Shape Modeling: Curves and Surfaces

Course web page:

http://goo.gl/EB3aA

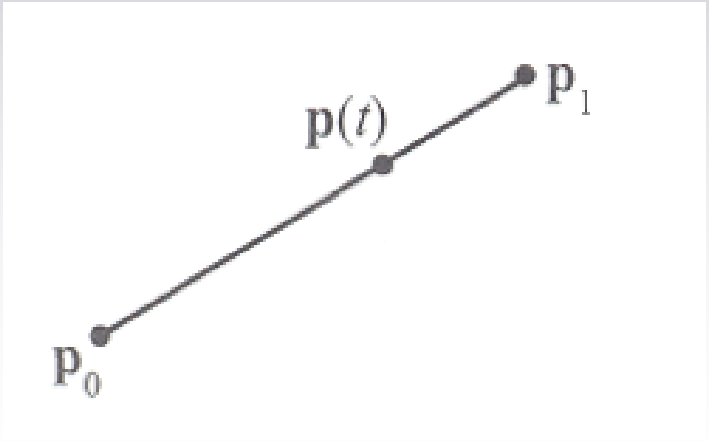
Outline

- Parametric curves and surfaces
  - Bezier curves
  - Catmull-Rom splines
  - Application: morphing

Parametric Lines

- Parametric definition of a line segment:

$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0), \text{ where } t \in [0, 1]$$
$$= \mathbf{p}_0 - t\,\mathbf{p}_0 + t\,\mathbf{p}_1$$
$$= (1 - t)\mathbf{p}_0 + t\,\mathbf{p}_1$$



from Akenine-Möller & Haines

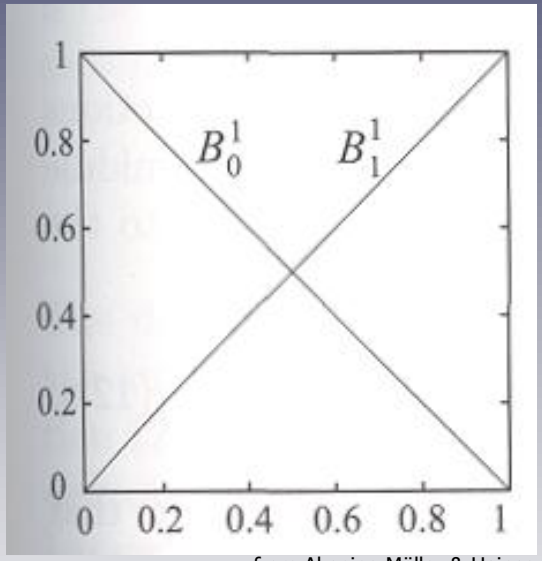
like a "blend" of the two endpoints

Linear Interpolation as Blending

- Consider each point on the line segment as a sum of **control points $\mathbf{p}_i$** weighted by **blending functions** B_i :

degree n = 1 for linear blending

$$\mathbf{p}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i$$



from Akenine-Möller & Haines

Blending functions for linear interpolation (2 control points)

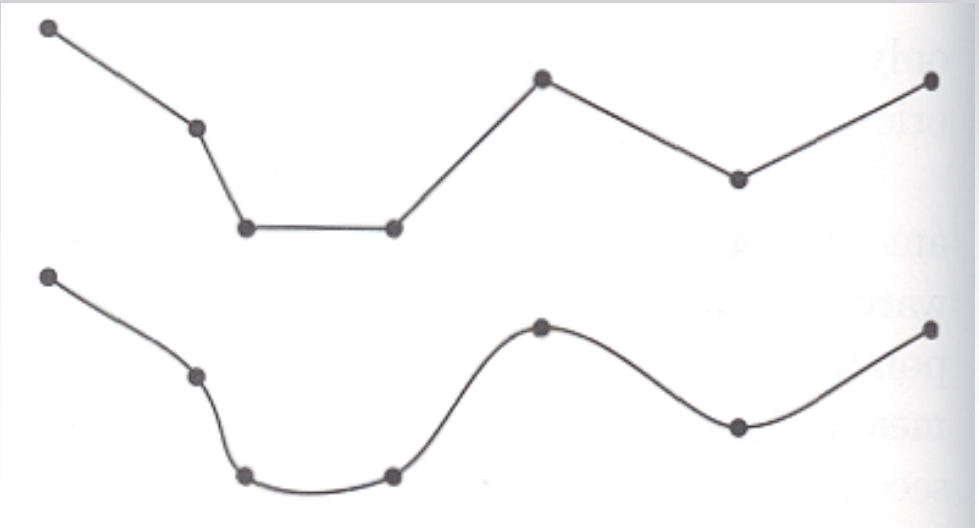
- Here we have $B_0 = 1 - t$ and $B_1 = t$

Improving Interpolation

- C^n continuity $\rightarrow$ n^{th} derivative is continuous everywhere on the curve
- Linear interpolation over multiple connected line segments has C^0 continuity, but not C^1 or higher continuity, which would make for a smoother curve

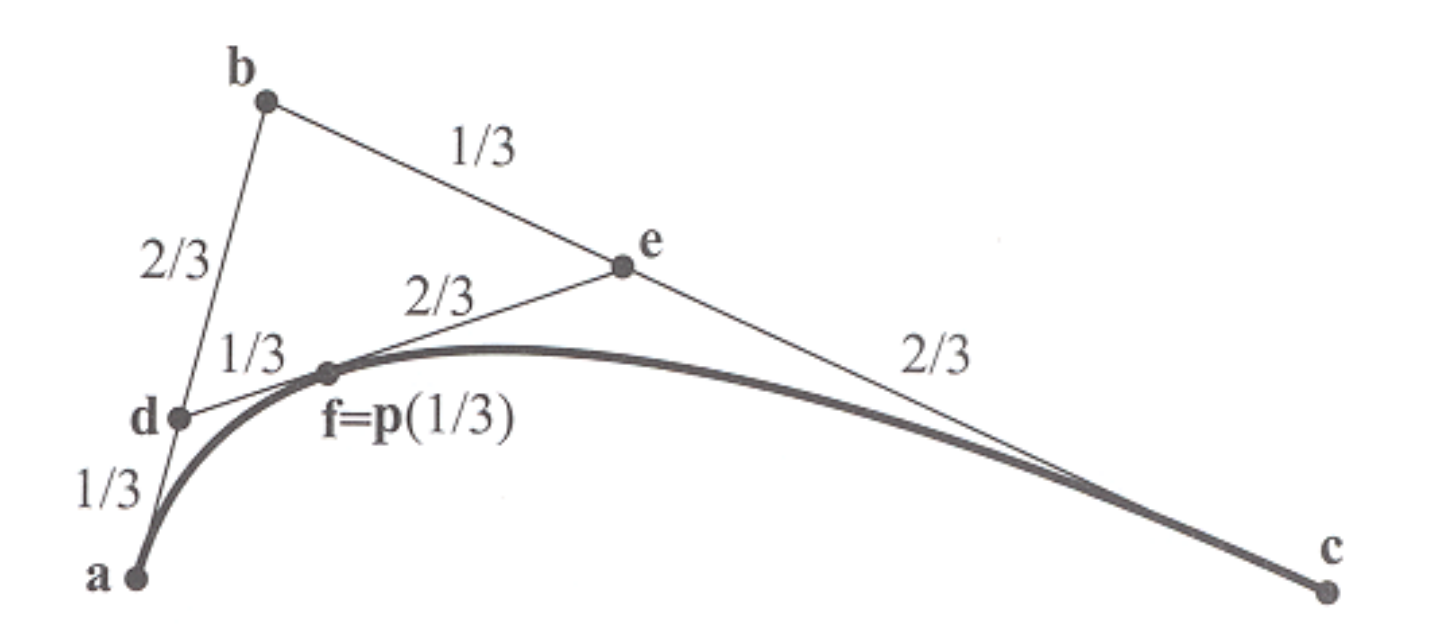
C^0 continuity

C^1 or higher continuity



from Akenine-Möller & Haines

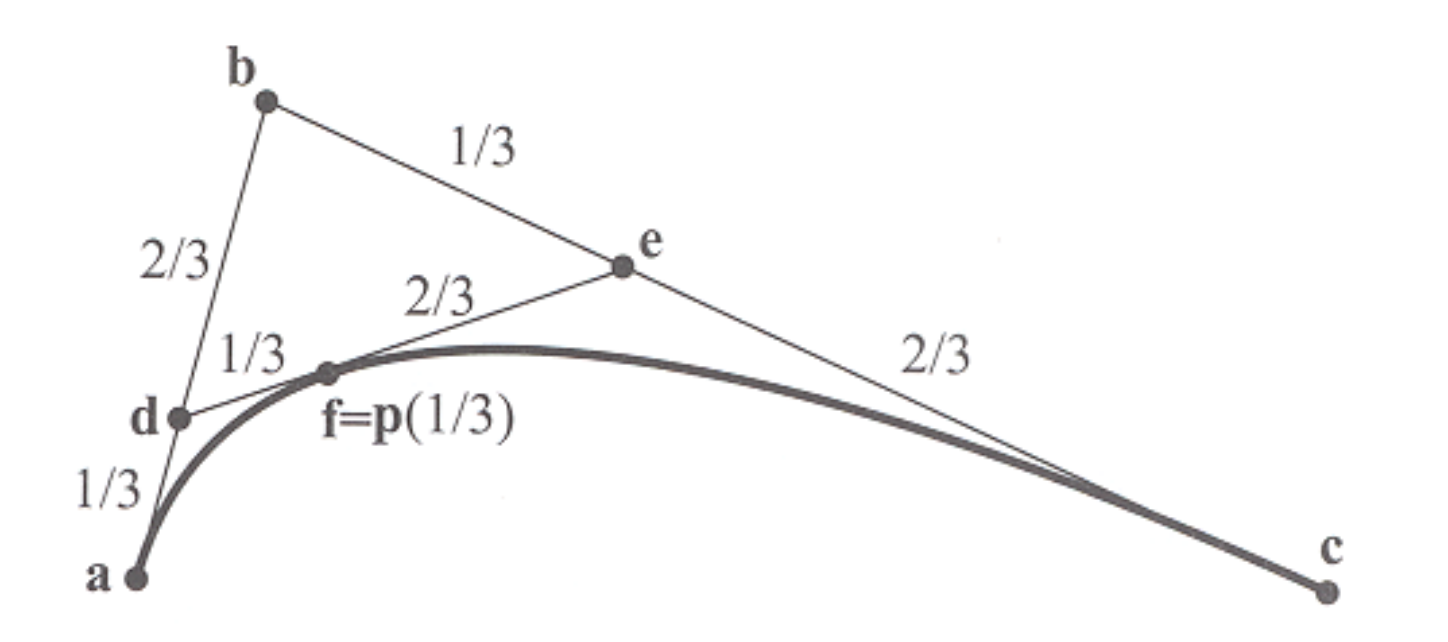
Interpolating Interpolants



from Akenine-Möller & Haines

- For 3 points **a**, **b**, and **c**, we can define a smoother curve by linearly interpolating along the line between points **d** and **e** linearly interpolated between **a**, **b** and **b**, **c**, respectively
- This curve **approximates a**, **b**, and **c**, because it doesn't go through all of them
- True **interpolating** curves include all of the original points

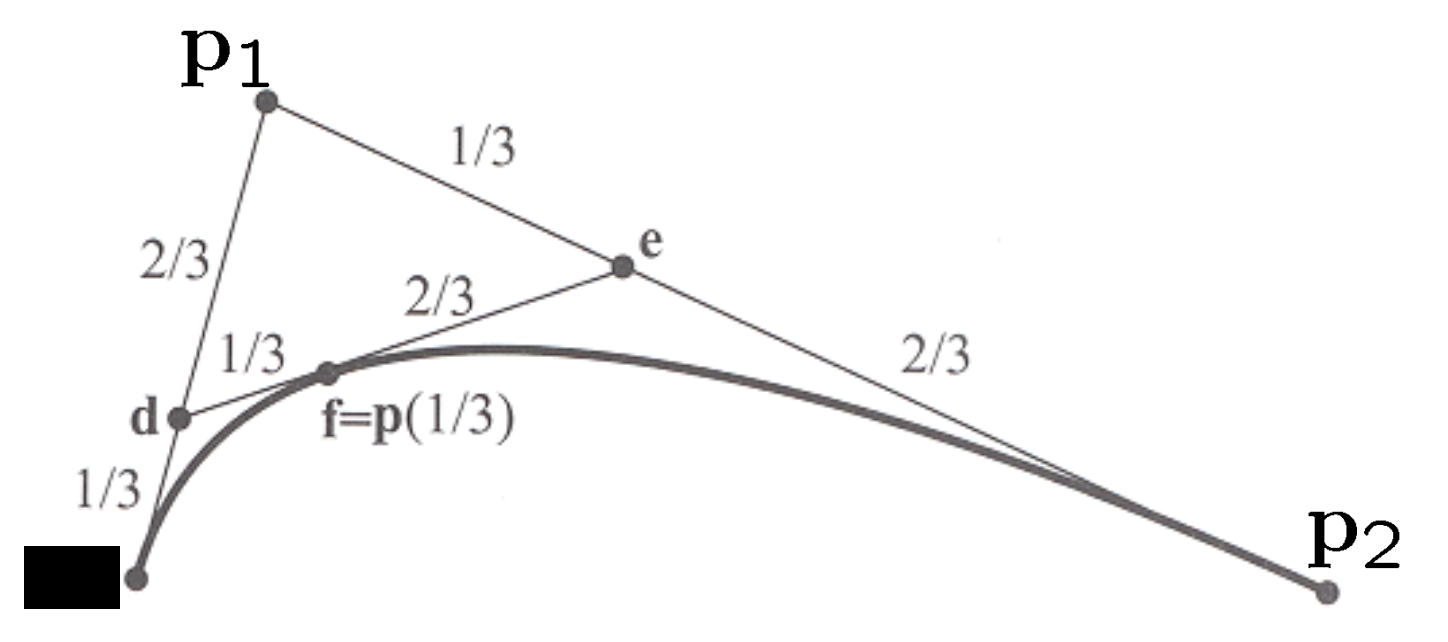
Interpolating Interpolants



from Akenine-Möller & Haines

$$\mathbf{p}(t) = (1 - t)\mathbf{d} + t\mathbf{e}$$
$$= (1 - t)[(1 - t)\mathbf{a} + t\mathbf{b}] + t[(1 - t)\mathbf{b} + t\mathbf{c}]$$

Changing notation…



from Akenine-Möller & Haines

$$\mathbf{p}(t) = (1 - t)[(1 - t)\mathbf{p}_0 + t\mathbf{p}_1] + t[(1 - t)\mathbf{p}_1 + t\mathbf{p}_2]$$

$$= (1 - t)^2\mathbf{p}_0 + 2t\,(1 - t)\mathbf{p}_1 + t^2\mathbf{p}_2$$

$$= \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i$$

now n = 2 → quadratic blending

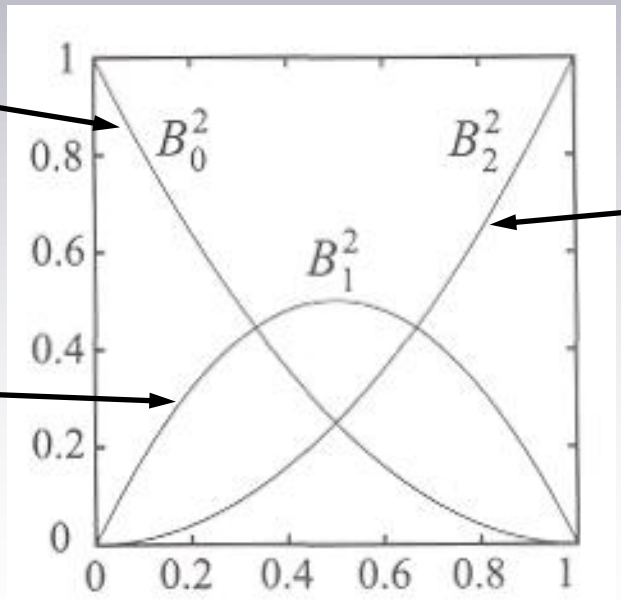
Quadratic Blending Functions

- Blending functions are also called **Bernstein** polynomials
- Magnitude of each proportional to control point's **influence** on the shape of the curve
  - Note that each is non-zero along entire curve

$(1-t)^2$

$2t(1-t)$

t^2

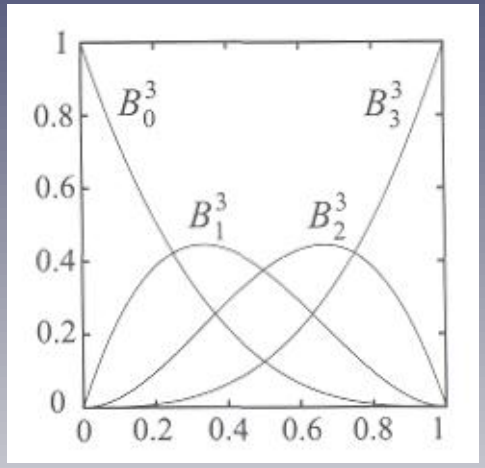


3 blending functions for 3 control points

from Akenine-Möller & Haines

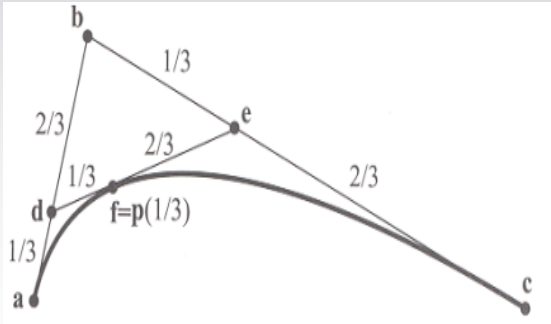
Bézier Curves

- Curve approximation through **recursive** application of linear interpolations
  - Linear: 2 control points, 2 linear Bernstein polynomials
  - Quadratic: 3 control points, 3 quadratic Bernstein polynomials
  - Cubic: 4 control points, 4 cubic polynomials
  - N control points = N - 1 degree curve
- Notes
  - Only endpoints are interpolated (i.e., on the curve)
  - Curve is tangent to linear segments at endpoints
  - Every control point affects **every** point on curve
    - Makes modeling harder



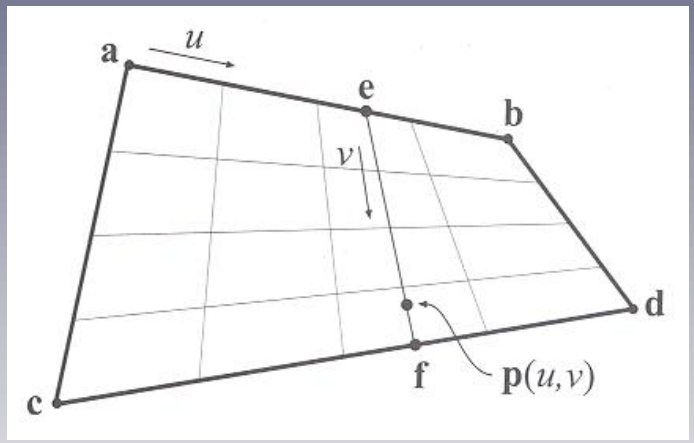
from Akenine-Möller & Haines

Cubic Bernstein polynomials for 4 control points

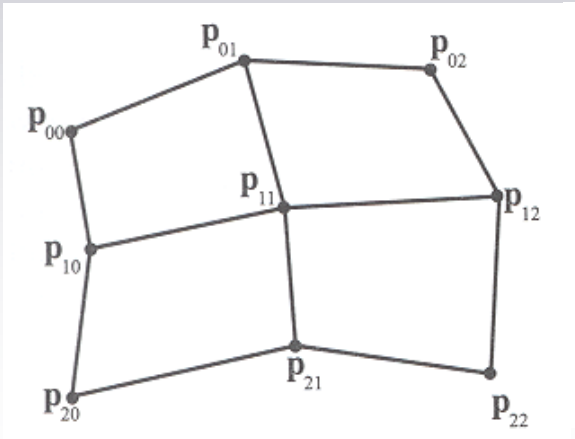


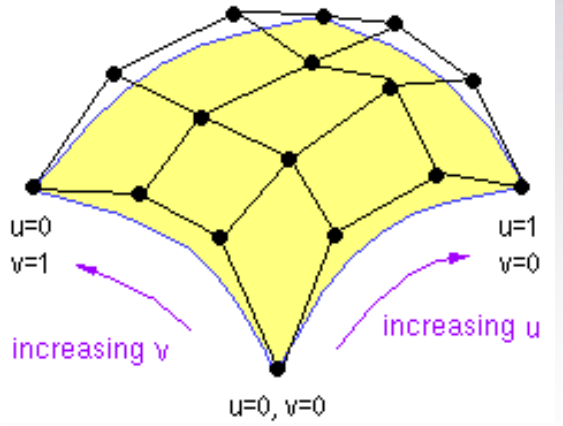
Extension to Surfaces

- Multiply two blending functions (one for each dimension) together
  - Bilinear patch
    - Need 2 x 2 control points
  - Biquadratic Bézier patch
    - Need 3 x 3 control points
  - Bicubic patch
    - 4 x 4 control points



from Akenine-Möller & Haines



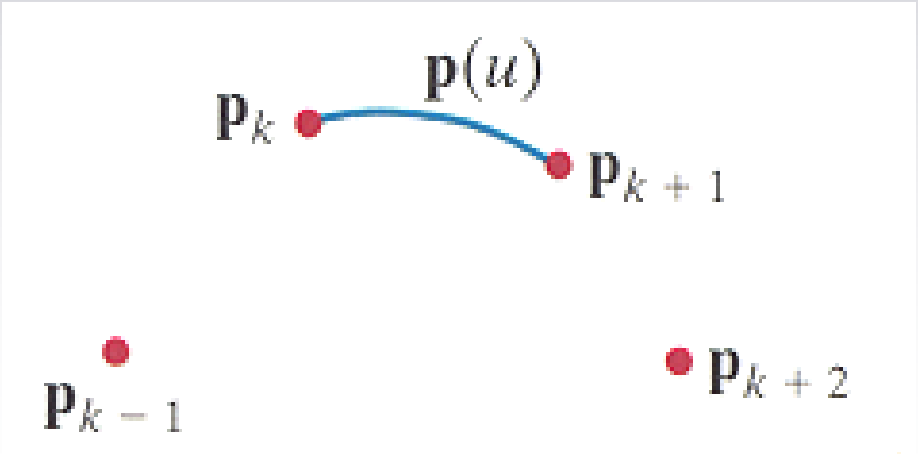


Interpolating Splines

- Idea: Use **key frames** to indicate a series of positions that must be "hit"
- For example:
  - Camera location
  - Path for character to follow
  - Animation of walking, gesturing, or facial expressions
    - Morphing
- Use splines for smooth interpolation
  - Must not be approximating!

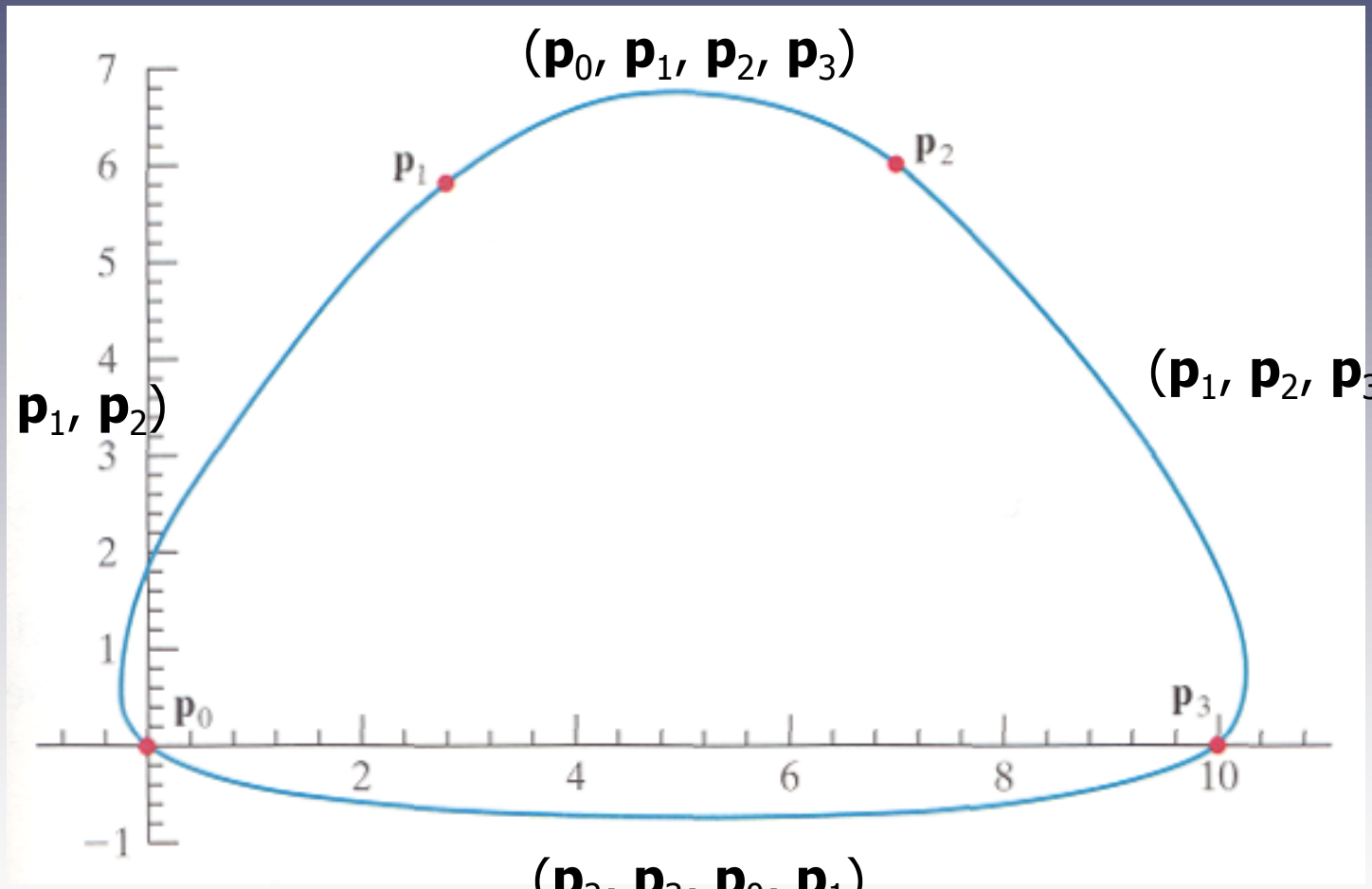
Catmull-Rom spline

- Different from Bezier curves in that we can have arbitrary number of control points, but **only 4 of them at a time** influence each section of curve
  - And it's **interpolating** (goes through points) instead of **approximating** (goes "near" points)
- Four points define curve between 2nd and 3rd



from Hearn & Baker

Catmull-Rom spline: Closed curve example

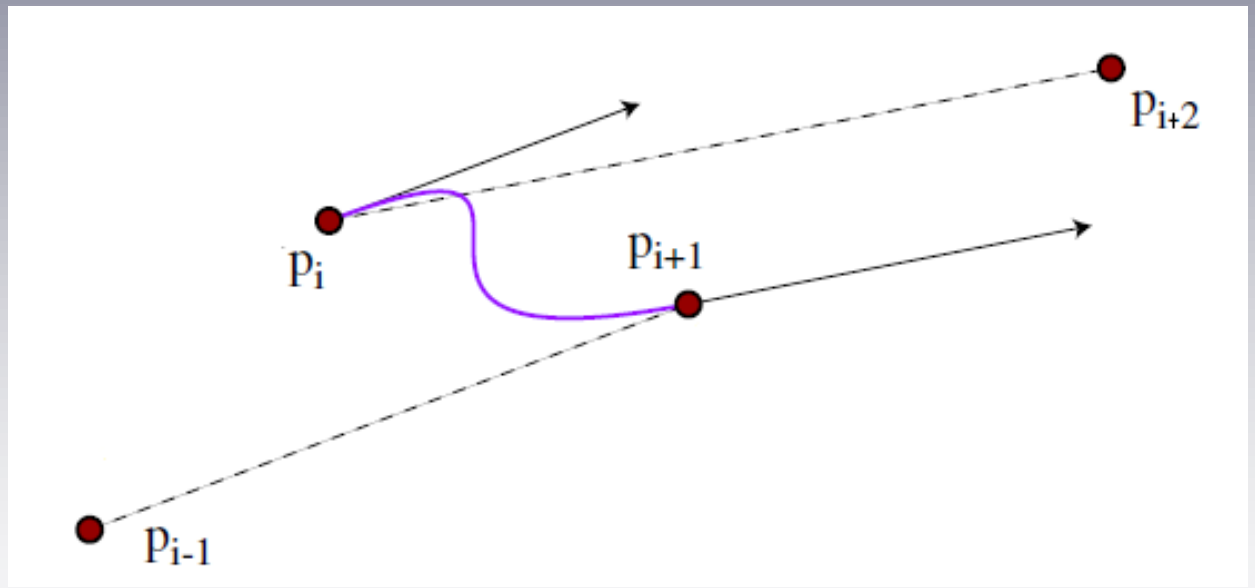


from Hearn & Baker

Applet: http://www.cse.unsw.edu.au/~lambert/splines/CatmullRom.html

Catmull-Rom spline

- Want cubic polynomial curve defined parametrically over interval $t \in [0, 1]$ with following constraints:
  - Starts at $\mathbf{P}(0) = \mathbf{P}_i$, ends at $\mathbf{P}(1) = \mathbf{P}_{i+1}$
  - Starting slope $\mathbf{P}'(0) = \mathbf{P}_{i+1} - \mathbf{P}_{i-1}$, ending slope $\mathbf{P}'(1) = \mathbf{P}_{i+2} - \mathbf{P}_i$



- Also require that it be cubic polynomial:

$$\mathbf{P}(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

Catmull-Rom: Blending matrix*

- Combine to get final **blending matrix**:

$$\mathbf{P}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_{i-1} \\ \mathbf{P}_i \\ \mathbf{P}_{i+1} \\ \mathbf{P}_{i+2} \end{bmatrix}$$

courtesy of K. Joy

- To trace curve, iterate through subsets of 4 control points ($\mathbf{p}_{i-1}$, $\mathbf{p}_i$, $\mathbf{p}_{i+1}$, $\mathbf{p}_{i+2}$)
  - Inner loop iterates over $t \in [0, 1]$

*Valid for all but first and last segments of **open curve** (use different method there for tangents)

Catmull-Rom spline properties

- Yields C_0, C_1 continuous curve which goes through every control point
  - Not C_2 continuous
- Curve does not necessarily lie within convex hull of control points

Splines for camera motion: Example

- Use Catmull-Rom spline to define smooth camera path—e.g., a roller coaster
- Then keep calling **`gluLookat()`** while tracing the curve*

- Example video: youtube.com/watch?v=WwLWwaMrIYM

*There are a few more details regarding keeping track of the camera's **up** vector—see http://www-2.cs.cmu.edu/~fp/courses/02-graphics/asst5/vectors.html

Application: Morphing

- Goal is smooth transformation between image of one object and another
- Simplest approach is **cross-fading**: Animate image blending as α varies from 1 to 0 smoothly

$\alpha = 1.0$

$\alpha = 0.75$

$\alpha = 0.5$

$\alpha = 0.25$

$\alpha = 0.0$

from G. Wolberg, CGI '96

Morphing: Cross-Fading Issues

- Problem with cross-fade is that if features don't line up exactly, we get a double image
- Can try shifting/scaling/etc. one **entire** image to get better alignment, but this doesn't completely fix problem
- Can handle more situations by applying different warps to different **pieces** of image
  - Manually chosen
  - Takes care of feature correspondences

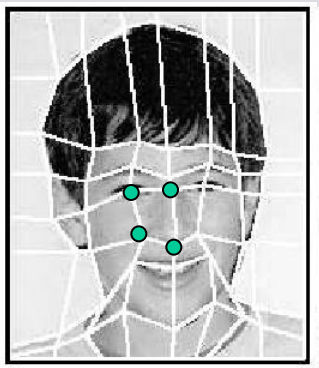


Image $\mathbf{I}_S$ with mesh $\mathbf{M}_S$ defining pieces

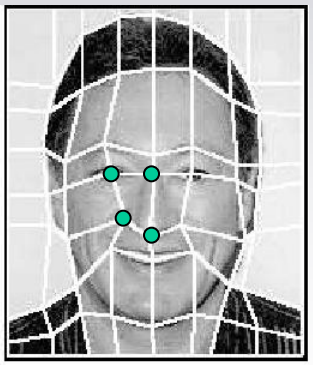


Image $\mathbf{I}_T$, mesh $\mathbf{M}_T$

from G. Wolberg, CGI '96

Morphing: Mesh Warping Algorithm

for f = 0 to 1 **do**

1. Linearly interpolate mesh vertices between $\mathbf{M}_S$ and $\mathbf{M}_T$ to get $\mathbf{M}^f$
2. Warp image $\mathbf{I}_S$ to $\mathbf{I}^f_S$ using $\mathbf{M}_S$ and $\mathbf{M}^f$
3. Warp $\mathbf{I}_T$ to $\mathbf{I}^f_T$ using $\mathbf{M}_T$ and $\mathbf{M}^f$
4. Linearly interpolate morphed image $\mathbf{I}^f$ between images $\mathbf{I}^f_S$ and $\mathbf{I}^f_T$ (i.e., blend them together with $\alpha = 1 - f$)

end

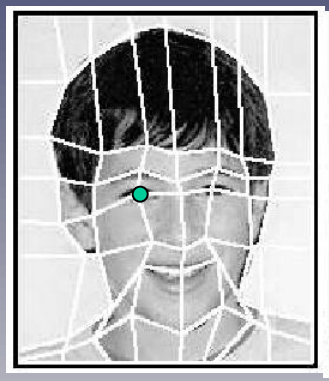


Image $\mathbf{I}_S$ with mesh $\mathbf{M}_S$

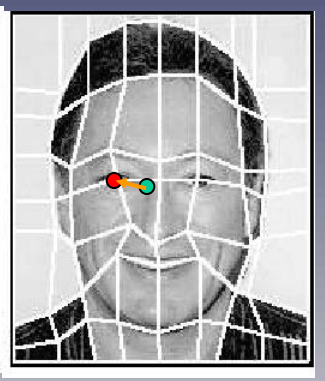


Image $\mathbf{I}_T$ with mesh $\mathbf{M}_T$

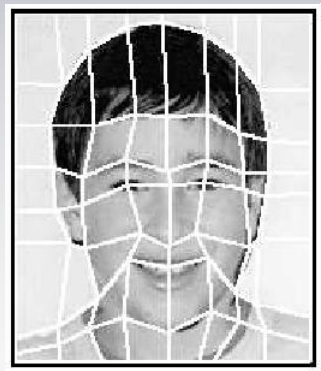


Image $\mathbf{I}^{0.5}_S$ with mesh $\mathbf{M}^{0.5}$

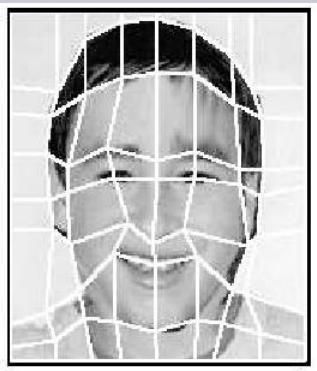
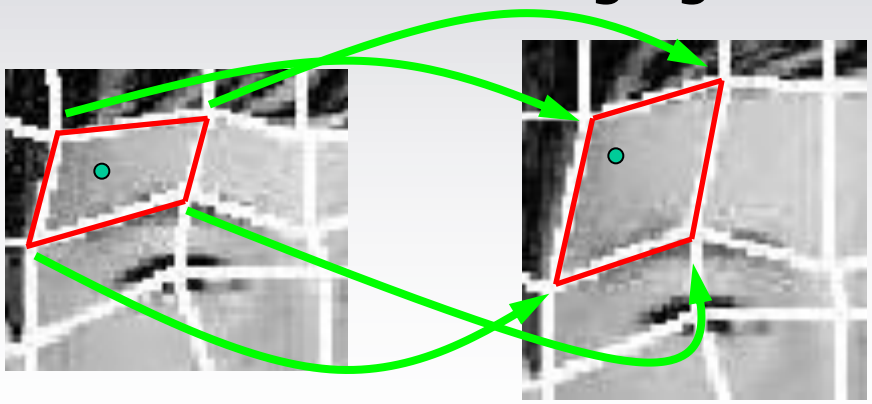


Image $\mathbf{I}^{1.0}_S$ with mesh $\mathbf{M}^{1.0}$

from G. Wolberg, CGI '96

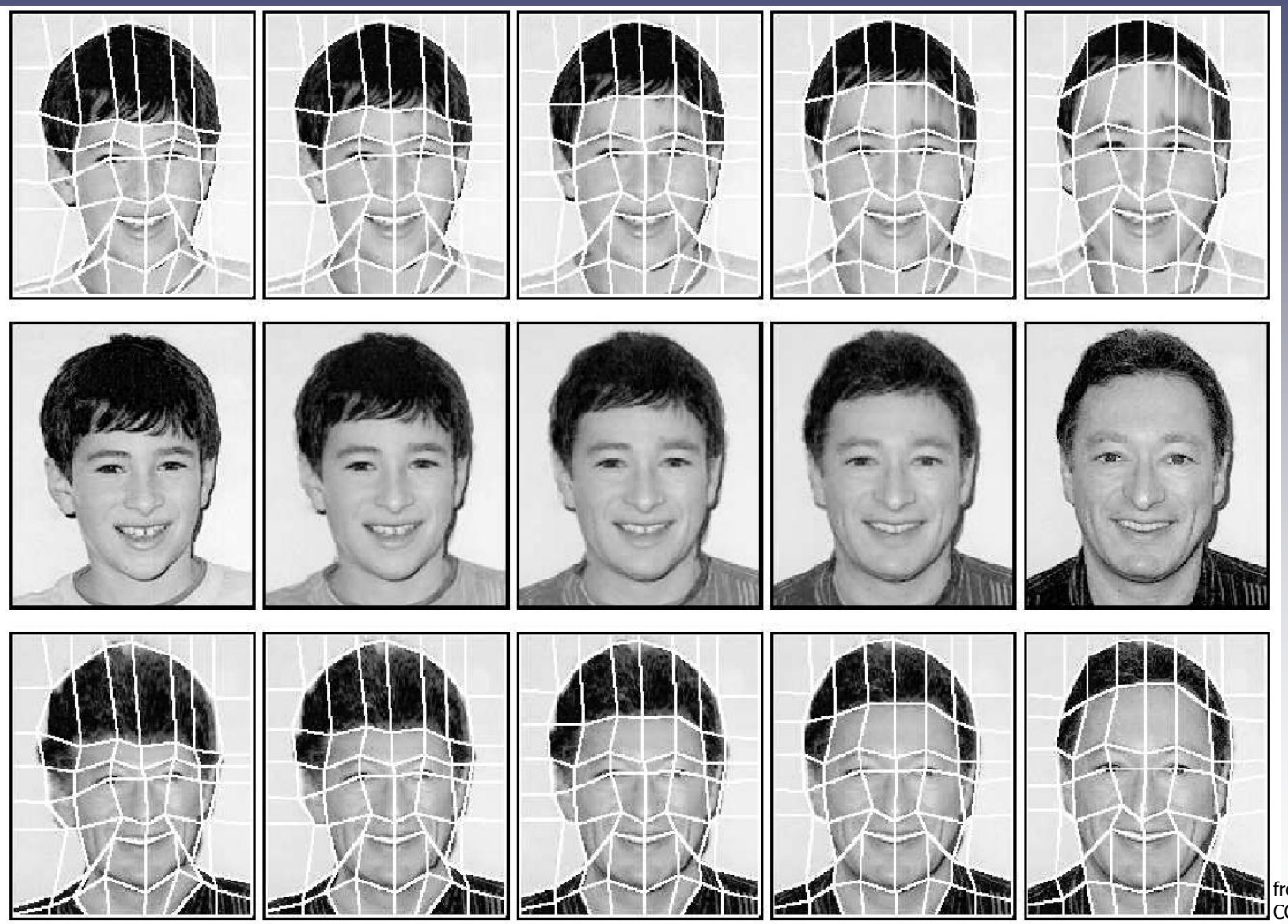
Mesh Warping: Splines

- For steps 2 & 3, use cubic splines to interpolate new pixel locations between warped mesh vertices
  - E.g., Catmull-Rom
- Could use bilinear patch for each piece, but wouldn't have C^1 continuity of **intensity** at borders
  - I.e., could get a faceted effect akin to Gouraud shading without normal averaging



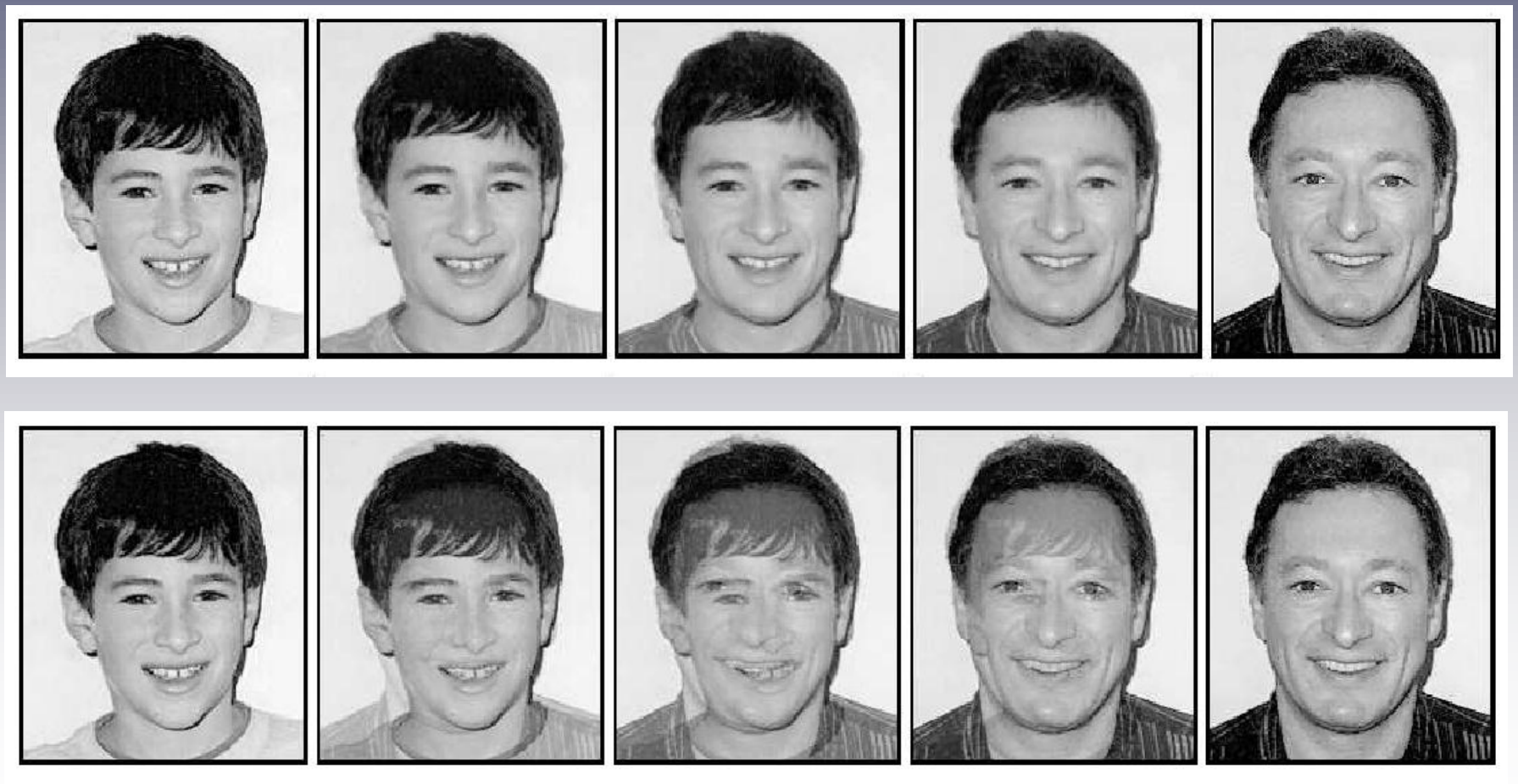
adapted from G. Wolberg, CGI '96

Morphing: Mesh Warping



from G. Wolberg, CGI '96

Mesh Warping vs. Cross-fading



from G. Wolberg, CGI '96

One more morphing example…

This is from the same paper, but using a line correspondence method rather than a mesh-based one

One more morphing example…