f;

    // Set the default output colour to the ambient light value for all pixels.
    color = ambientColor;

    // Calculate prjected coordinates, then into UV range
    projectTexCoord.xyz = input.lightViewPosition.xyz / input.lightViewPosition.z;

    // Calculate the projected texture coordinates.
    projectTexCoord.x =  (projectTexCoord.x / 2.0f) + 0.5f;
    projectTexCoord.y =  (-projectTexCoord.y / 2.0f) + 0.5f;
```

Shadow_ps.hlsl

```
// Determine if the projected coordinates are in the 0 to 1 range.  If so then this pixel is in the view of the light.
if((saturate(projectTexCoord.x) == projectTexCoord.x) && (saturate(projectTexCoord.y) == projectTexCoord.y))
{
   // Sample the shadow map depth value from the depth texture using the sampler at the      projected texture
coordinate location.
   depthValue = depthMapTexture.Sample(SampleTypeClamp, projectTexCoord).r;

   // Calculate the depth of the light.
   lightDepthValue = input.lightViewPosition.z / input.lightViewPosition.w;

   // Subtract the bias from the lightDepthValue.
   lightDepthValue = lightDepthValue - bias;

   // Compare the depth of the shadow map value and the depth of the light to determine      whether to shadow or to
light this pixel.
   // If the light is in front of the object then light the pixel, if not then shadow this   pixel since an object
(occluder) is casting a shadow on it.
   if(lightDepthValue < depthValue)
   {
```

Shadow_ps.hlsl

```
     // Calculate the amount of light on this pixel.
     lightIntensity = saturate(dot(input.normal, input.lightPos));

    if(lightIntensity > 0.0f)
    {
        // Determine the final diffuse color based on the diffuse color and the amount of light   intensity.
        color += (diffuseColor * lightIntensity);

        // Saturate the final light color.
        color = saturate(color);
    }
 }
}

// Sample the pixel color from the texture using the sampler at this texture coordinate location.
textureColor = shaderTexture.Sample(SampleTypeWrap, input.tex);

// Combine the light and texture color.
color = color * textureColor;

    return color;
}
```

Size of shadow map

- Render to texture
  - Shadow Map Texture

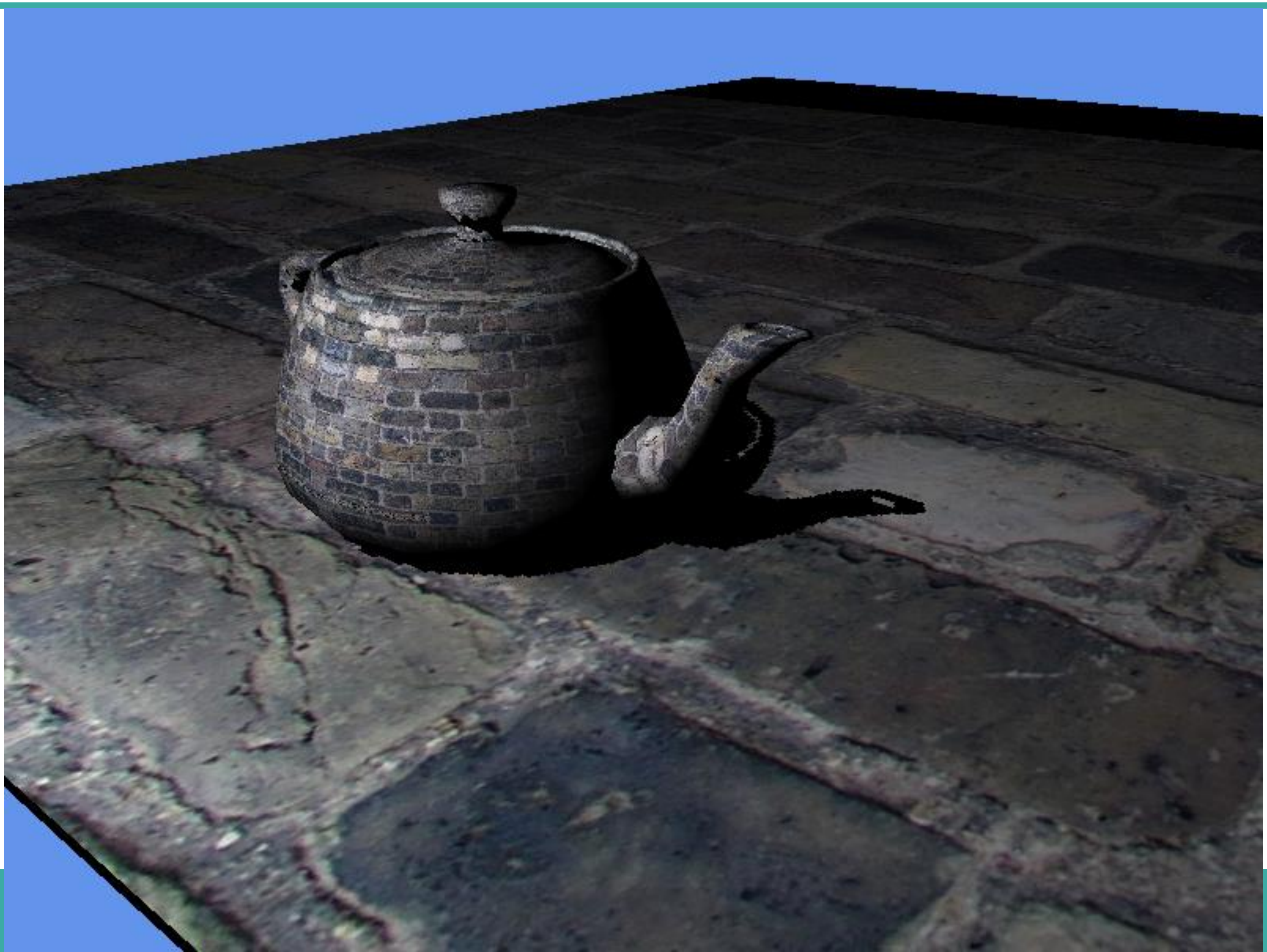
```
•const int SHADOWMAP_WIDTH = 1024;
•const int SHADOWMAP_HEIGHT = 1024;
```

  - The size affects the quality of the shadows
  - m_RenderTexture = new RenderTexture(m_Direct3D->GetDevice(), SHADOWMAP_WIDTH, SHADOWMAP_HEIGHT, SCREEN_NEAR, SCREEN_DEPTH);

Orthographic shadow map

- For directional lights
- Use orthographic matrix instead of projection

Directional shadow map

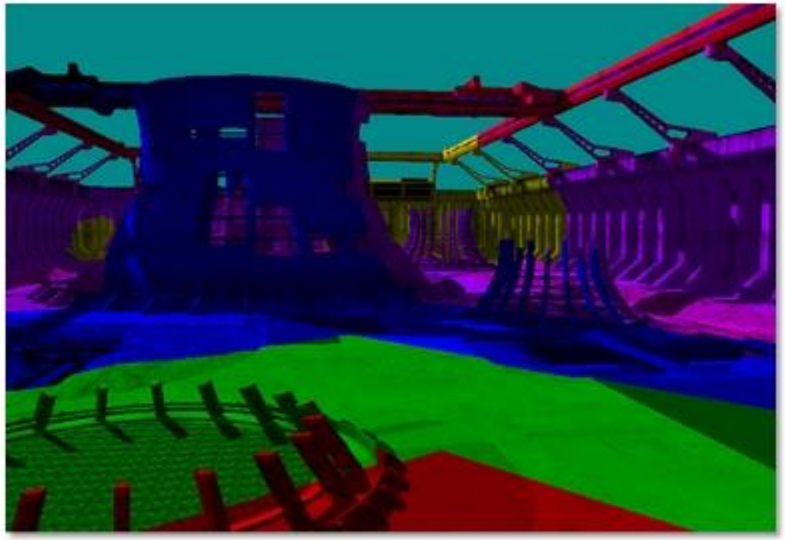


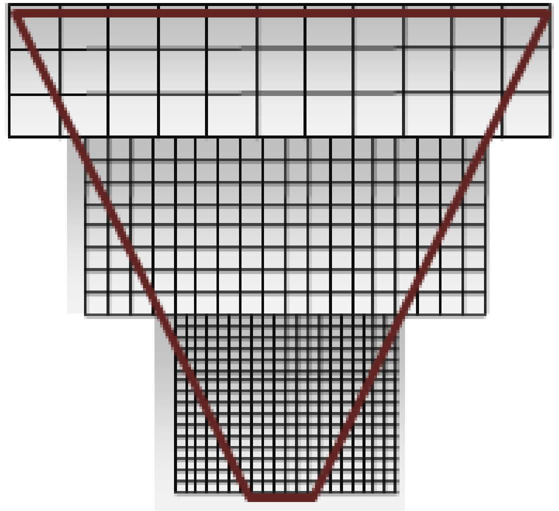
Going Further

- Multiple lights
- Different light types
- Soft shadows
- http://en.wikipedia.org/wiki/Shadow_mapping#Shadow_mapping_techniques

Cascaded Shadow Maps

- Use multiple shadow maps for level of detail
- Each with a different size
- Highest resolution closest to the camera





In the labs

- Building shadows
- Possible revision lecture
  - Any content you would like to cover?