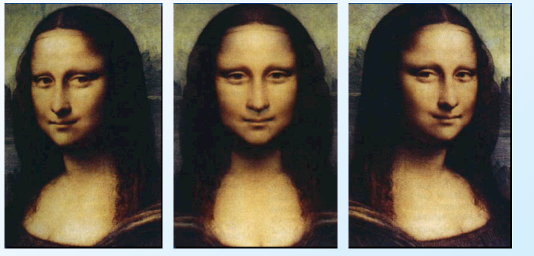
Interactive Computer Graphics

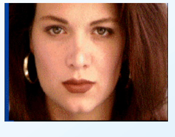
- Lecture 15+16: Warping and Morphing

Warping and morphing

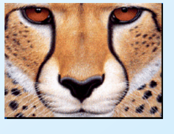


Warping and Morphing

- What is
  - warping ?
  - morphing ?



?



Warping

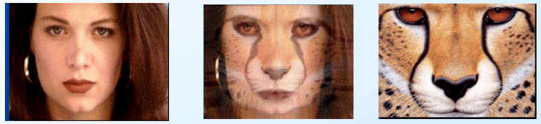
- The term warping refers to the geometric transformation of graphical objects (images, surfaces or volumes) from one coordinate system to another coordinate system.
- Warping does not affect the attributes of the underlying graphical objects.
- Attributes may be
  - color (RGB, HSV)
  - texture maps and coordinates
  - normals, etc.

Morphing

- The term morphing stands for metamorphosing and refers to an animation technique in which one graphical object is gradually turned into another.
- Morphing can affect both the shape and attributes of the graphical objects.

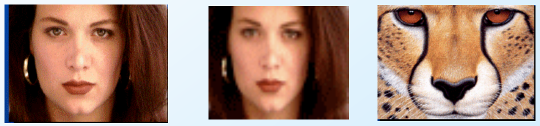
Warping and Morphing

- What is
  - warping ?
  - morphing ?



Warping and Morphing

- What is
  - warping ?
  - morphing ?

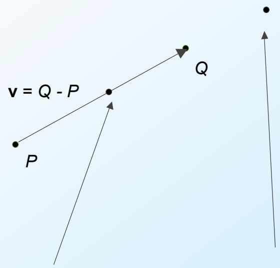


Morphing = Object Averaging

- The aim is to find “an average” between two objects
  - Not an average of two <u>images of objects</u>…
  - …but an image of the <u>average object</u>!
  - How can we make a smooth transition in time?
    - Do a “weighted average” over time t
- How do we know what the average object looks like?
  - Need an algorithm to compute the average geometry and appearance

Averaging

What's the average of P and Q?



Linear Interpolation (Affine Combination):
New point $aP + bQ$,
defined only when $a+b = 1$
So $aP+bQ = aP+(1-a)Q$

$P + 0.5v$
$= P + 0.5(Q - P)$
$= 0.5P + 0.5\ Q$

$P + 1.5v$
$= P + 1.5(Q - P)$
$= -0.5P + 1.5\ Q$
(extrapolation)

Morphing using cross-dissolve

- Interpolate whole images:

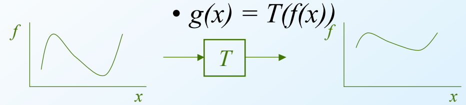
  $\text{Image}_{halfway} = t*\text{Image}_1 + (1\text{-}t)*\text{image}_2$
- This is called **cross-dissolve**
- But what is the images are not aligned?

Morphing using warping and cross-dissolve

- Align first, then cross-dissolve

Image warping

- image filtering: change ***range*** of image
  - $g(x) = T(f(x))$



- image warping: change ***domain*** of image
  - $g(x) = f(T(x))$

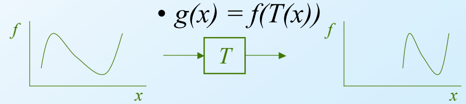
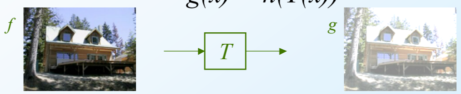
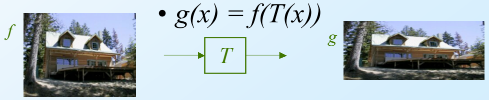


Image warping

- image filtering: change ***range*** of image
  - *g(x) = h(T(x))*

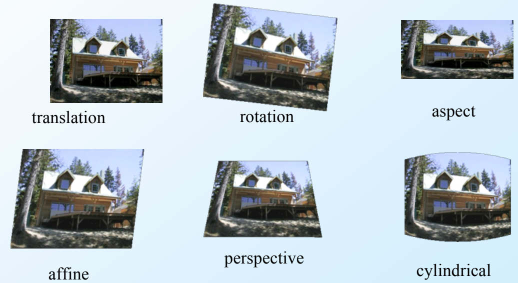


- image warping: change ***domain*** of image
  - *g(x) = f(T(x))*

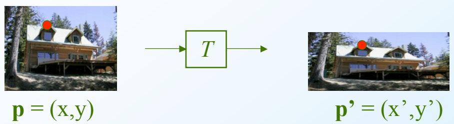


Parametric (global) warping

- Examples of parametric warps:



Parametric (global) warping



- Transformation T can be expressed as a mapping:

$$p' = T(p)$$

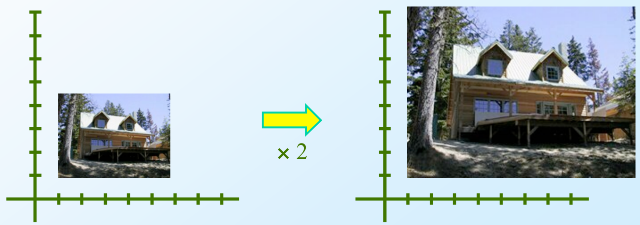
- Transformation T can be expressed as a matrix:

$$p' = \mathbf{M}*p$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

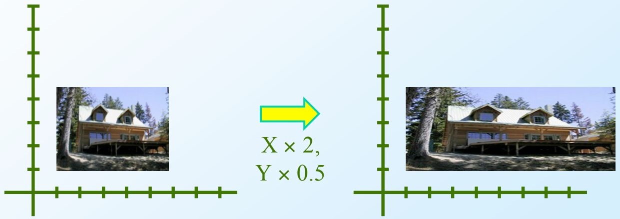
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Scaling

- Scaling operation:

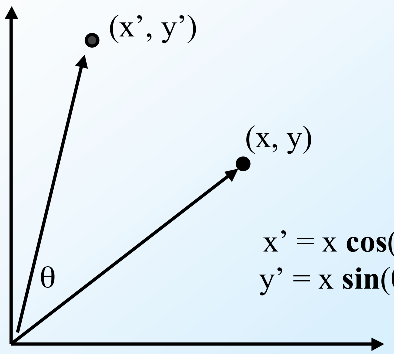
$$x' = ax$$
$$y' = by$$

- Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\textit{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

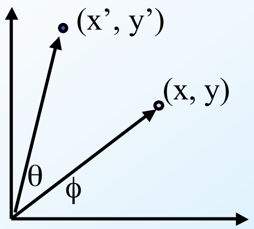
What is the inverse of S?

2-D Rotation



x’ = x **cos**(θ) - y **sin**(θ)
y’ = x **sin**(θ) + y **cos**(θ)

2-D Rotation



x = r cos (ϕ)
y = r sin (ϕ)
x’ = r cos (ϕ + θ)
y’ = r sin (ϕ + θ)

Trig Identity…
x’ = r cos(ϕ) cos(θ) – r sin(ϕ) sin(θ)
y’ = r sin(ϕ) sin(θ) + r cos(ϕ) cos(θ)

Substitute…
x’ = x **cos**(θ) - y **sin**(θ)
y’ = x **sin**(θ) + y **cos**(θ)

2-D Rotation

- This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Even though sin(θ) and cos(θ) are nonlinear functions of θ,
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y
- What is the inverse transformation?
  - Rotation by –θ
  - For rotation matrices, det(R) = 1 so $\mathbf{R}^{-1} = \mathbf{R}^T$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Identity?

$$x' = x \qquad y' = y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Scale around (0,0)?

$$\boldsymbol{x' = s_x * x} \qquad \boldsymbol{y' = s_y * y} \qquad \begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s_x} & 0 \\ 0 & \boldsymbol{s_y} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y \qquad y' = \sin\Theta * x + \cos\Theta * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Shear?

$$\boldsymbol{x' = x + sh_x * y} \qquad \boldsymbol{y' = sh_y * x + y} \qquad \begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh_x} \\ \boldsymbol{sh_y} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Mirror about Y axis?

$$x' = -x \qquad y' = y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror over (0,0)?

$$x' = -x \qquad y' = -y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$x' = x + t_x$$
$$y' = y + t_y$$

NO!

Only linear 2D transformations can be represented with a 2x2 matrix

All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous Coordinates

- Q: How can we represent translation as a matrix transformation?

$$x' = x + t_x$$
$$y' = y + t_y$$

- A: Using the translation parameters as the rightmost column:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

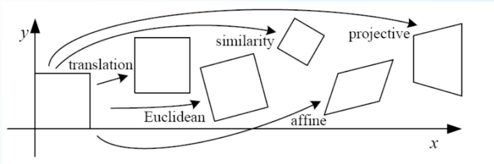
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

2D image transformations



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\ I \mid t\ \right]_{2\times3}$			
rigid (Euclidean)	$\left[\ R \mid t\ \right]_{2\times3}$			
similarity	$\left[\ sR \mid t\ \right]_{2\times3}$			
affine	$\left[\ A\ \right]_{2\times3}$			
projective	$\left[\ \tilde{H}\ \right]_{3\times3}$			

Transformations

- Dimensions of transformation
  - 1D: curves
  - 2D: images
  - 3D: volumes
- Types of transformations
  - rigid
  - affine
  - polynomial
    - quadratic
    - cubic
  - splines

Transformations in 3D: Rigid

- Rigid transformation (6 degrees of freedom)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} r_{01} & r_{02} & r_{03} & t_x \\ r_{11} & r_{12} & r_{13} & t_y \\ r_{21} & r_{22} & r_{23} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T^x_{rigid} \cdot T^y_{rigid} \cdot T^z_{rigid} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \\ 0 \end{pmatrix}$$

- t_x, t_y, t_z describe the 3 translations in x, y and z
- r_{11}, ..., r_{33} describe the 3 rotations around x, y, z

Transformations in 3D: Rigid

$$\mathbf{T}^x_{rigid} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}^y_{rigid} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T}^z_{rigid} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformations in 3D: Affine

- Affine transformations (12 degrees of freedom)

$$\mathbf{T}_{scale} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}_{shear}^{xy} = \begin{pmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T}(x, y, z) = \mathbf{T}_{shear} \cdot \mathbf{T}_{scale} \cdot \mathbf{T}_{rigid} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Non-rigid transformations

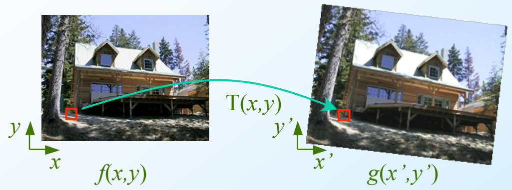
- Quadratic transformation (30 degrees of freedom)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} r_{00} & \cdots & r_{08} & r_{09} \\ r_{10} & \cdots & r_{18} & r_{19} \\ r_{20} & \cdots & r_{28} & r_{29} \\ 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x^2 \\ y^2 \\ \vdots \\ 1 \end{pmatrix}$$

Non-rigid transformations

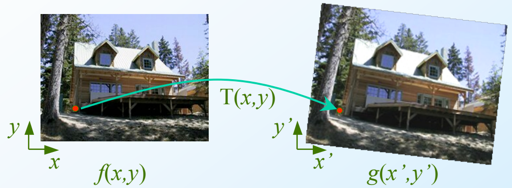
- Can be extended to other higher-order polynomials:
  - 3^{rd} order (60 DOF)
  - 4^{th} order (105 DOF)
  - 5^{th} order (168 DOF)
- Problems:
  - can model only global shape changes, not local shape changes
  - higher order polynomials introduce artifacts such as oscillations

Image warping



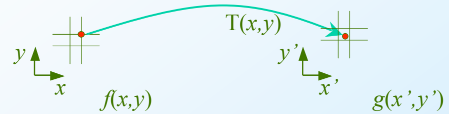
- Given a coordinate transform *(x',y')* = T(*x,y*) and a source image *f*(*x,y*), how do we compute a transformed image *g(x',y')* = *f*(T(*x,y*))?

Forward warping



- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = \mathrm{T}(x,y)$ in the second image

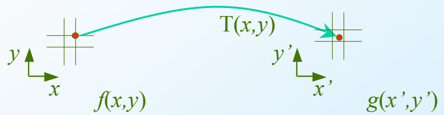
Forward warping



- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = \mathrm{T}(x,y)$ in the second image

  Q: what if pixel lands "between" two pixels?

Forward warping

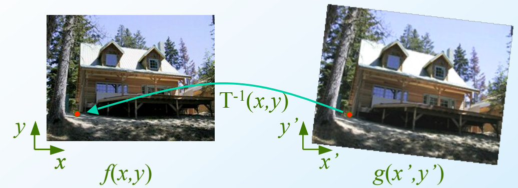


- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = \mathrm{T}(x,y)$ in the second image

  Q: what if pixel lands "between" two pixels?

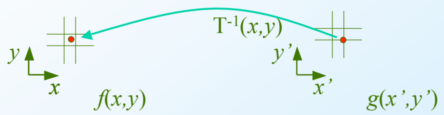
  A: distribute color among neighboring pixels (x',y')
  - known as "splatting"

Inverse warping



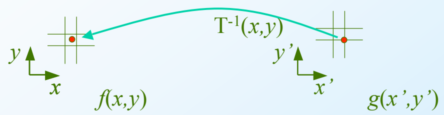
- Get each pixel $g(x',y')$ from its corresponding location $(x,y) = \mathrm{T}^{-1}(x',y')$ in the first image

Inverse warping



- Get each pixel $g(x',y')$ from its corresponding location $(x,y) = T^{-1}(x',y')$ in the first image
  Q: what if pixel comes from “between” two pixels?

Inverse warping



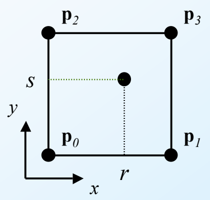
- Get each pixel $g(x',y')$ from its corresponding location $(x,y) = T^{-1}(x',y')$ in the first image
  Q: what if pixel comes from “between” two pixels?
  A: Interpolate color value from neighbors
  – nearest neighbor, bilinear, Gaussian, bicubic

Interpolation

Interpolation

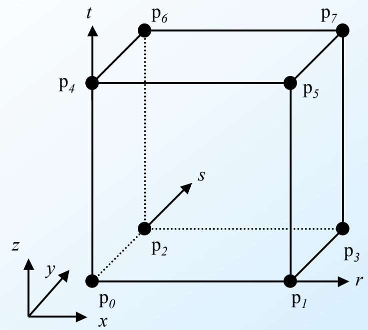
Interpolation: Linear, 2D

$$f(p) = \sum_{i=0}^{n-1} w_i f(p_i)$$



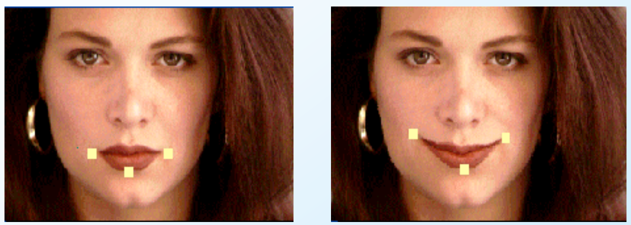
$$w_0 = (1-r)(1-s)$$
$$w_1 = r(1-s)$$
$$w_2 = (1-r)s$$
$$w_3 = rs$$

Interpolation: Linear, 3D

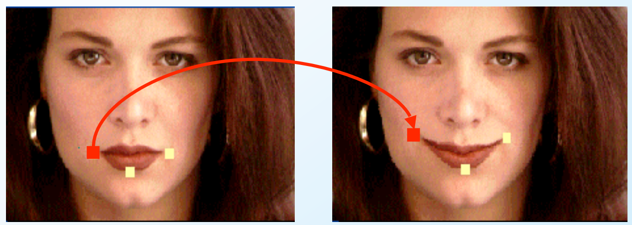


$$w_0 = (1-r)(1-s)(1-t)$$
$$w_1 = r(1-s)(1-t)$$
$$w_2 = (1-r)s(1-t)$$
$$w_3 = rs(1-t)$$
$$w_4 = (1-r)(1-s)t$$
$$w_5 = r(1-s)t$$
$$w_6 = (1-r)st$$
$$w_7 = rst$$

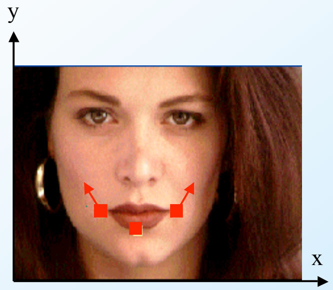
Non-rigid transformations



Non-rigid transformations: Correspondences

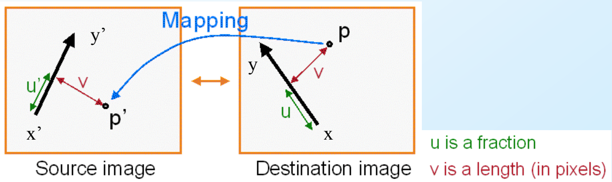


Non-rigid transformations: Correspondences



Feature-Based Warping: Beier-Neeley

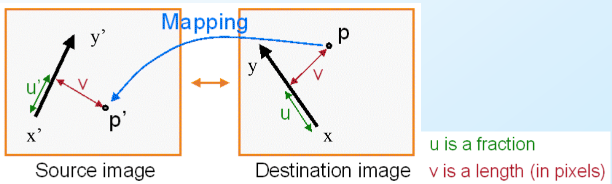
- Beier & Neeley use pairs of lines to specify warp
  - Given **p** in destination image, where is **p'** in source image?



Feature-Based Warping: Beier-Neeley

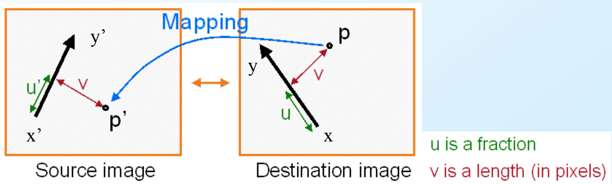
$$u = \frac{(p-x)\cdot(y-x)}{\|y-x\|^2} \qquad v = \frac{(p-x)\cdot Perpendicular(y-x)}{\|y-x\|}$$

$$p' = x + u\cdot(y'-x') + \frac{v\cdot Perpendicular(y'-x')}{\|y'-x'\|}$$



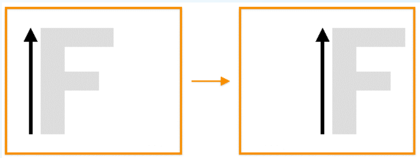
Feature-Based Warping: Beier-Neeley

- For each pixel p in the destination image
  - find the corresponding u,v
  - find the p' in the source image for that u,v
  - destination(p) = source(p')



Warping with One Line Pair: Beier-Neeley

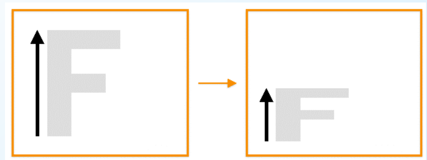
- What happens to the "F" ?



Translation !

Warping with One Line Pair (cont.): Beier-Neeley

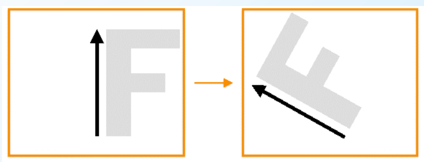
- What happens to the "F" ?



Scale !

Warping with One Line Pair (cont.): Beier-Neeley

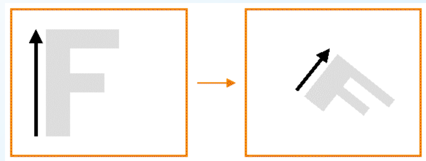
- What happens to the "F" ?



Rotation !

Warping with One Line Pair (cont.): Beier-Neeley

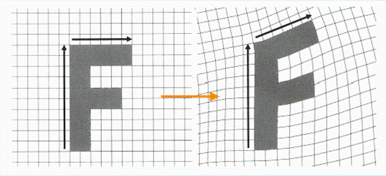
- What happens to the "F" ?



In general, similarity transformations

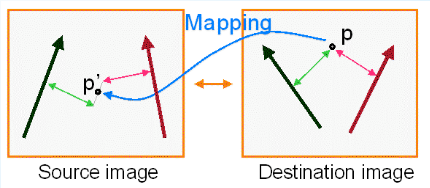
Warping with Multiple Line Pairs: Beier-Neeley

- Use weighted combination of points defined each pair of corresponding lines



Warping with Multiple Line Pairs: Beier-Neeley

- Use weighted combination of points defined by each pair corresponding lines



p' is a weighted average

Weighting Effect of Each Line Pair: Beier-Neeley

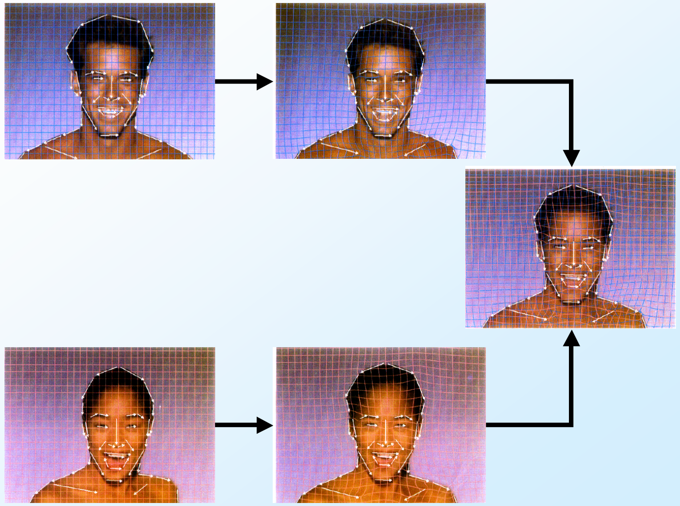
- To weight the contribution of each line pair

$$weight[i] = \left(\frac{length[i]^p}{a + dist[i]} \right)^b$$

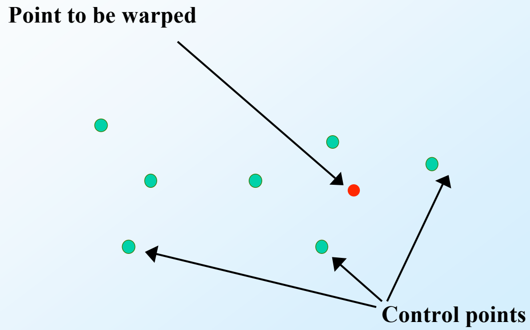
  - where
    - length[i] is the length of L[i]
    - dist[i] is the distance from X to L[i]
    - a, b, p are constants that control the warp

Warping Pseudocode: Beier-Neeley

```
foreach destination pixel p do
    psum = (0, 0)
    wsum = (0, 0)
    foreach line L[i] in destination do
        p'[i] = p transformed by (L[i], L'[i])
        psum = psum + p'[i] * weight[i]
        wsum += weight[i]
    end
    p' = psum / wsum
    destination(p) = source(p')
end
```

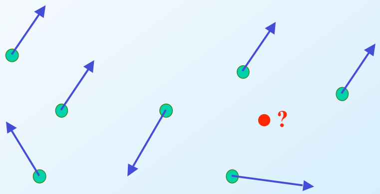


Non-rigid transformation



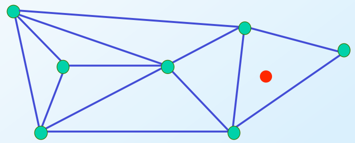
Non-rigid transformation

- For each control point we have a displacement vector
- How do we interpolate the displacement at a pixel?



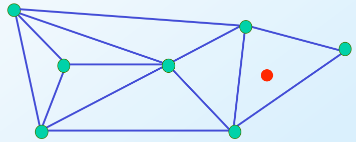
Non-rigid transformation: Piecewise affine

- Partition the convex hull of the control points into a set of triangles



Non-rigid transformation: Piecewise affine

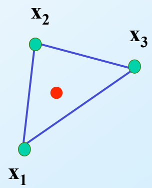
- Partition the convex hull of the control points into a set of triangles



Non-rigid transformation: Piecewise affine

- Find triangle which contains point **p** and express in terms of the vertices of the triangle:

$$\mathbf{p} = \mathbf{x}_1 + \alpha(\mathbf{x}_2 - \mathbf{x}_1) + \beta(\mathbf{x}_3 - \mathbf{x}_1)$$



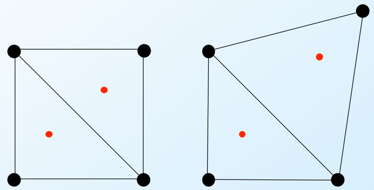
Non-rigid transformation: Piecewise affine

- Or $\mathbf{p} = \gamma\mathbf{x}_1 + \alpha\mathbf{x}_2 + \beta\mathbf{x}_3$ with $\gamma = 1 - (\alpha + \beta)$

- Under the affine transformation this point simply maps to

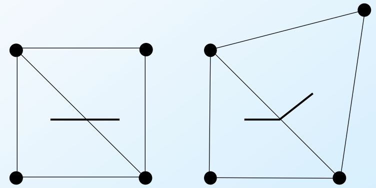
$$\mathbf{p}' = \gamma\mathbf{x}_1' + \alpha\mathbf{x}_2' + \beta\mathbf{x}_3'$$

Non-rigid transformation: Piecewise affine



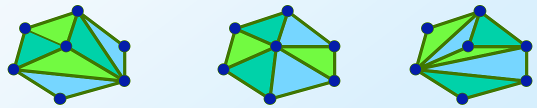
Non-rigid transformation: Piecewise affine

- Problem: Produces continuous deformations, but the deformation may not be smooth. Straight lines can be kinked across boundaries between triangles



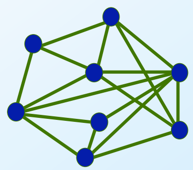
Triangulations

- A *triangulation* of set of points in the plane is a *partition* of the convex hull to triangles whose vertices are the points, and do not contain other points.
- There are an exponential number of triangulations of a point set.



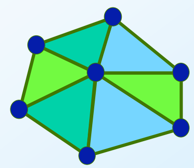
An O(n^3) Triangulation Algorithm

- Repeat until impossible:
  - Select two sites.
  - If the edge connecting them does not intersect previous edges, keep it.

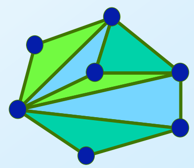


"Quality" Triangulations

- Let $\alpha(T) = (\alpha_1, \alpha_2, .., \alpha_{3t})$ be the vector of angles in the triangulation T in increasing order.
- A triangulation T_1 will be "better" than T_2 if $\alpha(T_1) > \alpha(T_2)$ lexicographically.
- The Delaunay triangulation is the "best"
  - Maximizes smallest angles

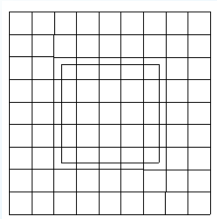


good

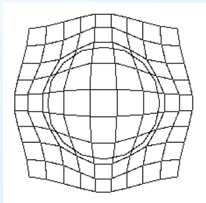


bad

Representing deformations



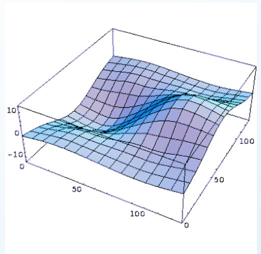
Before deformation



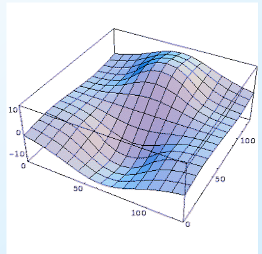
After deformation

Warping and Morphing: Slide 73

Representing deformations



Displacement in the horizontal direction



Displacement in the vertical direction

Warping and Morphing: Slide 74

B-splines

- Free-Form Deformation (FFD) are a common technique in Computer Graphics for modelling 3D deformable objects
- FFDs are defined by a mesh of control points with uniform spacing
- FFDs deform an underlying object by manipulating a mesh of control points
  - control point can be displaced from their original location
  - control points provide a parameterization of the transformation

Warping and Morphing: Slide 75

FFDs using linear B-splines

- FFDs based on linear B-splines can be expressed as a 2D (3D) tensor product of linear 1D B-splines:

$$\mathbf{u}(x,y) = \sum_{l=0}^{1}\sum_{m=0}^{1} B_l(u) B_m(v) \phi_{i+l,j+m}$$

where

$$i = \left\lfloor \frac{x}{\delta_x} \right\rfloor,\ j = \left\lfloor \frac{y}{\delta_y} \right\rfloor,\ u = \frac{x}{\delta_x} - \left\lfloor \frac{x}{\delta_x} \right\rfloor,\ v = \frac{y}{\delta_y} - \left\lfloor \frac{y}{\delta_y} \right\rfloor$$

and B_i corresponds to the B-spline basis functions

$$B_0(s) = 1 - s$$

$$B_1(s) = s$$

Warping and Morphing: Slide 76

FFDs using cubic B-splines

- FFDs based on cubic B-splines can be expressed as a 2D (3D) tensor product of cubic 1D B-splines:

$$\mathbf{u}(x,y)=\sum_{l=0}^{3}\sum_{m=0}^{3}B_l(u)B_m(v)\phi_{i+l,j+m}$$

where

$$i=\left\lfloor\frac{x}{\delta_x}\right\rfloor-1,\quad j=\left\lfloor\frac{y}{\delta_y}\right\rfloor-1,\quad u=\frac{x}{\delta_x}-\left\lfloor\frac{x}{\delta_x}\right\rfloor,\quad v=\frac{y}{\delta_y}-\left\lfloor\frac{y}{\delta_y}\right\rfloor$$

and B_i corresponds to the B-spline basis functions

$$B_0(s)=(1-s)^3/6 \qquad B_2(s)=(-3s^3+3s^2+3s+1)/6$$

$$B_1(s)=(3s^3-6s^2+4)/6 \qquad B_3(s)=s^3/6$$

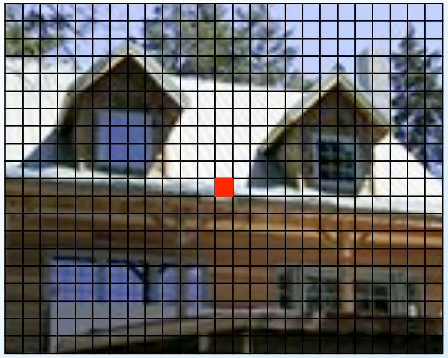
FFDs: Example

- Image
  - width 25 pixels
  - height 20 pixels
- Free-form deformation
  - 6 x 6 mesh of control points
  - linear B-splines
- Calculate new position of pixel
  - x = 12
  - y = 11

FFDs: 2D Example

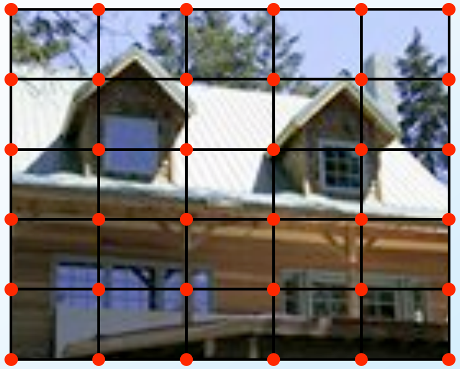
FFDs: 2D Example

0,0



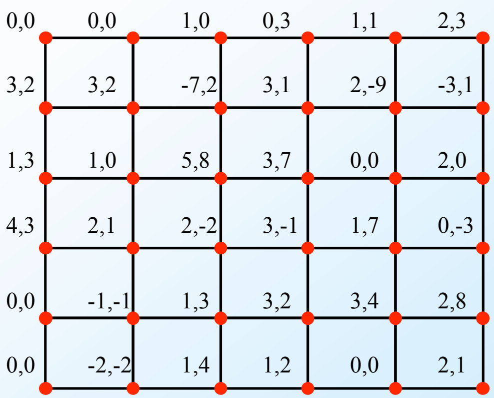
25,20

FFDs: 2D Example

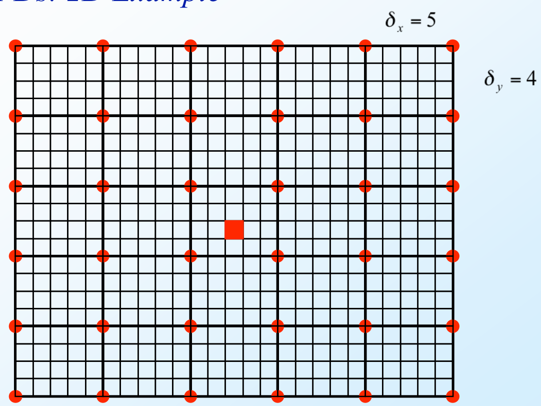


FFDs: 2D Example

FFDs: 2D Example



FFDs: 2D Example



FFDs: 2D Example

- Calculate integer lattice coordinates *i, j*:

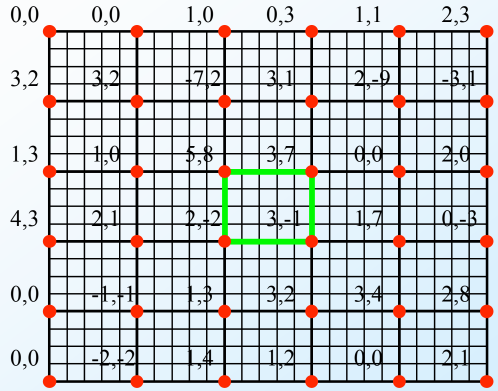
$$i = \left\lfloor \frac{12}{5} \right\rfloor = 2 \qquad j = \left\lfloor \frac{11}{4} \right\rfloor = 2$$

- Calculate fractional lattice coordinates *u,v*:

$$u = \frac{12}{5} - \left\lfloor \frac{12}{5} \right\rfloor = 0.4 \qquad v = \frac{11}{4} - \left\lfloor \frac{11}{4} \right\rfloor = 0.75$$

$$\mathbf{u}(x,y) = \sum_{l=0}^{1} \sum_{m=0}^{1} B_l(u) B_m(v) \phi_{2+l,2+m}$$

FFDs: 2D Example

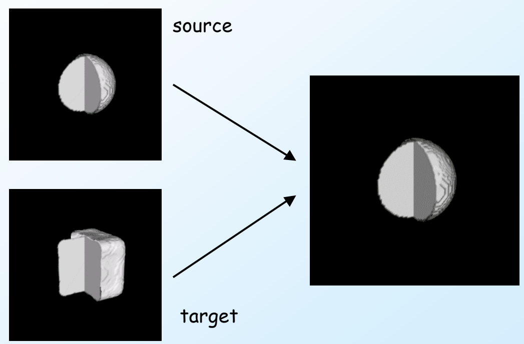


FFDs: 2D Example

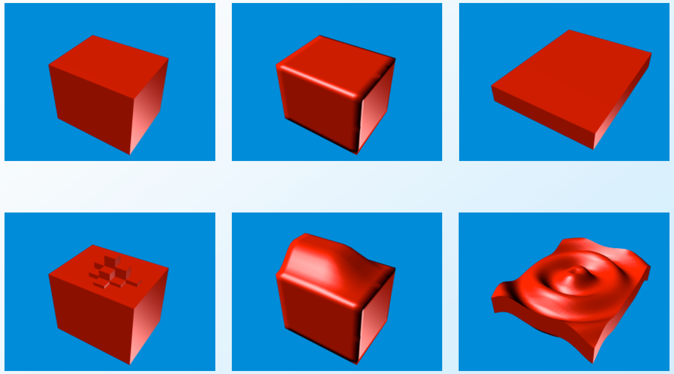
$$\begin{aligned} \mathbf{u}(x,y) &= B_0(0.4)B_0(0.75)\phi_{2,2} + \\ &+ B_0(0.4)B_1(0.75)\phi_{2,3} + \\ &+ B_1(0.4)B_0(0.75)\phi_{3,2} + \\ &+ B_1(0.4)B_1(0.75)\phi_{3,3} \end{aligned}$$

$$\begin{aligned} \mathbf{u}(x,y) &= 0.15 \cdot \begin{pmatrix} 5 \\ 8 \end{pmatrix} + 0.45 \cdot \begin{pmatrix} 2 \\ -2 \end{pmatrix} + \\ &+ 0.10 \cdot \begin{pmatrix} 3 \\ 7 \end{pmatrix} + 0.3 \cdot \begin{pmatrix} 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 2.34 \\ 0.7 \end{pmatrix} \end{aligned}$$

FFDs in 3D



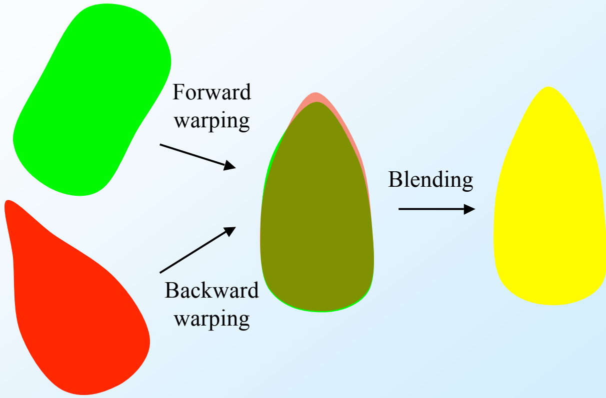
FFDs in 3D



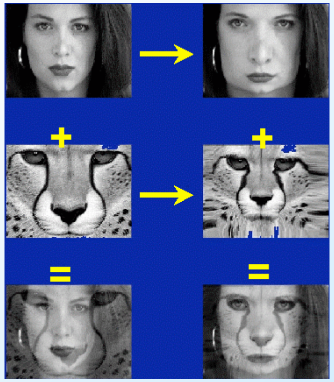
FFDs

- Used for warping:
  - Lee et al. (1997)
- Advantages:
  - Control points have local influence since the basis function has finite support
  - Fast
    - linear (in 3D: 2 x 2 x 2 = 8 operations per warp)
    - cubic (in 3D: 4 x 4 x 4 = 64 operations per warp)
- Disadvantages:
  - Control points must have uniform spatial distribution

Morphing = (warping)^2 + blending



Morphing = (warping)^2 + blending



Morphing

GenerateAnimation($Image_0$, $Image_1$)
begin
  foreach intermediate frame time t do
    $Warp_0$ = WarpImage($Image_0$, t)
    $Warp_1$ = WarpImage($Image_1$, t)
    foreach pixel p in FinalImage do
      Result(p) = $(1-t)Warp_0 + tWarp_1$
    end
  end
end

Image Combination

- Determines how to combine attributes associated with geometrical primitives. Attributes may include
  - color
  - texture coordinates
  - normals
- Blending
  - cross-dissolve
  - adaptive cross-dissolve
  - alpha-channel blending
  - z-buffer blending

Image Combination: Cross-dissolve

- Blending with cross-dissolve:

$$I = (1-t) \cdot I_A + t \cdot I_B$$

  - intensities
  - RGB space
  - HSV space
  - texture space

Image Combination: Cross-dissolve

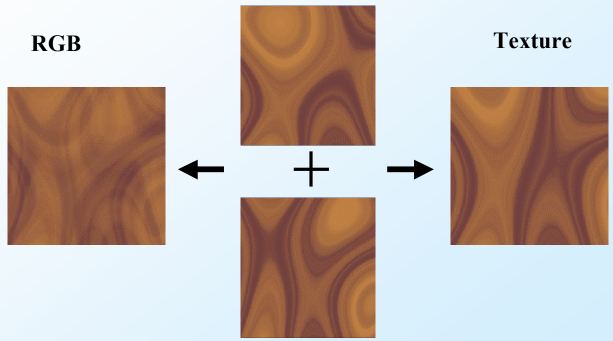


Image Combination: Adaptive cross-dissolve

- Adaptive cross-dissolve

$$I = (1 - w(\mathbf{p},\lambda)) \cdot I_A(\mathbf{p}) + w(\mathbf{p},\lambda) \cdot I_B(\mathbf{p})$$

  - similar to cross-dissolve but blending function depends on position in image

Image Combination: Alpha channel blending

- Blending using RGBA images

$$I = w_a \cdot I_A + w_b \cdot I_B$$

- Images are represented by quadruples:
  - R, G, B indicating color
  - Alpha channel encodes pixel coverage information
    - $\alpha = 0$ transparent
    - $0 < \alpha < 1$ semi-transparent
    - $\alpha = 1$ opaque

Image Combination: Alpha channel blending

- Convention:
  - RGBA represents a pixel with color C:

$$C = \left(R/\alpha, G/\alpha, B/\alpha \right)$$

  - What is meaning of:

(0, 1, 0, 1)	Full green, full opacity
(0, 1, 0, 0.5)	Full green, semi-transparent
(0, 0.5, 0, 1)	Half green, full opacity
(1, 1, 1, 0)	White, transparent

Image Combination: Alpha channel blending

Operation	w_a	w_b
A	1	0
B	0	1
A over B	1	$1-\alpha_a$
A in B	α_b	$0-\alpha_a$
A out B	$1-\alpha_b$	0
A plus A	1	1

Image Combination: Z-buffer blending

- Blending using Z-buffer values:

$$I = \begin{cases} I_a & \text{if } z_a < z_b \\ I_b & \text{else} \end{cases}$$

  - defines an ordering
  - can be used for layering