Image Alignment and Stitching

CS 450: Introduction to Digital Signal and Image Processing

Image Alignment

- Goal: warp one image to match another so their content aligns
- Uses:
  - Comparison
  - Stitching
  - And many more…

Alignment / Registration

- Choose the set of transformation parameters that causes the images to align "best"
- Sometimes called *image registration*
- Two common approaches:
  - Direct / Area based
  - Indirect / Feature based

Direct Alignment

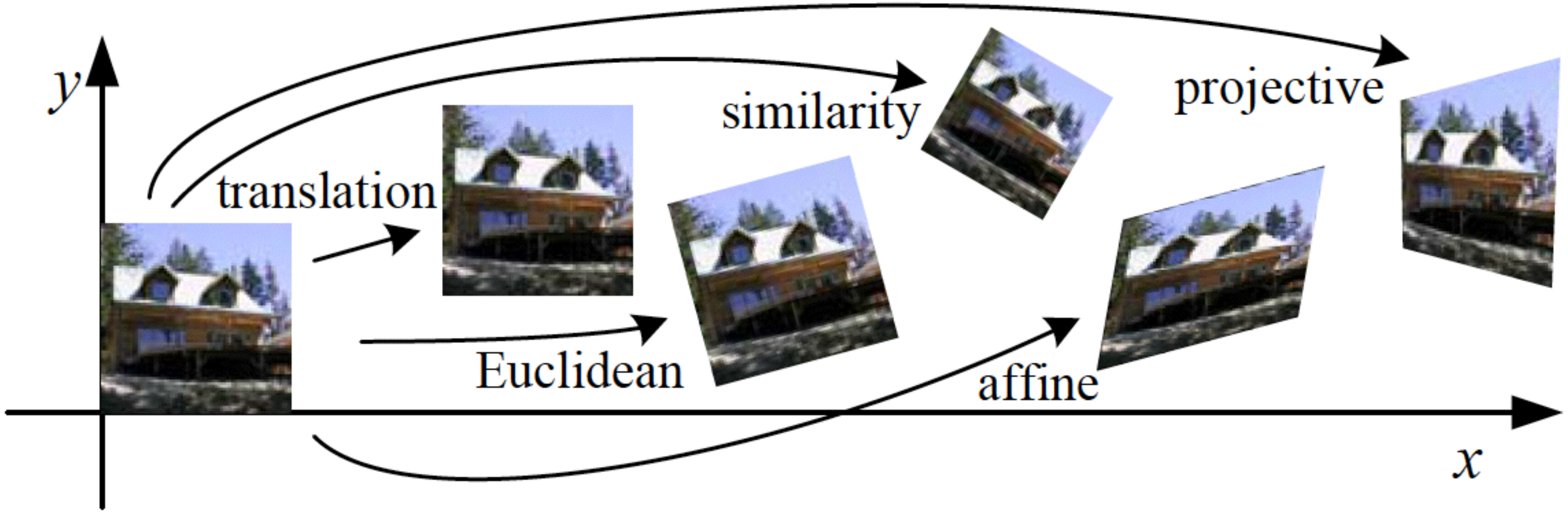
- *Direct* or *area-based* compares the pixels in the area of overlap
- Different metrics:
  - Sum of absolute differences
  - Sum of squared differences
  - Normalized cross-correlation
  - And many others…
- Usually done with a “greedy” optimization (more on that later)

Feature Alignment

- *Indirect* or *feature-based* matches only identified features in the images
- Steps:
  - Find feature points in *each* image (already talked about)
  - Match feature points *between* images
  - Solve for the transformation that best maps the matching feature points to each other



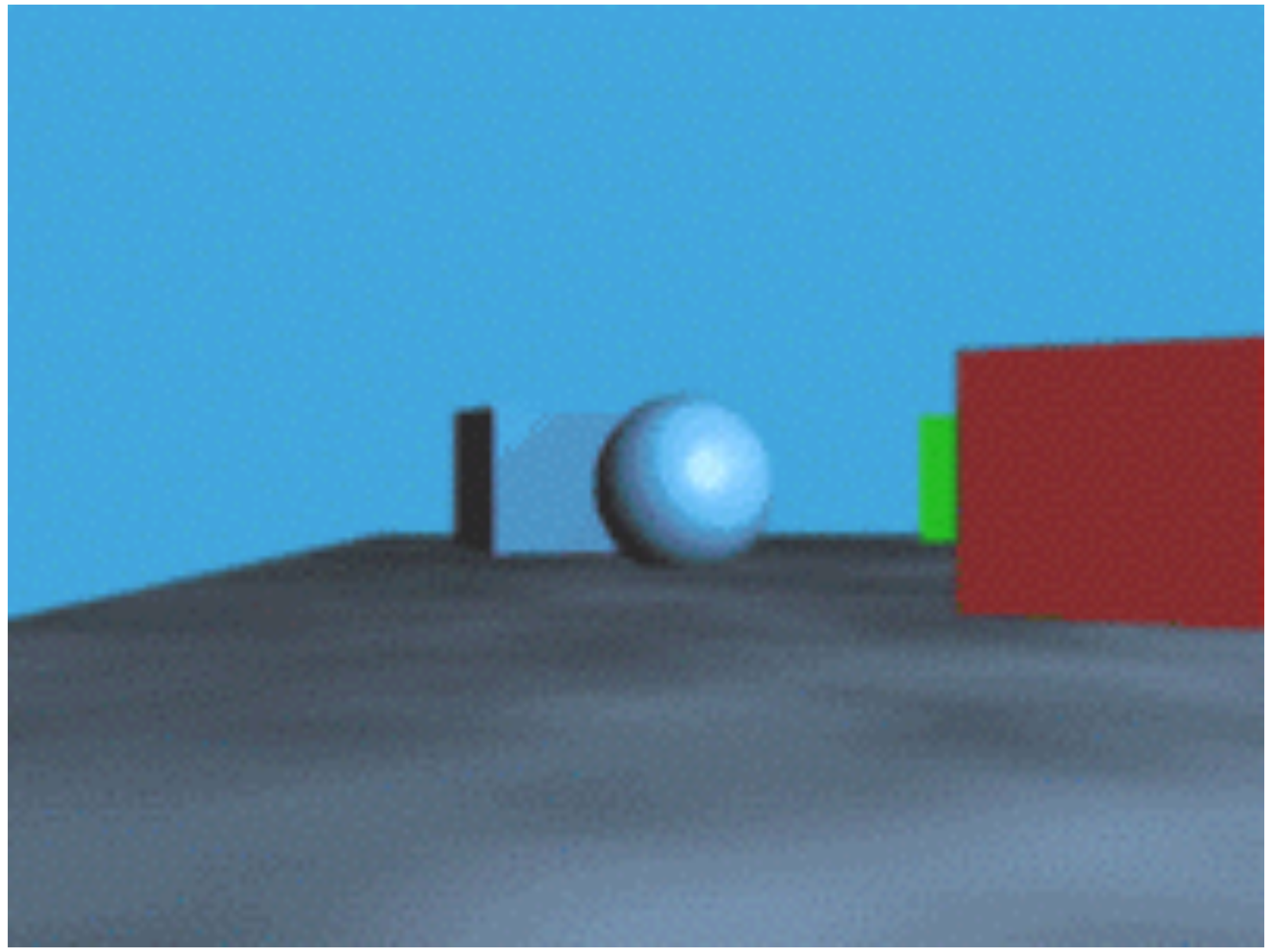
Linear Transformations (Revisited)



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Warping Images of 3D Scenes

- Problem: In general we can't easily warp images of 3D scenes to each other due to *parallax*
  - Nonuniform transformation
  - Depends on unknown depth (can figure out depth from multiple views, but that's a different problem)
  - May not even be 1-to-1
- *But it can be done for specific situations*



3D Perspective Projection

World-to-camera transformation

$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ Z_c/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \underbrace{\begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} \boxed{\begin{matrix} X_w \\ Y_w \\ Z_w \end{matrix}} \\ 1 \end{bmatrix}$$

Normalize Project Rotate Translate

3D Perspective Projection

World-to-camera transformation

$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ 1 \end{bmatrix} \sim \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Camera}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Project}} \overbrace{\underbrace{\begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Rotate}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Translate}}} \begin{bmatrix} \boxed{\begin{matrix} X_w \\ Y_w \\ Z_w \end{matrix}} \\ 1 \end{bmatrix}$$

K **P** **R** **T**

3D Perspective Projection

$$\mathbf{p} = \mathbf{K}\ \mathbf{P}\ \mathbf{R}\ \mathbf{T}\ \mathbf{p}_w$$

$$\mathbf{p} = \mathbf{M}\ \mathbf{p}_w$$

Two Perspective Images

- Two images of the same scene
  - Possibly different positions
  - Possibly different orientations
  - Possibly different camera parameters
  - Content may only overlap partially

$$\mathbf{p}_1 = \mathbf{M}_1 \ \mathbf{p}_w = \mathbf{K_1} \ \mathbf{P} \ \mathbf{R_1} \ \mathbf{T_1} \ \mathbf{p}_w$$

$$\mathbf{p}_2 = \mathbf{M}_2 \ \mathbf{p}_w = \mathbf{K_2} \ \mathbf{P} \ \mathbf{R_2} \ \mathbf{T_2} \ \mathbf{p}_w$$

Aligning Two Images

- Can transform one image to the other by just inverting one of the projections
- What is the shape of the resulting matrix?
- What's the problem with this?

$$\mathbf{p}_1 = \mathbf{M}_1\ \mathbf{p}_w = \mathbf{K_1}\ \mathbf{P}\ \mathbf{R_1}\ \mathbf{T_1}\ \mathbf{p}_w$$

$$\mathbf{p}_2 = \mathbf{M}_2\ \mathbf{p}_w = \mathbf{K_2}\ \mathbf{P}\ \mathbf{R_2}\ \mathbf{T_2}\ \mathbf{p}_w$$

so

$$\mathbf{p}_2 = \mathbf{M}_2\ \mathbf{M}_1^{-1}\ \mathbf{p}_1$$

Aligning Two Images

- General problem:
  You can't just invert a rank-deficient matrix without additional information

- Special cases that work:

  - Pure camera rotation
    (no translation, so no parallax)

  - You know that the "scene" is a planar surface

$$\mathbf{p}_1 = \mathbf{M}_1\ \mathbf{p}_w = \mathbf{K_1}\ \mathbf{P}\ \mathbf{R_1}\ \mathbf{T_1}\ \mathbf{p}_w$$

$$\mathbf{p}_2 = \mathbf{M}_2\ \mathbf{p}_w = \mathbf{K_2}\ \mathbf{P}\ \mathbf{R_2}\ \mathbf{T_2}\ \mathbf{p}_w$$

so

$$\mathbf{p}_2 = \mathbf{M}_2\ \mathbf{M}_1^{-1}\ \mathbf{p}_1$$

Case 1: Pure Camera Rotation

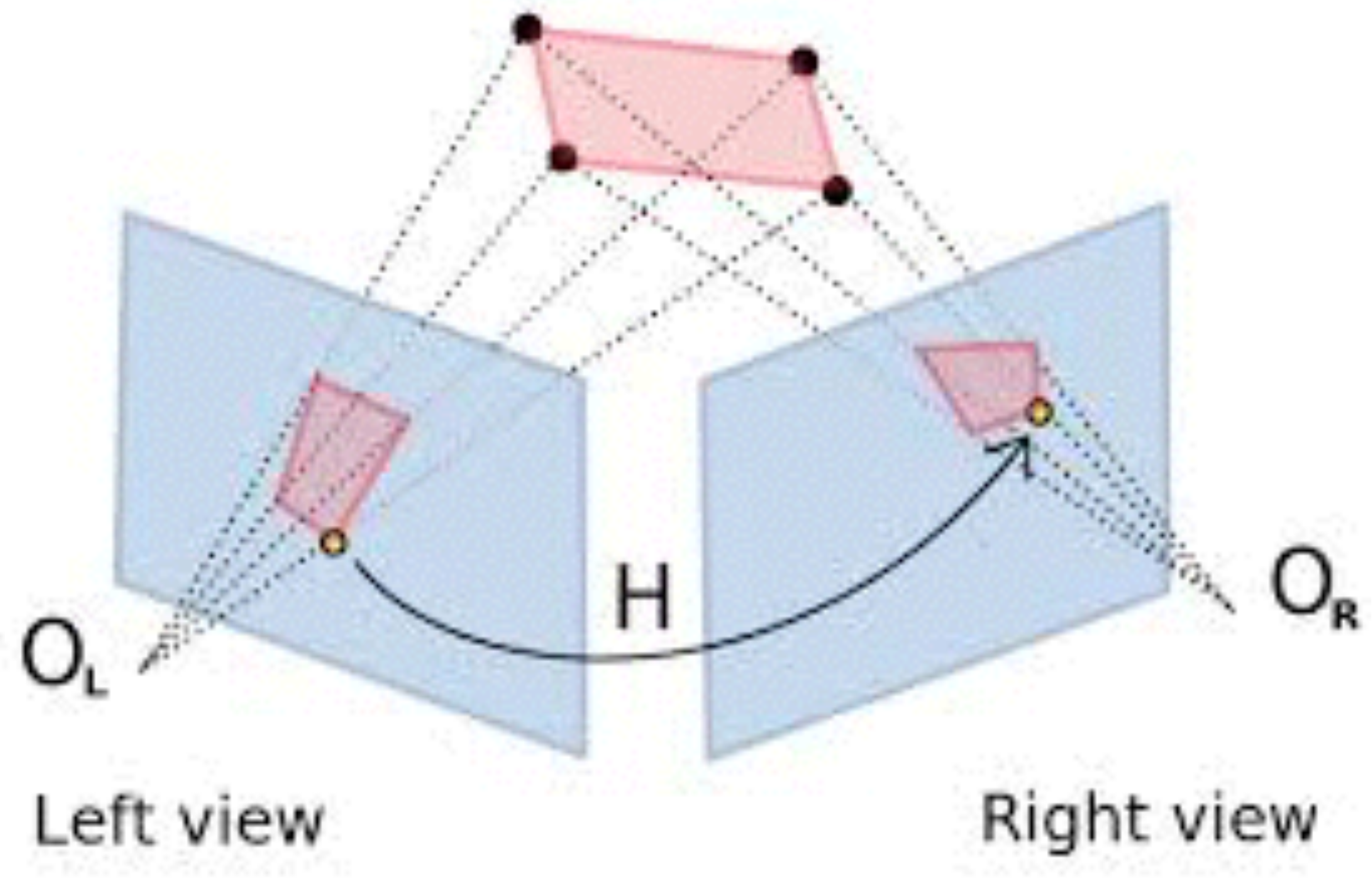
- If the camera only rotates around its own focal point and there isn't any translation, *there isn't any parallax*
- This is the basis for *circular panoramas*
- In practice, you can sometimes get away with a little camera movement if things are far away

$$\mathbf{p}_2 = \mathbf{M}_2\ \mathbf{M}_1^{-1}\ \mathbf{p}_1$$

$$\mathbf{p}_2 = \mathbf{R}_{12}\ \mathbf{p}_1$$

Case 2: Planar Surface

- A *homography* is a linear mapping between homogeneous coordinates
- Projection from a planar surface to another under perspective is a homography
  - From a surface to an image plane
  - From the same surface to another image plane
  - Between the projections onto the two image planes



$$\mathbf{p}_2 = \mathbf{M}_2 \ \mathbf{M}_1^{-1} \ \mathbf{p}_1$$

$$\mathbf{p}_2 = \mathbf{H} \ \mathbf{p}_1$$

Homographies

- Unique only up to a constant factor
  ➡ Only 8 degrees of freedom
- Generally full rank and invertible
  ➡ This means 1-to-1 mapping

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Computing Homographies

- Can compute analytically if you know
  - Relative positions of cameras
  - Relative orientations of cameras
  - Camera parameters
  - Orientation of the observed plane
  - Distance to the observed plane
- Problem: you may not know all these
- What if you just have the two images?

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Computing Homographies

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \qquad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + 1}$$

$$\begin{aligned} h_{00}x + h_{01}y + h_{02} - x'\left(h_{20}x + h_{21}y + 1\right) &= 0 \\ h_{10}x + h_{11}y + h_{12} - y'\left(h_{20}x + h_{21}y + 1\right) &= 0 \end{aligned}$$

Computing Homographies

$$
\begin{aligned}
h_{00}x + h_{01}y + h_{02} - x'\left(h_{20}x + h_{21}y + 1\right) &= 0 \\
h_{10}x + h_{11}y + h_{12} - y'\left(h_{20}x + h_{21}y + 1\right) &= 0
\end{aligned}
$$

$$
\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}
\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix}
= \begin{bmatrix} x' \\ y' \end{bmatrix}
$$

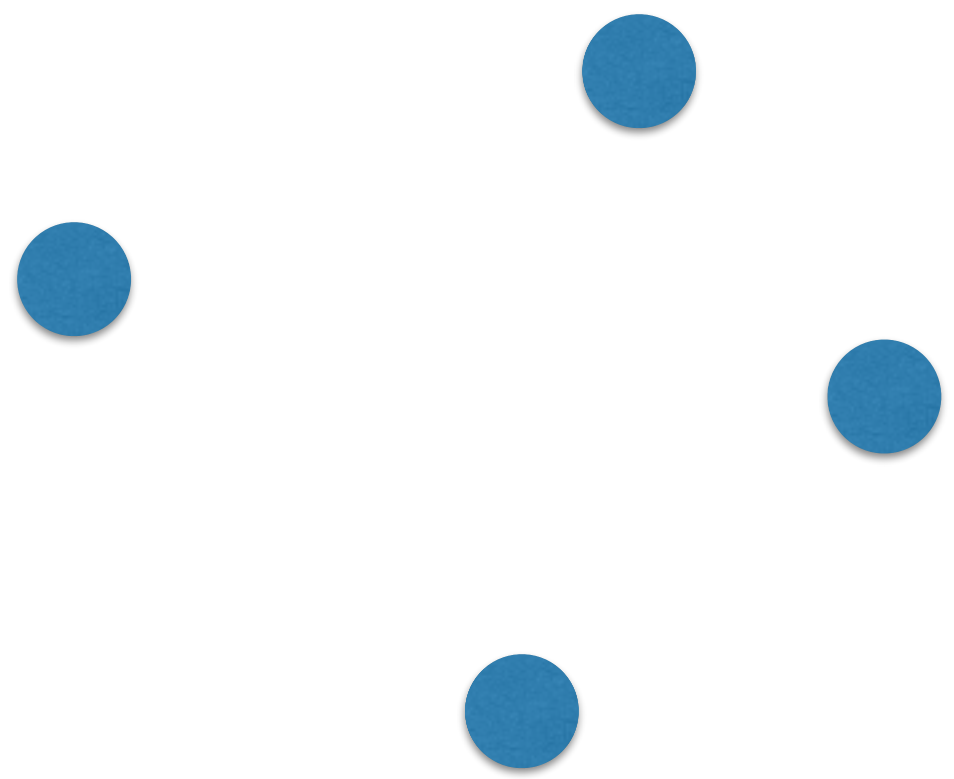
The Four-Point Algorithm

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2 x_2 & -x'_2 y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2 x_2 & -y'_2 y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3 x_3 & -x'_3 y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3 x_3 & -y'_3 y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4 x_4 & -x'_4 y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4 x_4 & -y'_4 y_4 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}$$

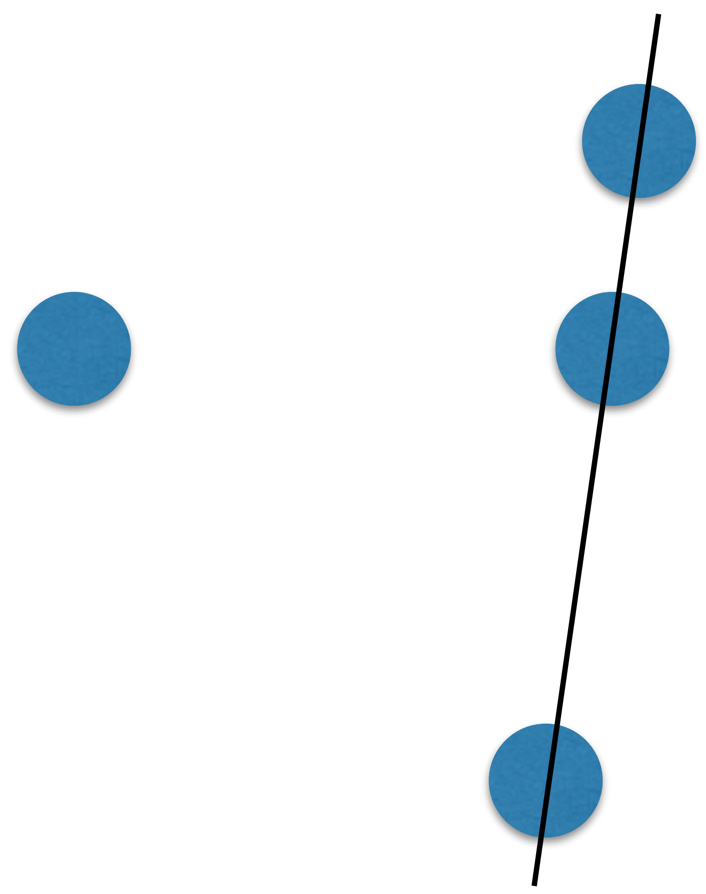
Caution: Avoid Co-Linear Points

- Solution fails if any 3 of the 4 points are co-linear
- Don't even get close to co-linear!

Good:



Bad:



Making It More Robust

- What if the point positions are noisy?
  - Discrete pixel sampling
  - Noise
  - Error
- Solution: get more points!
- Leads to an *overconstrained system*
- Solve using *least-squares solvers*
  - Linear
  - Nonlinear (requires starting point — seed with four-point result)

Bad Matches

- Some of the feature-point matches might be wrong
  - Ambiguous features
  - Side-effects of greedy matching
    (Fully optimal pairing is NP-hard)
- From a least-squares perspective these are *outliers*, which throw off the solution
- Goal is to simultaneously
  - Separate good matches from bad matches
  - Solve for best solution

RANSAC

- RANdom SAmpling with Consensus
- Idea: instead of starting with the full data and trying to prune outliers to a smaller subset of good data, start small and work to larger.
- Key parts:
    - With enough random sampling, you can guess the solution
    - Test solutions by seeing how many points agree with it

RANSAC

- Suppose that you have *n* data points that you want to fit
- RANSAC algorithm:
  - Randomly grab $m < n$ of the points, the fewest number that uniquely allows you to fit the model
  - See how many other points are "consistent" with that solution, i.e. within some tolerance — this is the *consensus set*
  - Repeat this some predefined number of times
  - Go with the solution with the largest consensus set and do a least-squares fit to only the consensus set

RANSAC: Line Fitting

- Example: using RANSAC to fit a line to points
  - Randomly grab two points and calculate line through them
  - Calculate the consensus set
  - Repeat some fixed number of times
    (perhaps stopping if you get a consensus set that is large enough)
  - Go with the solution with the largest consensus set

Why Does RANSAC Work?

- Suppose
  - n data points
  - m minimum points needed to define solution
  - p portion of the n points are inliers
  - k iterations of RANSAC
- One iteration: the probability of picking m good points is

$$p^m$$

- k iterations: the probability of at least once picking a set of good points is

$$1 - (1 - p^m)^k$$

Why Does RANSAC Work?

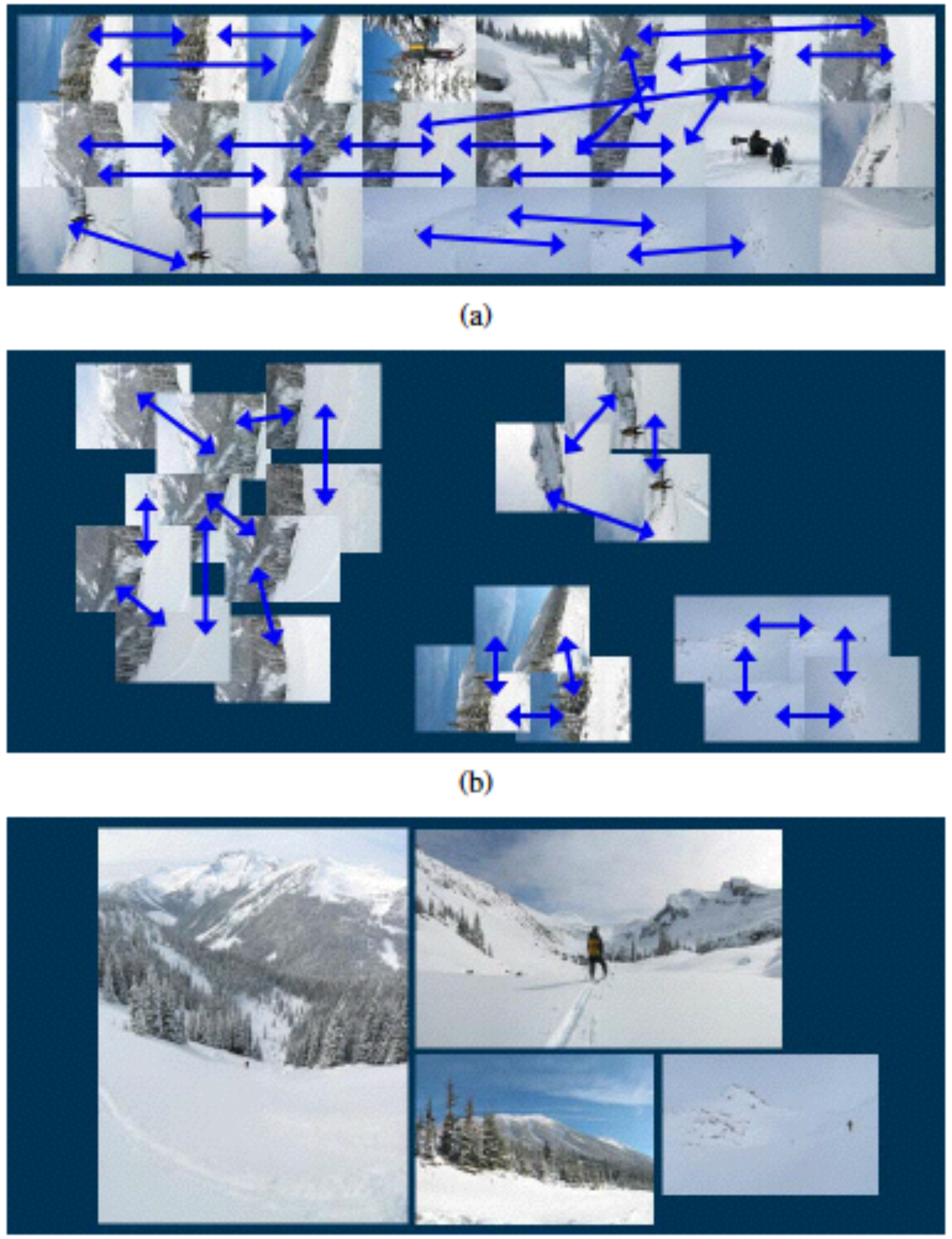
- Going back to our line example
  - n = 100 data points
  - m = 2 minimum points needed to define solution
  - p = 50% portion of the n points are inliers
  - k = 20 iterations of RANSAC
- What is the probability of finding the line and a good consensus set?

$$
\begin{aligned}
1-(1-p^{m})^{k} &= \\
1-\left(1-0.50^{2}\right)^{20} &= \\
99.68\%
\end{aligned}
$$

- Can use to tune number of iterations to get a high probability of success

Stitching Sets of Images

- What if you have more than just two?
- If unorganized, may have to first figure out what images overlap at all
    - Compare color histograms
    - Common subsets of SIFT features
- Organize into a minimum spanning tree(s) that tells you which images to align in a pairwise fashion



(a)

(b)

(c)

Accumulated Error

- Problem:
  - Align image 1 to image 2 — a little bit of error, but not bad
  - Align image 2 to image 3 — a little bit of error, but not bad

    …

  - Align image n-1 to image n — a little bit of error, but not bad
  - How far off is image 1 from image n?

Bundle Adjustment

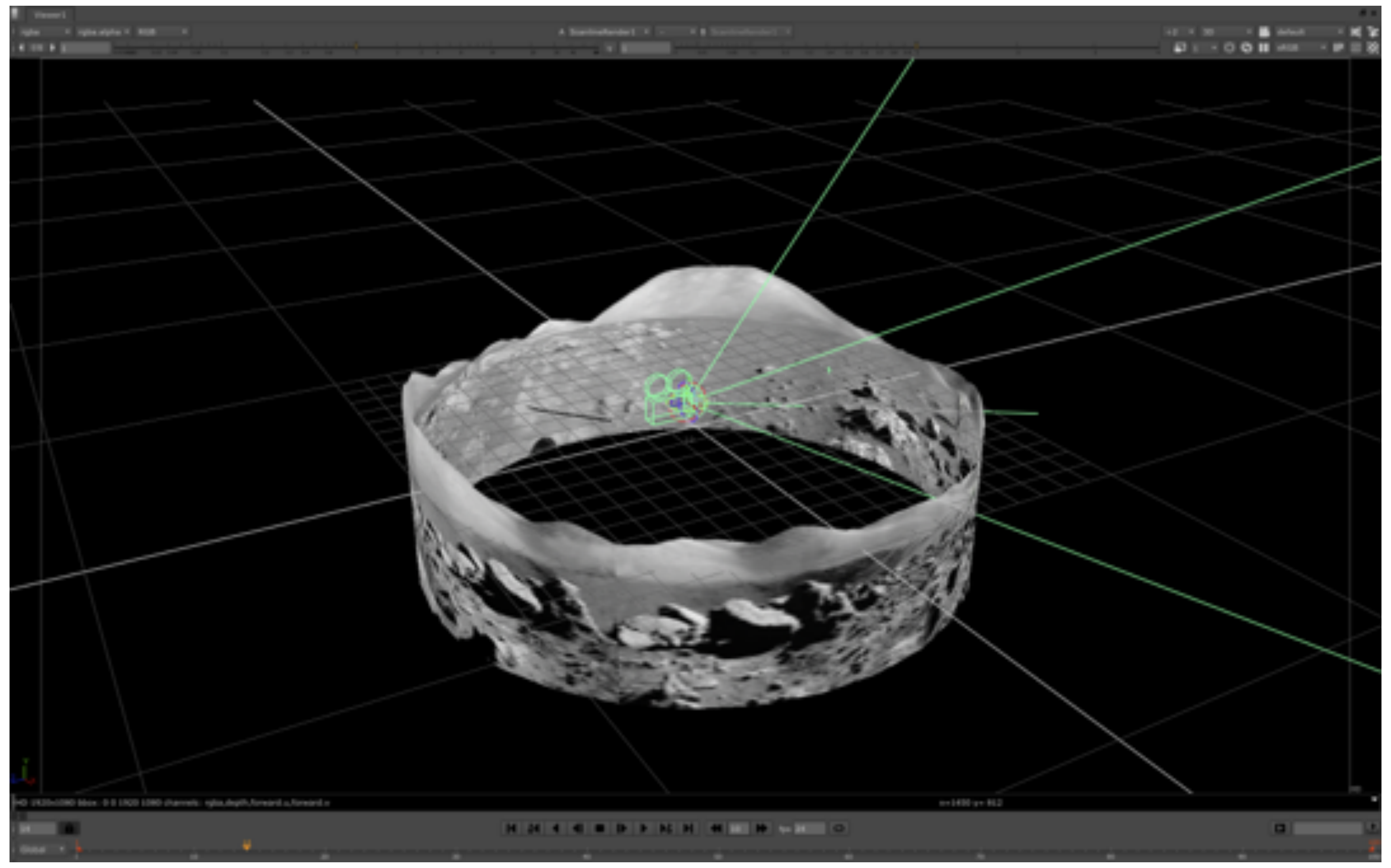
- Solution to accumulated error and drift: one big optimization!
    - All of the images
    - All of the feature point pairs (just keep inliers)
    - Global optimization
- This is called a *bundle adjustment*
- Can be hierarchical

Image Stitching

- Preprocess set of images to figure out which to pairwise align
- Find feature points in *each* image (Harris, SIFT, SURF, etc.)
- Match feature points *between* images of each pair
  - Usually just a greedy method since optimal pairing is NP-hard
  - Reject points with no good matches at all
- Use RANSAC and the four-point algorithm to simultaneously
  - Solve for the transformation
  - Weed out outliers (incorrectly matched feature points)
- Throw the entire resulting consensus set into a least-squares solver (optional)
- Once you’ve done all this pairwise, bundle adjust (optional but very useful)
- Iteratively refine solution using direct (area-based) alignment (optional)
- Composite aligned images

Compositing Surface

- Figure out what kind of surface to warp the images on to
  - Planar
  - Cylindrical
  - Spherical
- Project / warp images onto the compositing surface
- Blend



Blending

- Since the images overlap, you have to blend them together smoothly
- Lots of options:
  - Averaging
  - Median (if more than two)
  - Distance-weighted averaging
  - Voronoi
    (cut by closest and join at cuts)
  - Seam selection
    (least noticeable cuts)
  - Gradient-domain blending

Figure 9.14 Final composites computed by a variety of algorithms (Szeliski 2006a): (a) average, (b) median, (c) feathered average, (d) *p-norm* $p = 10$, (e) Voronoi, (f) weighted ROD vertex cover with feathering, (g) graph cut seams with Poisson blending and (h) with pyramid blending.