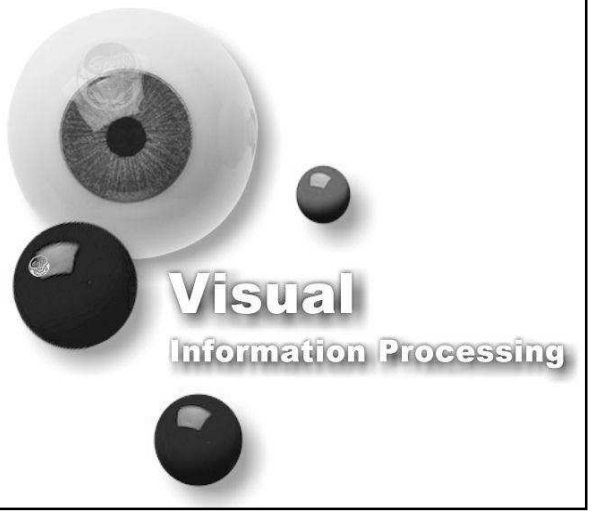


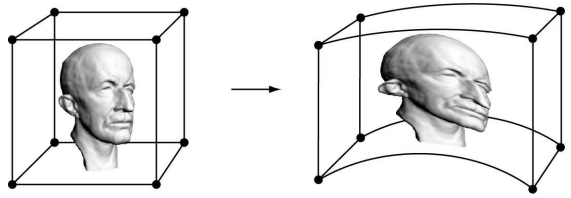
Interactive Computer Graphics

- Lecture 16: Warping and Morphing (cont.)

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

Non-rigid transformations: FFDs



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_l corresponds to the B-spline basis functions

$$B_0(s) = 1 - s$$
$$B_1(s) = s$$

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

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

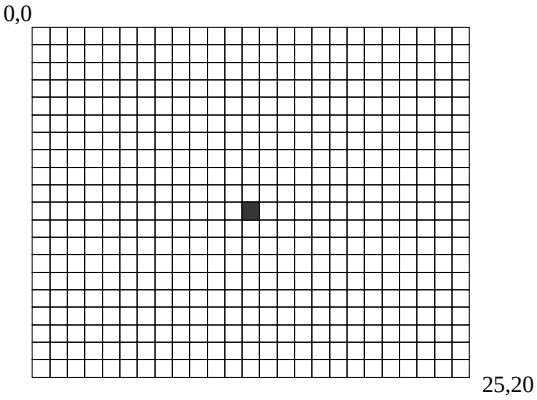
and B_l corresponds to the B-spline basis functions

$$B_0(s) = (1-s)^3/6 \qquad B_2(s) = (-3s^3 + 3s^2 + 3s + 1)/6$$
$$B_1(s) = (3s^3 - 6s^2 + 4)/6 \qquad B_3(s) = s^3/6$$

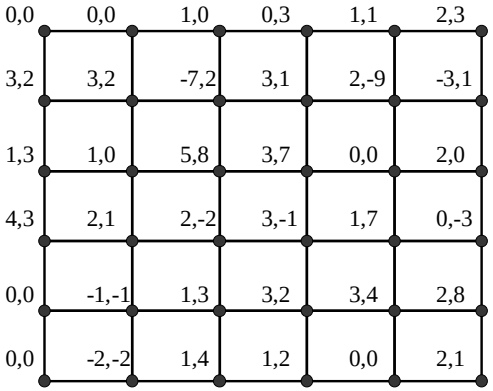
FFDs: Example

- 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

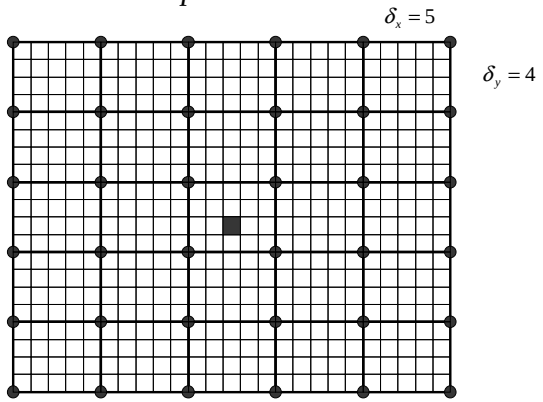
FFDs: 2D Example



FFDs: 2D Example



FFDs: 2D Example



FFDs: 2D Example

- Calculate integer lattice coordinates *i*, *j*:

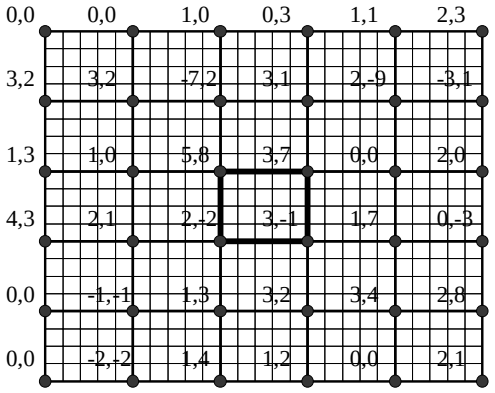
$$i = \left\lfloor \frac{12}{5} \right\rfloor = 2 \qquad j = \left\lfloor \frac{11}{4} \right\rfloor = 2$$

- 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

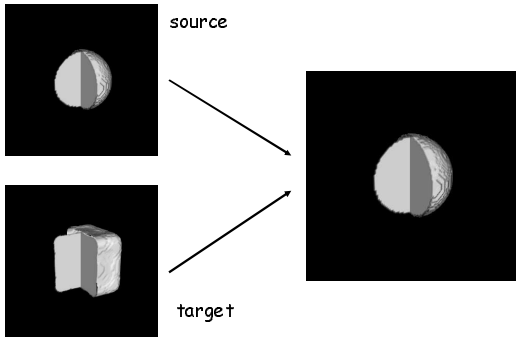


FFDs: 2D Example

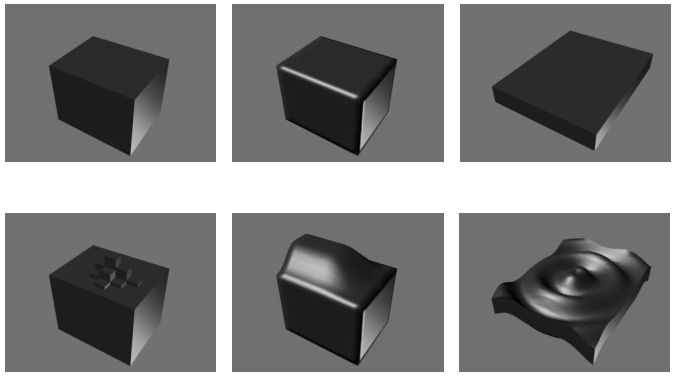
$$\begin{aligned}\mathbf{u}(x,y) &= B_0(0.4)B_0(0.75)\phi_{2,2} + \\ &+ B_0(0.4)B_1(0.75)\phi_{2,3} + \\ &+ B_1(0.4)B_0(0.75)\phi_{3,2} + \\ &+ B_1(0.4)B_1(0.75)\phi_{3,3}\end{aligned}$$

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

FFDs in 3D



FFDs in 3D

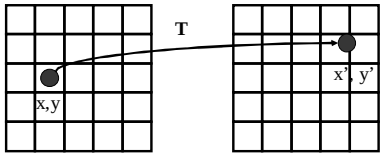


FFDs

- 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

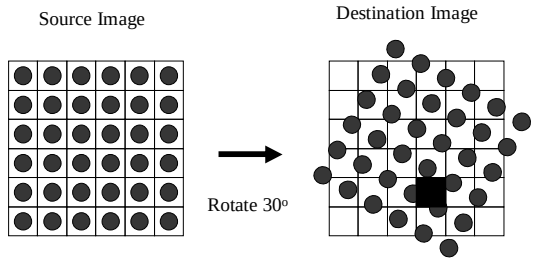
Forward mapping

```
for (int x = 0; x < X; x++) {
  for (int y = 0; y < Y; y++) {
      float x' = T_x(x, y);
      float y' = T_y(x, y);
      dst(x', y') = src(x, y);
  }
}
```



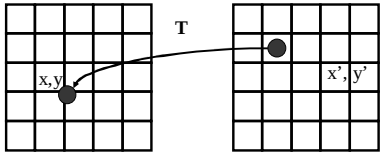
Forward mapping

- Iterate over source image
  - Some destination pixels may not be covered
  - Many source pixels can map to the same destination pixel



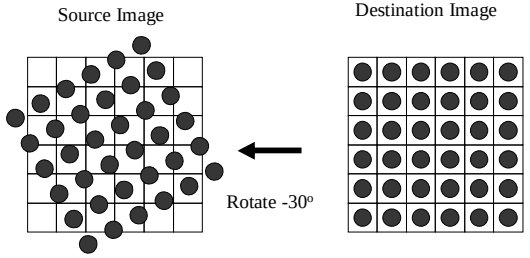
Backward mapping

```
for (int x' = 0; x' < X'; x'++) {
  for (int y' = 0; y' < V; y'++) {
      float x = T_x(x', y');
      float y = T_y(x', y');
      dst(x', y') = src(x, y);
  }
}
```



Backward mapping

- Iterate over destination image
  - All destination pixels may not be covered
  - Need to interpolate source pixels



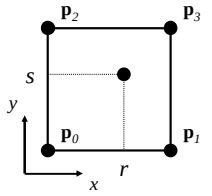
Forward and backward mapping (or warping)

- Forward mapping determines the mapped position of the input in the output.
  - – Forward mapping techniques are useful for warping points, lines or surfaces but not for images (since they tend to create holes).
- Backward mapping determines the mapped position of the output in the input.
  - – Backward mapping techniques are particularly useful for warping images since they avoid holes in the output image (which are created by forward mapping).

Interpolation

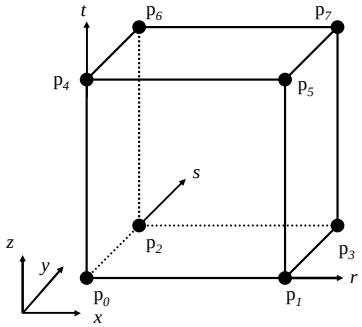
Interpolation: Linear, 2D

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



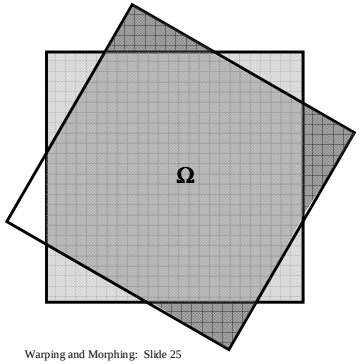
$$w_0 = (1-r)(1-s)$$
$$w_1 = r(1-s)$$
$$w_2 = (1-r)s$$
$$w_3 = rs$$

Interpolation: Linear, 3D



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

Image Mapping

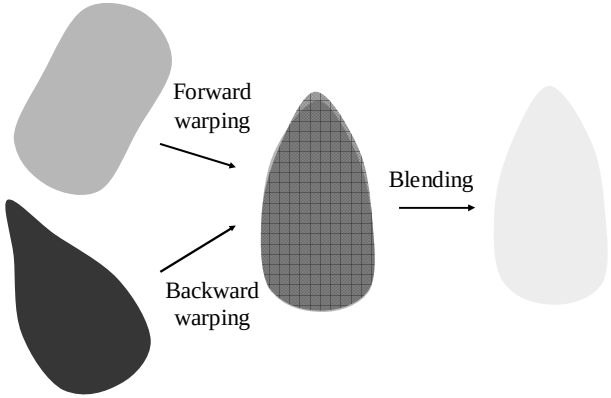


Source

Destination

Overlap between source and destination

Morphing = (warping)^2 + blending



Morphing = (warping)^2 + blending

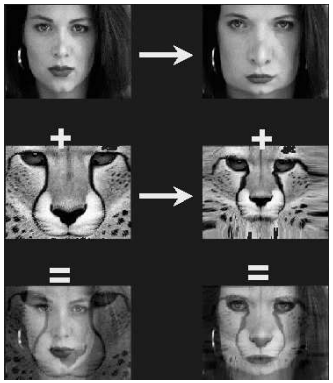


Image Combination

- 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

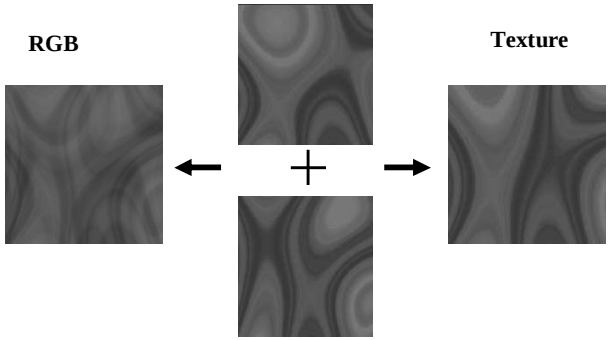


Image Combination: Adaptive cross-dissolve

- 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