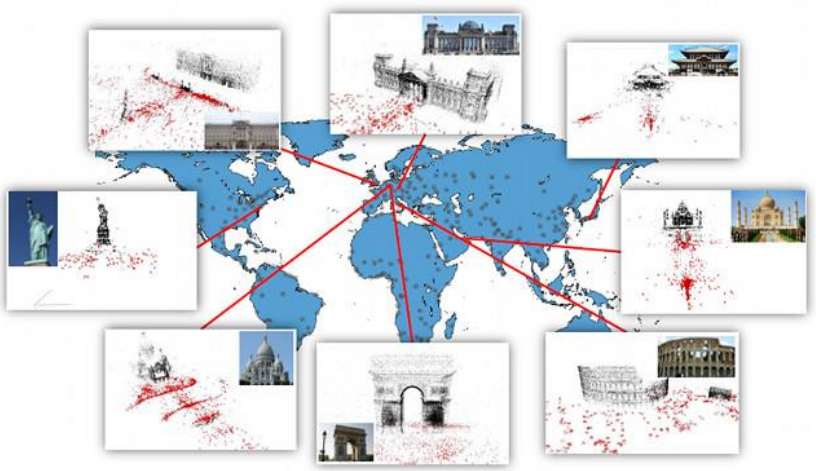
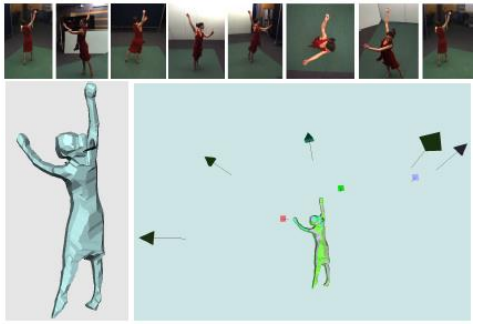
# Shape from X

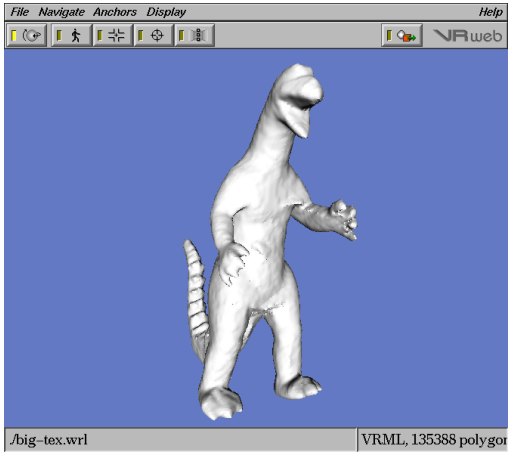




# Schedule

| # | Date | Topic |
|---|---|---|
| 1 | 20.09. | Introduction & Pinhole model |
| 2 | 27.09. | Feature extraction |
| 3 | 04.10. | Camera models & calibration |
| 4 | 11.10. | Optical flow & Particle filtering |
| 5 | 18.10. | Model fitting |
| 6 | 25.10. | - no lecture - |
| 7 | 01.11. | Stereo matching & Multi-view |
| 8 | 08.11. | Shape from X |
| 9 | 15.11. | Structure-from-Motion |
| 10 | 22.11. | Specific object recognition |
| 11 | 29.11. | Object category recognition |
| 12 | 06.12. | Image Segmentation |
| 13 | 13.12. | Research Overview & Lab Tours |
| 14 | 20.12. | Tracking |

# Shape from silhouettes

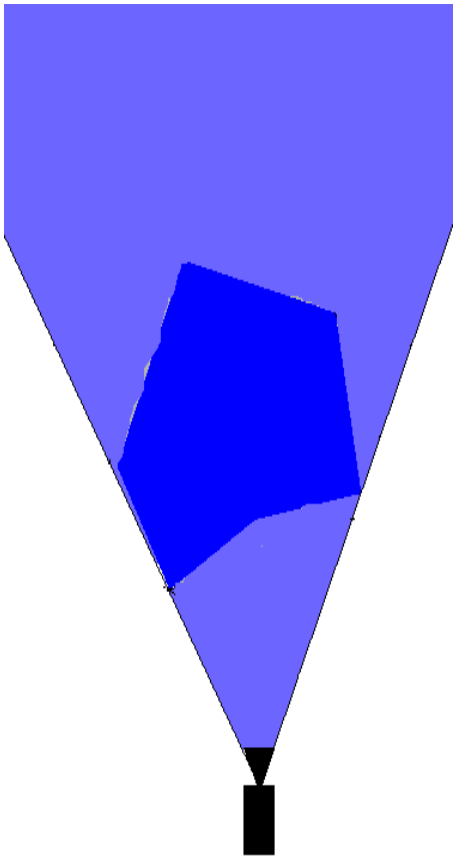




Automatic 3D Model Construction for Turn-Table Sequences,
A.W. Fitzgibbon, G. Cross, and A. Zisserman, SMILE 1998

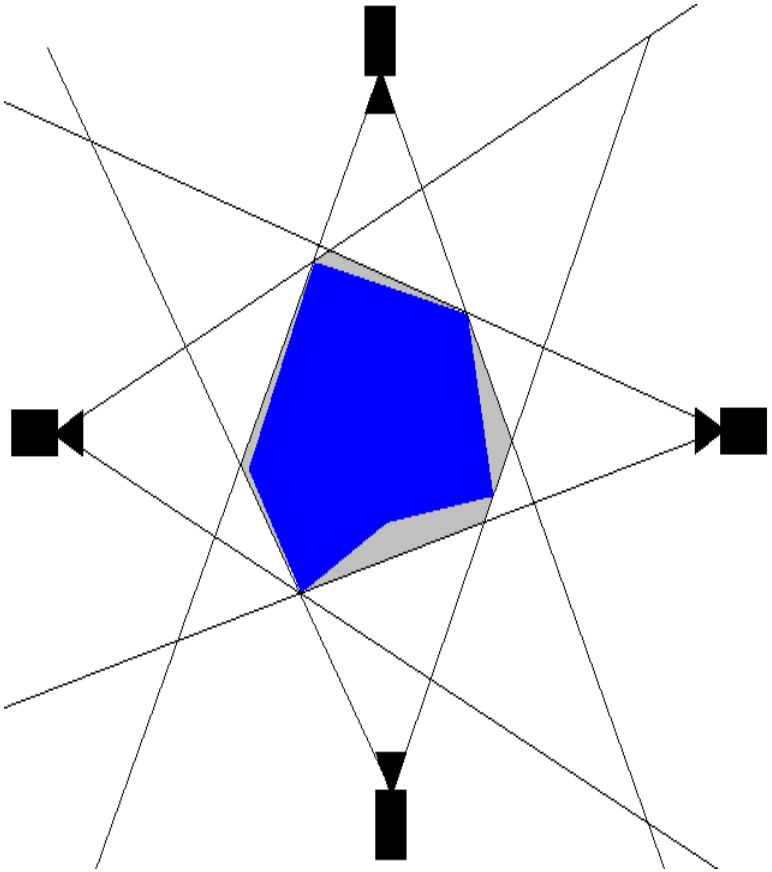
# Visual Hull (Blackboard)

# What is shape from silhouette?



- The silhouette, or occluding contour of an object in an image contains some information about the 3D shape of the object.
- Given a single silhouette image of an object, we know that the 3D object lies inside the volume generated by back-projecting the silhouette area using the camera parameters.

# What is shape from silhouette?



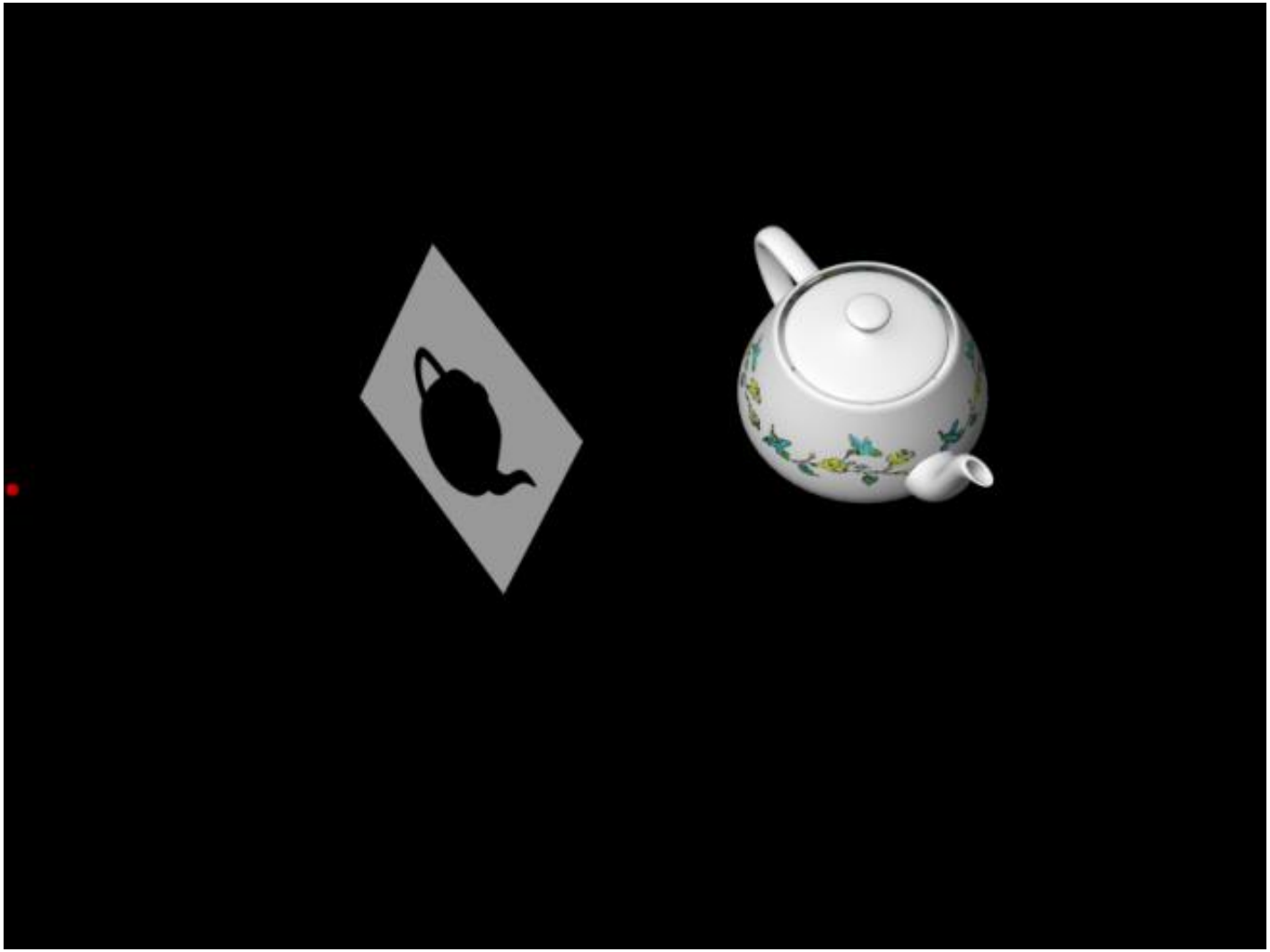
- With multiple views of the same object, we can intersect the *generalized cones* generated by each image, to build a volume which is guaranteed to contain the object.
- The limiting smallest volume obtainable in this way is known as the ***visual hull*** of the object.
- *What is the relationship to the convex hull of an object?*

# Literature

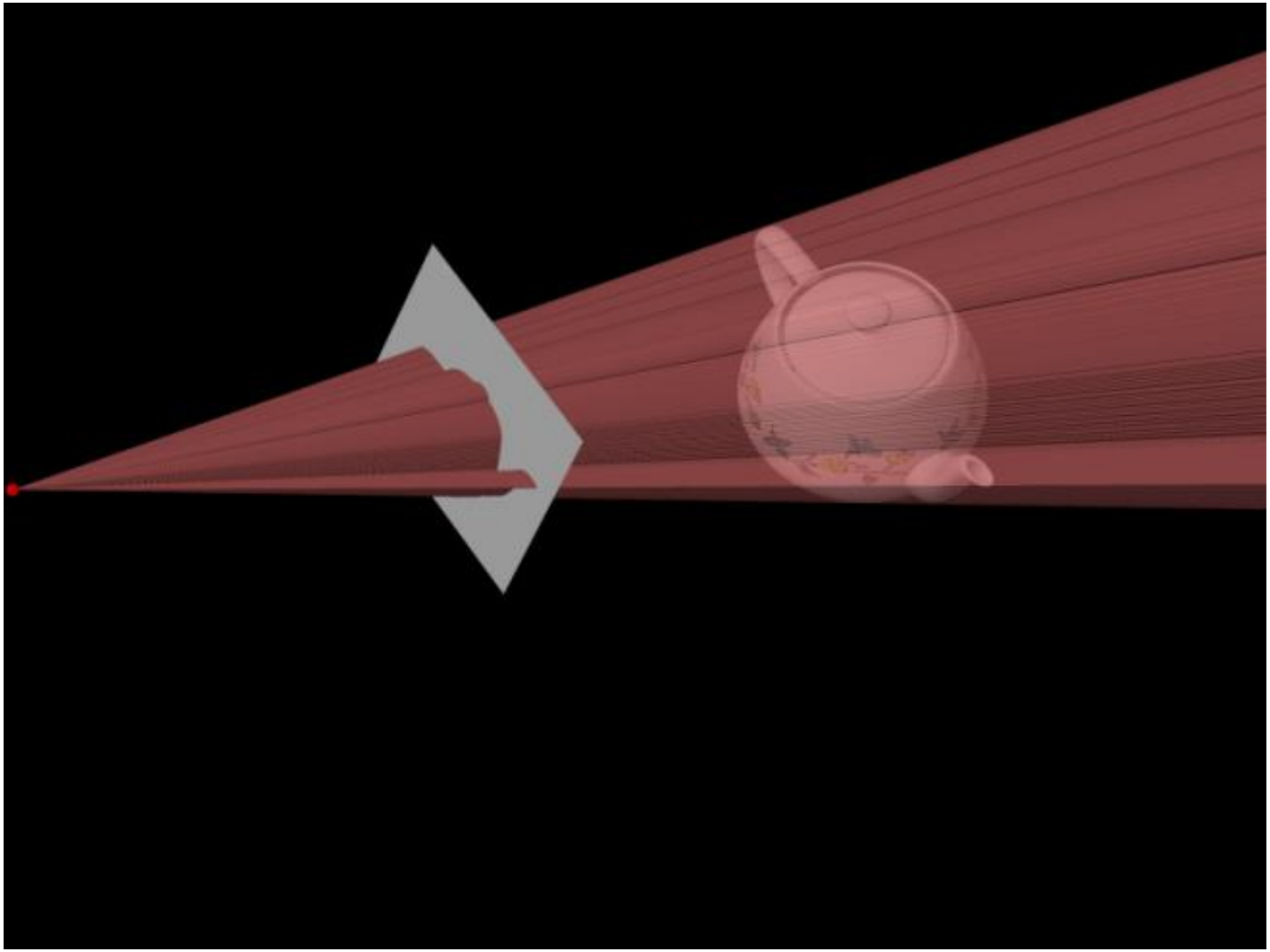
- Theory
  - Laurentini ’94, Petitjean ’98, Laurentini ’99
- Solid cone intersection:
  - Baumgart ’74 (polyhedra), Szeliski ’93 (octrees)
- Image-based visual hulls
  - Matusik et al. ’00, Matusik et al. ’01
- Advanced modeling
  - Sullivan & Ponce ’98, Cross & Zisserman ’00, Matusik et al. ’02
- Applications
  - Leibe et al. ’00, Lok ’01, Shlyakhter et al. ’01

# Visual Hull: A 3D Example

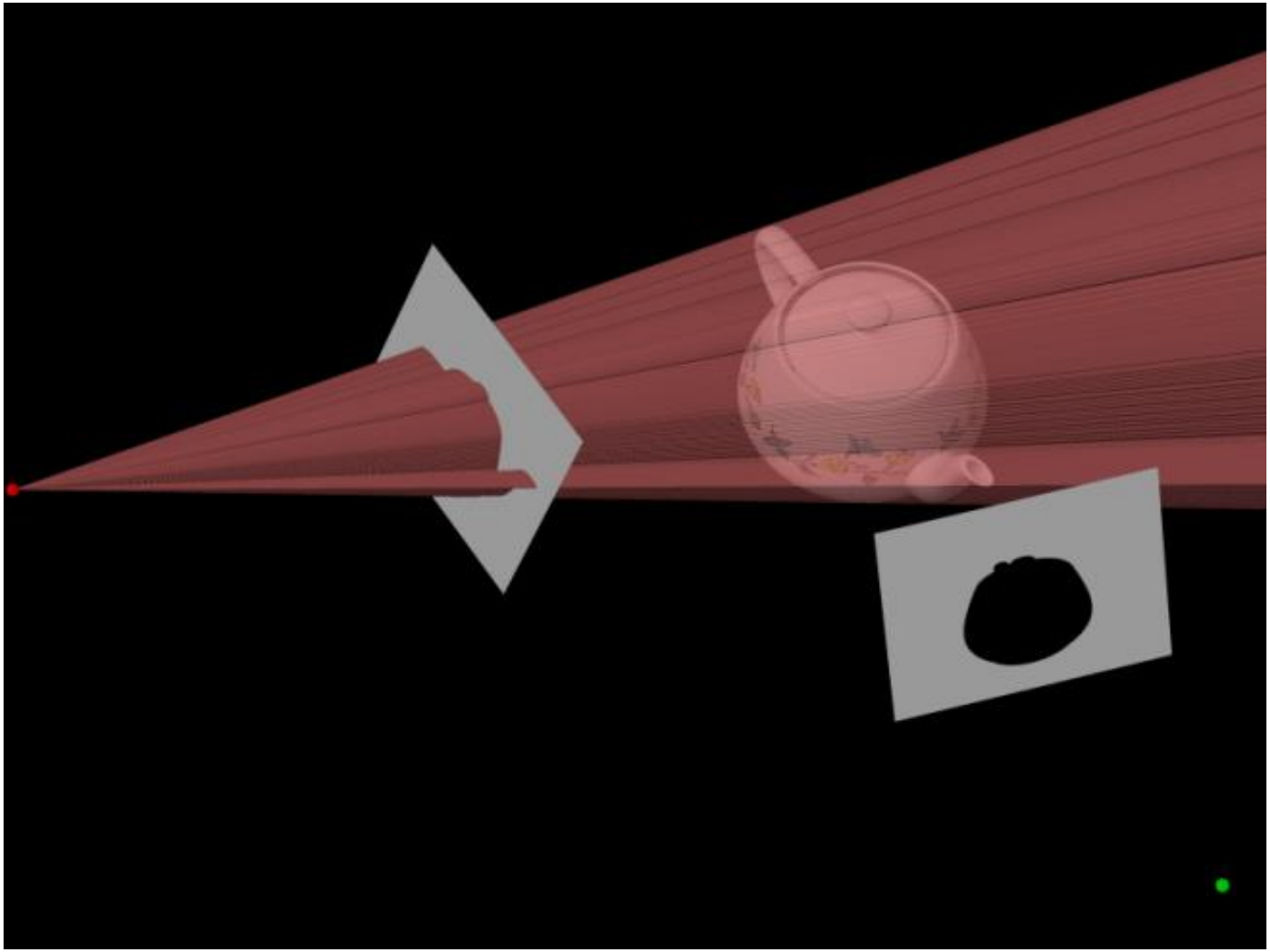
# Visual Hull: A 3D Example



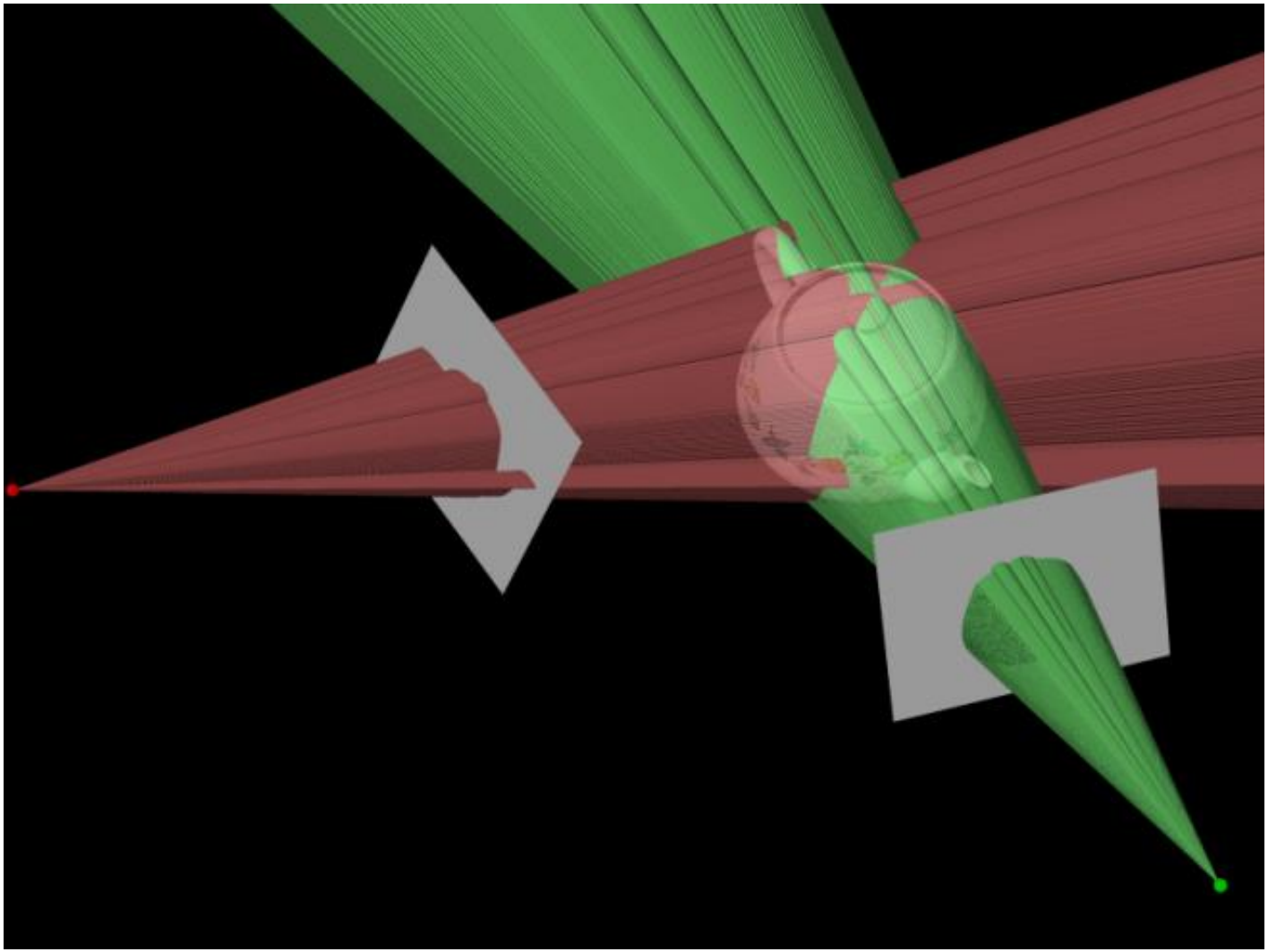
# Visual Hull: A 3D Example



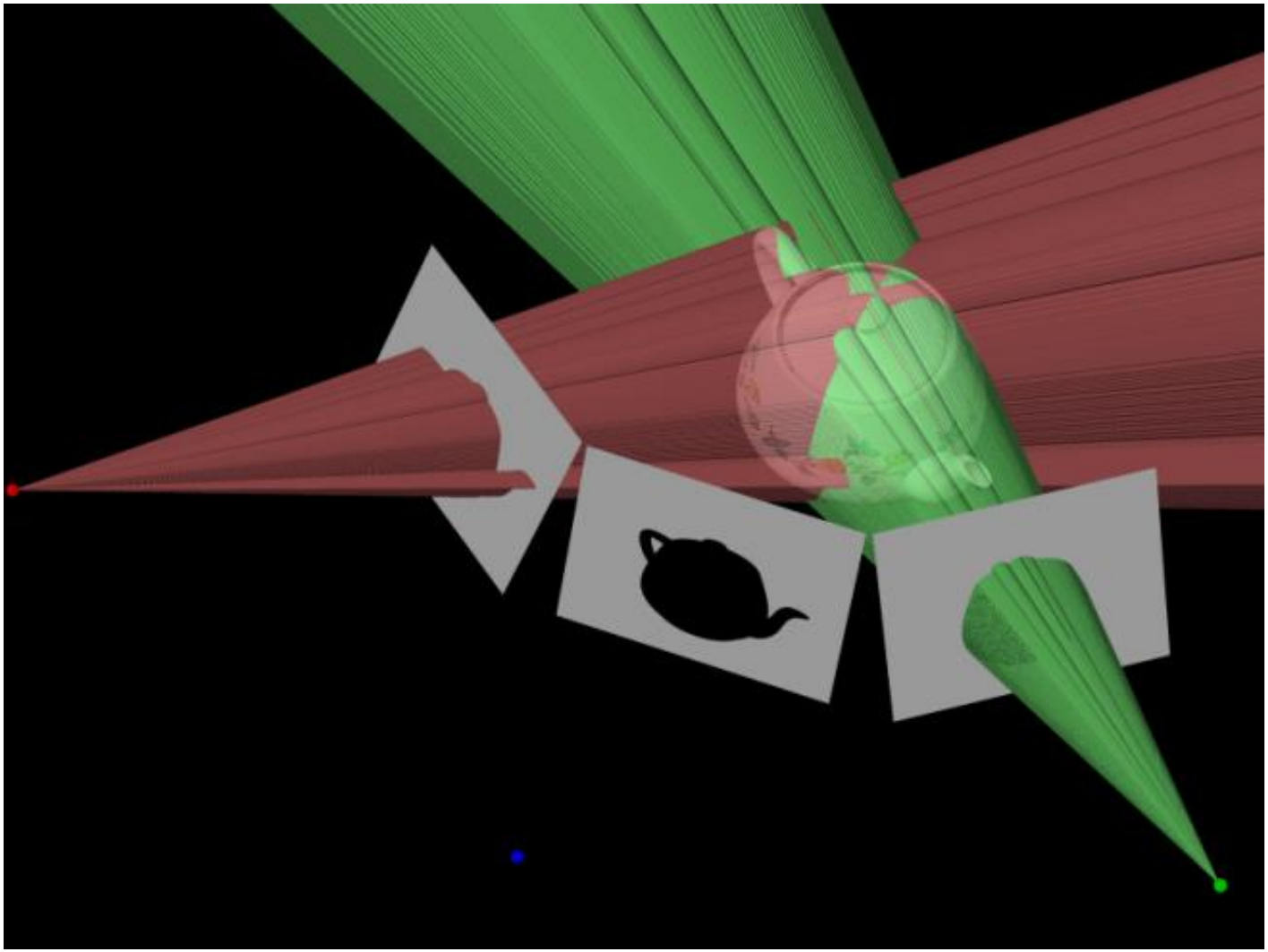
# Visual Hull: A 3D Example



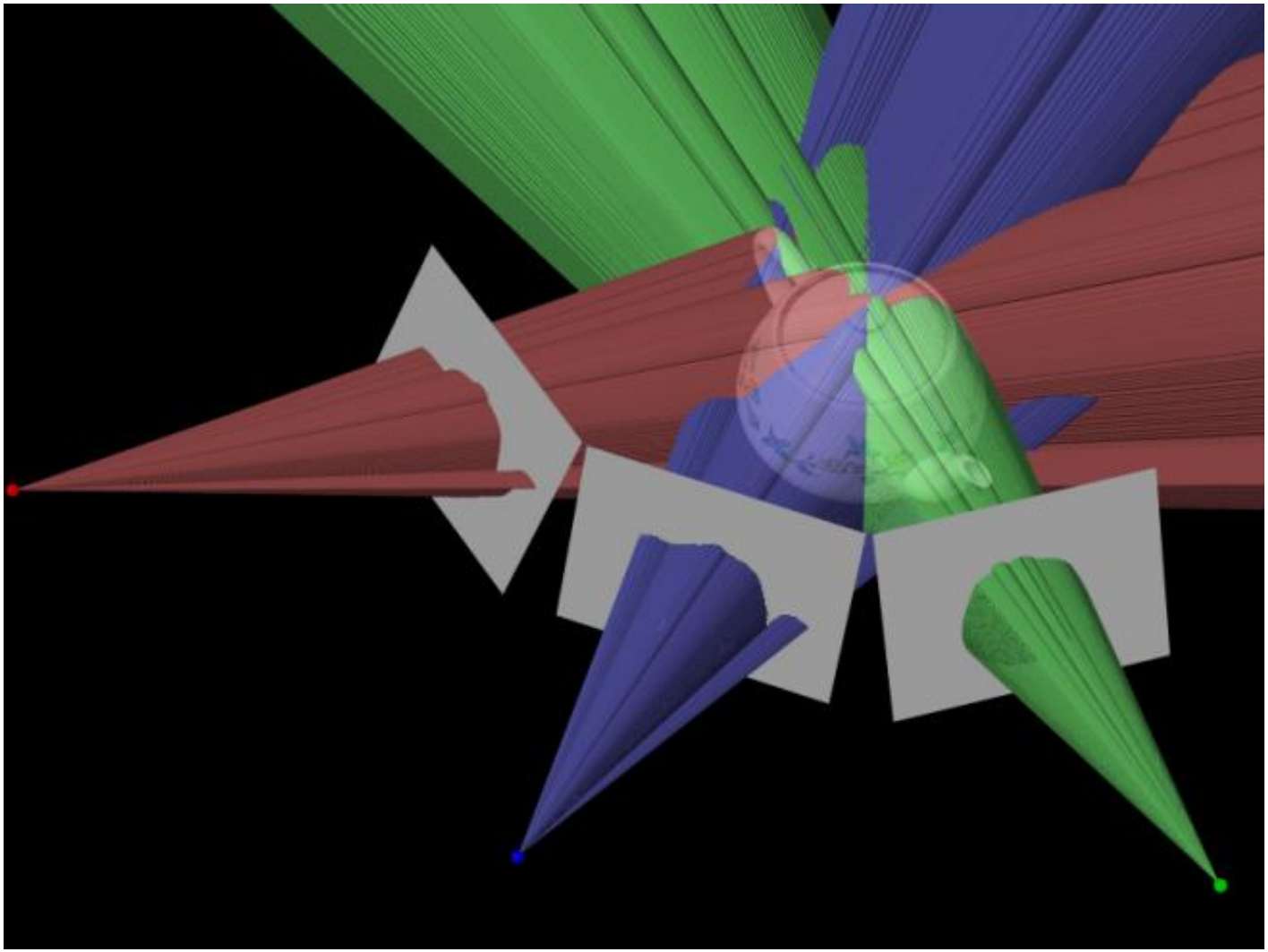
# Visual Hull: A 3D Example



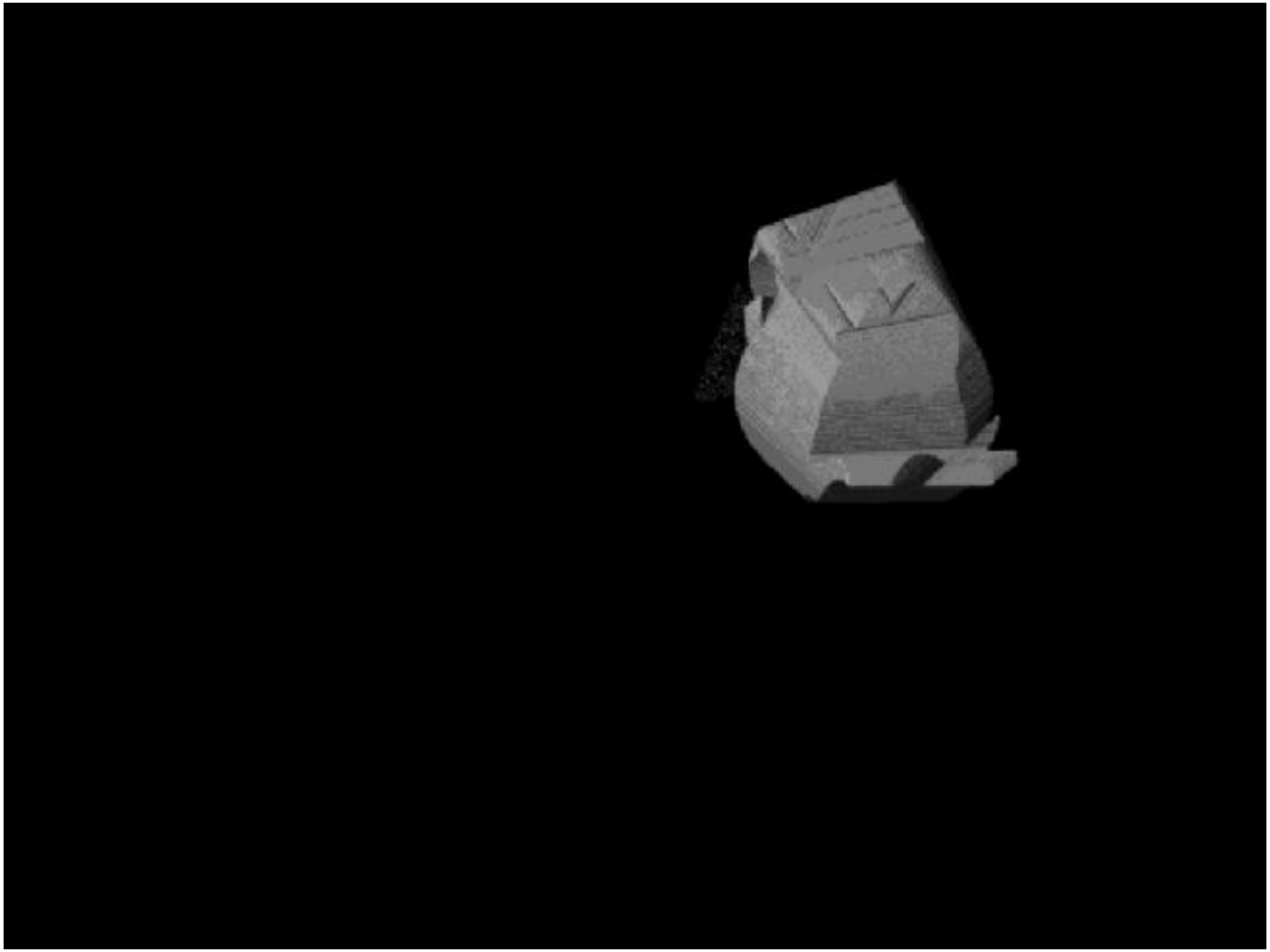
# Visual Hull: A 3D Example



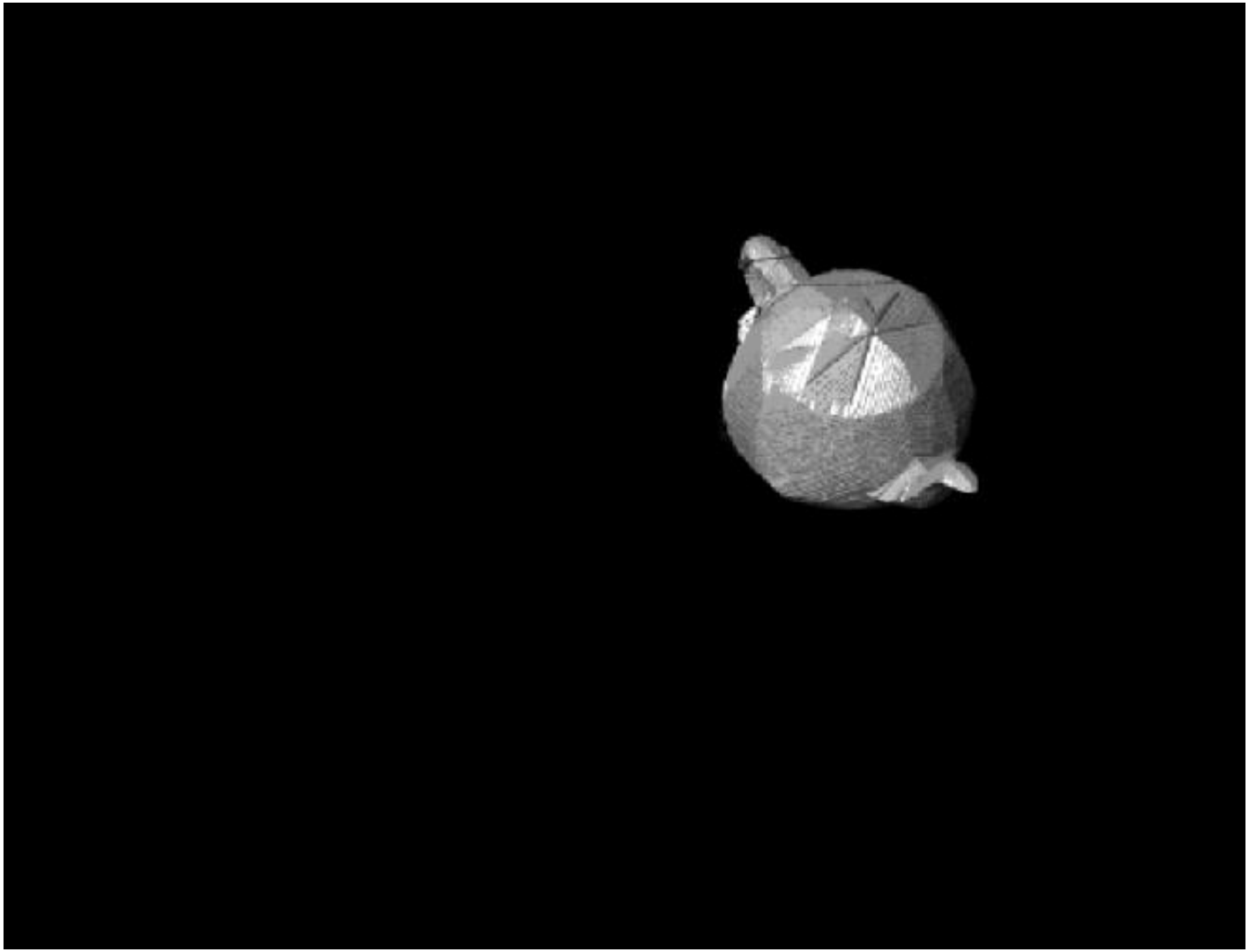
# Visual Hull: A 3D Example



# Visual Hull: A 3D Example

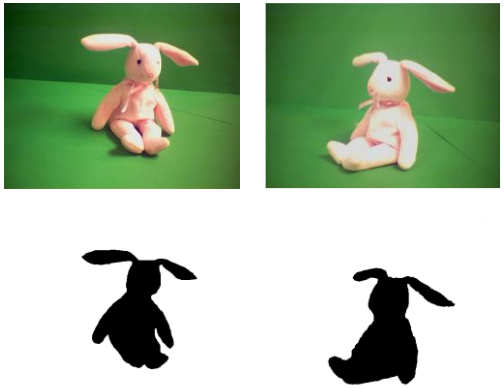


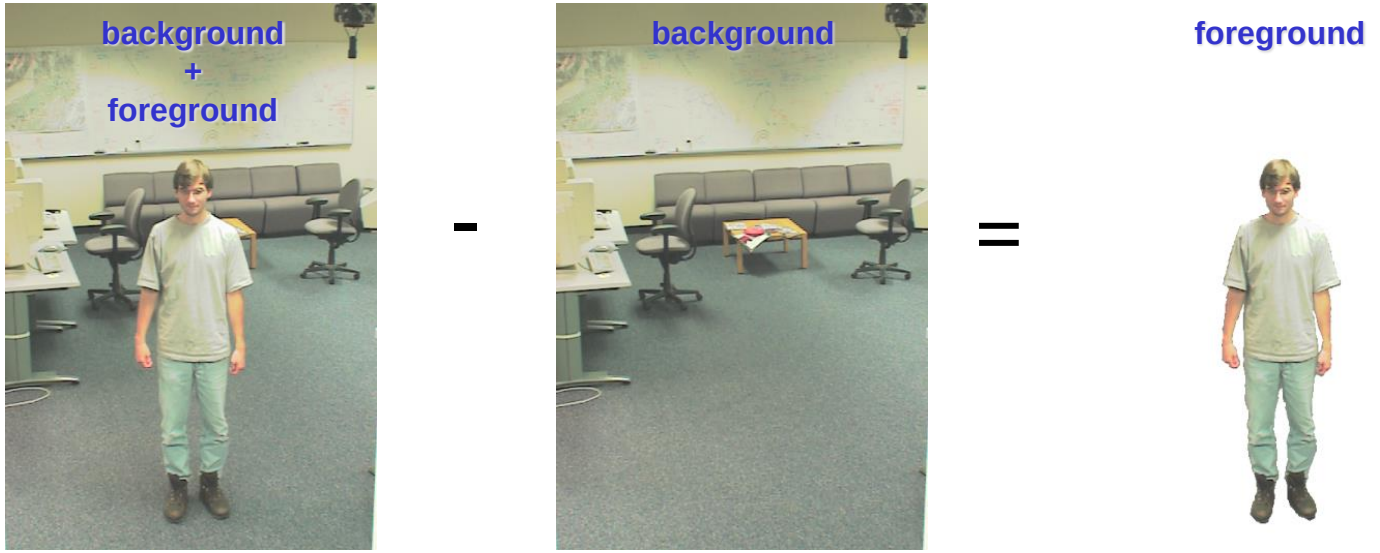
# Visual Hull: A 3D Example



# Why use a Visual Hull?

- Can be computed robustly
- Can be computed efficiently





# Computational complexity

- Intersection of many volumes can be slow
- Simple polyhedron-polyhedron intersection algorithms are inefficient

- To improve performance, most methods:
  - Quantize volumes and/or
  - Perform Intersection computations in 2D not 3D

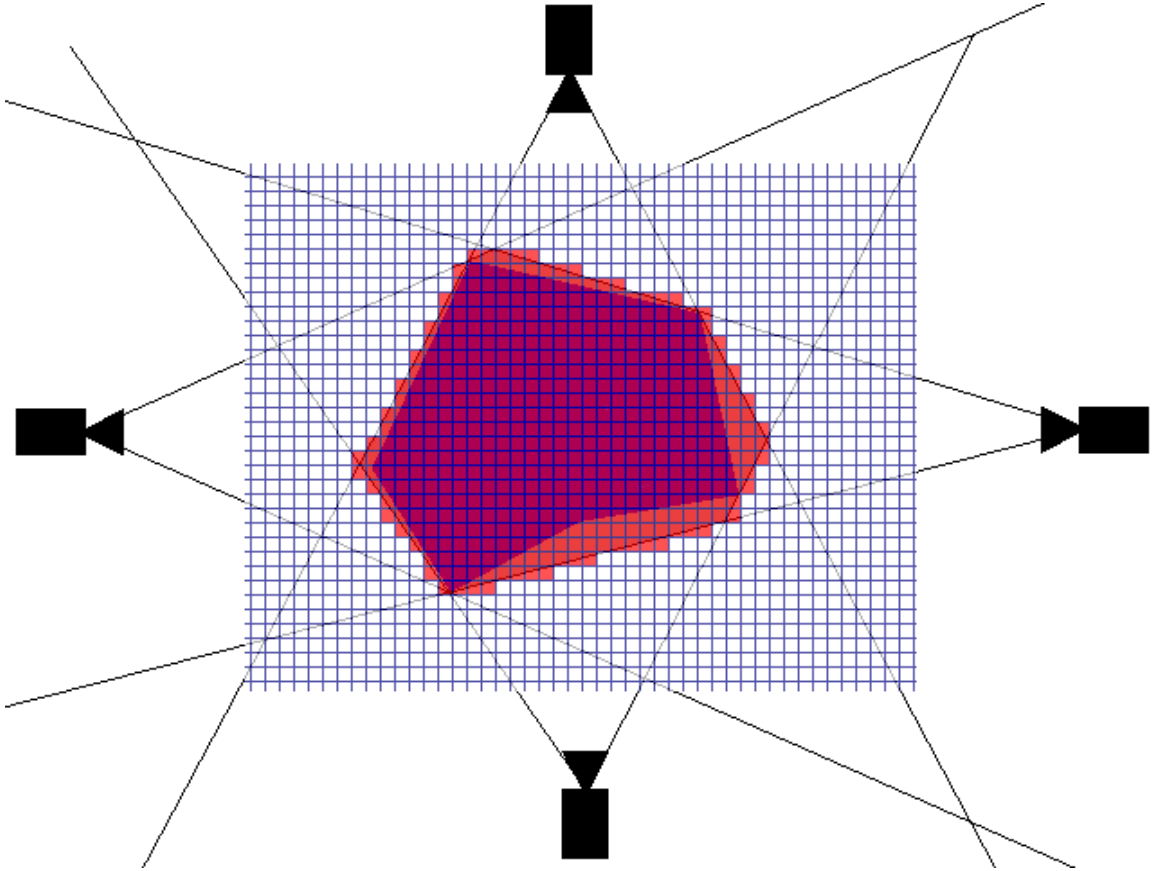
# Algorithms

- Standard voxel based method
- Marching intersections
- Image-based visual hulls
- Exact polyhedral methods

# Voxel based

- First the object space is split up into a 3D grid of voxels.
- Each voxel is intersected with each silhouette volume.
- Only voxels that lie inside all silhouette volumes remain part of the final shape.

# Voxel based - example

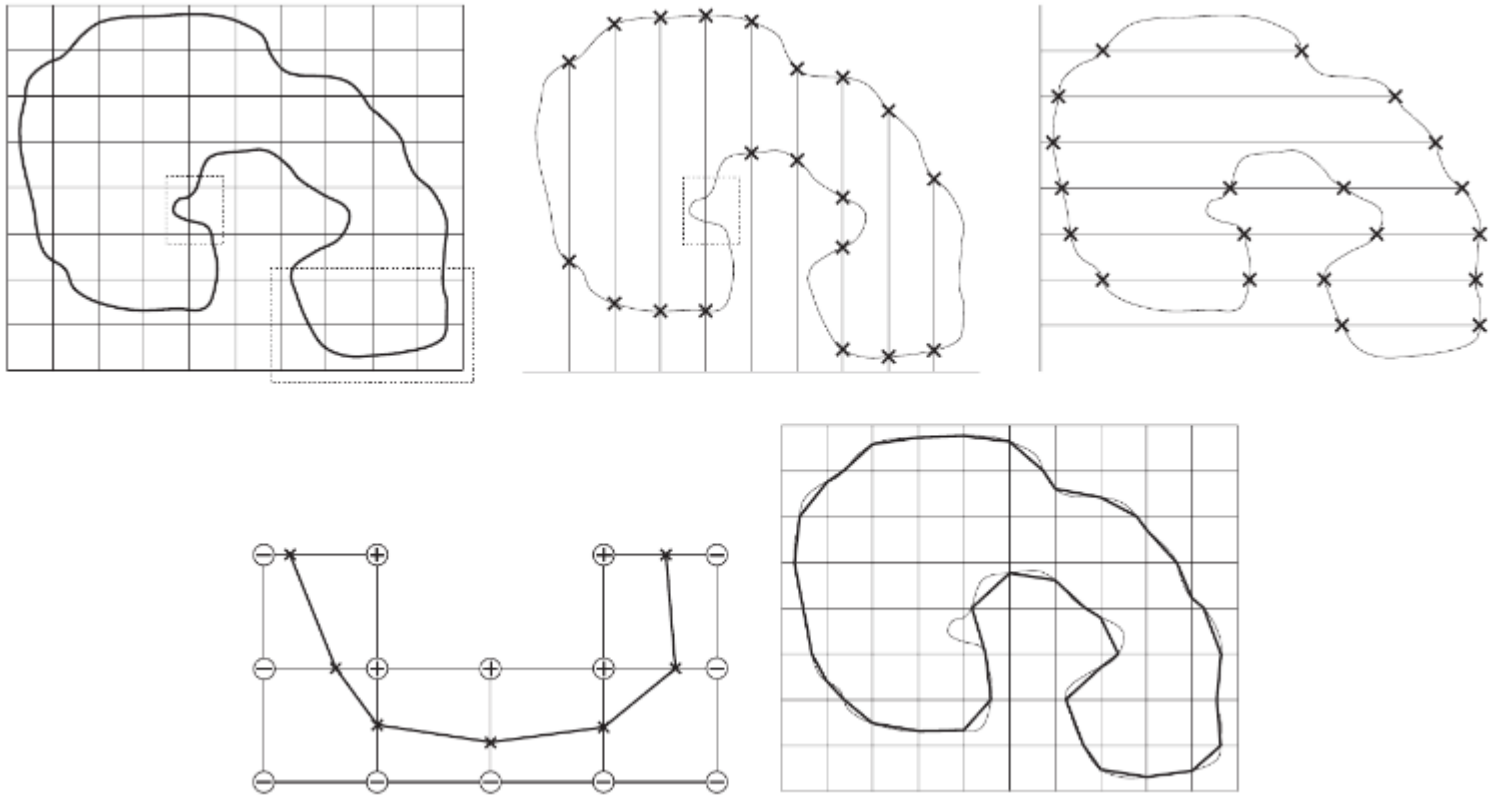


# Marching intersections

- The object space is again split up into a 3D grid.
- The grid that is used is made of 3 sets of rays, rather than voxels.
- Rays are aligned with the 3 axes, and store points of entry/exit into the volume
- Each silhouette cone can be converted to the marching intersections data structure.
- Then merging them is reduced to 1D intersections along each ray.

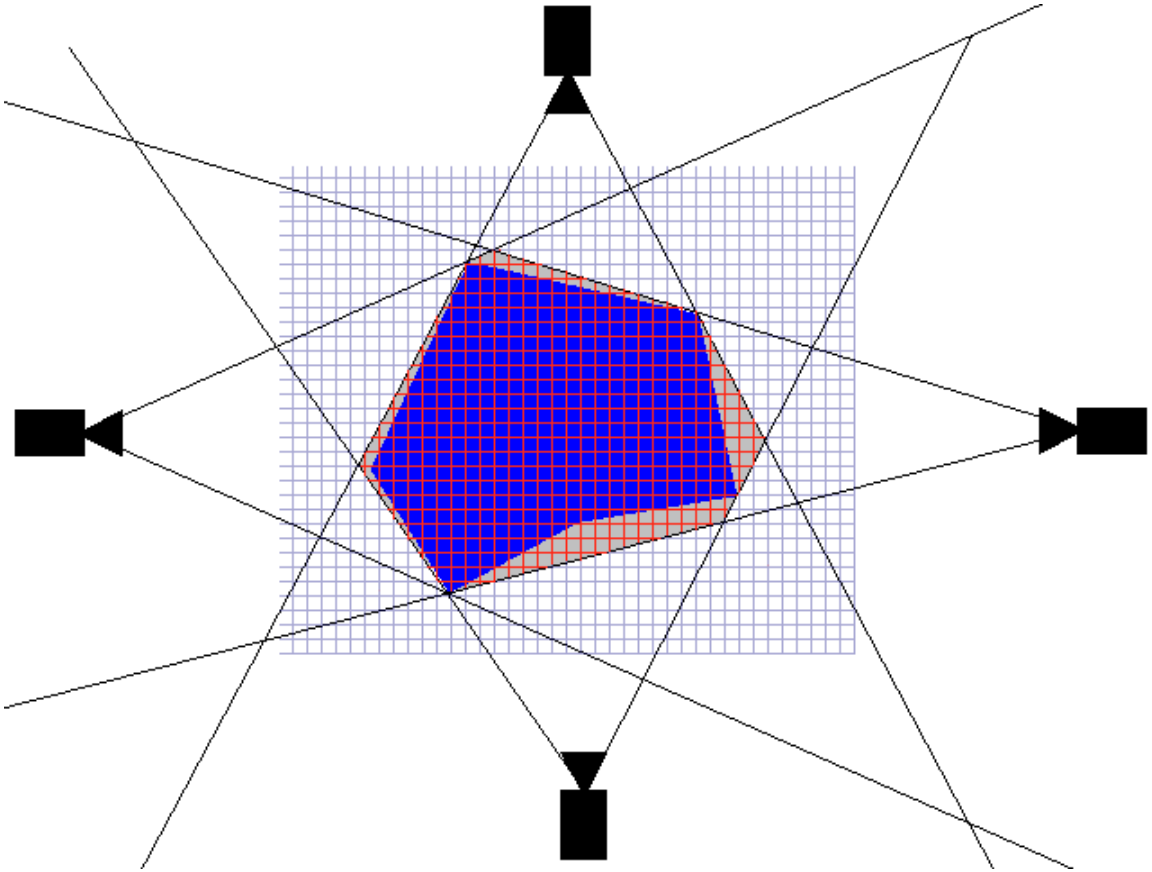
M. Tarini et al, *Marching intersections, An efficient Approach to Shape from Silhouette*

# Marching intersections



M. Tarini et al, *Marching intersections, An efficient Approach to Shape from Silhouette*

# Marching intersections - example

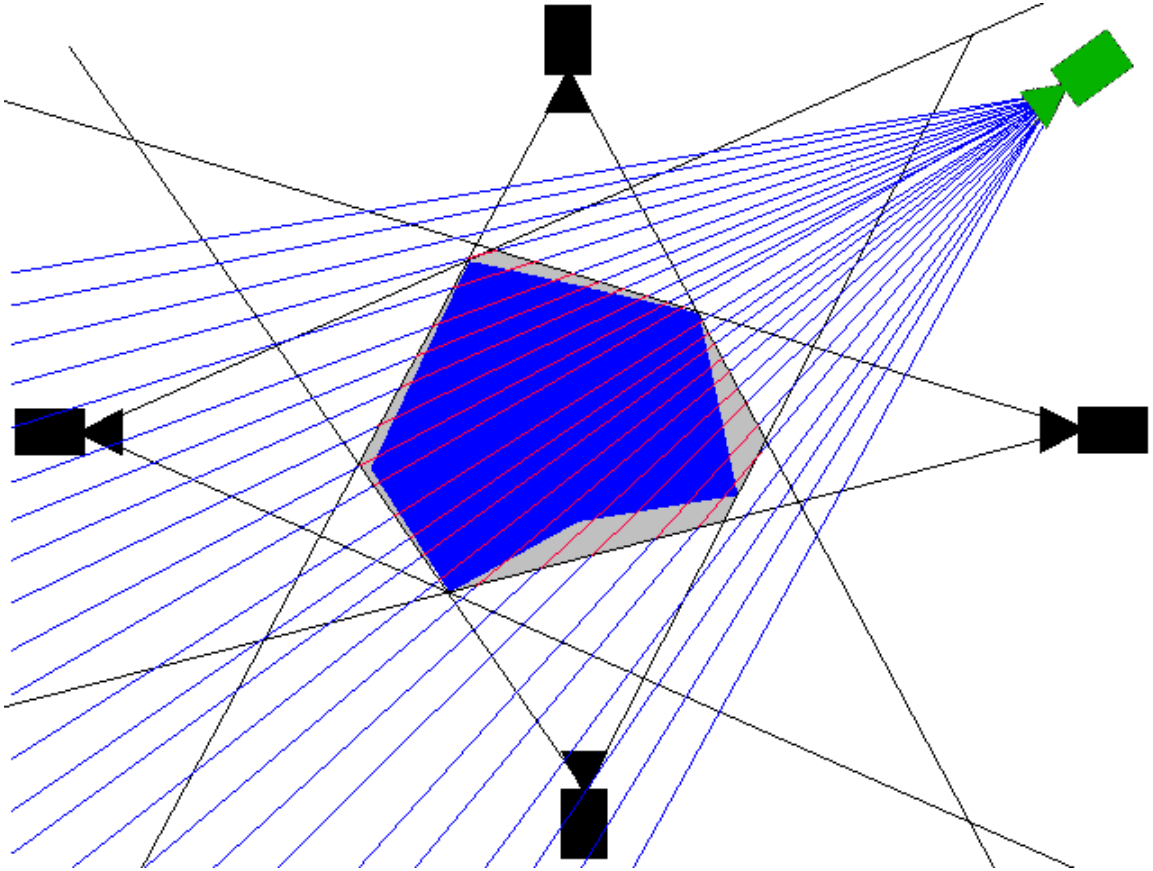


# Image based visual hulls

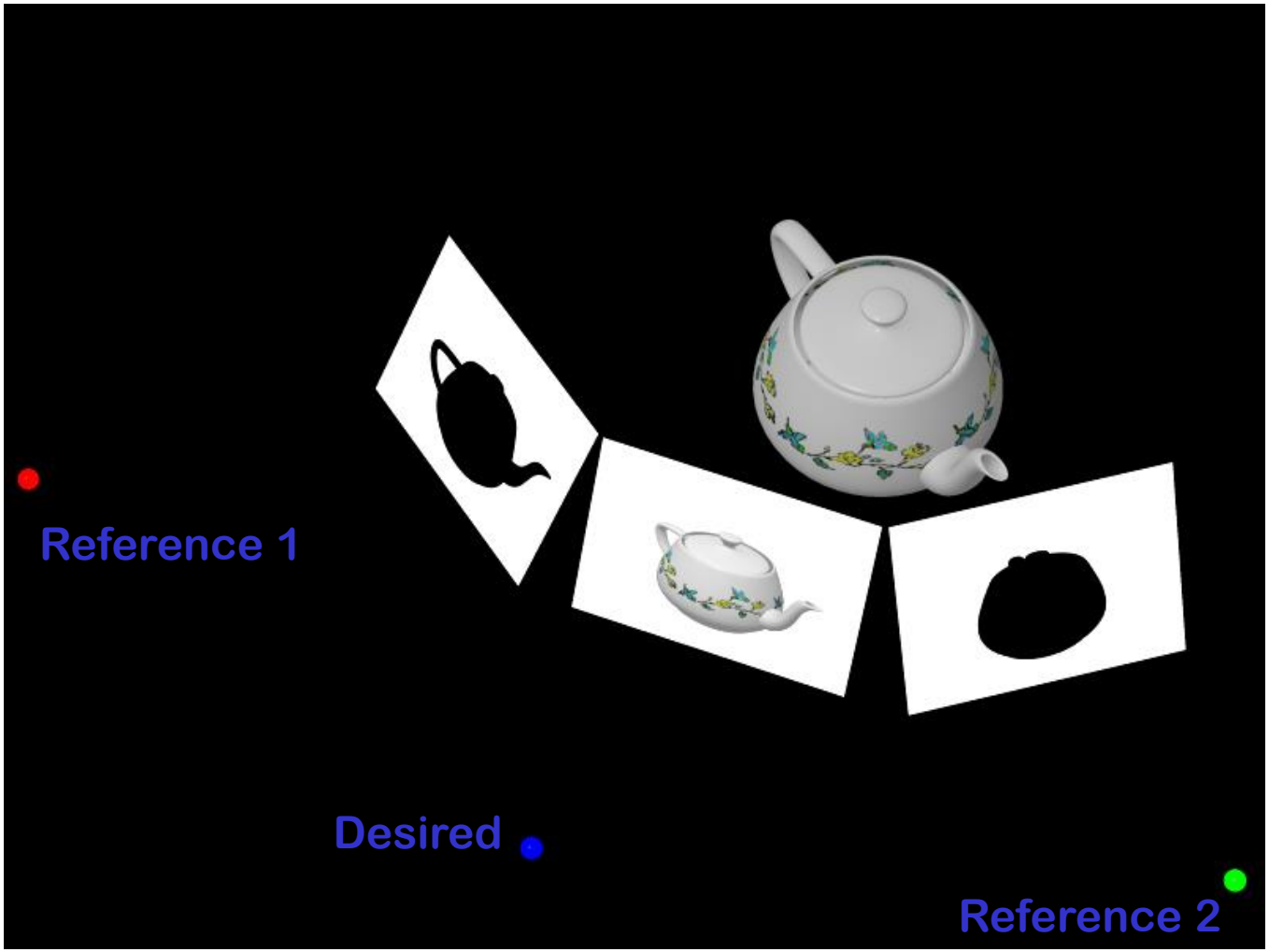
- This algorithm will only produce renderings of a visual hull from any view.
- Every pixel in the desired output image is back-projected to form a 3D ray.
- Each of those rays is intersected with each of the input silhouettes in the same way as the rays in the marching intersections method.
- Then a pixel in the output image is inside the new rendering of the visual hull if its ray has any segments left in it that are intersecting the visual hull. The depth of these pixels is known from the depth of the nearest entry point on the ray

Wojciech Matusik, *Image Based Visual Hulls*

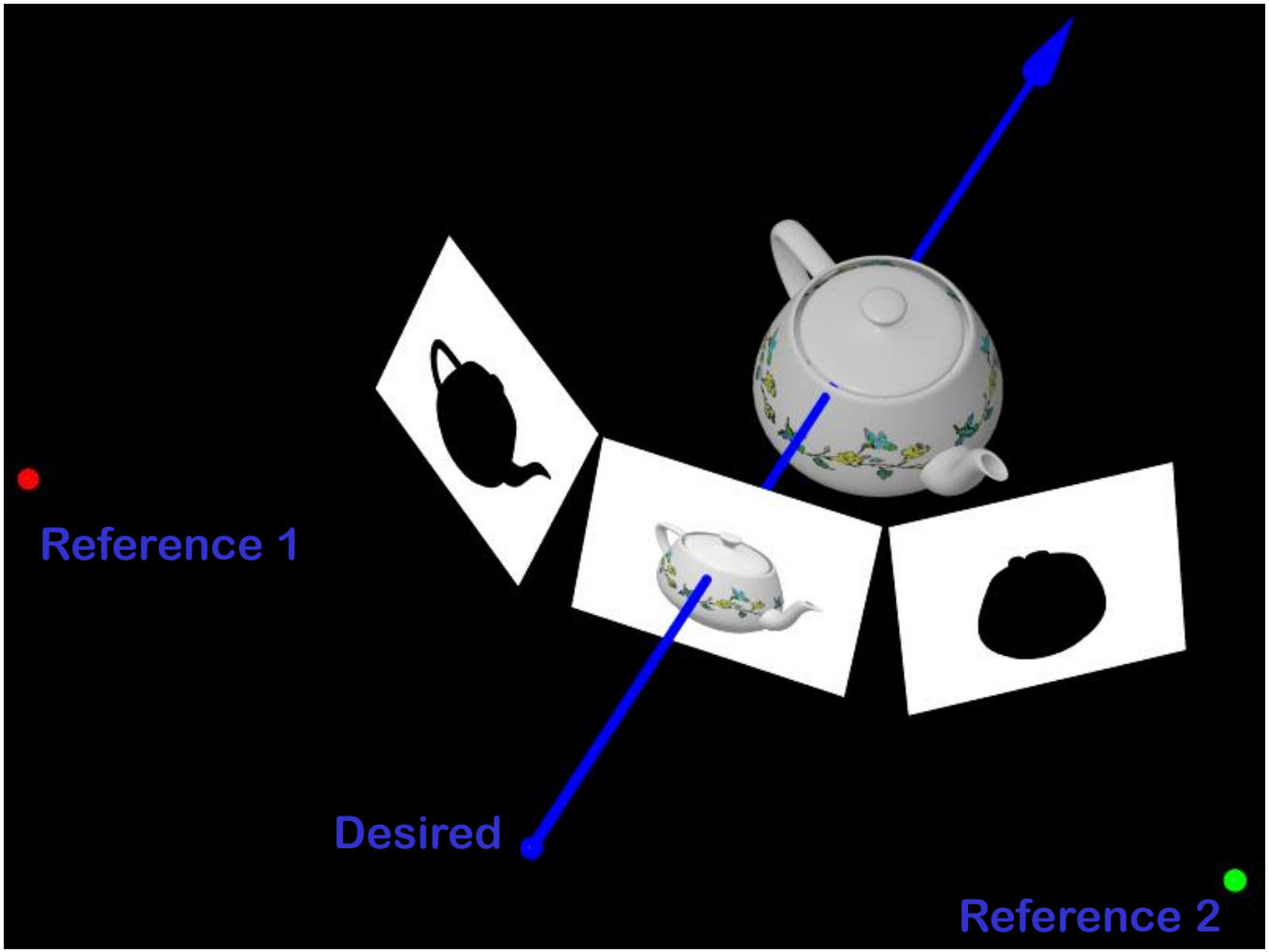
# Image based - example



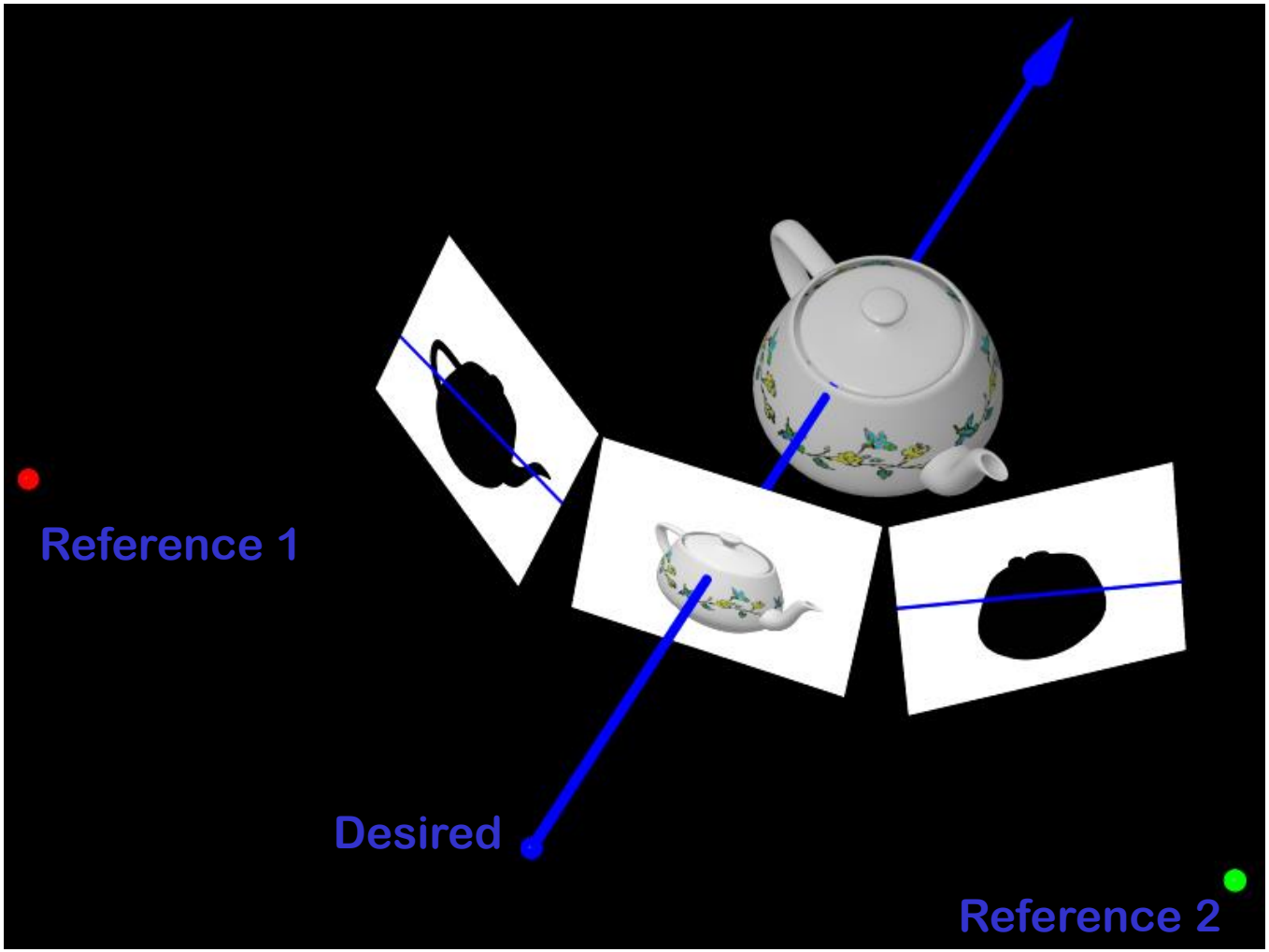
# Image-Based Computation



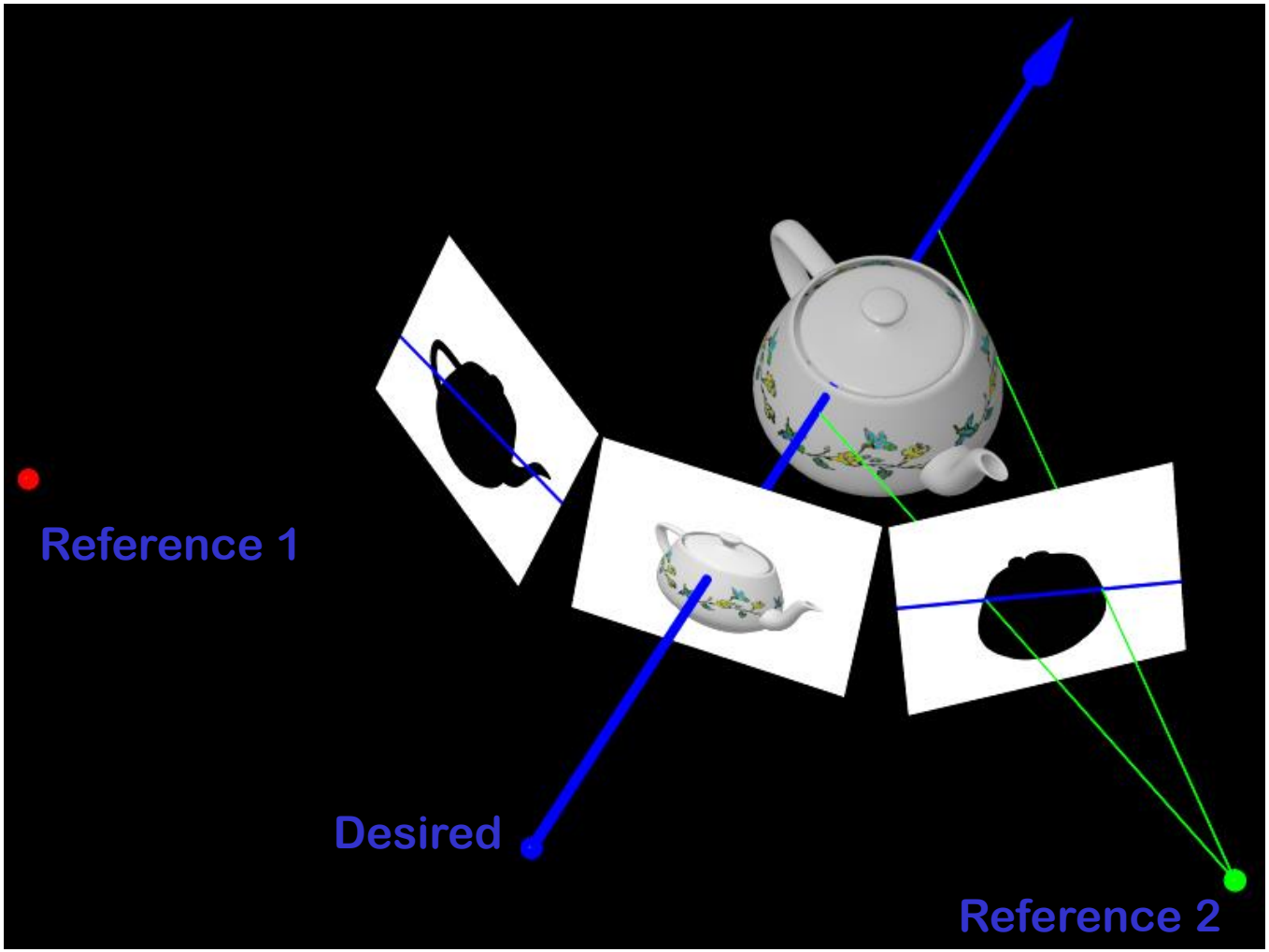
# Image-Based Computation



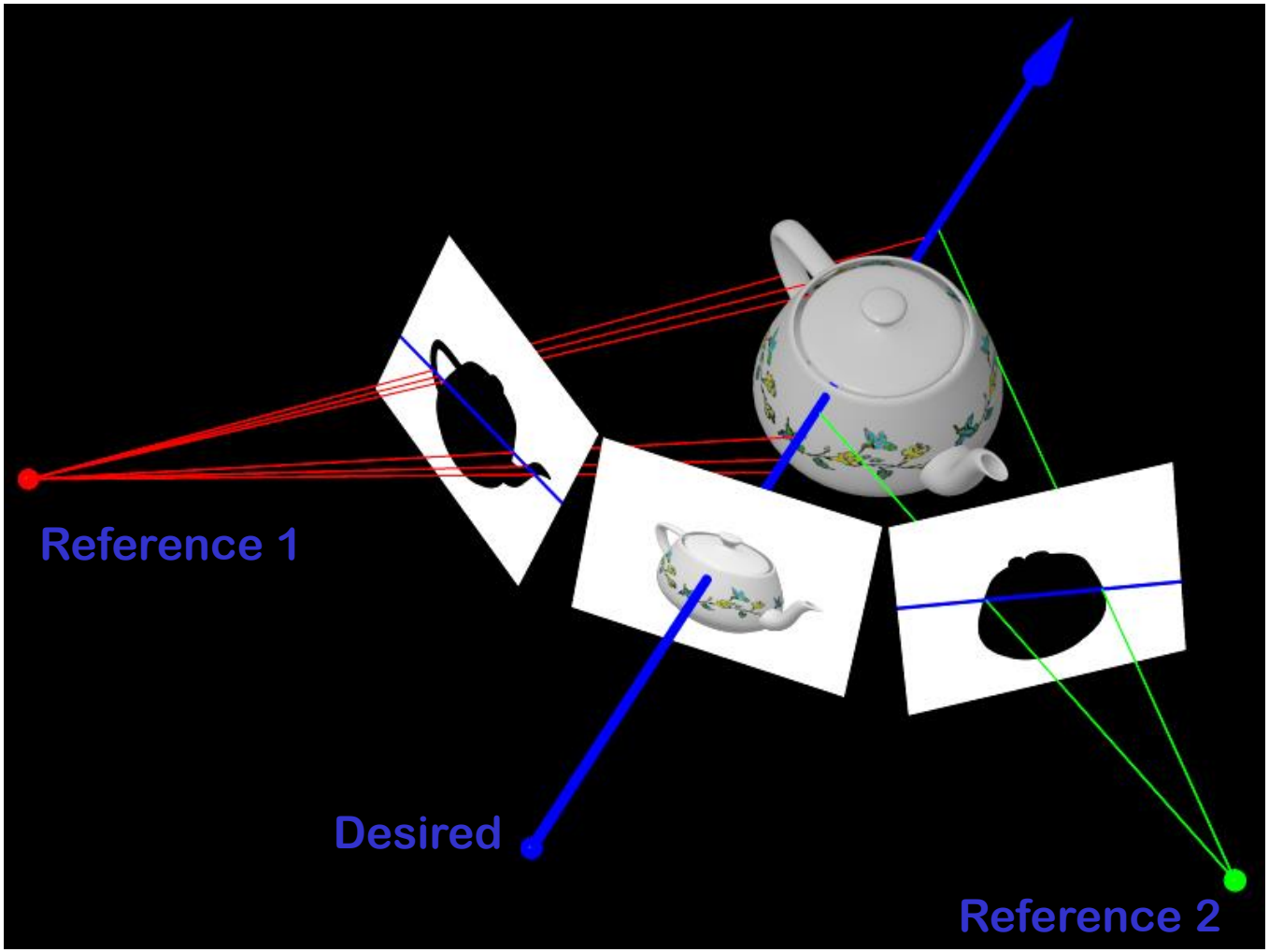
# Image-Based Computation



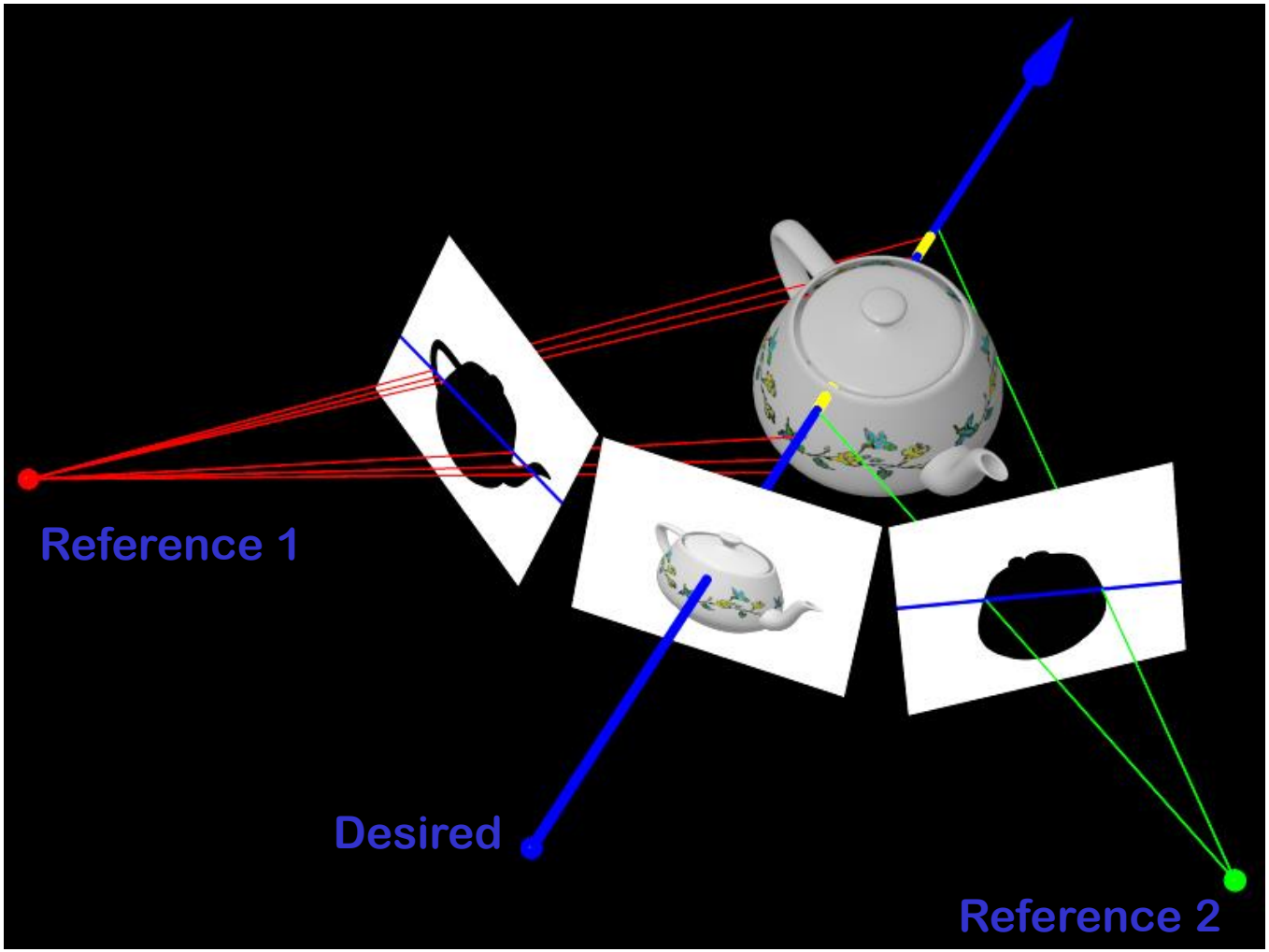
# Image-Based Computation



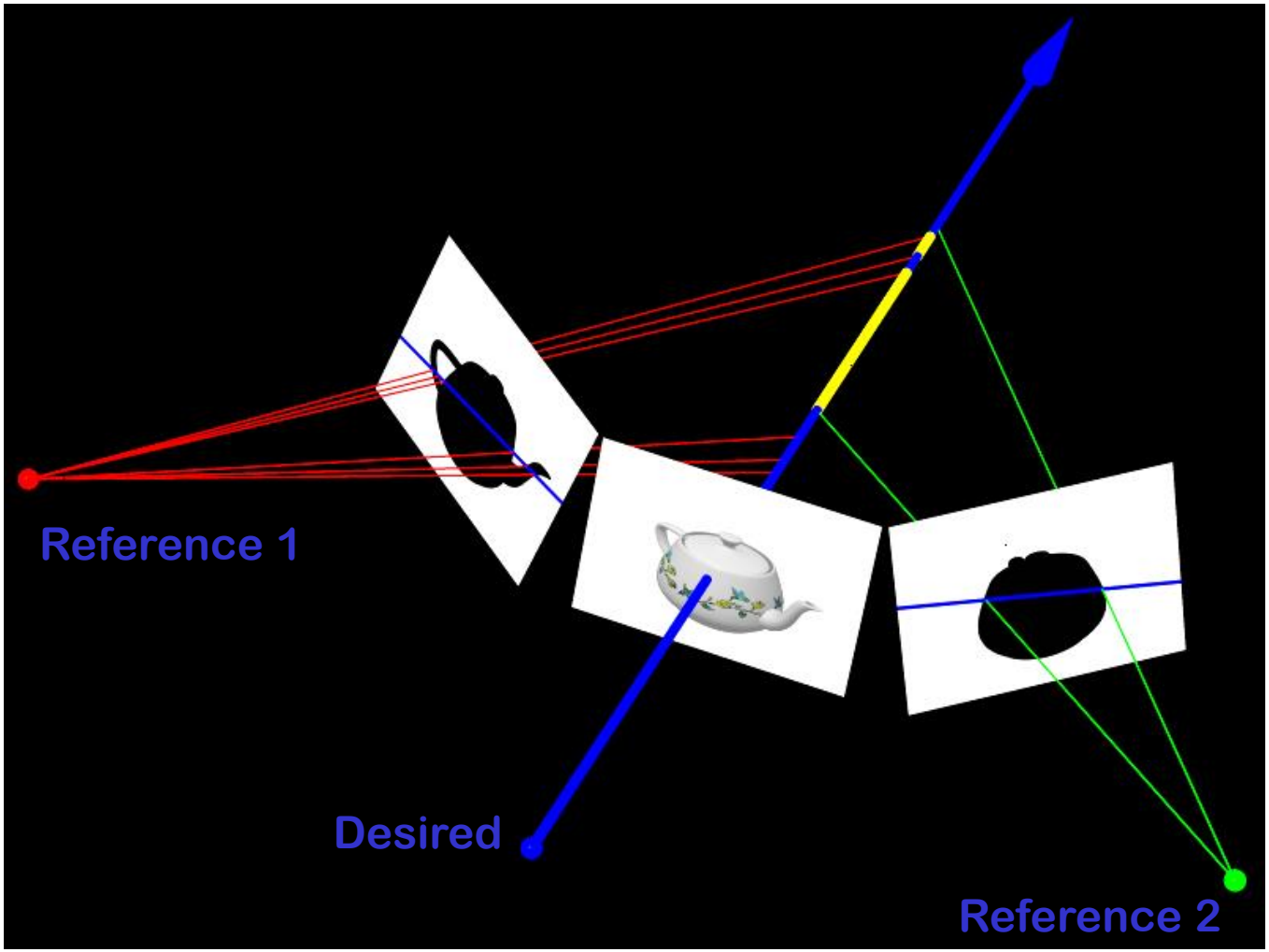
# Image-Based Computation



# Image-Based Computation



# Image-Based Computation



# Image-based Visual Hull

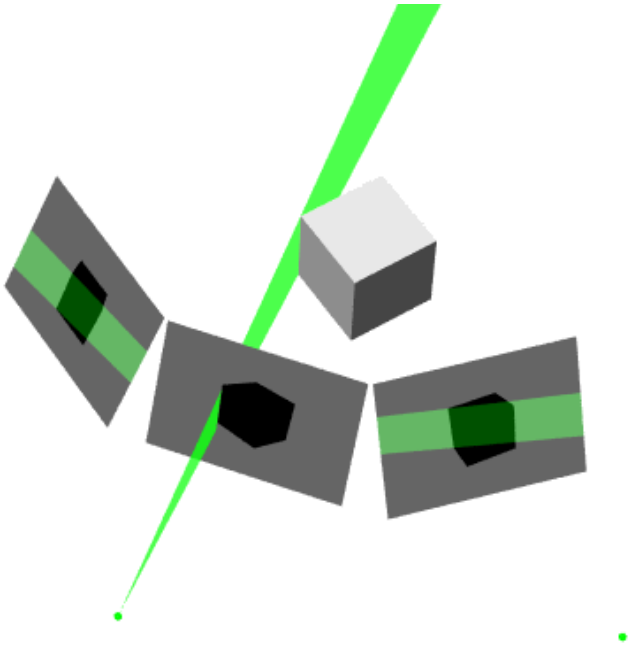
- Approximately constant computation per pixel per camera
- Parallelizes
- Consistent with input silhouettes

# Video - Depth

# Video Shading

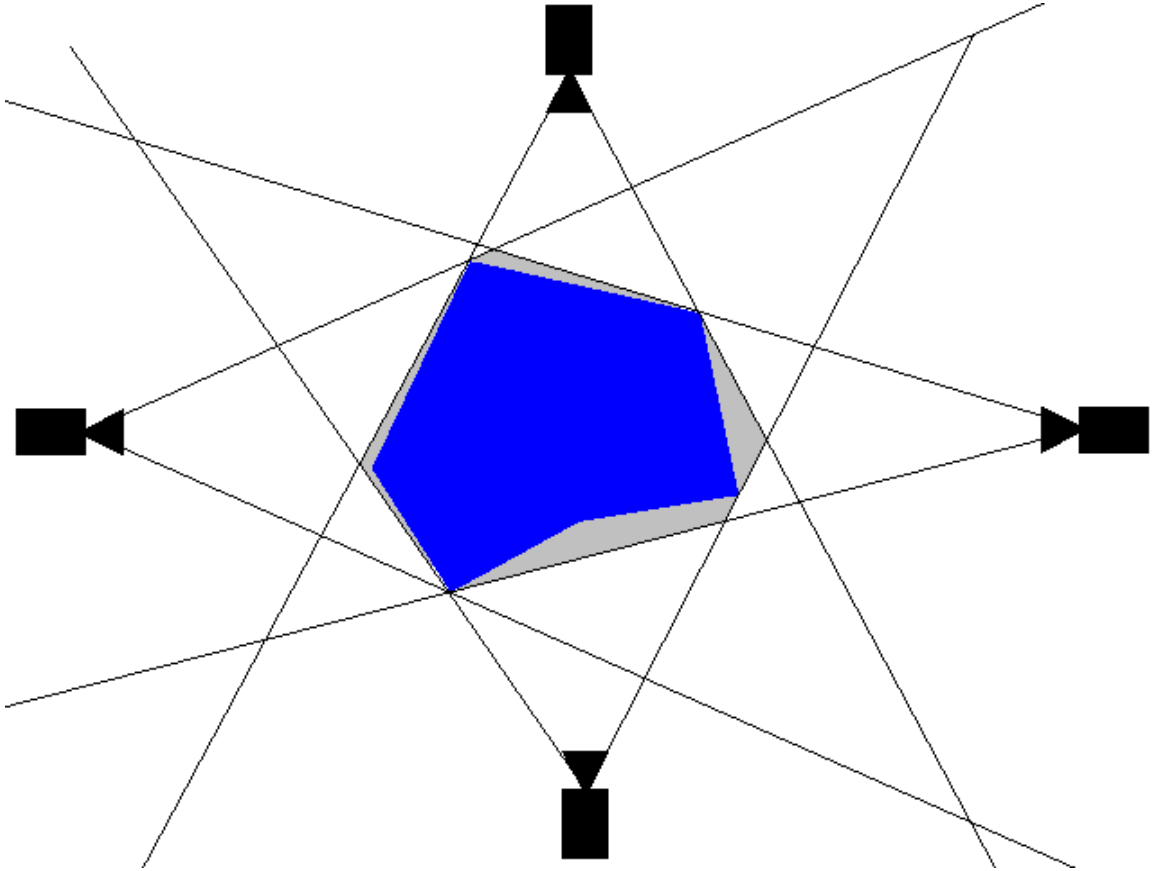
# Exact Polyhedral Methods

- First, silhouette images are converted to polygons (convex or non-convex, with holes allowed)
- Each edge is back projected to form a 3d polygon
- Then each polygon is projected onto each image, and intersected with each silhouette in 2D
- The resulting polygons are assembled to form the polyhedral visual hull



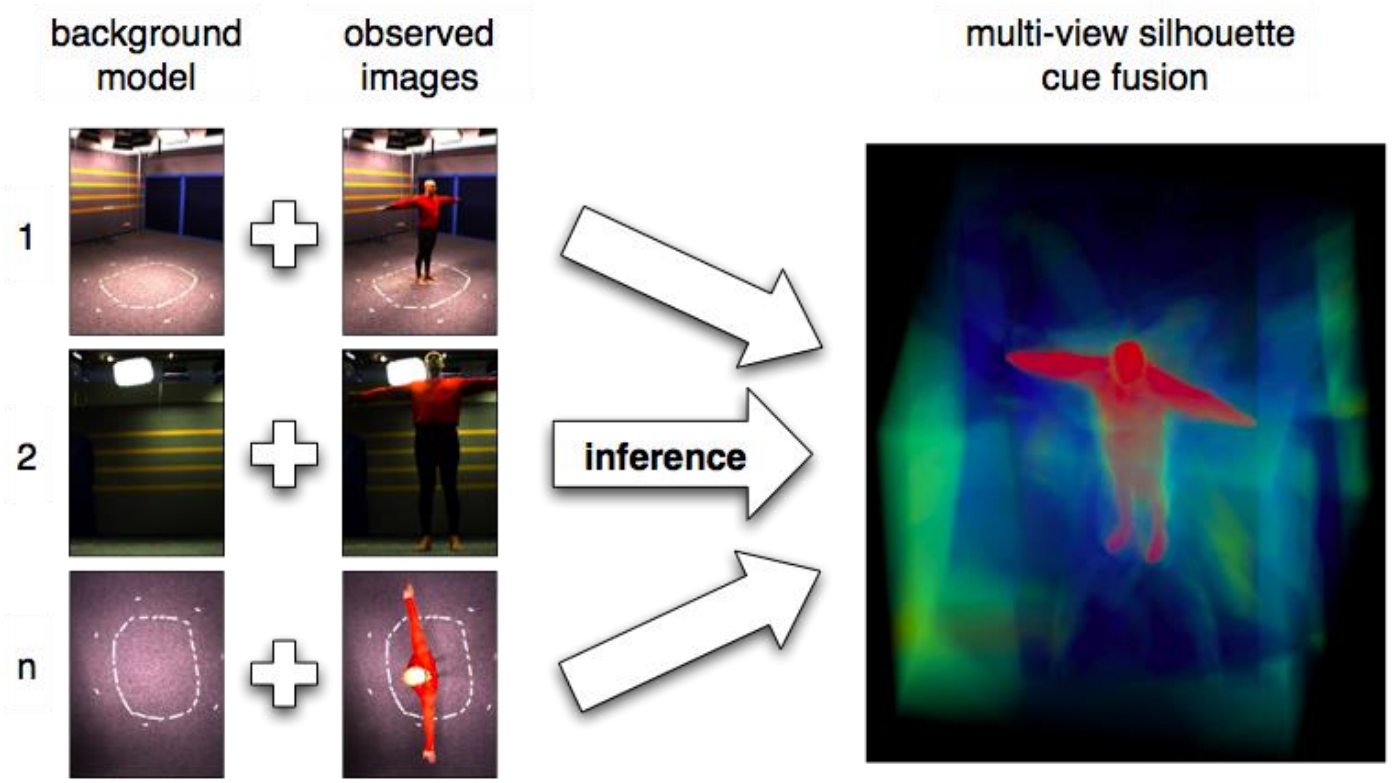
Wojciech Matusik, *An Efficient Visual Hull Computation Algorithm*

# Exact Polyhedral - Example



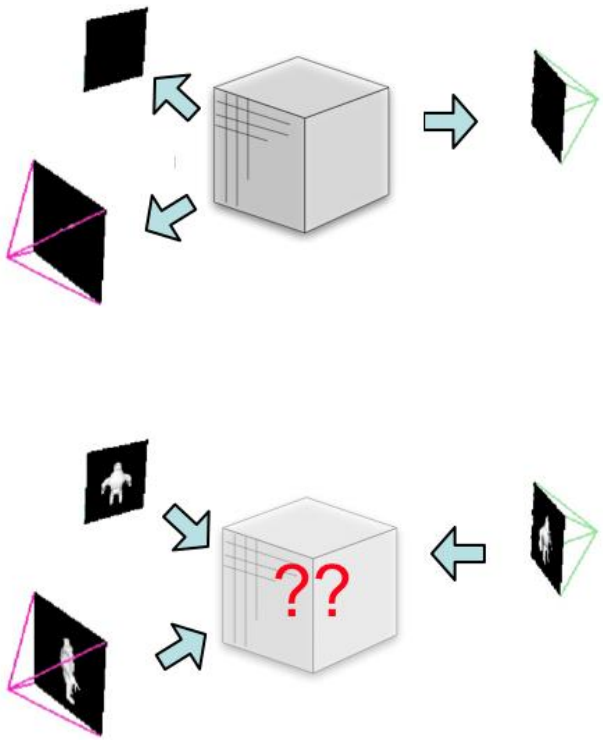
# Bayesian Silhouette Fusion

- Unreliable silhouettes: do not make decision about location
- Do sensor fusion: use all image information simultaneously



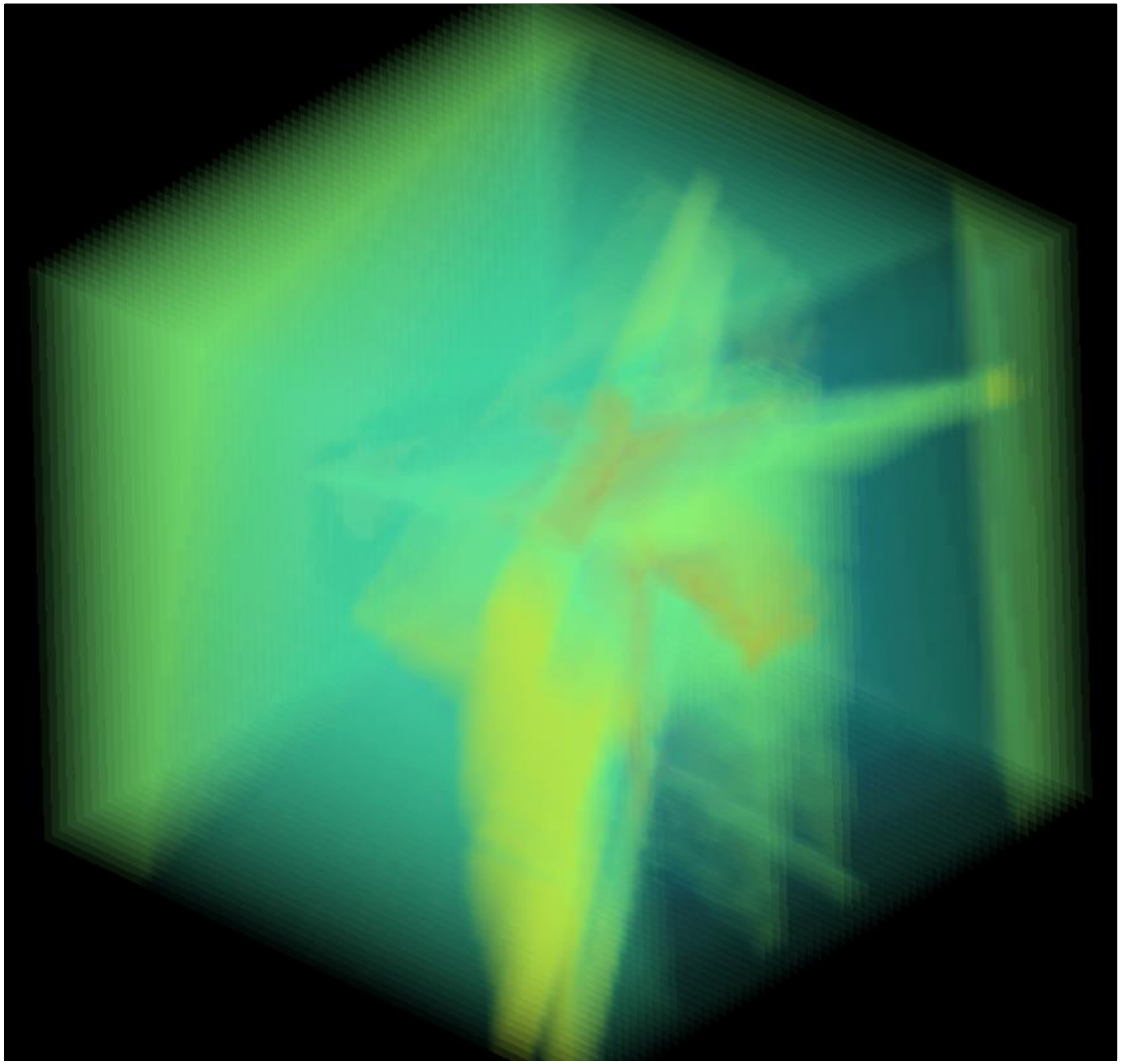

(Franco and Boyer, ICCV05)

# Bayesian Formulation



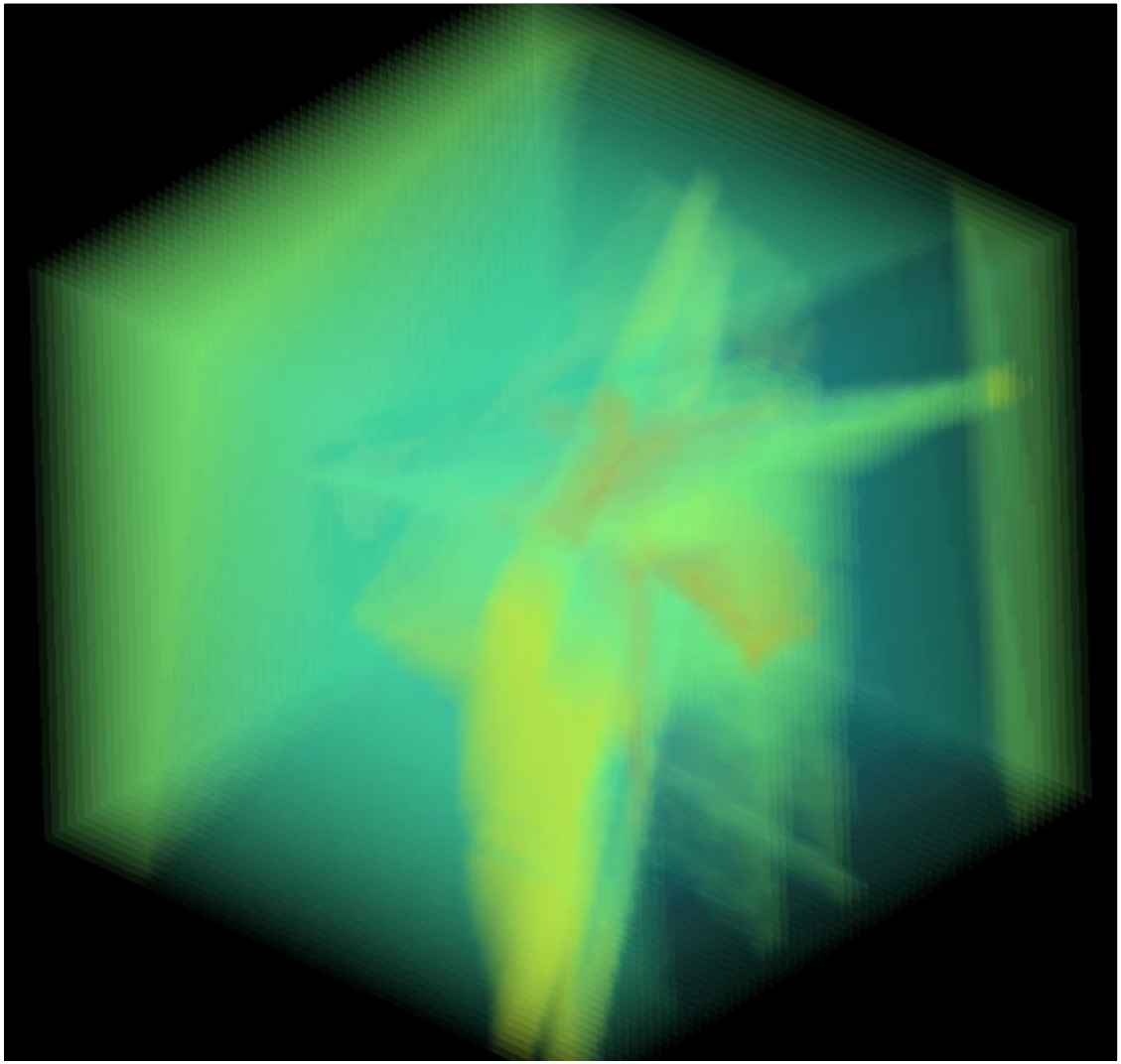

- Idea: we wish to find the content of the scene from images, as a probability grid
- Modeling the forward problem - explaining image observations given the grid state - is easy. It can be accounted for in a *sensor model*.
- Bayesian inference enables the formulation of our initial inverse problem from the sensor model
- Simplification for tractability: independent analysis and processing of voxels

# Visualization



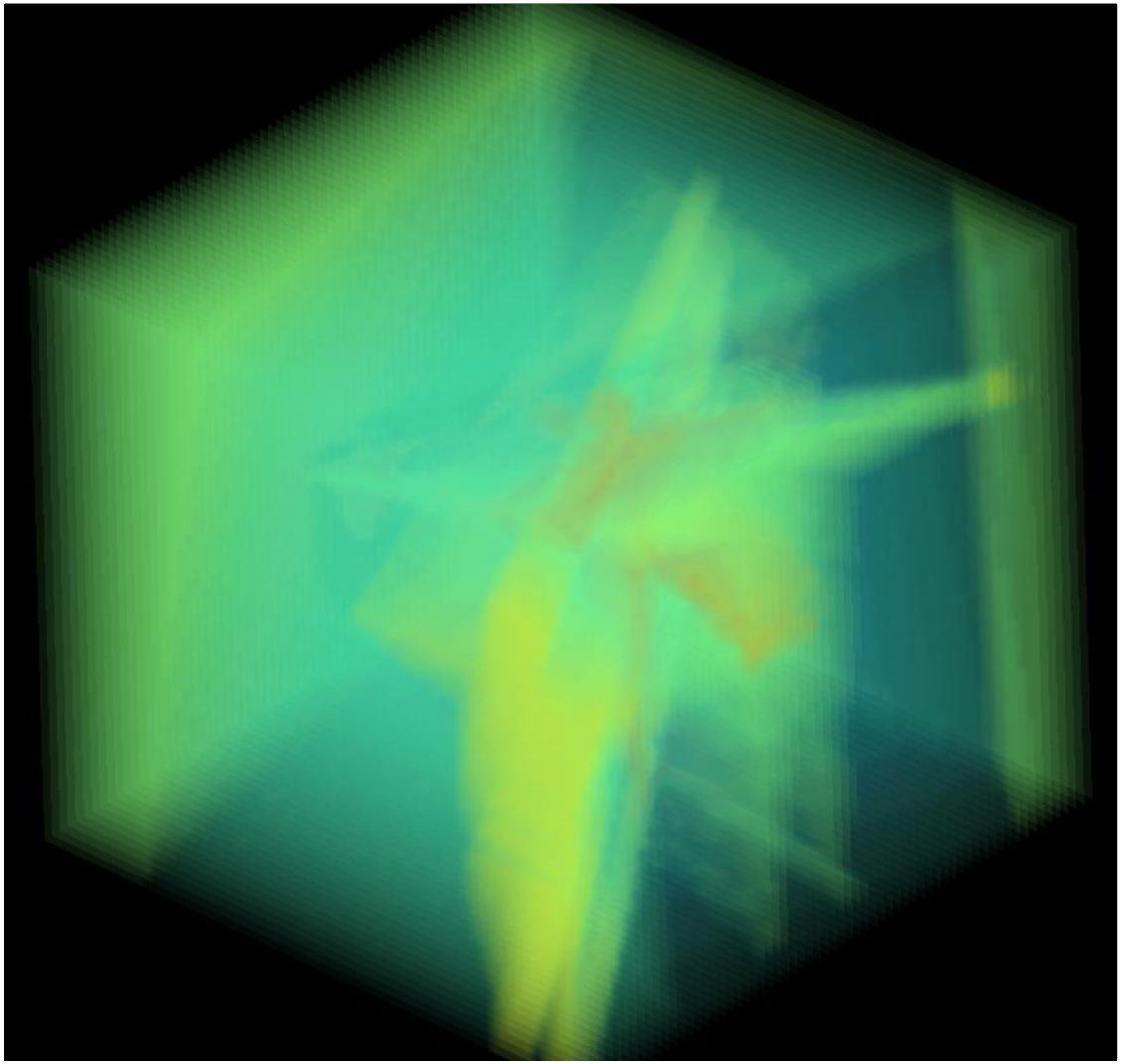
(Franco and Boyer, ICCV05)

# Visualization



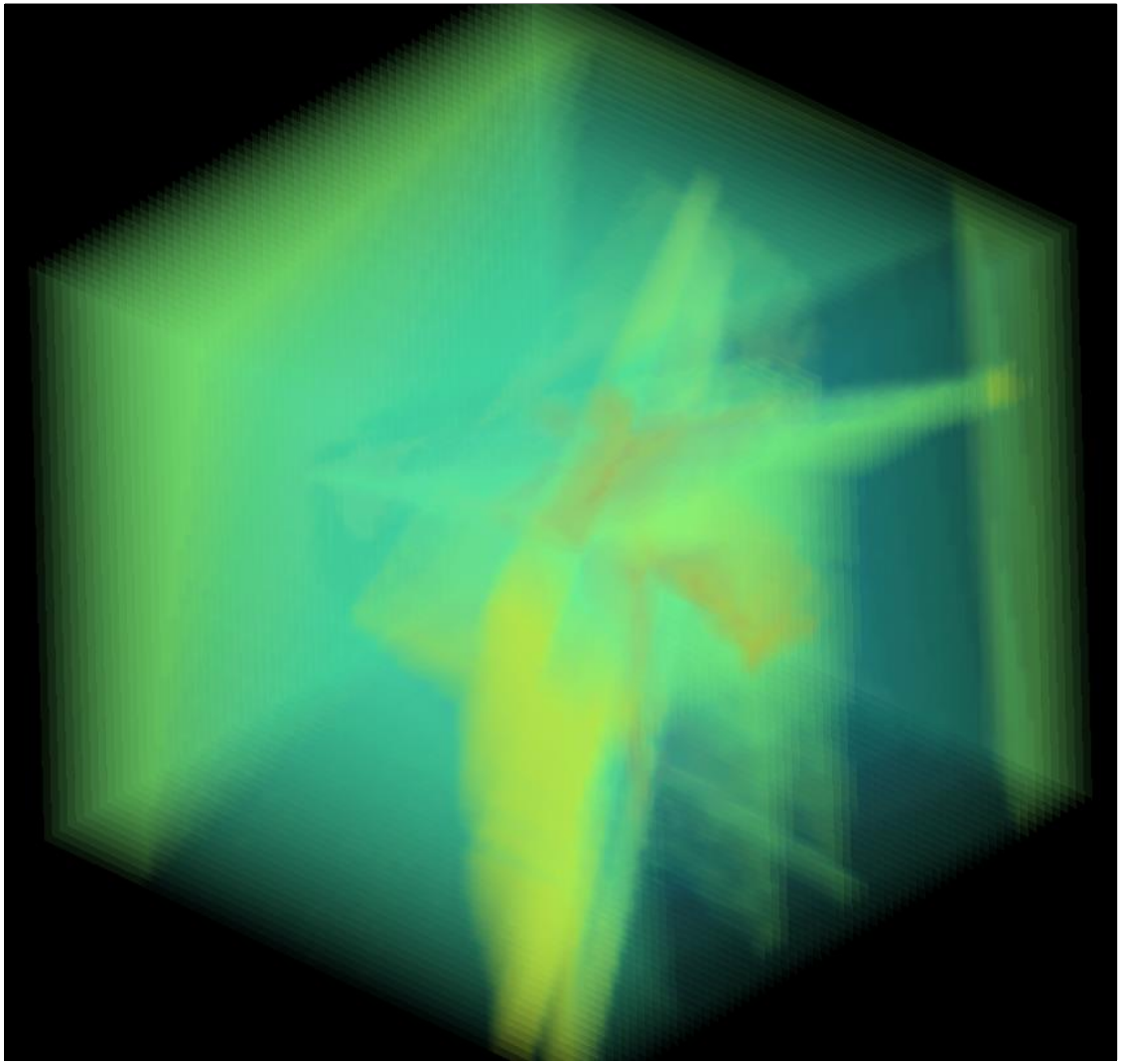
(Franco and Boyer, ICCV05)

# Visualization



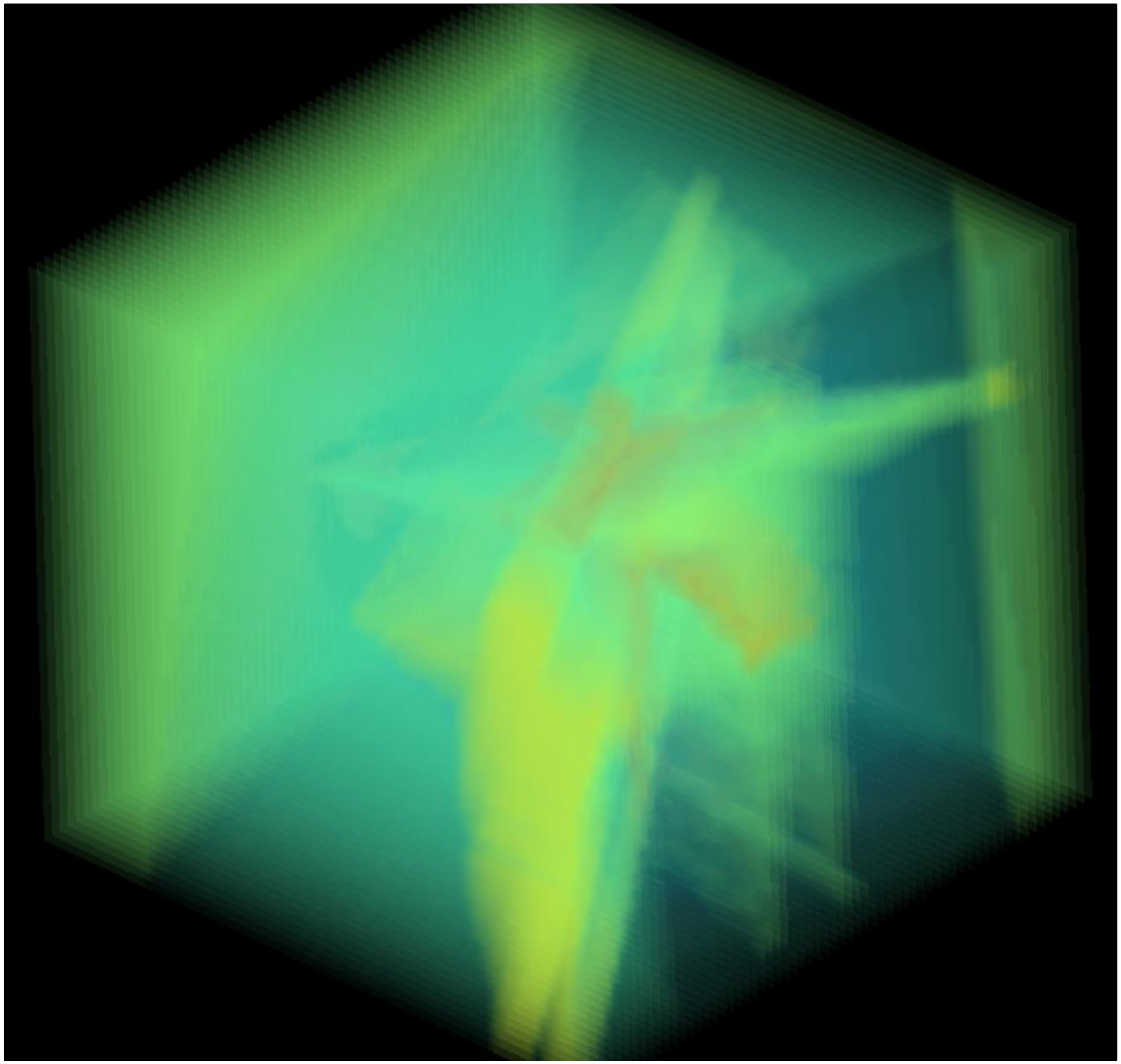
(Franco and Boyer, ICCV05)

# Visualization



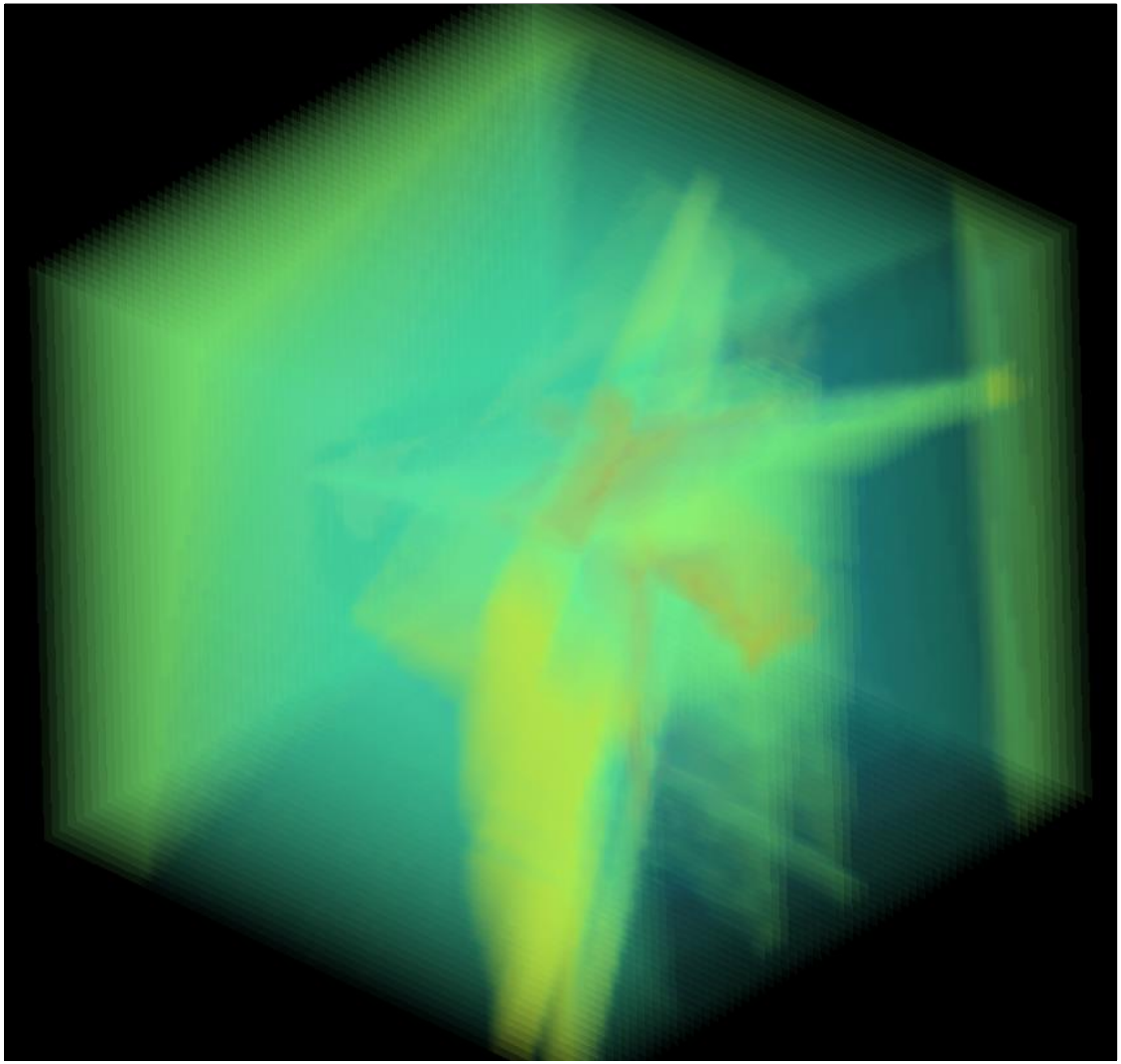
(Franco and Boyer, ICCV05)

# Visualization



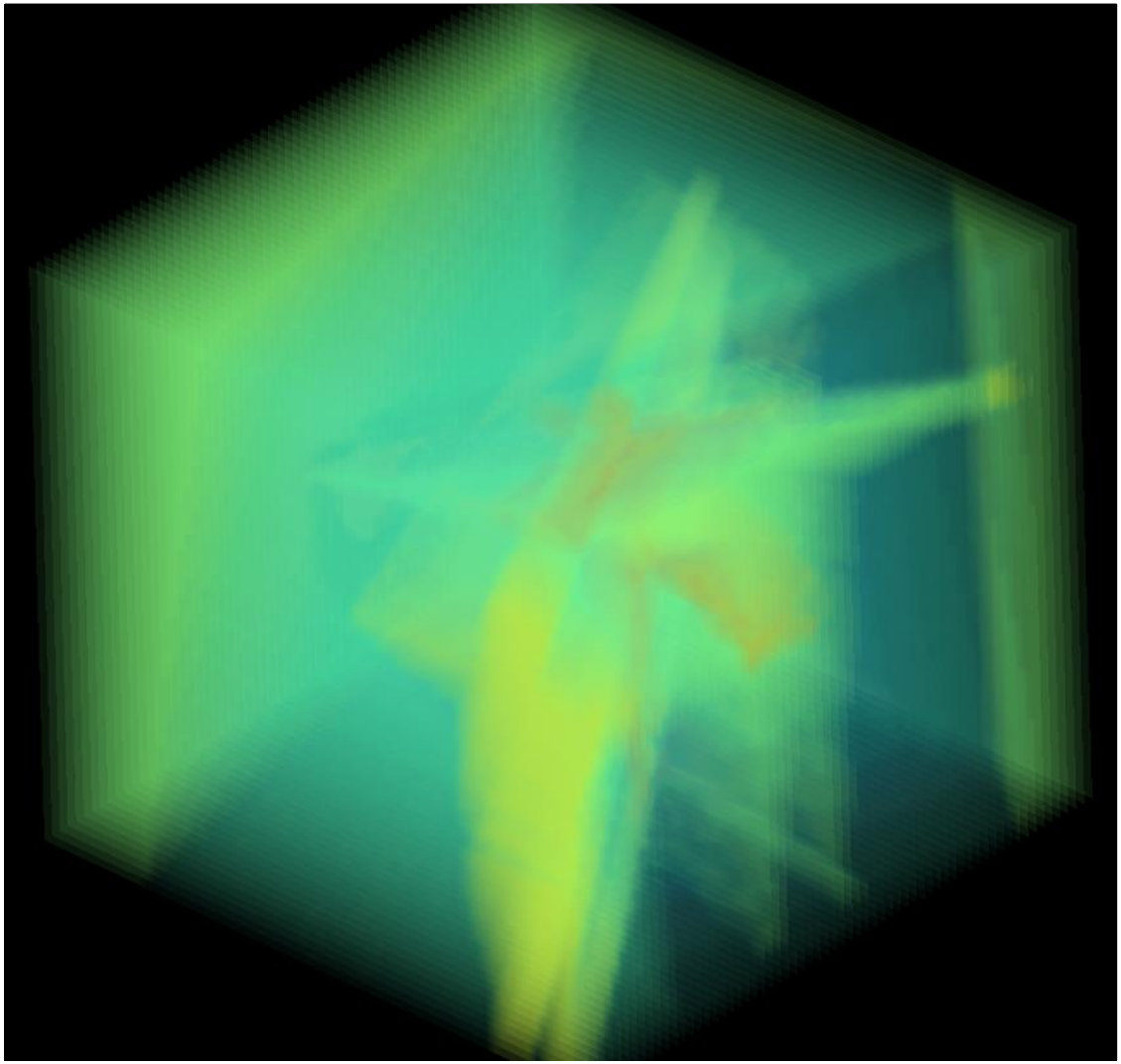
(Franco and Boyer, ICCV05)

# Visualization



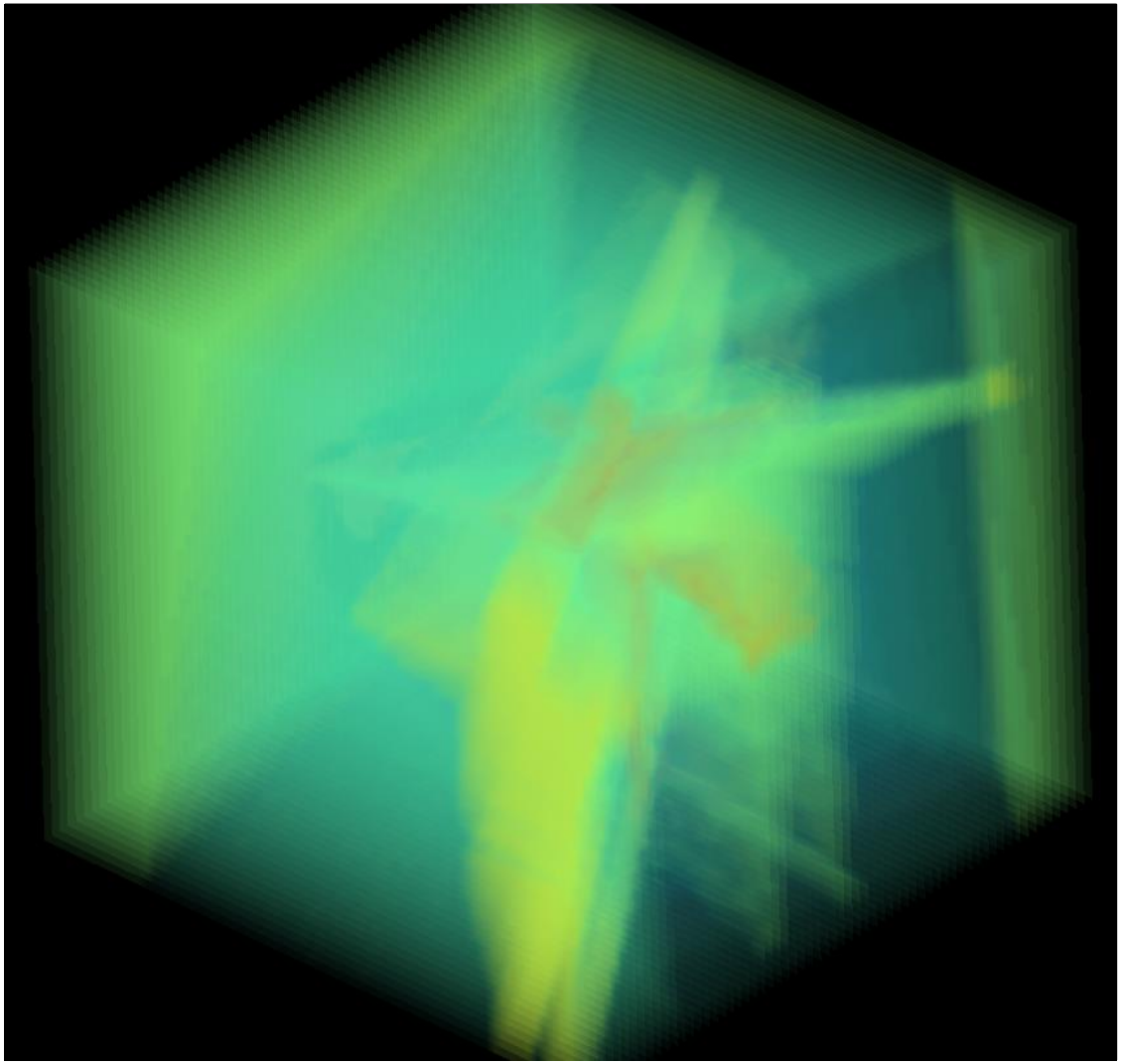
(Franco and Boyer, ICCV05)

# Visualization



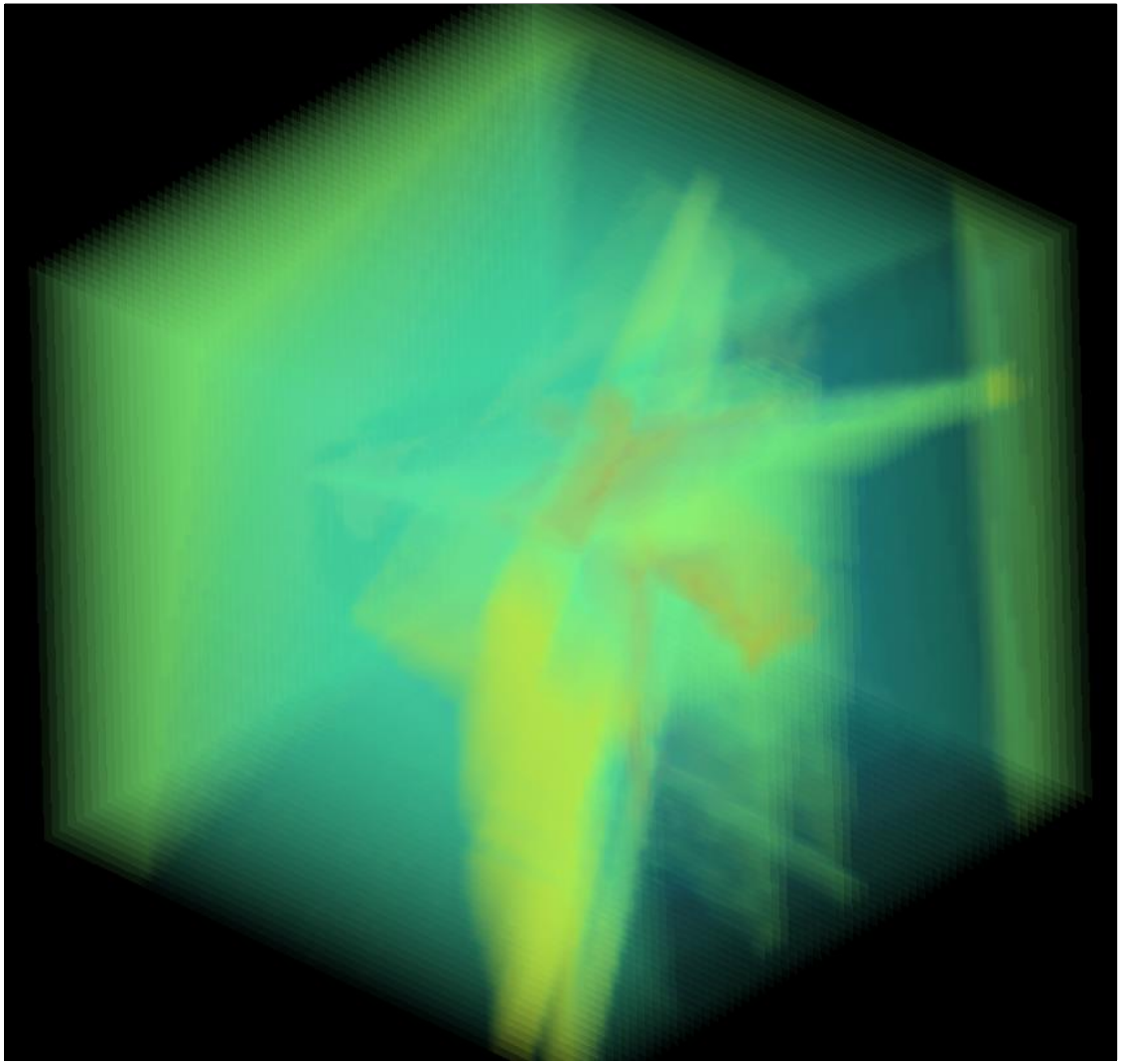
(Franco and Boyer, ICCV05)

# Visualization



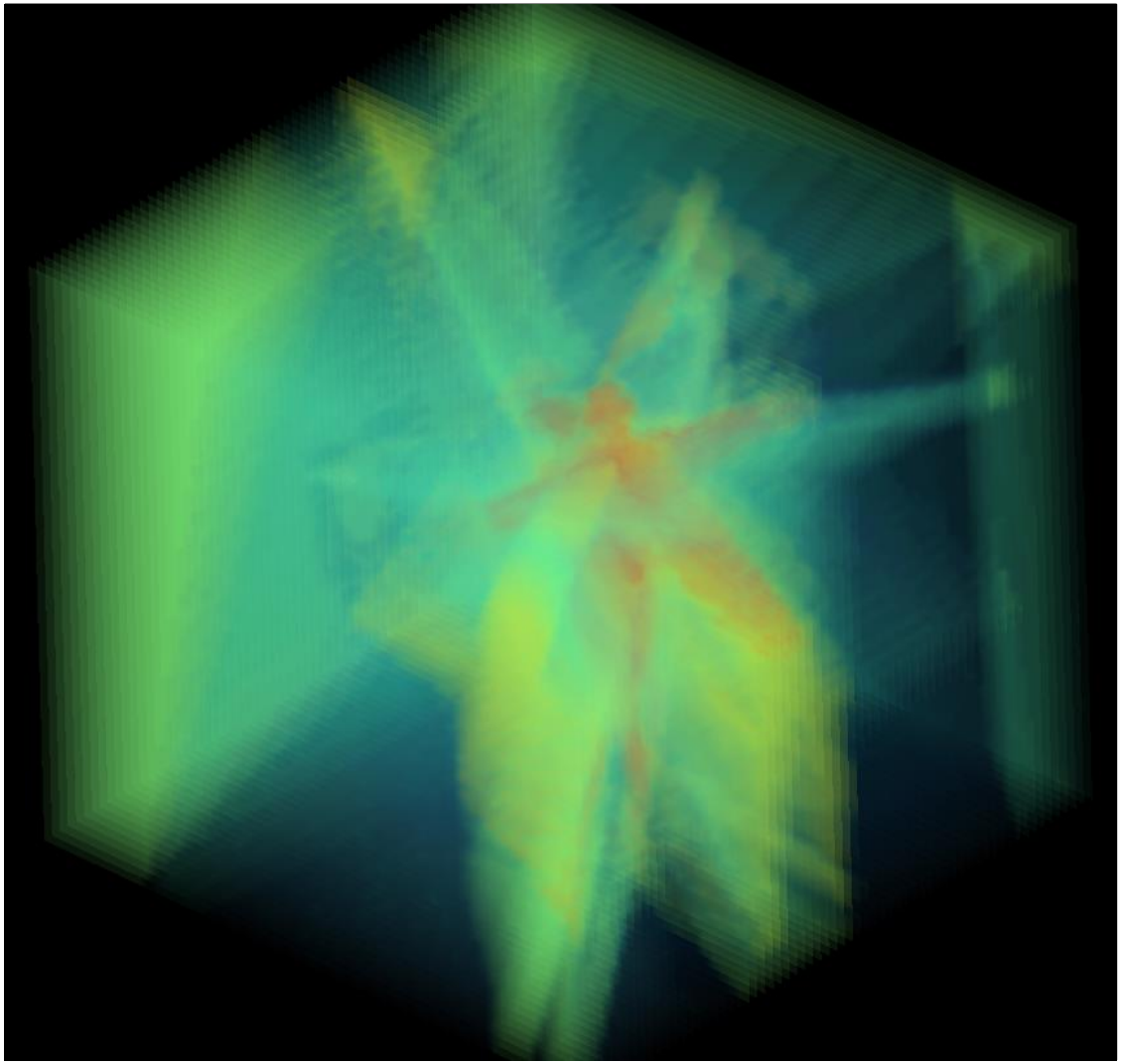
(Franco and Boyer, ICCV05)

# Visualization



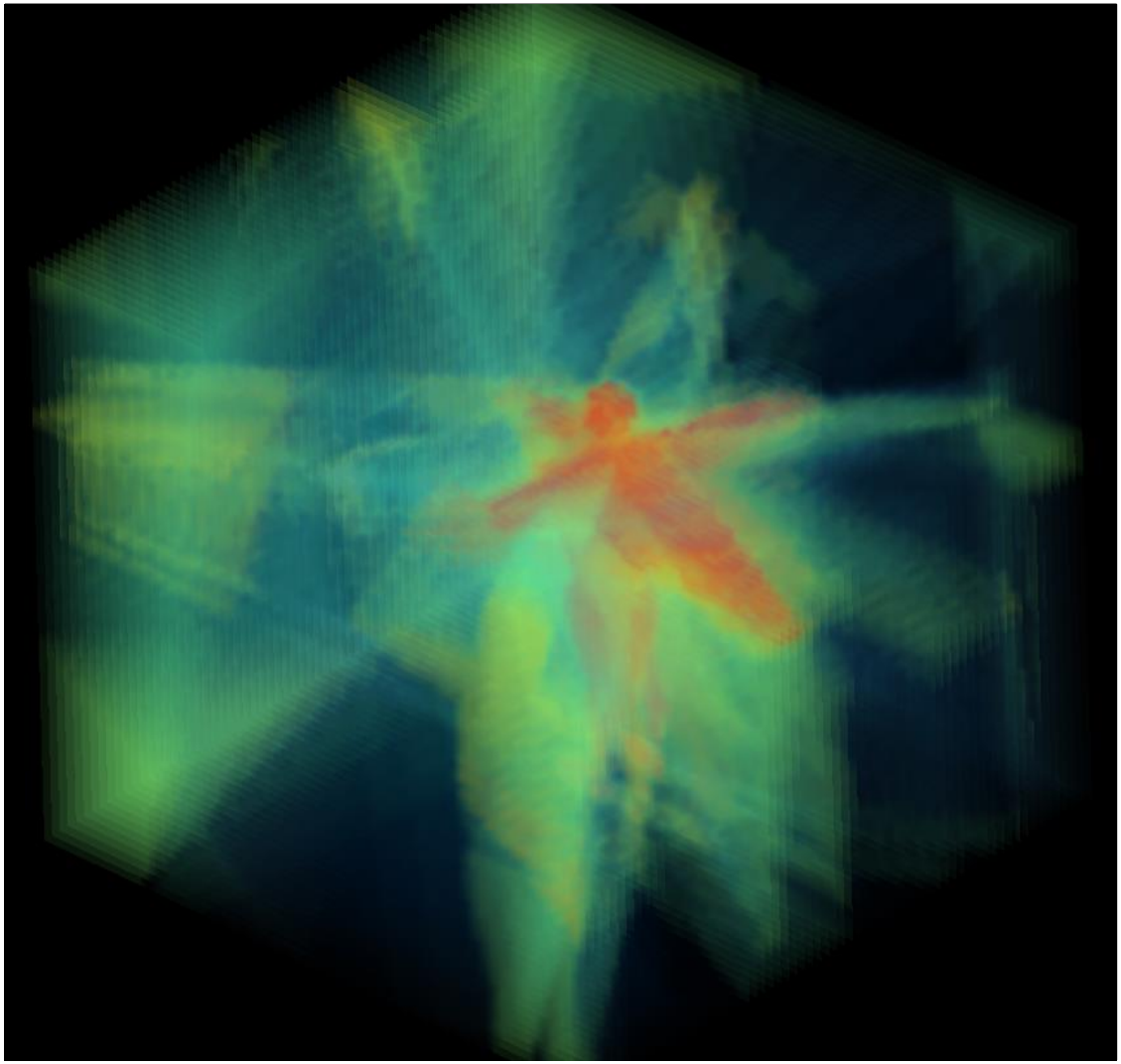
(Franco and Boyer, ICCV05)

# Visualization



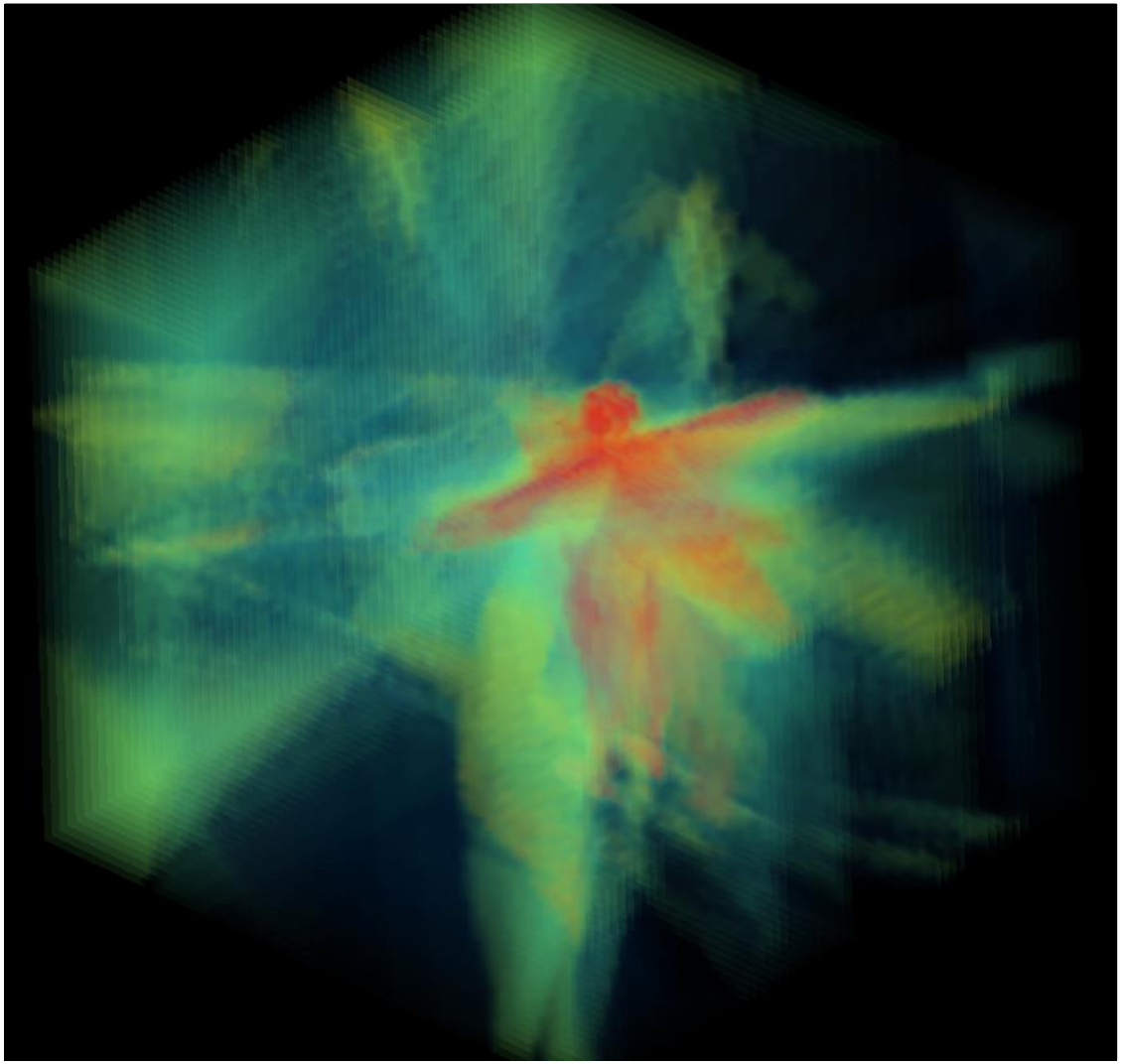
(Franco and Boyer, ICCV05)

# Visualization



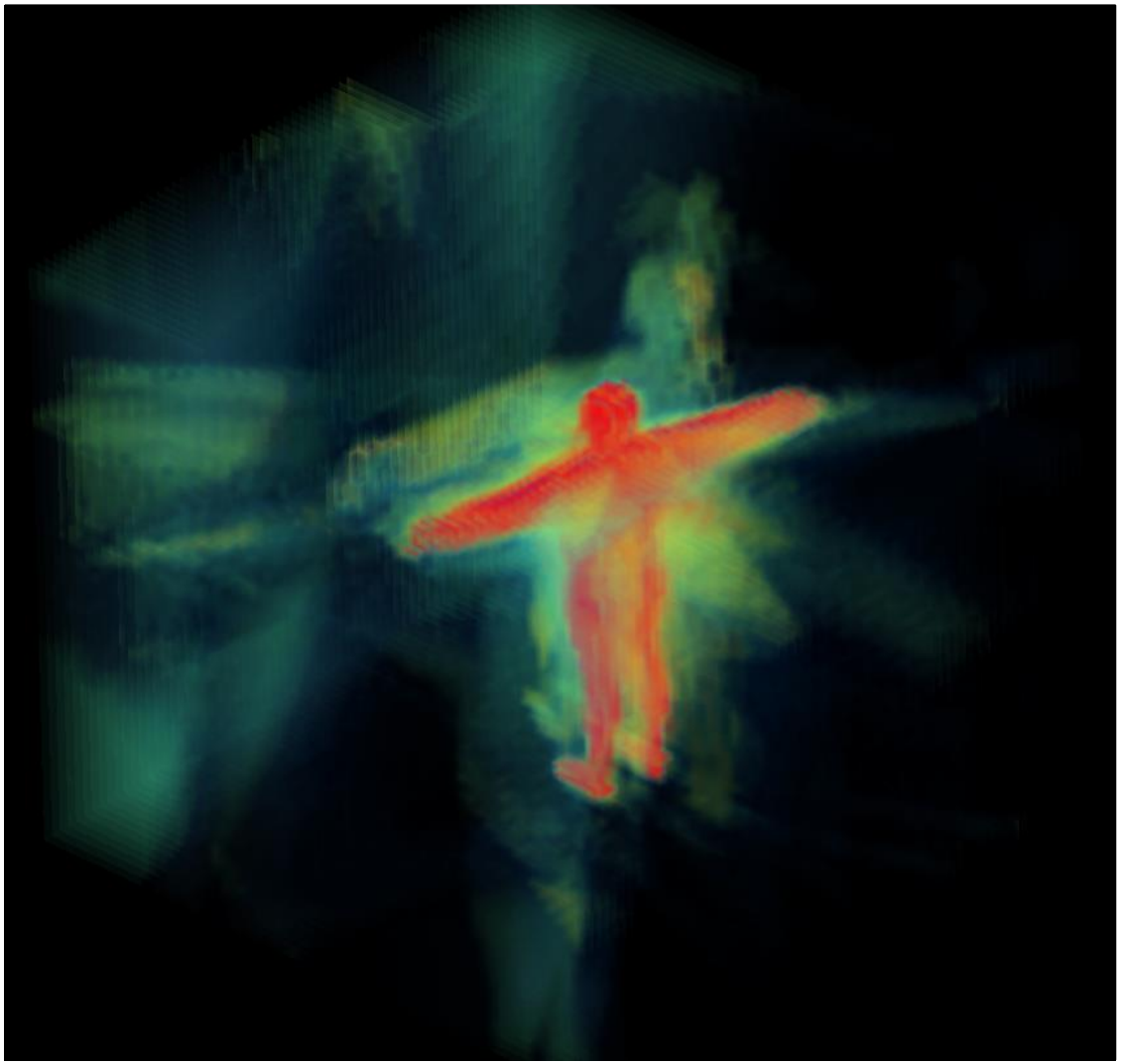
(Franco and Boyer, ICCV05)

# Visualization



(Franco and Boyer, ICCV05)

# Visualization

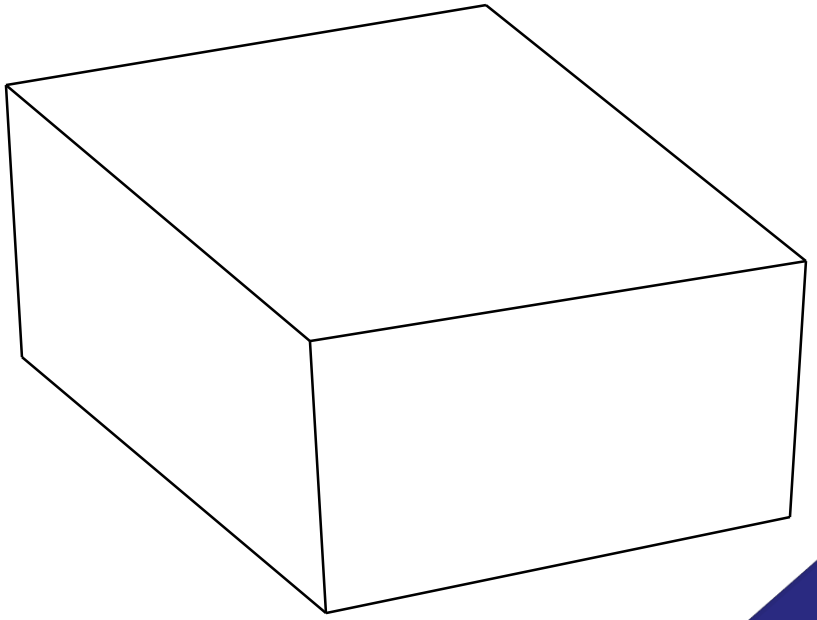


(Franco and Boyer, ICCV05)

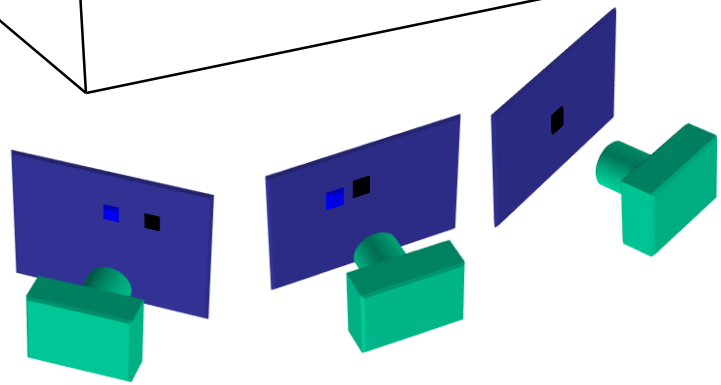
# Voxel Coloring

# 3D Reconstruction from Calibrated Images

Scene Volume
**V**



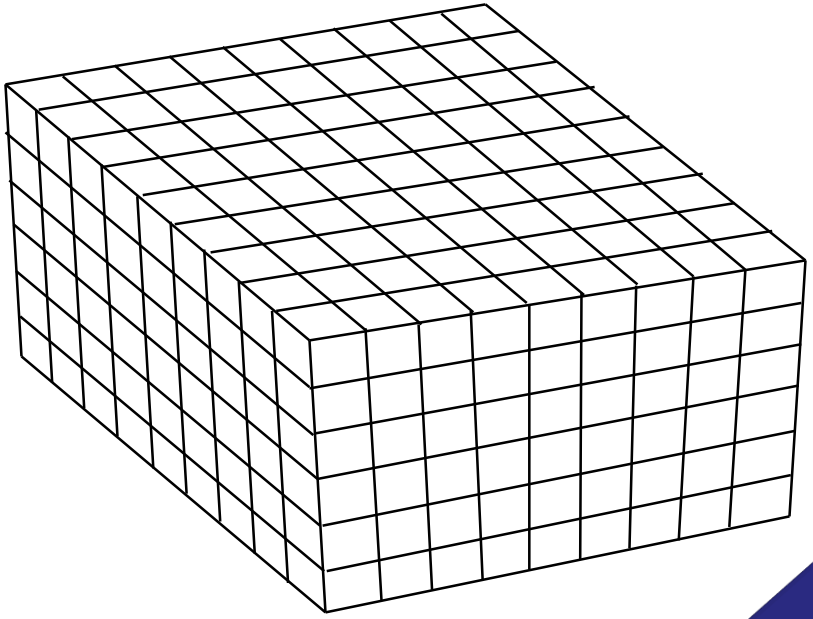
Input Images
(Calibrated)



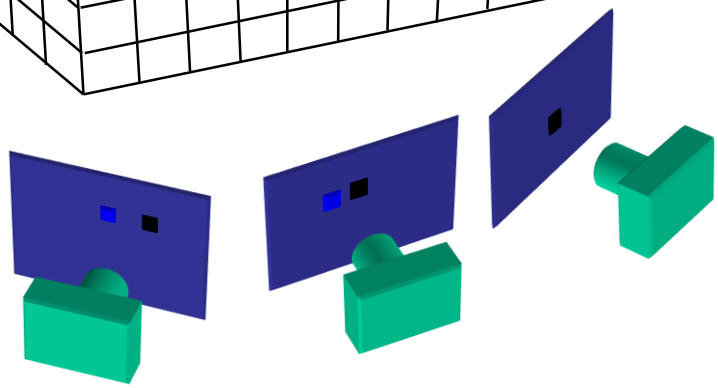
Goal: Determine transparency, radiance of points in V

# Discrete Formulation: Voxel Coloring

Discretized Scene Volume



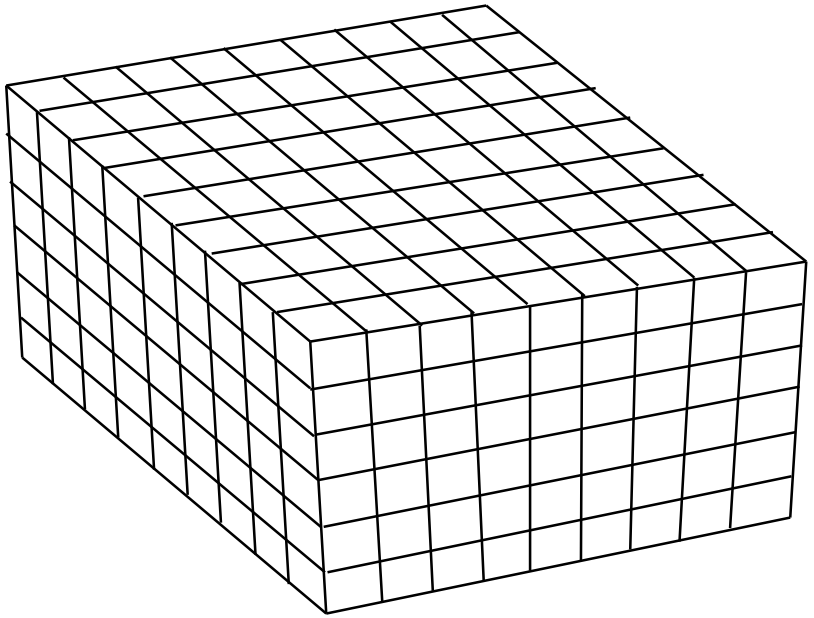
Input Images (Calibrated)

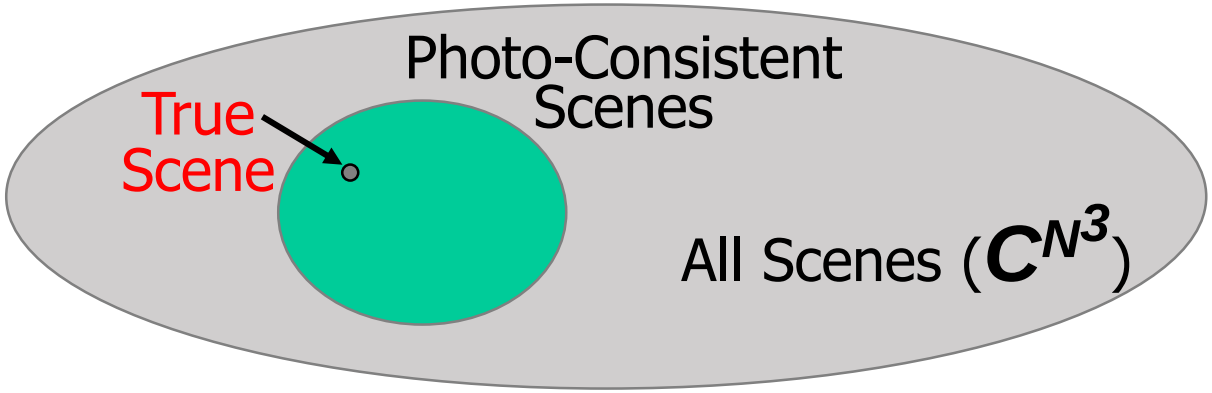


Goal: Assign RGBA values to voxels in V *photo-consistent* with images

# Complexity and Computability

Discretized
Scene Volume
$N^3$ voxels
$C$ colors





# Issues

## Theoretical Questions

Identify class of *all* photo-consistent scenes

## Practical Questions

How do we compute photo-consistent models?

# Voxel Coloring Solutions

1. C=2 (silhouettes)

   Volume intersection [Martin 81, Szeliski 93]

2. C unconstrained, viewpoint constraints

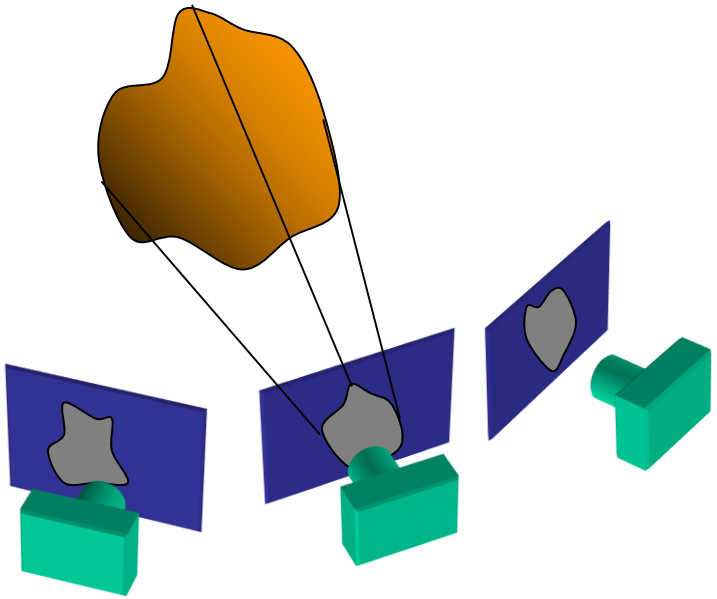
   Voxel coloring algorithm [Seitz & Dyer 97]

3. General Case

   Space carving [Kutulakos & Seitz 98]

# Reconstruction from Silhouettes (C = 2)

Binary Images →

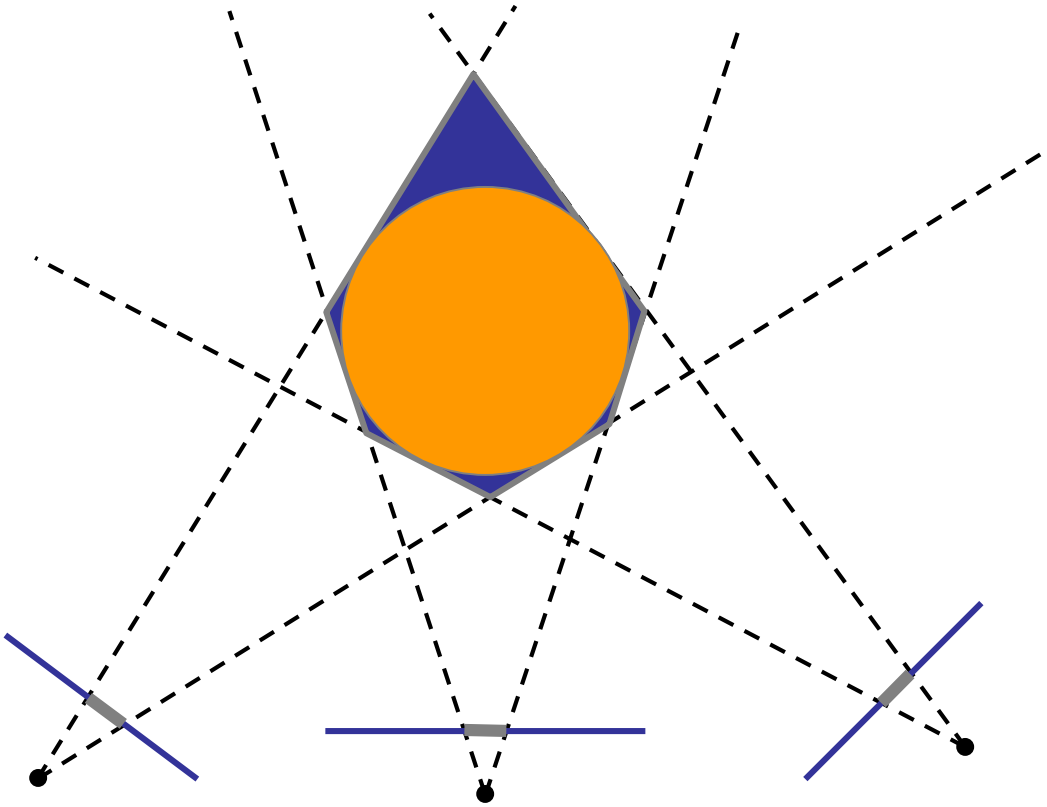


**Approach:**

1) Backproject each silhouette

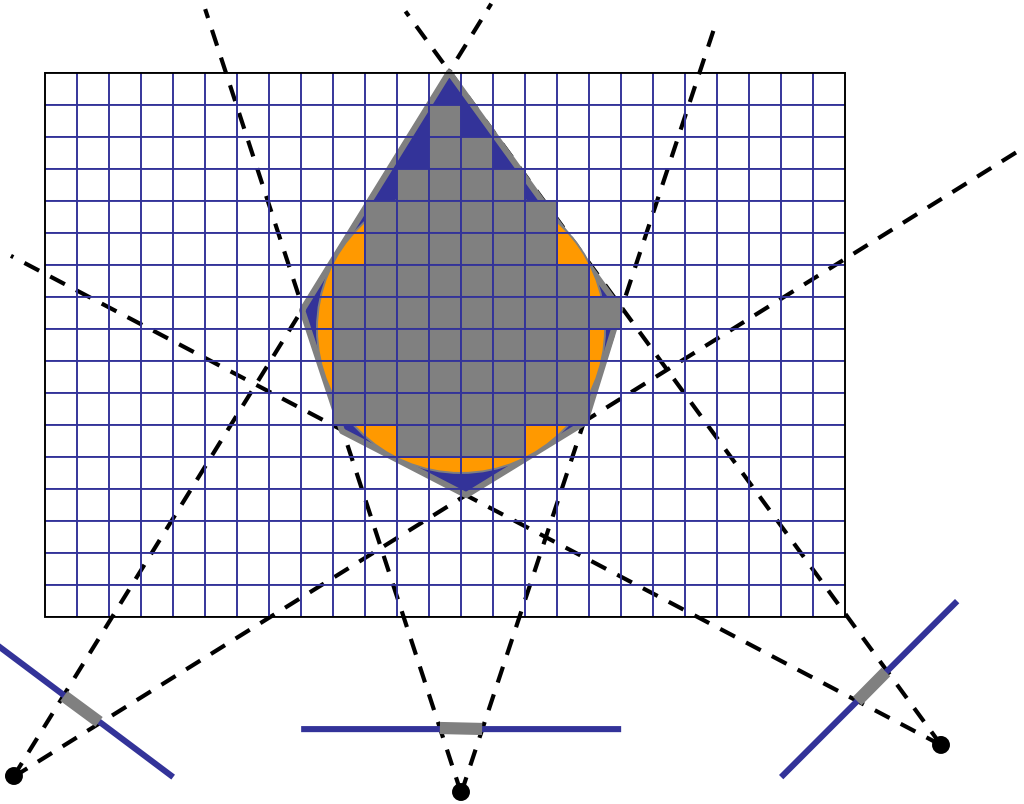
2) Intersect backprojected volumes

# Volume Intersection



- Reconstruction contains the true Scene
- But is generally not the same
- In the limit get *visual hull*
- Complement of all lines that don't intersect S

# Volume Intersection



- Color voxel black if on silhouette in every image
  - $O(MN^3)$, for M images, $N^3$ voxels
  - Don't have to search $2^{N^3}$ possible scenes!

# Properties of Volume Intersection

Pros

- Easy to implement, fast
- Accelerated via octrees [Szeliski 1993]

Cons

- No concavities
- Reconstruction is not photo-consistent
- Requires identification of silhouettes

# Voxel Coloring Solutions

1. C=2 (silhouettes)

   Volume intersection [Martin 81, Szeliski 93]

2. C unconstrained, viewpoint constraints

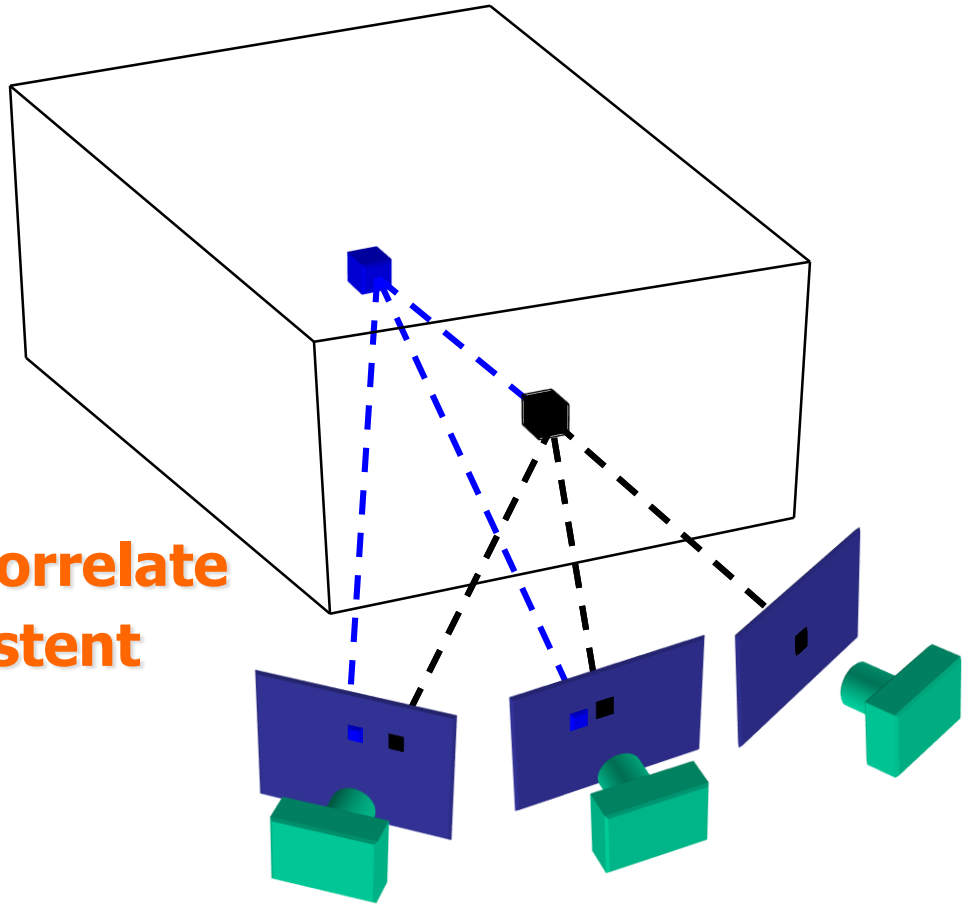
   Voxel coloring algorithm [Seitz & Dyer 97]

3. General Case

   Space carving [Kutulakos & Seitz 98]

# Voxel Coloring Approach

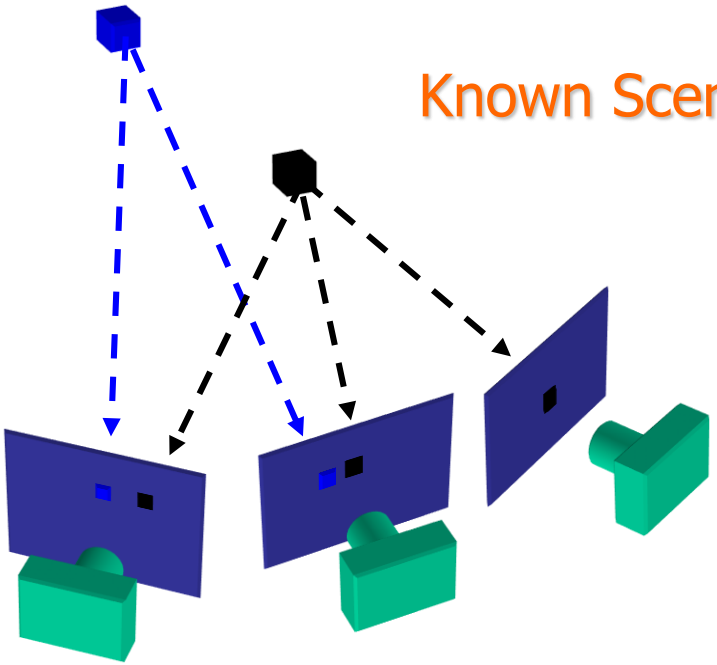
1. **Choose voxel**
2. **Project and correlate**
3. **Color if consistent**



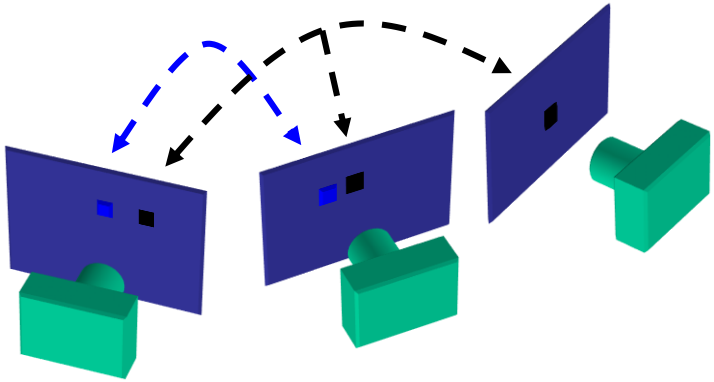
Visibility Problem: In which images is each voxel visible?

# The Global Visibility Problem

Which points are visible in which images?



Known Scene



Unknown Scene

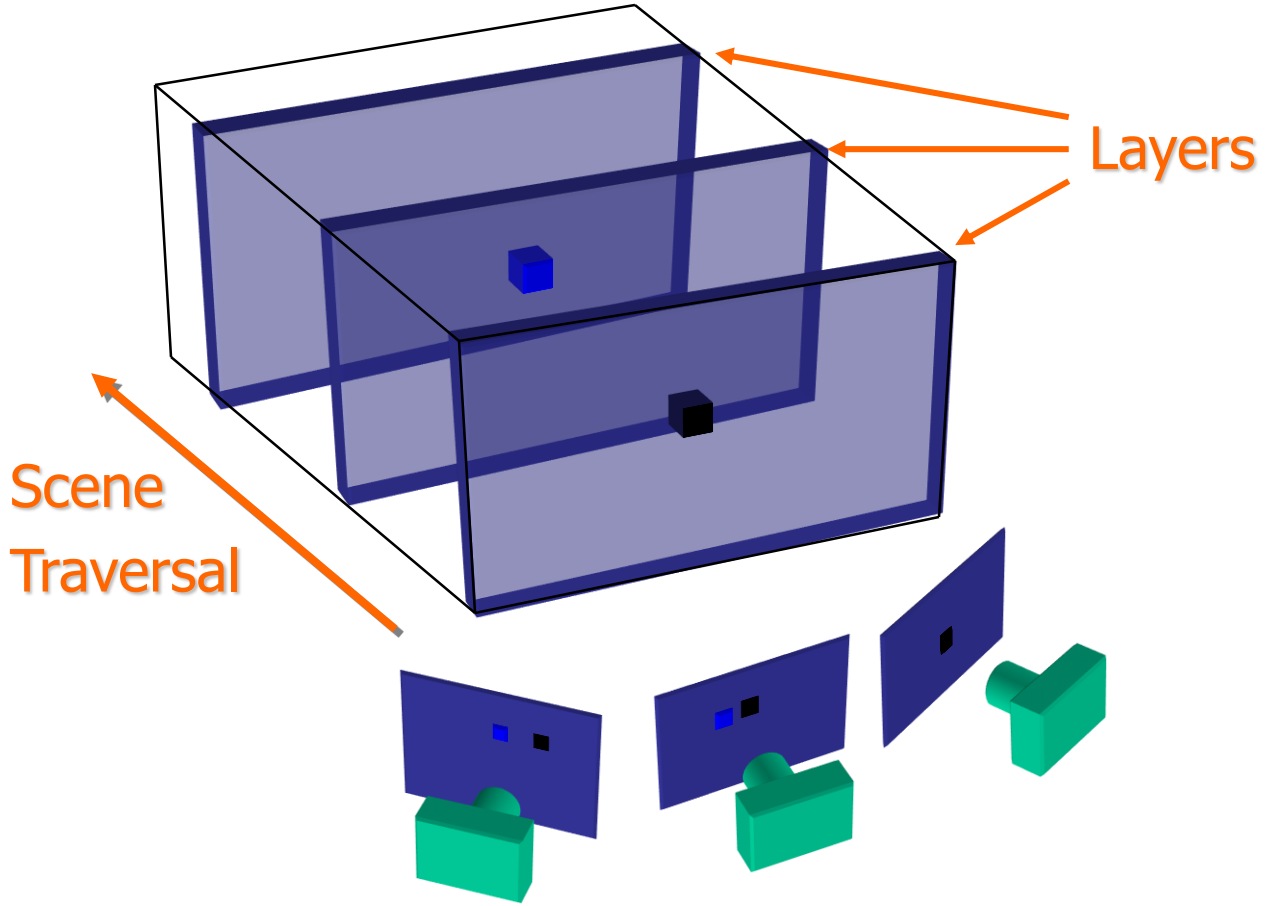
Forward Visibility

known scene

Inverse Visibility

known images

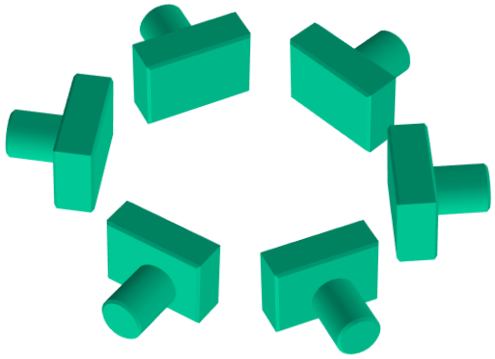
# Depth Ordering: Visit Occluders first!



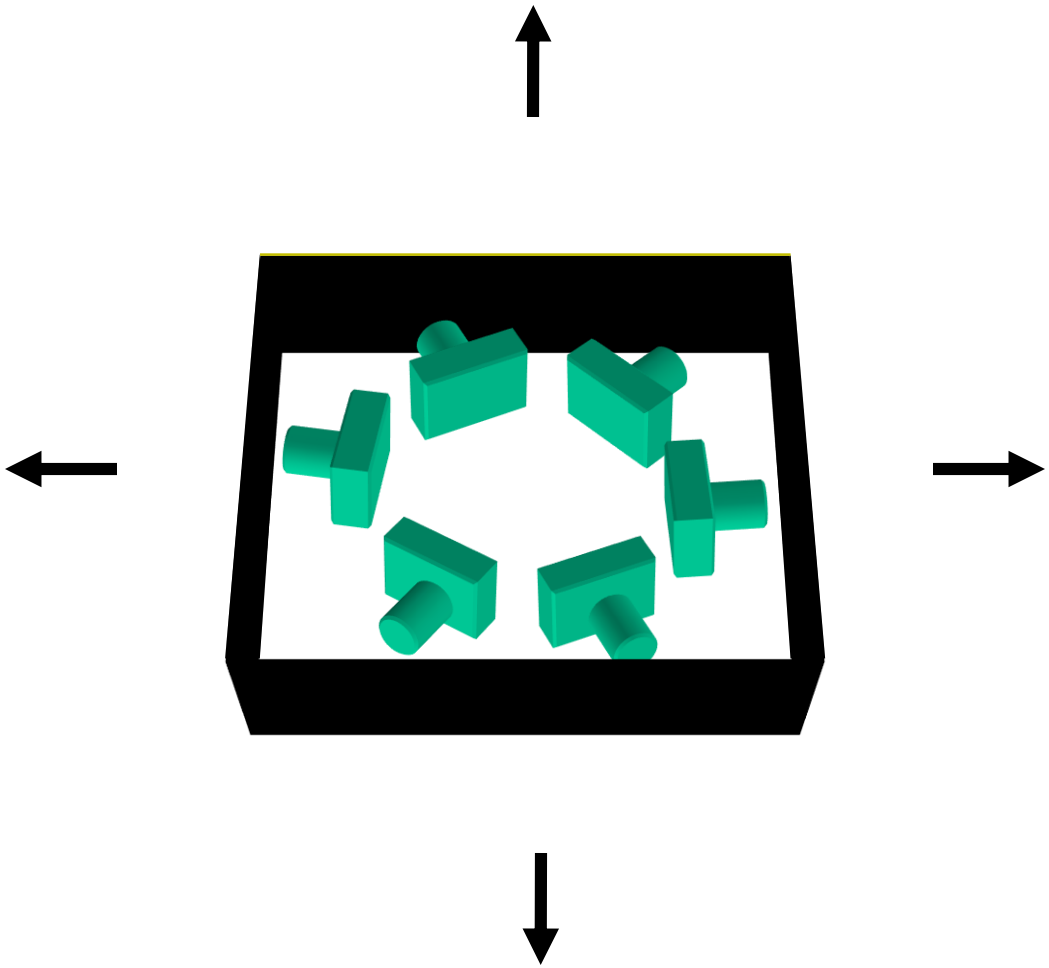

Condition: depth order is *view-independent*

# Panoramic Depth Ordering

- Cameras oriented in many different directions
- Planar depth ordering does not apply

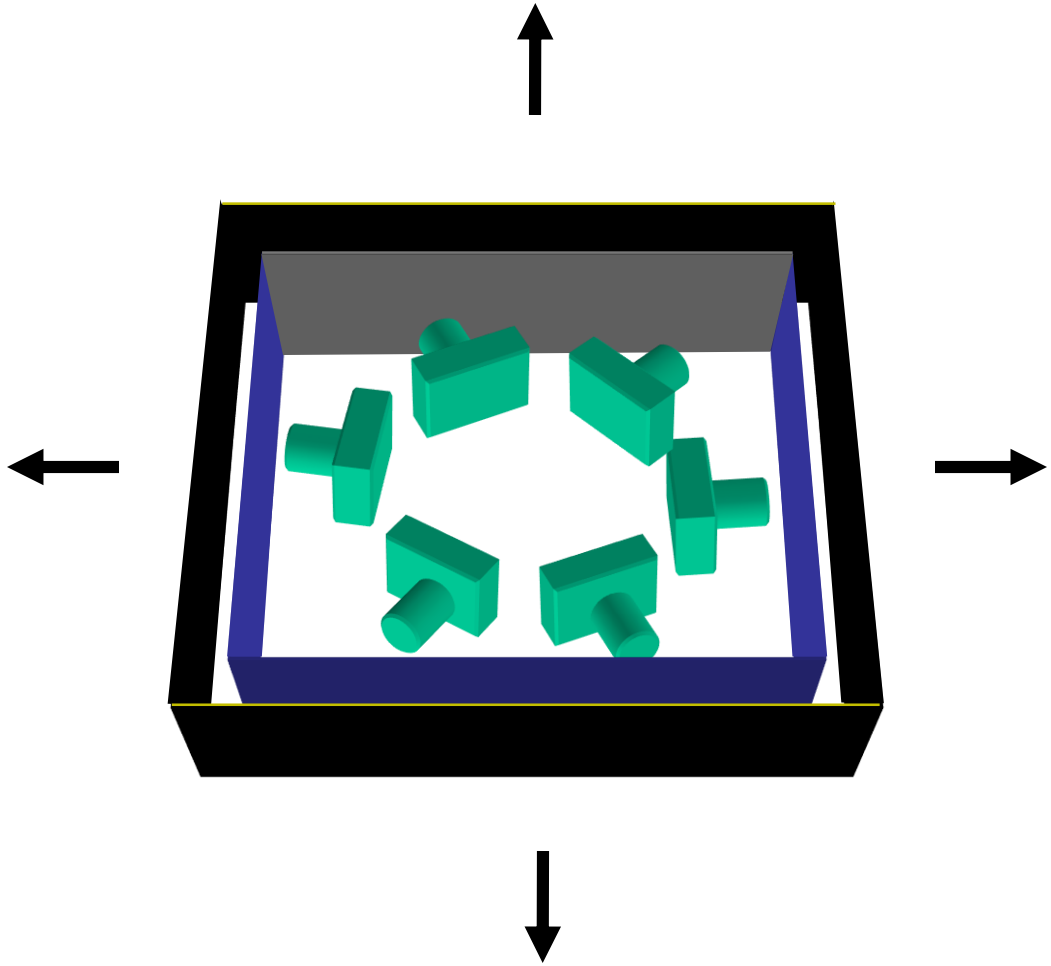


# Panoramic Layering



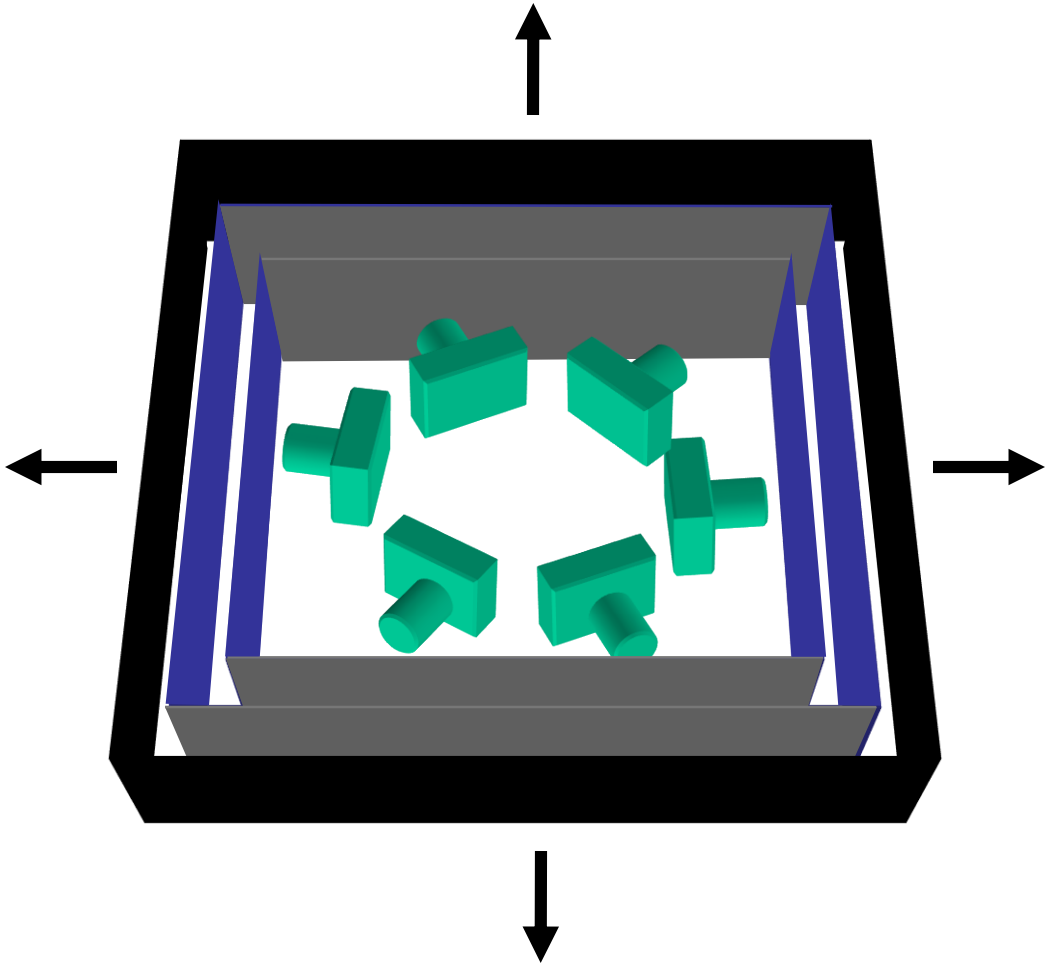
Layers radiate outwards from cameras

# Panoramic Layering



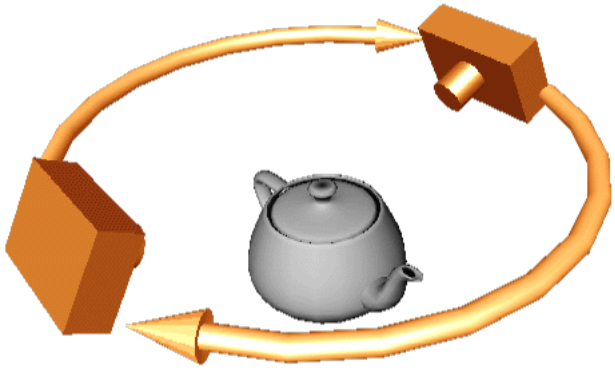
Layers radiate outwards from cameras

# Panoramic Layering

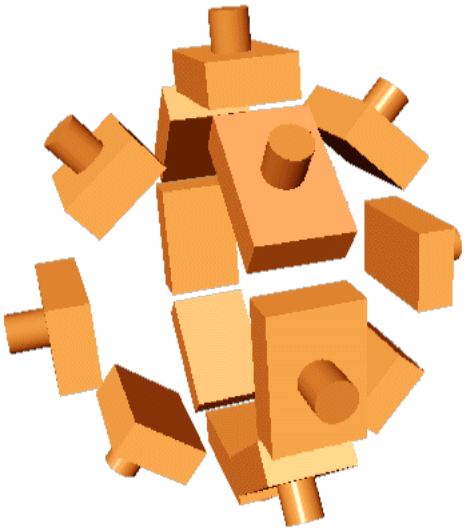


Layers radiate outwards from cameras

# Compatible Camera Configurations



Inward-Looking
cameras above scene



Outward-Looking
cameras inside scene

# Calibrated Image Acquisition

Calibrated
Turntable

360° rotation

(21 images)

Selected Dinosaur Images

Selected Flower Images

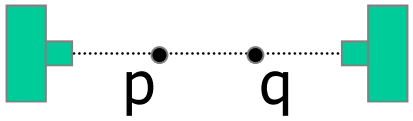
# Voxel Coloring Results (Video)

**Dinosaur Reconstruction**
**72 K voxels colored**
**7.6 M voxels tested**
**7 min. to compute**
**on a 250MHz SGI**

**Flower Reconstruction**
**70 K voxels colored**
**7.6 M voxels tested**
**7 min. to compute**
**on a 250MHz SGI**

# Limitations of Depth Ordering

A view-independent depth order may not exist




Need more powerful general-case algorithms

- Unconstrained camera positions
- Unconstrained scene geometry/topology

# Voxel Coloring Solutions

1. C=2 (silhouettes)

   Volume intersection [Martin 81, Szeliski 93]

2. C unconstrained, viewpoint constraints

   Voxel coloring algorithm [Seitz & Dyer 97]

3. General Case

   Space carving [Kutulakos & Seitz 98]

# Space Carving Algorithm


Space Carving Algorithm

Initialize to a volume V containing the true scene

Choose a voxel on the current surface
Project to visible input images
Carve if not photo-consistent
Repeat until convergence

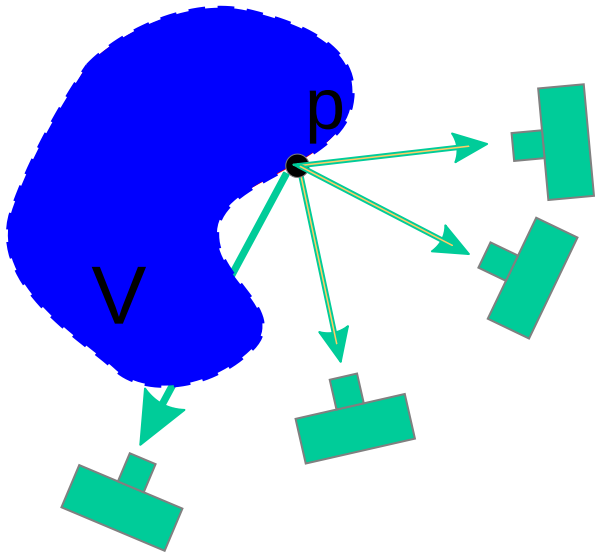
# Convergence

**Consistency Property**

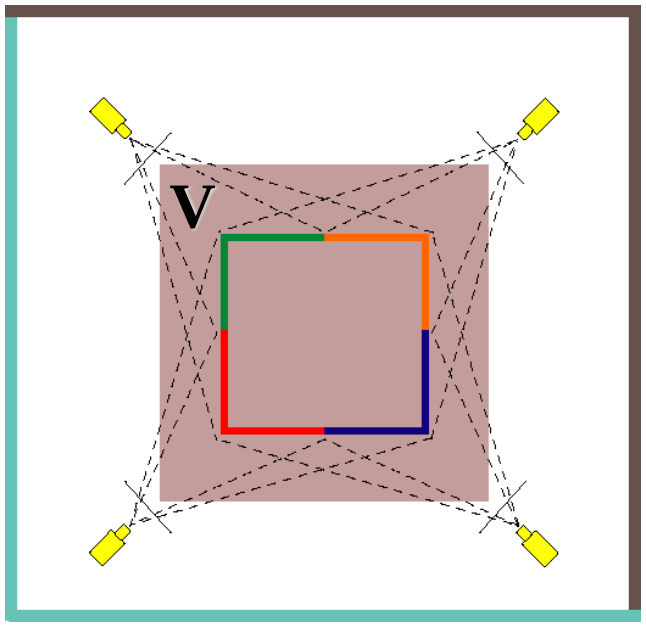
- The resulting shape is photo-consistent
- All inconsistent points are removed

**Convergence Property**

- Carving converges to a non-empty shape
- A point on the true scene is *never* removed



# What is Computable?




True Scene

4 Colors, 4 Cameras:

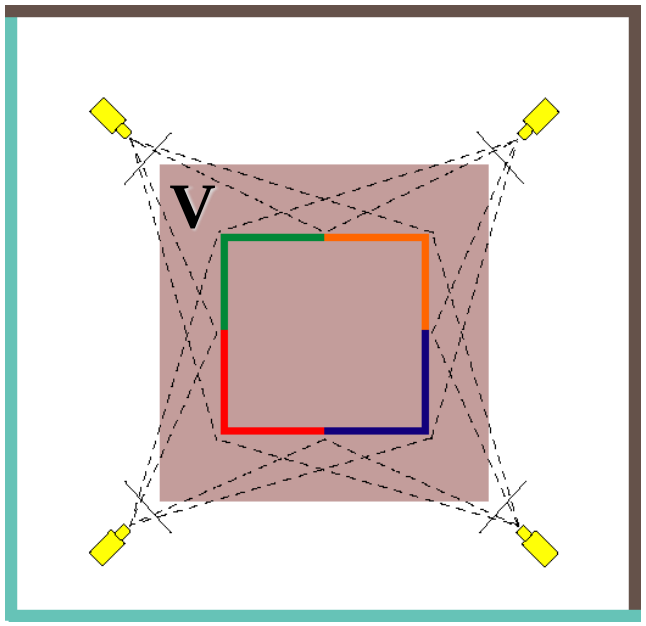
**What does the photo hull look like?**

Photo Hull *is UNION of all photo-consistent scenes in V*

- It is a photo-consistent scene reconstruction
- Tightest possible bound on the true scene
- Computable via provable *Space Carving Algorithm*

# Photo Hull
# (Blackboard)

# What is Computable?




True Scene

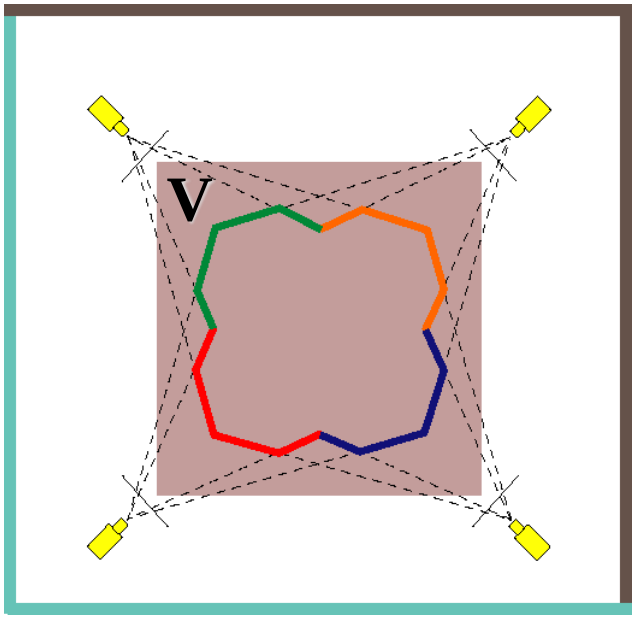



Photo Hull

Photo Hull *is UNION of all photo-consistent scenes in V*

- It is a photo-consistent scene reconstruction
- Tightest possible bound on the true scene
- Computable via provable *Space Carving Algorithm*

# Space Carving Algorithm

The Basic Algorithm is Unwieldy

- Complex update procedure

Alternative: Multi-Pass Plane Sweep

- Efficient, can use texture-mapping hardware
- Converges quickly in practice
- Easy to implement

# Space Carving Algorithm

Step 1:

– Initialize V to volume containing true scene

Step 2:

– For every voxel on surface of V

- test *photo-consistency* of voxel
- if voxel is inconsistent, carve it

Step 3:

– Repeat Step 2 until all voxels consistent

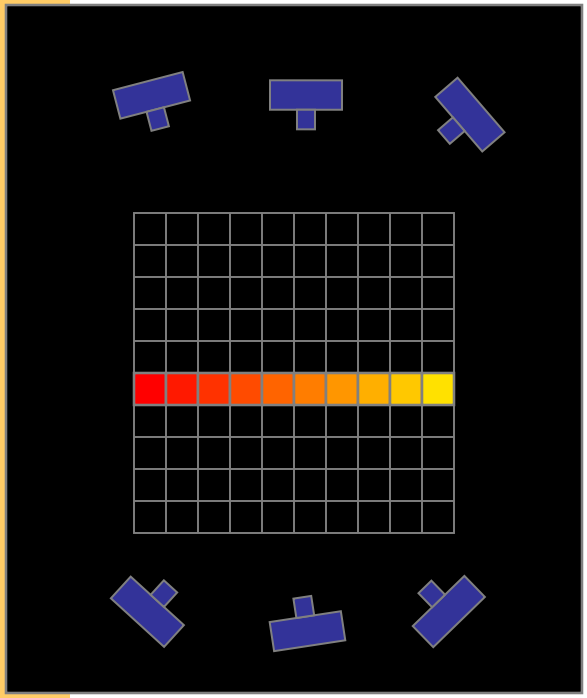
Convergence:

*Always* converges to a photo-consistent model
(when all assumptions are met)

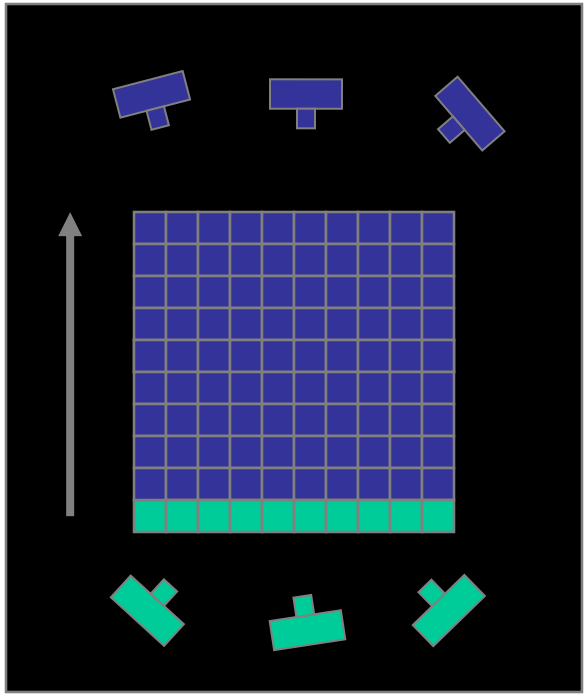
Good results on difficult real-world scenes

# Multi-Pass Plane Sweep

- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence



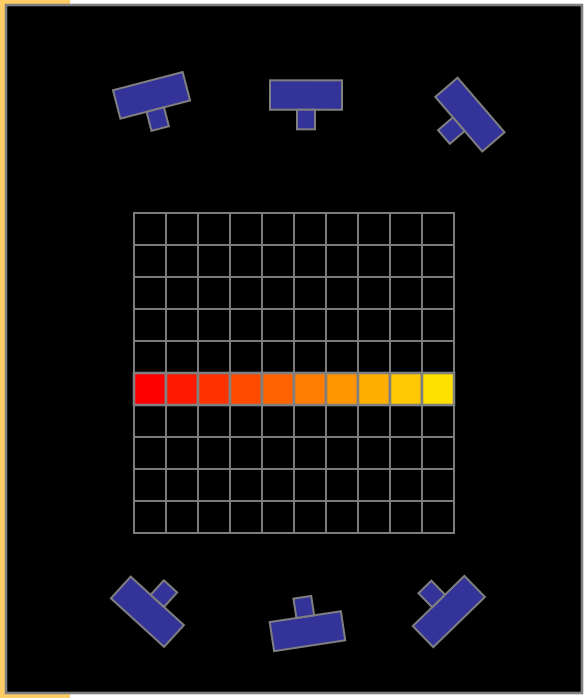
True Scene

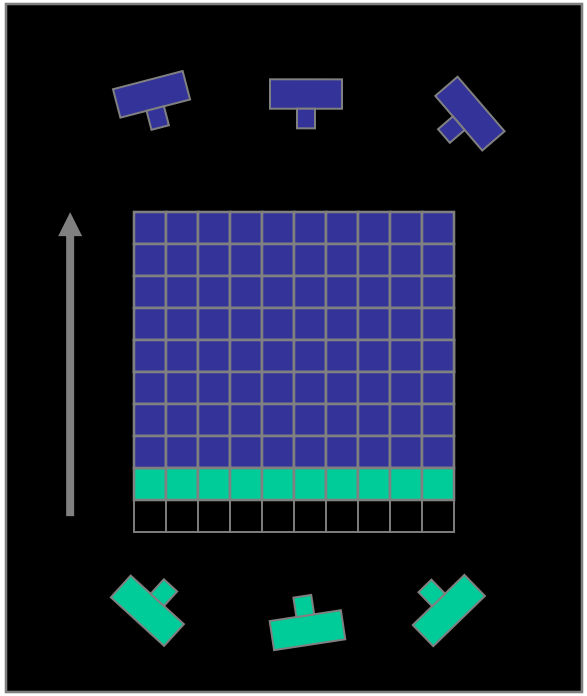


Reconstruction

# Multi-Pass Plane Sweep

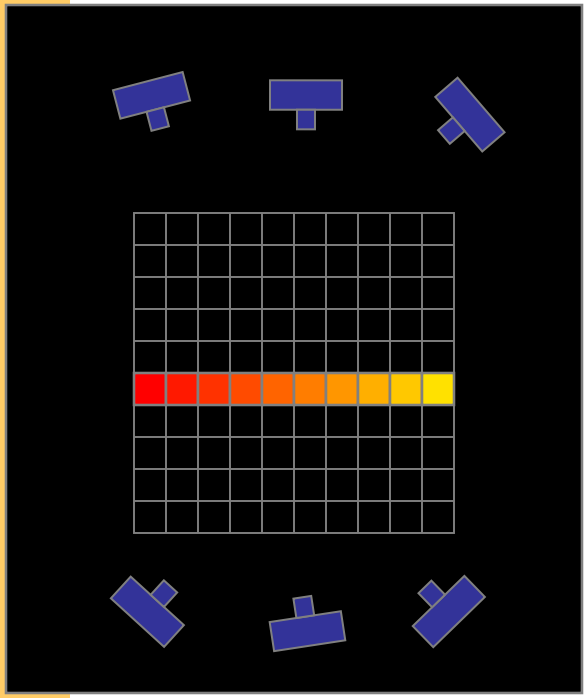
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

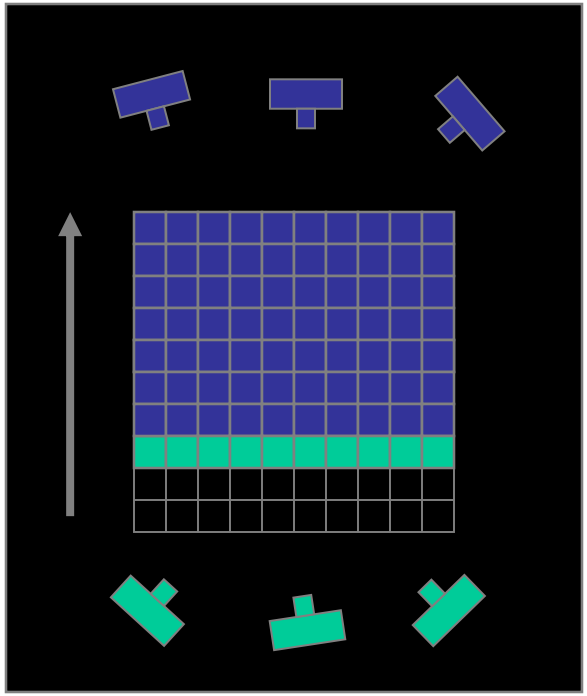




# Multi-Pass Plane Sweep

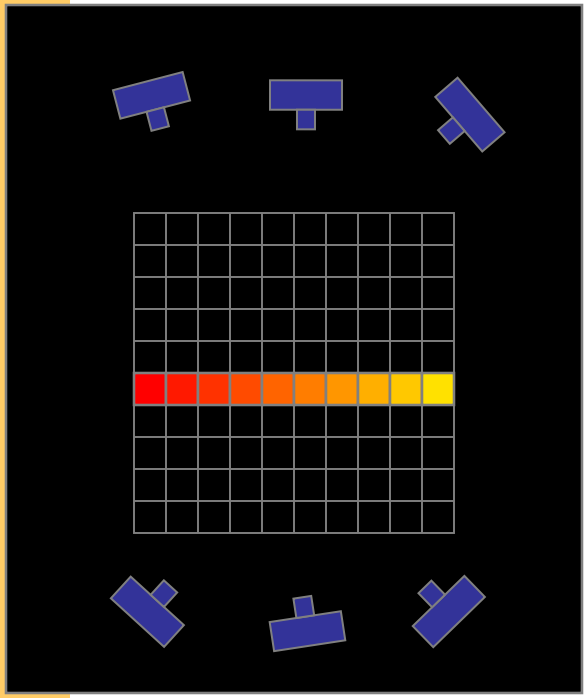
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

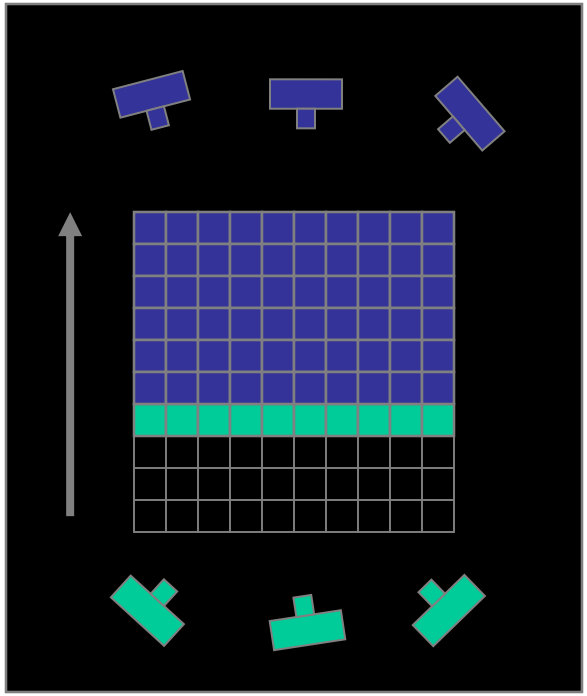




# Multi-Pass Plane Sweep

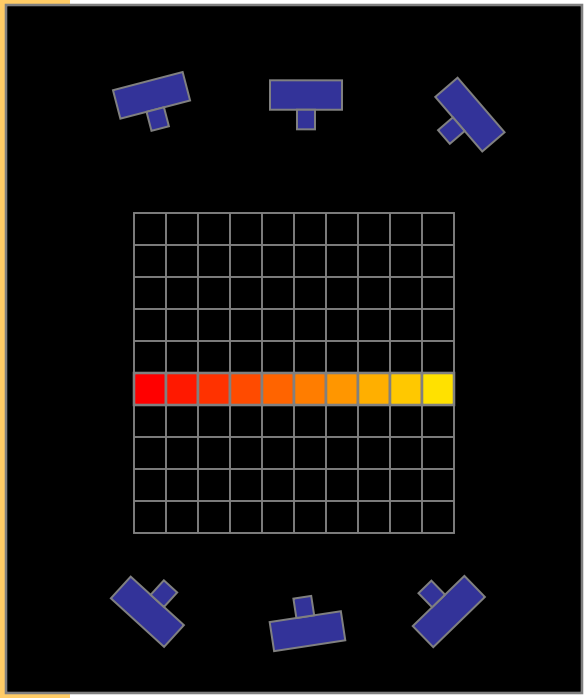
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

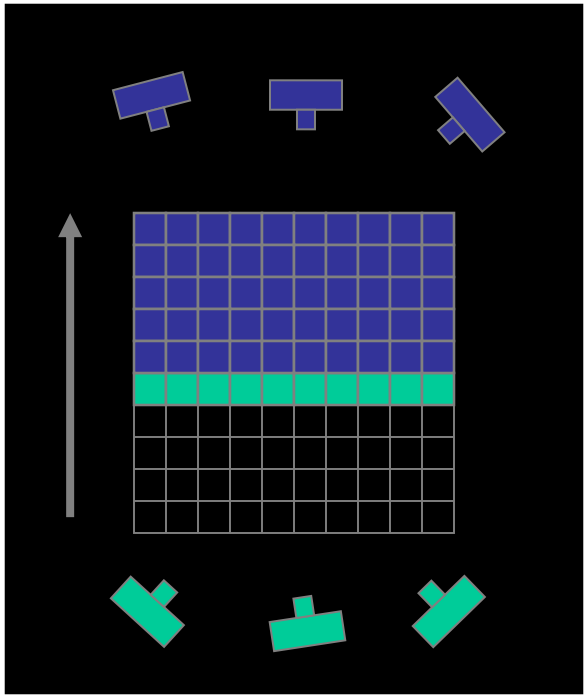




# Multi-Pass Plane Sweep

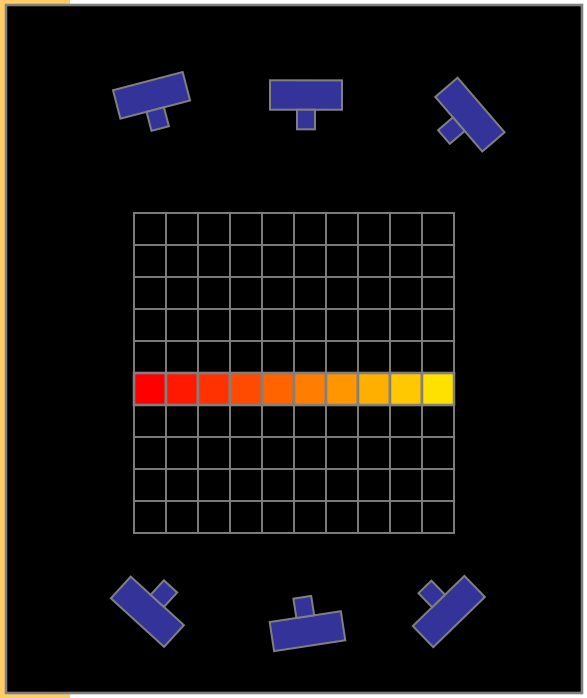
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

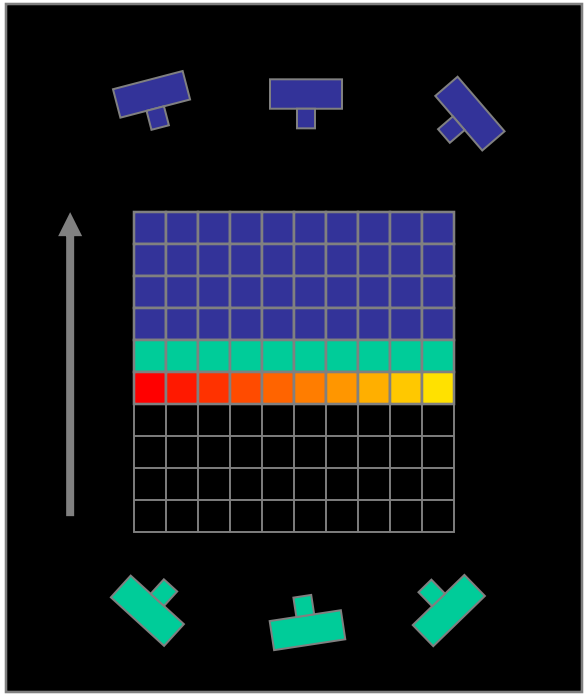




# Multi-Pass Plane Sweep

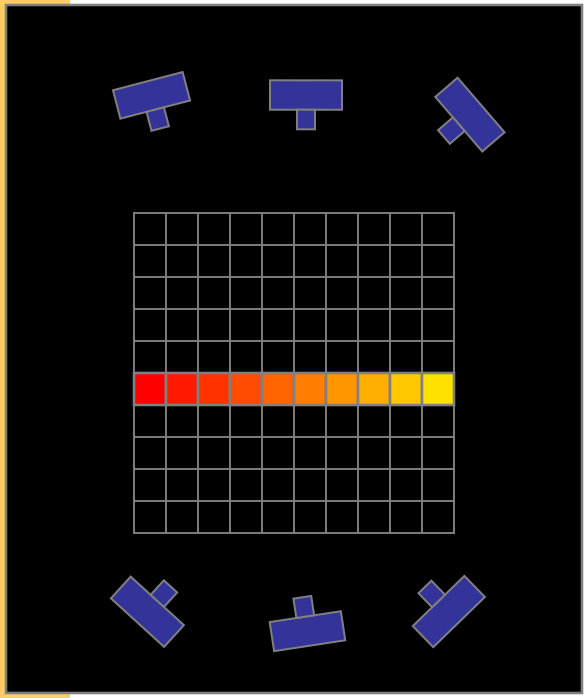
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

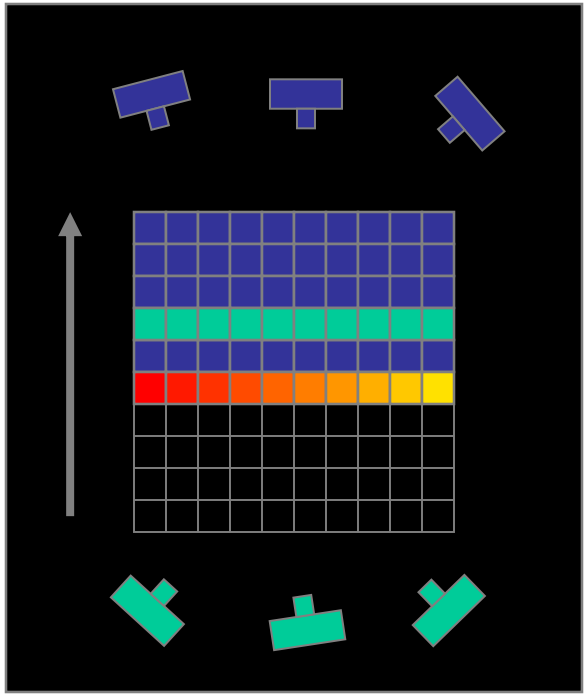




# Multi-Pass Plane Sweep

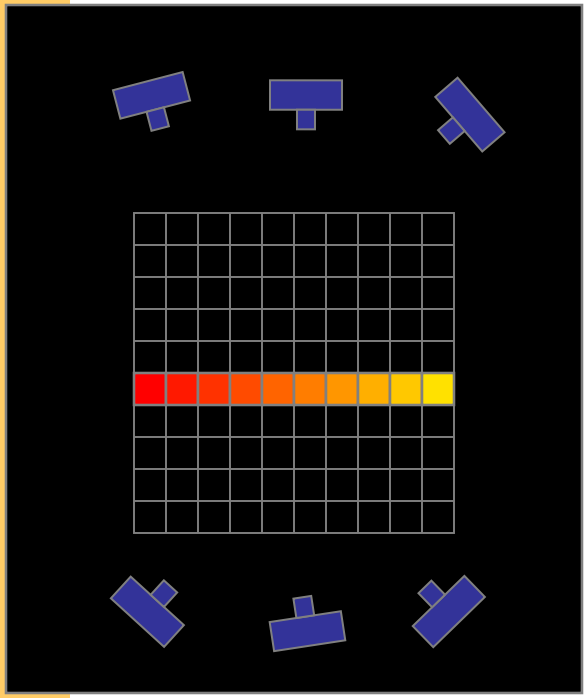
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

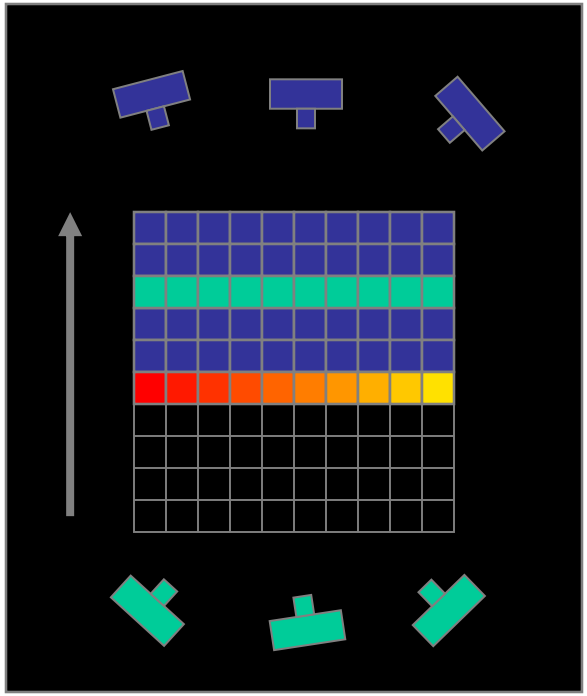




# Multi-Pass Plane Sweep

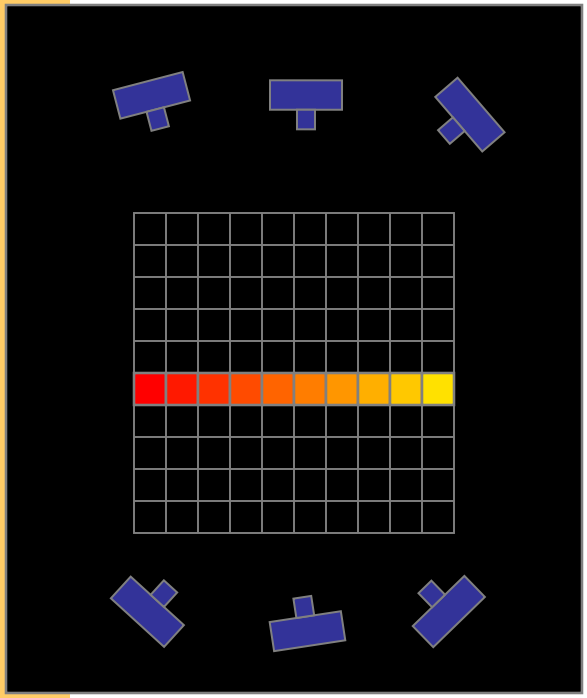
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

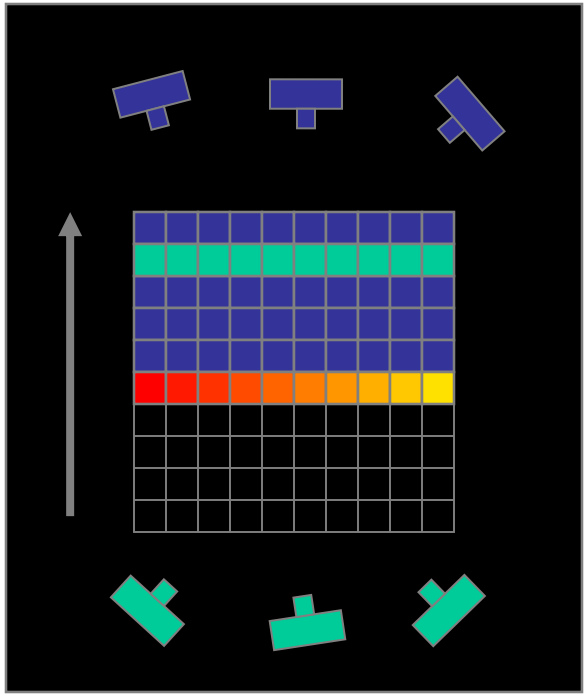




# Multi-Pass Plane Sweep

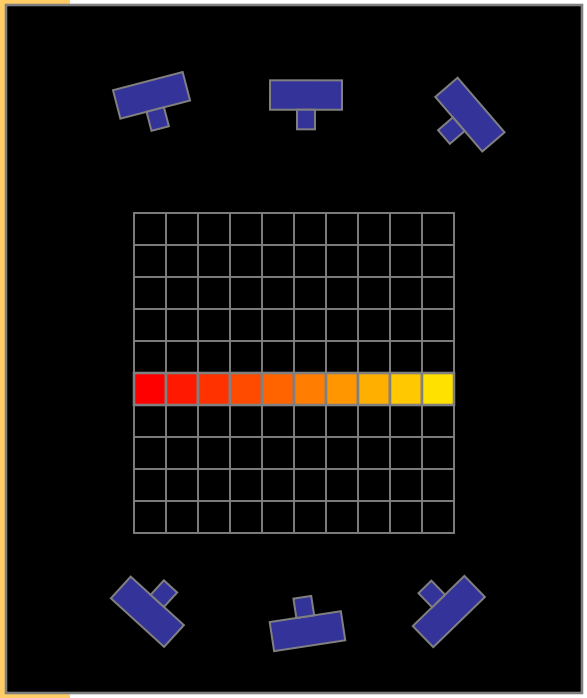
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

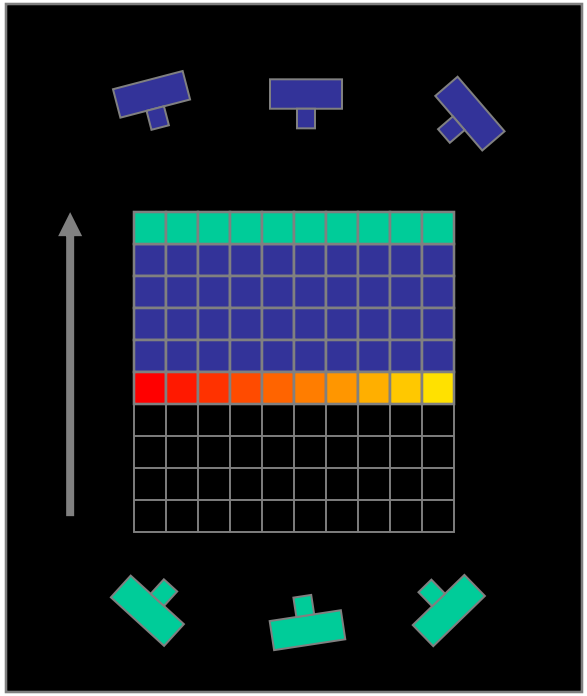




# Multi-Pass Plane Sweep

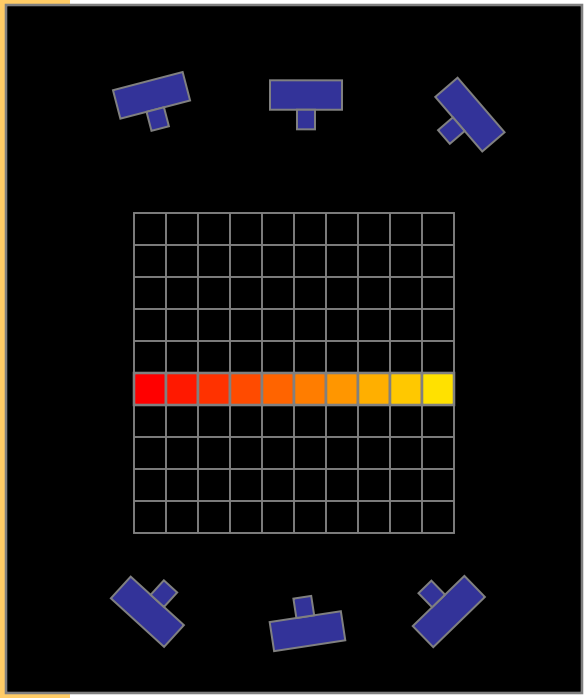
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

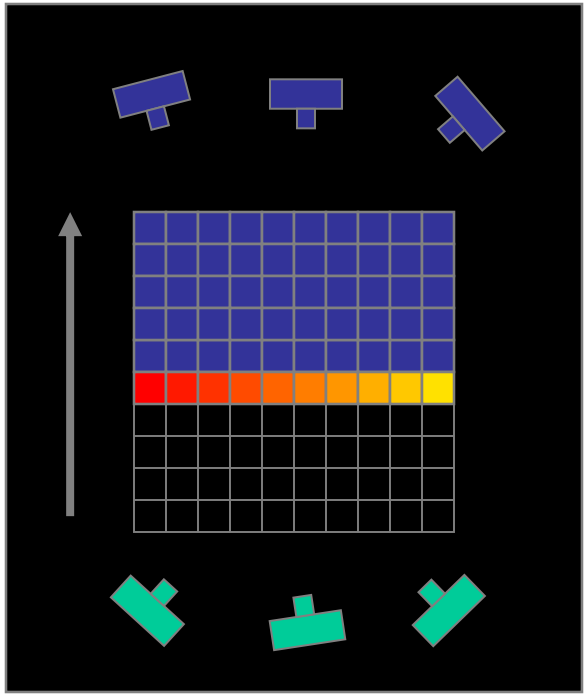




# Multi-Pass Plane Sweep

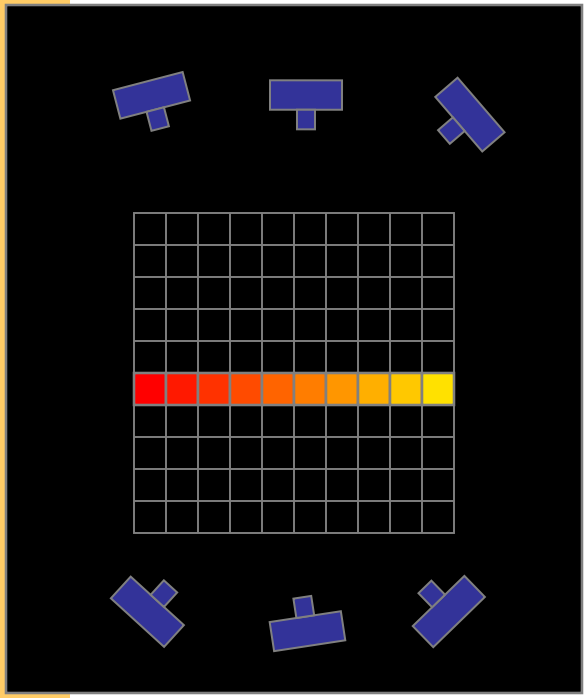
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

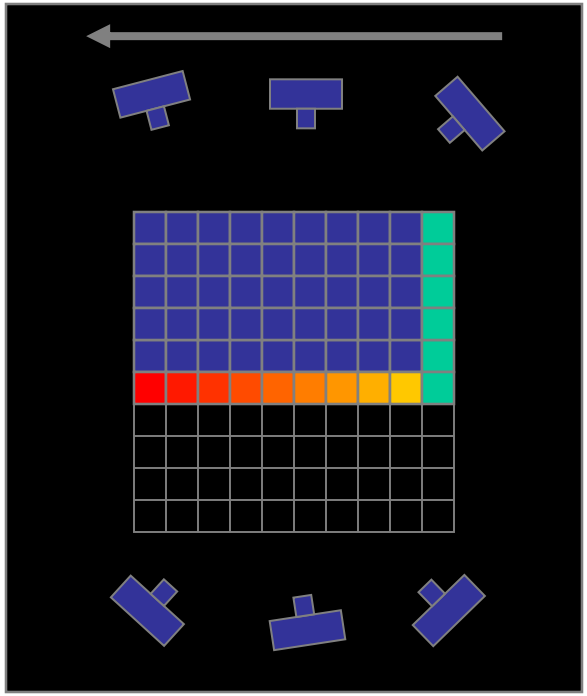




# Multi-Pass Plane Sweep

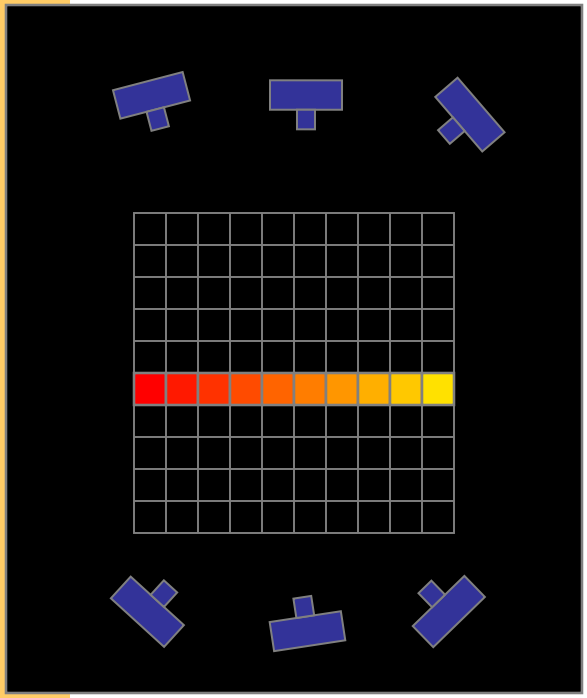
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

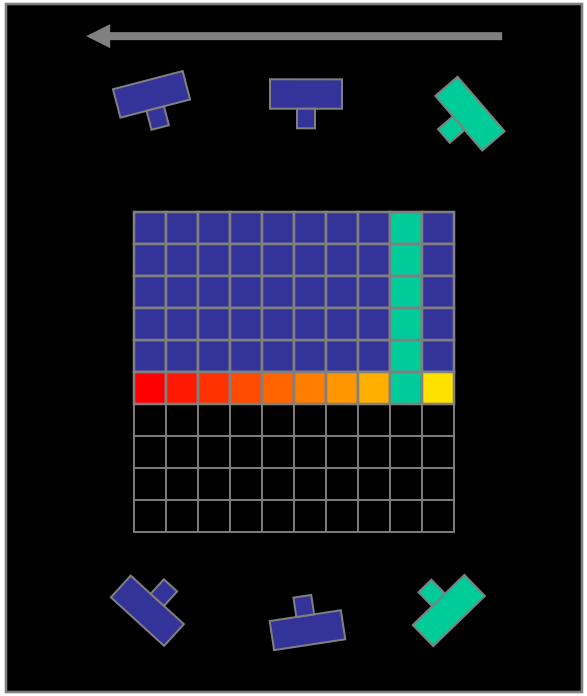




# Multi-Pass Plane Sweep

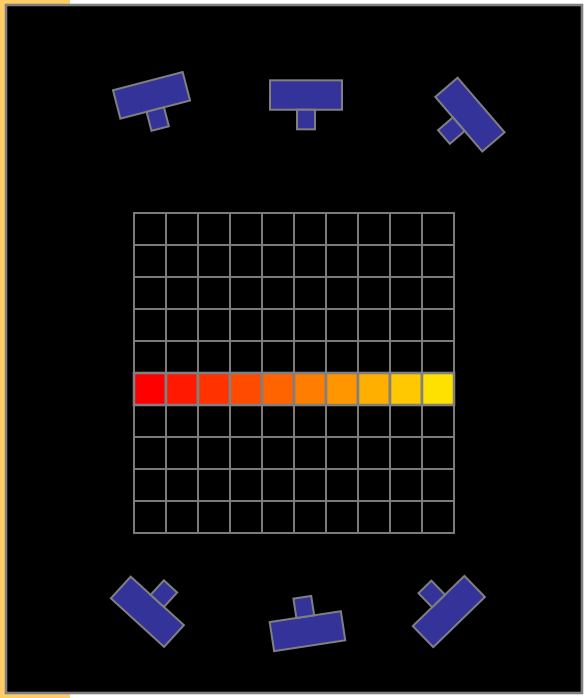
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

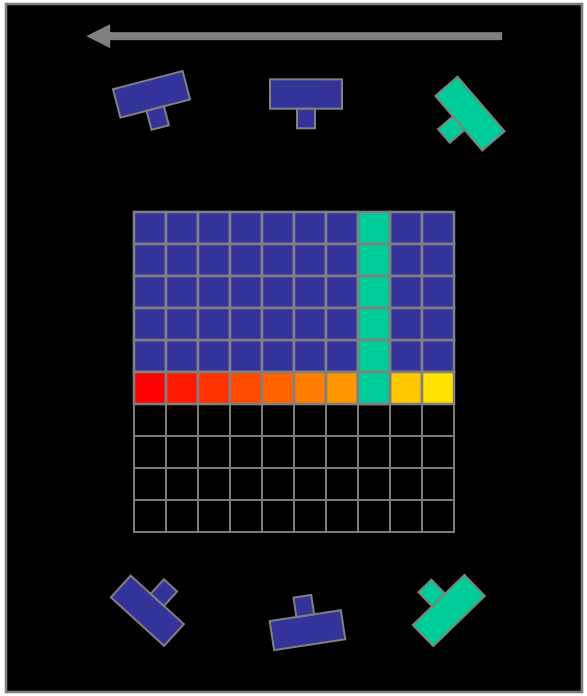




# Multi-Pass Plane Sweep

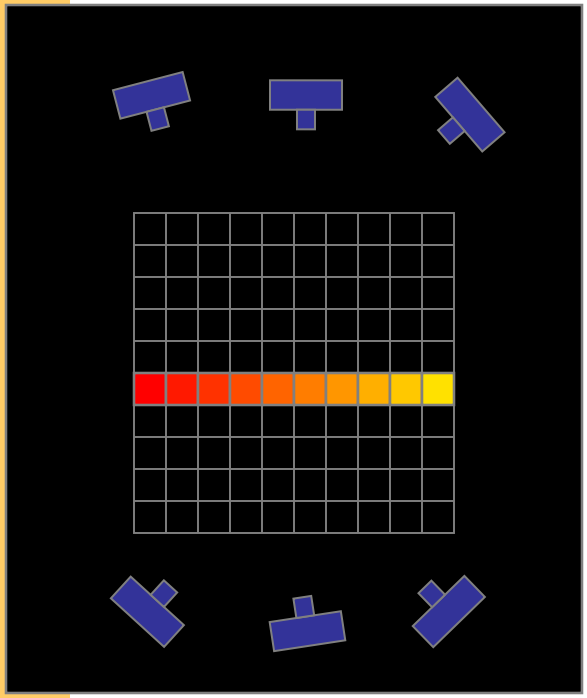
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

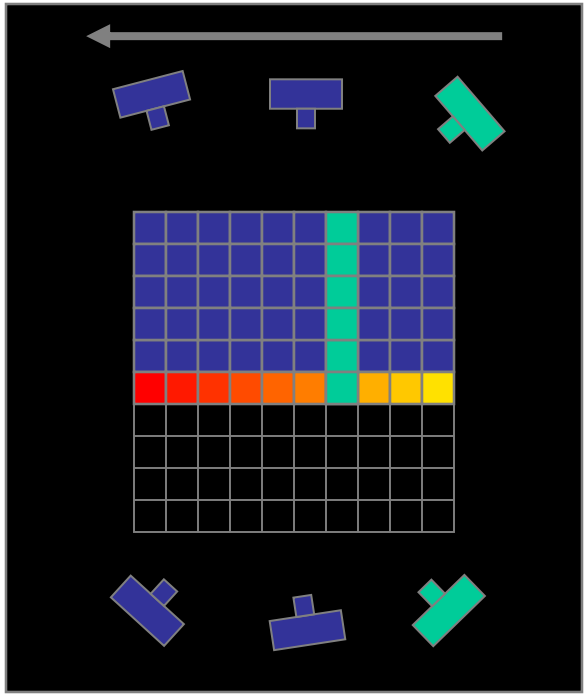




# Multi-Pass Plane Sweep

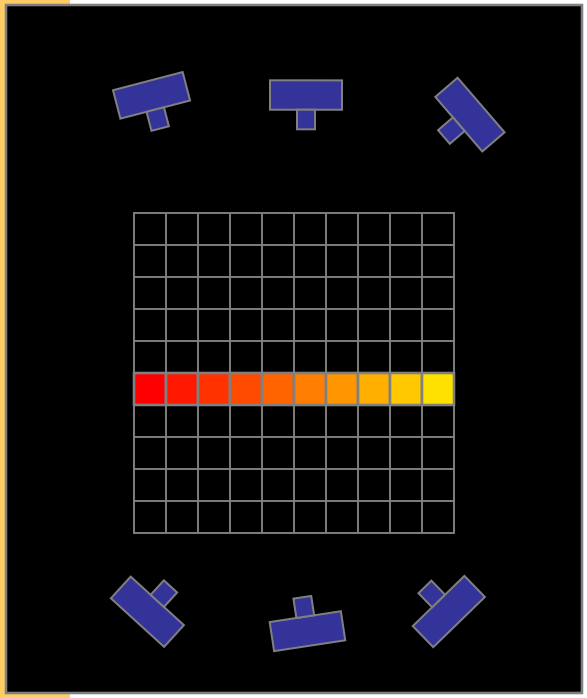
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

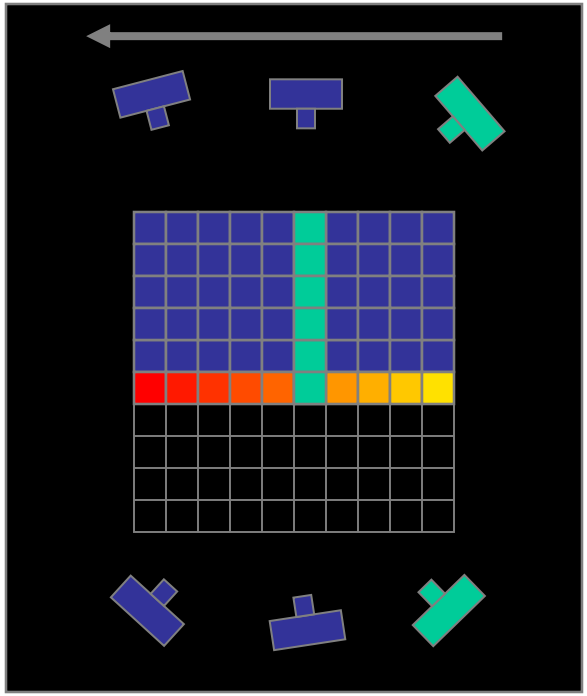




# Multi-Pass Plane Sweep

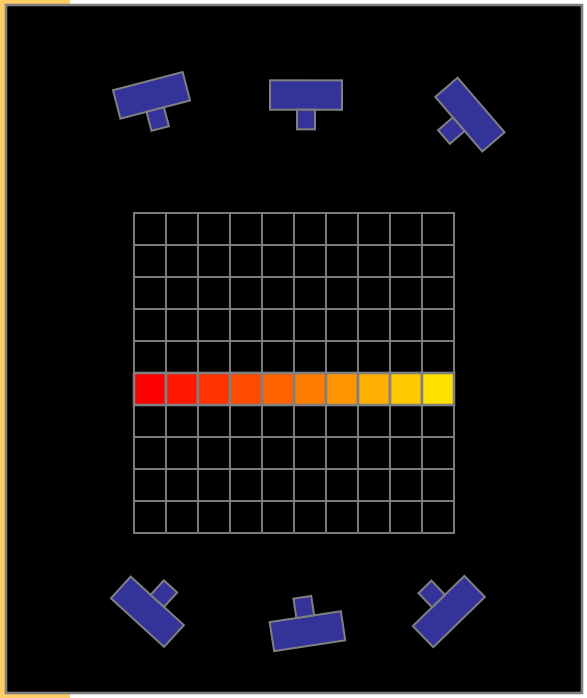
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

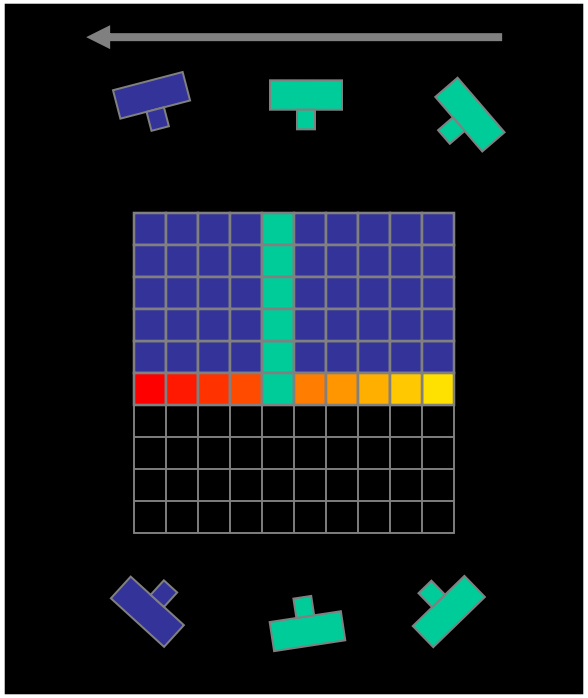




# Multi-Pass Plane Sweep

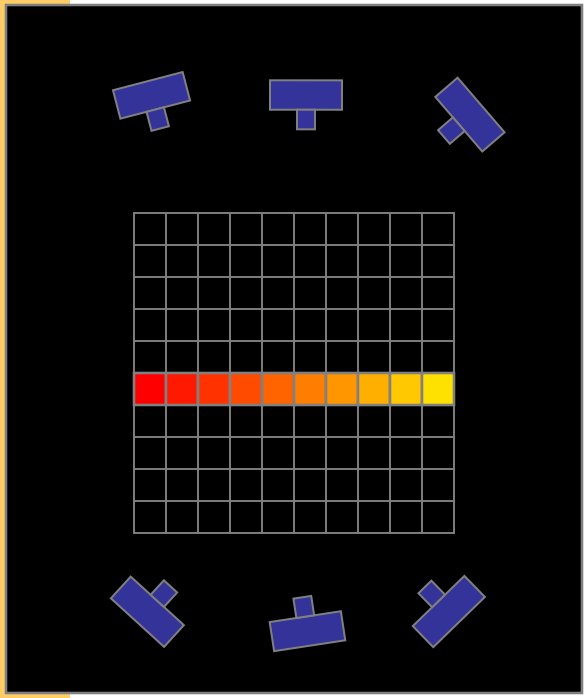
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

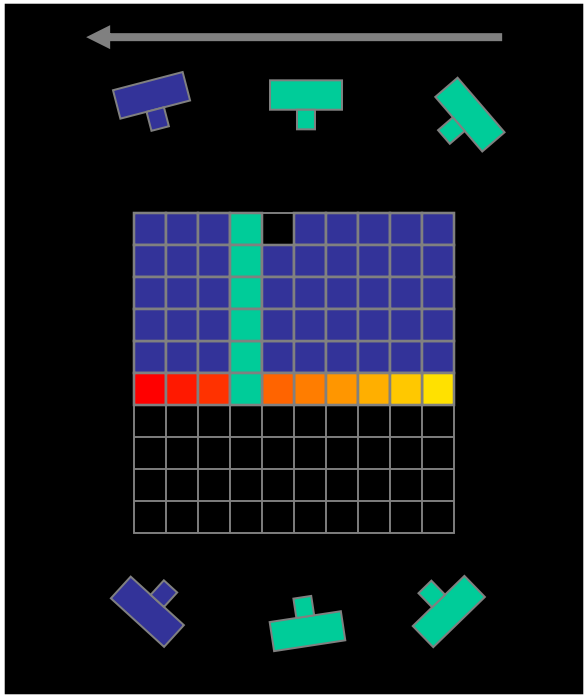




# Multi-Pass Plane Sweep

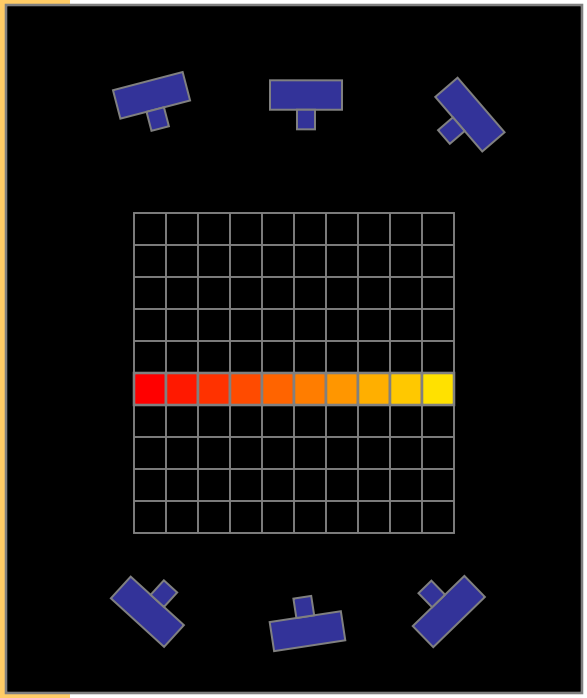
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

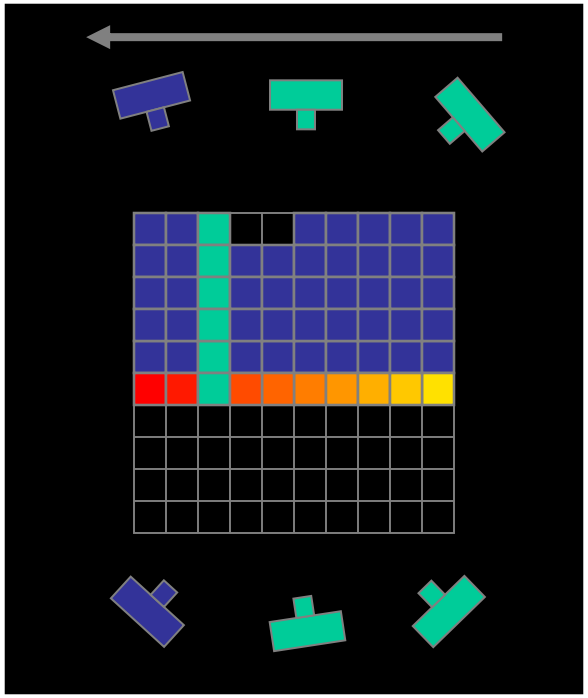




# Multi-Pass Plane Sweep

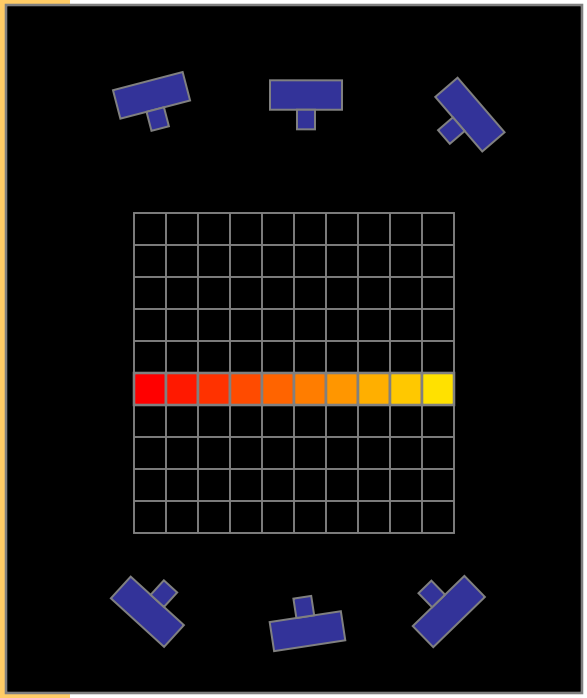
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

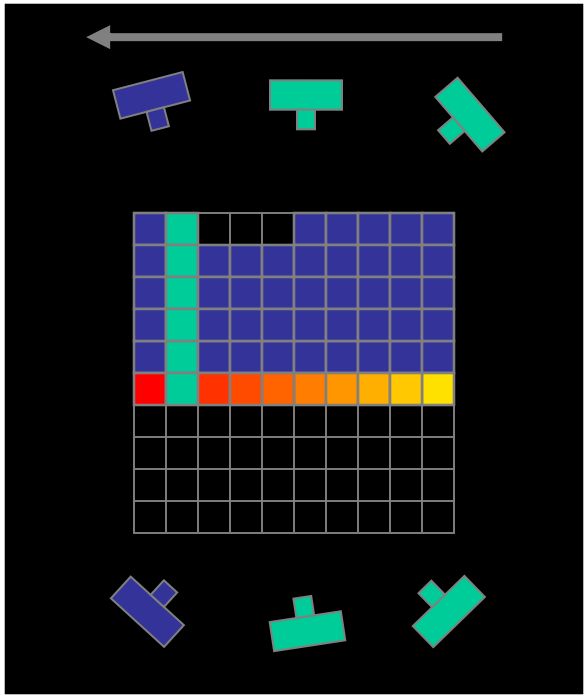




# Multi-Pass Plane Sweep

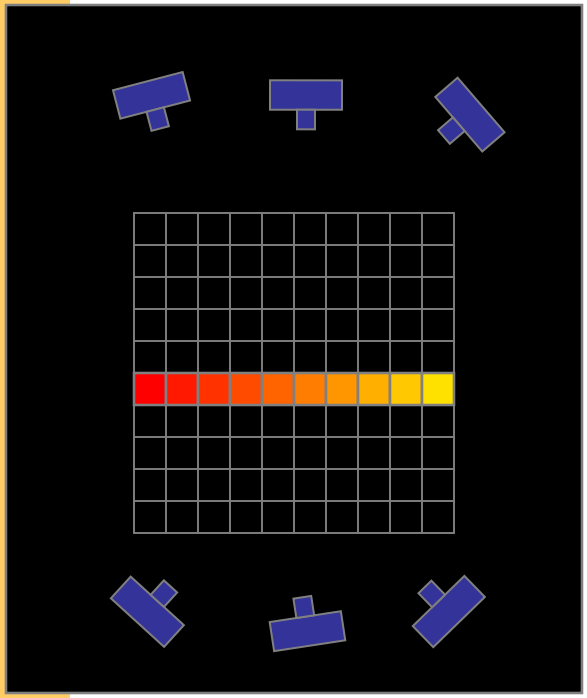
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

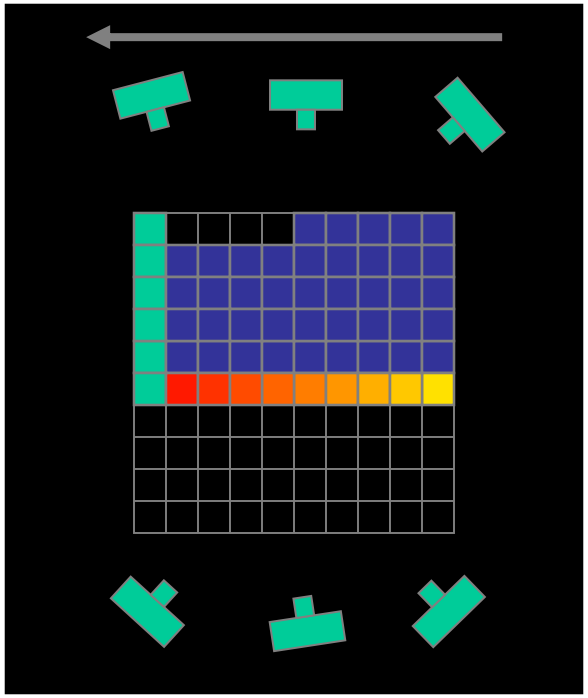




# Multi-Pass Plane Sweep

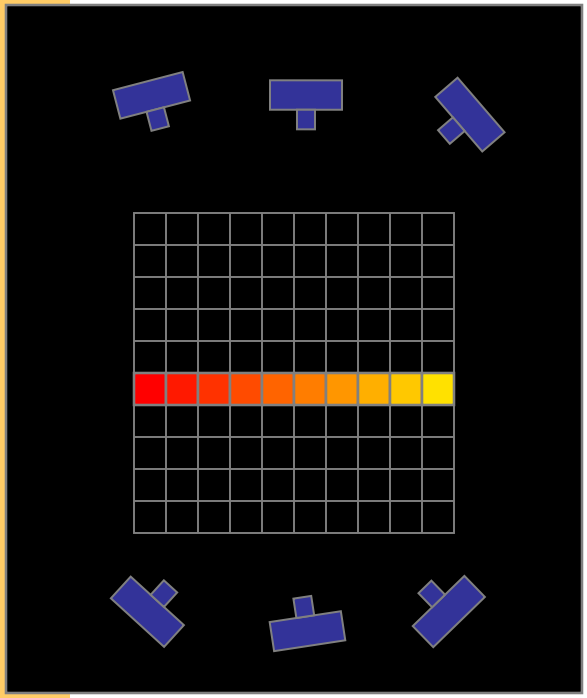
- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

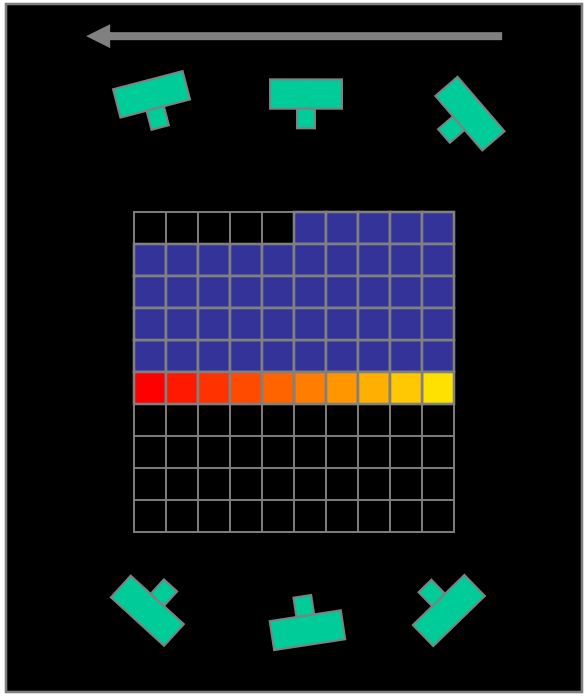




# Multi-Pass Plane Sweep

- Sweep plane in each of 6 principle directions
- Consider cameras on only one side of plane
- Repeat until convergence

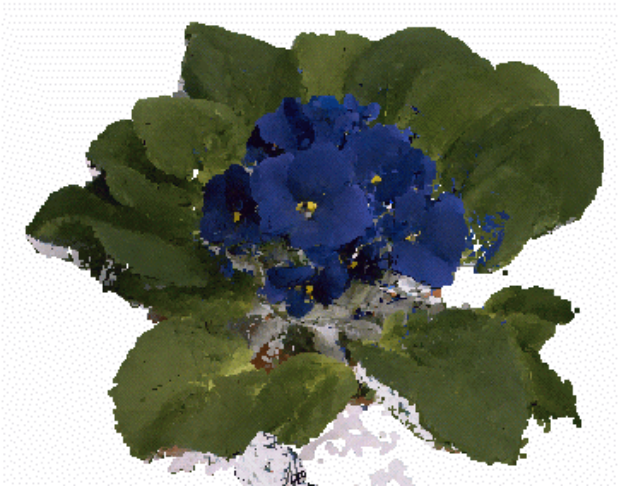




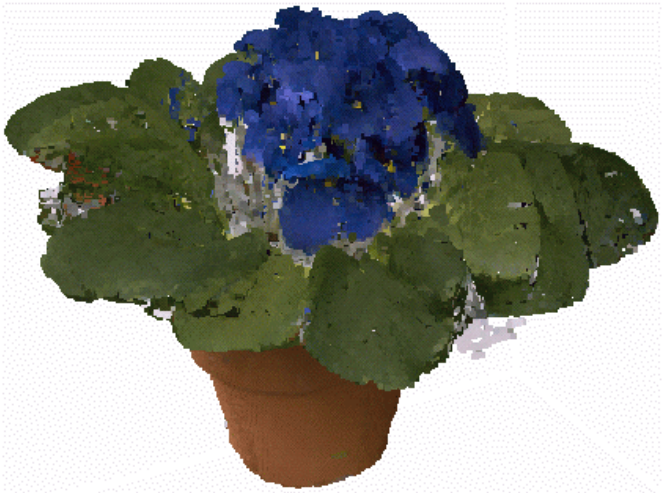
# Space Carving Results: African Violet

**Input Image (1 of 45)**

**Reconstruction**



**Reconstruction**



**Reconstruction**

# Space Carving Results: Hand

**Input Image (1 of 100)**

**Views of Reconstruction**

# Implementation Details

## Coarse-to-fine Reconstruction

- Represent scene as octree
- Reconstruct low-res model first, then refine

## Hardware-Acceleration

- Use texture-mapping to compute voxel projections
- Process voxels an entire plane at a time

## Limitations

- Need to acquire calibrated images
- Restriction to simple radiance models
- Transparency not supported
- Assumes good photo consistency estimate

# Where does it go wrong?

12%

9%

6%

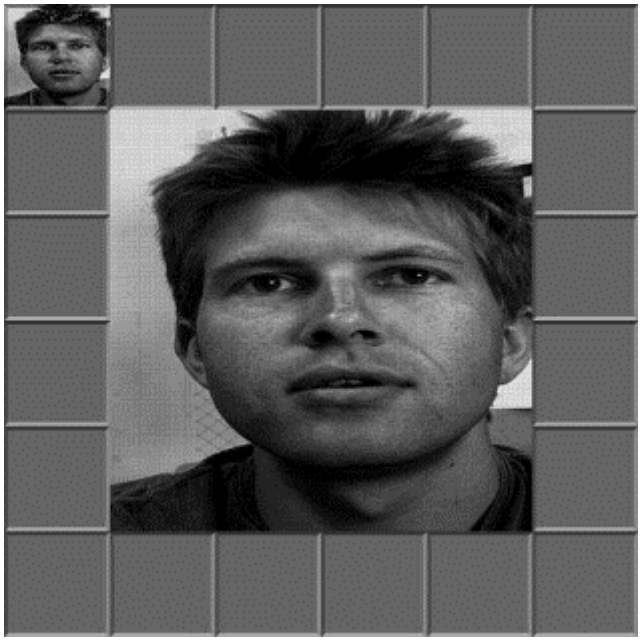
When the images contain noise, there is a trade off between the quality of the reconstruction and the number of holes.

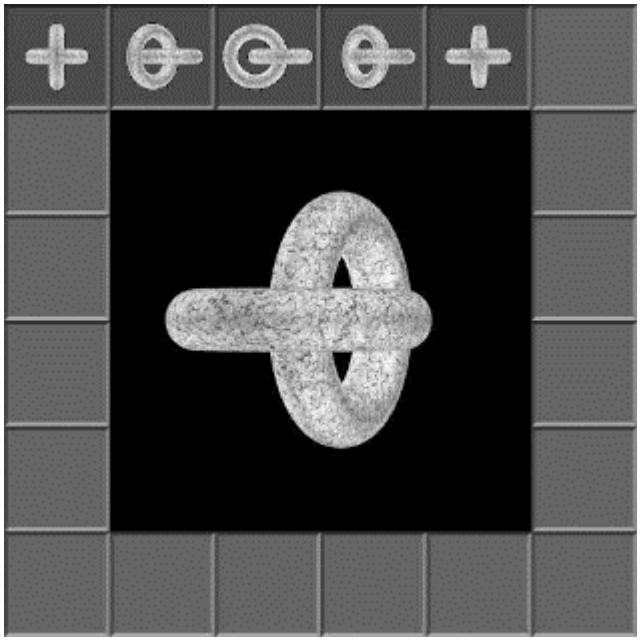
12% 9% 6%

# Other Approaches

Level-Set Methods [Faugeras & Keriven 1998]

- 3D shape as 4D "helper" function's zero crossing
- Evolve implicit function by solving PDE's





# Volumetric Graph cuts

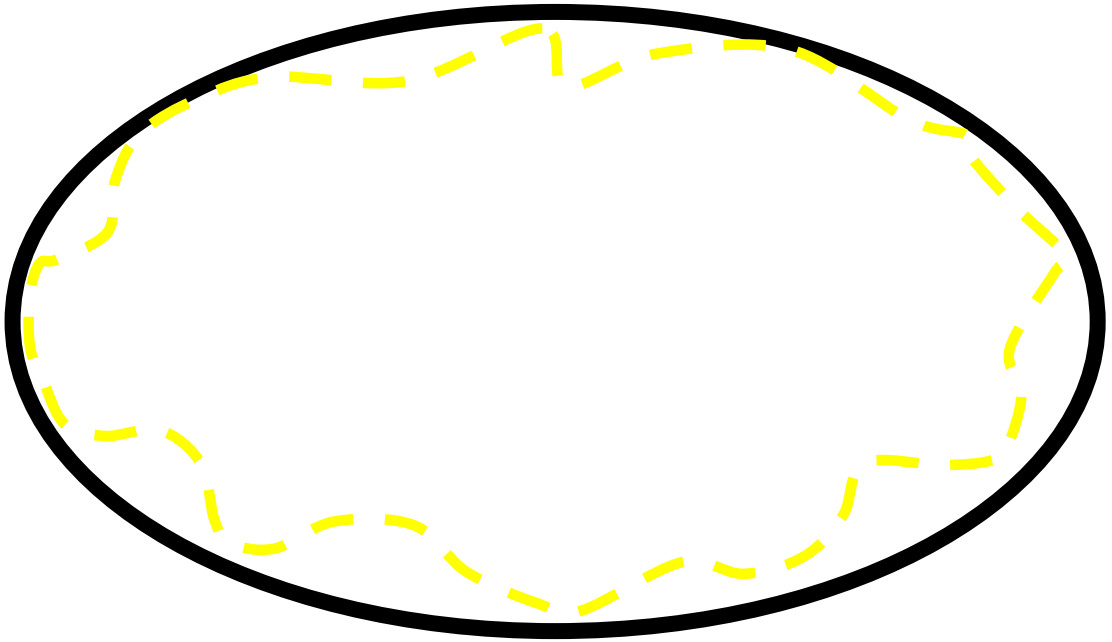
Input: Visual Hull



Slides from [Vogiatzis et al. CVPR2005]

# Volumetric Graph cuts

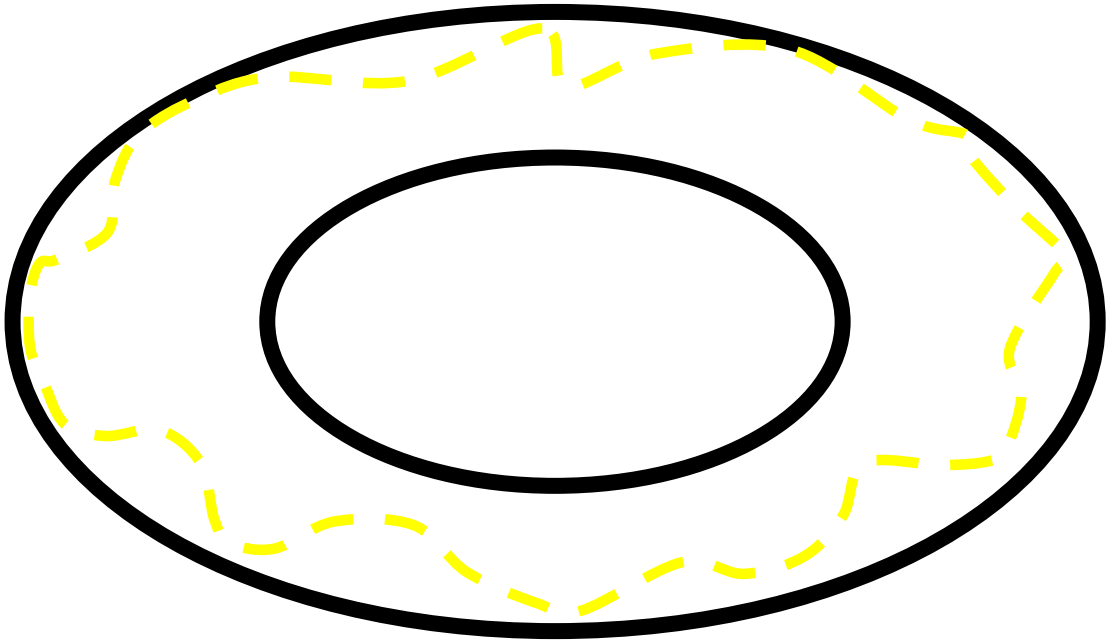
1. Outer surface



Slides from [Vogiatzis et al. CVPR2005]

# Volumetric Graph cuts

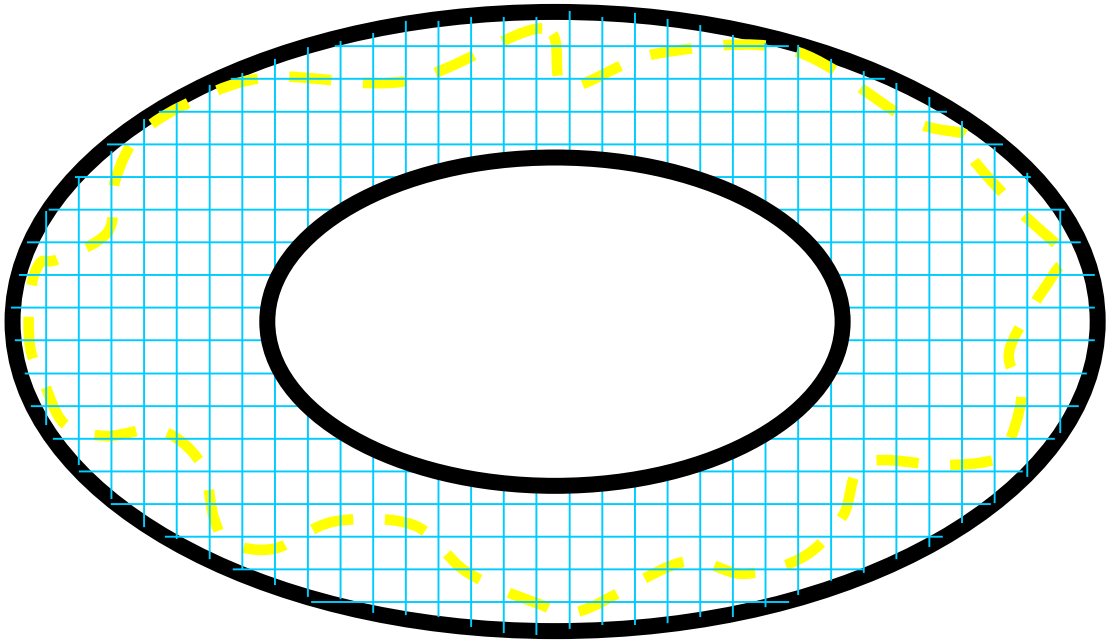
1. Outer surface
2. Inner surface
(at constant offset)



Slides from [Vogiatzis et al. CVPR2005]

# Volumetric Graph cuts

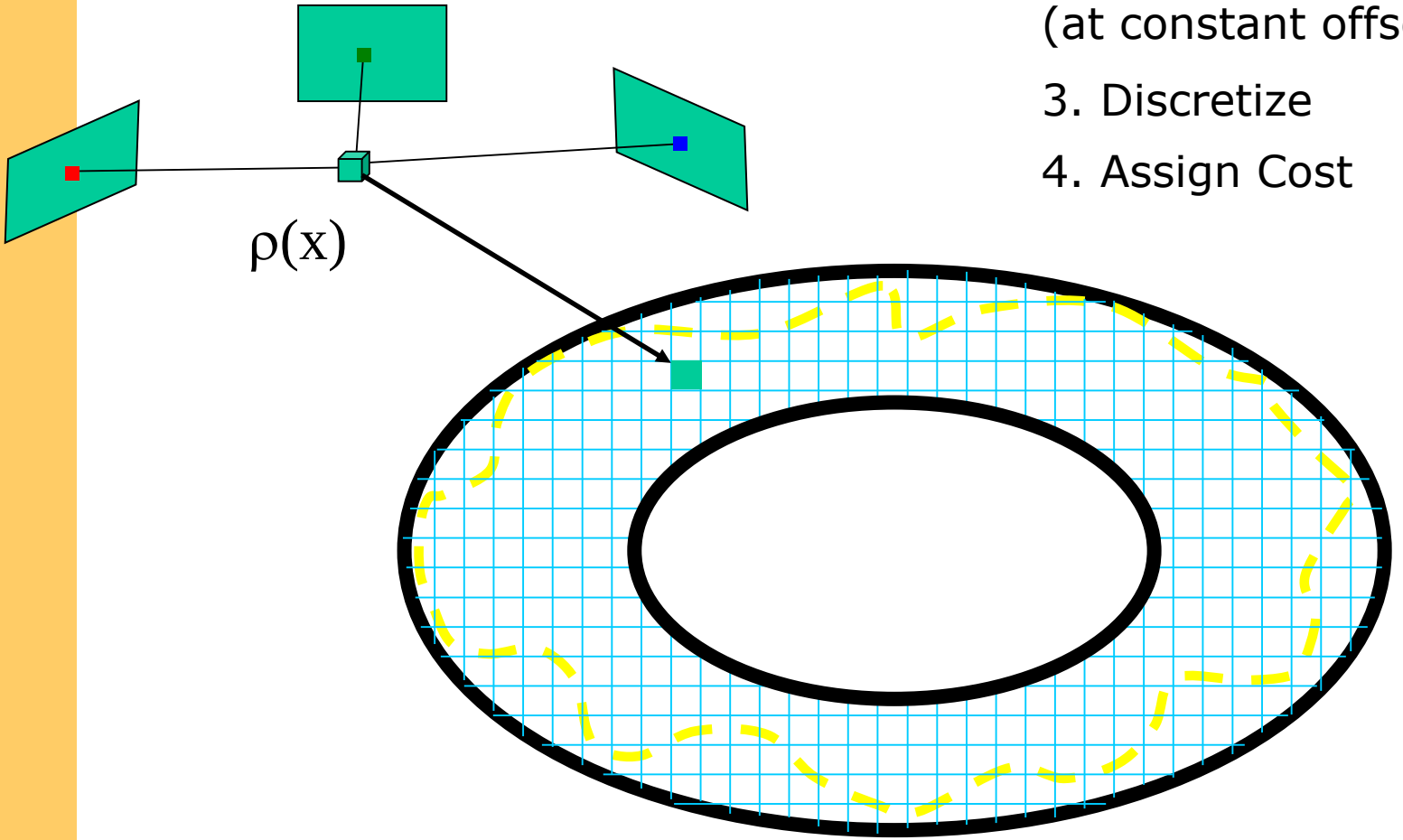
1. Outer surface
2. Inner surface (at constant offset)
3. Discretize



Slides from [Vogiatzis et al. CVPR2005]

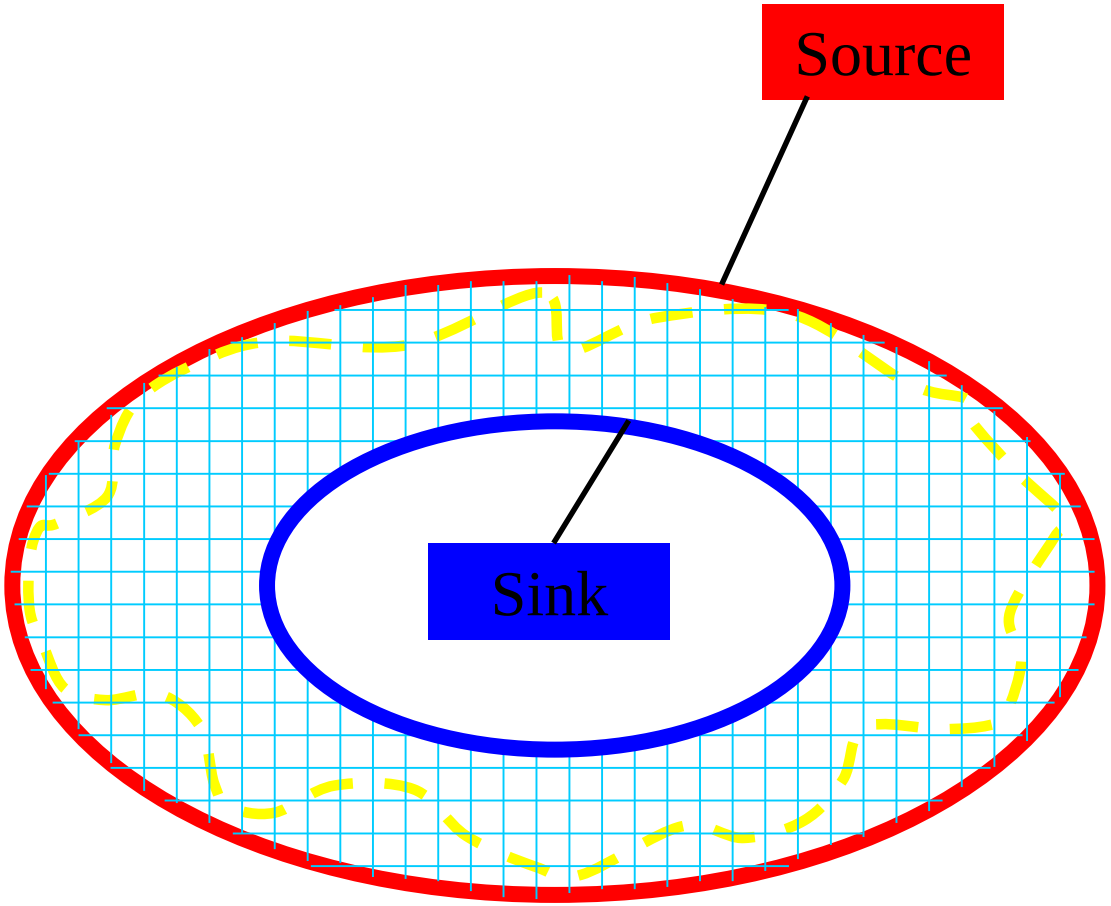
# Volumetric Graph cuts

1. Outer surface
2. Inner surface (at constant offset)
3. Discretize
4. Assign Cost




Slides from [Vogiatzis et al. CVPR2005]

# Volumetric Graph cuts

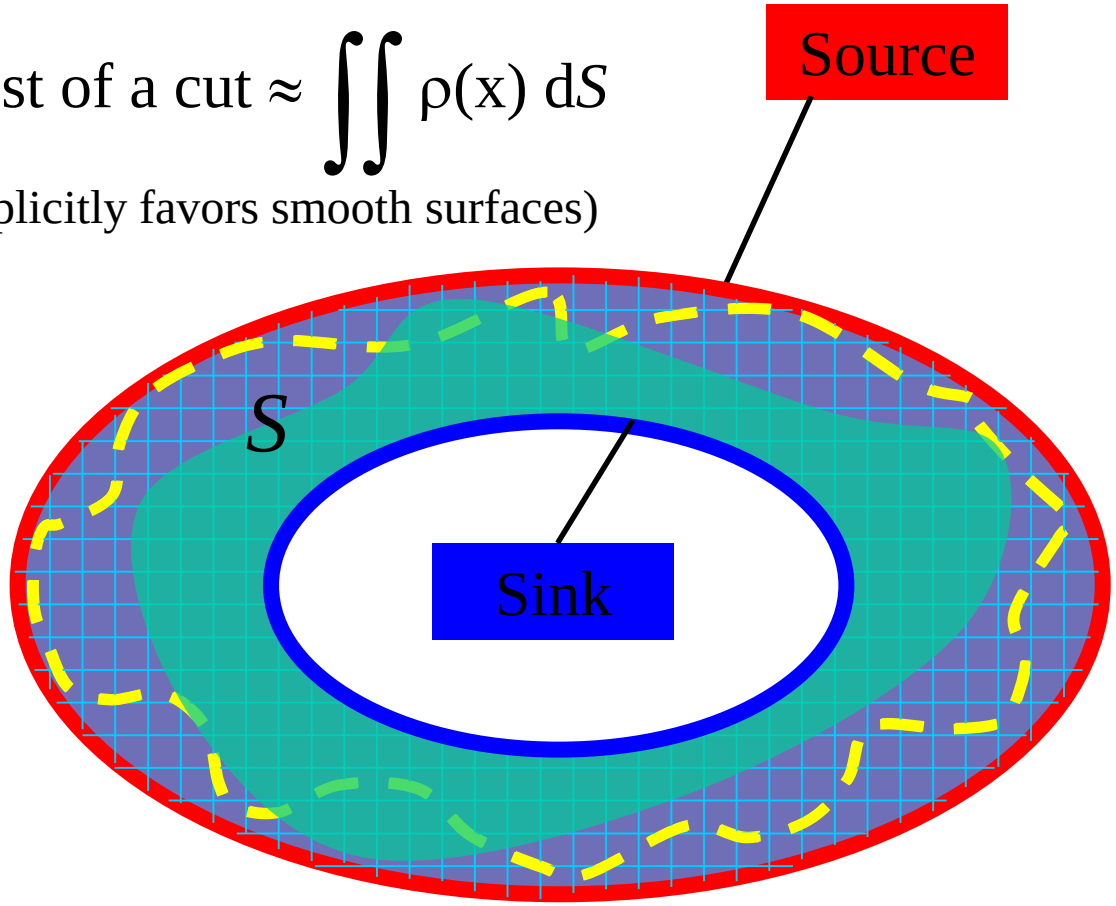


# Volumetric Graph cuts

Cut $\Leftrightarrow$ 3D Surface $S$

Cost of a cut $\approx \iint \rho(x)\, dS$

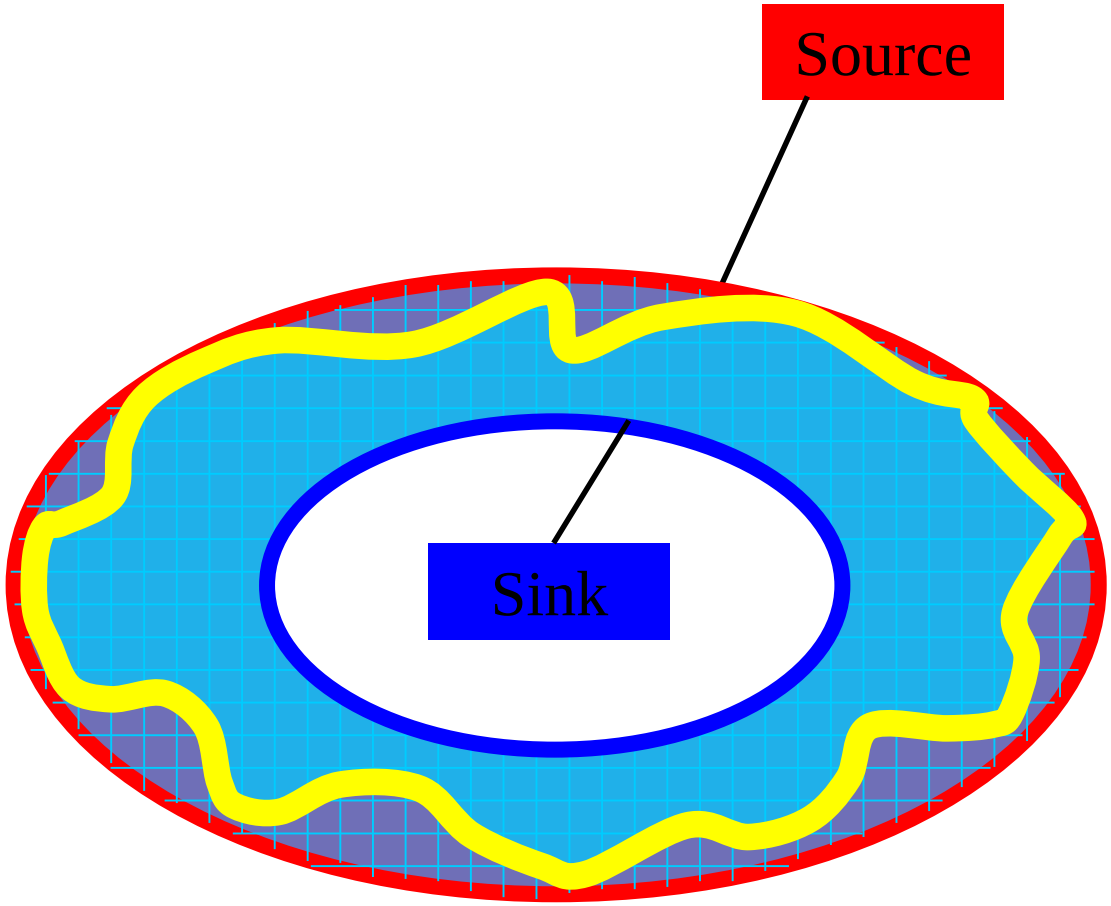
(implicitly favors smooth surfaces)



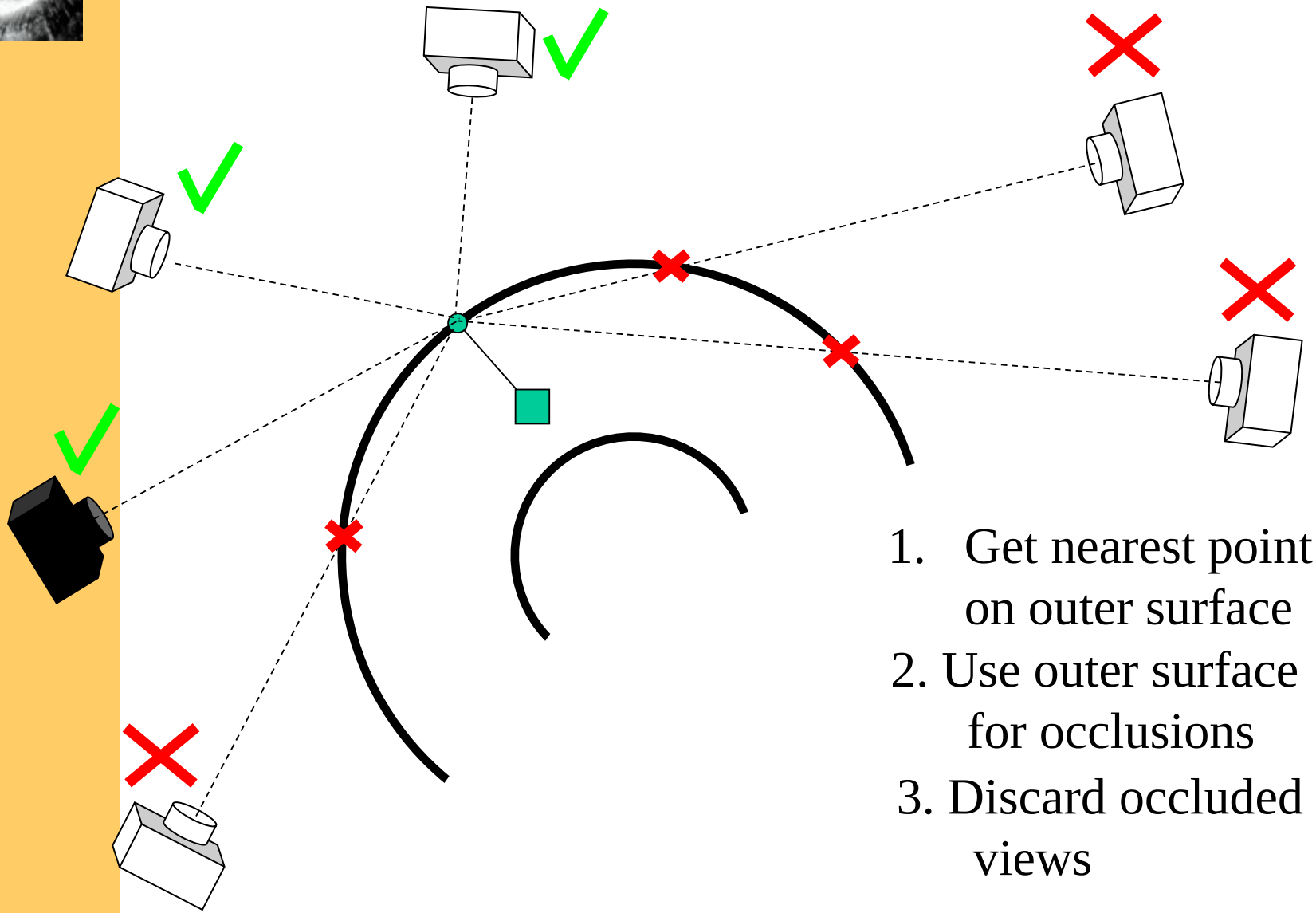

[Boykov and Kolmogorov ICCV 2001]

# Volumetric Graph cuts

Minimum cut ⇔ Minimal photo consistent 3D Surface

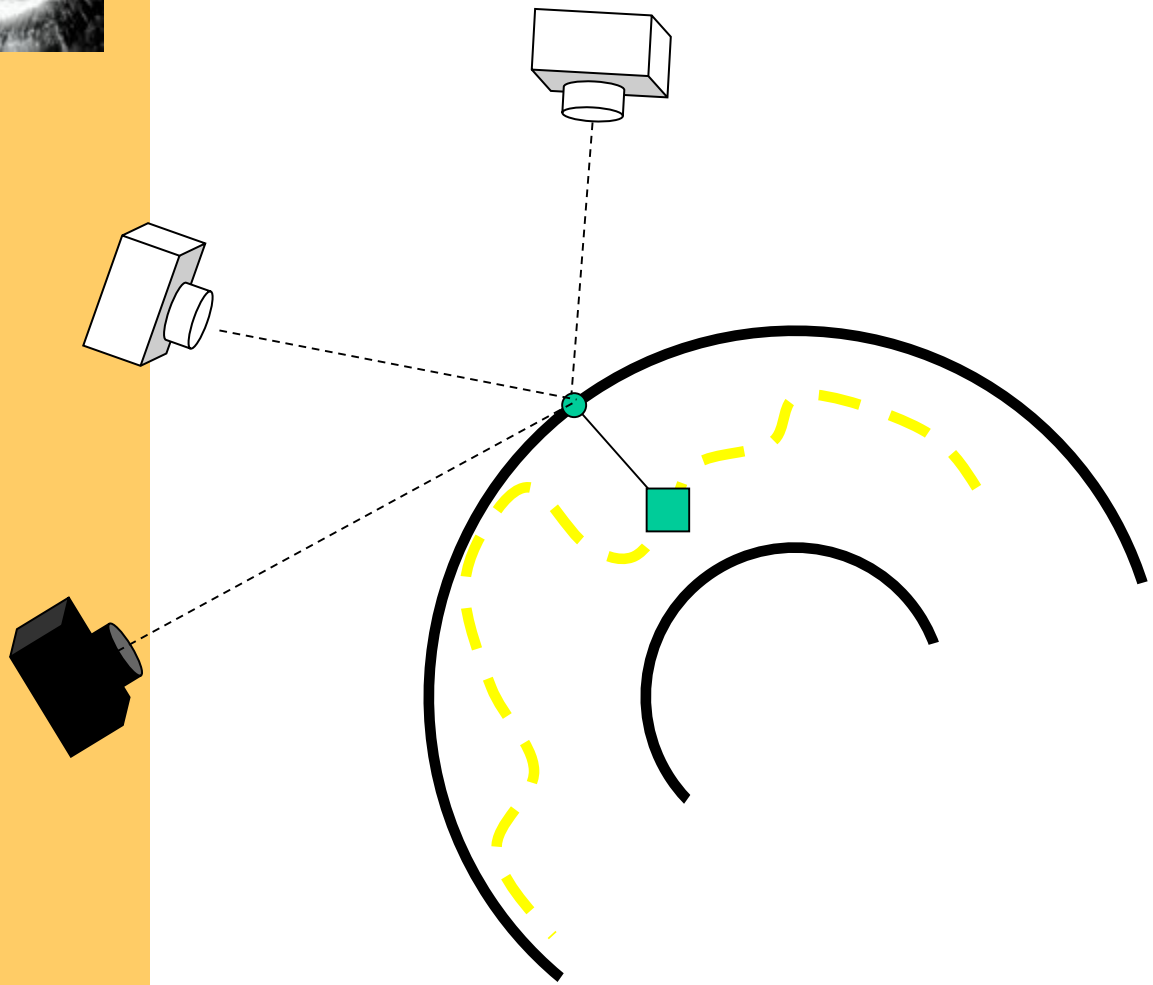


# Photo-consistency



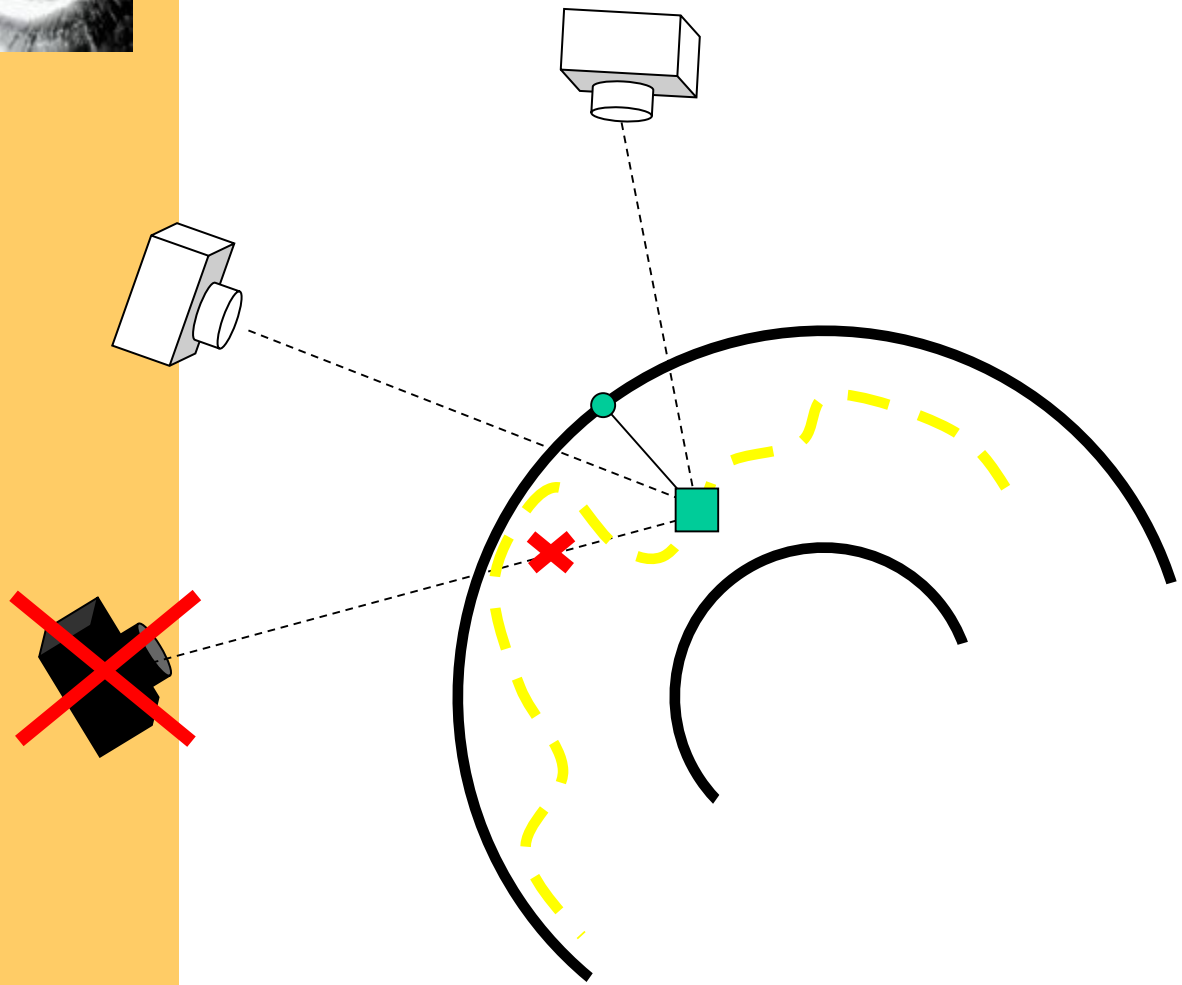
1. Get nearest point on outer surface
2. Use outer surface for occlusions
3. Discard occluded views

# Photo-consistency



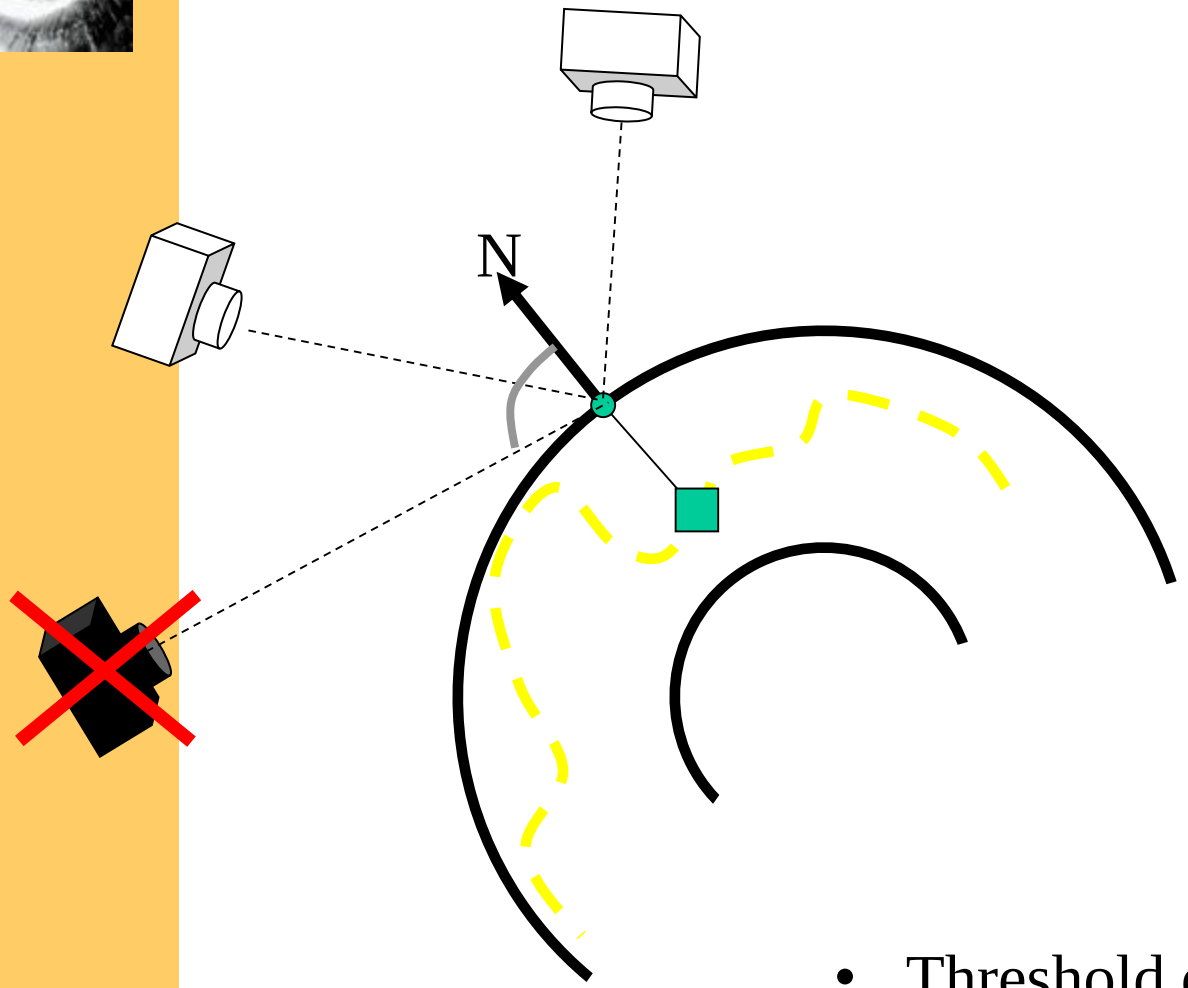
Self occlusion

# Photo-consistency



Self occlusion

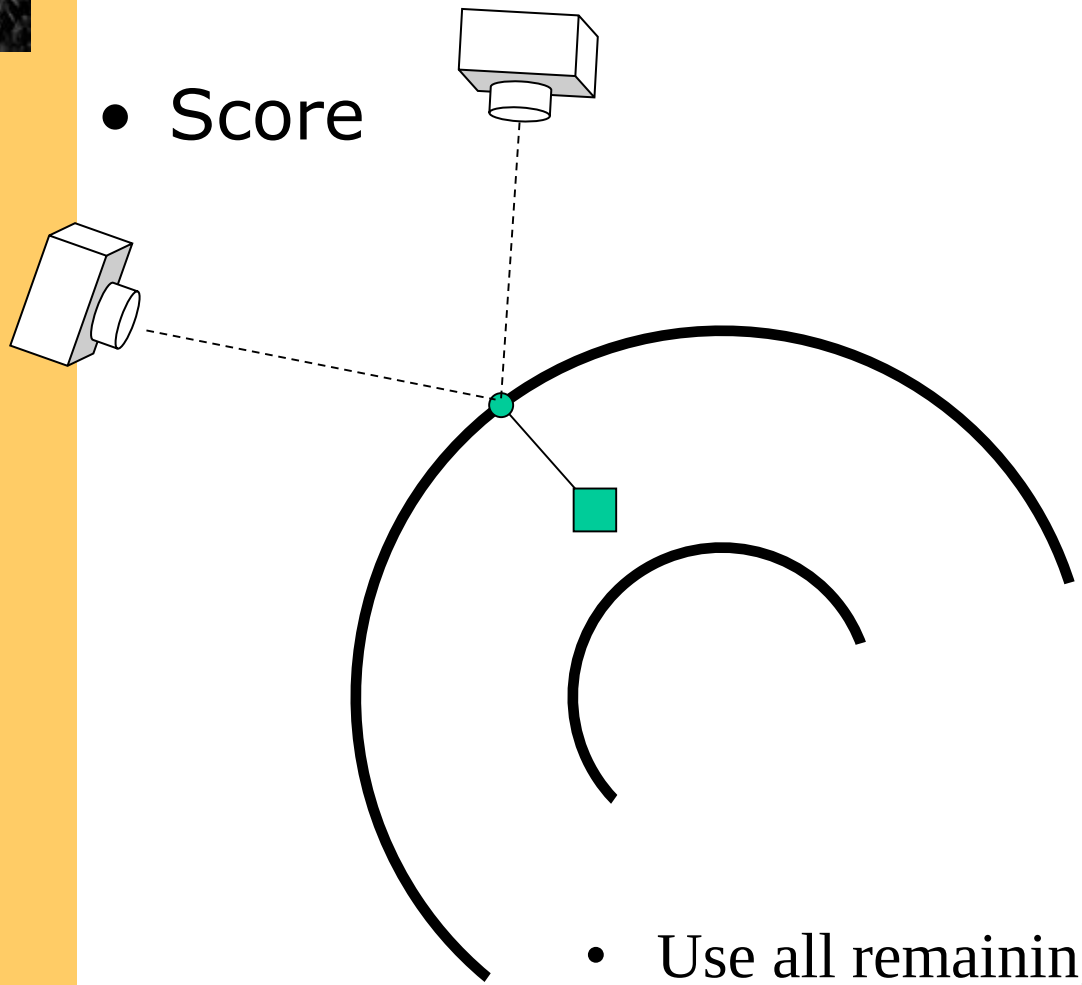
# Photo-consistency




- Threshold on angle between normal and viewing direction

# Photo-consistency

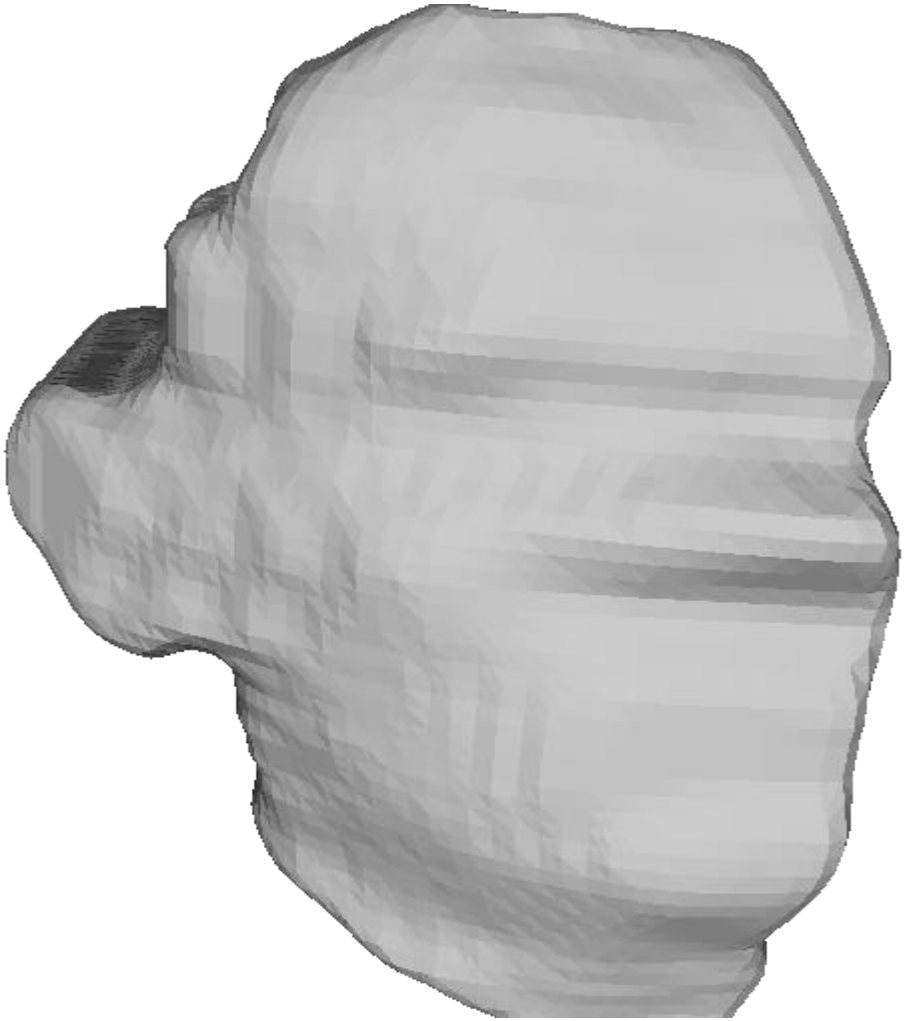
- Score



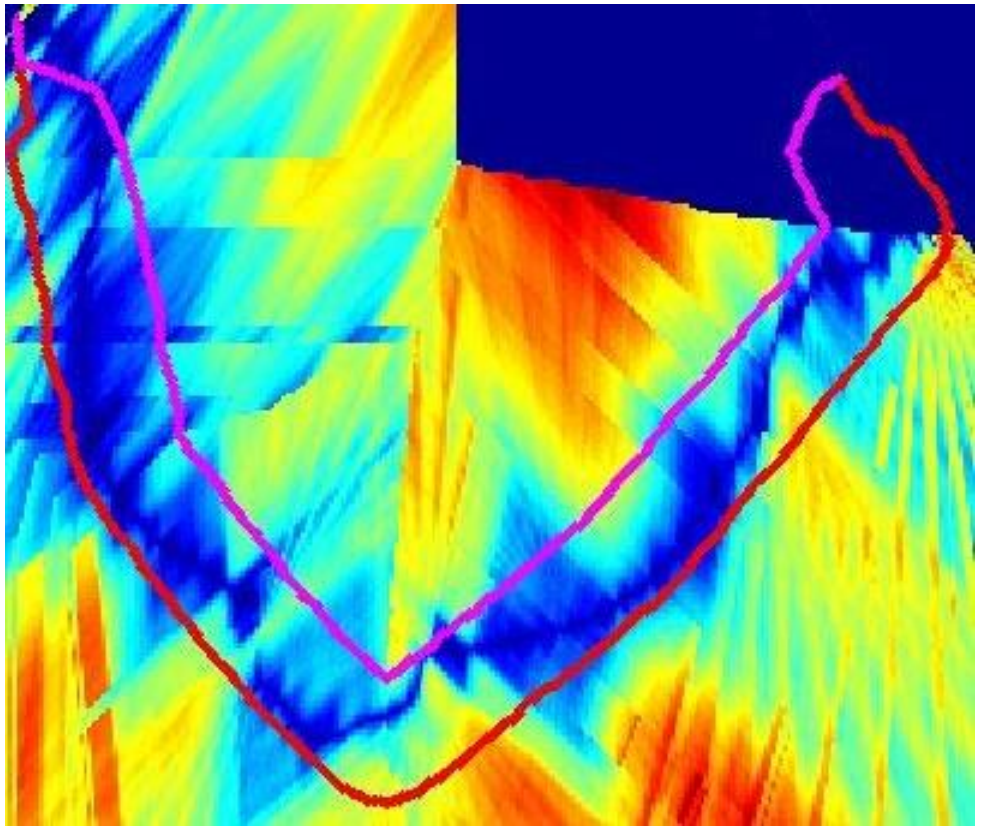
- Use all remaining camera pair wise
- Average all NCC scores

# Example

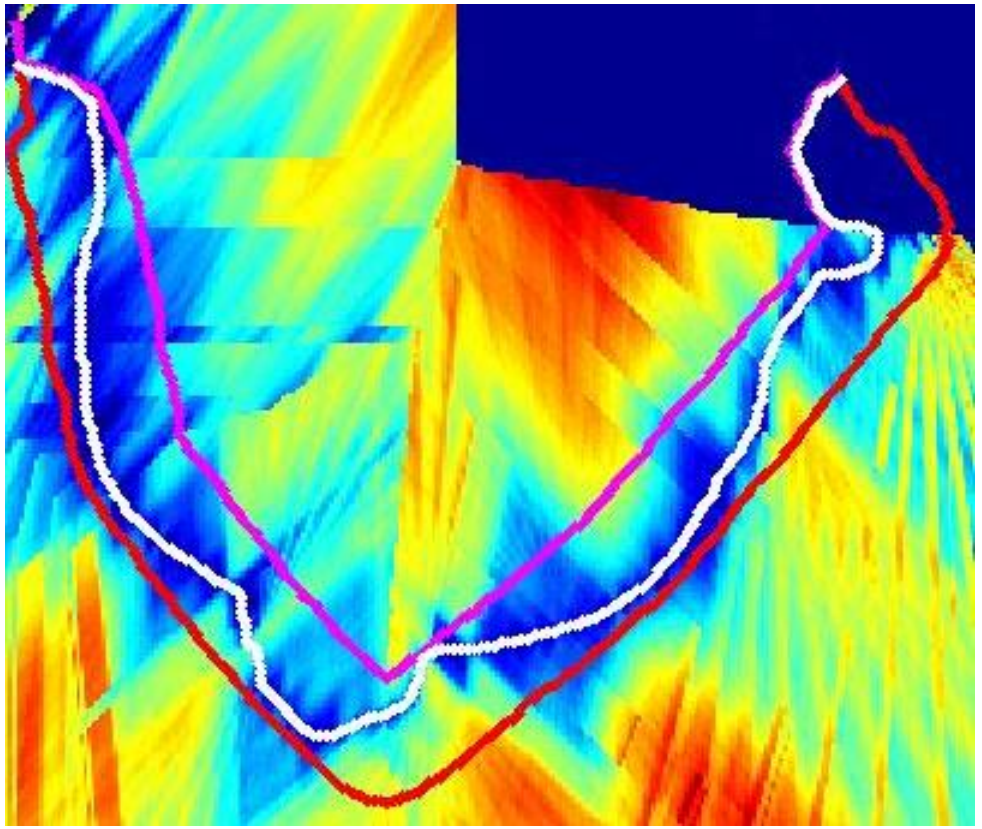
# Example - Visual Hull



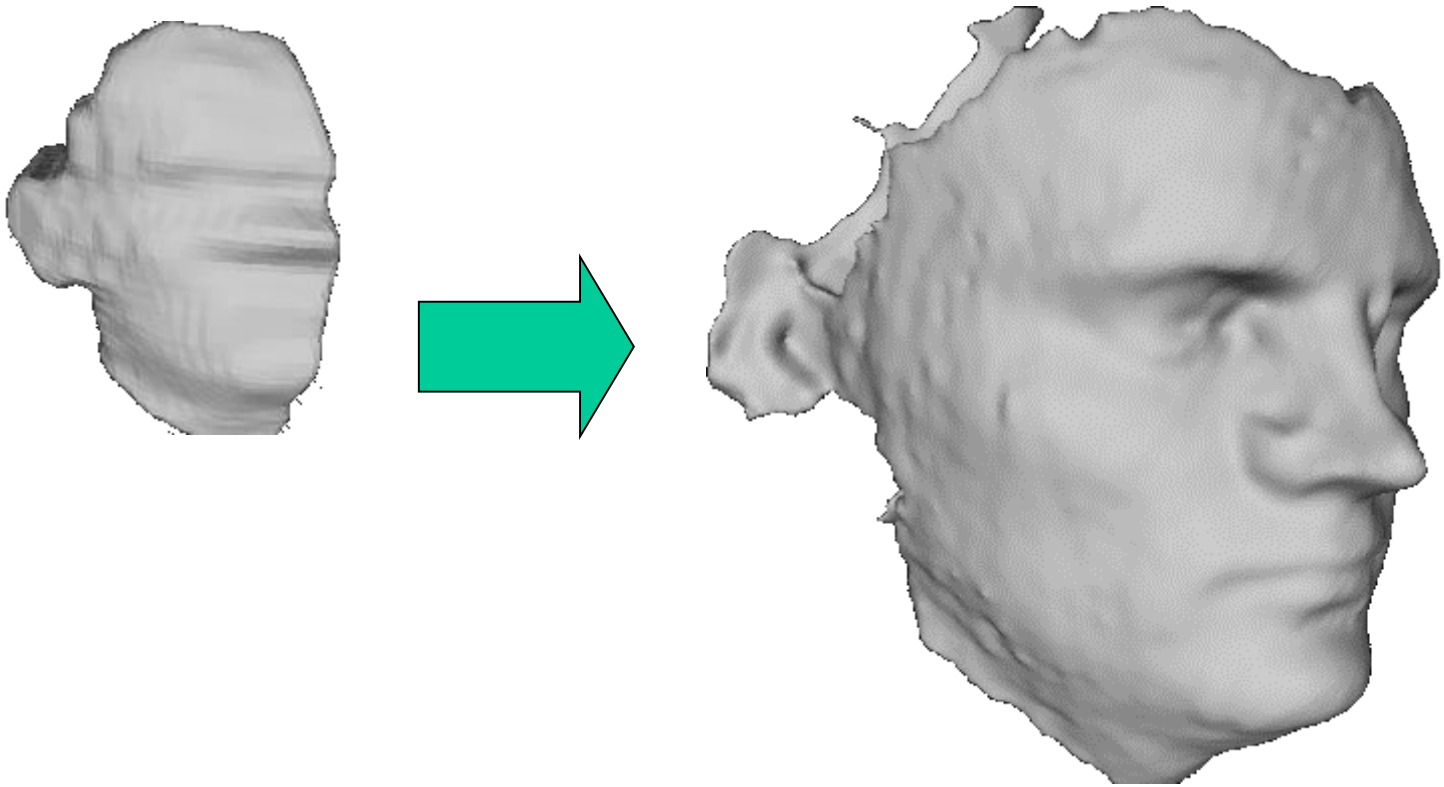
# Example - Slice



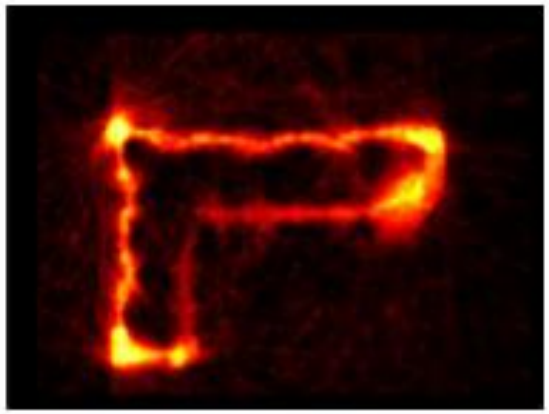
# Example – Graph Cut

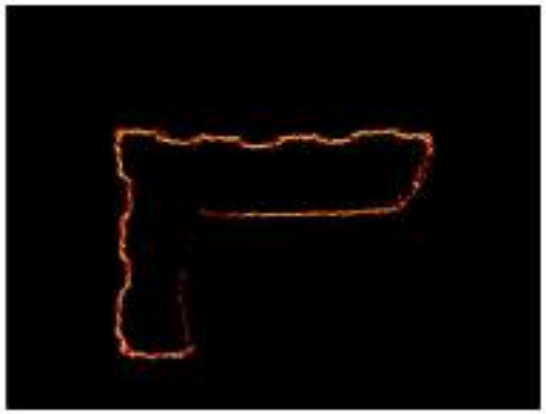


# Example – 3D

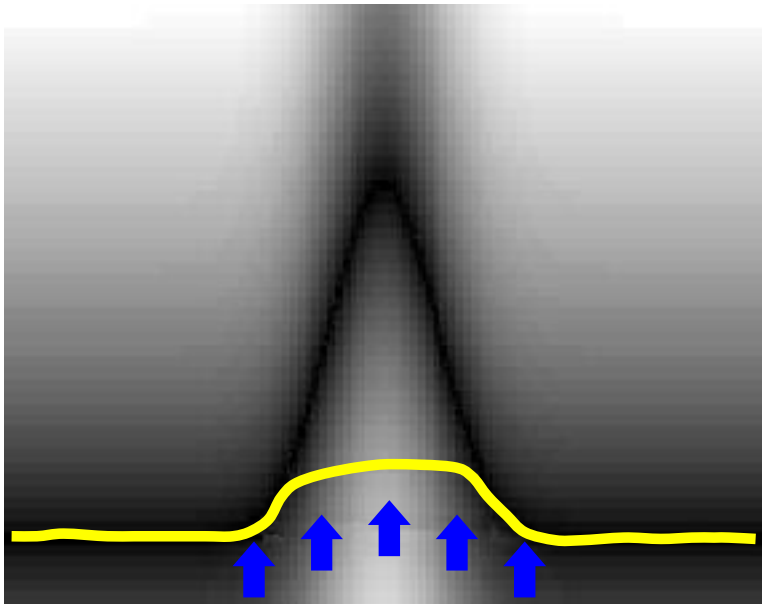


# Example – 3D





# Protrusion problem

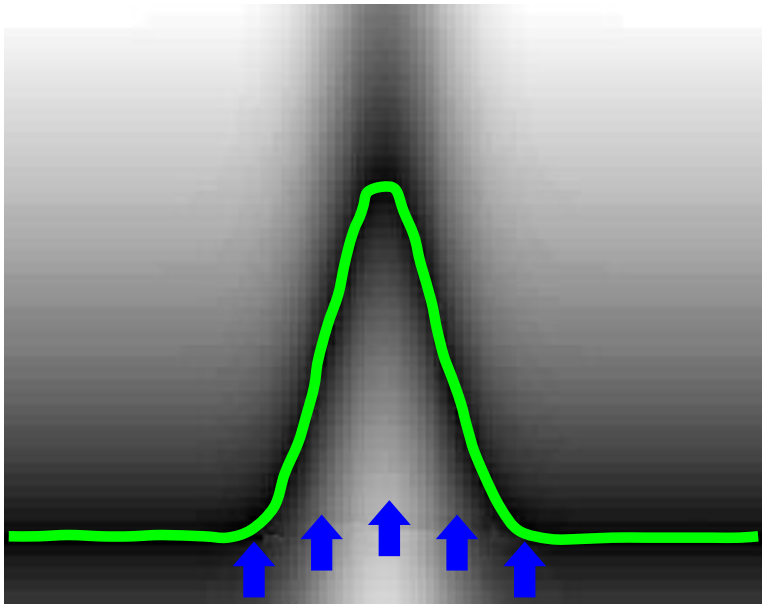


- Minimizing Riemannian surface area may result in removing protrusions
- Total surface integral may be larger even though lower cost per unit area

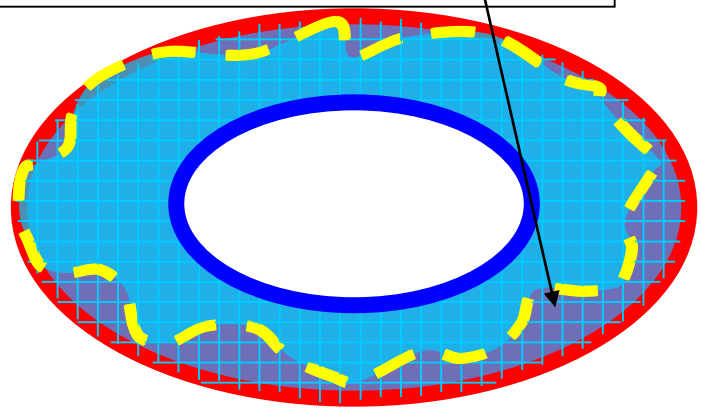
- 'Balooning' force
  - favouring bigger volumes that *fill* the visual hull

L.D. Cohen and I. Cohen. Finite-element methods for active contour models and balloons for 2-d and 3-d images. *PAMI*, 15(11):1131–1147, November 1993.

# Protrusion problem



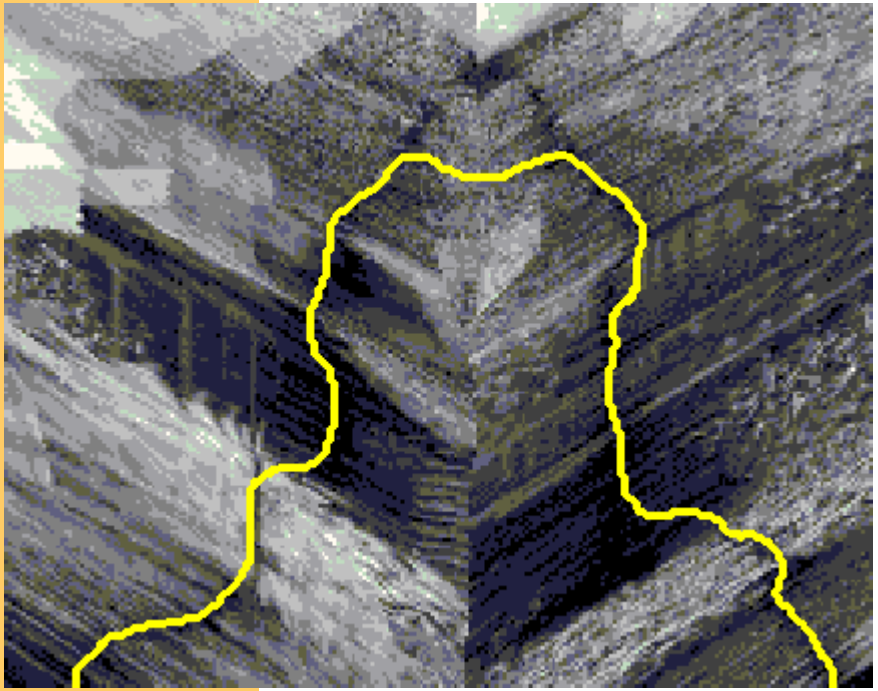
$$\iint_S \rho(x)\, dS - \lambda \iiint_V dV$$

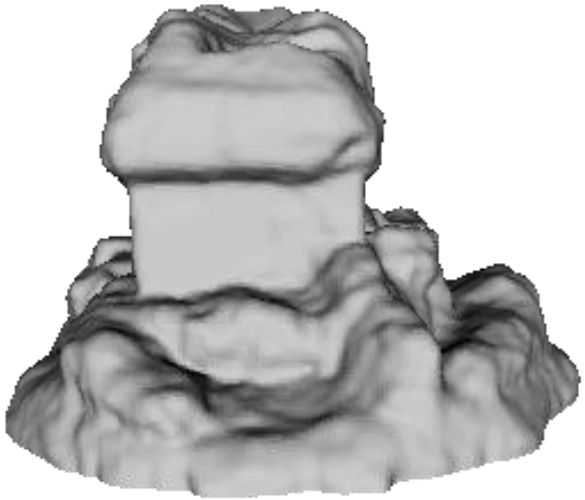


- 'Balooning' force
  - favouring bigger volumes that *fill* the visual hull

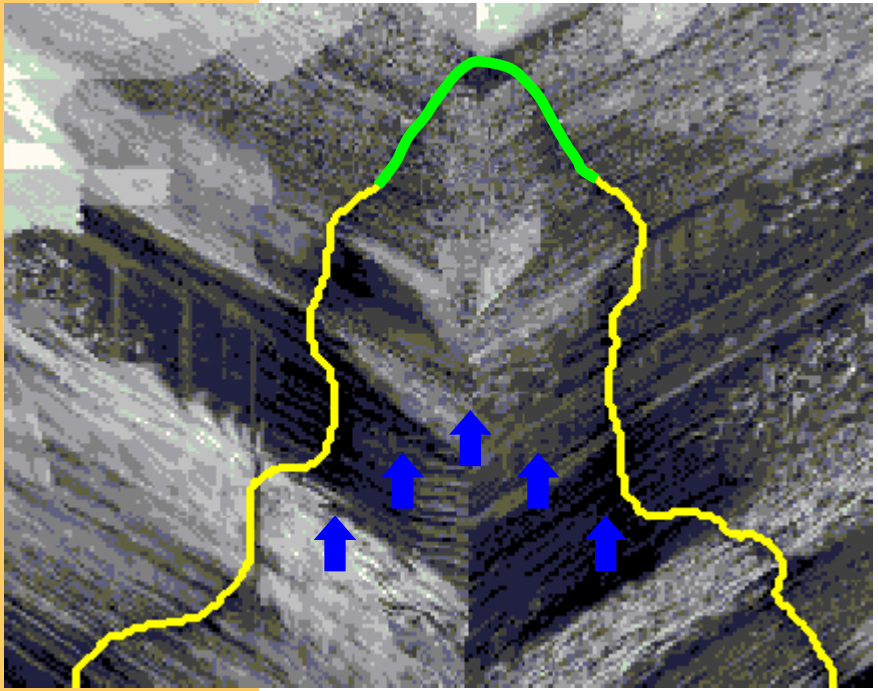
L.D. Cohen and I. Cohen. Finite-element methods for active contour models and balloons for 2-d and 3-d images. *PAMI*, 15(11):1131–1147, November 1993.

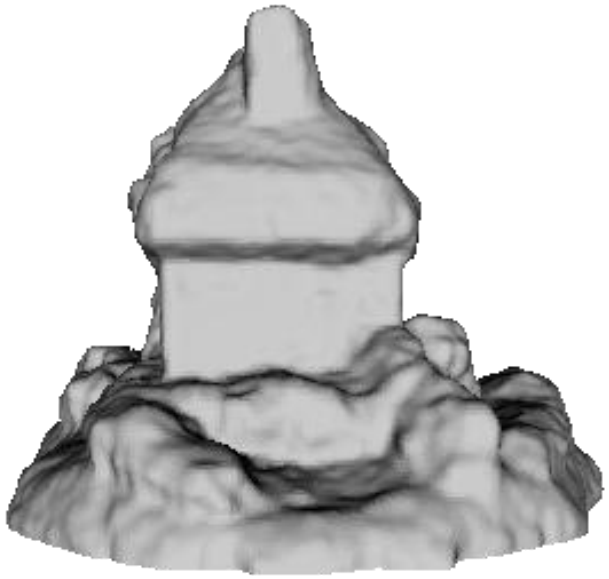
# Protrusion Problem





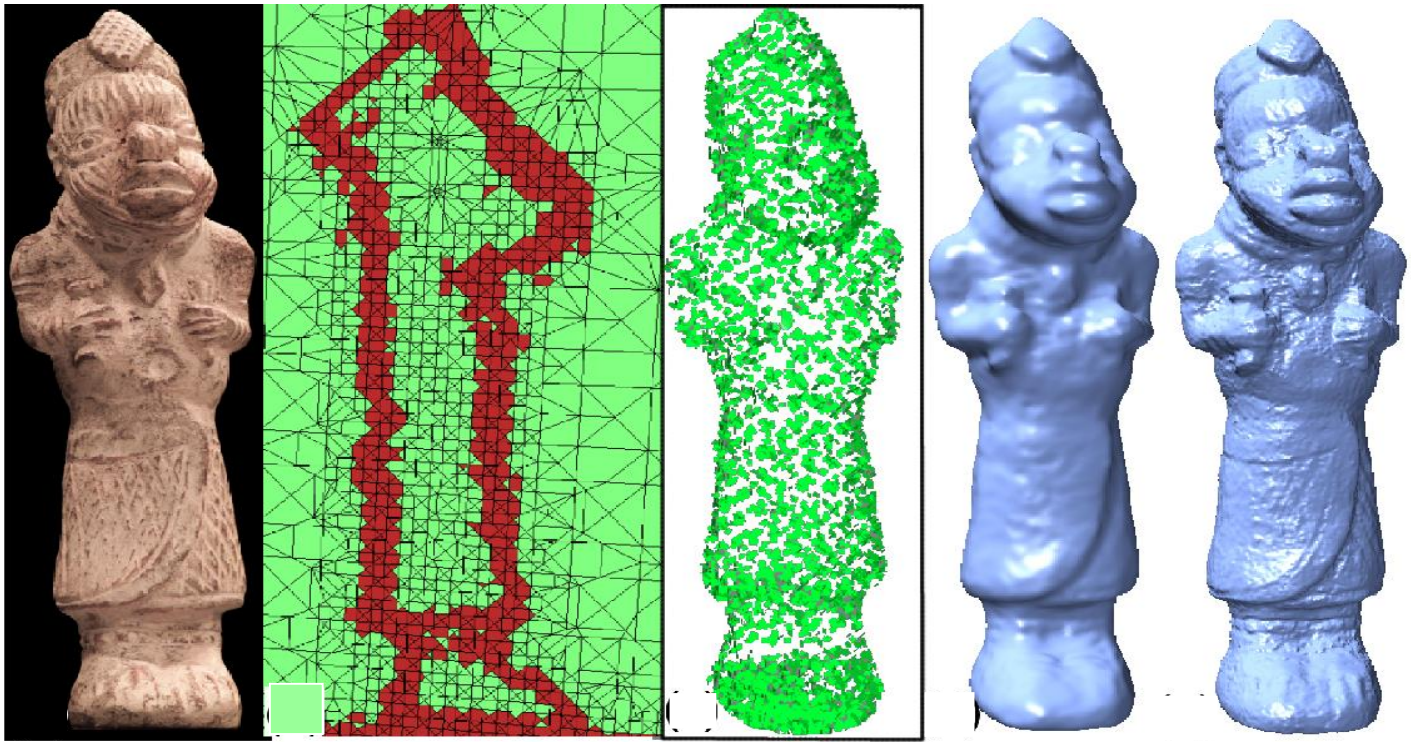
# Protrusion Problem





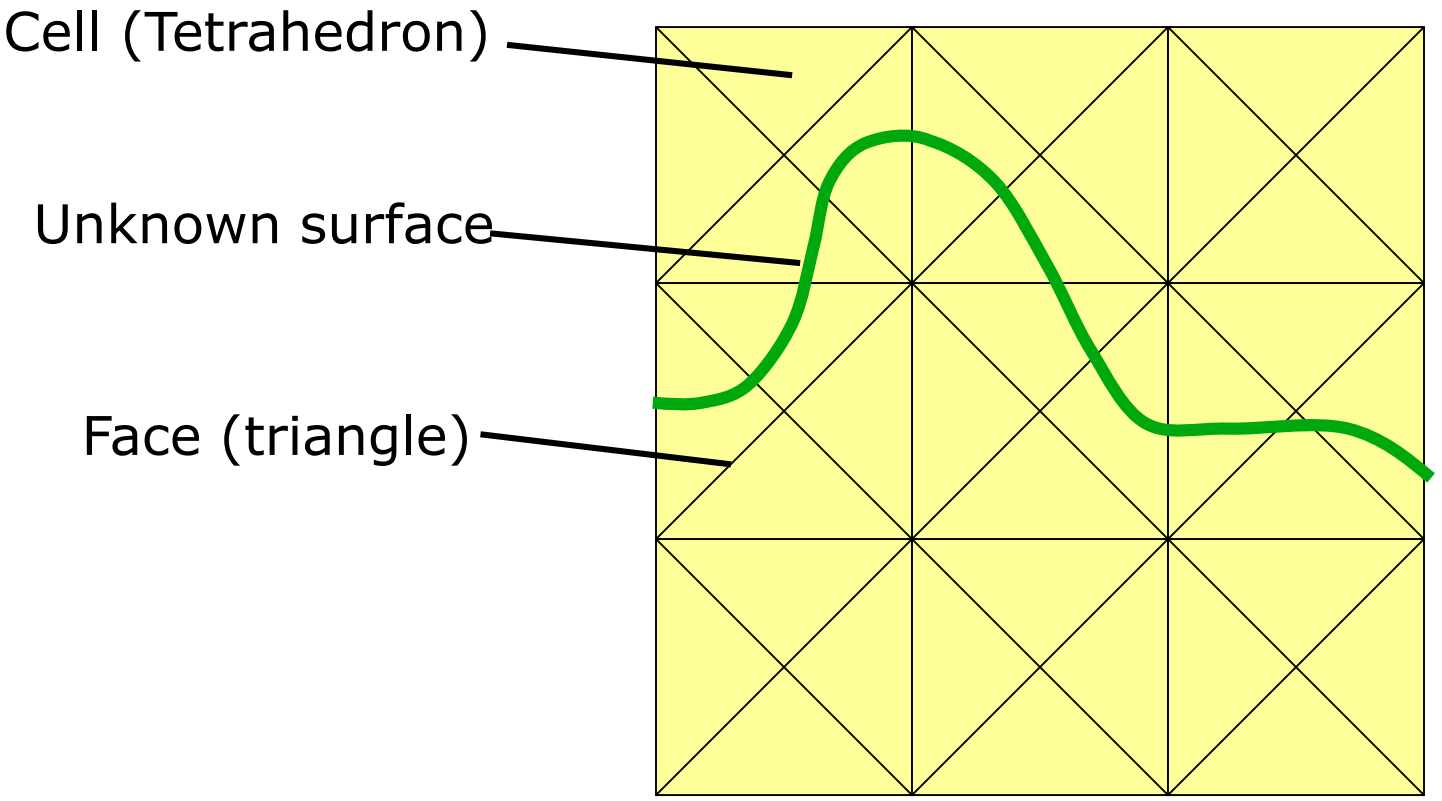
# Graph Cut on Tetrahedral Mesh

[Sinha et. al. 2007]



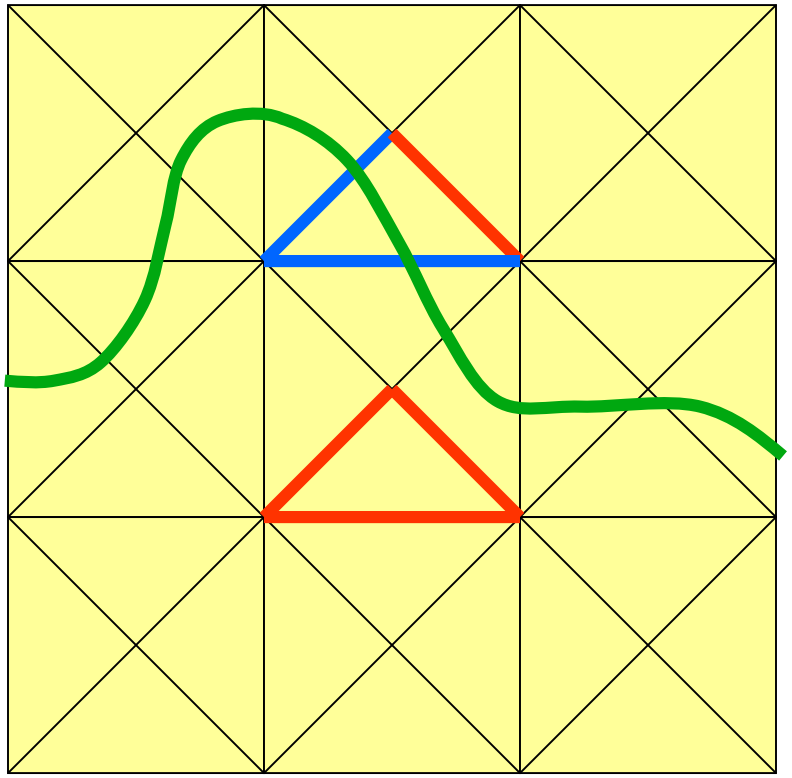
- Voxelized Graph Cut approaches don’t scale well!
- Compute Photo-consistency only where it is needed
- Use adaptive tetrahedral mesh representation

# Adaptive Mesh Refinement



# Adaptive Mesh Refinement

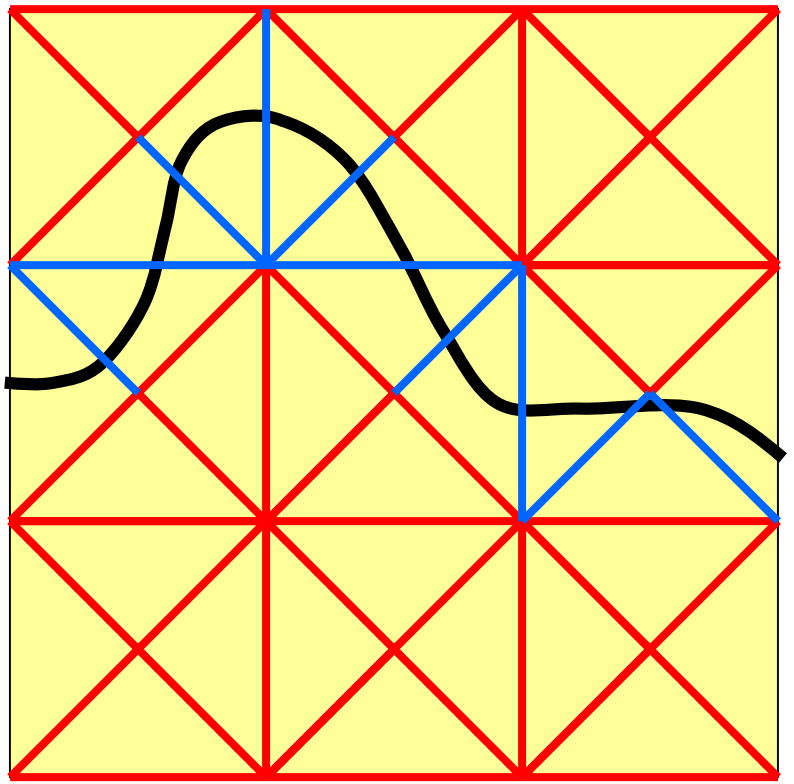
Detect *crossing faces* by testing photo-consistency of points sampled on the faces



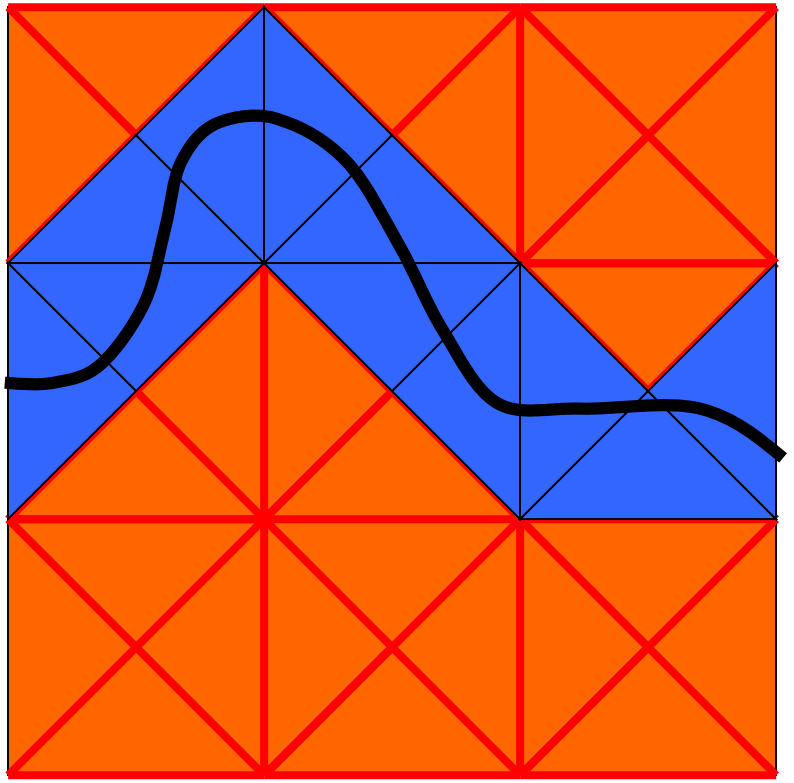
# Adaptive Mesh Refinement

If none of the faces of a cell are *crossing faces*,

that cell cannot contain any surface element.

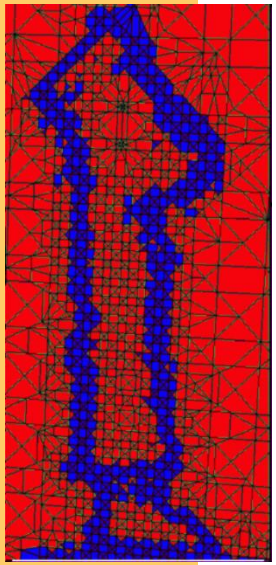


# Adaptive Mesh Refinement



- Prune the *Inactive cells*
- Refine the *Active cells*

# Mesh with Photo Consistency



Final Mesh

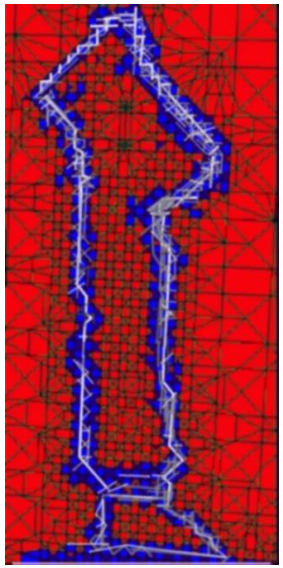
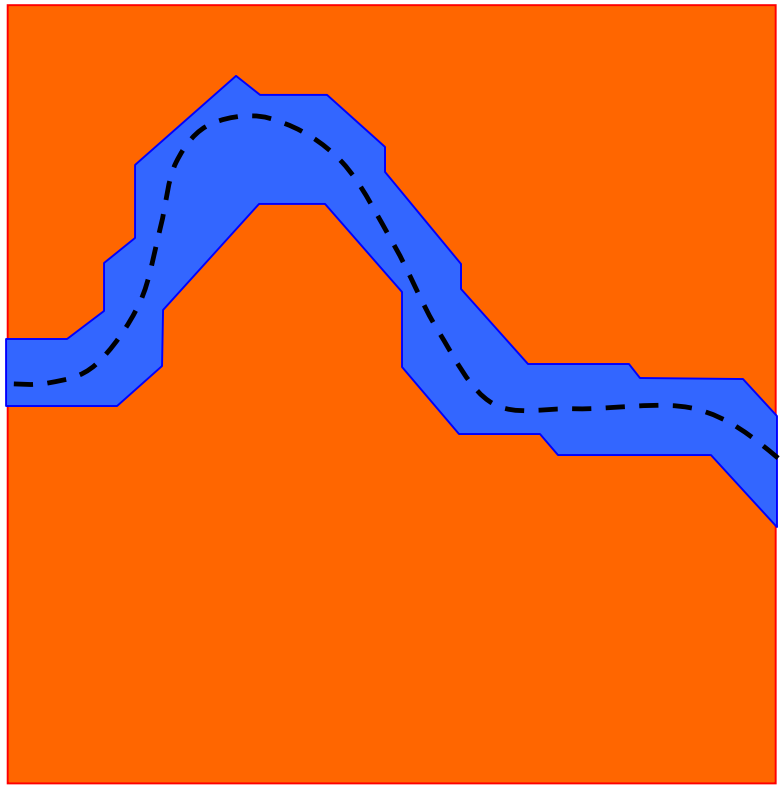
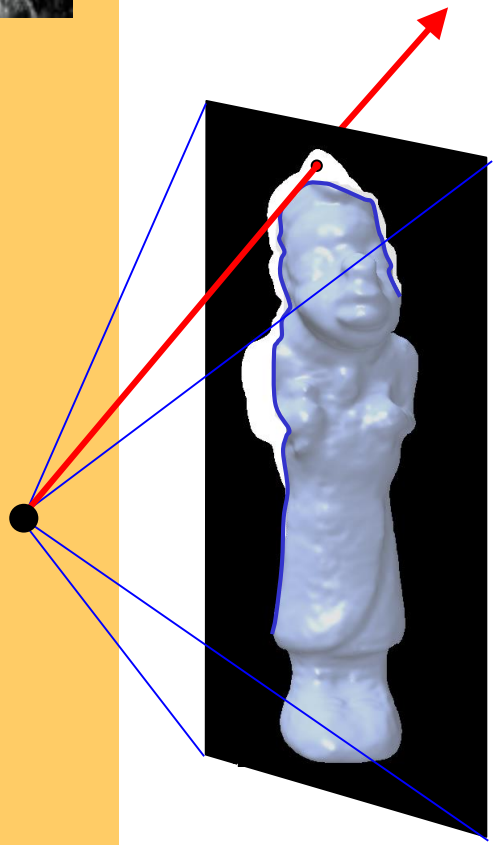
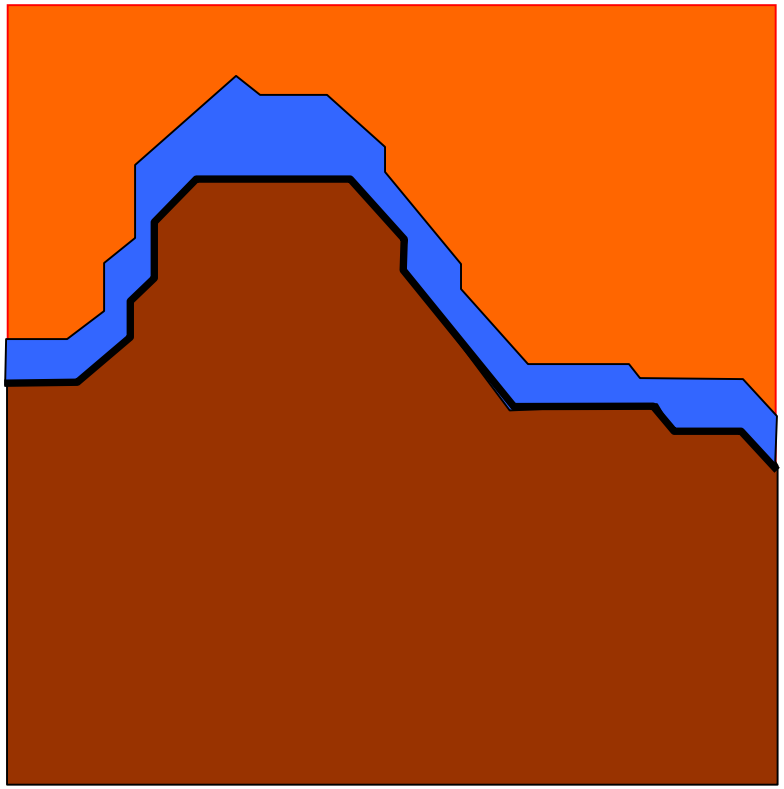


Photo-consistency



# Silhouette Constraints



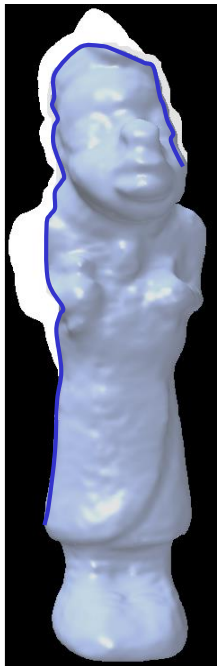


Every such ray must meet the real surface at least once

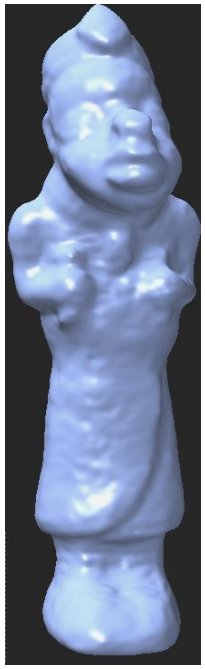


140

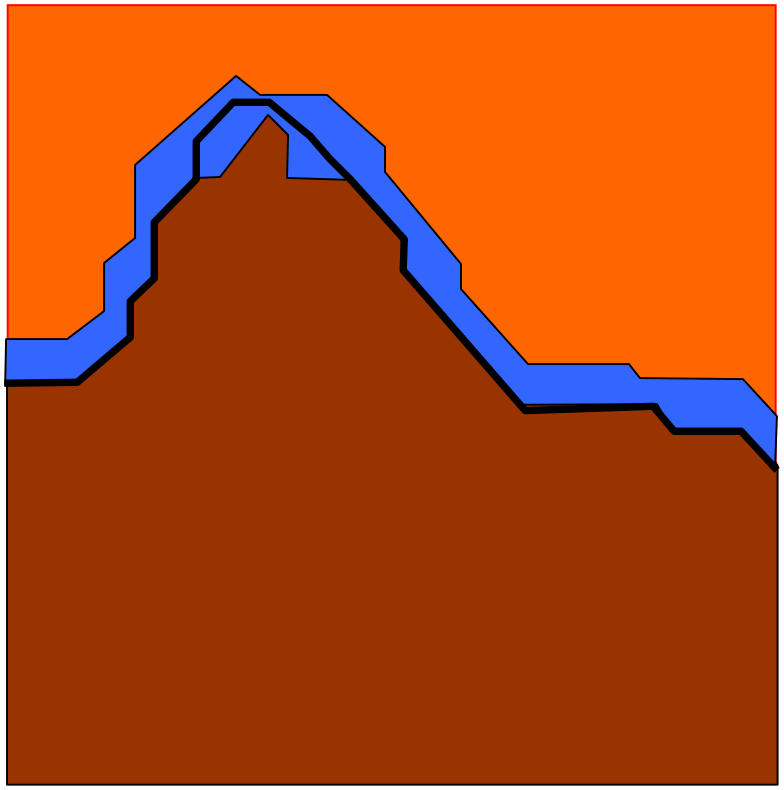
# Graph Cut



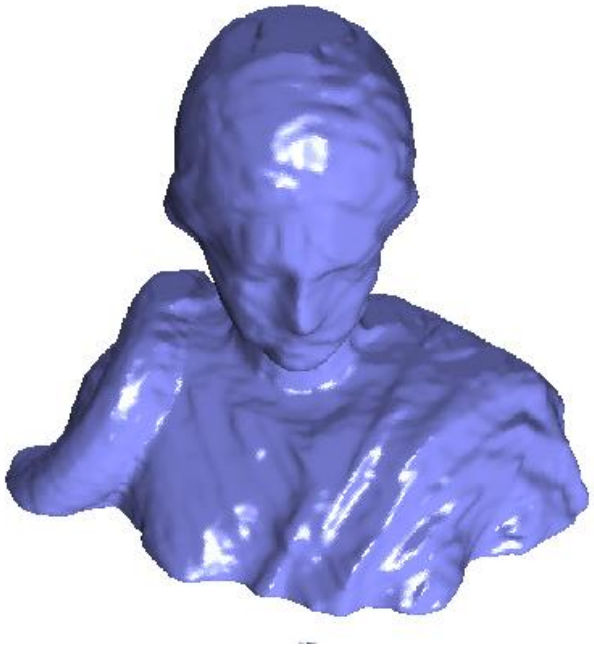
Before

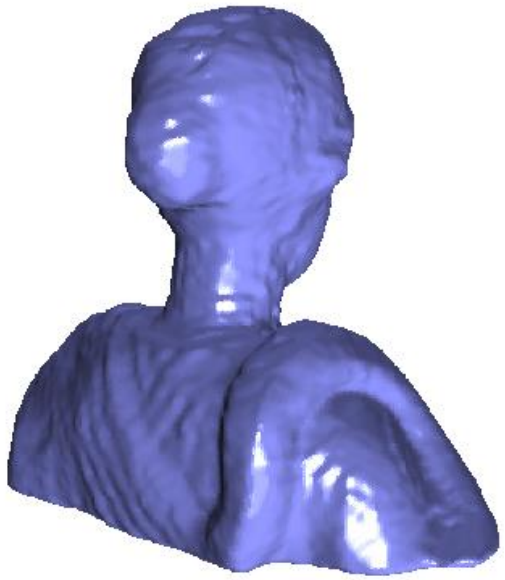


After

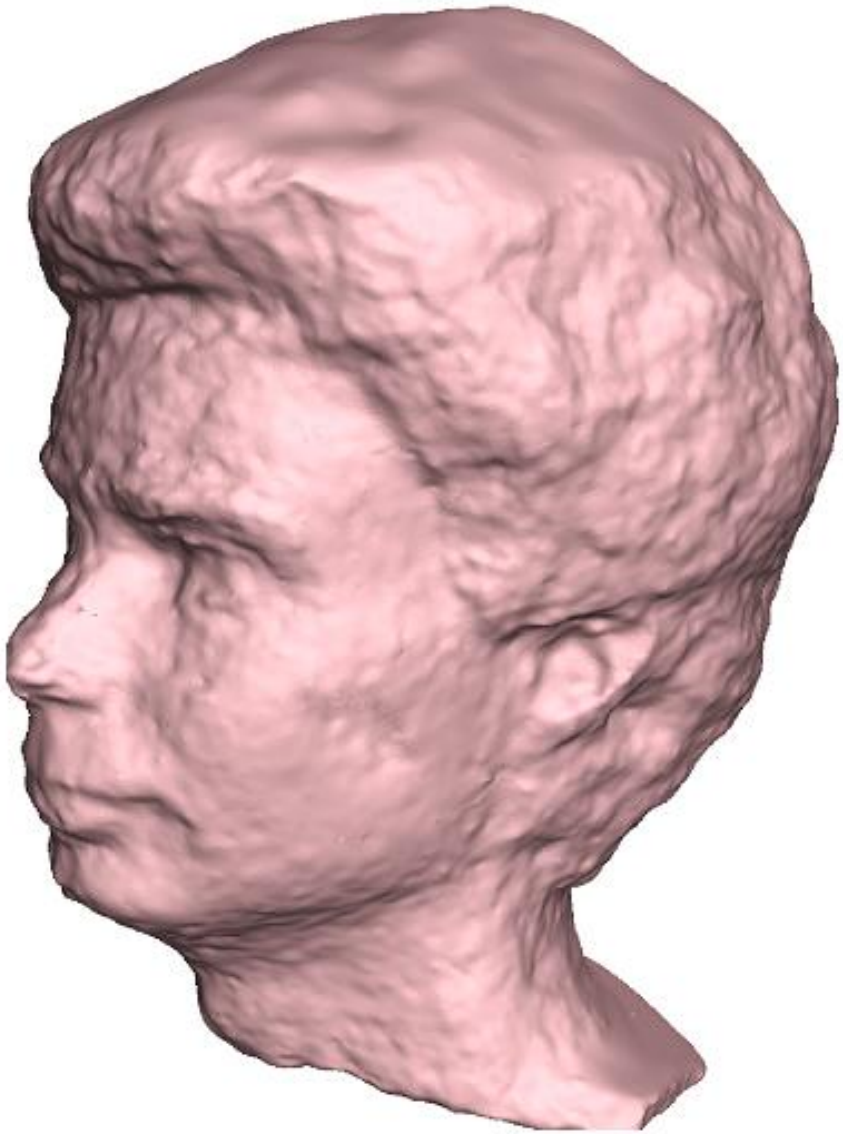


# Results





# 20 images, 640 x 480 pixels

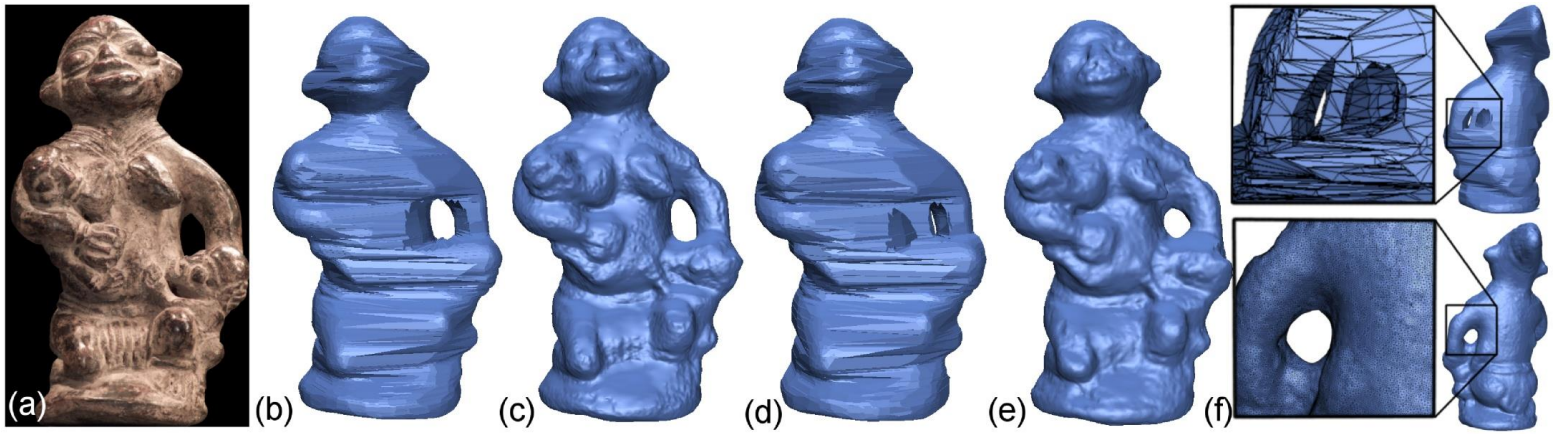


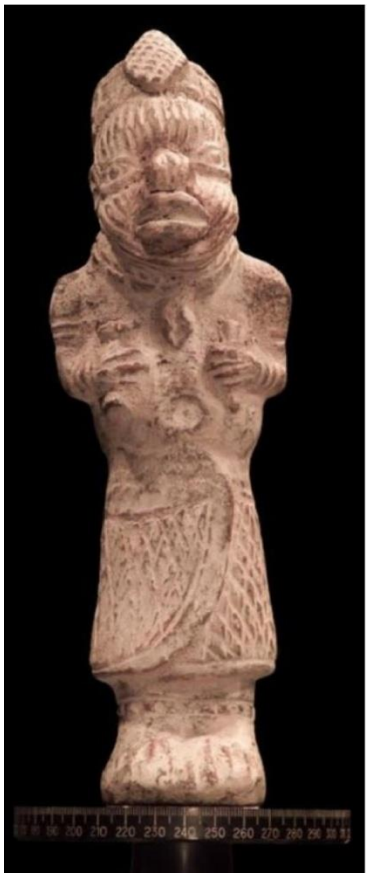
**Running Time:**

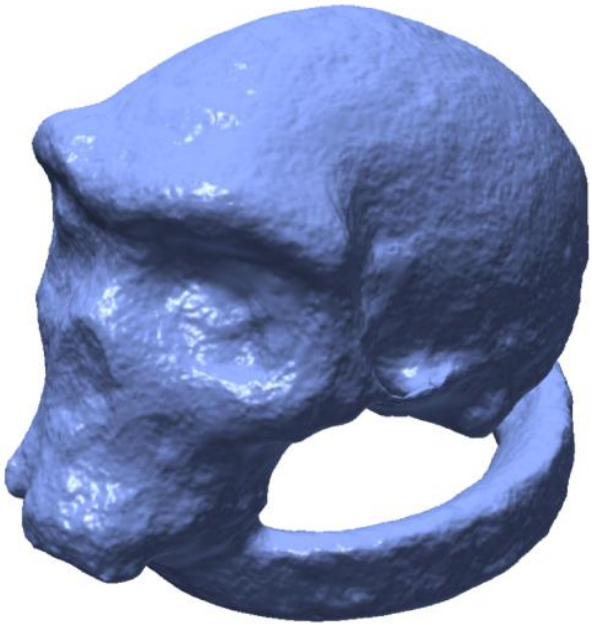
Graph Construction:
25 mins

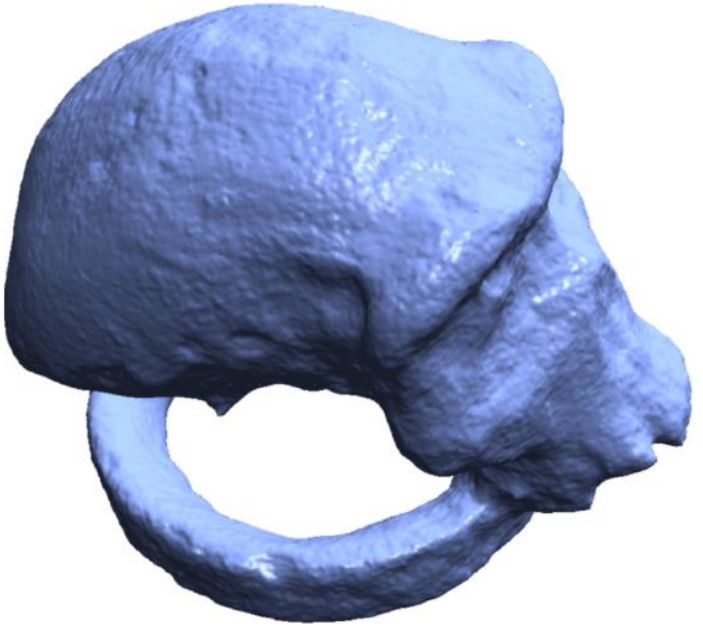
Graph-cut:
5 mins

Local Refinement:
20 mins

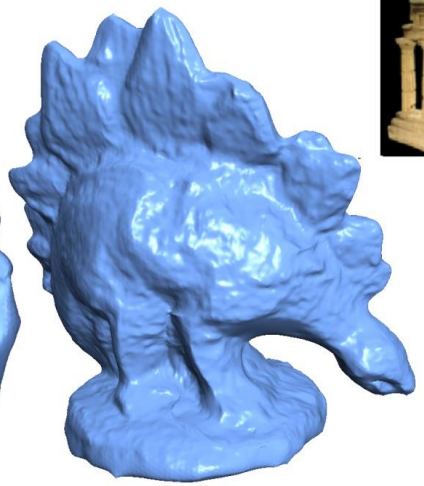

(a)
(b)
(c)
(d)
(e)
(f)

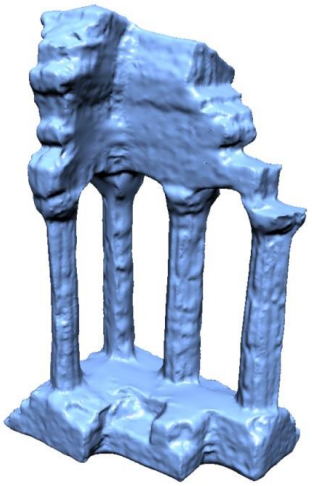




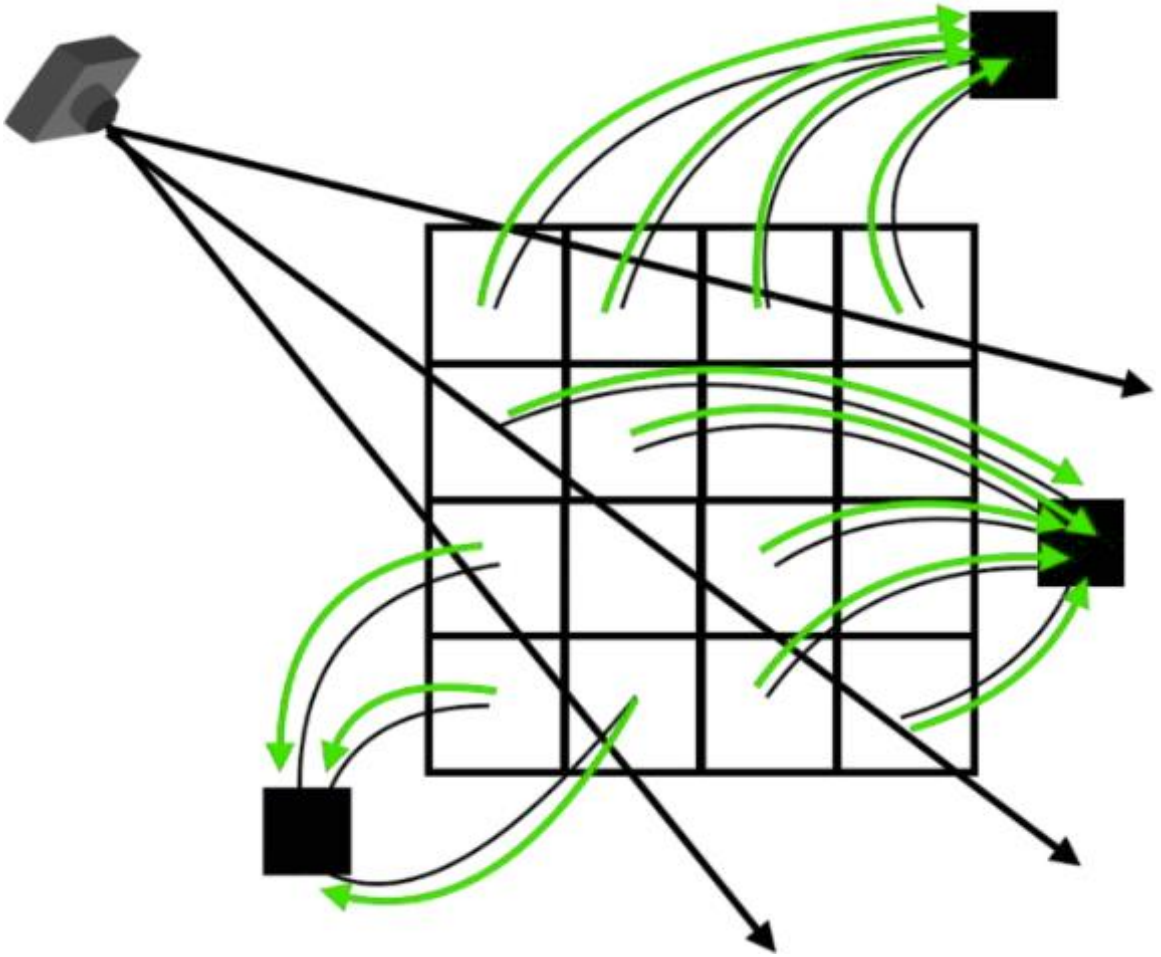








# Ray Potentials

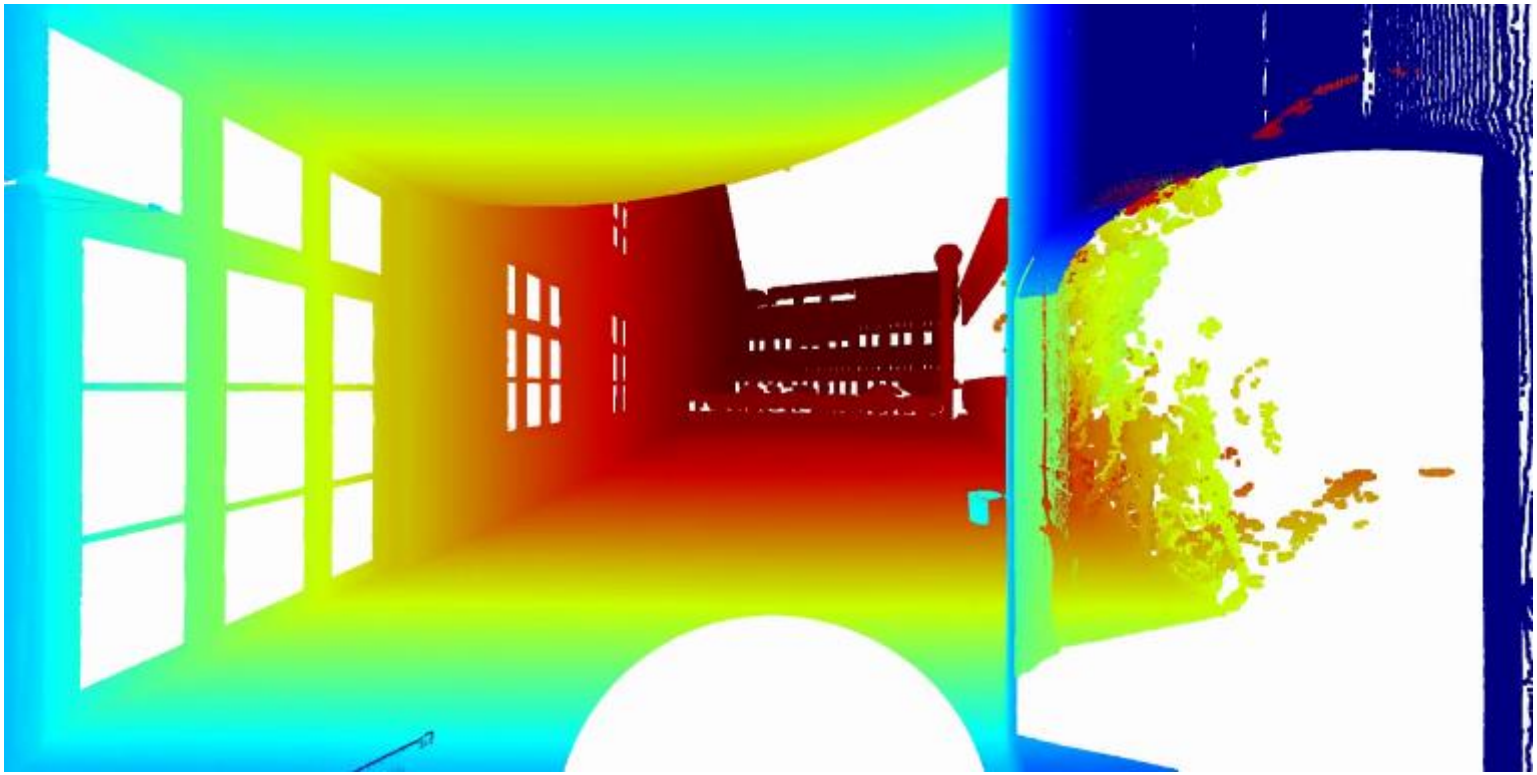


[Ulusoy et al., 2015/2016]
[Savinov et al., 2015/2016]

# Comparison of multi-view stereo approaches

http://vision.middlebury.edu/mview/

# ETH 3D MVS Benchmark



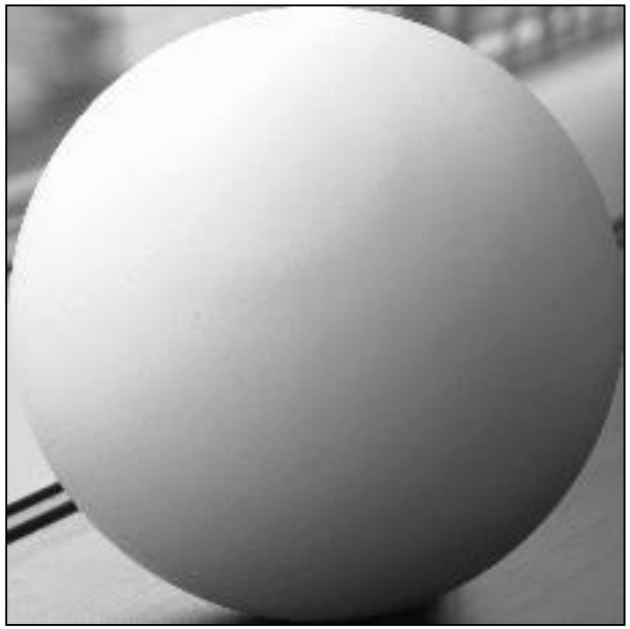
https://www.eth3d.net/

# Shape-from-X


# Shape-from-X

- X = Shading
- X = Multiple Light Sources (photometric stereo)
- X = Texture
- X = Focus/Defocus
- X = Specularities
- X = Shadows
- X = …

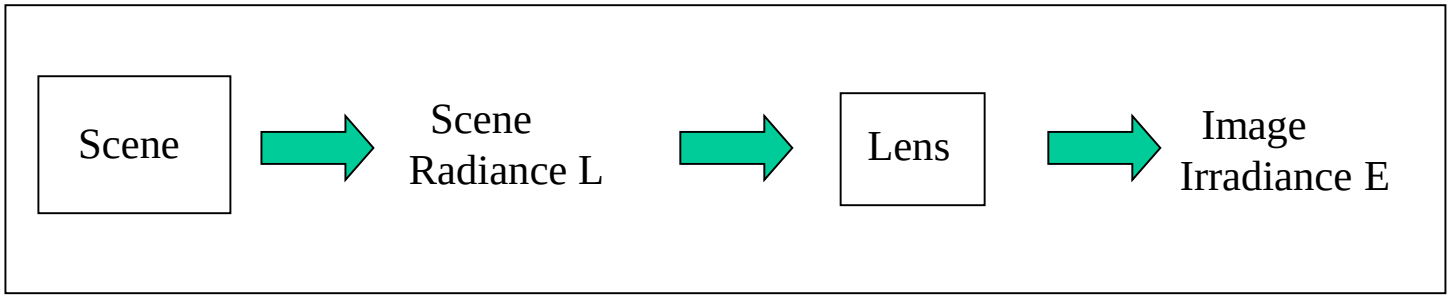
# Shape from Shading



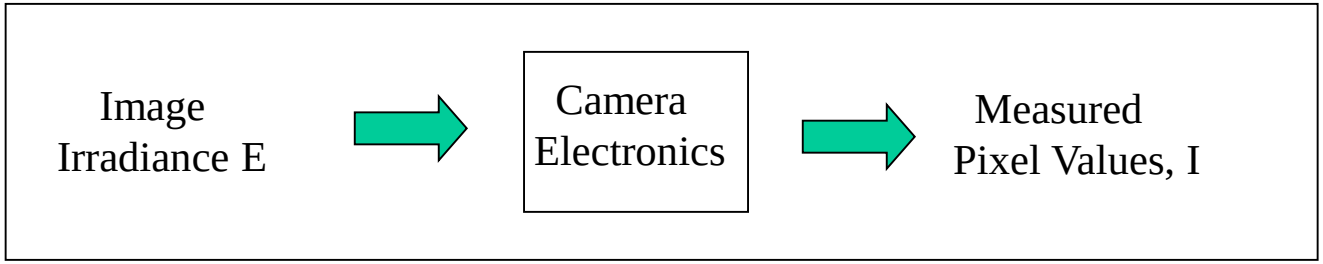
- Shading as a cue for shape reconstruction
- What is the relation between intensity and shape?
  - Reflectance Map
  - ie, how much light does reflect back under a given surface orientation

# Scene vs. Image Brightness

- Before light hits the image plane:




- After light hits the image plane:



# Reflectance Map

- Relates image irradiance I(x,y) to surface orientation **n** for given source direction and surface reflectance
- Lambertian case:

$k$ : source brightness

$\rho$ : surface albedo (reflectance)

$c$ : constant (optical system)

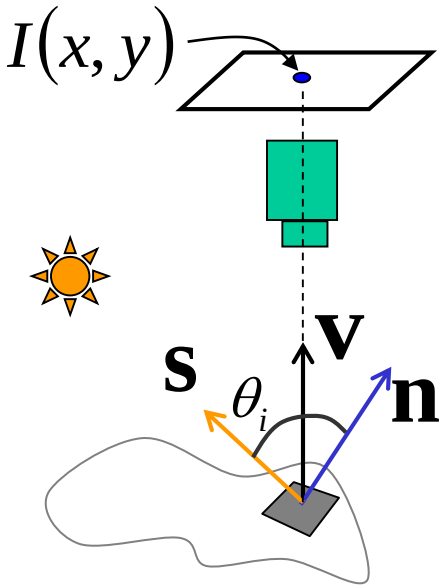



Image irradiance:

$$I = \frac{\rho}{\pi} kc \cos\theta_i = \frac{\rho}{\pi} kc\mathbf{n}\cdot\mathbf{s}$$

Let $\frac{\rho}{\pi} kc = 1$ then $\boxed{I = \cos\theta_i = \mathbf{n}\cdot\mathbf{s} = R(\mathbf{n})}$

# Reflectance Map

- Lambertian case
- Let (p,q) denote the negative gradients of the depth map z: $(p,q) = (-z_x, -z_y)$
- Surface normal: $\mathbf{n} = (p,q,1) / \sqrt{1 + p^2 + q^2}$
- The reflectance map is related to (p,q) as

$$I = \cos\theta_i = \mathbf{n} \cdot \mathbf{s} = \frac{(ps_x + qs_y + s_z)}{\sqrt{p^2 + q^2 + 1}} = R(p,q)$$

Reflectance Map

# Reflectance Map

- Lambertian case
- Let (p,q) denote the negative gradients of the depth map z: $(p,q) = (-z_x, -z_y)$
- Surface normal: $\mathbf{n} = (p,q,1) / \sqrt{1 + p^2 + q^2}$
- The reflectance map is related to (p,q) as

$$I = \cos\theta_i = \mathbf{n} \cdot \mathbf{s} = \frac{\left(ps_x + qs_y + s_z\right)}{\sqrt{p^2 + q^2 + 1}} = R\left(p,q\right)$$

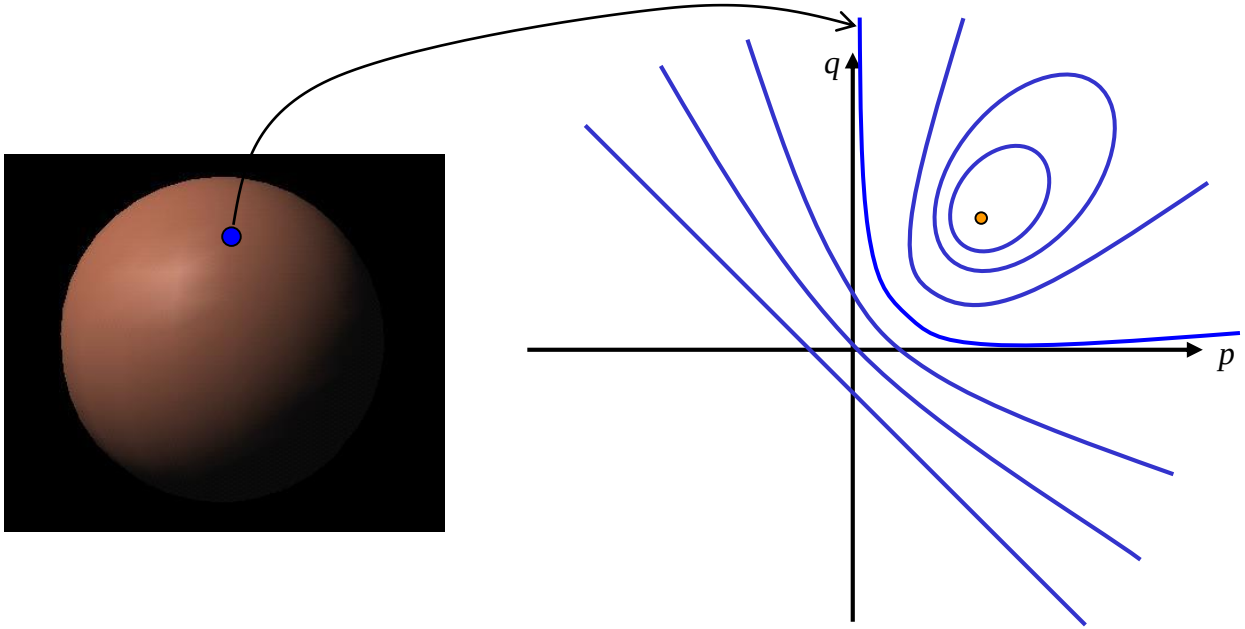
- Given a single image of an object with known surface reflectance taken under a known light source, can we recover the shape of the object?

- Given *R and* **s**, can we determine **n** uniquely for each image point?

# Shape from a Single Image?

- Given a single image of an object with known surface reflectance taken under a known light source, can we recover the shape of the object?

- Given *R and* **s**, can we determine *(p,q)* uniquely for each image point?



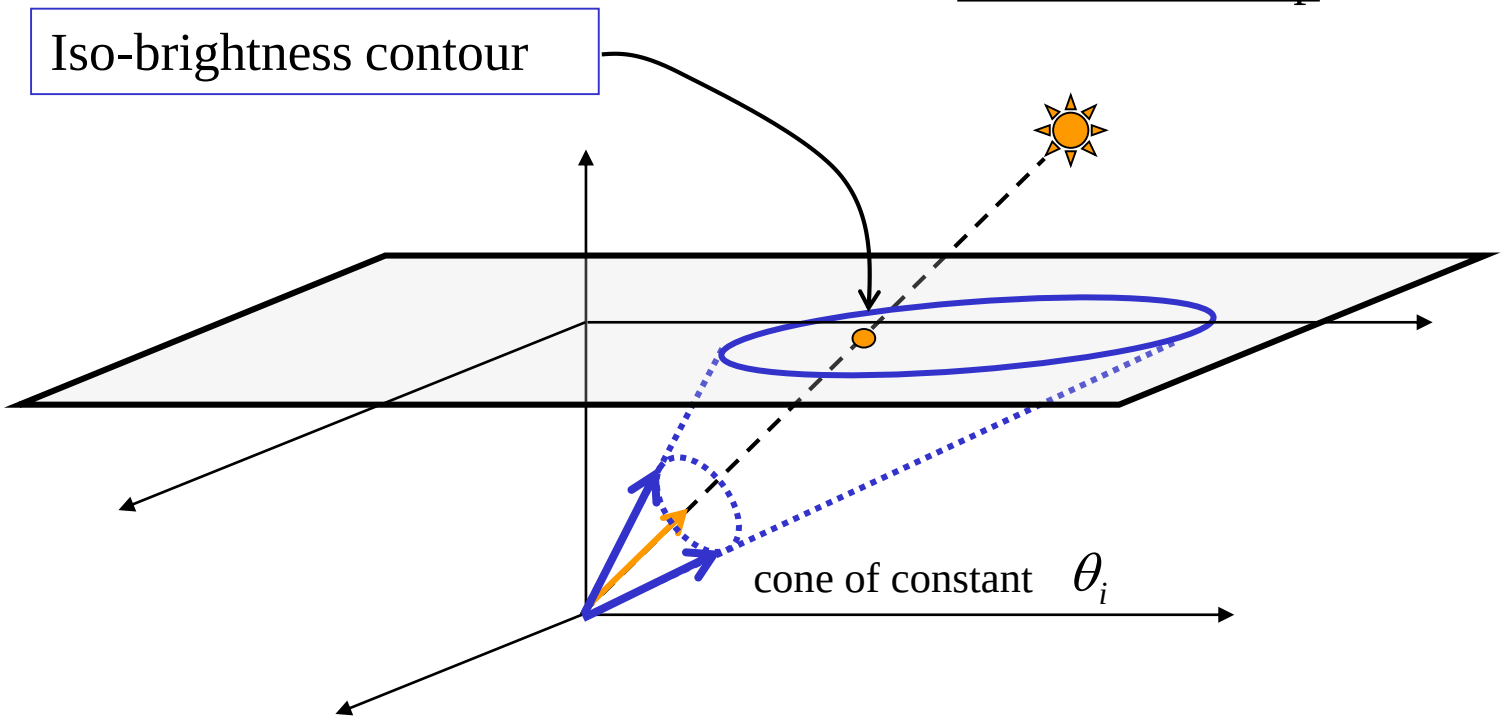

- Given R=I, which surfaces are possible?

# Reflectance Map

- Lambertian case

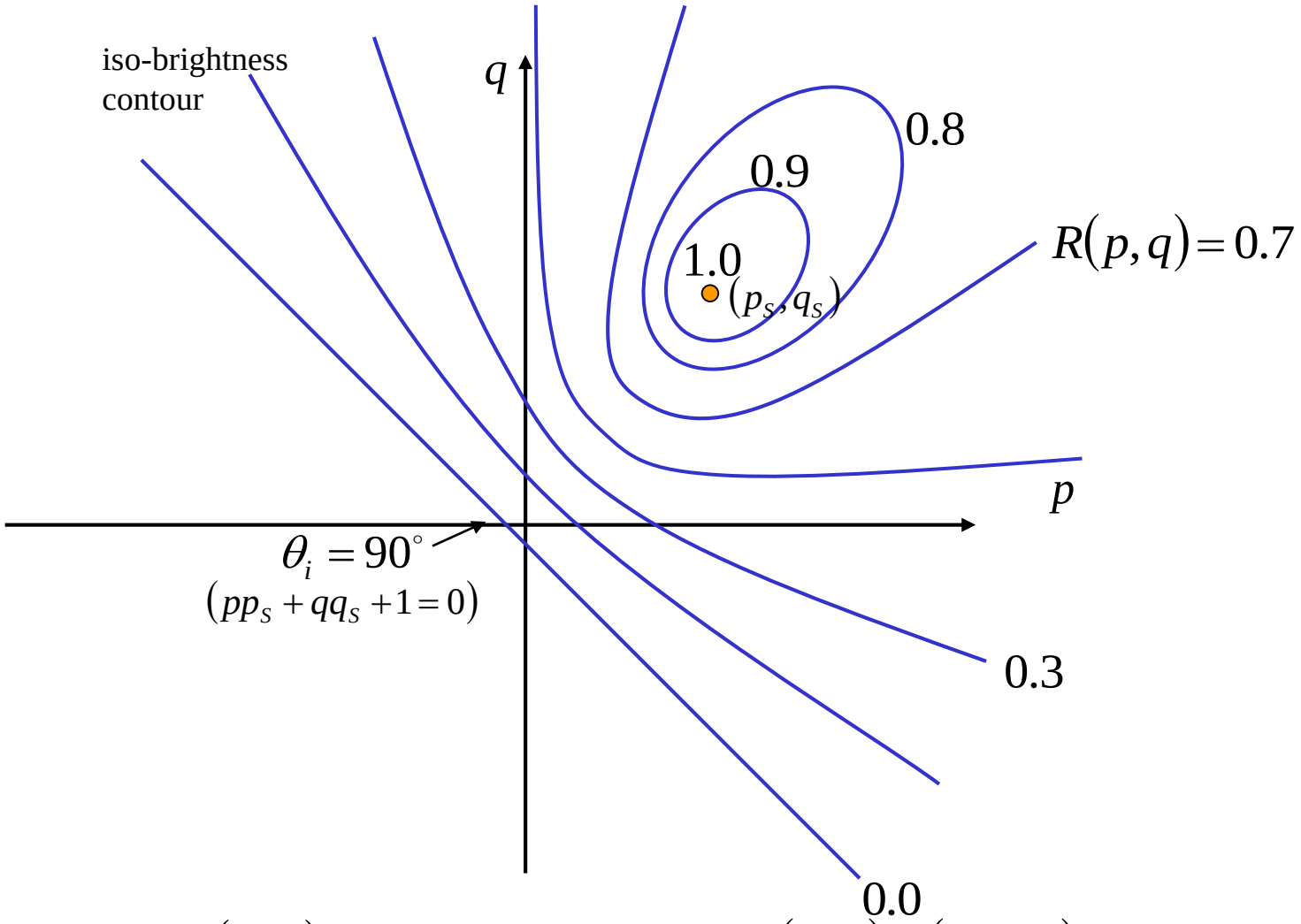
$$I = \cos\theta_i = \mathbf{n}\cdot\mathbf{s} = \frac{\left(ps_x + qs_y + s_z\right)}{\sqrt{p^2 + q^2 + 1}} = R(p,q)$$

Reflectance Map



# Reflectance Map

- Let $(p_S, q_S) = (s_x / s_z, s_y / s_z)$

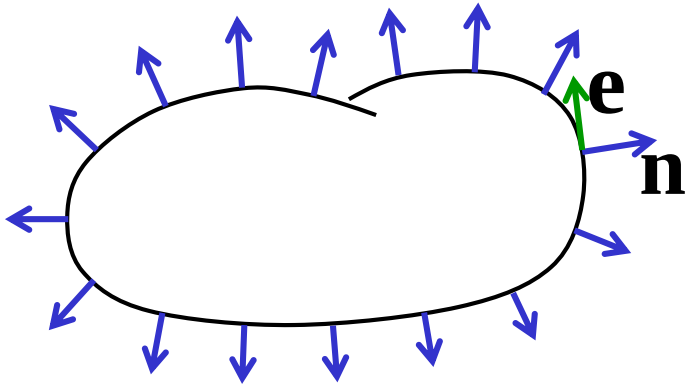


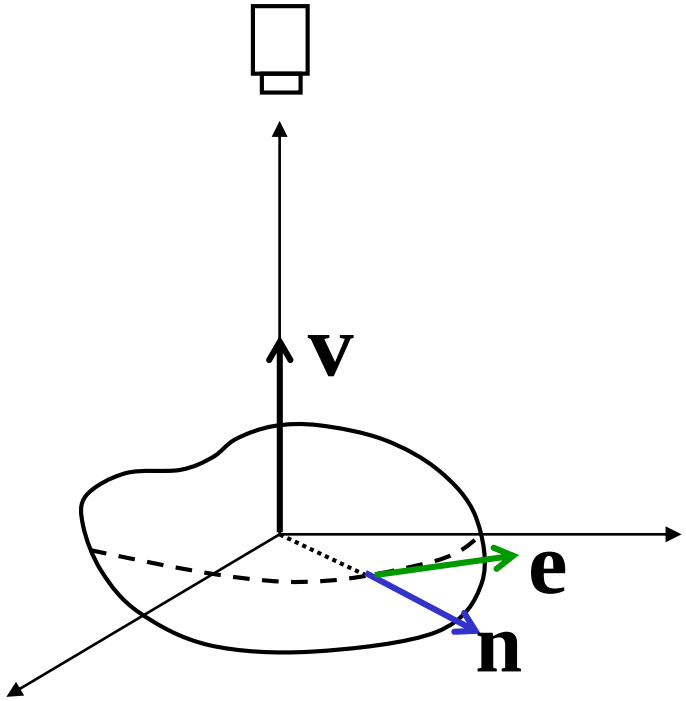

Note: $R(p,q)$ is maximum when $(p,q) = (p_S, q_S)$

# Possible Solutions

- Add more constraints
  - Shape-from-shading
- Take more images
  - Photometric stereo

# Occluding Boundaries




$\mathbf{n} \perp \mathbf{e}, \quad \mathbf{n} \perp \mathbf{v} \Rightarrow \mathbf{n} = \mathbf{e} \times \mathbf{v}$ $\mathbf{e}$ and $\mathbf{v}$ are known

The $\mathbf{n}$ values on the occluding boundary can be used as the boundary condition for shape-from-shading

# Energy Formulation

- Image irradiance should match the reflectance map (necessary but insufficient constraint)

Minimize

$$e_i = \iint_{\text{image}} \left(I(x,y) - R(f,g)\right)^2 dxdy$$

(minimize errors in image irradiance in the image)

# Smoothness Constraint

- Used to constrain shape-from-shading
- Relates orientations *(f,g)* of neighboring points

Minimize

$$e_s = \iint_{\text{image}} \left(f_x^2 + f_y^2\right) + \left(g_x^2 + g_y^2\right) dx dy$$

$(f, g)$: surface orientation under stereographic projection

$$f_x = \frac{\partial f}{\partial x}, f_y = \frac{\partial f}{\partial y}, g_x = \frac{\partial g}{\partial x}, g_y = \frac{\partial g}{\partial y}$$

(penalize rapid changes in surface orientation *f* and *g* over the image)

# Shape-from-Shading

- Find surface orientations *(f,g)* at all image points that minimize

$$e = e_s + \lambda e_i$$

weight

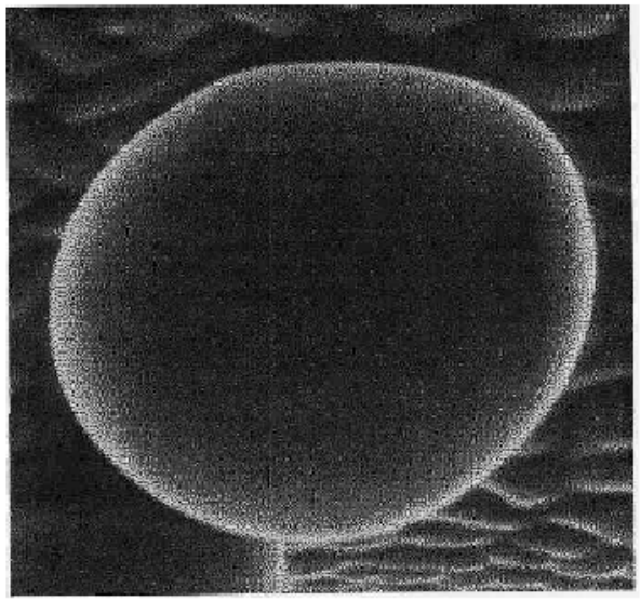
smoothness constraint

image irradiance error

Minimize:

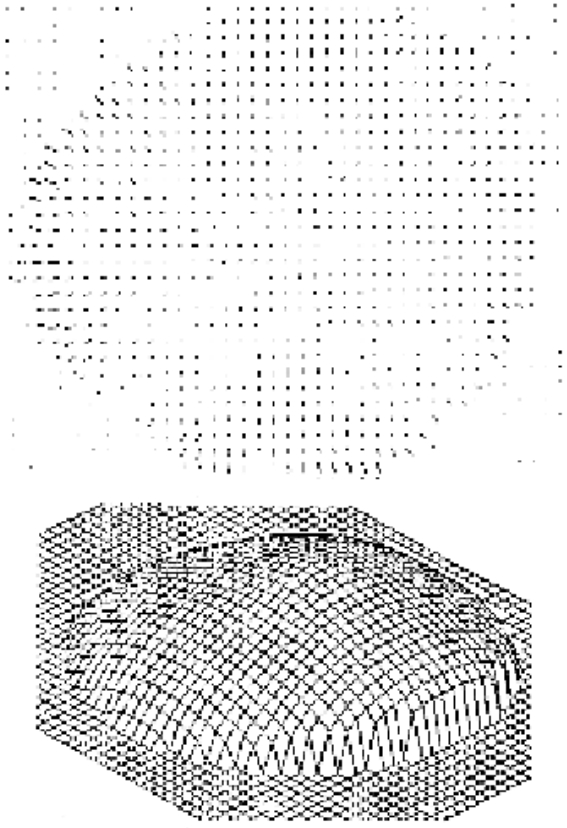
$$e = \iint_{\text{image}} \left(f_x^2 + f_y^2\right) + \left(g_x^2 + g_y^2\right) + \lambda\left(I(x,y) - R(f,g)\right)^2 dxdy$$

# Results



Scanning Electron Microscope image
(inverse intensity)

by Ikeuchi and Horn

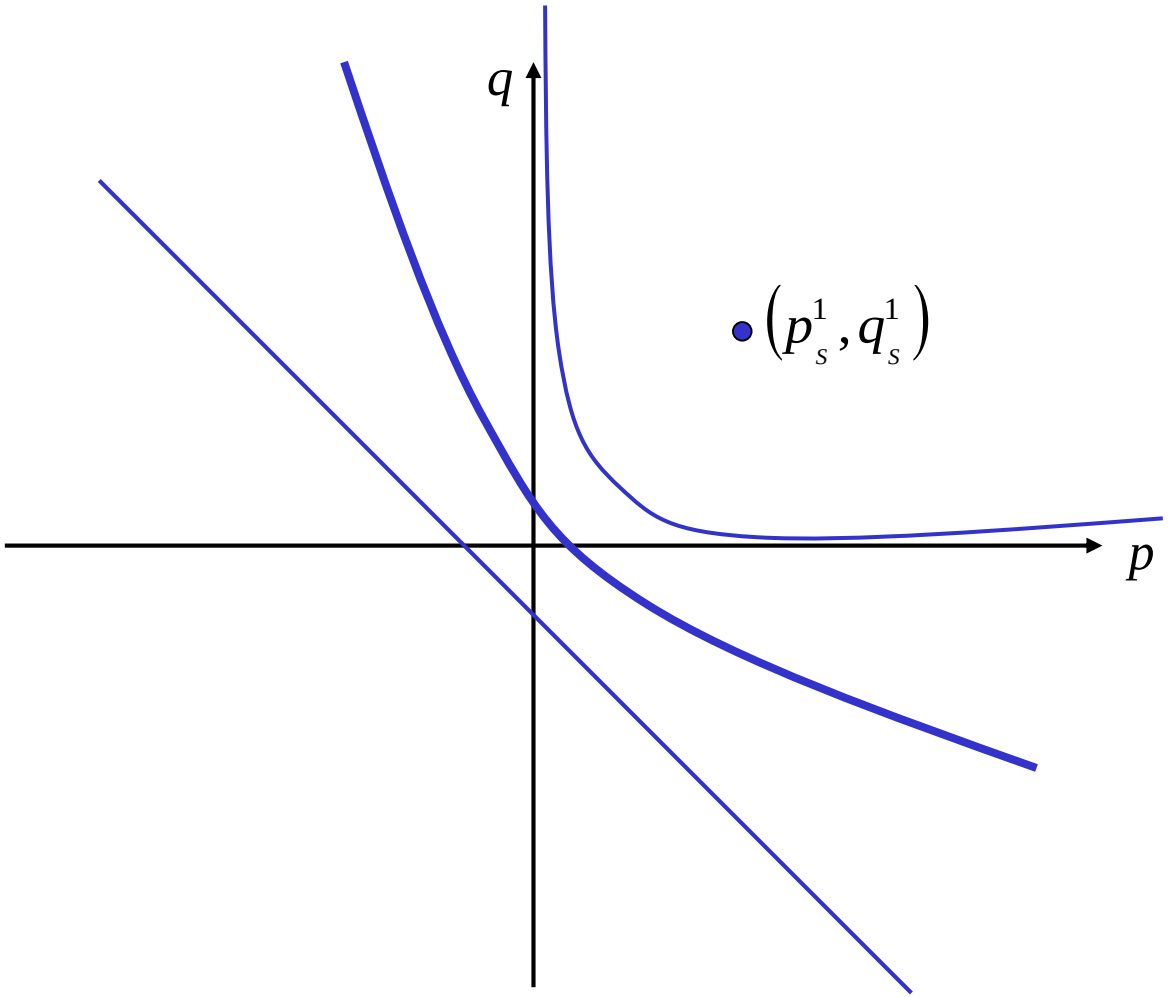


# Solution

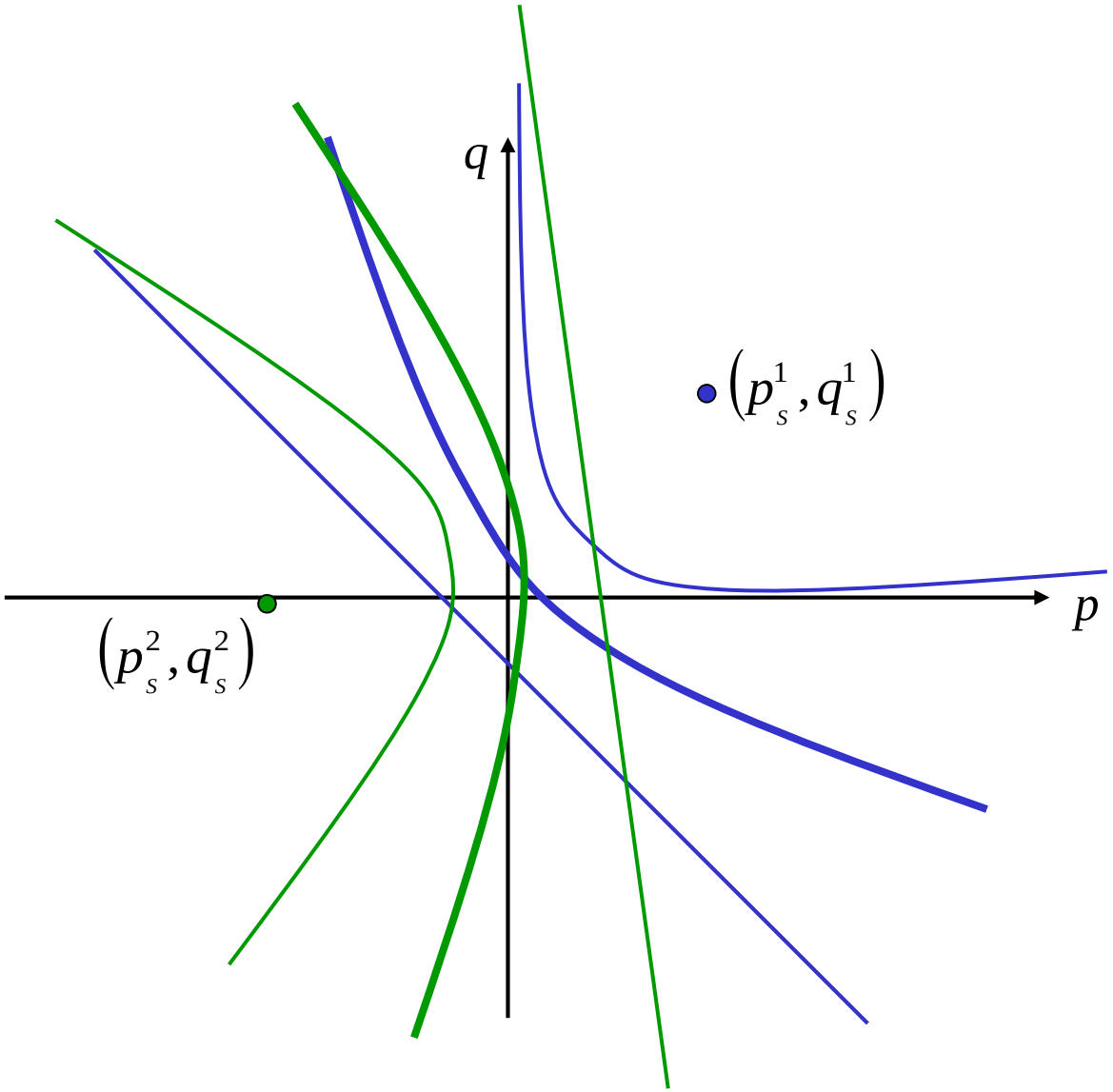
- Add more constraints
  - Shape-from-shading

- Take more images
  - Photometric stereo

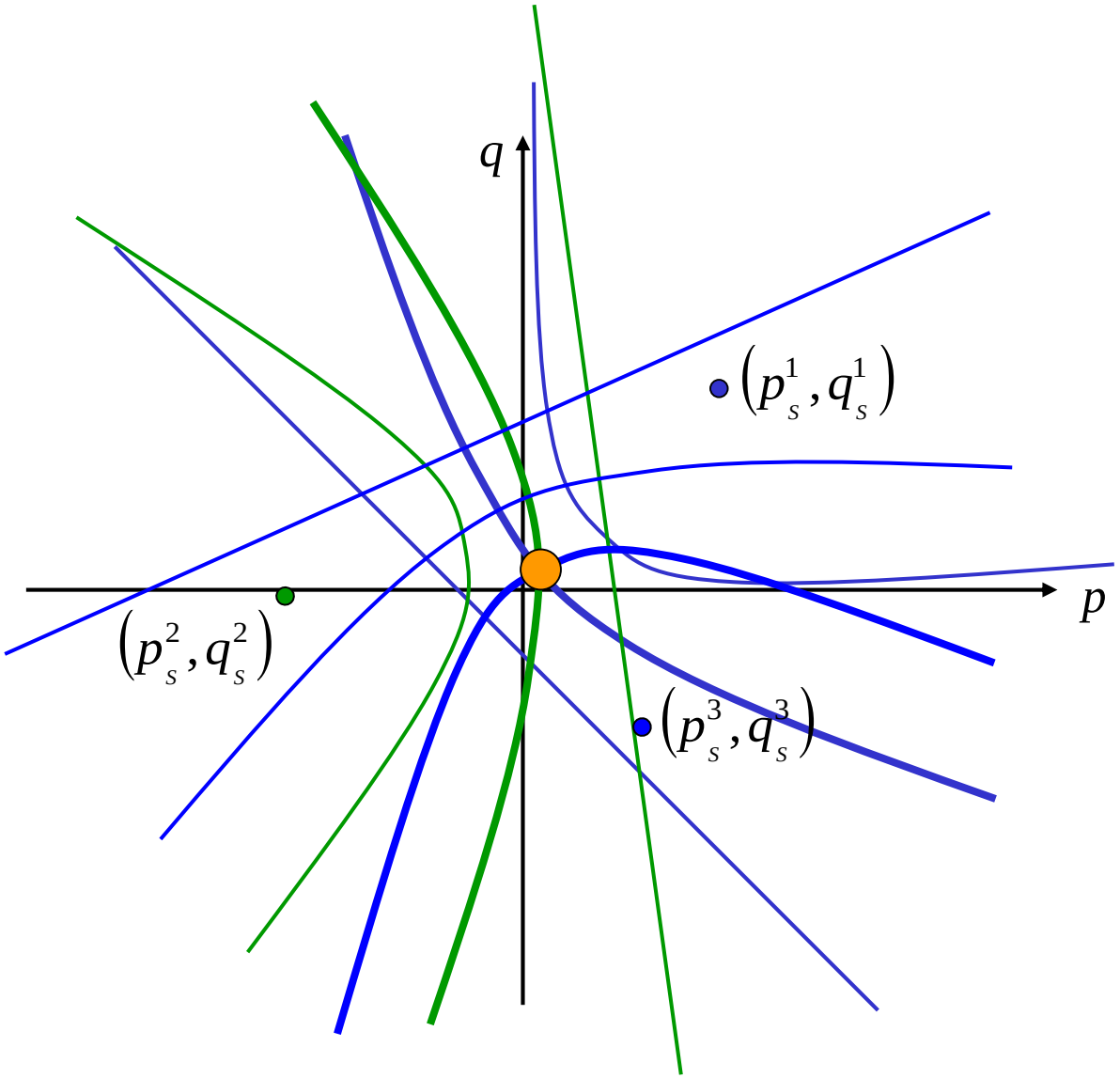
# Photometric Stereo



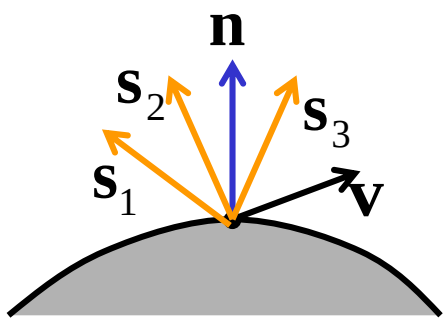
# Photometric Stereo



# Photometric Stereo



# Photometric Stereo




Lambertian case:

$$I = \frac{\rho}{\pi} kc \cos\theta_i = \rho \mathbf{n} \cdot \mathbf{s} \quad \left( \frac{kc}{\pi} = 1 \right)$$

Image irradiance:

$$I_1 = \rho \mathbf{n} \cdot \mathbf{s}_1$$
$$I_2 = \rho \mathbf{n} \cdot \mathbf{s}_2$$
$$I_3 = \rho \mathbf{n} \cdot \mathbf{s}_3$$

We can write this in matrix form:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_2 \end{bmatrix} = \rho \begin{bmatrix} \mathbf{s}_1^T \\ \mathbf{s}_2^T \\ \mathbf{s}_3^T \end{bmatrix} \mathbf{n}$$

$k$ : source brightness

$\rho$ : surface albedo (reflectance)

$c$ : constant (optical system)

# Solving the Equations

$$\underbrace{\begin{bmatrix} I_1 \\ I_2 \\ I_2 \end{bmatrix}}_{\mathbf{I}_{3\times 1}} = \underbrace{\begin{bmatrix} \mathbf{s}_1^T \\ \mathbf{s}_2^T \\ \mathbf{s}_3^T \end{bmatrix}}_{\mathbf{S}_{3\times 3}} \underbrace{\rho\mathbf{n}}_{\tilde{\mathbf{n}}_{3\times 1}}$$

$$\tilde{\mathbf{n}} = \mathbf{S}^{-1}\mathbf{I} \qquad \rho = \left|\tilde{\mathbf{n}}\right|$$

$$\mathbf{n} = \frac{\tilde{\mathbf{n}}}{\left|\tilde{\mathbf{n}}\right|} = \frac{\tilde{\mathbf{n}}}{\rho}$$

# Even more Light Sources

Better results by using more lights:

$$\begin{bmatrix} I_1 \\ \vdots \\ I_N \end{bmatrix} = \begin{bmatrix} \mathbf{s}_1^T \\ \vdots \\ \mathbf{s}_N^T \end{bmatrix} \rho \mathbf{n}$$

- Least squares solution:

$$\mathbf{I} = \mathbf{S}\tilde{\mathbf{n}} \quad \longleftarrow \quad N \times 1 = (N \times 3)(3 \times 1)$$

$$\mathbf{S}^T\mathbf{I} = \mathbf{S}^T\mathbf{S}\tilde{\mathbf{n}}$$

$$\tilde{\mathbf{n}} = \left(\mathbf{S}^T\mathbf{S}\right)^{-1}\mathbf{S}^T\mathbf{I}$$

Moore-Penrose pseudo inverse

- Solve for $\rho, \mathbf{n}$ as before

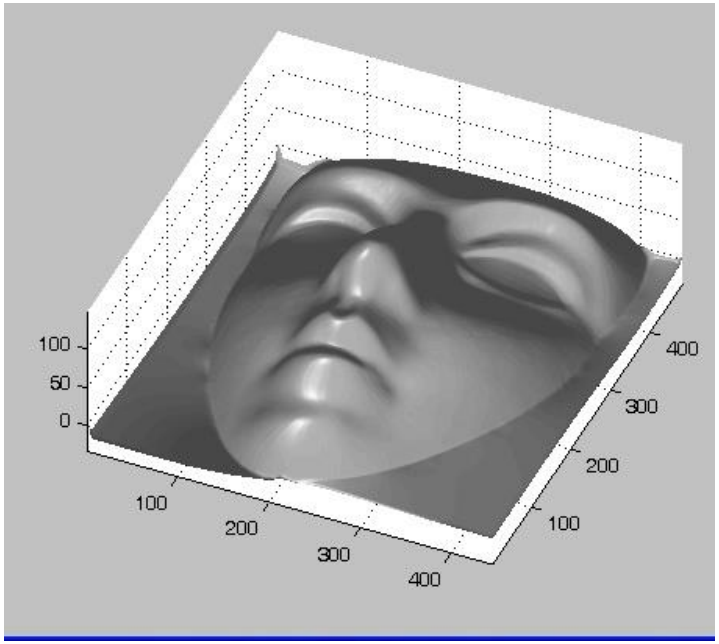
# Results

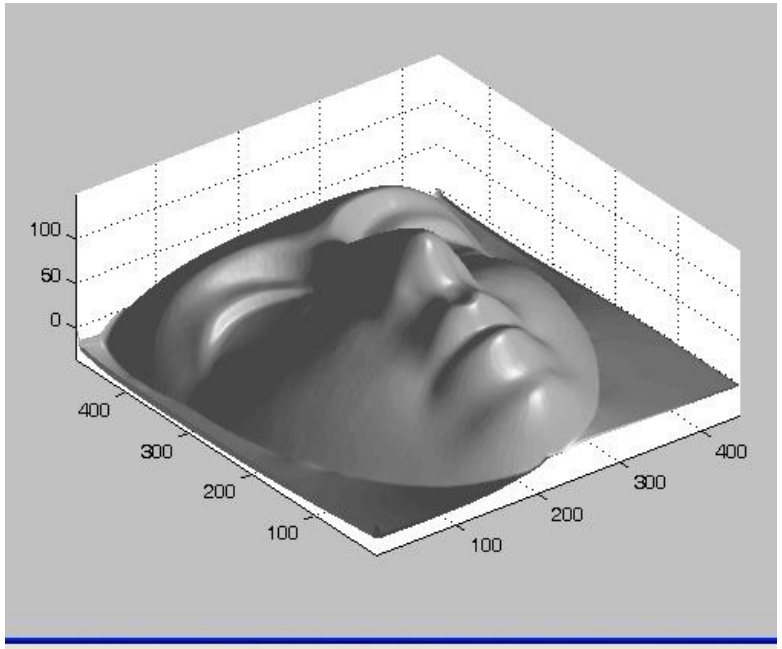
**Algorithm:**

1. Estimate light source directions
2. Compute surface normals
3. Compute albedo values
4. Estimate depth from surface normals
5. Relight the object (with original texture and uniform albedo)

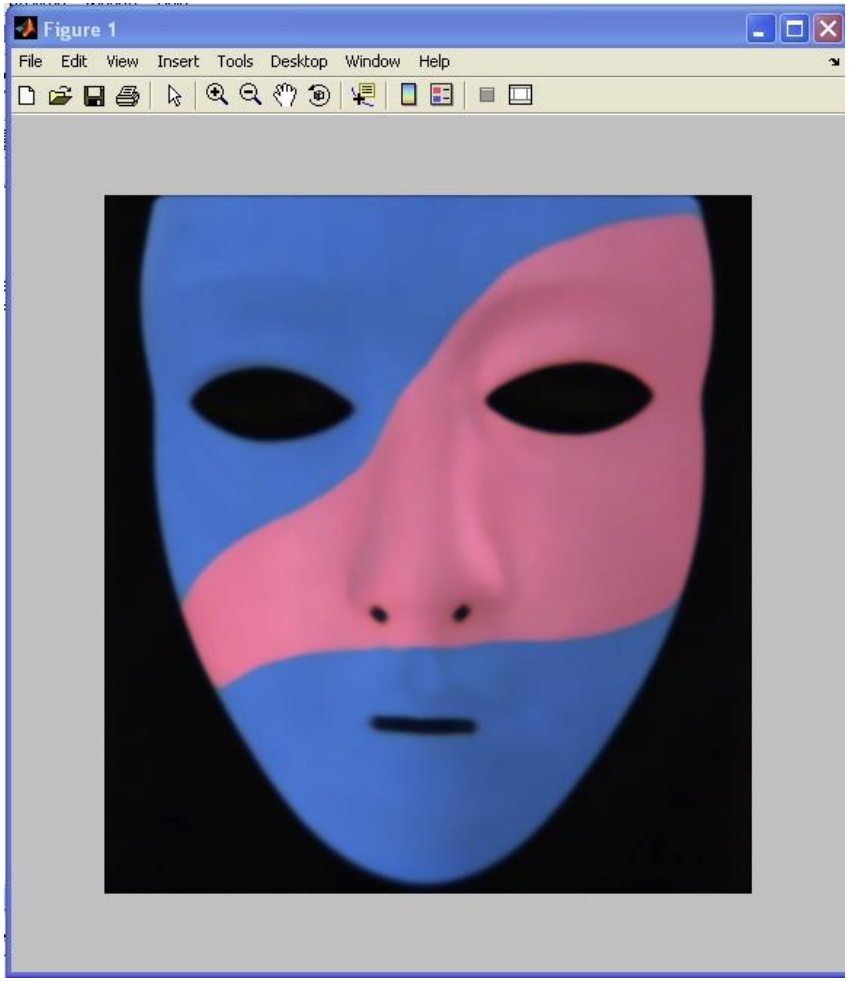
# Original Images

# Results - Shape



# Results - Albedo

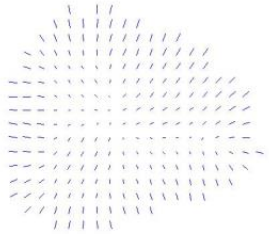
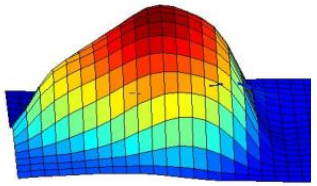


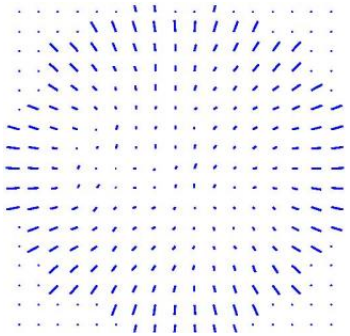
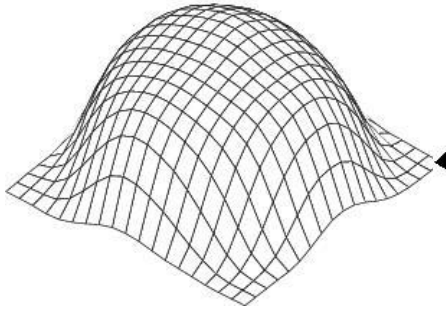
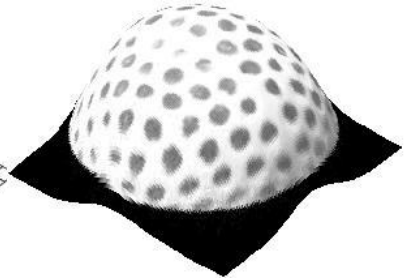
# Shape from texture

- Obtain normals from texture element deformations (or statistics)

Examples from Angie Loh

# Shape from texture

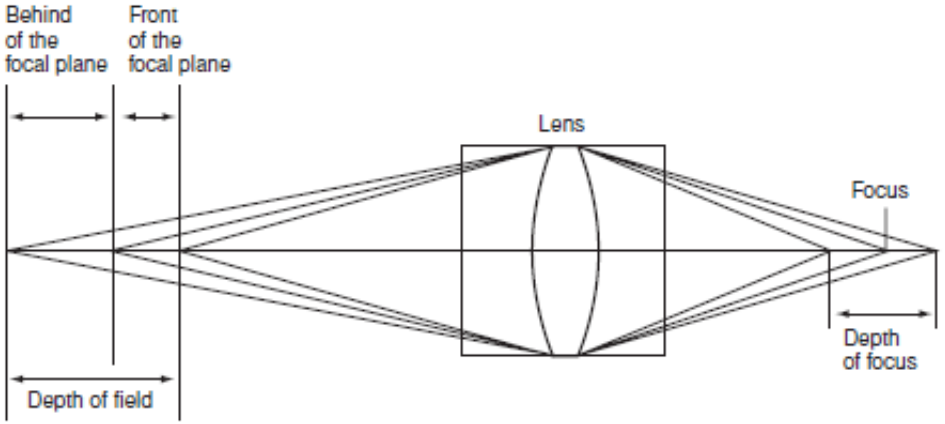




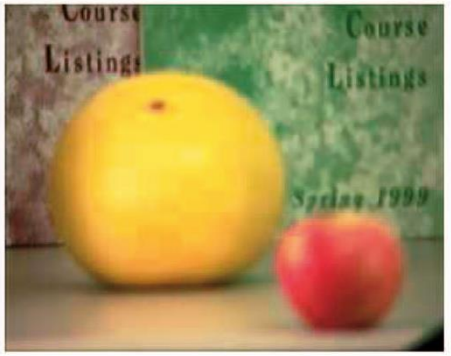
Examples from Angie Loh

# Depth from focus

- Sweep through focus settings
- "most sharp" pixels correspond to depth (most high frequencies)

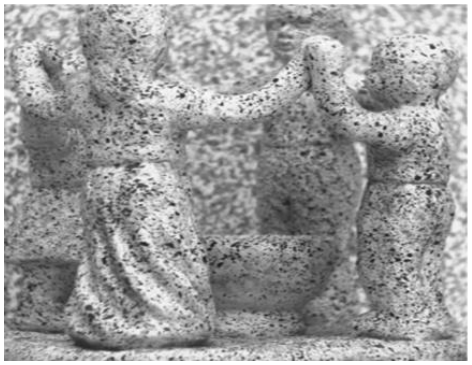


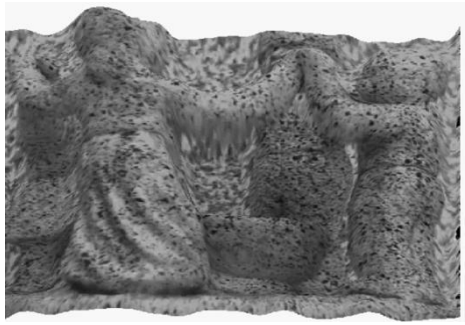




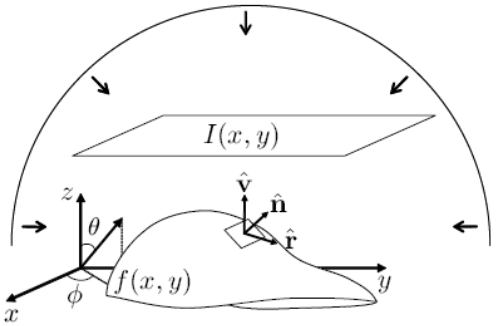
# Depth from Defocus

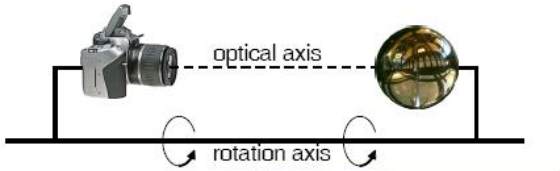
- Compare relative sharpness between images
- More complicated, but needs less images

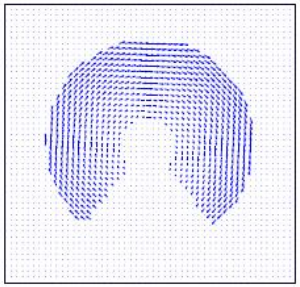


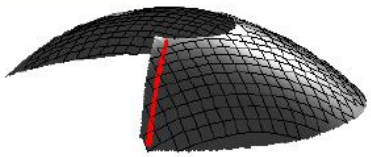



# Shape from specularities





**Toward a Theory of Shape from Specular Flow**
Y. Adato, Y. Vasilyev, O. Ben-Shahar, T. Zickler, ICCV'07.

Thanks!