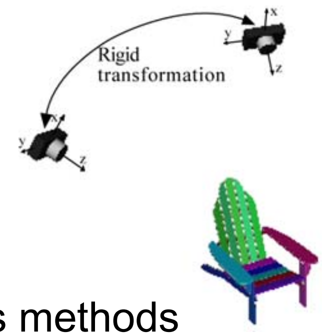
Projective Surface Matching of Colored 3D Scans

Kari Pulli Nokia Research Center & MIT CSAIL
Simo Piiroinen Nokia
Tom Duchamp University of Washington (Math & CS)
Werner Stuetzle University of Washington (Stat & CS)

Rigid registration

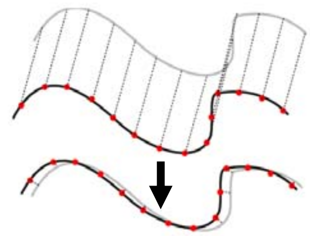
- Assume at least 2 scans
  - of the same object or scene
  - some overlap
- Find a rigid 3D transformation
  - so that samples corresponding to the same surface point are co-located
- Afterwards we can use various methods to reconstruct a single representation for
  - geometry
  - material properties



The generic motivation slide for 3D rigid registration: scan an object so it or the scanner move between scans, find the motion so the data aligns in some coordinate system.

ICP: match and align

- Correspondences
  - Heuristics to match scan points
    - take into account Euclidean distance, colors, local features, …
    - drop matches if over threshold, not coherent, at mesh boundary, …
- Alignment
  - find a rigid 3D motion that further reduces the distance between pairings
    - point-point: distance of paired points
    - point-plane: distance from a point to tangent plane
- Iterate until convergence



ICP (Iterated Closest Points) is a kind of generic brand name for several registration methods, consisting of two main parts.

First you use heuristics to find guesstimates which points might correspond to the same point on object surfaces, then you find a rigid 3D motion that brings the surfaces closer together, iterate as long as you improve.

Importance of color

- Color can disambiguate
  - geometry of overlap may not constrain registration
    - sweep surfaces (planar in one direction), surfaces of revolution (cylindrical)
- If alignment only due to geometry
  - color reconstruction is likely to be blurred

NOKIA CSAIL

As soon as people were able to capture both color and range, they noticed two reasons to use the color data in registration.

The first reason was that color can give you hints how to better register the scan, especially if the geometry is degenerate such that it doesn't constrain the registration. Examples would be planar, spherical, cylindrical surfaces.

The second reason is that if you also want to reconstruct color information, unless you explicitly do something to align color, that is, just align the geometry and hope for the best, you are likely to have more blurred color information.

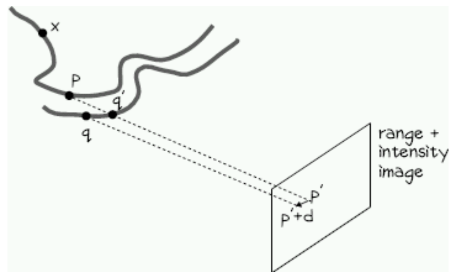
Projection for matching

- Projection used originally for acceleration
  - [Blais & Levine ’95]
  - no search of closest points
    - no building data structures and searching
    - usually O(log n) for each match
  - simply project one scan on the other scanner
    - requires knowledge of scanner calibration
    - usually O(1) for each match
  - then search for the best alignment

Many of the heuristics for point matching required 3D data structures and search, which always takes time, especially if you have a lot of points. Blais and Levine came up with an alternative. If your scanner scans regular range maps, as is usually the case, and you know the scanning geometry of your scanning device, you could project one of the scans to the range map of the other scan and simply pair a point with the one on which it projects in the other image. You still need to do the alignment part and iterate, but finding pairs can be considerably faster, and this heuristic is often equally valid to any other pairing heuristic.

Refine the projection match

- [Weik '97]
  - project point *p* and its intensity to *p'*
  - compare intensities of *p'* and *q'*
    - linearize: evaluate gradient on the image plane
    - take one step *d* to improve intensity match
  - move to *p'+d* on the plane, match *p* with with *q*
  - better matches lead to faster convergence

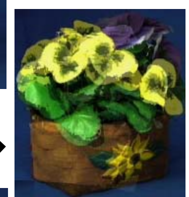


Weik noticed that you could look at the intensity data associated with his range data to make a quick local improvement, done in constant time. First project a point to the other scanner, look at the differences in intensity, linearize the change of intensity locally, and take a step to a direction where likely a better match for the color could be obtained. Only then pair the 3D points.

Use more context

- [Pulli '97]
  - processing each pixel in isolation yields mismatches
    - esp. where color data is flat
    - where local illumination differs
  - 2D alignment
    - project other scan to camera image plane
    - 2D image alignment (planar projective mapping)
  - 3D alignment
    - match 3D points falling at the same pixel
    - 3D alignment using pairs

3d meshes



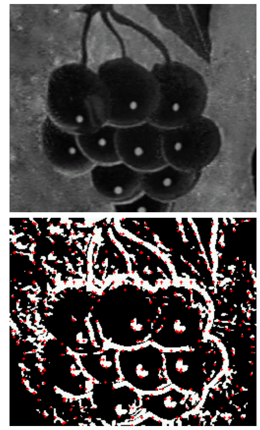
register 2D pictures, match corres-ponding 3D points

align paired points

I expanded that idea in my dissertation. Instead of making local changes separately at each pixel, the idea was to first align the projected color images and then the 3D points corresponding to overlapping 2D color points. The 2D alignment used a method developed for creating image panoramas. The method worked quite well in practise.

One more variation

- [Bernardini et al. ’01]
  - match only points with local intensity changes
    - avoids mismatches at flat colored areas
    - use cross-correlation
      - less affected by illumination changes
    - local search for local maximum
  - and align 3D points corresponding to matched 2D points



Bernardini, Martin, and Rushmeier refined this approach even further. Their observation was that you probably get better image matches if you do the matching in small windows, as opposed to the whole image, and concentrate on areas where the intensity has sudden changes and hence the match is more reliable than where the color data is almost constant.

What are we minimizing?

- Want to minimize the matching error of
  - geometry
  - colors
- But the errors are minimized separately
  - no single function to minimize
  - no guarantee of convergence
    - e.g., if there's any distortion of the geometry, the color and geometry errors are minimized at different poses
      - bounce back and forth
- We realized this in Dec '97
  - the new method mostly defined during '98
    - as well as the first implementation
    - but didn't have time to finish until last fall

So what's the problem, really?

The previous methods claim that they align colored scans so that the range and color information aligns.

That is true, but what is the error function that you minimize?

It turns out there isn't one. First color information is aligned, then based on color matches and pairings geometry is aligned. And when the data is good, this is usually enough.

But that fact that we couldn't even write out mathematically what we are minimizing was a problem (especially to some of my mathematically inclined co-authors). For example, if the color and range data do not agree, a different pose will yield the best color alignment than the one yielding the best geometry alignment.

We realized this around the time I was graduating, I worked on the problem after moving to Stanford to work on the Digital Michelangelo project, but never had time to finish the work until last fall.

Define the error function

- With idealized data and pose
  - points in different scans
    - project to the same spot on any plane
    - have the same depth and color
    - do not project outside the silhouette of the other
- Recipe for the algorithm
  - project scans
  - within overlap
    - minimize color and range
  - outside
    - minimize distance to silhouette
  - component weights

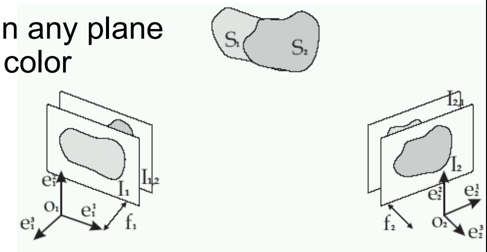


Figure 1. Two scans and their virtual cameras.

$$L(\mathbf{T}) = \int_{\Omega \cap \bar{\Omega}_T} \left((r - \bar{r}_T)^2 + \kappa_1 \|\mathbf{c} - \bar{\mathbf{c}}_T\|^2 \right) + \kappa_2 \int_{\bar{\Omega}_T} d^2(\mathbf{u}, \Omega).$$

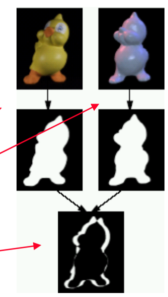
So let’s define the error function. Here’s a definition that works with a similar way of thinking than before. Assume that you have two colored meshes in 3D. Doesn’t even much matter with what kind of scanners they were obtained. Now if they align well, you should be able to project them to any plane, and at each point of the plane where both meshes project, both the color and the distance of the plane of the meshes should agree.

Further, if we associate a virtual pinhole camera with a scan, such that the camera would see no more of the object surface than the scan did, the second mesh would not project beyond the silhouette of the projection of the mesh associated with the virtual camera.

Now if the registration is not ideal, all these three components would have errors. The task is to transform one of the meshes, and its associated virtual camera, such that the error is minimized. We need to assign weights so we can relate colors to range to silhouette. The silhouette penalty we set based on how far outside the silhouette a point projects to.

Silhouettes

- Assumptions with using silhouettes
  - see the full object
  - no missing data
  - can separate background from foreground
- But when you can use them, they provide strong constraints
  - [Lensch et al. ’01]
  - registered a surface model to an image
    - extract image foreground
    - render the model white-on-black
    - evaluate with graphics HW (XOR)
    - minimize with Simplex



Silhouette information has been used for a long time as a basis for estimating object geometry from images. In registration Lensch et al. used only silhouettes to align a 3D model to color images.

3DIM 2005

Implementation

- Create textured meshes
  - associate a virtual pinhole camera with each scan
    - simple camera calibration to estimate camera location
- Render from camera viewpoint
  - only the “other” view changes
  - “this” view needs to be rendered just once
    - the virtual pinhole camera anchored with the view
- For each pixel
  - evaluate error (color, range, silhouette)
  - estimate image gradient (of each component)
  - analyze pixel flow as a function of 3D transformation **T**
    - propagate 2D error + gradient to 3D motion
- Minimize with Levenberg-Marquardt
  - just need error and its gradient w.r.t. **T**

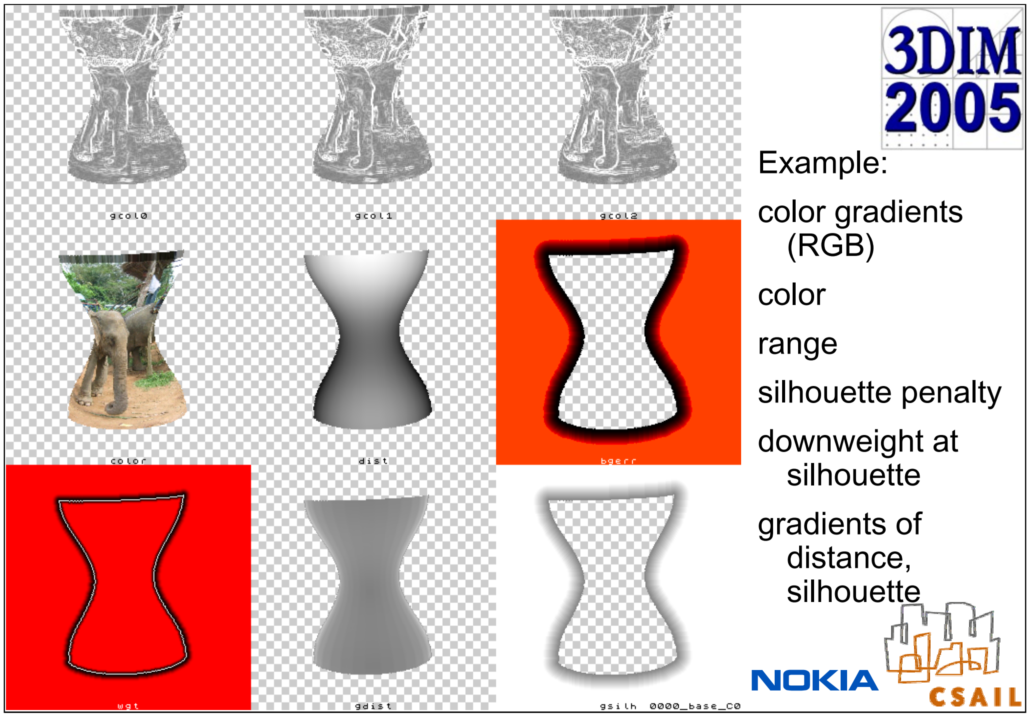
NOKIA CSAIL

Here’s how we implemented our system. First we take the range and color data, and convert it to textured triangle meshes.

We associate a virtual camera with each scan. If the range data is organized as a range map, you have a correspondence of 3D points with 2D rectangular array, and you can use a camera calibration algorithm to find good parameters for the virtual camera.

If we call the two scans A and B, we can first render A to its own virtual camera (using 3D graphics hardware). Since the camera of a mesh moves with the mesh, we never need to render that image again. Then we render B as seen from the camera of A. At each pixel where the images overlap, we evaluate range and color error, at pixels where B projects beyond A’s silhouette we add a penalty (we can precalculate how far each pixel is from the silhouette.

We then also estimate gradients of the data on the image plane (color gradient, range gradient, direction to the closest silhouette point), perform a kind of inverse image flow analysis to find a 3D transformation that would move pixels to directions where their errors would reduce using L-M, and iterate until convergence. We don’t treat either camera as special, but project A to B and vice versa.



Here we can see the various components we use from a test using two synthetic scans of a cylindrical vase with a texture map from a photograph.

Upper row: the color gradients (R,G,B bands)

Mid row: color data, range data, silhouette / background error (precalculated for each pixel, corresponds to a vector pointed to the closest point on the silhouette)

Bottom row: silhouette weight [just at the silhouette we downweight the error function, see paper for details], gradients for the range and silhouette penalty functions

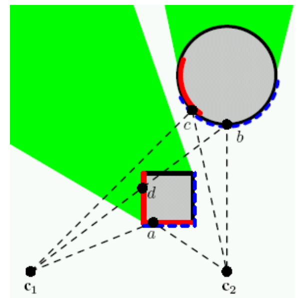
Color errors: start & end

On the left, misaligned projections generate ghosting, there are large structured color errors.

On the right the best solution is found. Since the color data has lots high frequencies, the sampling on two scans is different, the aliasing shows as (mostly) random noise.

Prevent false matches

- Surface seen in one camera (b)
  - may remain occluded in another
- Threshold approach
  - pair too far behind: mismatch (b,d)
    - [Weik ‘97, Pulli ‘97, Bernardini ’01, …]
    - need a larger threshold at start than in the end
- Extruded silhouettes
  - like shadow volumes
  - disallow matches covered by extended silhouettes (d,b)
  - conservative: disallows also (c)
  - but no need to choose a threshold



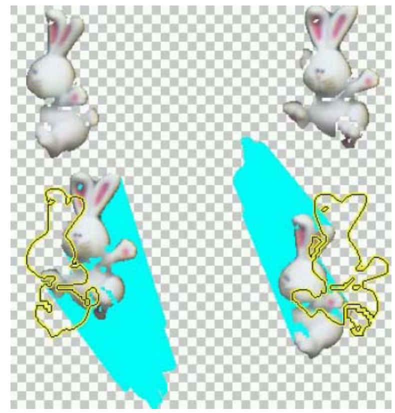
We also created a new method disallowing certain matches, even if they project to the same point on the image plane. A typical approach has been to compare distances, and if the difference is too large, disallow match. The problem is that it is difficult to choose a good threshold, as it depends also on how much alignment error there is before you start.

Our approach can be understood as virtual shadow volumes. Think of a virtual camera being a light source. The visible surface will then cast a shadow. Since we don’t know the extent of the object in the shadow volume, we take the conservative approach and prevent projection across the shadow volume.

In this case both cameras 1 and 2 see point a, but the false pairing of b and d on camera 1 is prevented. Also matches of c are prevented, but such configurations are quite rare.

False match prevention

- Implemented as
  - silhouette edges extruded as cyan polygons
  - skip any cyan pixels



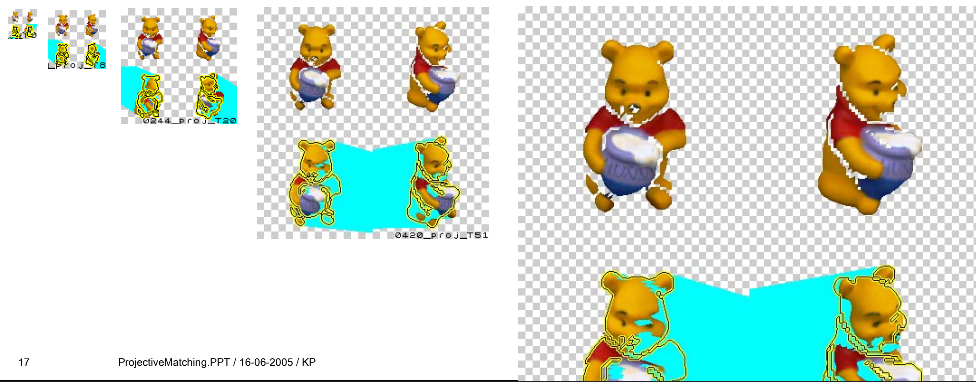
We implemented this by extruding the silhouette edges and rendering them in a special color. All pixels where such colored pixels project to are simply ignored.

Here, if we look at the lower left image, we have drawn the silhouette of the rabbit above, and we are looking at the camera associated with the rabbit. The other rabbit (from right top) is projected, mostly beyond the silhouette, with its shadow volume. In this case the error is mostly silhouette error.

Hierarchical minimization

3DIM 2005

- Faster
- More robust
  - error function is smoothed
  - fewer local minima



The registration was implemented in a hierarchical manner. It is pretty well known that the processing can be both much faster, and since the small versions are in effect low-pass filtered versions of the data, the registration is less likely to get stuck in a local minimum. A coarse level gives a rough alignment fast, which is then improved.

Summary

- Define good color & range registration
  - project to image plane
  - match projections
- Minimize the defined registration error
  - Levenberg-Marquardt
    - just for the small improvement around the current pose estimate
  - error gradients
    - direct numerical evaluation (sanity check)
    - lift 2D errors and gradients to 3D
    - not much difference in performance, method reported on the paper requires fewer evaluations => faster

As a summary: we have defined what we mean with good color and range registration, which is done by projecting data to image plane and matching projections.

For minimizing the error we actually tried several approaches, just to see that our implementation does the right thing. In the method presented in the paper we developed the maths for propagating the errors in the image plane, and gradients of the data on the image plane into 3D rotations and translations and solved it incrementally using Levenberg-Marquardt. We also evaluated error gradients directly numerically, as a sanity test. Both worked qualitatively in a similar fashion, the analysis way requires fewer evaluations, hence is faster.

Equations: how to read and use

- Levenberg-Marquardt needs
  - evaluate error
    - easy: just the different at a pixel (or dist. to silhouette)
  - Jacobian of error function w.r.t. 3D xform params
    - a matrix with 6 columns (d = [a b g x y z] 3 rot, 3 trans)
    - a bit different for different components (depth, color, silh.)
- Let's just analyze one color component
  - how much does the error change as a function of a rigid 3D motion?
    - gradient on the image plane (estimate numerically)
      times
      image flow of a surface point due to 3D motion

$$\nabla_{\mathbf{d}} \bar{\mathbf{c}}_{\mathbf{d}} = -\nabla_{\mathbf{u}} \bar{\mathbf{c}} \cdot \nabla_{\mathbf{d}} \psi_{\mathbf{d}}$$

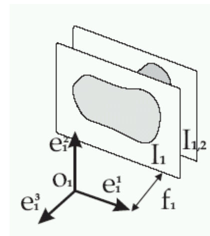
2-vector row/col gradient; 2x6 - matrix image flow due to xform d

This is just additional material on the equations if there's time in the presentation (and if you really want to understand the equations since you'd like to implement the method).

Projection 2D-3D equations

- Project image point **u** (at distance r) using camera at **o** with focal length *f*

$$\mathbf{P}(\mathbf{u}) = \mathbf{o} + \frac{r(\mathbf{u})}{f}(u\,\mathbf{e}^1 + v\,\mathbf{e}^2 + f\,\mathbf{e}^3)$$



and translate it around **m** by rotation **R** and translation **t**

$$\mathbf{T}(\mathbf{x}) = \mathbf{R} \cdot (\mathbf{x} - \mathbf{m}) + \mathbf{m} + \mathbf{t}$$

Note that in our case f is negative, as the image plane intersects e3 at (0,0,-f). View direction is along negative z as in OpenGL.

Projection 3D-2D equations

- Project 3D point **x** to image point *(u,v)*

$$\mathbf{\Pi}(\mathbf{x}) = f\left(\frac{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_1}{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_3}, \frac{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_2}{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_3}\right) = (u,v)$$

- Motion of point's projection when it moves in 3D

$$\nabla_{\mathbf{x}}\mathbf{\Pi} = \begin{bmatrix}\frac{\partial u}{\partial \mathbf{x}} & \frac{\partial v}{\partial \mathbf{x}}\end{bmatrix}^T$$

  - its *u* component (*v* is similar)

$$\begin{aligned}\frac{\partial u}{\partial \mathbf{x}} &= f\frac{\mathbf{e}_1}{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_3} - f\frac{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_1}{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_3}\frac{\mathbf{e}_3}{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_3} \\ &= \frac{1}{(\mathbf{x}-\mathbf{o})\cdot\mathbf{e}_3}(f\mathbf{e}_1 - u\mathbf{e}_3) \\ &= \frac{1}{\mathbf{r}(u,v)}(f\mathbf{e}_1 - u\mathbf{e}_3).\end{aligned}$$

Combine: Image flow

- How does a pixel move when a mesh (camera) is moved by a small transformation **d**?
  - project pixel from image plane to 3D
  - change in 3D by a small transformation
  - that transformation projected to image plane

$$\nabla_{\mathbf{d}}\psi = \nabla_{\mathbf{x}}\mathbf{\Pi} \cdot \nabla_{\mathbf{d}}(\mathbf{T} \circ \bar{\mathbf{P}})$$

$$\nabla_{\mathbf{x}}\mathbf{\Pi} = \frac{1}{r(\mathbf{u})}\begin{bmatrix} f\mathbf{e}^1 - u\mathbf{e}^3 \\ f\mathbf{e}^2 - v\mathbf{e}^3 \end{bmatrix}$$

2x3 - matrix
from 3D to 2D

$$\nabla_{\mathbf{d}}\mathbf{T}(\mathbf{x}) = \begin{bmatrix} 0 & z - m_z & -(y - m_y) & 1 & 0 & 0 \\ -(z - m_z) & 0 & (x - m_x) & 0 & 1 & 0 \\ (y - m_y) & -(x - m_x) & 0 & 0 & 0 & 1 \end{bmatrix}$$

3x6 - matrix
rotate 3D point x around “mass center” m, translate

Pbar is the point on the image, projected to 3D using the geometry of the pinhole camera

T transforms it

nabla x of Pi is the gradient of 2D image coordinates when 3D point x moves

nabla d of T is easy to interpret: with a small transformation, the translation is included directly, rotation induces a cross product

x, y, z are the 3D coordinates of the pixel at u,v

e1, e2, e3 are the orientation vectors of the virtual camera

Get the whole Jacobian

- Jacobian for a color component (e.g., red)

$$\nabla_{\mathbf{d}} \bar{\mathbf{c}}_{\mathbf{d}} = -\nabla_{\mathbf{u}} \bar{\mathbf{c}} \cdot \frac{1}{r(\mathbf{u})} \begin{bmatrix} f\mathbf{e}^1 - u\mathbf{e}^3 \\ f\mathbf{e}^2 - v\mathbf{e}^3 \end{bmatrix} \begin{bmatrix} 0 & z - m_z & -(y - m_y) & 1 & 0 & 0 \\ -(z - m_z) & 0 & (x - m_x) & 0 & 1 & 0 \\ (y - m_y) & -(x - m_x) & 0 & 0 & 0 & 1 \end{bmatrix}$$

1x6
row vector

- Need to modify for range and normal
  - since the transformation changes those as well
    - not just where they project to

$$\begin{aligned} \nabla_{\mathbf{d}} \bar{\mathbf{c}}_{\mathbf{d}} &= -\nabla_{\mathbf{u}} \bar{\mathbf{c}} \cdot \nabla_{\mathbf{d}} \psi_{\mathbf{d}}, && (12) \\ \nabla_{\mathbf{d}} \bar{r}_{\mathbf{d}} &= -\nabla_{\mathbf{u}} \bar{r} \cdot \nabla_{\mathbf{d}} \psi_{\mathbf{d}} + \mathbf{e}^3 \cdot \left(\nabla_{\mathbf{d}} \mathbf{T} \circ \bar{\mathbf{P}}\right), && (13) \\ \nabla_{\mathbf{d}} \bar{\mathbf{n}}_{\mathbf{d}} &= -\nabla_{\mathbf{u}} \bar{\mathbf{n}} \cdot \nabla_{\mathbf{d}} \psi_{\mathbf{d}} + \nabla_{\mathbf{d}} \mathbf{R} \cdot \bar{\mathbf{n}}. && (14) \end{aligned}$$

x, y, z are the 3D coordinates of the pixel at u,v

e1, e2, e3 are the orientation vectors of the virtual camera

L-M for incremental changes

- Use L-M to calculate small xform **d**
  - those equations were for projecting scan A on B
    - we want to use all information, so project A on B as well as B on A
    - when projecting B on A, just flip the sign of the gradient
  - append transformation **d** to the current estimate of the registration pose
  - draw a new image
  - repeat until convergence