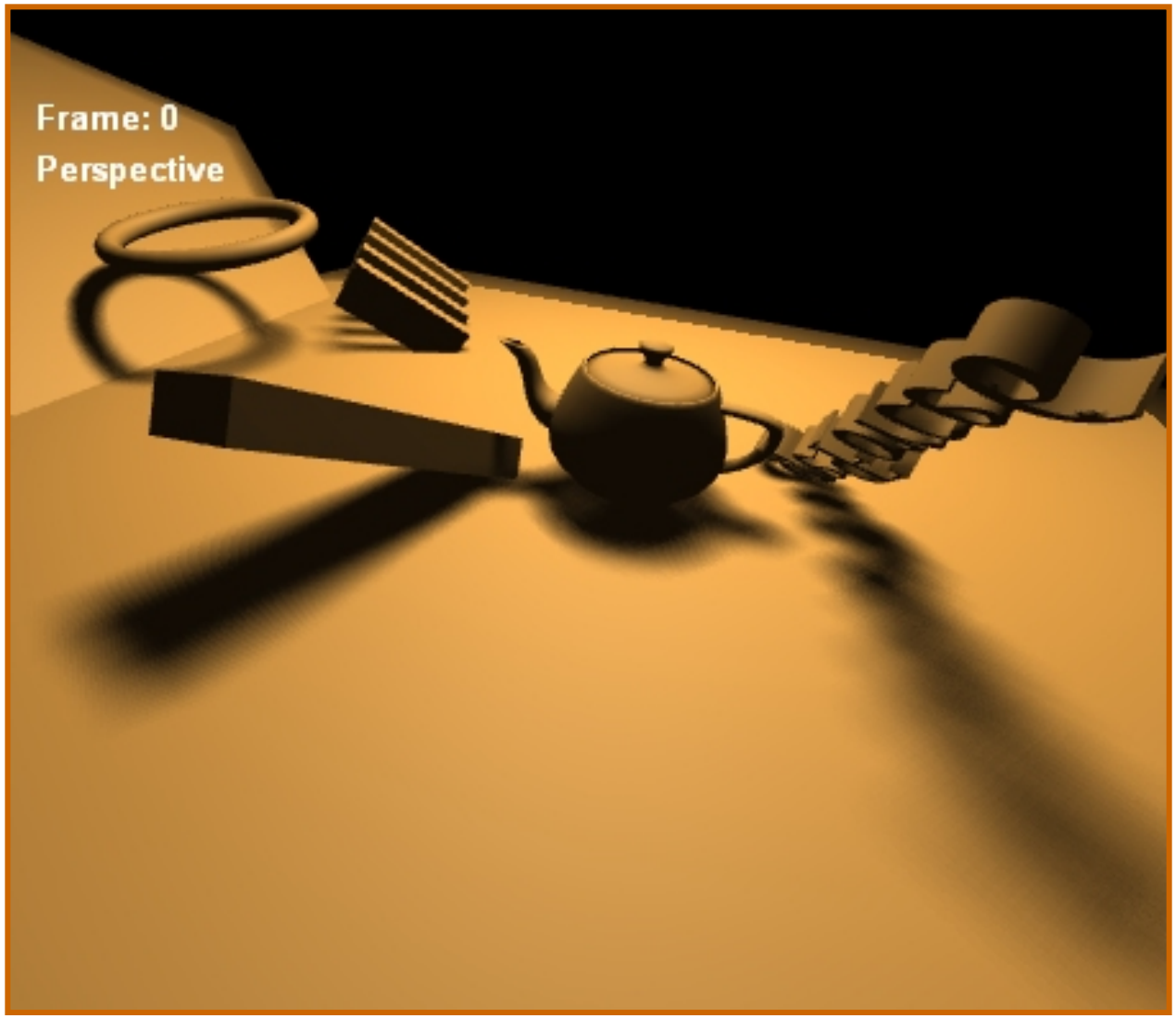
Percentage-Closer Soft Shadows

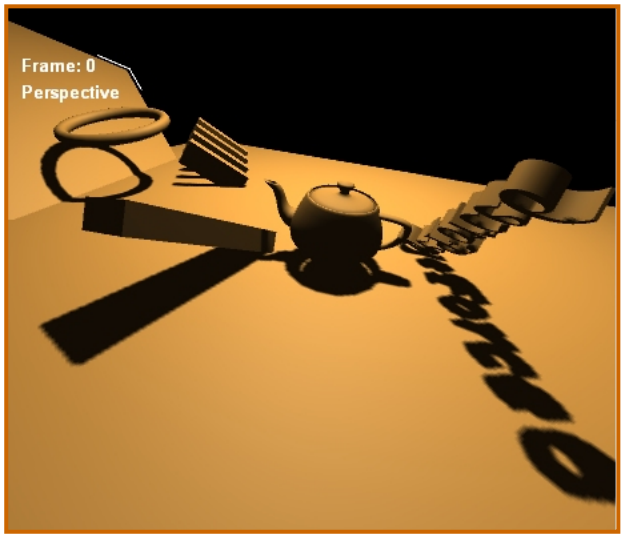
Randima (Randy) Fernando

NVIDIA Developer Technology Group

Demo

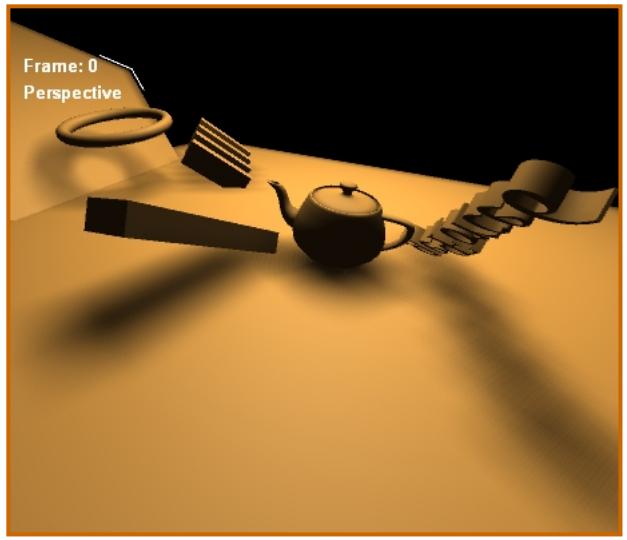


Algorithm Comparison



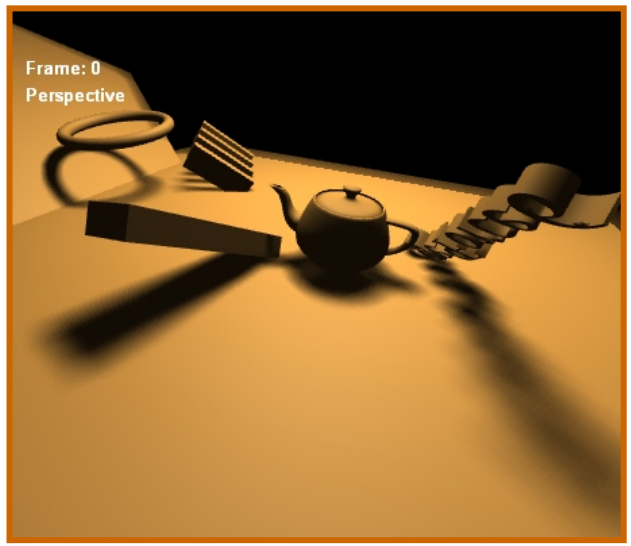
Regular Shadow Maps

- Always hard
- Noticeable Aliasing



Uniform Soft Shadows

- Always soft
- Aliasing is hidden



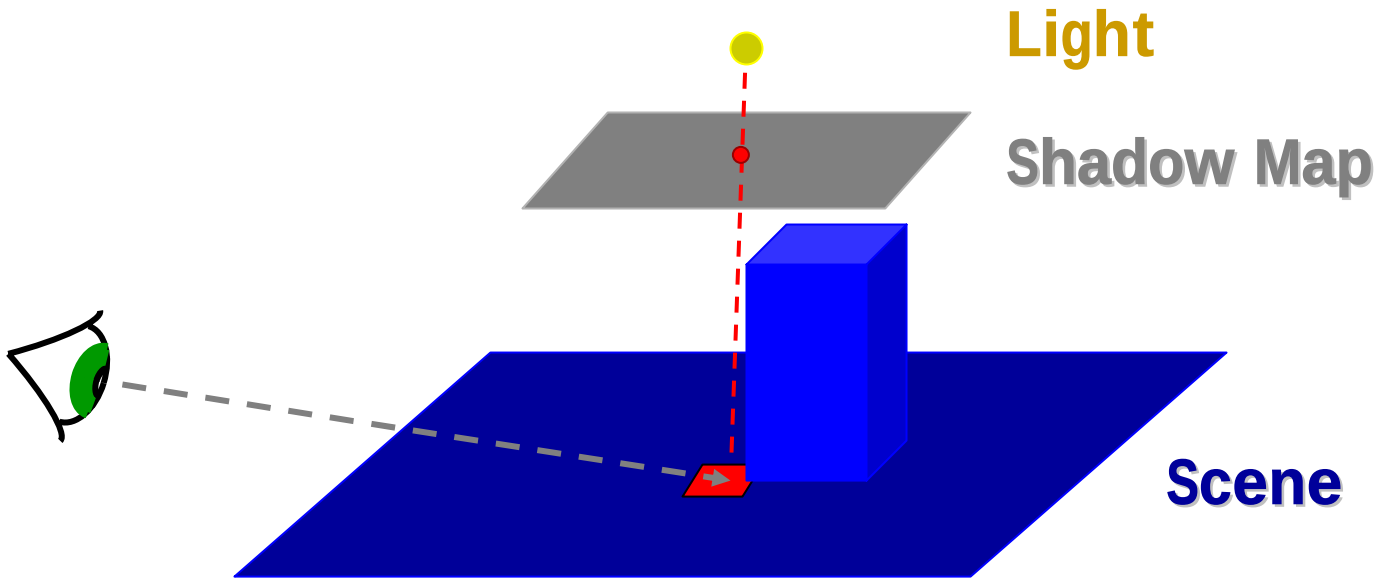
Percentage-Closer Soft Shadows

- Shadows harden on contact
- Aliasing is hidden

Features

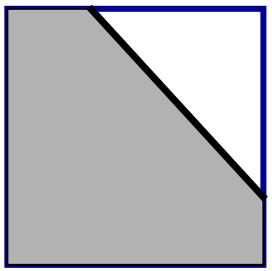
- Perceptually-correct soft shadows (good visual cues)
- Artifacts vary smoothly (no popping)
- Benefits from shadow mapping features
  - Independent of geometric complexity
  - Works with alpha testing, displacement mapping, etc…
- Integrates easily
  - Single floating-point shadow map and one shader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

Ordinary Shadow Mapping

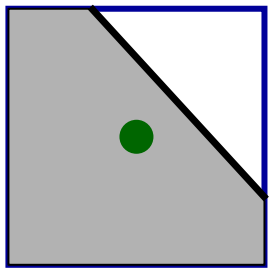


Percentage-Closer Filtering

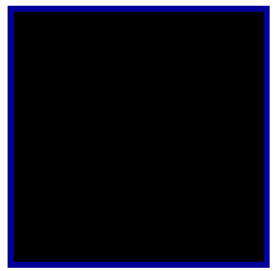
- **Extension to shadow mapping**
- **How Percentage-Closer Filtering (PCF) works:**



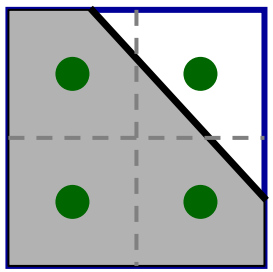
Pixel to be Shaded



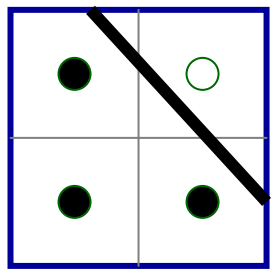
Typical Shadow Map Test



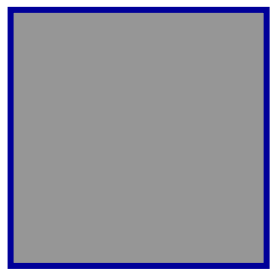
Shaded Pixel (Black or White)



4-Sample PCF



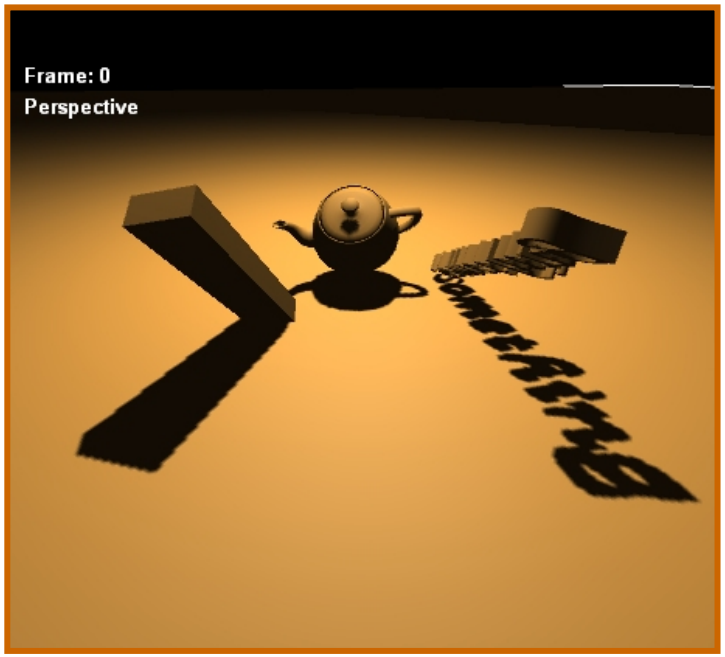
Perform 4 Depth Tests

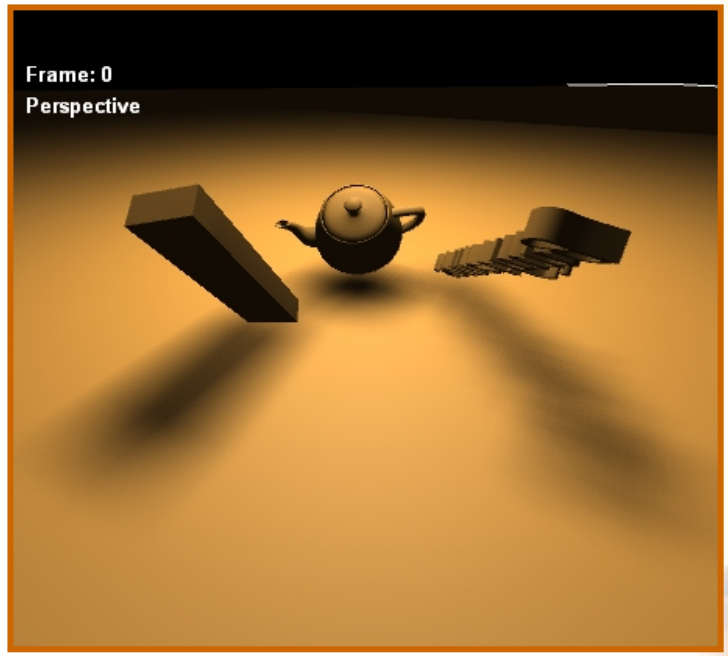


Shaded Pixel (0, 0.25, 0.50, 0.75, 1.0)

Basic Idea

- Soften shadows by varying PCF kernel width





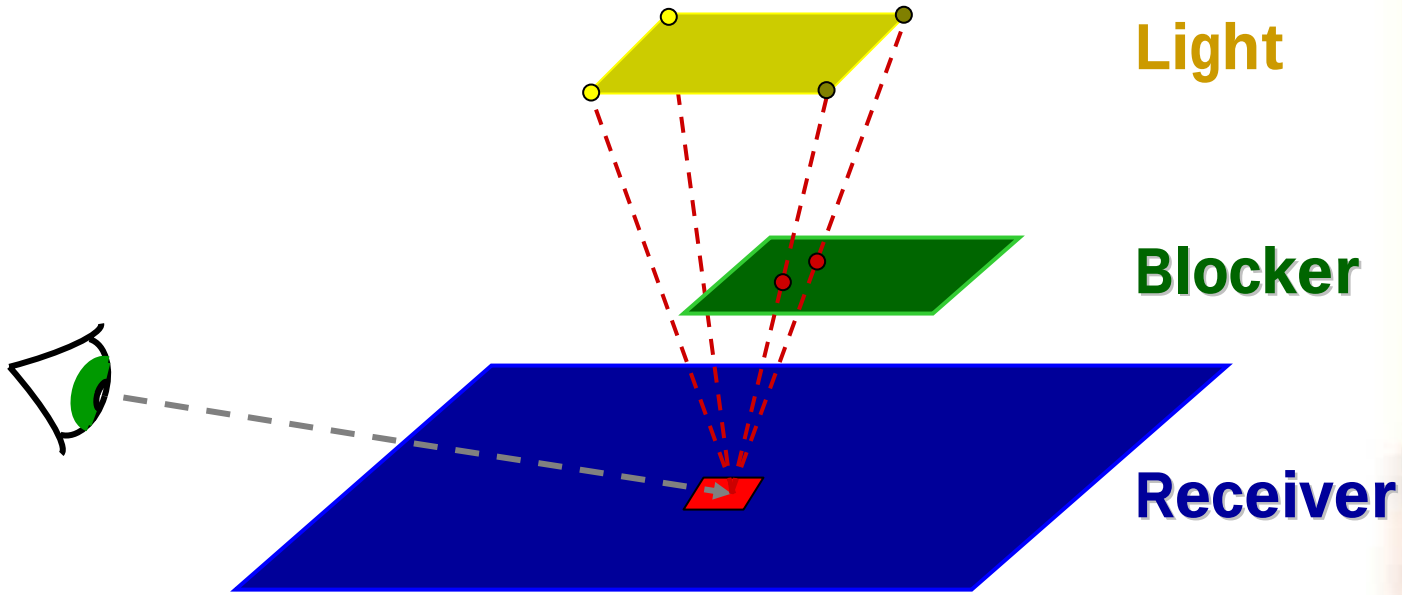
Small Kernel
(Narrow Filter)

Large Kernel
(Wide Filter)

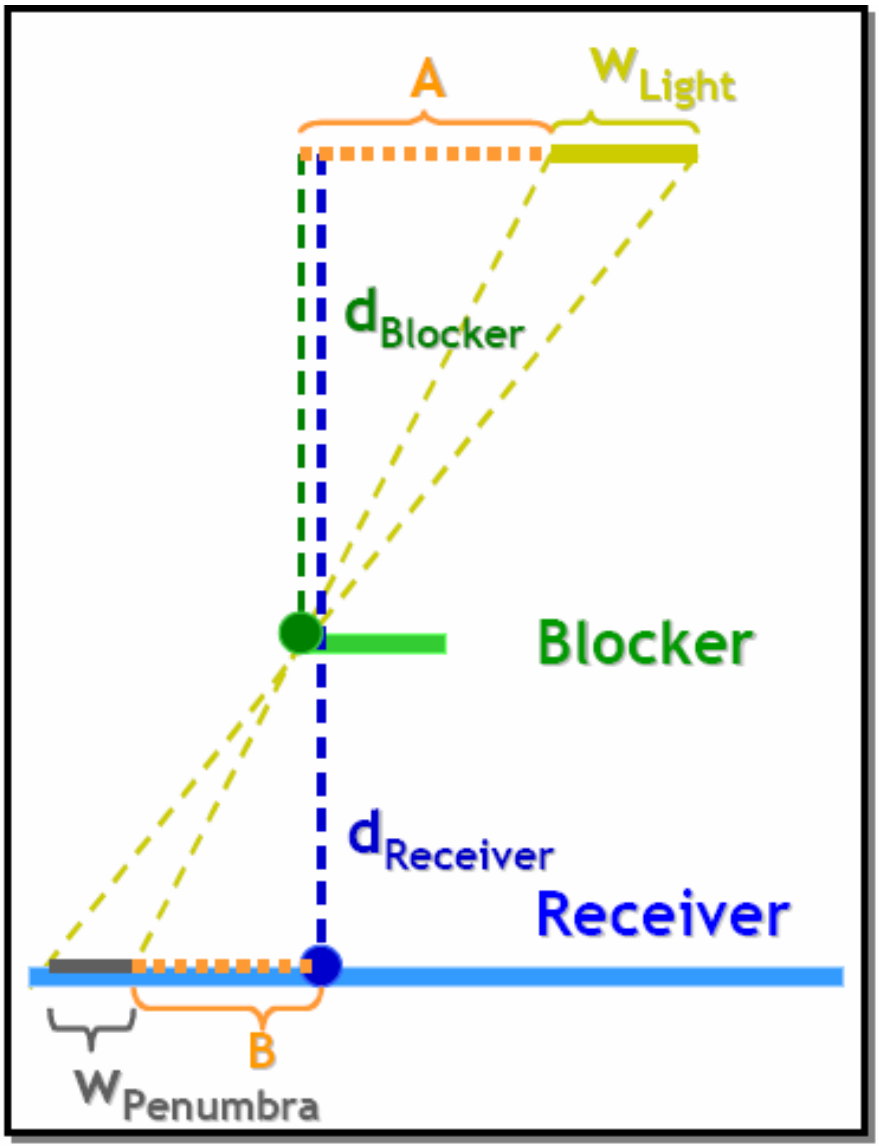
Penumbra Estimation

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

“Blockers” and “Receivers”



Penumbra Size Estimation



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

- **Assumes that blocker, receiver, and light are parallel**

Penumbra Size Estimation

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

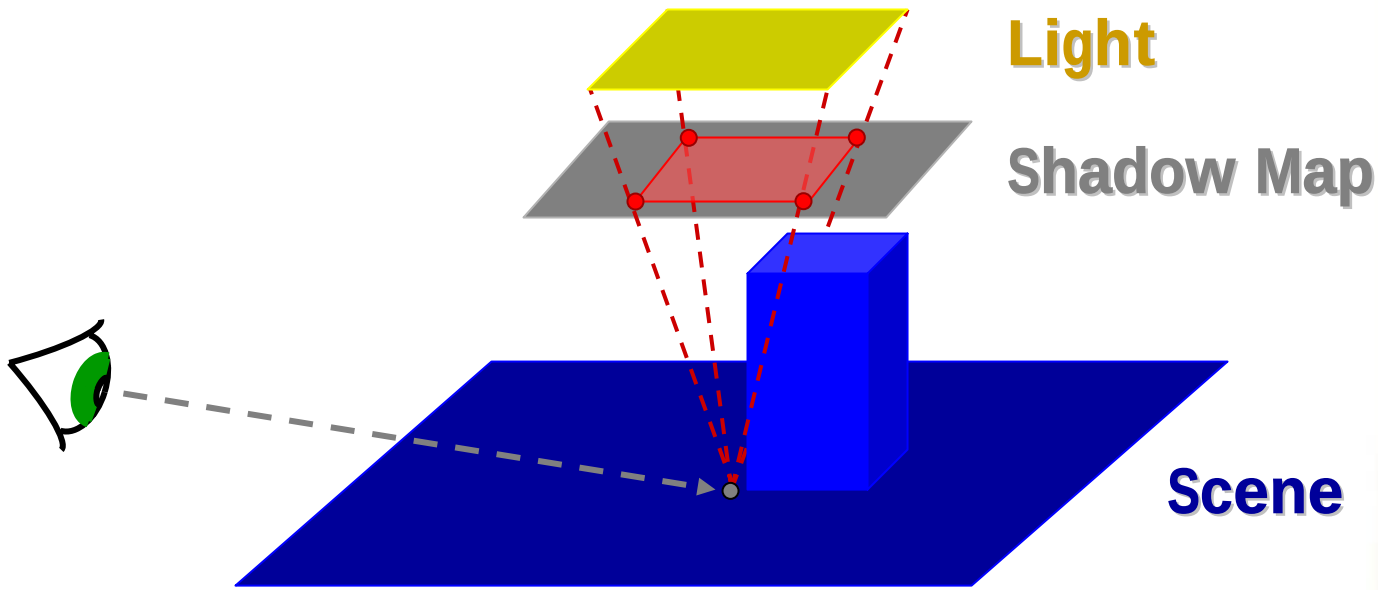
- **We need:**
  - **Distance from blocker to light source**
    - **Don't know this… yet.**
  - **Distance from receiver to light source**
    - **Depth of the point we're shading**
  - **Light size**
    - **Uniform input to the shader**

Main Algorithm

- Generate a shadow map (as usual)
- When shading each pixel on the screen:
  - Blocker Search
  - Penumbra Size Estimation
  - Variable Percentage-Closer Filtering

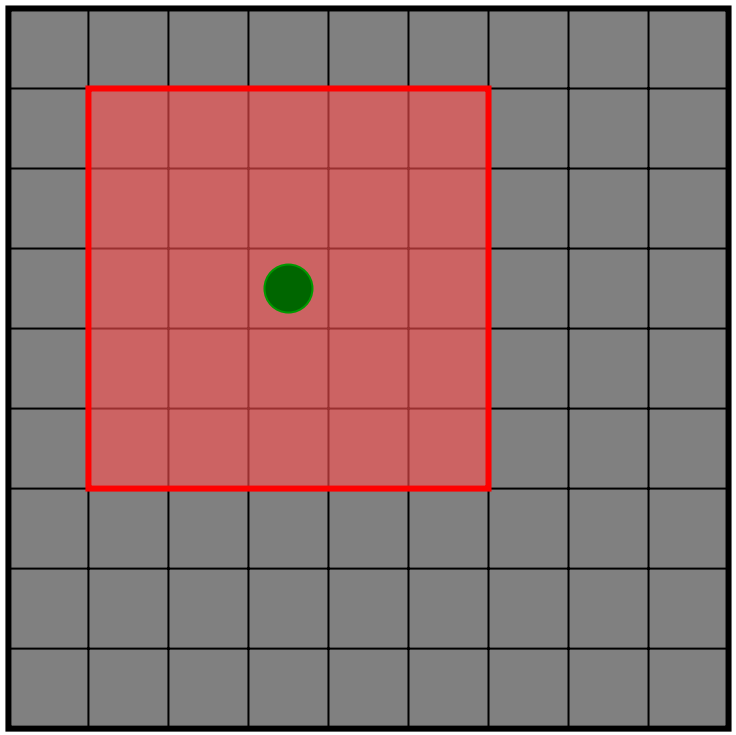
Blocker Search

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



Blocker Search

- Iterate through all texels in 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**
  - **Further exploration in progress…**

Penumbra Size Estimation

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

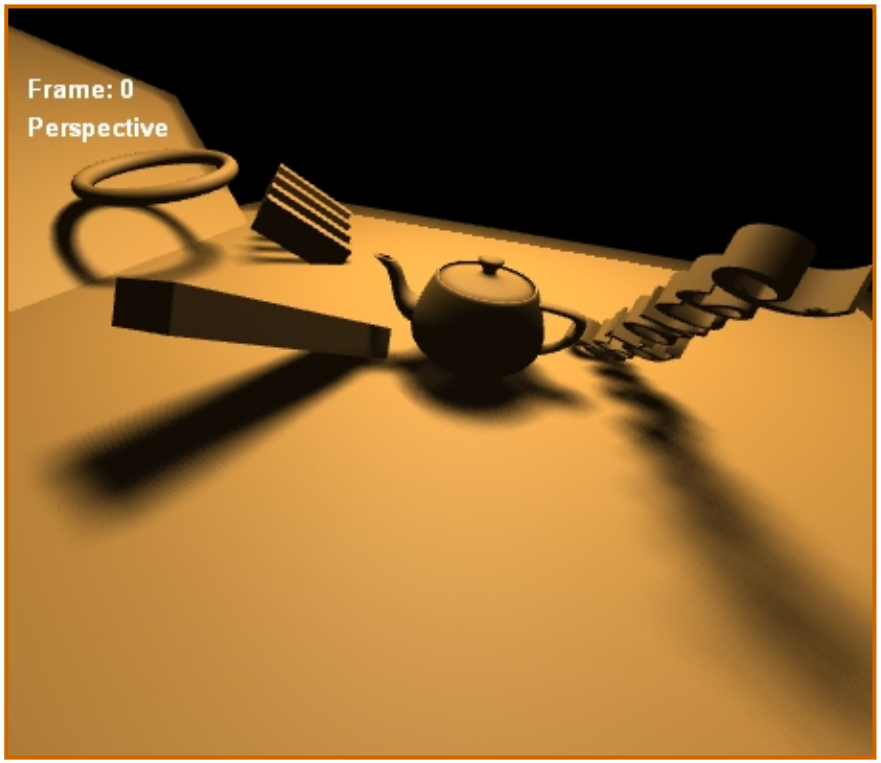
- **We have:**
  - **Distance from blocker to light source**
    - **Result of blocker search**
  - **Distance from receiver to light source**
    - **Depth of the point we're shading**
  - **Light size**
    - **Uniform input to the shader**
- **Estimate penumbra per pixel**
  - **Change PCF kernel based on the result**

Variable Percentage-Closer Filtering

- **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)**

Demo

- **Observed current performance:**
  - **~8 fps @ 640 x 480 in FX Composer**
  - **GeForce 6800 Ultra**



Improvements

- **Performance**
  - **Need to implement early exit for PCF**
    - **Currently very wasteful (256 samples always!)**
  - **No profiling/tuning done yet**
  - **Mask out umbras and fully-lit regions**
- **Quality**
  - **Better blocker-search heuristics**
  - **Better filtering to remove banding in large penumbras**

Parting Thoughts

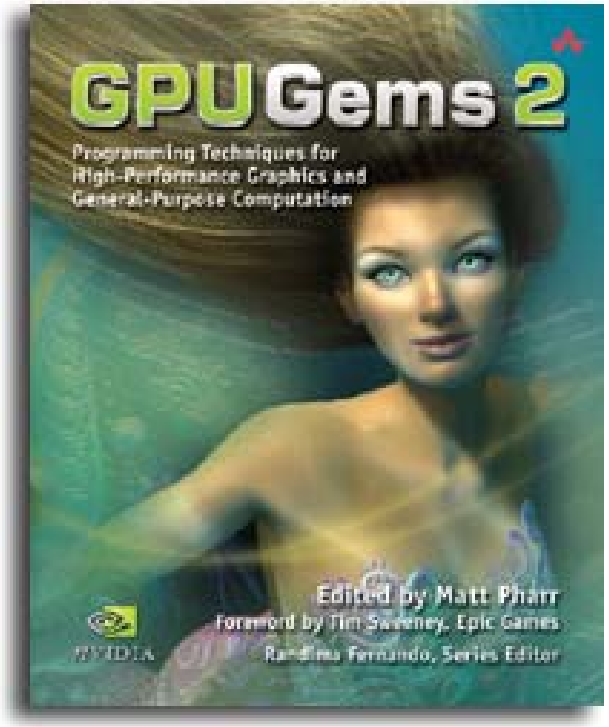
- Algorithm is completely encapsulated in one shader file for easy integration
- Try it out – please let us know what you find
- Tweak “Near Plane Factor” and “Shadow Map Bias” to match your scene
- Applications: DCC/CAD applications, pre-visualization, future games
- Improved version, video, and slides on the way…

Suggestions/Questions Welcome

- Lots of relevant references in:
  - http://www.randima.com/MastersThesis.pdf
- Slides and code on
  http://developer.nvidia.com
  The Source for GPU Programming
- E-mail: rfernando@nvidia.com
- Thanks to Kevin Bjorke, whose basic PCF shader I based this work on.
- Thanks to Chris Maughan and FX Composer for making shader development so easy.

GPU Gems 2

Programming Techniques for High-Performance Graphi and General-Purpose Computation



- **880 full-color pages, 330 figures, hard cover**
- **$59.99**
- **Experts from universities and industry**

"The topics covered in *GPU Gems 2* are critical to the next generation of game engines."

— Gary McTaggart, Software Engineer at Valve, Creators of Half-Life and Counter-Strike

"*GPU Gems 2* isn't meant to simply adorn your bookshelf—it's required reading for anyone trying to keep pace with the rapid evolution of programmable graphics. If you're serious about graphics, this book will take you to the edge of what the GPU can do."

—Rémi Arnaud, Graphics Architect at Sony Computer Entertainment