The image as a virtual stage
Today

• Brief review of last class

• Inserting objects into legacy photos

• Using Blender
Mirror ball -> equirectangular
Mirror ball -> equirectangular

Mirror ball
Normals
Reflection vectors
Phi/theta of reflection vecs
Equirectangular
Phi/theta equirectangular domain
Mirror ball -> equirectangular

• Domain transformation in matlab
  – Create an interpolation function F with TriScatteredInterp
  – Compute values for each pixel in new domain

Pseudocode:

for i=1:d
    F = TriScatteredInterp(phi_ball, theta_ball, mirrorball(:,:,i));
    latlon(:,:,i) = F(phi_latlon, theta_latlon);
end
The polygonal mesh

• Discrete representation of a surface
  – Represented by vertices -> edges -> polygons (faces)
Insert these...
...into this
...into this
Or, remove this
Or, remove this
Problem statement

• From a single image, we want to seamlessly insert objects into and delete objects from an image while automatically handling perspective, occlusion, collision, and lighting.
Useful for...

• Home planning/redecoration
• Movies (visual effects)
• Video games
However...

- Tedious with current modeling tools
  - Blender, Photoshop, etc

- Alternatives require scene measurements and/or multiple photographs
What do we need?

• A 3D representation of the scene
  – Camera parameters (focal length, positioning, etc)
  – Scene geometry
  – Light positions and intensities

• Which requires
  – Projective geometry (vanishing pts, homographies, etc) [Camera, geometry]
  – Segmentation [Geometry]
  – Numerical optimization [Lights]
Overview

• Inserting objects
  – Perspective
  – Occlusion
  – Relighting
• Recap
• Unsolved problems
What’s wrong here?
That’s better, sort of...
Single view reconstruction

Many scenes can be represented as an axis-aligned box volume
Single view reconstruction

Many scenes can be represented as an axis-aligned box volume
Ideal example
Ideal example
Computing a projection

- Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[ K = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \]
Computing a projection

- Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[
e_i = (1, 0, 0)^T, \quad e_j = (0, 1, 0)^T, \quad e_k = (0, 0, 1)^T
\]
\[
v_i = KR e_i, \quad v_j = KR e_j, \quad v_k = KR e_k
\]
\[
(KR)^{-1} v_i = e^i, \quad (KR)^{-1} v_j = e^j, \quad (KR)^{-1} v_k = e^k
\]

\[
e_i^T e_j = e_j^T e_k = e_i^T e_k = 0
\]
\[
v_i^T K^{-T} RR^{-1} K^{-1} v_j = v_j^T K^{-T} RR^{-1} K^{-1} v_k = v_i^T K^{-T} RR^{-1} K^{-1} v_k = 0
\]
\[
v_i^T K^{-T} K^{-1} v_j = v_j^T K^{-T} K^{-1} v_k = v_i^T K^{-T} K^{-1} v_k = 0
\]
Computing a projection

• Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[ e_i = (1, 0, 0)^T, e_j = (0, 1, 0)^T, e_k = (0, 0, 1)^T \]

\[ v_i = KRe_i, v_j = KRe_j, v_k = KRe_k \]

\[ (KR)^{-1}v_i = e^i, (KR)^{-1}v_j = e^j, (KR)^{-1}v_k = e^k \]

\[ e_i^T e_j = e_j^T e_k = e_i^T e_k = 0 \]

\[ v_i^T K^{-T} RR^{-1} K^{-1} v_j = v_j^T K^{-T} RR^{-1} K^{-1} v_k = v_i^T K^{-T} RR^{-1} K^{-1} v_k = 0 \]

\[ v_i^T K^{-T} K^{-1} v_j = v_j^T K^{-T} K^{-1} v_k = v_i^T K^{-T} K^{-1} v_k = 0 \]
Computing a projection

- Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[ e_i = (1, 0, 0)^T, \quad e_j = (0, 1, 0)^T, \quad e_k = (0, 0, 1)^T \]

\[ v_i = KRe_i, \quad v_j = KRe_j, \quad v_k = KRe_k \]

\[ (KR)^{-1}v_i = e^i, \quad (KR)^{-1}v_j = e^j, \quad (KR)^{-1}v_k = e^k \]

\[ e_i^T e_j = e_j^T e_k = e_i^T e_k = 0 \]

\[ v_i^T K^{-T} RR^{-1} K^{-1} v_j = v_j^T K^{-T} RR^{-1} K^{-1} v_k = v_i^T K^{-T} RR^{-1} K^{-1} v_k = 0 \]

\[ v_i^T K^{-T} K^{-1} v_j = v_j^T K^{-T} K^{-1} v_k = v_i^T K^{-T} K^{-1} v_k = 0 \]
Computing a projection

• Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[ e_i = (1, 0, 0)^T, e_j = (0, 1, 0)^T, e_k = (0, 0, 1)^T \]

\[ v_i = KR e_i, v_j = KR e_j, v_k = KR e_k \]

\[ (KR)^{-1} v_i = e^i, (KR)^{-1} v_j = e^j, (KR)^{-1} v_k = e^k \]

\[ e_i^T e_j = e_j^T e_k = e_i^T e_k = 0 \]

\[ v_i^T K^{-T} RR^{-1} K^{-1} v_j = v_j^T K^{-T} RR^{-1} K^{-1} v_k = v_i^T K^{-T} RR^{-1} K^{-1} v_k = 0 \]

\[ v_i^T K^{-T} K^{-1} v_j = v_j^T K^{-T} K^{-1} v_k = v_i^T K^{-T} K^{-1} v_k = 0 \]
Computing a projection

- Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[ e_i = (1,0,0)^T, \ e_j = (0,1,0)^T, \ e_k = (0,0,1)^T \]

\[ v_i = KRe_i, \ v_j = KRe_j, \ v_k = KRe_k \]

\[ (KR)^{-1}v_i = e^i, \ (KR)^{-1}v_j = e^j, \ (KR)^{-1}v_k = e^k \]

\[ e_i^T e_j = e_j^T e_k = e_i^T e_k = 0 \]

\[ v_i^T K^{-T} RR^{-1} K^{-1} v_j = v_j^T K^{-T} RR^{-1} K^{-1} v_k = v_i^T K^{-T} RR^{-1} K^{-1} v_k = 0 \]

\[ v_i^T K^{-T} K^{-1} v_j = v_j^T K^{-T} K^{-1} v_k = v_i^T K^{-T} K^{-1} v_k = 0 \]
Computing a projection

- Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[e_i = (1, 0, 0)^T, \quad e_j = (0, 1, 0)^T, \quad e_k = (0, 0, 1)^T\]
\[v_i = KR e_i, \quad v_j = KR e_j, \quad v_k = KR e_k\]
\[(KR)^{-1} v_i = e^i, \quad (KR)^{-1} v_j = e^j, \quad (KR)^{-1} v_k = e^k\]

\[e_i^T e_j = e_j^T e_k = e_i^T e_k = 0\]
\[v_i^T K^{-T} RR^{-1} K^{-1} v_j = v_j^T K^{-T} RR^{-1} K^{-1} v_k = v_i^T K^{-T} RR^{-1} K^{-1} v_k = 0\]
\[v_i^T K^{-T} K^{-1} v_j = v_j^T K^{-T} K^{-1} v_k = v_i^T K^{-T} K^{-1} v_k = 0\]
Computing a projection

• Given 3 orthogonal VPs (at least two finite), can compute projection operator

\[
R = \begin{bmatrix} R_{1c} & R_{2c} & R_{3c} \end{bmatrix}
\]

\[
\lambda v_i = KRe_i
\]

\[
e_i = [1, 0, 0]^T
\]

\[
R_{iC} = \lambda K^{-1}v_i
\]
Projecting to image space

- Given $K$, $R$, and a position in 3D ($v_{object}$), we can find its corresponding 2D image location:

$$v_{image} = KRv_{object}$$
What about the reverse?

• Given K, R, and a 2D position on the image \((v_{\text{image}})\), what do we know about its 3D location?
What about the reverse?

• Given $K$, $R$, and a 2D position on the image ($v_{image}$), what do we know about its 3D location?

$$\lambda v_{object} = (KR)^{-1}v_{image}$$

• Implies a line along which the 3D point lies

• Allows for image space interactions to be localized in 3D!
Homographies

(Projective warping from one domain to another)
Image Rectification
Overview

• Inserting objects
  – Perspective & collision
  – Occlusion
  – Relighting
  – Animation

• Removing objects

• Recap + Unsolved problems
Modeling occlusions
User-defined boundary

- Tedious/inaccurate
- How can we make this better?
Refined segmentation
Segmentation with graph cuts

\[ \text{Energy}(\mathbf{y}; \theta, \text{data}) = \sum_{i} \psi_1(y_i; \theta, \text{data}) \sum_{i, j \in \text{edges}} \psi_2(y_i, y_j; \theta, \text{data}) \]
Segmentation with graph cuts

\[
\text{Energy}(y; \theta, \text{data}) = \sum_{i} \psi_1(y_i; \theta, \text{data}) \sum_{i,j \in \text{edges}} \psi_2(y_i, y_j; \theta, \text{data})
\]
A spectral approach
A spectral approach
A spectral approach

• Create NxN matrix describing neighboring pixel similarity (Laplacian matrix, L)
• Extract “smallest” eigenvectors of L
• Segmentation defined by linear combination of eigenvectors
  – Scribbles as constraints
A spectral approach
A spectral approach
Segmentations as “billboards”
Segmentations as “billboards”
Rendering via ray tracing

- Camera
- Light Source
- Scene Object
- Image
- View Ray
- Shadow Ray
Rendering via ray tracing

Camera → View Ray → Scene Object → Shadow Ray → Light Source → Image
Insertion without relighting
...with relighting
Understanding light: our approach

• Hypothesize physical light sources in the scene
  – Physical $\rightarrow$ CG representations of light sources
    found in the real world (area lights, etc)

• Visible sources in image marked by user
  – Refined to best match geometry and materials

• User annotates light shafts; direction vector
  – Shafts automatically matted and refined
Lighting estimation
Lighting estimation
Lighting estimation
Light refinement

- Match original image to rendered image
Initial light parameters
Refined light parameters
External light shafts
External light shafts
External light shafts

Shadow matting via Guo et al. [2011]
External light shafts

• Can we find the direction of the shaft in 3D?
Example on board

\[ \mathbf{x}_2 = \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} \]

\[ \mathbf{x}_1 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \]
Light shaft result
Light shaft result
Inserting objects

- Representation of geometry, materials and lights compatible with 3D modeling software
- Two methods of insertion/interaction
  - Novice: image space editing
  - Professional: 3D modeling tools (e.g. Maya)
- Scene rendered with physically based renderer (e.g. LuxRender, Blender’s Cycles)
Final composite

- Additive differential technique [Debevec 1998]

\[
\text{composite} = M .* \text{R} + (1-M) .* \text{I} + (1-M) .* (\text{R} - \text{E}) .* c
\]

I (background)  composite

R (rendered)  E (empty)  M (mask)
Blender demos
Putting it all together

Video
Unsolved problems

• Can we “do better” with
  – Multiple images?
  – Videos?
  – Depth?
• Better scene understanding?
• How to insert image fragments (Poisson Blending style)?