Rendering Fake Soft Shadows with Smoothies

Eric Chan
Massachusetts Institute of Technology

SIGGRAPH2004

CSAIL

Clarification

Clarification

Real-Time Soft Shadows

SIGGRAPH2004

Goals:

- Interactive framerates
- Hardware-accelerated
- Good image quality
- Dynamic environments

NVIDIA

Challenge:

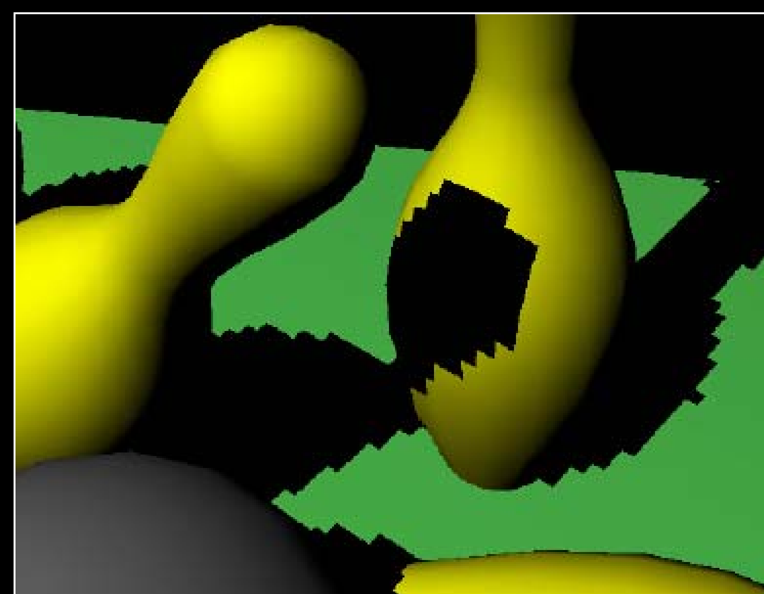
- How to balance quality and performance?

Ordinary Shadow Maps

Image-space algorithm:

- Fast and simple
- Supported in hardware
- Aliasing artifacts

NVIDIA



Sen et al. [SIGGRAPH 2003]

Soft Shadow Maps

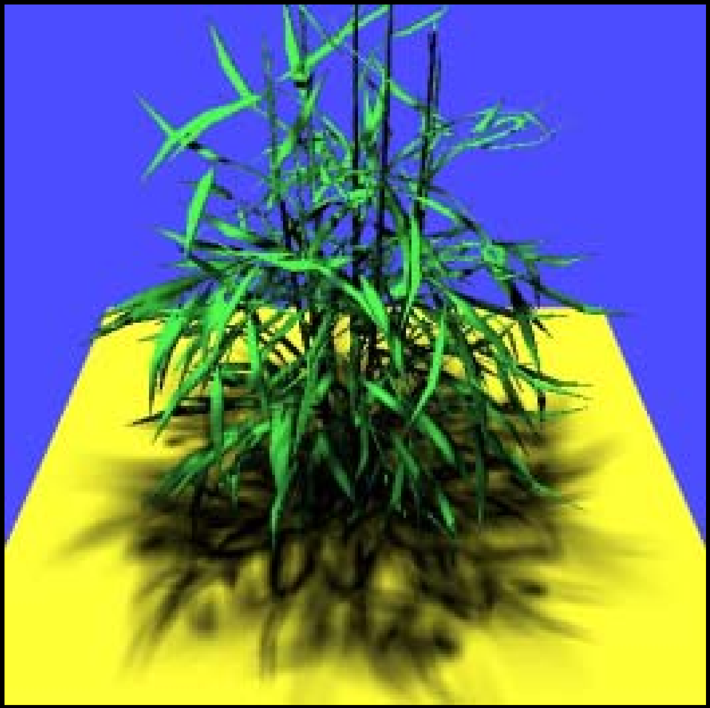
SIGGRAPH2004

Techniques:

- Filtering
- Stochastic sampling
- Image warping

Examples:

- Percentage closer filtering (Reeves et al., SIG1987)
- Deep shadow maps (Lokovic and Veach, SIG2000)



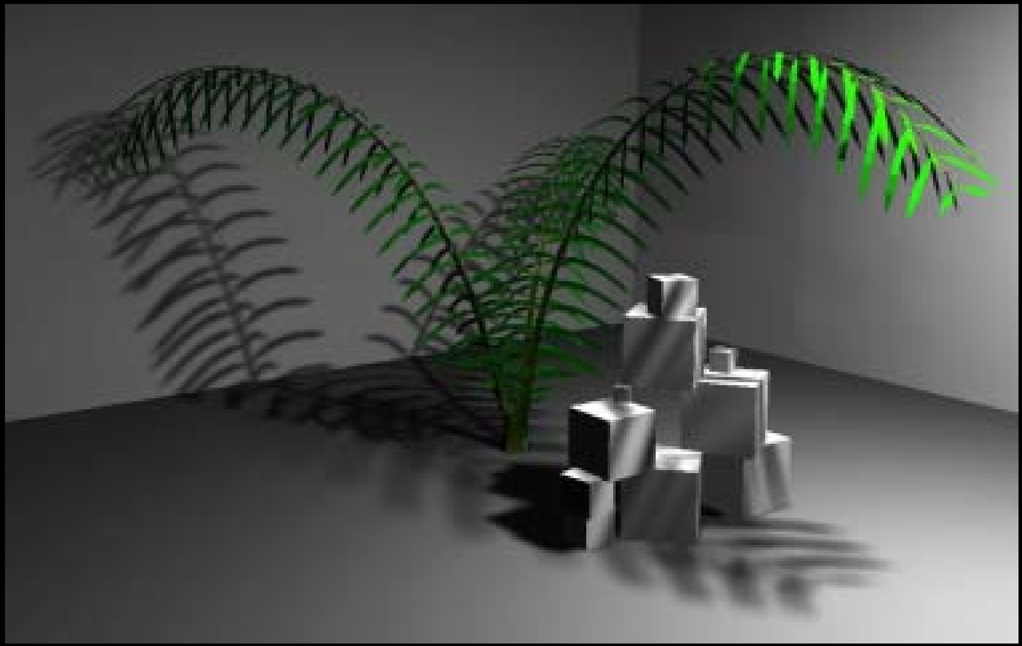
Agrawala et al. [SIGGRAPH 2000]

But: need dense sampling to minimize artifacts

Soft Shadow Maps (cont.)

SIGGRAPH2004

Approximations



Soler and Sillion

Examples:

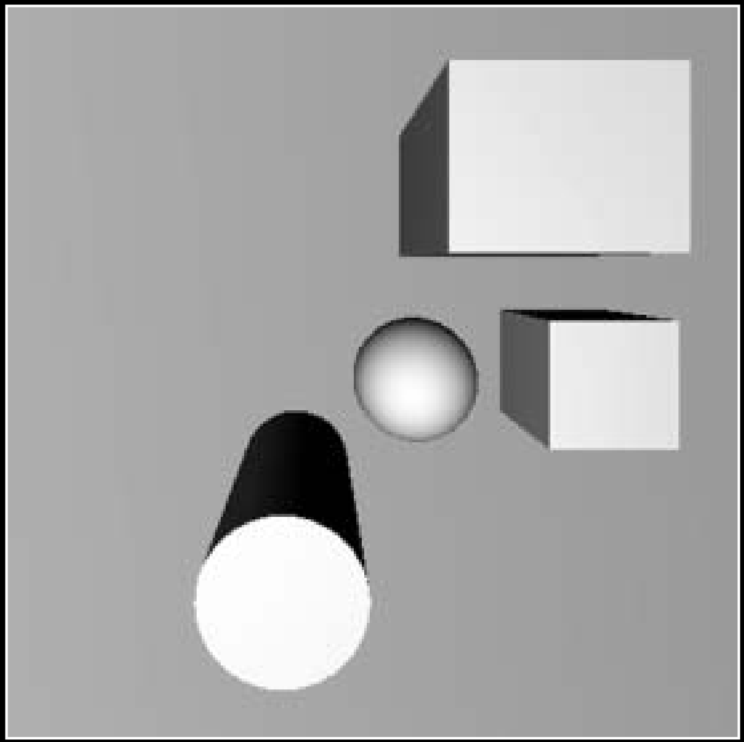
- Convolution (Soler and Sillion, SIGGRAPH 1998)
- Linear lights (Heidrich et al., EGRW 2000)

Idea

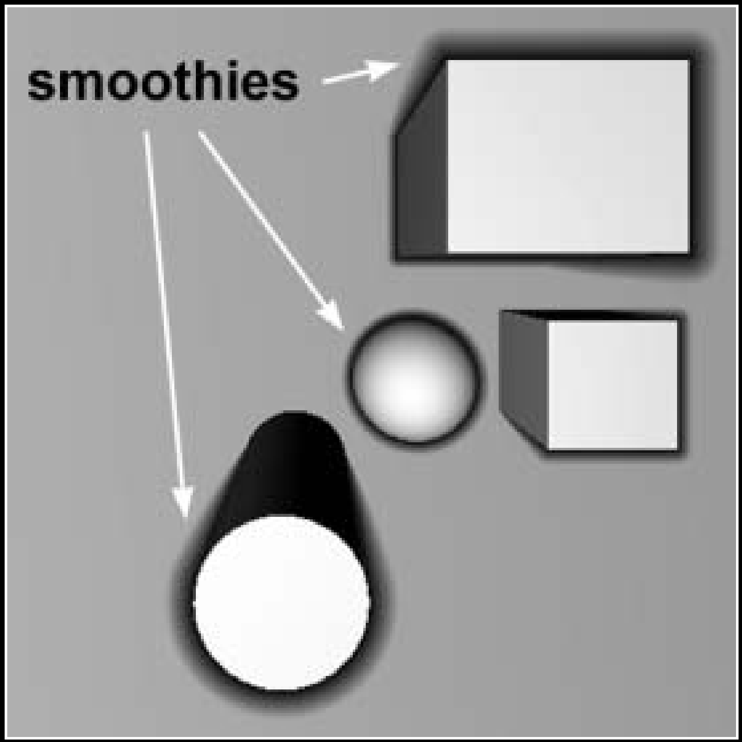
SIGGRAPH2004

Extend basic shadow map approach

Extra primitives (smoothies) soften shadows



light's view (blockers only)



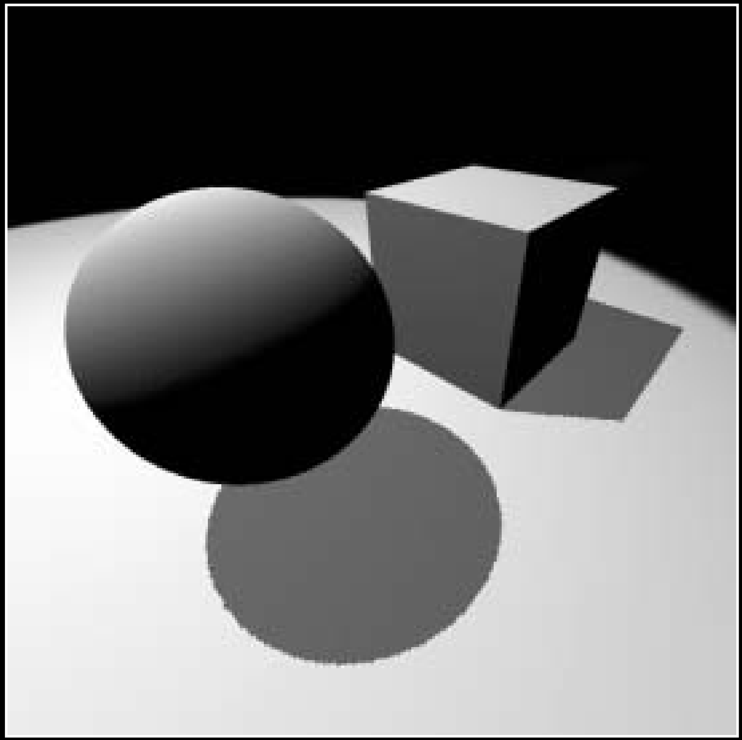
light's view (blockers + smoothies)

Fake Soft Shadows

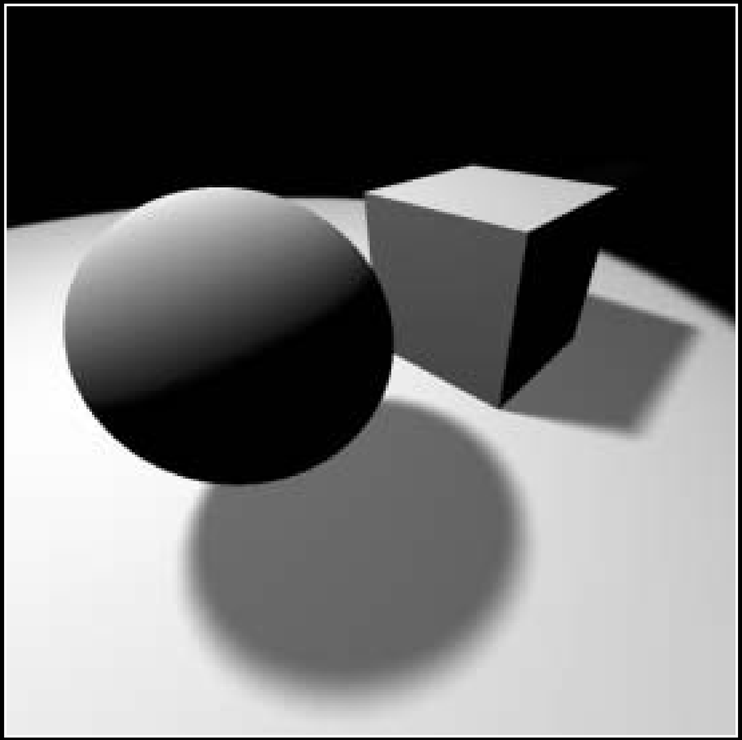
SIGGRAPH2004

Shadows not geometrically correct

Shadows appear qualitatively like soft shadows



Hard shadows



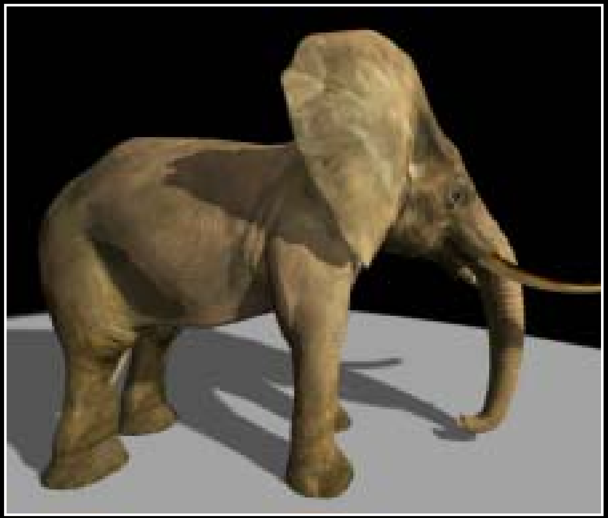
Fake soft shadows

Smoothie Algorithm

SIGGRAPH2004

Properties:

- Creates soft shadow edges
- Hides aliasing artifacts
- Efficient (object / image space)
- Hardware-accelerated
- Supports dynamic scenes



References

Rendering Fake Soft Shadows with Smoothies

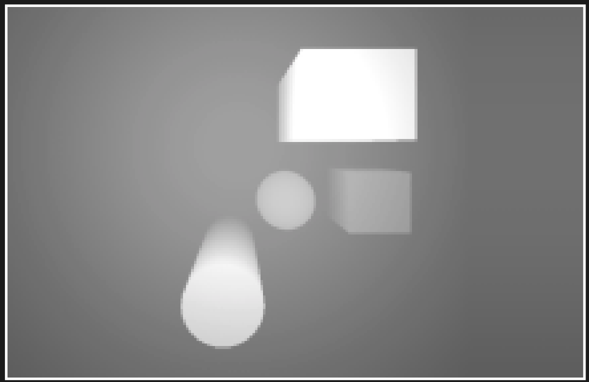
- E. Chan and F. Durand [EGSR 2003]

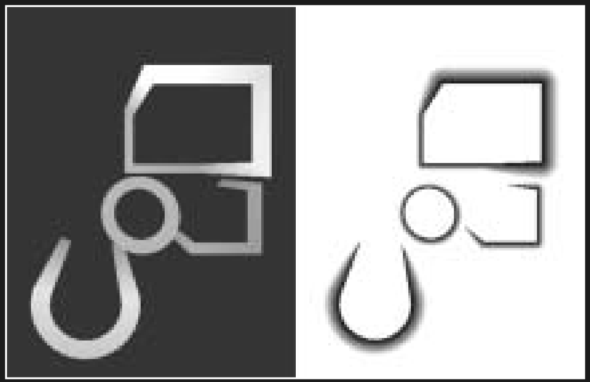
Penumbra Maps

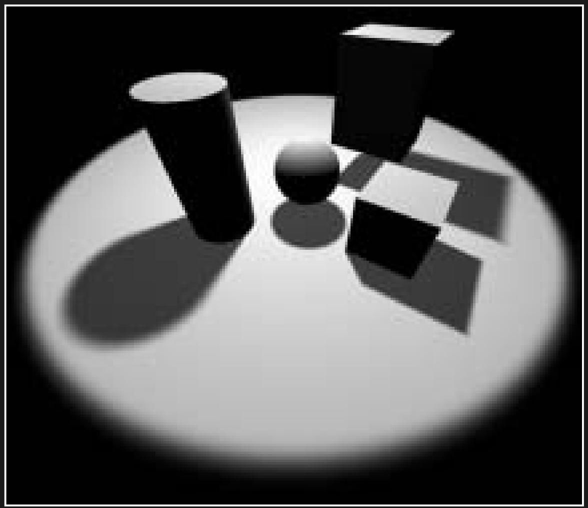
- C. Wyman and C. Hansen [EGSR 2003]

Algorithm

Algorithm Overview





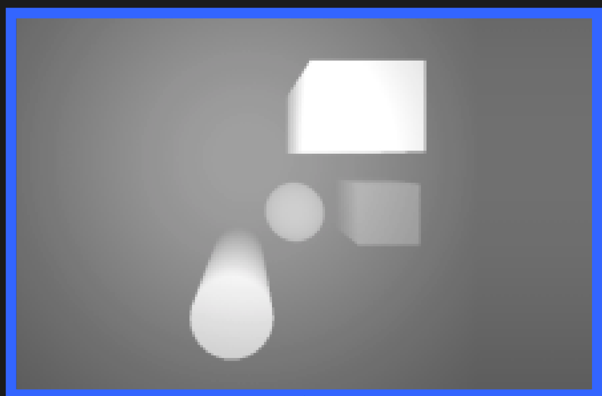


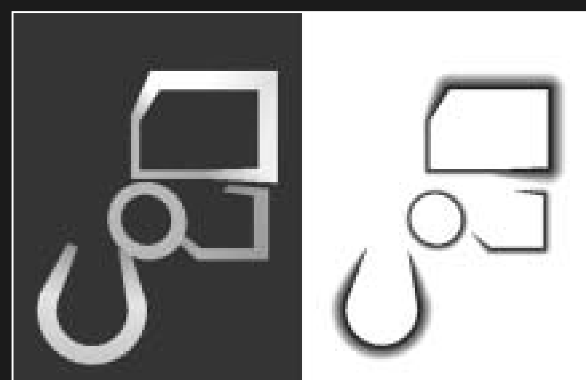
Focus on concepts

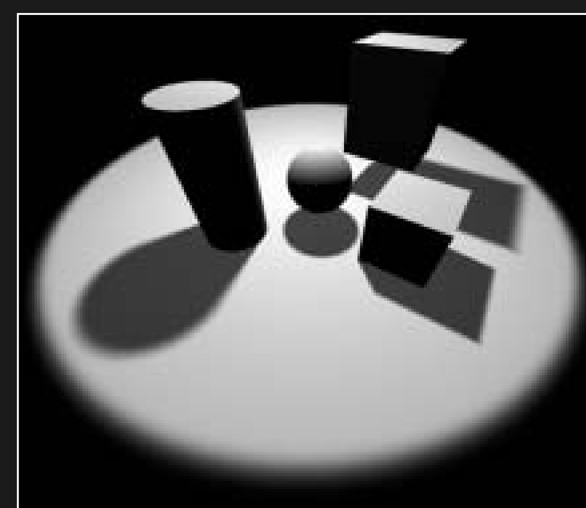
Implementation details later

Algorithm Overview

Step 1



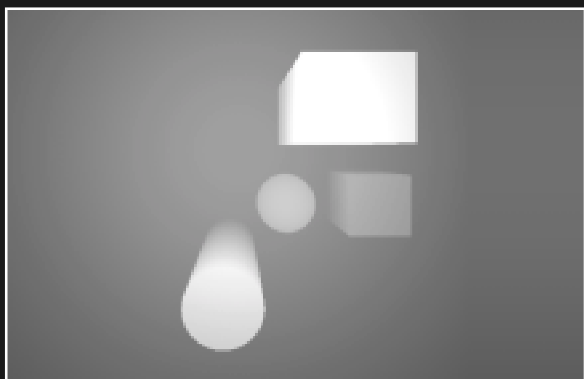


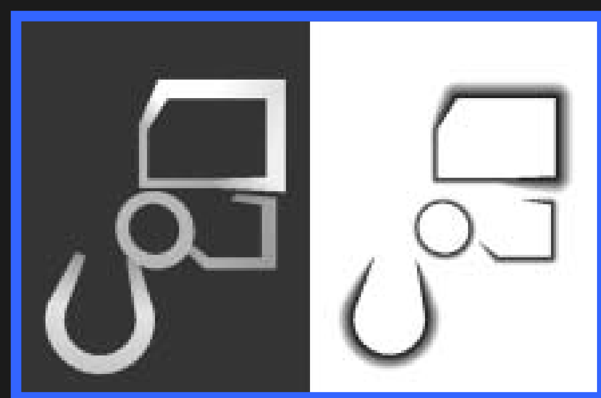


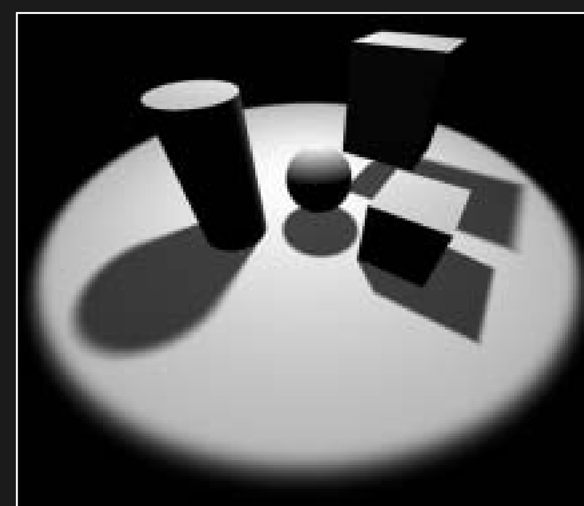
Create depth map

Algorithm Overview

Step 2





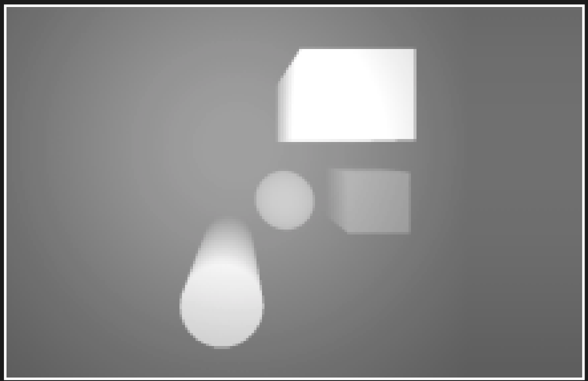


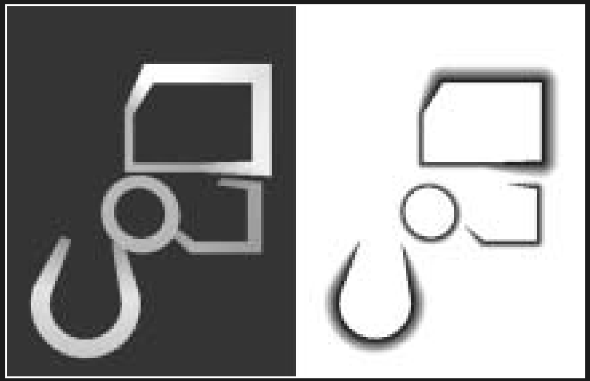
Create smoothie buffer

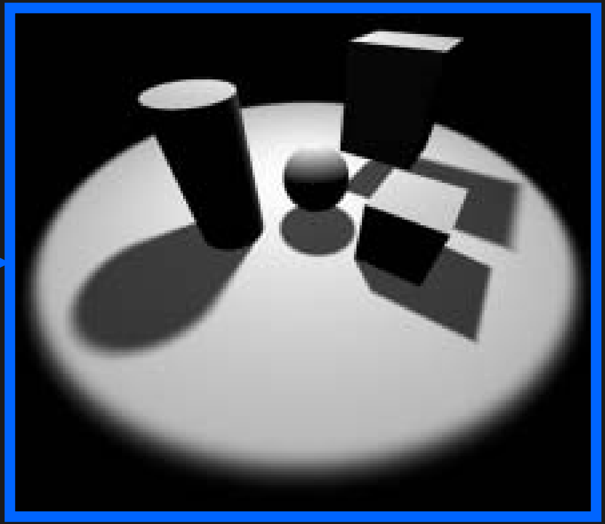
Algorithm Overview

SIGGRAPH2004

Step 3



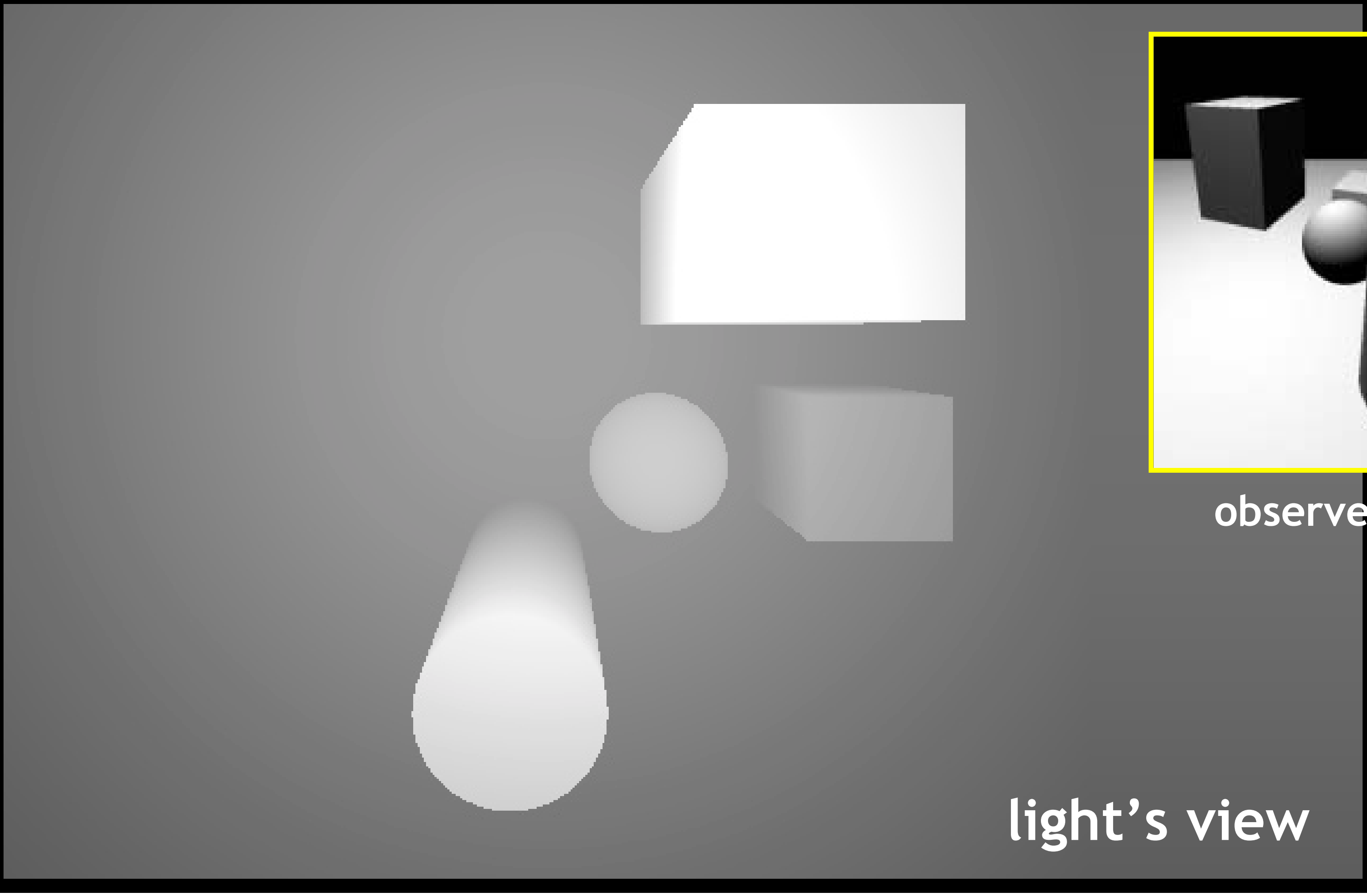


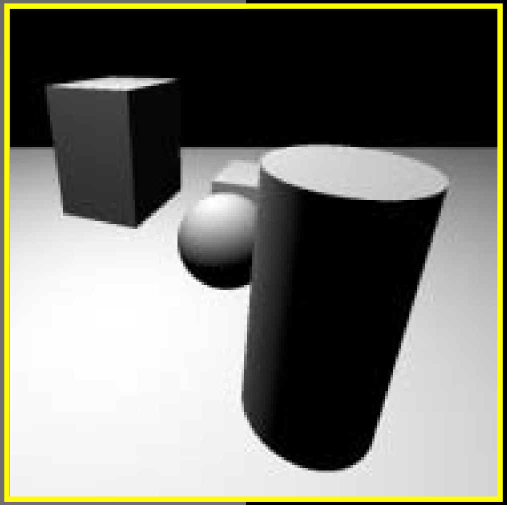


Render scene + shadows

Create Shadow Map

Render blockers into depth map

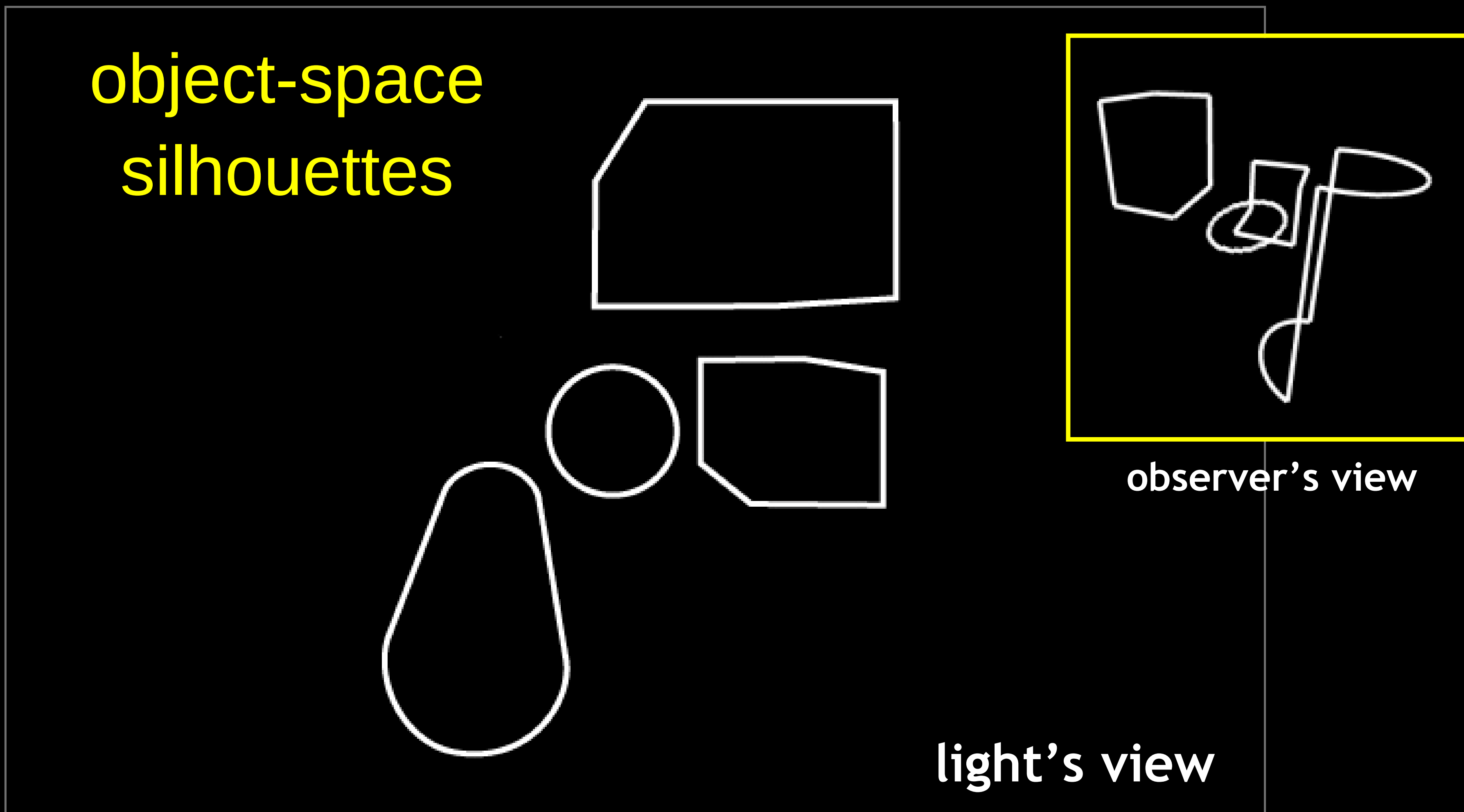




observer's view

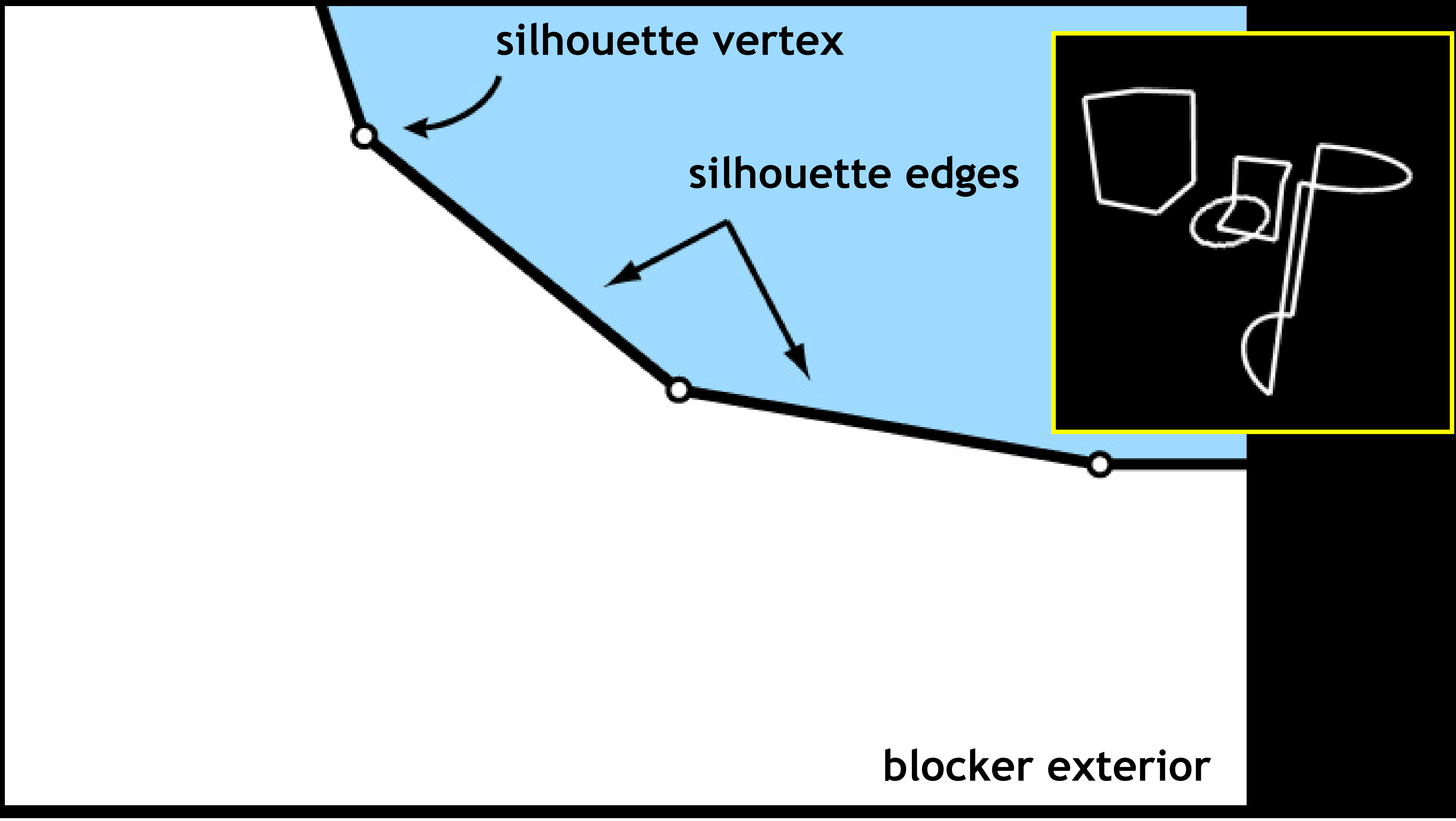
Find Silhouette Edges

Find blockers' silhouette edges in object space



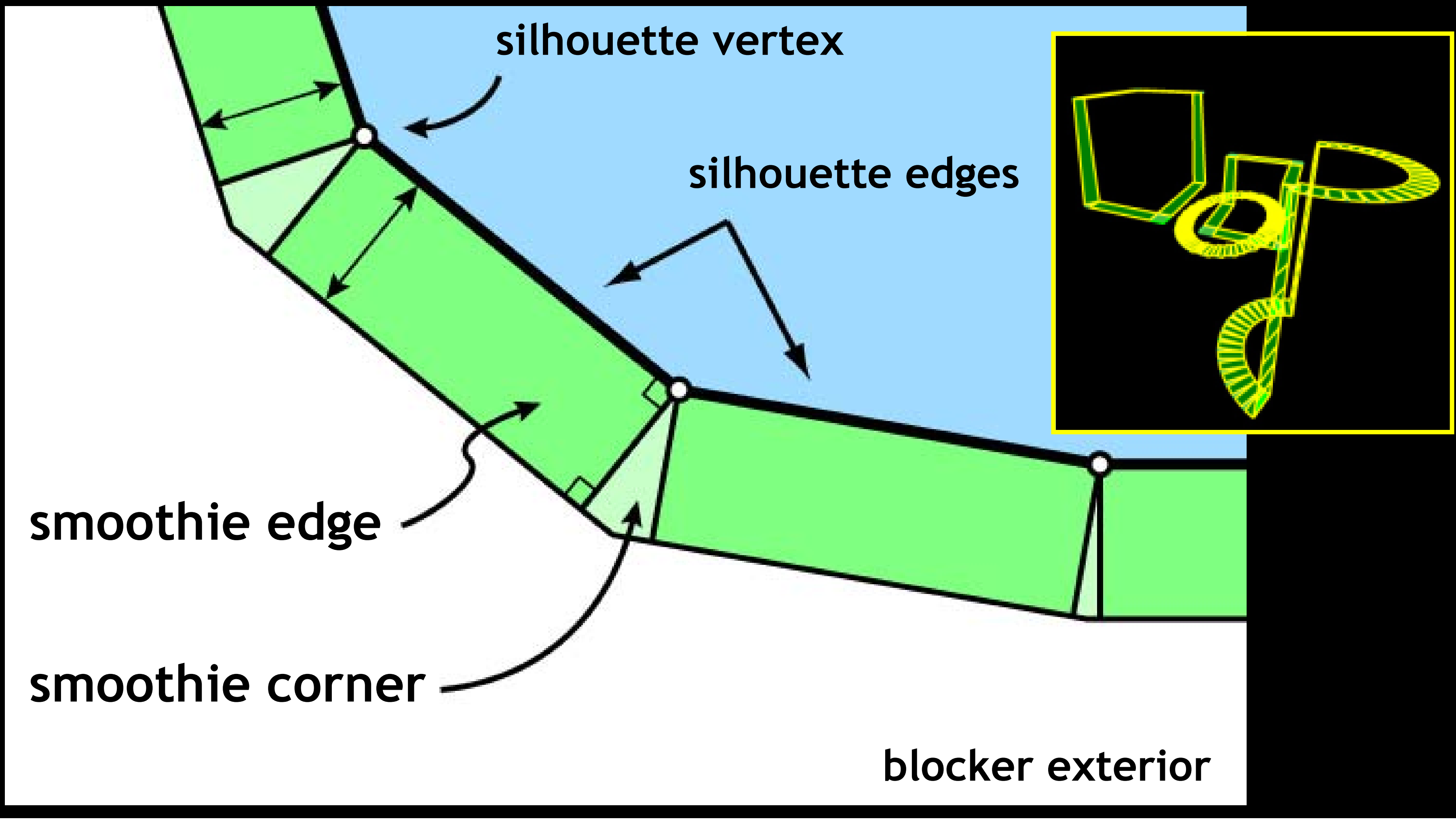
Construct Smoothies

Blocker only:



Construct Smoothies

Blocker + smoothies:

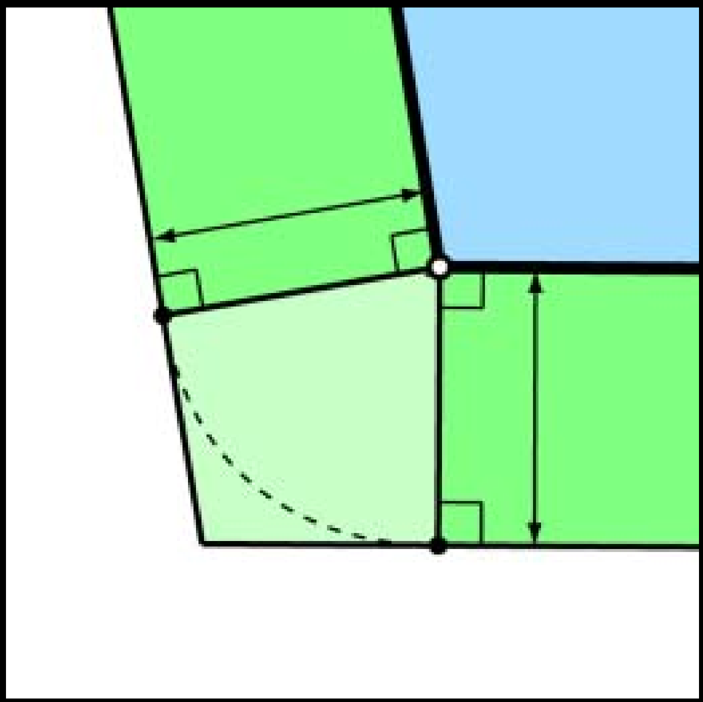


Construct Smoothies

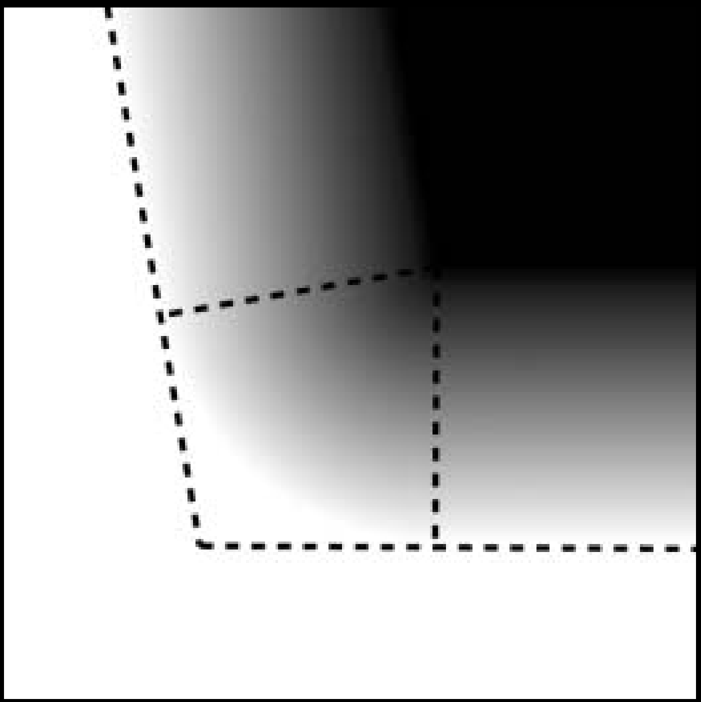
SIGGRAPH2004

Smoothie edges are fixed-width rectangles in screen space

Smoothie corners connect adjacent smoothie edges



geometry

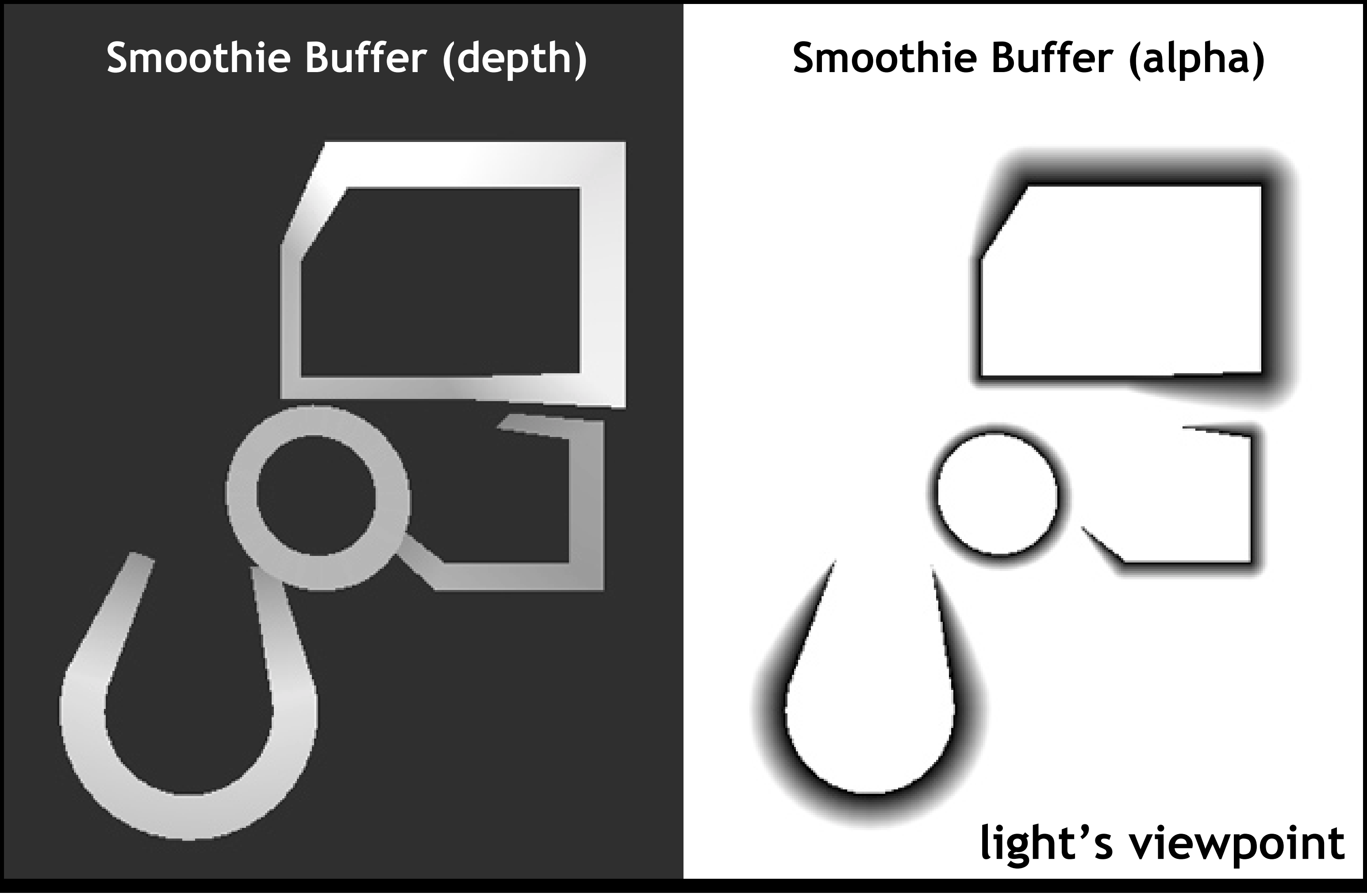


shading

Render Smoothies

SIGGRAPH2004

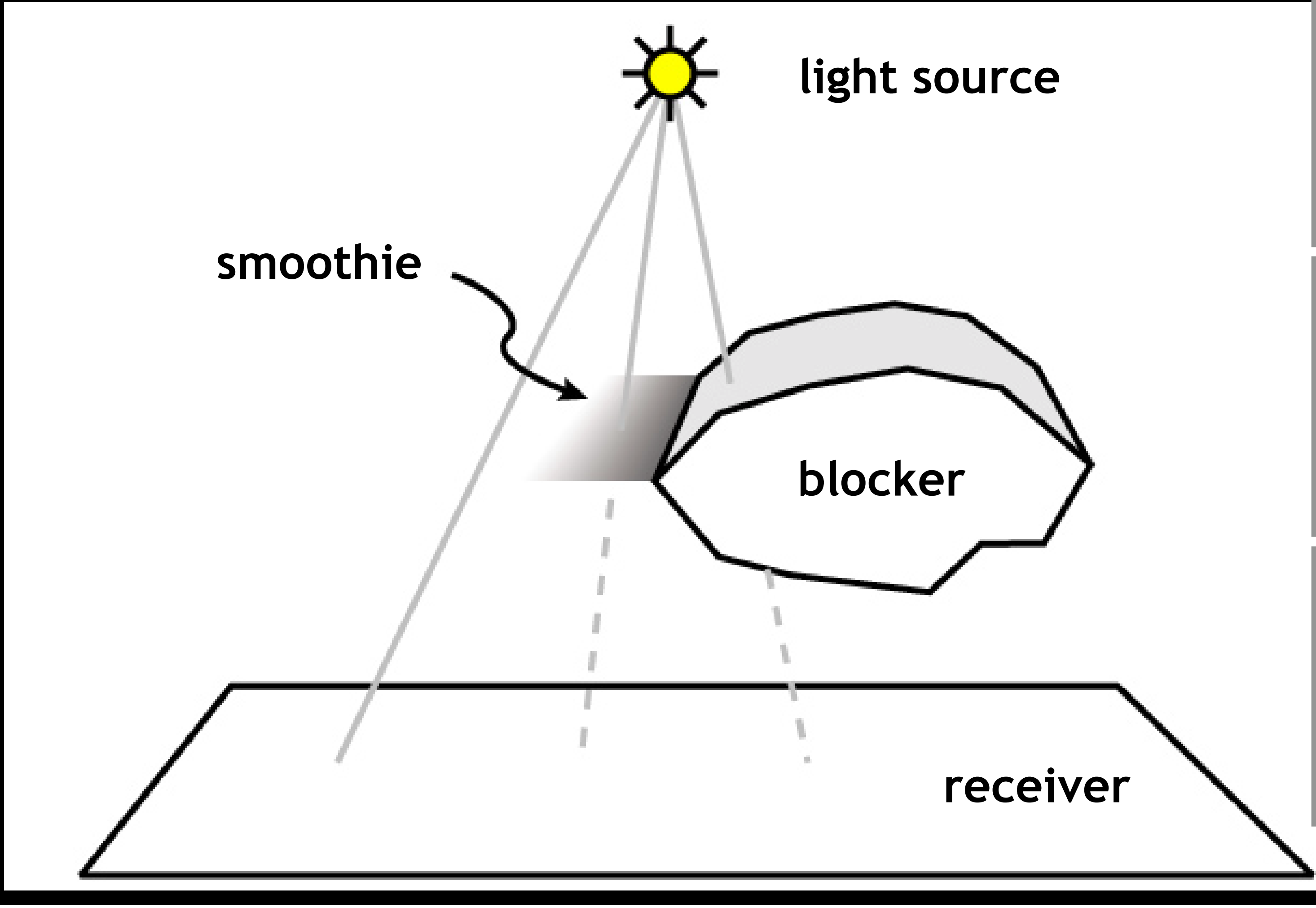
Store depth and alpha values into smoothie buffer



Compute Shadows

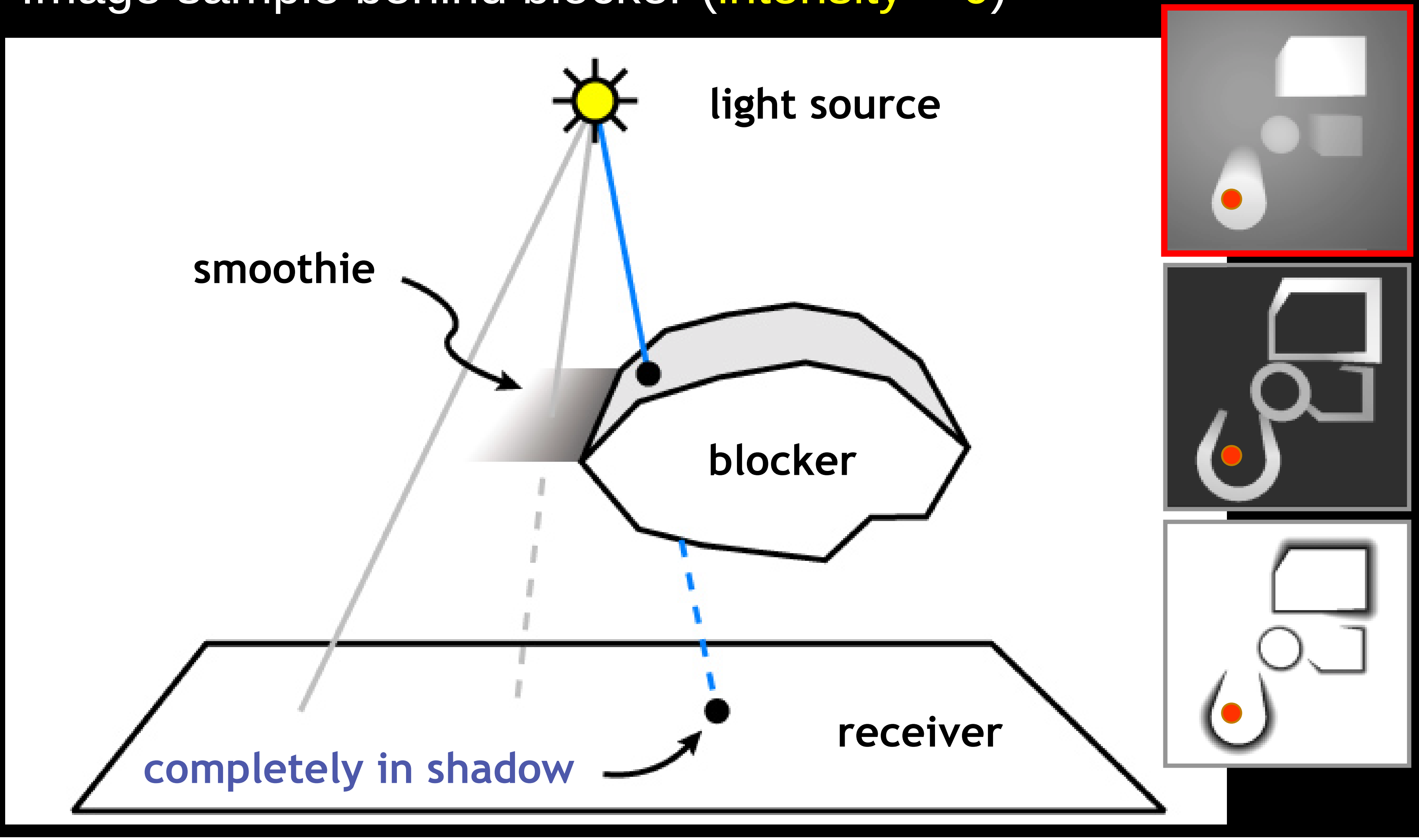
SIGGRAPH2004

Compute intensity using depth comparisons



Compute Shadows (1 of 3)

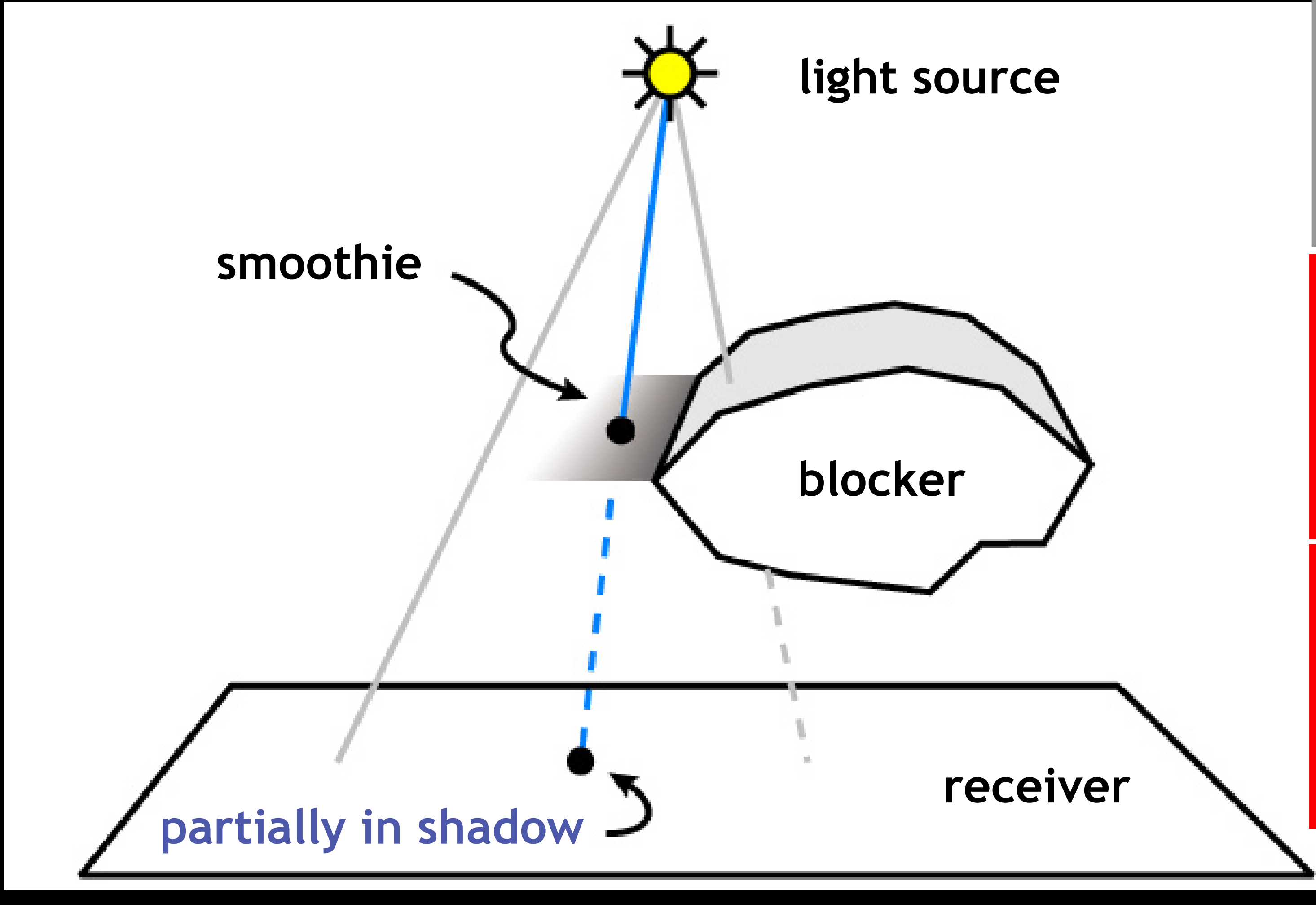
Image sample behind blocker (intensity = 0)



Compute Shadows (2 of 3)

SIGGRAPH2004

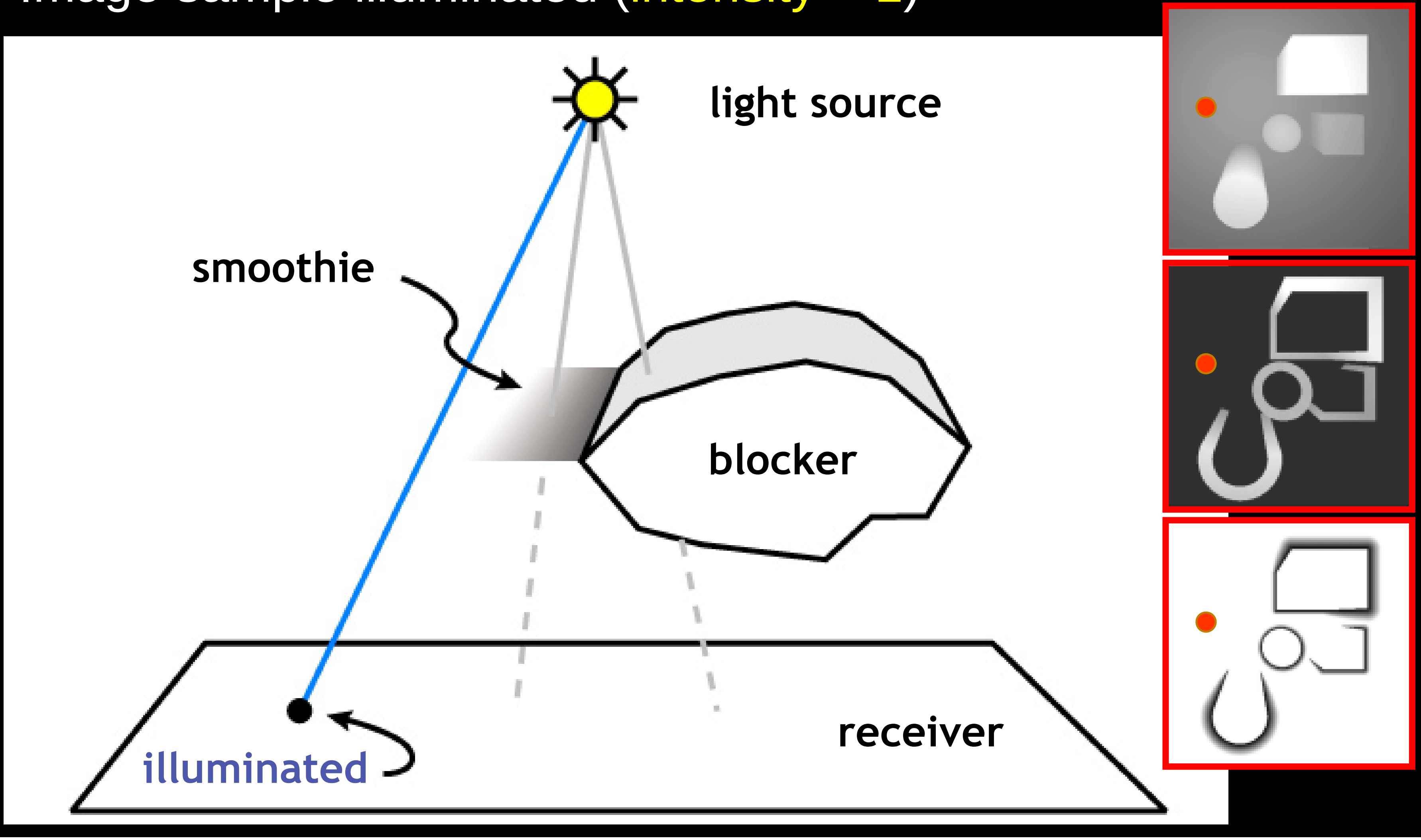
Image sample behind smoothie (intensity = α)



Compute Shadows (3 of 3)

SIGGRAPH2004

Image sample illuminated (intensity = 1)

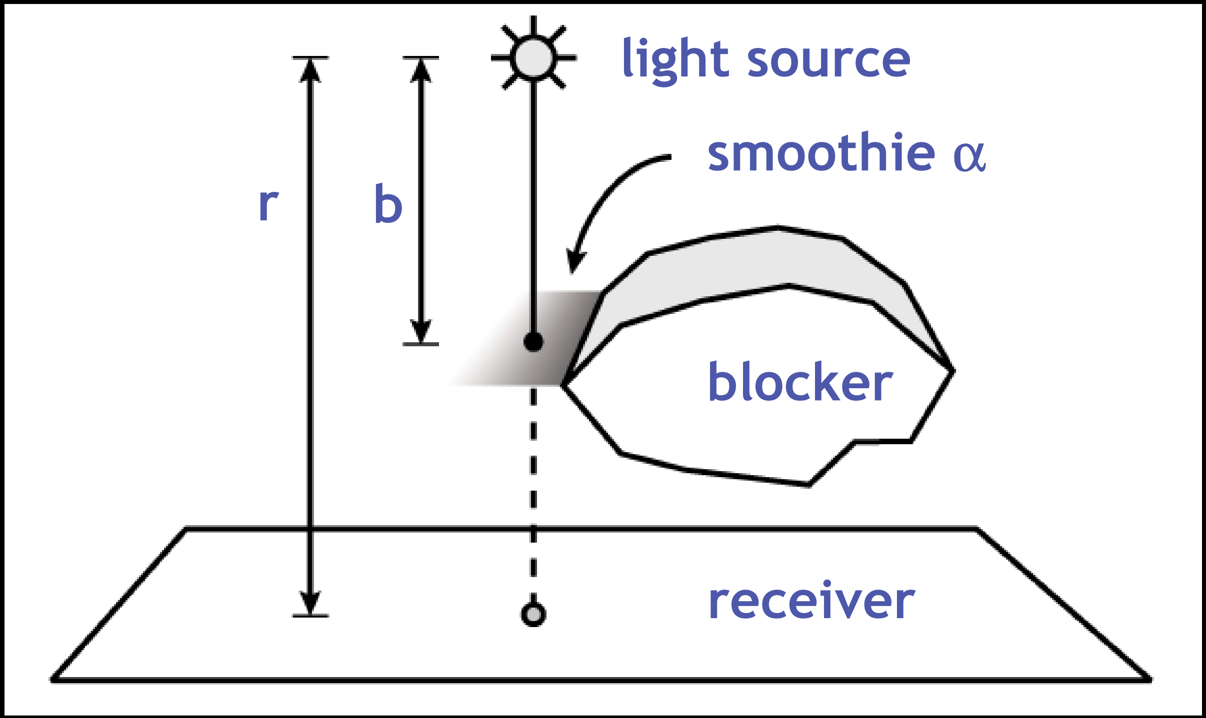


Computing Alpha Values

SIGGRAPH2004

Intuition:

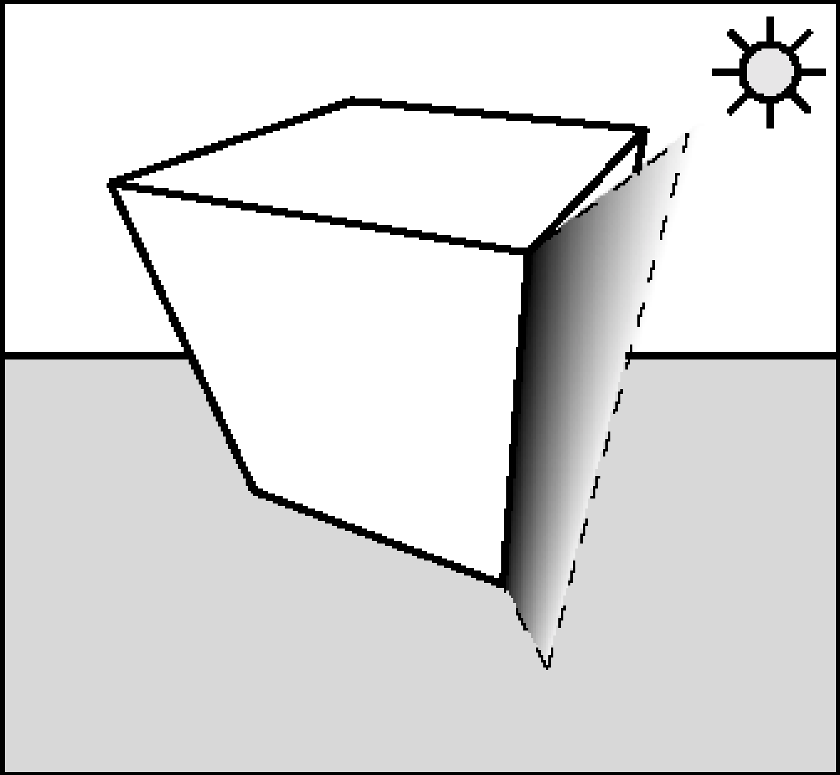
- Alpha defines penumbra shape
- Should vary with ratio b/r



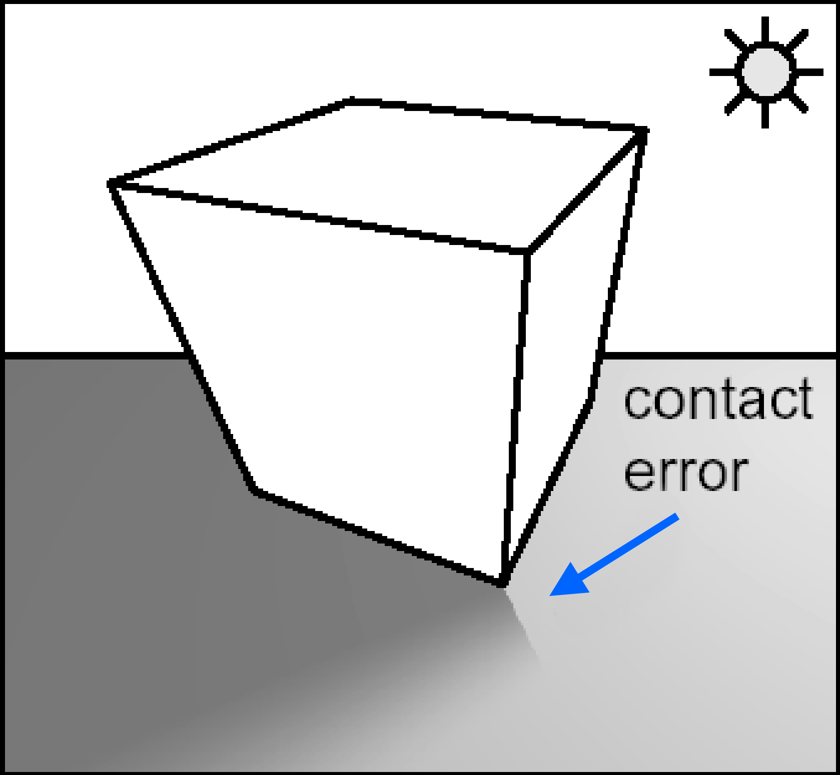
Without Alpha Remapping

SIGGRAPH2004

Linearly interpolated alpha → undesired results!



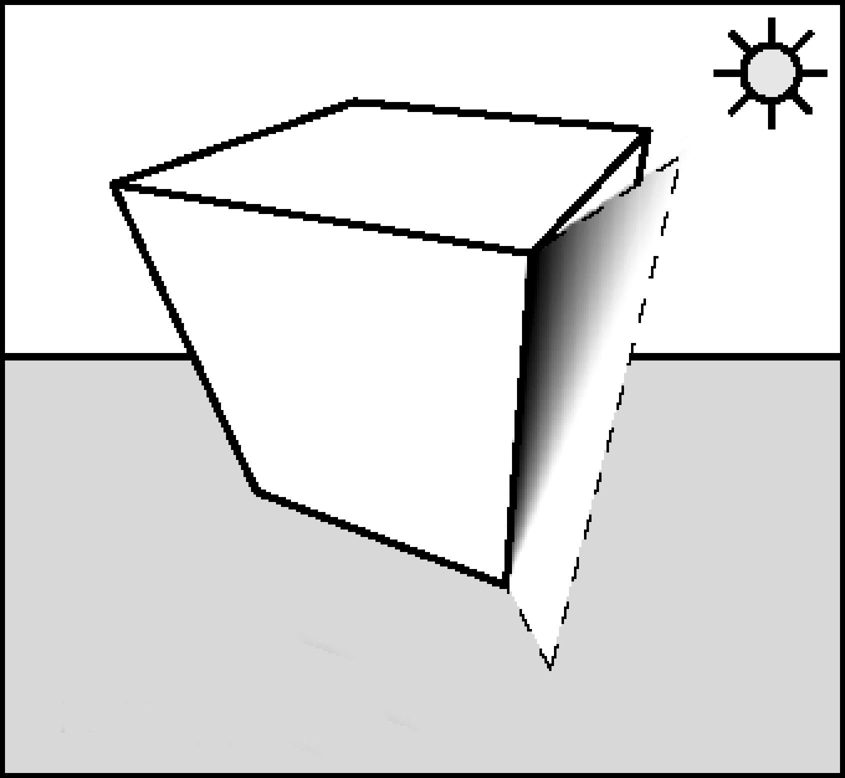
smoothie



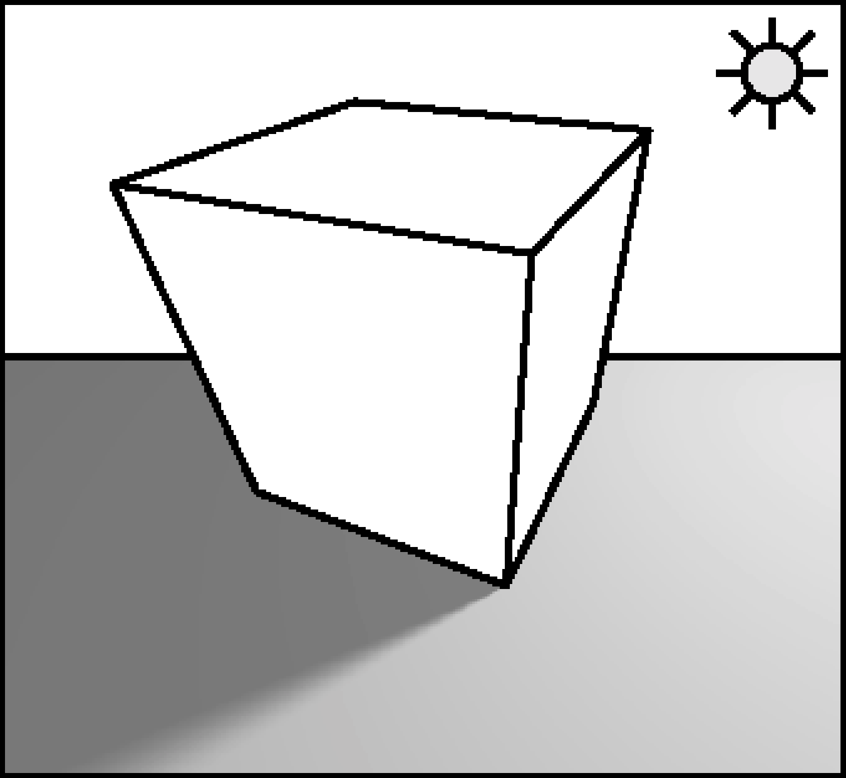
contact problem

With Alpha Remapping

Remap alpha at each pixel using ratio b/r: $\alpha' = \alpha / (1 - b/r)$



smoothie



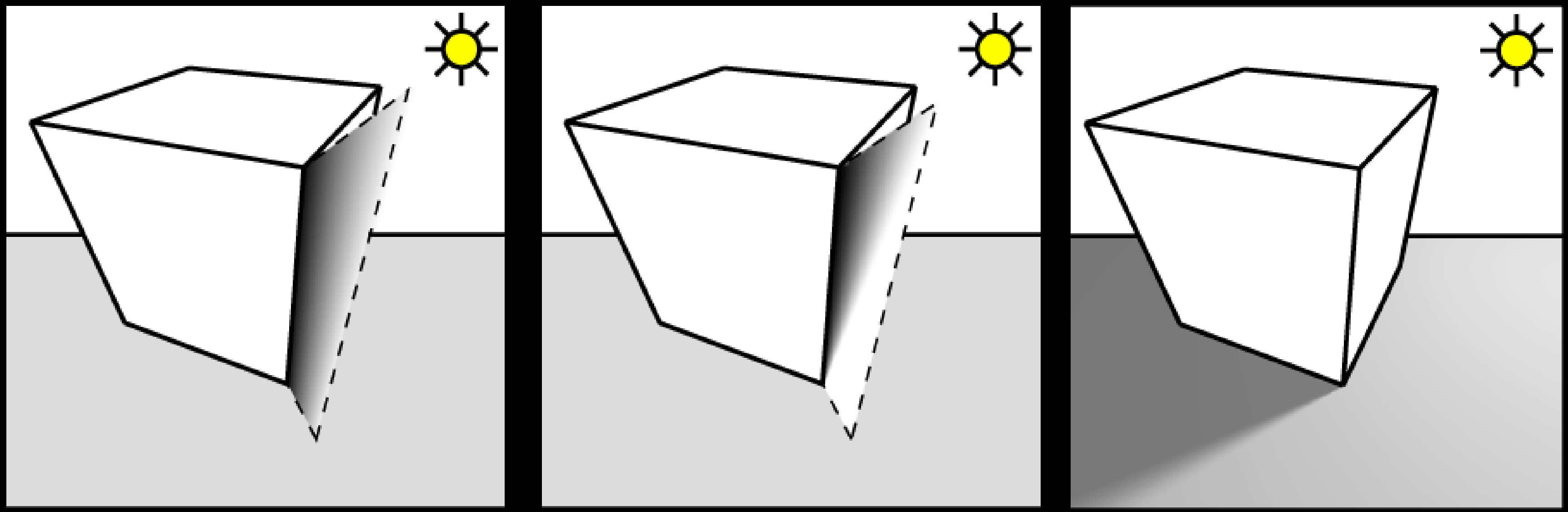
fixed contact problem

Computing Alpha Values

SIGGRAPH2004

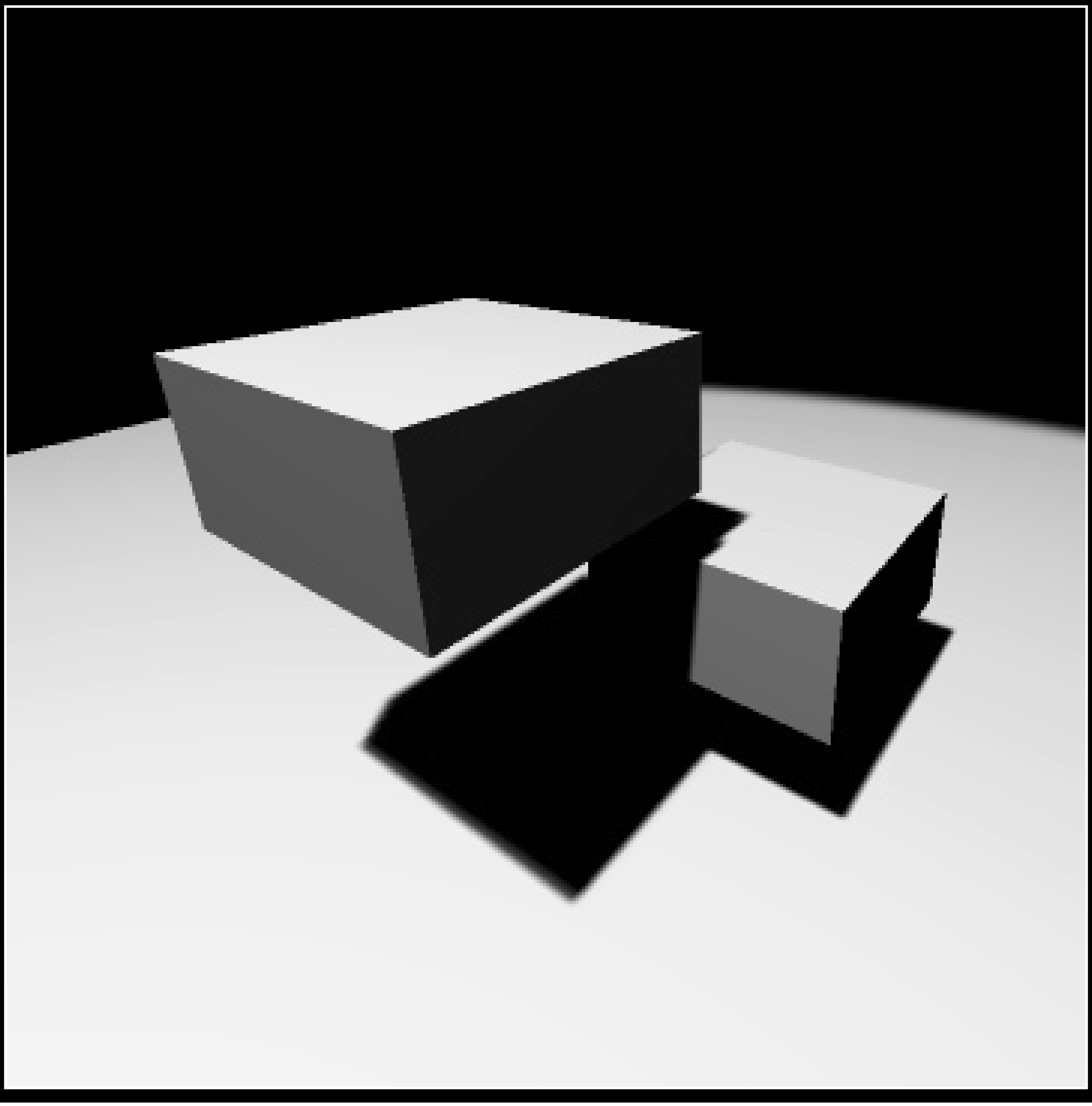
1. Linearly interpolate alpha
2. Remap alpha at each pixel using ratio b/r:

$$\boxed{\alpha' = \alpha / (1 - b/r)}$$

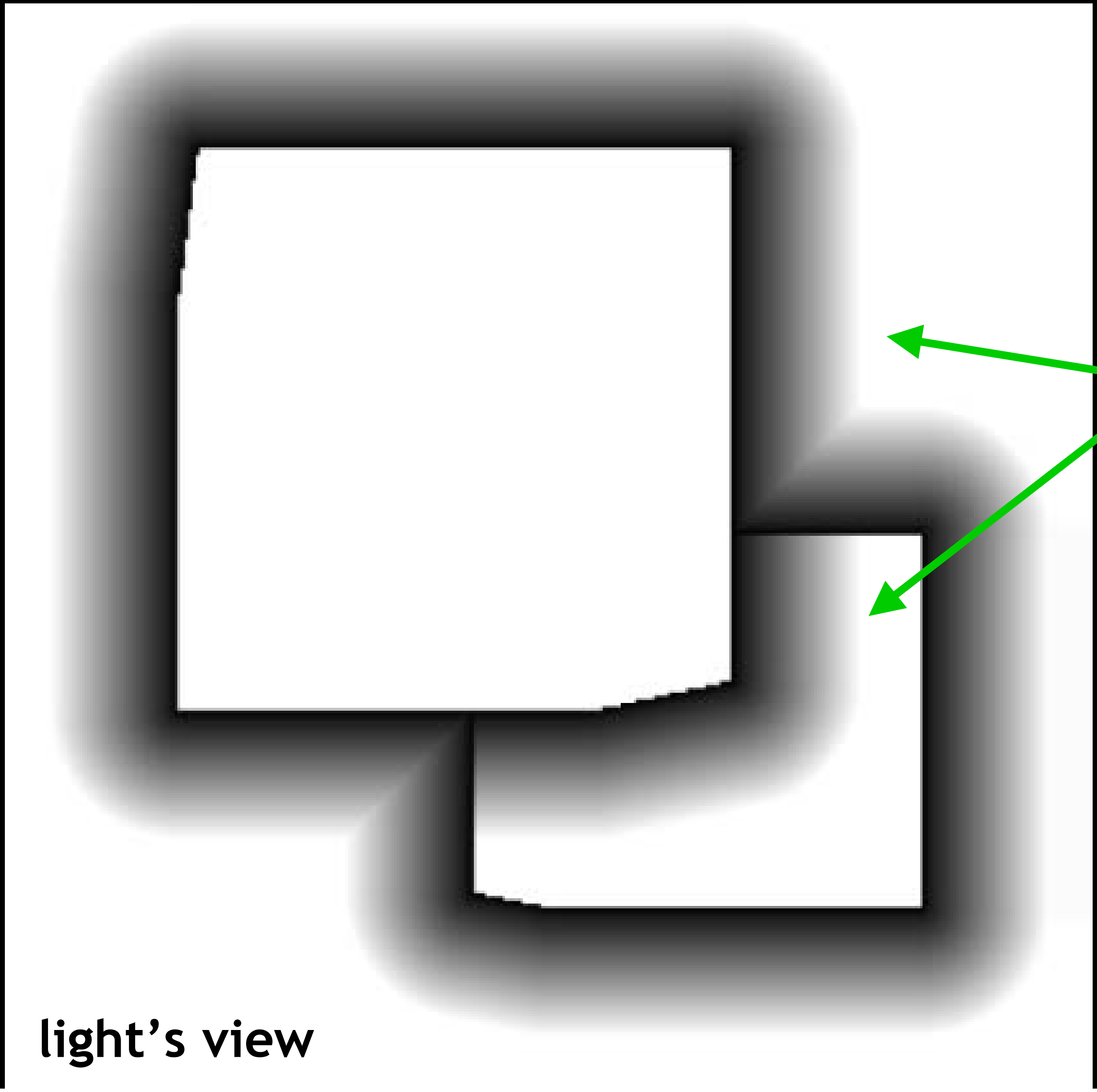


original α | remapped α | result

Multiple Objects



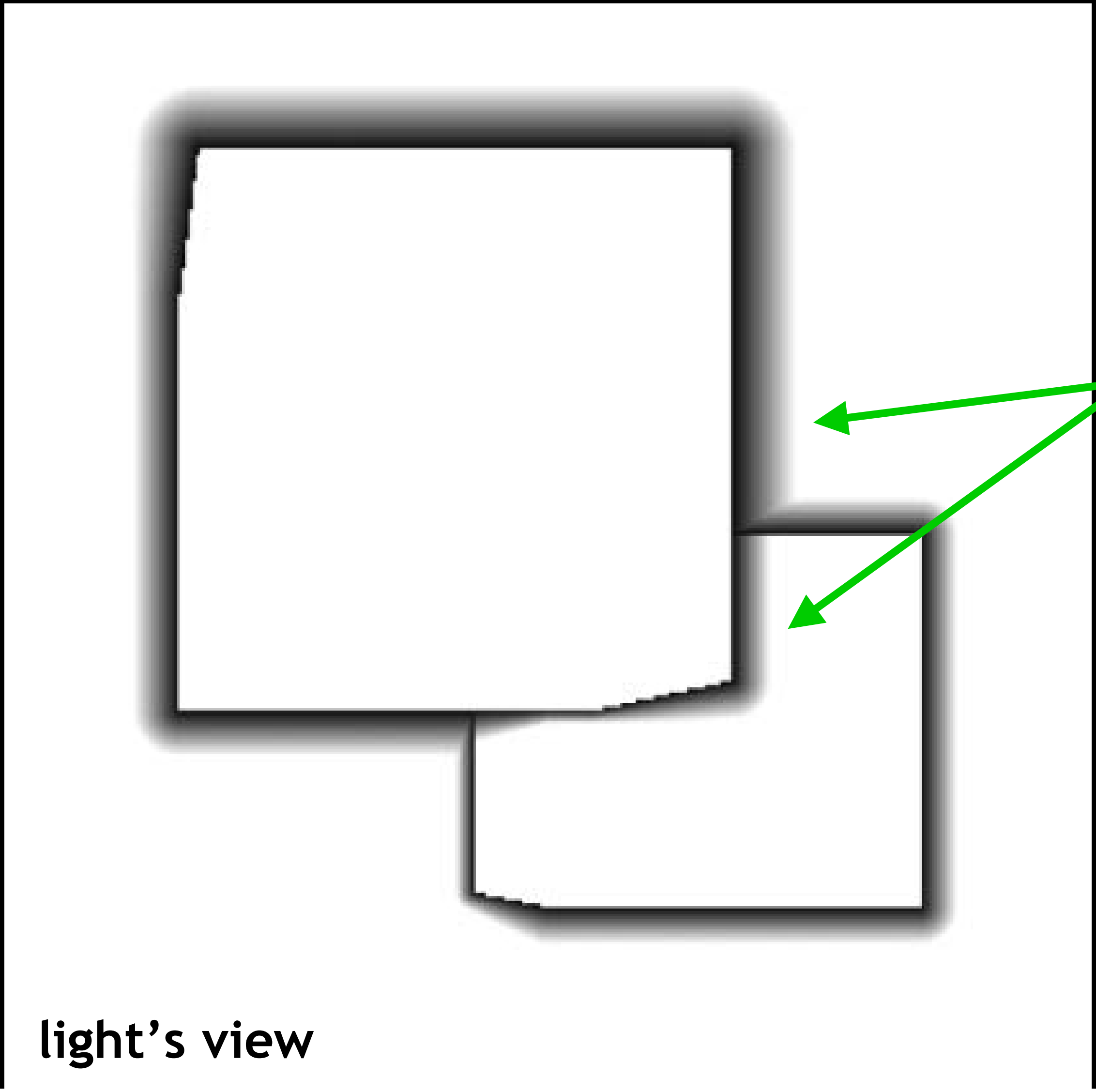
Multiple Receivers



Smoothie buffer
(linearly-interpolated α)

same thickness

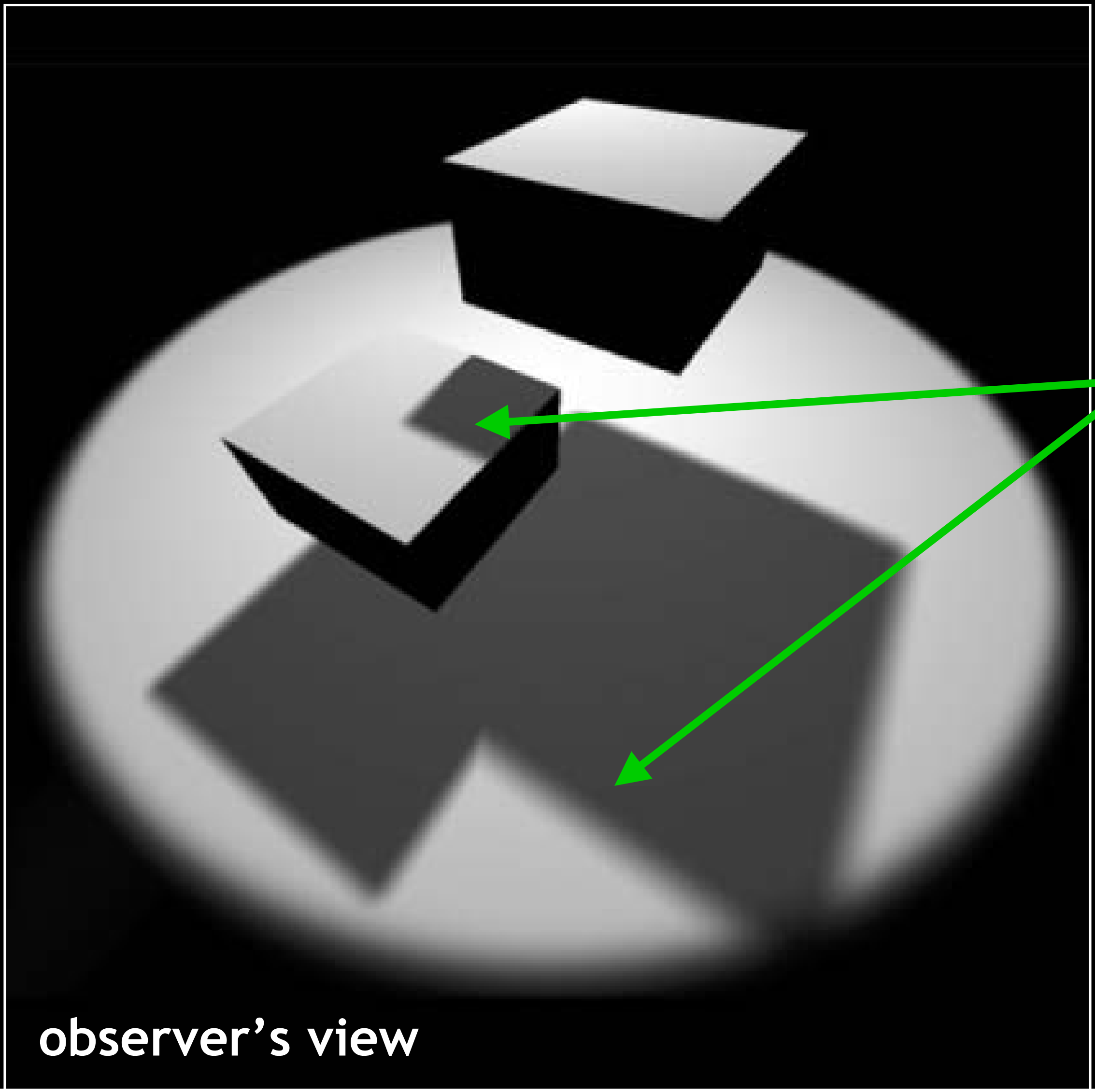
Multiple Receivers



Smoothie buffer
(remapped α)

different thickness

Multiple Receivers



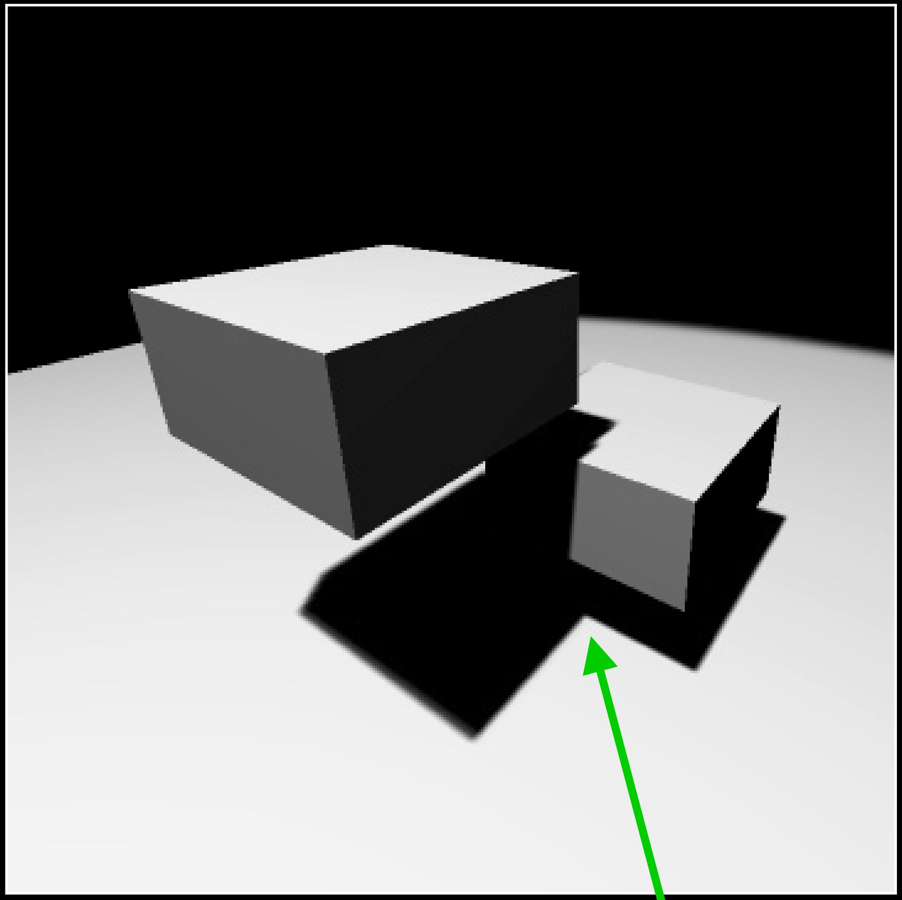
Final image

different thickness

Multiple Blockers

SIGGRAPH2004

What happens when smoothies overlap?

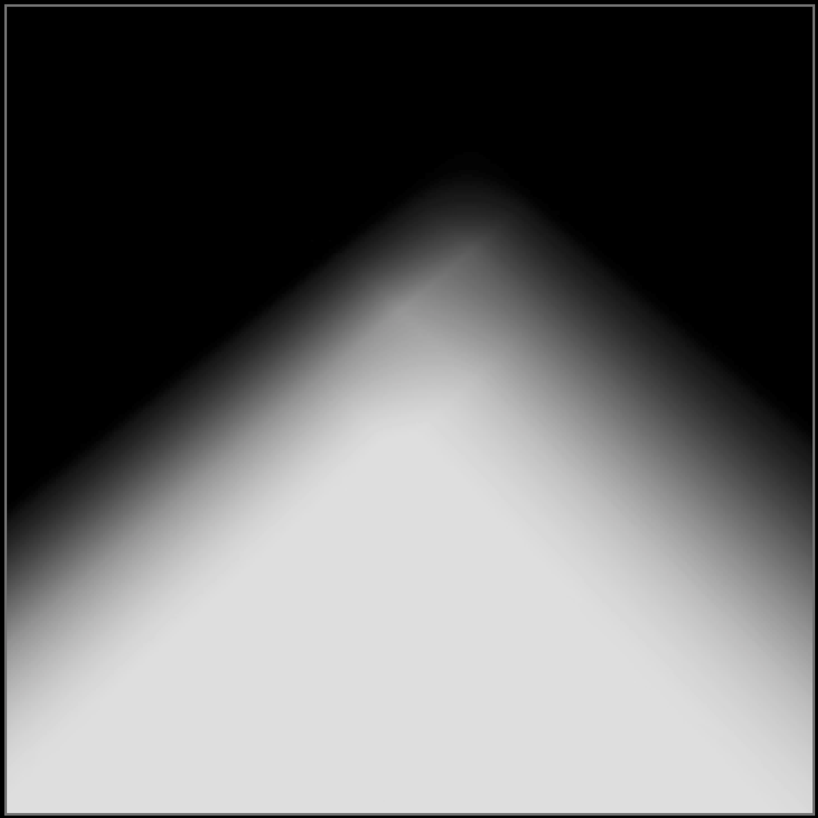


smoothie overlap

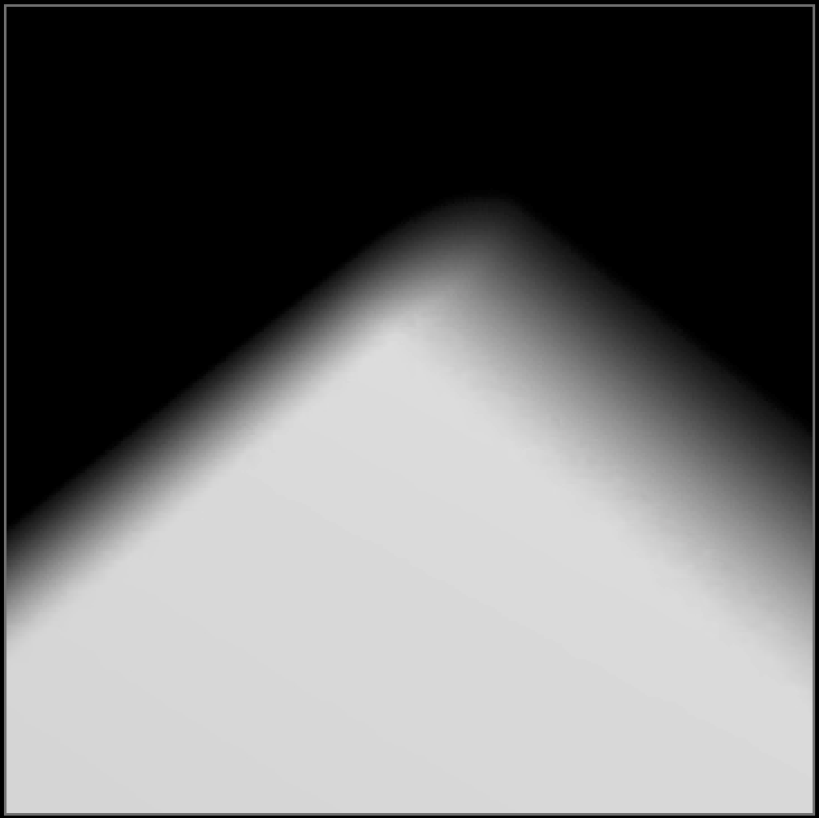
Multiple Blockers

SIGGRAPH2004

Minimum blending: just keep minimum of alpha values



smoothie



ray tracer

Implementation

Implementation

SIGGRAPH2004

- Details (OpenGL)
- Hardware acceleration
- Optimizations

Create Shadow Map

Render to standard OpenGL depth buffer

- 24-bit, window space
- Post-perspective, non-linear distribution of z

Also write to color buffer (using fragment program)

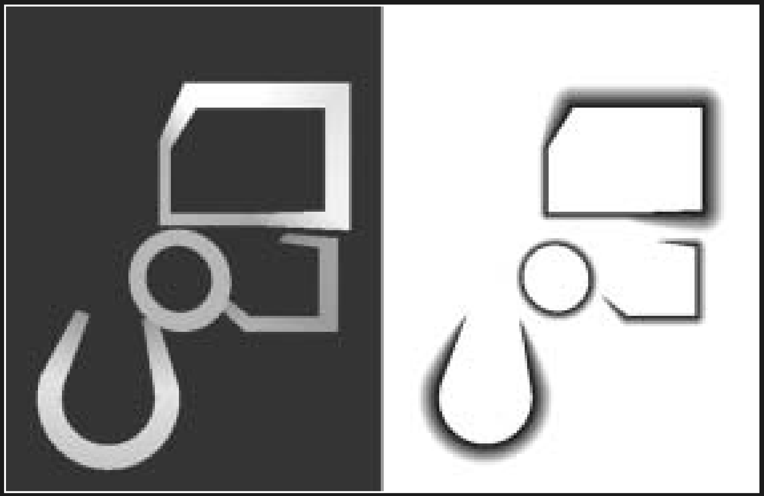
- Floating-point, eye space
- Pre-perspective, linear distribution of z
- Unlike regular shadow maps

Why? Need linear depth for next rendering pass

Create Smoothie Buffer

Conceptually, draw the smoothies once:

- store depth and alpha into a buffer



In practice, draw smoothies twice:

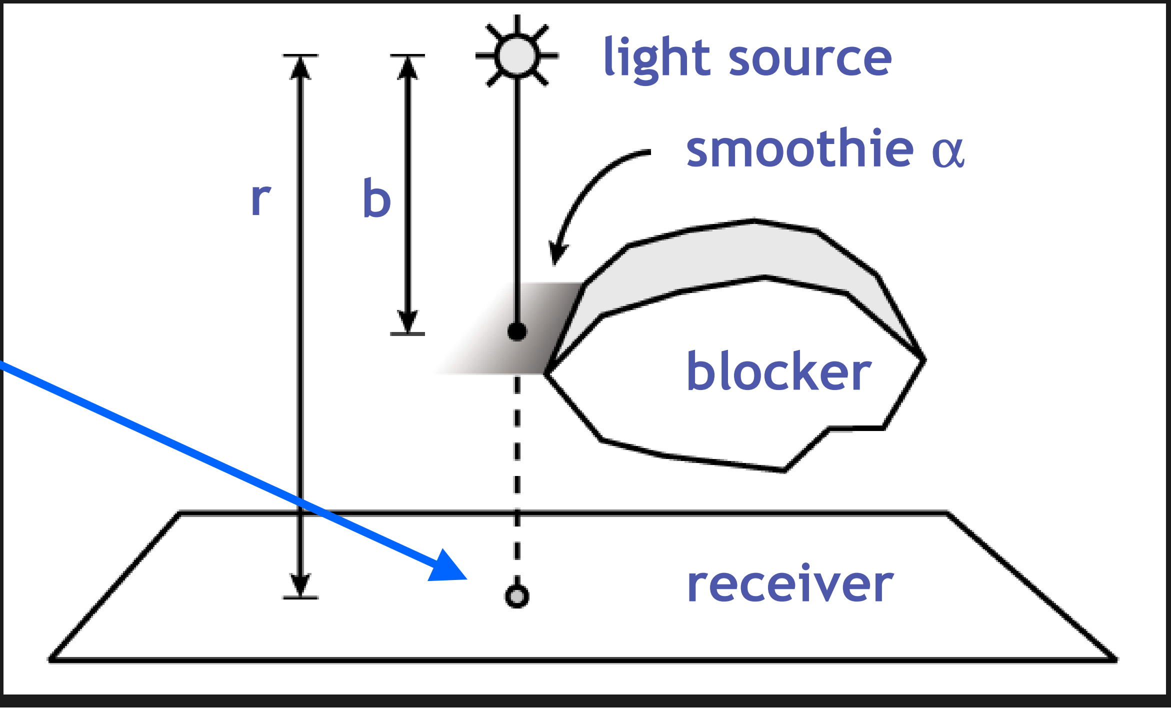
1. store nearest depth value into depth buffer
2. blend alpha values into color buffer

Computing Alpha

How to compute alpha? Recall $\boxed{\alpha' = \alpha / (1 - b/r)}$

- α is linearly interpolated from 0 to 1 across quad
- b is computed in fragment program
- r is obtained from shadow map (linear depth!)

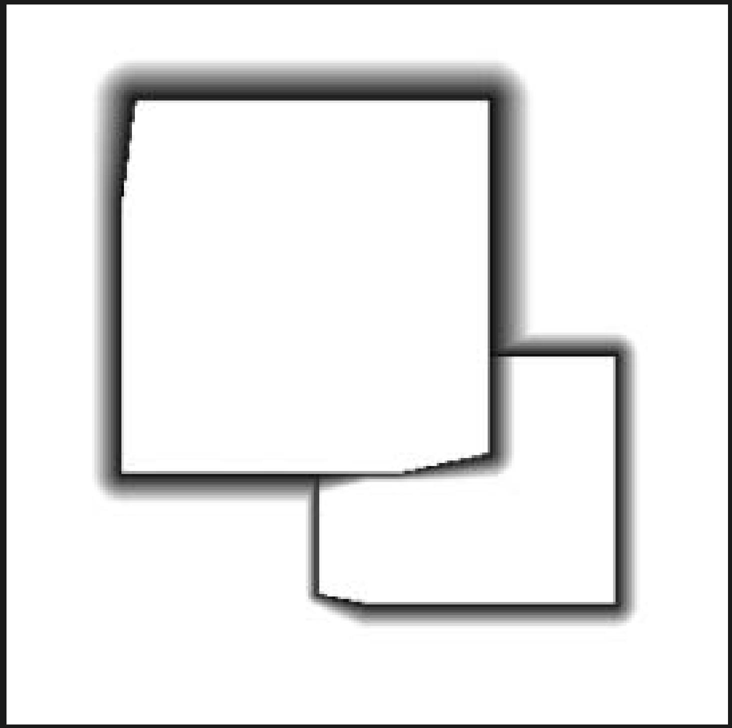
current sample



Minimum Blending

Implementation in OpenGL:

- Supported natively in hardware
- use glBlendEquationEXT(GL_MIN_EXT)



Final Rendering Pass

Implementation using fragment program:

- Project each sample into light space
- Multiple texture lookups

shadow map
(depth)

smoothie buffer
(depth)

smoothie buffer
(alpha)

Additional Details

SIGGRAPH2004

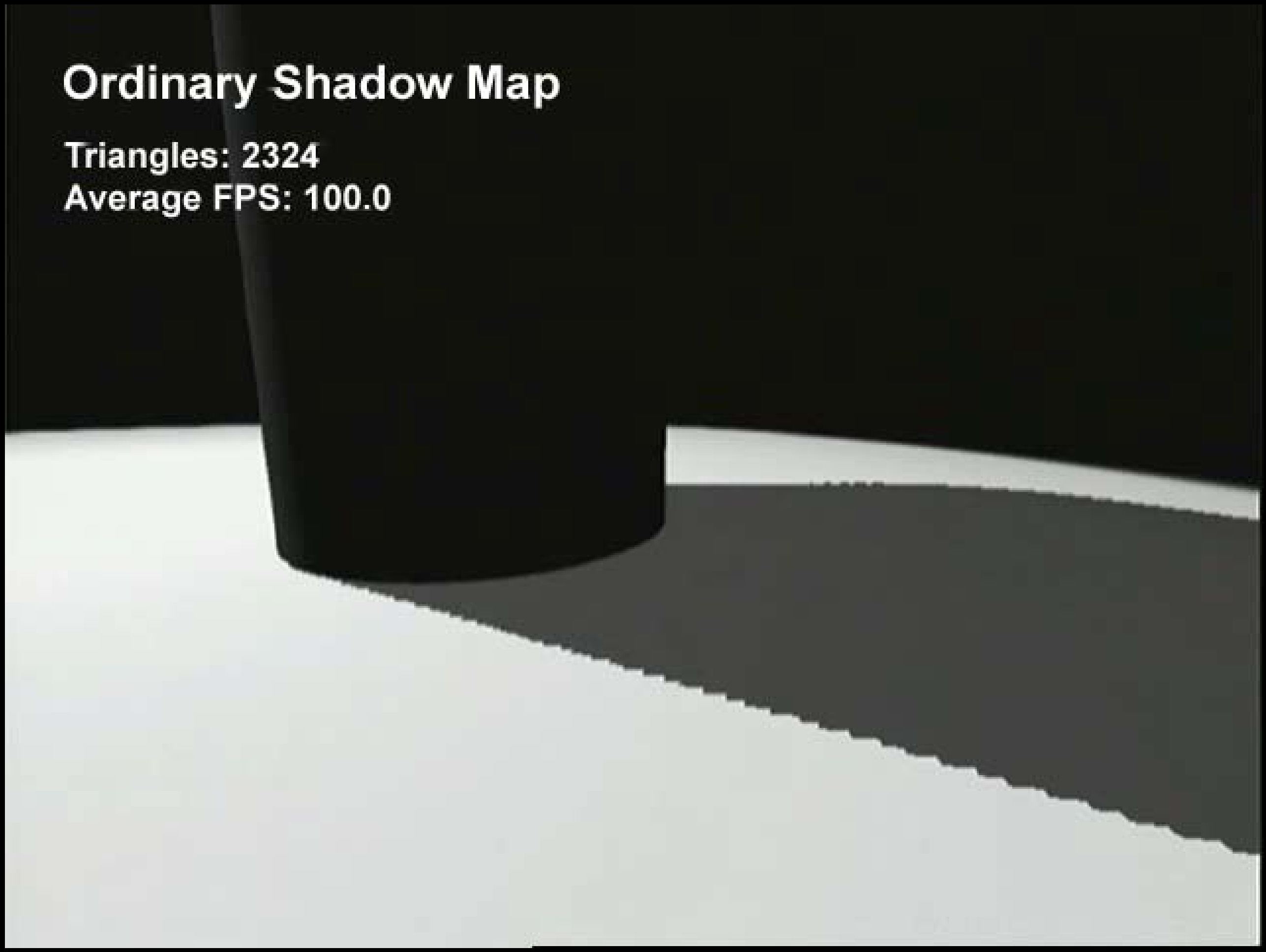
Combination of methods:

- percentage closer filtering (2 x 2 filtering in shader)
- perspective shadow maps

See paper (course notes) for Cg shader code

Examples

Video

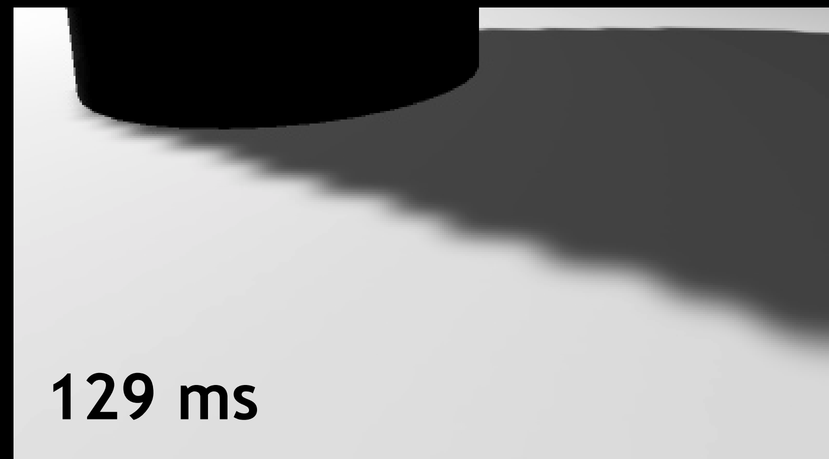


SIGGRAPH2004

Hiding Aliasing (256 x 256)

SIGGRAPH2004

shadow map



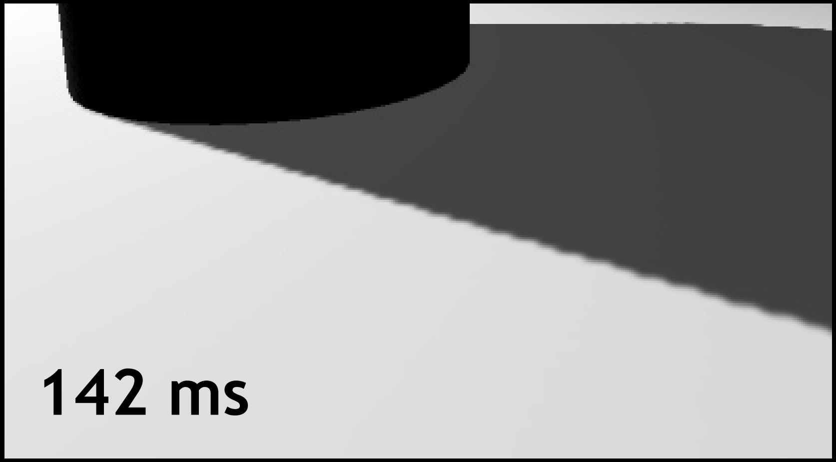
bicubic filter

smoothie (t = 0.02)

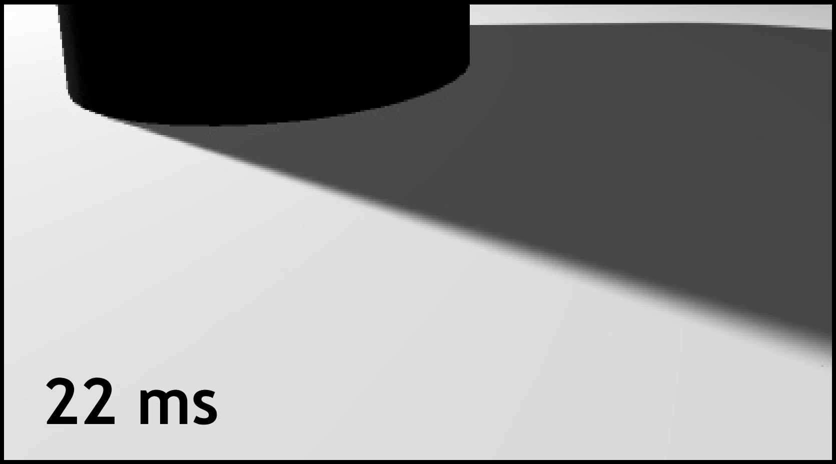
smoothie (t = 0.08)

Hiding Aliasing (1k x 1k)

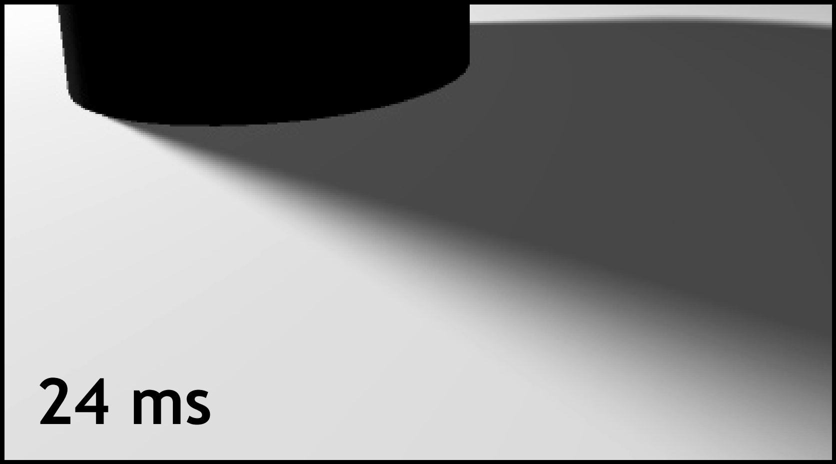
shadow map



bicubic filter



smoothie (t = 0.02)

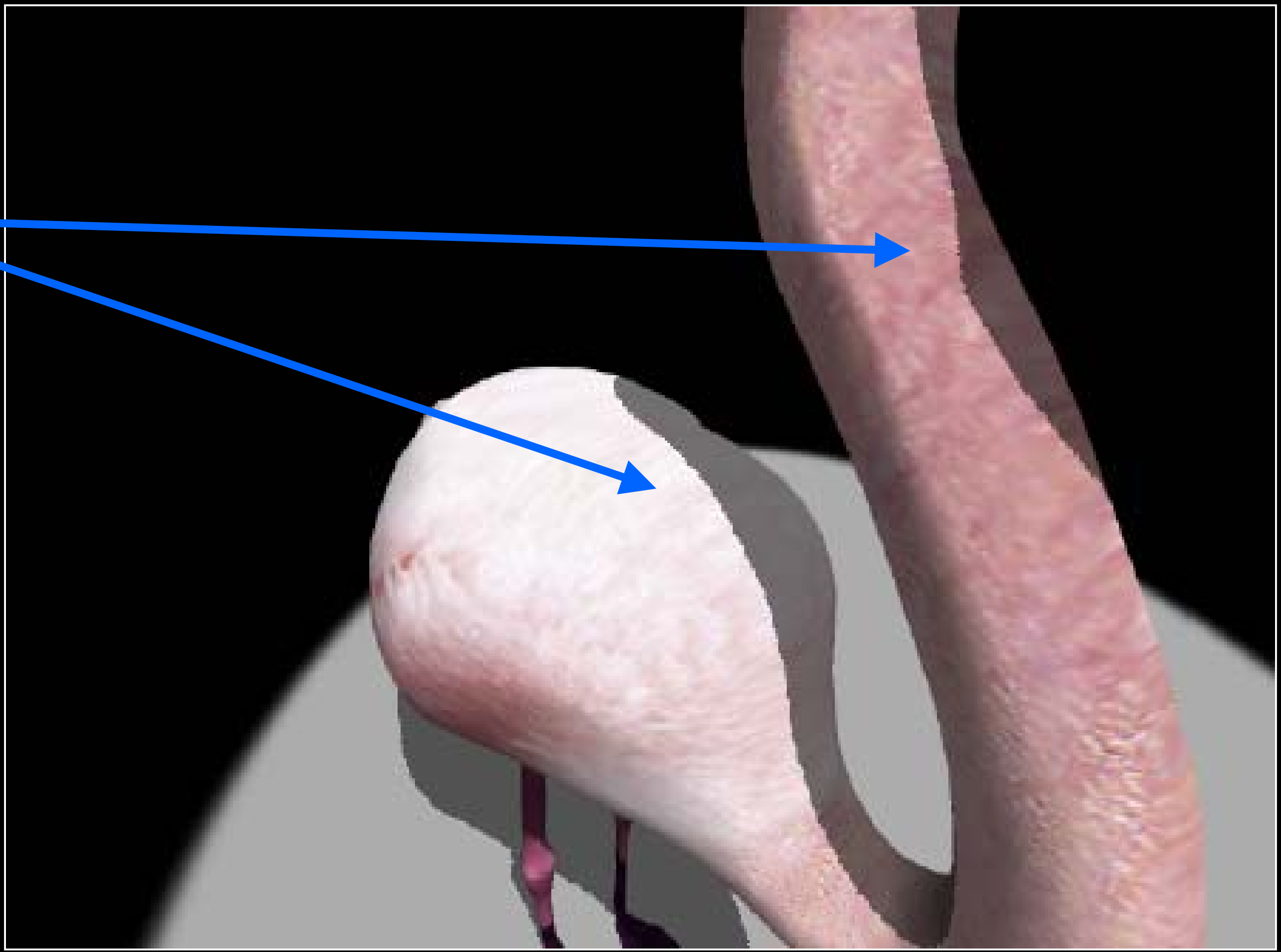


smoothie (t = 0.08)

Antialiasing Example #1

SIGGRAPH2004

hard shadows
(aliased)

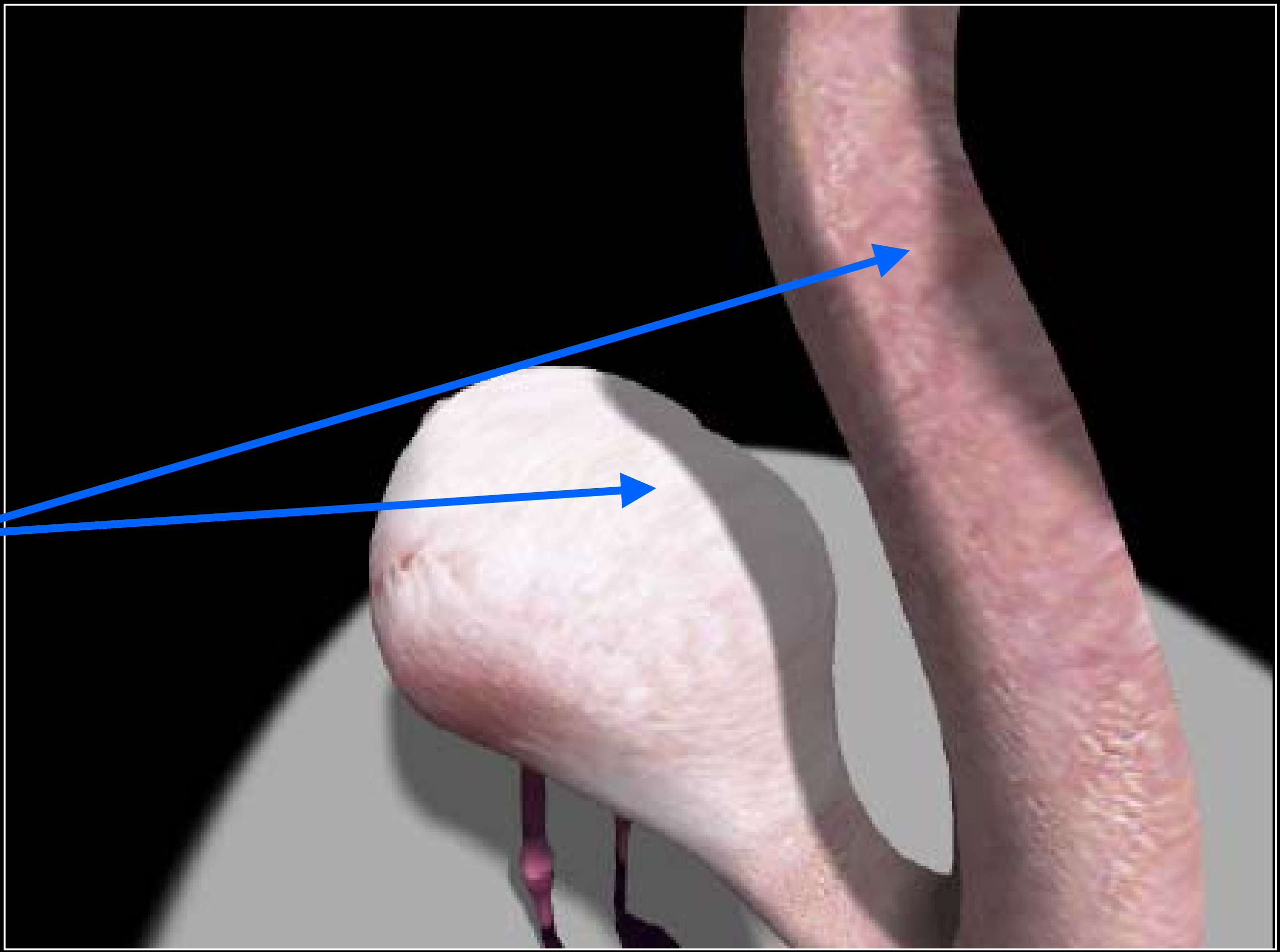


shadow map

Antialiasing Example #1

SIGGRAPH2004

soft shadows
(antialiased)

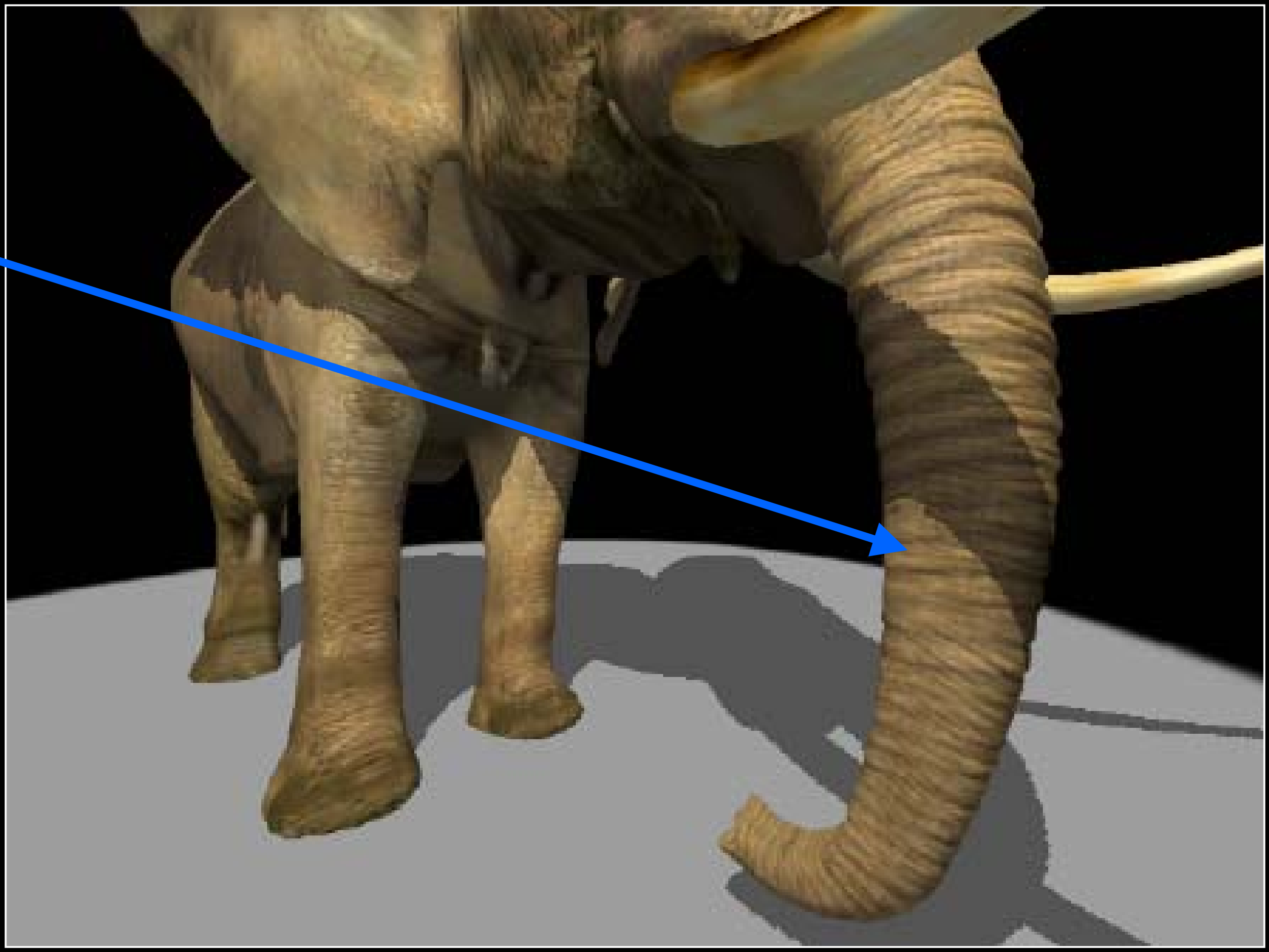


smoothies

Antialiasing Example #2

SIGGRAPH2004

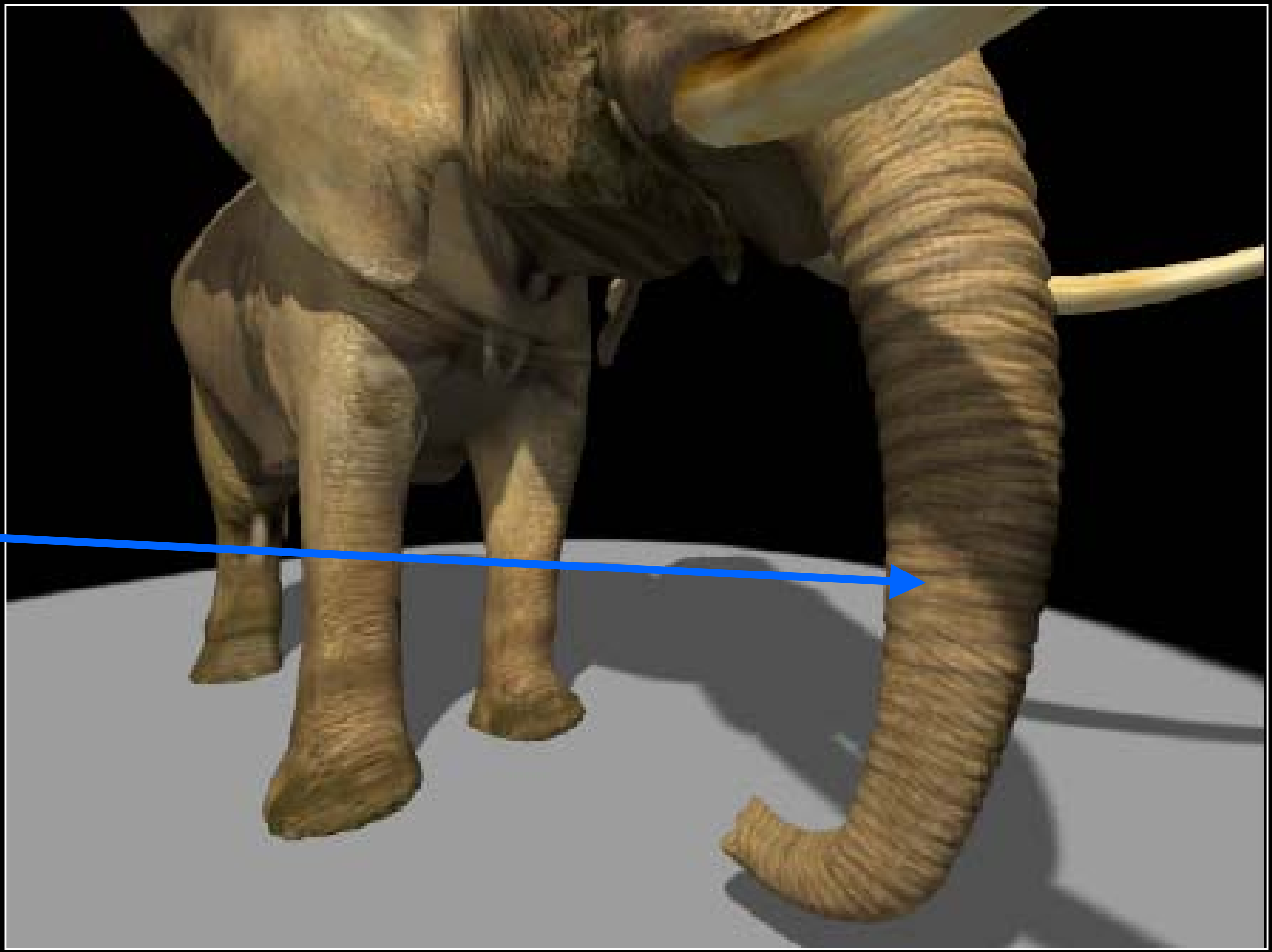
hard shadows
(aliased)



shadow map

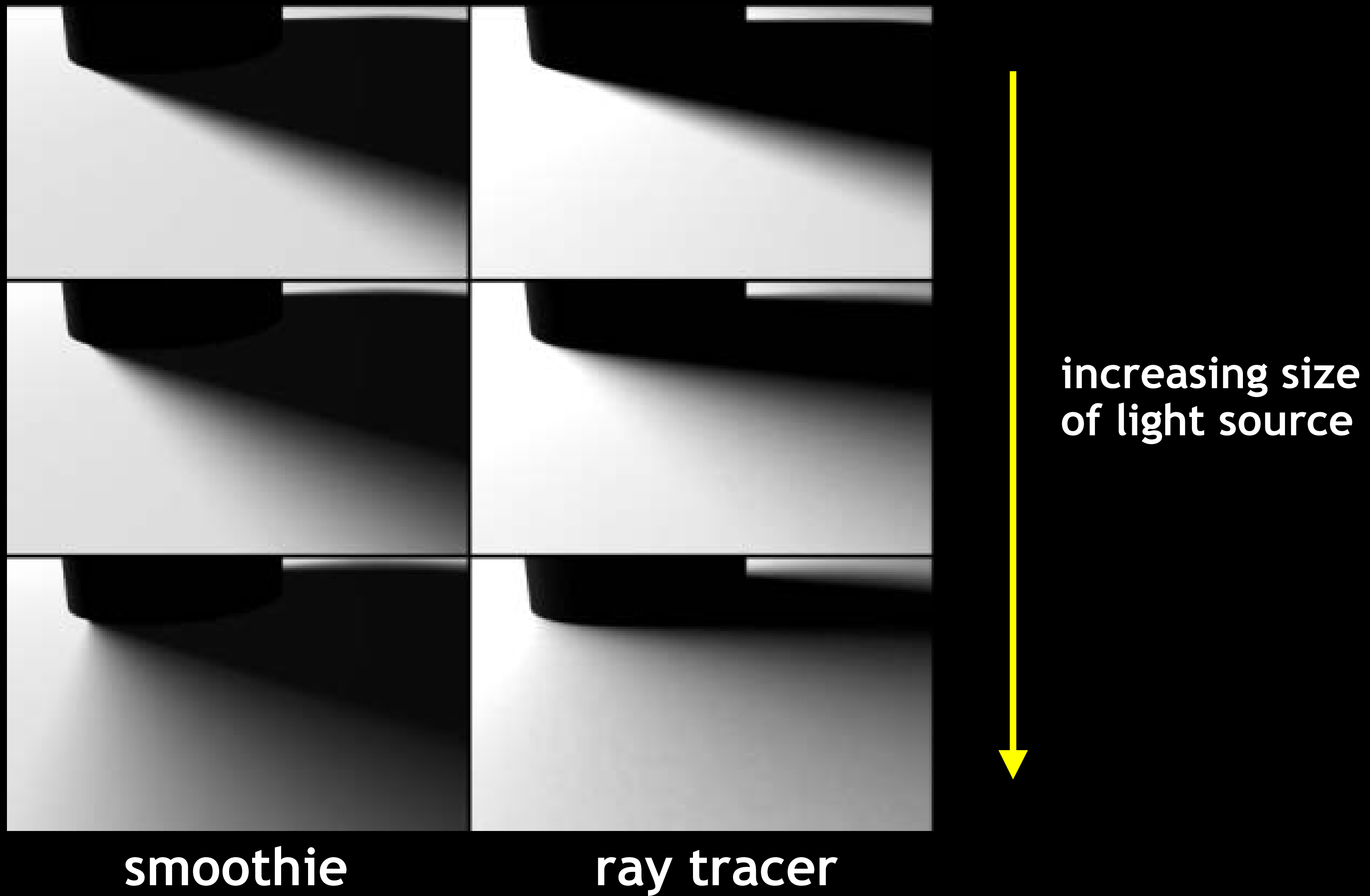
Antialiasing Example #2

soft shadows
(antialiased)



smoothies

Limitations



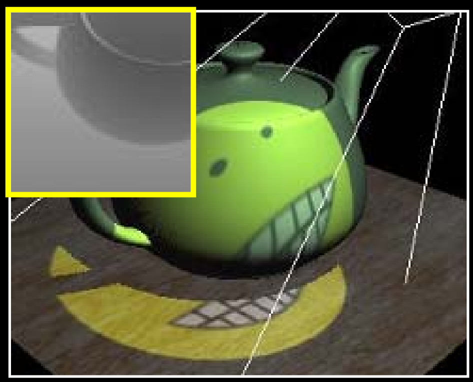
Video

original md2shader demo courtesy of Mark Kilgard

Tradeoffs

Shadow maps:

- Assumes directional light or spotlight
- Discrete buffer samples



Tradeoffs

SIGGRAPH2004

Shadow maps:

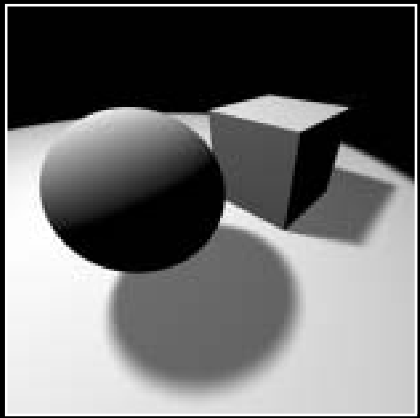
- Assumes directional light or spotlight
- Discrete buffer samples

Shadow volumes:

- Assumes blockers are closed triangle meshes
- Silhouettes identified in object space

Tradeoffs

SIGGRAPH2004



Shadow maps:

- Assumes directional light or spotlight
- Discrete buffer samples

Shadow volumes:

- Assumes blockers are closed triangle meshes
- Silhouettes identified in object space

Smoothies:

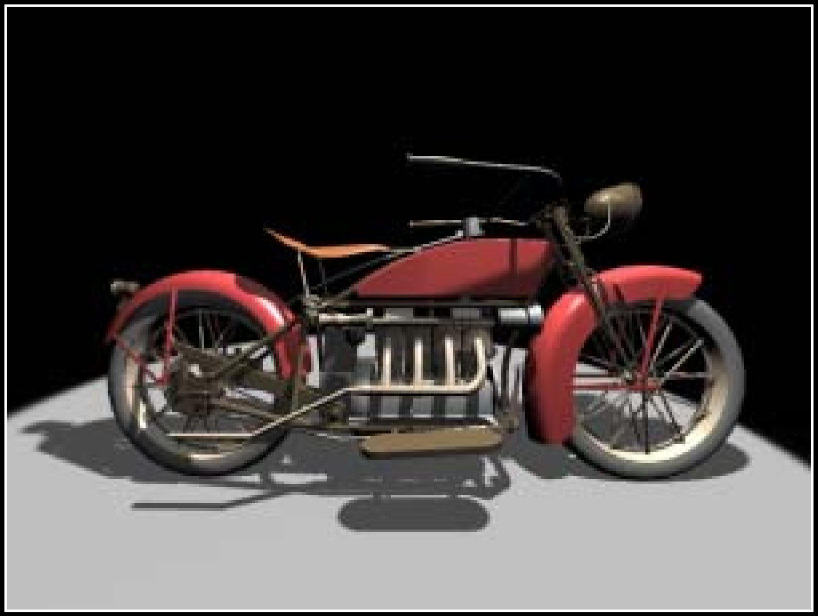
- Rendered from light’s viewpoint
- Occupy small screen area → inexpensive

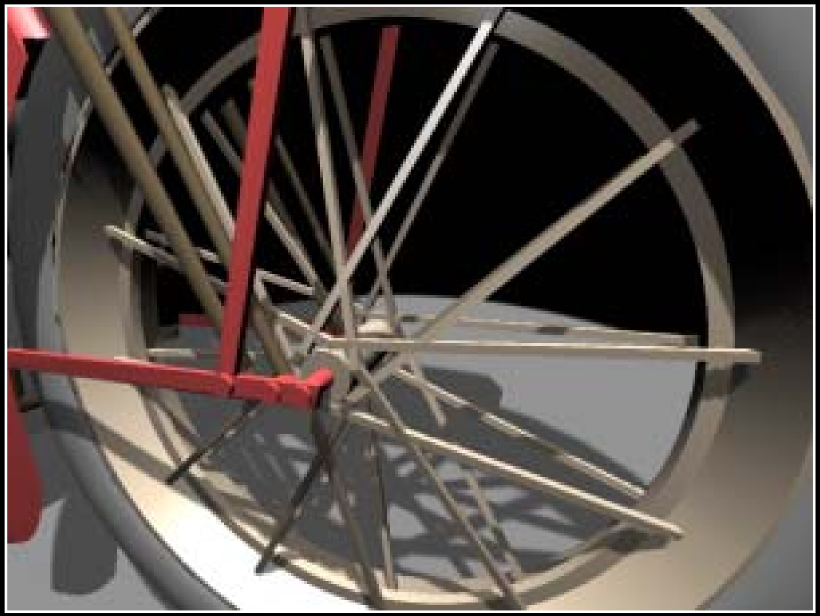
Summary

SIGGRAPH2004

Main points:

- Simple extension to shadow maps
- Shadows edges are fake, but look like soft shadows
- Fast, maps well to graphics hardware





Acknowledgments

SIGGRAPH2004

Hardware, drivers, and bug fixes

- Mark Kilgard, Cass Everitt, David Kirk, Matt Papakipos (NVIDIA)
- Michael Doggett, Evan Hart, James Percy (ATI)

Writing and code

- Sylvain Lefebvre, George Drettakis, Janet Chen, Bill Mark
- Xavier Décoret, Henrik Wann Jensen

SIGGRAPH2004