SIGGRAPH2005

Explicit Early-Z Culling and Dynamic Flow Control on Graphics Hardware

Pedro V. Sander
John Isidoro
ATI Research

Outline

- Architecture
  - Hardware Z culling
  - Hardware dynamic flow control
  - Tradeoffs of early-Z and dynamic flow control
- Applications
  - Fluid Flow
  - Shadow mapping:
    - Skin Shading with texture space shadow mapping
    - Computation masking
    - Camera-chasing shadow maps

Early-Z

- The pixel test against the z buffer is done prior to execution of the pixel shader
- Can avoid execution of pixel shader
- On current ATI chips, if the pixel shader computes and writes z then Early-Z is disabled
  - Of course, it is available again on subsequent passes

Being Clever with Early-Z

- Some applications don’t need z for hidden surface removal
  - Fluid flow simulation
  - Texture-space skin lighting
- Some other applications don’t need z after a certain point
  - Deferred shading
- Hence, you can think of z as a control code which can be used to control processing
  - You’ll sometimes see this called a *Computation Mask*

Dynamic flow control

- Efficient culling of computation with conditionals on shader
- Examples:
  - Skipping lighting computation when facing away from light
  - Fully fogged pixel
- Replaces lerp() of result of executing both branches
- DFC can have inefficiencies if different pixel pipes take different branches
- Compiler can analyze code and decide whether or not to use DFC

Tradeoffs

Early-Z

- Cost:
  - Often requires pass to populate Z buffer. Needs to store state in auxiliary buffers
- Benefit:
  - Shader execution completely culled for some pixels
- Considerations:
  - Takes advantage of hierarchical-z buffer to cull larger blocks
  - Useful for screen-space processing of a subset of pixels
  - Z-buffer not available for other computations

Tradeoffs

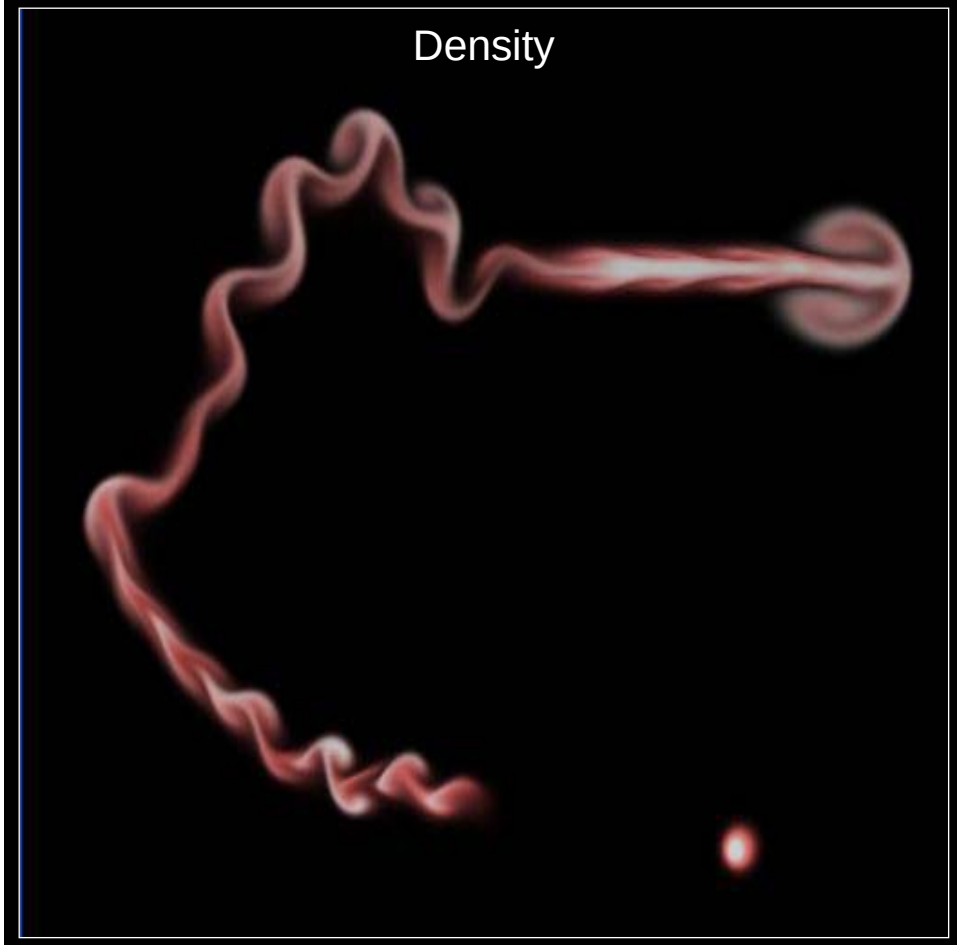
Dynamic flow control

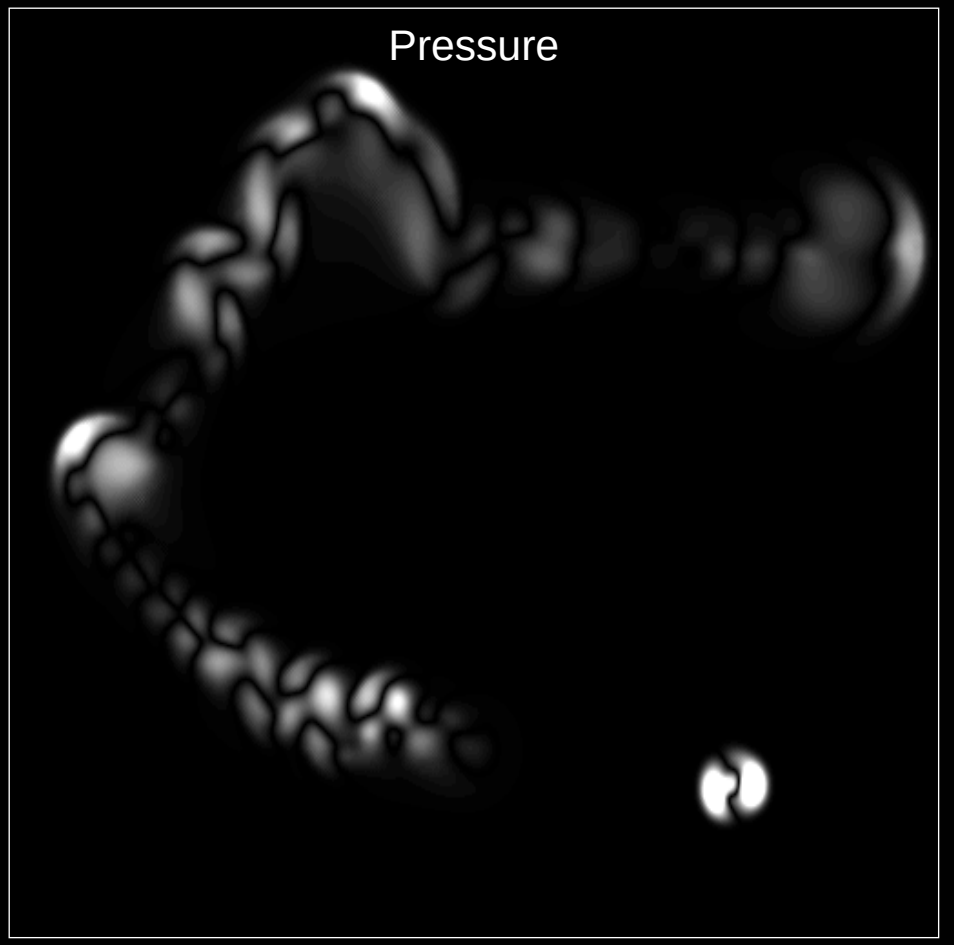
- Cost:
  - The branching instruction
  - Might execute pixel shader unnecessarily
- Benefit:
  - Certain code paths are culled
  - Efficiency depends on whether different branches are taken
- Considerations:
  - Single pass over geometry
  - Z-buffer available to store depth

Fluid Flow

- Fluid flow and other kinds of simulation are now possible on the GPU
- We can use Early-Z to cull computation in some cases, since the z buffer is not needed for traditional hidden surface removal
- Fluid flow can be sparse, making it a candidate for Early-Z optimizations
- Can reduce computation in areas of low pressure to achieve faster / better simulation results

Flow Density and Pressure

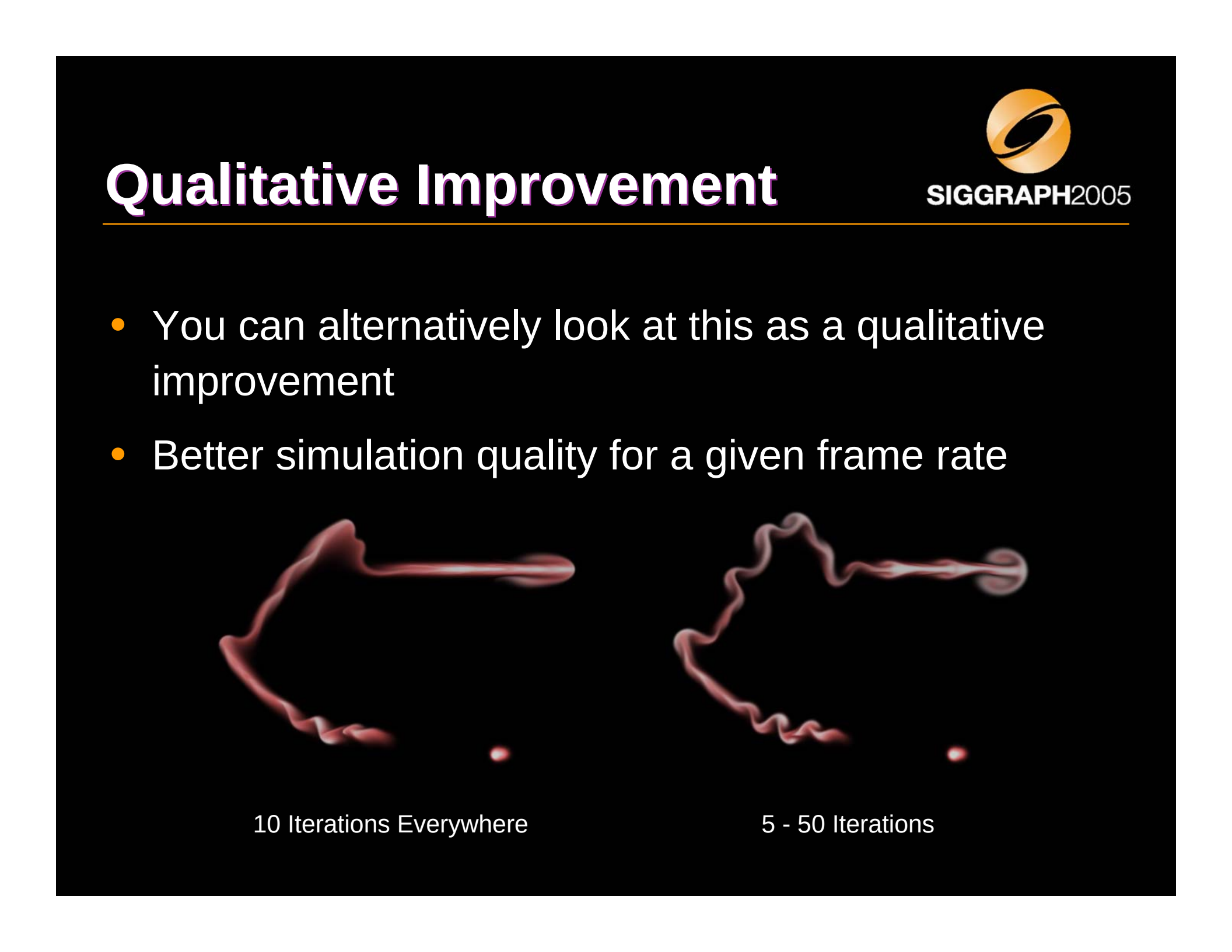




Iterations Vary With Pressure

- Assume that low-pressure regions need fewer computation iterations for convergence
- Set z buffer according to pressure buffer
- Draw 30 full screen quads in projection step
  - Vary the z from quad to quad so that the number of iterations varies with pressure
- Early-Z complete culls pixel shader instances
- Up to 3x performance improvement

Qualitative Improvement



- You can alternatively look at this as a qualitative improvement
- Better simulation quality for a given frame rate

10 Iterations Everywhere

5 - 50 Iterations

Shadows on skin

- Render diffuse lighting into an off-screen texture, unwrapping the character’s head

Blurred lighting

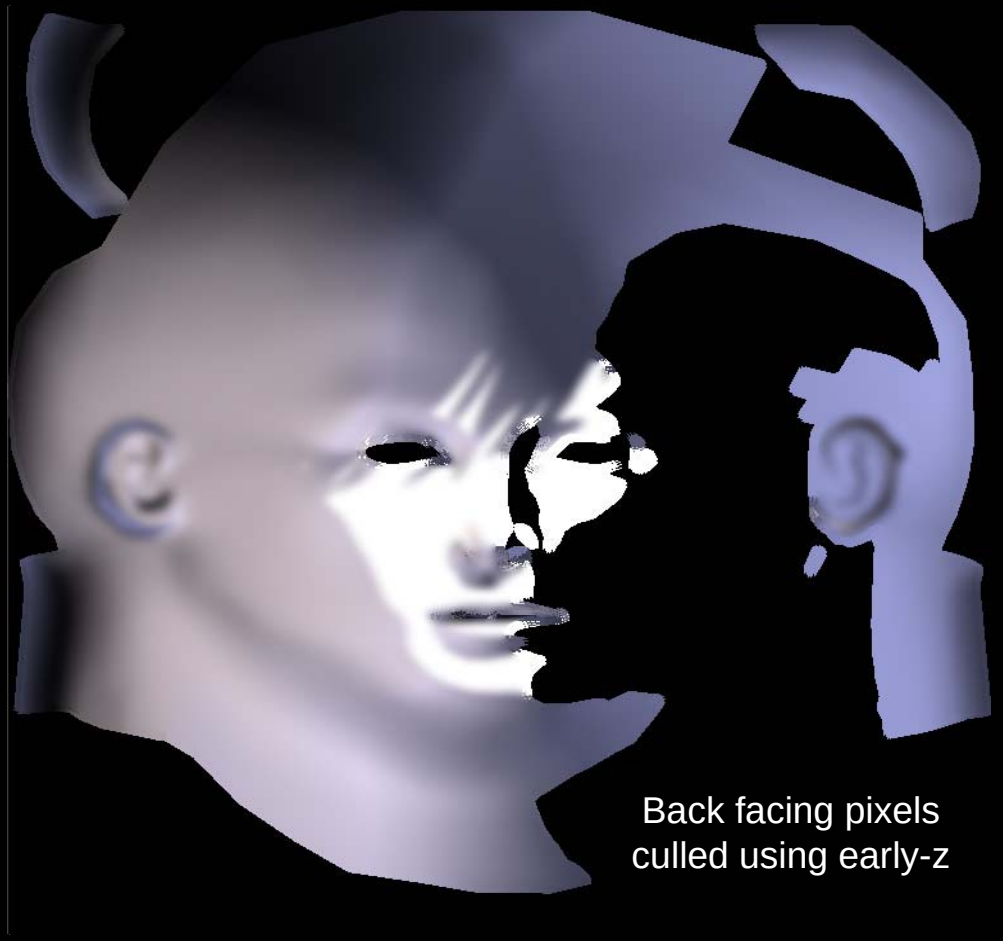
- Blur the off-screen diffuse lighting

Using Early-Z for Culling

- This texture-space operation doesn't need the z buffer for hidden surface removal
- Can use Early-Z to cull computation
  - Back face culling
  - Distance and frustum culling
- Set z buffer on lighting pass according to distance from viewer and facing-ness of polygons
- Reduces cost of image-space blurs in regions that don't need it

Back Face Culling

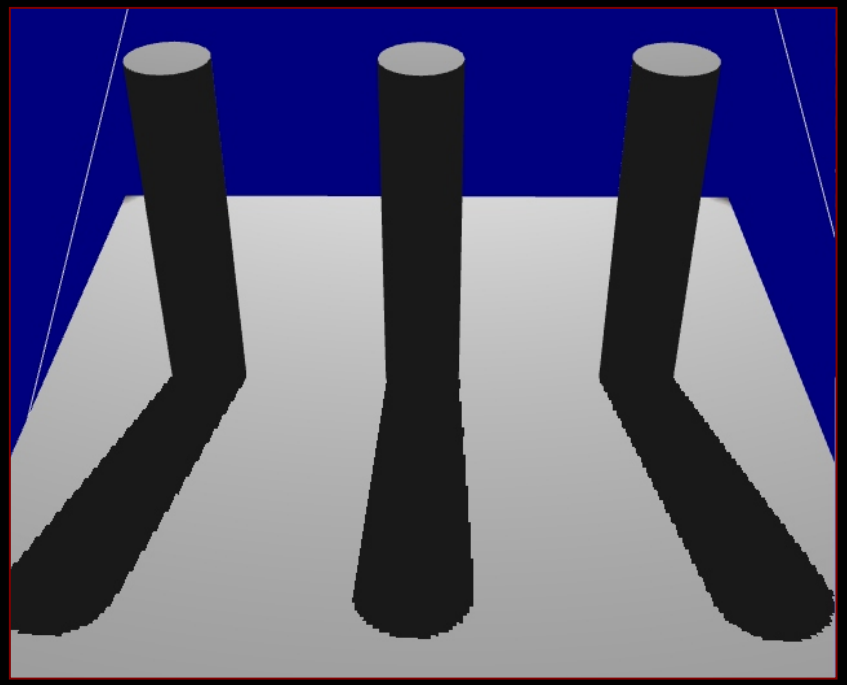
Over the shoulder view of Ruby



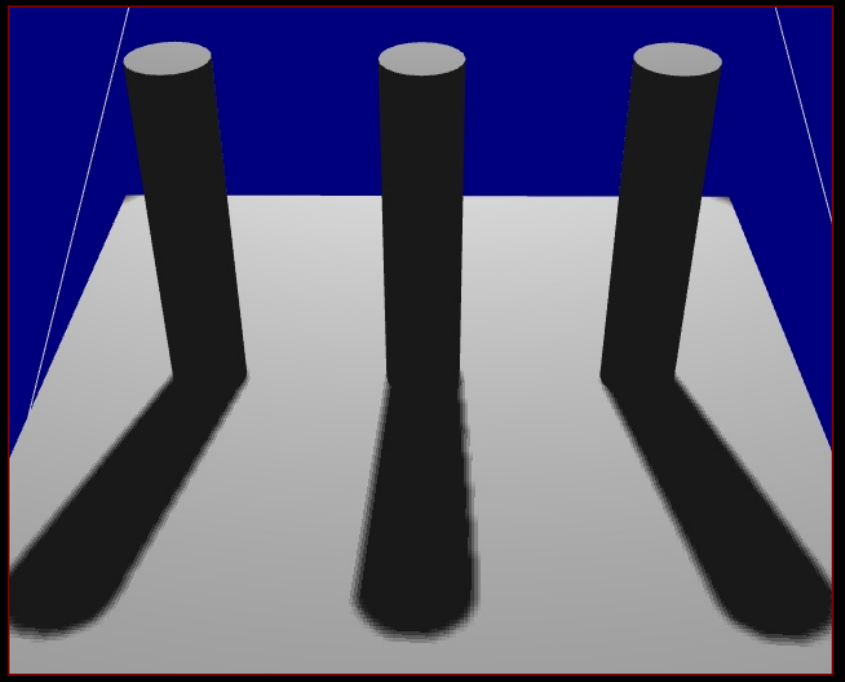
Back facing pixels
culled using early-z

Demo

Percentage Closer Filtering



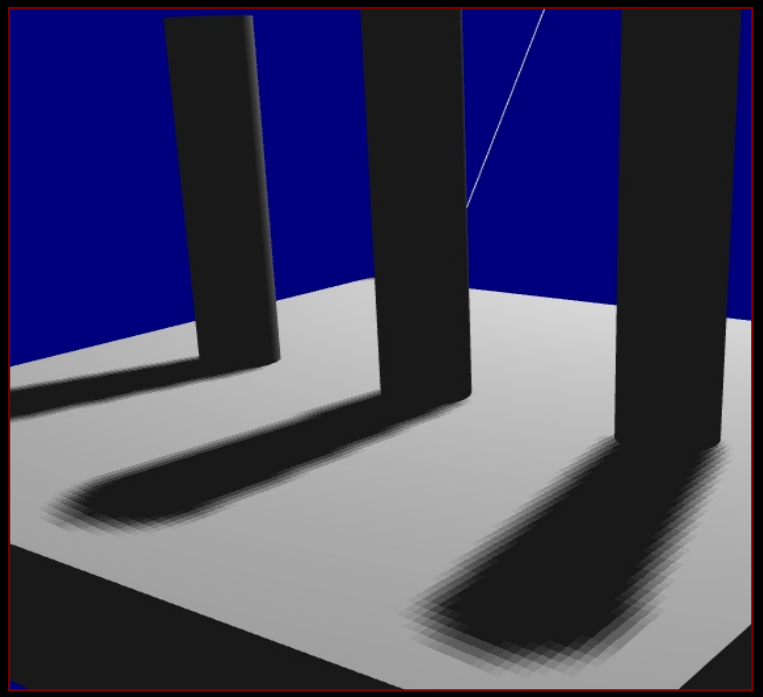
1-Tap Hard Shadow mapping



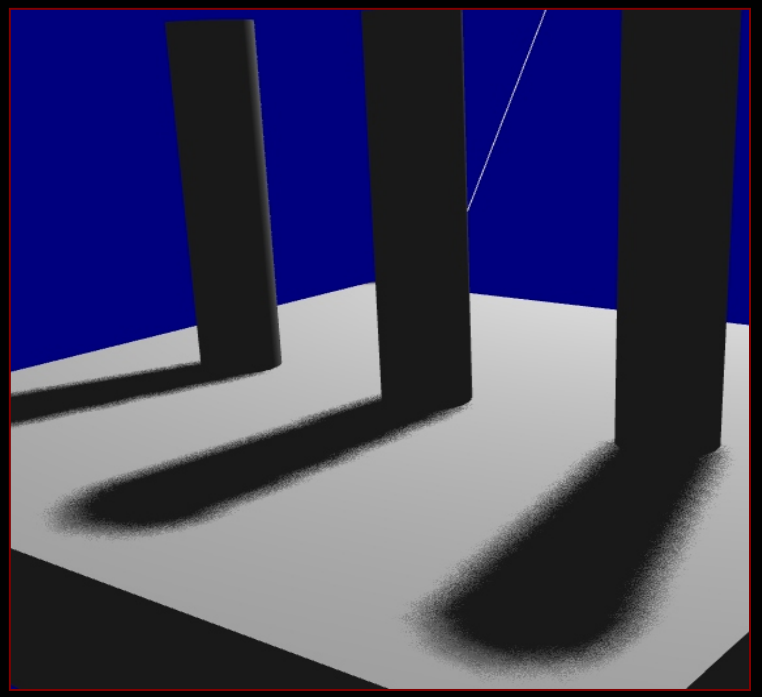
4×4 (16-tap) Percentage Closer Filtering

- Helps to alleviate aliasing problems
- Uses multiple samples from the shadow map
- First performs depth comparisons, then filters

Spatially-varying PCF Offsets



4×4 (16-tap) PCF



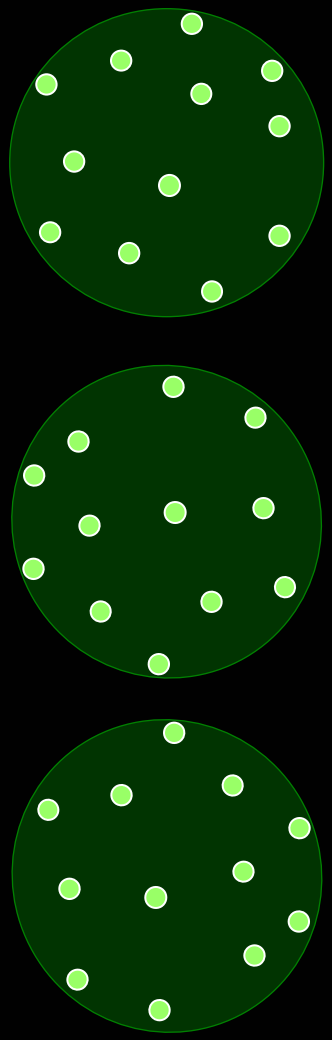
12-tap Spatially Varying PCF

- Grid-based PCF kernel needs to be fairly large to eliminate aliasing
  - Particularly in cases with small detail popping in and out of the underlying hard shadow.
- Irregular sampling allows us to get away with fewer samples
  - Error is still present, only the error is “unstructured” and thus less noticeable
  - Per-pixel spatially varying rotation of kernel is used to provide even more variation.

Spatially-Varying Rotation

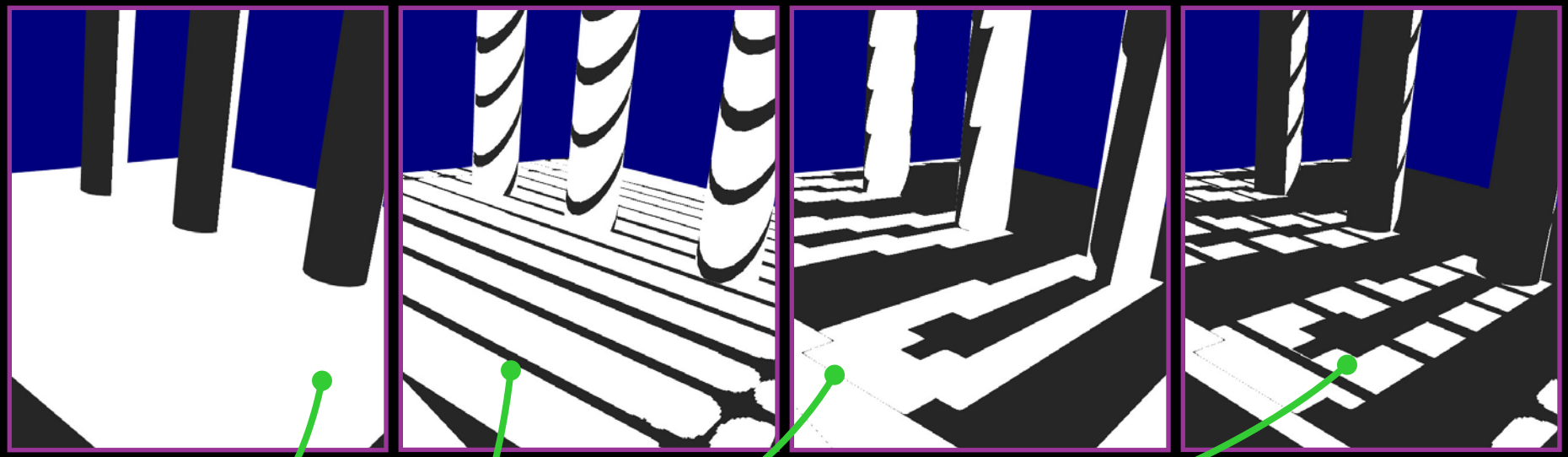

```
// Look up rotation for this pixel
float2 rot = BX2( tex2Dlod(RotSampler,
                  float4(vPos.xy * g_vTexelOffset.xy, 0, 0) ));

for(int i=0; i<12; i++) // Loop over taps
{
   // Rotate tap for this pixel location and scale relative to center
   rotOff.x =  rot.r * quadOff[i].x + rot.g * quadOff[i].y;
   rotOff.y = -rot.g * quadOff[i].x + rot.r * quadOff[i].y;
   offsetInTexels = g_fSampRadius * rotOff;

   // Sample the shadow map
   float shadowMapVal = tex2Dlod(ShadowSampler,
   float4(projCoords.xy + (g_vTexelOffset.xy * offsetInTexels.xy), 0, 0));

   // Determine whether tap is in light
   inLight = ( dist < shadowMapVal );

   // Accumulate
   percentInLight += inLight;
}
```

With unrolled loop, whole filter is roughly 120 cycles



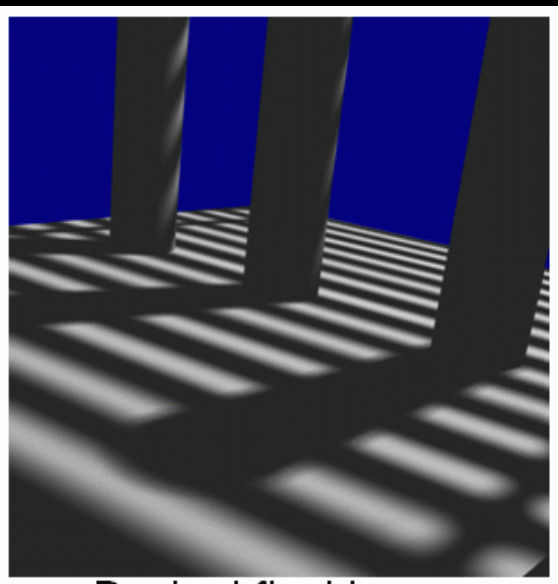
- Better quality, but still computationally expensive

Computation masking

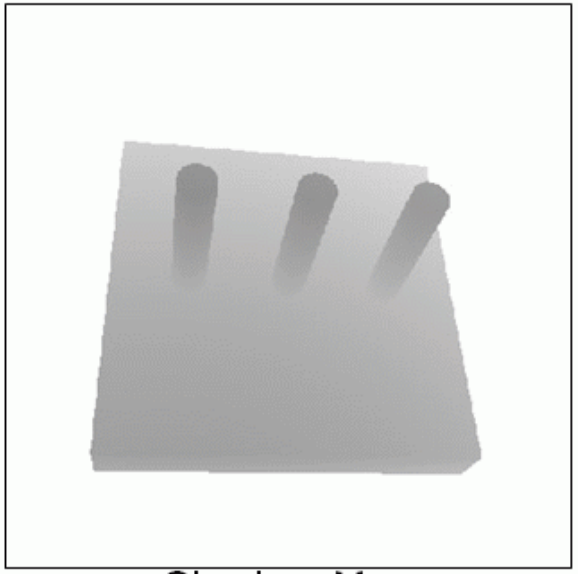


- Only do expensive filtering in areas likely to be penumbra regions using dynamic flow control. Can mask with a variety of values.
  - Backfacingness to light (no shadows on these regions)
  - Gobo (projected light pattern to mask out portions of the light source)
  - Shadow map edge filtering →

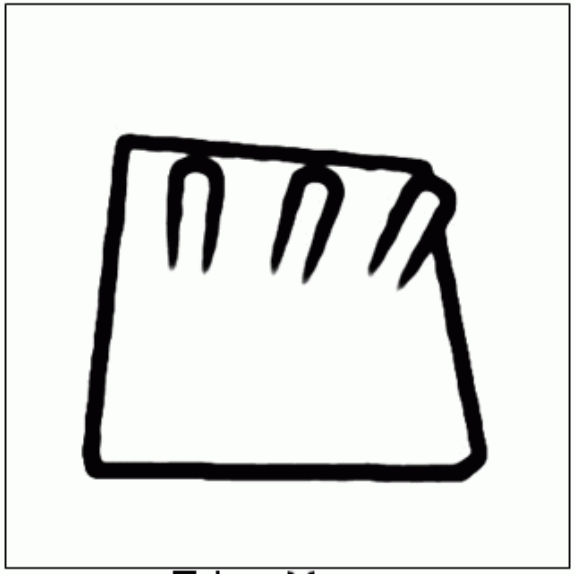
Computation masking



Desired final image



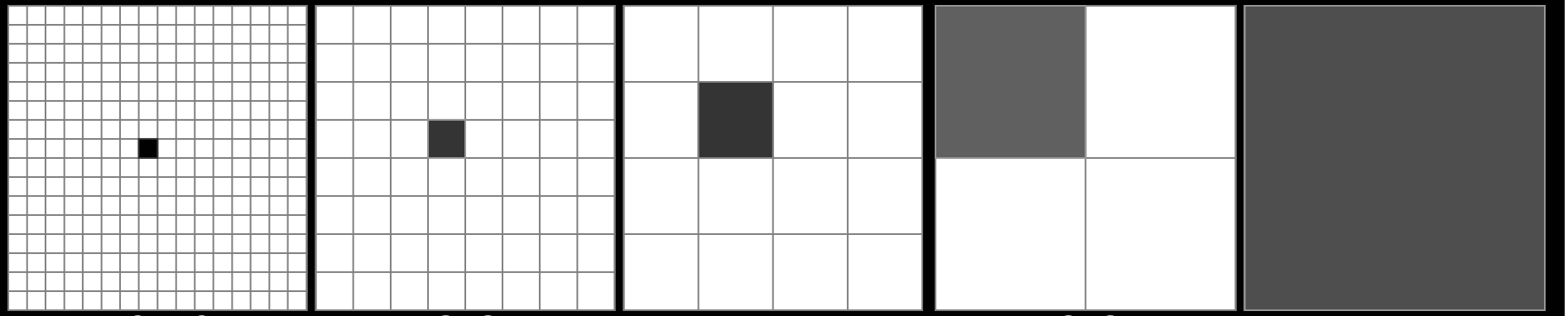
Shadow Map



Edge Map

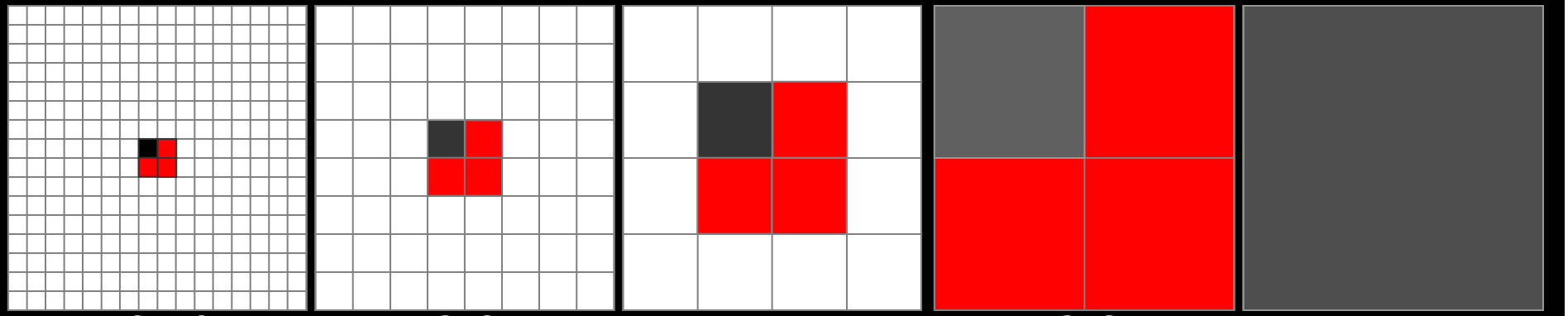
- An important observation for shadow mapping, is that the penumbra regions only exist near depth discontinuities (edges) in the shadow map.
  - Blocker/receiver ratio sufficient to cause penumbras.
  - Decide whether or not to use expensive high quality PCF (penumbral regions), or simpler single tap shadow mapping (umbral)
- Shadow map edge map must be dilated to at least the width of the filtering kernel

Fast mask expansion using mip chain dilation



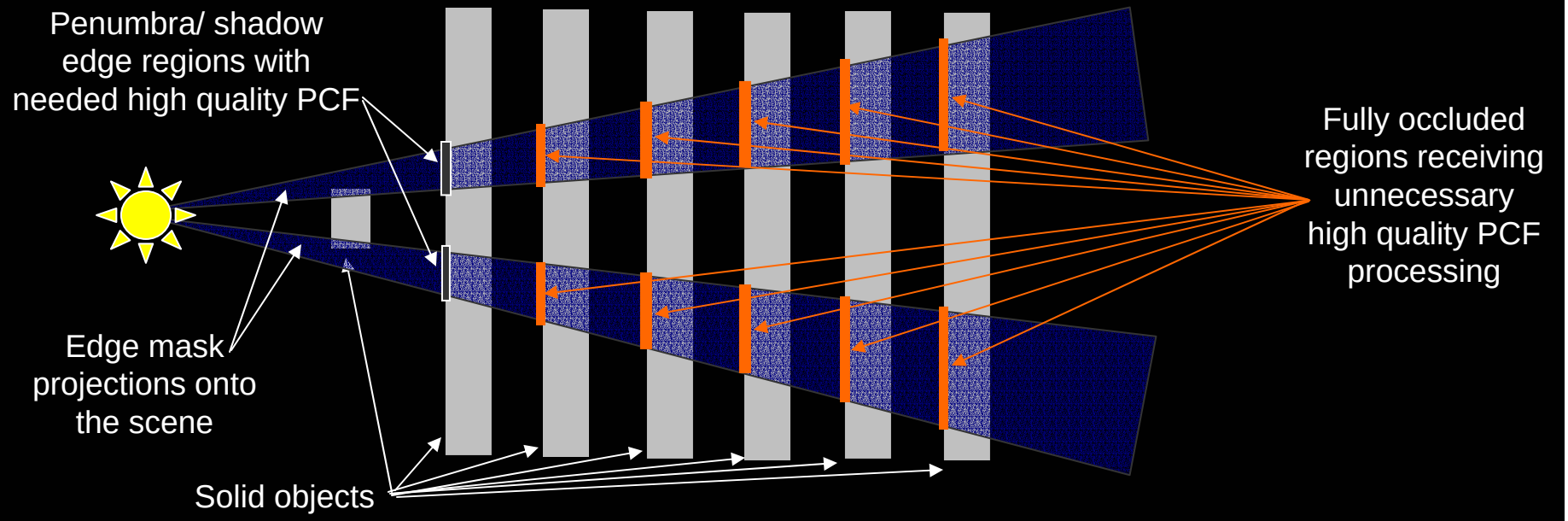
- Standard HW mipchain generation (2x2 box) is a fast way to expand the extent of a computation mask for wide kernels.
  - We call this mipchain dilation.
- However, using only point sampling the extent is not expanded equally in all directions.
  - Notice how the mask region texel does not get expanded leftward or downward until the 1x1 case.

Fast mask expansion using mip chain generation



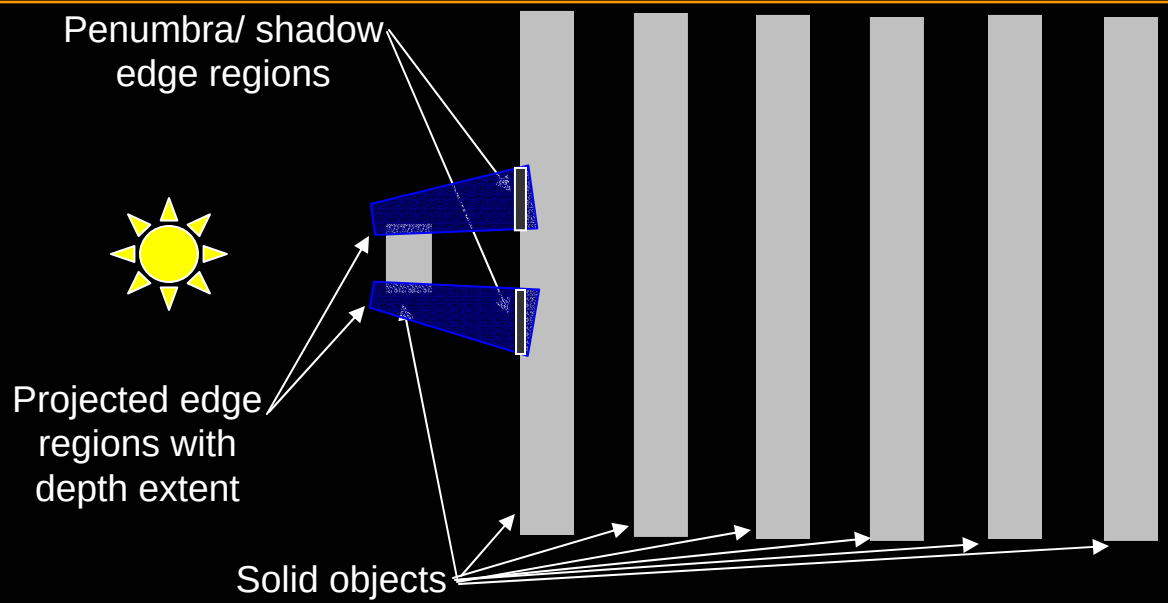
- But.. using bilinear filtering when fetching from the resulting mip levels and testing for non-zero fixes this problem!
  - The red texels cover region in texture space the bilinearly filtered texel expands out to.
- Miplevel chosen is determined by size of PCF kernel

Scene Depth Complexity



- One limitation with the edge masking approach is that the projection of the shadow map edge mask is unbounded in depth.
  - Edge masking on the shadow mask works best when the scene has a low depth complexity from the light's point of view.
  - In the case of high depth complexity, the penumbras are not present onto the third depths and further, but the regions still receive high quality PCF.

Per-Texel Depth Extent Masking



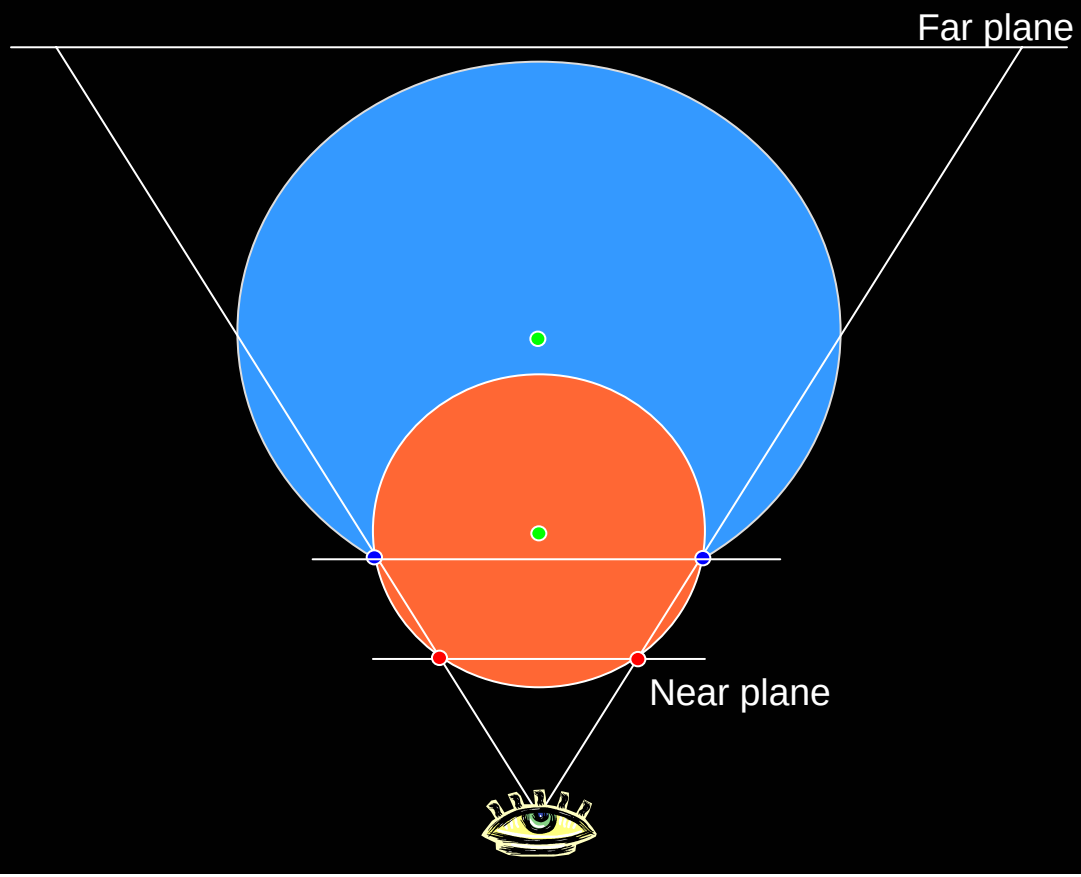
- In addition to edge masking the shadow map, compute min/max depths for the region as well if there is an edge.
- During mip-chain dilation, propagate min and max depths
  - Bilinear filtering trick doesn't work here, filtering min and max doesn't work, so use 3x3 neighborhood of texels when computing min/max mipchain.

Why we use DFC

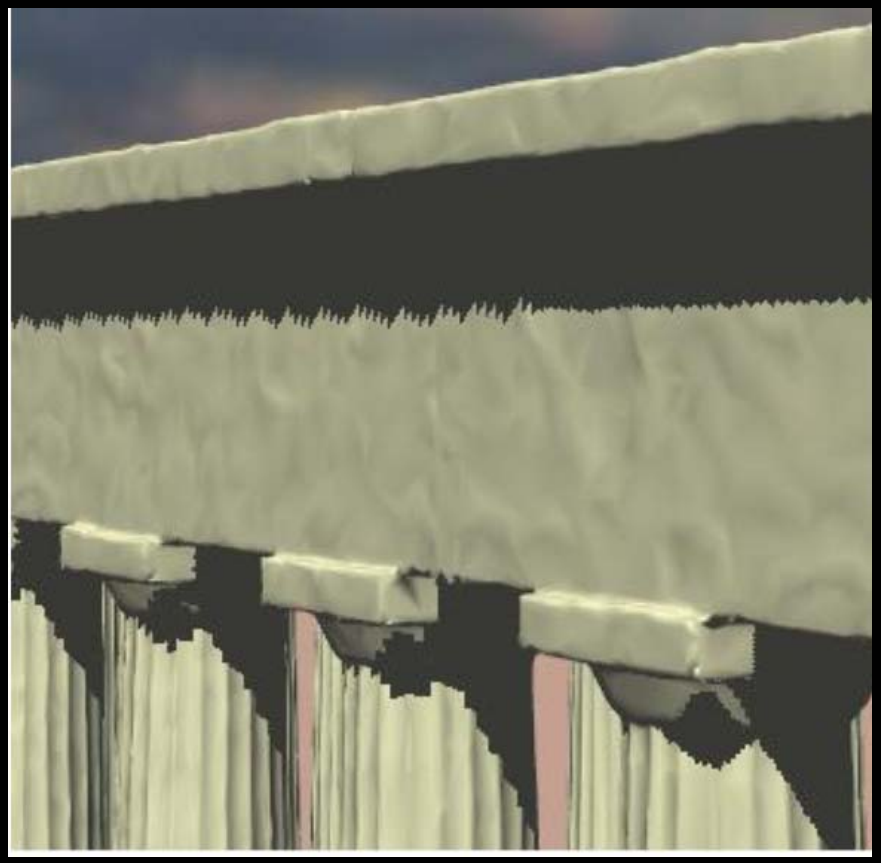
- Tough to do early-Z in this case
  - Since we are rendering the scene geometry we need the z-buffer for depth sorting.
  - For cases that don’t require depth sorting, we’d still have to render the geometry twice in the light pass to use early-z.

Camera-chasing shadow maps

- Multiple shadow maps with different projections for higher spatial sampling rate near camera



Multi-frustum example

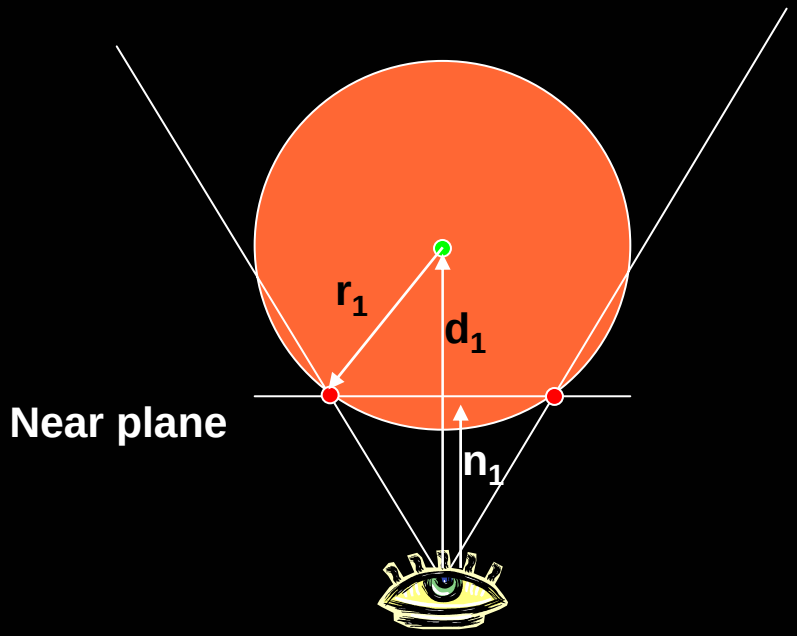


Two frusta
(both are 1024x1024)

Computing first frustum

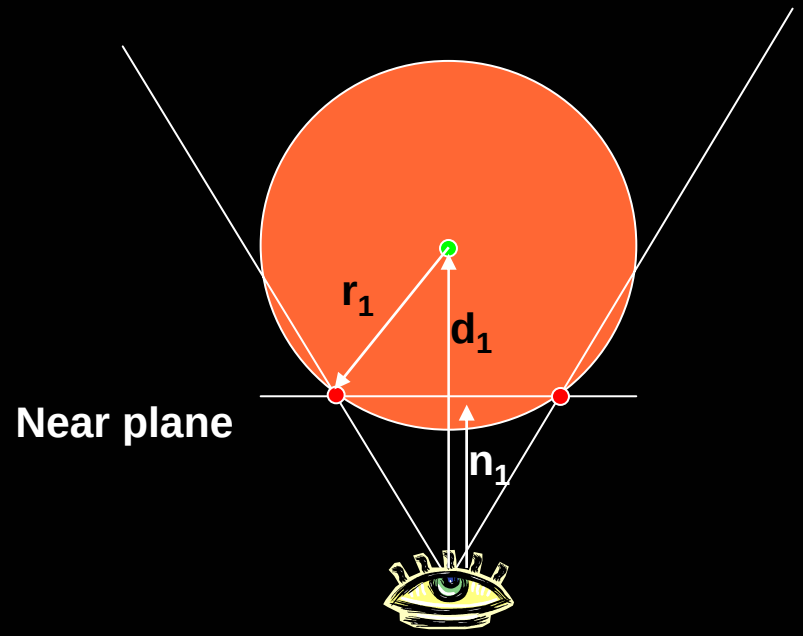
- Encapsulates sphere whose center is at distance d_1 from the camera

$$d_1 = \mathrm{sqrt}(r_1^2 - (w_1/2)^2 - (h_1/2)^2) + n_1$$



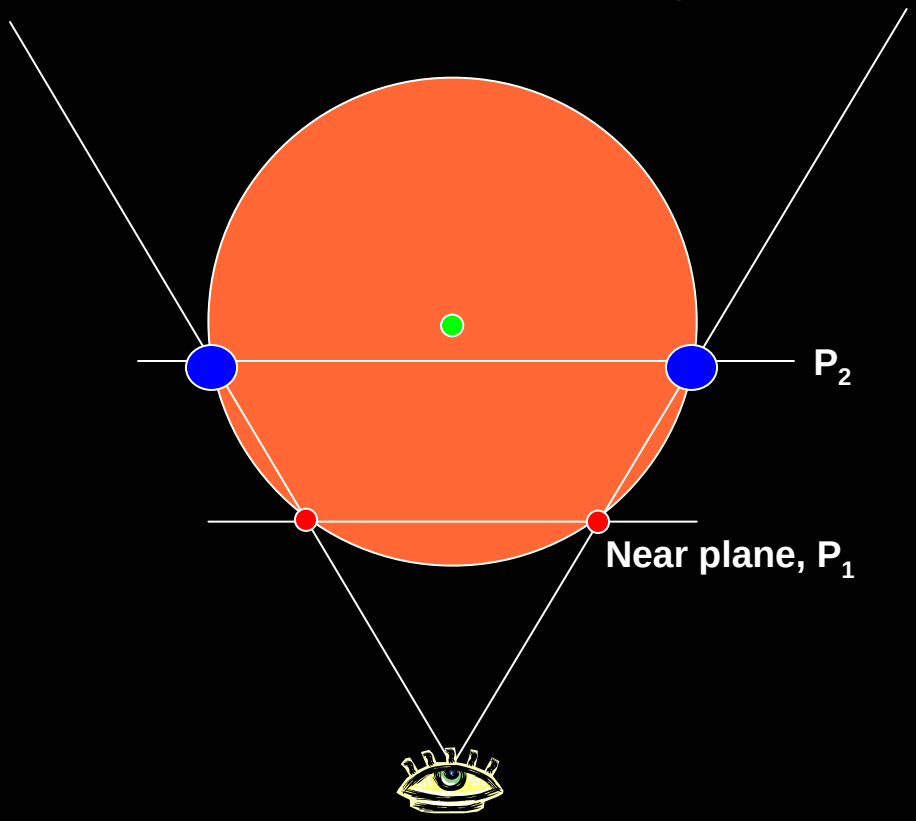
Computing first frustum

- Given d_1 and the view matrix
  - Compute the sphere center in world space
  - Generate a shadow frustum encapsulating sphere



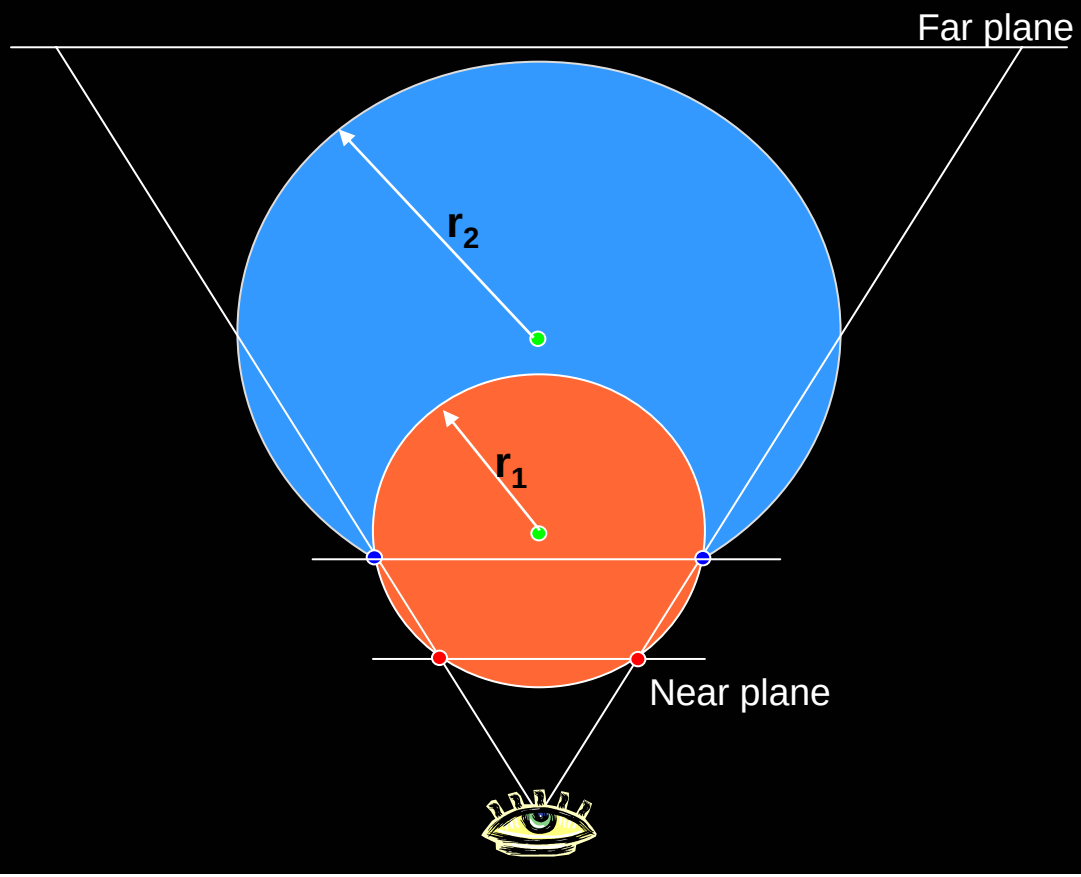
Subsequent frusta

- Next sphere should intersect P_2 at the blue points
- Radius r_2 can be chosen arbitrarily



Subsequent frusta

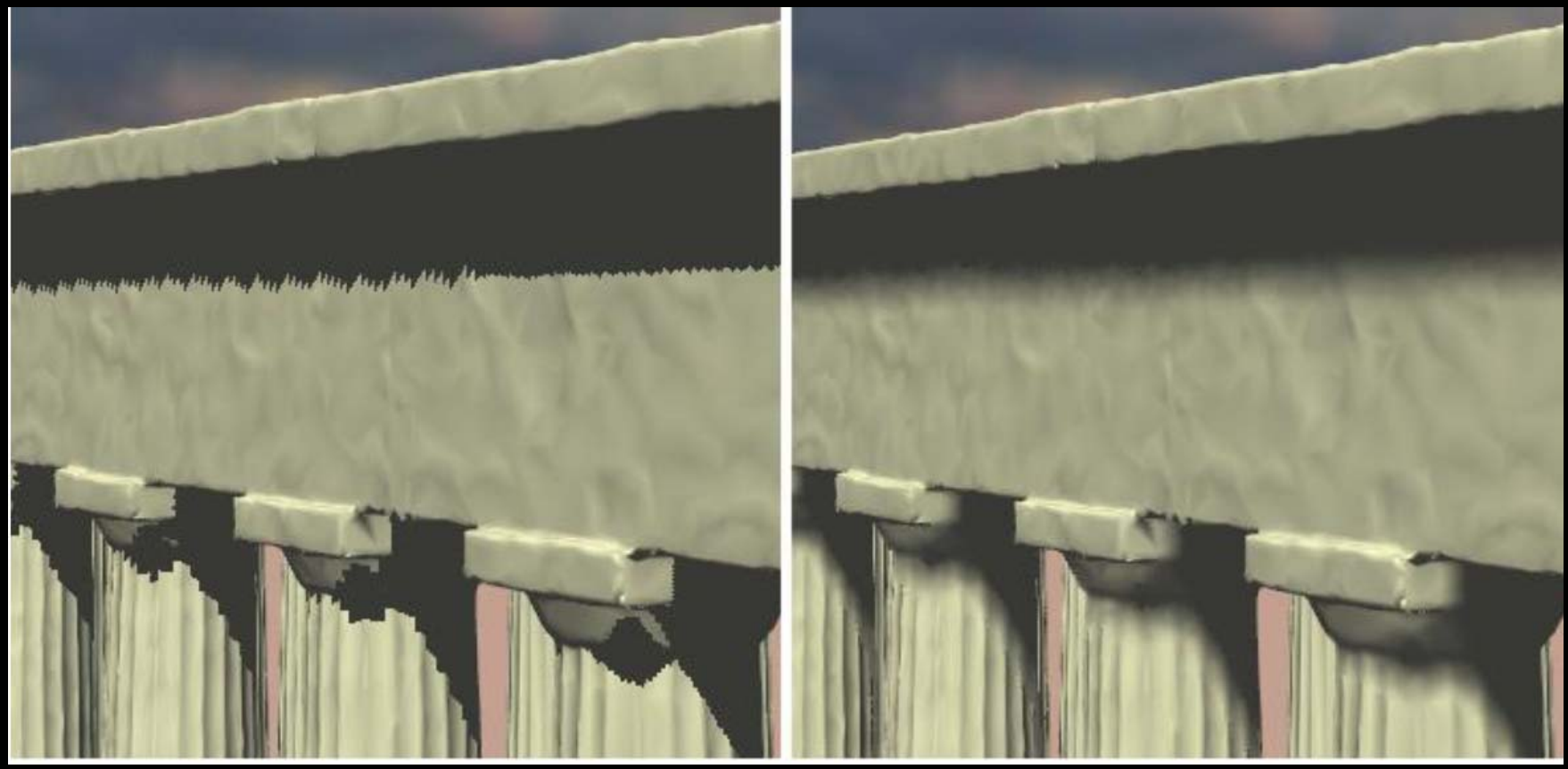
- To maintain roughly uniform sampling rate: $d_1/d_2 = r_1/r_2$
- $d_2 = f(r_2)$, so solve for r_2



Discussion

- Advantage
  - Focuses shadow map samples on regions near the camera
- Drawbacks
  - Not beneficial if light at or near camera
  - Requires generating multiple shadow maps *(experimentally, in most of the scenes we tested, two shadow frusta suffice)*

Reducing aliasing



Single tap *After AA*

Two improvements

- Multiple-tap sampling with screen-space rotation
- Post-process denoising

Sampling

- Common approach: PCF
- We do PCF with:
  - We use a disc of 12 samples
  - Rotate disc using per-pixel angle specified by image-space lookup table
- Removes shadow-map aliasing
- Introduces fine-grained unstructured noise

Denoising

- Need to remove per-pixel noise
- Post-process blur only on shadow transitions
- Keep pixels that are not on shadow transitions intact

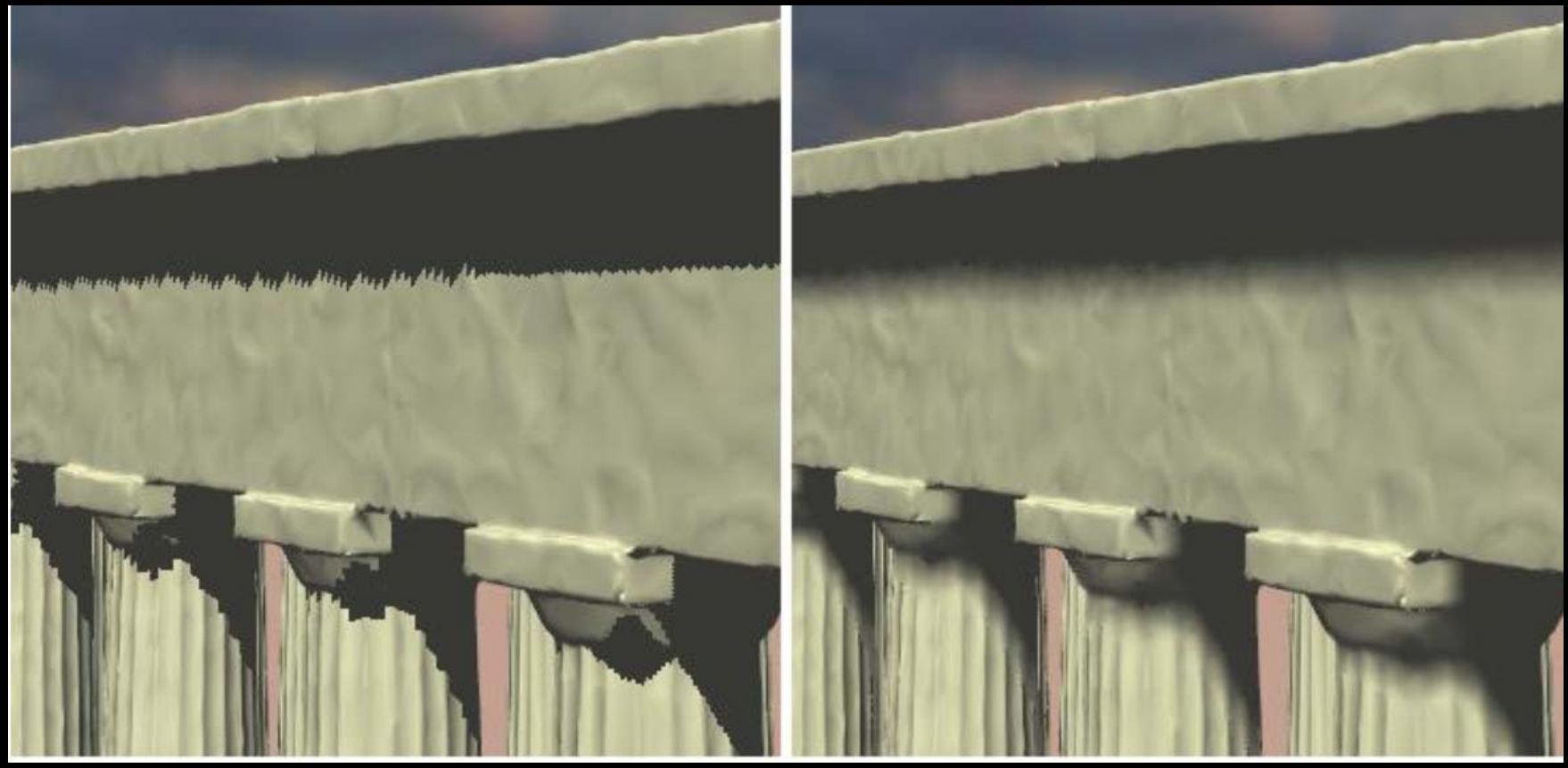
Sampling optimization

- Dynamic flow control:
  - Determine in pixel shader which shadow map to fetch from
  - Only fetch from that shadow map
- Why not Early-Z?
  - Decision wasn’t whether to execute shader, but rather which path to take
  - Avoids multiple passes on complex geometry

Denoising optimization

- Early-Z culling with two full screen quad passes:
  - Z = (lightVis == 0 || lightVis == 1) ? 1 : 0
  - Render blurring pixel shader;
    Ztest: Z == 0
- Why early-Z?
  - We want to completely cull sample
  - We don’t need Z buffer at that stage
  - Full screen quad geometry is inexpensive

Demo



Single tap *After AA*

Summary

- Architecture
  - Hardware Z culling
  - Hardware dynamic flow control
  - Tradeoffs of early-Z and dynamic flow control
- Applications
  - Fluid Flow
  - Skin Shading
  - Shadow mapping:
    - Camera-chasing shadow maps
    - Computation masking

Acknowledgements

- Jason Mitchell for discussions and some of the earlier slides
- Eli Turner for help with some of the models used in this talk
- Natasha Tatarchuk for some of the work on the flow optimization