Image-Based Rendering and Modeling

- **Image-based rendering** (IBR): A scene is represented as a collection of images
- **3D model-based rendering** (MBR): A scene is represented by a 3D model plus texture maps
- Differences
  - Many scene details need not be explicitly modeled in IBR
  - IBR simplifies model acquisition process
  - IBR processing speed independent of scene complexity
  - 3D models (MBR) are more space efficient than storing many images (IBR)
  - MBR uses conventional graphics "pipeline," whereas IBR uses pixel reprojection
  - IBR can sometimes use uncalibrated images, MBR cannot

IBR Approaches for View Synthesis

- Non-physically based image mapping
  - **Image morphing**
- Geometrically-correct pixel reprojection
  - **Image transfer** methods, e.g., in photogrammetry
- Mosaics
  - Combine two or more images into a single large image or higher resolution image
- Interpolation from dense image samples
  - Direct representation of **plenoptic function**

Image Metamorphosis (Morphing)

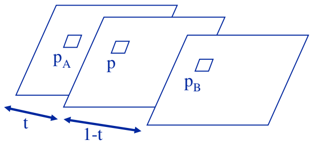
- **Goal**: Synthesize a sequence of images that smoothly and realistically transforms objects in source image A into objects in destination image B
- Method 1: 3D **Volume Morphing**
  - Create 3D model of each object
  - Transform one 3D object into another
  - Render synthesized 3D object
  - Hard/expensive to accurately model real 3D objects
  - Expensive to accurately render surfaces such as skin, feathers, fur

[Lerios, Garfinkle, & Levoy 95]

Image Morphing

- Method 2: **Image Cross-Dissolving**
  - Pixel-by-pixel color interpolation
  - Each pixel p at time $t \in [0, 1]$ is computed by combining a fraction of each pixel's color at the same coordinates in images A and B:

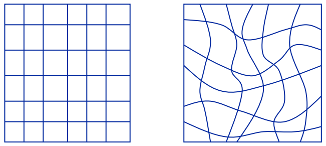
$$p = (1 - t)\, p_A + t\, p_B$$



  - Easy, but looks artificial, non-physical

Image Morphing

- Method 3: **Mesh-based image morphing**
  - G. Wolberg, *Digital Image Warping*, 1990
  - Warp between corresponding grid points in source and destination images
  - Interpolate between grid points, e.g., linearly using three closest grid points

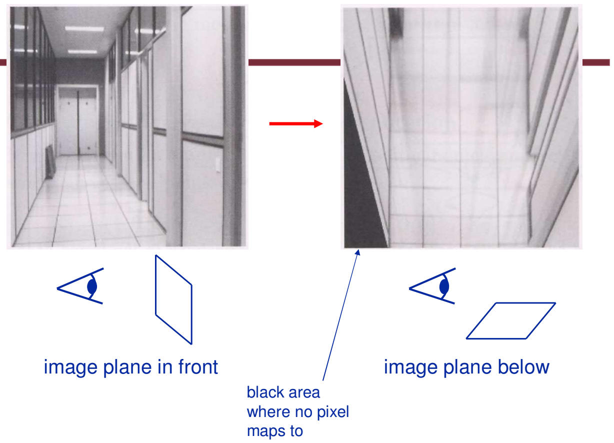


  - Fast, but hard to control so as to avoid unwanted distortions

Image Warping

- Goal: Rearrange pixels in an image. I.e., map pixels in source image A to new coordinates in destination image B
- Applications
  - Geometric Correction (e.g., due to lens pincushion or barrel distortion)
  - Texture mapping
  - View synthesis
  - Mosaics
- Aka **geometric transformation**, **geometric correction**, **image distortion**
- Some simple mappings: 2D translation, rotation, scale, affine, projective

Image Warping



Homographies

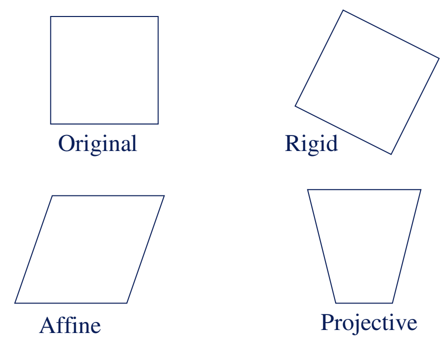
- Perspective projection of a plane
  - Lots of names for this:
    - **homography**, texture-map, colineation, planar projective map
  - Modeled as a 2D warp using homogeneous coordinates

$$\underbrace{\begin{bmatrix} sx' \\ sy' \\ s \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$

To apply a homography **H**

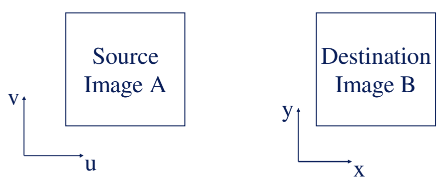
- Compute **p′** = **Hp** (regular matrix multiply)
- Convert **p′** from homogeneous to image coordinates
  - divide by *s* (third) coordinate

Examples of 2D Transformations



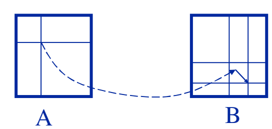
Mapping Techniques

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



Mapping Techniques

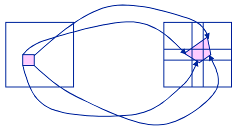
- **Forward, point-based**
  - Apply forward mapping **X**, **Y** at point (u,v) to obtain real-valued point (x,y)
  - Assign (u,v)'s gray level to pixel closest to (x,y)



  - 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

Mapping Techniques

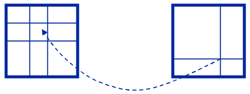
- **Forward, square-pixel based**
  - Consider pixel at (u,v) as a unit square in source image. Map square to a quadrilateral in destination image
  - Assign (u,v)'s gray level to pixels that the quadrilateral overlaps



  - 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

Mapping Techniques

- **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 Mapping

- For x = xmin to xmax
  - for y = ymin to ymax
    - u = **U**(x, y)
    - v = **V**(x, y)
    - B[x, y] = A[u, v]

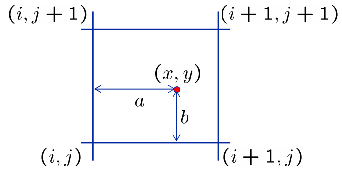
- But (u, v) may not be at a pixel in A
- (u, v) may be out of A's domain
- If **U** and/or **V** are discontinuous, A may not be connected!
- Digital transformations in general don't commute

Pixel Interpolation

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

Bilinear Interpolation

- A simple method for resampling images



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

Example of Backward Mapping

- **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 2 0 … 0
- 0-order interpolation, I.e., $u = \lfloor x/2 \rfloor$

  B = 0 … 0 2 2 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 2 1 0 … 0

Image Morphing

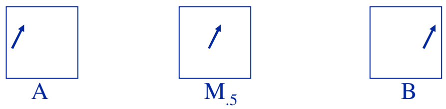
- Method 4: **Feature-based image morphing**
  - T. Beier and S. Neely, *Proc. SIGGRAPH '92*
  - Distort color *and* shape
    - ⇒ image warping + cross-dissolving
  - Warping transformation partially defined by user interactively specifying corresponding pairs of line segment features in the source and destination images; only a sparse set is required (but carefully chosen)
  - Compute dense pixel correspondences, defining continuous mapping function, based on weighted combination of displacement vectors of a pixel from all of the line segments
  - Interpolate pixel positions and colors (2D linear interpolation)

Beier and Neely Algorithm

- **Given**: 2 images, A and B, and their corresponding sets of line segments, L_A and L_B, respectively
- **Foreach** intermediate frame time t ∈ [0, 1] **do**
  - **Linearly interpolate** the position of each line
    - $L_t[i]$ = Interpolate($L_A[i]$, $L_B[i]$, t)
  - **Warp** image A to destination shape
    - WA = Warp(A, L_A, L_t)
  - **Warp** image B to destination shape
    - WB = Warp(B, L_B, L_t)
  - **Cross-dissolve** by fraction t
    - MorphImage = CrossDissolve(WA, WB, t)

Example: Translation

- Consider images where there is one line segment pair, and it is **translated** from image A to image B:

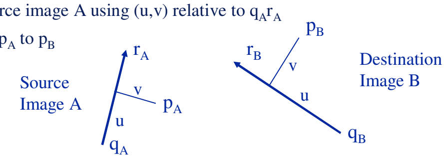


- First, linearly interpolate position of line segment in M
- Second, for each pixel (x, y) in M, find corresponding pixels in A (x-a, y) and B (x+a, y), and average them

Feature-based Warping

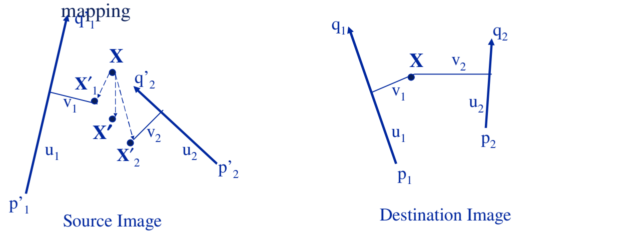
- **Goal**: Define a continuous function that warps a source image to a destination image from a sparse set of corresponding, oriented, **line segment features -** each pixel's position defined relative to these line segments
- Warping with one line pair:

foreach pixel p_B in destination image B **do**
find dimensionless coordinates (u,v) relative to oriented line segment $q_B r_B$
find p_A in source image A using (u,v) relative to $q_A r_A$
copy color at p_A to p_B

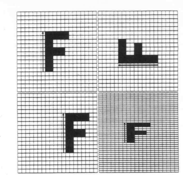


Feature-based Warping (cont.)

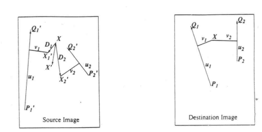
- Warping with multiple line pairs
  - Use a weighted combination of the points defined by the same mapping

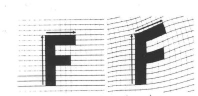


$\mathbf{X'}$ = weighted average of D_1 and D_2, where $D_i = \mathbf{X'}_i - \mathbf{X}$, and weight = $(\text{length}(p_i q_i)^c / (a + |v_i|))^b$, for constants a, b, c



- Warping with multiple line pairs:
  Use a weighted combination of the points defined by the same mapping





Geometrically-Correct Pixel Reprojection

- What geometric information is needed to generate virtual camera views?
  - Dense pixel correspondences between two input views
  - Known geometric relationship between the two cameras
    - Epipolar geometry

View Interpolation from Range Maps

- Chen and Williams, *Proc. SIGGRAPH '93* (seminal paper on image-based rendering)
- **Given**: Static 3D scene with Lambertian surfaces, and two images of that scene, each with **known camera pose** and **range map**
- **Algorithm**:

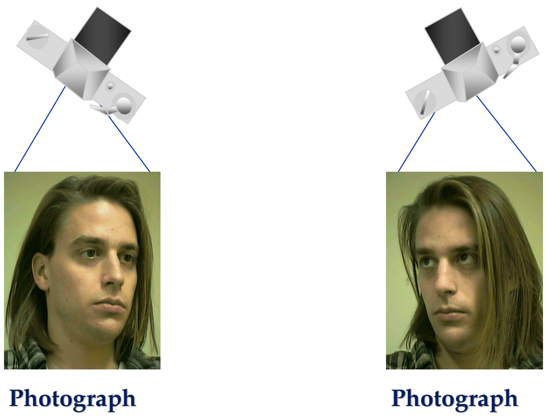
1. Recover dense pixel correspondence using known camera calibration and range maps
2. Compute forward mapping, X_F, Y_F, and backward mapping, X_B, Y_B. Each "morph map" defines an *offset vector* for each pixel

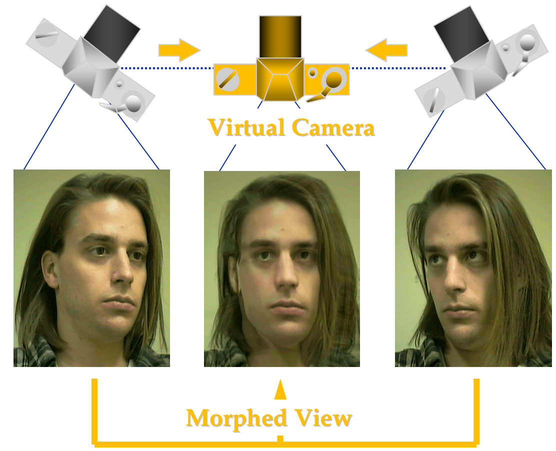
View Interpolation (cont.)

3. Compute interpolated "morph map" by linearly interpolating forward and backward offset vectors, given intermediate frame, t, $0 \le t \le 1$
4. Apply forward mapping from A given interpolated morph map (approximating perspective transformation for new view)
5. Use a z-buffer (and A's range map) to keep only closest pixel (so, handles folds)
6. For each pixel in interpolated image that has no color, interpolate color from all adjacent colored pixels

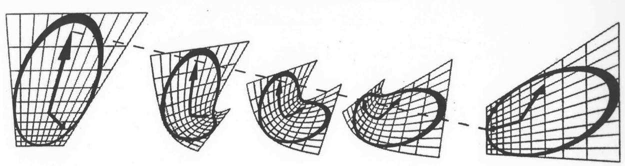
View Morphing

- Seitz and Dyer, *Proc. SIGGRAPH '95*
- **Given**: Two views of an unknown rigid scene, with no camera information known, compute new views from a virtual camera at viewpoints in-between two input views





Why NOT Image Morphing?



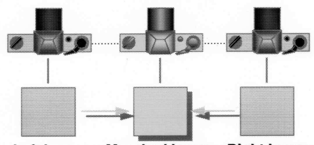
– Results are not physically consistent

11

When is View Synthesis Feasible?

- Given two images, I_0 and I_1, with optical centers C_0 and C_1, if the monotonicity constraint holds for C_0 and C_1, then there is sufficient information to completely predict the appearance of the scene from all in-between viewpoints along line segment C_0C_1
- **Monotonicity Constraint**: All visible scene points appear in the *same order* along conjugate epipolar line in I_0 and I_1
- Any number of distinct scenes could produce I_0 and I_1, but each one produces the **same** in-between images

View Morphing: Parallel Views



Morphing parallel views $\Rightarrow$ new parallel views

Parallel projection matrices have special form:

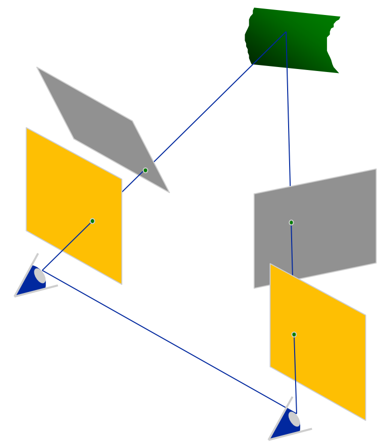
$$\mathbf{\Pi}_0 = \begin{bmatrix} 1 & 0 & 0 & C_0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{\Pi}_1 = \begin{bmatrix} 1 & 0 & 0 & C_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Linear combination of image point positions: $\mathbf{p}_0 + \mathbf{p}_1$

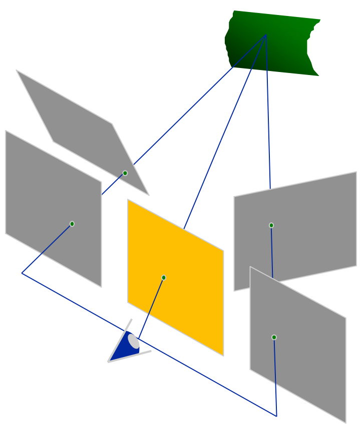
$$\Longleftrightarrow$$

Linear combination of camera positions: $\mathbf{\Pi}_0 + \mathbf{\Pi}_1$

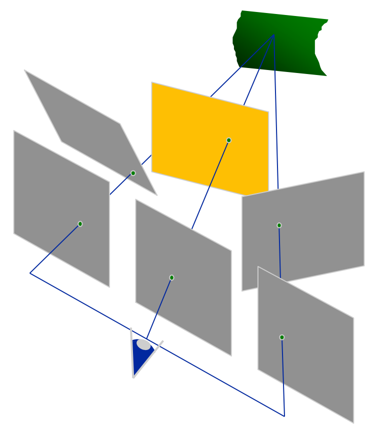
1. Prewarp
   - *align views*



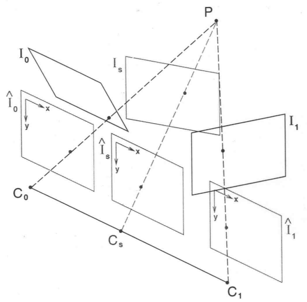
1. Prewarp
   - *align views*
2. Morph
   - *move camera*



1. Prewarp
   - *align views*
2. Morph
   - *move camera*
3. Postwarp
   - *point camera*



Interpolating Non-Parallel Views



Three Step Algorithm

1. **Prewarp**: Reproject images to parallel configuration
2. **Morph**: Interpolate parallel images
3. **Postwarp**: Reproject images to new configuration

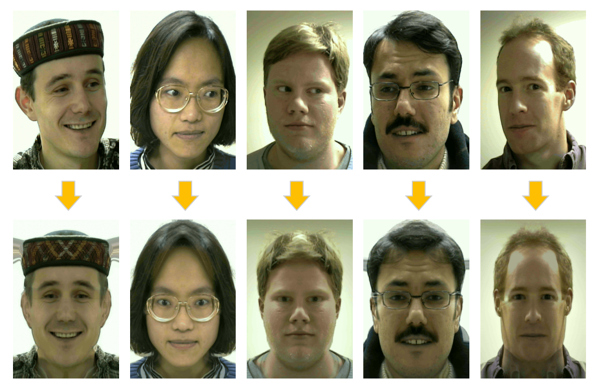
Features of View Morphing

- Provides
  - A mobile virtual camera
  - Better image morphs
- Requires no prior knowledge
  - No 3D shape information
  - No camera information
  - No training on other images

Application: Photo Correction

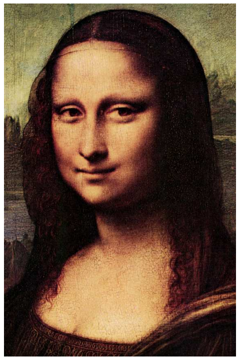
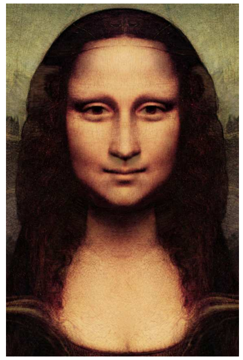
- Image Postprocessing
  - Alter image perspective in the lab
- Image Databases
  - Normalize images for better indexing
  - Simplify face recognition tasks

Original Photographs



Corrected Photographs

Another Example

Application: Better Image Transitions

- Avoid bending and shearing ***distortions***
- Shapes do not need to be ***aligned***

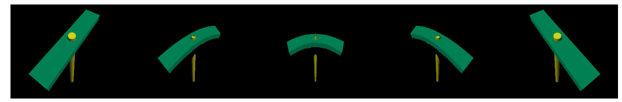


Image Morph

View Morph

Mosaics

- **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**"
- **Motivation**: Image-based modeling of 3D scenes benefits visualization, navigation, exploration, VR walkthroughs, video compression, video stabilization, super-resolution
- **Example**: Apple's Quicktime VR (Chen, *SIGGRAPH '95*)

Image Mosaics




- Goal
  - Stitch together several images into a seamless composite

Mosaicing Method

- **Registration**: Given n input images, $I_1, \ldots, I_n$, compute an image-to-image transformation that will map each image $I_2, \ldots, I_n$ into the coordinate frame of reference image, I_1
- **Warp**: Warp each image I_i, i=2, …, n, using transform
- **Interpolate**: Resample warped image
- **Composite**: Blend images together to create single output image based on the reference image's coordinate frame

When can Two Images be Aligned?

- Problems
  - In general, warping function depends on the **depth** of the corresponding scene point since image projection defined by $x' = fx/z$, $y' = fy/z$
  - Different views means, 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
  - **Panoramic mosaic: Camera rotates (i.e., pans) about its optical center, arbitrary 3D scene**
    - No motion parallax as camera rotates, so depth unimportant
    - 2D projective transformation relates any 2 images (⇒ 8 unknowns)
  - **Planar mosaic**: Arbitrary camera views of a **planar scene**
    - 2D projective transformation relates any 2 images

PANORAMIC MOSAICING

- ANY 2 IMAGES of AN ARBITRARY SCENE TAKEN FROM 2 CAMERAS WITH THE SAME CAMERA CENTER ARE RELATED BY THE PLANAR PROJECTIVE TRANSFORMATION:

$$\tilde{w}' = KRK^{-1}\tilde{w}$$

called the "homography induced by the plane at infinity"

where $\tilde{w}$, $\tilde{w}'$ are homogeneous coordinates of 2 corresponding points in the two input images

K is the 3×3 upper-triangular camera calibration matrix: $\begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$

R is the 3×3 rotation matrix:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

4 dof + 3 dof ⇒ 4 point correspondences needed

Or, generalizing to the projective camera:

$$\tilde{w}' = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \tilde{w}$$

8 degrees of freedom

$m_{33} = 1$

4 point correspondences needed

How to Determine Alignment Assuming a Planar Projective Transformation between Cameras?

- Method 1: Find 4 point correspondences, and then solve for 8 unknowns
  - Requires reliable detection of four corresponding features, at sub-pixel location accuracy
- Method 2: Use image-based (intensity) correlation to determine best matching transformation
  - No correspondences needed
  - Statistically optimal (gives maximum likelihood estimate)
  - Useful for local image registration

Example: 2D Rigid Warp Mosaics

- **Assume**: Planar scene, camera motion restricted to plane parallel to scene, optical axis perpendicular to scene, intensity 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 tem 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, $\nabla E = (\partial E/\partial a, \partial E/\partial b, \partial E/\partial\theta)$
- Iteratively update total motion estimate, (a, b, θ), while warping I' towards I until E < threshold

2D Rigid Warp Algorithm

- Because the displacement between images may not be small enough to solve directly for the motion parameters, use an iterative algorithm instead:
  1. $a^{(0)} = 0$; $b^{(0)} = 0$; $\theta^{(0)} = 0$; $\mathbf{m} = (0, 0, 0)$; $t = 1$
  2. Solve for $a^{(t)}$, $b^{(t)}$, $\theta^{(t)}$ from the 3 equations
  3. Update the total motion estimate:
     $\mathbf{m}^{(t+1)} = \mathbf{m}^{(t)} + (a^{(t)}, b^{(t)}, \theta^{(t)})$
  4. Warp I' toward I: $I' = \text{warp}(I', a^{(t)}, b^{(t)}, \theta^{(t)})$
  5. **If** ($a^{(t)} < \varepsilon_1$ **and** $b^{(t)} < \varepsilon_2$ **and** $\theta^{(t)} < \varepsilon_3$)
     then halt
     else $t = t + 1$; **goto** step 2

More generally, Szeliski paper

minimizes $E = \sum_i \left[I'(x_i', y_i') - I(x_i, y_i) \right]^2$

$= \sum_i e_i^2$

by computing $\frac{\partial e_i}{\partial m_0}, \ldots, \frac{\partial e_i}{\partial m_7}$

E.g., $\frac{\partial e_i}{\partial m_0} = \frac{x_i}{m_6 x_i + m_7 y_i + 1} \cdot \frac{\partial I'}{\partial x'}$

Then uses Levenberg-Marquardt iterative, nonlinear minimization algorithm to solve for $m_0, \ldots, m_7$.

Dealing with Noisy Data: RANSAC

- How to find the best fitting data to a global model when the data are noisy – especially because of the presence of outliers, i.e., missing features and extraneous features?
- **RANSAC** (Random Sample Consensus) Method
  - Iteratively select a small subset of data and fit model to that data
  - Then check all of data to see how many fit the model

RANSAC Algorithm for Robust Estimation

bestcnt := 0
until there is a good fit or *k* iterations **do**
  randomly choose a sample of *n* points from the dataset
  compute the best fitting model parameters to the selected subset of the *n* data points
  cnt := 0
  foreach remaining data point **do**
    if the distance from the current point to the model is < T **then** cnt++
  if cnt ≥ D **then** there is a good fit, so re-compute the best fitting model parameters using all of the good points, and **halt**
  else if cnt > bestcnt **then** bestcnt := cnt

Other Robust Parameter Estimation Methods

- How to deal with outliers, i.e., occasional large-scale measurement errors?
- **M-Estimators**

$$\arg\min_{\theta} \sum_{x_i \in X} \rho(r_i(x_i, \theta); \sigma)$$

  - Generalization of
    - MLE (Maximum Likelihood Estimation)
    - LMedS (Least Median of Squares)
  - Θ is the set of model parameters, ρ is a "robust loss function" whose shape is parameterized by σ, and $r_i(x_i, \theta)$ is the residual error of the model θ with the i^{th} data element

Least Median of Squares (LMedS)

- Defined as

$$\arg\min_{\theta} \operatorname{median}_{x_i \in X} r^2(x_i, \theta)$$

- Up to ½ of data points can be arbitrarily far from the optimum estimate without changing the result
- Median is not differentiable, so how to search for optimum?
  - Use randomization:
    - Randomly select a subset of the data
    - Compute model, θ, from the selected subset
    - Each candidate model, θ, is tested by computing r^2 using the remaining data points and selecting the median of these values

Global Image Registration

- When images are not sufficiently close, must do global registration
- Method 1: **Coarse-to-fine matching using Gaussian Pyramid**
  1. Compute Gaussian pyramids
  2. *level* = N // Set initial processing level to coarse 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

Global Image Registration

- Method 2: **Coarse-to-fine matching using Laplacian Pyramid**
  - Use Laplacian pyramids, *LA* and *LB*
  - **for** *l* = N **to** 0 **step** -1 **do**

Use motion estimate from previous iteration to warp level *l* in LA

for $-1 \le i, j \le 1$ **do**

compute cross-correlation at level *l*:

$$CC_{i,j}(x, y) = LA_l(x, y) LB_l(x+i, y+j)$$

smooth using Gaussian pyramid to level *S*:

$$C_{i,j}(x, y) = CC_{i,j}(x, y) * w(x, y)$$

foreach (*x*,*y*) interpolate 3 x 3 correlation surface centered at (*x*,*y*) at level S to find peak, corresponding to best local motion estimate

find best-fit global motion model to flow field

Planar Mosaics and Panoramic Mosaics

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

Panoramic Mosaics

- Large field of view ⇒ can’t map all images onto a plane
- Instead, map onto cylinder, sphere, or cube
- Example: With a cylinder, first warp all images from rectilinear to cylindrical coordinates, then mosaic them
- “Undistort” (perspective correct) image from this representation prior to viewing

Cylindrical panoramas

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

Cylindrical reprojection

Image 384x300 **f = 180 (pixels)** **f = 280** **f = 380**

- Map image to cylindrical coordinates
  - need to know the camera focal length

Cylindrical image stitching

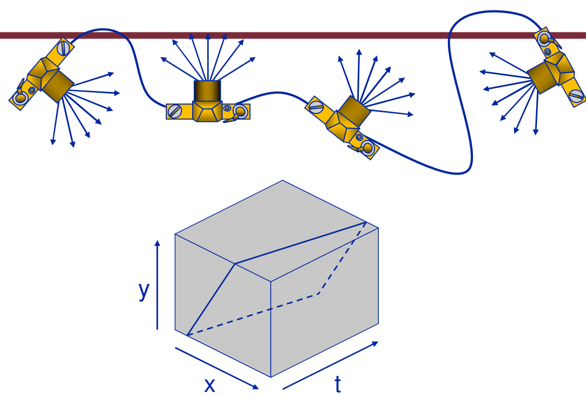
- What if you don’t know the camera rotation?
  - Solve for the camera rotations
    - Note that a rotation of the camera is a **translation** of the cylinder!
    - Use Lucas-Kanade to solve for translations of cylindrically-warped images

Full-view Panorama

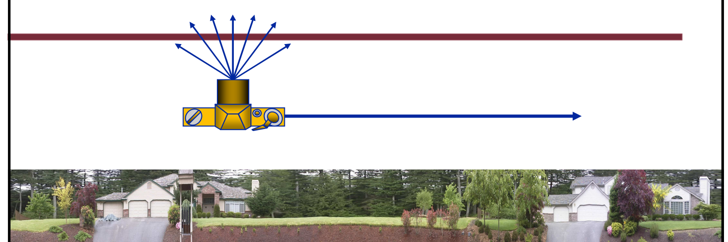
Other types of mosaics

- Can mosaic onto *any* surface if you know the geometry
  - See NASA's Visible Earth project for some stunning earth mosaics
    - http://earthobservatory.nasa.gov/Newsroom/BlueMarble/

Path Images

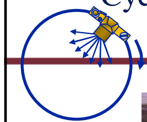


Pushbroom image



http://grail.cs.washington.edu/projects/stereo/

Cyclograph



http://grail.cs.washington.edu/projects/stereo/

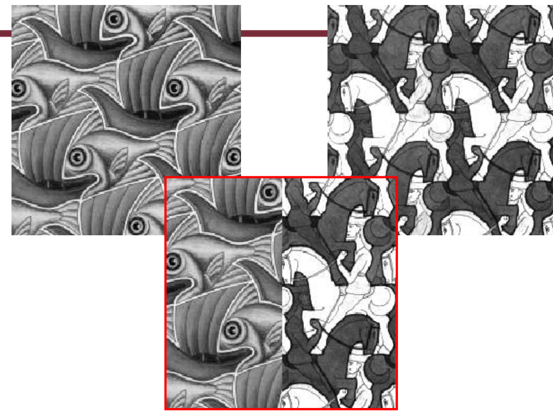
Multi-Perspective Image

Rademacher and Bishop,1998

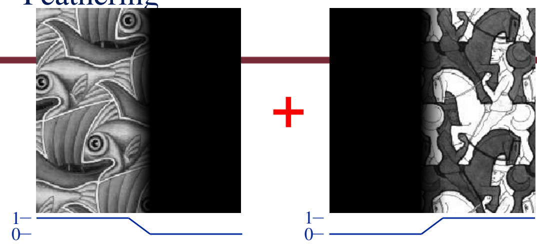
Some Blending Methods

- Average intensity of overlapping pixels
- Median intensity of overlapping pixels
- Newest (i.e., most recent frame) pixel's intensity
- Burt's pyramid blending method
- Bilinear blending
  - Weighted average with pixels near center of each image contributing more
  - Let w_t be a 1D triangle (hat) function of size equal to width of image, with value 0 at edges and value 1 at midpoint. Then use 2D weighting function: $w(x_i', y_i') = w_t(x_i')w_t(y_i')$

Image Blending



Feathering



=

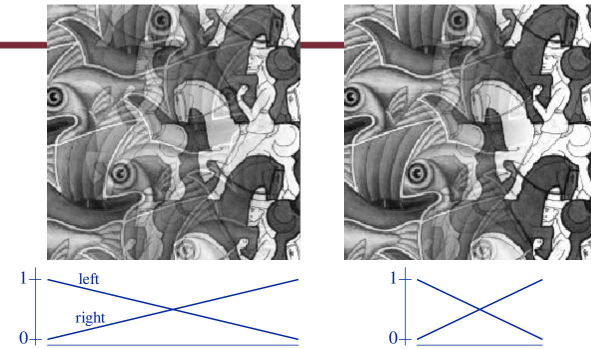
Encoding transparency

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

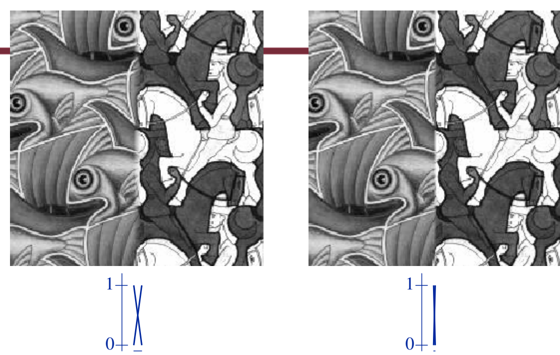
$I_{blend} = I_{left} + I_{right}$

(J. Blinn, *CGA*, 1994) for details

Effect of window size



Effect of window size



Good window size

- "Optimal" window: smooth but not ghosted
  - Doesn't always work...

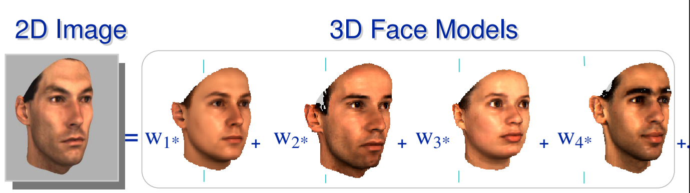
Mosaicing Arbitrary Scenes

- For general scenes and arbitrary viewpoints, we must recover **depth**
- **Method 1**: Depth map given ala Chen and Williams
- **Method 2**: Segment image into piecewise-planar patches and use 2D projective method for each patch
- **Method 3**: Recover dense 3D depth map using either **stereo reconstruction** (assuming known motion between cameras) or **structure from motion** (assuming no camera motion is known)

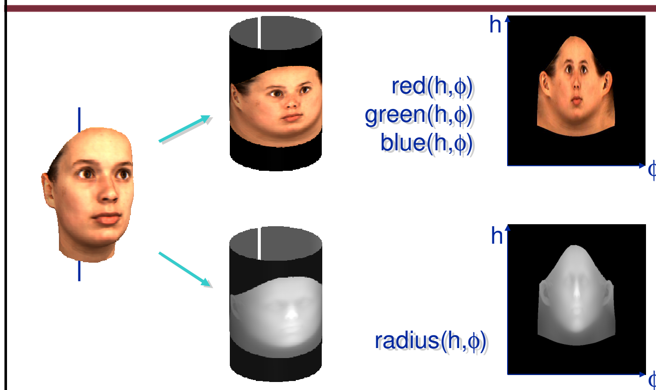
Learning-based View Synthesis from 1 View

- Vetter and Poggio, *IEEE Trans. PAMI*, 1997
- **Given**: A set of views of several training objects, and a single view of a new object
- Single view of new object considered to be approximated by a linear combination of views of the training objects, all at the same pose as the new object
- Learn set of weights that specify best match between given view of new object and same view of the training objects
- Use learned weights with other views of training objects to synthesize novel view of new object

Approach: Example-based Modeling of Faces

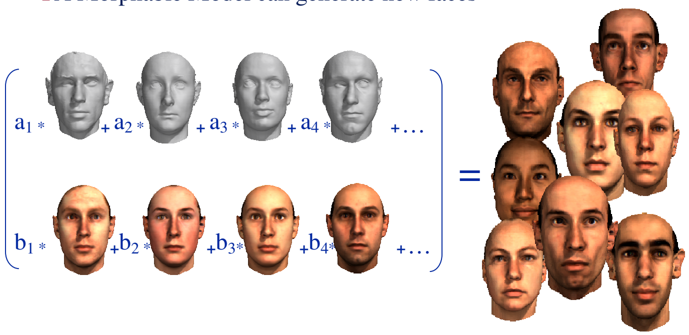


Cylindrical Coordinates

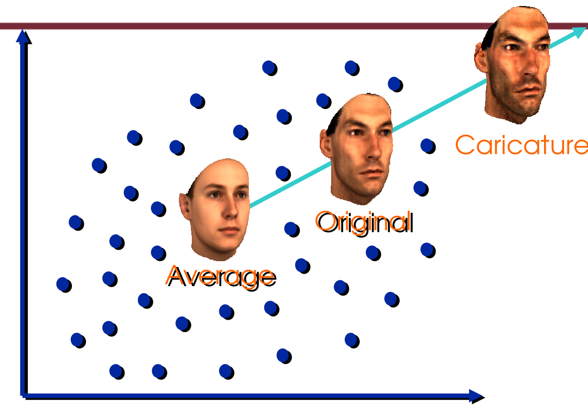


Vector Space of 3D Faces

- A Morphable Model can generate new faces



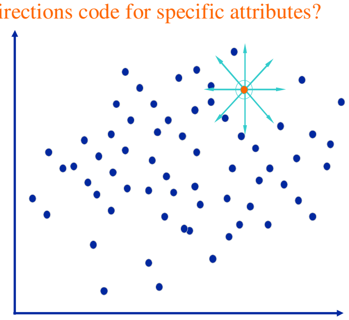
Modeling in Face Space



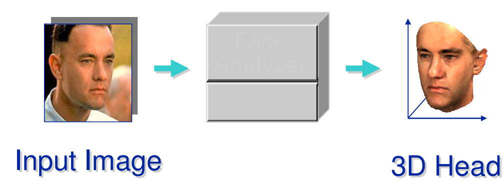
Modeling the Appearance of Faces

A face is represented as a point in face space

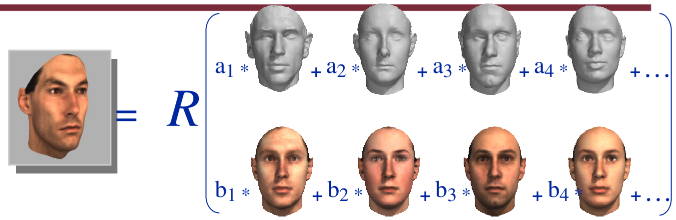
- Which directions code for specific attributes?



3D Shape from Images



Matching a Morphable 3D Face Model



Optimization problem!

Vetter's Method

Given an input image of a face

1. Compute correspondence with a "reference face" in the same pose

   (Uses coarse-to-fine optical flow algorithm using a Laplacian pyramid)

   Result is a "shape vector"

   $s = (x_1, y_1, \dots, x_n, y_n)$

2. Compute "texture vector"

   Difference map between corresponding pixels' image intensities

   $t = (i_1, \dots, i_n)$

3. Linear Shape Model of Faces

   Given a database of 3D models of faces, any new face can be described by

   $$S = \sum_{i=1}^{g} \beta_i S_i$$

   where $S = (x_1, y_1, z_1, \dots, x_n, y_n, z_n)$ is the new face's 3D shape and S_i is the 3D shape of the i^{th} face in the DB.

   * Arbitrarily rotating 3D shape and projecting into an image does <u>not</u> change the β_i's!

   $$s_r = \sum_{i=1}^{g} \beta_i s_i^r$$

   where $s_r = PS^r$, P = perspective projection

4. Compute linear shape approximation

   → Find best β_i's (min error) so that

   $$s^r = \sum_i \beta_i s_i^r$$

   where s^r = input image face

   s_i^r = i^{th} reference face image at same pose

5. Compute new view's shape vector

   $$s^f = \sum_i \beta_i s_i^f$$

6. Compute linear texture approximation

   $$t^r = \sum \alpha_i t_i^r$$

7. Compute new view's texture vector

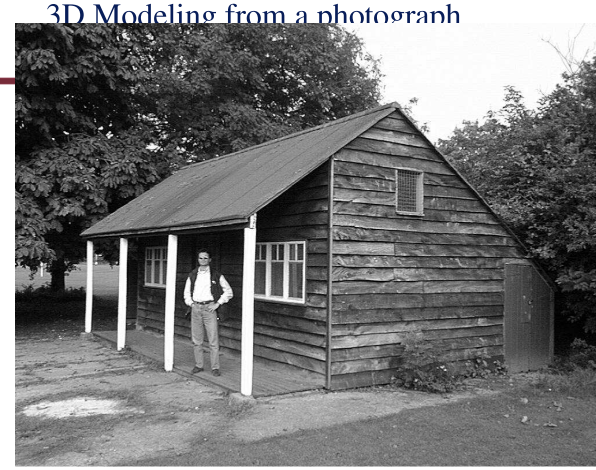
$$t^f = \sum \alpha_i t_i^f$$

Note: This texture mapping is not correct except for Lambertian surfaces.
Alternatively, use 3D head model to remap texture

8. Warp texture onto shape

$$s^f + t^f$$

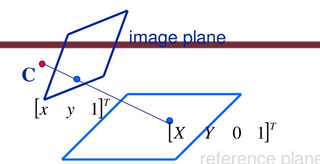
3D Modeling from a photograph



Criminisi et al., ICCV 99

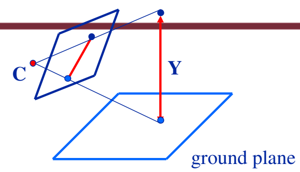
- Complete approach
  - Load in an image
  - Click on lines parallel to X axis
    - repeat for Y, Z axes
  - Compute vanishing points
  - Specify 3D and 2D positions of 4 points on reference plane
  - Compute homography H
  - Specify a reference height
  - Compute 3D positions of several points
  - Create a 3D model from these points
  - Extract texture maps
  - Output a VRML model

Measurements within reference plane



- Solve for homography **H** relating reference plane to image plane
  - **H** maps reference plane (X,Y) coords to image plane (x,y) coords
  - Fully determined from 4 known points on ground plane
    - Option A: physically measure 4 points on ground
    - Option B: find a square, guess the dimensions
  - Given (x, y), can find (X,Y) by $\mathbf{H}^{-1}$

Measuring height without a ruler

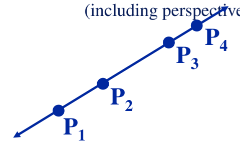


- Compute Y from image measurements
  - Need more than vanishing points to do this

The cross ratio

- A Projective Invariant
  - Something that does not change under projective transformations (including perspective projection)

The cross-ratio of 4 collinear points

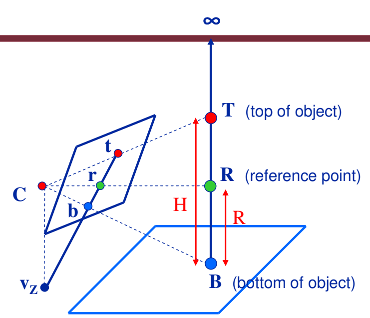


$$\frac{\|\mathbf{P}_3-\mathbf{P}_1\|\,\|\mathbf{P}_4-\mathbf{P}_2\|}{\|\mathbf{P}_3-\mathbf{P}_2\|\,\|\mathbf{P}_4-\mathbf{P}_1\|}$$

$$\mathbf{P}_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

- Can permute the point ordering $\frac{\|\mathbf{P}_1-\mathbf{P}_3\|\,\|\mathbf{P}_4-\mathbf{P}_2\|}{\|\mathbf{P}_1-\mathbf{P}_2\|\,\|\mathbf{P}_4-\mathbf{P}_3\|}$
  - 4! = 24 different orders (but only 6 distinct values)
- This is the fundamental invariant of projective geometry

Measuring height



$$\frac{\|\mathbf{T}-\mathbf{B}\|\,\|\infty-\mathbf{R}\|}{\|\mathbf{R}-\mathbf{B}\|\,\|\infty-\mathbf{T}\|} = \frac{H}{R}$$

scene cross ratio

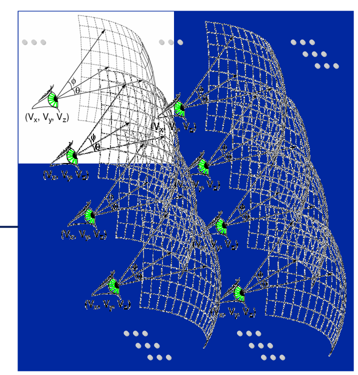
$$\frac{\|\mathbf{t}-\mathbf{b}\|\,\|\mathbf{v}_Z-\mathbf{r}\|}{\|\mathbf{r}-\mathbf{b}\|\,\|\mathbf{v}_Z-\mathbf{t}\|} = \frac{H}{R}$$

image cross ratio

scene points represented as $\mathbf{P} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ image points as $\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Plenoptic Function (Adelson and Bergen 1991)

- For a given scene, describe ALL rays through
  - ALL pixels, of
  - ALL cameras, at
  - ALL wavelengths,
  - ALL time



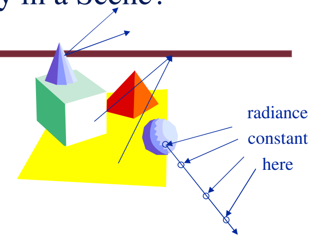
F(x,y,z, ϕ,θ, λ, t)
"eyeballs everywhere" function (5D x 2D)

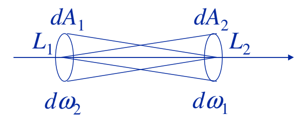
Plenoptic Array: 'The Matrix Effect'

- Brute force! Simple arc, line, or ring array of cameras
- Synchronized shutter – Dayton Taylor's TimeTrack
- Warp/blend between images to change viewpoint on 'time-frozen' scene:


How Much Light is Really in a Scene?

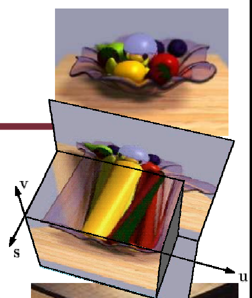
- Light transported throughout scene along rays
  - Anchor
    - Any point in 3D space
    - 3 coordinates
  - Direction
    - Any 3D unit vector
    - 2 angles
  - Total of 5 dimensions
- Radiance remains constant along ray as long as in empty space
  - Removes one dimension
  - Total of 4 dimensions

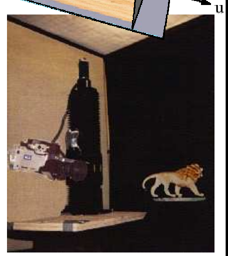




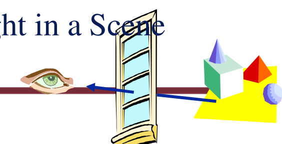
Light Field and Lumigraph

- Light Field
  - Levoy & Hanrahan, Siggraph 1996
- Lumigraph
  - Gortler *et al.*, Siggraph 1996
- Consider (u,v) the image plane and (s,t) the viewpoint plane
- Photographs taken from a bunch of different viewpoints
- Reconstructed photographs of scene are 2D slices of 4D light field

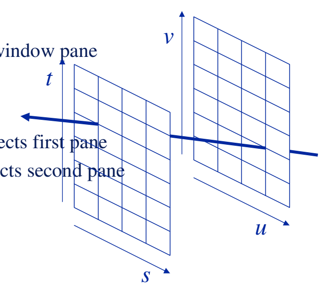




Representing All of the Light in a Scene



- View scene through a window
- All visible light from scene must have passed through window
- Window light is 4D
  - 2 coordinates where ray intersects window pane
  - 2 angles for ray direction
- Use a double-paned window
  - 2 coordinates (u,v) where ray intersects first pane
  - 2 coordinates (s,t) where ray intersects second pane



Object Based

- Can be made a cube around object

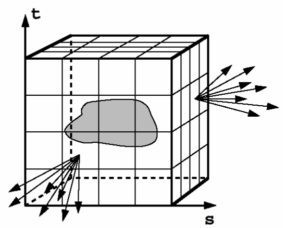
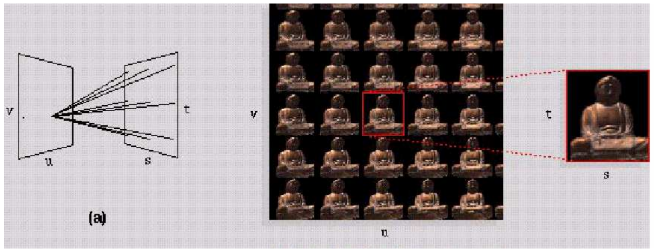
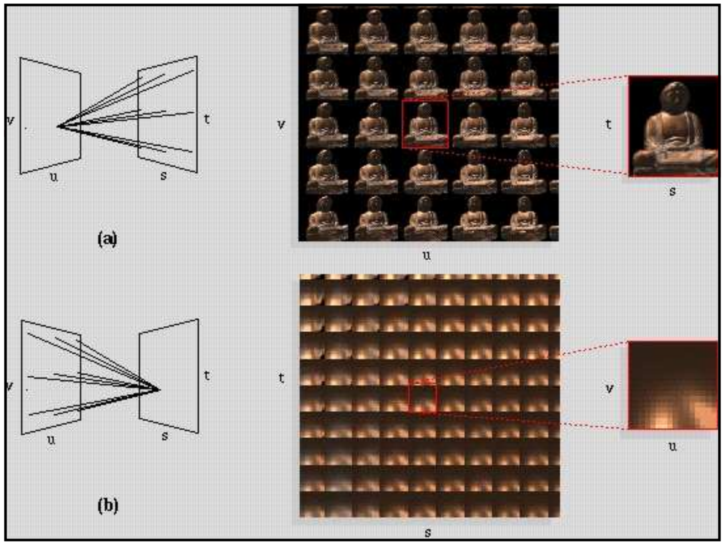


Figure 1: The surface of a cube holds all the radiance information due to the enclosed object.

Images from uv Plane





Making Light Field/Lumigraph

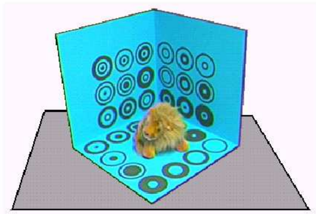
- Rendered from synthetic model
- Made from real world
  - With gantry
  - Handheld

Gantry

- Lazy Susan
  - Manually rotated
- XY Positioner
- Lights turn with lazy susan
- Correctness by construction

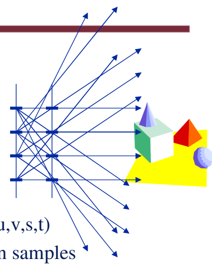
Handheld Camera

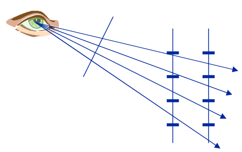
- Blue screen on stage
  - Walls moveable, can turn



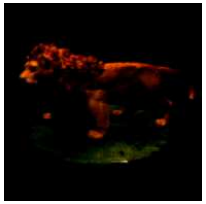
Ray Tracing and Light Fields

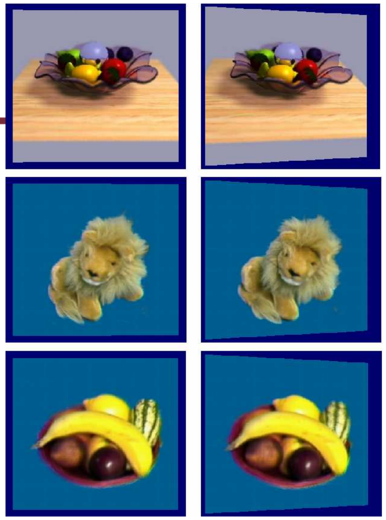
- Rendering **into** a light field
  - Cast rays between all pairs of points in panes
  - Store resulting radiance at (u,v,s,t)
- Rendering **from** a light field
  - Cast rays through pixels into light field
  - Compute two ray-plane intersections to find (u,v,s,t)
  - Interpolate u,v and s,t to find radiance between samples
  - Plot radiance in pixel





Results





Results

- Light Field

	buddha	**kidney**	**hallway**	**lion**
Number of slabs	1	1	4	4
Images per slab	16x16	64x64	64x32	32x16
Total images	256	4096	8192	2048
Pixels per image	256^2	128^2	256^2	256^2
Raw size (MB)	50	201	1608	402
Prefiltering	uvst	st only	uvst	st only

Table I: Statistics of the light fields shown in figure 14.