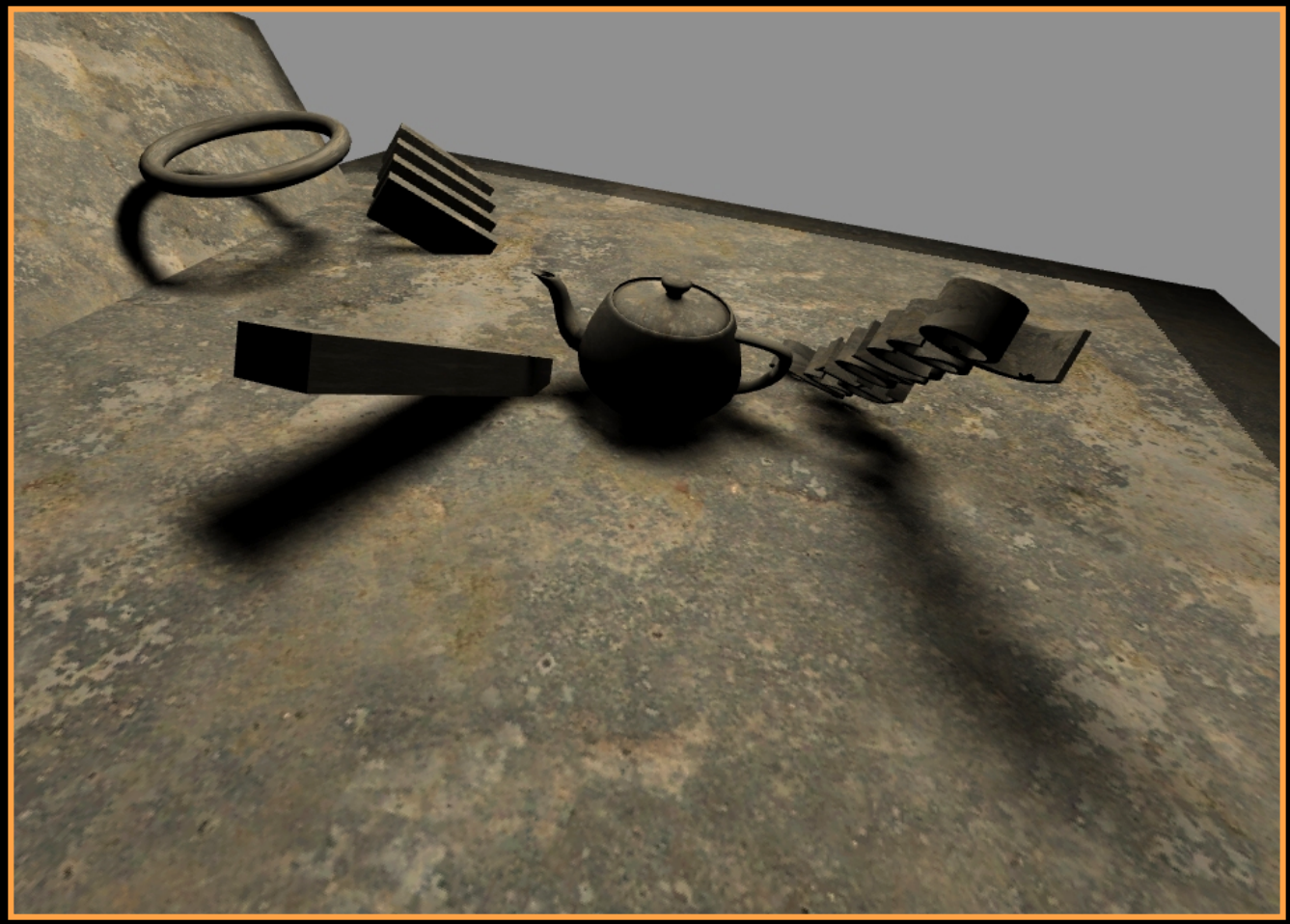
Percentage-Closer Soft Shadows

Randima (Randy) Fernando

NVIDIA Corporation

Demo



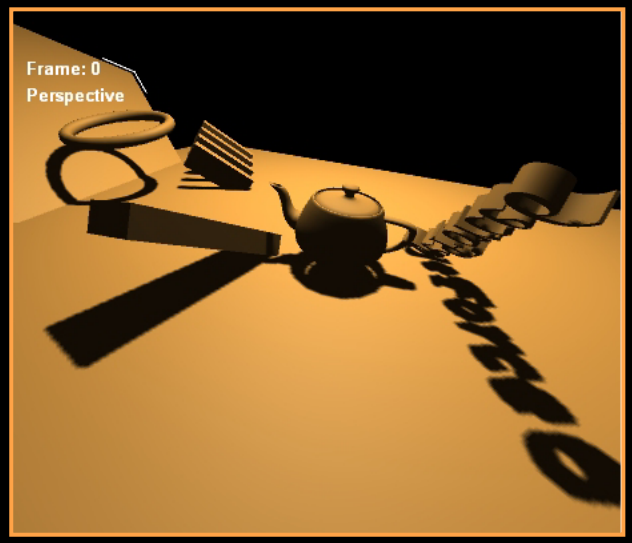
Recorded in Real Time on a GeForce 7800 GTX

Previous Work

SIGGRAPH2005

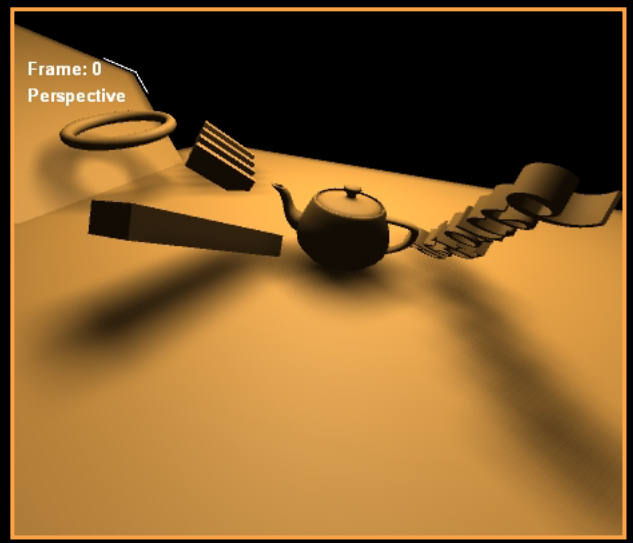
- Fundamentally based on [Williams1978] and [Reeves1987]
- Lots of previous work in this area
- Recent survey of real-time algorithms:
  - http://artis.inrialpes.fr/Publications/2003/HLHS03a/
- Related background and references in:
  - http://www.randima.com/MastersThesis.pdf

Algorithm Comparison



Regular Shadow Maps

- Always hard
- Noticeable Aliasing



Uniform Soft Shadows

- Always soft
- Aliasing is hidden

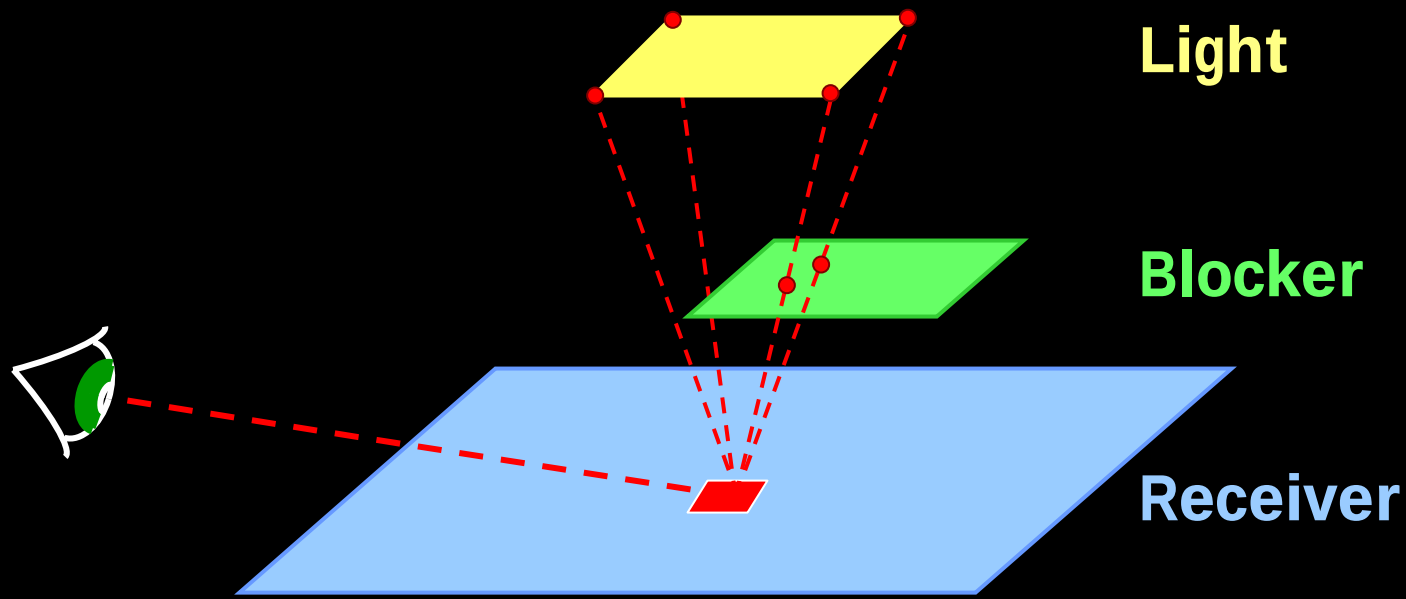
Perceptually-Correct Soft Shadows

- Shadows harden on contact
- Aliasing is hidden

Features

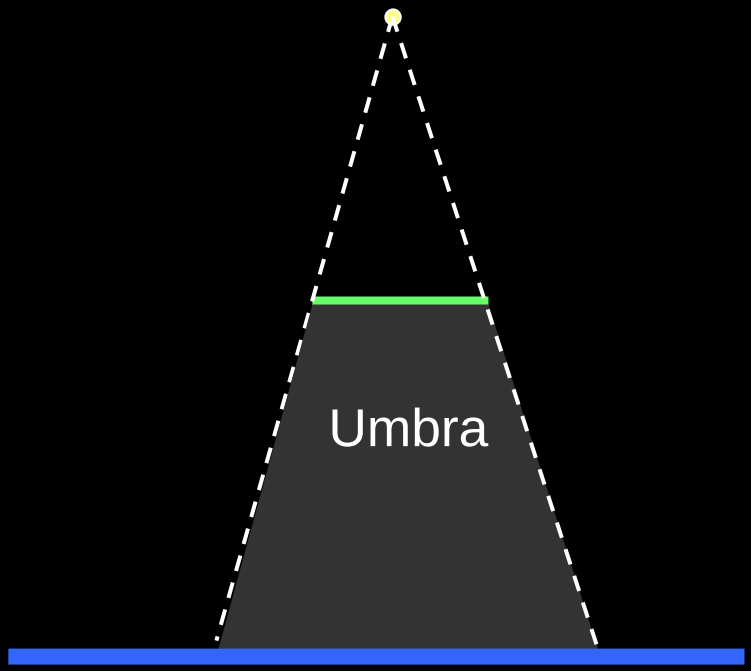
- Perceptually-correct soft shadows (good visual cues)
- Runs at real-time rates
- Artifacts vary smoothly (no popping)
- Benefits from shadow mapping features
  - Independent of geometric complexity
  - Works with alpha testing, displacement mapping, etc…
- Integrates easily
  - Single floating-point shadow map and one shader
  - No special steps, preprocessing, etc…

A Simple Scene

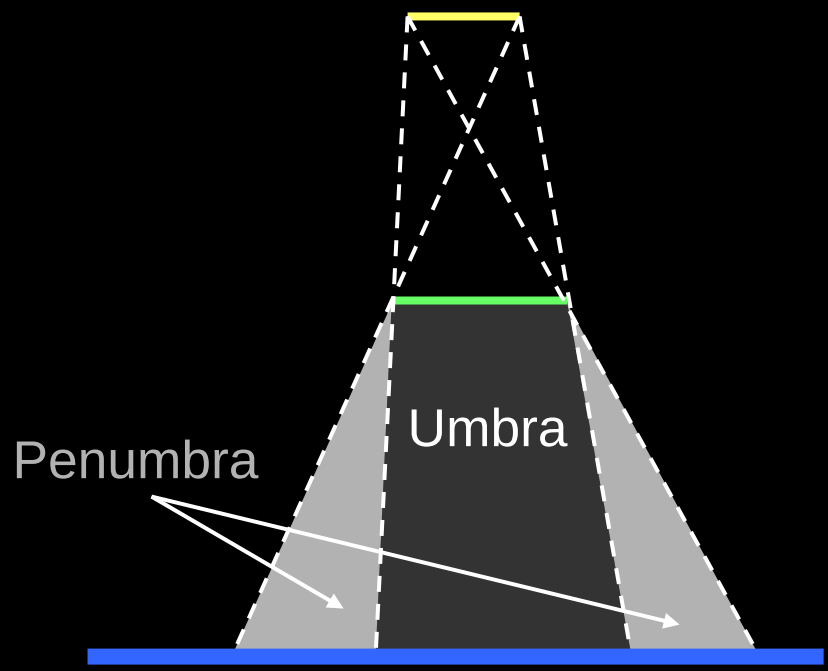


Point Light vs. Area Light

SIGGRAPH2005



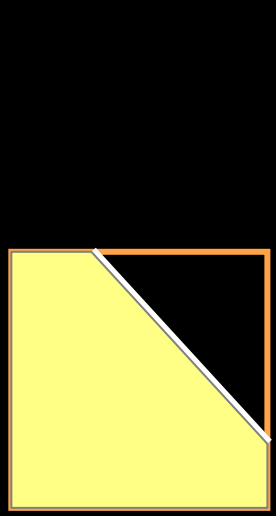
Point Light



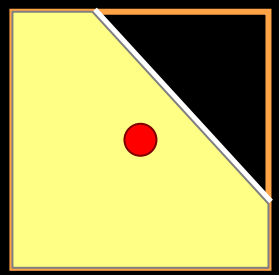
Area Light

Percentage-Closer Filtering

- Extension to shadow mapping
- How Percentage-Closer Filtering (PCF) works:

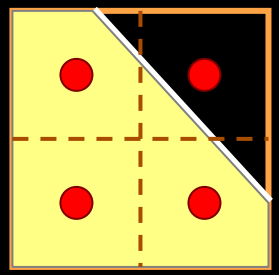


Pixel to be Shaded

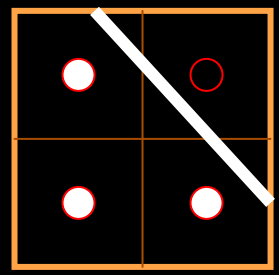


Typical Shadow Map Test

Shaded Pixel (Black or White)



4-Sample PCF



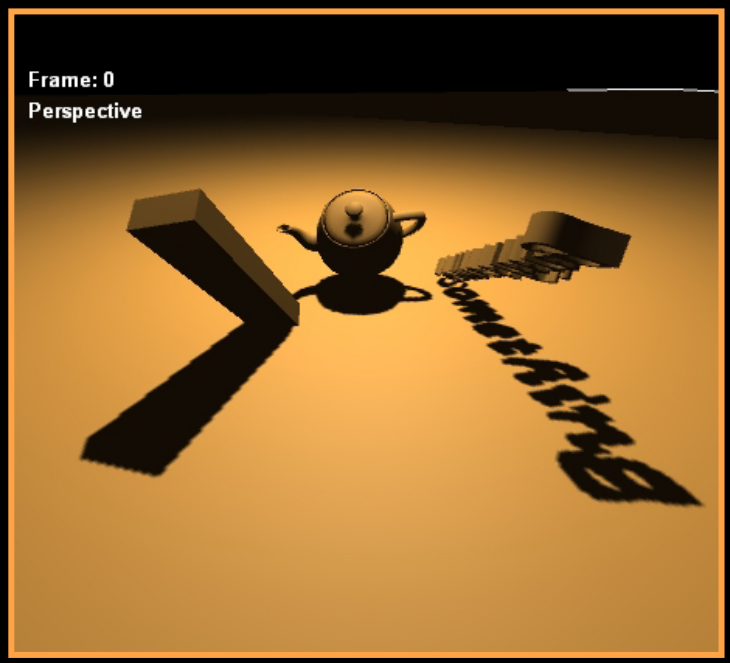
Perform 4 Depth Tests

Shaded Pixel (0, 0.25, 0.50, 0.75, 1.0)

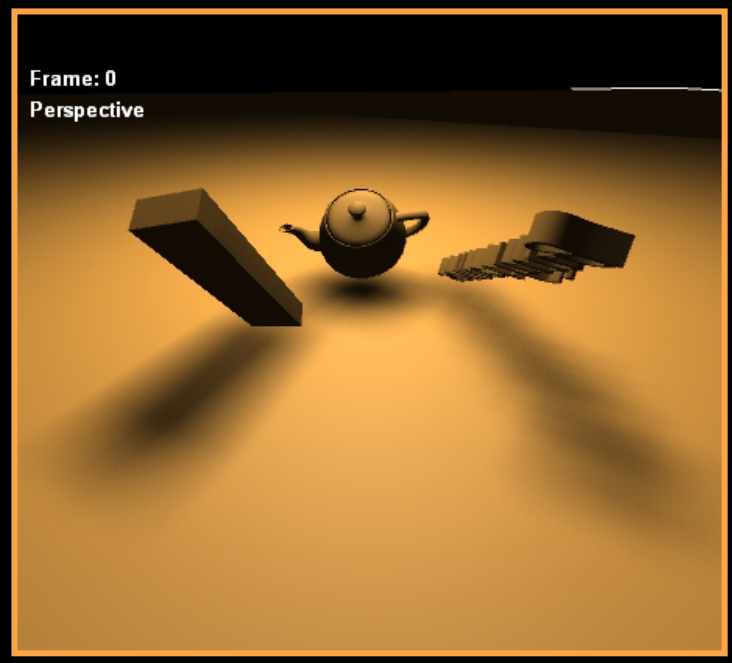
Basic Idea

SIGGRAPH2005

- Control shadow softness with PCF kernel width



Small Kernel
(Narrow Filter)

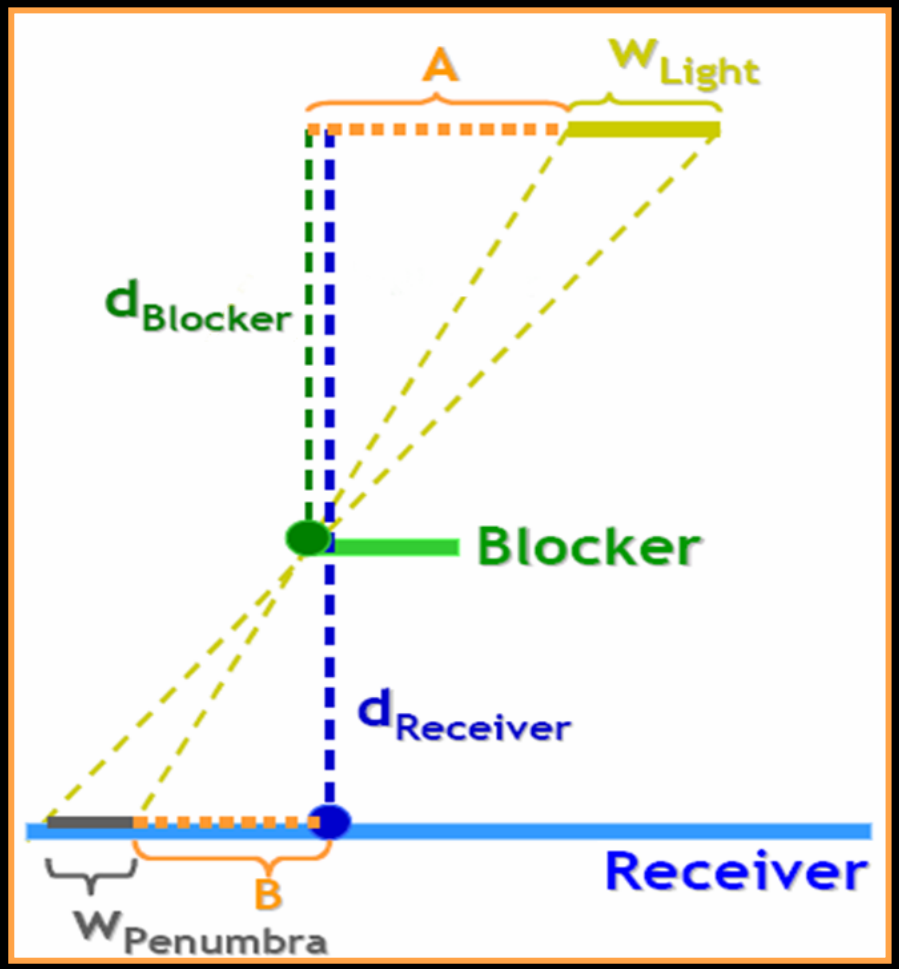


Large Kernel
(Wide Filter)

Penumbra Estimation

- Vary amount of softening
  - Based on penumbra size
- Penumbra size estimate based on:
  - Blocker depth
  - Receiver depth
  - Light size

Penumbra Size Estimation



$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

- Equation comes from similar triangles
- Assumes that blocker, receiver, and light are parallel

Penumbra Size Estimation

$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

- We need:
  - Distance from receiver to light source
    - ✓ Depth of the point we're shading
  - Light size
    - ✓ Uniform input to the shader
  - Distance from blocker to light source
    - × **Don't know this… yet.**

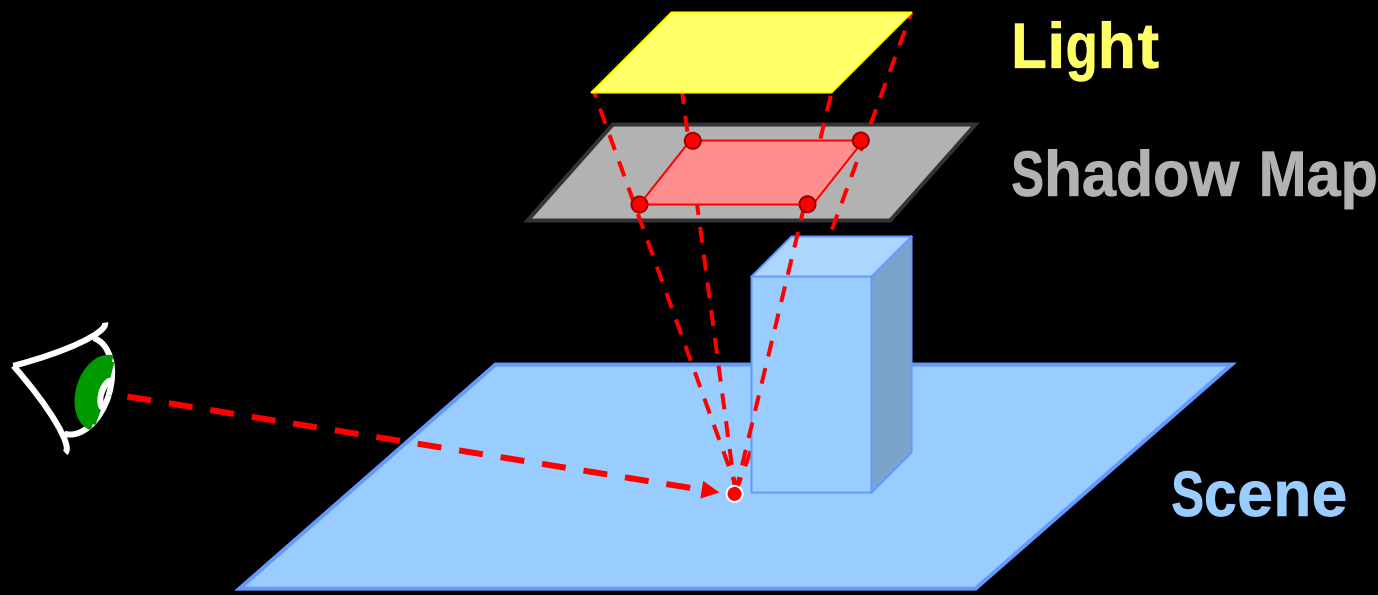
Main Algorithm

- Generate a shadow map from center of light source (as usual)

- When shading each pixel on the screen:
  - Blocker Search
  - Penumbra Size Estimation
  - Variable Percentage-Closer Filtering

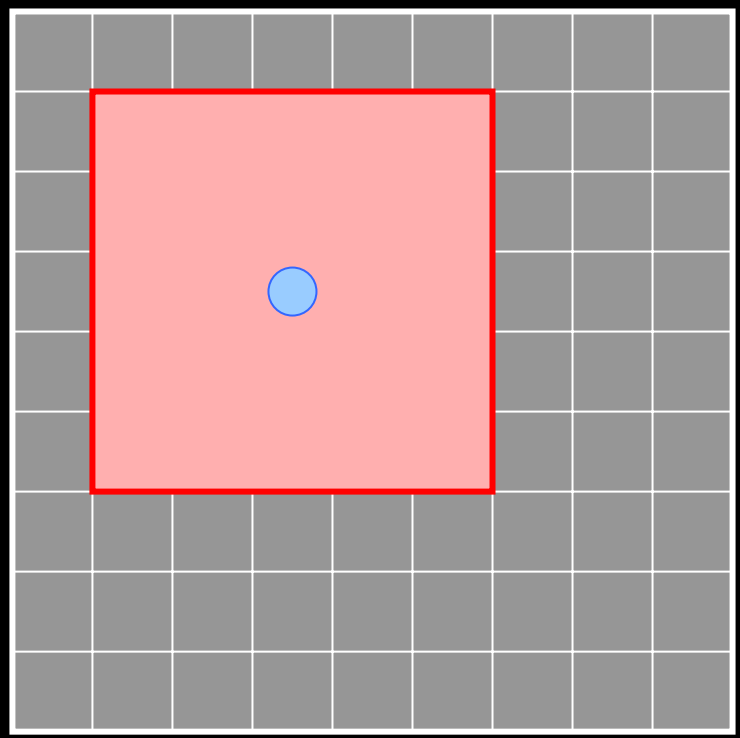
Blocker Search

- **Search region** depends on light size and distance to light



Blocker Search

- Sample the texels in the **search region**
- Do something with the depth values…



Shadow Map

What to do with Blockers?

- Take minimum?
  - Artifacts when transitioning between blockers
- Need some kind of average
  - Average all blockers (depth < receiver)
  - Flag the case if no blockers were found
    - Fully lit – no need to perform filtering
  - Gives good results

Penumbra Size Estimation

- We now have what we need:
  - Distance from receiver to light source
    - ✓ Depth of the point we're shading
  - Light size
    - ✓ Uniform input to the shader
  - Distance from blocker to light source
    - ✓ Result of blocker search
- Estimate penumbra per pixel:

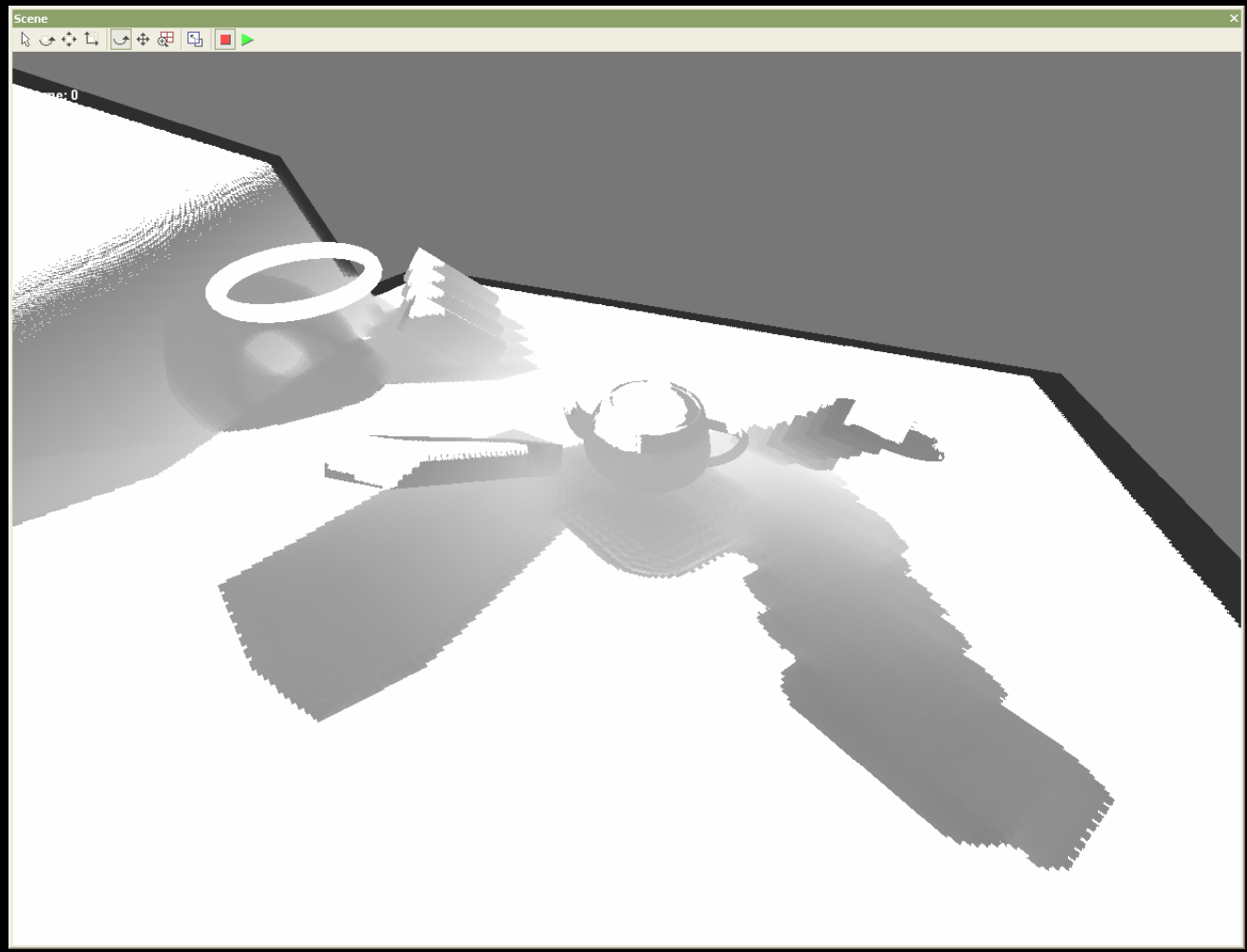
$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

- Change PCF kernel based on the result

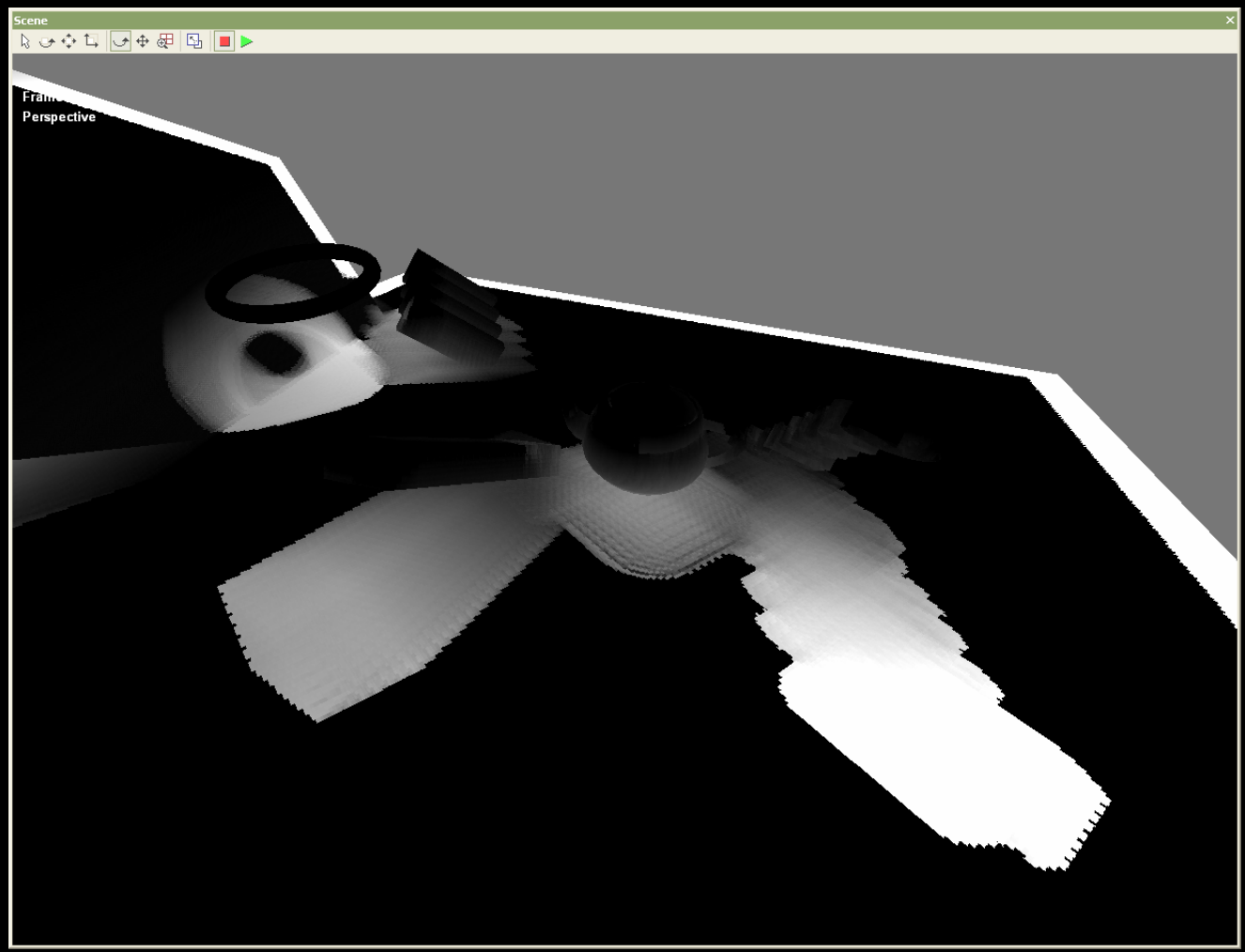
Variable PCF

- Use a flexible PCF kernel that can vary:
  - Filter width
  - Number of samples
- Vary kernel parameters based on penumbra estimate
  - Actually, projection of penumbra in screen space (but not yet implemented)

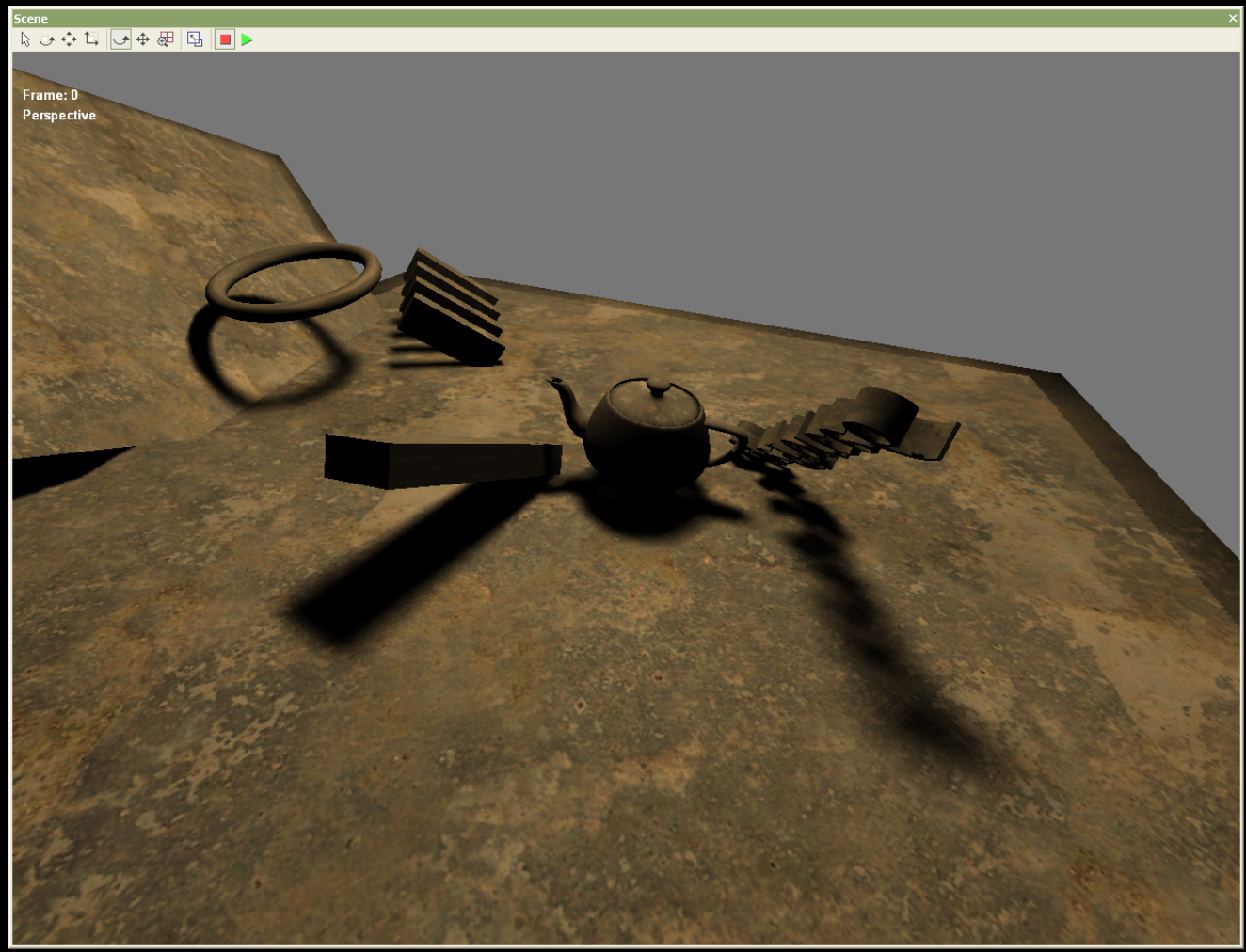
Blocker Search Results



Penumbra Estimates



After Percentage-Closer Filtering



Performance

- Up to **191 fps @ 640 x 480** in a standalone application
- **<10%** hit to add 4xAA, 8xAF on GeForce 7800 GTX

# SAMPLES						FPS	6800	7800 GTX	7800 GTX SLI
Search		PCF		Total			12 pipes	24 pipes	48 pipes
36	+	36	=	72			49	104	191
36	+	64	=	100			35	75	141
36	+	256	=	292			13	31	60

- **292 texture fetches/pixel = ~900 cycles/pixel = 60 fps!**

Performance Characteristics

- Scales very well with number of pipes
- Each blocker search sample is ~30% more expensive than each PCF sample
- 36 search samples works well for most scenes
- 64 PCF samples works great with a reasonably complex texture

Performance Improvements

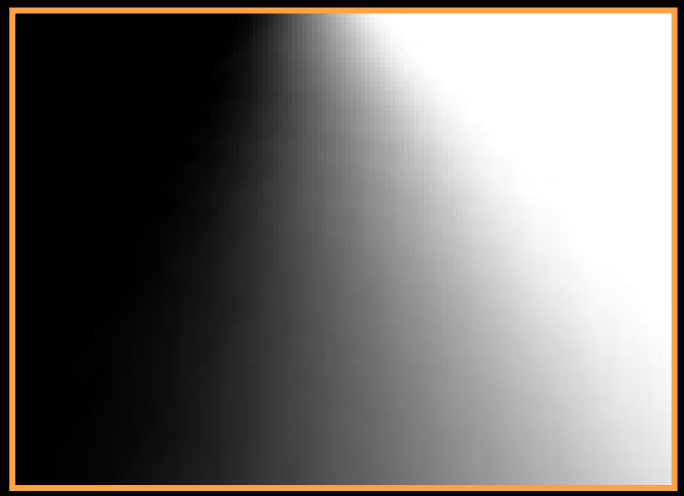
- Mask out umbras and fully-lit regions
  - Pass 1: Estimate penumbras
  - Pass 2: Perform PCF only where penumbra exists
- Vectorize
  - Much of the code is very scalar
  - Can try to work on 4 texels simultaneously
- Profiling/tuning

Quality Improvements

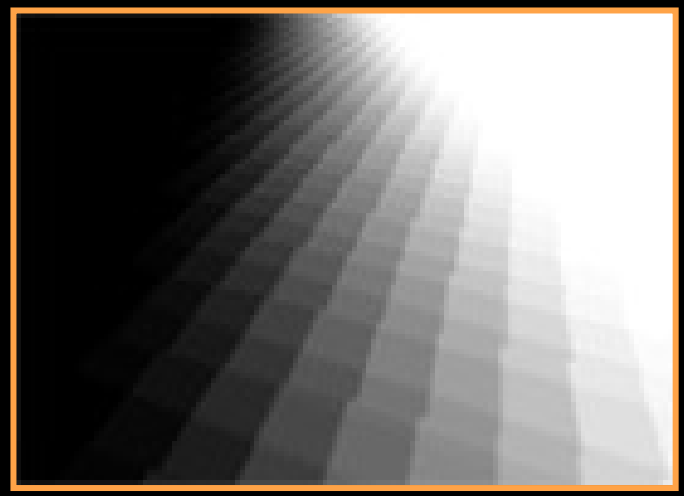
- Better blocker search heuristics
  - 5 shadow maps (4 corners + center of light)
  - Estimate penumbra using all five
  - Filter using center shadow map only
- Better filtering to remove banding in large penumbras
  - Add offset jitter for samples
  - Screen-space blur to save on PCF work
  - Combine filtering from several resolutions of shadow maps?

Quality Improvements

- Take advantage of NVIDIA's hardware PCF
  - 32-bit floating-point shadow map for analysis
  - 24-bit integer shadow map for filtering
  - Small performance hit for large quality improvement



With HW Filtering



Without HW Filtering

Parting Thoughts

- Algorithm is completely encapsulated in one shader file for easy integration
- Try it out – please let me know what you find
- Tweak "**Near Plane Distance**" (in world units) and "**Shadow Map Bias**" to match your scene
- Applications: DCC/CAD applications, pre-visualization, future games

Thanks to…

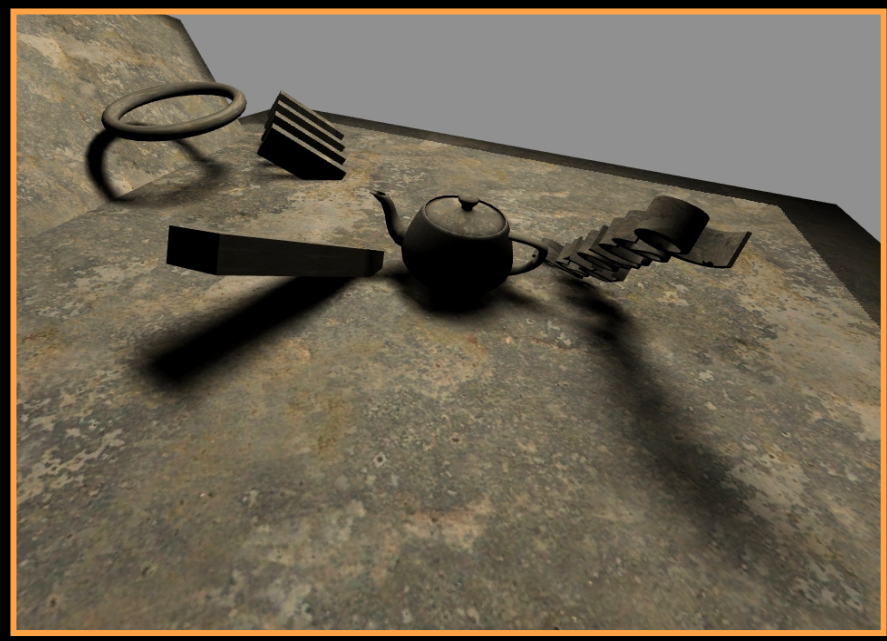
- Kevin Bjorke, whose basic PCF shader I started with
- Chris Maughan and the FX Composer team for making shader development so easy
- Various folks at NVIDIA, and especially Matthias Wloka, for comments and suggestions

Suggestions/Questions Welcome

- Slides, short whitepaper, code, and video for this FX Composer effect are in the NVIDIA SDK: http://download.developer.nvidia.com/developer/SDK/Individual_Samples/featured_effects.html#PCSS

- A standalone code sample is coming soon

- E-mail: RFernando@nvidia.com



Percentage-Closer Soft Shadows

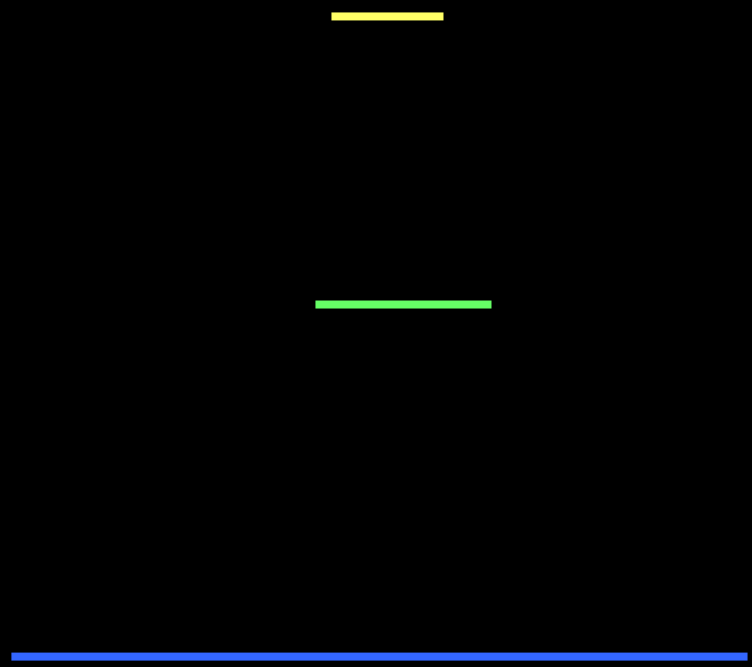
Randima (Randy) Fernando

NVIDIA Corporation

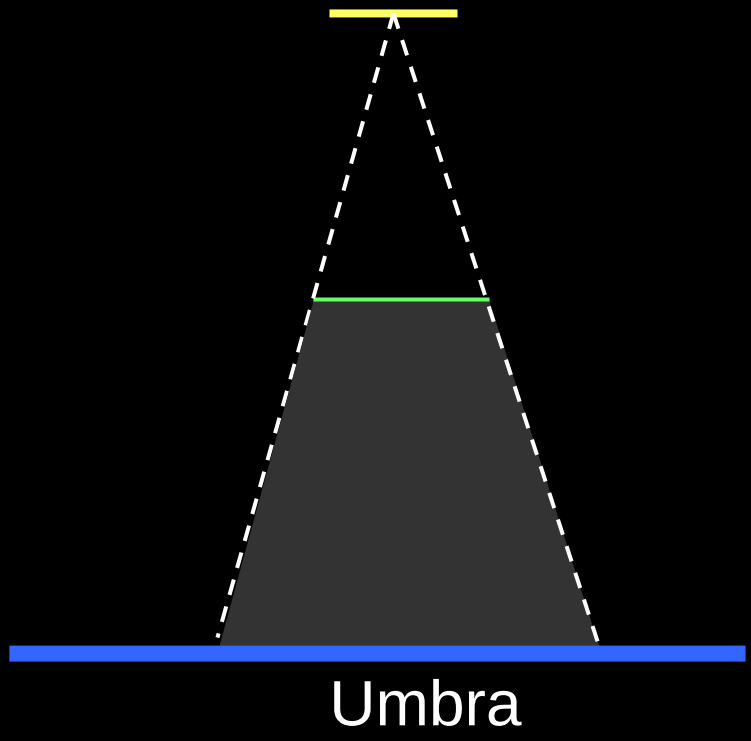
Extras

SIGGRAPH2005

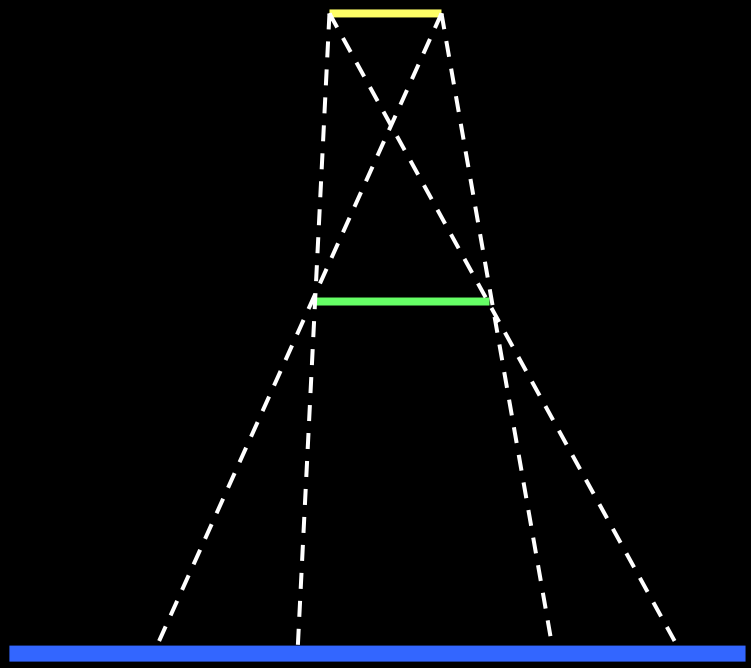
A Simple Scene



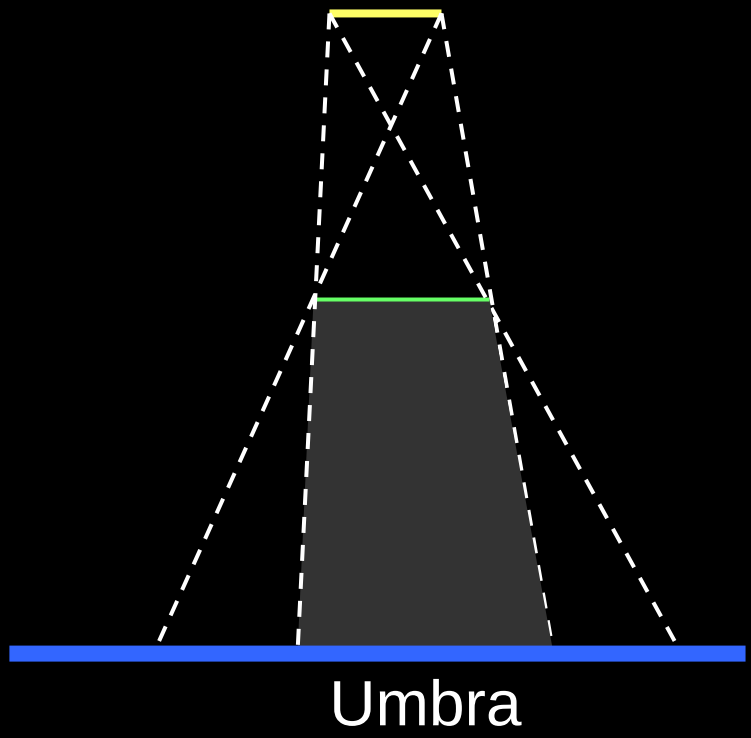
Shadows from a Point Light



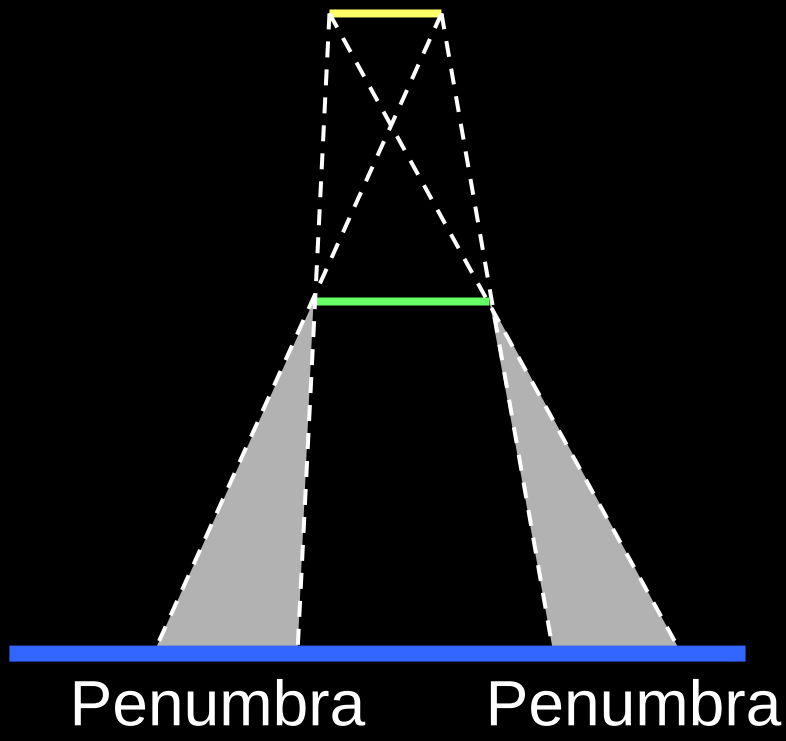
Shadows from an Area Light



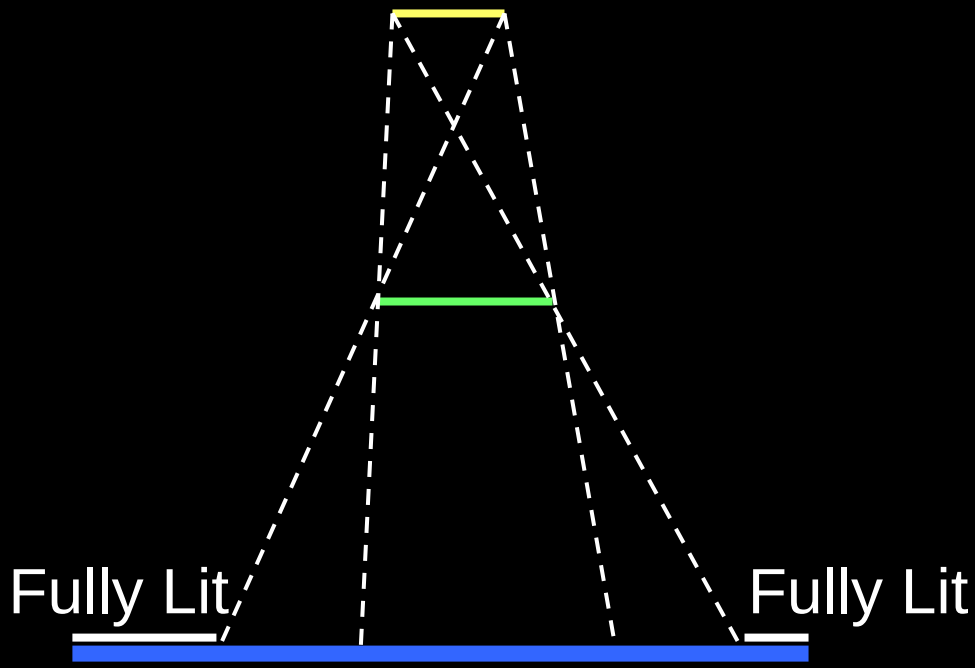
Shadows from an Area Light



Shadows from an Area Light

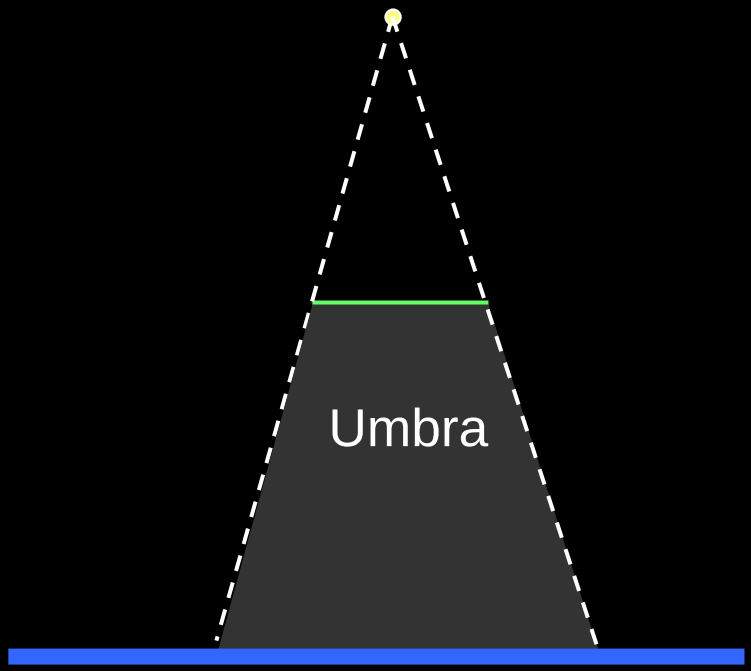


Shadows from an Area Light

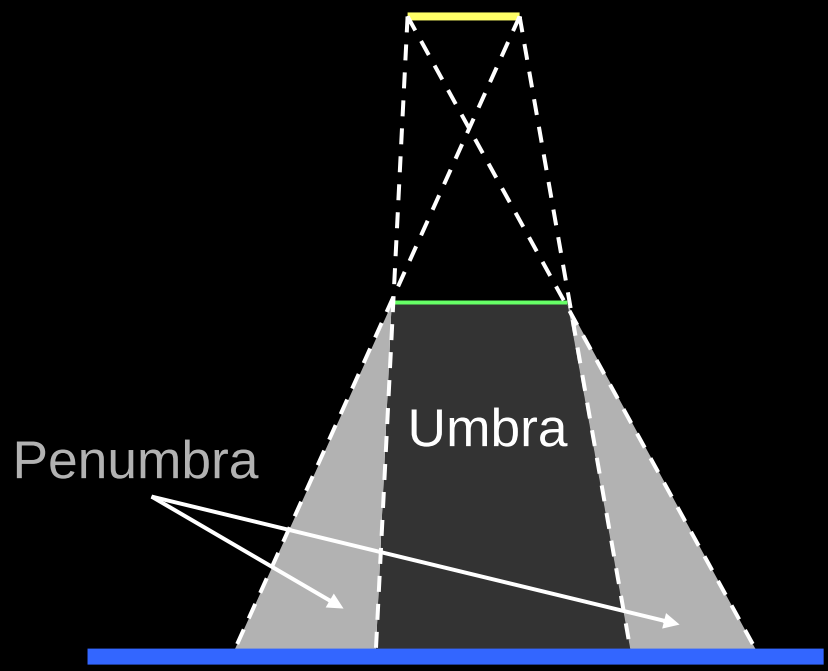


Point Light vs. Area Light

SIGGRAPH2005



Point Light



Area Light