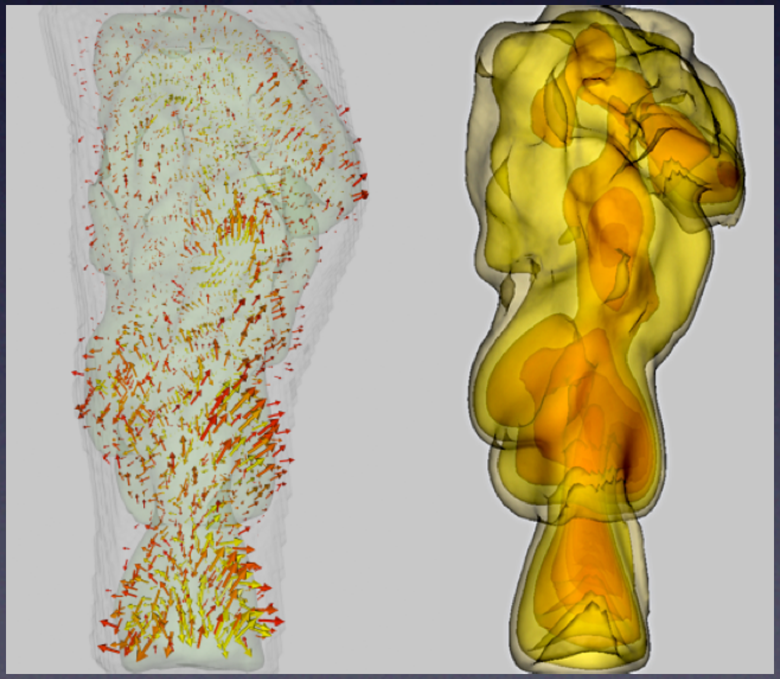
Schlieren Tomography

PSM Brainstorming Week
June 2011

Inhomogeneous translucent materials



Internal structure as well as surface geometry

$$f(\vec{x}) = \vec{b}$$

Image formation model

General function f

Linear A

Real world vs model

$$f(\vec{x}) = \vec{b}$$

Image formation model
General function f
Linear A
Real world vs model

Model parameters
Refractive index field
Reflection, scattering, etc
Discretisation

$$f(\vec{x}) = \vec{b}$$

Image formation model
General function f
Linear A
Real world vs model

Model parameters
Refractive index field
Reflection, scattering, etc
Discretisation

$$f(\vec{x}) = \vec{b}$$

Physical artefact
Measurements
Noise
Problem-specific

Image formation model
General function f
Linear A
Real world vs model

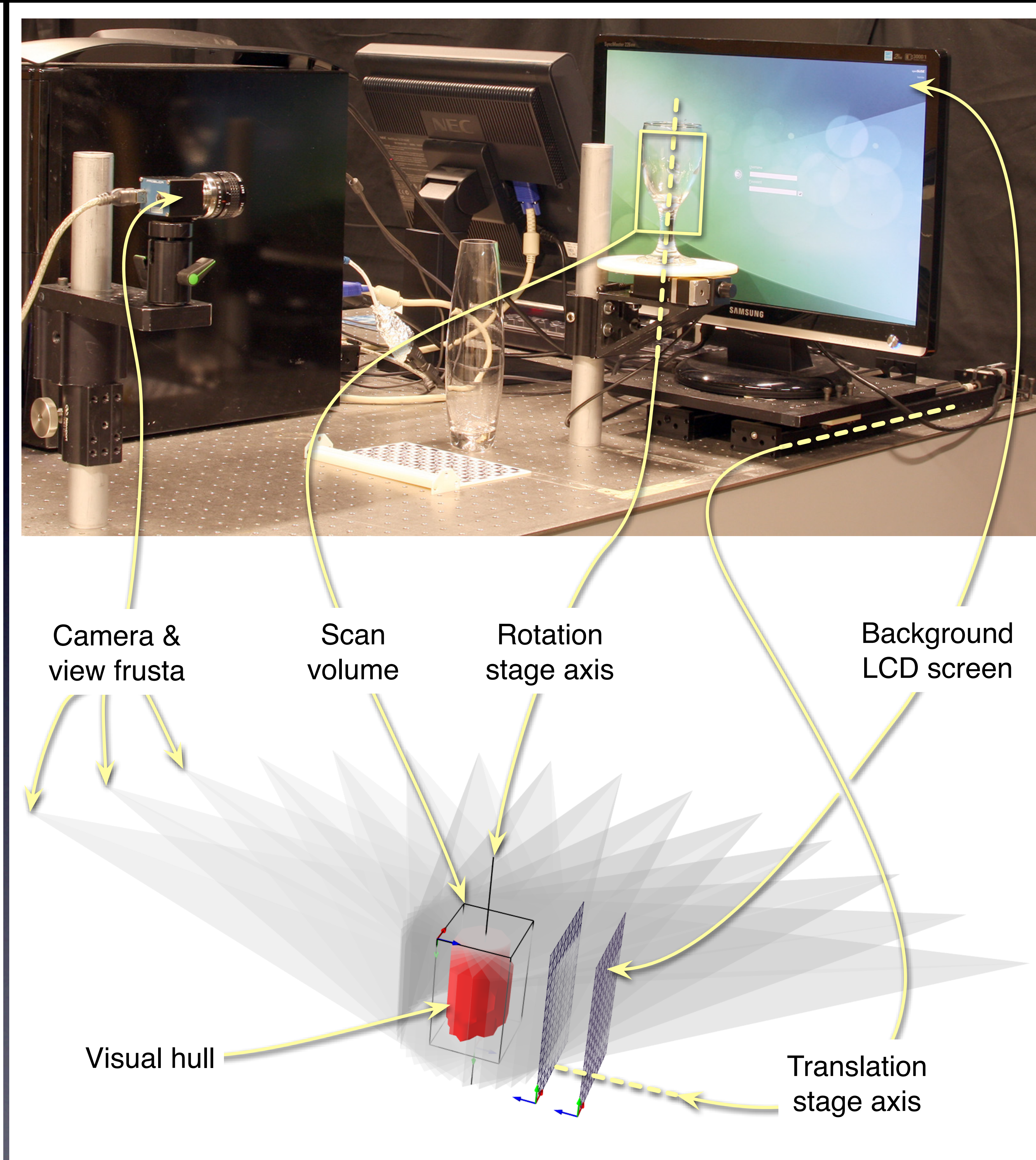
Model parameters
Refractive index field
Reflection, scattering, etc
Discretisation

$$f(\vec{x}) = \vec{b}$$

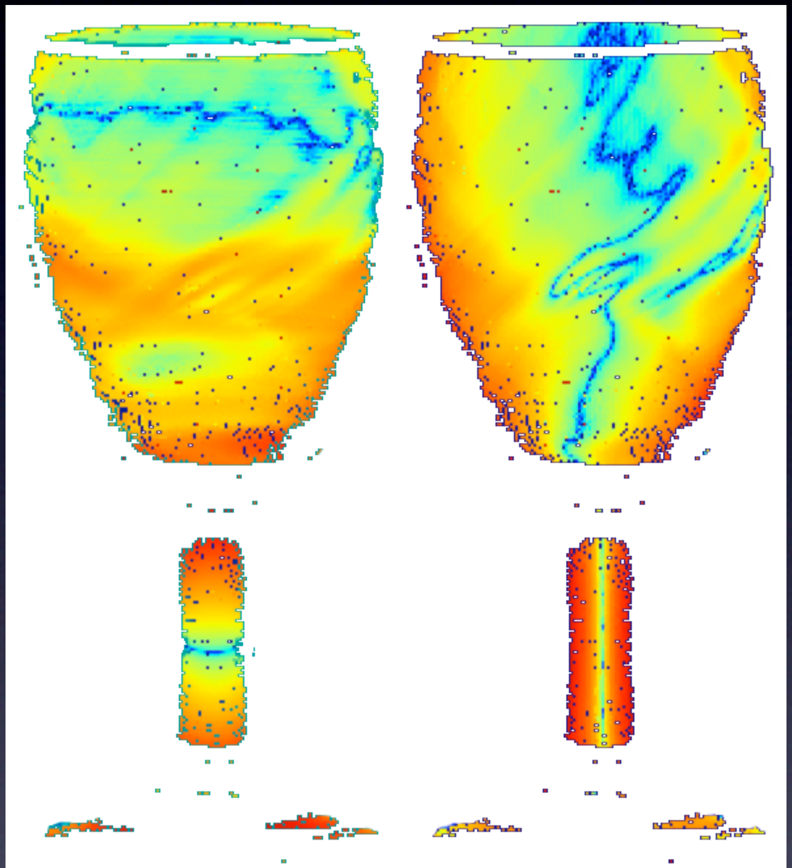
Physical artefact
Measurements
Noise
Problem-specific

Equation
Forward & inverse problems

Measurements



Acquired data



Very repeatable
80-90dB scan-to-scan SNR

Model Parameters

- Refractive index field (scalar n)
- Discretised onto voxel grid (local basis functions)
- Grid resolution (increase iteratively)
- Regular vs unstructured grid
- Alternative: global wavelet basis functions

Image formation model

- Ray equation of geometric optics
- To system of 1st order ODEs
- RK4 IVP (although we have exit data)
- Adaptive steps: quickly jump over empty/homogeneous regions
- Step-size plays large role in difficult regions (glancing angles)
- 20k rays/sec in uniform 1k voxels

$$\frac{d}{ds}\left(n\frac{d\vec{x}}{ds}\right) = \nabla n$$

$$\frac{d\vec{d}}{ds} = \nabla n$$

$$n\frac{d\vec{x}}{ds} = \vec{d}$$

Gradient computation

- Automatic numerical estimation easy but requires many function evals and large storage
- Raytracer can compute Jacobian: precompute gradient of local basis functions then integrate those gradients along ray path

The equation

- In general f(x) is nonlinear

$$\min_{\vec{x}\in\mathcal{X}} ||f(\vec{x}) - \vec{b}||_2$$

- BPDN

$$\min_{\vec{x}\in\mathcal{X}} ||x||_1 \ s.t. \ ||f(\vec{x}) - \vec{b}||_2 < \sigma$$

- Constrained X
- Cast as general optimisation problem
- Trust-region, Gauss-Newton, Interior point, Levenberg-Marquardt, Quasi-Newton...
- Implicit linear approximation made at each iteration of the solver

Linearisation

- Implicit local approximations in nonlinear solver
- Explicitly linearise at each iteration
- Approach used by Cha & Vest
- Trace rays, solve, retrace, resolve, retrace...
- Overlay pyramid resolution iteration, or just use AMG (questionable benefits over geometric multigrid)

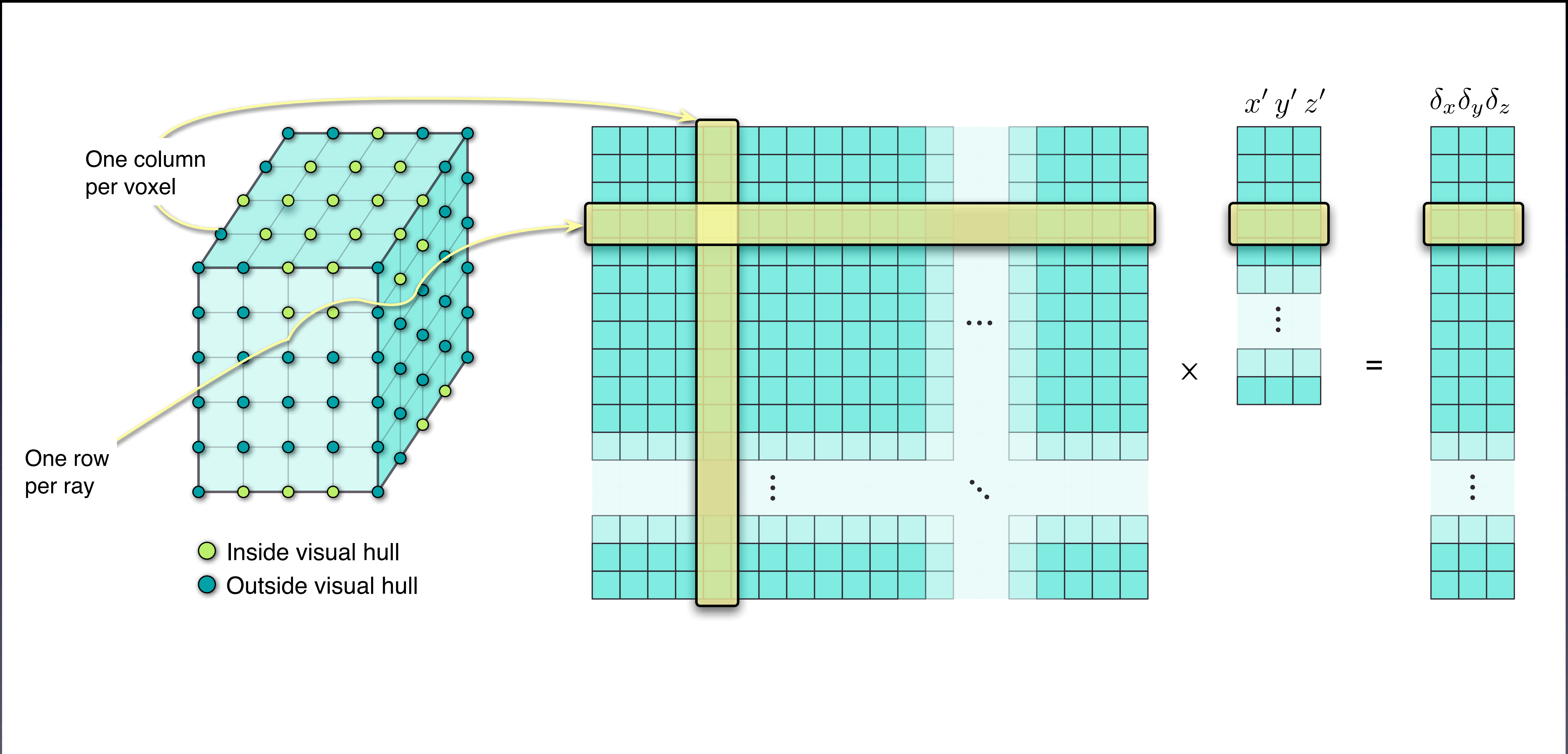
$$Ax = b$$

Solving for gradients

$$f(n_x) = \delta_x$$
$$f(n_y) = \delta_y$$
$$f(n_z) = \delta_z$$

- 3 equations, same f
- Integrate vector field to scalar
- Problems with nonuniform boundaries
- Replace vector equation with scalar
  - delta = angle between in/out ray
  - minimise norm of delta residual

The matrix A



One column per voxel
One row per ray
Inside visual hull
Outside visual hull
$x'\ y'\ z'$
$\delta_x \delta_y \delta_z$
×
=

Advantages of A

- Fast and easy to compute Ax and $A^T b$
- All the machinery of linear algebra
- Understand system properties (solution exists if sufficient rank, solution stable if condition number low, how does condition number change as camera geometry changes...)
- Only useful in debugging/analysis sense. For just solving system, matrix-free preferable.

Disadvantages of A

- Sheer size $V * K * 4\text{b}/1024^3 = 107\text{Gb}$

  # voxels (V): $512^3 * 1/10$

  # nonzeros/row (K): $\sqrt[3]{V} * 9$

- Sparse COO matrix: row/col indices highly compressible, add ~10%
- Need not store whole A: trace one camera and discard rays. Then trace again when we need to perform $A^T b$. Low mem for 2X time

Linear Solvers

- QR
  - least-squares solution
  - requires full matrix
  - adapt to IRLS for outlier reflections
- SART
  - trace and store A_i each iteration, accumulate products
- SPGL1
  - sparsity constraint on x
  - supports matrix-free operation but implementing $A^T b$ without matrix requires at least 2X tracing time per iteration

Nonlinear solvers

- No need for full matrix
- But useful for sparsity pattern of Jacobian
- However that pattern is fixed at initialisation, so would need large kernel support for conservative ray-tunnel
- Many function evals for Hessian

Conclusions

- Solve for indices rather than gradients
- Prefer matrix-free solver over explicit A
- Prefer analytical gradients to numerically computed gradients from solver