Visual Computing I:

Interactive Computer Graphics and Vision

Stencil Buffers, Render to Texture, Shadow Mapping

Stefanie Zollmann and Tobias Langlotz

Recap

Recap: Advanced Rendering with Buffers

Buffers

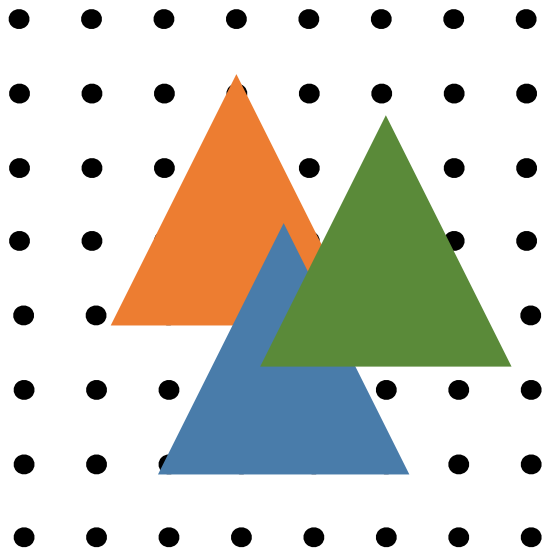
- Buffer is simply a contiguous block of memory
- Different types:
  - Framebuffer: The conceptual container for all screen-related buffers
  - Colour Buffer: Visible output, handled with double buffering for smooth animation
  - Depth Buffer: Essential for correct 3D spatial relationships and occlusion
  - Stencil Buffer: Mask for advanced rendering effects
  - Data Buffers (Vertex Buffer Objects, Index Buffer Objects): Efficiently store and transfer geometric and uniform data to the GPU

Frame Buffer

- Container: Framebuffer is a collection of images (buffers) that hold the data for a single frame
- Goal: To render a complete 2D image from a 3D scene
- Abstract Representation: Framebuffer is not directly manipulated
- Can attach specific "renderbuffers" or textures to it
- Default Framebuffer: Provided by windowing system that gets displayed on the screen

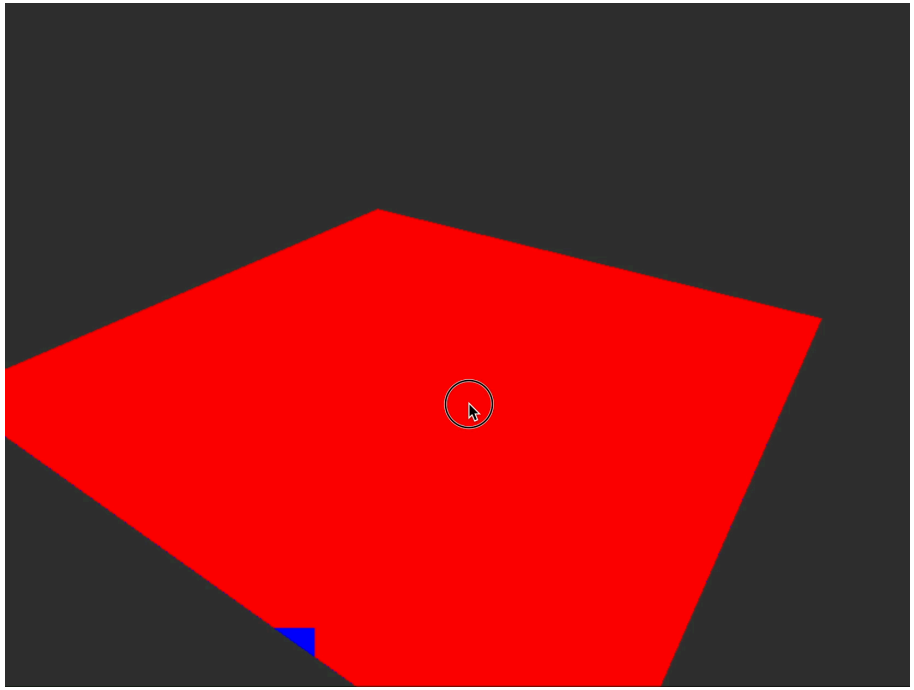
Colour Buffer

- What it is:
  - 2D array storing the final colour value (RGB or RGBA) for each pixel
  - Multiple Colour Attachments: You can have multiple colour buffers for advanced techniques (e.g., deferred shading)
- Double Buffering: Essential for smooth animation:
  - Front Buffer: Currently displayed on screen
  - Back Buffer: Where the next frame is being drawn.
  - Buffer Swap: When drawing is complete, the front and back buffers switch roles



Z-Buffer fighting

- Visual artefact (Flickering) where two or more surfaces with very similar depths "fight" for the same fragment
- Why?
  - Depth buffer does not have infinite precision (grid of floating-point numbers representing depth, e.g. 24-bit)
  - When two fragments are extremely close, their calculated depth values might be (nearly) identical due to this limited precision
  - GPU cannot decide which one is closer, so the "winner" might be whichever fragment happens to get processed last



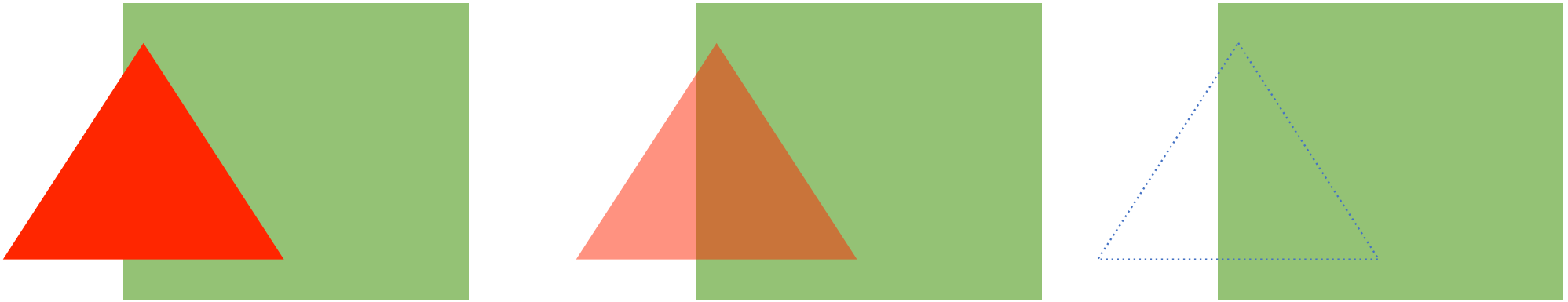
Alpha Testing

- Test discards a fragment if its alpha is below a certain threshold
- Binary: the fragment is either fully drawn or fully invisible
- Use Case: Fences, grass, leaves. (Objects with hard cutouts)
- How: Done manually in the Fragment Shader

```
// If the fragment's alpha is below the threshold...
if (textureColor.a < alphaThreshold)
{
    // ...discard the fragment completely.
    // It will not be written to the color buffer
    // OR the depth buffer. It simply ceases to exist.
    discard;
}
```

Compositing

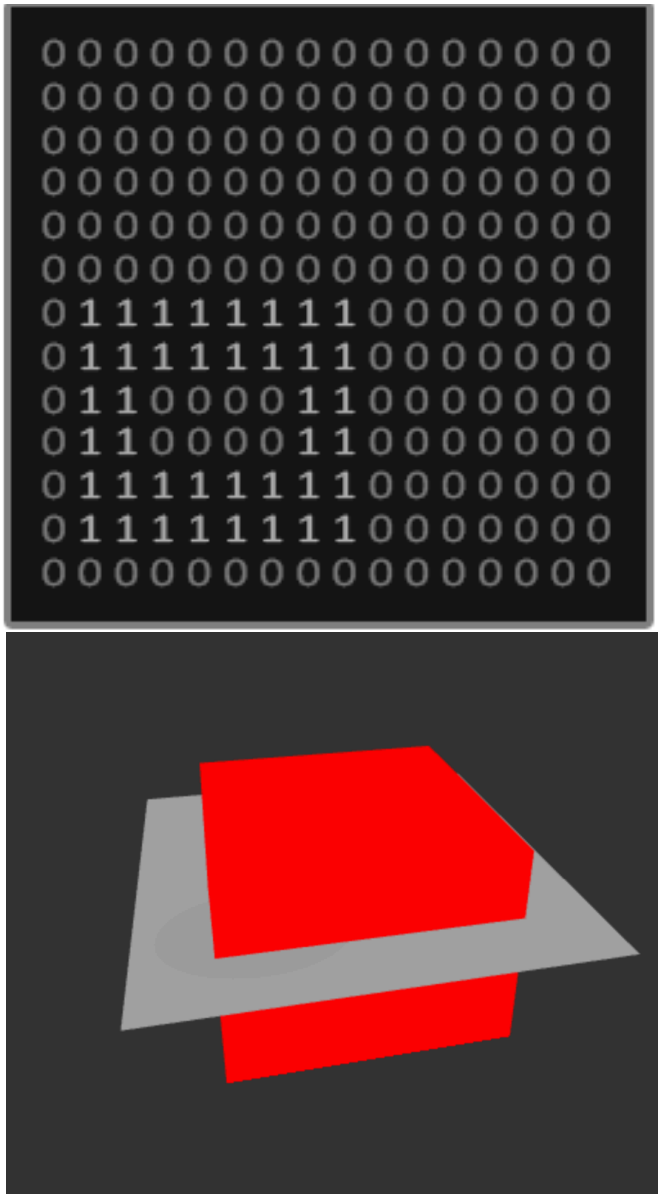
- Alpha describes the opacity of an object
- Fully opaque surface: $\alpha = 1$
- 50% transparent surface: $\alpha = 0.5$
- Fully transparent surface: $\alpha = 0$



Stencil Buffer

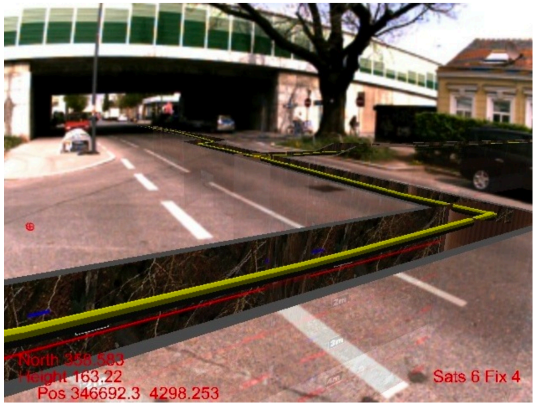
Stencil Testing

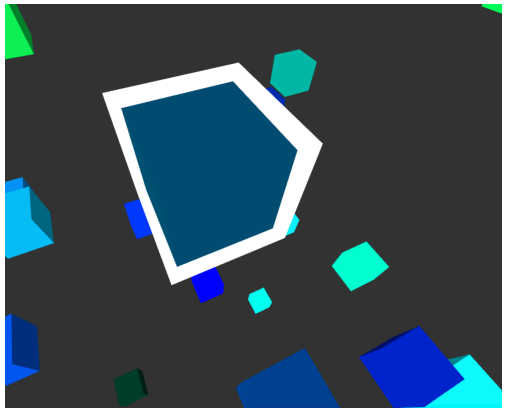
- 2D array storing integer values (e.g., 8-bit, 0-255) for each pixel
- Purpose:
  - Acts as a per-pixel mask or filter
  - It allows to control when and where rendering operations can occur
- How it Works (Stencil Test):
- "Draw" a pattern into the stencil buffer first.
- Configure OpenGL to only draw subsequent objects where the stencil buffer meets certain conditions (e.g., "stencil value equals 1")

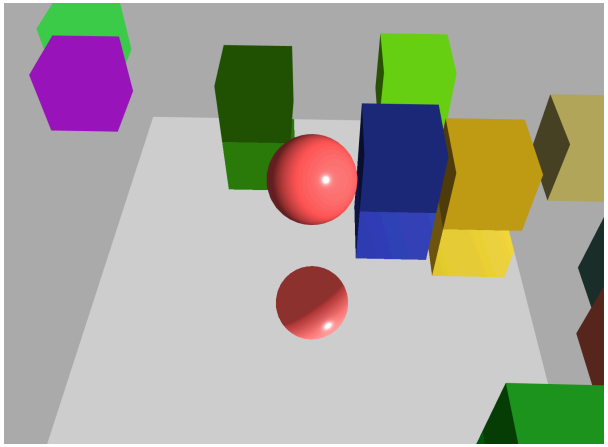


Stencil Buffer

- Can be used to create advanced rendering effects:
  - Cutouts
  - Outline
  - Reflection



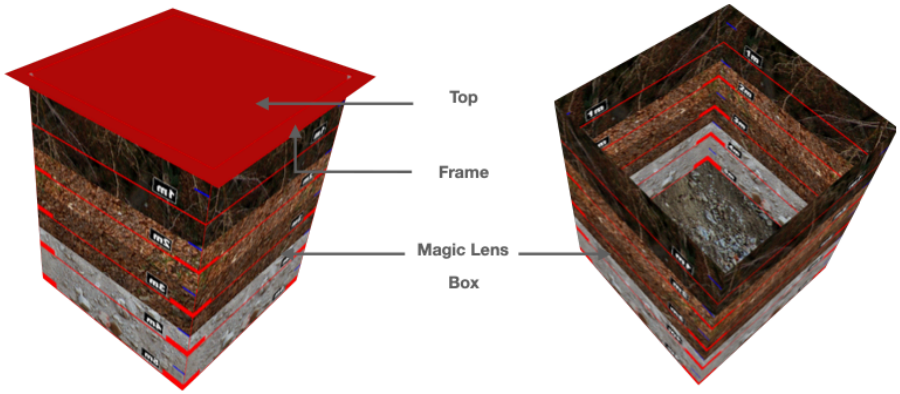


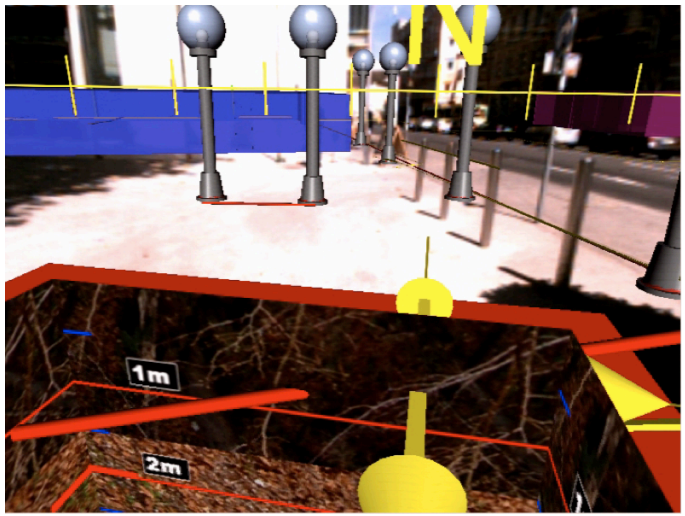


Stencil Buffer for X-Ray Vision in AR

Rendering Method

1. Render video texture
2. Render magic lens box
3. Render the top of the magic lens box into the stencil buffer
4. Render virtual data
5. Render transparent video texture with enabled stencil test
6. Render frame geometry



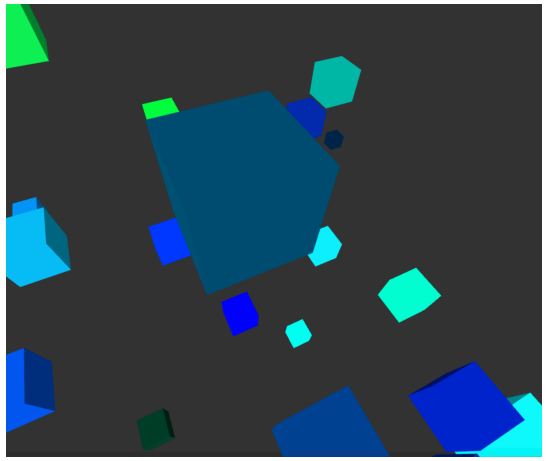


Zollmann et al. “Comprehensible and interactive visualizations of GIS data in augmented reality”

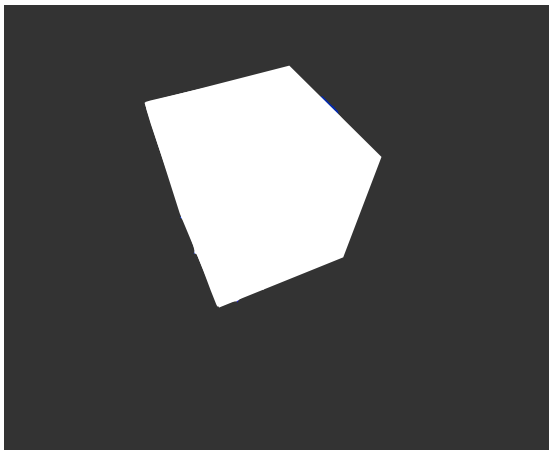
Stencil Buffer - Creating Outlines

- 1. Render original scene
- 2. Write to Stencil Buffer:
  - Enable the stencil buffer and disable writing to the colour buffer
  - Render scaled model and configure the stencil operation to write a value (e.g., 1)
- 3. Render normal-sized mode
  - Configure the stencil operation to write a new value (e.g., 0) for every pixel that is rendered.
  - Leaves only the surrounding outline pixels with a value of 1
- 4. Render the model at its larger size
  - Enabling the colour buffer
  - Configure stencil test to only pass fragments where the stencil buffer's value is 1

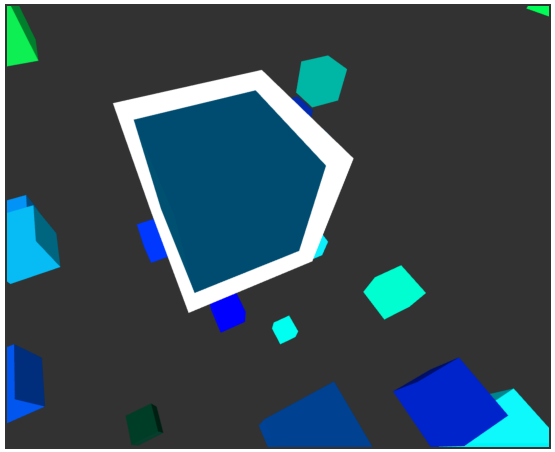
Original Scene



Stencil Mask

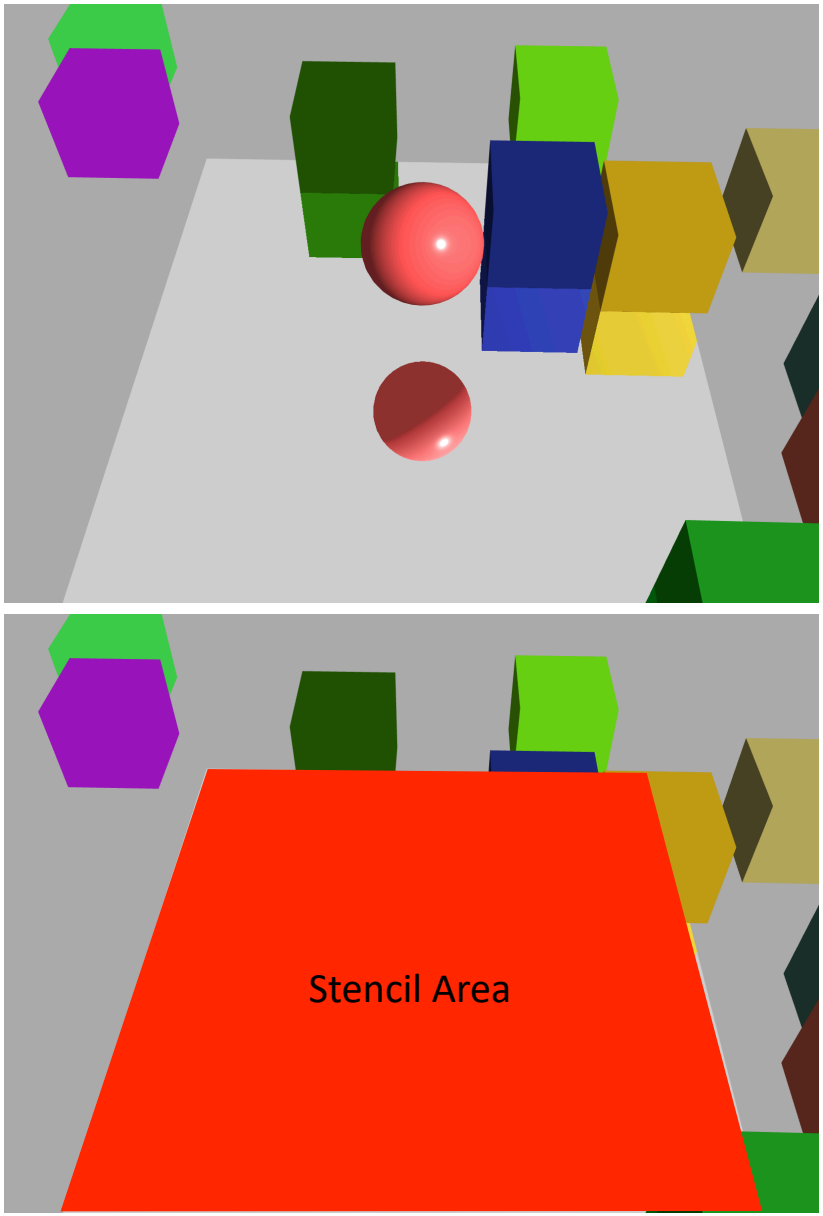


Outline



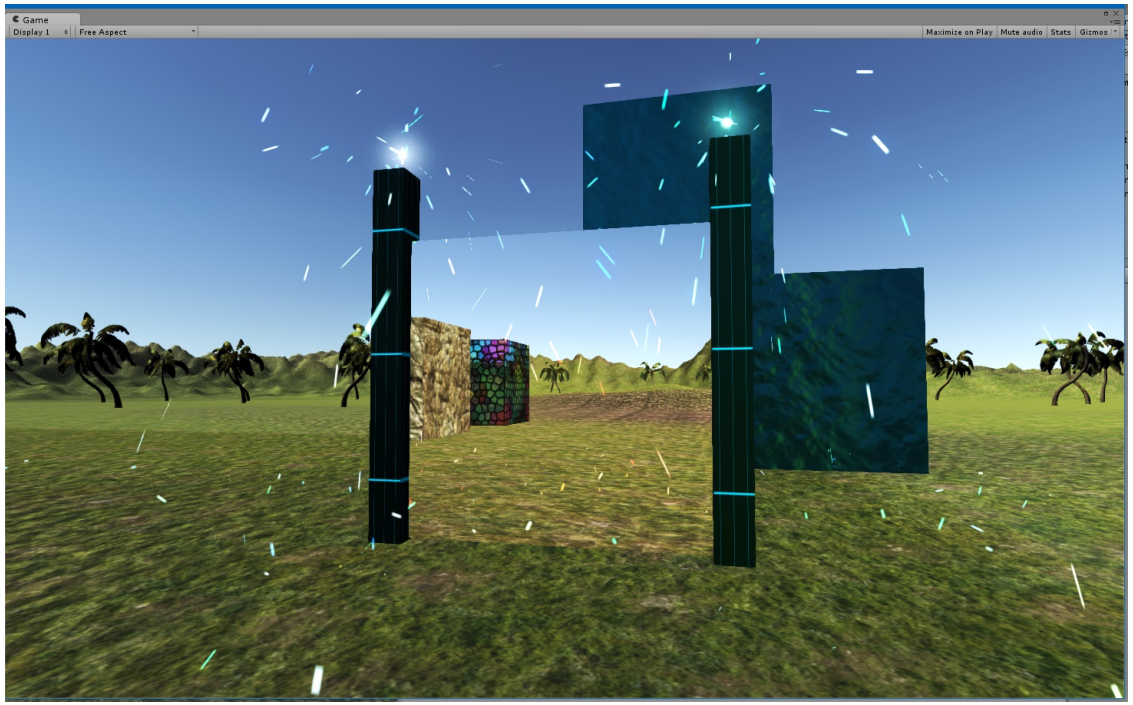
Stencil Testing for Reflections

- 1. Render the original models normally (writing to the colour and depth buffers)
- 2. Render the mirror plane:
  - Configure the stencil operation to write a reference value (e.g., 1) to the stencil buffer for every pixel that passes the depth test (glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE)).
  - Depth writing is disabled (glDepthMask(false)) to prevent mirror from occluding the reflection
- 3. Render the mirrored (flipped) models:
  - Configure the stencil test to only pass fragments with stencil buffer's value = 1 (glStencilFunc(GL_EQUAL, 1, 0xFF)).
  - "Masks" reflection, confining it to the pixels on mirror plane



3-Min Discussion: Create a portal

- You are asked to build a portal into your game. How can you make use of Stencil Buffers to implement this?

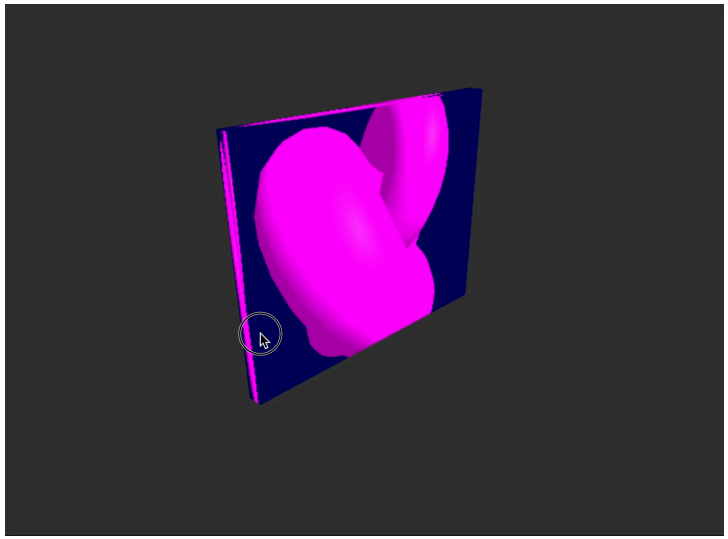


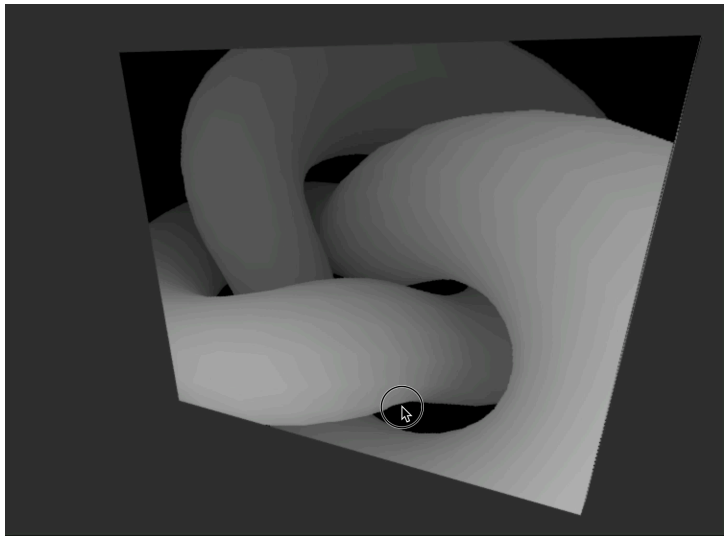
https://discussions.unity.com/t/creating-an-effective-portal-system/621535/14

03:00

Render to Texture

- Concept:
  - Instead of drawing just once to the screen, scene is rendered in multiple stages
  - Key is to render the first pass into an off-screen Framebuffer Object (FBO), which saves the result as a texture
- Pipeline:
  - Pass 1 (Render-to-Texture): Render scene (e.g., all 3D objects) into the FBO -> colour buffer is now a texture
  - Pass 2 (Render-to-Screen):
    - Draw a simple full-screen quad
    - Apply Texture: Apply the texture you just created in Pass 1 to this quad





Multi-Pass Rendering

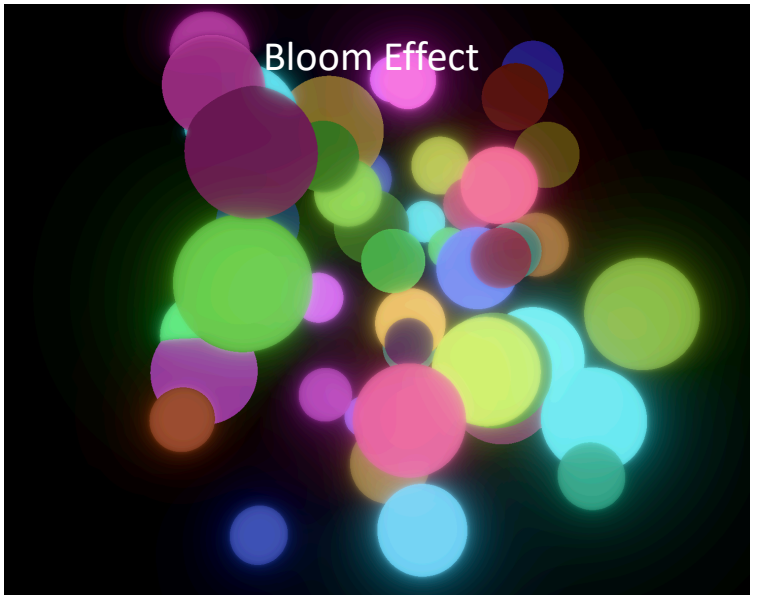
- Multi-pass rendering is the technique of rendering a scene multiple times within a single frame to build up a complex final image
- Output of one pass is saved as a texture and then used as the input for the next rendering pass
- "Ping-Pong" Technique: Using at least two FBOs (A and B) to "ping-pong" the image between, it is not possible to (safely) read from and write to the same texture in one pass
- Key Techniques:
  - Frame Buffer Objects (FBOs)
  - Render to Texture

Multi-Pass Rendering

- How:
  - Pass 1: Render scene (e.g., scene colours) → FBO "A" (generates sceneTexture).
  - Pass 2: Draw a full-screen quad. Read from sceneTexture → Apply blur shader → FBO "B" (generates blurredTexture).
  - Final Pass: Draw another quad. Read from blurredTexture → Draw to Main Screen

Multi-Pass Rendering - Examples

- Post-Processing Effects (Blur, Bloom):
  - Pass 1: Render your full 3D scene to an FBO
  - Pass 2: Draw a full-screen quad, read the FBO's texture, and apply an effect shader (like a blur)
- Deferred Shading:
  - Pass 1: Render scene data (position, normals, colour) into multiple textures attached to one FBO (this is called a "G-Buffer")
  - Pass 2: Read from the G-Buffer textures to calculate complex lighting for hundreds of lights at once



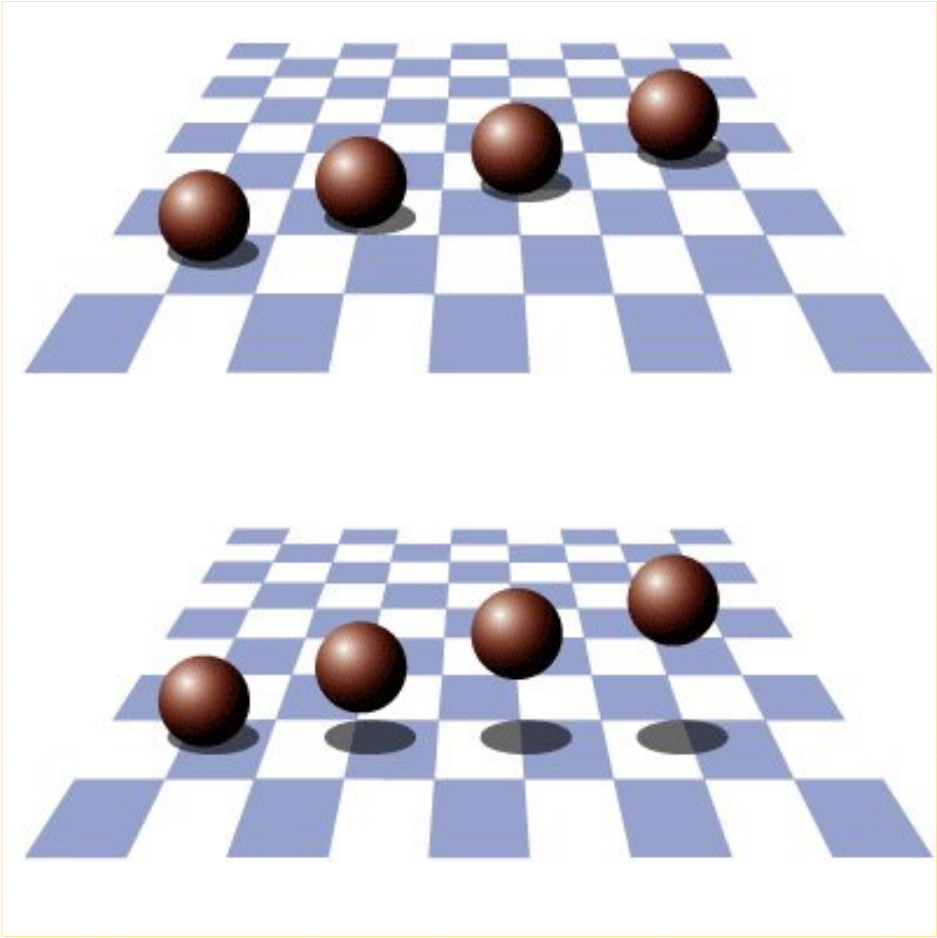
https://docs.unity3d.com/

Shadows

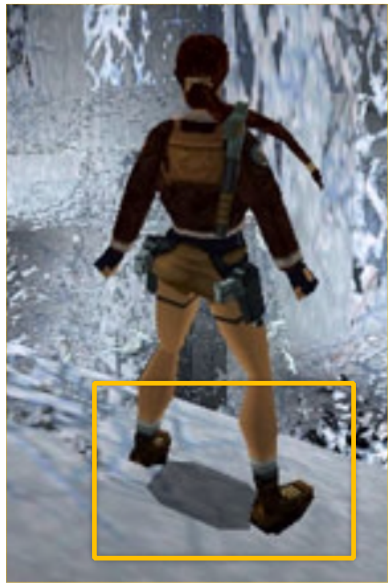
- Important depth cue
- Spatial Relationship between objects
- Realism
- Provides information about scene lighting

Shadows as Depth cue

- Important depth cues
- Provide information about relative locations



Shadows For Realism

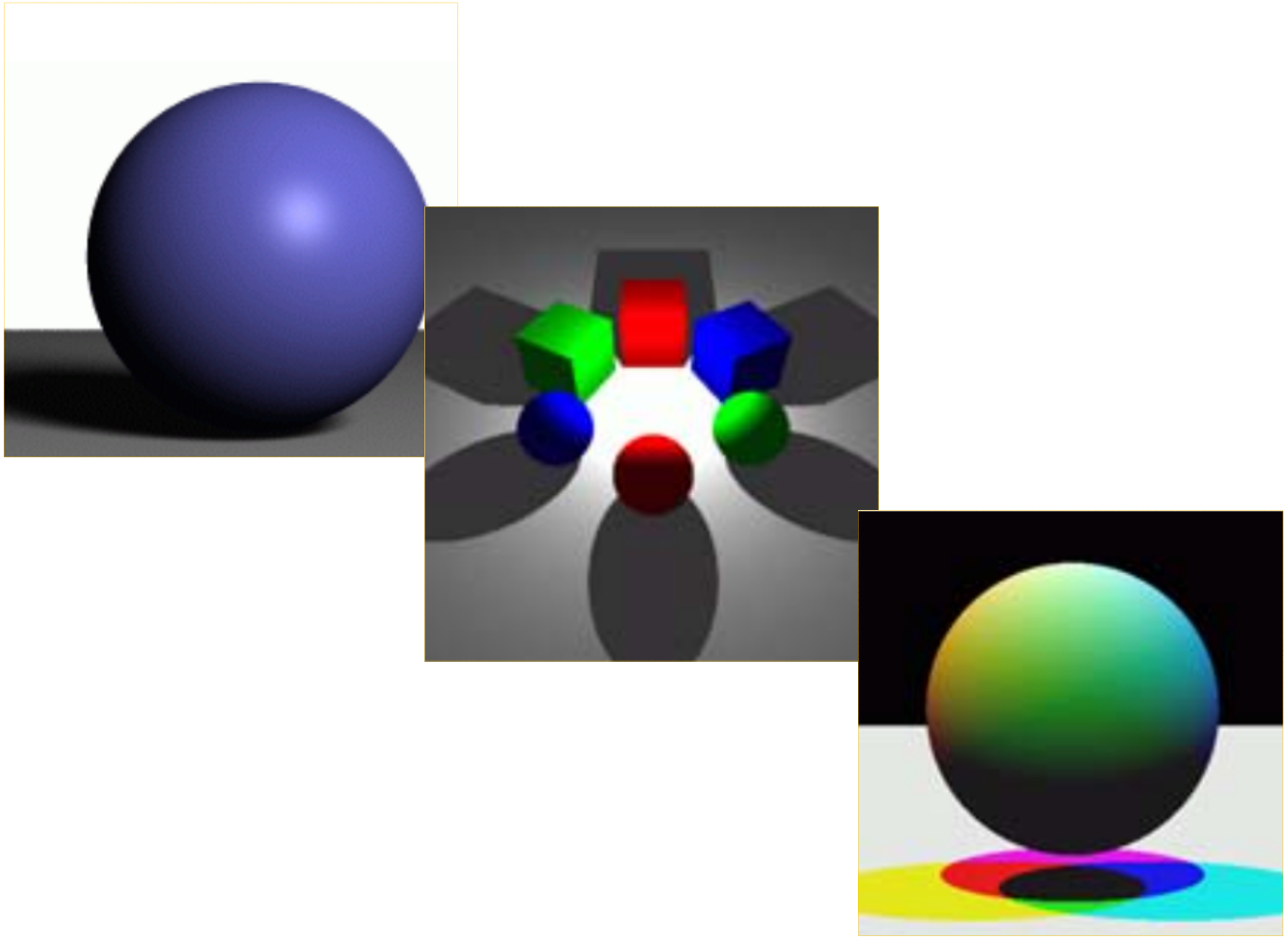


Examples:

- Missing Shadows
- Wrong Shadows

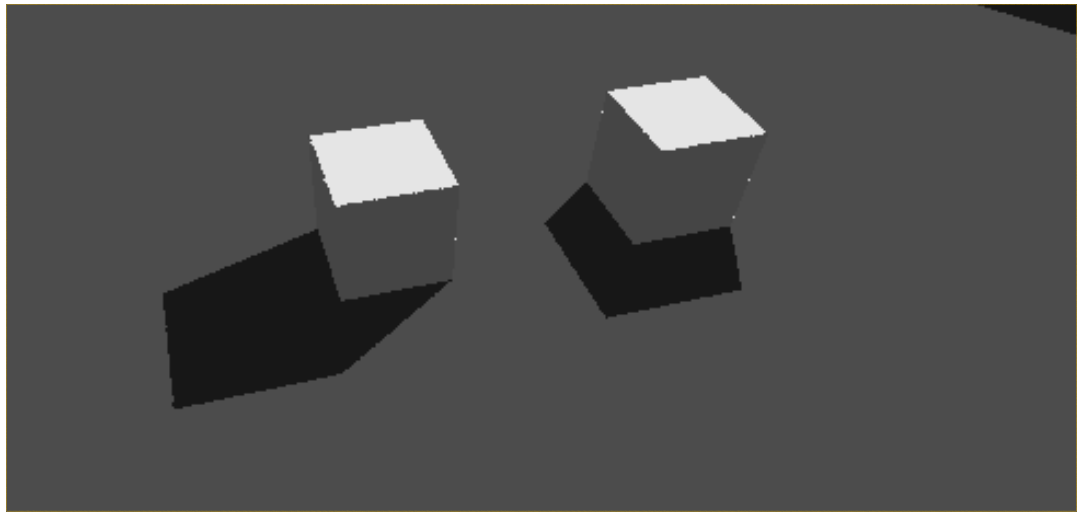
Scene Lighting

- Position of light sources
- Types of lights:
- Hard shadows vs. soft shadows
- Point lights have hard edges
- Area lights have soft edges
- Coloured light sources



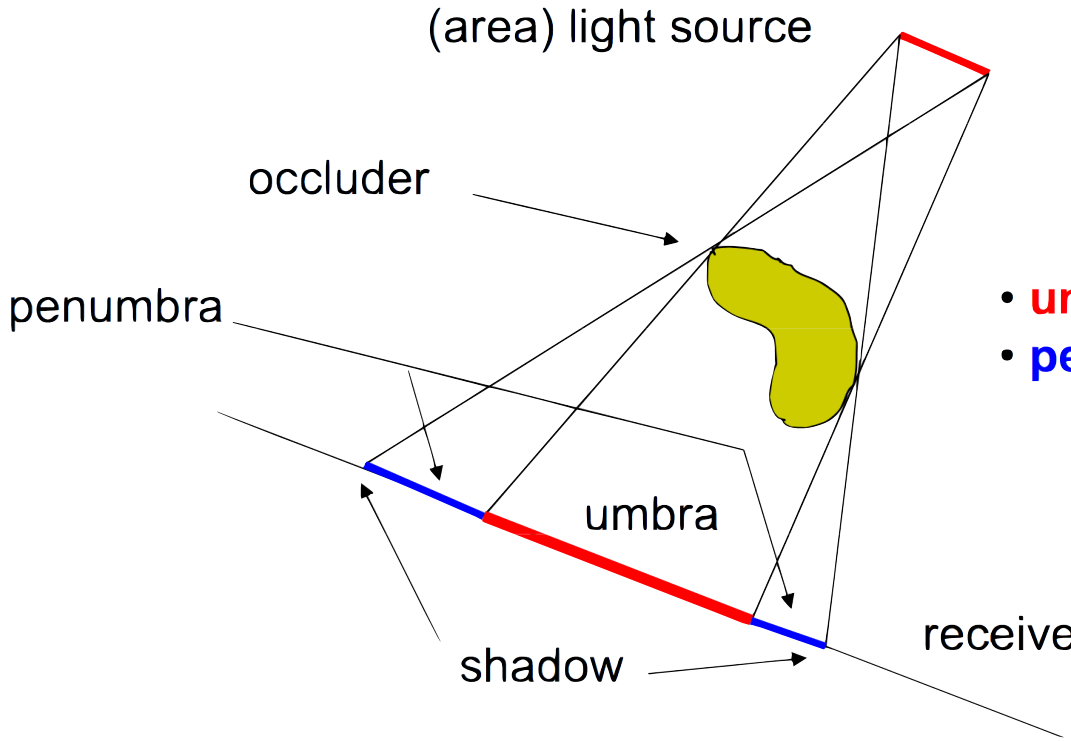
What are shadows?

- Shadows are areas hidden from a light source
- Surface is only illuminated if nothing blocks its view of the light



Terminology

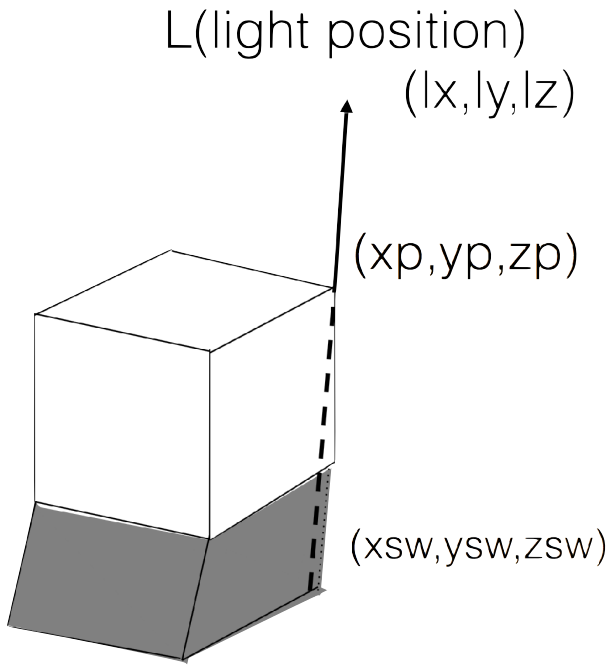
- Point lights have hard edges
- Area lights have soft edges



- **umbra** – fully shadowed region
- **penumbra** – partially shadowed region

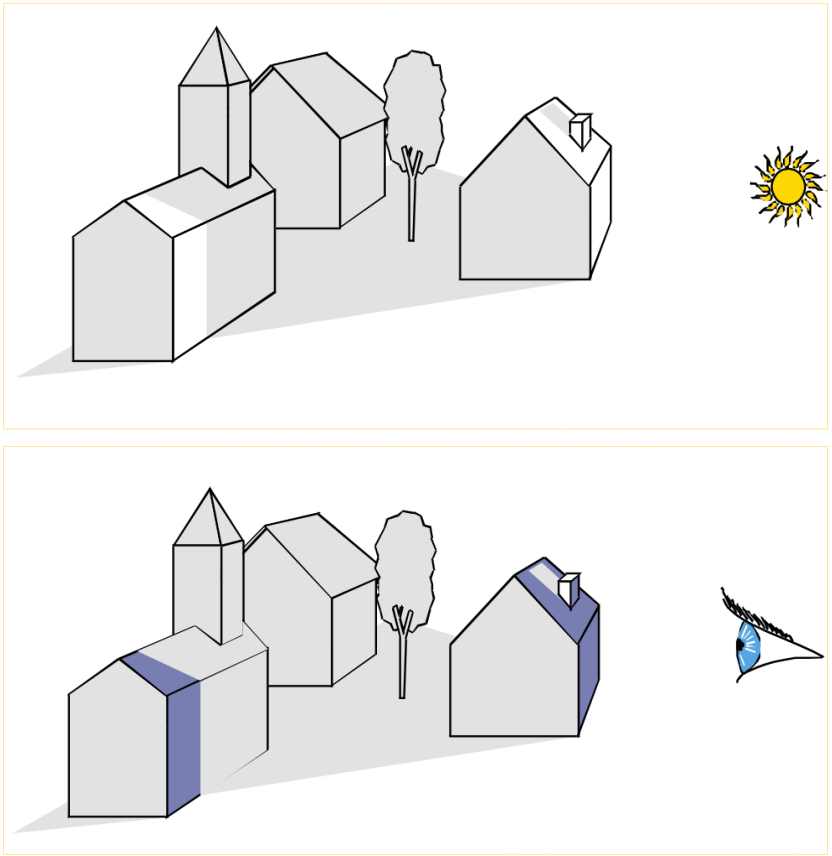
Projective Shadows

- Easy to compute
- Simply project geometry on ground plane
- Only works on flat shadow receivers, e.g. car on street, player on ground
- No self-shadows
- Expensive: needs to be rendered every frame even for static scenes



Shadow/View Duality

- A point is in shadow if it not visible from the perspective of the light source
- Use the view from the light source to compute shadows

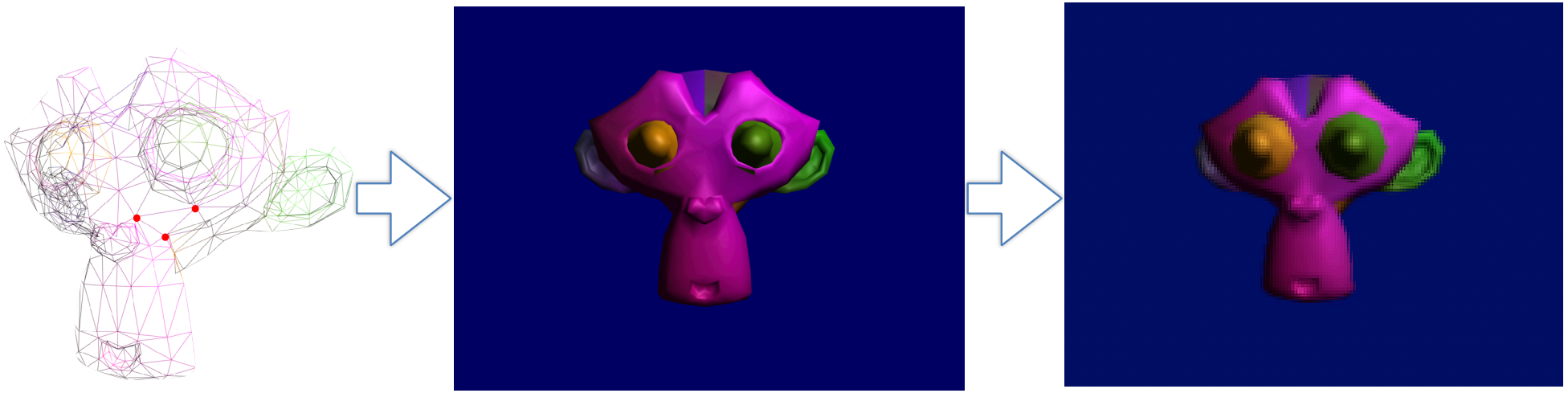


Shadow Mapping

- Use shadow/view duality
- Two rendering passes
  - 1st Pass: Rendering shadow map representing the depth from light source
  - 2nd Pass: Render final image from camera view and check shadow map to see if points are in shadow

Remember: Render-to-Texture

- Render scene into a texture
- Can be used for post render effects
- Can be used for shadow mapping



3D Mesh

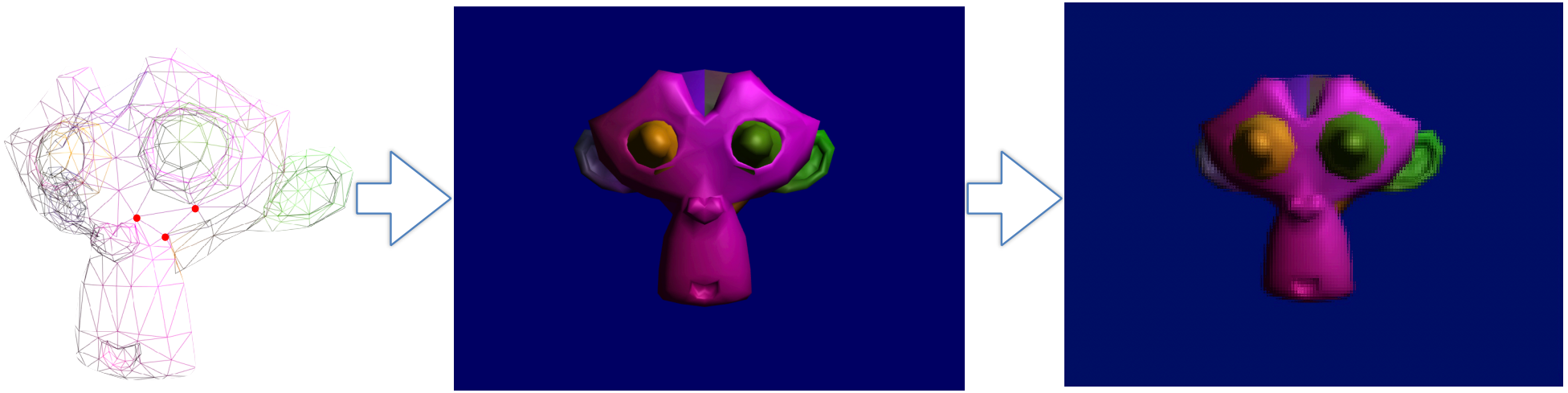
Rendered into a texture

Post processing step

Remember: Render-to-Texture

```
// Render to framebuffer
  glBindFramebuffer(GL_FRAMEBUFFER, FramebufferName);
  myScene->render(myCamera);
```

```
// Render to screen
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, renderedTexture);
outputQuad->directRender();
```



3D Mesh

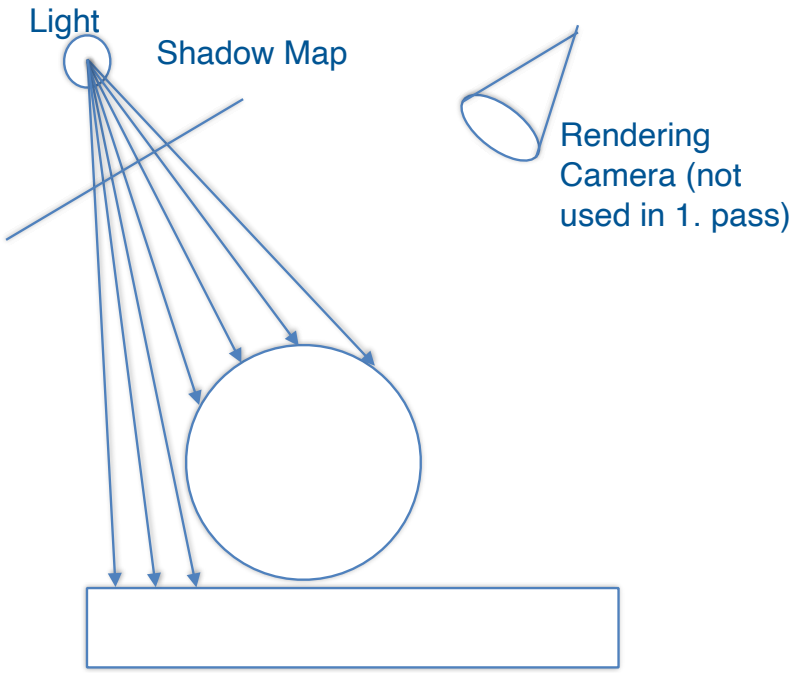
Rendered into a texture

Post processing step

First Pass: Render Shadow Map

Create a map of depth values as seen from the light's point of view

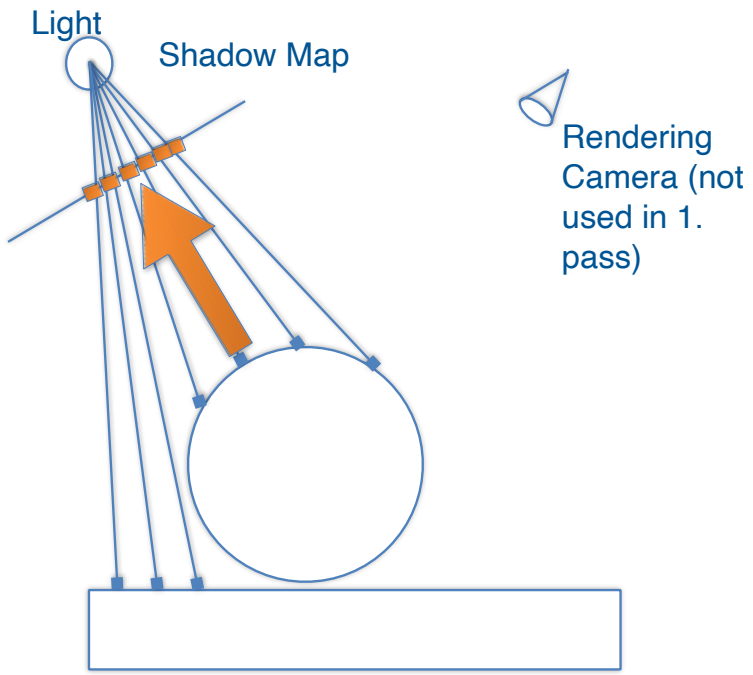
- Geometry is rendered into a depth buffer from the point of view of the light
- Transform the geometry into light-view space



First Pass: Render Shadow Map

Create a map of depth values as seen from the light's point of view

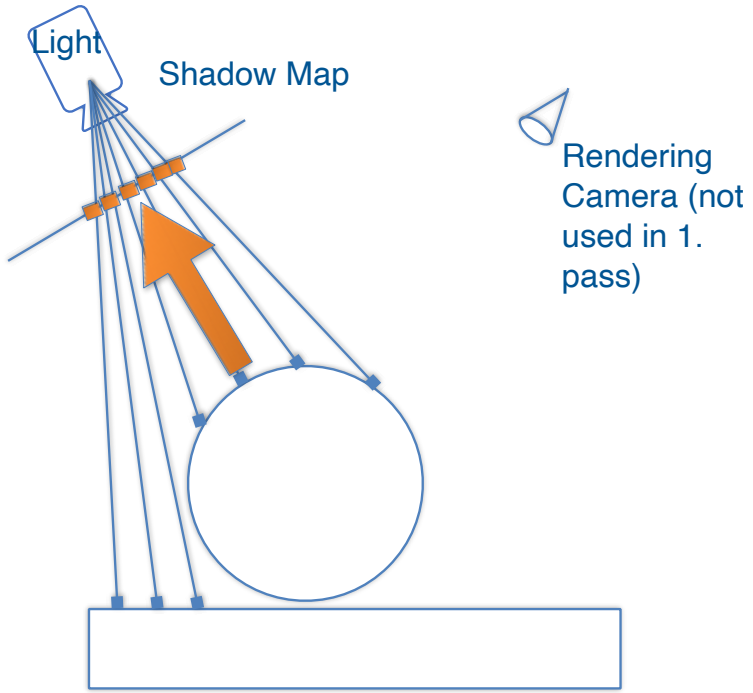
- Geometry is rendered into a depth buffer from the point of view of the light
- Transform the geometry into light-view space



First Pass: Render Shadow Map

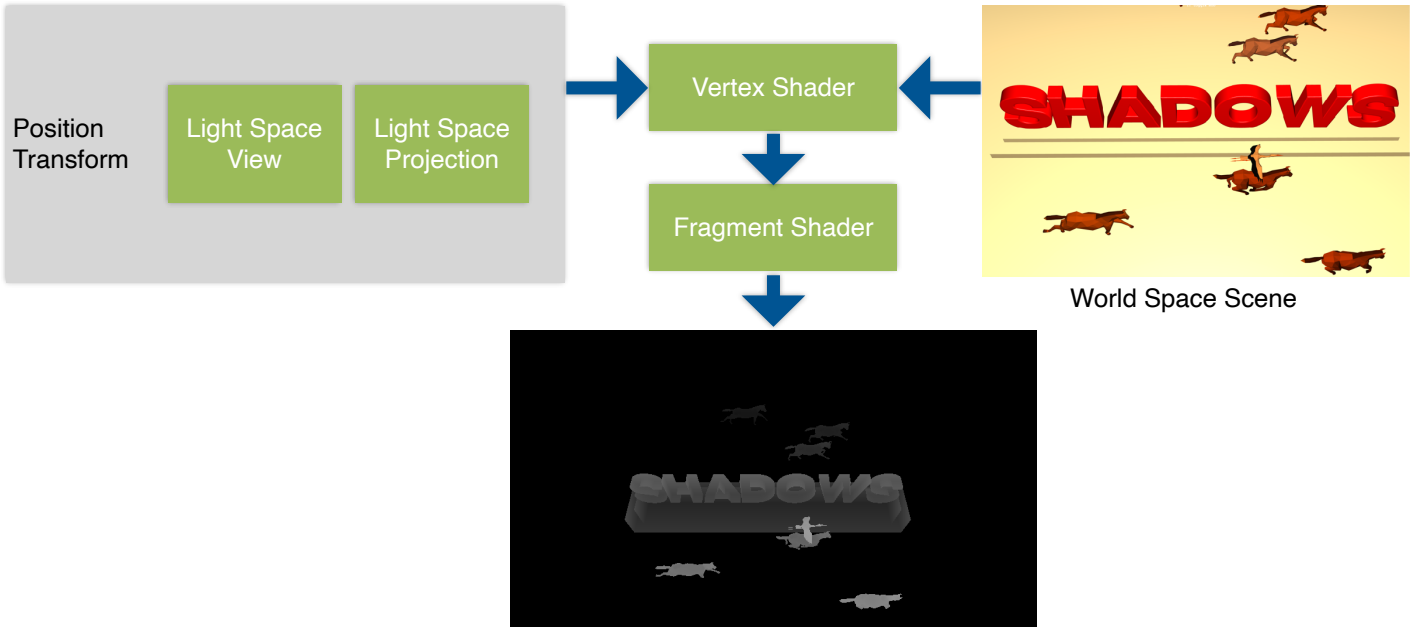
Create a map of depth values as seen from the light's point of view

- Geometry is rendered into a depth buffer from the point of view of the light
- Transform the geometry into light-view space



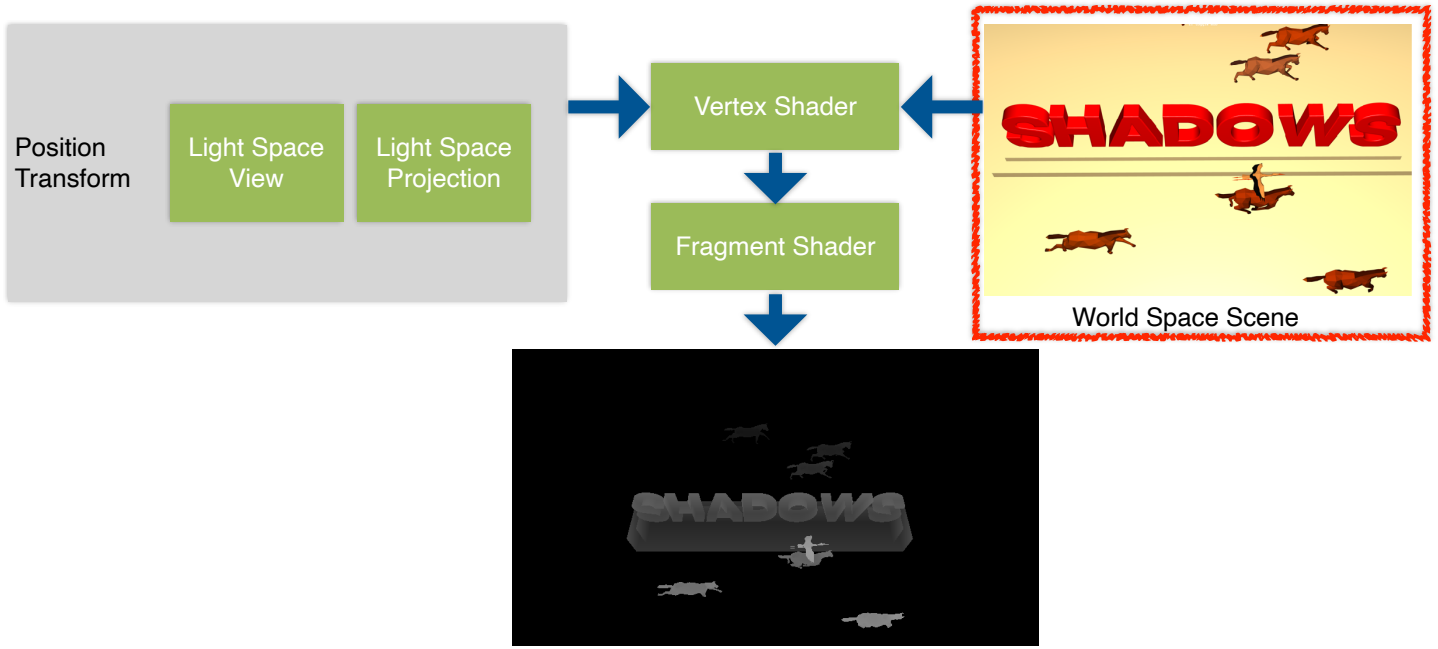
First Pass: Render Shadow Map

In practice: Vertex shader transforms the geometry into light-view space



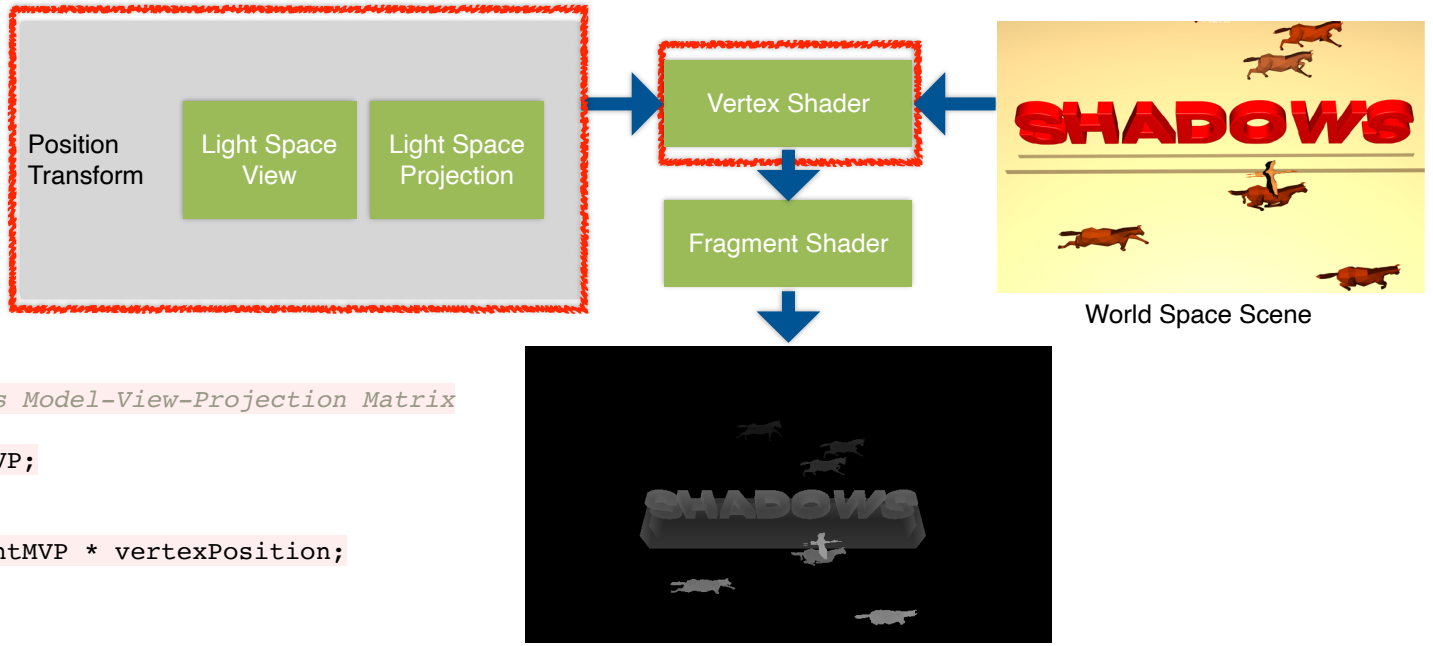
First Pass: Render Shadow Map

In practice: Vertex shader transforms the geometry into light-view space



First Pass: Render Shadow Map

In practice: Vertex shader transforms the geometry into light-view space

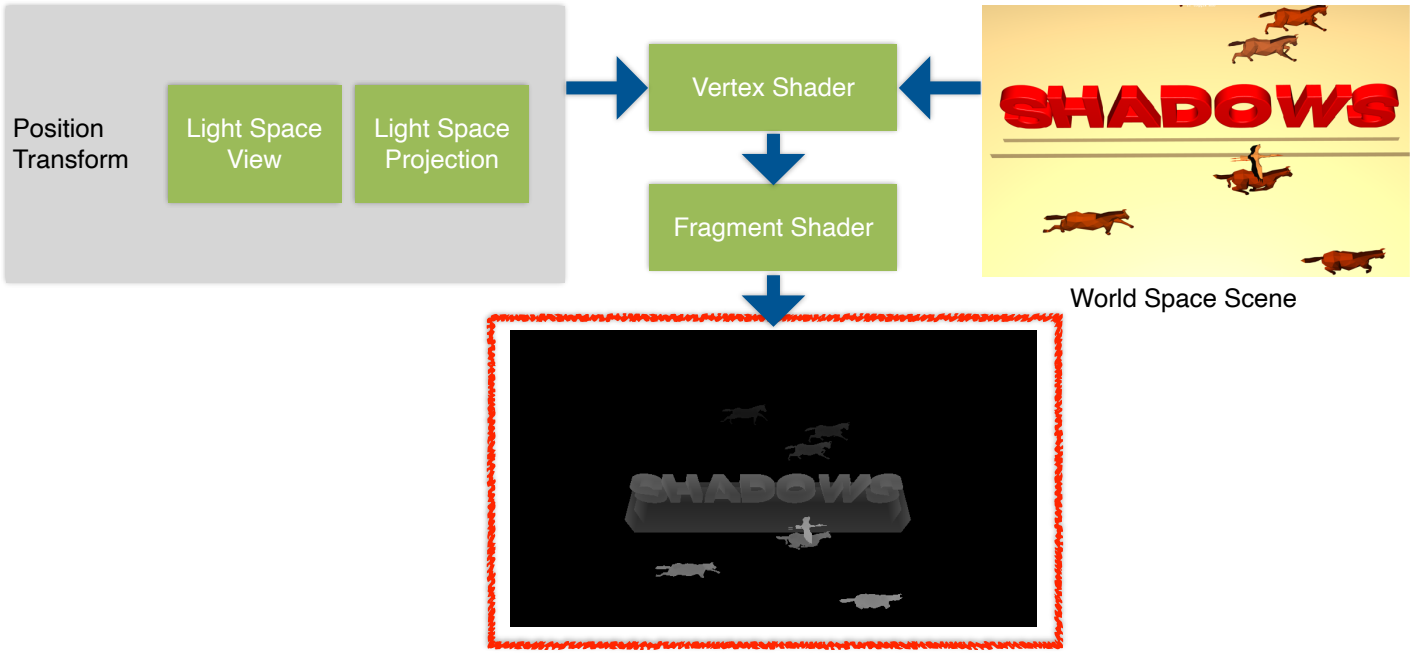



```
// lightMVP contains Model-View-Projection Matrix
// for light source
uniform mat4 lightMVP;

void main(){
 gl_Position =  lightMVP * vertexPosition;
}
```

First Pass: Render Shadow Map

In practice: Vertex shader transforms the geometry into light-view space

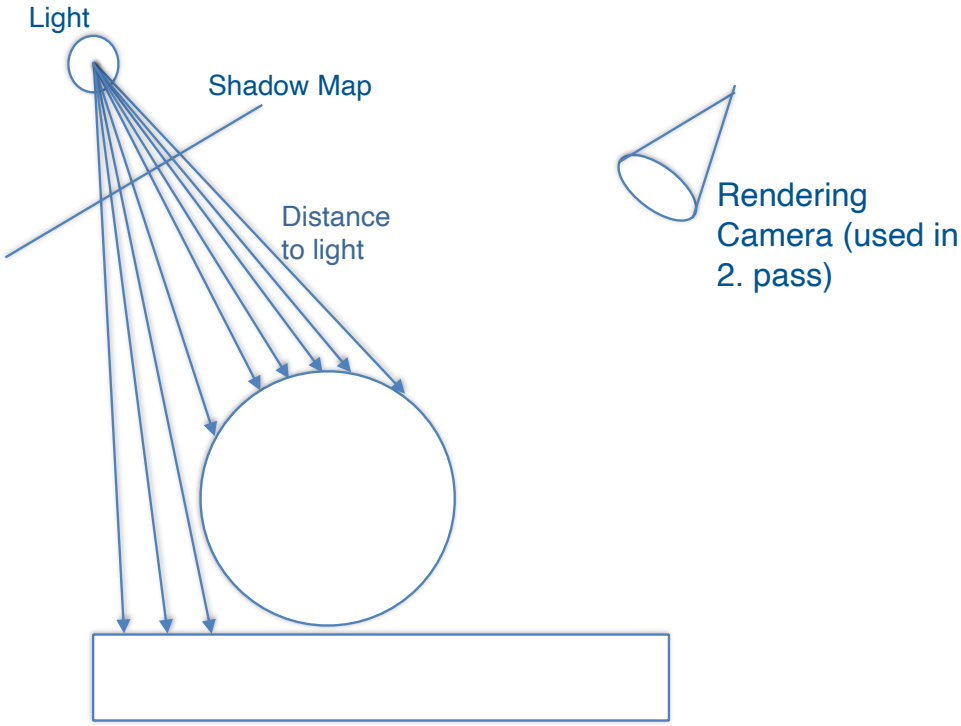


First Pass: Render Shadow Map

- Result from render to texture: shadowMap
- Use shadowMap as input for second rendering pass

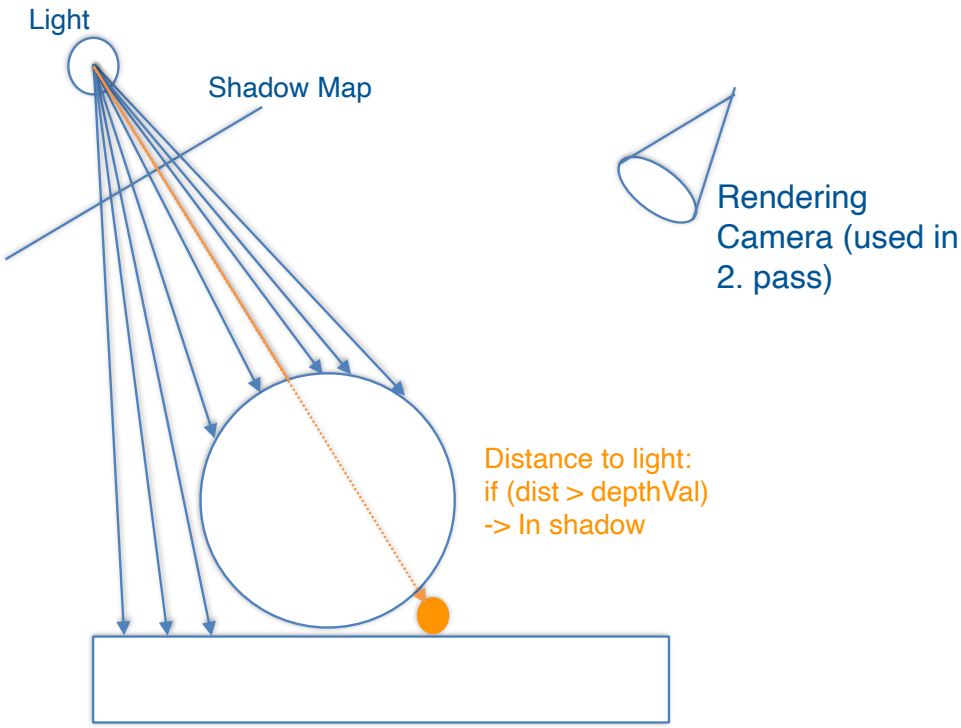
Second Pass: Render Scene

- Render final output from camera view and check shadow map to see if points are in shadow
- Compare distance between point and light source and value in shadow map
- If distance larger than value in shadow map point is in shadow



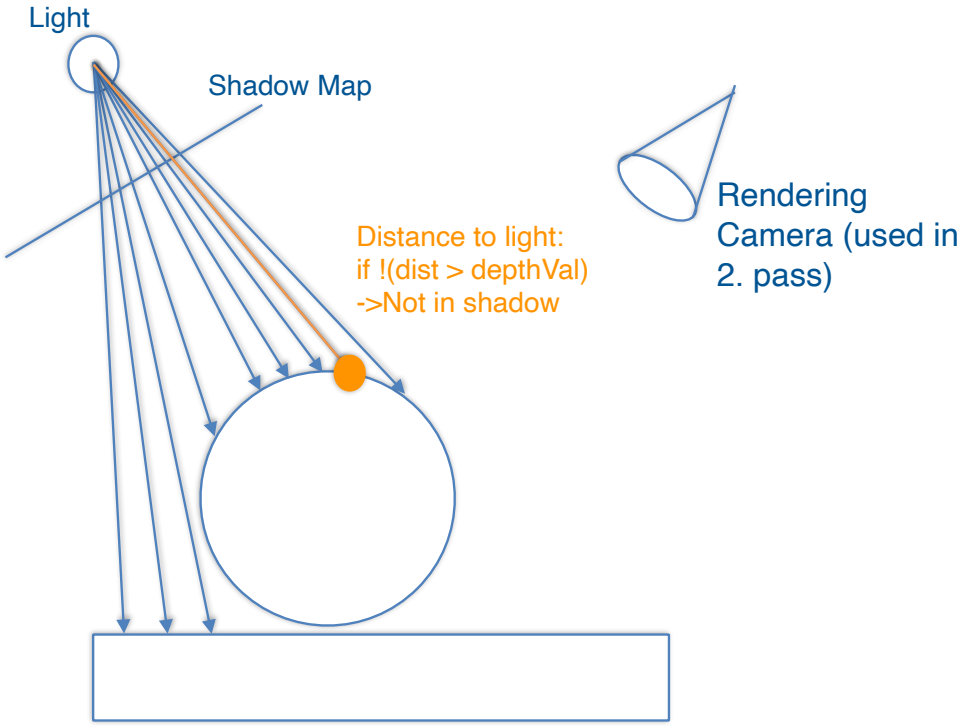
Second Pass: Render Scene

- Render final output from camera view and check shadow map to see if points are in shadow
- Compare distance between point and light source and value in shadow map
- If distance larger than value in shadow map point is in shadow



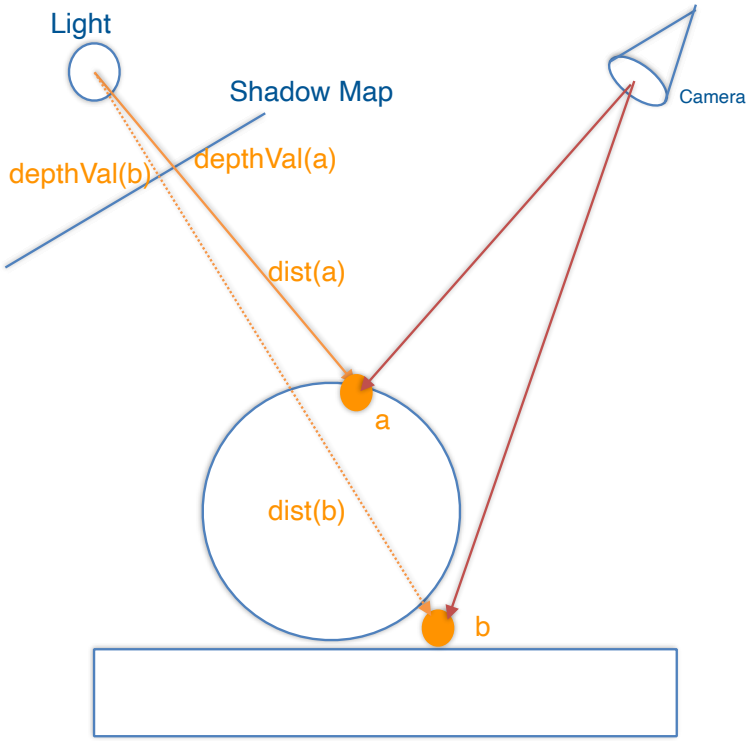
Second Pass: Render Scene

- Render final output from camera view and check shadow map to see if points are in shadow
- Compare distance between point and light source and value in shadow map
- If distance larger than value in shadow map point is in shadow



Second Pass: Render Scene

- Transform points (p) to light space
- Transform points to shadow map
- Look up value from shadow map (depthVal(p))
- Compute visibility based on:
  - dist(p) > depthVal(p)
- **In practice**: use vertex shader and fragment shader



Second Pass: Render Scene

- Transform points (p) to light space
- Transform points to shadow map
- Look up value from shadow map (depthVal(p))
- Compute visibility based on:
  - dist(p) > depthVal(p)
- **In practice**: use vertex shader and fragment shader

Vertexshader:

```
// Output position of the vertex, in clip space : MVP * position
gl_Position =  MVP * vec4(vertexPos_modelspace,1);

// Mapping into shadow map coords using the light's view matrix
shadowCoord = textureSpace* lightMVP * vec4(vertPos_modelspace,1);
```

Fragmentshader:

```
float visibility = 1.0;
float depthVal = texture( shadowMap, shadowCoord.xy ).z;
float dist = shadowCoord.z;
if ( dist > depthVal){
    visibility = 0.0;
}
```

Second Pass: Render Scene

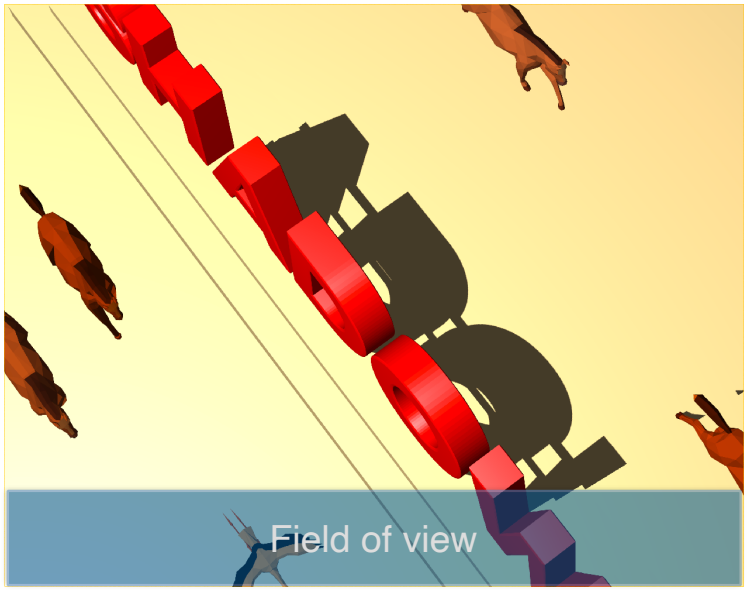
Final Shading Step

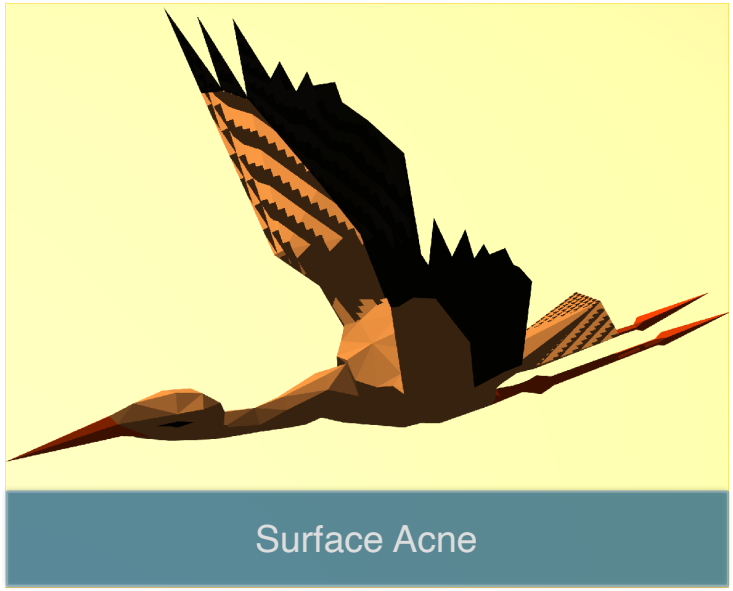
- Use calculated visibility to compute fragment's output colour
- **In practice:** use visibility in fragment shader:

```
outputColor =
 // Ambient : simulates indirect lighting
 ambientColor +
 // Diffuse : "color" of the object
 visibility * diffuseColor
 // Specular : reflective highlight, like a mirror
 visibility * specularColor;
```

Limitations

- Field of View
- Surface Acne
- Aliasing

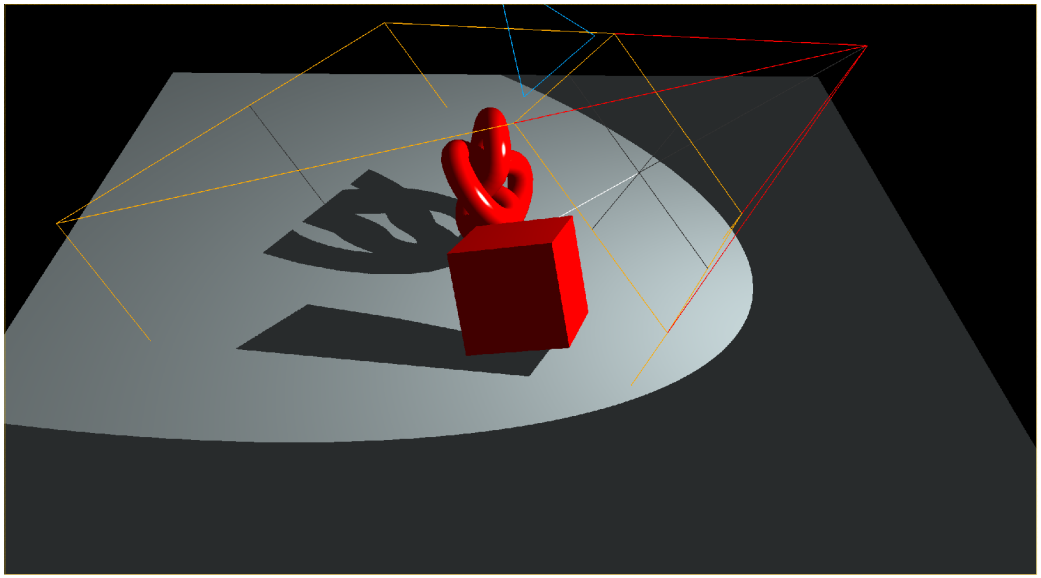

Field of view


Surface Acne


Shadow Map Aliasing

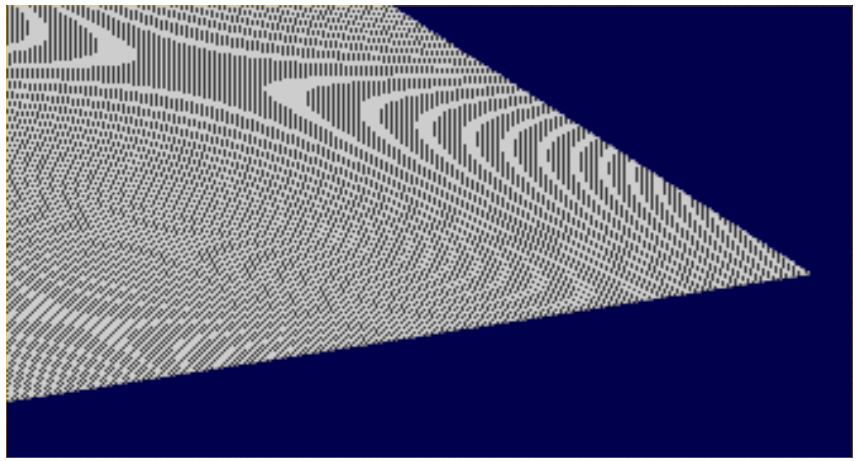
Field of View Problem

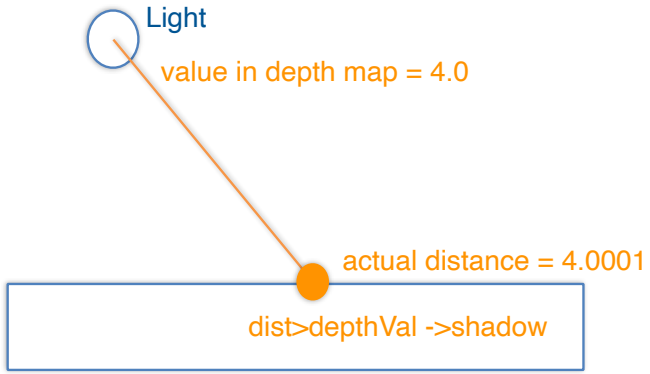
- What if object is outside field of view of shadow map?
- No shadows or partial shadows
- For spot lights, this can be changed by tweaking its range
- Problem in particular for larger scenes



Surface acne

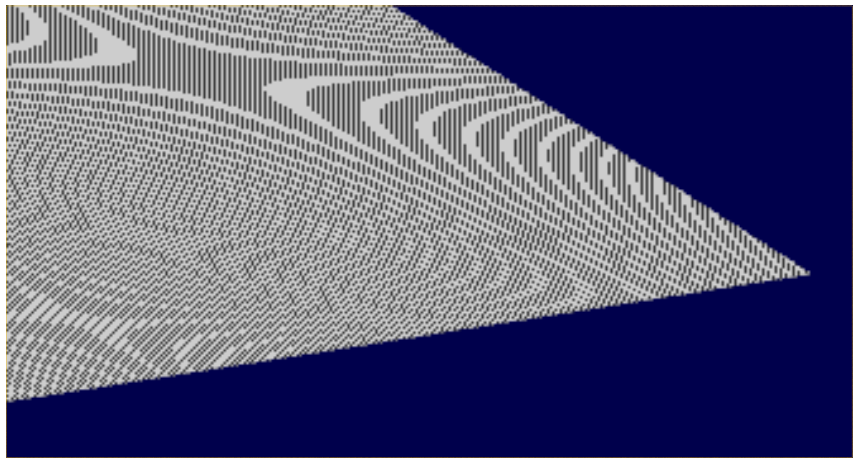
- Self-shadowing problem due to precision and depth map resolution
  - Depth value in map can differs from actual distance between object and light source
  - Sampling problem: neighbouring vertices map to the same depth map pixel

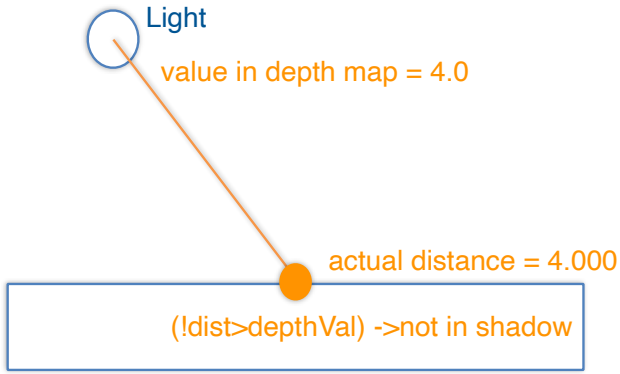




surface acne

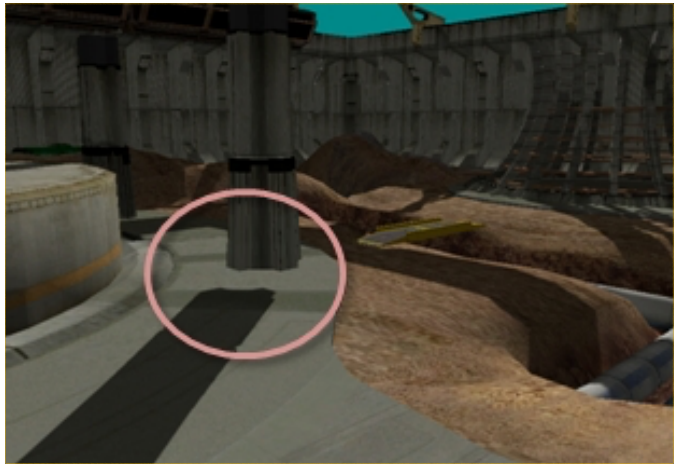
- Self-shadowing problem due to precision and depth map resolution
  - Depth value in map can differs from actual distance between object and light source
  - Sampling problem: neighbouring vertices map to the same depth map pixel

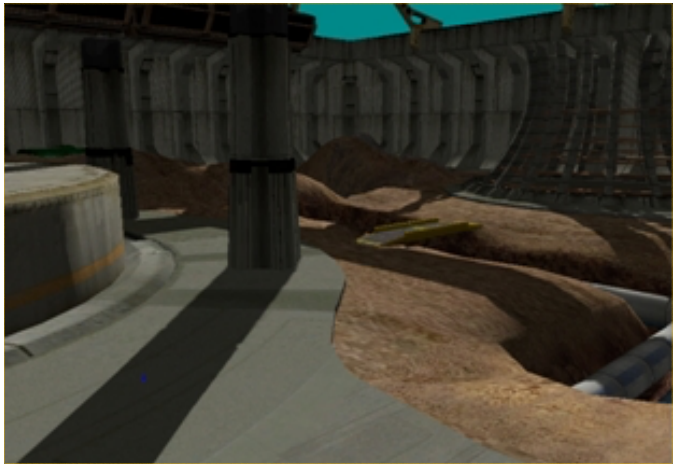




Shadow Map Bias

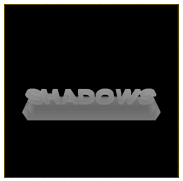
- Solution for surface acne:
  - Shadow map bias for shadow test: ShadowMap(x,y) + bias < dist
  - Choosing a good bias value can be tricky - otherwise Peter Panning



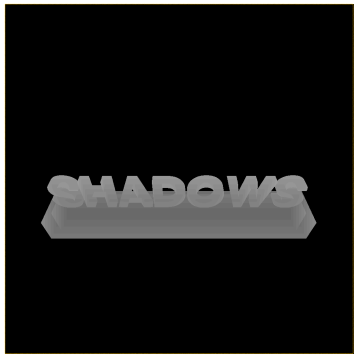


Shadow Map Aliasing

- Quality depends on shadow map resolution
- Higher resolution:
  - Higher quality
  - More memory required



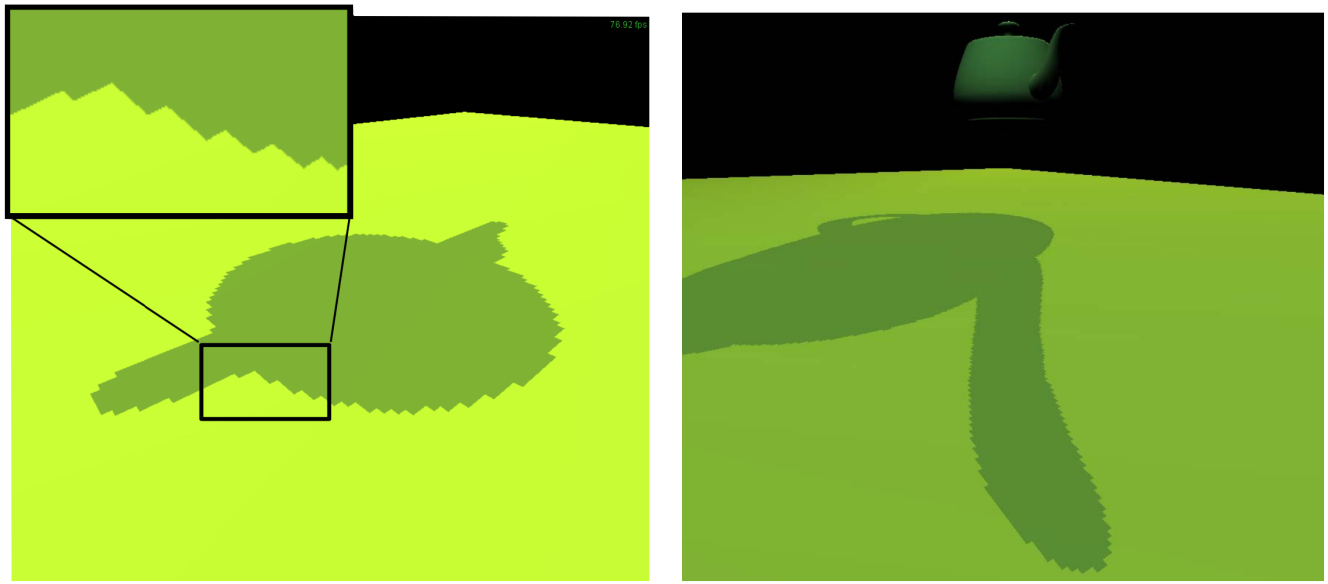
1024x1024



2048x2048

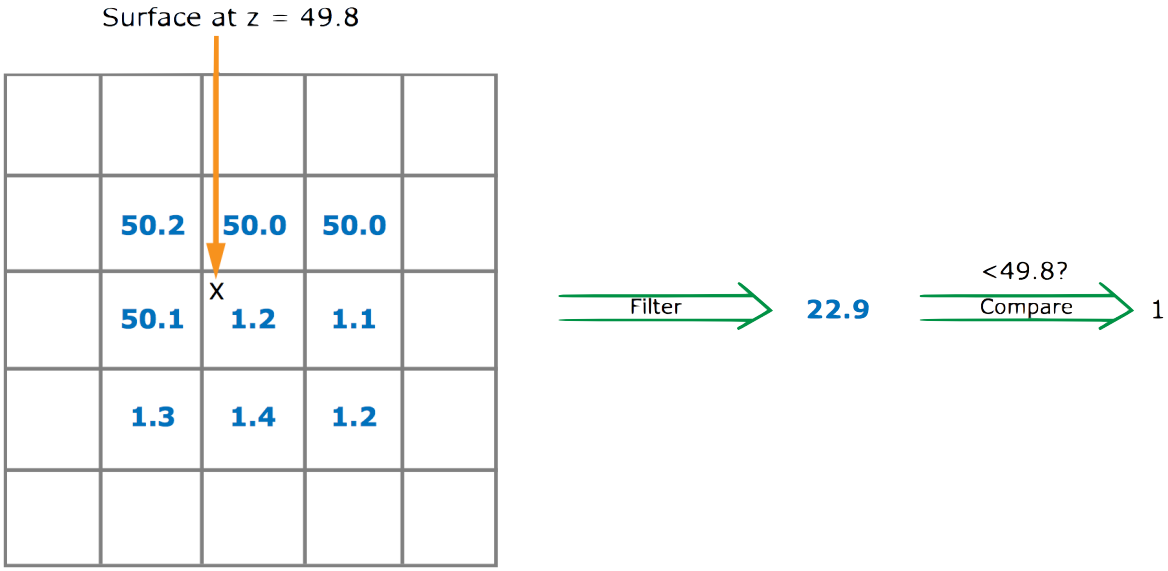
More Solutions for Shadow Map Aliasing

- Under-sampling of the shadow map – Jagged shadow edges
- Solution: Percentage closer filtering (PCF)



Percentage Closer Filtering (PCF)

- Creating softer shadows
- Unlike normal textures, shadow map textures cannot be pre-filtered to remove aliasing
- Weighted average of neighbouring depth values is not meaningful



Ordinary texture map filtering. Does not work for depth maps.

Image by MIT OpenCourseWare.

Percentage Closer Filtering (PCF)

- Creating softer shadows
- Unlike normal textures, shadow map textures cannot be pre-filtered to remove aliasing
- Weighted average of neighbouring depth values is not meaningful

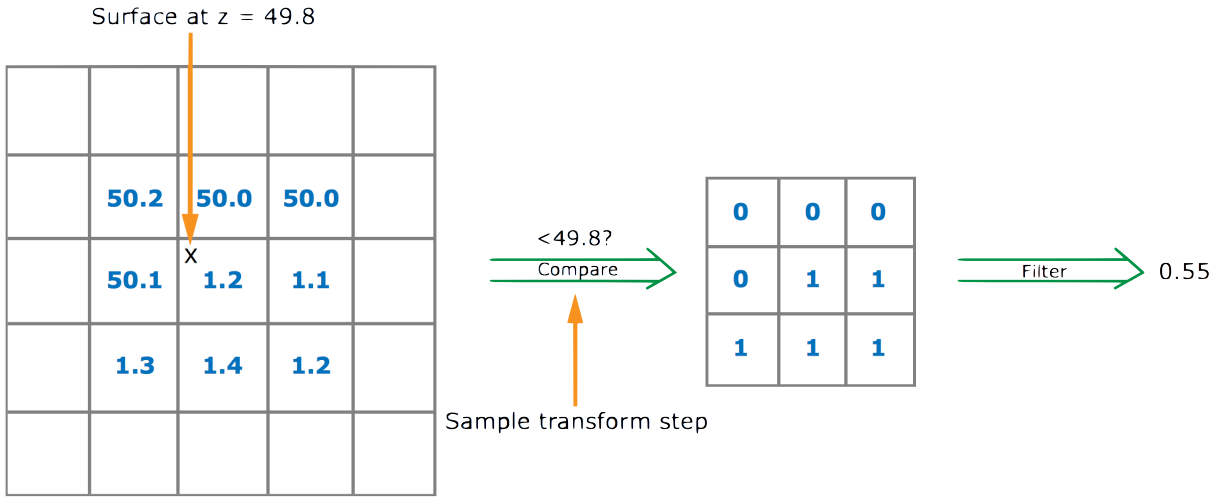
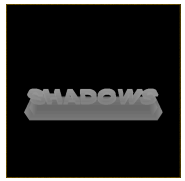


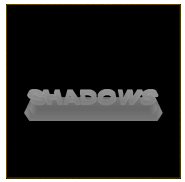
Image by MIT OpenCourseWare.

Percentage Closer Filtering (PCF)

- Interpolation of the results of the shadow map computation
- Quality increases with Percentage closer filtering
- Softer appearance of shadows



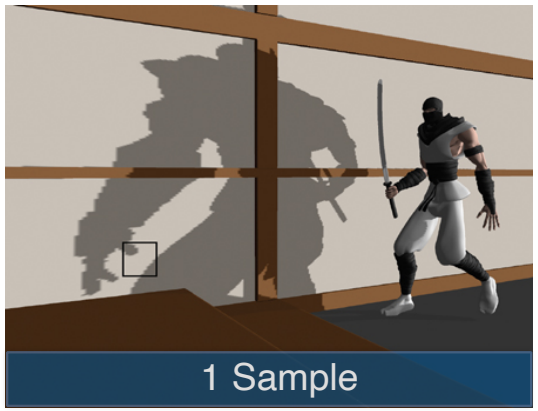
1024x1024

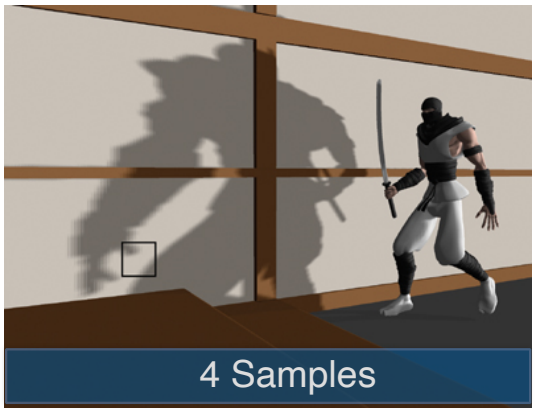


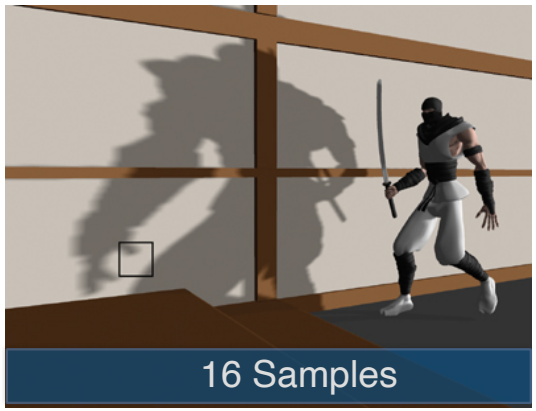
1024x1024

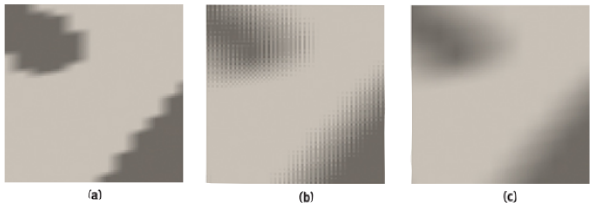
Percentage Closer Filtering (PCF)

- Results for different filter size



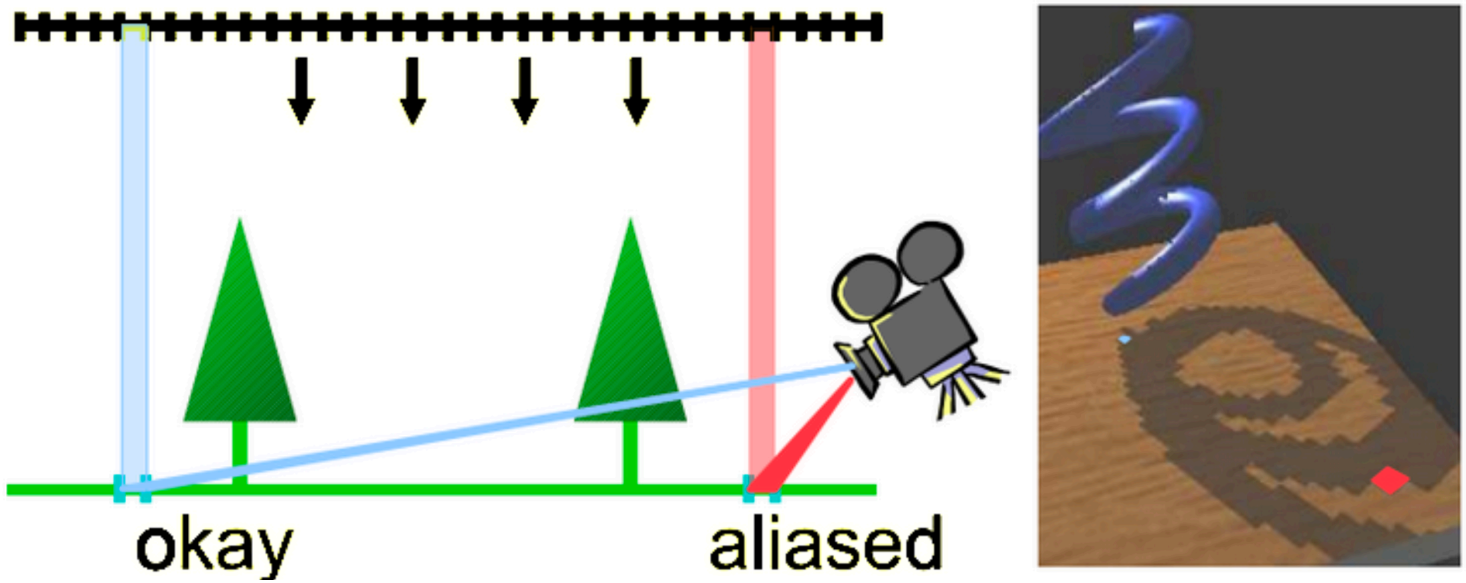






Perspective Aliasing

- View space resolution vs. shadow map resolution

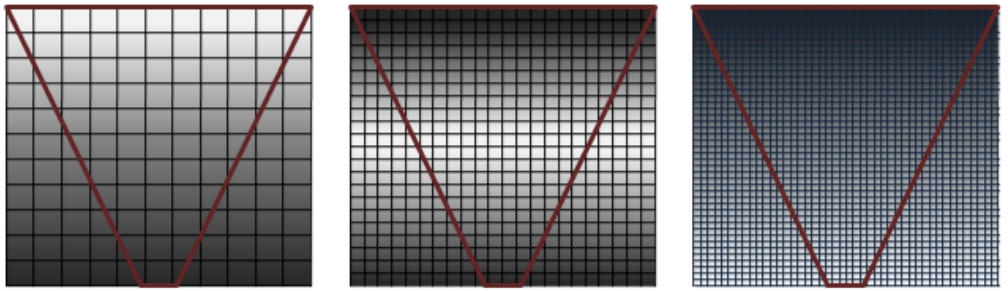


http://www.cg.tuwien.ac.at/~scherzer/files/papers/LispSM_survey.pdf

Cascade Shadow Maps

- Providing higher resolution of the depth texture near the viewer and lower resolution far away

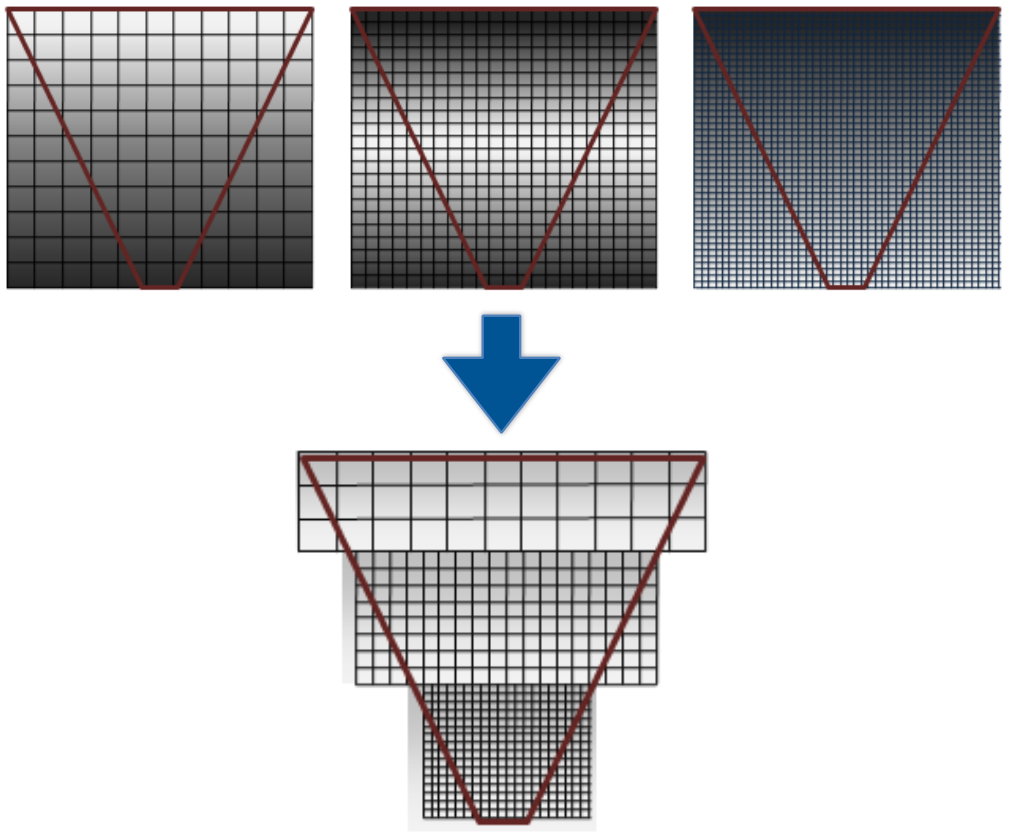
http://msdn.microsoft.com/en-us/library/windows/desktop/ee416307(v=vs.85).aspx



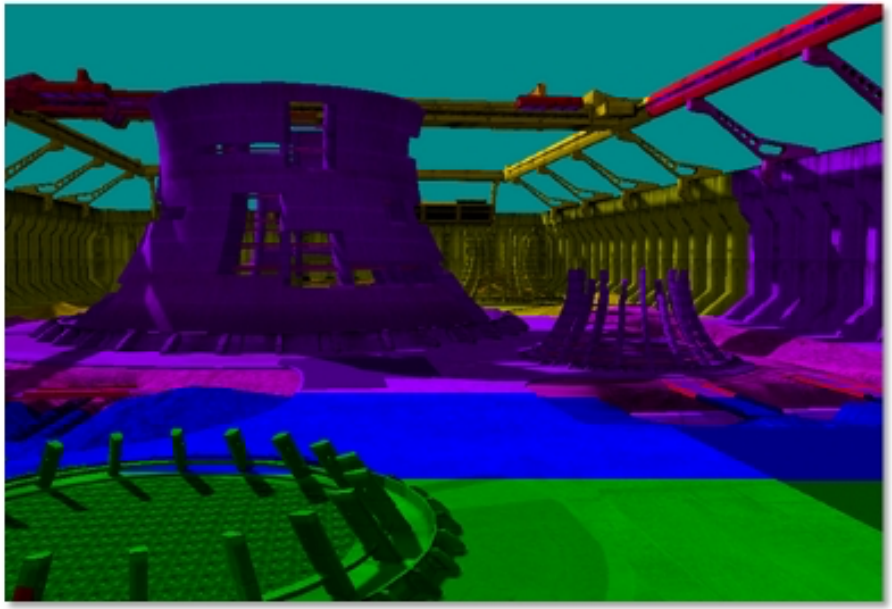
Cascade Shadow Maps

- Providing higher resolution of the depth texture near the viewer and lower resolution far away
- Splitting the camera view frustum and creating a separate depth-map for each partition

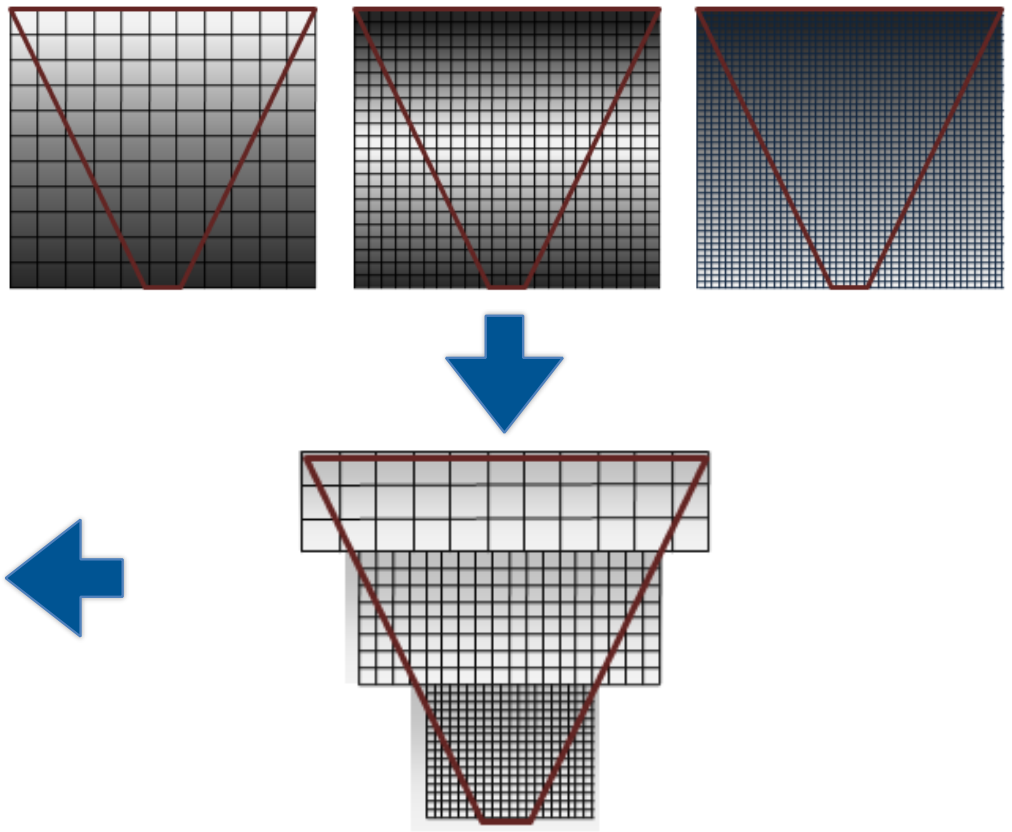
http://msdn.microsoft.com/en-us/library/windows/desktop/ee416307(v=vs.85).aspx



Cascade Shadow Maps



http://msdn.microsoft.com/en-us/library/windows/desktop/ee416307(v=vs.85).aspx



The end!