Surface Modeling and Display from Range and Color Data

Kari	Pulli	UW
Tony	DeRose	Pixar
Tom	Duchamp	UW
Linda	Shapiro	UW
Werner	Stuetzle	UW
Michael	Cohen	MSR
Hugues	Hoppe	MSR
John	McDonald	UW

UW = University of Washington
MSR = Microsoft Research

Overview, part I

Surface reconstruction

- Range data acquisition
- 3D registration
- Creating initial surface estimate
- Mesh optimization

Overview, part II

View-dependent texturing, two approaches:

Single surface, projected textures.

Partial models, integrate during display.

Related work

Model construction from range data:

- Hoppe et al. ['92,'93,'94]
- Curless and Levoy ['96]
- other work at Stanford, GRASP, Laval, McGill, NRCC, CMU, INRIA, USC, ...

Image-based rendering:

- Gortler et al. ['96], Levoy & Hanrahan ['96]
- Debevec et al. ['96]
- other work at Apple, UNC, SUNY, ...

Surface Reconstruction

Step 1: Data acquisition

Obtain range data that covers the object. Filter, remove background.

Step 2: Registration

Step 3: Integration

Step 4: Optimization

Scanner setup

Dense range (and color) from stereo + active light

4 Sony video cameras

light projector

turntable

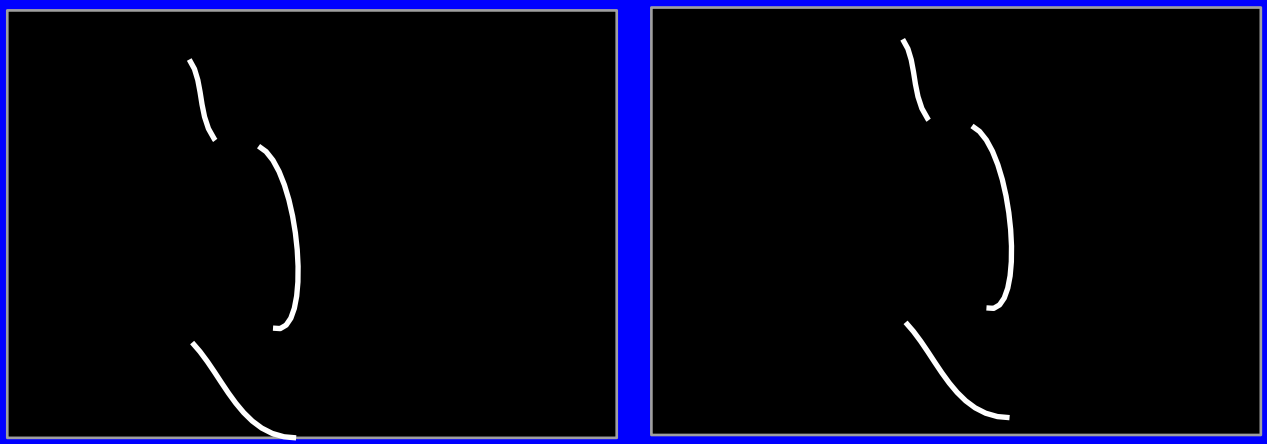
lamps

Calibrate cameras

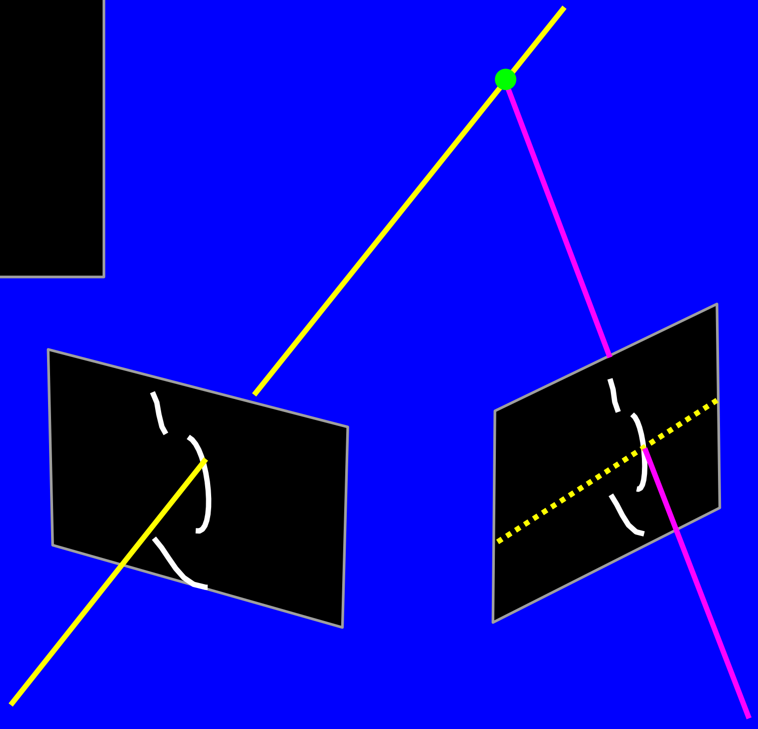
Calibrate cameras from known 3D-2D point correspondences [Tsai '87]

Scanning

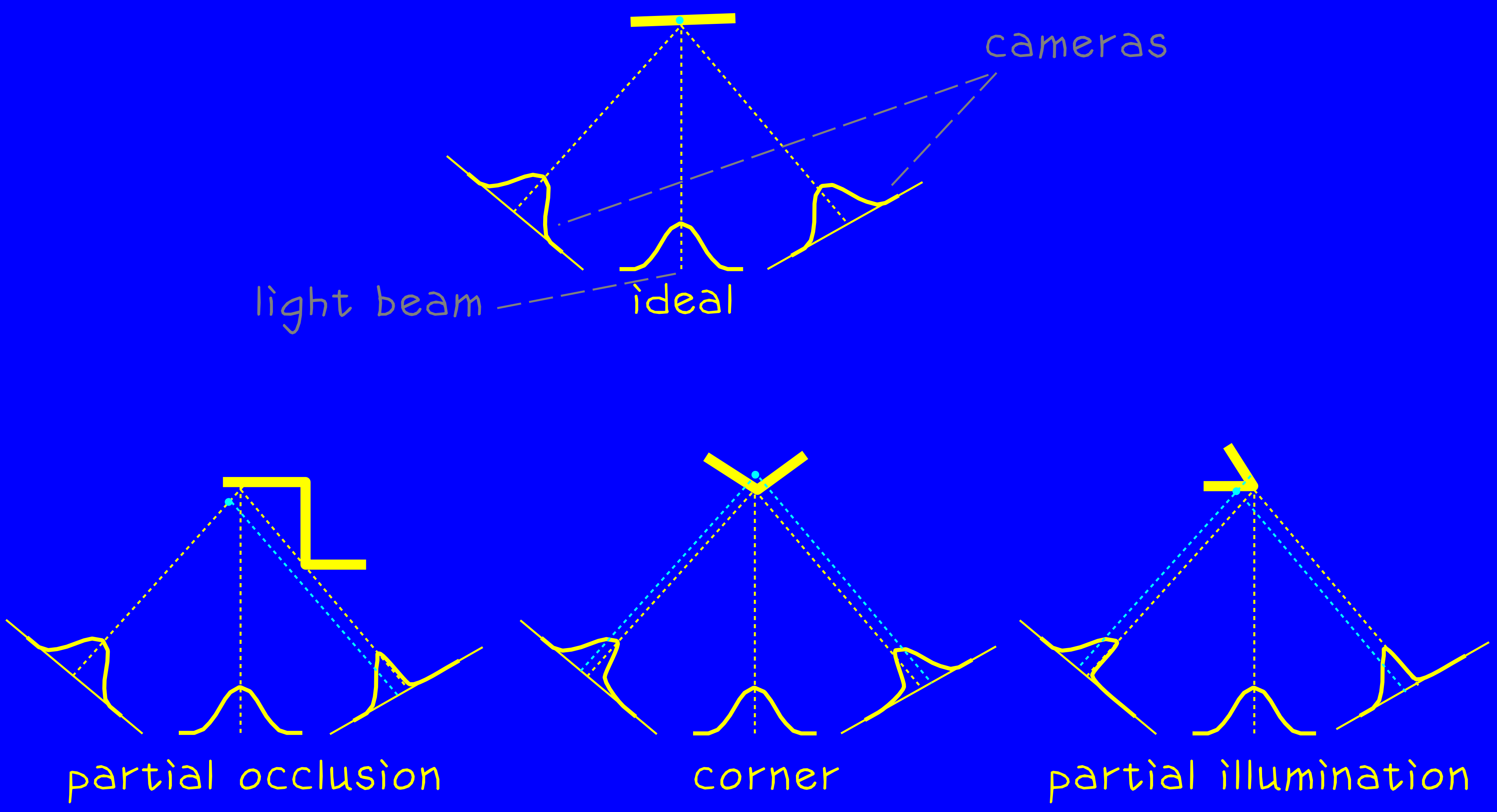
- Sweep a vertical light stripe in small steps
- Detect the stripe from camera images



- Project a line from a pixel to the right image
- Triangulate



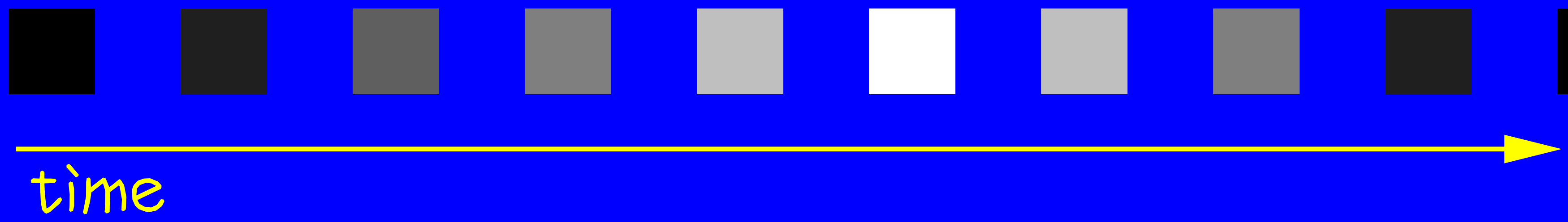
Beam location errors



Spacetime analysis

- Don't look at a single image where the beam is centered

- Look at a sequence of images when the beam was centered at each pixel



- Invert the function to locate beam in each image [Curless and Levoy '95]

Surface Reconstruction

Step 1: Data acquisition

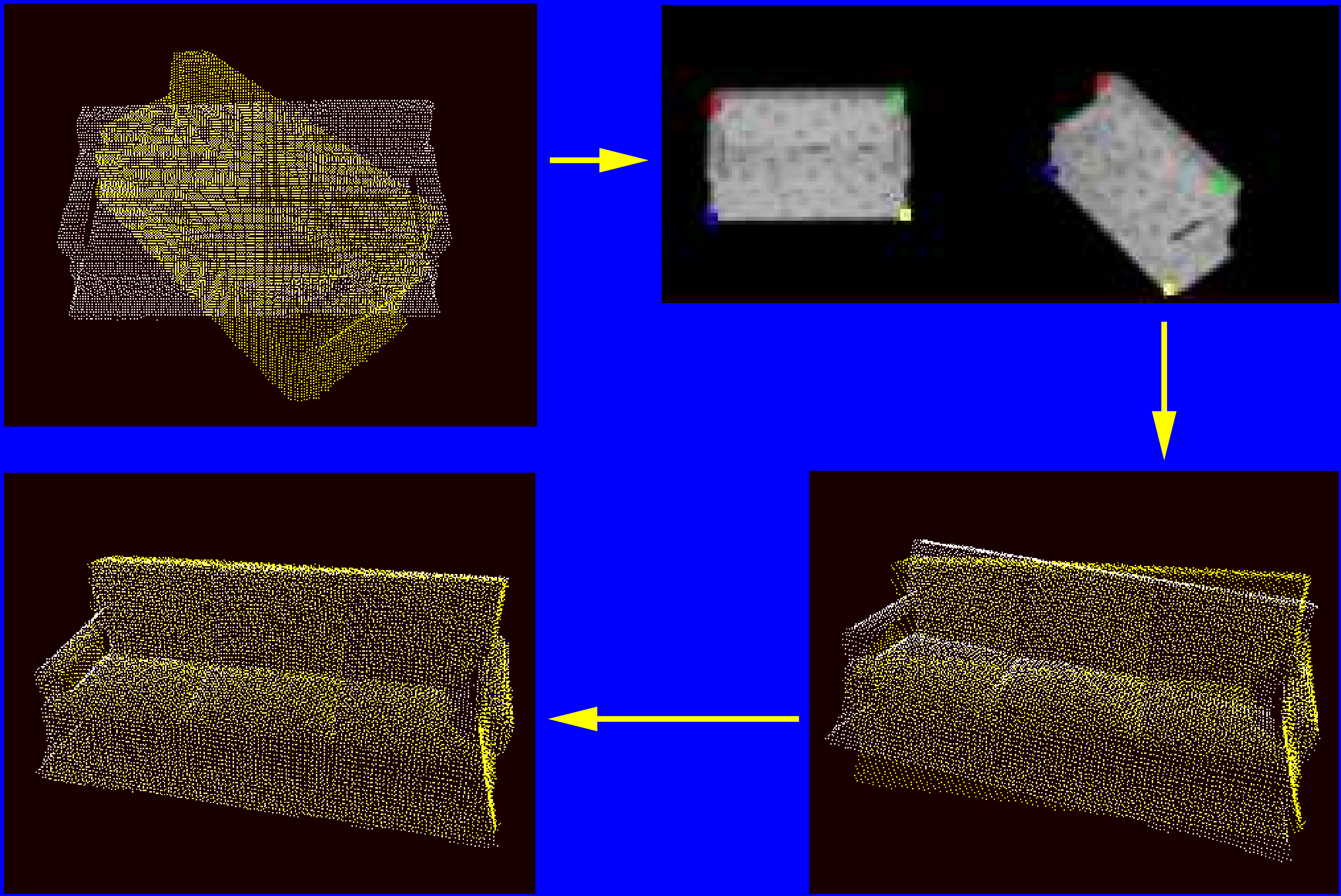
Step 2: Registration

Register the range maps into a common coordinate system.

Step 3: Integration

Step 4: Optimization

Registration: overview



Two subproblems

Correspondence

Given a point in one view, which point in the other view corresponds to the same surface point?

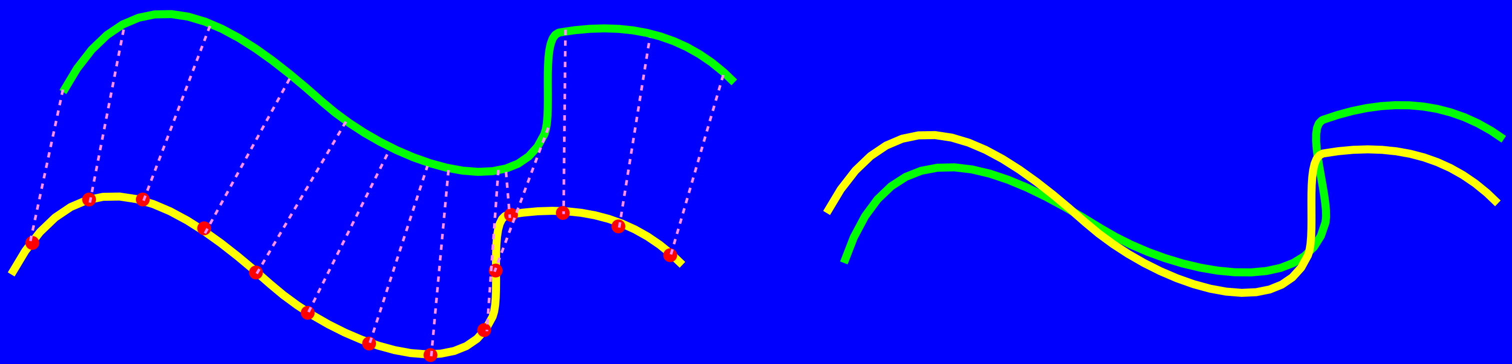
Alignment

Given point correspondences, which rigid motion aligns the surfaces (or paired points)?

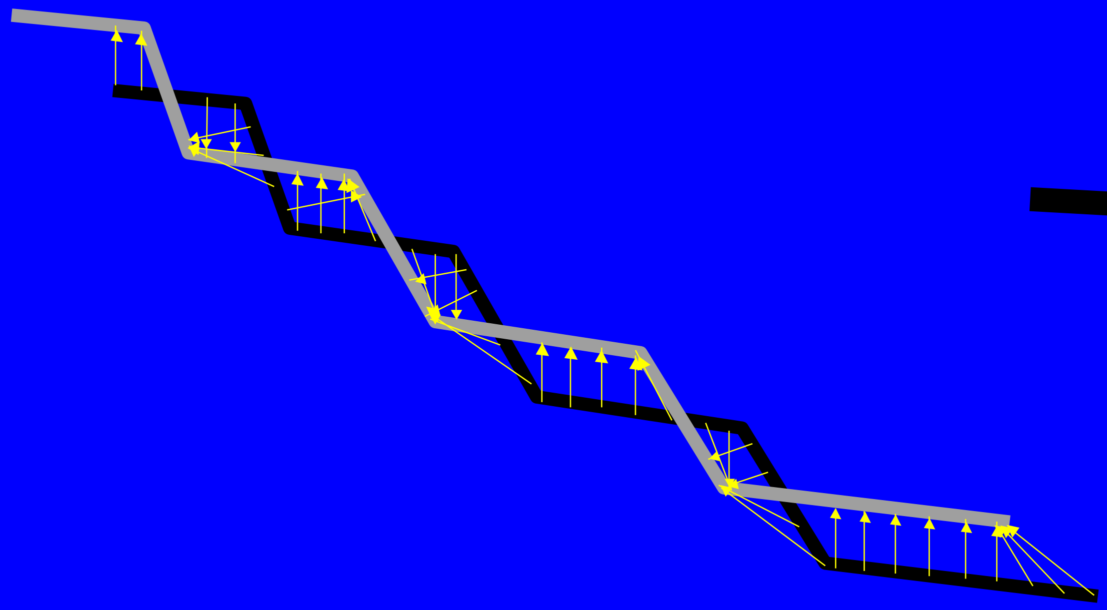
Iterative closest points (ICP)

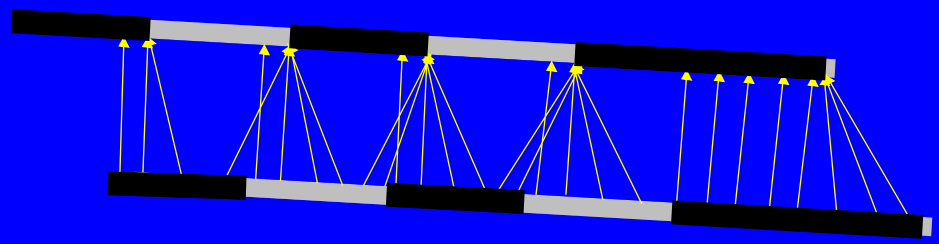
- Start from an approximate registration
- Pairing heuristic: match a point with the closest compatible point

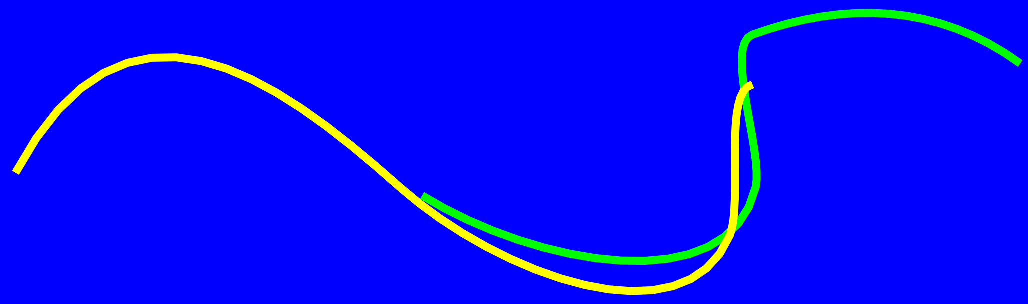
```
while (NOT DONE) {
        for each point {
                find closest point in the other view
        }
        move the surfaces closer
}
```



Finding good pairs is hard







Find all pairs at once

Project the scan data of sensor 2 over sensor 1's view

Align images
[Szeliski & Shum '97]

Pair points that project to the same pixel

3D alignment

Multiview registration

- Traditional approach: add another loop to ICP

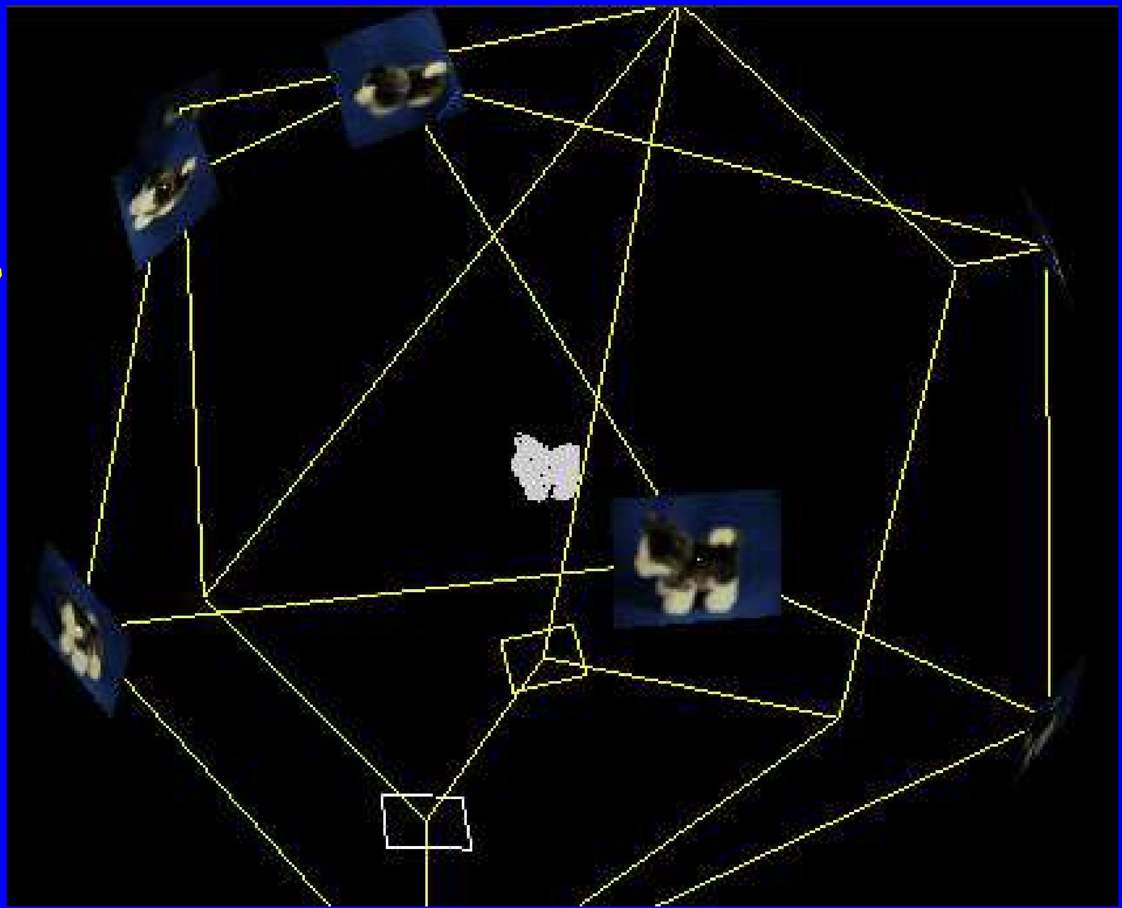
```
while (NOT DONE) {
        for each view {
                for each point {
                    find closest points in the other views
                }
                move the current surface closer
        }
}
```

SLOW!

Multiview registration

First find correspondences, then align

* register pairwise
* store reliable pairs
* find all registrations simultaneously



Surface Reconstruction

Step 1: Data acquisition

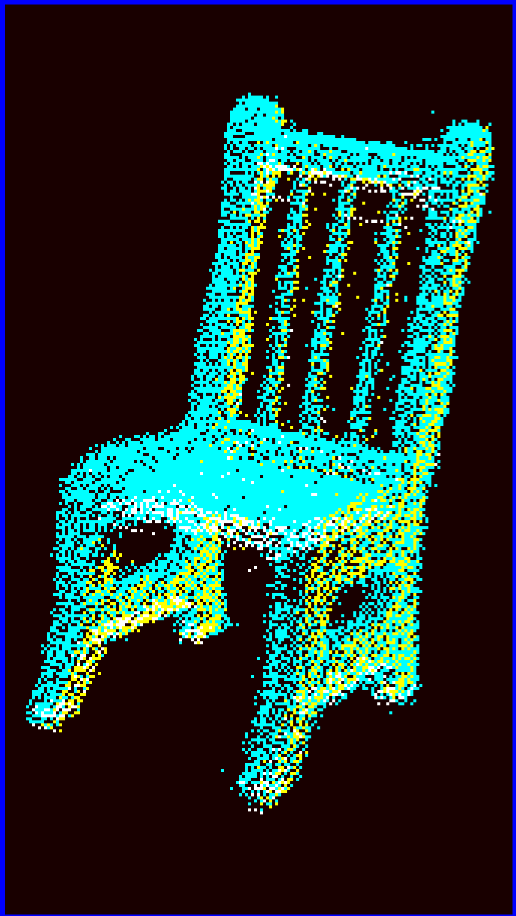
Step 2: Registration

Step 3: Integration

Integrate the registered range data into a single surface representation.

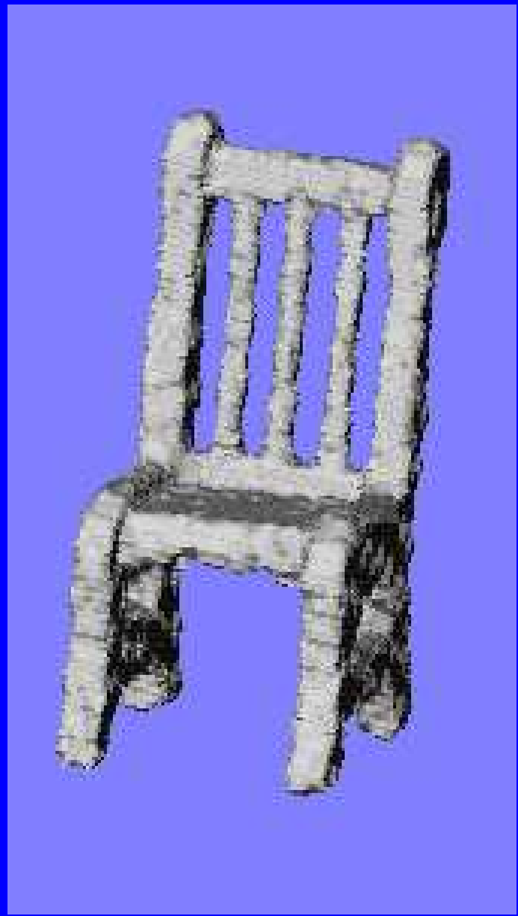
Step 4: Optimization

Problem



Noisy registered data

Signed distance fn & marching cubes

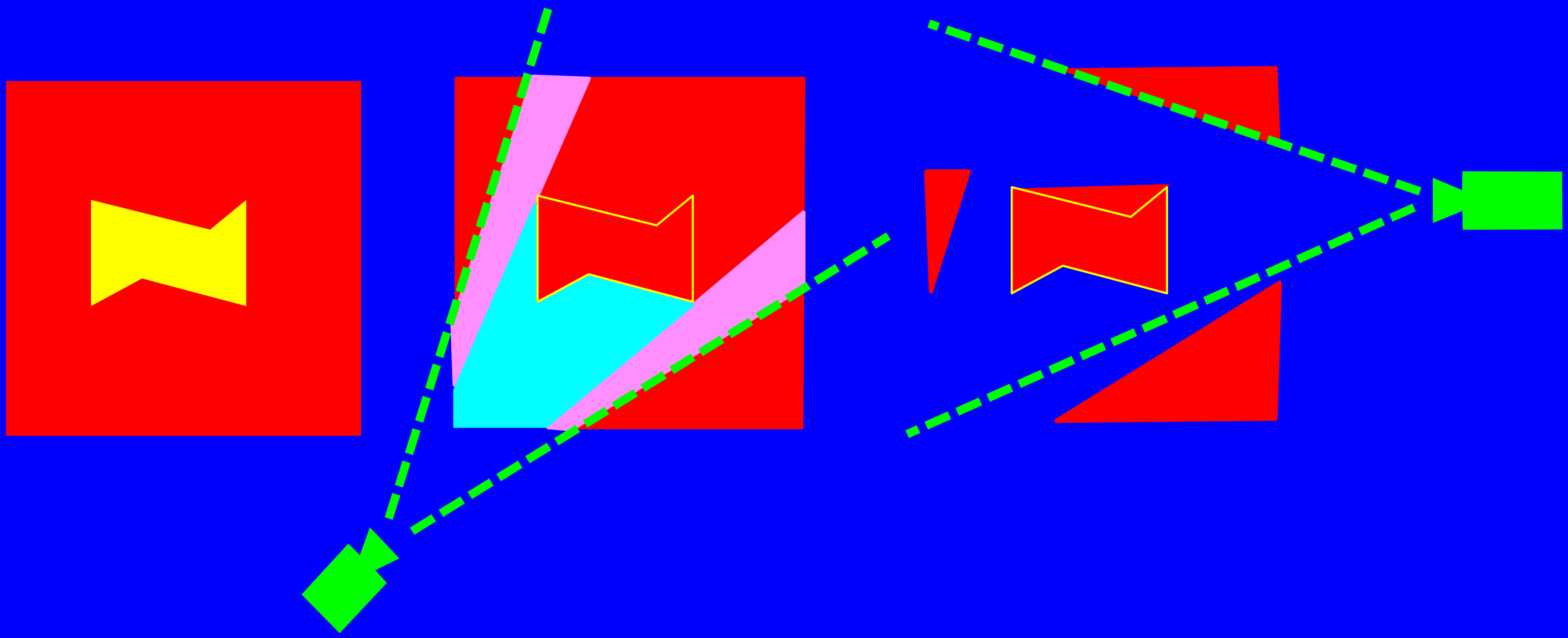


Hierarchical & directional space carving

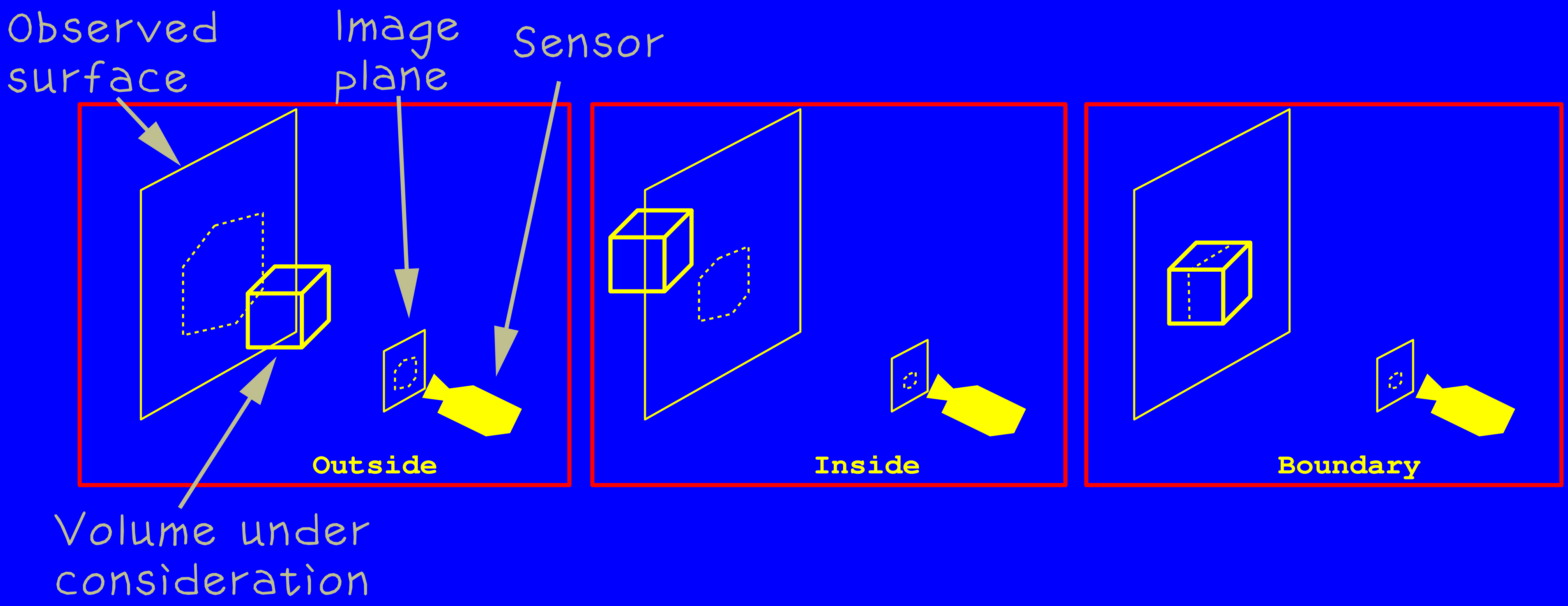
Space carving

- Curless & Levoy ['96]

- Concentrate on volumes



Carve space in cubes



Label cubes

- Project cube to image plane (hexagon)
- Test against data in the hexagon

Several views

Processing order:

FOR EACH cube

  FOR EACH view

Rules:

<u>any</u> view thinks cube's <u>out</u>

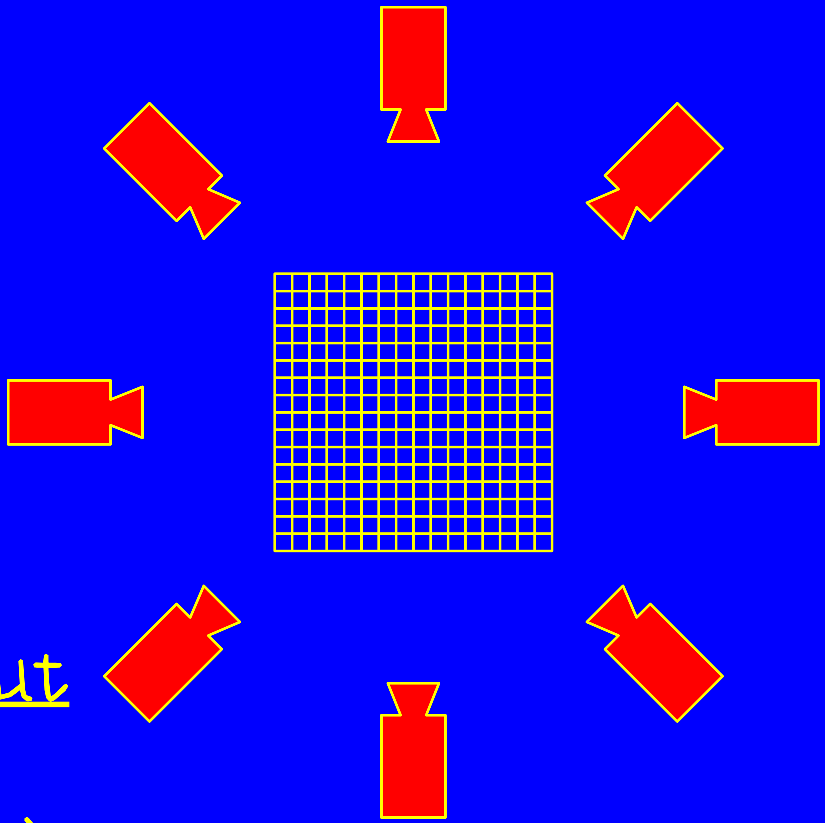
 => it's <u>out</u>

<u>every</u> view thinks cube's <u>in</u>

 => it's <u>in</u>

else

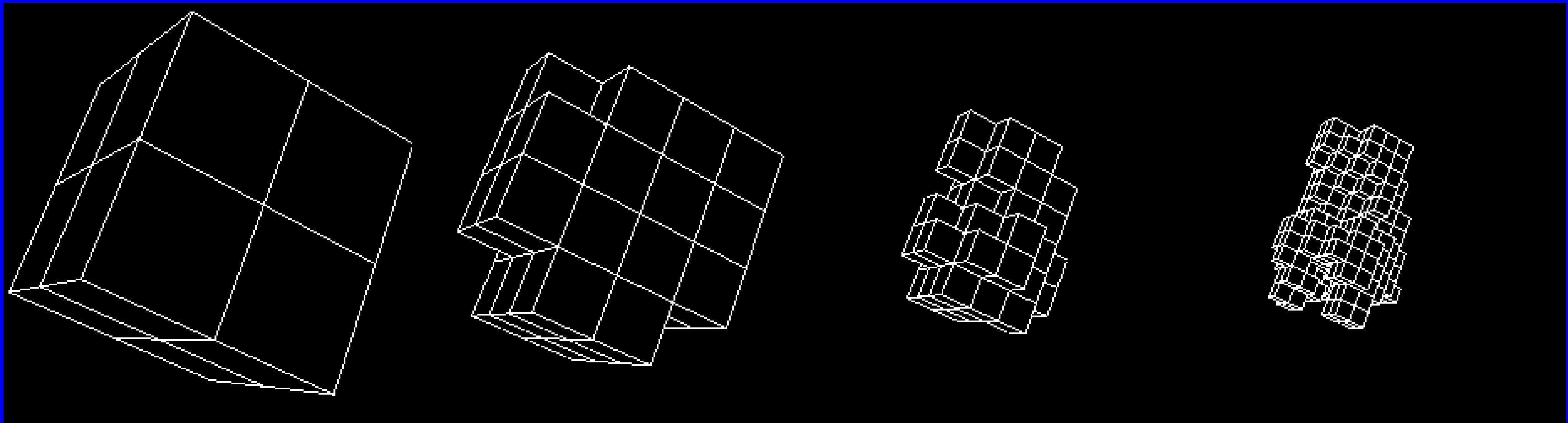
 => it's at boundary



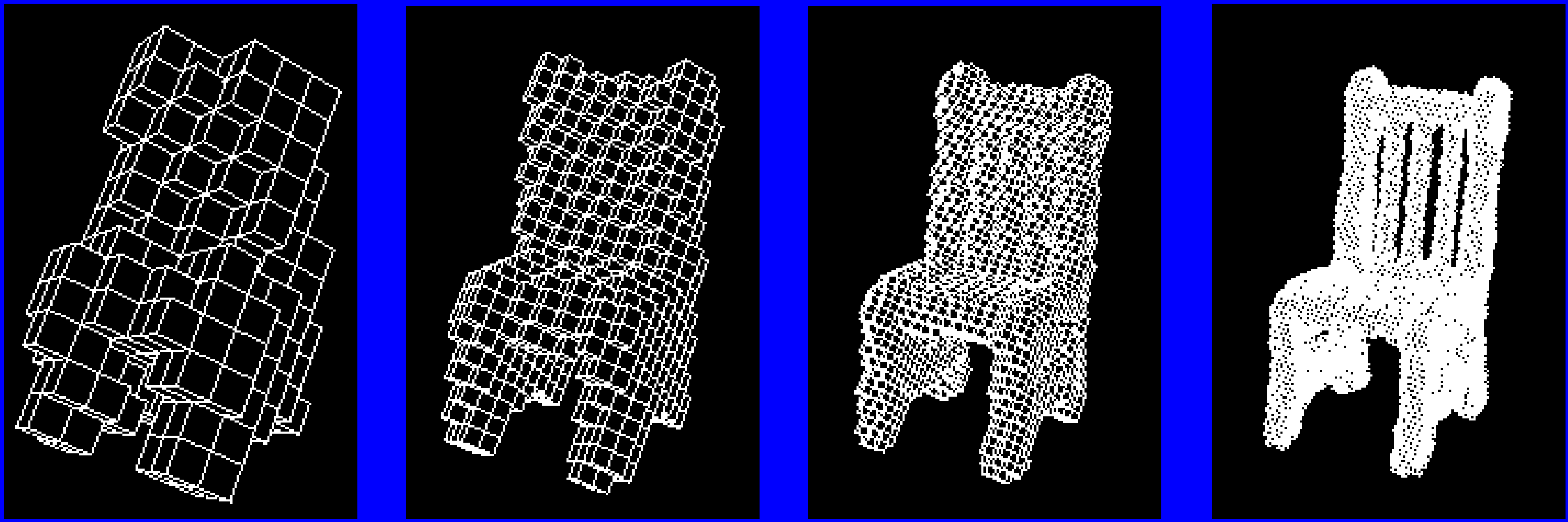
Hierarchical space carving

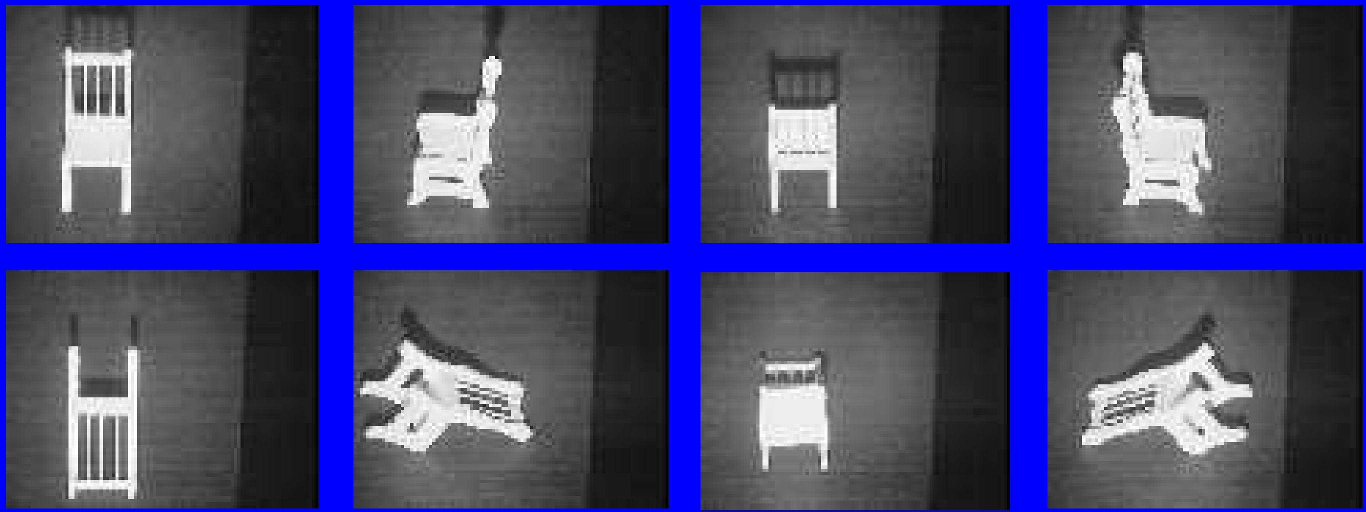
- Big cubes => fast, poor results
- Small cubes => slow, more accurate results
- Combination = octrees

RULES:
- cube is out => done
- cube is in => done
- else => recurse

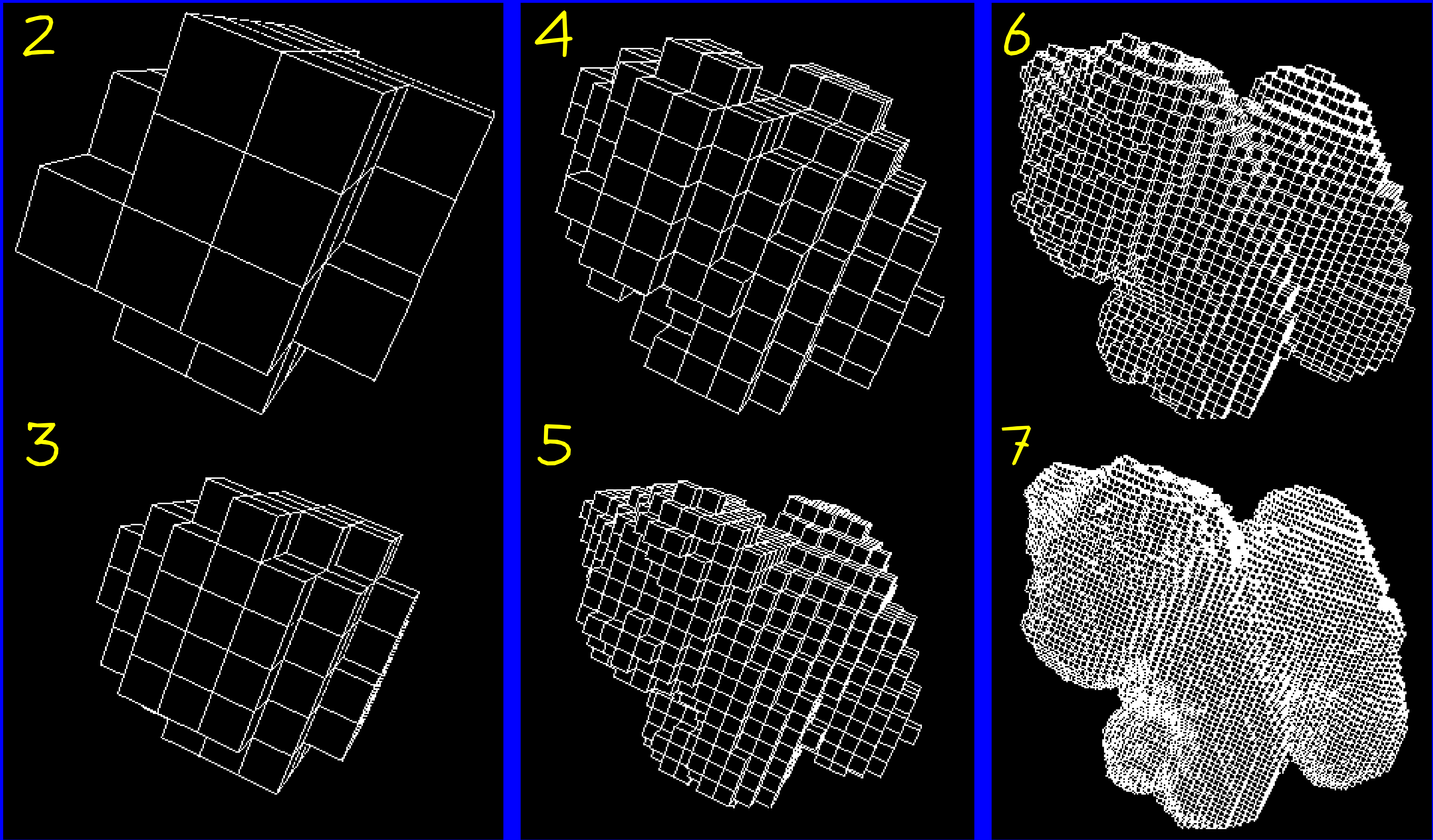


The rest of the chair

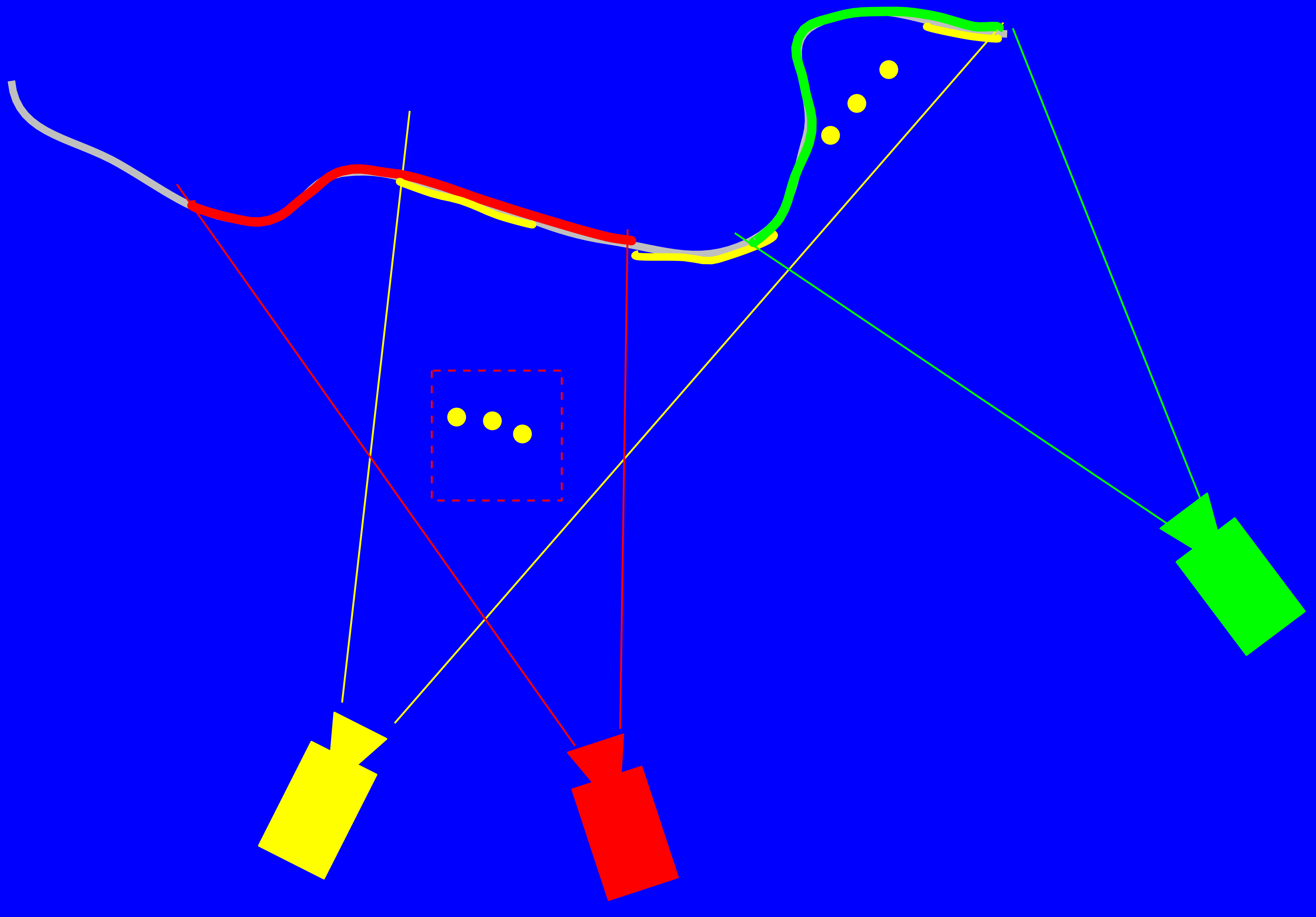




Same for a toy husky dog



Implicit outlier removal

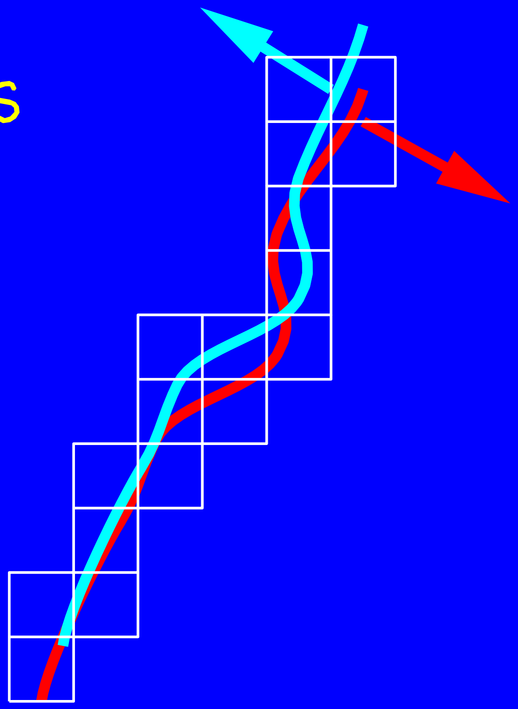


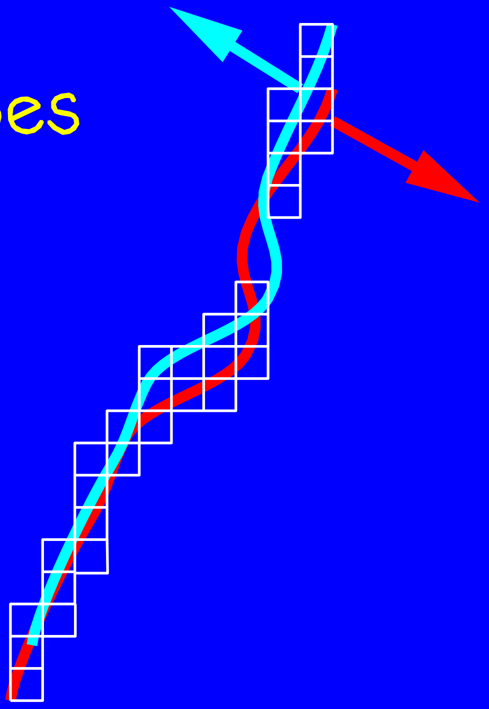
Thin surfaces

- Use of directional information
  allows reconstructing thin objects

- Surface seen from left and right
  Reconstruction:
  a shell around the surface

- Don't use too small cubes
  Size > max of
  * registration error
  * sampling error
  * sampling density





Surface Reconstruction

Step 1: Data acquisition

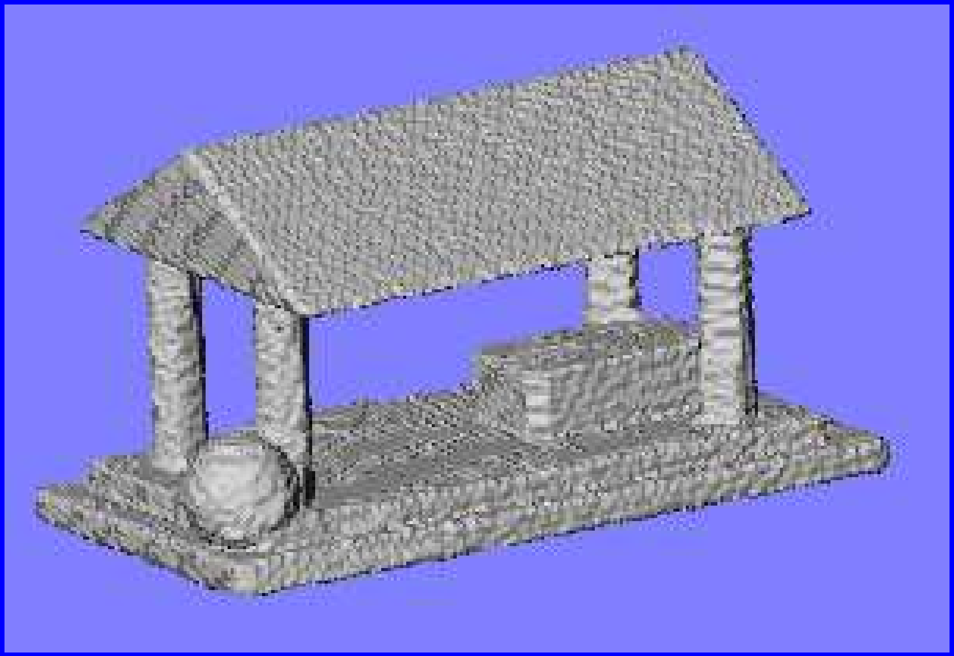
Step 2: Registration

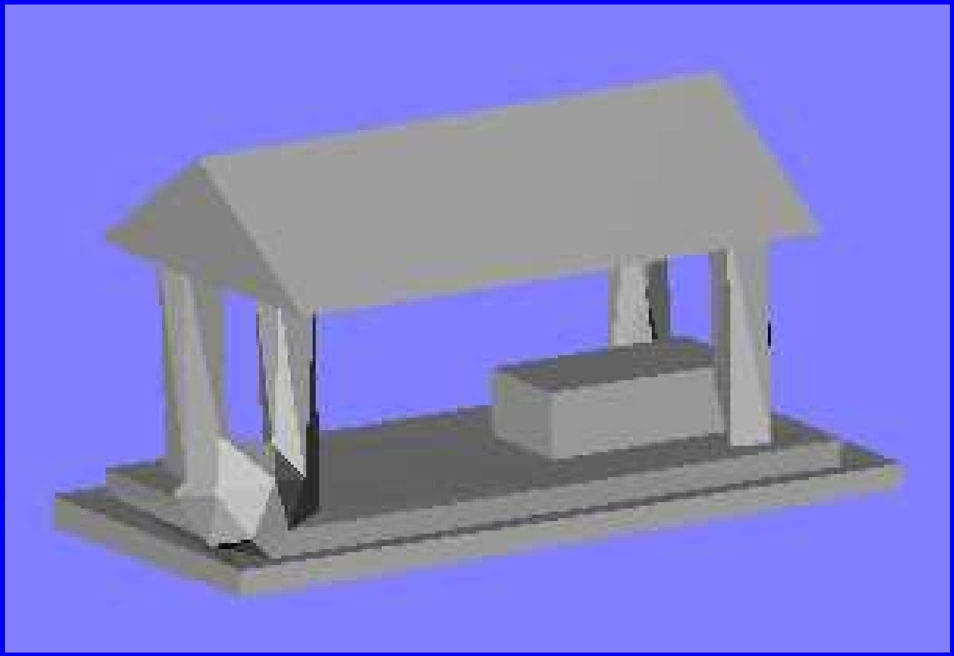
Step 3: Integration

Step 4: Optimization

Fit the surface more accurately to the data, simplify the representation.

Optimization example



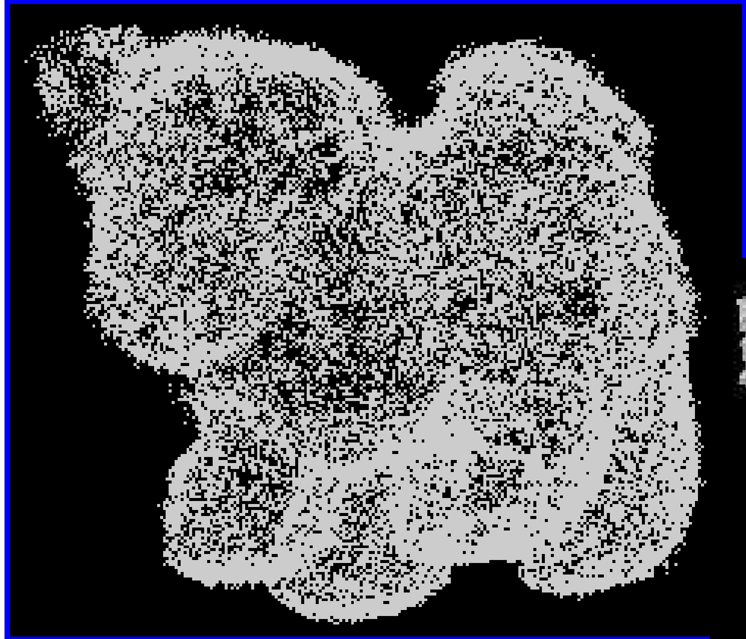


Our result (smoothed)

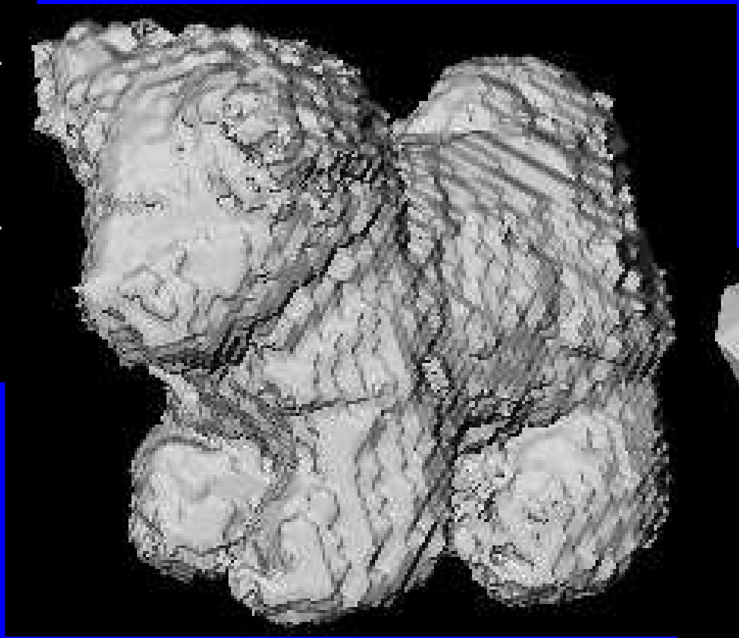
Fitted and simplified

[Hoppe '93]

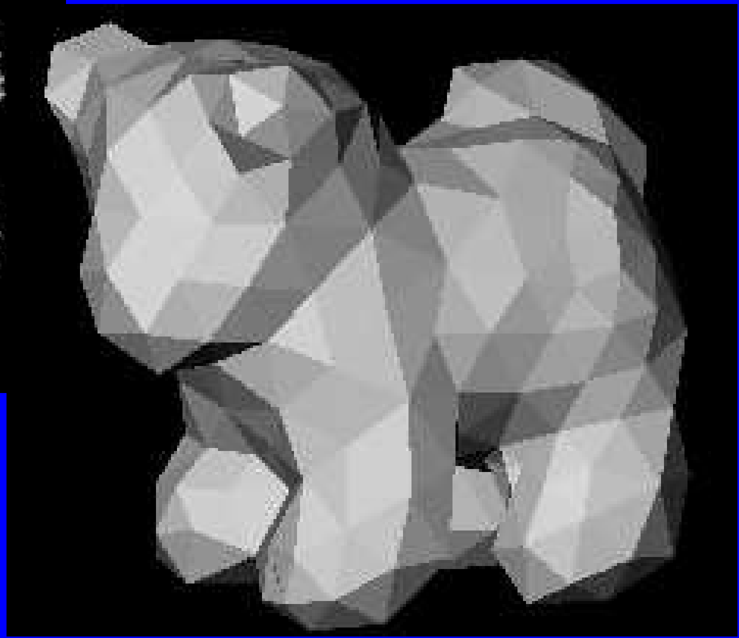
Optimizing the dog mesh



Registered points



Initial mesh



Optimized mesh

View-dependent texturing

Sampling problems

No single texture is good

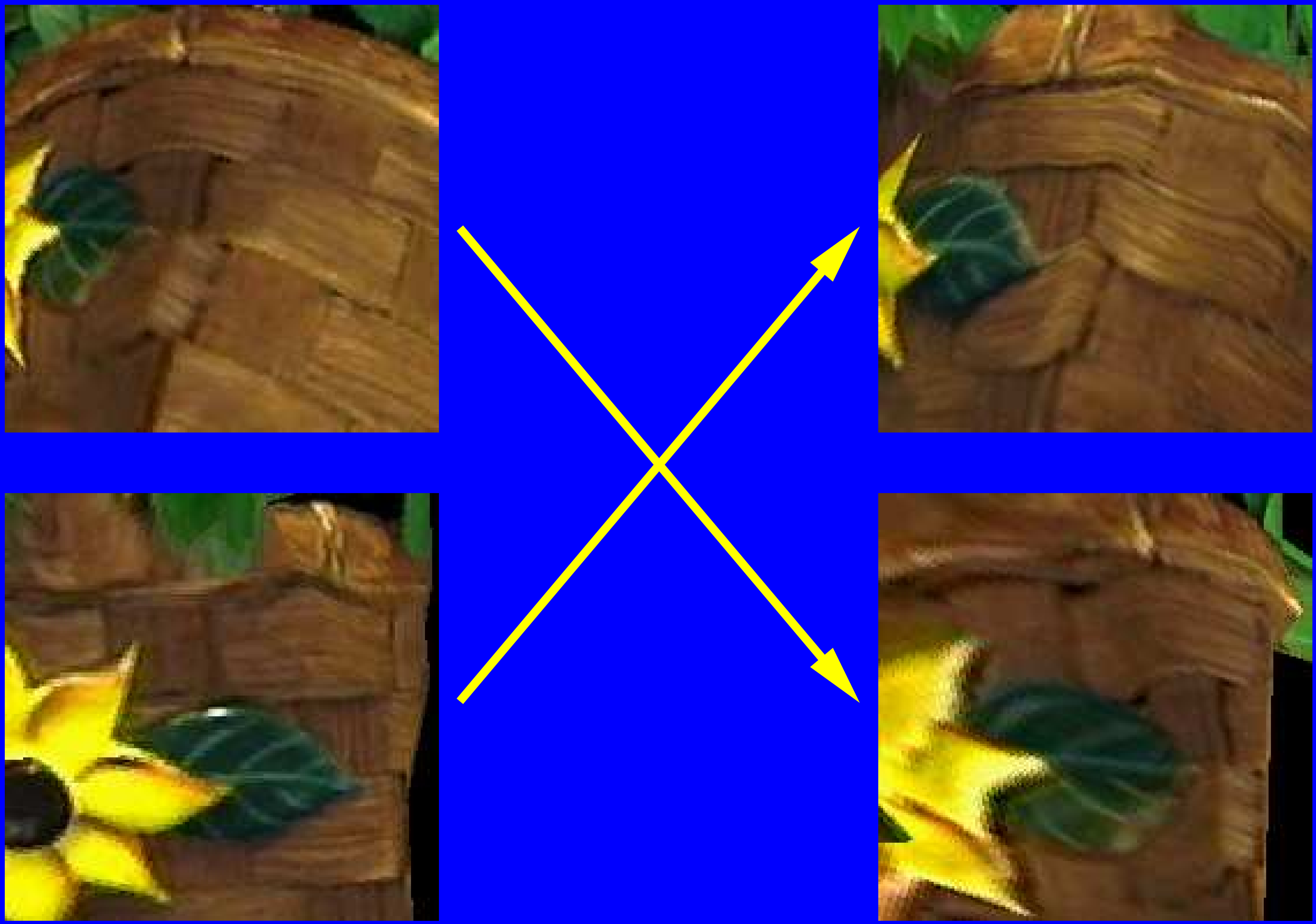
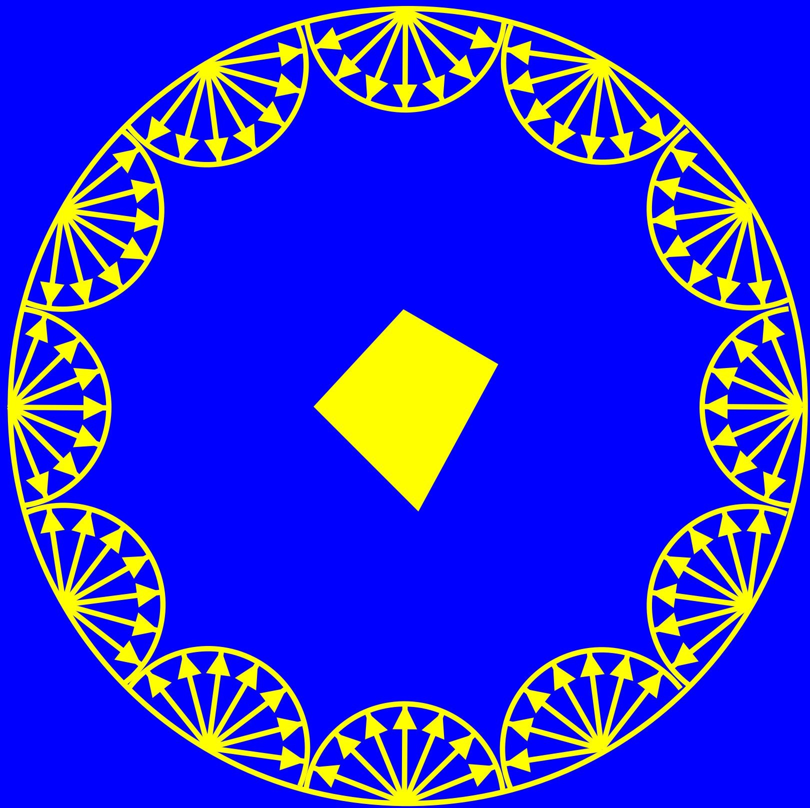


Image-based rendering

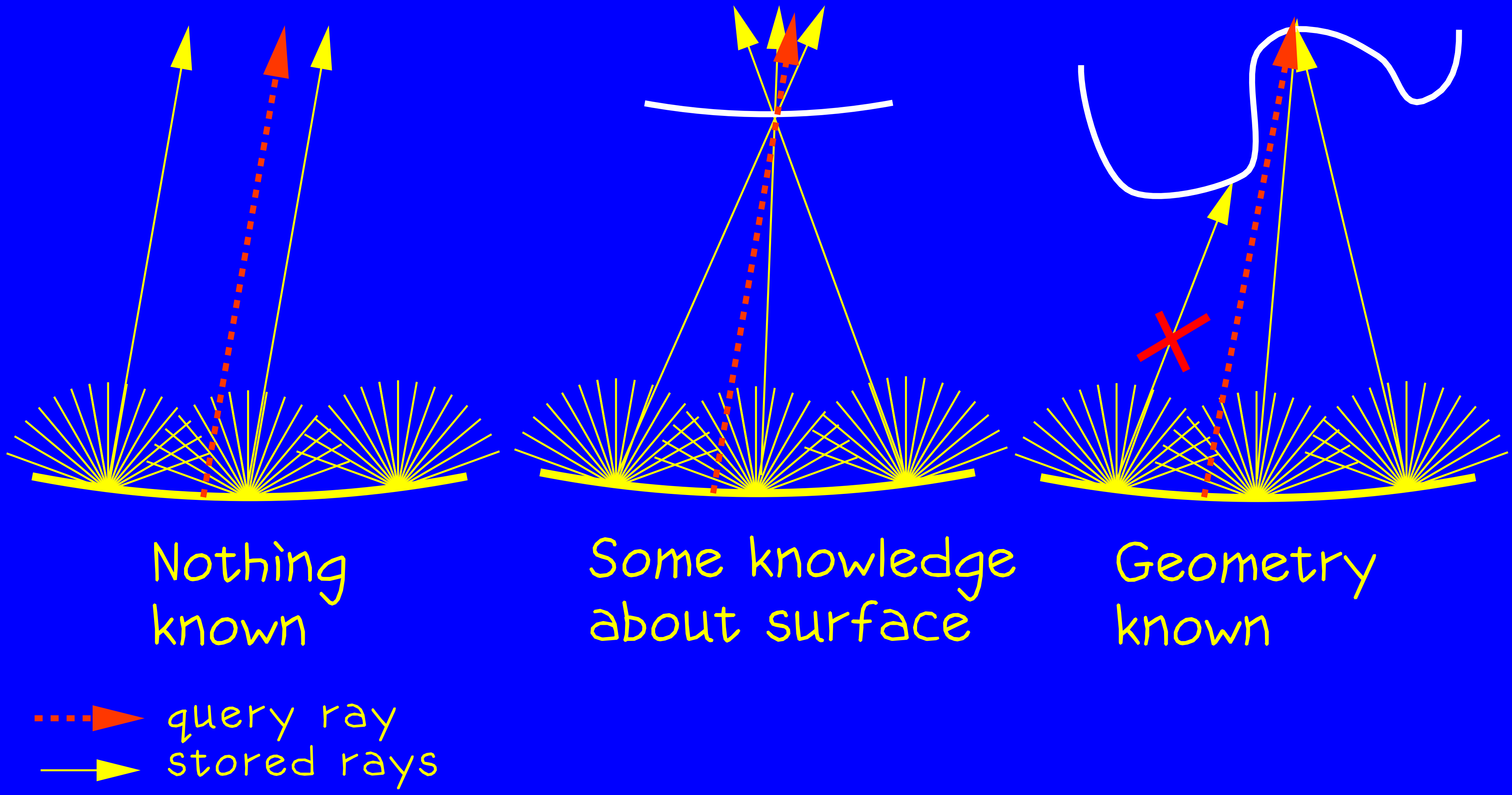
- Input: images
  Output: a new image, reuse the pixels
- Think of pixels as rays

SIGGRAPH '96
Light Field Rendering
The Lumigraph

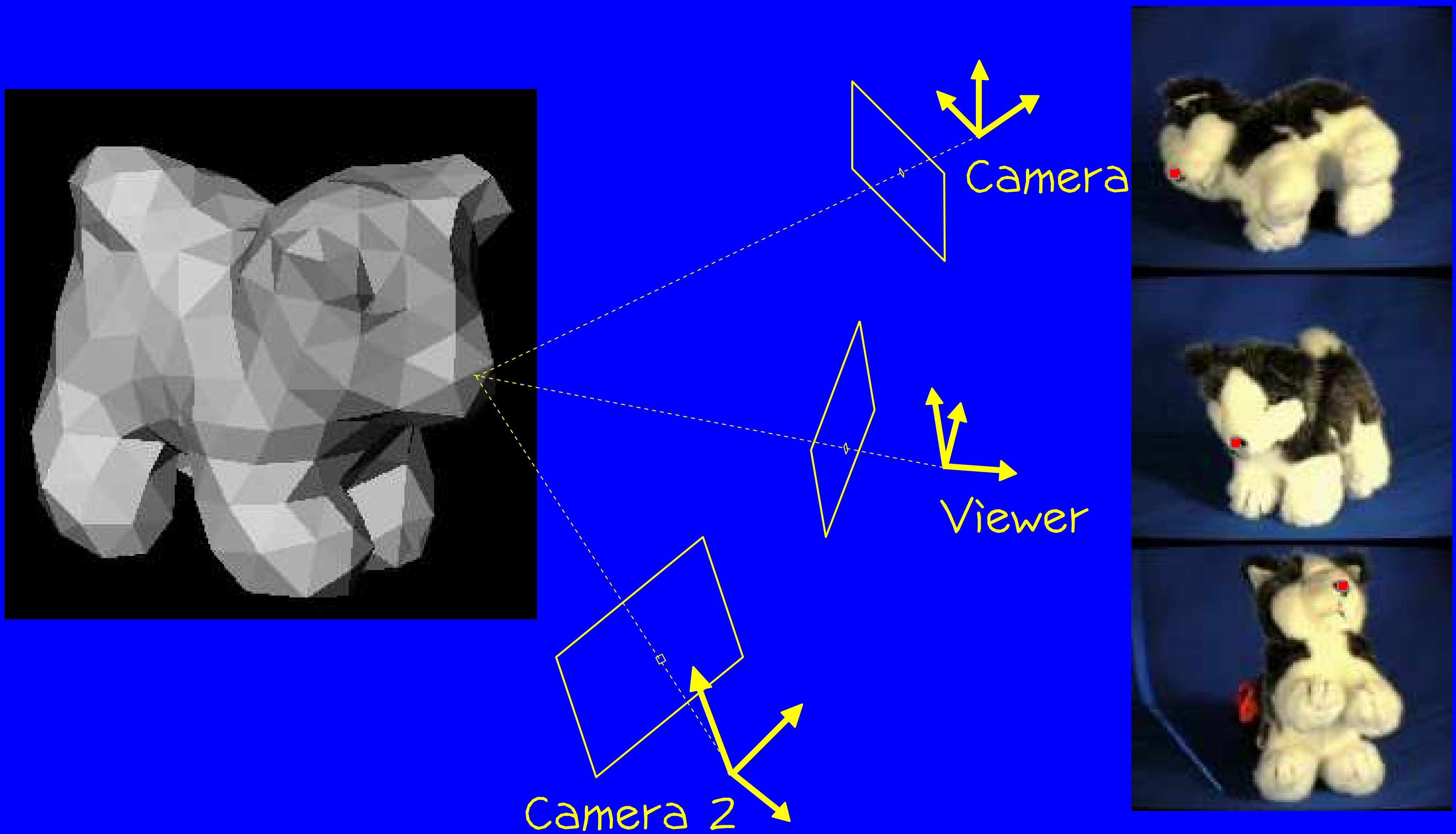


- Reorganize rays
- Which rays in input are right for a pixel?

Ray-surface intersection



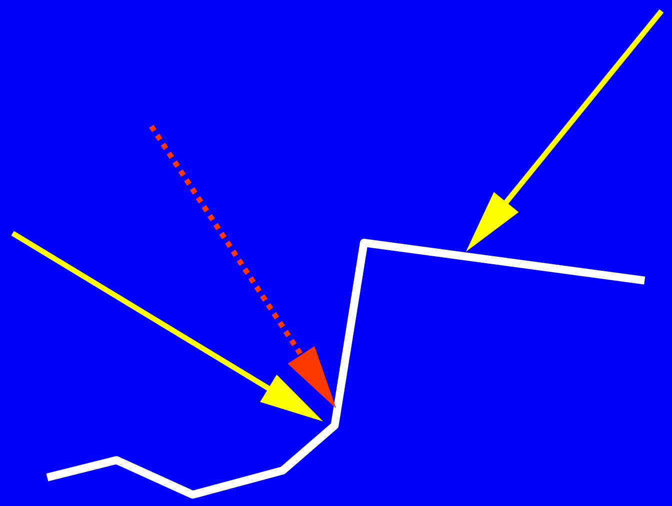
Find rays using projection

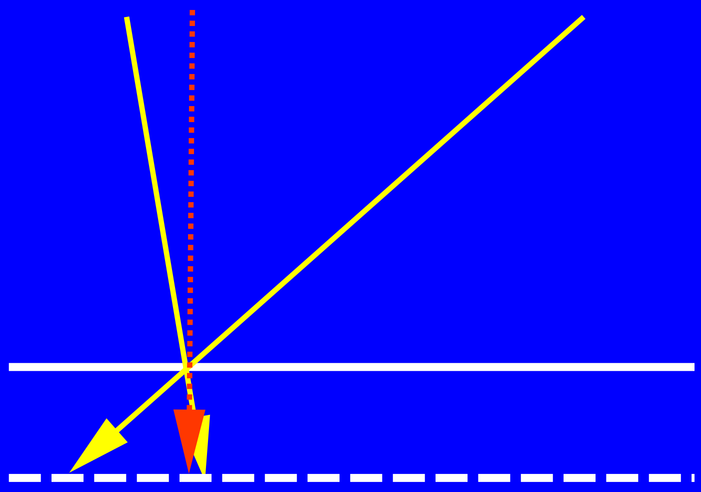


Ray direction

- View-dependent reflectance
  specularities, hair, ...

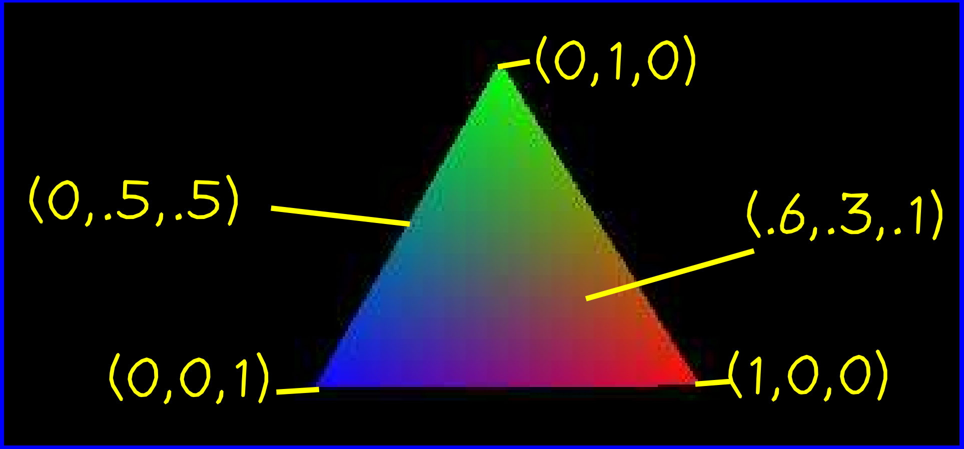
- Higher likelihood of correct match
  for surface intersection

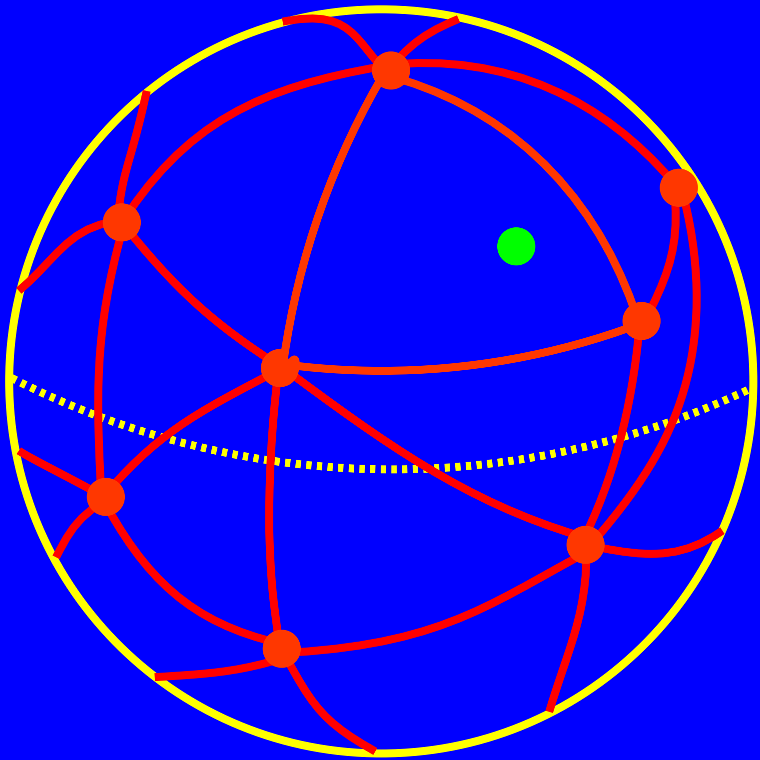




Directional weight

- Place the viewpoints on unit sphere
- Triangulate (Delaunay)
- Weight = barycentric coordinates

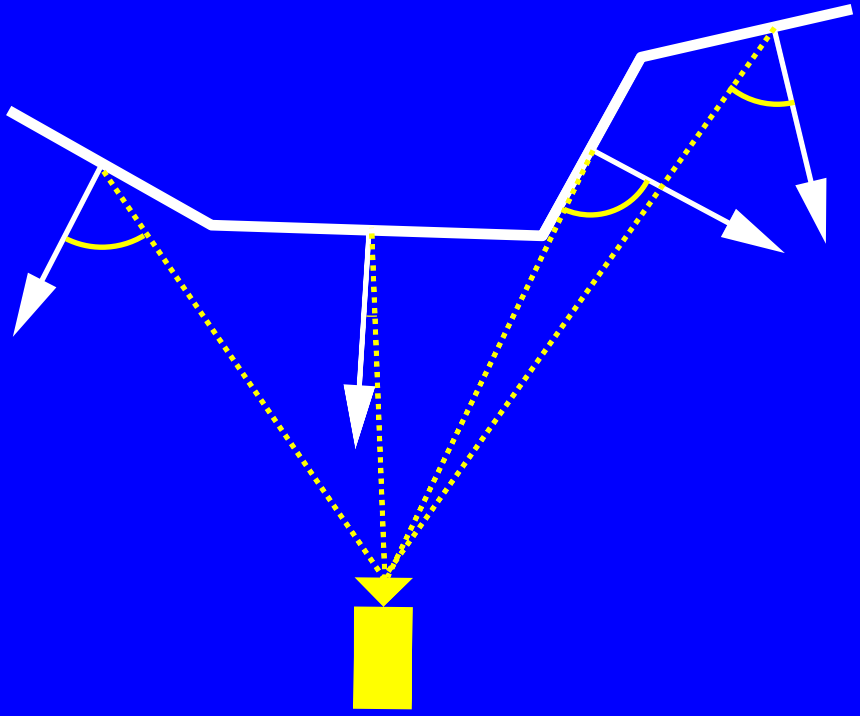


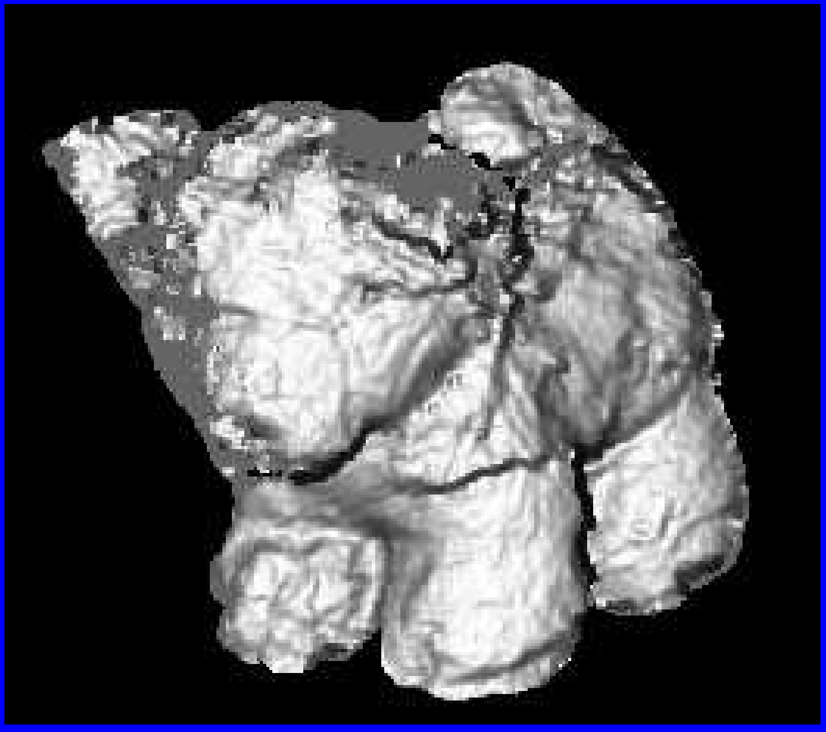


Quality weight

Downweight locations with poor sampling

Weight = cosine of the relative orientation





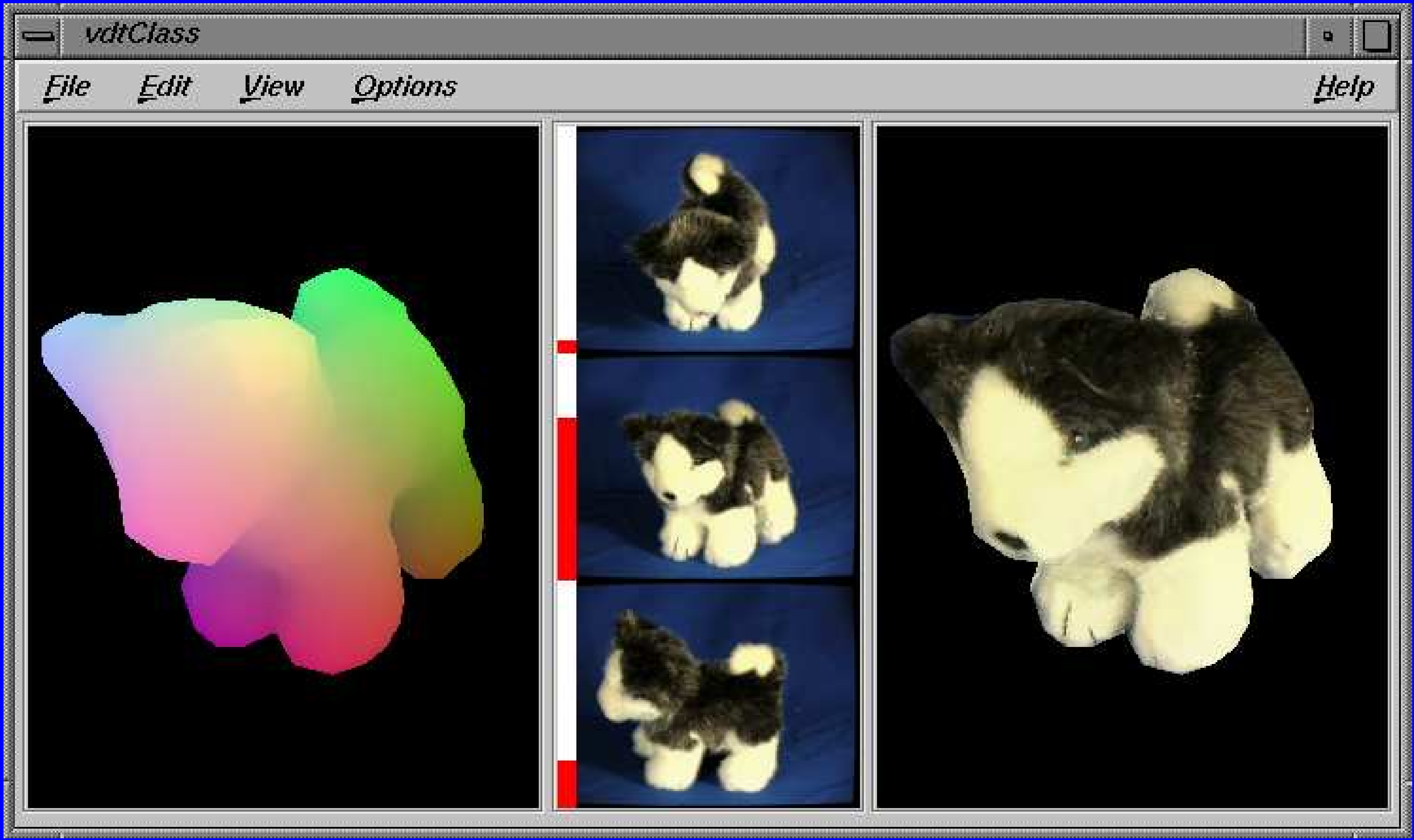
Feathering weight

- An image typically does not cover all the visible surface
- Lighting conditions differ
  => artifacts at mesh boundaries
- Decrease the influence of a view towards its silhouette

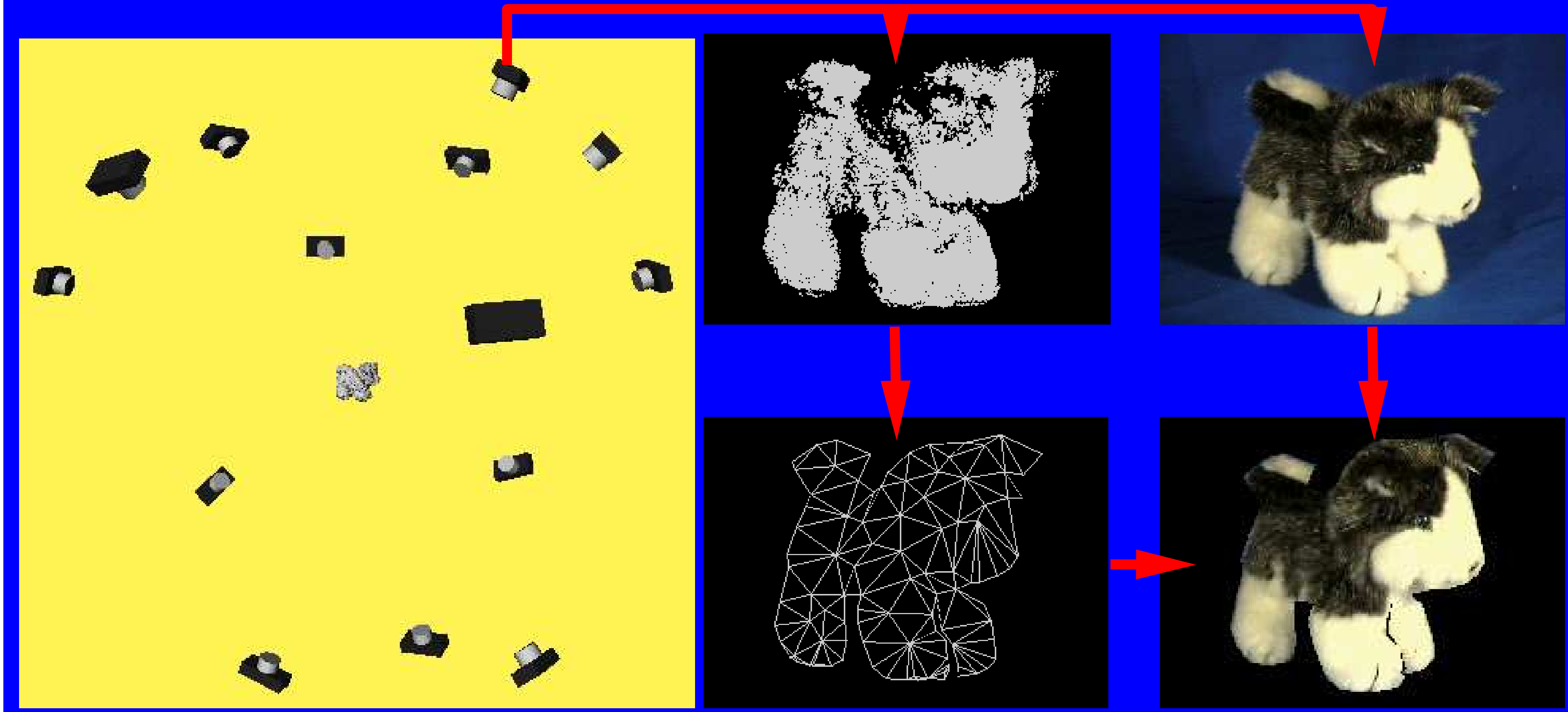
no feathering

with feathering

Our viewer



View-based models



VB Rendering: quick and dirty

- Render several views into the same image
  * hardware interpolates rays that intersect the surface close to where the query ray does
  * directions are not considered, no blending

Better results through blending

Overview of VBR

Choose 3 close views

Render meshes from the current viewpoint

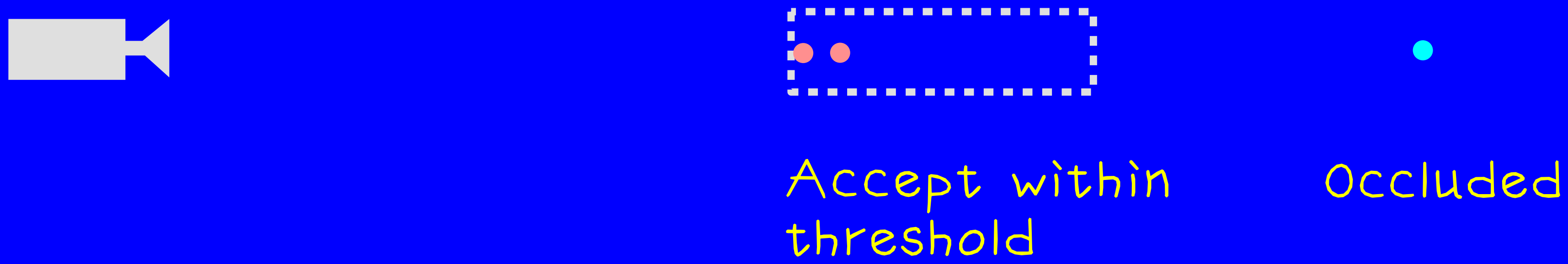
For each pixel

- * read it from each view
- * remove occluded ones
- * calculate a weighted average
- * paint the pixel

Soft Z-buffering for blending

- Hard(ware) Z-buffering
  closest point wins

- Soft Z-buffering
  several close ones survive



Summary

Two approaches:

- Integrate range data into a single model
  View-dependent texture mapping
- Model each view separately
  Integrate during display in screen space

Advantages

<u>Vs. traditional rendering</u>

- Realism: copying nature easier than synthesizing
- Speed: simple geometry, though it looks detailed

<u>Vs. Lumigraph-type IBR</u>

- Input and storage requirements small (because we have geometry)
- The geometry eases composition, interaction

Contributions

- Data acquisition
  - * implemented working range & color scanner
  - * adapted spacetime analysis to our scanner
- Registration
  - * global correspondence determination
  - * fast and robust multiview registration
- Surface reconstruction
  - * hierarchical space carving
- View-dependent rendering
  - * two approaches for view-dependent texturing of reconstructed surfaces