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
  - Aka **image transfer** methods 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

[Images of skull morphing]

[Lerios, Gershman, & 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

- Method 3: **Mesh-based image morphing**
  - 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

Homogeneous Coordinates

- Represent a point (x, y) in the Euclidean plane in the 3D projective plane: (x, y) → (x’, y’, w) = (xw, yw, w), where w is any non-zero value
- Homogeneous coordinates (x, y, w) correspond to Cartesian coordinates (x/w, y/w). I.e., a point projects to the w=1 plane
2D Mappings

- 2D translation - 2 DOFs
  \[ \begin{cases} x = u + a \\ y = v + b \end{cases} \]
  \[ [x, y, 1]^T = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \]

- 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 transformation (translation and rotation)

2D Mappings (cont.)

- 2D scale (similarity transformation) - 2 DOFs
  \[ \begin{cases} x = \alpha u \\ y = \beta v \end{cases} \]
  \[ [x, y, 1]^T = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \]

- Composite translation, rotation, scale - 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 (allows skewing) - 6 DOFs
  \[
  \begin{align*}
  x &= a_{11}u + a_{12}v + a_{13} \\
  y &= a_{21}u + a_{22}v + a_{23} \\
  [x, y, 1]^T &= \begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    0 & 0 & 1
  \end{bmatrix} \begin{bmatrix}
    u \\
    v \\
    1
  \end{bmatrix}
  \end{align*}
  \]

- Projective - 8 DOFs (\(a_{33}\) is a scale factor)
  \[
  \begin{align*}
  x &= \frac{a_{1}u + a_{2}v + a_{3}}{a_{4}u + a_{5}v + 1} \\
  y &= \frac{b_{1}u + b_{2}v + b_{3}}{a_{4}u + a_{5}v + 1} \\
  [x, y, 1]^T &= \begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
  \end{bmatrix} \begin{bmatrix}
    u \\
    v \\
    1
  \end{bmatrix}
  \end{align*}
  \]

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
  \[
  \begin{bmatrix}
    wx' \\
    wy' \\
    w
  \end{bmatrix} = \begin{bmatrix}
    * & * & * \\
    * & * & * \\
    * & * & *
  \end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    1
  \end{bmatrix}
  \]
  \[
  \begin{bmatrix}
    w \\
    w
  \end{bmatrix} = \begin{bmatrix}
    * & * & * \\
    * & * & *
  \end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    1
  \end{bmatrix}
  \]
  \[
  p' = H \cdot p
  \]
  - To apply a homography \(H\)
    - Compute \(p' = Hp\) (regular matrix multiply)
    - Convert \(p'\) from homogeneous to image coordinates
      - divide by \(w\) (third) coordinate
Examples of 2D Transformations

Original

Rigid

Affine

Projective

3D Mappings

- Cartesian coordinates \((x, y, z) \rightarrow (x, y, z, w)\) in homogeneous coordinates
- \(4 \times 4\) matrix for affine transformations:

\[
\begin{bmatrix}
    r_{11} & r_{12} & r_{13} & t_x \\
    r_{21} & r_{22} & r_{23} & t_y \\
    r_{31} & r_{32} & r_{33} & t_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

where \(r_{ij}\) specify aggregate rotation and scale change, and \(t_i\) specify translation
Mapping Techniques

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

![Diagram of Source Image A and Destination Image B](image)

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 \((x, y)\)

![Diagram of A and B](image)

- 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 $x = \text{xmin to xmax}$
  - for $y = \text{ymin to ymax}$
    - $u = U(x, y)$
    - $v = V(x, y)$

- 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**
  - $g(x, y) = \text{gray level at nearest pixel (i.e., round (x, y) to nearest integers)}$
  - 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 $g(x, y)$, 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

\[
\begin{array}{c}
(i, j) \\
\downarrow u \\
(x, y) \\
\downarrow b \\
(i + 1, j)
\end{array}
\]

\[
f(x, y) = (1 - a)(1 - b) f[i, j] \\
+ a(1 - b) f[i + 1, j] \\
+ ab f[i + 1, j + 1] \\
+ (1 - a)b f[i, j + 1]
\]

Example of Backward Mapping

- **Goal**: Define a transformation that performs a scale change, which expands size of image by 2, i.e., \( U(x) = x/2 \)
- \( A = 0 \ldots 0 2 2 2 0 \ldots 0 \)
- 0-order interpolation, i.e., \( u = \lfloor x/2 \rfloor \)
  \[
  B = 0 \ldots 0 2 2 2 2 2 0 \ldots 0
  \]
- Bilinear interpolation, i.e., \( u = x/2 \) and average 2 nearest pixels if \( u \) is not at a pixel
  \[
  B = 0 \ldots 0 1 2 2 2 2 1 0 \ldots 0
  \]
Image Morphing

- **Method 4: Feature-based image morphing**
  - T. Beier and S. Neely, *Proc. SIGGRAPH ‘92*
  - Distort color and shape
    \[\Rightarrow\] 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 \(t \in [0, 1]\) do
  - **Linearly interpolate** the position of each line
    \[L_{it}[i] = \text{Interpolate}(L_{tA}[i], L_{tB}[i], t)\]
  - **Warp** image A to destination shape
    \[WA = \text{Warp}(A, L_A, L_{it})\]
  - **Warp** image B to destination shape
    \[WB = \text{Warp}(B, L_B, L_{it})\]
  - **Cross-dissolve** by fraction t
    \[\text{MorphImage} = \text{CrossDissolve}(WA, WB, t)\]
Example: Translation

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

\[ \text{A} \qquad \text{M}_{0.5} \qquad \text{B} \]

- 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:
  
  \[
  \text{foreach pixel } p_{B} \text{ in destination image } B \text{ do} \\
  \text{find dimensionless coordinates } (u,v) \text{ relative to oriented line segment } q_{B}r_{B} \\
  \text{find } p_{A} \text{ in source image } A \text{ using } (u,v) \text{ relative to } q_{A}r_{A} \\
  \text{copy color at } p_{A} \text{ to } p_{B} \\
  \]
Feature-based Warping (cont.)

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

\[
\begin{align*}
X' &= \text{weighted average of } D_1 \text{ and } D_2, \text{ where } D_i = X'_i - X, \\
\text{and weight} &= (\text{length}(p_iq_i)^a / (a + |v_i|)^b), \text{ for constants } a, b, c
\end{align*}
\]
Morphing between Two Image Sequences

- **Goal:** Given two animated sequences of images, create a morph sequence
- User defines corresponding line segments in pairs of key frames in the two sequences
- At frame $i$, compute the two sets of line segments by interpolating between the nearest bracketing key frame’s line sets
- Apply 2-image morph algorithm for $t = 0.5$ only to obtain morph frame $i$

Learning-based View Synthesis from 1 View

- **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
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, $t$, $0 \leq t \leq 1$
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
When is View Synthesis Feasible?

- Given two images, I₀ and I₁, with optical centers C₀ and C₁, if the monotonicity constraint holds for C₀ and C₁, then there is sufficient information to completely predict the appearance of the scene from all in-between viewpoints along line segment C₀C₁
- **Monotonicity Constraint**: All visible scene points appear in the *same order* along conjugate epipolar line in I₀ and I₁
- Any number of distinct scenes could produce I₀ and I₁, but each one produces the *same* in-between images
Morphing parallel views ⇒ new parallel views

Parallel projection matrices have special form:

\[
P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad P_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

Linear combination of image point positions: \( p_0 + p_1 \)

Linear combination of camera positions: \( P_0 + P_1 \)

1. Prewarp

⇒ align views
1. Prewarp
   ➡️ *align views*

2. Morph
   ➡️ *move camera*

3. Postwarp
   ➡️ *point camera*
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

Three Step Algorithm
1. **Prewarp**: Reproject images to parallel configuration
2. **Morph**: Interpolate parallel images
3. **Postwarp**: Reproject images to new configuration
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
Application: Better Image Transitions

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

![Image Morph](image1.png)

![View Morph](image2.png)

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, I₁, …, Iₙ, compute an image-to-image transformation that will map each image I₂, …, Iₙ into the coordinate frame of reference image, I₁
- **Warp**: Warp each image Iᵢ, 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 (\( \Rightarrow 8 \) 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:
  \[
  \tilde{w}' = KRK\tilde{w}
  \]
  where \( \tilde{w}, \tilde{w}' \) are homogeneous coordinates of 2 corresponding points in the two input images
  - \( K \) is the \( 3 \times 3 \) upper-triangular camera calibration matrix:
    \[
    K = \begin{bmatrix}
    f_x & 0 & u_0 \\
    0 & f_y & v_0 \\
    0 & 0 & 1
    \end{bmatrix}
    \]
  - \( R \) is the \( 3 \times 3 \) rotation matrix:
    \[
    R = \begin{bmatrix}
    r_{11} & r_{12} & r_{13} \\
    r_{21} & r_{22} & r_{23} \\
    r_{31} & r_{32} & r_{33}
    \end{bmatrix}
    \]
  - \( 4 \) dof + \( 3 \) dof \( \Rightarrow 4 \) 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

Desired 2D Transformation

$$\begin{cases} x_i' = \frac{m_0 x_i + m_1 y_i + m_2}{m_6 x_i + m_7 y_i + 1} \\ y_i' = \frac{m_3 x_i + m_4 y_i + m_5}{m_6 x_i + m_7 y_i + 1} \end{cases}$$

Solve for $m_0, \ldots, m_7$
How to Determine Alignment?

- **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 \((\theta)\)
- 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:
  \[
  I'(x', y') = I(x, y) + (a - y \theta - x \theta^2 / 2) \partial I / \partial x + (b + x \theta - y \theta^2 / 2) \partial I / \partial y
  \]
Example: 2D Rigid Warp Mosaics

- Solve for $a$, $b$, $\theta$ that minimizes SSD error:

$$E = \sum_{x,y} (I'(x,y)-I(x,y))^2$$

$$= \sum_{x,y} (I(x,y) + (a-y\theta + x\theta^2/2)\partial I/\partial x + (b+x\theta - y\theta^2/2)\partial I/\partial y - I(x,y))^2$$

- 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, $(a, b, \theta)$, while warping $I'$ towards $I$ until $E < \text{threshold}$

Example: 2D Rigid Warp Mosaics

- Differentiating each term and setting equal to 0, we get

$$\sum I_x^2 a + \sum I_x I_y b + \sum AI_x \theta = I_x I_t$$

$$\sum I_x I_y a + \sum I_y^2 b + \sum AI_y \theta = I_y I_t$$

$$\sum AI_x a + \sum AI_y b + \sum A^2 \theta = AI_t$$

where $I_x = \partial I / \partial x$, $I_y = \partial I / \partial y$

$$I_t = \partial I / \partial t = I' - I$$

$$A = xI_y - yI_x$$
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 \( (a^{(t)} < \varepsilon_1 \text{ and } b^{(t)} < \varepsilon_2 \text{ and } \theta^{(t)} < \varepsilon_3) \)
   - then halt
   - else \( t = t + 1; \text{ goto step 2} \)

More generally, Székely paper

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

= \[ \sum_i \varepsilon_i^2 \]

by computing \( \frac{\partial E_i}{\partial m_0}, \ldots, \frac{\partial E_i}{\partial m_7} \)

Example:

\[ \frac{\partial E_i}{\partial m_0} = \frac{x_i}{m_0 x_i + m_7 y_i + 1} \cdot \frac{\partial I'}{\partial x'} \]

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

```
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
```

62
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)
    - LMS (Least Median of Squares)
  - \( \Theta \) is the set of model parameters, \( \rho \) is a “robust loss function” whose shape is parameterized by \( \sigma \), and \( r_i(x_i, \theta) \) is the residual error of the model \( \theta \) with the \( i \)th data element

Least Median of Squares (LMS)

- Defined as
  \[ \arg \min_{\theta} \text{median} r^2(x_i, \theta) \]
  - Up to \( \frac{1}{2} \) 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, \( \theta \), from the selected subset
    - Each candidate model, \( \theta \), 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. \( \text{level} = N \) // Set initial processing level to coarse level
  3. Solve for motion parameters at level \( \text{level} \)
  4. **If** \( \text{level} = 0 \) **then halt**
  5. Interpolate motion parameters at level \( \text{level} - 1 \)
  6. \( \text{level} = \text{level} - 1 \)
  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** \(-1 \leq i, j \leq 1\) **do**
      - compute cross-correlation at level \( l \):
        \[ CC_{i,j}(x, y) = LA_i(x, y) LB_j(x + i, y + j) \]
      - 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 projection

- Map 3D point \((X,Y,Z)\) onto cylinder
  \[
  (\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2 + Z^2}}(X, Y, Z)
  \]
- Convert to cylindrical coordinates
  \[
  (\sin \theta, h, \cos \theta) = (\hat{x}, \hat{y}, \hat{z})
  \]
- Convert to cylindrical image coordinates
  \[
  (\hat{x}, \hat{y}) = (f \theta, fh) + (\bar{x}_c, \bar{y}_c)
  \]

Cylindrical reprojection

- How to map from a cylinder to a planar image?

- Apply camera projection matrix
  - for project 2, account for focal length and assume principle point is at center of image
  - \(\hat{x}' = \frac{1}{2}\) image width, \(\hat{y}' = \frac{1}{2}\) image height
  - \[
  \begin{bmatrix}
  wx' \\
  wy' \\
  w
  \end{bmatrix} = \begin{bmatrix}
  -f & 0 & w/2 & 0 \\
  0 & -f & h/2 & 0 \\
  0 & 0 & 1 & 0
  \end{bmatrix} \begin{bmatrix}
  \hat{x} \\
  \hat{y} \\
  \hat{z} \\
  1
  \end{bmatrix}
  \]
- Convert to image coordinates
  - divide by third coordinate (w)
Cylindrical reprojection

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

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 $w_i$ 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: $w(x'_i, y'_i) = w_i(x'_i)w_j(y'_i)$
Image Blending

Feathering

Encoding transparency

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

\[ I_{\text{blend}} = I_{\text{left}} + I_{\text{right}} \]

(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)
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"
   \[ S = (X_1, Y_1, \ldots, X_n, Y_n) \]

2. Compute "texture vector"
   Difference map between corresponding pixels' image intensities
   \[ t = (t_1, \ldots, t_n) \]

3. Linear Shape Model of Faces
   Given a database of n 3D models of faces, any new face can be described by
   \[ S = \sum_{i=1}^{n} \beta_i S_i \]
   where \( S = (X_1, Y_1, \beta_1, \ldots, X_n, Y_n, \beta_n) \)
   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 not change the \( \beta_i \)’s!
   \[ S_p = \sum_{i=1}^{n} \beta_i S_i^p \]
   where \( S_p = PS_p \), \( P = \) perspective projection
4. Compute linear shape approximation
   \[ s^f = \sum \beta_i s^i \]
   where \( s^f \) = input image face
   \( s^i \) = \( i \)th reference face image at same pose

5. Compute new view's shape vector
   \[ s^f = \sum \beta_i s^i \]

6. Compute linear texture approximation
   \[ t^f = \sum \alpha_i t^i \]

7. Compute new view's texture vector
   \[ t^f = \sum \alpha_i t^i \]
   Note: This texture mapping is not correct except for Lambertian surfaces.
   Alternately, use 3D head model to remap texture

8. Warp texture onto shape
   \[ s^f + t^f \]
For a given scene, describe ALL rays through ALL pixels of ALL cameras, at ALL wavelengths, and ALL time.

\[ F(x,y,z, \phi, \theta, \lambda, t) \]

“eyeballs everywhere” function \((5D \times 2D)\)

Plenoptic Array: ‘The Matrix Effect’

- Brute force! Simple arc, line, or ring array of cameras
- Synchronized shutter [http://www.ruffy.com/firingline.html](http://www.ruffy.com/firingline.html)
- 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

- Levoy and Hanrahan, *Proc. SIGGRAPH ’96*
- Aka **Lumigraph**
- Equivalent to the Plenoptic function in that this is a representation of the radiance at a point in a given direction, except Light Field is 4D instead of 5D
- Assume: Radiance along a ray is constant (e.g., air is transparent)
- Only represent light radiating from a single object/scene from its convex hull (i.e., no occlusion or inter-reflections)
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-pane window
  - 2 coordinates \((u,v)\) where ray intersects first pane
  - 2 coordinates \((s,t)\) where ray intersects second pane

Light Field Representation

- Construct enclosing cube around scene
- For each face, represent all rays emanating from that face
- \((u,v)\) represents position of camera optical center on face, and \((s,t)\) represents orientation
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

Quadrilinear Interpolation

- Interpolate over hypercube of nearest locations
  - \(u, v, s, t\)
  - \(u, v, s, t + 1\)
  - \(u, v, s + 1, t\)
  - \(u, v, s + 1, t + 1\)
  - ...
Results

- Light Field

<table>
<thead>
<tr>
<th></th>
<th>buddha</th>
<th>kidney</th>
<th>hallway</th>
<th>lion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slabs</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Images per slab</td>
<td>16x16</td>
<td>64x64</td>
<td>64x32</td>
<td>32x16</td>
</tr>
<tr>
<td>Total images</td>
<td>256</td>
<td>4096</td>
<td>8192</td>
<td>2048</td>
</tr>
<tr>
<td>Pixels per image</td>
<td>$256^2$</td>
<td>$128^2$</td>
<td>$256^2$</td>
<td>$256^2$</td>
</tr>
<tr>
<td>Raw size (MB)</td>
<td>50</td>
<td>201</td>
<td>1608</td>
<td>402</td>
</tr>
<tr>
<td>Prefiltering</td>
<td>uvst</td>
<td>st only</td>
<td>uvst</td>
<td>st only</td>
</tr>
</tbody>
</table>

*Table 1*: Statistics of the light fields shown in figure 14.