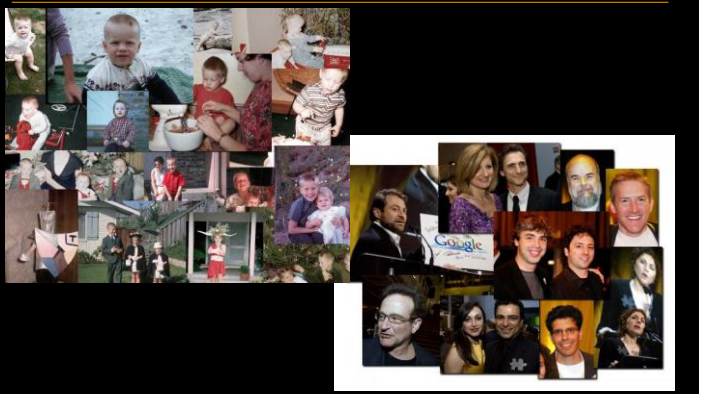
Combining Images

Goal: Create (large) images by putting many photographs together

Slide sources from A. Efros, S. Seitz, V. Vaish, M. Brown, D. Lowe, S. Lazebnik, P. Perez, R. Szeliski

What: Photo Collages



What: Photomosaics

Invented in 1993 by Joseph Francis, and patented by Robert Silvers in 2000

What: Joiners

"Pearblossum Highway" (1986)

David Hockney

Kelsey Bloomquist, 2010

What: Text-to-Picture Communication

First the farmer gives hay to the goat. Then the farmer gets milk from the cow.

What: Slide Shows

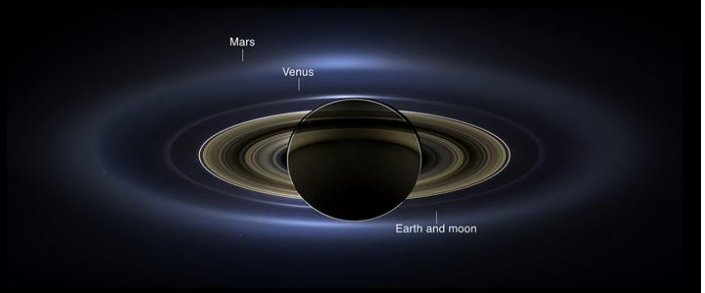
What: Panoramas

Goal**: Given a static scene and a set of images (or video) of it, combine the images into a single "**panoramic image**" or "**panoramic mosaic**" that is **optically correct

Why

- Ca
  ma

49



With the Cassini satellite's wide-angle camera aimed at Saturn, Cassini was able to capture 323 images in just over four hours in 2013. This final mosaic uses 141 of those images taken using red, green and blue spectral filters.

Why Panoramas ?

- Virtual reality walkthroughs

Demo: Quicktime VR [Chen & Williams 95]

Why Panoramas ?

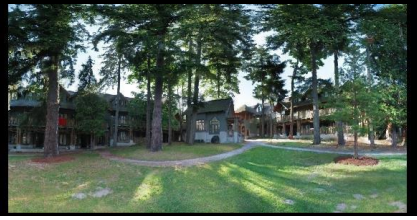
- Getting the whole picture
  - Consumer camera: 50˚ x 35˚

[Brown 2003]

Why Panoramas ?

- Getting the whole picture
  - Consumer camera: 50° x 35°
  - Human Vision: 176° x 135°

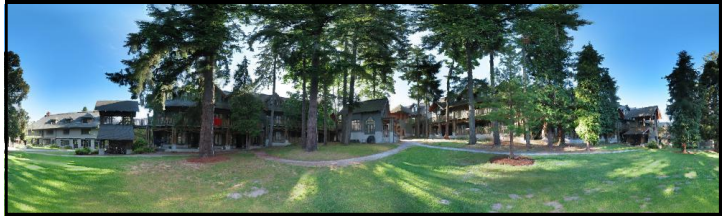
[Brown 2003]



Why Panoramas ?

- Getting the whole picture
  - Consumer camera: 50° x 35°
  - Human Vision: 176° x 135°
  - Panoramic mosaics: up to 360° x 180°

[Brown 2003]



Madison Panoramas

Hans Werner, 2011

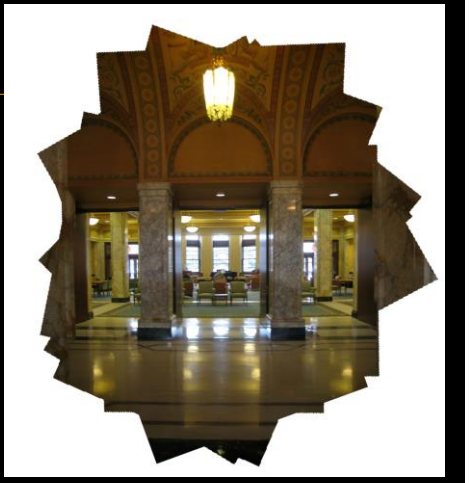
Madison Panoramas

Jean Forde, 2011

Madison Panoramas

Ali Bramson, 2011

Madison Panoramas



Rachel Wroblewski, 2010

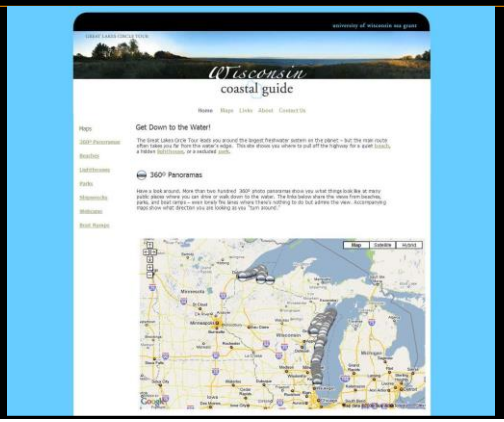
Madison Panoramas

George Wanant, 2010

Madison Panoramas

Tyler Ambroziak, 2010

Wisconsin Coastal Guide Panoramas



Panoramas from Video

One frame from video

Mosaic constructed

Video stabilization, compression and summarization

The First Panoramas …

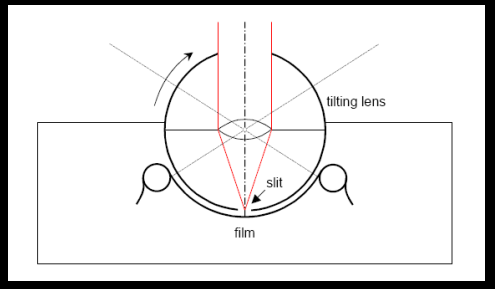
Paris, c. 1845-50, photographer unknown

San Francisco from Rincon Hill, 1851, by Martin Behrmanx

… and Panoramic Cameras

Al-Vista, 1899 ($20)

How they work



Swing lens (1843 – 1980s)

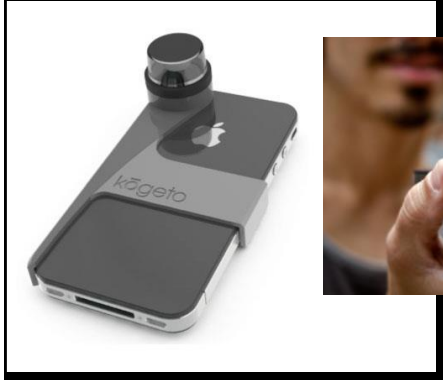
Panorama Capture Hardware

0-360

Point Grey Ladybug

Panoscan MK-3

Kogeto Dot 360 Camera for iPhone



Panorama Stitching Algorithm

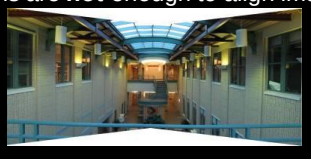
- **Capture Images:** Capture a set of images of a static scene
- **Alignment**: Compute an image-to-image transformation that will map pixel coordinates in one image into corresponding pixel coordinates in a second image
- **Warp**: Warp each image using transform onto output compositing surface (e.g., plane, cylinder, sphere, cube)
- **Interpolate**: Resample warped image
- **Composite**: Blend images together so as to hide seams, exposure differences, lens distortion, scene motion, etc.

When can Images be Aligned?


left on top

right on top

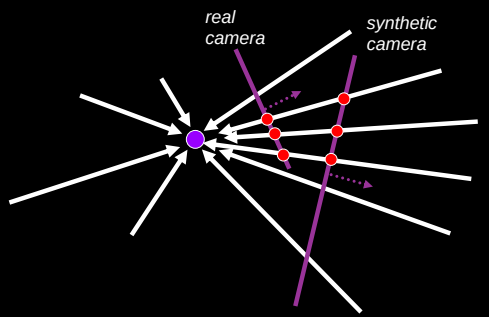
Translations are ***not*** enough to align images in general



When can Two Images be Aligned?

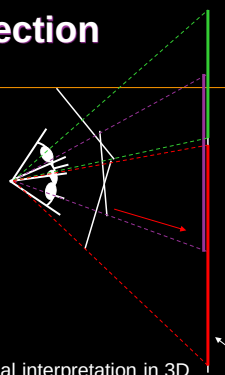
- Problems
  - In general, warping function depends on the **depth** of the corresponding scene point since perspective projection defined by $x' = fx/z, \quad y' = fy/z$
  - Different views mean, in general, that parts that are visible in one image may be occluded in the other
- Special cases where the above problems ***can't*** occur
  - **Panorama: Camera rotates about its optical center, arbitrary 3D scene**
    - No motion parallax as camera rotates, so depth unimportant
    - 2D projective transformation relates any 2 images ($\Rightarrow$ 8 unknowns)
  - **Planar mosaic**: Arbitrary camera views of a **planar scene**
    - 2D projective transformation relates any 2 images

Panoramas: A pencil of rays contains all views



Can generate any synthetic camera view as long as it has a **single center of projection (pinhole)**

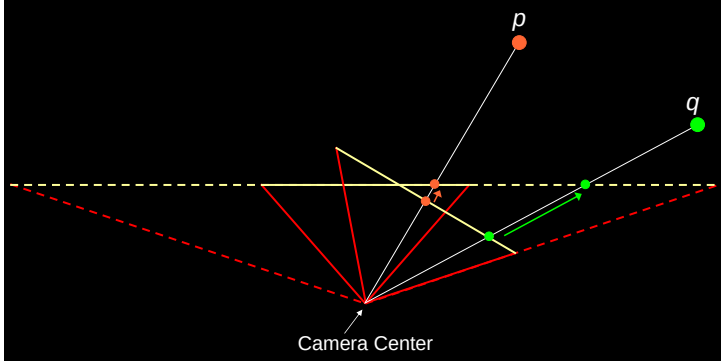
Image Reprojection



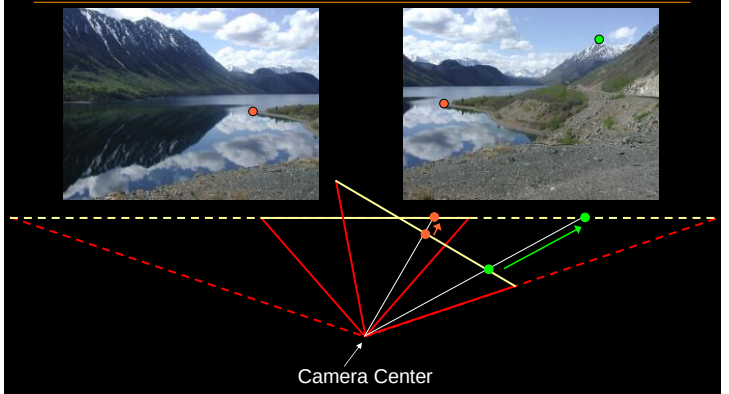
panorama's image plane

- The panorama has a natural interpretation in 3D
  - The images are reprojected onto a common plane
  - The panorama is formed on this plane
  - A panorama is a ***synthetic , wide-angle camera***

Increasing the Field of View



Example



Projection on to Common Image Plane

What is required to project the image on to the desired plane ?

- Scaling ?
- Translation ?
- Rotation ?
- Affine transform ?
- Perspective projection ?

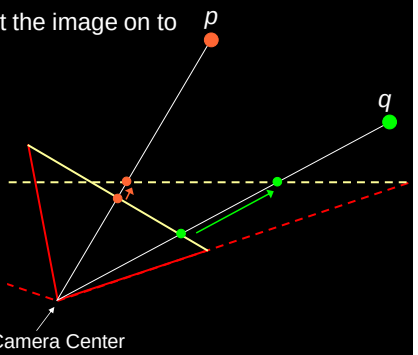
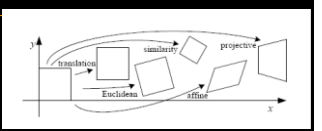
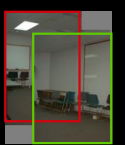


Image Warping

Which transform is the right one for warping PP1 into PP2?

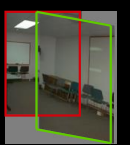


Translation



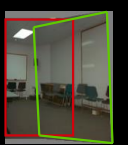
2 unknowns

Affine



6 unknowns

Perspective



8 unknowns

Projection on to Common Image Plane

What is required to project the image on to the desired plane ?

- Scaling
- Translation
- Rotation
- Affine transform
- Perspective projection

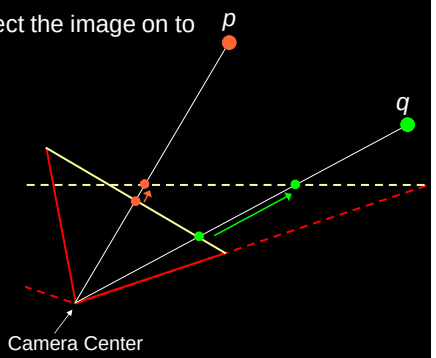
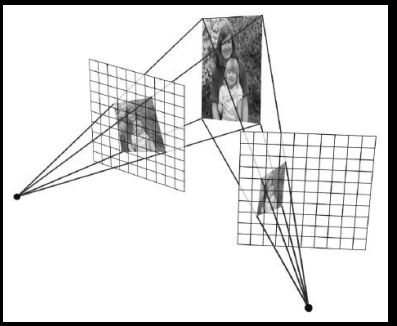


Image Reprojection

- Rather than thinking of this as a 3D reprojection, think of it as a **2D image warp** from one image to another



Aligning Images

What's the relation between corresponding points?

Alignment: Homography

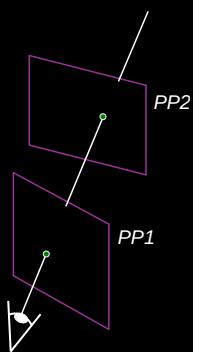
- Perspective projection of a *plane*
  - called **homography**, colineation, or planar projective transformation

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$\mathbf{p}'$ $\quad$ $\mathbf{H}$ $\quad$ $\mathbf{p}$

To apply a homography **H**

- *Compute* $\mathbf{p}' = \mathbf{Hp}$ *(regular matrix multiply)*
- *Convert* $\mathbf{p}'$ *from homogeneous to image coordinates*



Camera Transformations using Homogeneous Coordinates

- Computer vision and computer graphics usually represent points in **Homogeneous coordinates** instead of Cartesian coordinates
- Homogeneous coordinates are useful for representing perspective projection, camera projection, points at infinity, etc.
- Cartesian coordinates (*x*, *y*) represented as Homogeneous coordinates (*wx*, *wy*, *w*) for any scale factor $w \neq 0$
- Given 3D homogeneous coordinates (*x*, *y*, *w*), the 2D Cartesian coordinates are (*x/w*, *y/w*)

Homogeneous Coordinates

*Converting **to** homogeneous coordinates:*

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous scene coordinates

*Converting **from** homogeneous coordinates:*

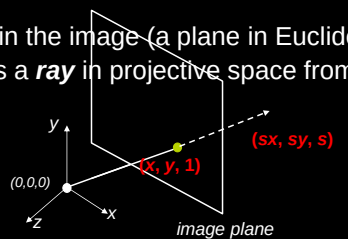
$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

Slide by Steve Seitz

The Projective Plane

- Geometric intuition
  - A ***point*** in the image (a plane in Euclidean space) is a ***ray*** in projective space from origin



- *Each **point** (x, y) in the image is represented by a **ray** (sx, sy, s)*
  - *all points on the ray are equivalent*: **(x, y, 1) ≅ (sx, sy, s)**
  - (*x*, *y*, 0) is the point "at infinity"

2D Mappings

- 2D **translation** - 2 DOFs

$$\begin{cases} x = u + a \\ y = v + b \end{cases}$$

Using Cartesian coords

$$[x, y, 1]^T = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Using Homogeneous coords

- 2D **rotation** (counterclockwise about the origin) - 1 DOF

$$\begin{cases} x = u\cos\theta + v\sin\theta \\ y = -u\sin\theta + v\cos\theta \end{cases}$$

$$[x, y, 1]^T = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- 2D **rigid** (Euclidean) transformation: translation and rotation – 3 DOFs

2D Mappings (cont.)

- 2D **scale** - 2 DOFs

$$\begin{cases} x = \alpha u \\ y = \beta v \end{cases} \qquad [x, y, 1]^T = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Composite translation, rotation, and scale (called a **similarity** transformation) - 5 DOFs

$$[x, y, 1]^T = \begin{bmatrix} \alpha\cos\theta & \beta\cos\theta & \alpha(a\cos\theta - b\sin\theta) \\ -\alpha\sin\theta & \beta\cos\theta & \beta(a\sin\theta + b\cos\theta) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

2D Mappings (cont.)

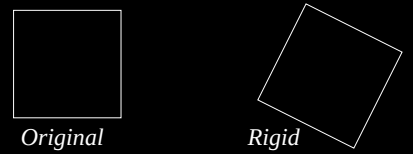
- **Affine** (linear) - 6 DOFs

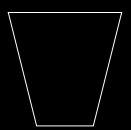
$$\begin{cases} x = a_{11}u + a_{12}v + a_{13} \\ y = a_{21}u + a_{22}v + a_{23} \end{cases} \qquad [x, y, 1]^T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- **Projective** (allows skewing) - 8 DOFs (a_{33} is a scale factor)

$$\begin{cases} x = \dfrac{a_{11}u + a_{12}v + a_{13}}{a_{31}u + a_{32}v + a_{33}} \\ y = \dfrac{a_{21}u + a_{22}v + a_{23}}{a_{31}u + a_{32}v + a_{33}} \end{cases} \qquad [x, y, 1]^T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Examples of 2D Transformations

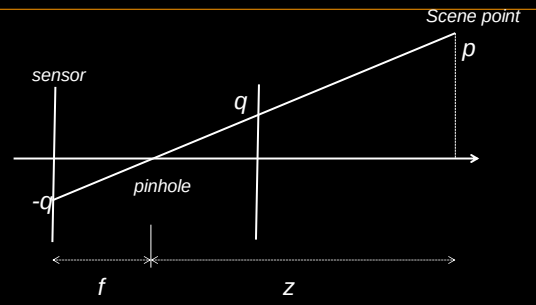




Properties of Transformations

- **Projective**
  - Preserves collinearity, concurrency, order of contact
- **Affine** (linear transformations)
  - Preserves above plus parallelism, ratio of areas, …
- **Similarity** (rotation, translation, scale)
  - Preserves above plus ratio of lengths, angle
- **Rigid** (rotation and translation)
  - Preserves above plus length, area

Perspective Projection: Pinhole Optics



$q = (f/z)p$

Perspective Projection

- Projection is a matrix multiply using homogeneous coords!

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z/f \end{bmatrix} \Rightarrow \left(\frac{fX}{Z}, \frac{fY}{Z} \right)$$

- This 3 x 4 matrix is called the **camera perspective projection matrix**

Perspective Projection

- Or, equivalently, after multiplying the projection matrix by *f*, we get the same transformation:

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} \Rightarrow \left(\frac{fX}{Z}, \frac{fY}{Z} \right)$$

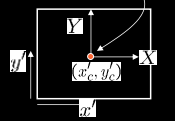
Modeling a Perspective Camera

A real camera is modeled by several parameters

- Translation ***T*** of the optical center from the origin of world coords
- Rotation ***R*** of the image plane
- focal length ***f***, principle point (***x'c, y'c***) , pixel size (***sx, sy***)
- blue parameters are called "**extrinsics**," red are "**intrinsics**"

Projection equation

$$\mathbf{x} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \Pi \mathbf{X}$$



- The projection matrix models the cumulative effect of ***all*** parameters
- Useful to decompose into a series of operations

identity matrix

$$\Pi = \begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}$$

intrinsics $\mathbf{K}$ ***projection*** ***rotation*** ***translation***

Note: Can also add other parameters to model lens distortion

Projective Camera

- More general than Perspective Camera matrix, Π
- Most general mapping from P^3 to P^2
- Transformation matrix has only 11 DOFs since only the *ratios* of elements are important

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix}$$

Affine Camera

- Most general ***linear*** transformation
- 8 DOFs
- Reasonable assumption when scene objects are far away from camera (relative to focal length)

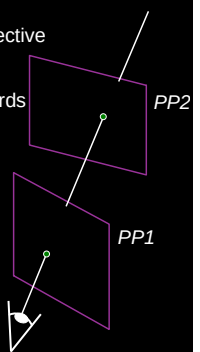
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix}$$

Homography: Viewing a Plane

- Perspective projection of a ***plane***
  - Called **homography**, colineation, or planar projective transformation, **H**
  - Modeled as a 2D warp using homogeneous coords

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

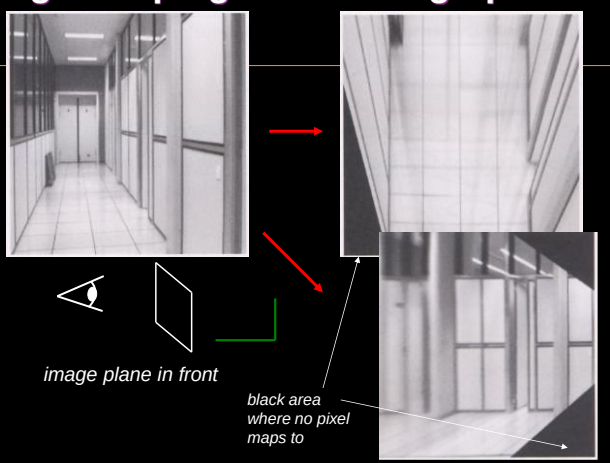
p′ **H** ***p***



To apply a homography **H**

- Compute ***p′*** = **H*p*** (regular matrix multiply)
- Convert ***p′*** from homogeneous to image coordinates by dividing by 3rd coord

Image Warping with Homographies



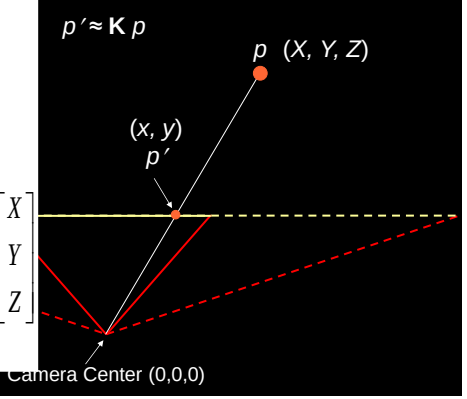
Perspective Warps (Homographies)

$$x = \frac{-fs_x X}{Z} + o_x$$

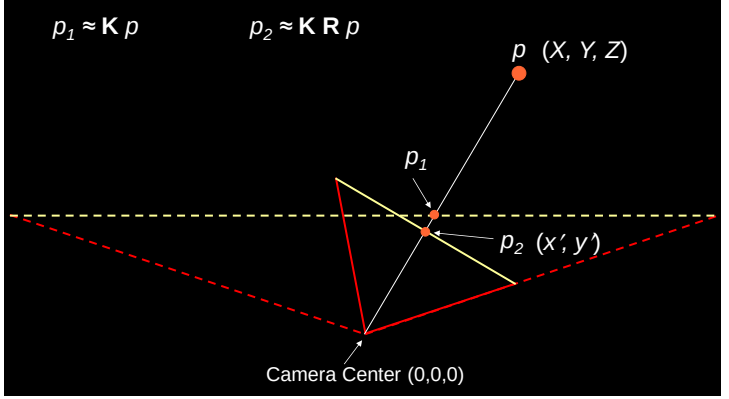
$$y = \frac{-fs_y Y}{Z} + o_y$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \approx \begin{bmatrix} -fs_x & 0 & o_x \\ 0 & -fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

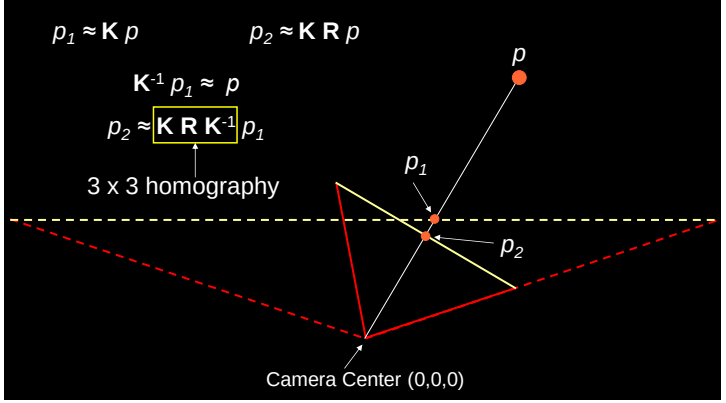
K



Perspective Warps (Homographies)



Perspective Warps (Homographies)



Combining Images into Panoramas

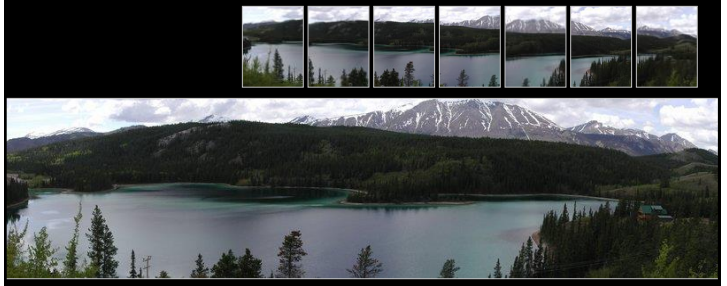
- Theorem: Any 2 images of an arbitrary scene taken from 2 cameras with *same* camera center are related by $p_2 \approx \mathbf{K R K}^{-1} p_1$ where p_1 and p_2 are homogeneous coords of 2 corresponding points, **K** is 3 x 3 camera calibration matrix, and **R** is 3 x 3 rotation matrix
- $\mathbf{K R K}^{-1}$ is 3 x 3 matrix called the "homography induced by the plane at infinity"

Finding the Homographies

How can we compute the homographies required for alignment?

- From calibration parameters
  - Works, but these aren't always known
- By matching features across images

How: Making Panoramas



Goal: *Combine pixels from multiple images to compute a bigger image*

How: Panorama Making Software

- Autostitch
- Windows Live Photo Gallery
- Adobe Photoshop's Photomerge
- Hugin
- AutoPano
- Autopano-SIFT

Panorama Stitching Algorithm

- **Capture Images:** Take a sequence of images , I_1, …, I_n, from the same position by rotating the camera around its optical center
- **Alignment**: Compute an image-to-image transformation that will map pixel coordinates in one image into corresponding pixel coordinates in second image
- **Warp**: Warp each image using transform onto output compositing surface (e.g., plane, cylinder, sphere, cube)
- **Interpolate**: Resample warped image
- **Composite**: Blend images together so as to hide seams, exposure differences, lens distortion, scene motion, etc.

Alignment

- Direct methods
  - Search over space of possible image warps and compare images by pixel intensity/color matching to find warp with minimum matching error
- Feature-based methods
  - Extract distinctive (point) features from each image and match these features to establish global correspondence, and then estimate geometric transformation

Alignment

- Direct method: Uses image-based (intensity) correlation to determine best matching transformation
  - No correspondences needed
  - Statistically optimal (gives maximum likelihood estimate)
  - Useful for local image registration
- Feature-based method: Finds point correspondences, and then solves for unknowns in "motion model"
  - Requires reliable detection of a sufficient number of corresponding features, at sub-pixel location accuracy

Example of Direct Method: 2D Rigid Warp Mosaics

- **Assume**: Planar scene, camera motion restricted to plane parallel to scene, optical axis perpendicular to scene, brightness constancy assumption, local displacement
- **3 unknowns**: 2D translation (*a*, *b*) and 2D rotation (θ)
- Relation between two images, *I* and *I′*, given by:

$$I'(x', y') = I(x\cos\theta - y\sin\theta + a, x\sin\theta + y\cos\theta + b)$$

Expanding $\sin\theta$ and $\cos\theta$ to first two terms in Taylor series:

$$I'(x', y') \approx I(x + a - y\theta - x\theta^2/2, y + b + x\theta - y\theta^2/2)$$

Expanding I to first term of its Taylor series expansion:

$$I'(x', y') \approx I(x, y) + (a - y\theta - x\theta^2/2)\partial I/\partial x + (b + x\theta - y\theta^2/2)\partial I/\partial y$$

Example: 2D Rigid Warp Mosaics

- Solve for *a, b,* θ that minimizes SSD error:

$$E = \sum\sum (I' - I)^2$$
$$= \sum_x \sum_y (I(x, y) + (a - y\theta + x\theta^2/2)\partial I/\partial x + (b + x\theta - y\theta^2/2)\partial I/\partial y - I(x, y))^2$$

- Assuming small displacement, use gradient descent to minimize *E*, ∇E = ($\partial E/\partial a$, $\partial E/\partial b$, $\partial E/\partial\theta$)
- Iteratively update total motion estimate, (*a, b,* θ), while warping *I′* towards *I* until *E* < threshold

Global Image Registration

- When images are not sufficiently close, must do global registration
- **Coarse-to-fine matching using Gaussian Pyramid**
  1. Compute Gaussian pyramids
  2. *level* = N // Set initial processing level to coarsest level
  3. Solve for motion parameters at level *level*
  4. **If** *level* = 0 **then halt**
  5. Interpolate motion parameters at level *level* - 1
  6. *level* = *level* - 1
  7. **Goto** step 3

Direct Method for Computing Panoramic Mosaics

- Motion model is 2D projective transformation, so 8 parameters (DOFs)
- Assuming small displacement, minimize SSD error
- Apply (nonlinear) minimization algorithm to solve

Finding the Homography


- By matching features across images
  - What features should we match?
  - How many features?

Finding the Homography

What features do we match across images ?

- Pixel values ?
- Edges ?
- Corners ?
- Lines ?
- SIFT features ?
- Other ?

Finding the Homography

What features do we match across images ?

- Pixel values
- Edges
- Corners
- Lines
- SIFT features
- Other

Homography by Feature Matching

$p_2 \approx \mathbf{K}\,\mathbf{R}\,\mathbf{K}^{-1}\,p_1$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \approx \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography by Feature Matching

$p_2 \approx \mathbf{K}\,\mathbf{R}\,\mathbf{K}^{-1}\,p_1$

$$x' = \frac{a_1 x + a_2 y + a_3}{c_1 x + c_2 y + c_3}$$

$$y' = \frac{b_1 x + b_2 y + b_3}{c_1 x + c_2 y + c_3}$$

Two linear equations per matching feature

Solving for Homography

$p' = \mathbf{H}p$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Set scale factor i = 1. So, there are 8 unknowns
- Set up a system of linear equations:

$$\mathbf{Ah} = \mathbf{b}$$

where vector of unknowns $\mathbf{h} = [a,b,c,d,e,f,g,h]^T$

- Need at least 8 equations, but the more the better
- Solve for **h**. If overconstrained, solve using least-squares:

$$\min \|\mathbf{Ah} - \mathbf{b}\|^2$$

- Can be done in Matlab using "\" command (see "help lmdivide")

Solving for Homography

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_i' = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y_i' = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$\begin{aligned} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Solving for Homography

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

A *2n × 9* **h** *9* **0** *2n*

Linear least squares

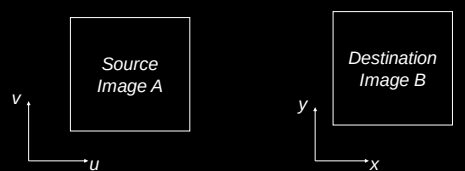
- Since **h** is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Minimize $\|\mathbf{A}\hat{\mathbf{h}}\|^2$

$$\|\mathbf{A}\hat{\mathbf{h}}\|^2 = (\mathbf{A}\hat{\mathbf{h}})^T\mathbf{A}\hat{\mathbf{h}} = \hat{\mathbf{h}}^T\mathbf{A}^T\mathbf{A}\hat{\mathbf{h}}$$

- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points

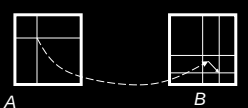
Warping

- Define transformation as either
  - **Forward**: $x = \mathbf{X}(u, v)$, $y = \mathbf{Y}(u, v)$
  - **Backward**: $u = \mathbf{U}(x, y)$, $v = \mathbf{V}(x, y)$



Warping Methods

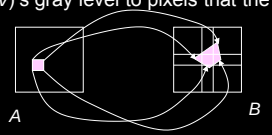
- **Forward, point-based**
  - Apply forward mapping **X**, **Y** at point (*u*,*v*) to obtain real-valued point (*x*,*y*)
  - Assign (*u*,*v*)'s gray level to pixel ***closest to*** (*x*,*y*)



  - Problem: "**measles**," i.e., "**holes**" (pixel in destination image that is not assigned a gray level) and "**folds**" (pixel in destination image is assigned multiple gray levels)
  - Example: Rotation, since preserving length cannot preserve number of pixels

Warping Methods

- **Forward, square-pixel based**
  - Consider pixel at (u,v) as a unit square in source image. Map square to a quadrilateral in destination image
  - Assign (u,v)'s gray level to pixels that the quadrilateral overlaps



  - Integrate source pixels' contributions to each output pixel. Destination pixel's gray level is weighted sum of intersecting source pixels' gray levels, where weight proportional to coverage of destination pixel
  - Avoids holes, but not folds, and requires intersection test

Warping Methods

- **Backward, point-based**
  - For each destination pixel at coordinates (x,y), apply backward mapping, **U**, **V**, to determine real-valued source coordinates (u,v)
  - Interpolate gray level at (u,v) from neighboring pixels, and copy gray level to (x,y)

  - Interpolation may cause artifacts such as aliasing, blockiness, and false contours
  - Avoids holes and folds problems
  - Method of choice

Backward Warping

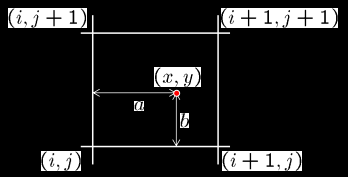
- **for** x = xmin **:** xmax
  for y = ymin **:** ymax
  u = **U**(x, y);
  v = **V**(x, y);
  B(x, y) = A(u, v);
  end
  end
- But (u, v) may not be at a pixel in A
- (u, v) may be out of A's domain
- If **U** and/or **V** are discontinuous, A may not be connected!
- Digital transformations in general don't commute

Pixel Interpolation

- **Nearest-neighbor (0-order) interpolation**
  - $A(u, v)$ = gray level at nearest pixel (i.e., round (u, v) to nearest integers)
  - May introduce artifacts if image contains fine detail
- **Bilinear (1st-order) interpolation**
  - Given the 4 nearest neighbors, $A(0, 0)$, $A(0, 1)$, $A(1, 0)$, $A(1, 1)$, of a desired point $A(u, v)$, compute gray level at $A(u, v)$:
    - Interpolate linearly between $A(0,0)$ and $A(1,0)$ to obtain $A(u,0)$
    - Interpolate linearly between $A(0,1)$ and $A(1,1)$ to obtain $A(u,1)$
    - Interpolate linearly between $A(u,0)$ and $A(u,1)$ to obtain $A(u,v)$
  - Combining all three interpolation steps into one we get:
    - $A(u,v) = (1-u)(1-v)\,A(0,0) + (1-u)v\,A(0,1) + u(1-v)\,A(1,0) + uv\,A(1,1)$
- **Bicubic spline interpolation**

Bilinear Interpolation

- A simple method for resampling images



$$f(x,y) = \begin{array}{ll} (1-a)(1-b) & f[i,j] \\ +a(1-b) & f[i+1,j] \\ +ab & f[i+1,j+1] \\ +(1-a)b & f[i,j+1] \end{array}$$

Example of Backward Warping

- **Goal**: Define a transformation that performs a **scale change,** which expands size of image by 2, i.e., **U**(*x*) = *x*/2
- A = 0 … 0 2 2 2 0 … 0
- 0-order interpolation, i.e., $u = \lfloor x/2 \rfloor$

  B = 0 … 0 2 2 2 2 2 2 0 … 0
- Bilinear interpolation, i.e., *u* = *x*/2 and average 2 nearest pixels if *u* is not at a pixel

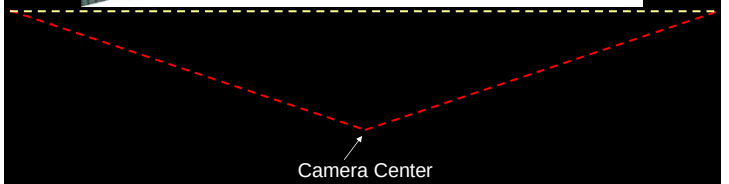
  B = 0 … 0 1 2 2 2 2 2 1 0 … 0

Panoramic Stitching Algorithm

Input: *N* images from camera rotating about center

1. Detect point features and their descriptors in all images
2. For adjacent images:
   1. Match features to get pairs of corresponding points
   2. [Optional: Eliminate bad matches]
   3. Solve for homography
3. Project images on common "image plane"
4. Blend overlapping images to obtain panorama

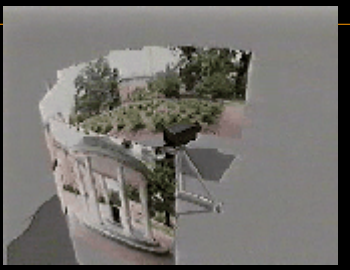
Do we have to Project on to a Plane ?



Panorama Images

- Large field of view ⇒ should ***not*** map all images onto a plane
- Instead, map onto cylinder, sphere, or cube
- With a cylinder, first warp all images from rectilinear to cylindrical coordinates, then combine them
- "Undistort" (perspective correct) image from this representation prior to viewing

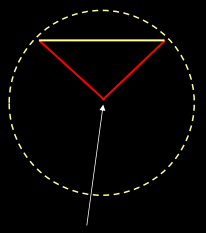
Cylindrical Panoramas



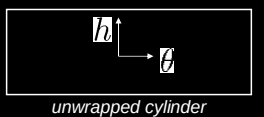
- Steps
  - Reproject each image onto a cylinder
  - Blend
  - Output the resulting panorama

Cylindrical Projection

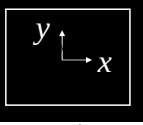
360˚ Panorama
[Szeliski & Shum 97]



Cylindrical Projection

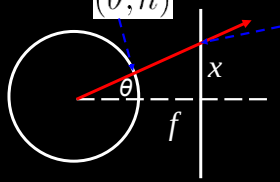


unwrapped cylinder



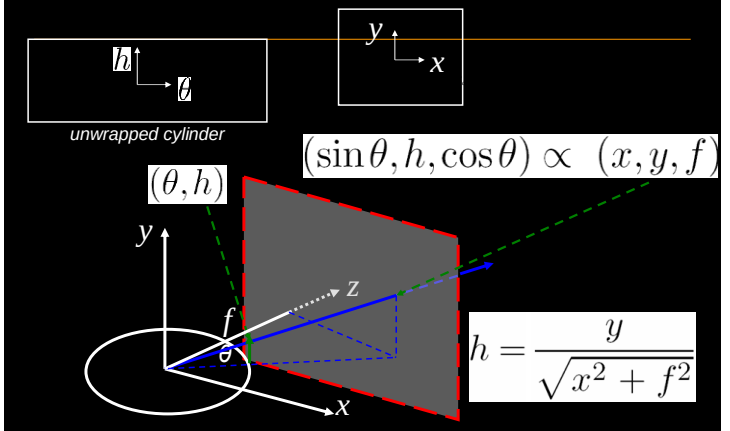
$$(\sin\theta, h, \cos\theta) \propto (x, y, f)$$

(θ, h)



$$\theta = \tan^{-1}\frac{x}{f}$$

Cylindrical Projection

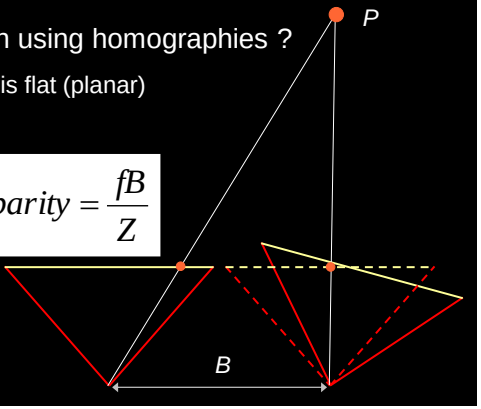


Beyond Panoramas: General Camera Motion

Can we still stitch using homographies ?

- When the scene is flat (planar)
- When *Z* >> B

$$disparity = \frac{fB}{Z}$$



Blending: Getting Rid of Seams

Some Causes of Seams:

- Differences in exposure
- Vignetting
- Small misalignments

[Brown 2003]

Method 1: Averaging

- For each output pixel, compute average of warped pixels

$$C(x) = \sum_k w_k(x)\tilde{I}_k(x) / \sum_k w_k(x)$$

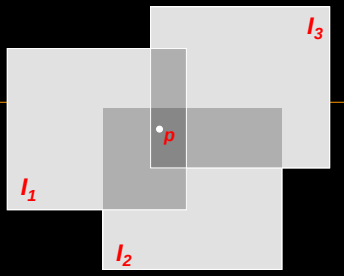
where $\tilde{I}_k(x)$ are the warped images and $w_k(x)$ is 1 at valid pixels and 0 elsewhere

- Weakness: Doesn't work well with exposure differences, mis-registration, etc.

Method 2: Weighted Averaging

- Aka **Feathering** or **Alpha Blending**
- Weight pixels near center of each warped image more heavily than pixels near image border
- If image has holes, also down-weight values near border of hole
- Implement by computing a **distance map** = distance to nearest invalid pixel
- Weakness: blurs details such as edges

Feathering



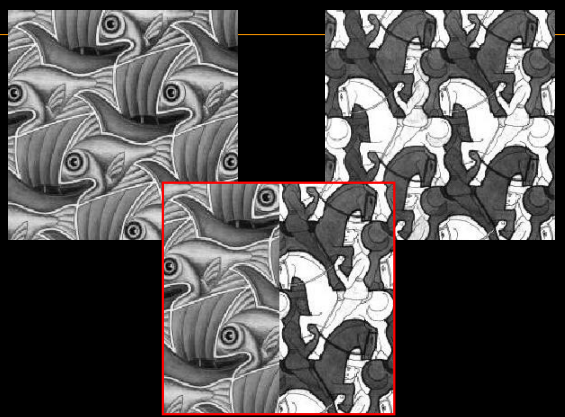
Encoding blend weights: $I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$

color at p = $\frac{(\alpha_1 R_1,\ \alpha_1 G_1,\ \alpha_1 B_1) + (\alpha_2 R_2,\ \alpha_2 G_2,\ \alpha_2 B_2) + (\alpha_3 R_3,\ \alpha_3 G_3,\ \alpha_3 B_3)}{\alpha_1 + \alpha_2 + \alpha_3}$

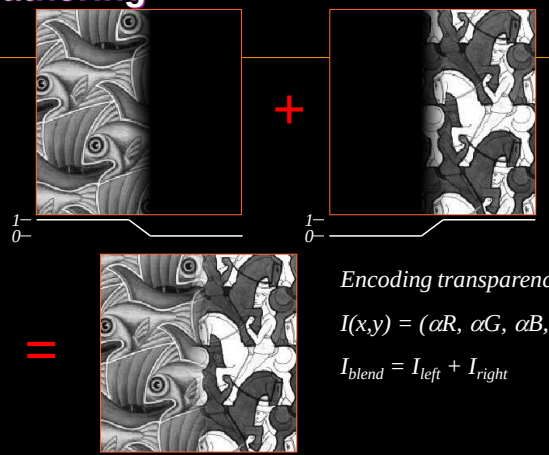
Implement in two steps:

1. *Accumulate: add up the (α premultiplied) RGBα values at each pixel*
2. *Normalize: divide each pixel's accumulated RGB by its α value*

Image Blending



Feathering

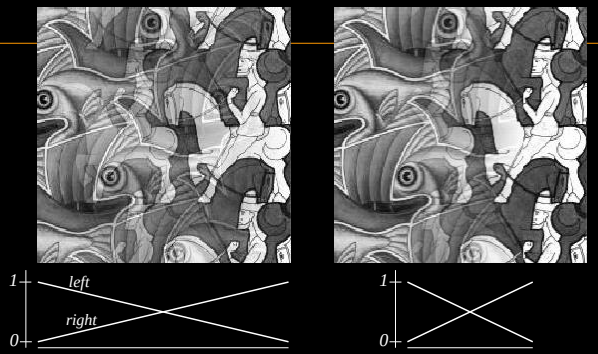


Encoding transparency

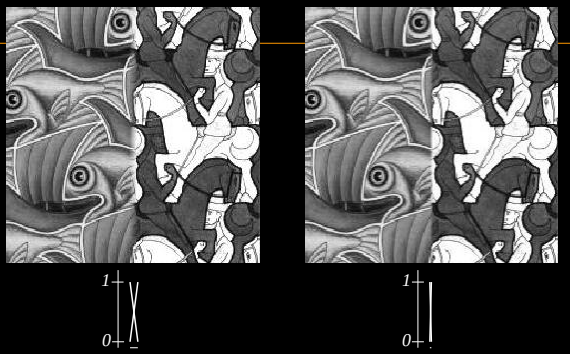
$I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$

$I_{blend} = I_{left} + I_{right}$

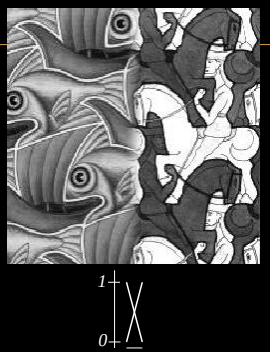
Effect of Window Size



Effect of Window Size



Good Window Size



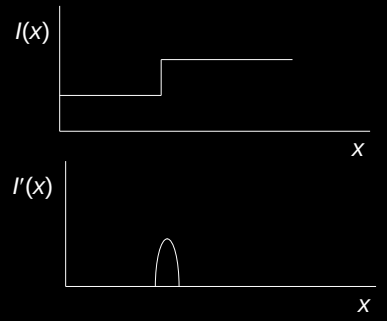
"Optimal" Window: smooth but not ghosted

Method 3: Laplacian Pyramid Blending

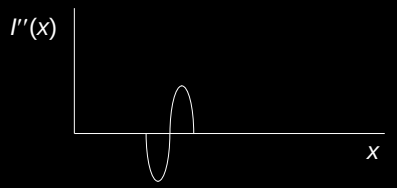
- [Burt and Adelson 1983]
- Content-based blending using edge features
- Multi-resolution technique using image pyramid
- Hides seams but preserves sharp detail

1D Edge Detection

- An ideal edge is a step function



1D Edge Detection



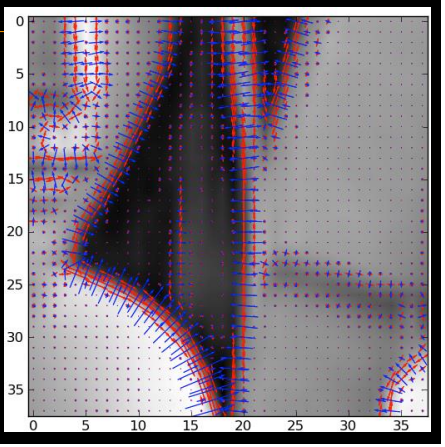
- The first derivative of I(x) has a **peak** (local max or local min) at the edge
- The second derivative of I(x) has a **zero crossing** at the edge

Edge Detection in 2D

- Let I(x,y) be the image intensity function. It has derivatives in all directions
  - $\partial I(x, y)/\partial x = \lim I(x+\Delta x, y) - I(x, y) / \Delta x \approx I(u+1, v) - I(u,v)$
  - **Gradient of I(x, y) is a vector $\nabla I(x, y) = [\partial I/\partial x, \partial I/\partial y]^T$** specifying the direction of greatest rate of change in intensity (i.e., perpendicular to the edge's direction)
  - From gradient can determine the **direction** in which the first derivative is highest, and the **magnitude** of the first derivative in that direction
  - Magnitude = $[(\partial I/\partial x)^2 + (\partial I/\partial y)^2]^{1/2}$
  - Direction = $\tan^{-1} (\partial I/\partial y)/(\partial I/\partial x)$

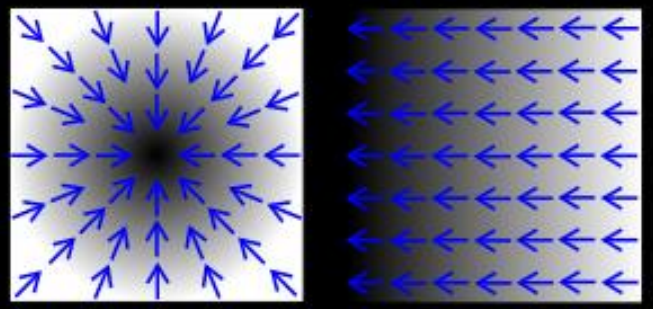
Image Gradient

- Vector field of image gradients are shown in blue; the red vectors are perpendicular to the gradient and along edge direction



Source: F. Blanco-Silva

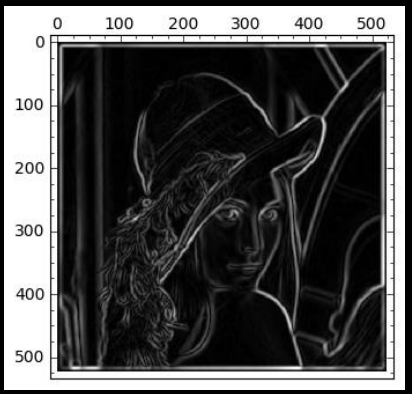
Image Gradient



Blue arrows show the *direction* of the gradient

Source: Wikipedia

Magnitude of the Gradient



Edge Detection in 2D

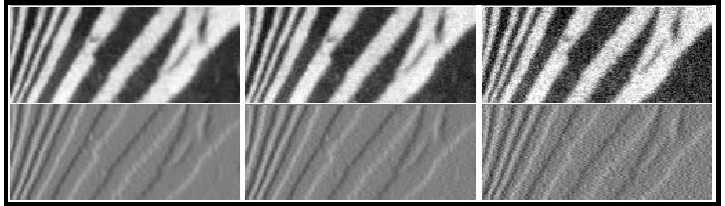
- With a digital image, the partial derivatives are replaced by finite differences:
  - $\Delta_x I = I(u+1, v) - I(u, v)$
  - $\Delta_y I = I(u, v) - I(u, v+1)$

- An alternative (Sobel)
  - $\Delta_{sobel\_X} I = I(u+1, v+1) + 2I(u+1, v) + I(u+1, v-1) - I(u-1, v+1) - 2I(u-1, v) - I(u-1, v-1)$
  - $\Delta_{sobel\_Y} I = I(u-1, v-1) + 2I(u, v-1) + I(u+1, v-1) - I(u-1, v+1) - 2I(u, v+1) - I(u+1, v+1)$
- Roberts's "Cross"
  - $\Delta_+ I = I(u, v) - I(u+1, v-1)$ → $\begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$
  - $\Delta_- I = I(u, v-1) - I(u+1, v)$ → $\begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$

Finite Differences and Noise

- Finite difference filters respond strongly to noise
  - obvious reason: image noise results in pixels that look very different from their neighbors
- Generally, the more noise, the stronger the response
- What is to be done?
  - intuitively, most pixels in images look quite a lot like their neighbors
  - this is true even at an edge; along the edge they're similar, across the edge they're not
  - suggests that ***smoothing*** the image should help, by forcing pixels different from their neighbors (=noise pixels?) to look more like their neighbors

Finite Differences Responding to Noise



Increasing noise →

(this is zero mean additive Gaussian noise)

Summary of Basic Edge Detection Steps

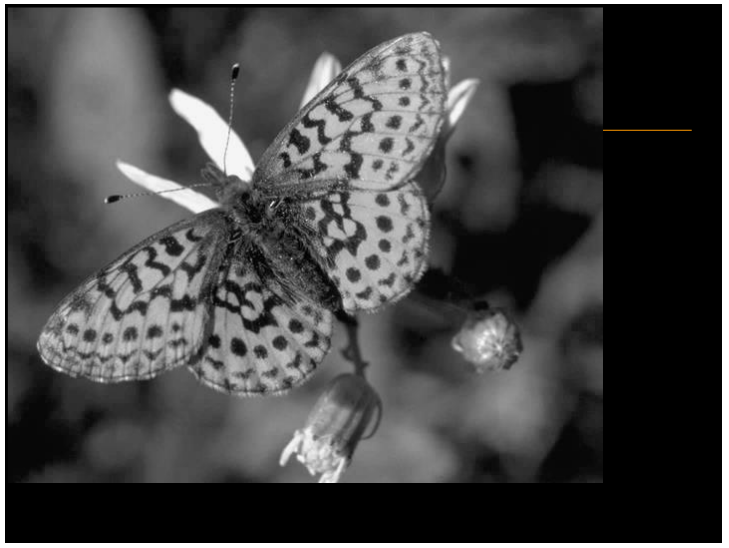
- Smooth the image to reduce the effects of local intensity variations
  - Choice of smoothing operator is practically important
- Differentiate the smoothed image using a digital gradient operator that assigns a magnitude and direction of the gradient at each pixel
  - Mathematically, we can apply the digital gradient operator to the digital smoothing filter, and then just convolve the differentiated smoothing filter to the image
  - Requires using a slightly larger smoothing array to avoid border effects

Summary of Basic Edge Detection Steps

- Threshold the gradient magnitude to eliminate low contrast edges
- Apply a non-maximum suppression step to thin the edges to single pixel wide edges
  - Smoothing will produce an image in which the contrast at an edge is spread out in the neighborhood of the edge
  - Thresholding operation will produce thick edges

The Scale Space Problem

- Usually, any single choice of σ does not produce a good edge map
  - a large σ will produce edges from only the largest objects, and they will not accurately delineate the object because the smoothing reduces shape detail
  - a small σ will produce many edges and very jagged boundaries of many objects
- Scale-space approaches
  - detect edges at a range of scales $[\sigma_1, \sigma_2]$
  - combine the resulting edge maps



fine scale high threshold

coarse scale, high threshold

Laplacian Edge Detectors

- Directional second derivative in direction of gradient has a **zero crossing** at gradient maximum
- Can approximate directional second derivative with **Laplacian**:

$$\nabla^2 I = \partial^2 I / \partial x^2 + \partial^2 I / \partial y^2$$

- Laplacian is lowest order linear isotropic operator
- Digital approximation (2nd forward difference) is
  - $\nabla^2 I(u,v) = [I(u+1,v) + I(u-1,v) + I(u,v+1) + I(u,v-1)] - 4I(u,v)$

    $= [I(u+1,v) - I(u,v)] - [I(u,v) - I(u-1,v)] + [I(u,v+1) - I(u,v)] - [I(u,v) - I(u,v-1)]$

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Laplacian Examples

- I = ... 2 2 2 8 8 8 ...
- ∇^2 = 1 -2 1

  $\Rightarrow \nabla^2 I$ = ... 0 0 0 6 -6 0 0 0 ...

- I = ... 2 2 2 5 8 8 8 ...

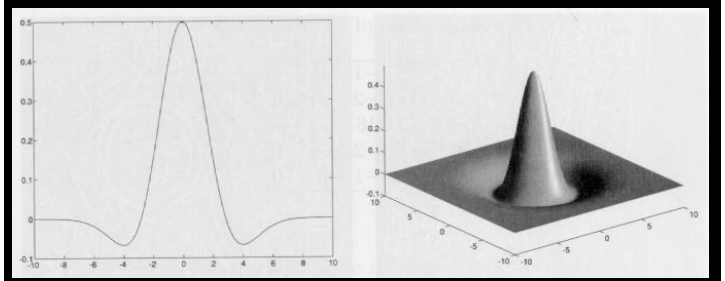
  $\Rightarrow \nabla^2 I$ = ... 0 0 0 3 0 -3 0 0 0 ...

Laplacian Edge Detectors

- Laplacians are also combined with smoothing for edge detectors
  - Take the Laplacian of a Gaussian-smoothed image - called the Laplacian-of-Gaussian (**LoG**), Mexican Hat operator, Difference-of-Gaussians (**DoG**), Marr-Hildreth, $\nabla^2 G$ operator
  - Locate the zero-crossing of the operator
    - these are pixels whose LoG is positive and which have neighbor's whose LoG is negative or zero
  - Usually, measure the gradient or directional first derivatives at these points to eliminate low contrast edges

Laplacian of Gaussian (LoG)

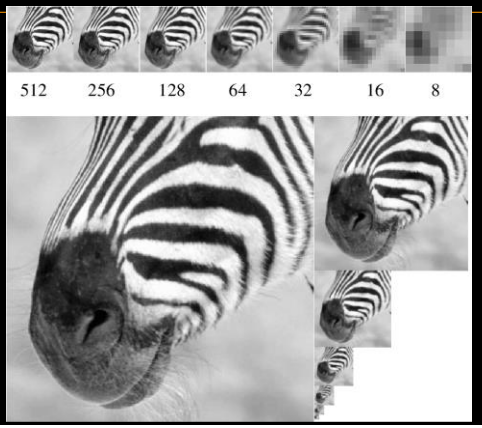
$$\nabla^2 G_\sigma(x, y) = -[1/2\pi\sigma^4] (2 - (x^2 + y^2)/\sigma^2) e^{-(x^2 + y^2)/2\sigma^2}$$



LoG Properties

- Linear, shift invariant $\Rightarrow$ convolution
- Circularly symmetric $\Rightarrow$ isotropic
- Size of LoG operator approximately 6σ x 6σ
- LoG is separable
- LoG $\approx G_{\sigma_1} - G_{\sigma_2}$, where $\sigma_1 = 1.6\sigma_2$
- Analogous to spatial organization of receptive fields of retinal ganglion cells, with a central excitatory region and an inhibitory surround

Gaussian Pyramids



Gaussian Pyramid

- Multiresolution, low-pass filter
- Hierarchical convolution
  - G0 = input image
  - $G'_k(u, v) = \sum\sum w(m, n)\, G_{k-1}(u-m, v-n)$; smooth
  - $G_k(u, v) = G'_k(2u, 2v), \quad 0 < u, v < 2^{N-k}$; sub-sample
- *w* is a small (e.g., 5 x 5) separable generating kernel, e.g., 1/16 [1 4 6 4 1]
- Cascading *w* equivalent to applying large kernel
  - Effective size of kernel at level k = $2M(2^k - 1) + 1$, where *w* has width 2M+1
  - Example: Let M=1. If k=1 then equivalent size = 5; k=2 then equivalent size = 13; k=3 then equivalent size = 27

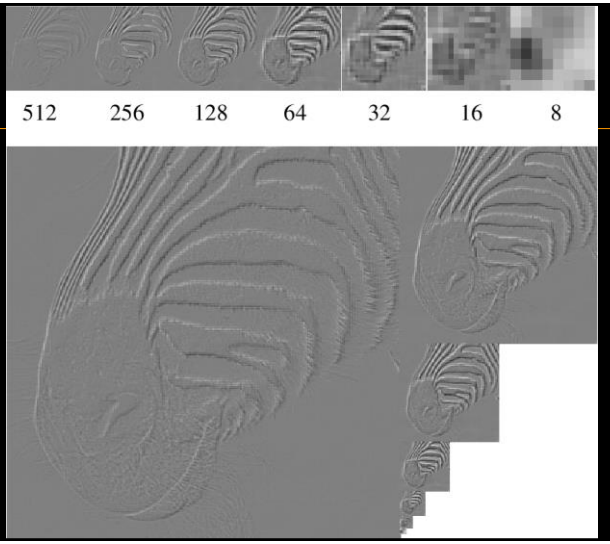
Laplacian Pyramids

- Similar to results of edge detection
- Most pixels are 0
- Can be used for image compression

$$L_1 = g_1 - EXPAND[g_2]$$

$$L_2 = g_2 - EXPAND[g_3]$$

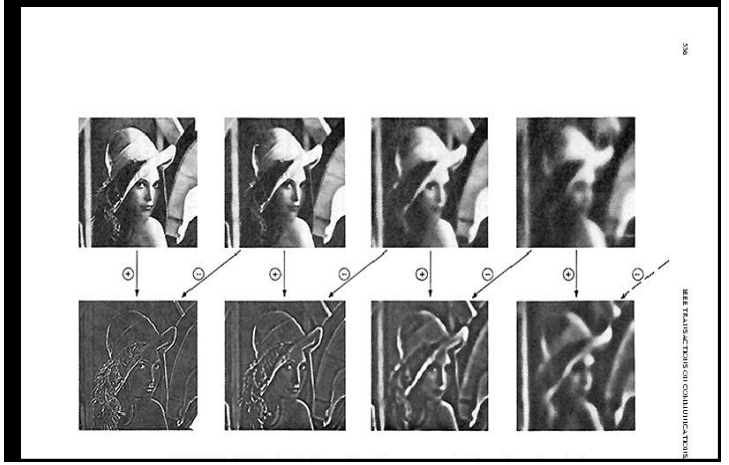
$$L_3 = g_3 - EXPAND[g_4]$$



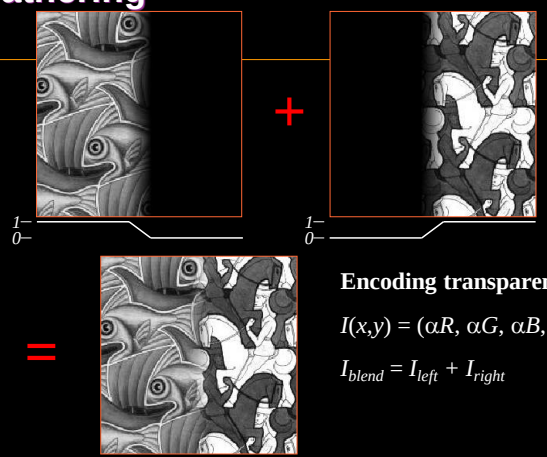
Laplacian Pyramid

- Computes a set of "**bandpass filtered**" versions of image
- $L_k = G_k – (w * G_k)$
  $\approx G_k - \text{Expand}(G_{k+1})$
- $L_N = G_N$ (apex of Laplacian pyr = apex of Gaussian pyr)
- Separates features by their scale (size)
- Enhances features
- Compact representation
- $\sum L_k = (G_0 - G_1) + (G_1 - G_2) + \ldots + (G_{N-1} - G_N) + G_N$
  $= G_0$

Gaussian and Laplacian Pyramids



Feathering

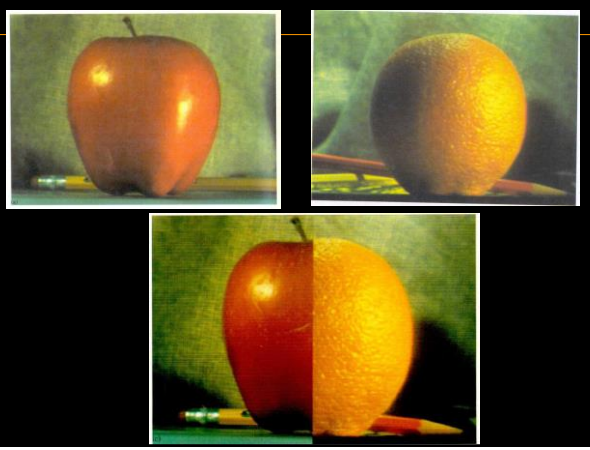


Encoding transparency

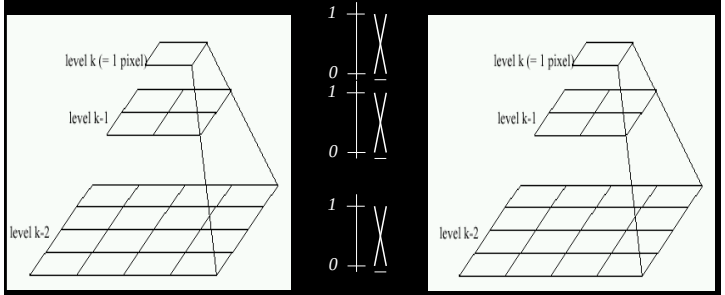
$I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$

$I_{blend} = I_{left} + I_{right}$

How much should we Blend?



Pyramid Blending



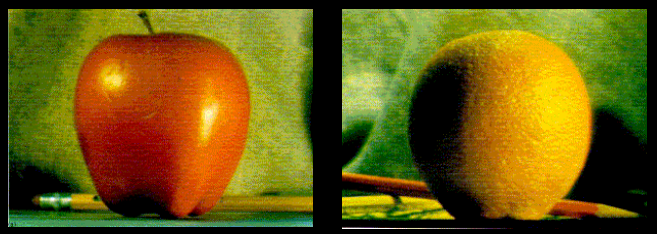
Left pyramid *blend* *Right pyramid*

Idea: At low frequencies, blend a lot; at high frequencies, blend little

Image Compositing by Pyramid Blending

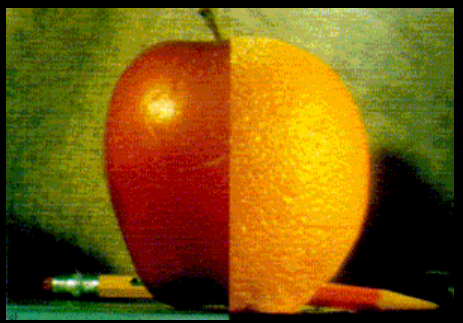
- **Given**: Two 2^n x 2^n images
- **Goal**: Create an image that contains left half of image A and right half of image B
- **Algorithm**
  - Compute Laplacian pyramids, LA and LB, from images A and B
  - Compute Laplacian pyramid LS by copying left half of LA and right half of LB. Pyramid nodes down the center line = average of corresponding LA and LB nodes $\Rightarrow$ blend along center line
  - Expand and sum levels of LS to obtain output image S

Example

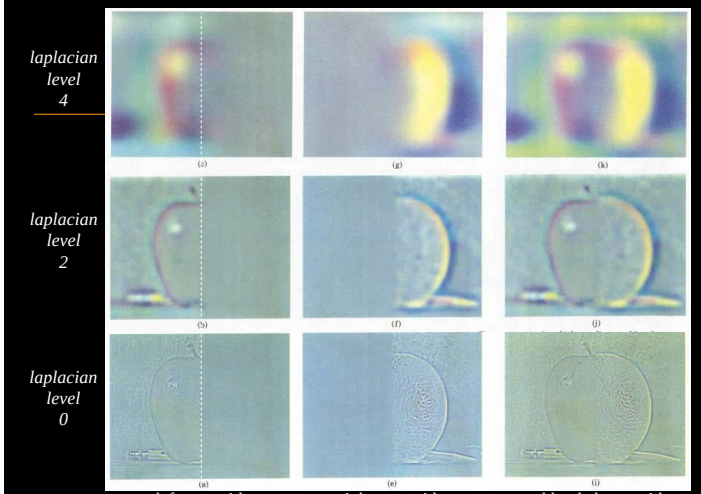


Input images A and B

Combining Apple & Orange



Left half of A + right half of B



Combining Apple & Orange using Laplacian Pyramids

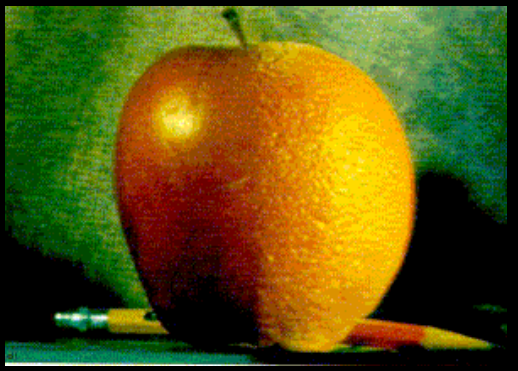


Image Compositing from Arbitrary Regions

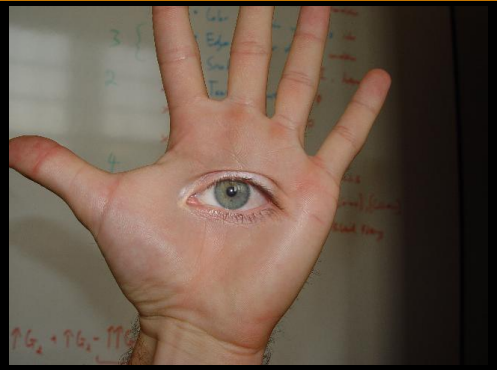
- Given: Two 2^n x 2^n images and one 2^n x 2^n binary mask
- Goal: Output image containing image A where mask=1, and image B where mask=0
- Algorithm:
  - Construct Laplacian pyramids LA and LB from images A and B
  - Construct Gaussian pyramid GR from mask R
  - Construct Laplacian pyramid LS:

$$LS_k(u, v) = GR_k(u, v)\, LA_k(u, v) + (1 - GR_k(u, v))\, LB_k(u, v)$$

  - Expand and sum levels of LS to obtain output image S

Blending Regions

Horror Photo



© prof. dmartin

Method 4: Gradient Domain Blending

- Perez, Gangnet and Blake, Poisson Image Editing, *Proc. Siggraph,* 2003
- Aka **Poisson Blending** or **Poisson Cloning**
- Similar to Photoshop's "Healing Brush"

Example: Input Images

source image

target image

Simple Cut-and-Paste Result

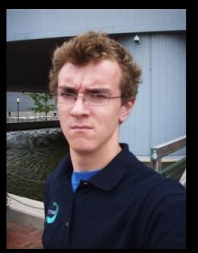
Poisson Blending Result

Poisson Blending

- Key Idea: Allow the absolute colors in foreground region to change, but preserve all the details (i.e., edges, corners)
- Blend should preserve the *gradients* of foreground region AND match background colors at seam, without changing the background
- Treat pixels as variables to be solved
  - Minimize squared difference between *gradients* of foreground region and *gradients* of output region
  - Keep background pixels constant

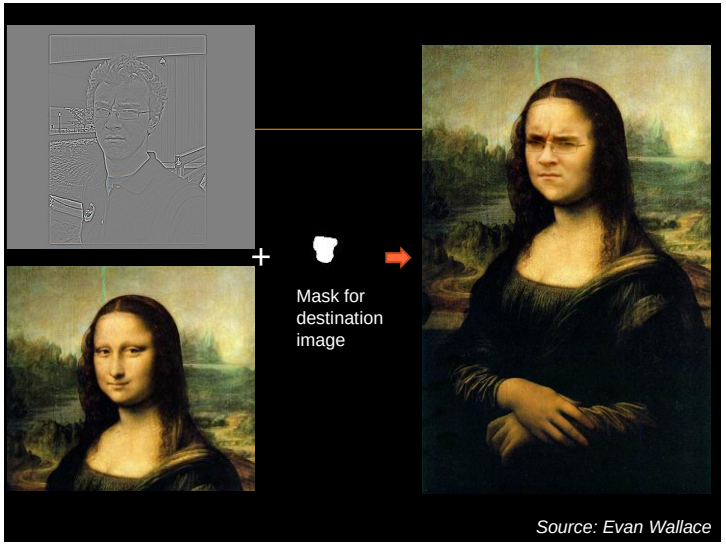
- Rather than copying pixels, copy the ***gradients*** instead; then compute the pixel values by solving a Poisson equation that matches the gradients while also satisfying fixed boundary conditions at seam

Example

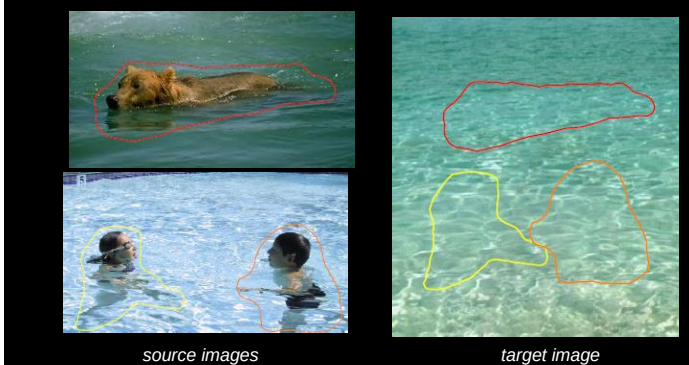


Gradient values

Source: Evan Wallace



More Results



More Results

More Results

source images

Poisson blending

Smooth Image Completion

Goal #1: Fill hole as *smoothly* as possible

Produces blurry result

*f** – *the background image*
f – *the image in the unknown area*
Ω – *the unknown area (domain of f)*
$\partial\Omega$ – boundary of Ω

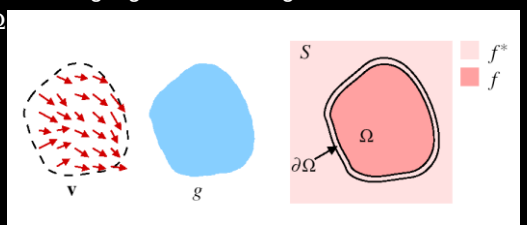
Fill missing pixels by solving:

$$\arg\min_f \iint_\Omega |\nabla f|^2 \quad s.t. \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

where $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$ = *gradient*

Poisson Blending: "Guiding" the Completion

- Use gradients from the source image (i.e., the foreground region) to *guide* the completion
- Find new pixels' values in output image's target region, *f*, so that their *gradients* are close to the gradients (vector field ***v***) of the foreground image, *g*, while holding $f = f^*$ at the boundary, $\partial\Omega$



Poisson Blending

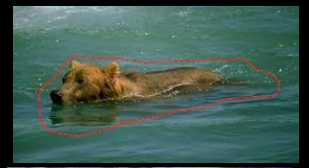
- Treat pixels as variables to be solved
  - Minimize squared difference between gradients of foreground region and gradients of output pixels
  - Match background's boundary pixels

$$\arg\min_f \iint_\Omega |\nabla f - v|^2 \quad s.t. \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Equivalent to solving a Poisson equation, which can be formulated as a discrete quadratic optimization problem and solved using Gauss-Seidel or Jacobi methods

Perez et al. 2003

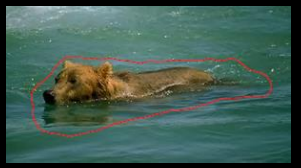
Example Result



source images, g

*target image, f**

Example Result



source images

Poisson blending result

Example Result

Note: Target and source images must be (manually) aligned

What do we Lose?

- Foreground color changes
- Background pixels in target region are replaced

Perez et al. 2003

Poisson Blending: Mixing Gradients

- Mixing gradients
  - There are situations where it is desirable to combine properties of *f** with those of *g*, for example to add objects with holes, or partially transparent ones, on top of a textured or cluttered background
  - Use gradients of source and/or destination

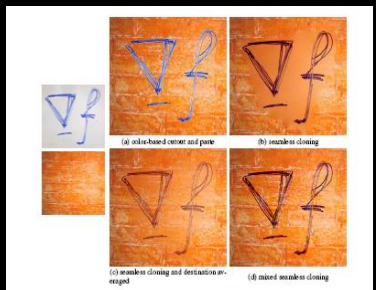


Figure 6: **Inserting objects with holes**. (a) The classic method, color-based selection and alpha masking might be time consuming and often leaves an undesirable halo; (b-c) seamless cloning, even averaged with the original image, is not effective; (d) mixed seamless cloning based on a loose selection proves effective.

Mixing Gradients

- At each point of Ω, retain the *stronger* of the variations in *f** or in *g*, using the following "guidance field:"

$$\text{for all } x \in \Omega, v(x) = \begin{cases} \nabla f^*(x) & \text{if } |\nabla f^*(x)| > |\nabla g(x)| \\ \nabla g(x) & \text{otherwise} \end{cases}$$

- The discrete counterpart of this guidance field is

$$v_{pq} = \begin{cases} f_p^* - f_q^* & \text{if } |f_p^* - f_q^*| > |g_p - g_q| \\ g_p - g_q & \text{otherwise} \end{cases}$$

In other words, look at the Laplacian at a pixel in both the source image and the target image and take whichever one is ***stronger***

Mixing Gradients: Inserting Transparent Objects

Non-linear mixing of gradient fields picks out most salient structure at each pixel

Mixing Gradients: Inserting Objects with Holes