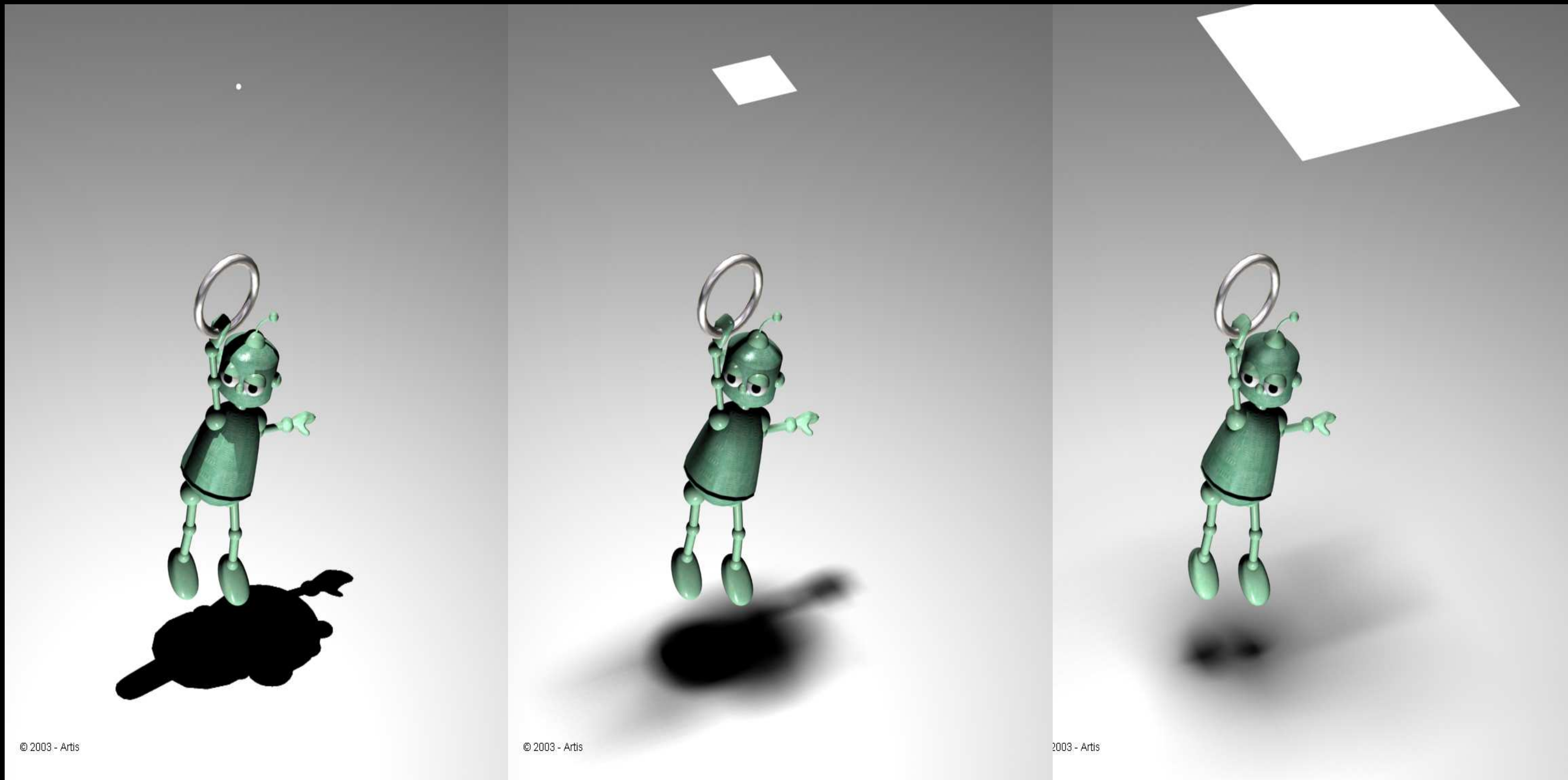
Soft Shadows

COMP 770 Computer Graphics

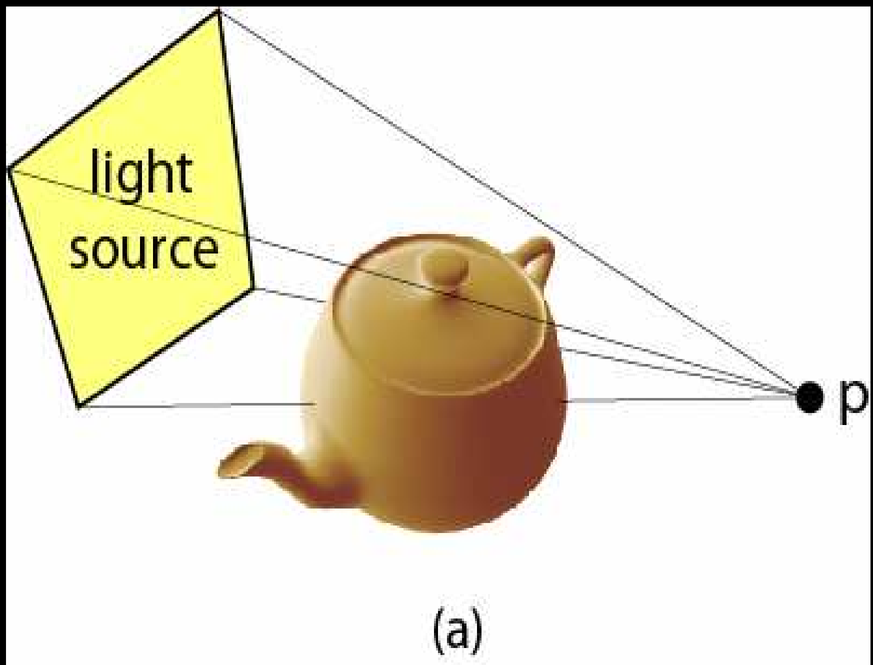
Qi Mo

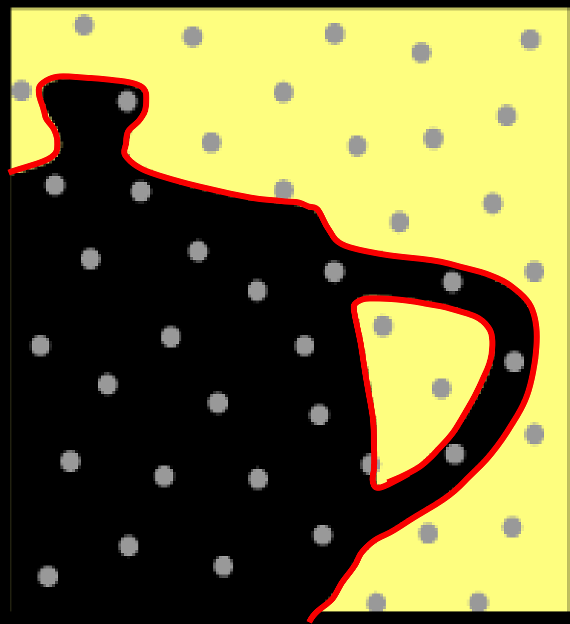
Why Soft Shadows?



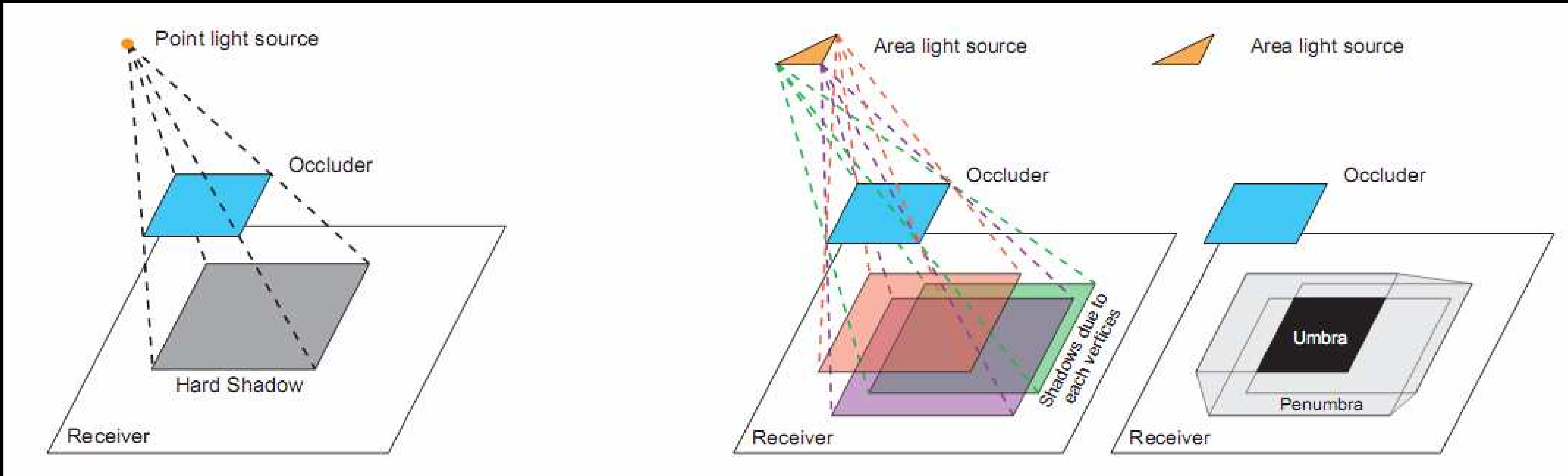
Challenge

- Visibility function
- Between light source and every point





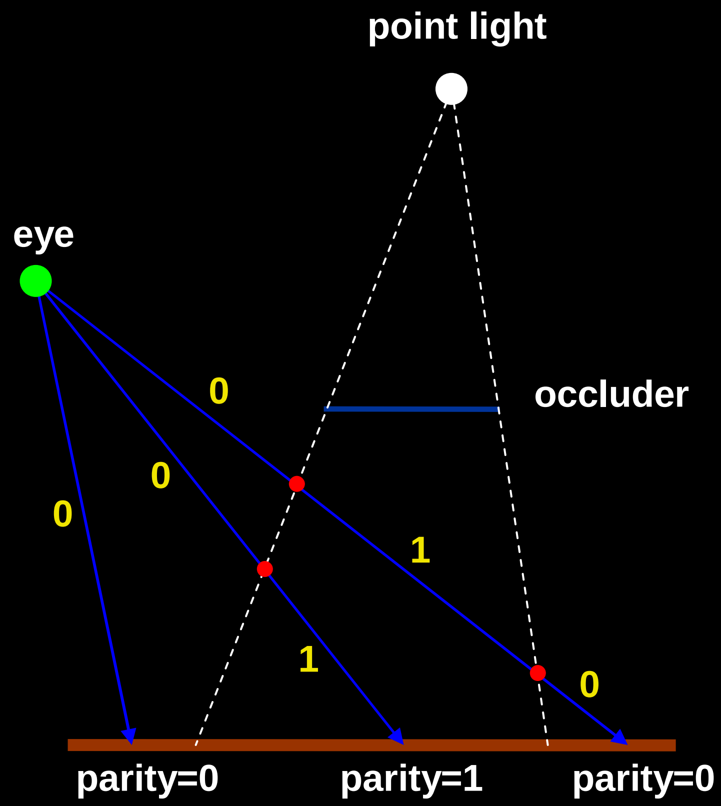
Anatomy of Soft Shadows



Previous Work

- Geometry-based methods
  - *Shadow-volume-based*
- Image-based methods
  - *Shadow-map-based*

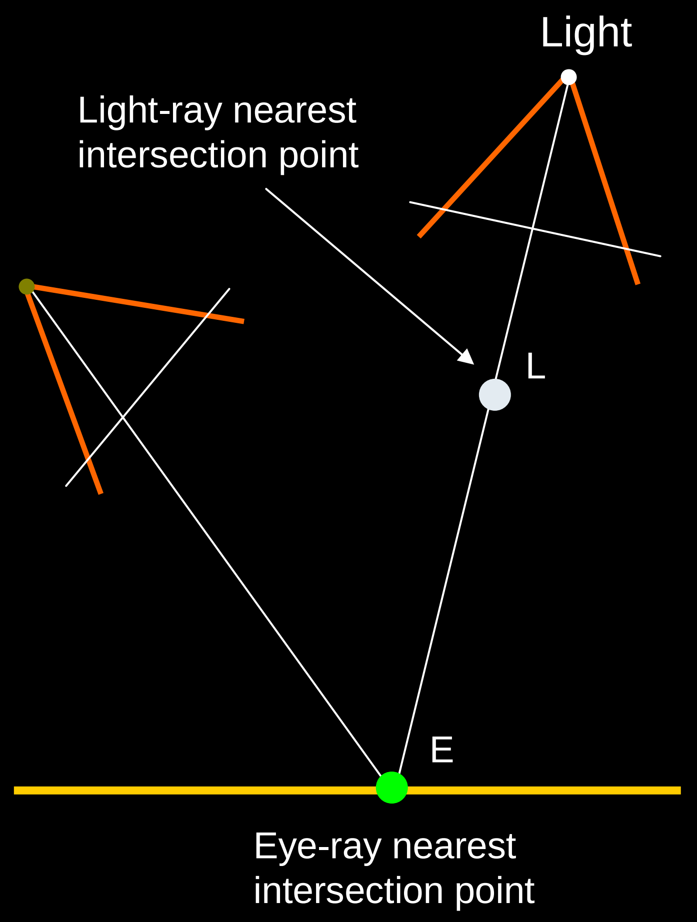
Review: Shadow Volumes



Pros and Cons

- Pros
  - No aliasing
  - Unlimited light field of view
- Cons
  - Poor scalability with scene complexity
  - Fill-rate limited
  - Polygonal models only

Review: Shadow Maps



Pros and Cons

- Pros
  - Efficiency and scalability
  - Polygons, parameterized surfaces, alpha textures, etc.
- Cons
  - Shadow aliasing and acne
  - No omni-directional light

Methods to cover

- Distributed ray tracing soft shadows

- Penumbra wedges
- Soft shadow volumes

- Soft shadow mapping by backprojection

Methods to cover

- Distributed ray tracing soft shadows

- Penumbra wedges
- Soft shadow volumes

- Soft shadow mapping by backprojection

distribution ray tracing

distribution ray tracing

use many rays to compute average values over pixel areas, time, area lights, reflected directions, …

antialiasing origin

- compute average color subtended by a pixel

pixel center $C = \mathrm{raytrace}(\mathbf{E}, \mathbf{Q} - \mathbf{E})$

pixel average $C = \frac{1}{A_P} \cdot \int_{\mathbf{Q} \in P} \mathrm{raytrace}(\mathbf{E}, \mathbf{Q} - \mathbf{E}) \cdot dA_{\mathbf{Q}}$

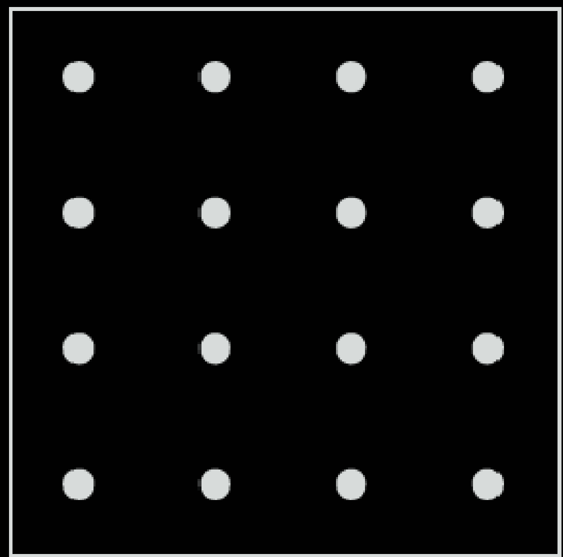
$\mathbf{E}$: camera origin

$\mathbf{Q}$: point on image plane

P : pixel of area A_P

antialiasing by deterministic integration

- subdivide the pixel in squares
- cast rays through squares centers
- average result

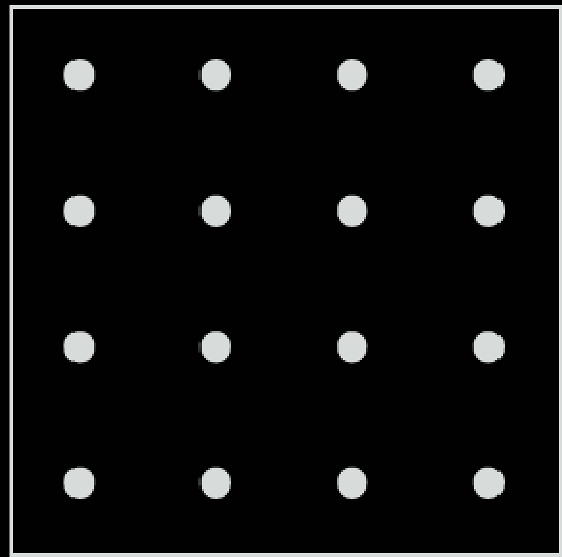


[Shirley]

deterministic antialiasing pseudocode

- antialising pixel (i, j)

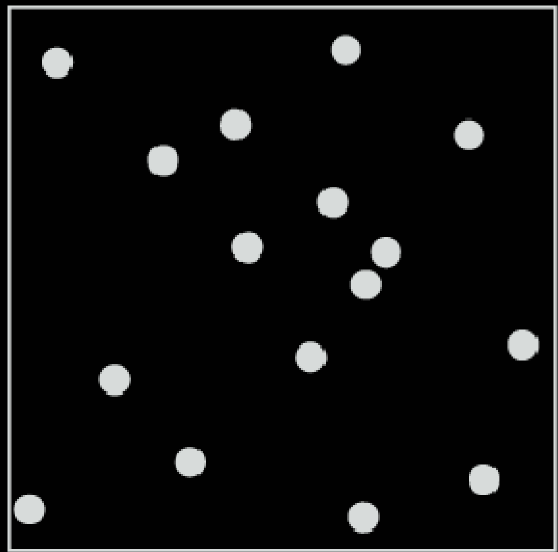
```
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    u = (i + (sx+0.5)/ns) / width
    v = (j + (sj+0.5)/ns) / height
    Q = imagePlanePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns²
```



[Shirley]

antialiasing by Monte Carlo estimation

- pick random points in pixel area
- cast rays through them
- average result

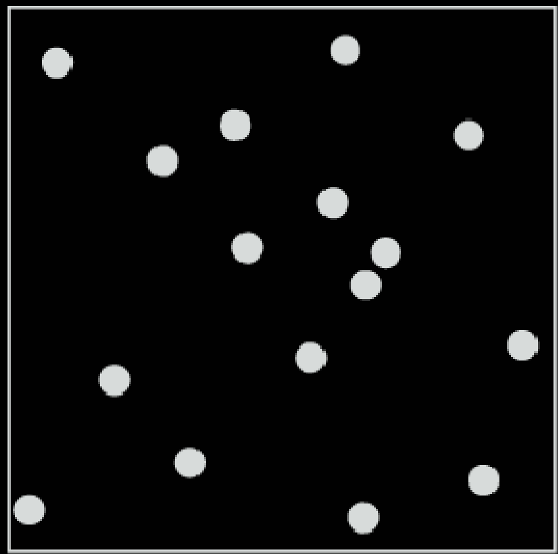


[Shirley]

Monte Carlo antialiasing pseudocode

- antialiasing pixel (i, j)

```
c = 0
for s = 0 to ns²
    (rx,ry) = random2d();
    u = (i + rx) / width
    v = (j + ry) / height
    Q = imagePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns²
```

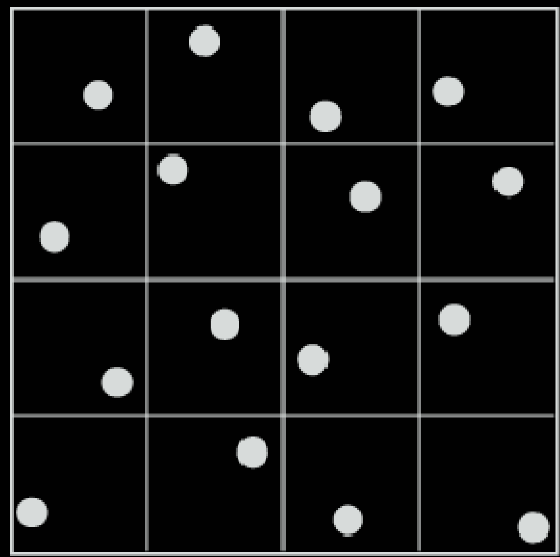


[Shirley]

Monte Carlo antialiasing pseudocode

- antialiasing pixel (i, j)

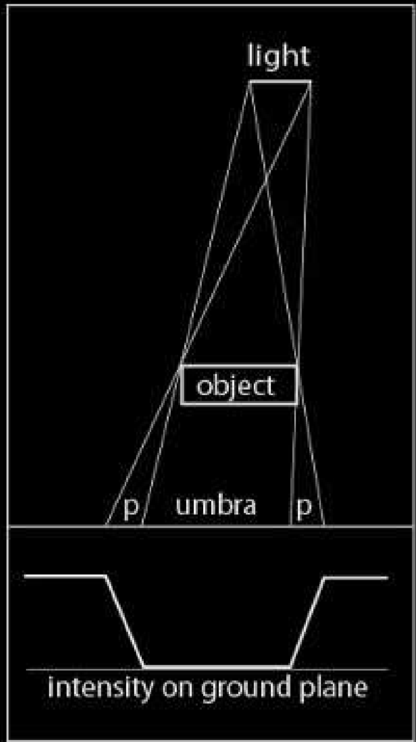
```
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    (rx,ry) = random2d();
    u = (i + (sx+rx)/ns) / width
    v = (j + (sy+ry)/ns) / height
    Q = imagePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns²
```



[Shirley]

soft shadows origin

- area lights create penumbras
  - light is only partially visible from a given point
  - want to compute how much light hits the point



[Shirley]

approximate soft shadows principle

point light $$C = C_l \cdot V(\mathbf{P},\mathbf{S}) \cdot \text{shading}(\mathbf{P},\mathbf{S})$$

area light $$C = \frac{C_l}{A_L} \cdot \int_{\mathrm{S} \in L} V(\mathbf{P},\mathbf{S}) \cdot \text{shading}(\mathbf{P},\mathbf{S}) \cdot dA_{\mathrm{S}}$$

$\mathbf{P}$: point on the surface

$\mathbf{S}$: point on the light

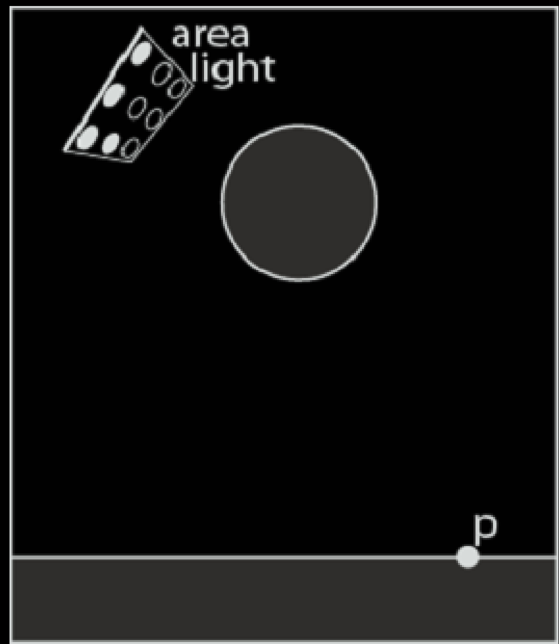
V : visibility function (0 or 1)

L : light of area A_L

C_l : total light intensity

soft shadows by deterministic integration

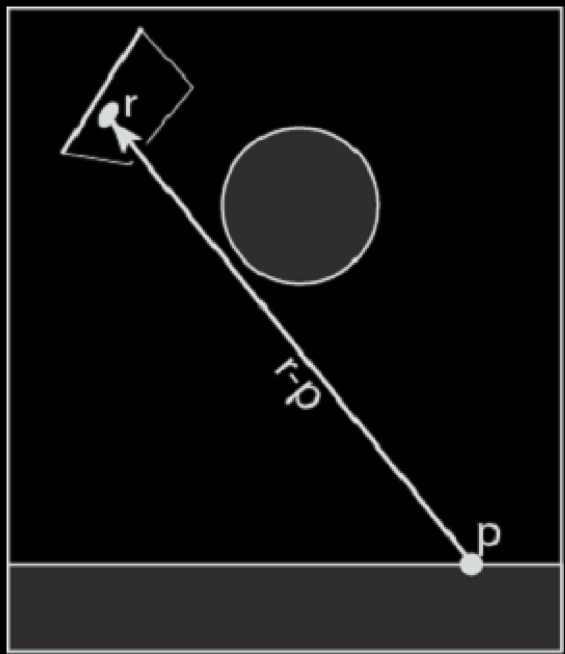
- approximate area light as a set of point lights
  - equivalent to quadrature rule
- for each point, compute shadows and lighting
- average results



[Shirley]

soft shadows by Monte Carlo estimation

- use Monte Carlo integration
- pick random points on the light
  - easy for quad lights, hard (but possible) for others
- compute shadows and lighting
- average results



[Shirley]

soft shadows by Monte Carlo estimation

for quads

$$\mathbf{S}_i = \mathbf{S}_c + (0.5 - r_{i,1})l\mathbf{u} + (0.5 - r_{i,2})l\mathbf{v}$$

$$\langle C \rangle = \frac{C_l}{N} \cdot \sum_i^N V(\mathbf{P}, \mathbf{S}_i) \cdot \text{shading}(\mathbf{P}, \mathbf{S}_i)$$

$\mathbf{S}_c$: light source center

$\mathbf{u}, \mathbf{v}$: light source tangent vectors

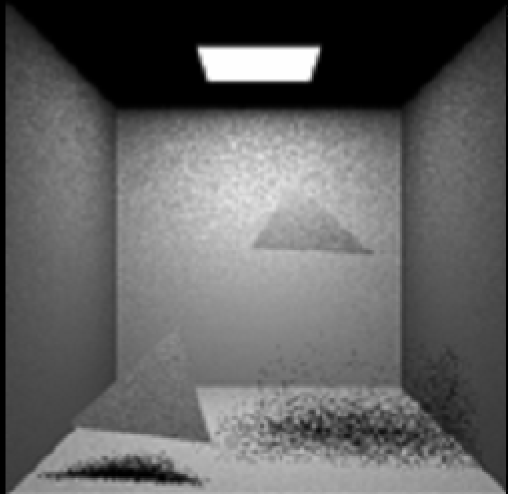
l : light source size

$\mathbf{r}_i$: uniformly sampled random 2d vector in $[0,1]^2$

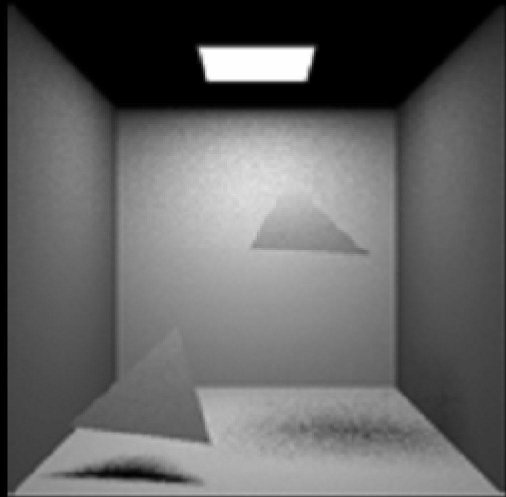
N : total number of samples

how many samples?

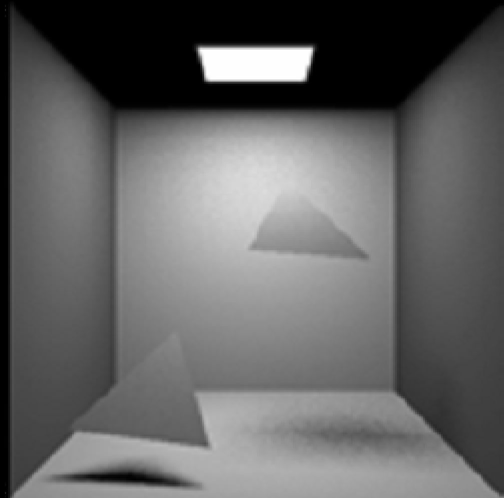
1 sample



9 samples



36 samples



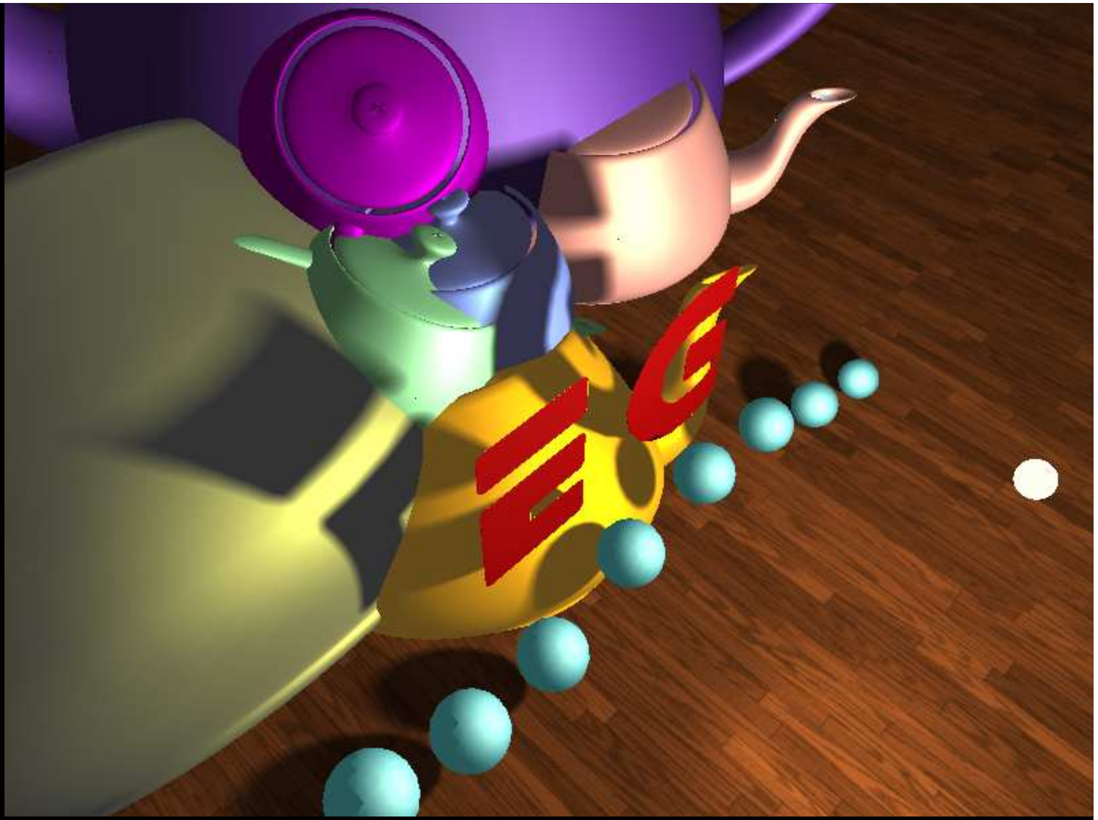
100 samples

[Bala]

Methods to cover

- Distributed ray tracing soft shadows
- Penumbra wedges
- Soft shadow volumes
- Soft shadow mapping by backprojection

Penumbra Wedges



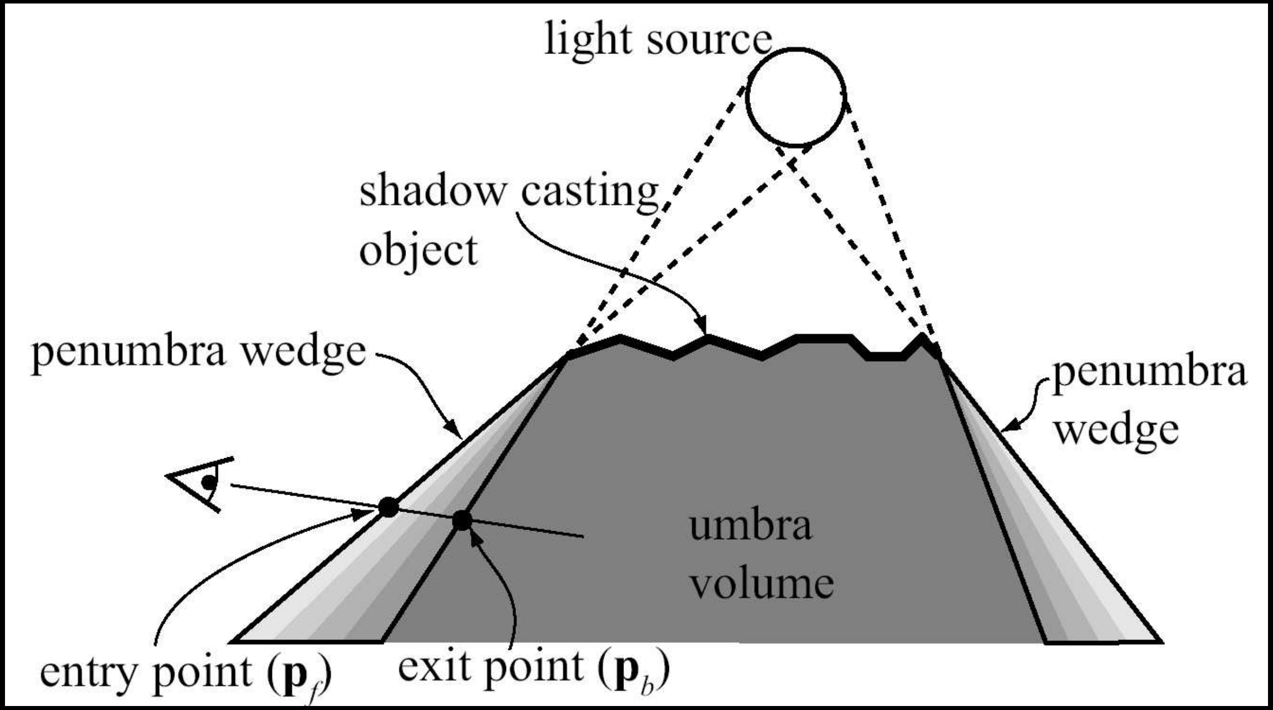
Tomas Akenine-Möller
Ulf Assarsson
Department of Computer Engineering,
Chalmers University of Technology
Sweden

Idea

- Extend the shadow volume algorithm
- In the shadow volume algorithm

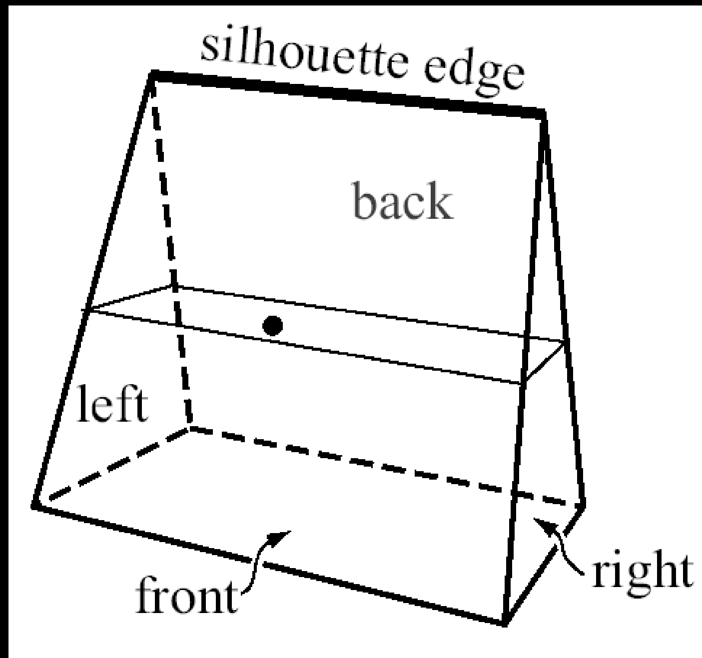
  each silhouette edge → shadow quad
- For soft shadows, instead

  each edge → penumbra wedge



In 3D, ...

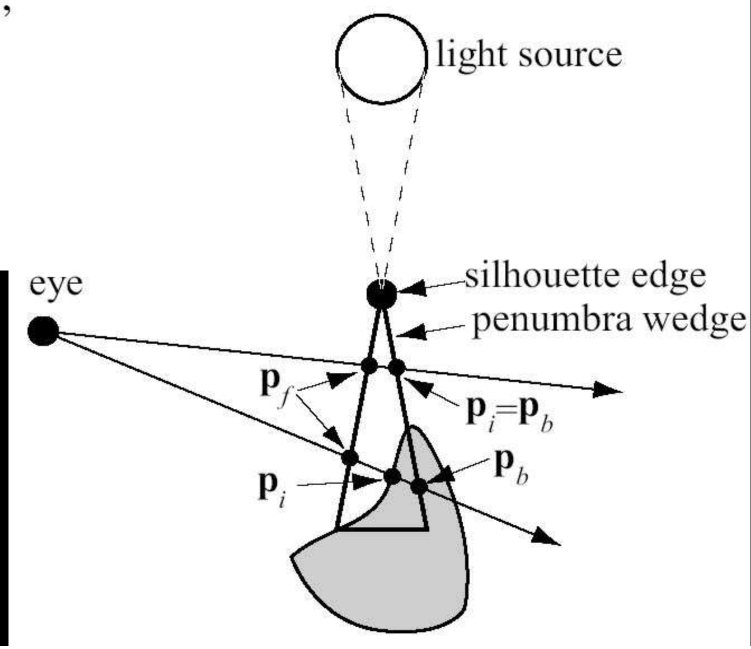
- Simplifications:
  - Spherical light sources
  - Only use silhouette as seen from center of light source
  - Bound the penumbra volume with 4 planes, sharing a silhouette edge
- Also, use a hires stencil buffer (we use 16 bits) – called light intensity buffer here



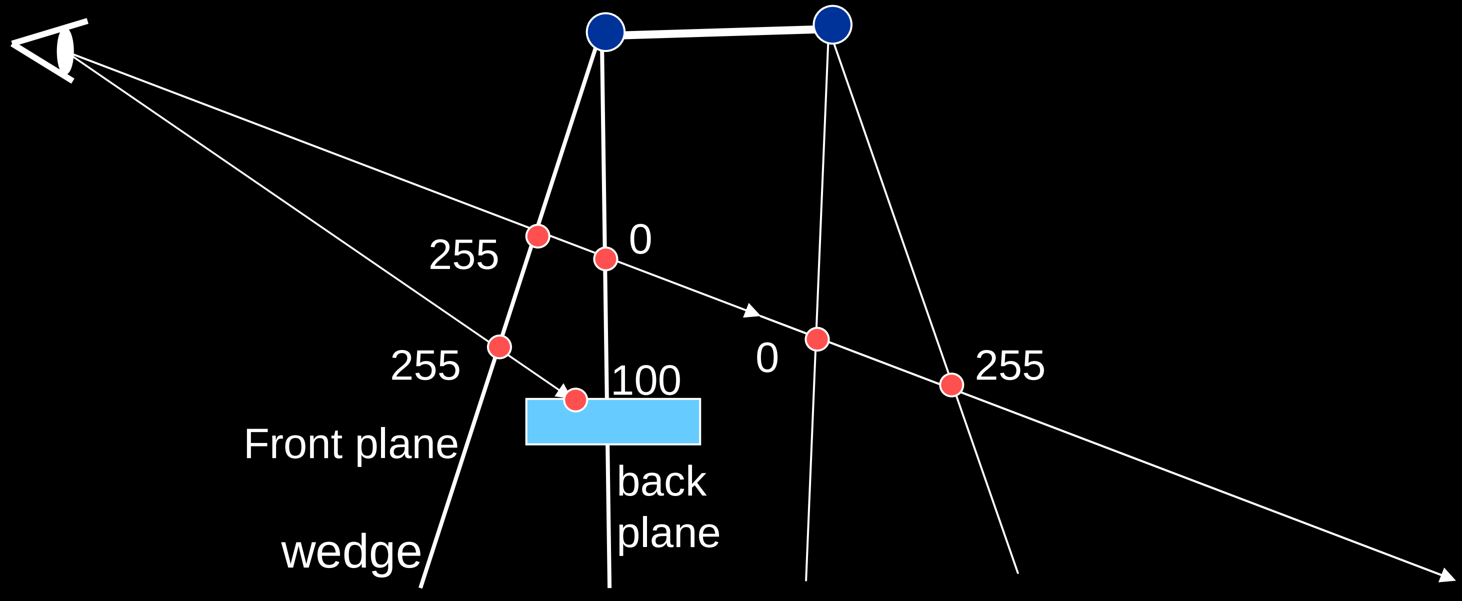
How to rasterize a wedge…

- Init light intensity (LI) buffer to 255 before
- 255 = full light, 0 = no light, 0 < x < 255 → penumbra

```
rasterizeWedge()
foreach visible fragment(x,y)...
  ...on front facing triangles of wedge
  p_f = computeEntryPointOnWedge(x,y);
  p_b = computeExitPointOnWedge(x,y);
  p = point(x,y,z); – z is the Z-buffer value at (x,y)
  p_i = choosePointClosestToEye(p,p_b);
  s_f = computeLightIntensity(p_f);
  s_i = computeLightIntensity(p_i);
  addToLIBuffer(round(255*(s_i - s_f)));
end;
```



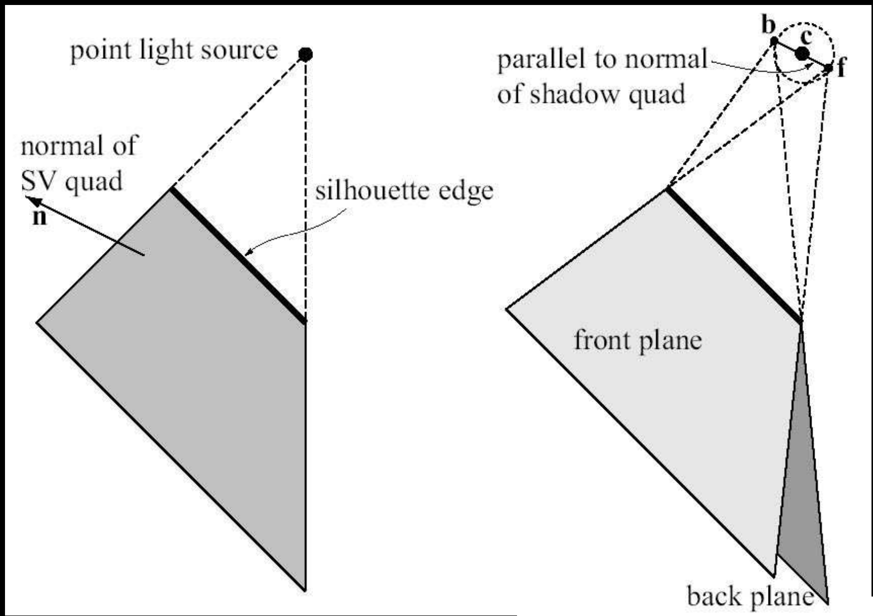
Examples of rasterization in 2D

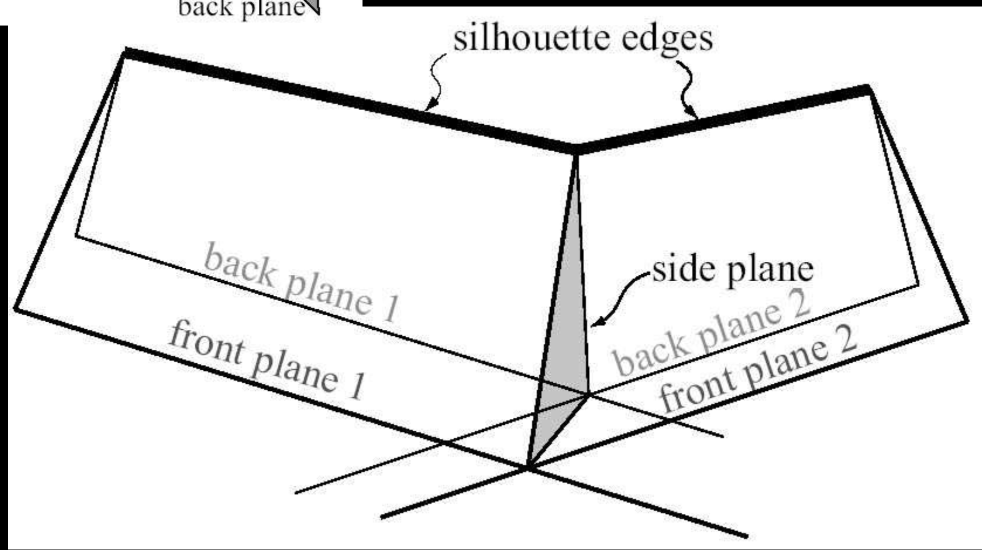


LI-buffer =255 + (0 – 255) = 0 (umbra)

- Next, describe missing pieces:
  - Construction of wedges
  - Light intensity interpolation

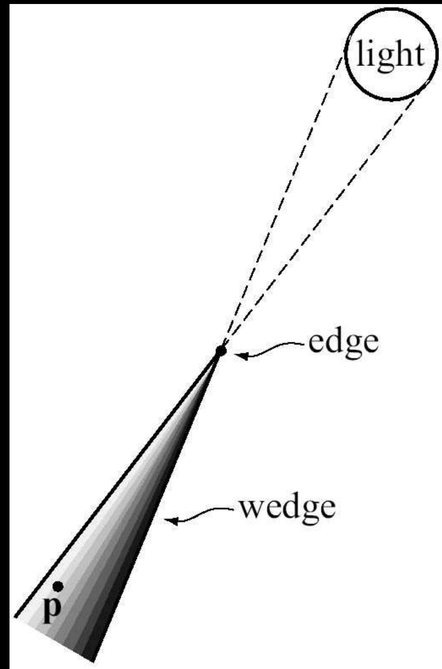
Penumbra wedge construction





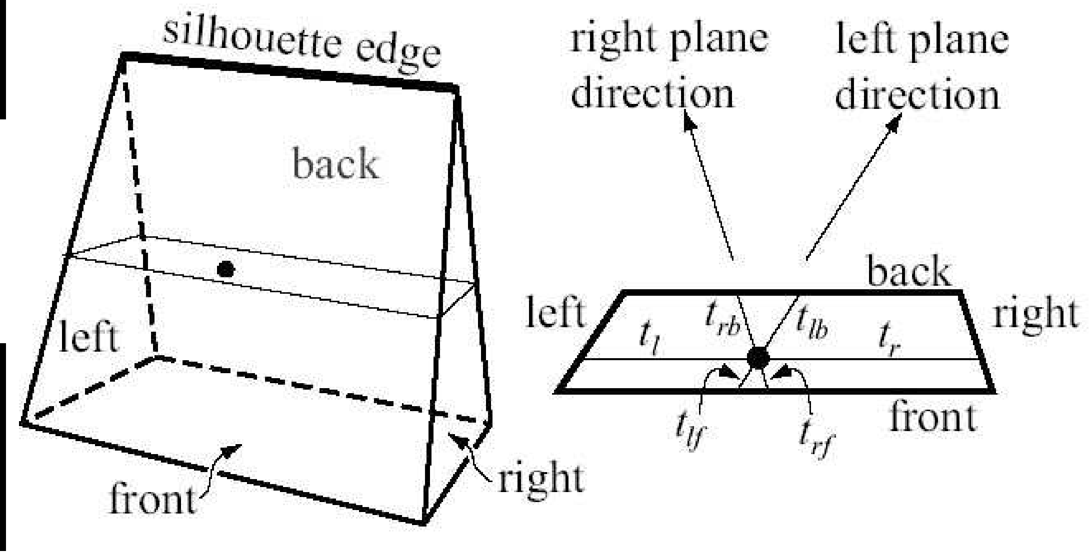
Light intensity interpolation

- To make it possible to implement using programmable hardware:
  - Our only requirement was C^0 continuity across wedges (side planes)

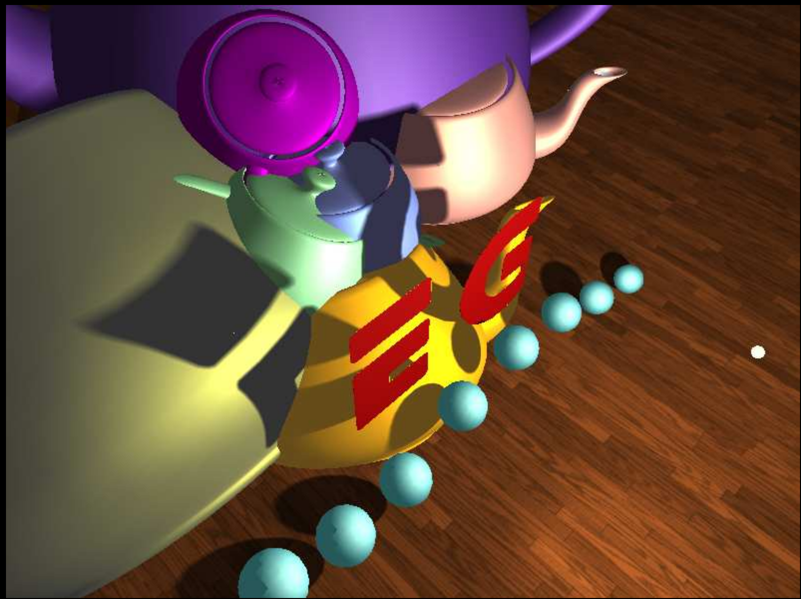
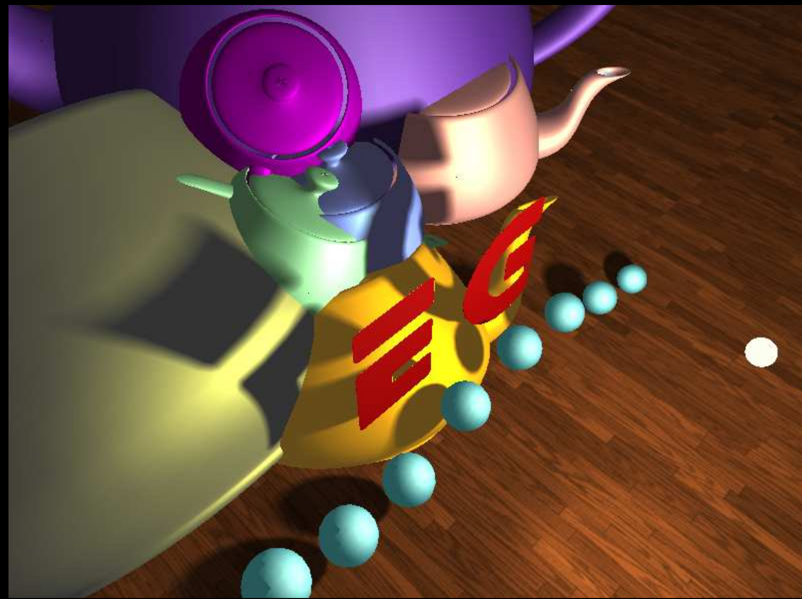
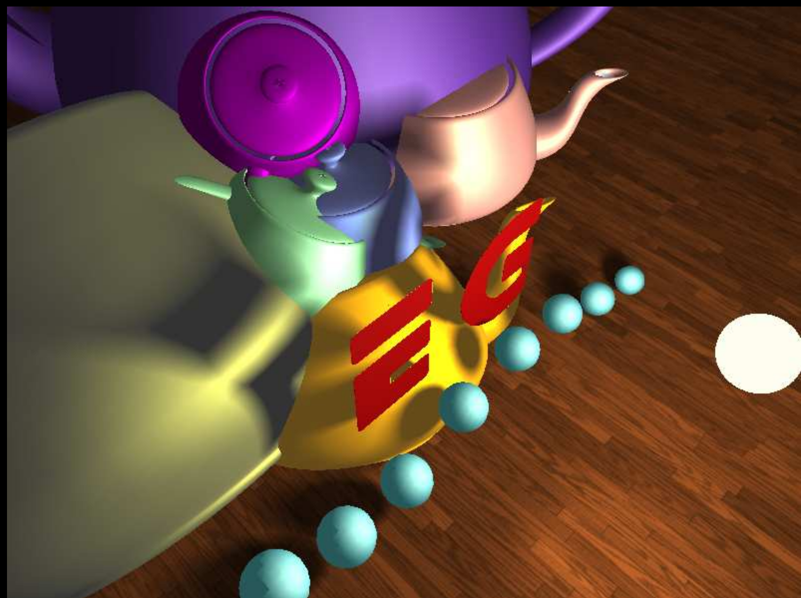


$$s_l = \frac{t_{lb}}{t_{lf} + t_{lb}}, \quad s_r = \frac{t_{rb}}{t_{rf} + t_{rb}}$$

$$s = \frac{t_r}{t_r + t_l} s_l + \frac{t_l}{t_r + t_l} s_r$$



Results

Methods to cover

- Distributed ray tracing soft shadows

- Penumbra wedges
- Soft shadow volumes

- Soft shadow mapping by backprojection

Soft Shadow Volumes for Ray Tracing

Samuli Laine — Helsinki University of Technology, Hybrid Graphics Ltd.

Timo Aila — Helsinki University of Technology, Hybrid Graphics Ltd.

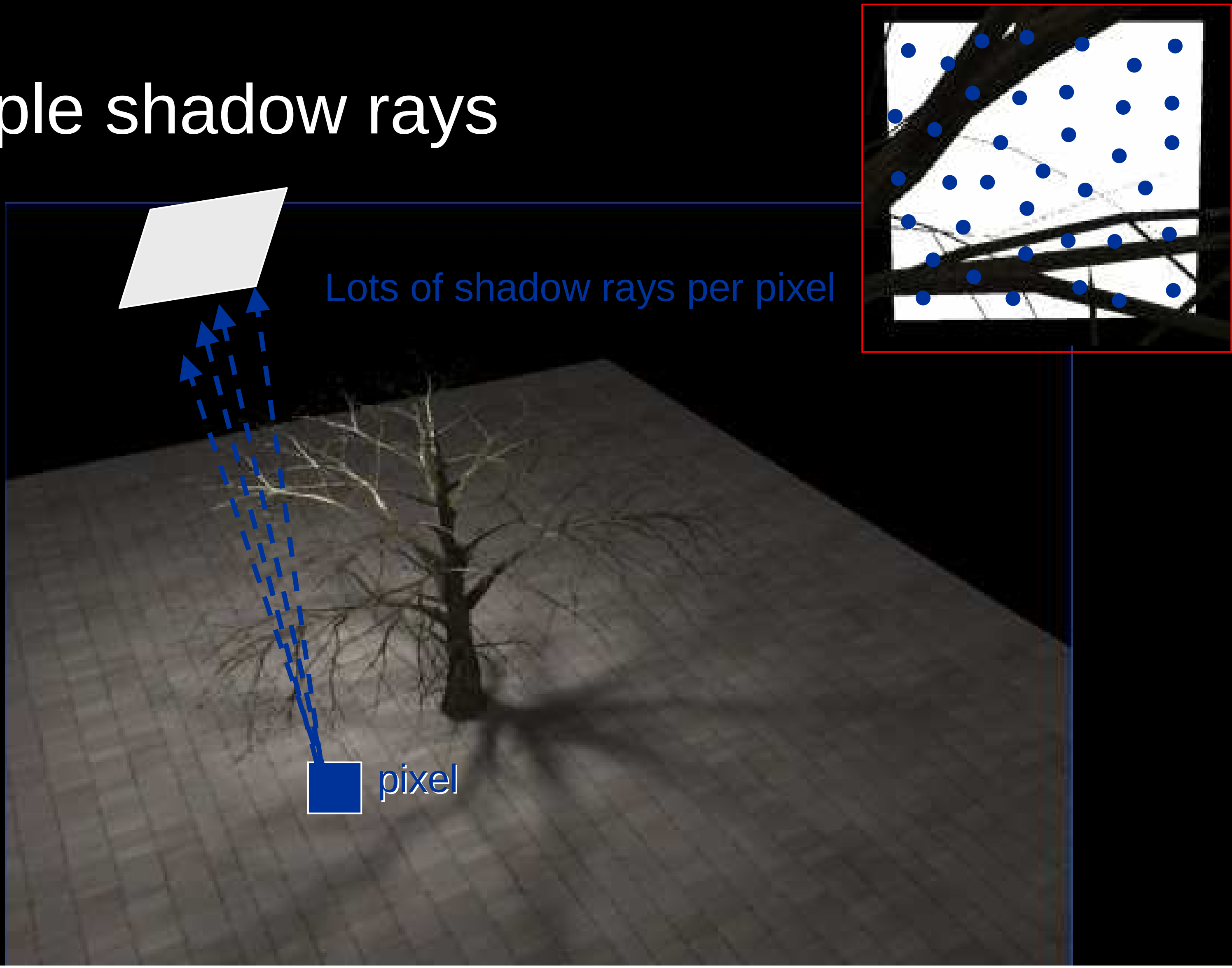
Ulf Assarsson — ARTIS, GRAVIR /IMAG INRIA, Chalmers University of Technology

Jaakko Lehtinen — Helsinki University of Technology, Remedy Entertainment Ltd.

Tomas Akenine-Möller — Lund University

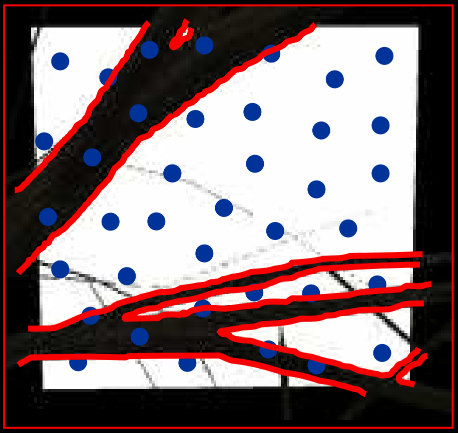
Classic solution

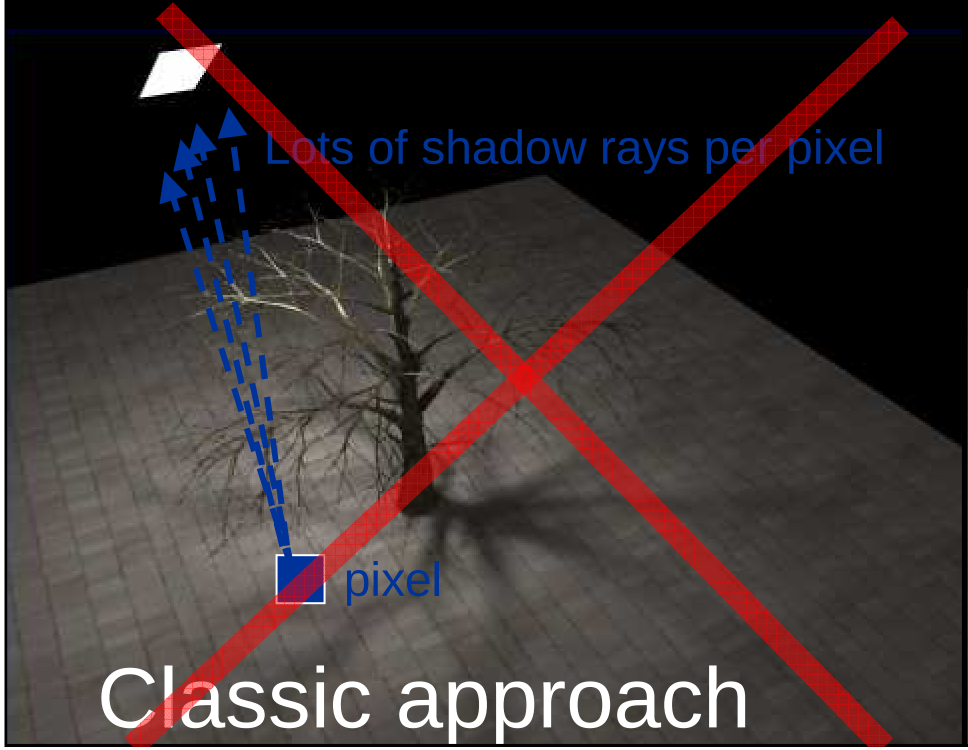
- Multiple shadow rays

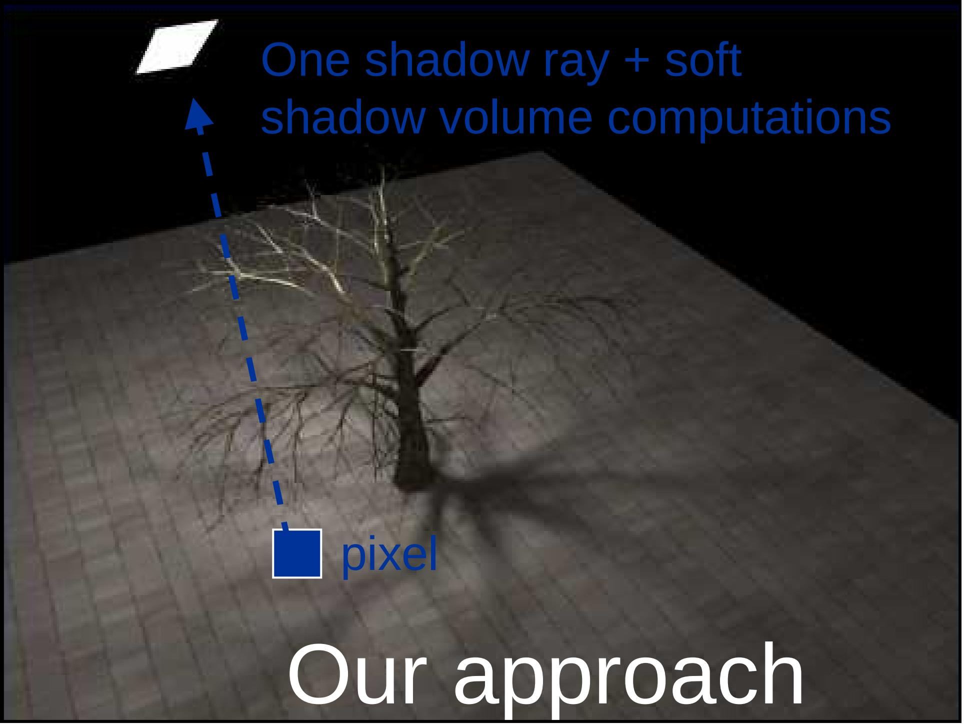


New solution - overview

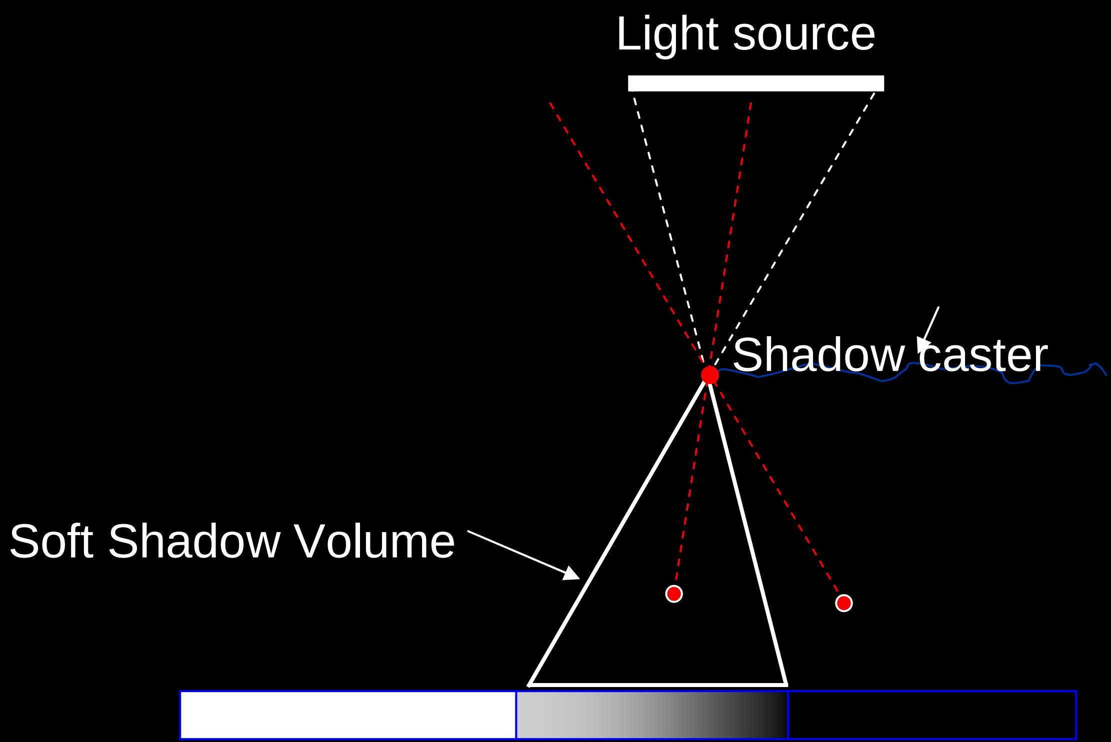
- Replace the shadow rays
  - With soft shadow volume computations
  - Plus one reference shadow ray

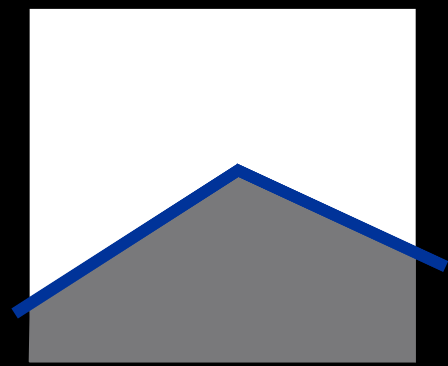






What’s a Soft Shadow Volume?

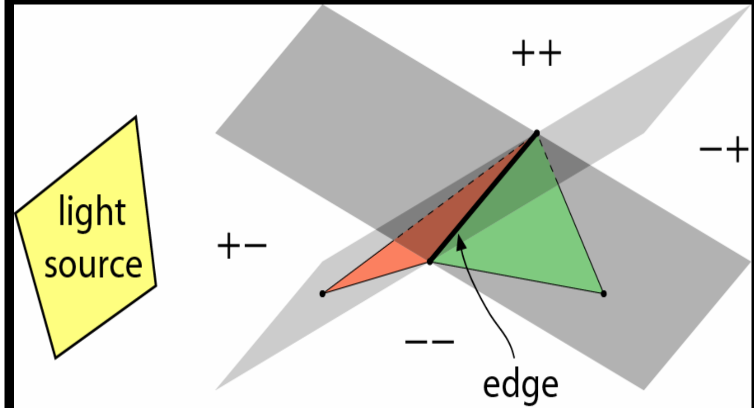


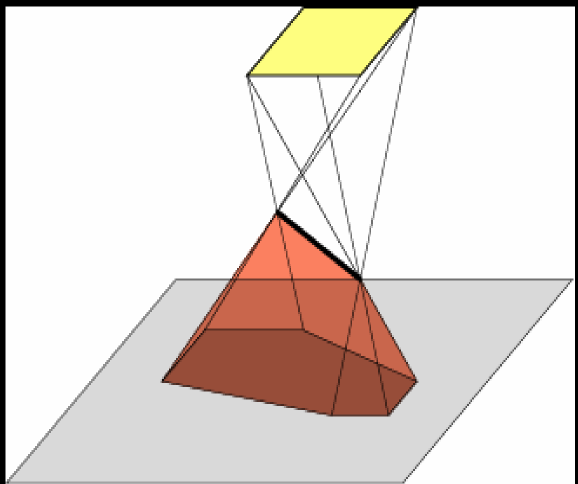


Soft Shadow Volume =

A. Volume from which an edge projects onto the light source
B. Region of penumbra caused by an edge

Wedge Creation Criterions



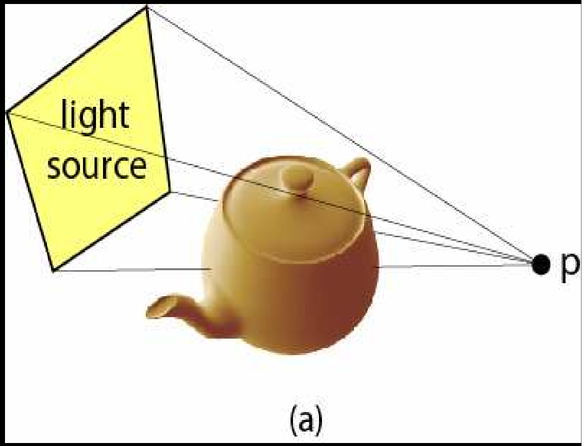


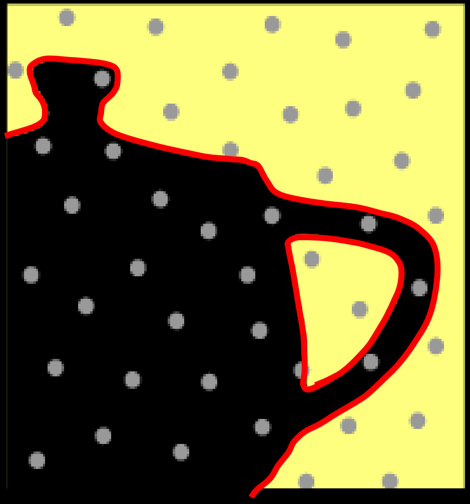
1. Wedges are created for all edges that are silhouettes from any point on the light source
2. The wedge includes all positions from which the edge projects onto (occludes) the light source.

Our solution - overview

Two parts:

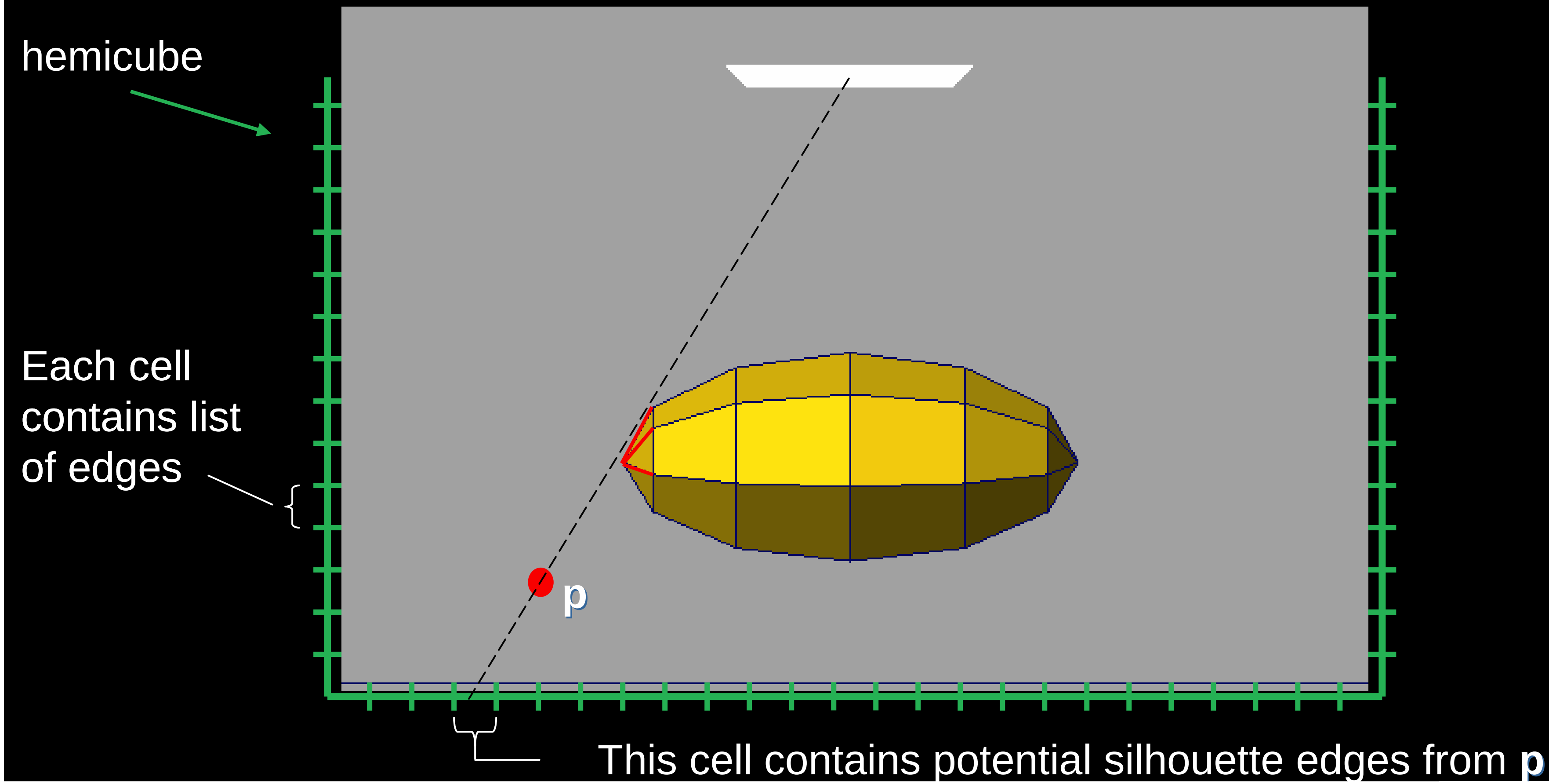
- from any receiving point **p**, we need to find silhouette edges affecting the visibility
- A method for computing the visibility from silhouette information





Hemicube Construction

Rasterize soft shadow volumes into a hemicube for each light source

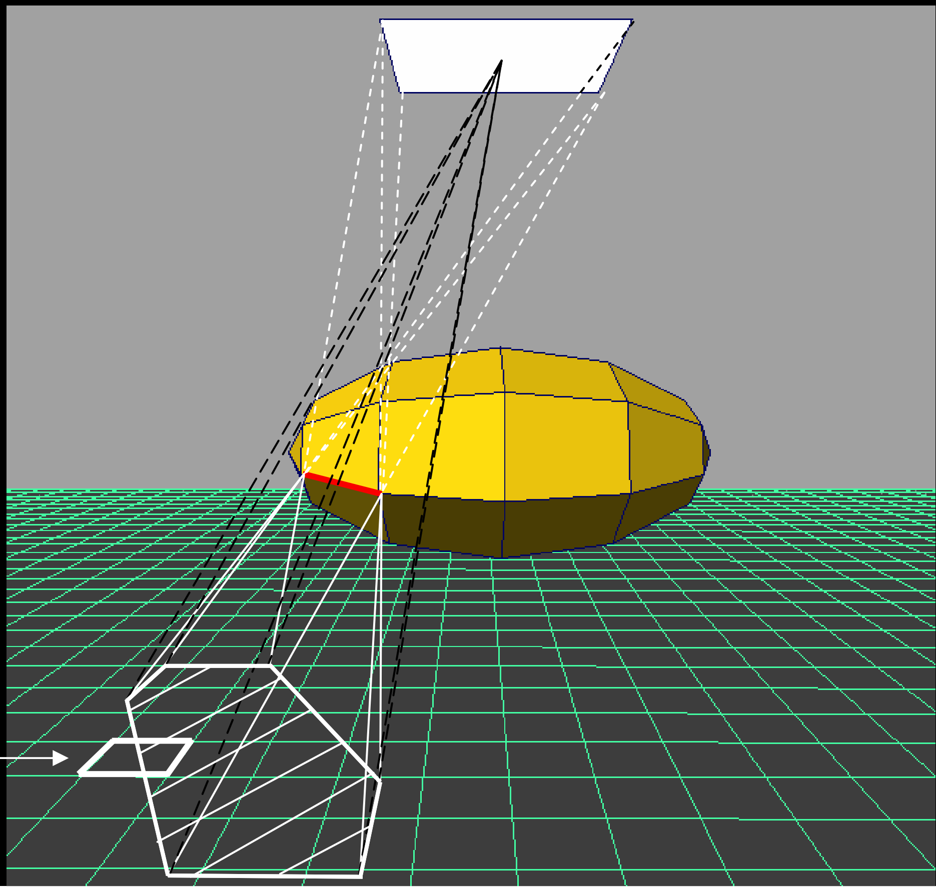


Hemicube Construction

Wedge marked to all cells it even partially overlaps → no artifacts

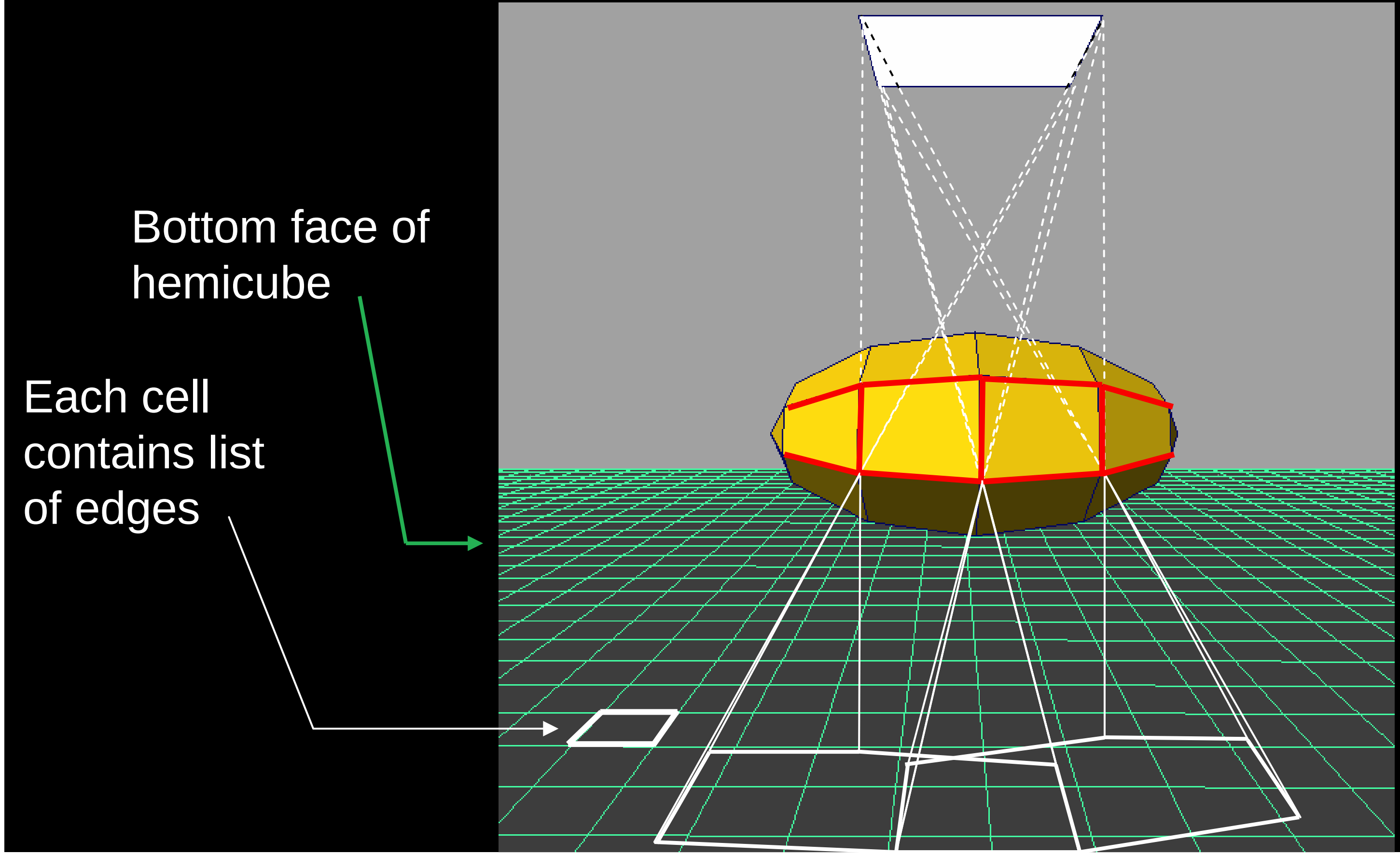
Bottom face of hemicube

Each cell contains list of edges



Hemicube Construction

Wedge marked to all cells it even partially overlaps → no artifacts

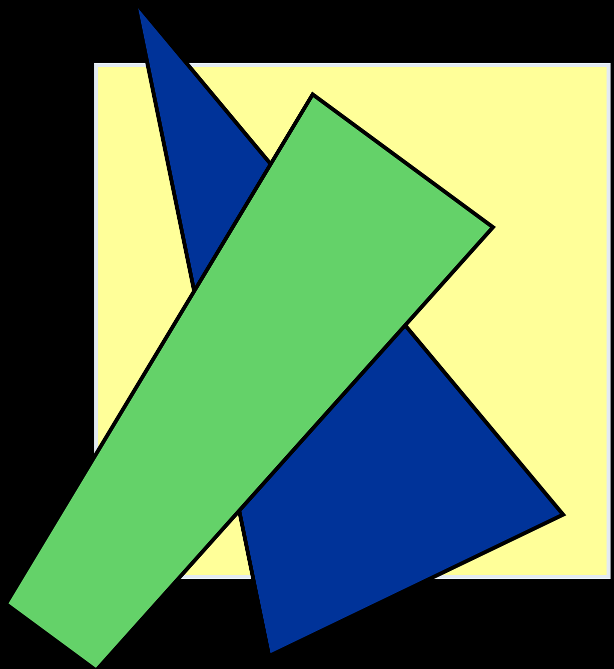


Visibility Reconstruction

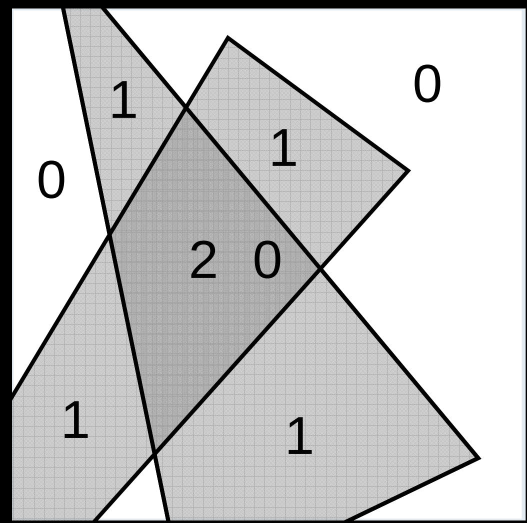
- Which light samples s_i are visible from point p?
- Brute force: cast a shadow ray for each s_i
- Our recipe:
  1. Find silhouette edges between p and light source
  2. Project them onto light source → reduces to 2D
  3. Compute relative depth complexity for every s_i
  4. Solve visibility with a single shadow ray
  5. Profit

Depth Complexity

- Depth complexity of s_i = number of surfaces between p and s_i



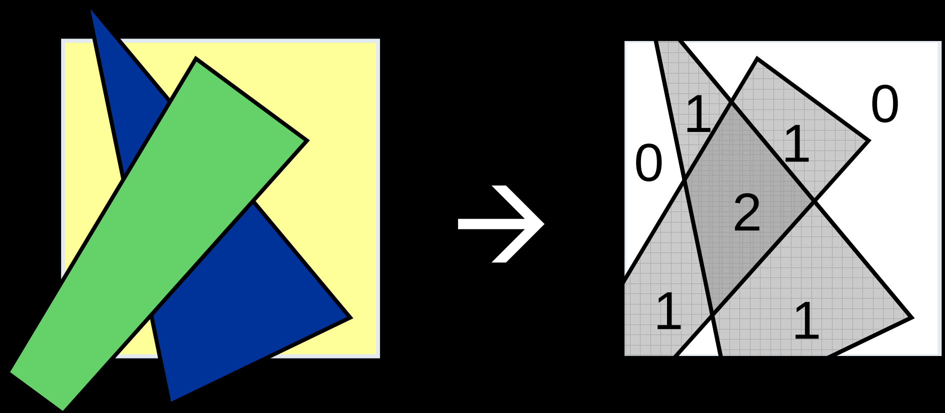
Light source as seen from p



Depth complexity function

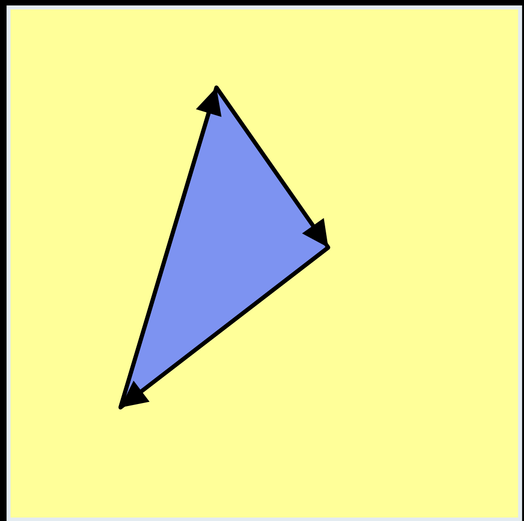
From Silhouette Edges to Relative Depth Complexity

- Projected silhouette edges define the first derivative of the depth complexity function
- Hence, **relative** depth complexity can be solved by integrating the silhouette edges over the light source
- Integration is linear → can be performed one edge at a time

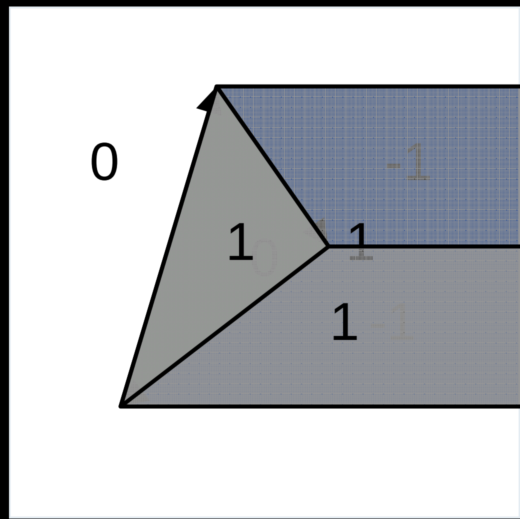


Integration: Example

- Left-to-right integration of a triangular silhouette



→

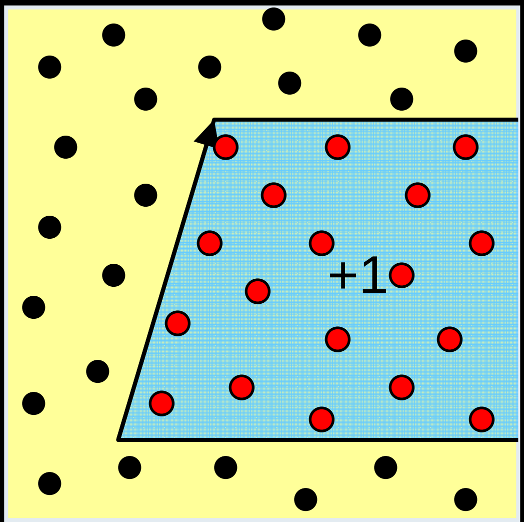


Light source as seen from *p*

Depth complexity function

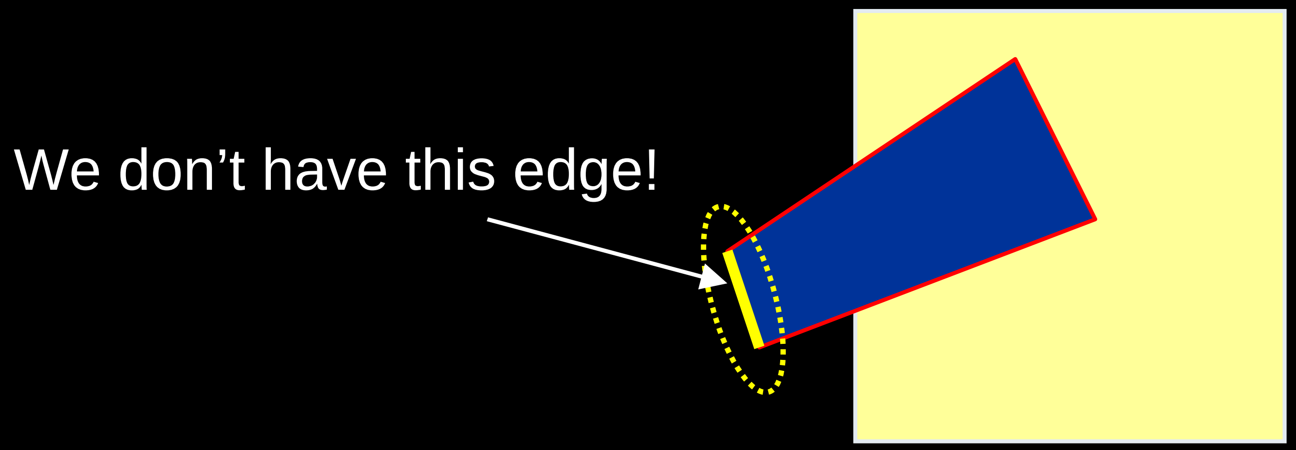
Integration: Sampling Points

- We don’t need the value of the depth complexity function except at the sampling points s_i
- Sufficient to maintain a depth complexity counter for each s_i
- Integration: find s_i that are inside update region and update their depth complexity counters



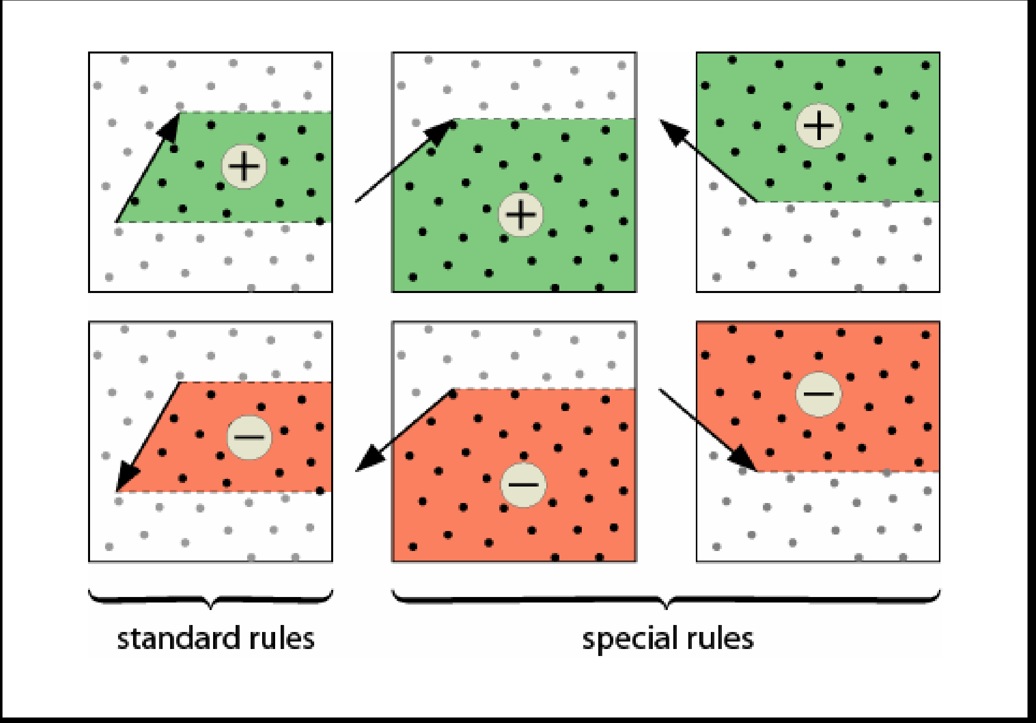
Integration: Rules

- There's a caveat - we only have the edges that overlap the light source
- Loops are not necessarily closed, since parts outside the light source may be missing



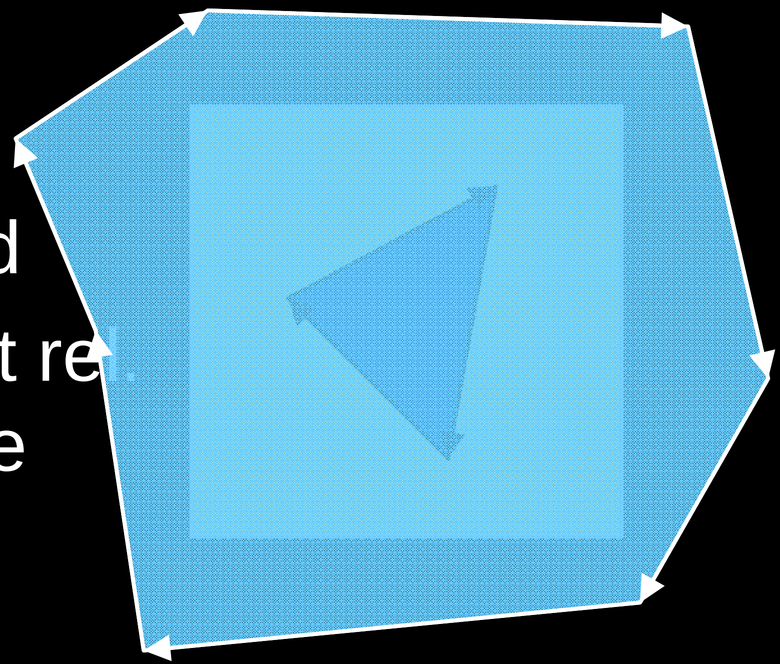
Integration: Rules

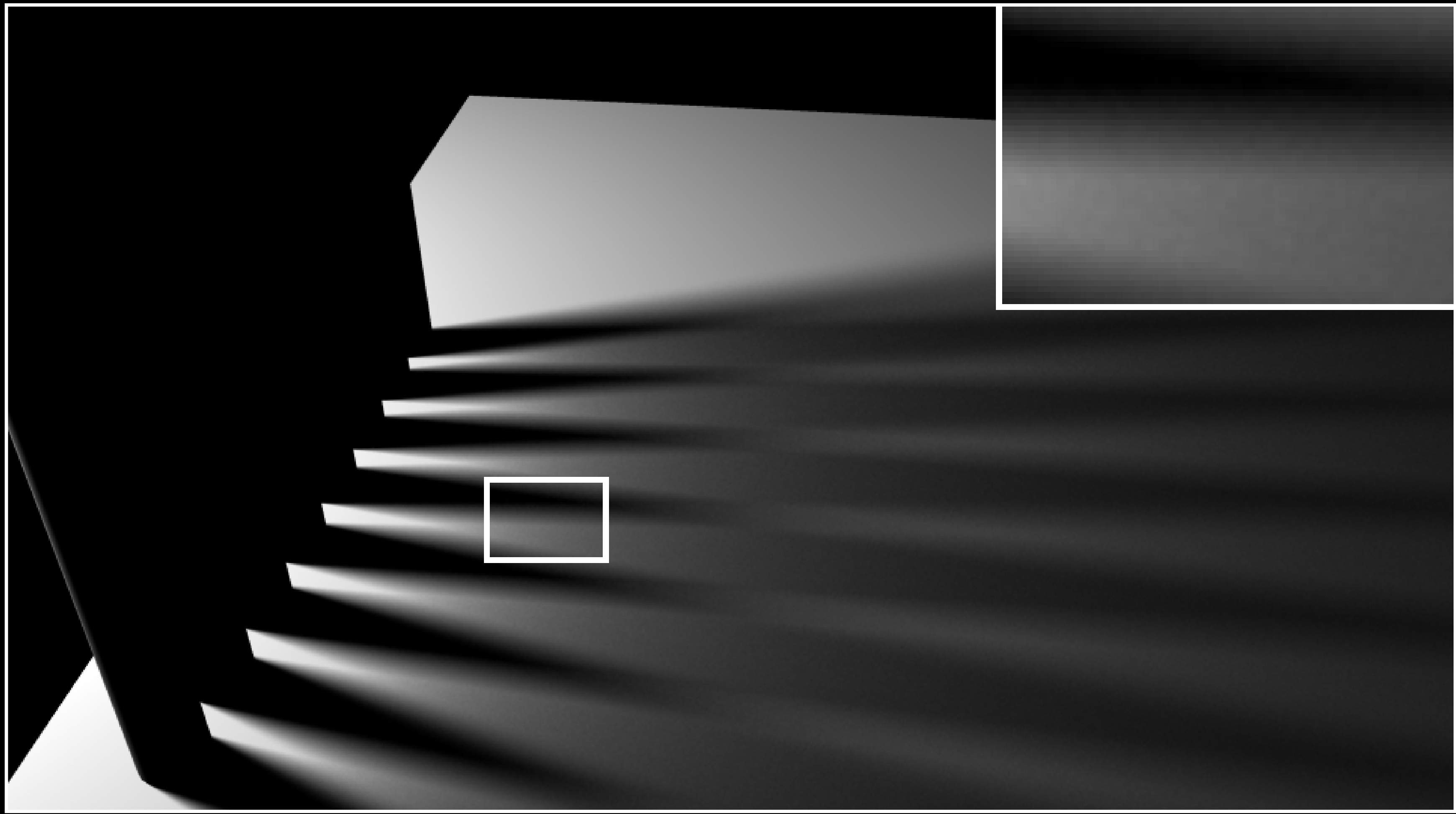
- Solution: apply special rules to edges that cross the left side of the light source
- This accounts for potentially missing edges



From Relative Depth Complexity to Visibility

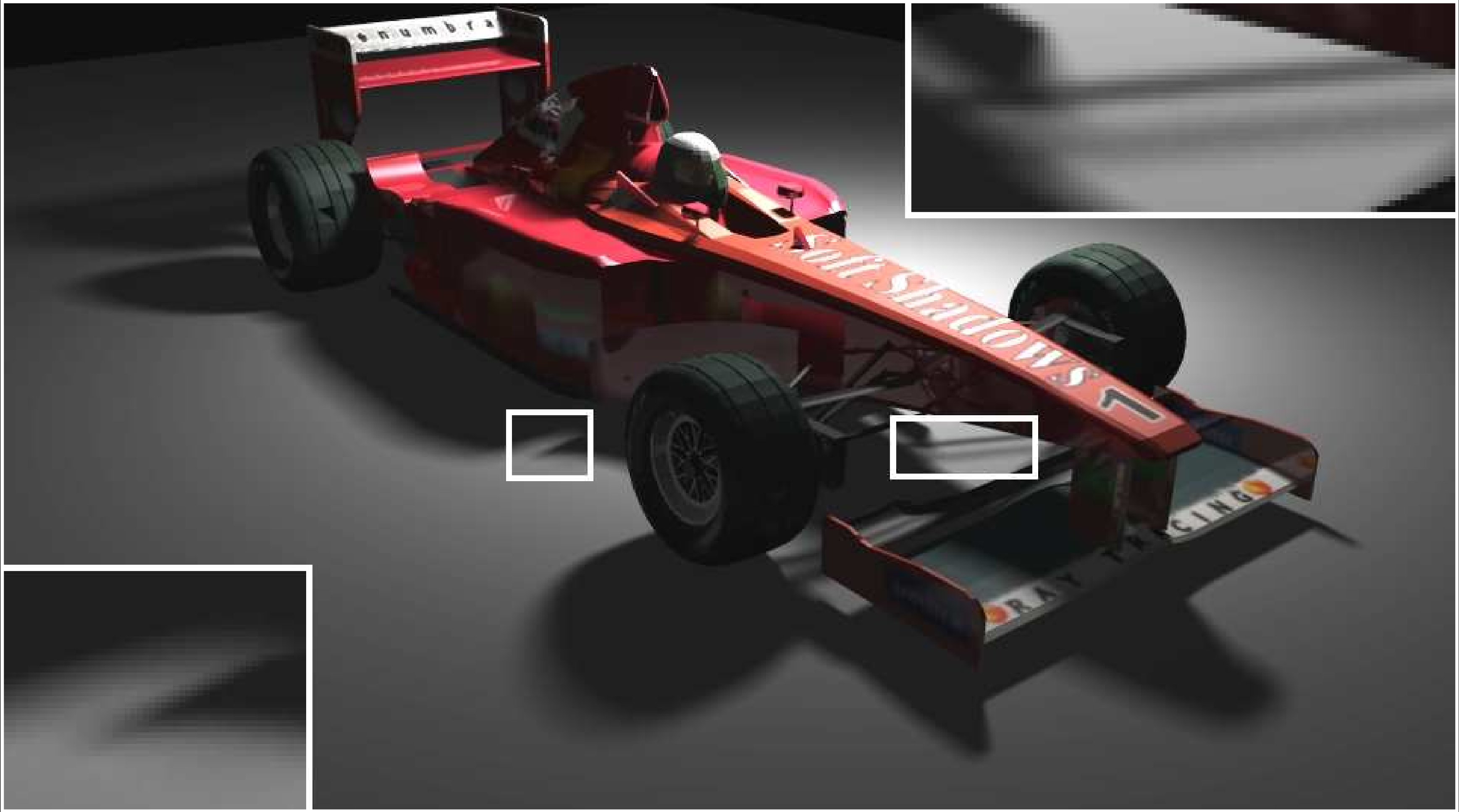
- We are not done yet, since the constant of integration is not known → cannot solve visibility
- Solution: cast a shadow ray ray to one s_i with lowest relative depth complexity
- If blocked, all s_i are blocked
- Otherwise, all s_i with lowest rel. depth complexity are visible





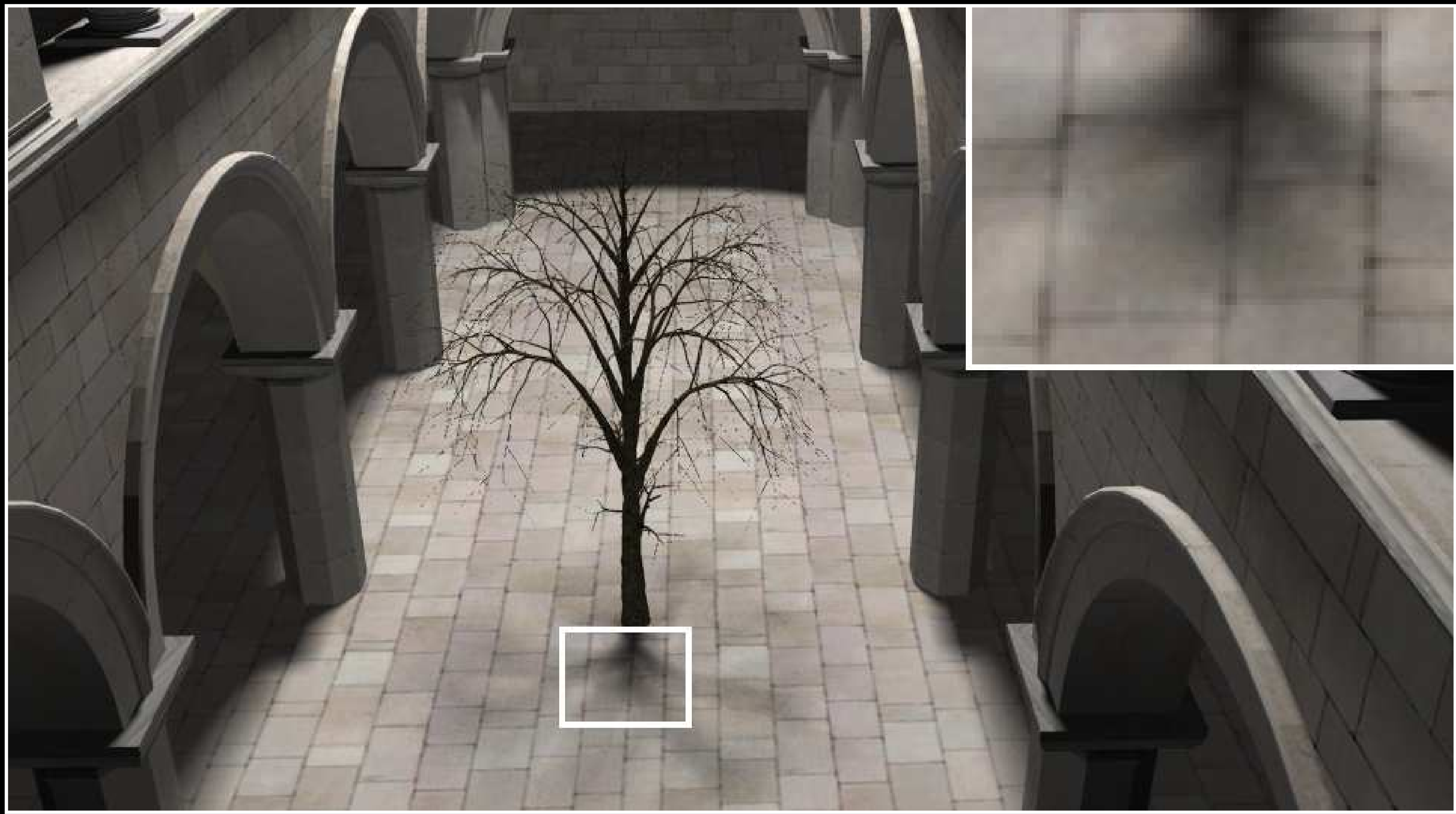
Columns: 580 triangles, adaptive AA, 960x540

	L = 256	L = 1024
Speedup factor	**103**	**242**



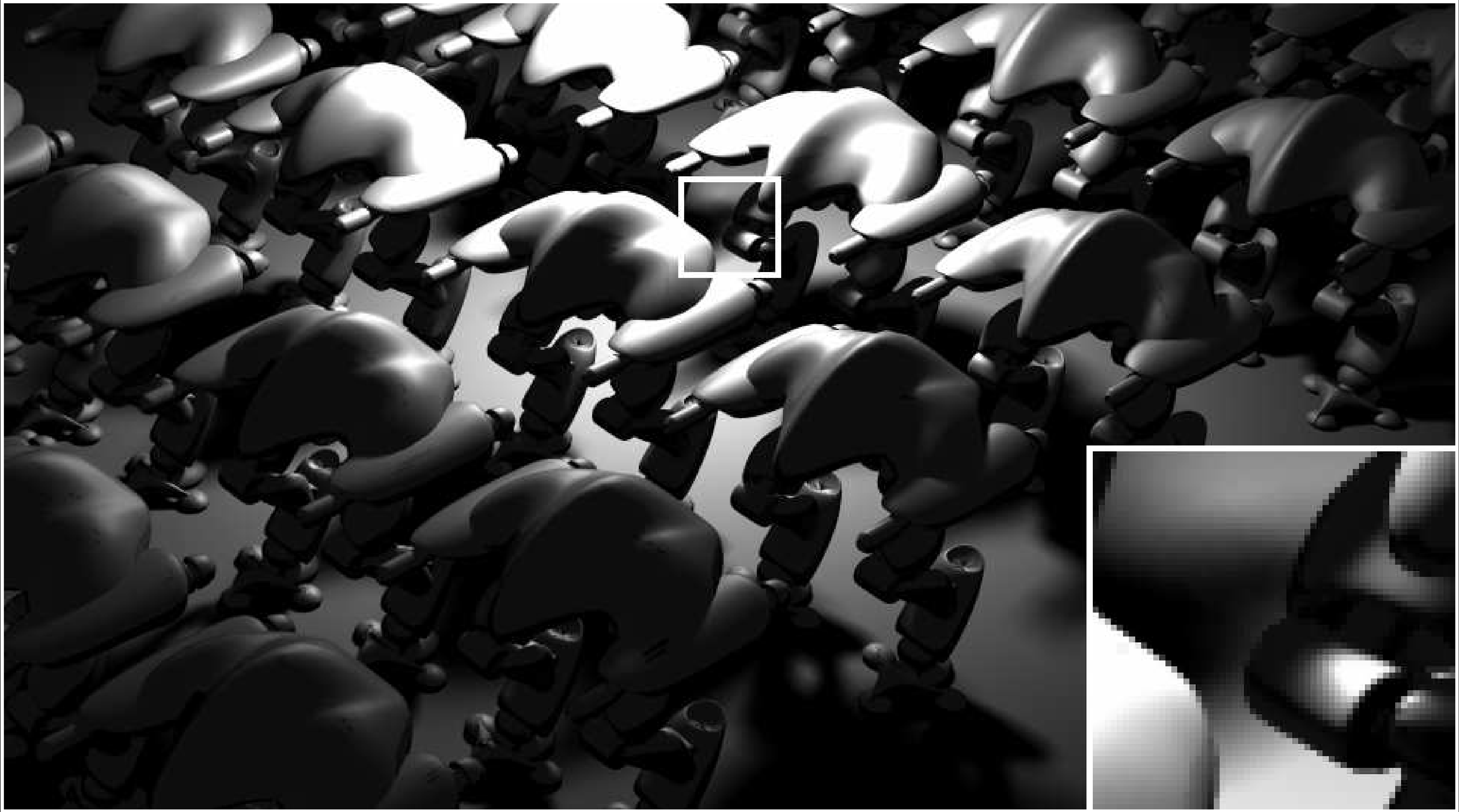
Formula: 60K triangles, adaptive AA, 960x540

	L = 200	L = 800
Speedup factor	**30**	**65**



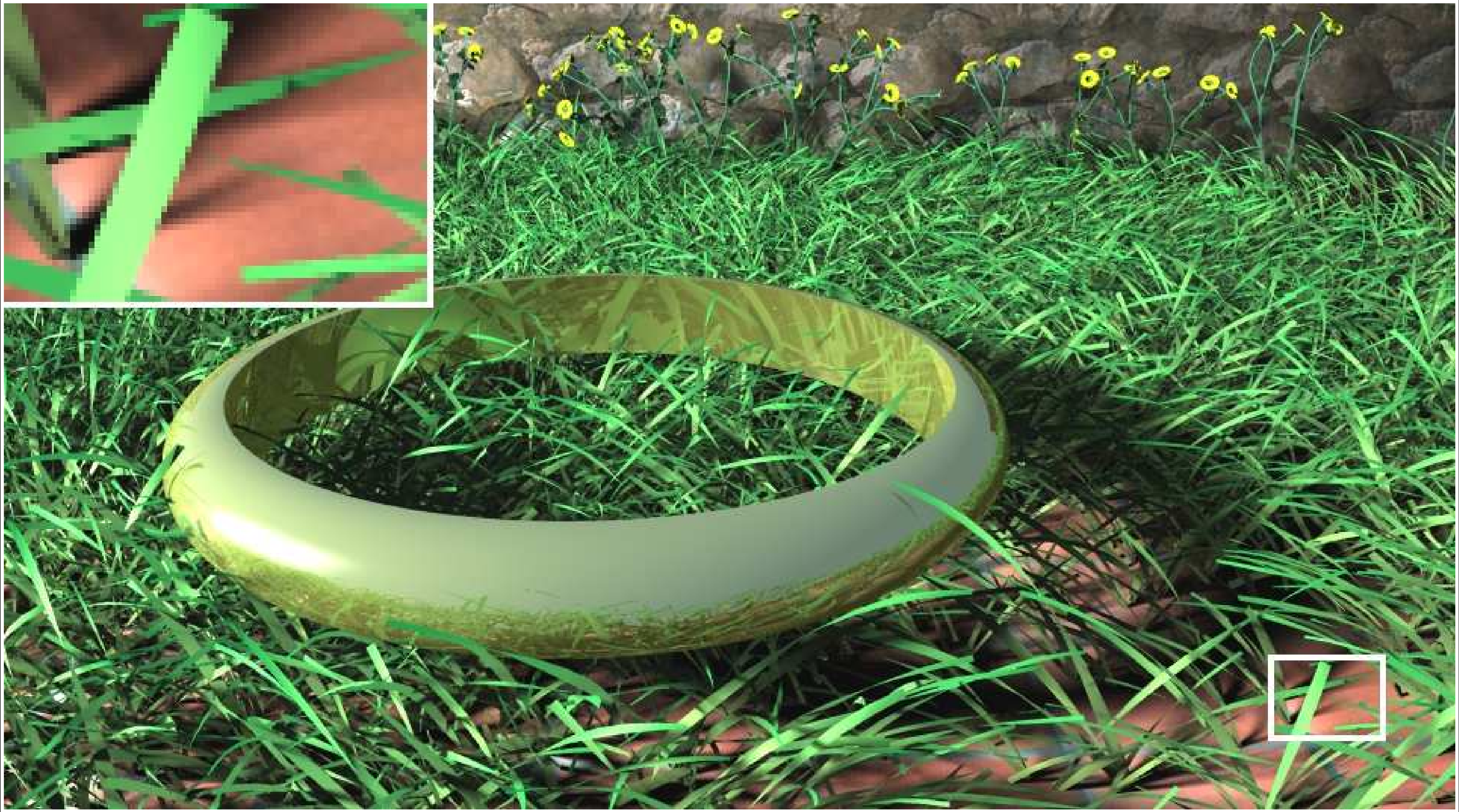
Sponza: 109K triangles, adaptive AA, 960x540

	L = 150	L = 600
Speedup factor	**11**	**33**



Robots: 1.3M triangles, adaptive AA, 960x540

	L = 200	L = 800
Speedup factor	**21**	**58**



Ring: 374K triangles, adaptive AA, 960x540

	L = 150	L = 600
Speedup factor	**13**	**32**

Conclusions

- Fast shadow algorithm in wide range of scenes
- Easy to plug into an existing ray tracer
- Scalability considerations
  - Number of light samples: excellent (~ sqrt M)
  - Number of triangles: good (silhouettes: ~ $N^{(\text{something} \leqslant 1)}$)
  - Output resolution: not so good (linear)
  - Spatial size of the light source: not so good (~ linear)

Methods to cover

- Distributed ray tracing soft shadows

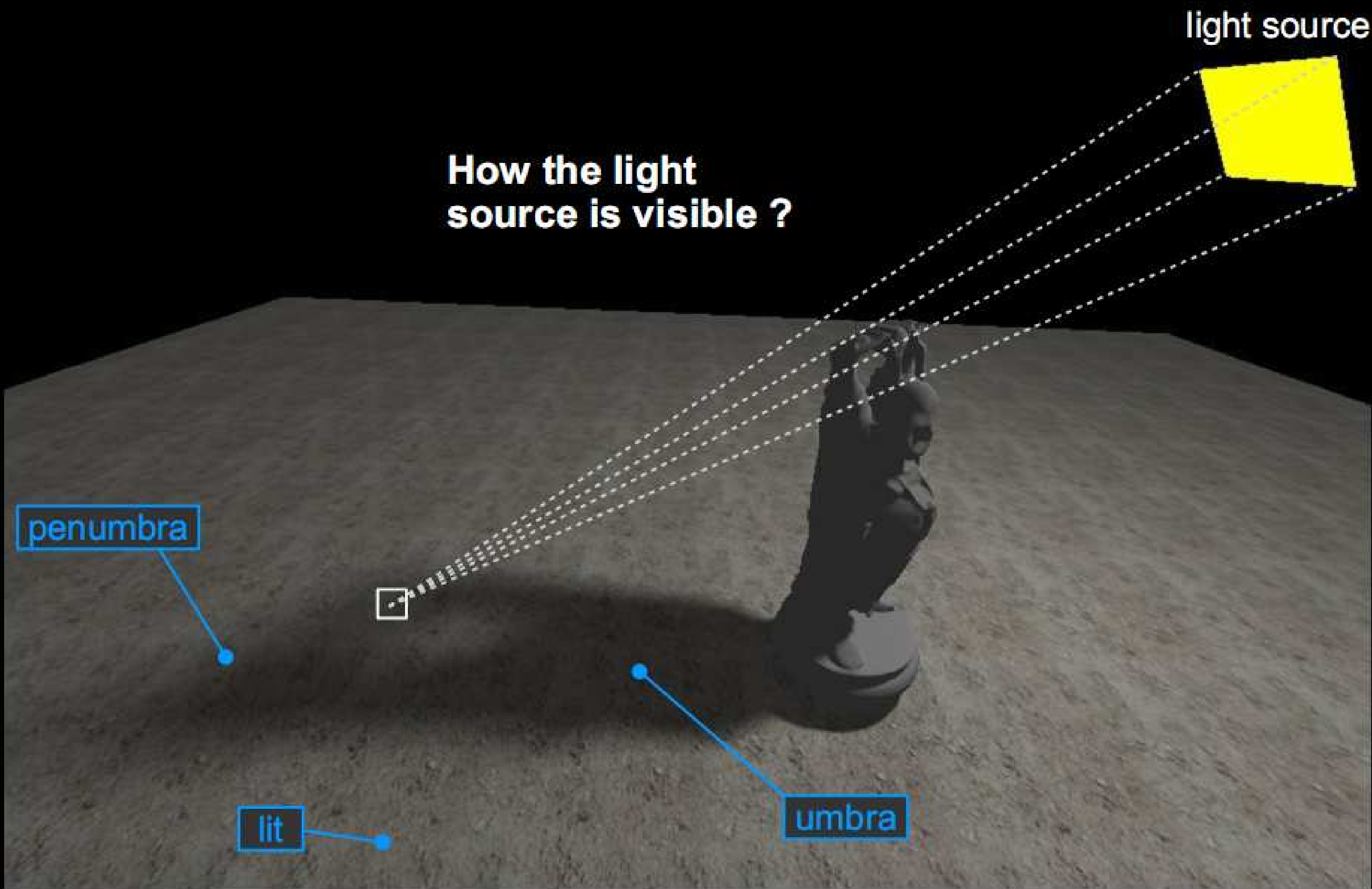
- Penumbra wedges
- Soft shadow volumes

- Soft shadow mapping by backprojection

Real-time **Soft Shadow Mapping** by back-projection

Gaël Guennebaud
Loïc Barthe, Mathias Paulin
IRIT – UPS – CNRS
TOULOUSE – FRANCE
http://www.irit.fr/~Gael.Guennebaud/

Soft Shadows

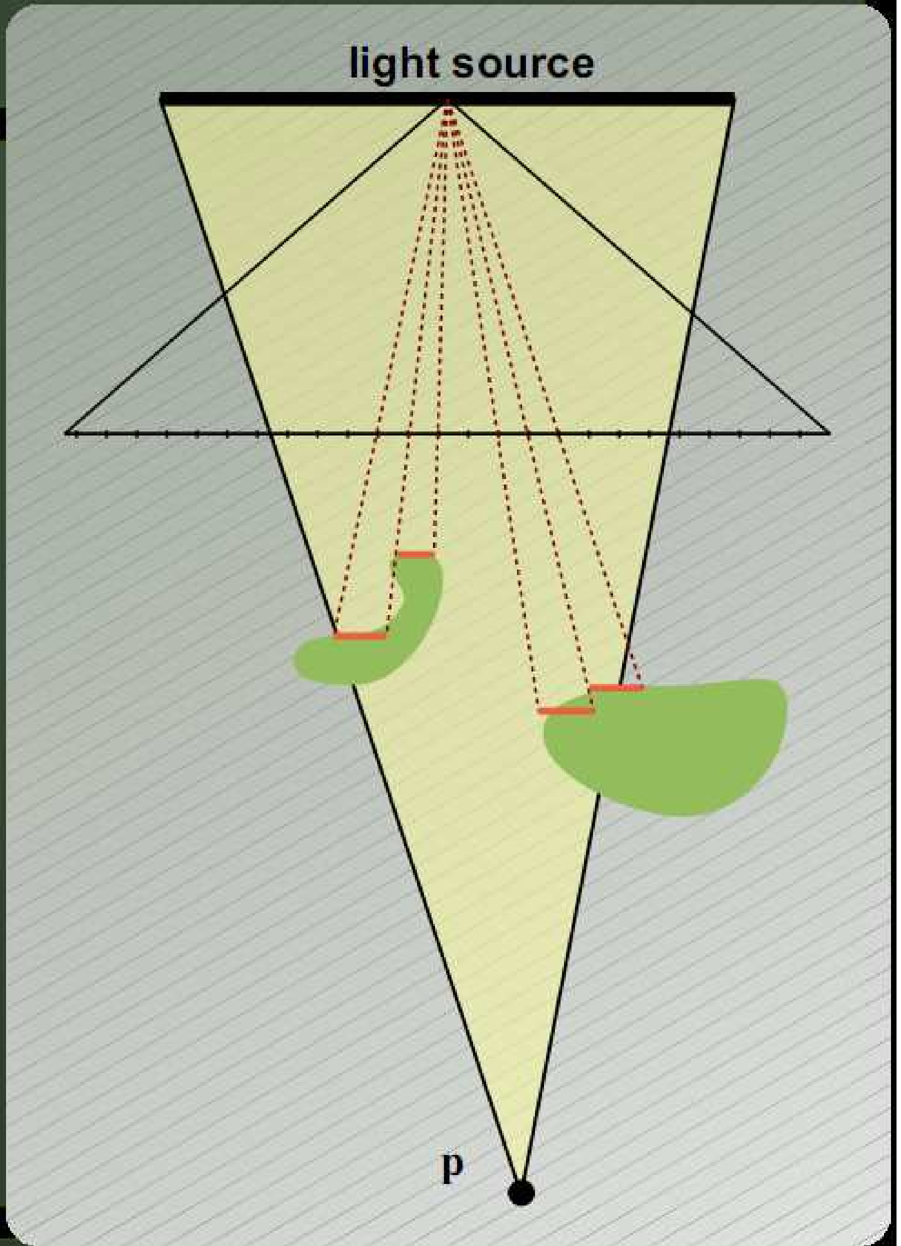


Principle

- What is the visibility percentage v_p between a point **p** and the light source ?

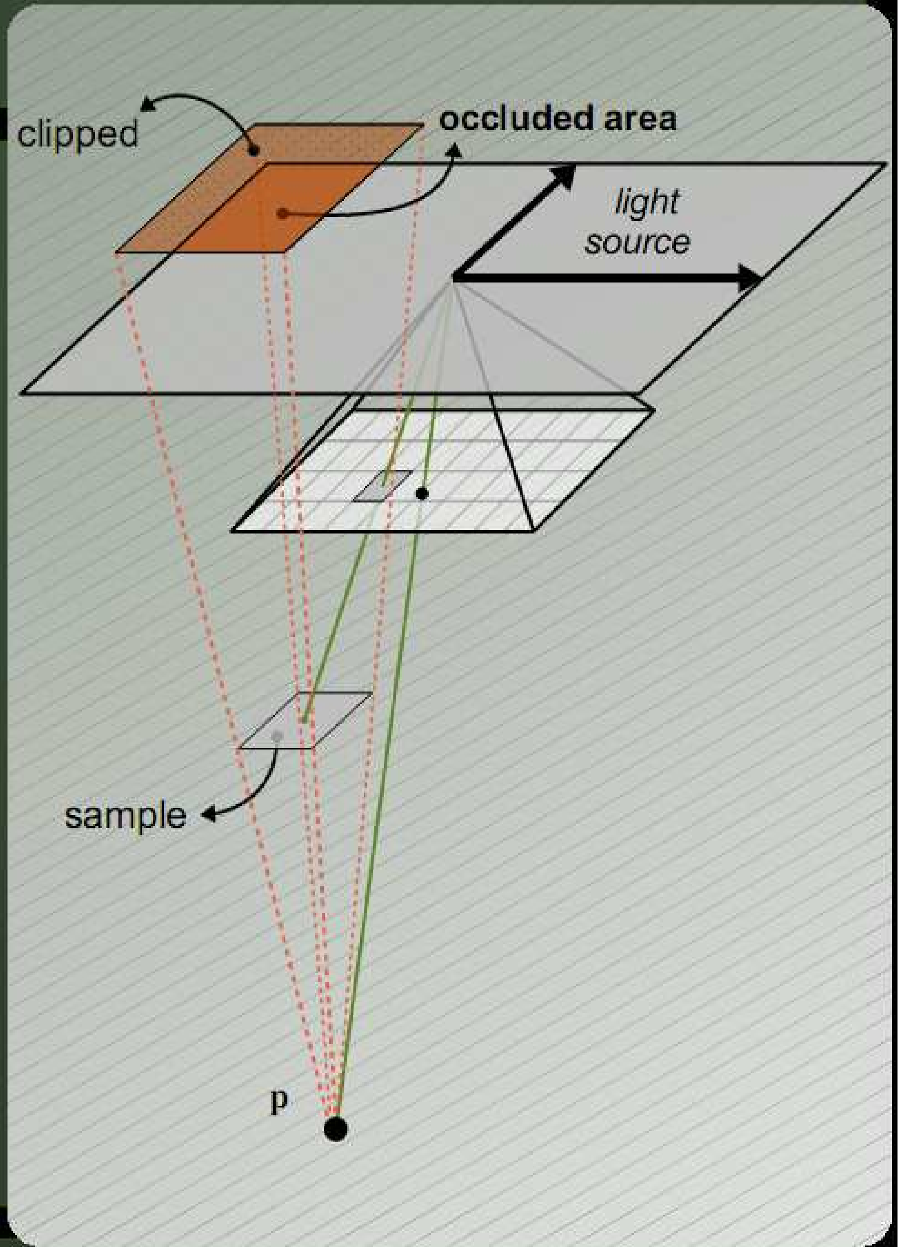
- Our approach:

  key idea: use the **shadow map** as a simplified and discrete representation of the scene



Principle

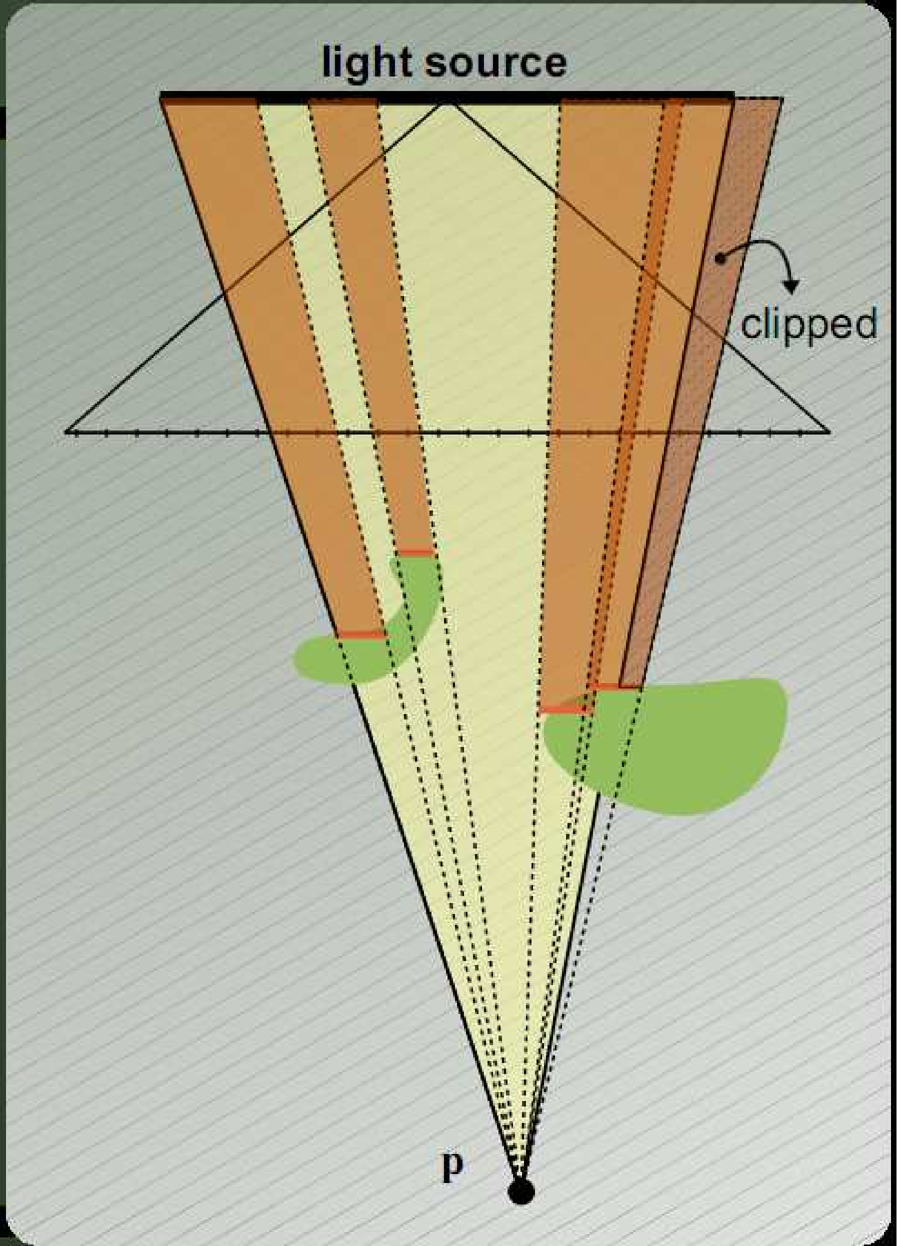
- Area occluded by a shadow map sample ?
    - back-projection on the light source
    - + clipping (trivial)



Principle

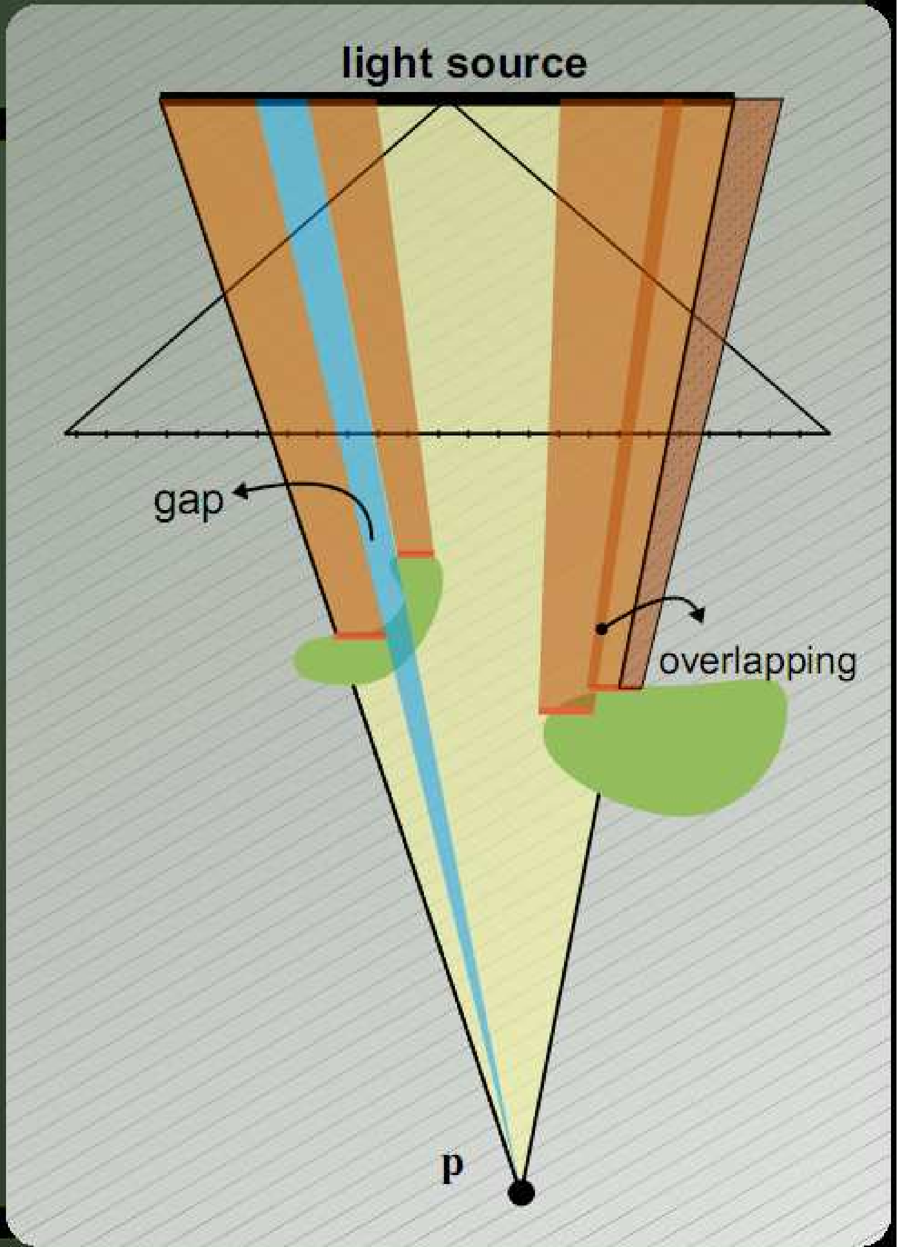
- What is the visibility percentage v_p between a point **p** and the light source ?
  - algorithm:

    subtract the area occluded by each shadow map sample



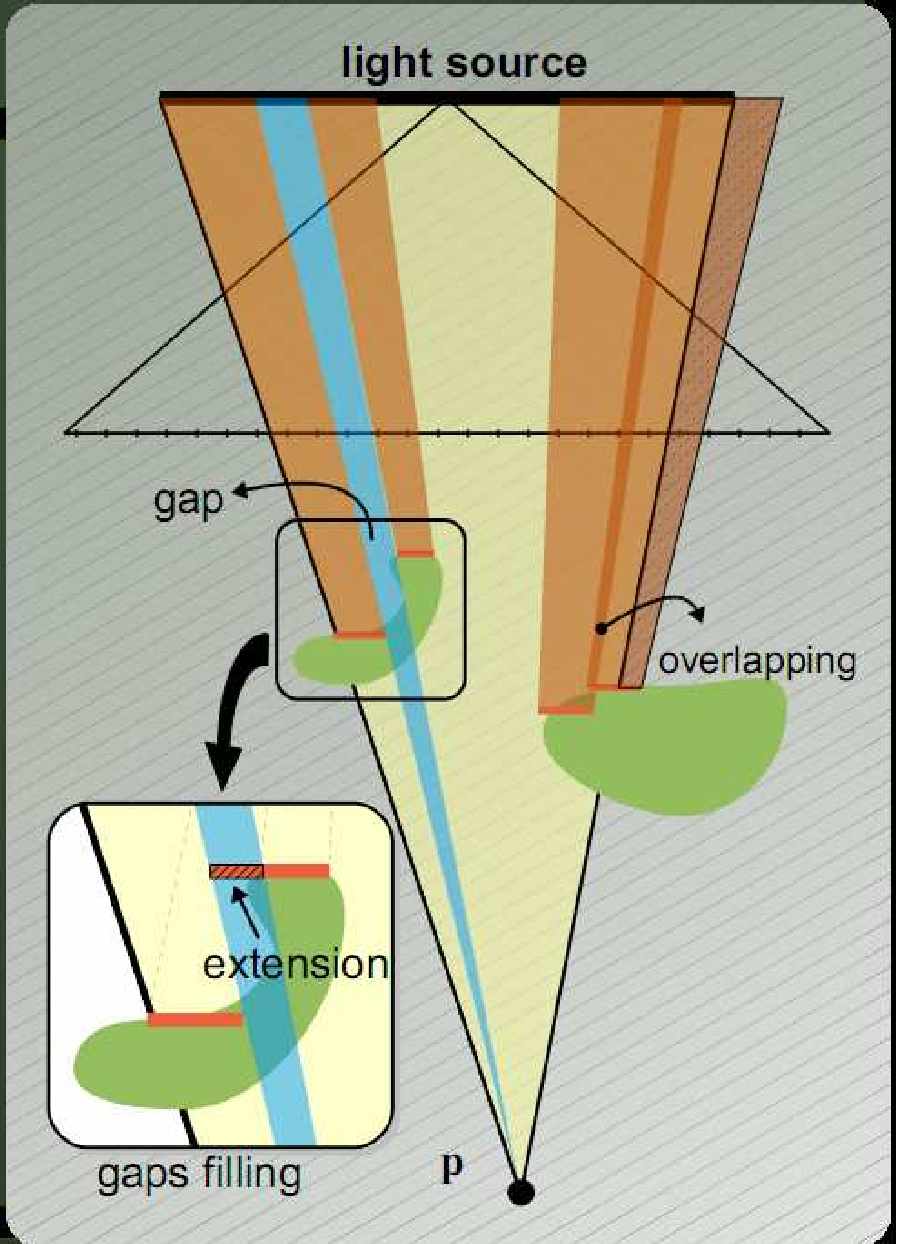
Main issue

- gaps & overlaps
  - simple in 1D
  - very complex in 2D

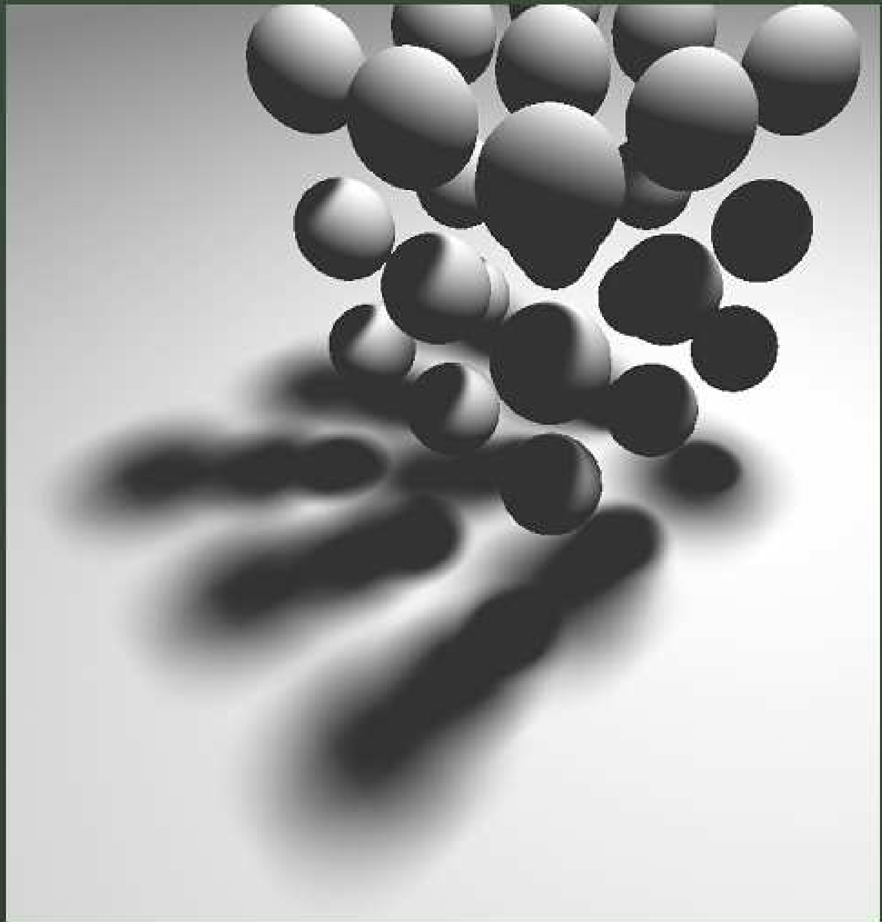


Gaps filling

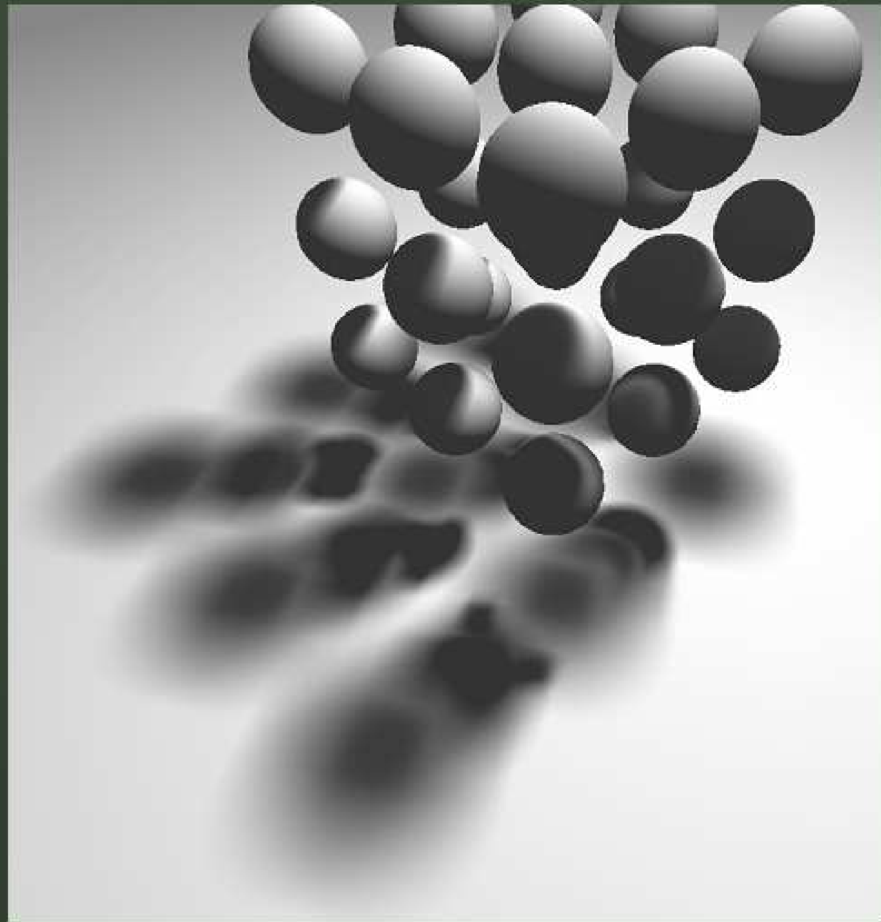
- gaps & overlaps
  - simple in 1D
  - very complex in 2D
- overlap artifacts are acceptable
- => at this time, we just fill the gaps



Gaps filling

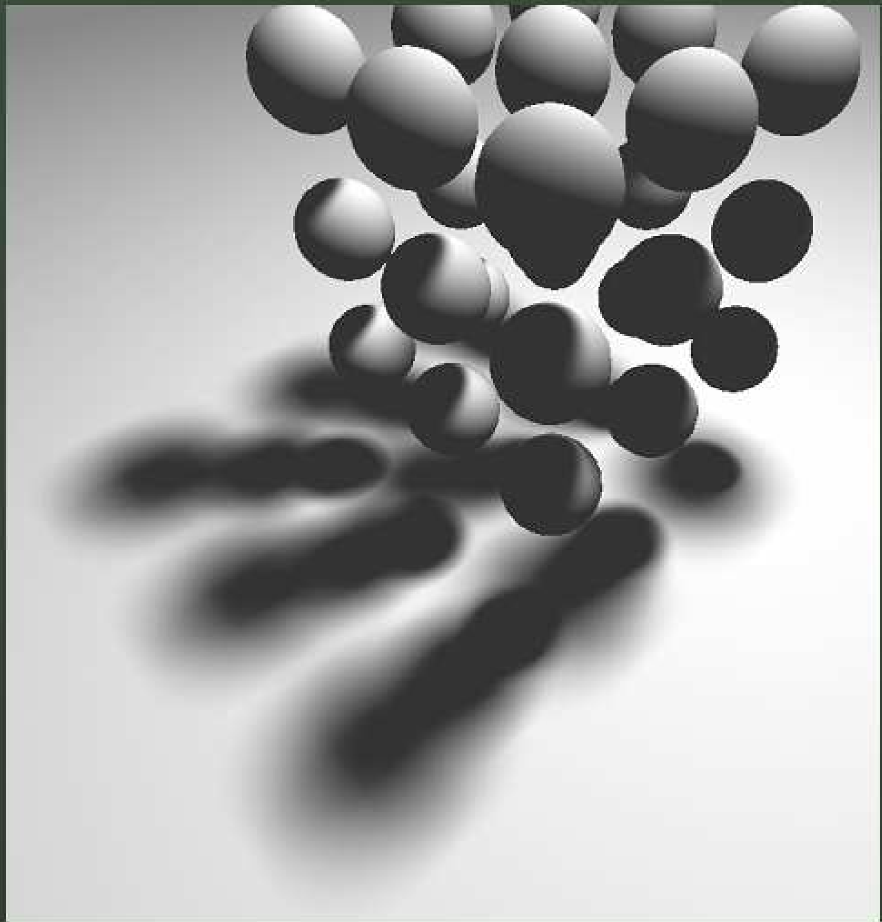


reference

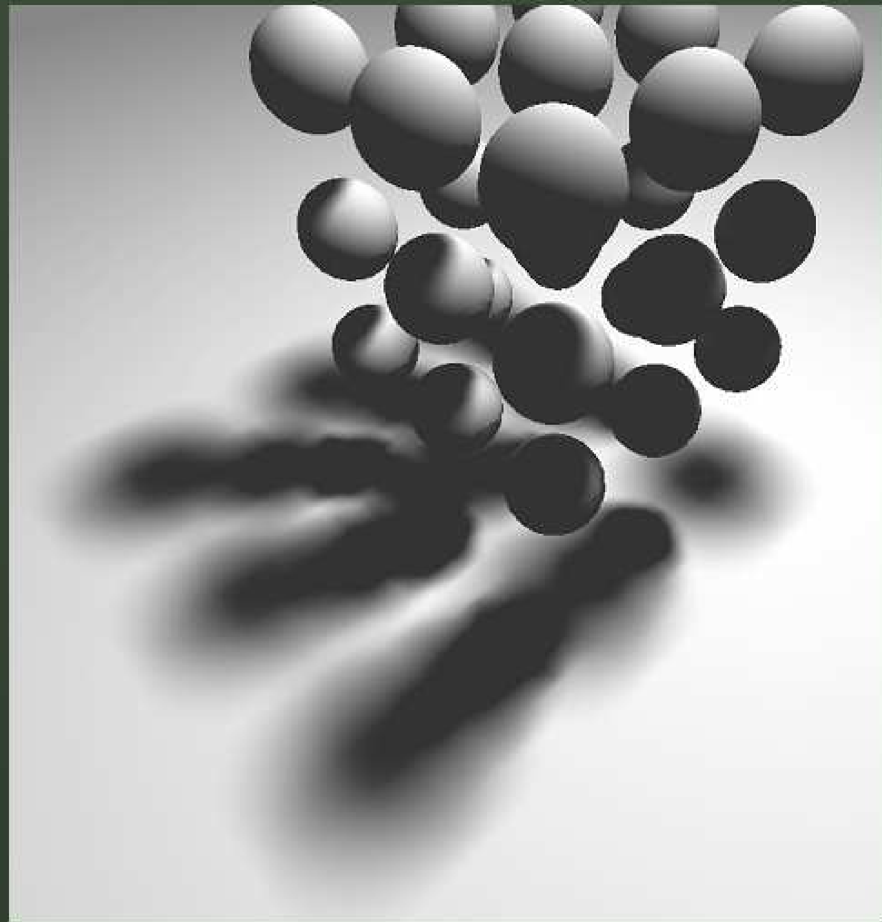


naive algorithm

Gaps filling



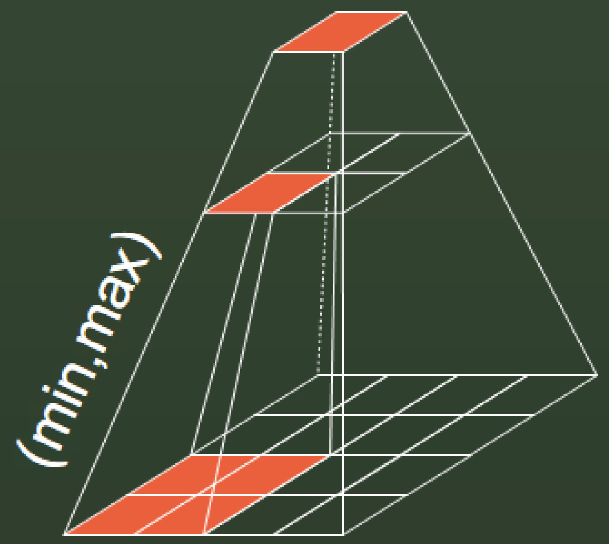
reference



with gap filling

Optimizations
hierarchical shadow map (HSM)

- shadow map → hierarchical shadow map (HSM)
  - similar to mipmaps
  - each pixel stores the min and max depth values



Summary of the algorithm

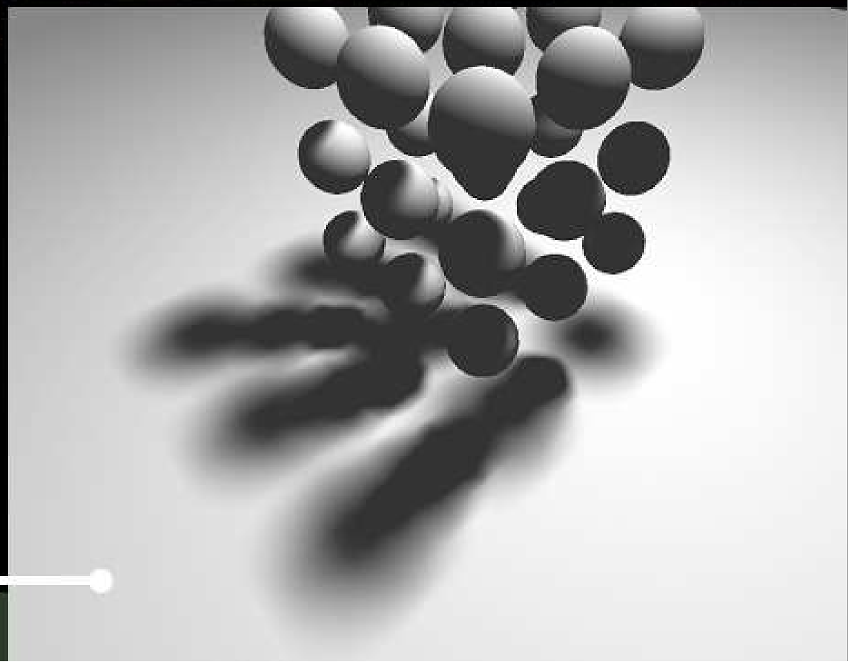
- Draw the scene in the shadow map
- Compute the HSM (GPGPU, ~3 ms)
- Draw the scene from the view point in a depth buffer
  - ~ deferred shading
- Compute the visibility buffer:
  - **for** each pixel **p** *(draw a quad)*
    - estimate the occluder search area (HSM)
    - if p is lit or in the umbra then **OK**
    - **else** loop over the occluder samples...
      - ~ 15 instructions / sample

dynamic branching

- Draw the scene with lighting and soft shadows !

performances
(on a GeForce 7800)

Scene	Fig. 7	Fig. 1	Fig. 8
Shadow map	1.7	2.6	8.7
Camera depth map	0.7	1.3	7.6
HSM construction	3.1	3.1	3.1
Visibility pass 1	0.9	0.9	0.9
Visibility pass 2	39	28	15
Final rendering pass	0.8	1.6	8.2
Total (ms)	46.2	37.5	43.5
fps	21.6	26.6	23



Soft shadow mapping conclusion

- Summary
  - provides high quality soft shadows in real-time
    - not physically exact, but close in most cases
  - has all the advantages of shadow maps
    - suitable for complex scenes
    - suitable for any rasterizable geometry
    - no pre-computation => dynamic scenes

Geometry-based Methods

- Discontinuity mesh and backprojection
- ✓ Penumbra wedges
- ✓ Soft shadow volumes

Image-based Methods

- Multi-layered shadow map
- Extended shadow map
- ✓ Soft shadow mapping by backprojection

Other Methods

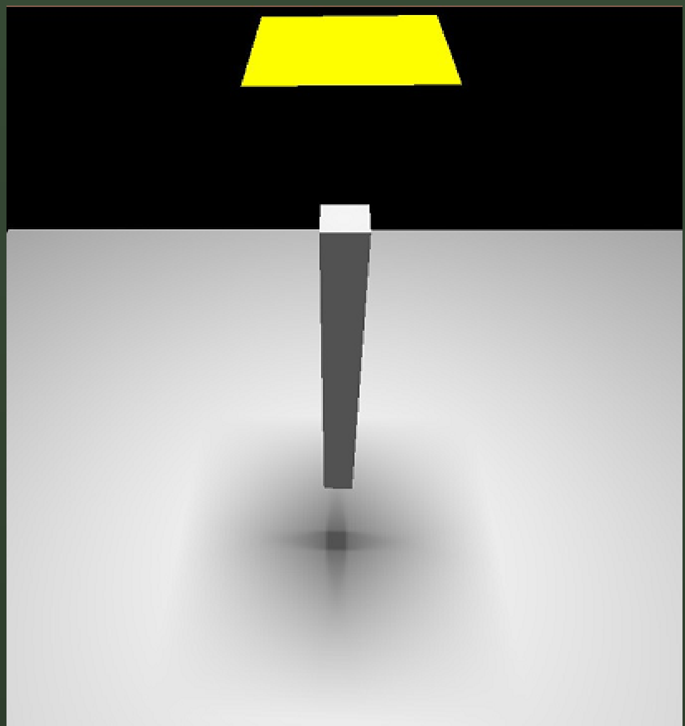
- Occlusion camera
- Spherical harmonics
- ✓ Ray tracing/Radiosity

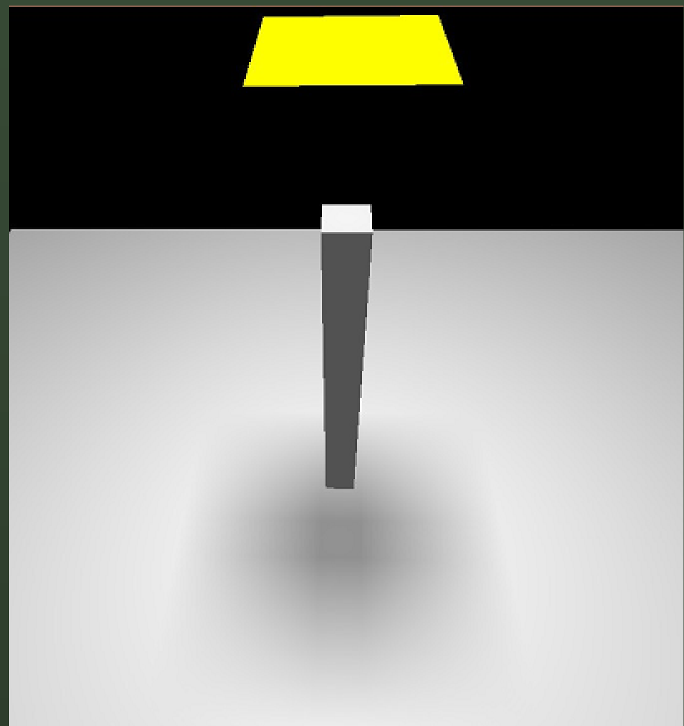
Summary

- Performance / Scalability
- Polygonal scenes or Other scenes
- Plausible or Physically accurate
- Common artifacts
  - *- Single sample artifact*
  - *- Occlusion fusion artifact*

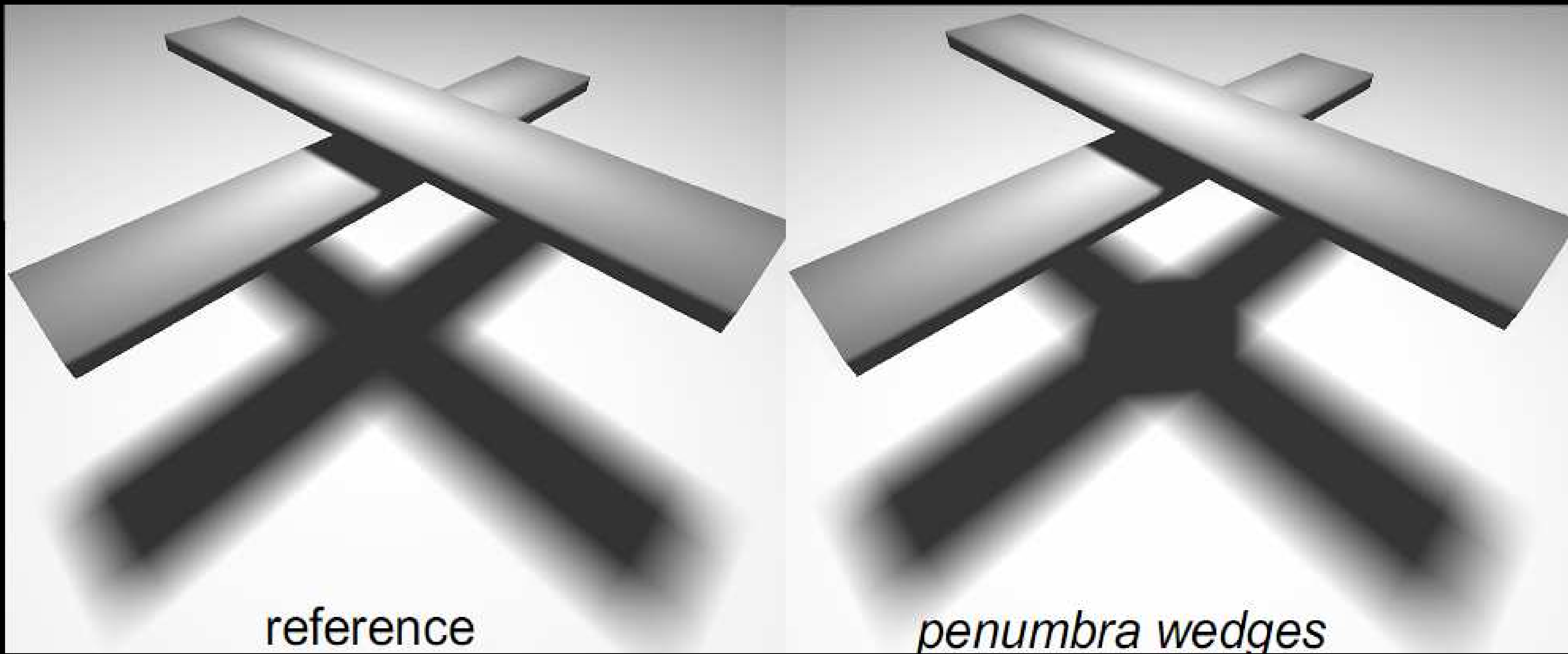
Single Sample Artifact

- Only parts visible from the light center are taken into account in the visibility computation





Occlusion Fusion Artifact



Reading List

- Tomas Akenine-Möller and Ulf Assarsson, “Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges.” *13th Eurographics Workshop on Rendering 2002*, pp. 309-318, June 2002.
- Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen and Tomas Akenine-Möller, “Soft Shadow Volumes for Ray Tracing." *ACM SIGGRAPH 2005*.
- Randima Fernando, “Percentage-closer soft shadows.” ACM SIGGRAPH 2005 Sketches.
- Gaël Guennebaud, Loïc Barthe and Mathias Paulin. “Real-time Soft Shadow Mapping by Backprojection.” *Eurographics Symposium on Rendering 2006*

Additional Reference

- A survey of Real-Time Soft Shadows Algorithms

  http://artis.inrialpes.fr/Publications/2003/HLHS03a/
- Shadow Rendering Page at Lund University

  http://graphics.cs.lth.se/research/shadows/
- Real-time Rendering

  http://www.realtimerendering.com/