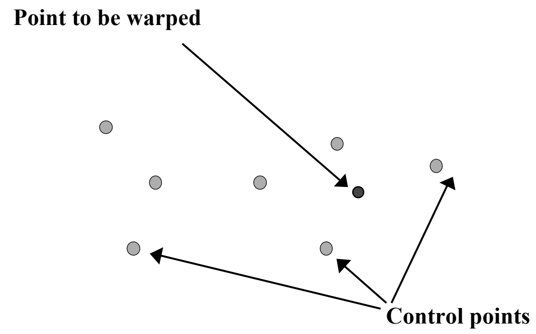
## *Interactive Computer Graphics*

- Lecture 15: Warping and Morphing (cont.)



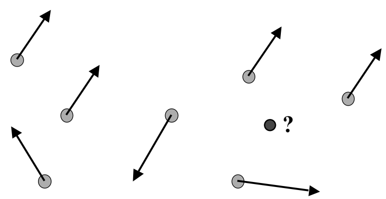
## *Non-rigid transformation*






## *Non-rigid transformation*

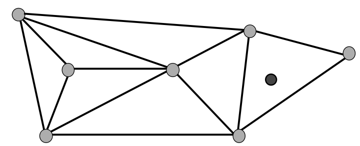
- For each control point we have a displacement vector
- How do we interpolate the displacement at a pixel?





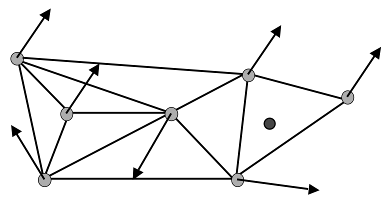

## *Non-rigid transformation: Piecewise affine*

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



## *Non-rigid transformation: Piecewise affine*

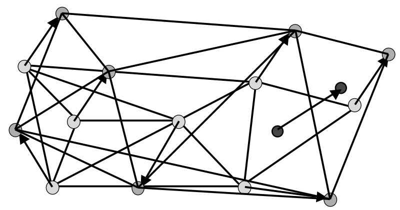
- Partition the convex hull of the control points into a set of triangles





## *Non-rigid transformation: Piecewise affine*

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

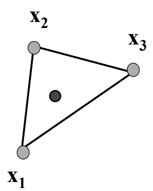




## *Non-rigid transformation: Piecewise affine*

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

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





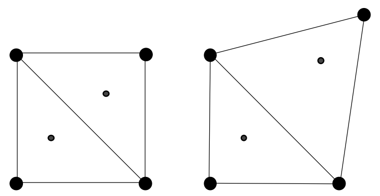

## *Non-rigid transformation: Piecewise affine*

- 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

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

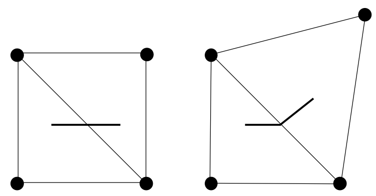
## *Non-rigid transformation: Piecewise affine*





## *Non-rigid transformation: Piecewise affine*

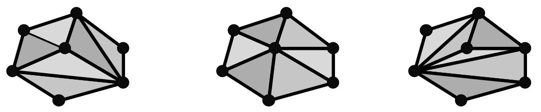
- Problem: Produces continuous deformations, but the deformation may not be smooth. Straight lines can be kinked across boundaries between triangles





## *Triangulations*

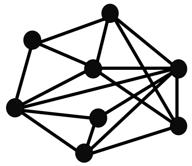
- A *triangulation* of set of points in the plane is a *partition* of the convex hull to triangles whose vertices are the points, and do not contain other points.
- There are an exponential number of triangulations of a point set.





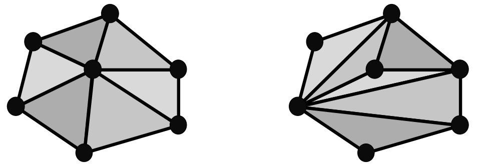
## *An O(*$n^3$*) Triangulation Algorithm*

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



## *"Quality" Triangulations*

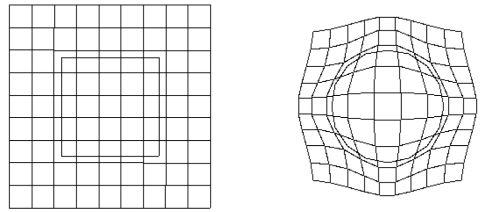
- Let $\alpha(T) = (\alpha_1, \alpha_2, .., \alpha_{3t})$ be the vector of angles in the triangulation $T$ in increasing order.
- A triangulation $T_1$ will be "better" than $T_2$ if $\alpha(T_1) > \alpha(T_2)$ lexicographically.
- The Delaunay triangulation is the "best"
  - Maximizes smallest angles



good bad



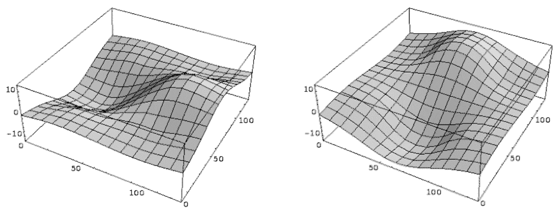
## *Representing deformations*



Before deformation After deformation



## *Representing deformations*



Displacement in the horizontal direction

Displacement in the vertical direction



## *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

## *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

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



## *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 $B_i$ corresponds to the B-spline basis functions

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



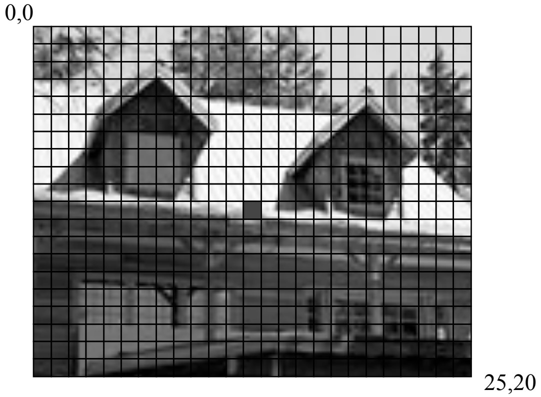
## *FFDs: Example*

- 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





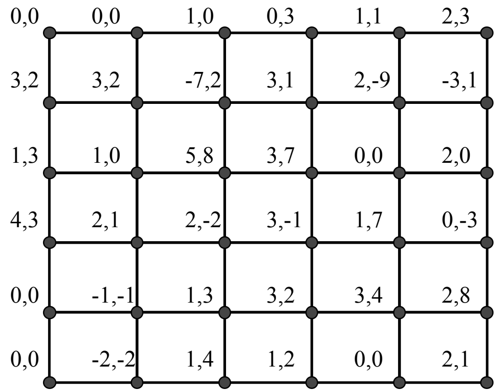

## FFDs: 2D Example



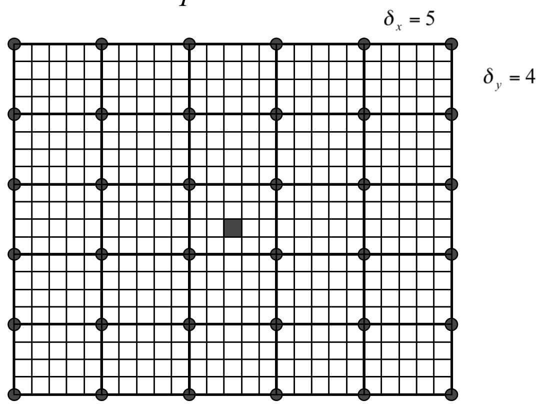
## FFDs: 2D Example



## FFDs: 2D Example



## *FFDs: 2D Example*






## *FFDs: 2D Example*

- Calculate integer lattice coordinates *i, j*:

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

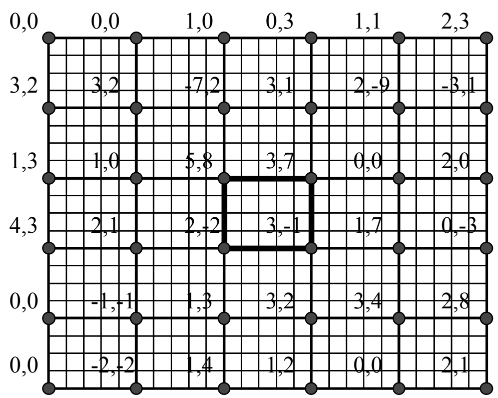
- Calculate fractional lattice coordinates *u,v*:

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

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



## *FFDs: 2D Example*



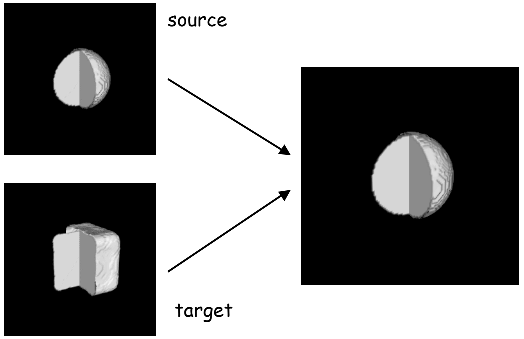



## *FFDs: 2D Example*

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

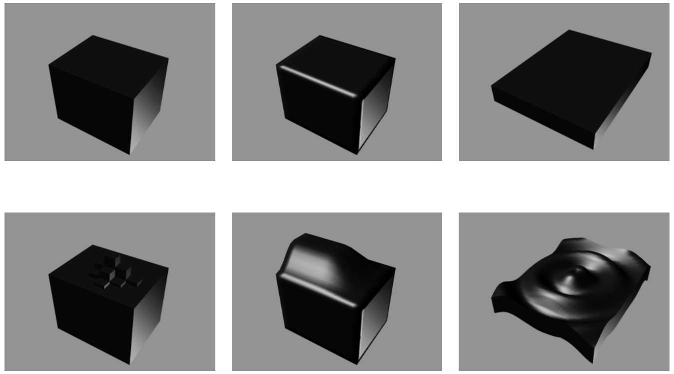
$$\begin{aligned}
\mathbf{u}(x,y) &= 0.15 \cdot \begin{pmatrix} 5 \\ 8 \end{pmatrix} + 0.45 \cdot \begin{pmatrix} 2 \\ -2 \end{pmatrix} + \\
&+ 0.10 \cdot \begin{pmatrix} 3 \\ 7 \end{pmatrix} + 0.3 \cdot \begin{pmatrix} 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 2.34 \\ 0.7 \end{pmatrix}
\end{aligned}$$

## *FFDs in 3D*






## *FFDs in 3D*



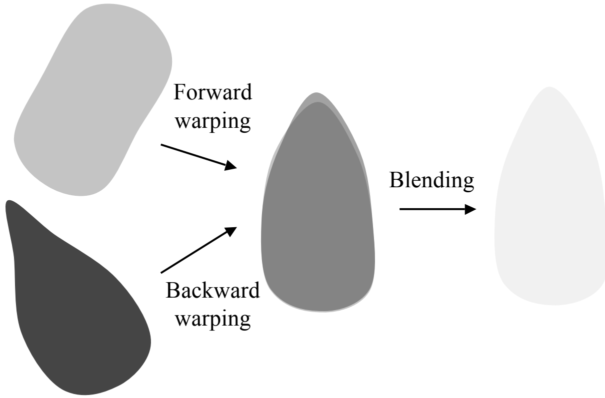


## *FFDs*

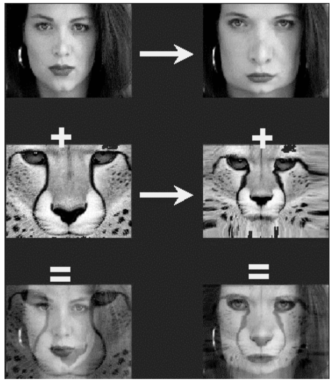
- Used for warping:
  - Lee et al. (1997)
- Advantages:
  - Control points have local influence since the basis function has finite support
  - Fast
    - linear (in 3D: 2 x 2 x 2 = 8 operations per warp)
    - cubic (in 3D: 4 x 4 x 4 = 64 operations per warp)
- Disadvantages:
  - Control points must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

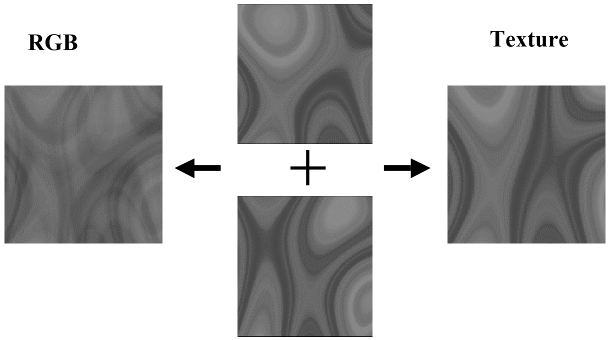
- Blending with cross-dissolve:

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

  - intensities
  - RGB space
  - HSV space
  - texture space

## *Image Combination: Cross-dissolve*

**RGB** **Texture**





## *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

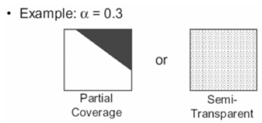


## *Image Combination: Alpha channel blending*

- Blending using RGBA images

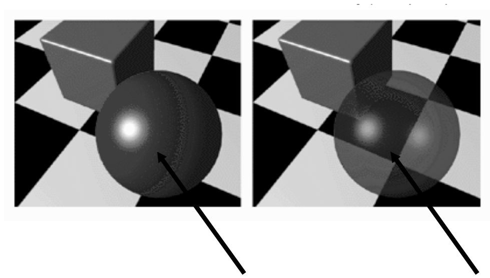
$$I = \alpha_a \cdot I_A + \alpha_b \cdot I_B$$

- 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*

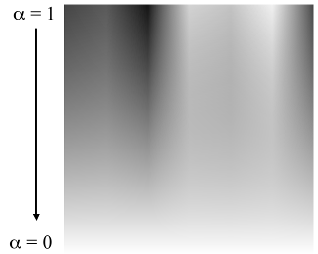


$\alpha = 1$ $\alpha = 0.5$

## *Image Combination: Alpha channel blending*

- Convention:
  - RGBA represents a pixel with color $C = (R, G, B)$ as

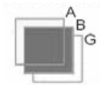
$$C = (\alpha r, \alpha g, \alpha b, \alpha)$$






## *Image Combination: Alpha channel blending*

- Suppose we put A over B over background G




  - How much of B is blocked by A?

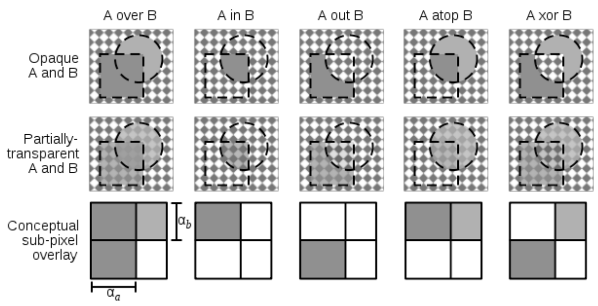
    $\alpha_A$
  - How much of B shows through A?

    $(1 - \alpha_A)$
  - How much of G shows through both A and B?

    $(1 - \alpha_A)(1 - \alpha_B)$



## *Image Combination: Alpha channel blending*






## *Image Combination: Alpha channel blending*

- Example: C = A over B

- For colors that are <u>not</u> premultiplied:
  - $C = \alpha_A A + (1-\alpha_A)\,\alpha_B B$
  - $\alpha = \alpha_A + (1-\alpha_A)\,\alpha_B$
- For colors that <u>are</u> premultiplied:
  - $C' = A' + (1-\alpha_A)\,B'$
  - $\alpha = \alpha_A + (1-\alpha_A)\,\alpha_B$

Assumption: coverages of A and B are uncorrelated for each pixel

A over B

## *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