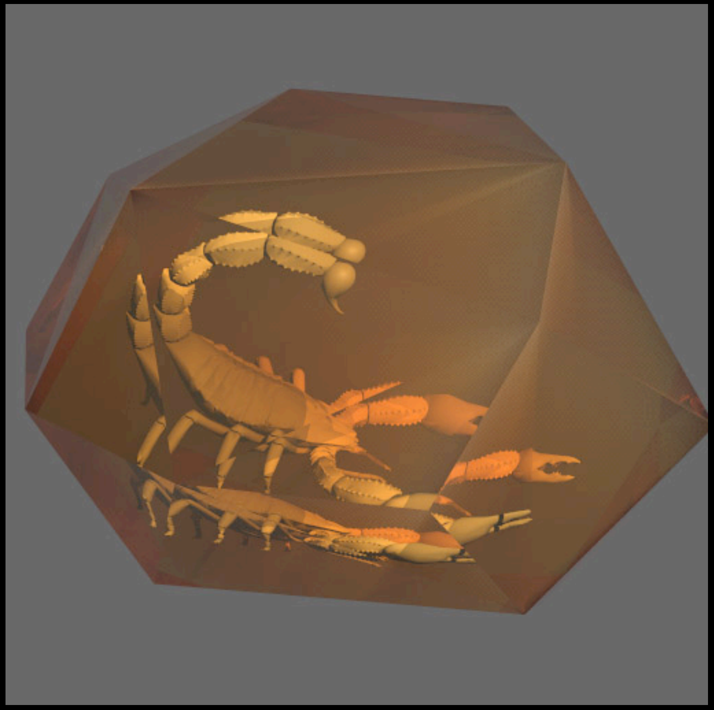
Single Scattering in Refractive Media with Triangle Mesh Boundaries

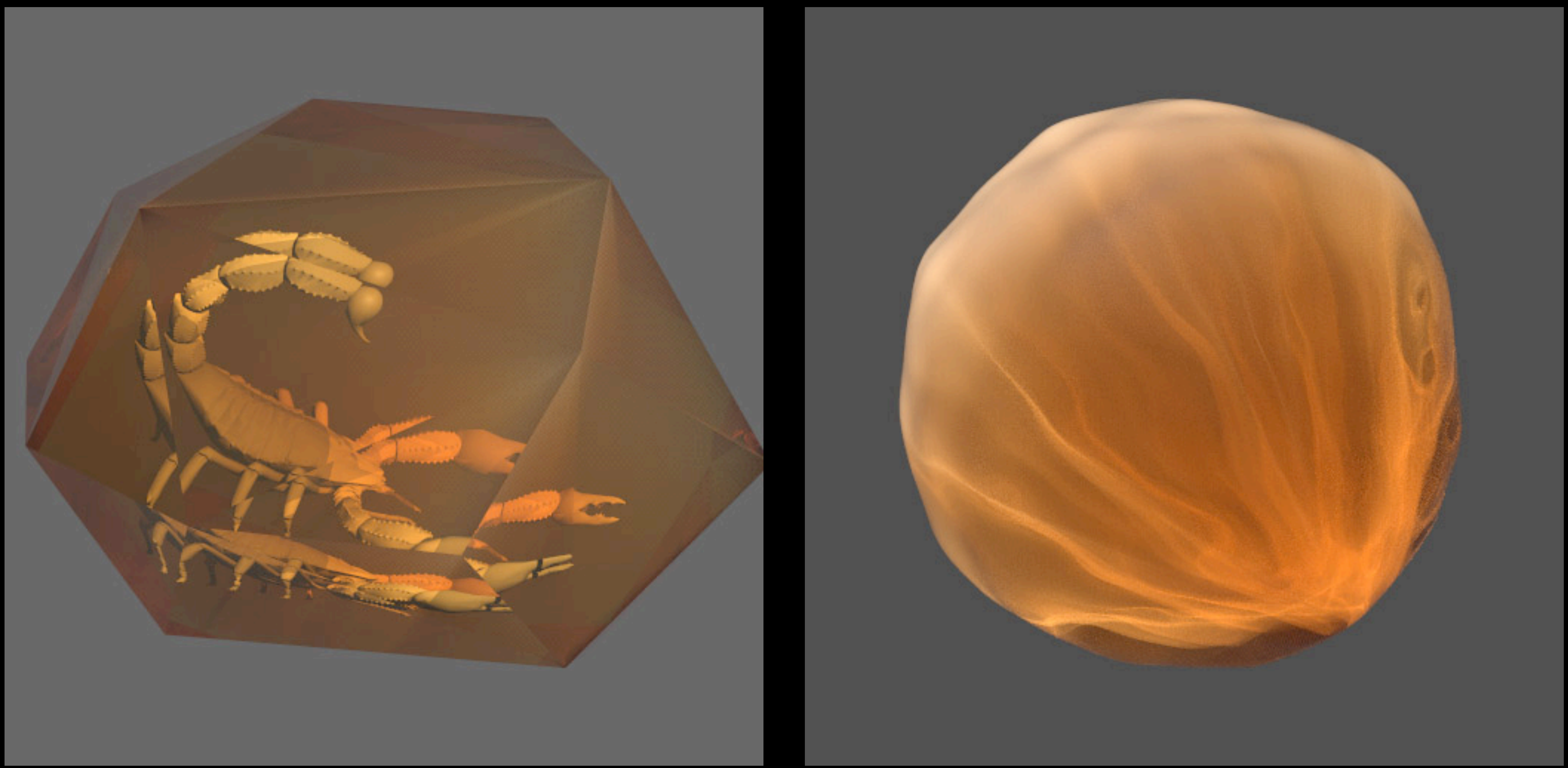


Bruce Walter Cornell Univ.
Shuang Zhao Cornell Univ.
Nicolas Holzschuch Grenoble Univ.
Kavita Bala Cornell Univ.

Presented at SIGGRAPH 2009

Single Scattering

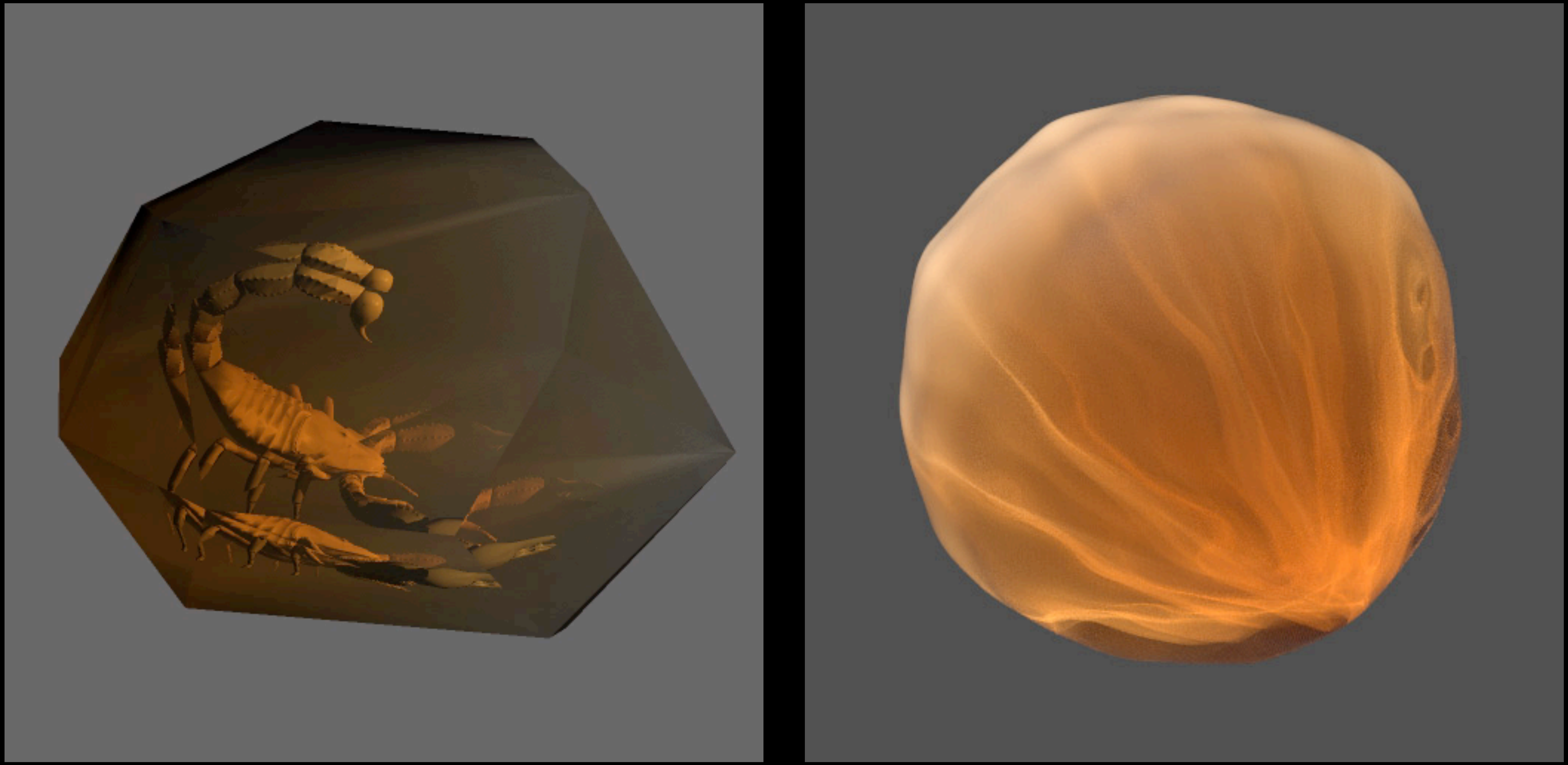
- Direct illumination in refractive objects is hard



Single scatter from a single point light source

Single Scattering

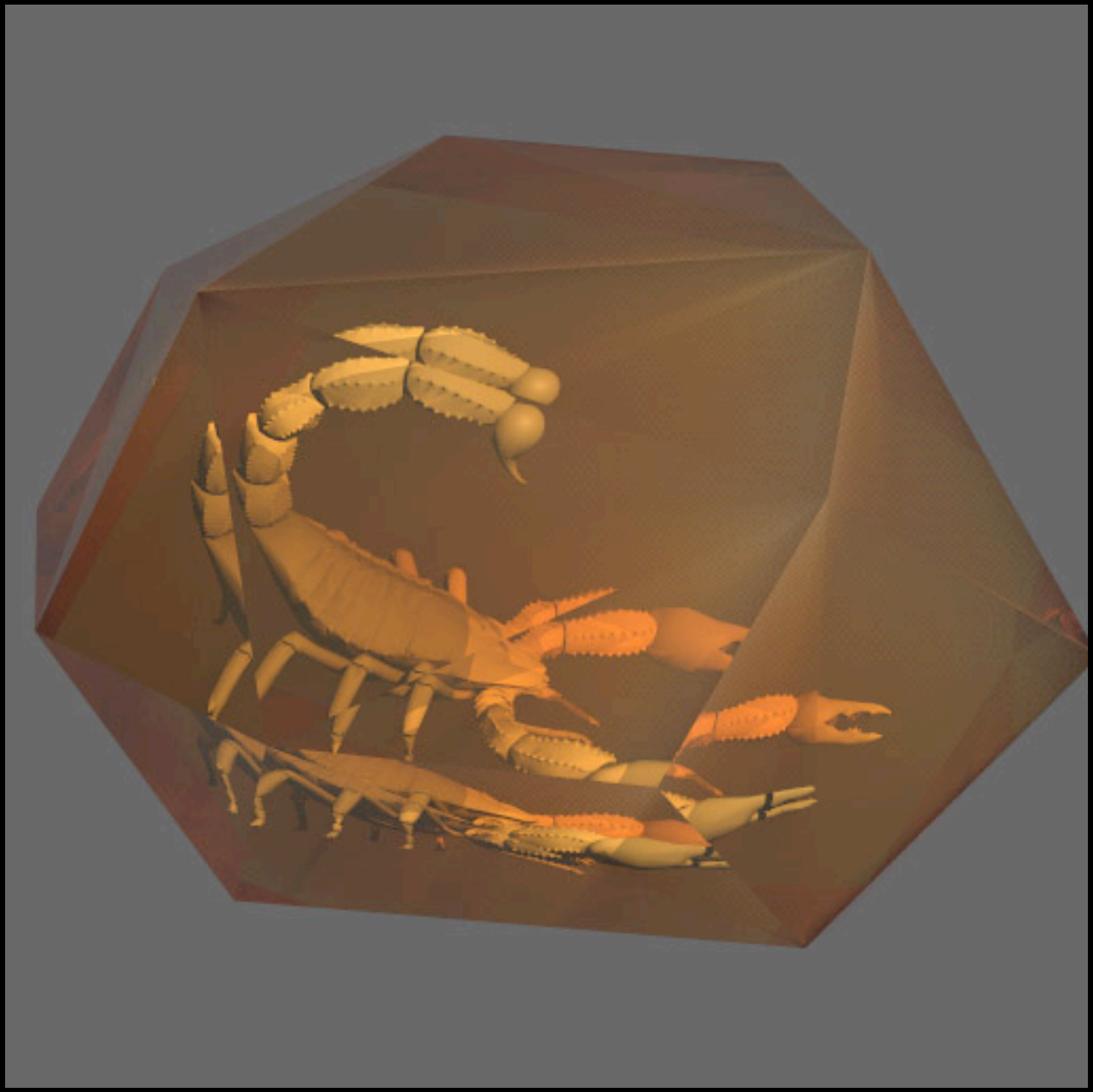
- Direct illumination in refractive objects is hard

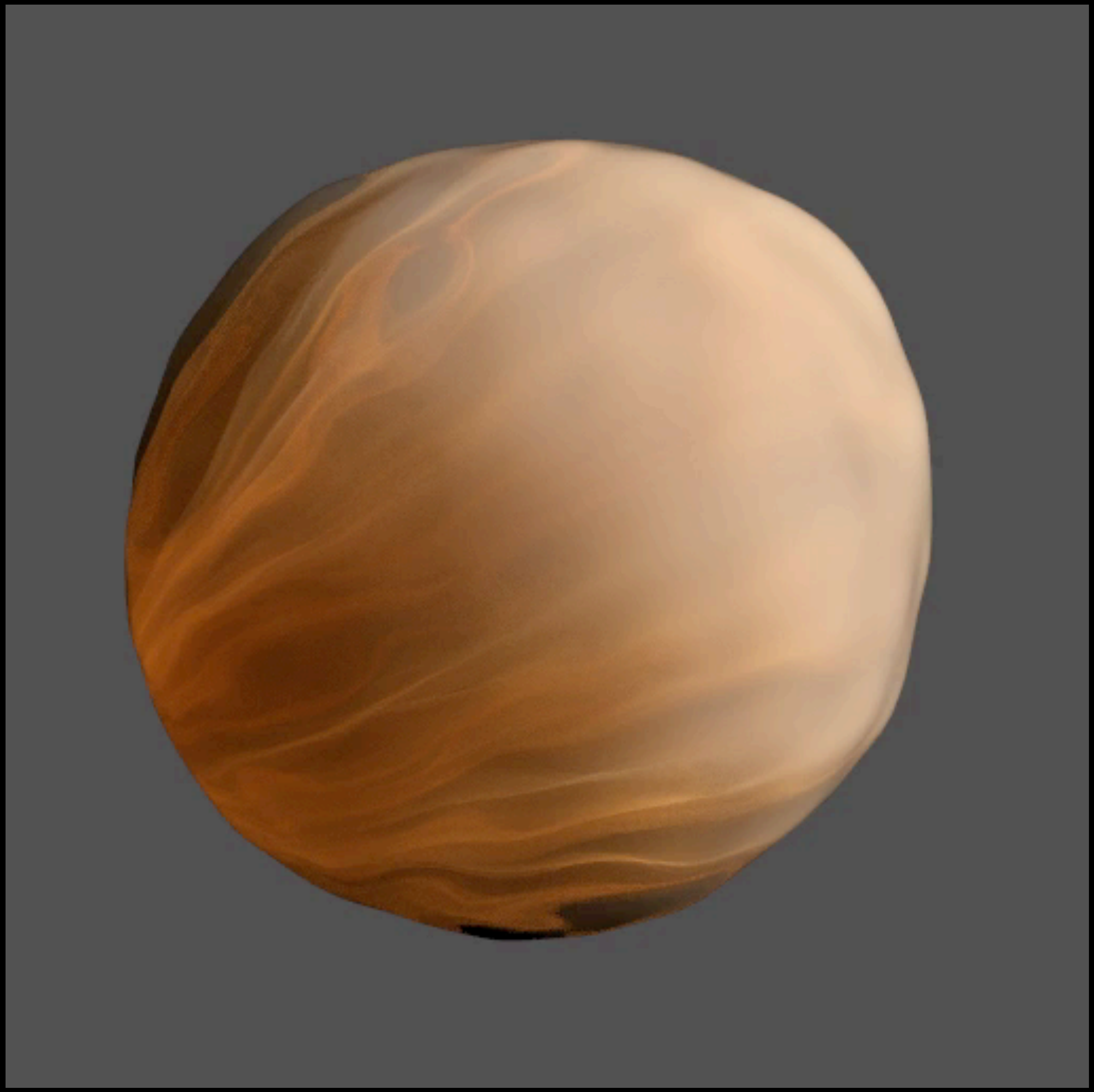


Single scatter from a single point light source

Single Scattering

- Direct illumination in refractive objects is hard



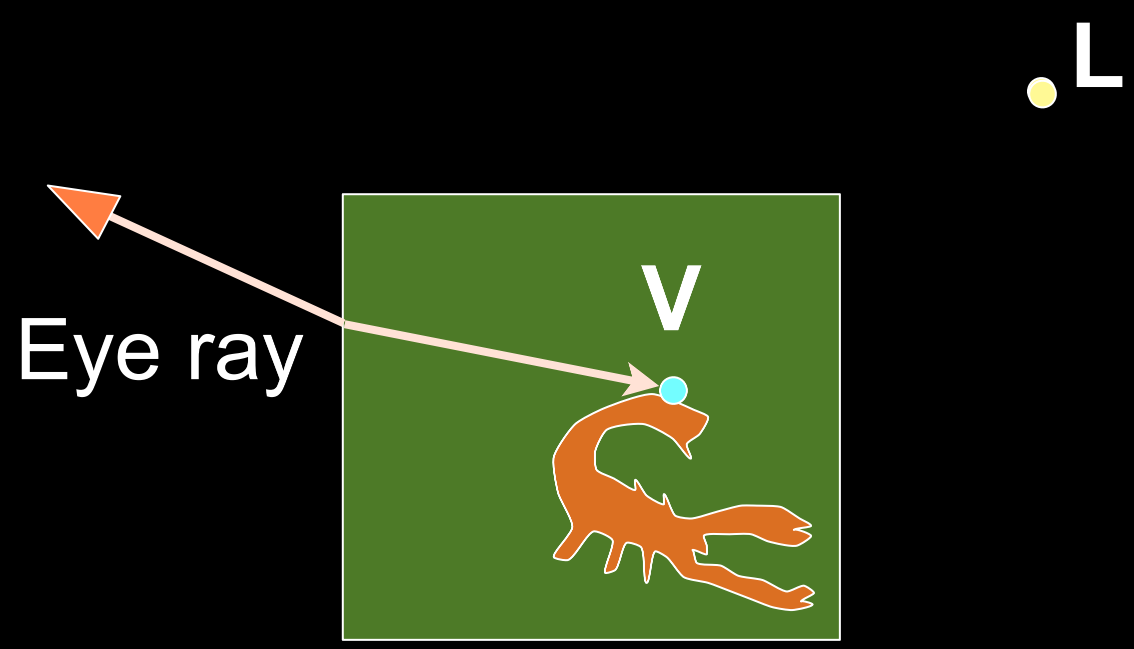


Single scatter from a single point light source

Problem

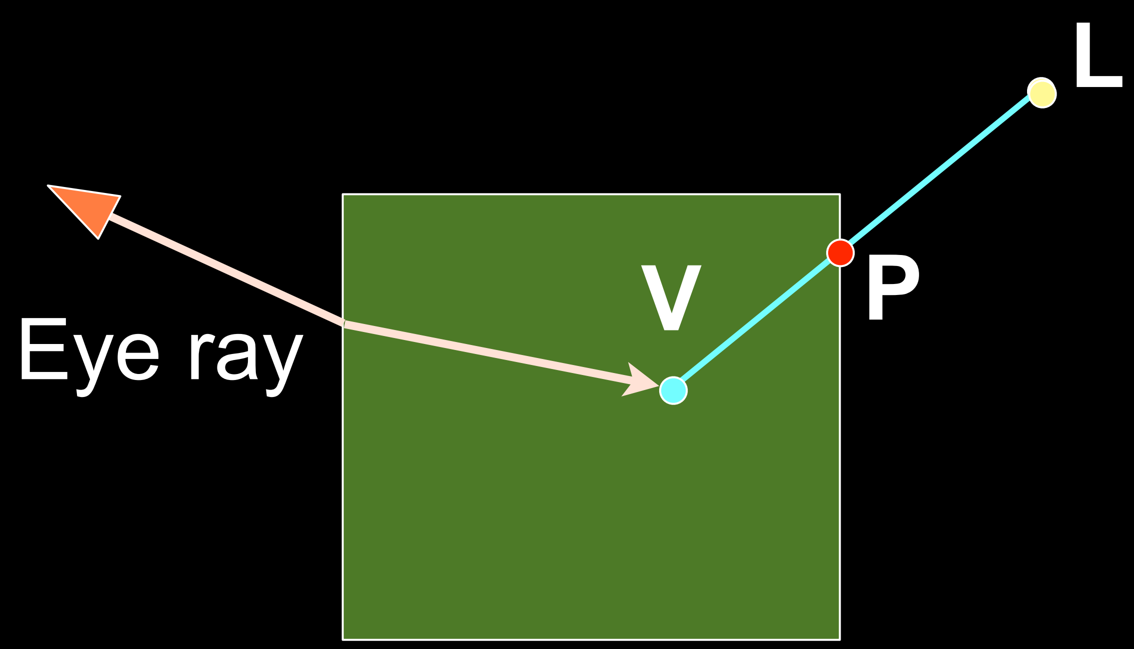
Problem

- Find direct illumination at V (receiver) from L (light)



Problem

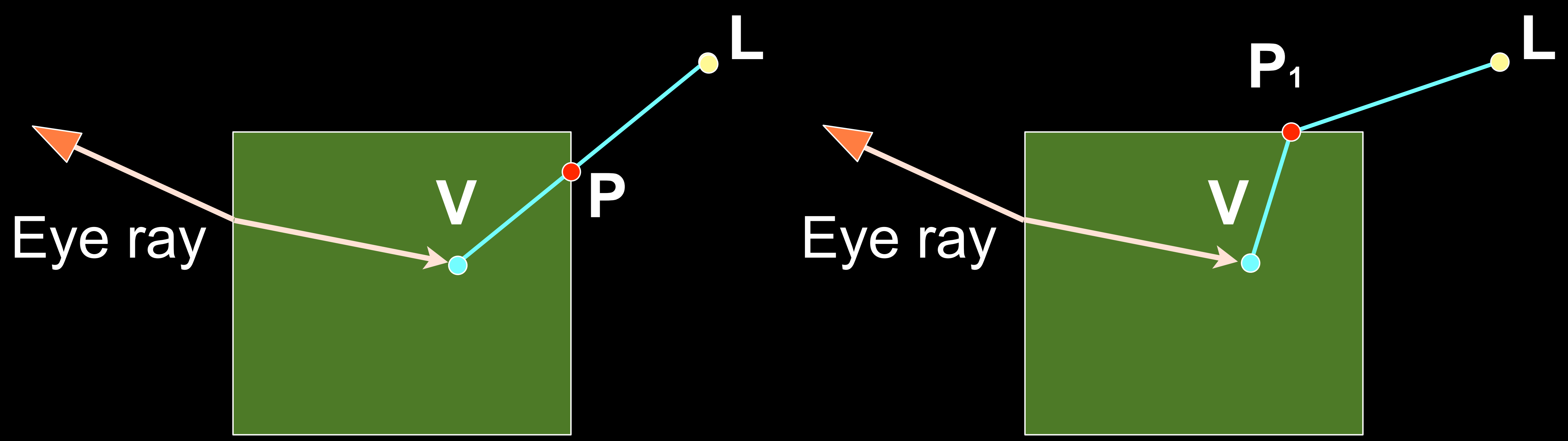
- Find direct illumination at V (receiver) from L (light)



Ignoring refraction

Challenges: Bending of Path

- Find direct illumination at V (receiver) from L (light)
- Light bends at interface according to Snell's Law

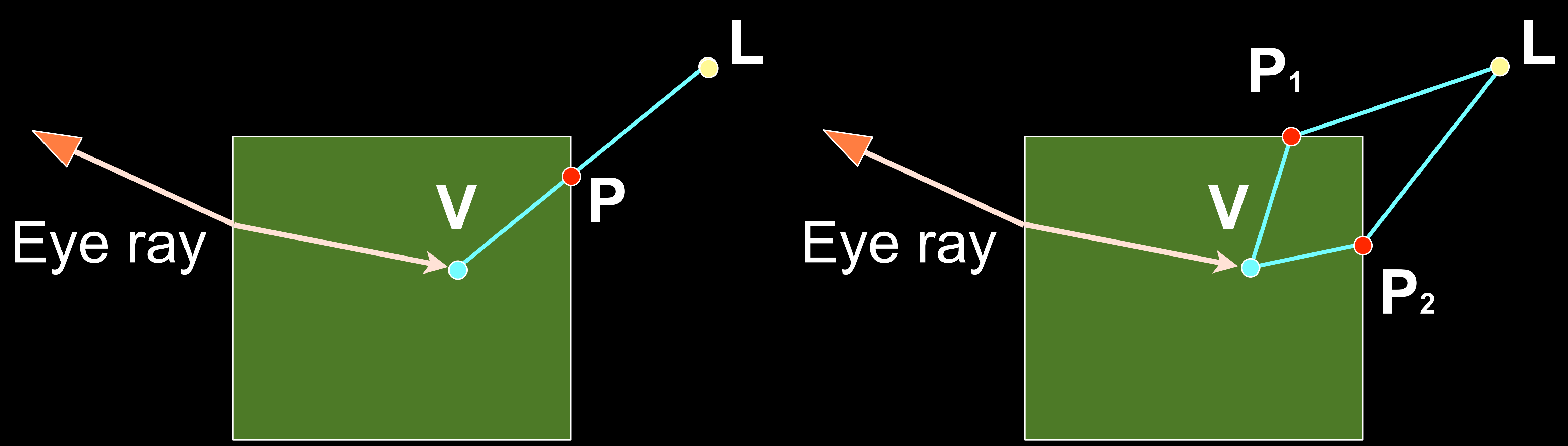


Ignoring refraction

With refraction

Challenges: Multiple Paths

- Find direct illumination at V (receiver) from L (light)
- Light bends at interface according to Snell's Law
  - Can have zero, one, or many such paths (and P's)

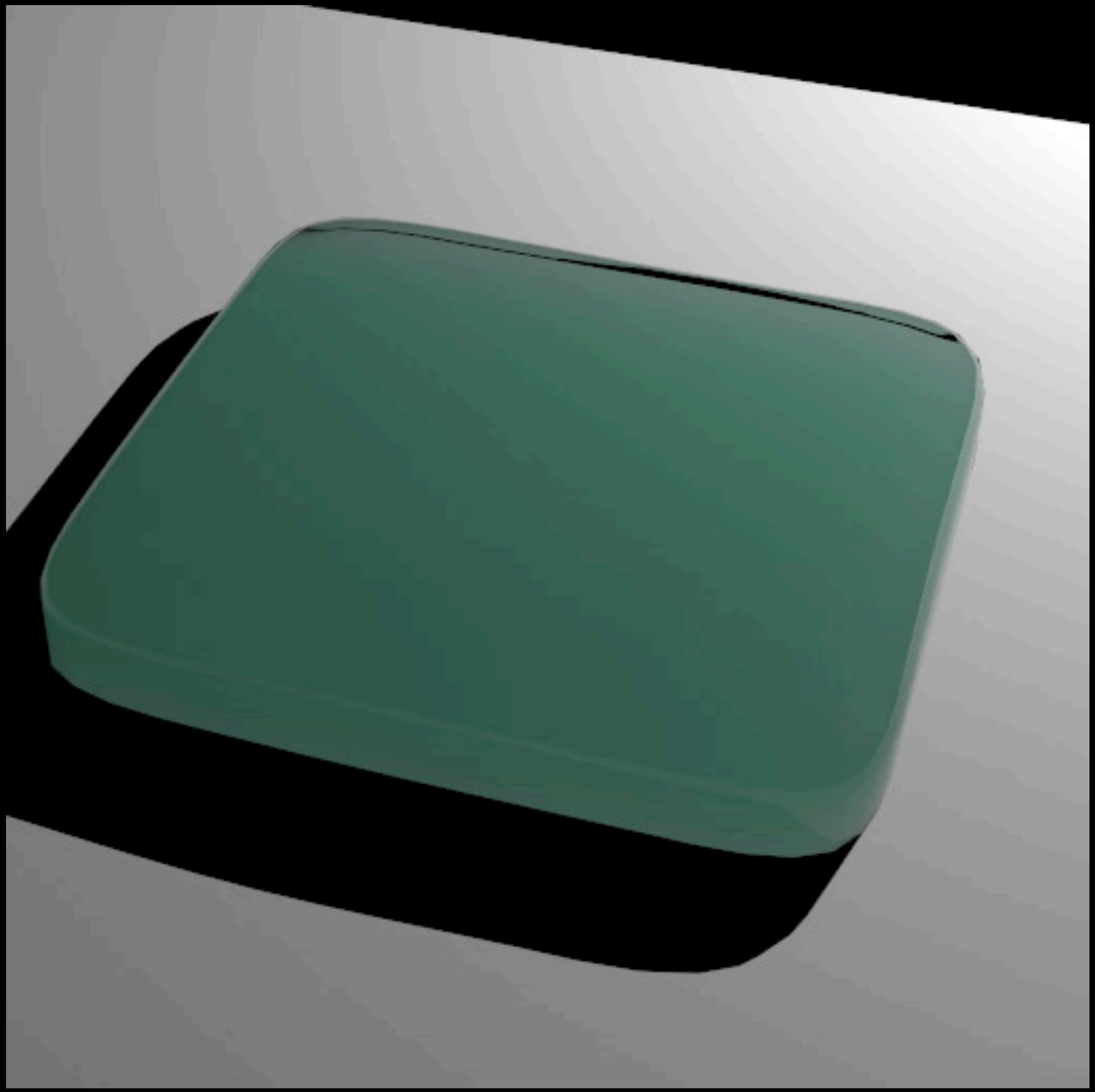


Ignoring refraction

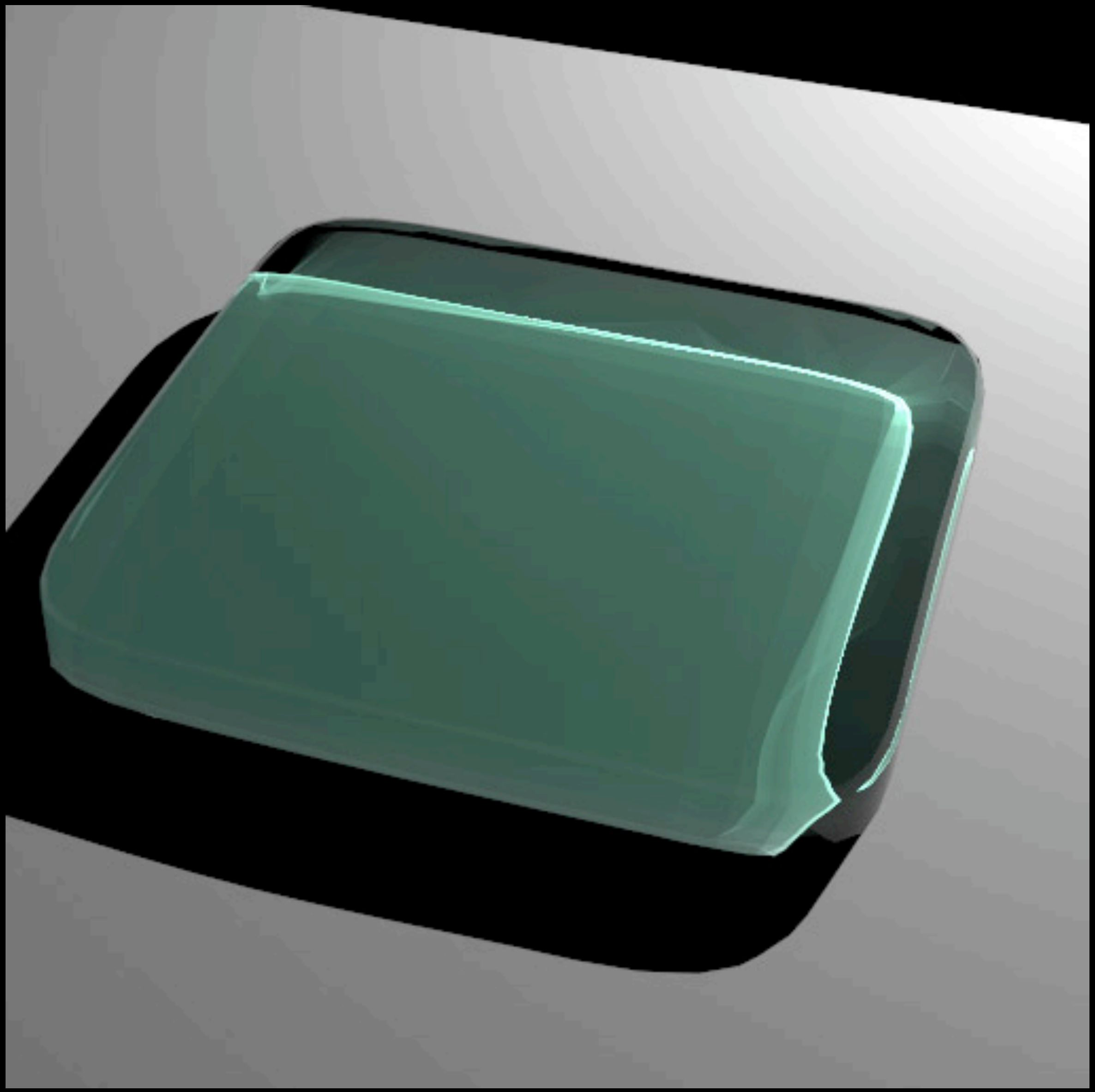
With refraction

Is it important?

- Glass tile quality comparison



Shadow rays ignore refraction



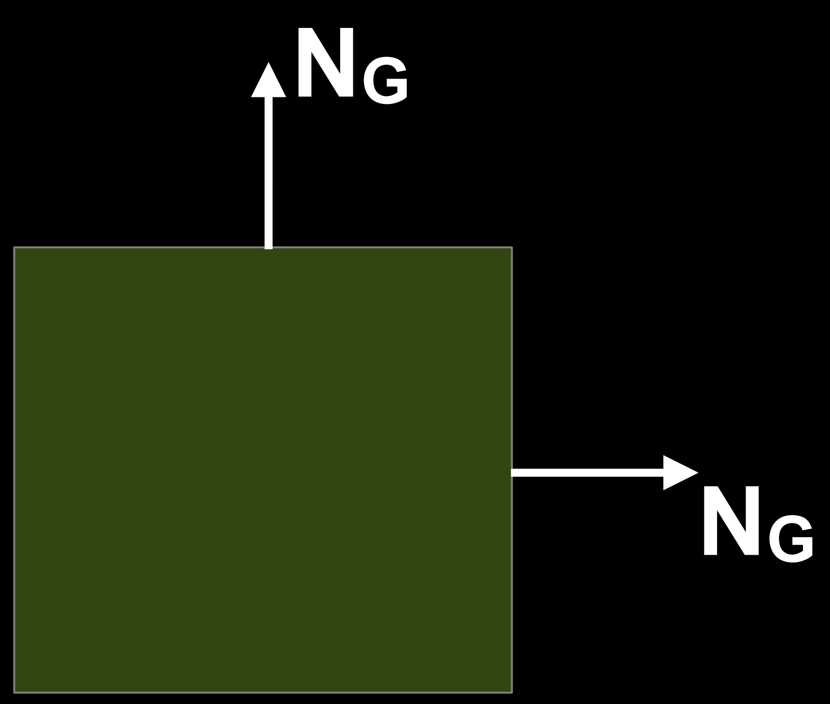
Our method

Challenges Summary

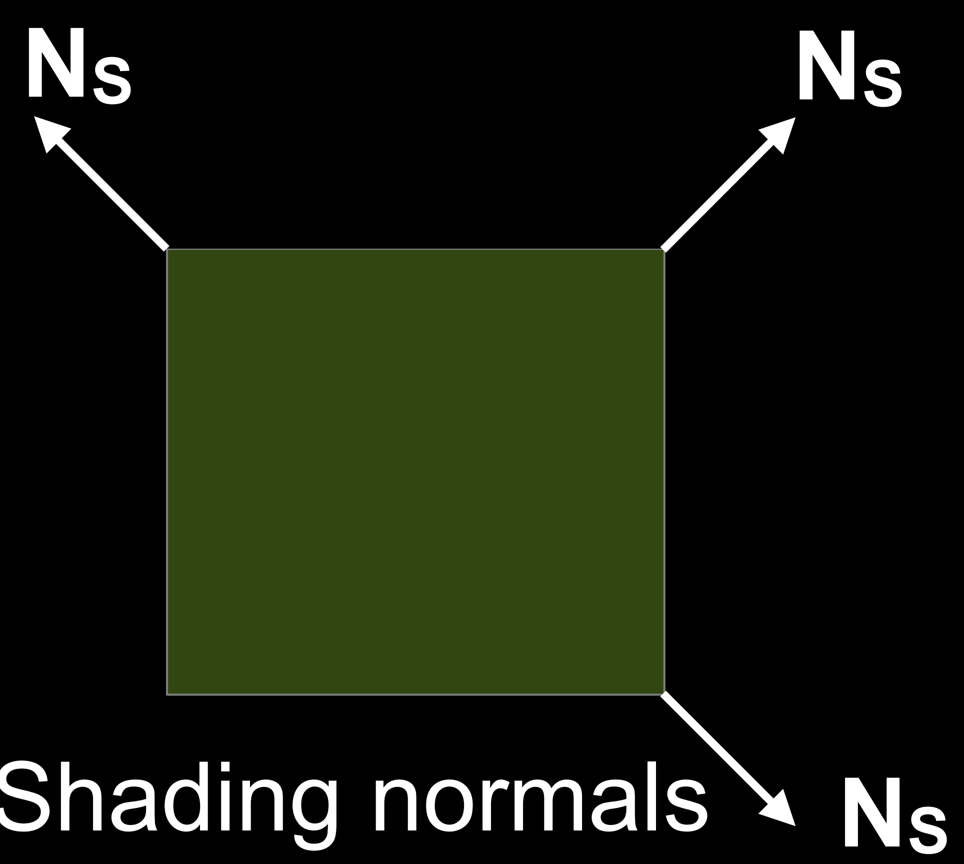
- Bending of paths
- Multiple paths
- Shading normals
- Large triangle meshes

Challenges: Shading Normals

- Geometric normals (N_G) vs. shading normals (N_S)
  - E.g., interpolated normals, bump maps, normal maps
  - Alters directions and intensities of light paths



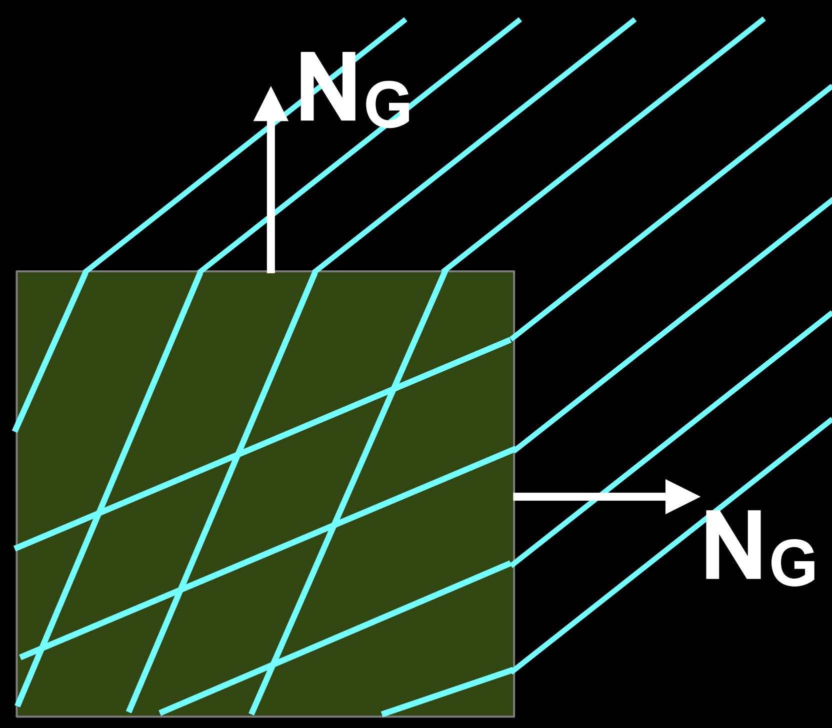
Geometric normals



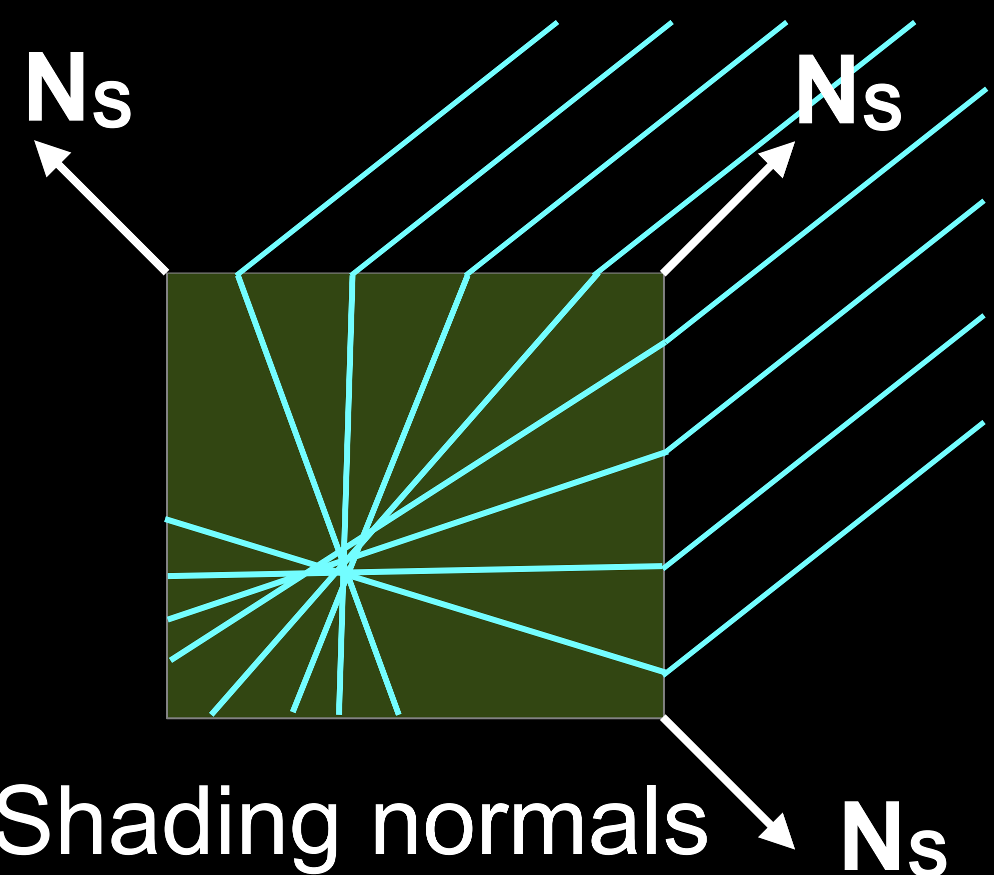
Shading normals

Challenges: Shading Normals

- Geometric normals (N_G) vs. shading normals (N_S)
  - E.g., interpolated normals, bump maps, normal maps
  - Alters directions and intensities of light paths



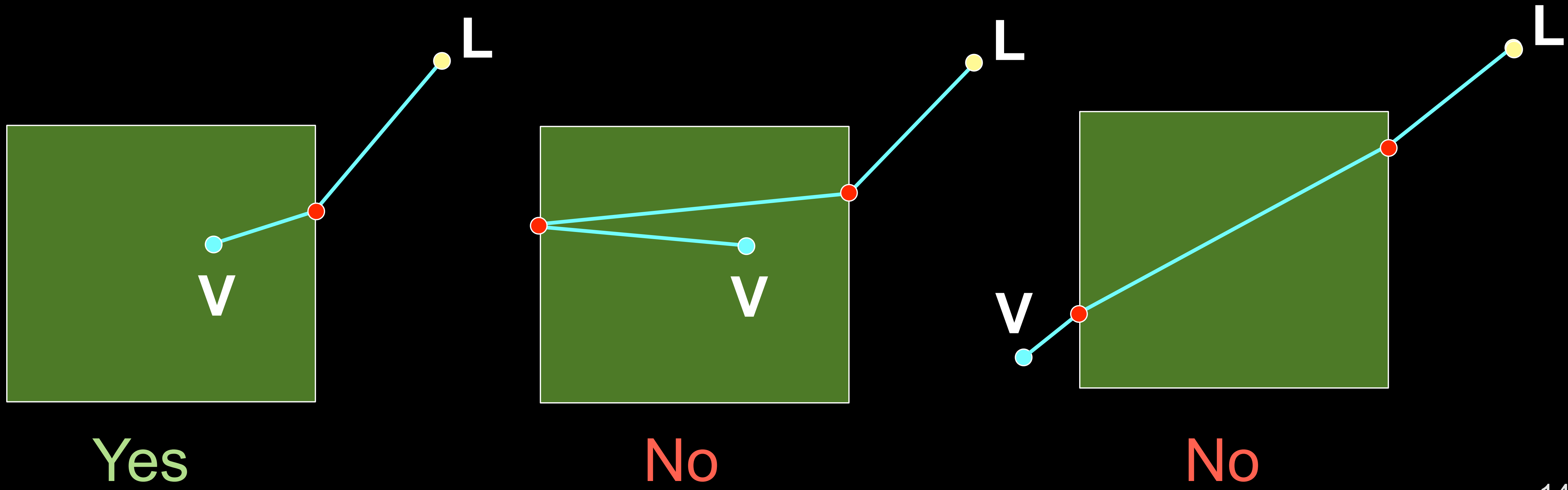
Geometric normals



Shading normals

Limitations

- Finds connections that
  - Cross the boundary exactly once
  - Have no other changes in direction
  - Cost depends on path count and boundary

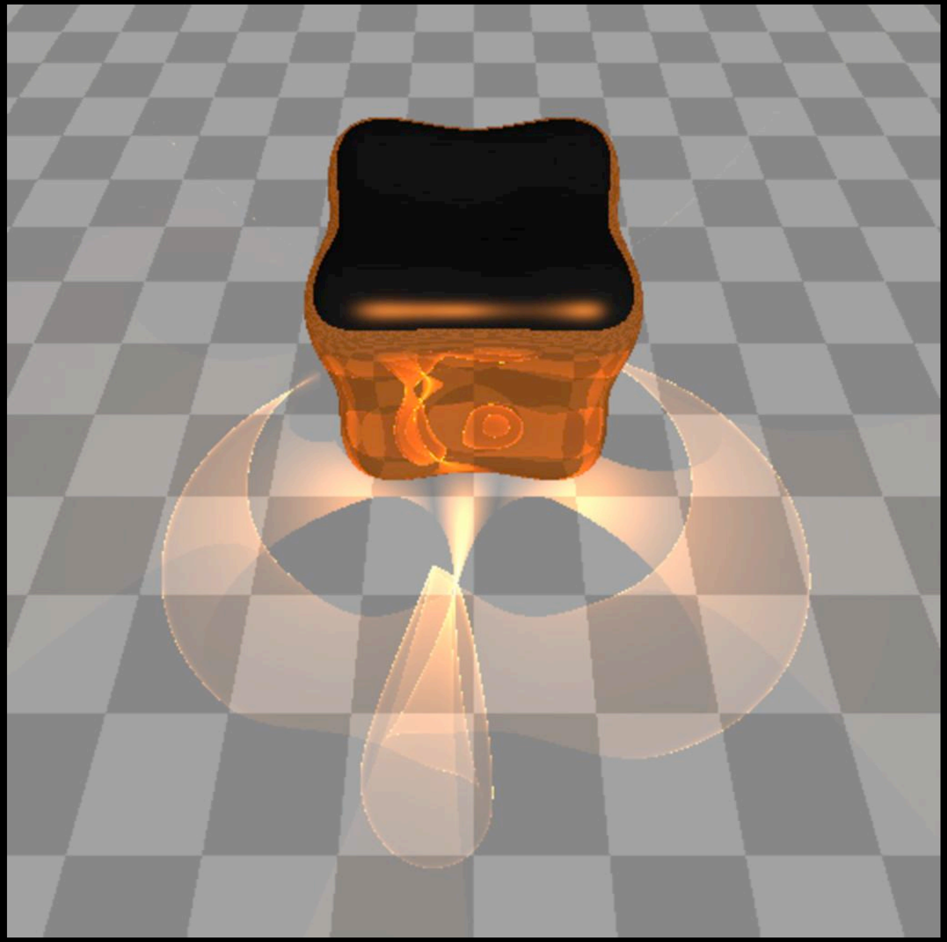


Prior Work

- Diffusion and multiple scatter
  - [eg, Stam 95, Jensen et al. 01, Wang et al. 08]
- Monte Carlo
  - [eg, Kajiya 86, Veach 97]
- Beam tracing
  - [eg, Nishita & Nakamae 94, Iwasaki et al. 03, Ernst et al. 05]
- Photon mapping
  - [eg, Jensen 01, Sun et al. 08, Jarosz et al. 08]
- Fermat's principle
  - [eg, Mitchell & Hanrahan 92, Chen & Arvo 00]

Prior Work

- Mitchell & Hanrahan 92
  - Used Fermat's principle and Newton's method
  - Reflection (shown) and refraction
- Limitations
  - Only supported implicit surfaces
  - Cannot handle shading normals
  - Expensive 3D search
  - Not scalable to complex geometry



Mitchell & Hanrahan 92

Contributions

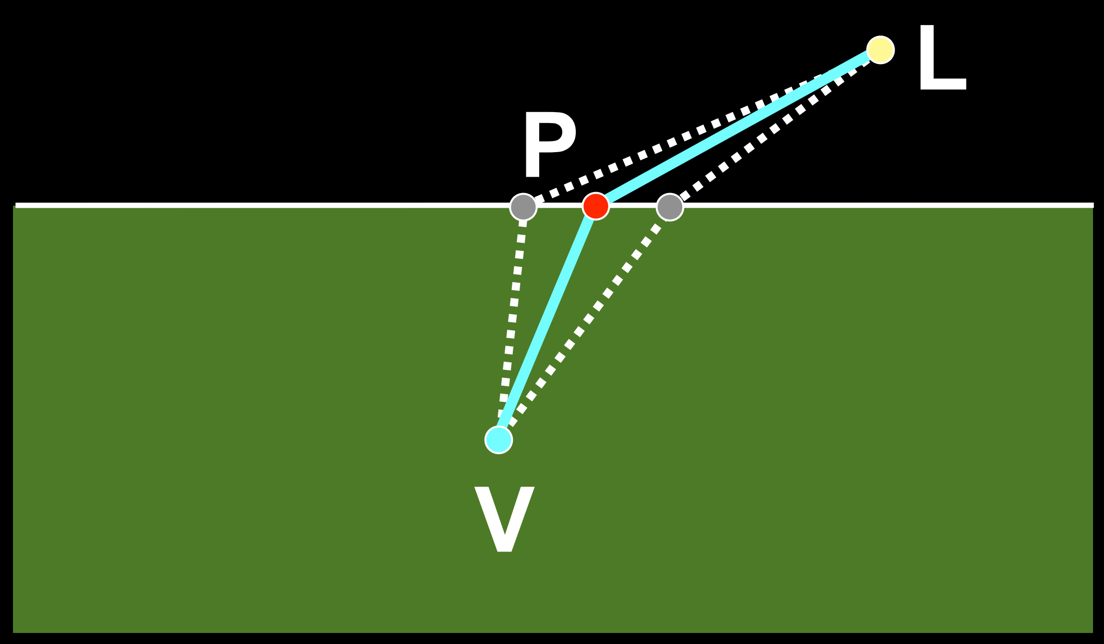
- Support triangles with shading normals
  - Most widely used geometry format
  - Required fundamental problem reformulation
  - New search methods and intensity equations
- Hierarchical culling
  - Scales to complex objects with many triangles
- CPU and GPU implementations
  - Interactive performance on some scenes

Outline

- Half-vector formulation
- Solving for a single triangle
- Hierarchical culling for meshes
- Results

Fermat's Principle

- Define optical path length
  - $\mathbf{d}(P) = \eta \, ||V-P|| + ||P-L||$
  - Extrema of $\mathbf{d}(P)$ are the refraction points
- Cannot handle shading normals

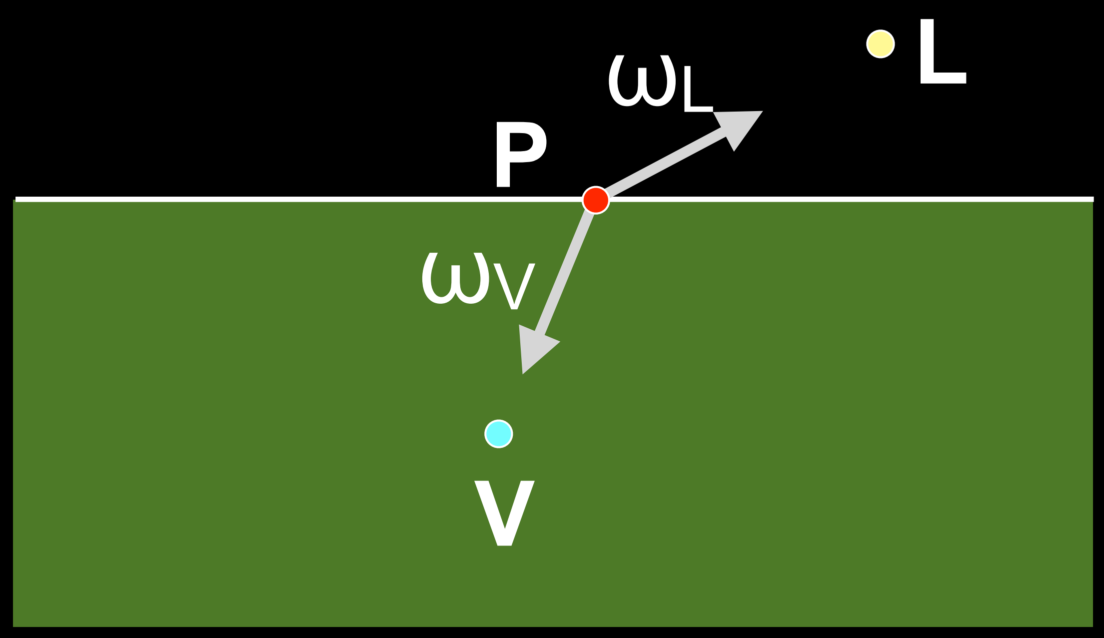


Index of refraction = η

Half-Vector Formulation

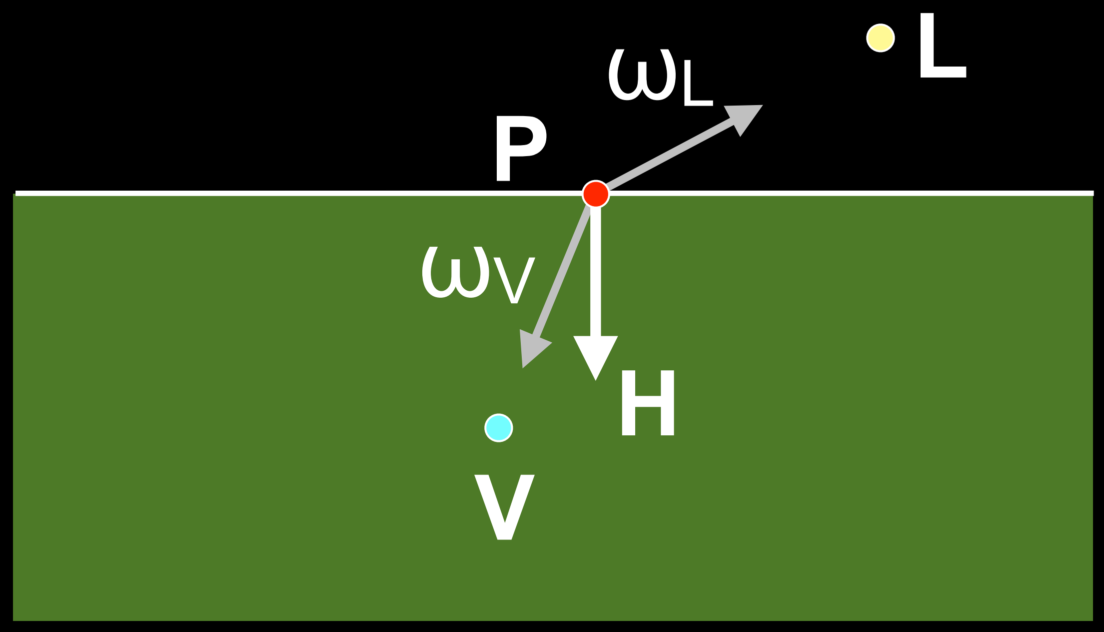
- Used in micro-facet model [Walter et al. 07]

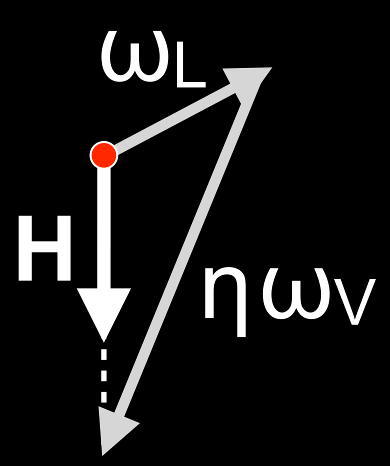
- Direction to receiver: $\omega_V = (V - P) / ||V - P||$
- Direction to light: $\omega_L = (L - P) / ||L - P||$



Half-Vector Formulation

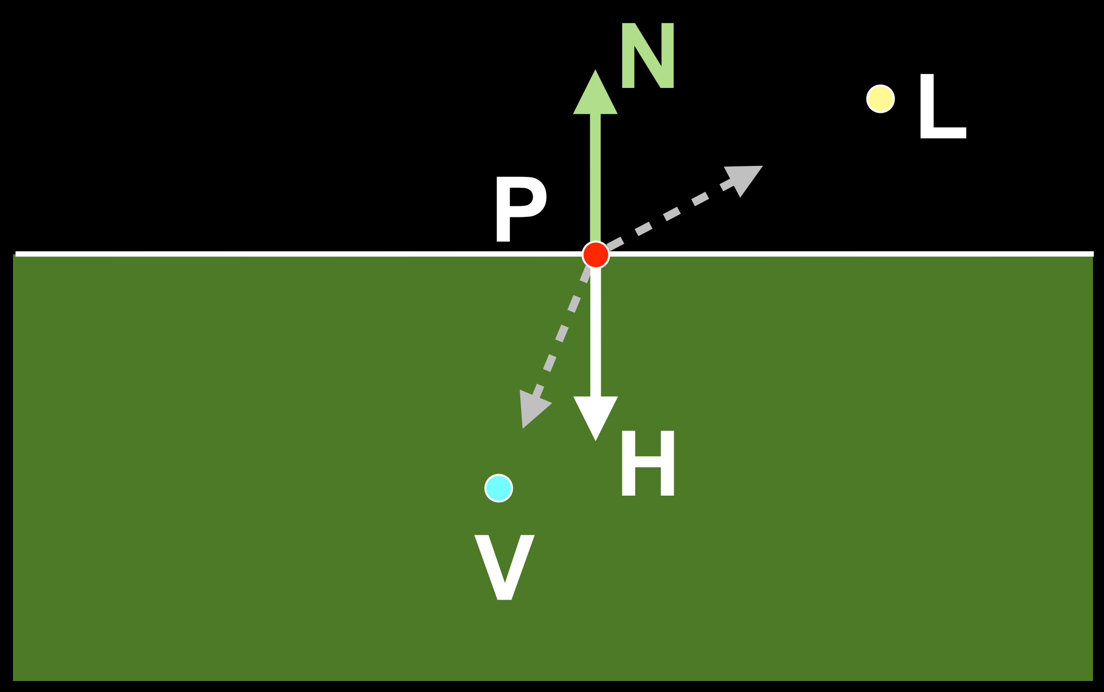
- Used in micro-facet model [Walter et al. 07]
- Direction to receiver: $\omega_V = (V - P) / ||V - P||$
- Direction to light: $\omega_L = (L - P) / ||L - P||$
- Half-vector: $H = (\eta\,\omega_V + \omega_L) / ||\eta\,\omega_V + \omega_L||$





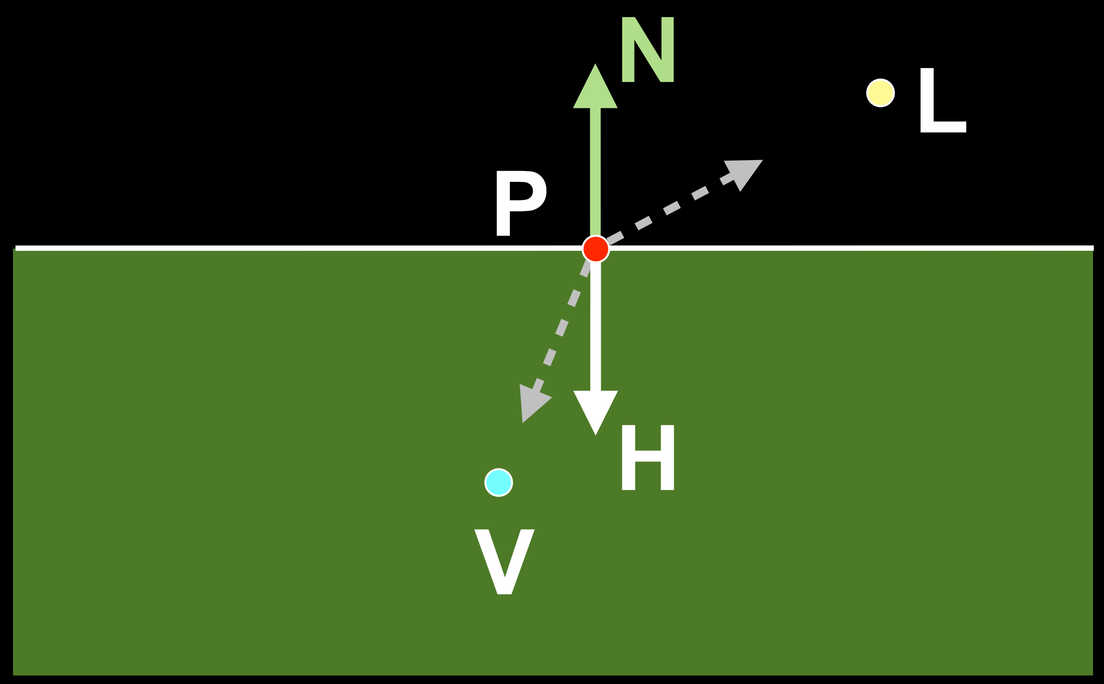
Half-Vector Formulation

- If H = -N (surface normal) then
  - V, P, L, and N are coplanar
  - Angles obey Snell's Law: $\eta \sin(\theta_V) = \sin(\theta_L)$
- It is a refraction solution
  - Assuming V and L lie on the correct sides of the normal



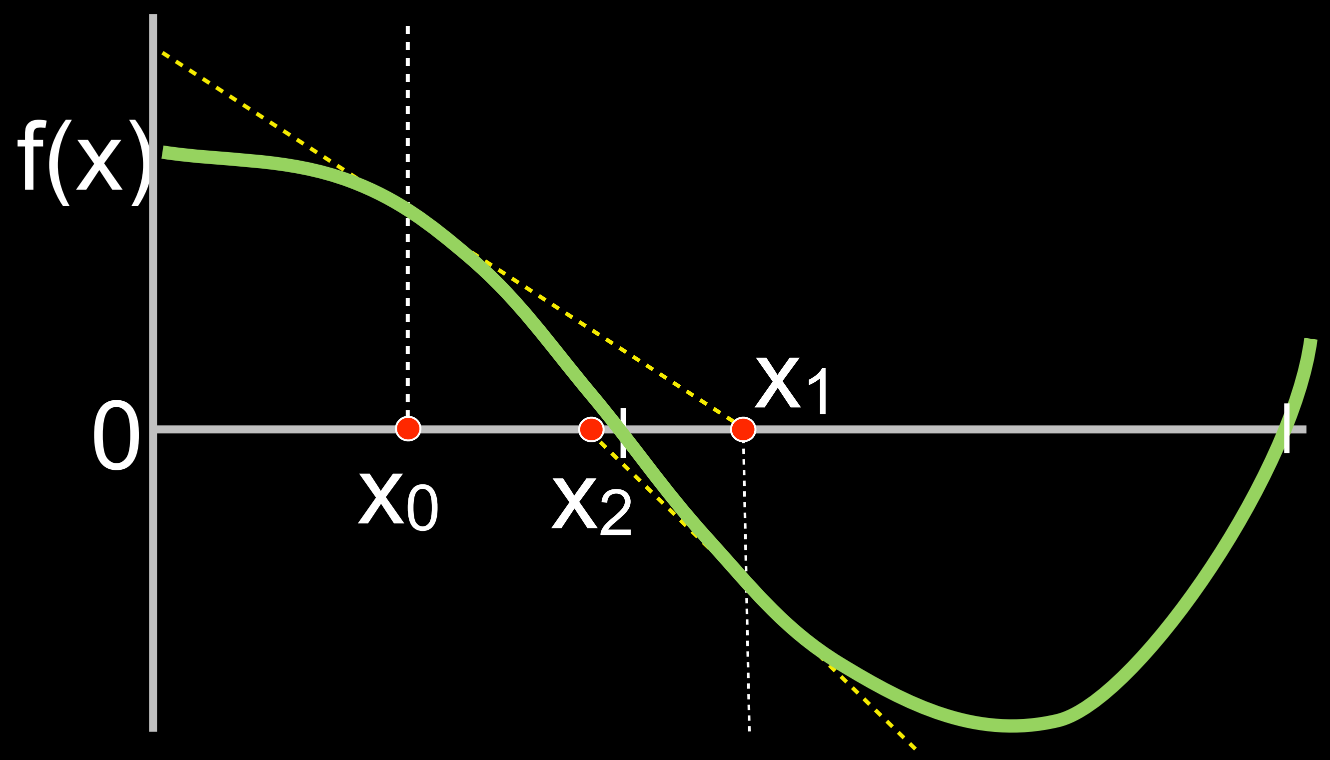
Half-Vector Formulation

- Find all P such that: H + N = 0
  - Natural extension to shading normals: $H + N_s = 0$
- Newton's method to find zeroes of: $\mathbf{f}(P) = H + N$



Newton's Method Review

- Quadratically convergent near a root
  - Each iteration doubles the precision
- Chaotic behavior far from a root
  - May diverge or converge to other roots

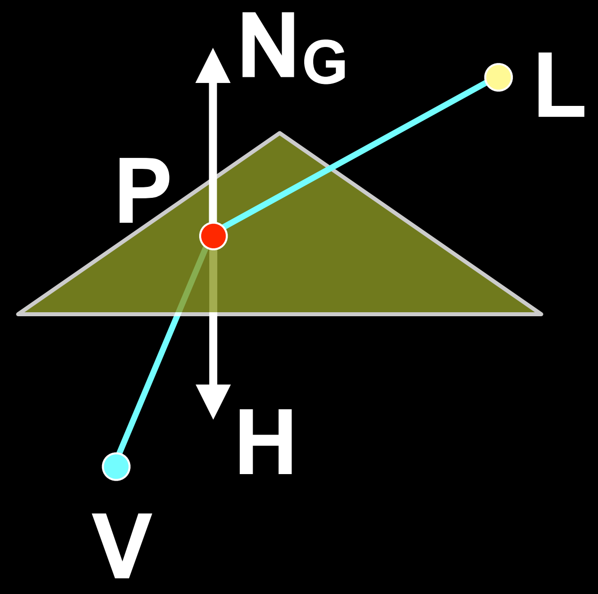


Outline

- Half-vector formulation
- Solving for a single triangle
  - Geometric normal - 1D Newton
  - Shading normals - 2D Newton
  - Subdivision oracles
- Hierarchical culling for meshes
- Results

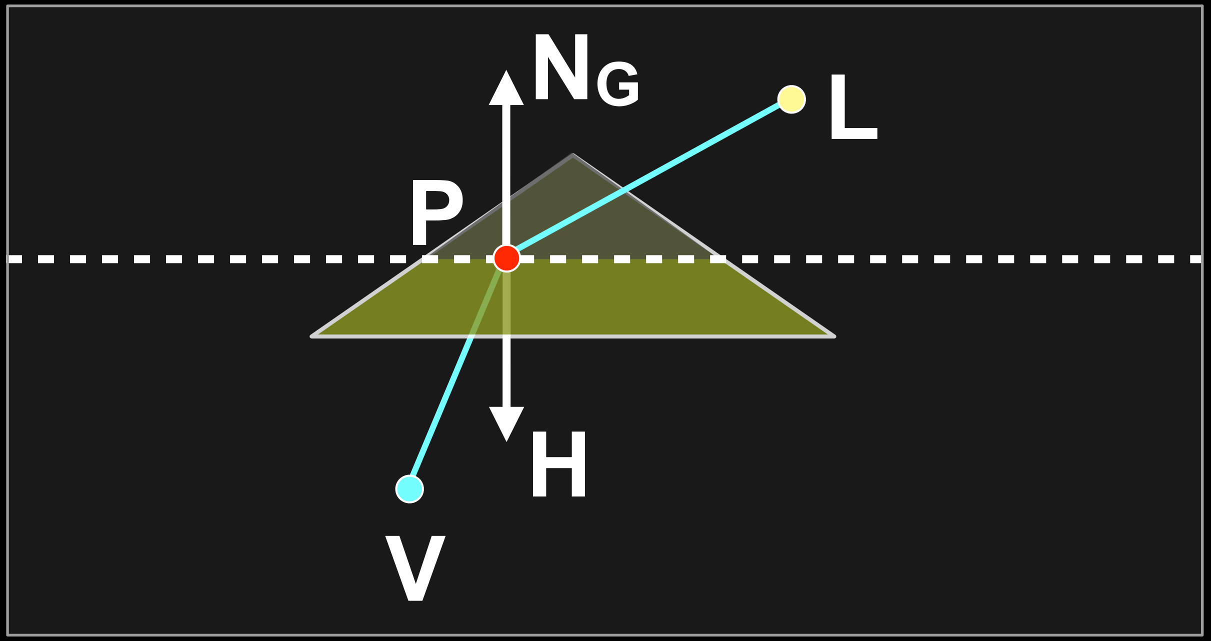
Triangle Without Shading Normals

- Solution must lie in plane containing V, L, and N_G
  - Unique solution always exists
  - Simple 1D Newton's method converges
    - Typically in just 2 to 4 iterations
  - Check if solution lies within the triangle



Triangle Without Shading Normals

- Solution must lie in plane containing V, L, and N_G
  - Unique solution always exists
  - Simple 1D Newton's method converges
    - Typically in just 2 to 4 iterations
  - Check if solution lies within the triangle



Triangle with Shading Normal

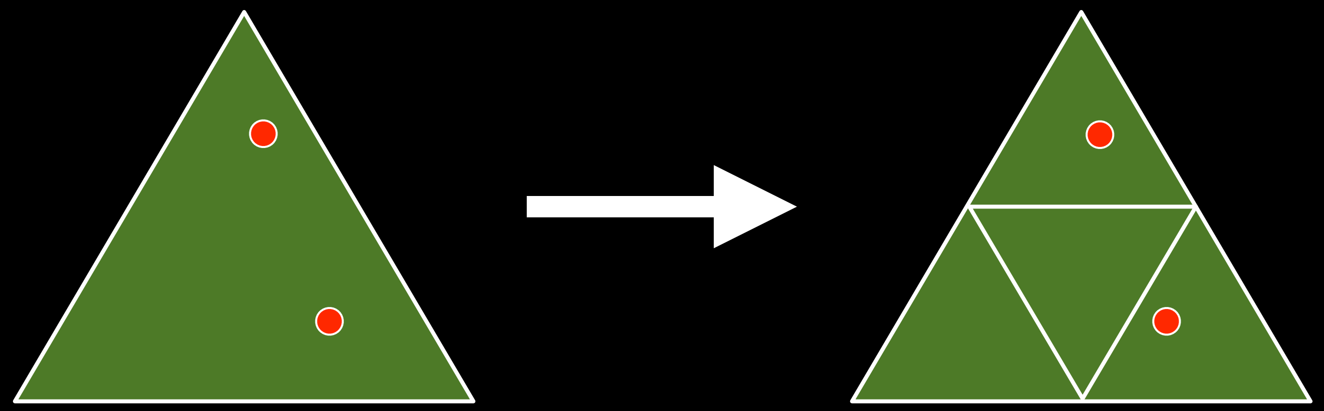
- Shading normal, N_S, varies over triangle
  - Full 2D search over triangle's area
- Function **f**(P) = H + N_S maps 2D to 3D
  - Derivative is 2X3 Jacobian matrix is non-invertible
  - Use pseudoinverse for Newton's method:

$$J^+ = (J^T J)^{-1} J^T$$

$$X_{i+1} = X_i - J^+(X_i)\mathbf{f}(X_i)$$

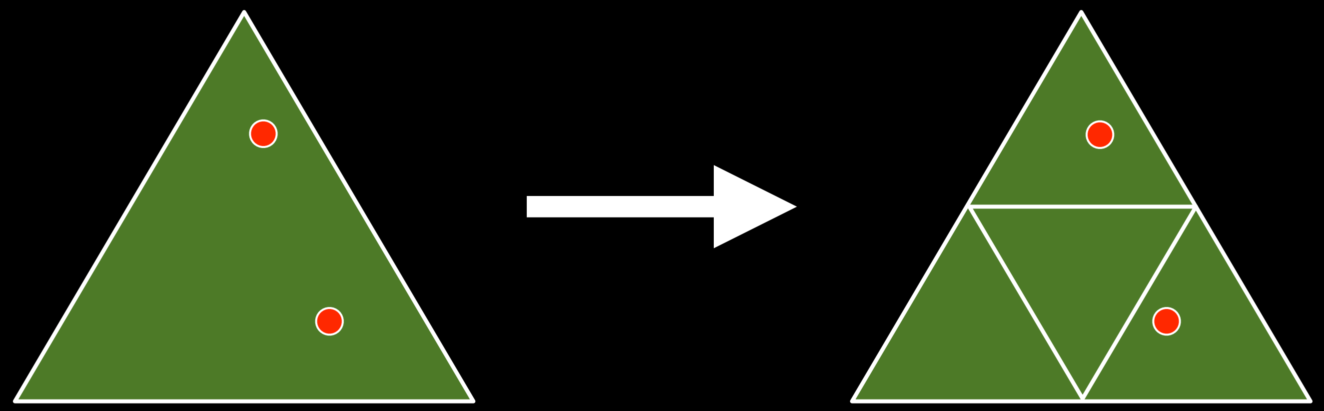
Triangle with Shading Normal

- Need good starting points
- May have zero, one, or multiple solutions
  - Subdivide triangle as needed to isolate solutions



Two Triangle Subdivision Oracles

- Test with strong guarantees
  - Based on [Krawczyk 69], [Mitchell&Hanrahan 92]
  - Conditions guarantee uniqueness and convergence
- Fast empirical heuristic
  - Based on solid angles of triangle and normals



Triangle summary

- For each triangle:
  - If no shading normals
    - Solve for P with 1D Newton
  - Else if passes the subdivision oracle
    - Solve for P with 2D Newton
  - Else
    - Subdivide into 4 triangles and try again
  - Test if P lies within the triangle

Outline

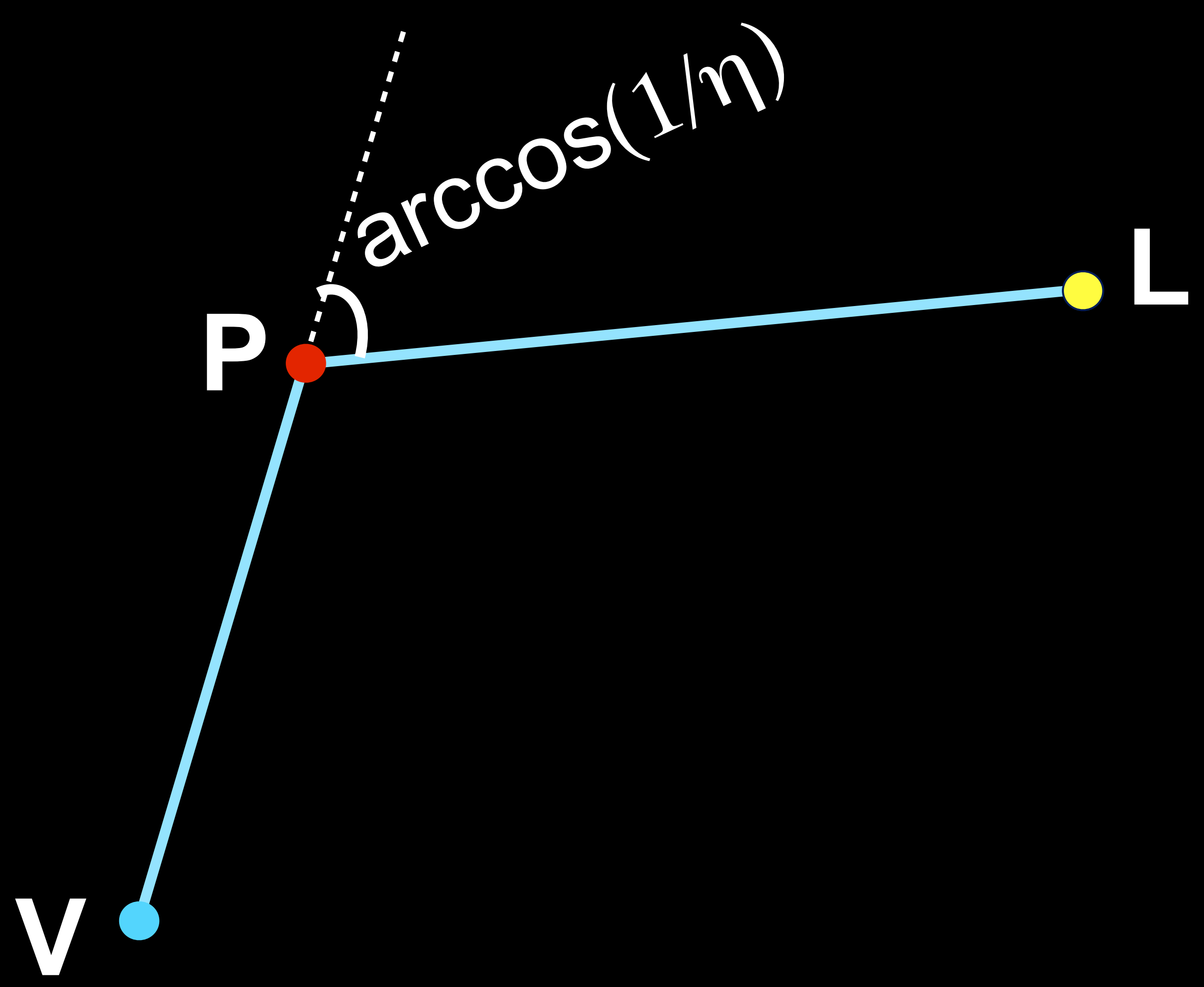
- Half-vector formulation
- Solving for a single triangle
- Hierarchical culling for meshes
- Results

Culling Tests

- Most triangles contain no solutions for P
- Three quick culling tests
  - Spindle
  - Sidedness
  - Interval

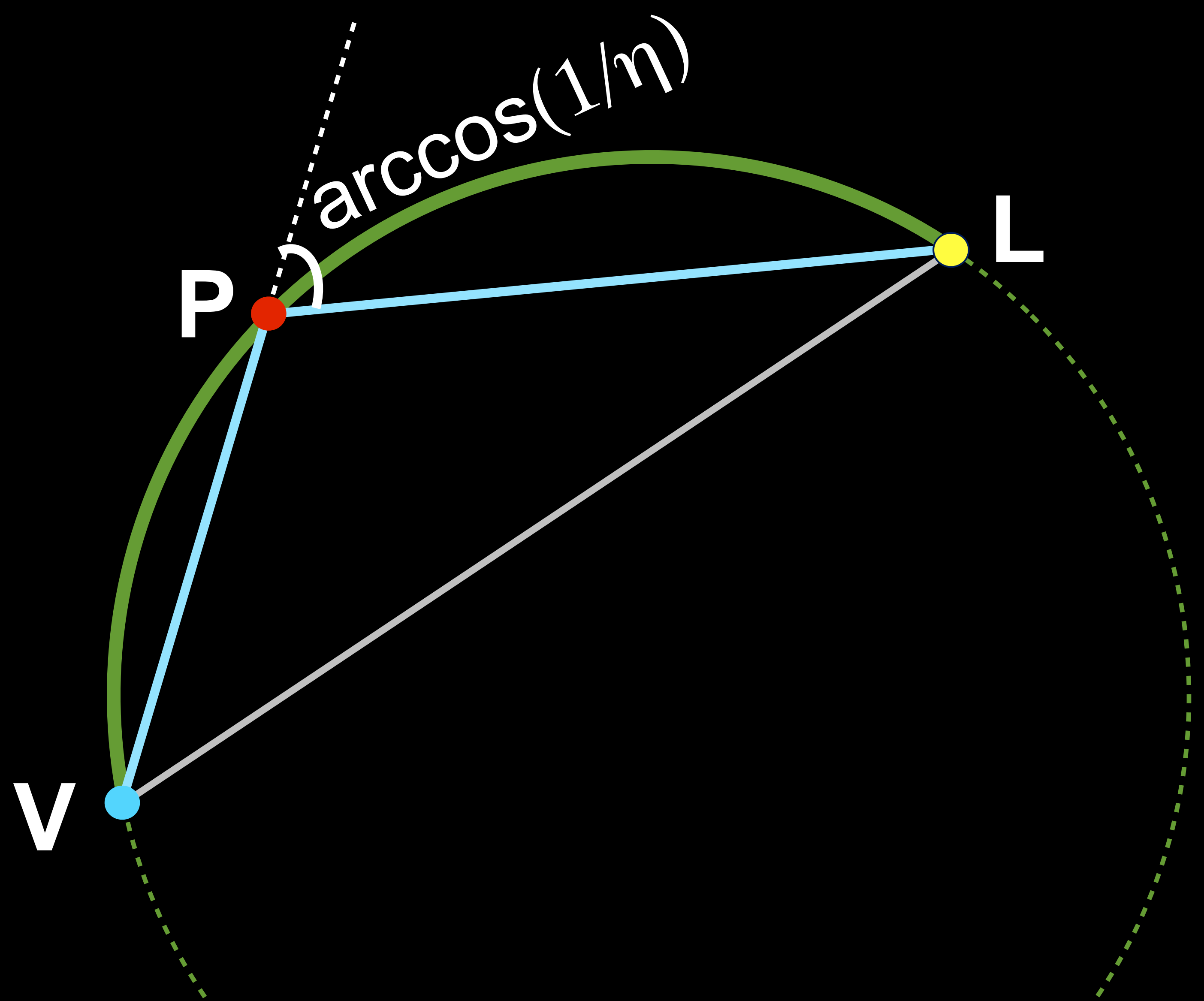
Spindle Culling Test

- Refraction bends path by angle ≤ arccos($1/\eta$)
  - Solutions must lie within circular arc (2d) or spindle (3d)



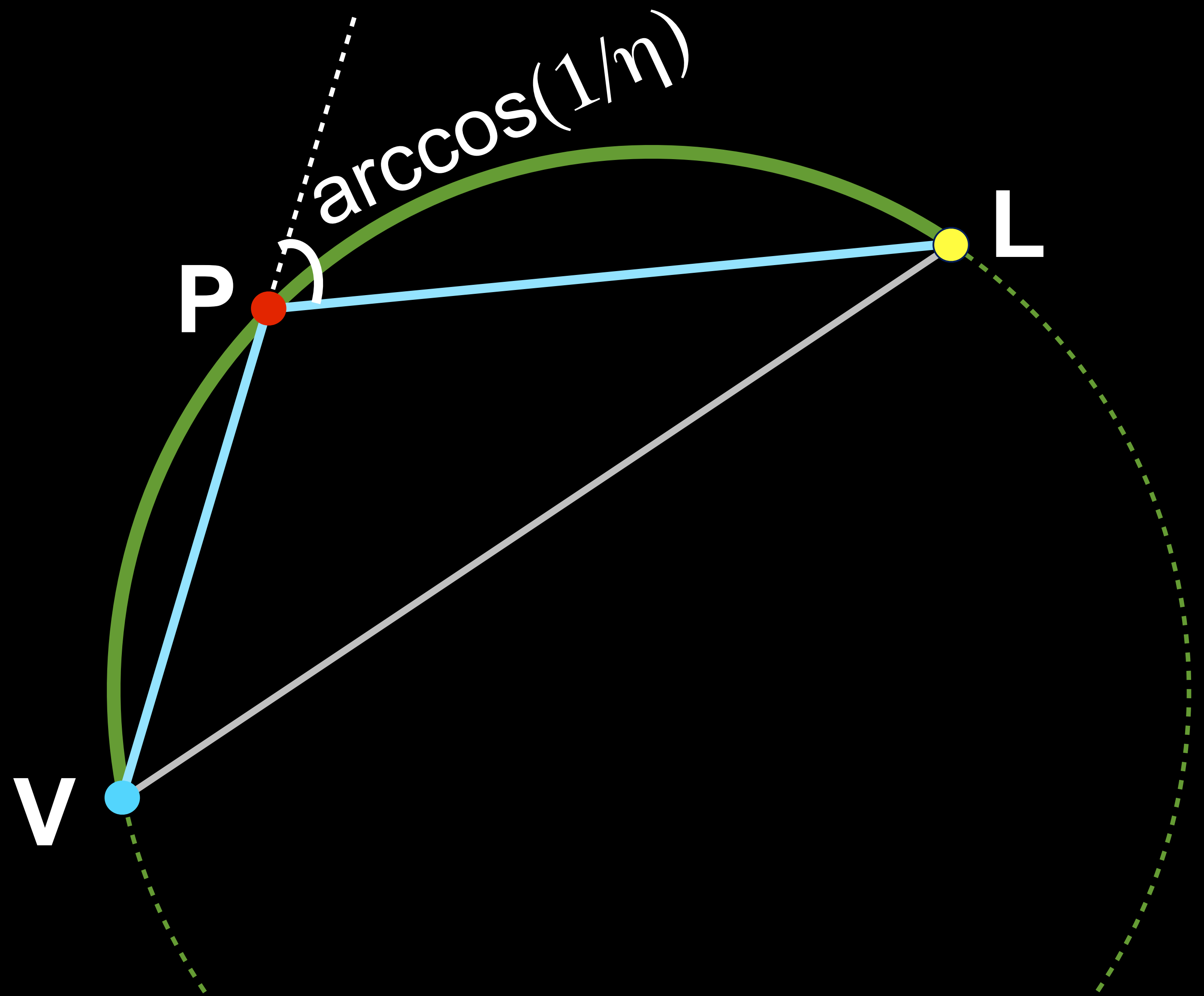
Spindle Culling Test

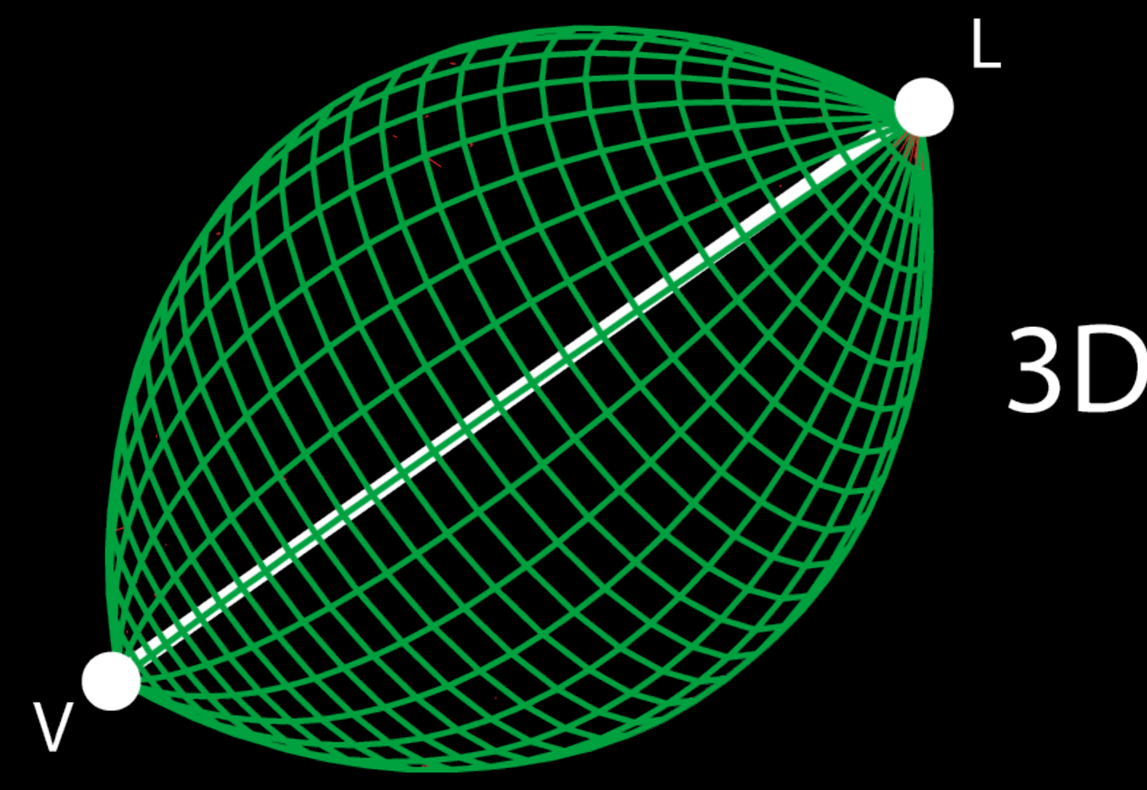
- Refraction bends path by angle ≤ arccos($1/\eta$)
  - Solutions must lie within circular arc (2d) or spindle (3d)



Spindle Culling Test

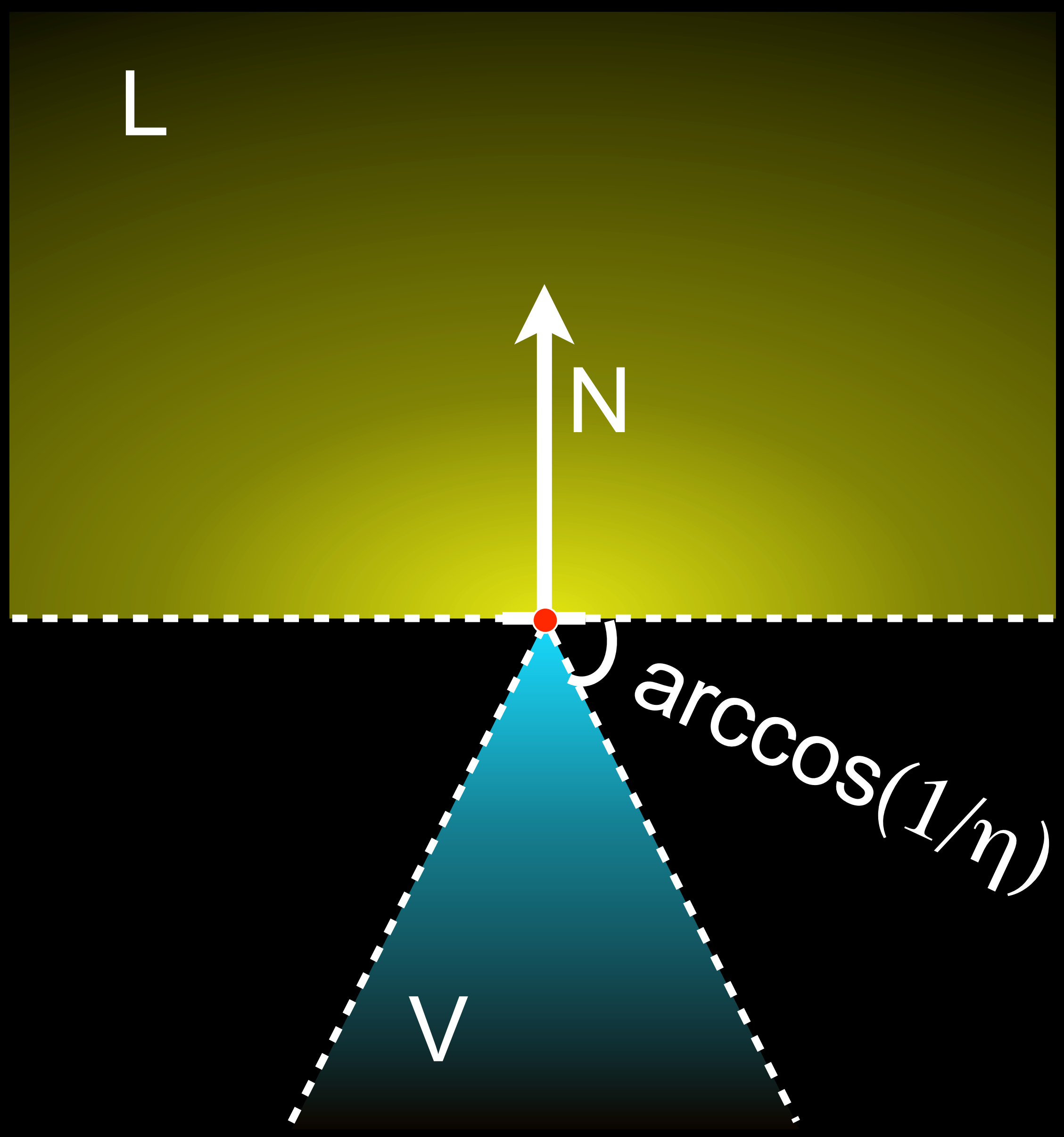
- Refraction bends path by angle ≤ arccos($1/\eta$)
  - Solutions must lie within circular arc (2d) or spindle (3d)





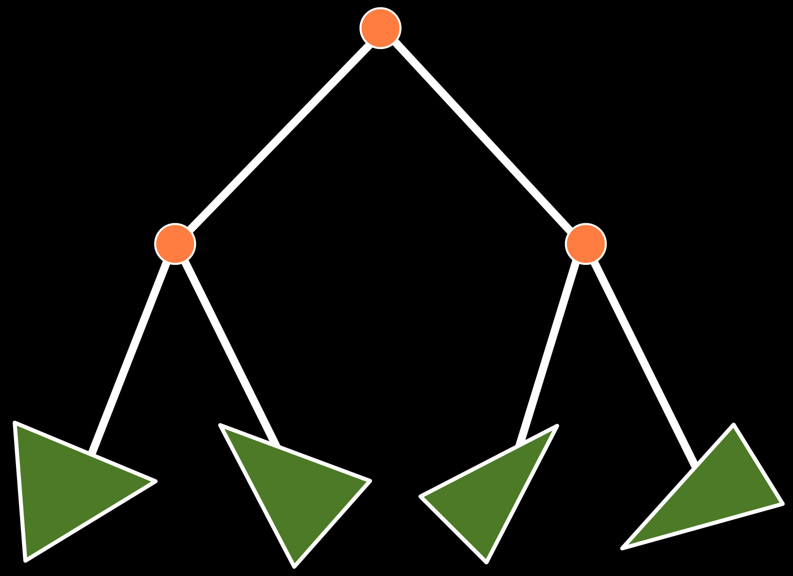
Sidedness Culling Test

- Light L must be on the outside of surface at P
- Receiver V must be inside within the critical angle



Hierarchical Culling for Meshes

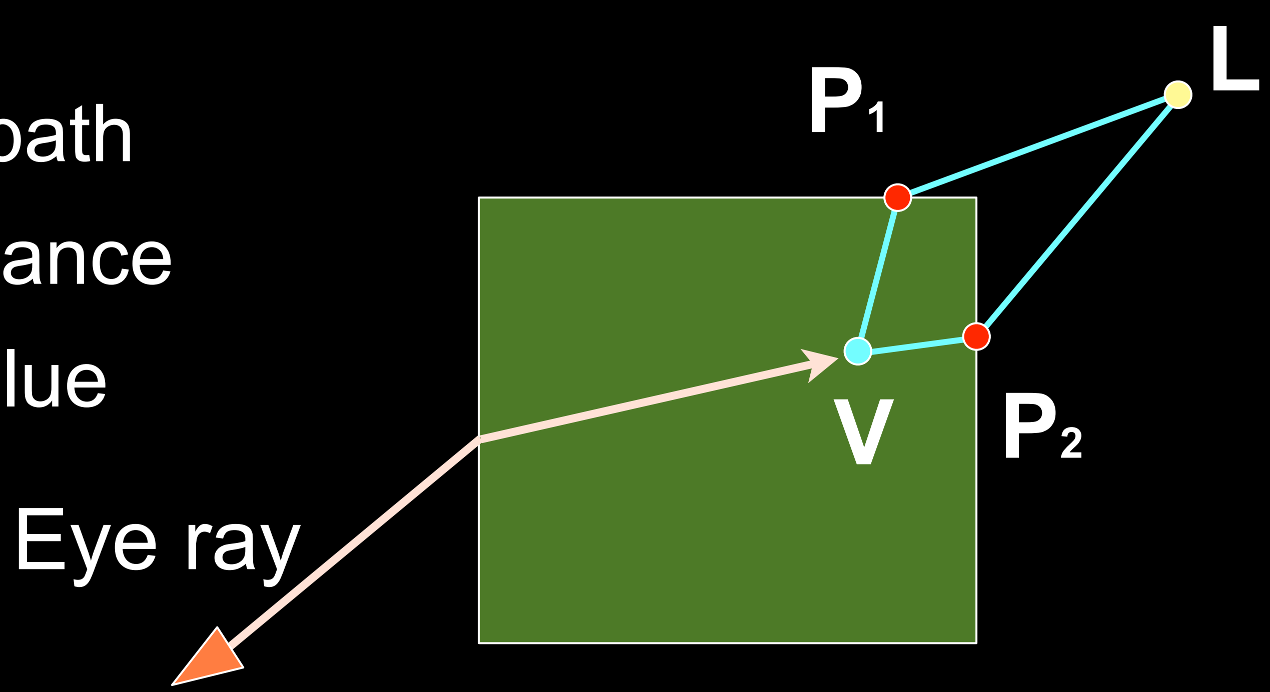
- Apply culling test to groups of triangles
- Use 6D position-normal tree [Bala et al. 03]
  - Leaves are boundary triangles
  - Boxes and cones bound positions and normals
  - Traverse top-down



Position-normal tree

Algorithm Summary

- Build position-normal tree for each boundary mesh
- For each eye ray
  - Trace until hits surface or volume-scatters at V
  - Select a light source point, L
  - Traverse tree to solve for all P on boundary
  - For each solution point P
    - Check for occlusion along path
    - Compute effective light distance
    - Add contribution to pixel value



Effective Distance to Source

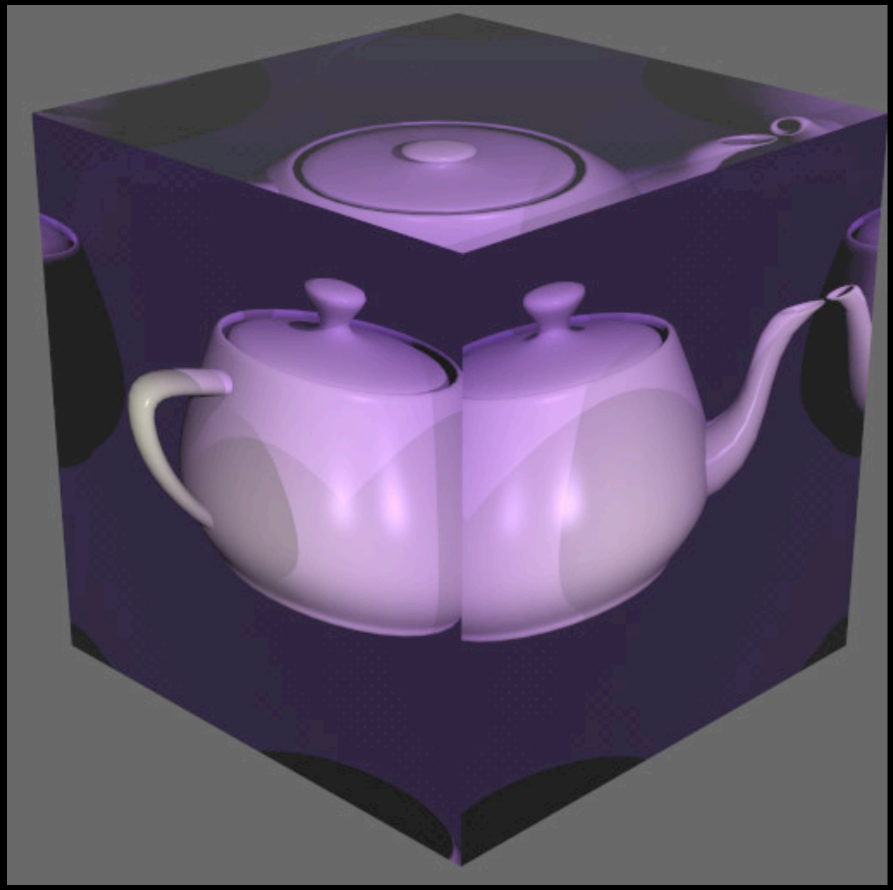
- Refraction alters usual $1/r^2$ intensity falloff
  - –Can focus or defocus the light
- Compute effective light distance for each path
  - –Simple formula for triangles without shading normal
  - –Use ray differentials [Igehy 99] for shading normal case
  - –See paper for details

Outline

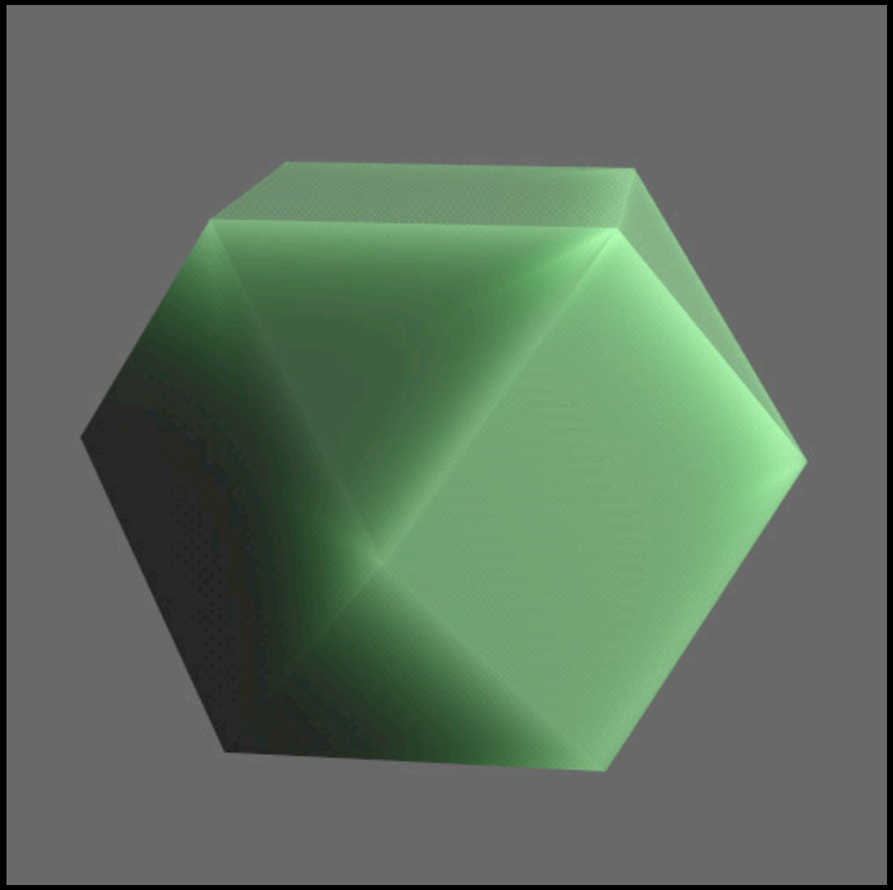
- Half-vector formulation
- Solving for a single triangle
- Hierarchical culling for meshes
- Results

Results - CPU

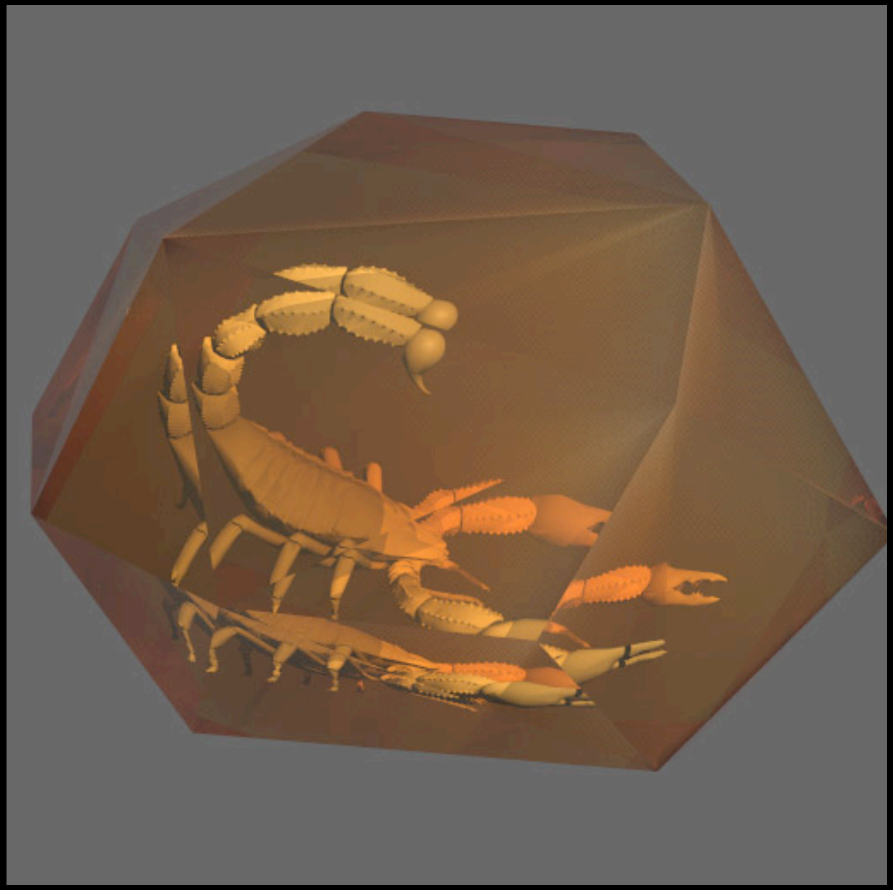
- Three scenes without shading normals



Teapot



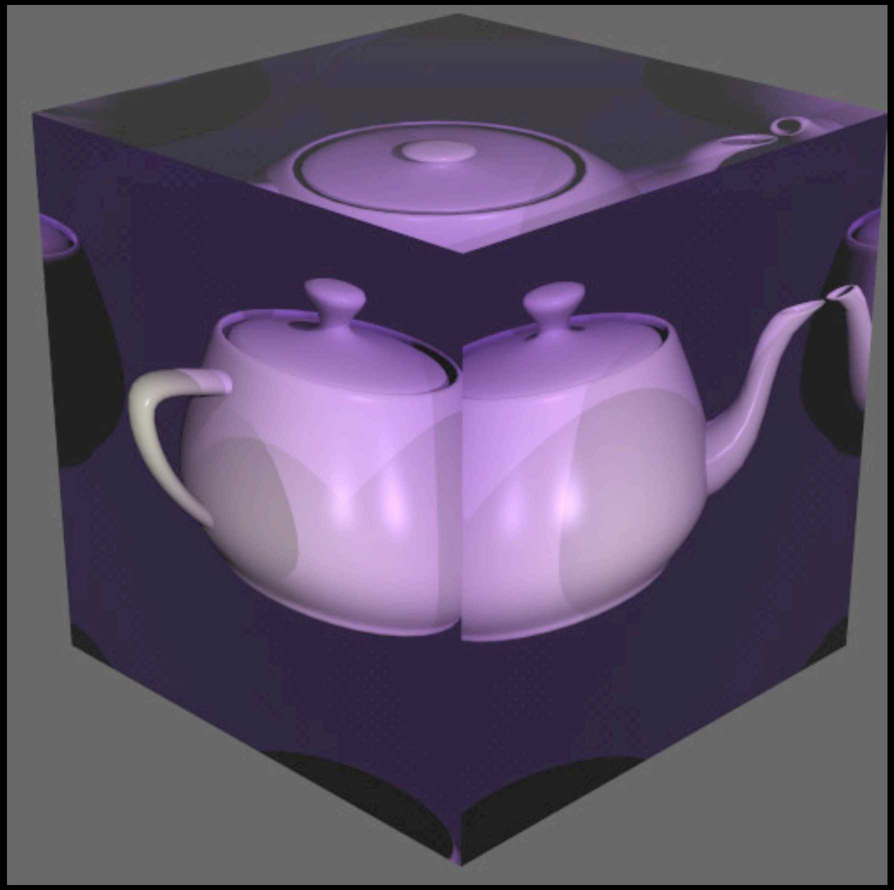
Cuboctahedron



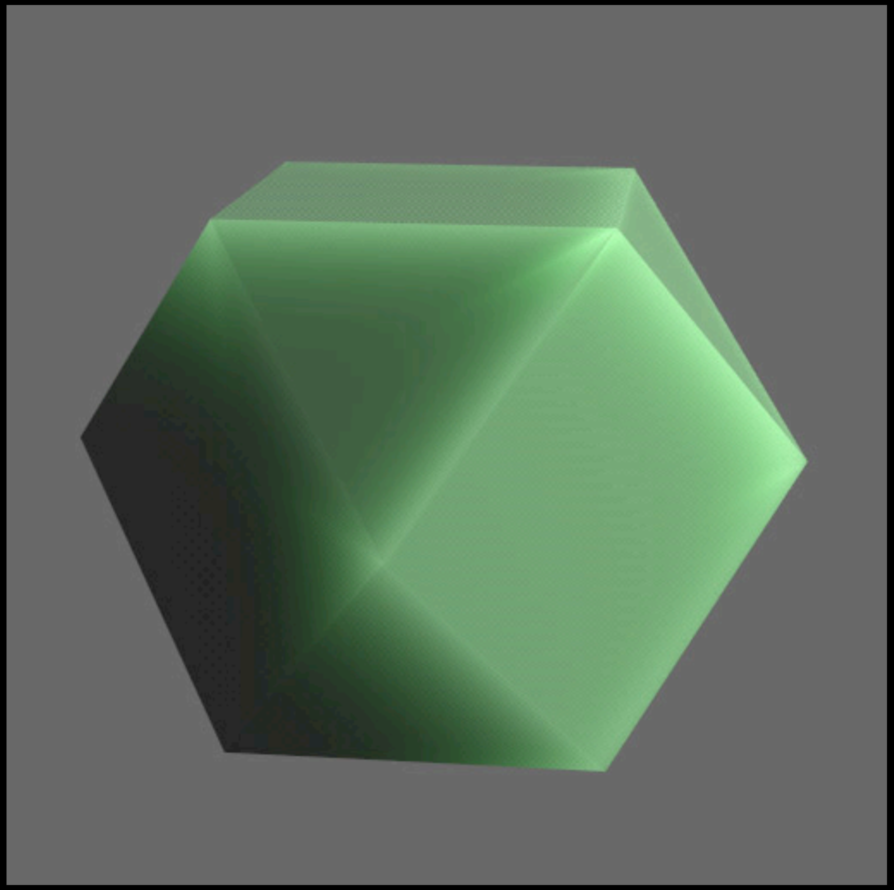
Amber

Results - CPU

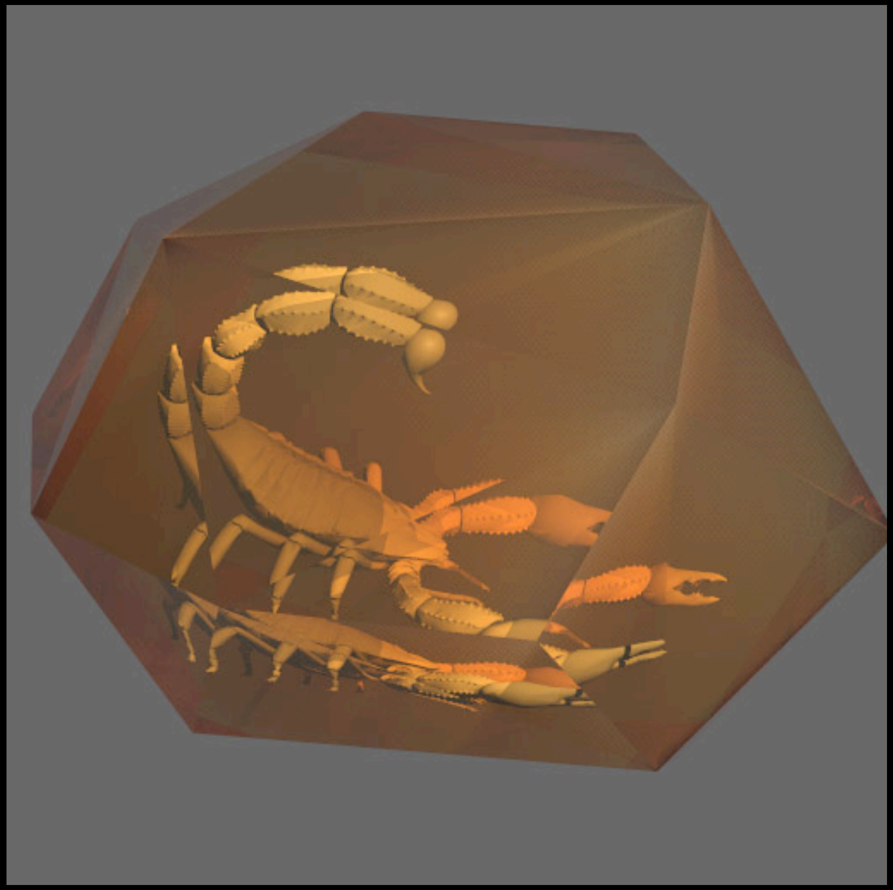
- Three scenes without shading normals



Teapot
15.3s



Cuboctahedron
13.9s

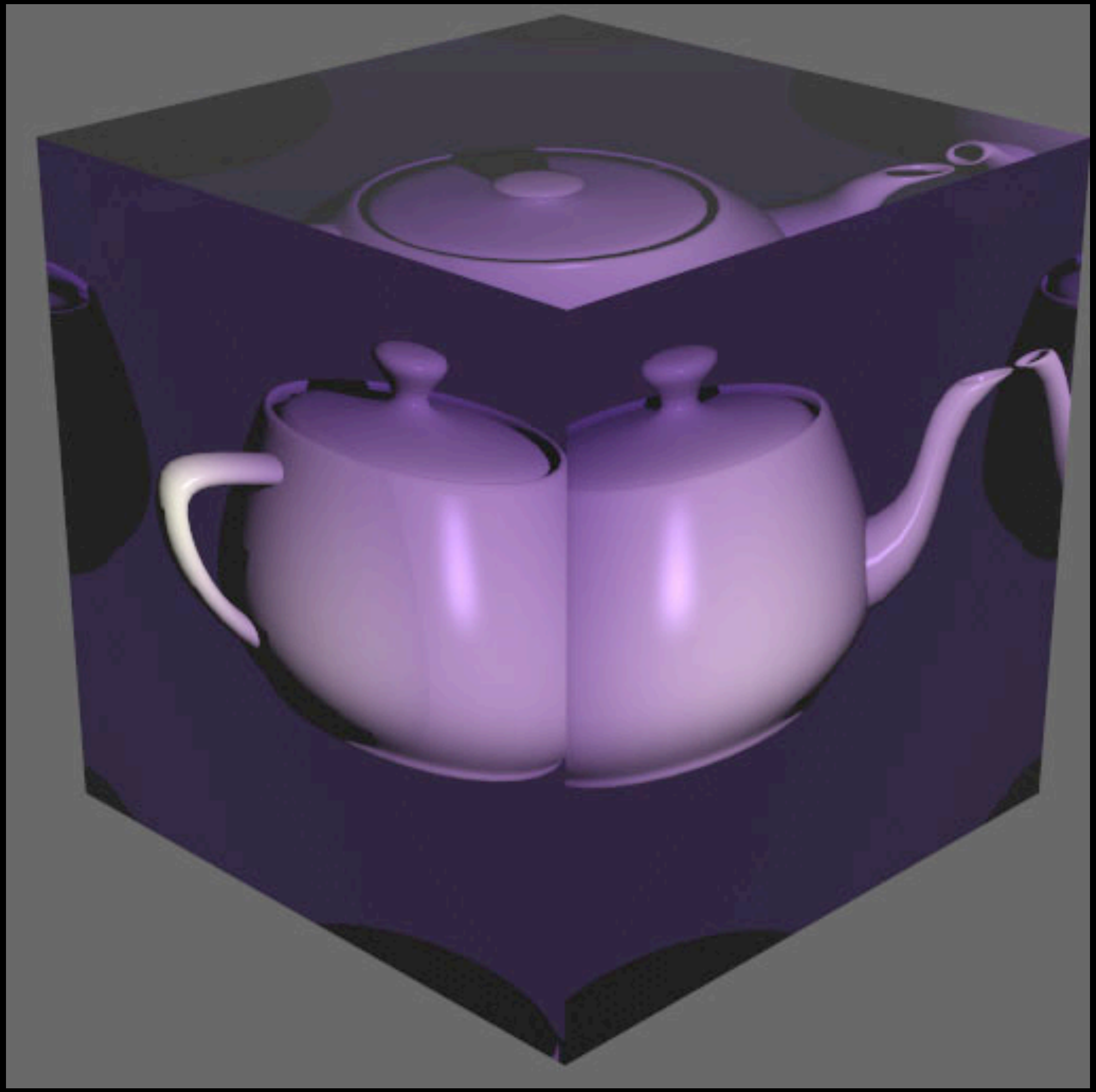


Amber
19.2s

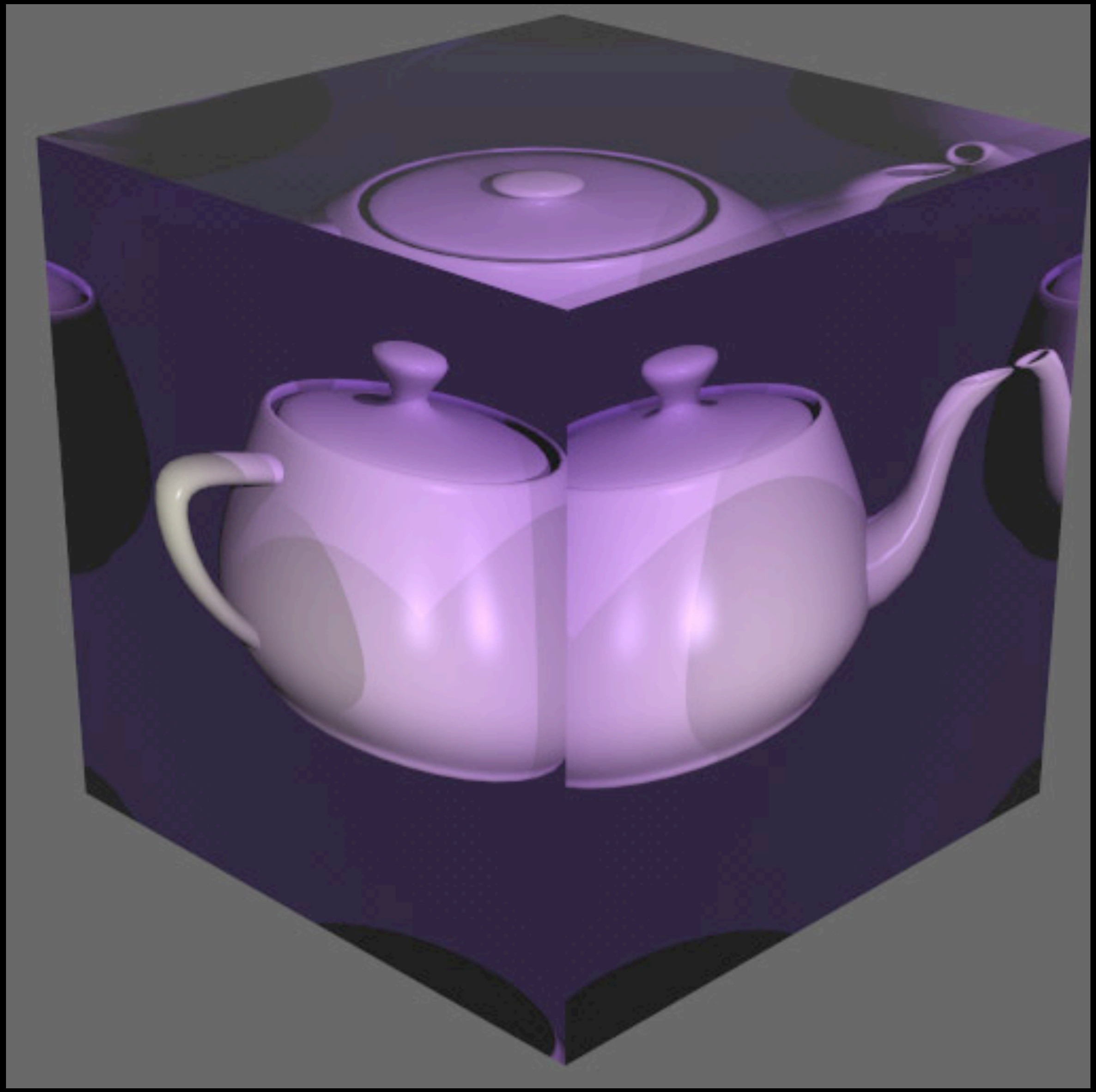
512x512 images, 64 samples per pixel, 8-core 2.83GHz Intel Core2

Results - Teapot

- Teapot quality comparison



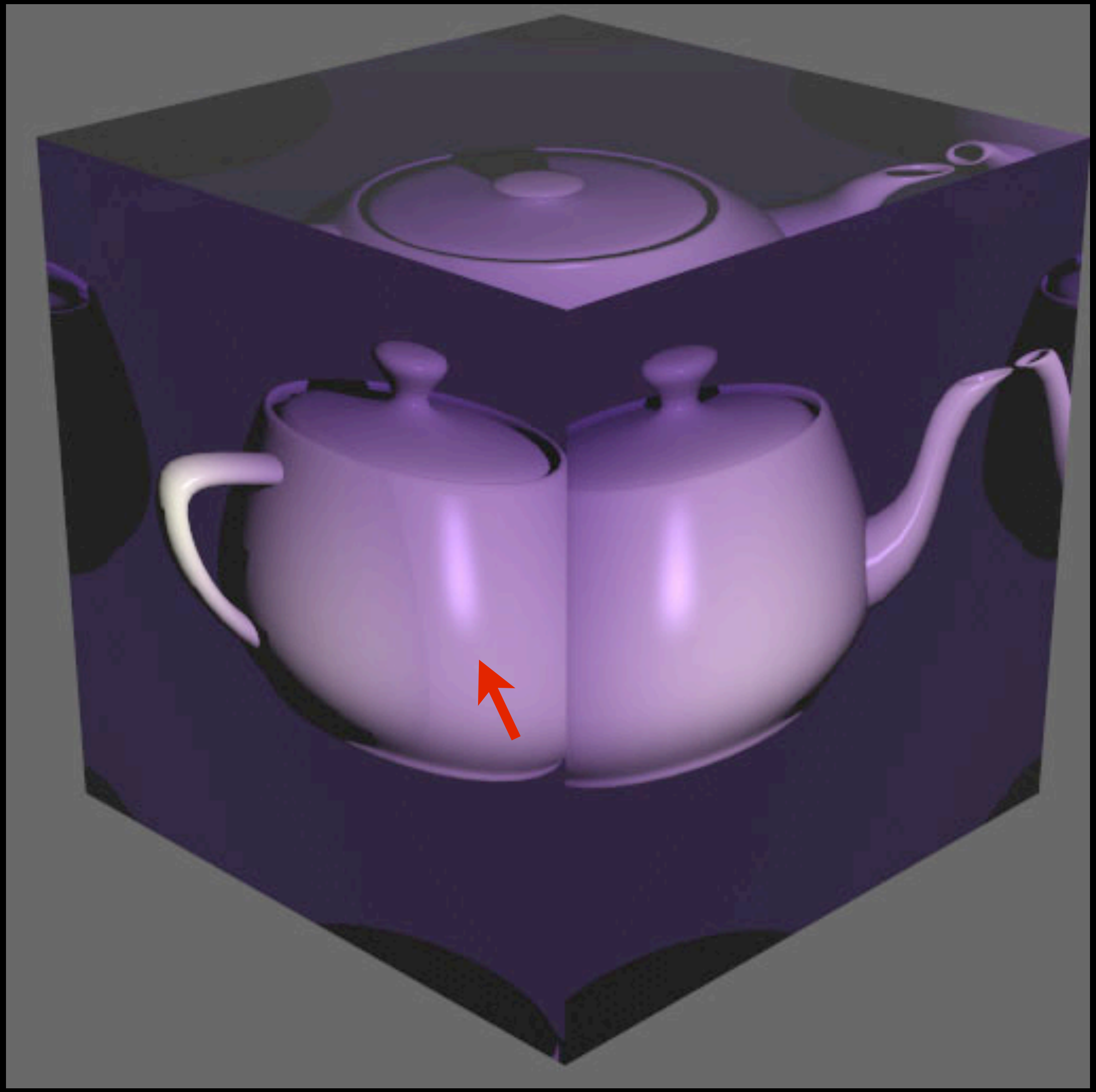
Shadow rays ignore refraction



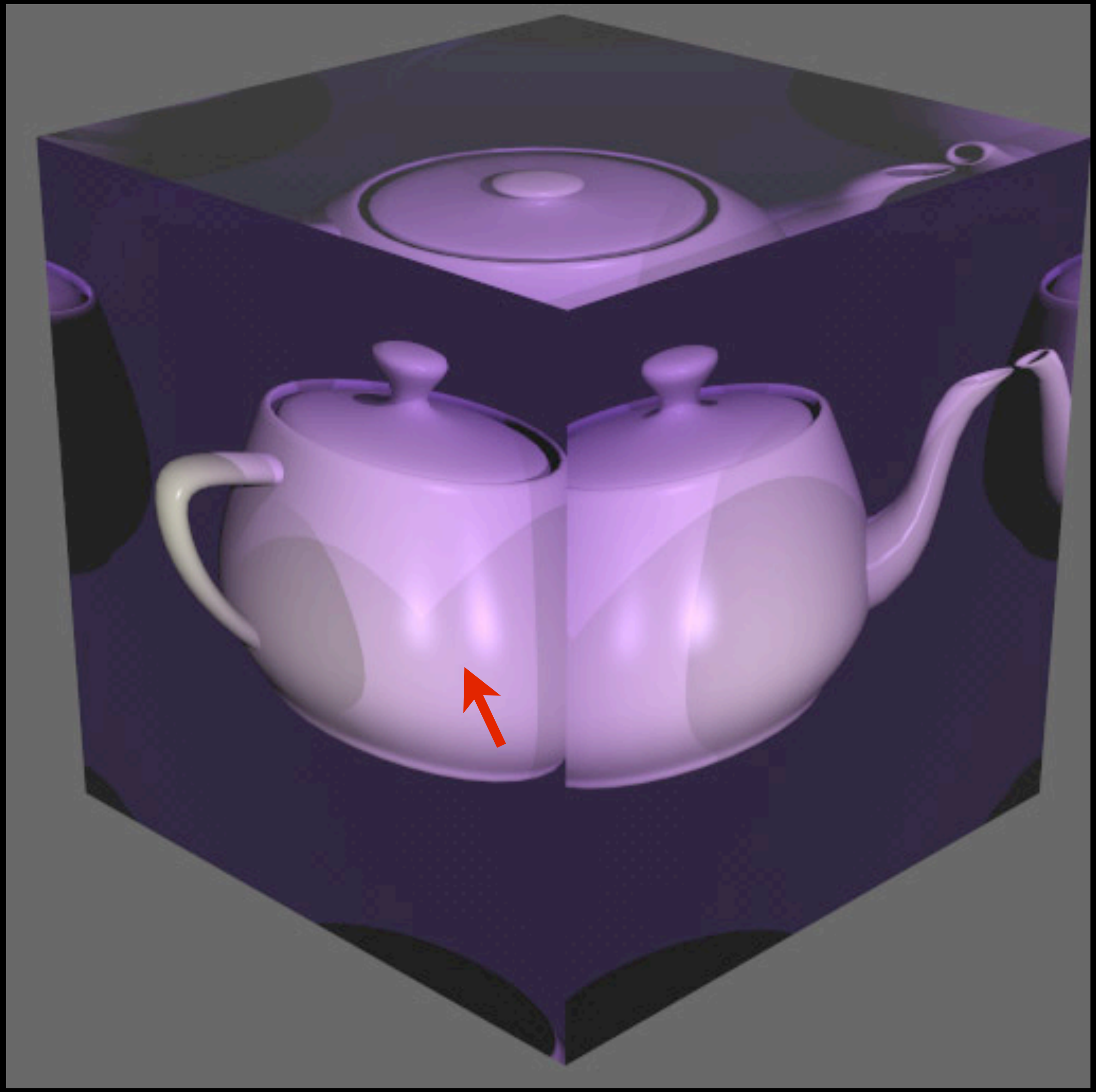
Our method (15.4s)

Results - Teapot

- Teapot quality comparison



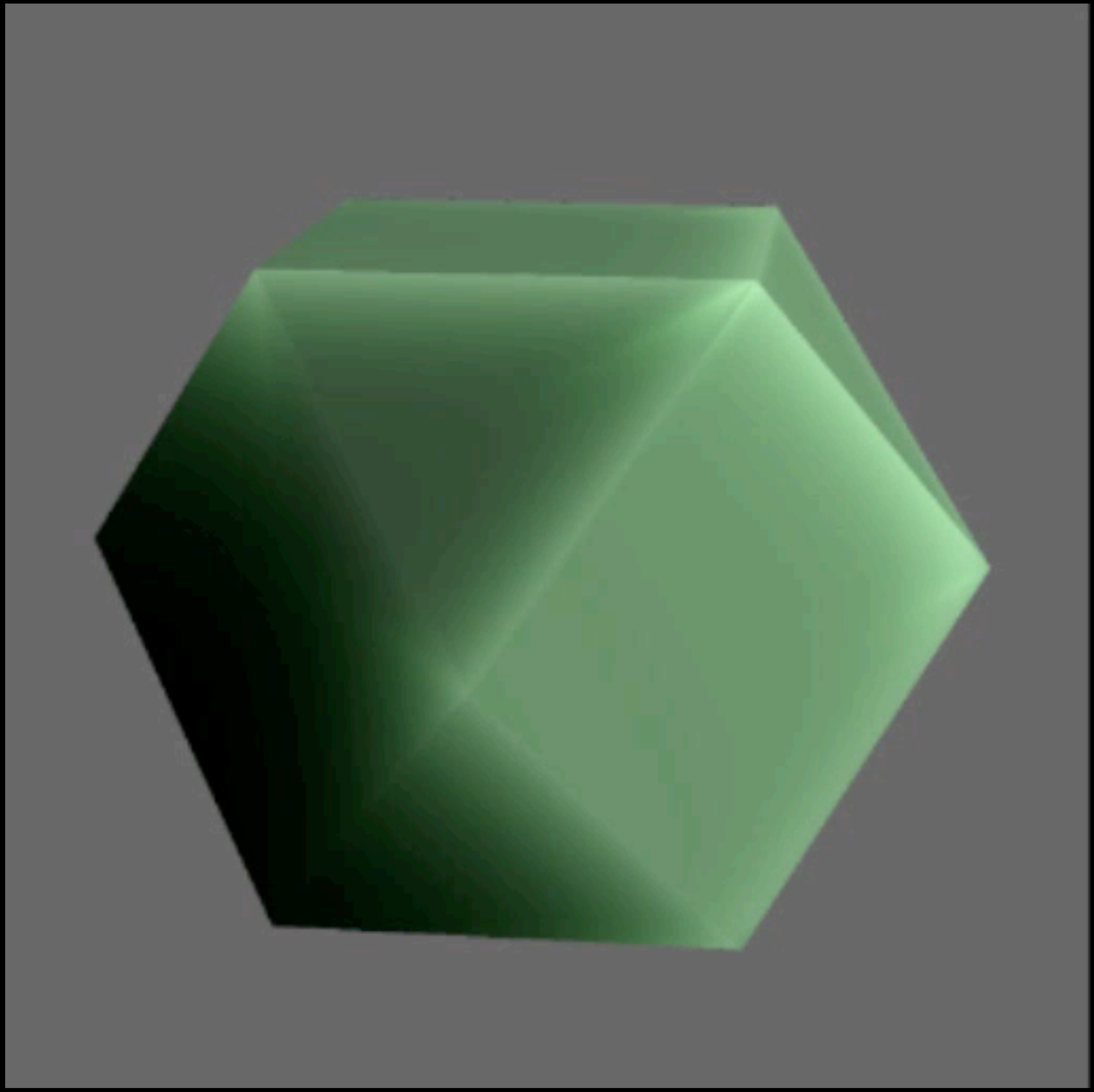
Shadow rays ignore refraction



Our method (15.4s)

Results - Cuboctahedron

- Cuboctahedron movie (13.9s)



Results - GPU

- Implemented on GPU using CUDA 2.0
  - 1D, 2D Newton iteration
  - Hierarchical pruning
  - Ray tracing based on [Popov et al. 2007]
  - One kernel thread per eye ray
  - Does not yet support all scenes

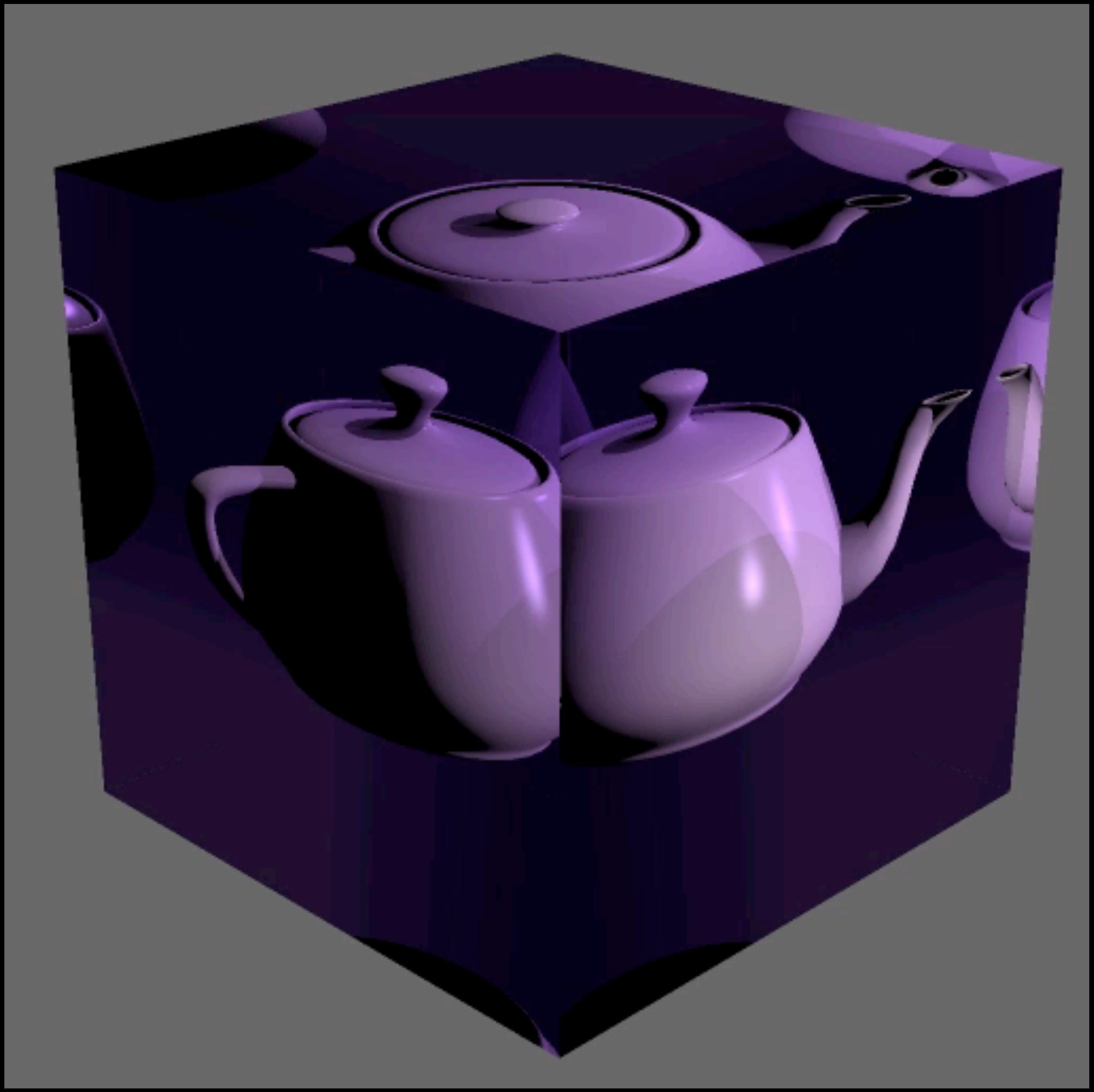
Results - GPU

Name	Render Time	FPS
Teapot	0.1 s	10 fps
Cuboctahedron	0.14 s	7 fps
Amber	0.3 s	3 fps

512x512 images, 2 eye rays per pixel + 40-60 volume samples, nVIDIA GTX 280, CUDA 2.0

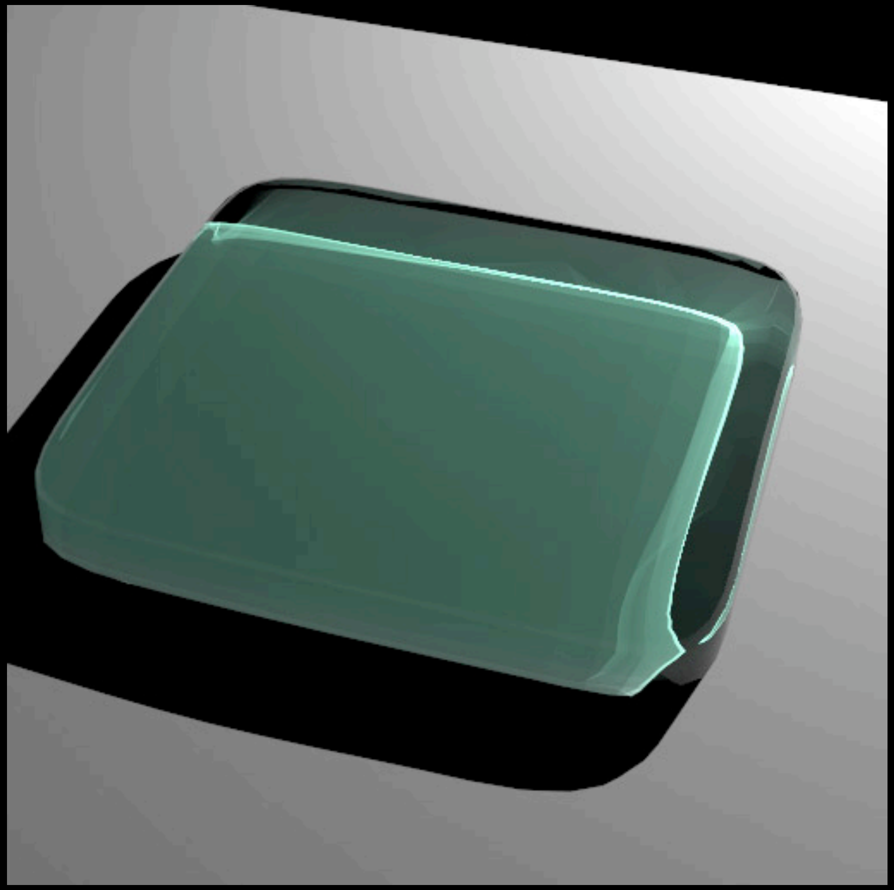
Results - GPU

- Teapot example
  - 10 fps on GPU

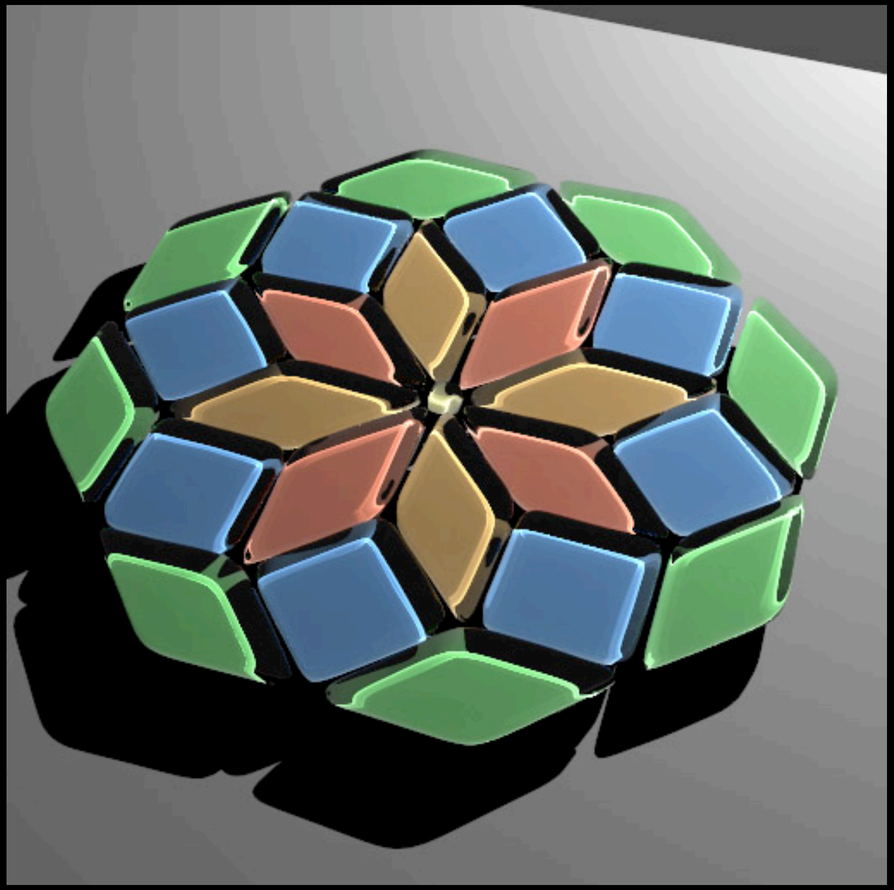


Results - CPU

- Three scenes with shading normals



Glass tile

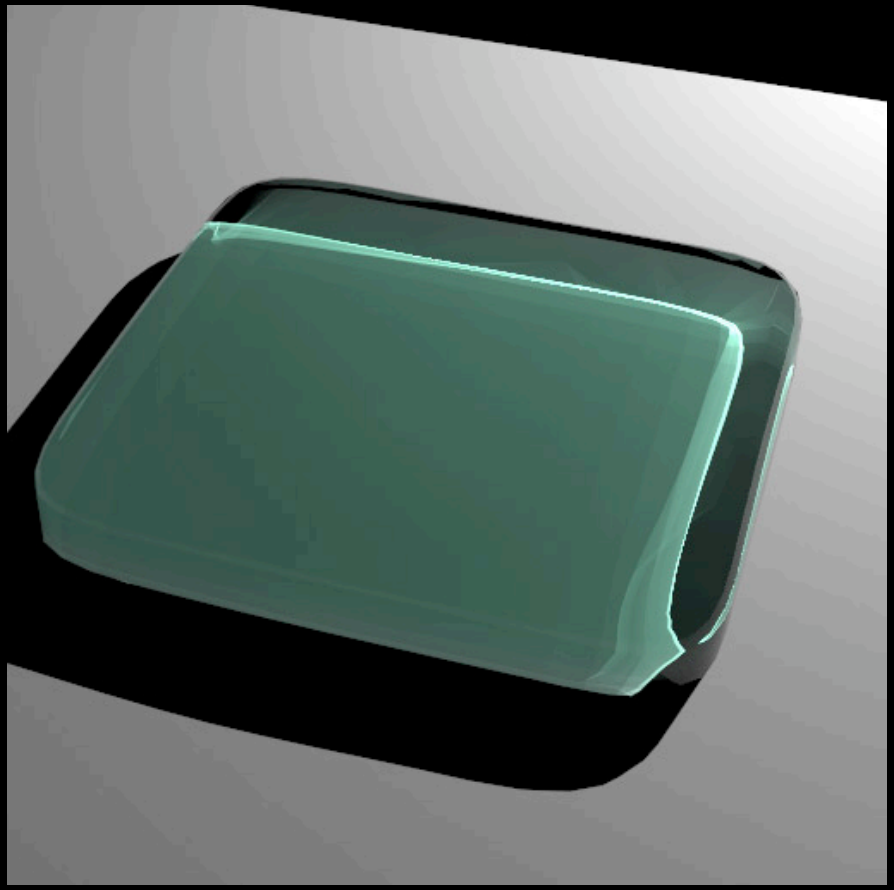


Glass mosaic

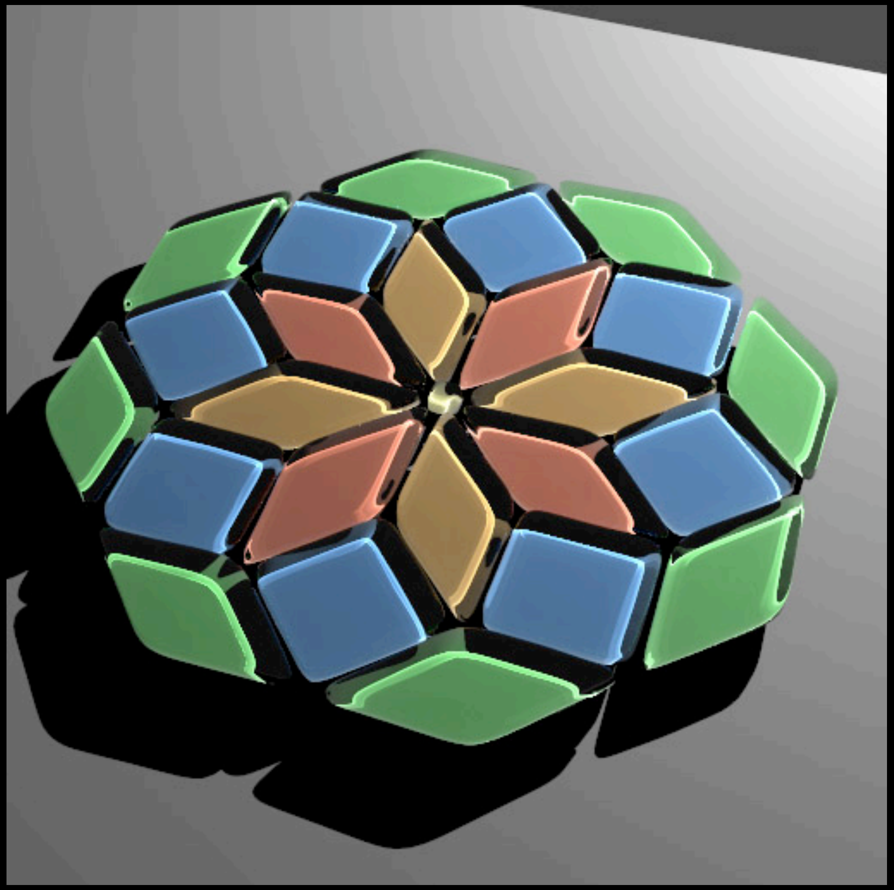
Pool

Results - CPU

- Three scenes with shading normals



Glass tile
66.9s

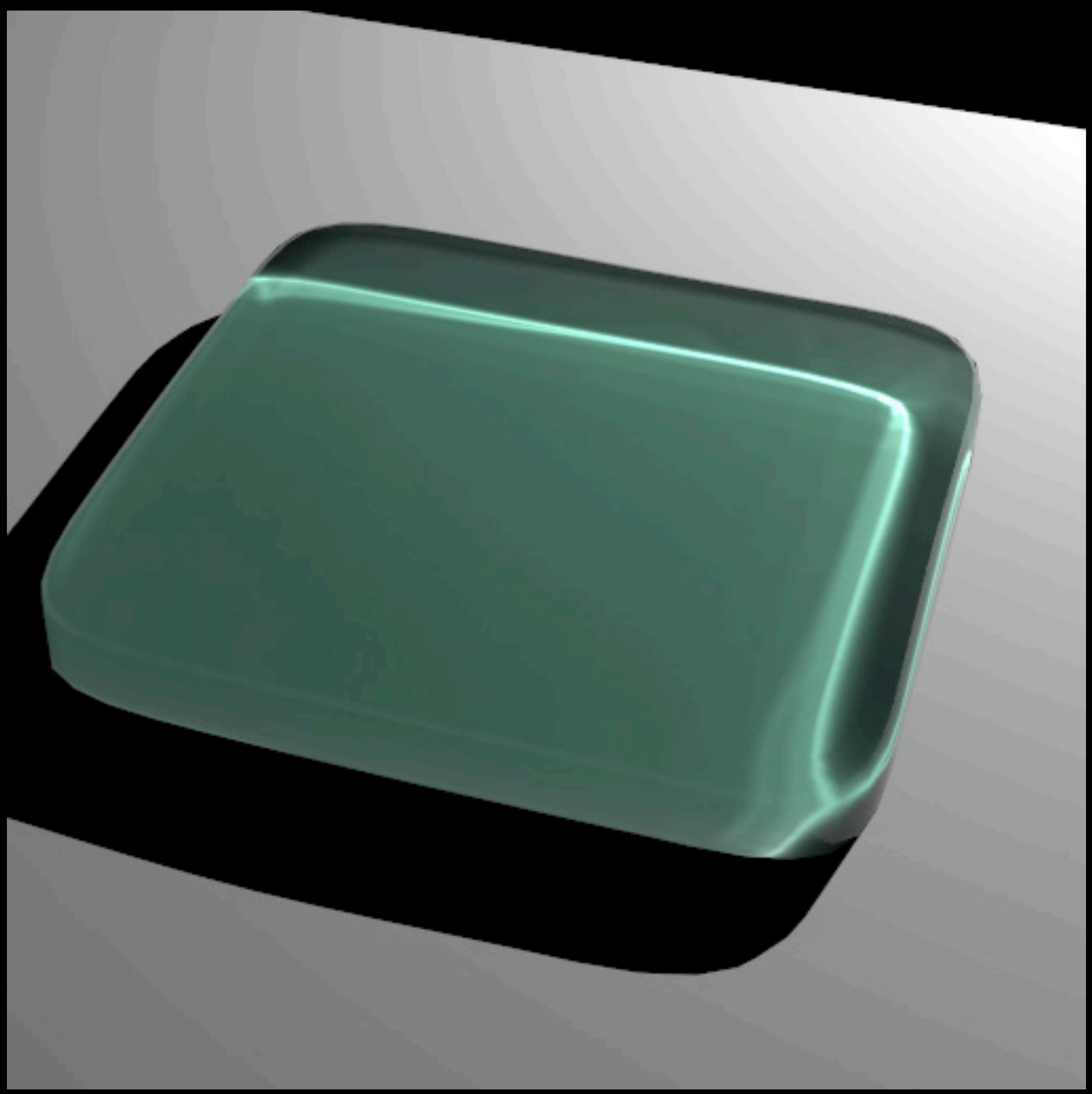


Glass mosaic
87.8s

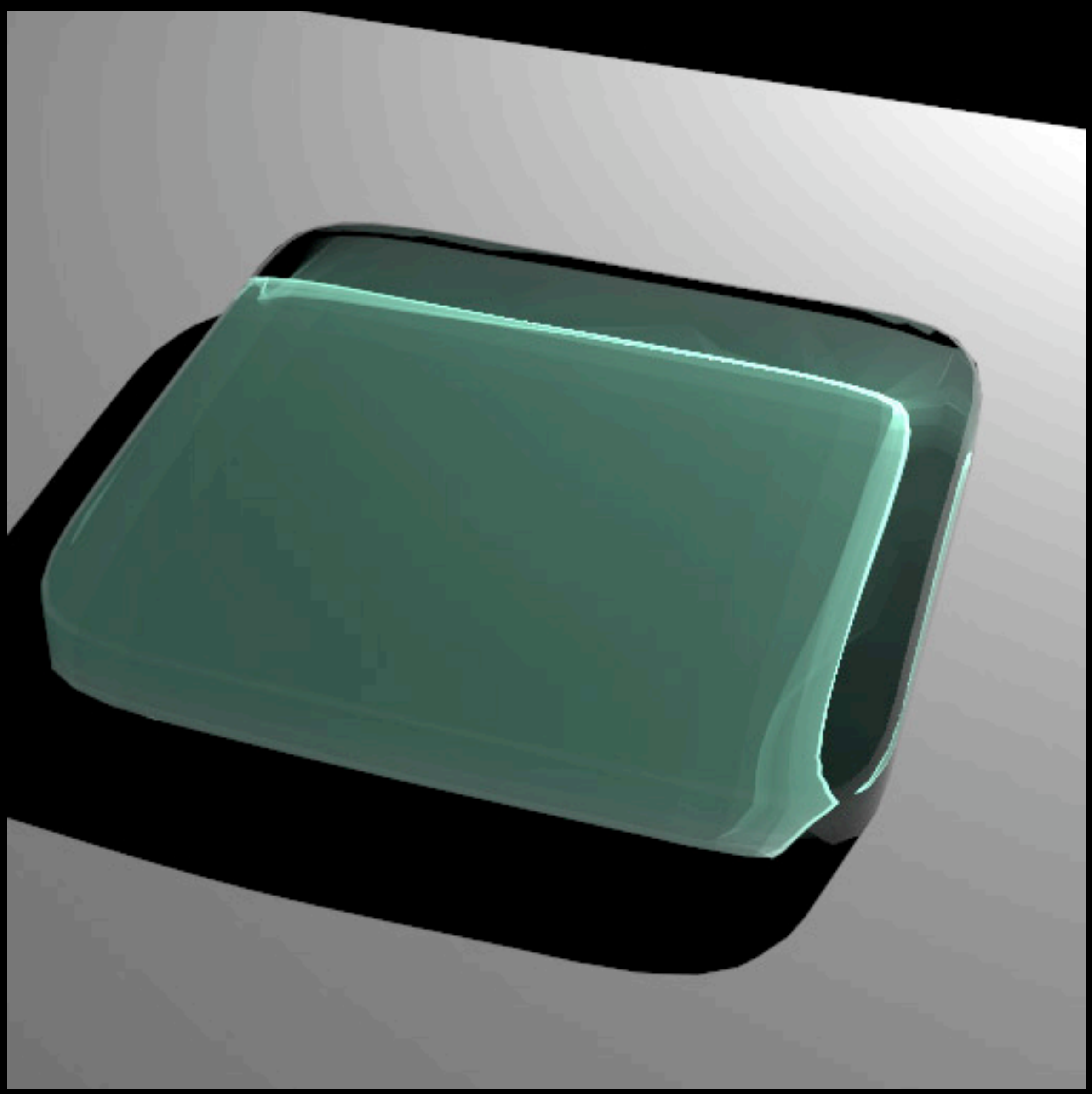
Pool
59.4s

512x512 images, 64 samples per pixel, 8-core 2.83GHz Intel Core2

Results - Glass Tile



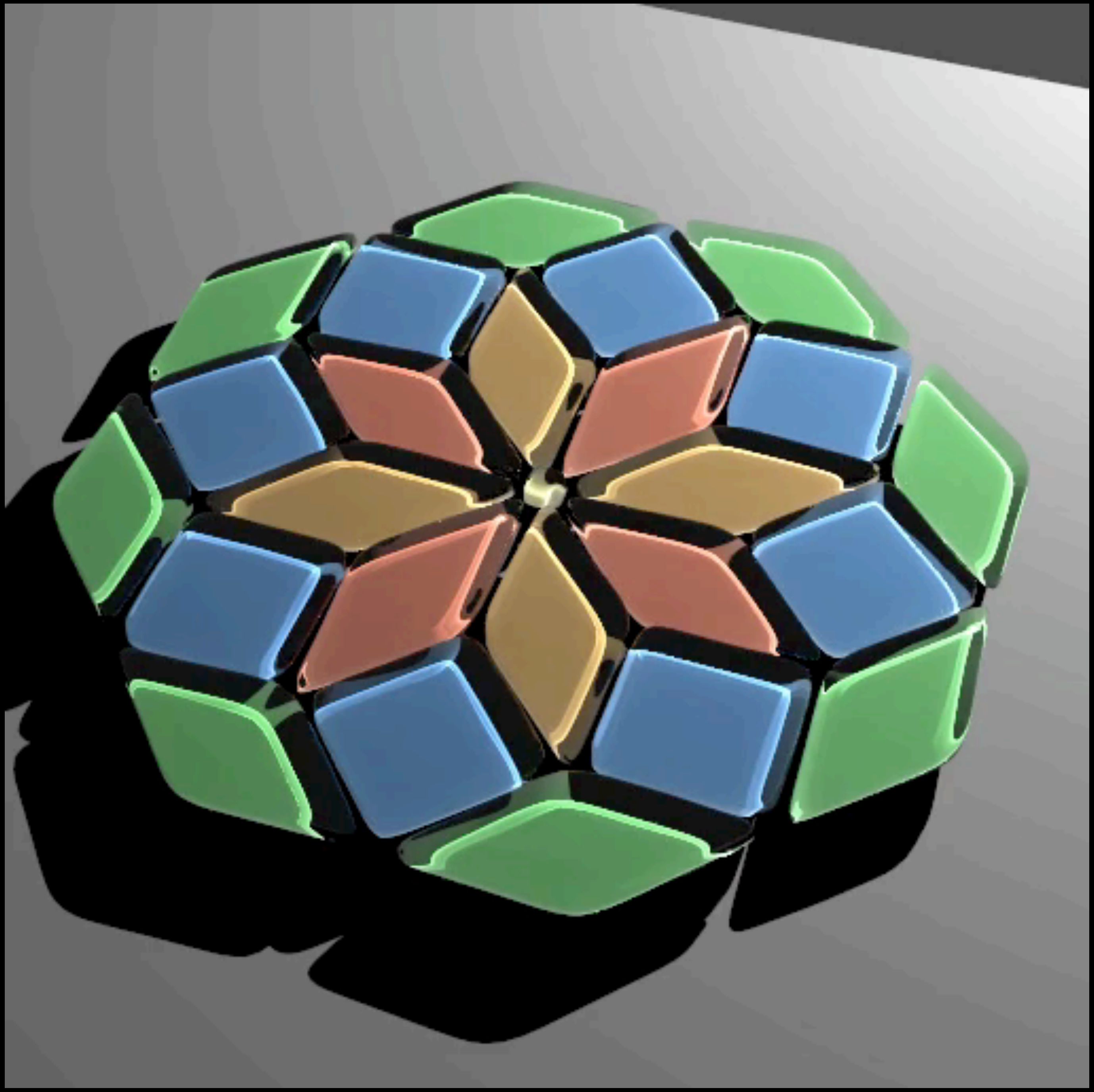
Photon map (equal time)



Our method (66.9s)

Results - Glass Mosaic

- Glass mosaic movie (87.8s)



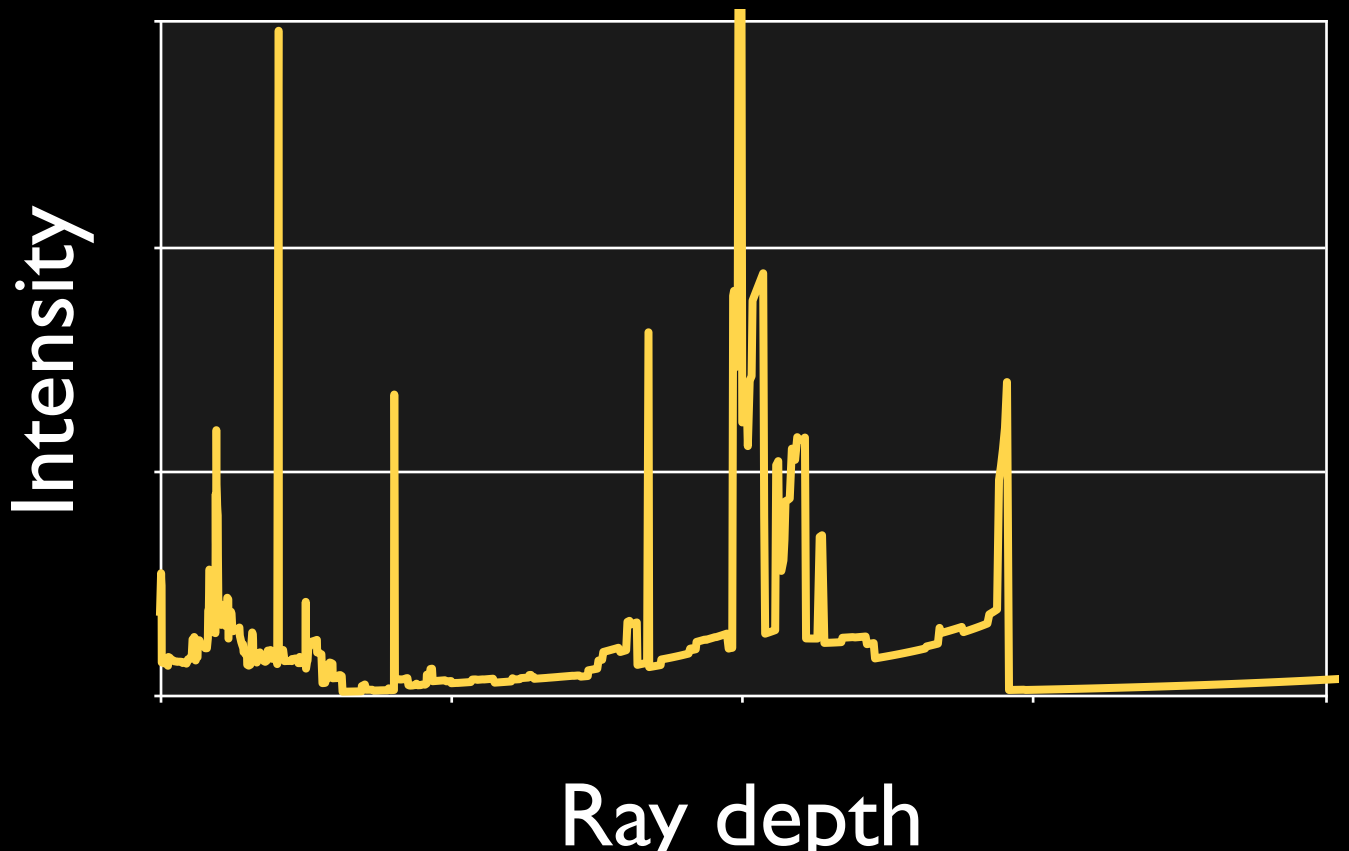
Results - Component Evaluation

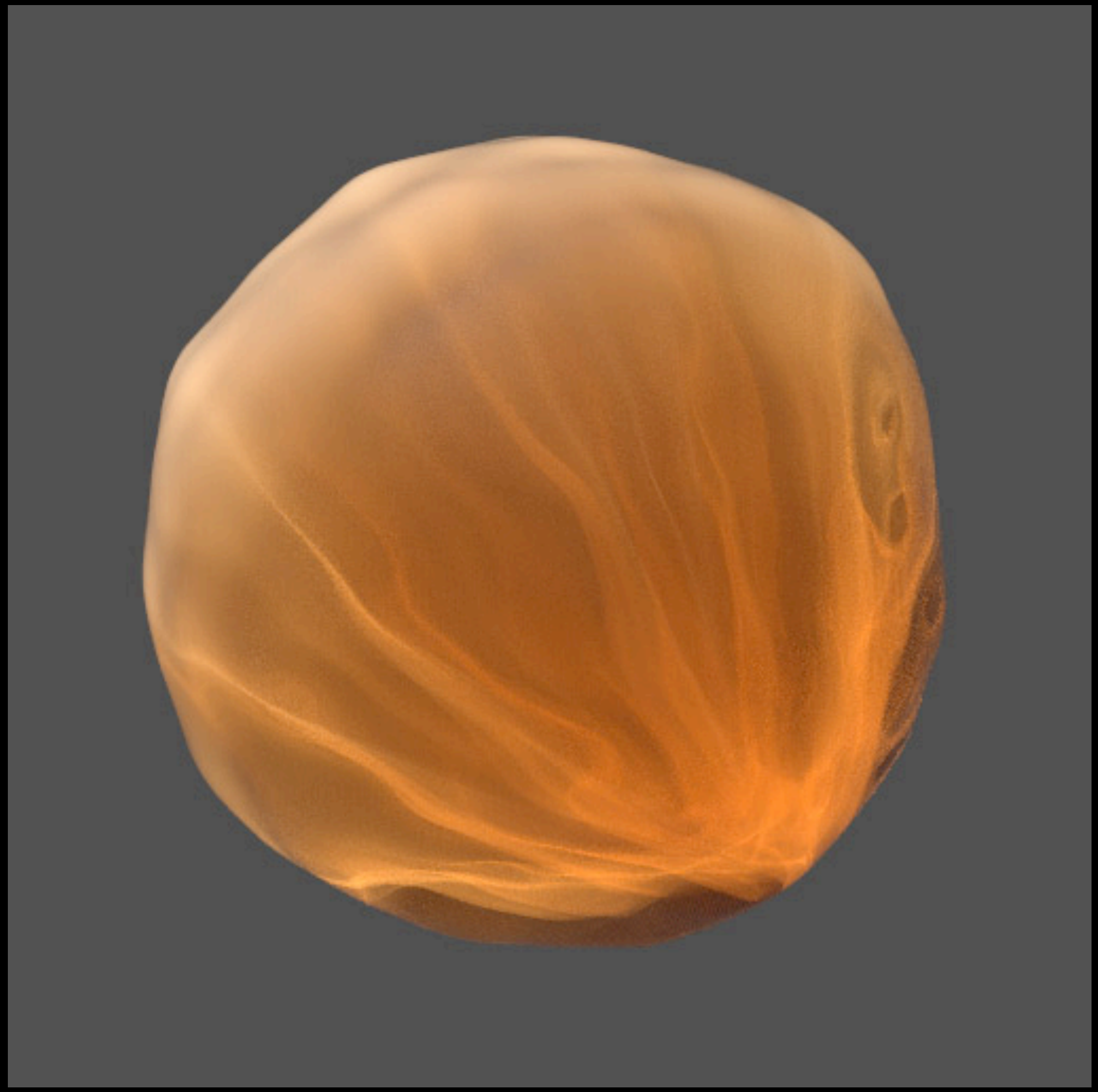
- Evaluation of algorithm components
  - Pool (2632 triangles in boundary)

	Time	Ratio
Without Hierarchy	1934.6s	32x
Guaranteed Convergence	141.1s	2.4x
Subdivision Heuristic	59.4s	1x

Results - Bumpy sphere

- Bumpy sphere (9680 triangles)
  - Volume sampling noise
    - Used 128 samples per pixel
    - Effective distance clamping



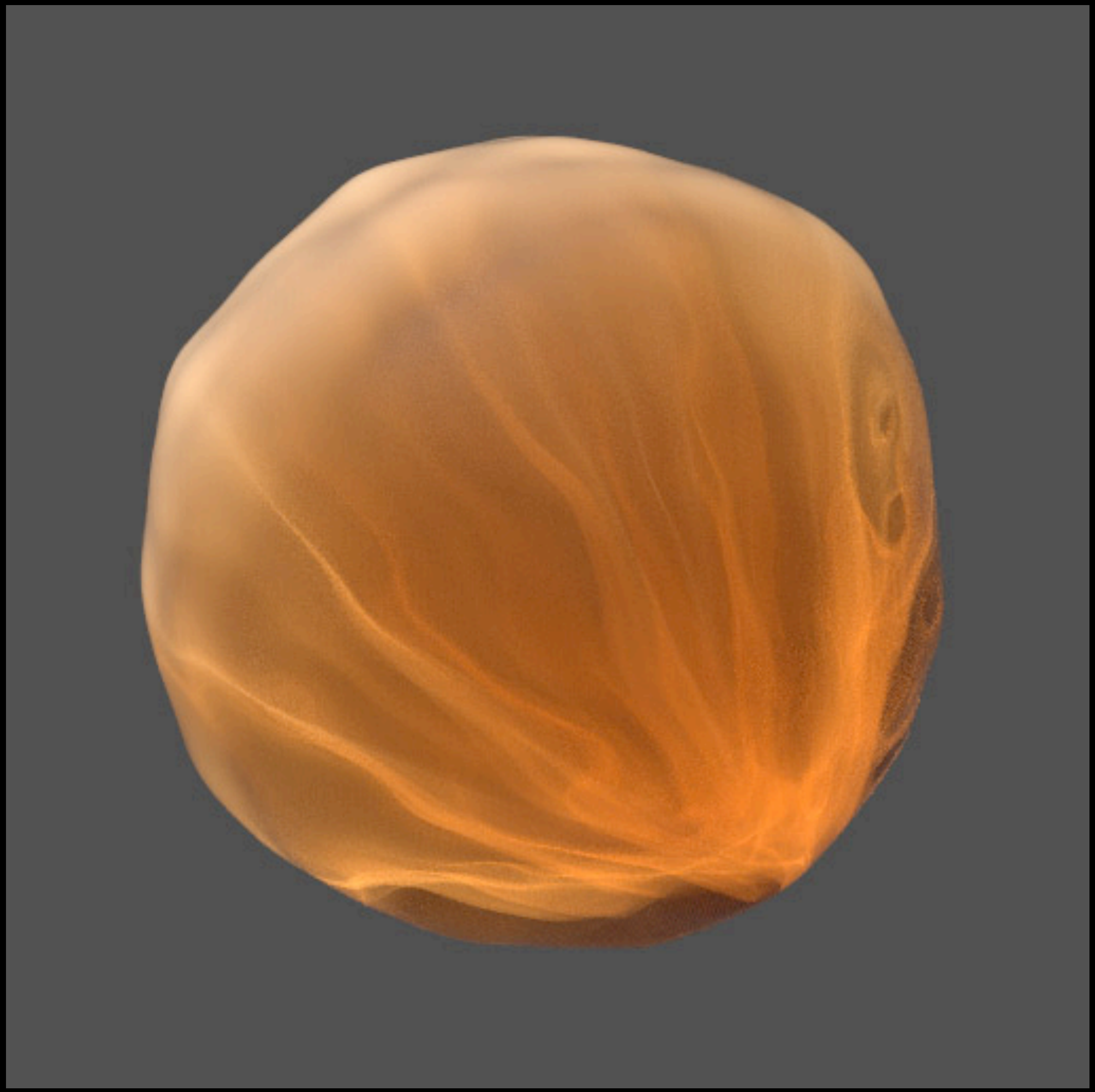


Our method 304.3 s

Results - Bumpy sphere

- Bumpy sphere (9680 triangles)

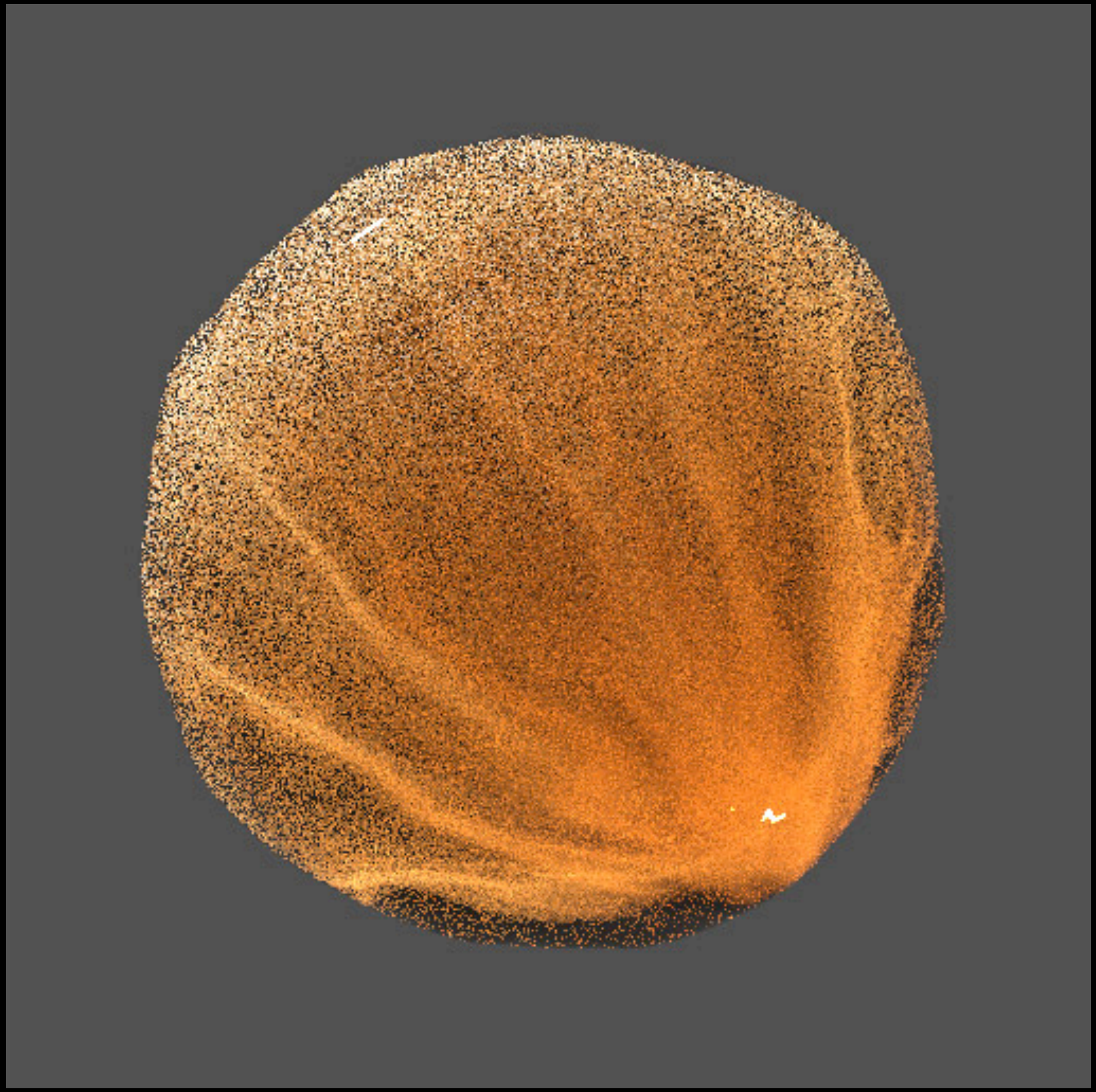
Shadow rays ignore refraction



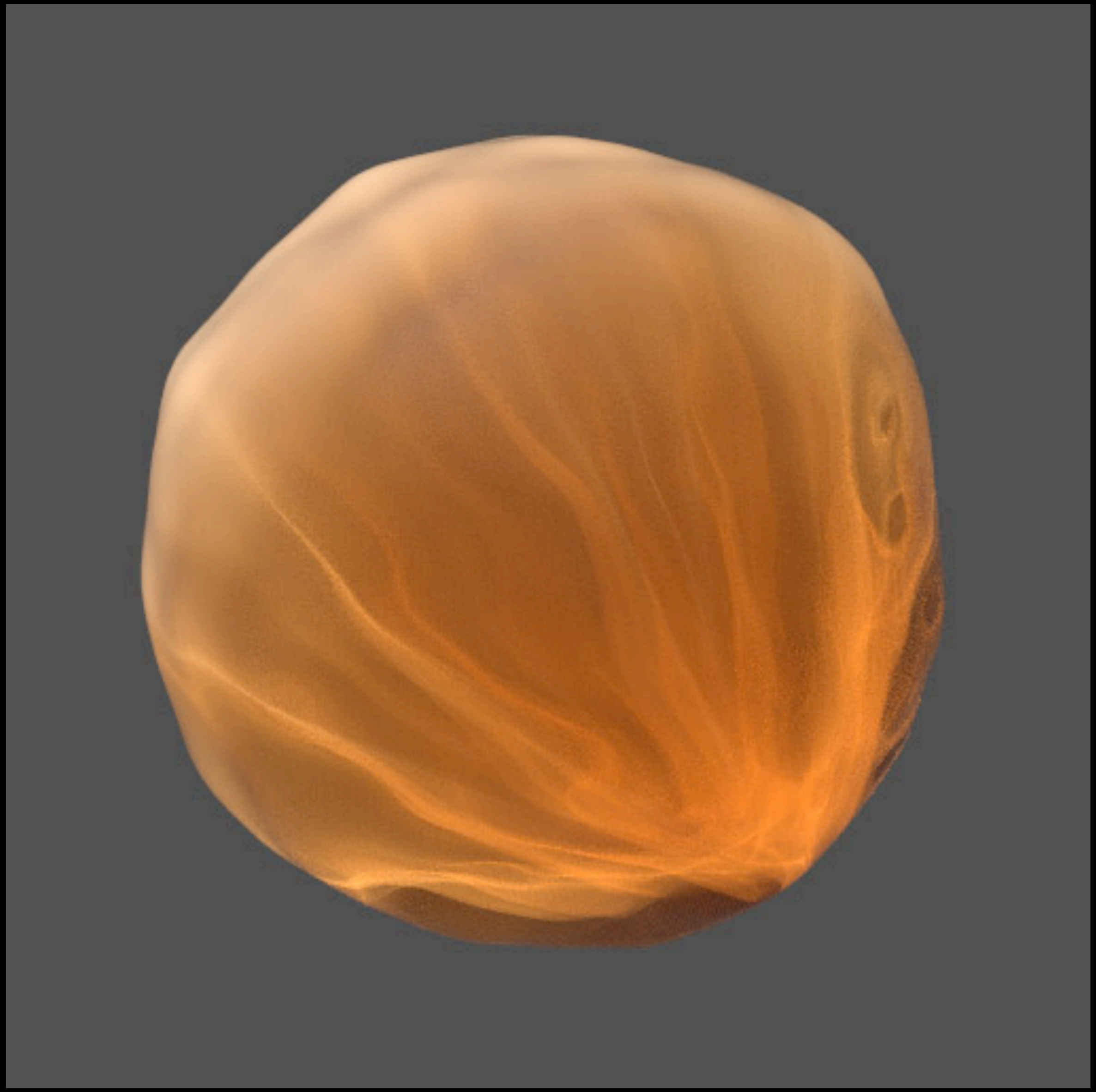
Our method

Results - Bumpy sphere

- Bumpy sphere (9680 triangles)



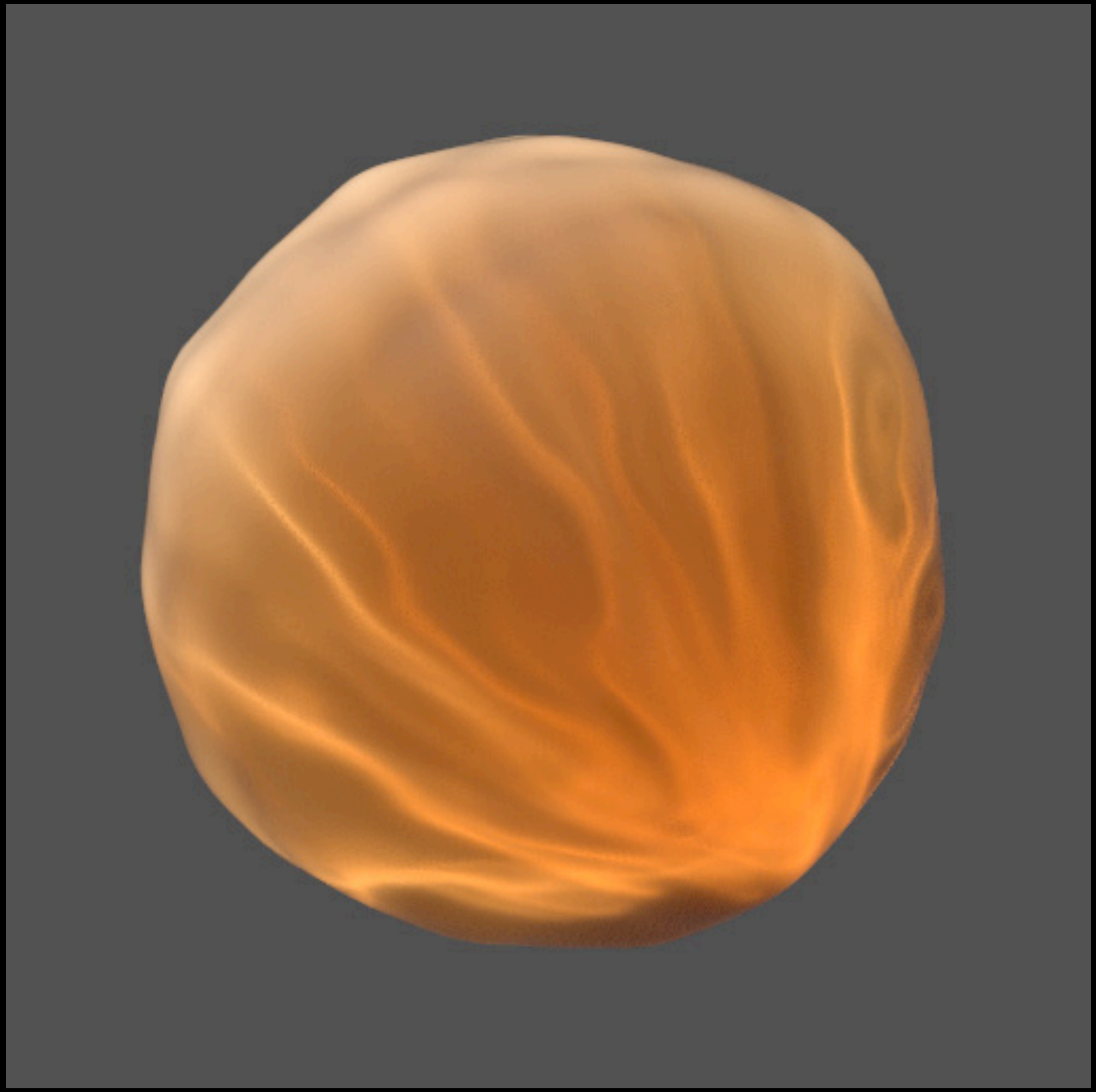
Path tracing(16x time)



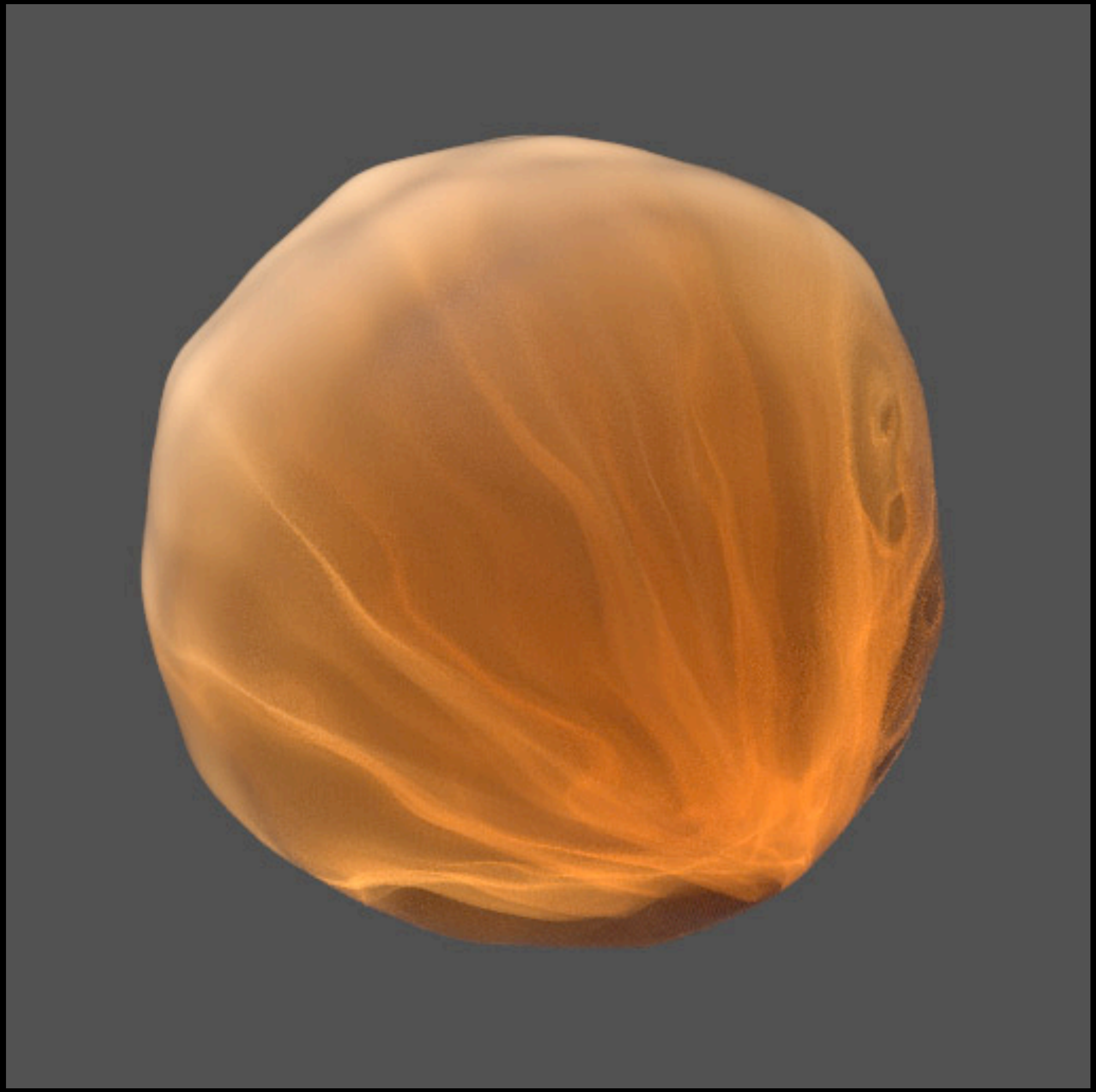
Our method

Results - Bumpy sphere

- Bumpy sphere (9680 triangles)



Photon map 10M (equal time)



Our method

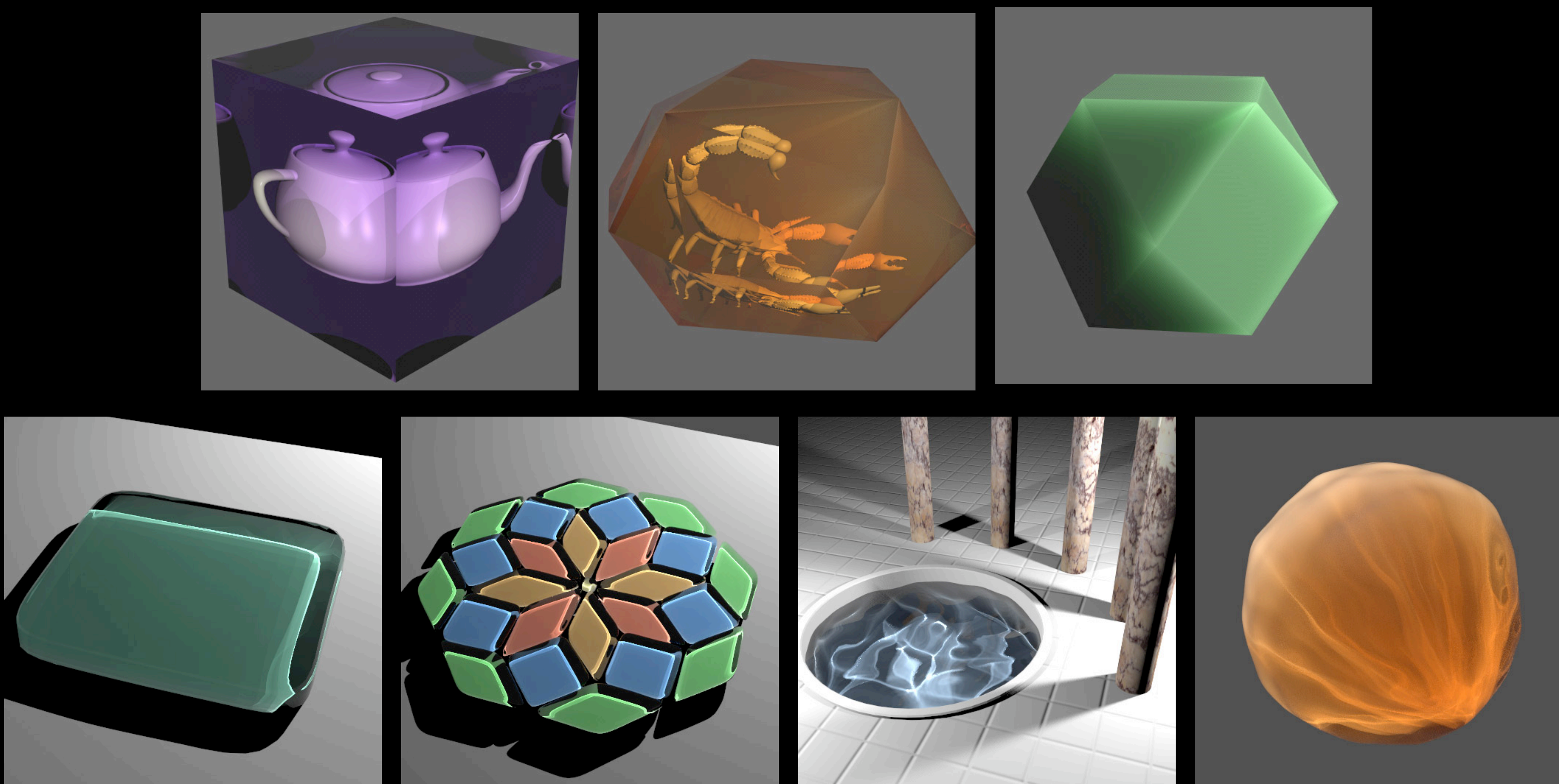
Conclusion

- New method for single scatter in refractive media
  - Applicable to many rendering algorithms
  - New half-vector formulation
  - Efficient culling and search methods
  - Supports shading normals and large triangle meshes
  - Interactive performance for some scenes
- Future work
  - Better culling
  - Reflections and low-order scattering
  - Multiple interfaces

Acknowledgements

- Sponsors
  - NSF
    - Career 0644175, CPA 0811680, CNS 0615240, CNS 0403340
  - Intel
  - NVidia
  - Microsoft
  - INRIA sabbatical program
- PCG Graphics Lab and Elizabeth Popolo

The End



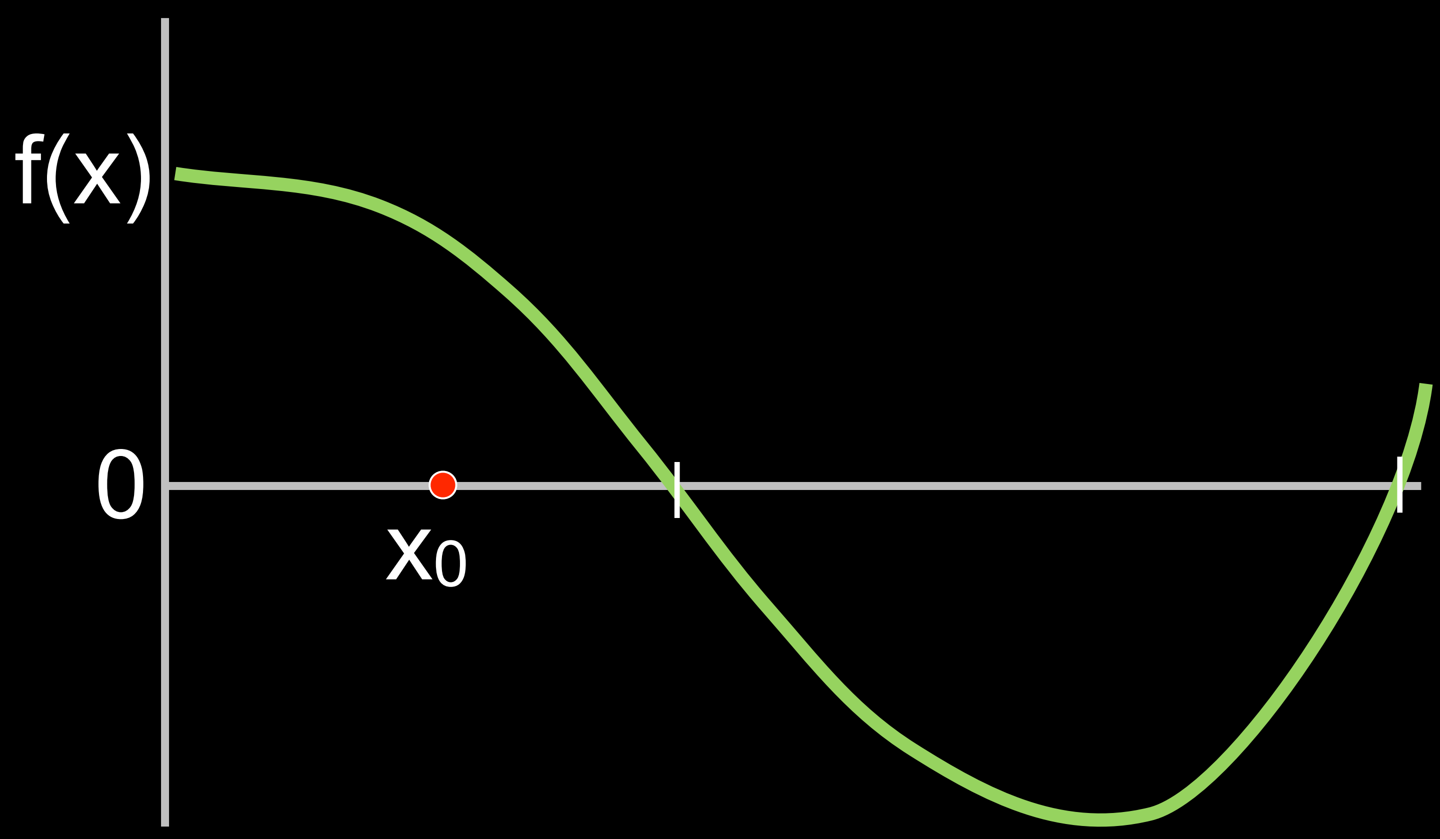
Results - CPU timings

Name	Render Time	Triangles		Shading Normals
		Surface	Other	
Teapot	15.3 s	12	4096	No
Cuboctahedron	13.9 s	20	0	No
Amber	19.2 s	36	60556	No
Glass tile	66.9 s	798	60	Yes
Glass mosaic	87.8 s	20813	1450	Yes
Pool	59.4 s	2632	4324	Yes
Bumpy Sphere	304.3 s	9680	0	Yes

512x512 images, 64 samples per pixel (128 for bumpy sphere), 8-core 2.83GHz Intel Core2 CPU

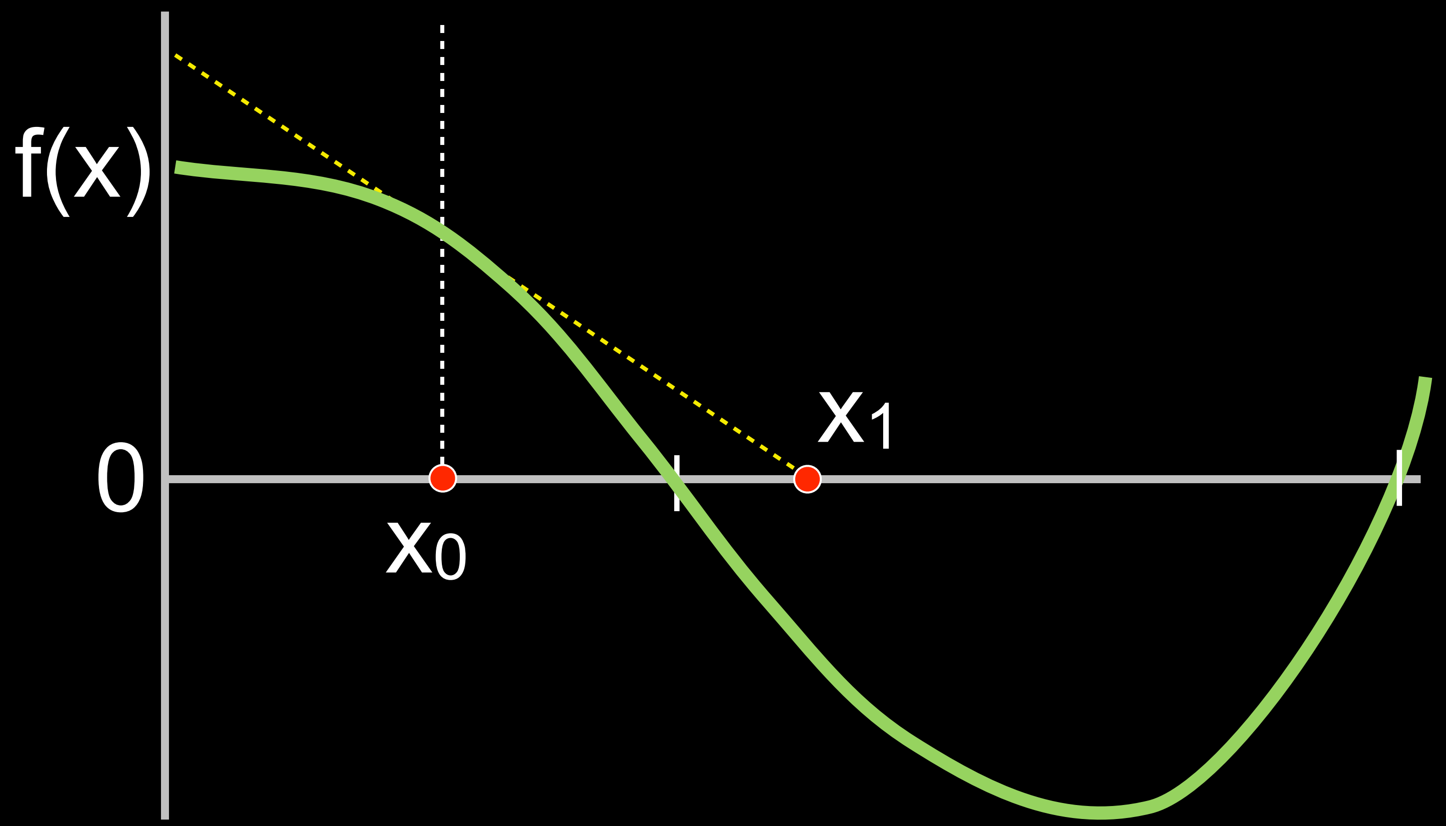
Newton's Method

- Iterative root finding method
  - Start with initial guess x0
  - Iteration: $x_{i+1} = x_i - f(x_i) / f'(x_i)$



Newton's Method

- Iterative root finding method
  - Start with initial guess x0
  - $x_{i+1} = x_i - f(x_i) / f'(x_i)$



Newton's Method

- Iterative root finding method
  - Start with initial guess x0
  - $x_{i+1} = x_i - f(x_i) / f'(x_i)$