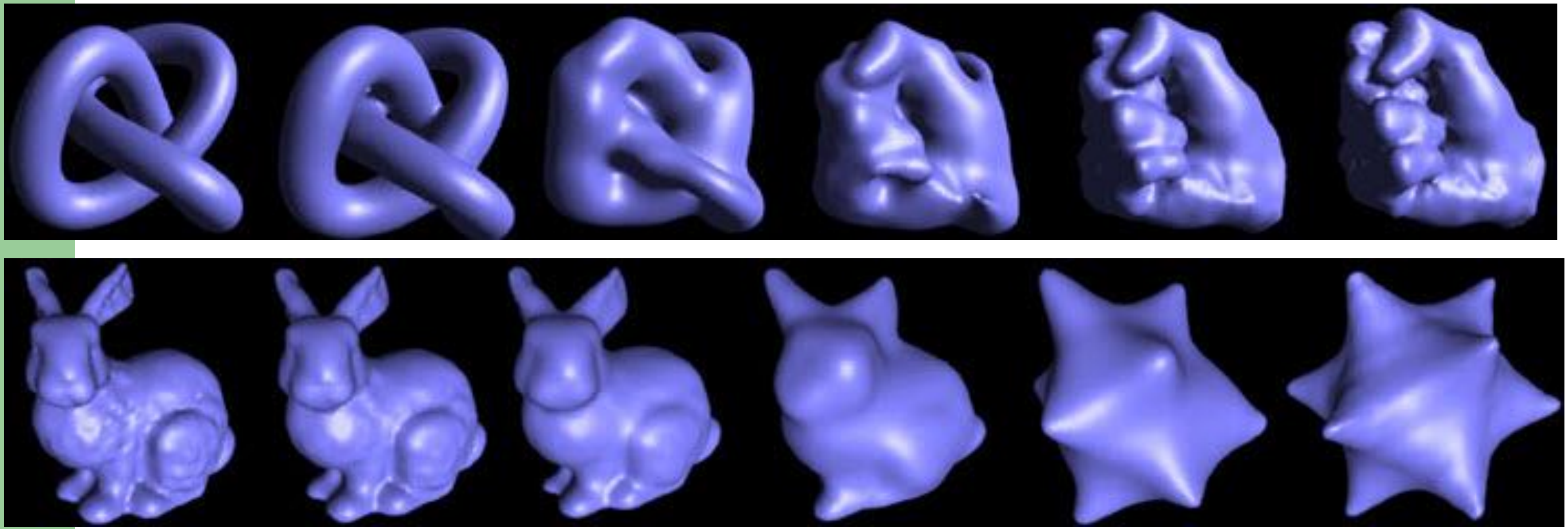
4.4 3D Shape Interpolation

- changing one 3D object into another



Turk/O'Brien

Two categories:

surface based approach (2^{nd} case)

volume based approach (1^{st} case)

Surface based approach (2nd case)

- modify boundary representations of the objects so that vertex-edge topologies would match then perform vertex interpolation
- sensitive to different object topologies (holes)

(horse_sph_morph_loop.avi)

Volume based approach (1st case)

- blend one set of volume elements into another set of volume elements
- less sensitive to different object topologies
- computationally more expensive
- a more promising approach but will not be covered here

Terminologies:

Donut and teacup have the same topology (one hole, one component)

object - 3D entity with finite volume

shape - an object's surface

model - rep of the shape of an object

topology - connectivity of the surface of an object (no. of holes, components)

homeomorphism

genus

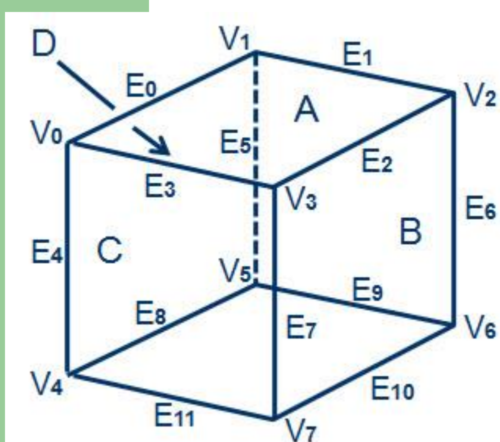
\- vertex/edge/face connectivity of an object (vertex -> vertex; edge -> edge; loop -> loop; face -> face; sphere (a simply-connected closed surface -> sphere)

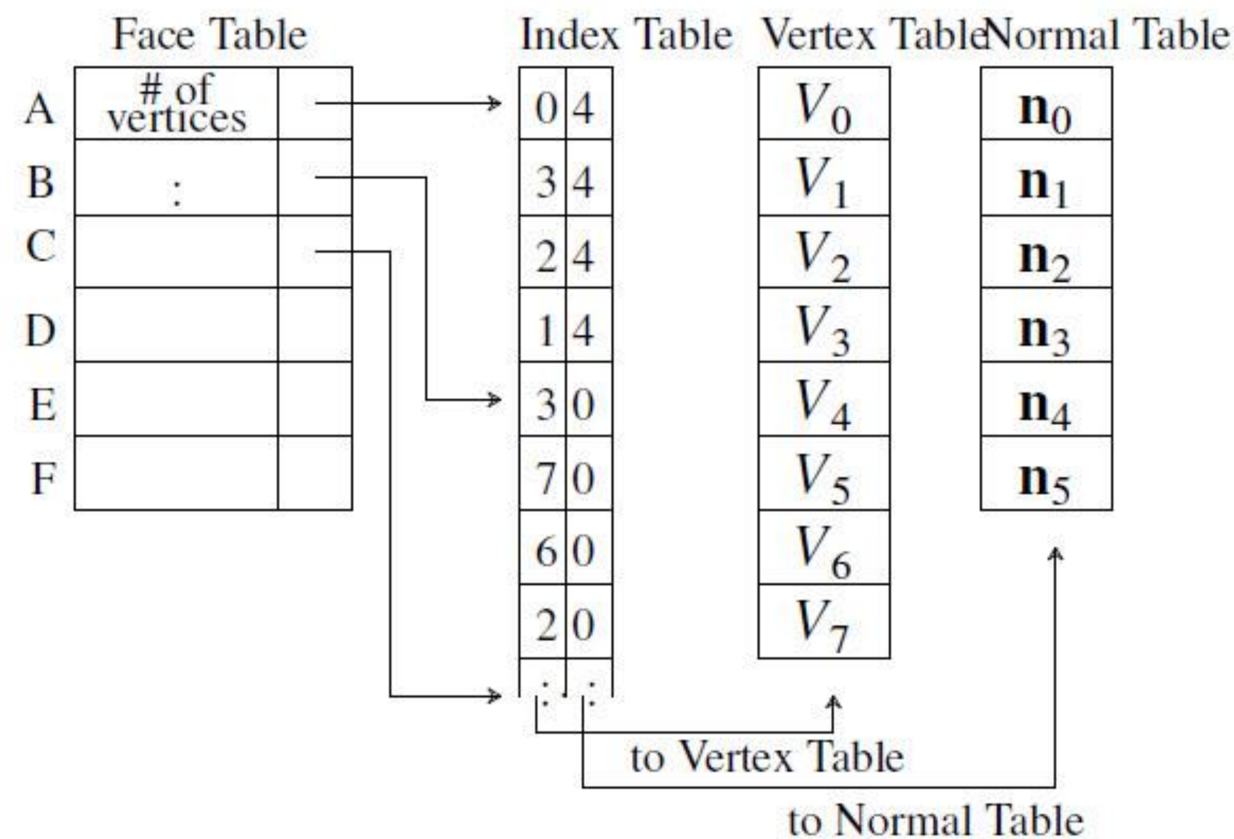
Question: if 2 objects are not topologically equivalent in the 1st sense, can they be topologically equivalent in the 2nd sense? **NO**

(the other direction?) **YES**

Shape Representation: a review

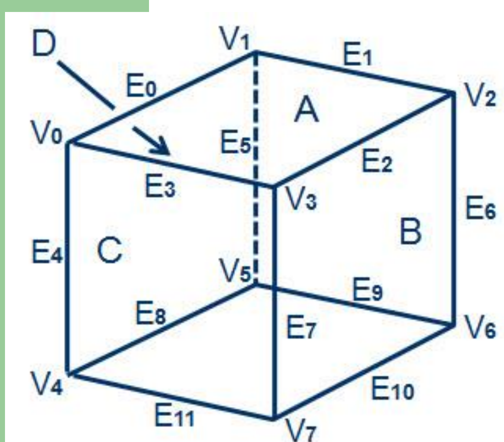
A representation of a 3D object:

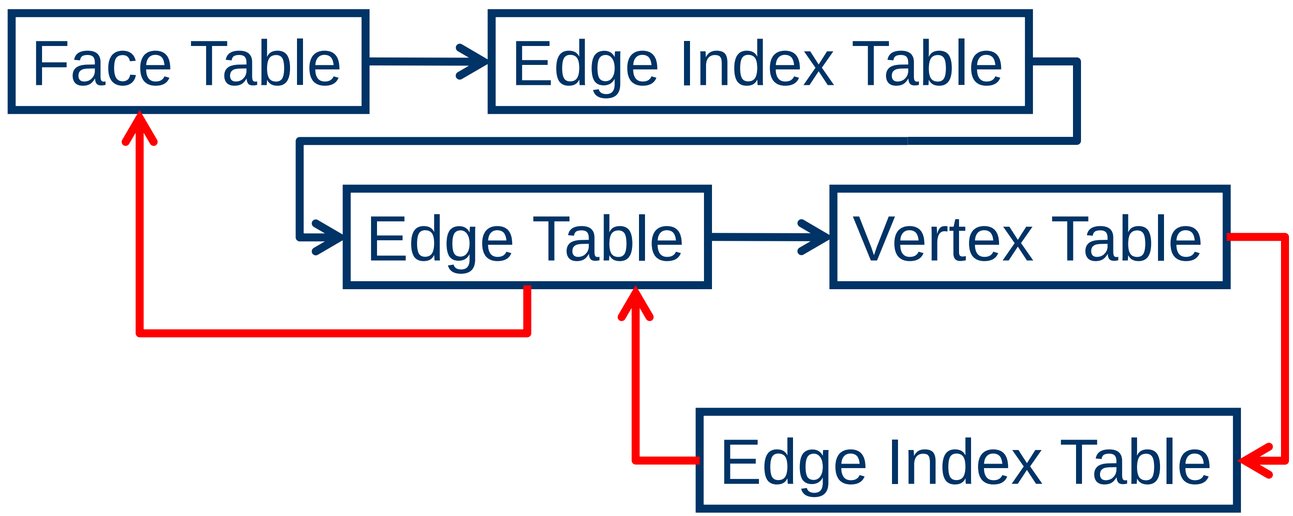




Shape Representation: a review

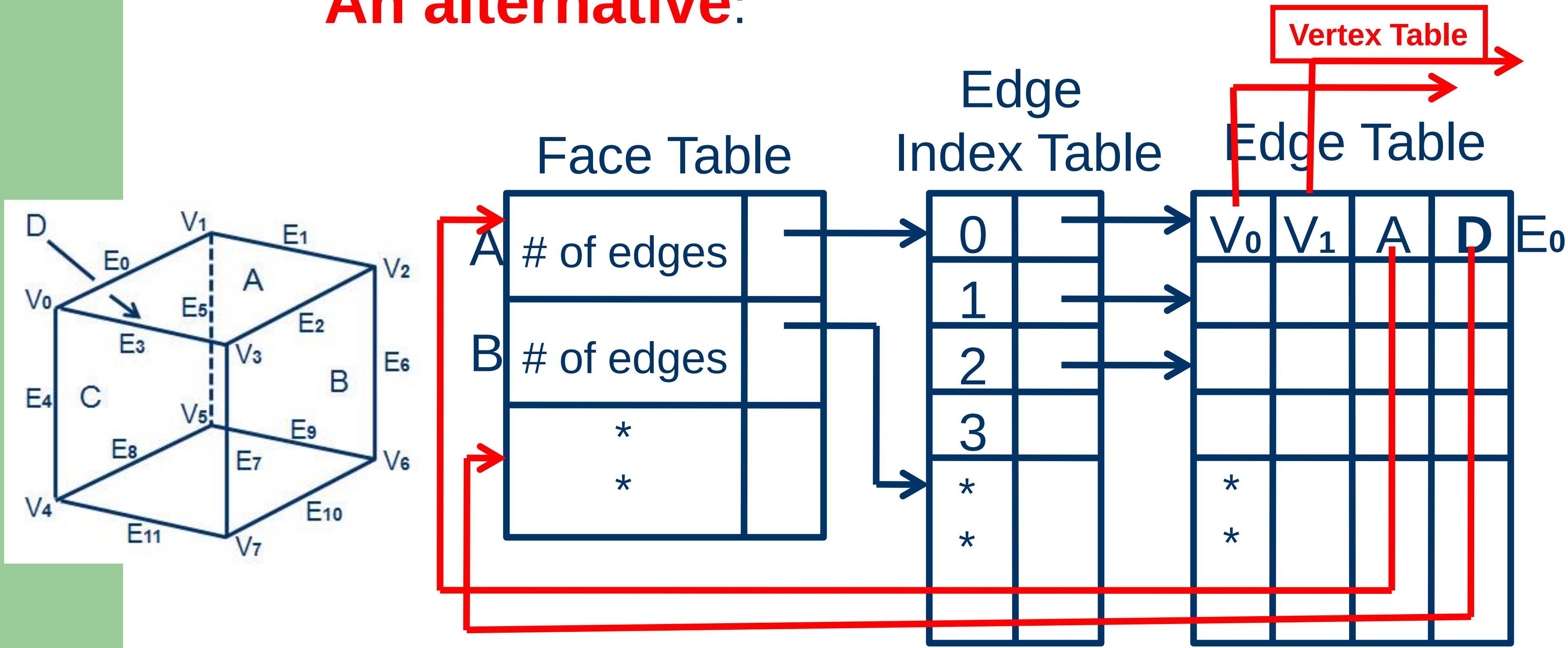
An alternative:





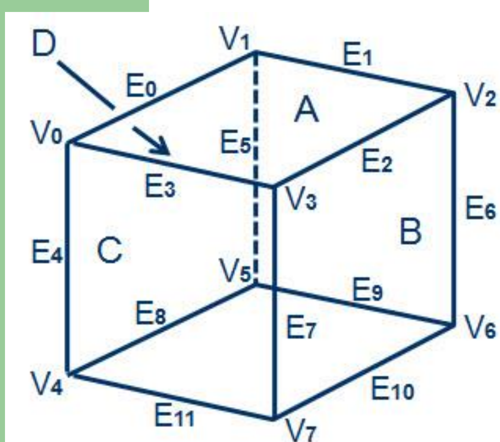
Shape Representation: a review

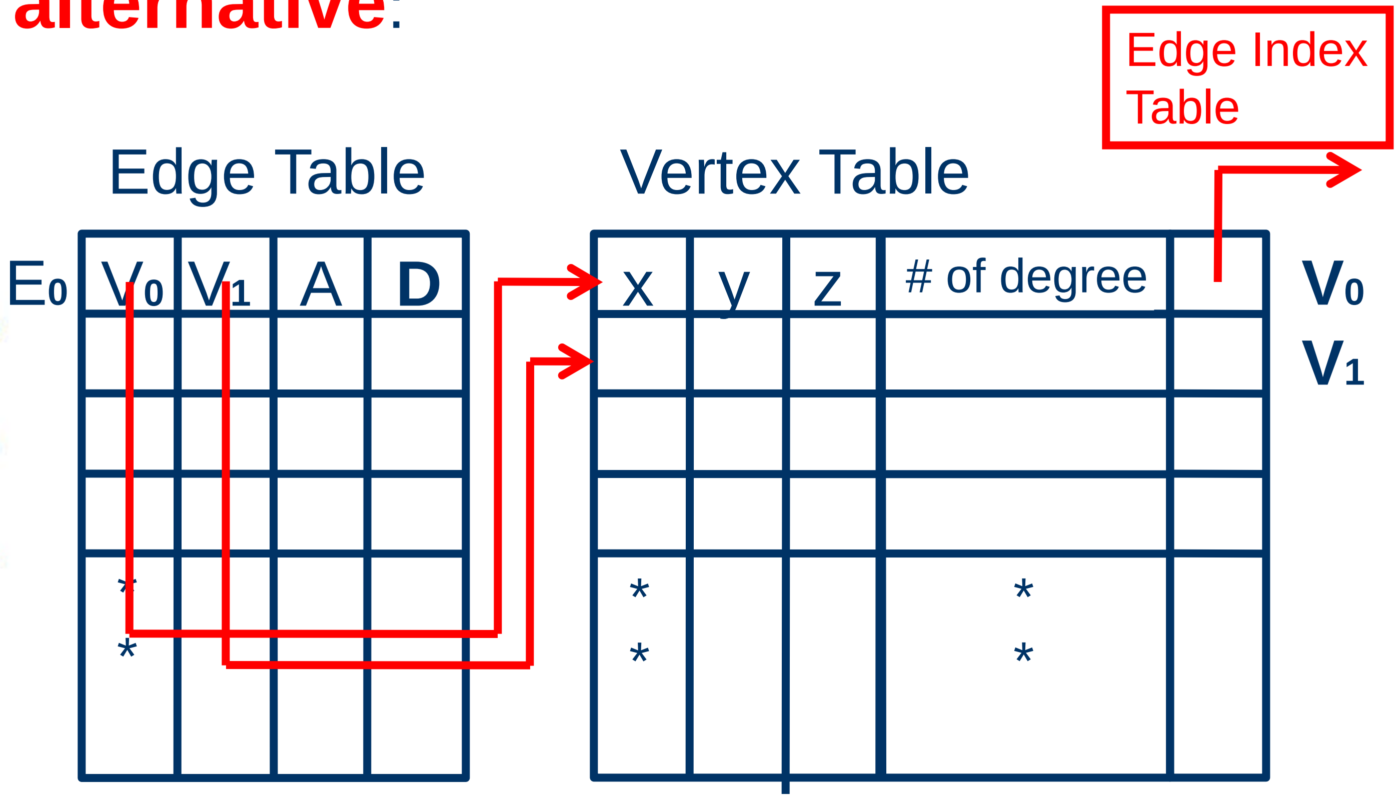
An alternative:



Shape Representation: a review

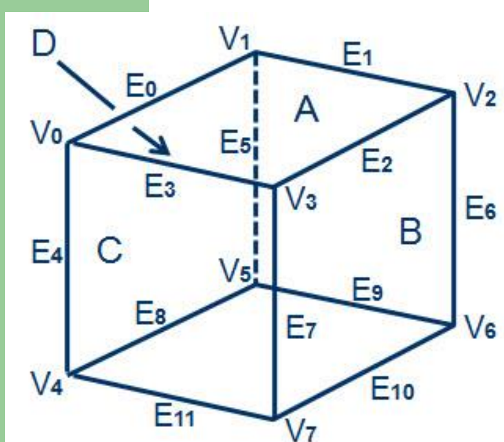
An alternative:





Shape Representation: a review

An alternative:



Vertex Table

V_0	x_0	y_0	z_0	# of degree	→
V_1	x_1	y_1	z_1	# of degree	→
	*			*	
	*			*	

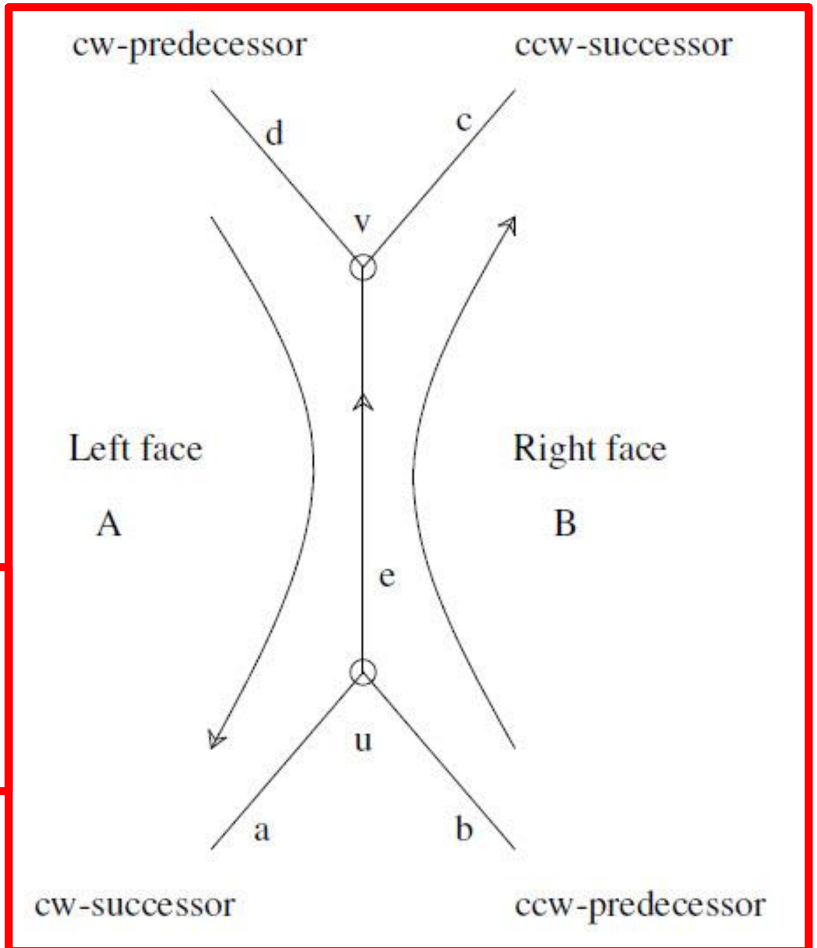
Edge Index Table

0	→ Edge Table
3	→
4	
0	
*	
*	

Winged-Edge Data Structure: a review

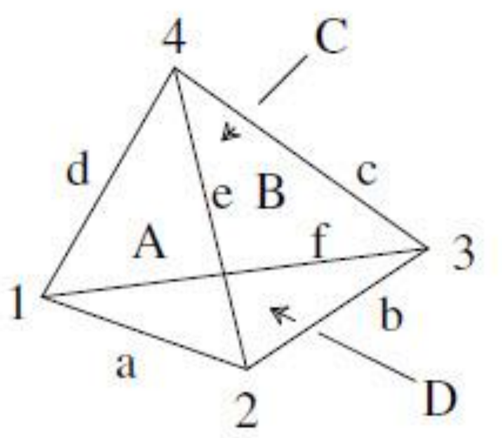
- To avoid using variable-length data structures
- Hide the implementation behind a class interface

Edge	Vertices		Faces		Clockwise		Counter-Clockwise	
	from	to	left	right	pred	succ	pred	succ
e	u	v	A	B	d	a	b	c



Winged-Edge Data Structure: a review

Example: representing a tetrahedron

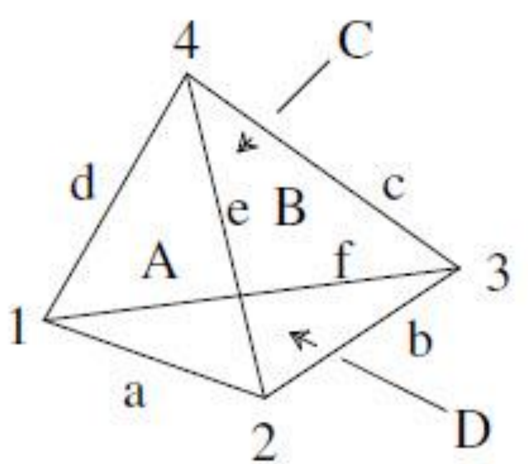


Edge Table

Edge Name	Vertices from	Vertices to	Faces left	Faces right	Clockwise pred	Clockwise succ	Counter-Clockwise pred	Counter-Clockwise succ
a	1	2	A	D	e	d	f	b
b	2	3	B	D	c	e	a	f
f	3	1	C	D	d	c	b	a
c	3	4	B	C	e	b	f	d
d	1	4	C	A	c	f	a	e
e	2	4	A	B	d	a	b	c

Winged-Edge Data Structure: a review

Example: representing a tetrahedron



Vertex-Edge Table

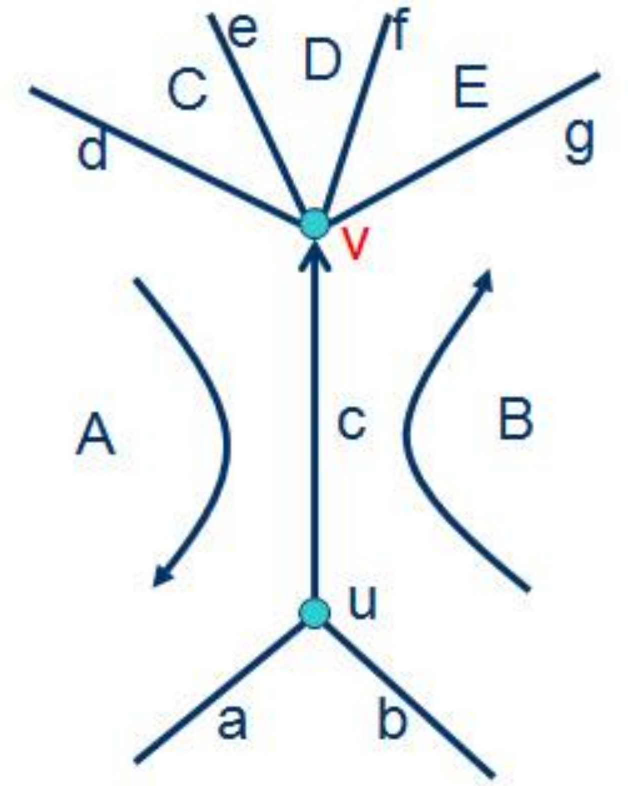
Vertex	Edge
1	d
2	b
3	b
4	c

Face-Edge Table

Face	Edge
A	d
B	e
C	d
D	b

Winged-Edge Data Structure: a review

- How to find edges adjacent to a given vertex v ?



Vertex-Edge Table (to find c)

↓

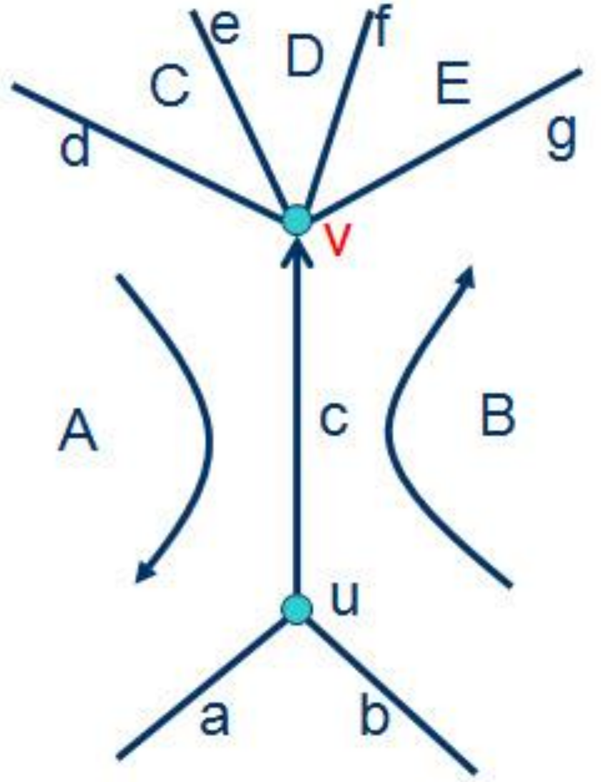
Edge Table (to find g)

↓

Edge Table (to find f)

Winged-Edge Data Structure: a review

- How to find edges adjacent to a given vertex v ?



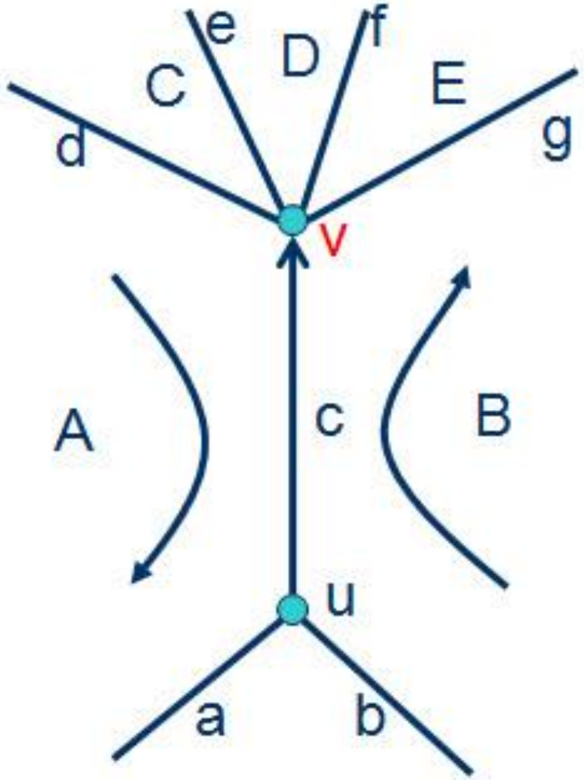
Edge Table (to find e)

Edge Table (to find d)

Edge Table (to find c, stop)

Winged-Edge Data Structure: a review

- How to find faces adjacent to a given vertex v ?



Vertex-Edge Table (to find c)

↓

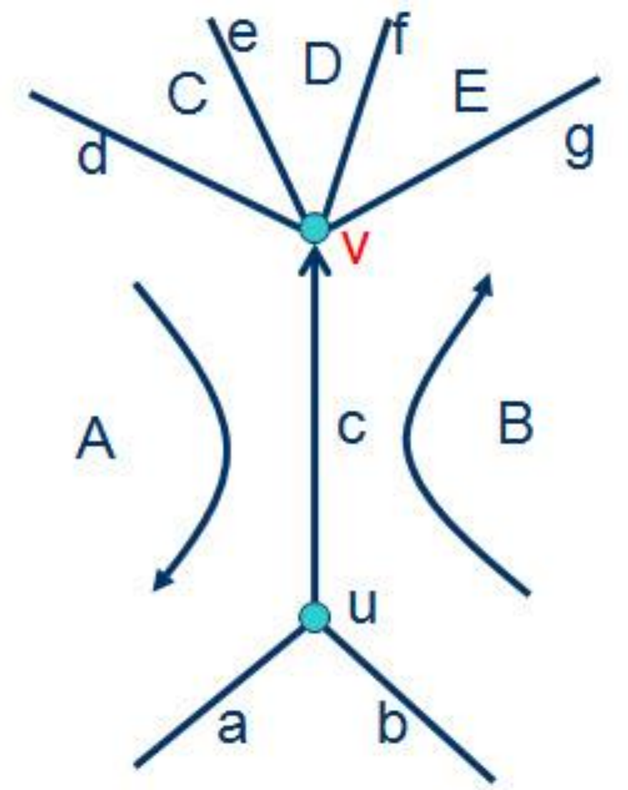
Edge Table (to find B and g)

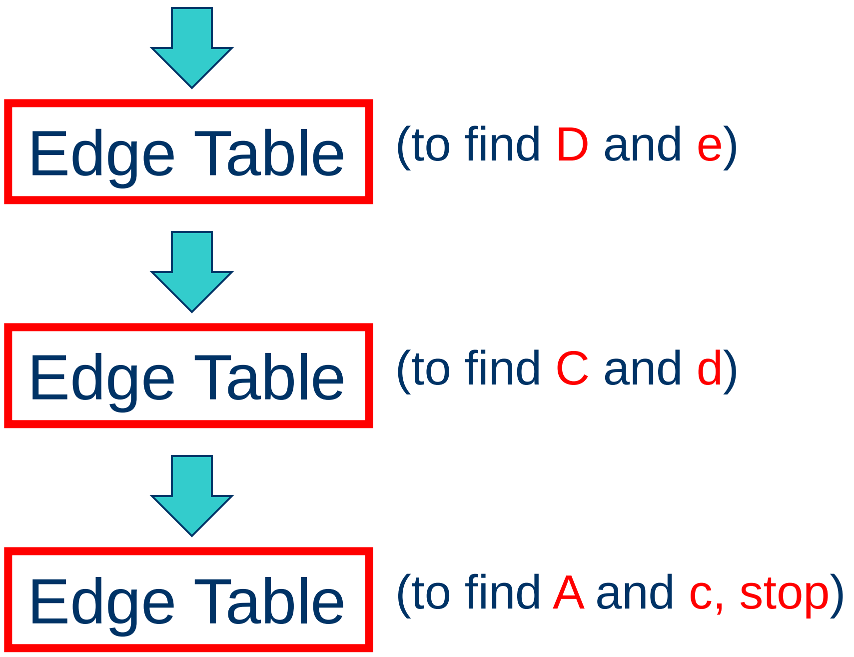
↓

Edge Table (to find E and f)

Winged-Edge Data Structure: a review

- How to find faces adjacent to a given vertex v ?



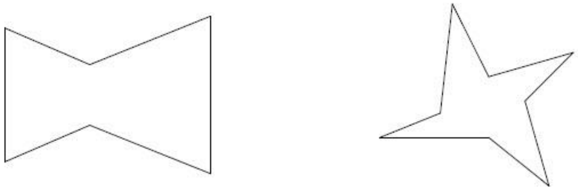


CS Dept, Univ of Kentucky

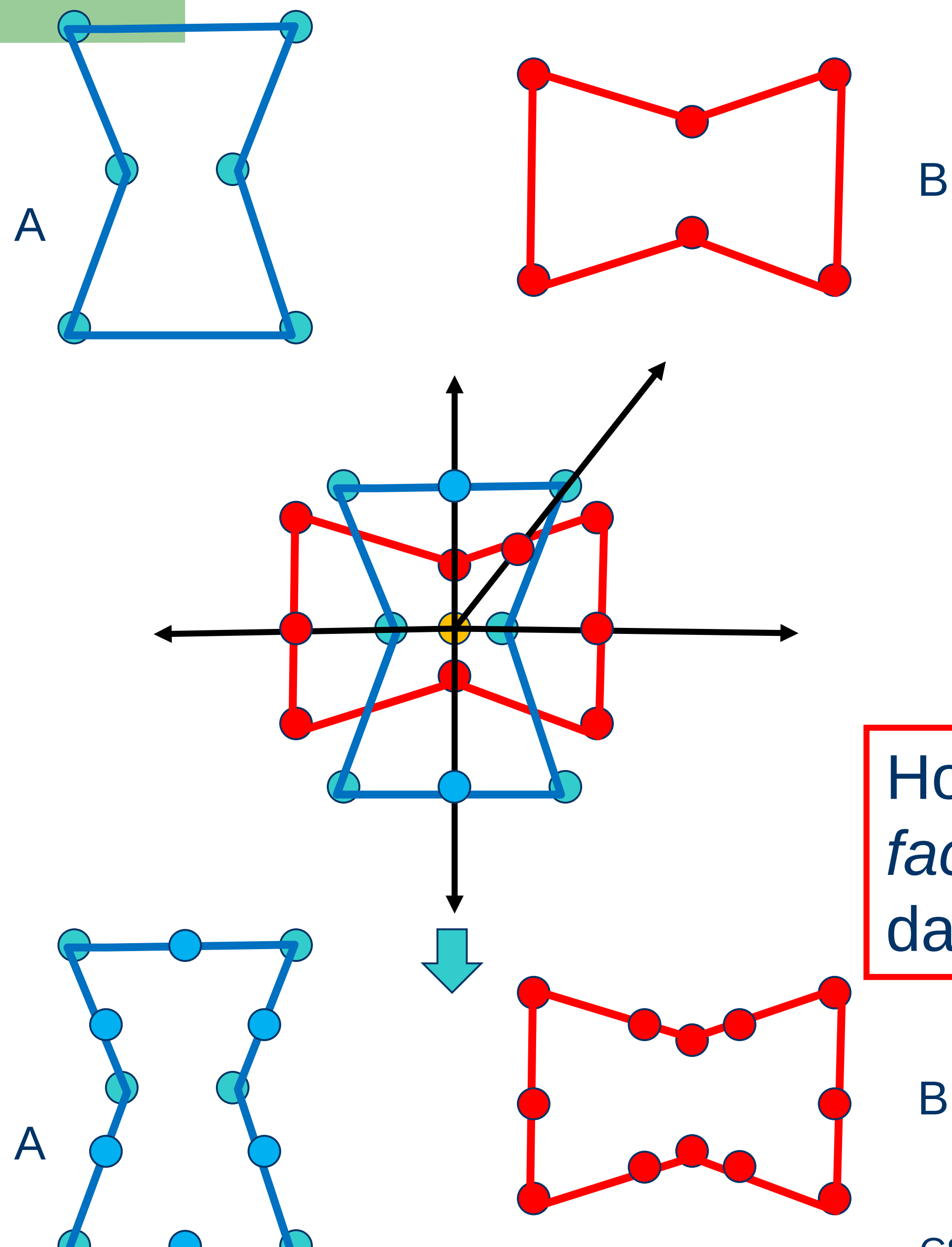
Matching Topology

- only the interpolation step is needed
- how do you tell if two objects have the same topology? (2nd sense)

Star-shaped polyhedra

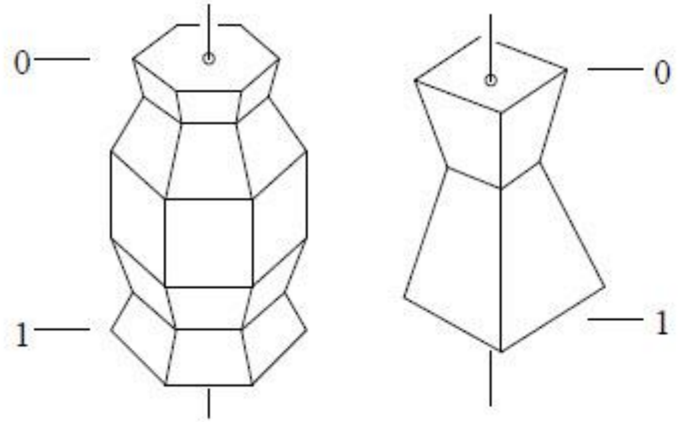


- generating new vertices and edges thru ray emanating from a central point in the kernel of the objects so that the objects would have the same topology

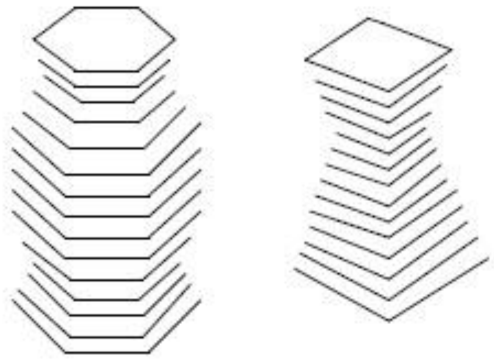


How to build new *face-edge-vertex* data structure?

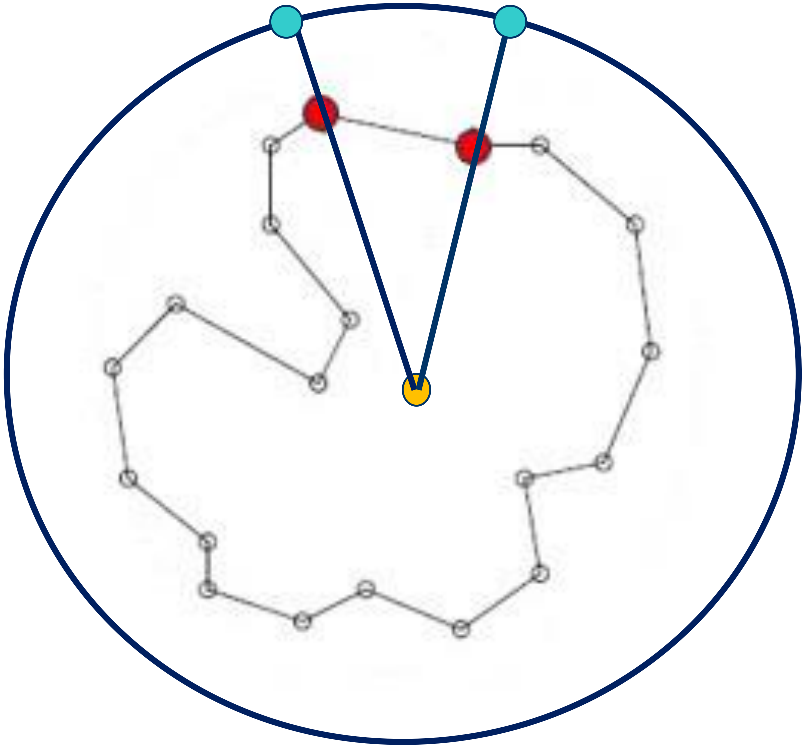
Axial Slices



- object has a central axis and slices of the object with respect to the axis are star shaped
- the axis should be parametrized from 0 to 1
- corresponding slices are interpolated



Map to Sphere



- map both objects onto a sphere
- construct a union of the projected vertex-edge topologies
- map the new vertex-edge topology back onto each original object
- the new models for the objects are transformed by a vertex-by-vertex interpolation

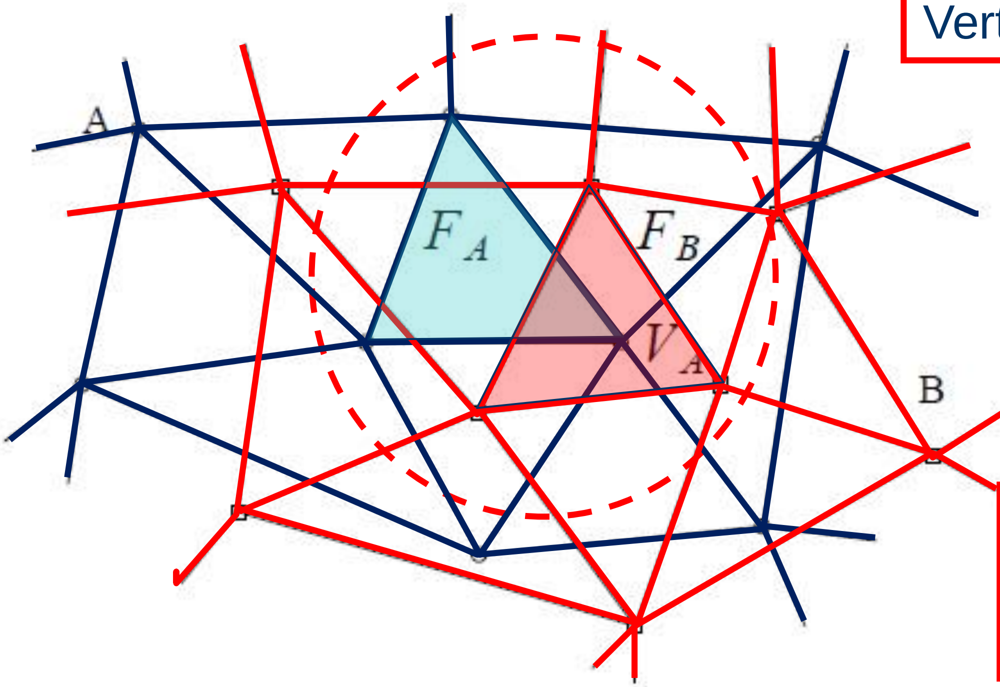
Map to Sphere

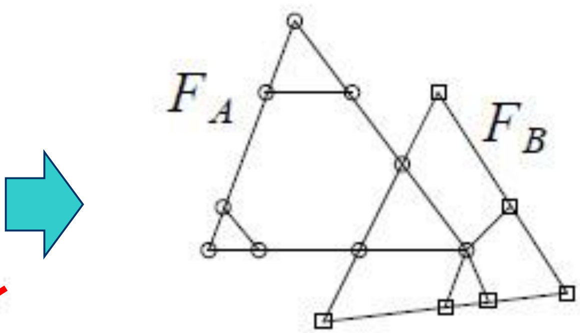
How to build new *face-edge-vertex* data structure?

UNION of the projected vertex-edge topologies

- costly process
- projected edges are intersected and merged into one topology

Face -> Edge -> Vertex ?
Vertex -> Edge -> Face ?





Face-Edge intersection:
to find new edges and vertices
Then form new faces

Recursive subdivision

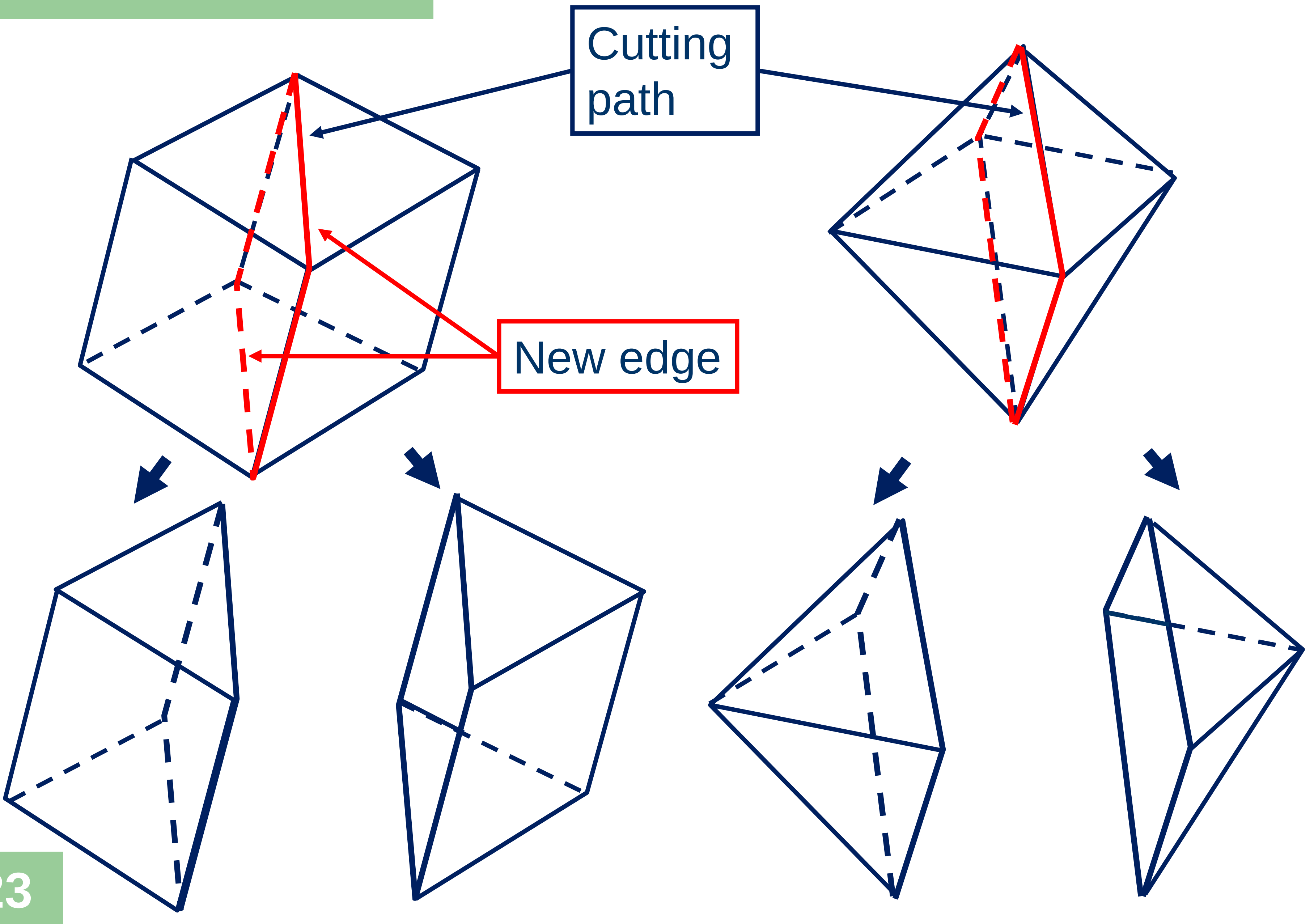
- recursively split the surfaces of the objects into disjoint meshes
- splitting is done by selecting appropriate splitting paths (sometime needs to add new edges)
- corresponding meshes must maintain adjacency relationship and have the same boundary topology (sometime needs to add new vertices)
- recursive subdivision stops when all the meshes have been reduced to triangles
- at that point both objects have the same topology
- then perform vertex-to-vertex interpolation of vertices to carry out object transformation

Recursive subdivision

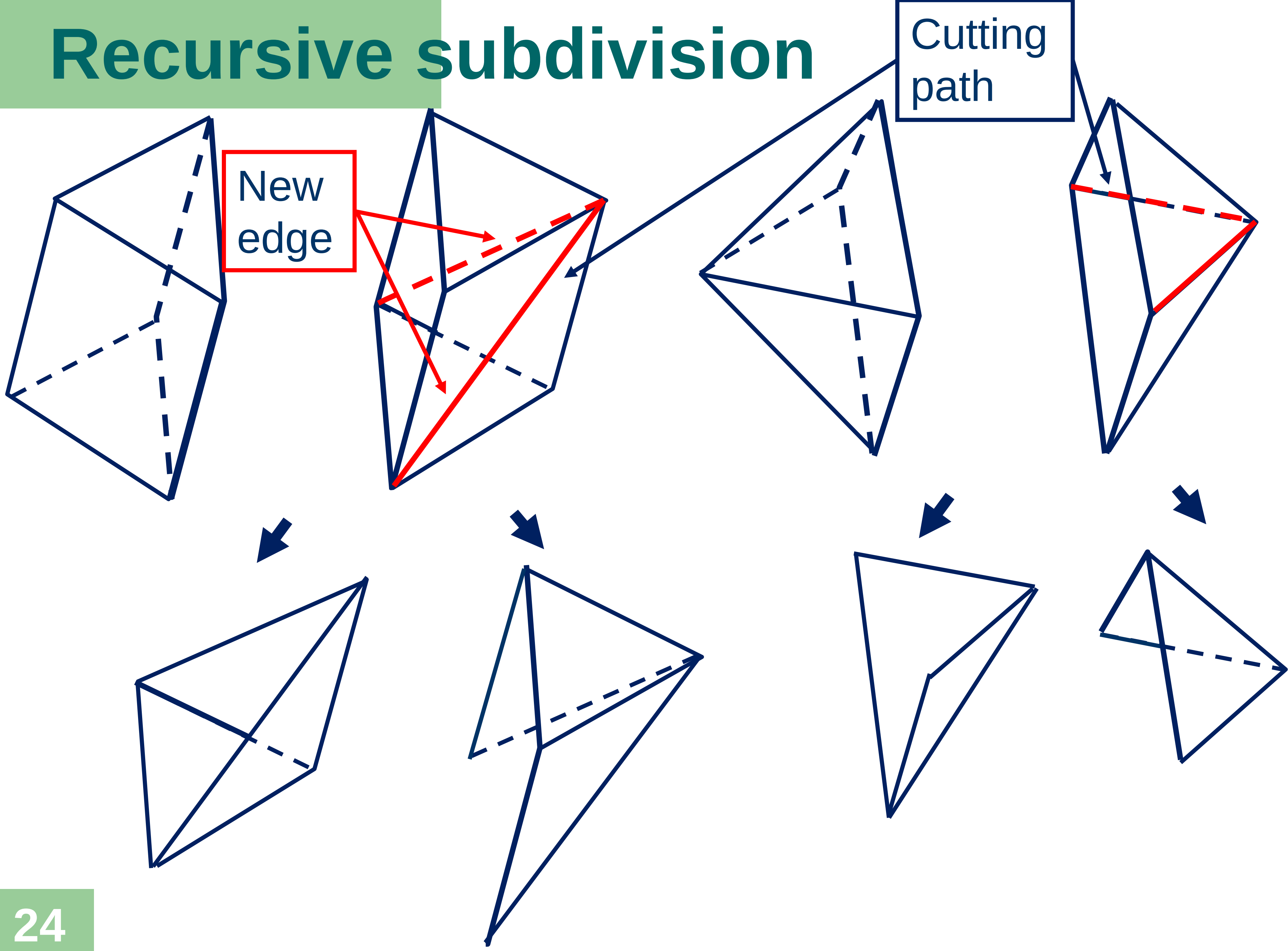
Key Idea:

- break each object into two meshes (using the shortest paths between the topmost, bottommost, leftmost and rightmost vertices)
- add new edges to the cutting path if necessary
- building a one-to-one correspondence between vertices of the two cutting paths (add vertices if necessary

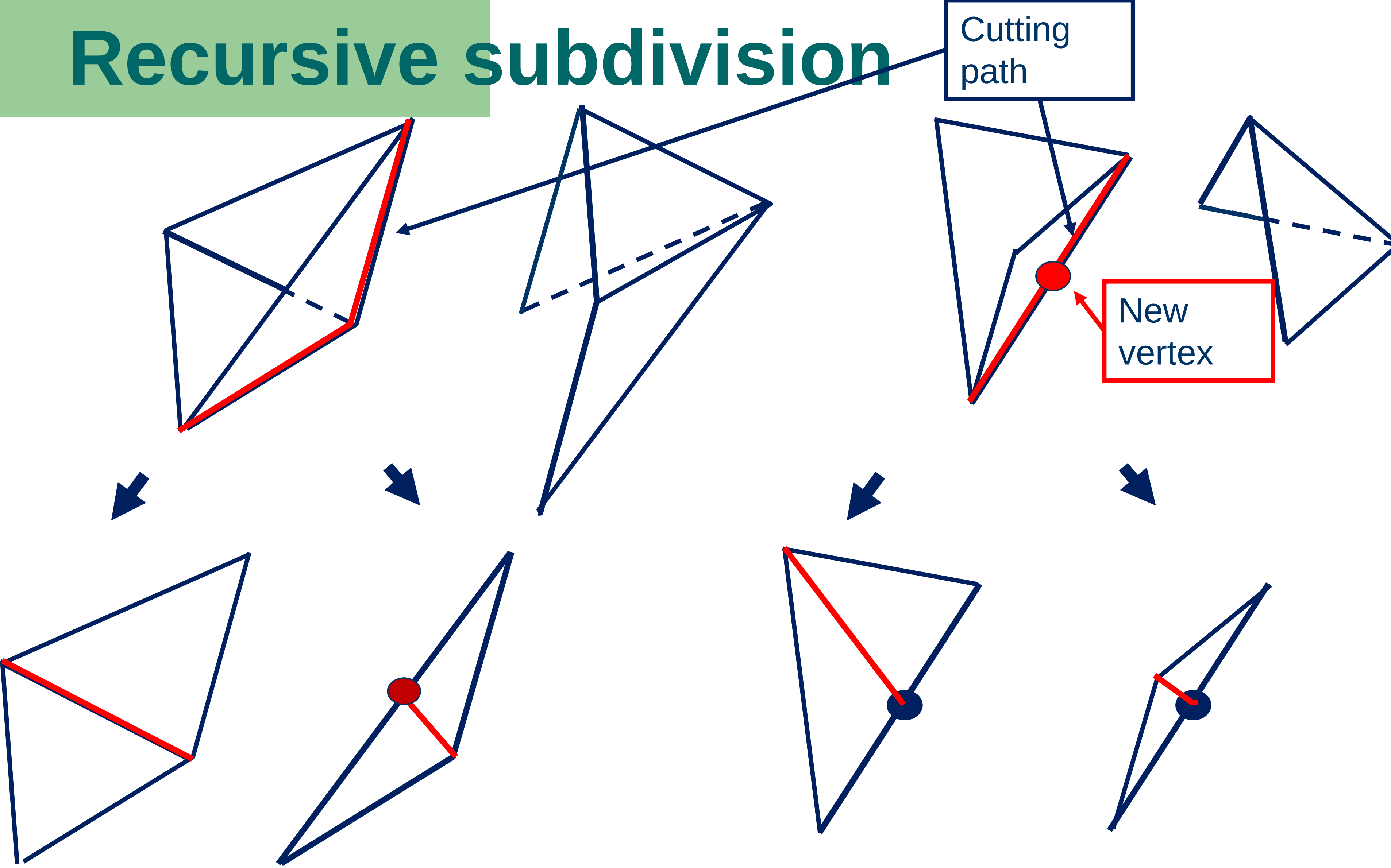
Recursive subdivision



Recursive subdivision



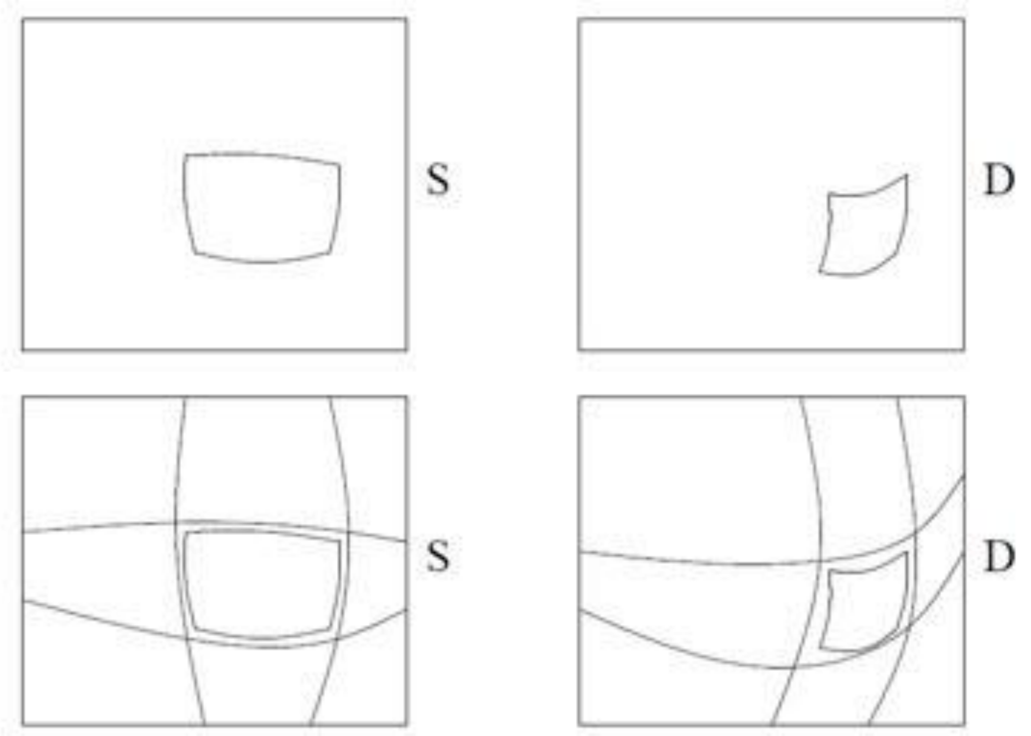
Recursive subdivision



4.5 Morphing (2D): transforming one image (*source image*) into another image (*destination image*)

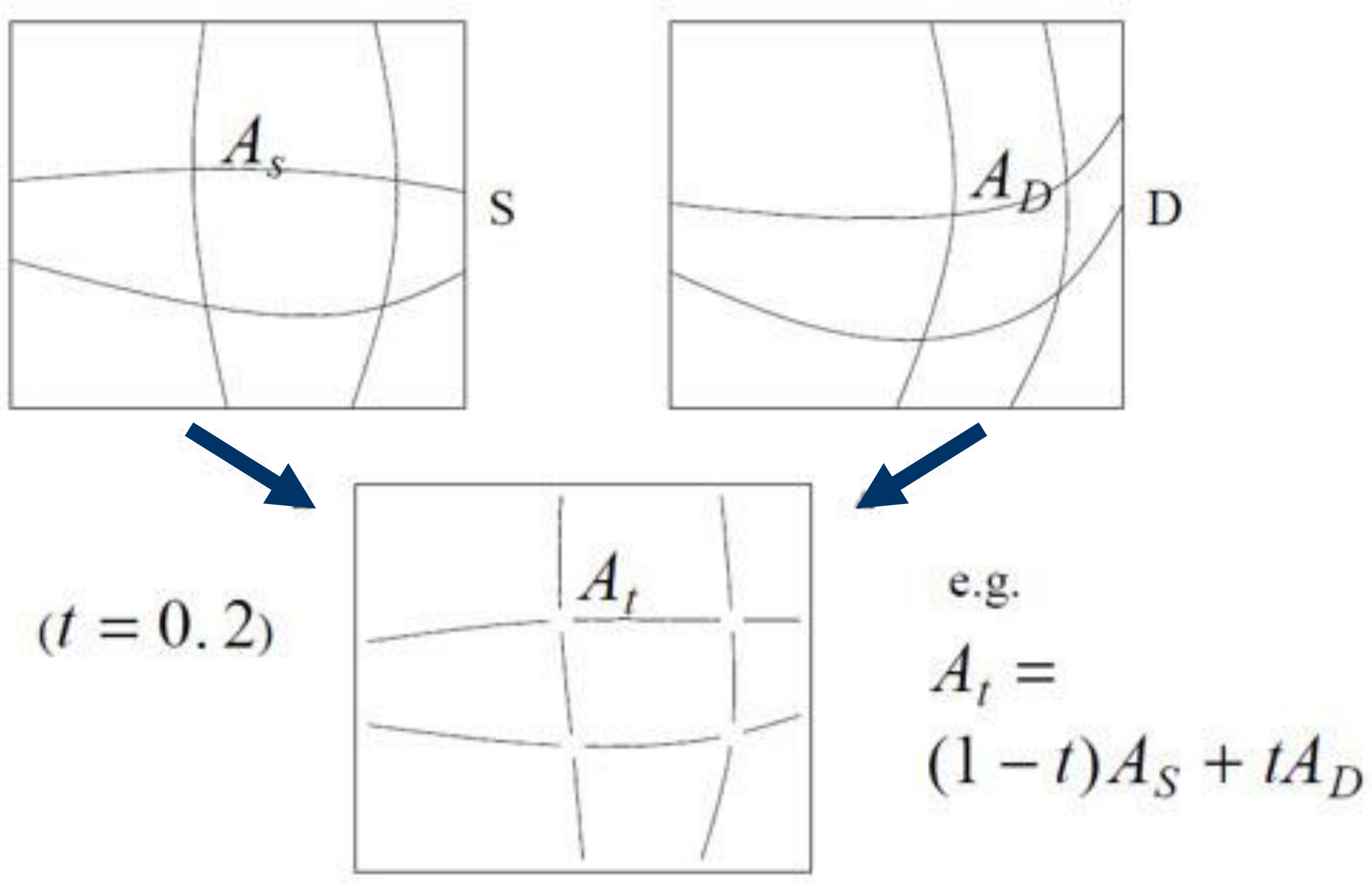
Coordinate Grid Approach

- based on user-defined coordinate grids super-imposed on each image
- corresponding elements must be in corresponding cells of the grids



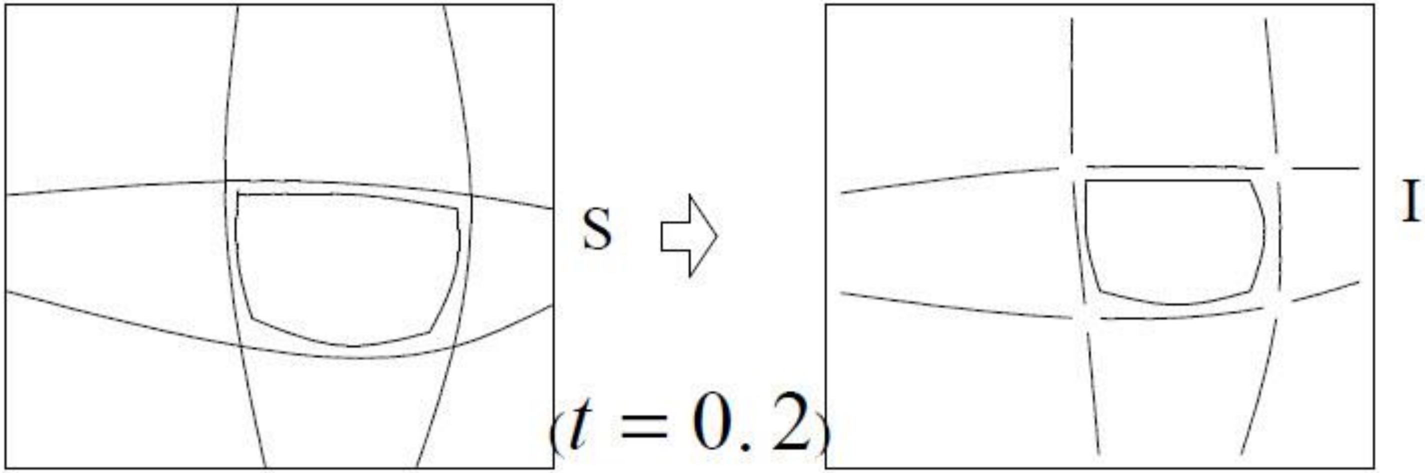
Generation of intermediate images:

1. For given 0 < *t* < *1*, generate an intermediate grid

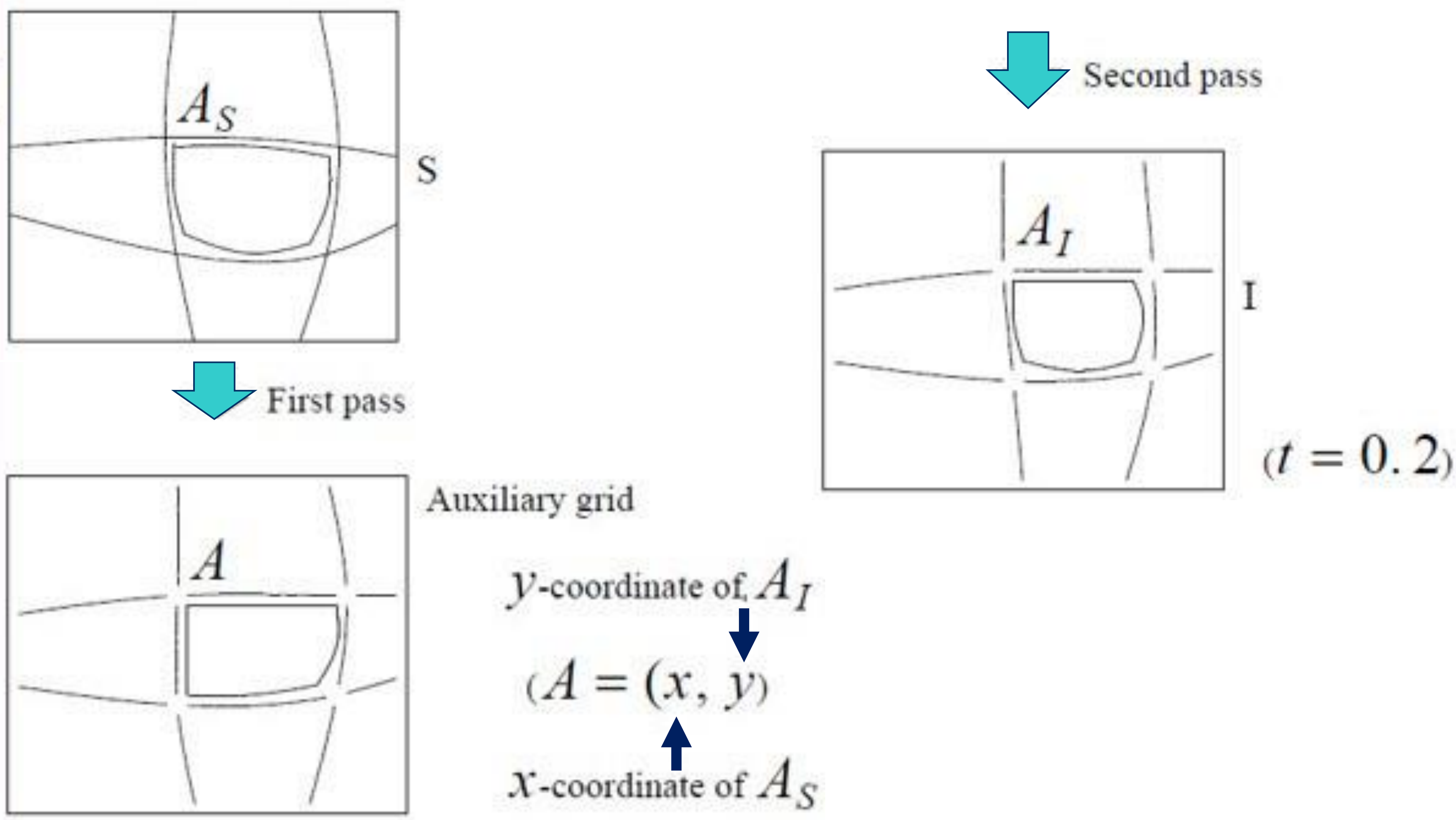


Generation of intermediate images (*conti*):

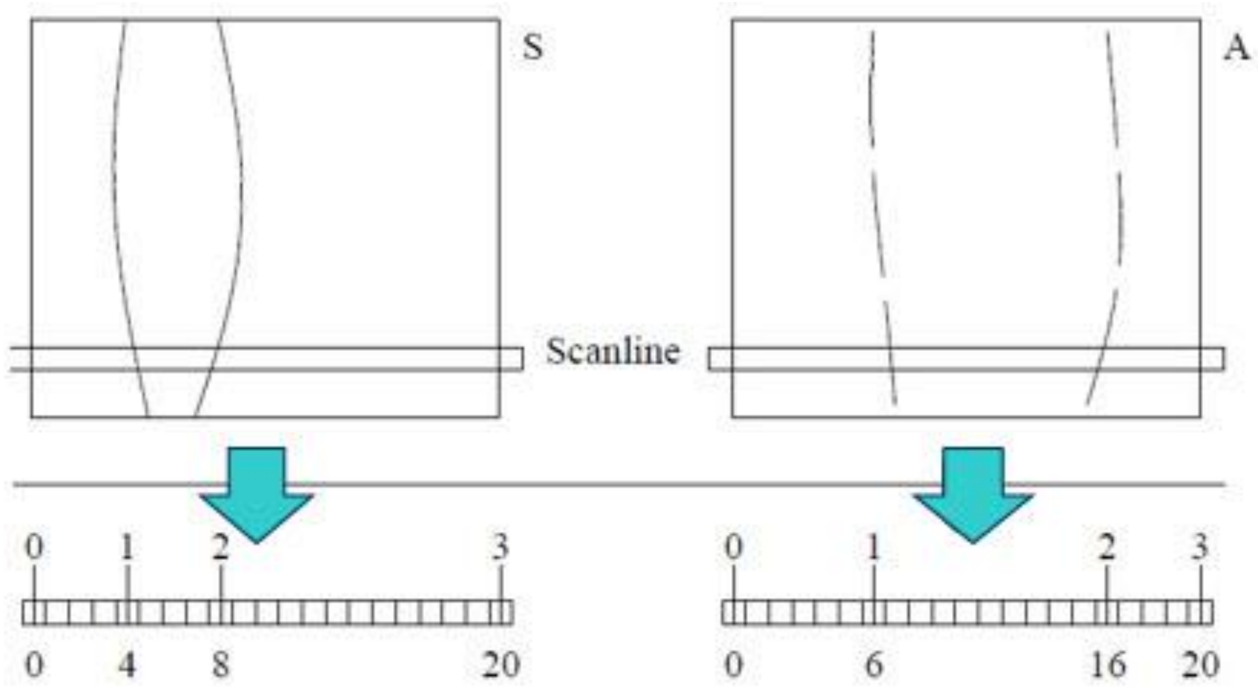
2. Source image is warped according to the intermediate grid in a two-pass process



Two-pass process:



First pass: distort source image in *x*-direction

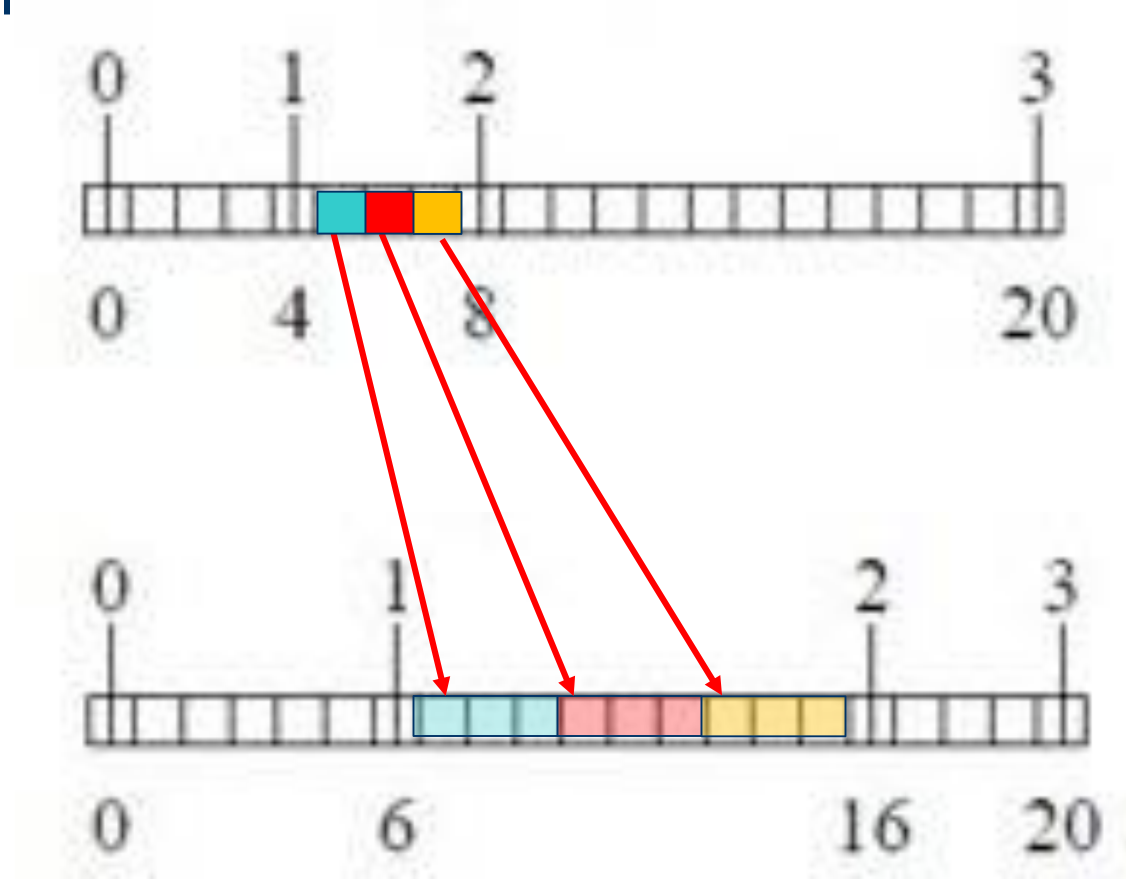


(i) For each scan line, x-intercepts of the vertical grid curves with the scan line are computed

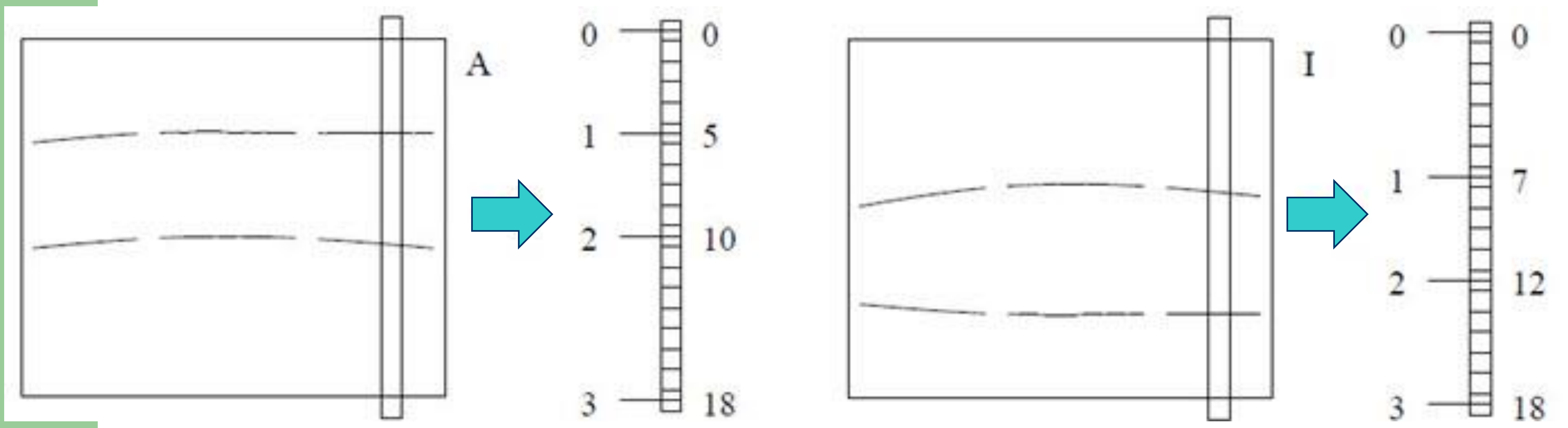
(ii) Use relative position of each pixel on the scanline in the auxiliary image to determine which portion of the scanline in the source image should be used to color the pixel

First pass: distort source image in *x*-direction

(ii) Use relative position of each pixel on the scanline in the auxiliary image to determine which portion of the scanline in the source image should be used to color the pixel



Second pass: distort auxiliary image in *y*-direction

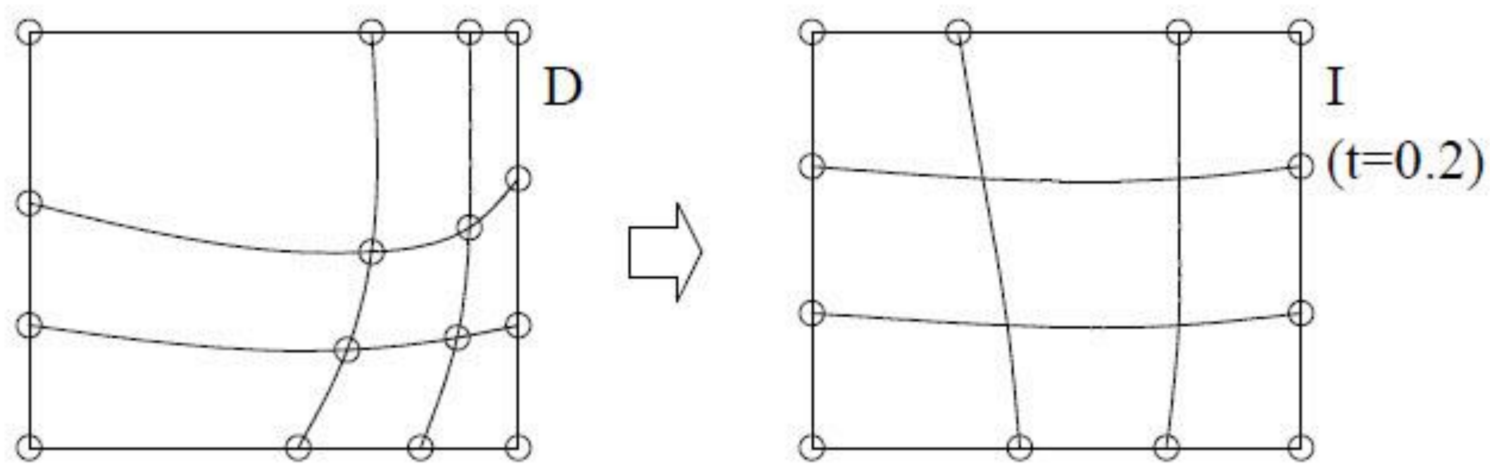


(i) For each column line, y-intercepts of the horizontal grid curves with the column line are computed

(ii) Averaging auxiliary image pixel colors to form the intermediate image using information from (i)

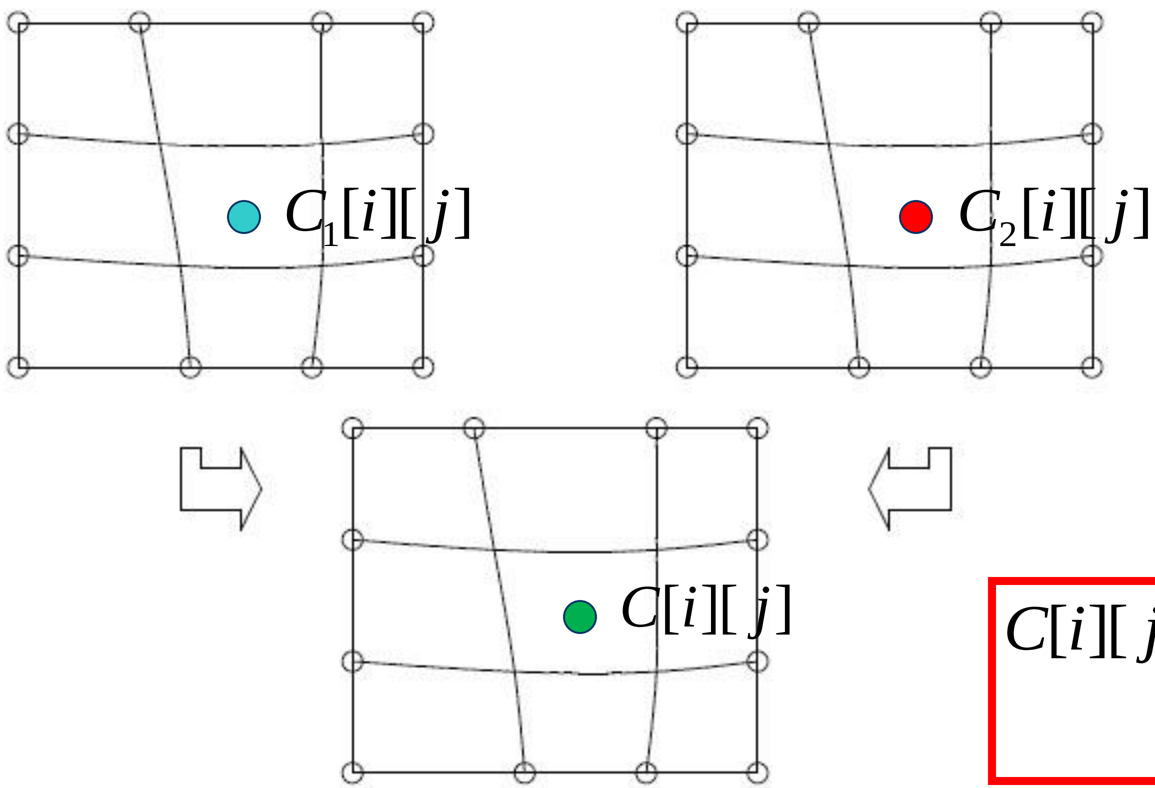
Generation of intermediate images (*conti*):

3. Destination image is also warped according to the intermediate grid in a two-pass process



Generation of intermediate images (*conti*):

4. Performing *cross-dissolve* on a pixel-by-pixel basis between the two warped images to generate the final image



α is a function of the current frame number and the range of the frame numbers over which the morph is to take place

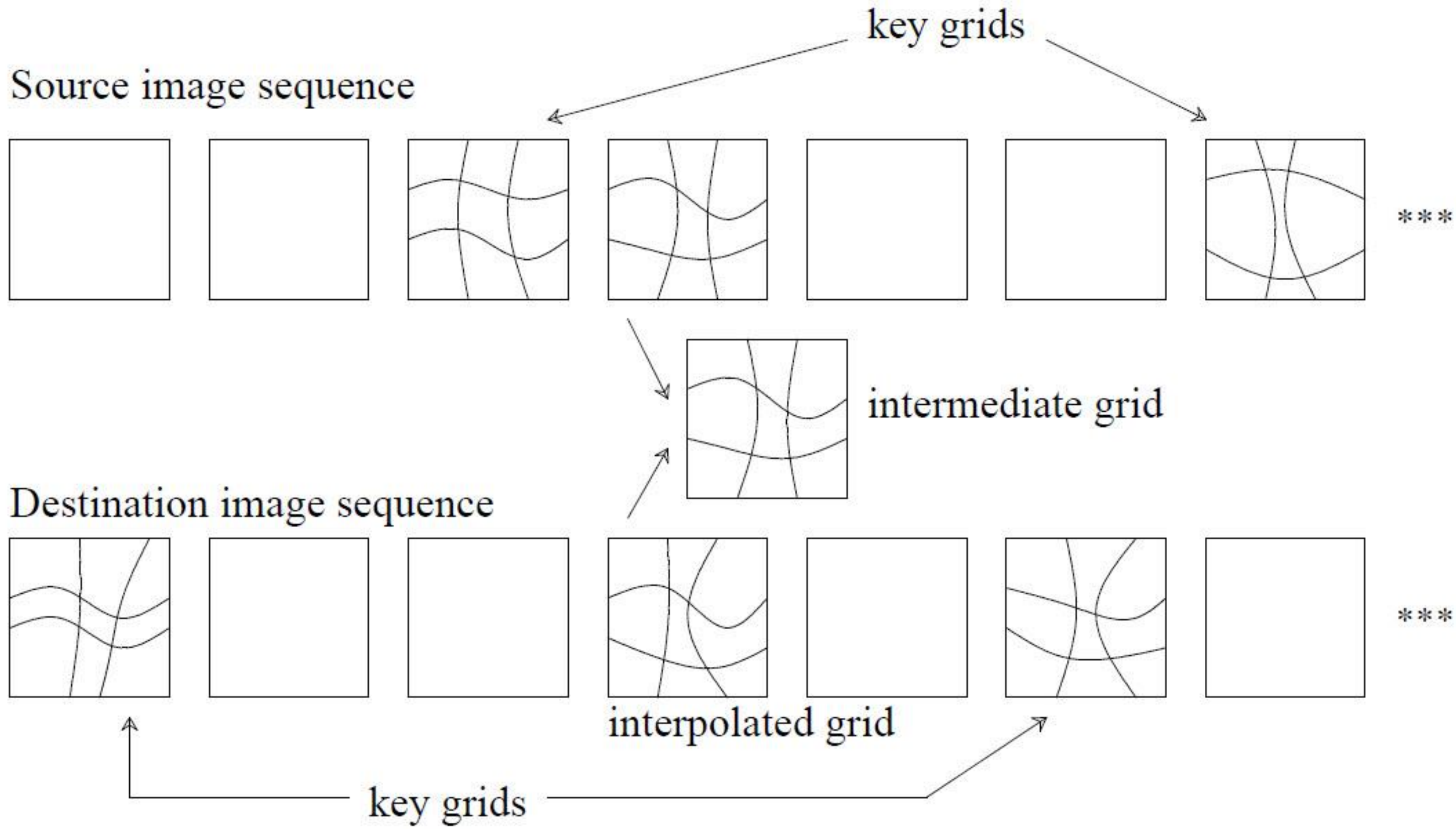
$$C[i][j] = \alpha C_1[i][j] + (1-\alpha) C_2[i][j]$$

Generation of intermediate images (*conti*):

4. Performing *cross-dissolve* on a pixel-by-pixel basis between the two warped images to generate the final image

Coordinate Grid Approach examples

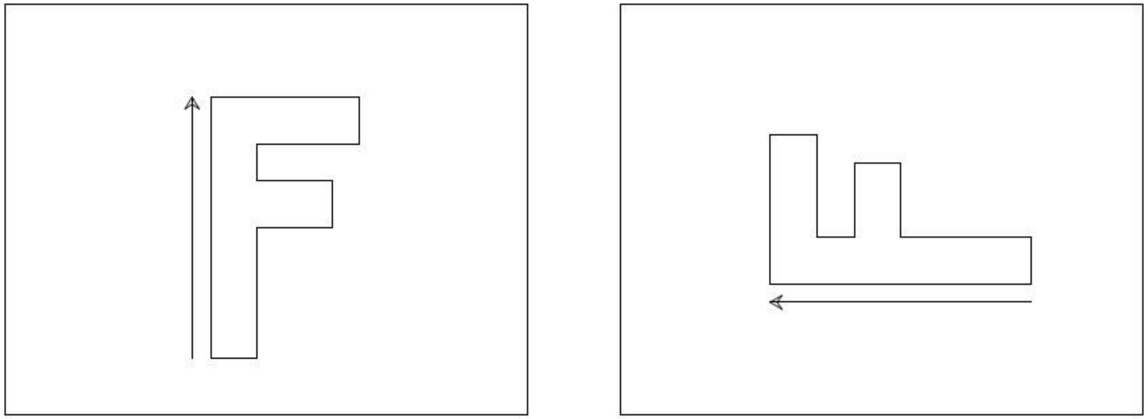
Morphing of Animated Images:



1) define coordinate grids for key images in each sequence
2) generate interpolated grid for each frame
3) perform static morphing on corresponding frames

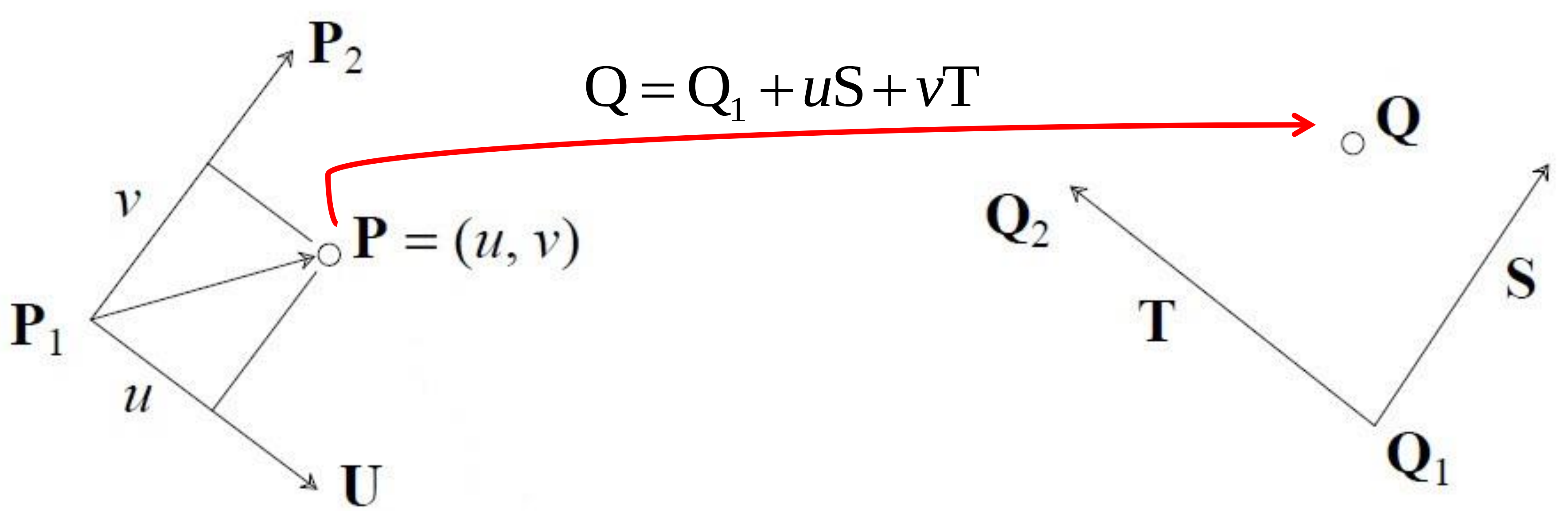
Feature-based Morphing

- correspondence between images are established by using feature lines



- feature lines are interpolated to form intermediate feature lines (based on interpolation endpoints or center points and orientation)
- a *mapping* is used on the source image to form an intermediate image, and on the destination image to form its intermediate image as well
- the two intermediate images are *cross-dissolved* to form the final intermediate image

Morphing defined by a single feature line:

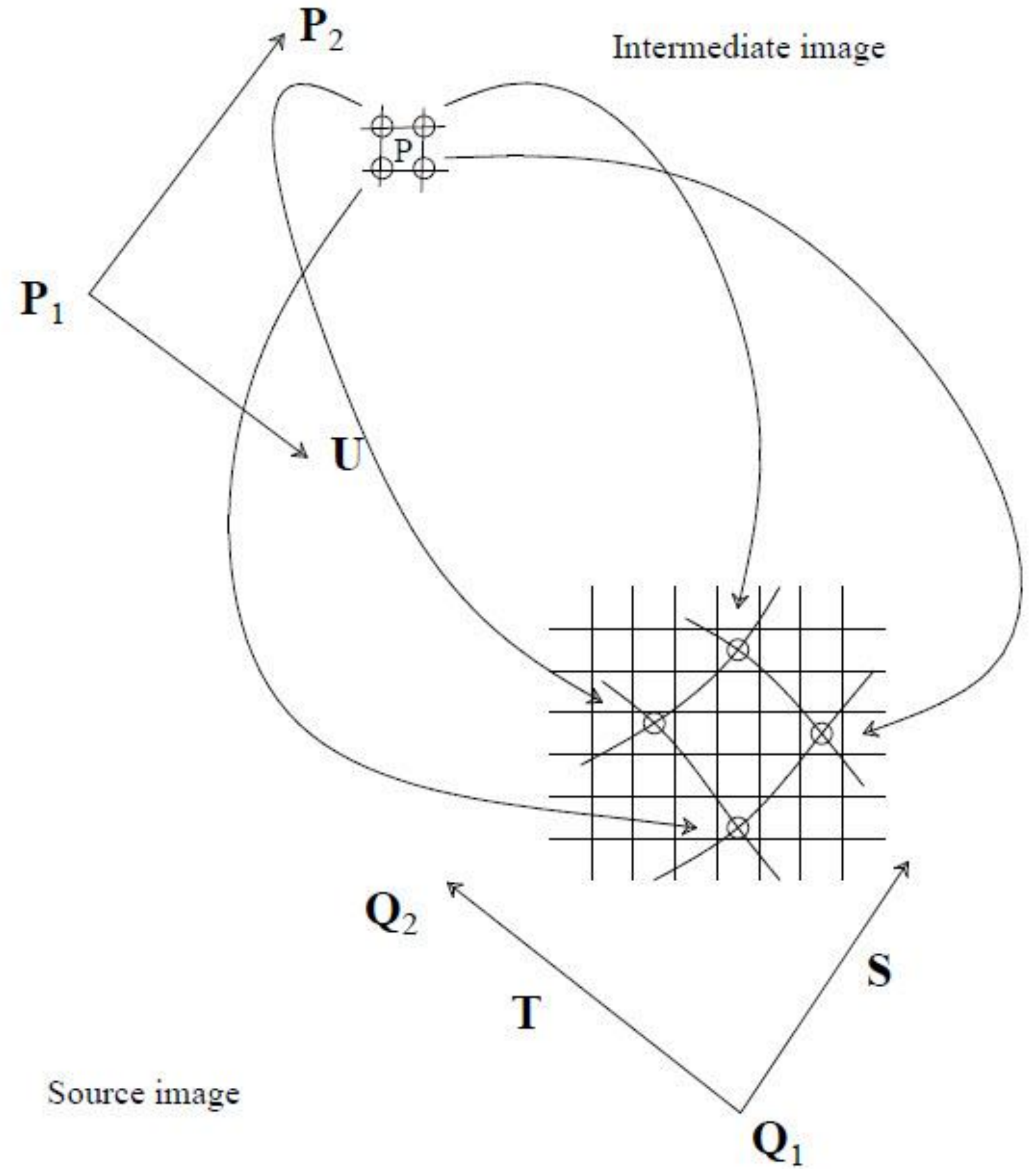


P_1P_2 : feature line in
an inermediate image

Q_1Q_2 : feature line
in source image

The intensity values of **Q** in source image are used to color the pixel **P** in the intermediate image

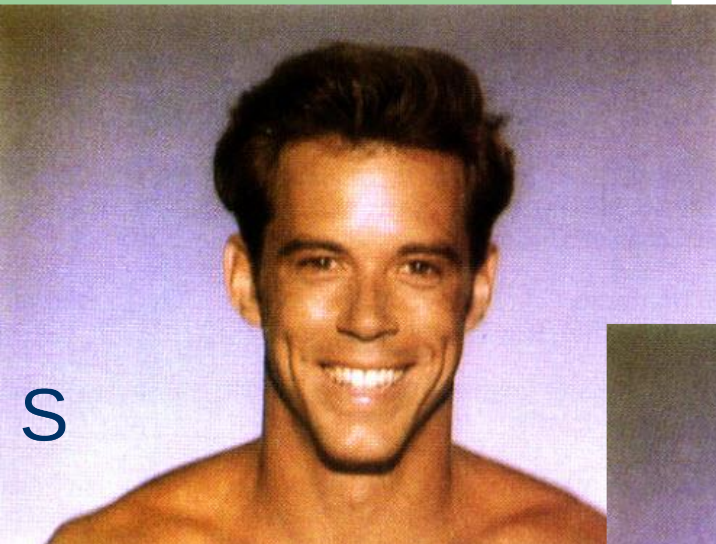
A better approach:

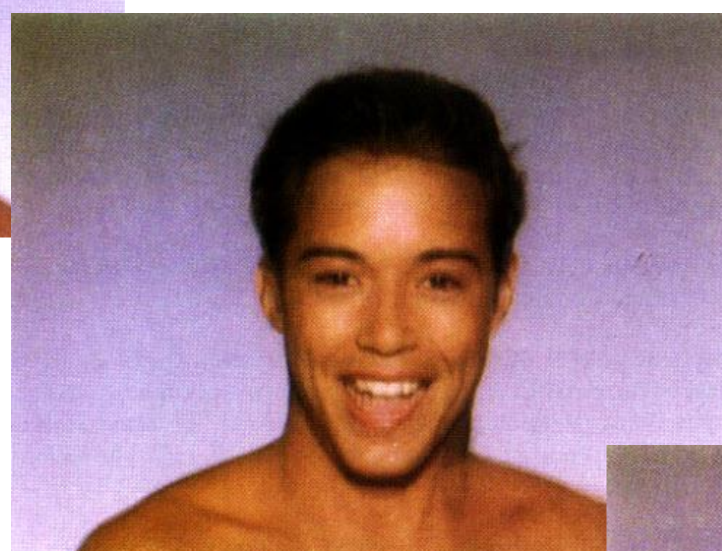


Mapping defined by multiple feature lines:

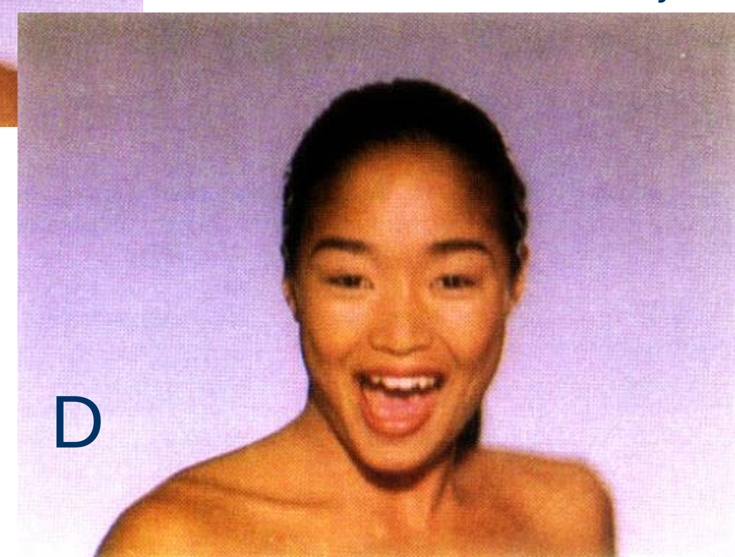
- for each interpolated feature line, a mapping like the one described above is created
- a relative weight is computed that indicates the amount of influence that feature line should have for each intermediate image pixel
- the mapping is used in the source image to identify the corresponding pixel for a pixel in the intermediate image
- the relative weights are used to average the source image locations generated by multiple line features into a final source image location
- this location is used to color the intermediate image pixel

Feature-based Morphing Example

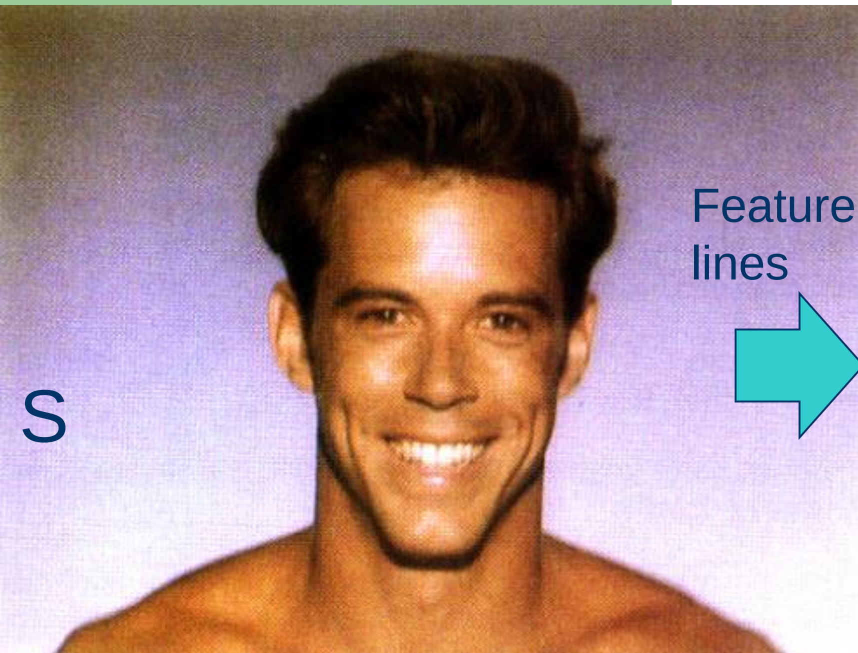




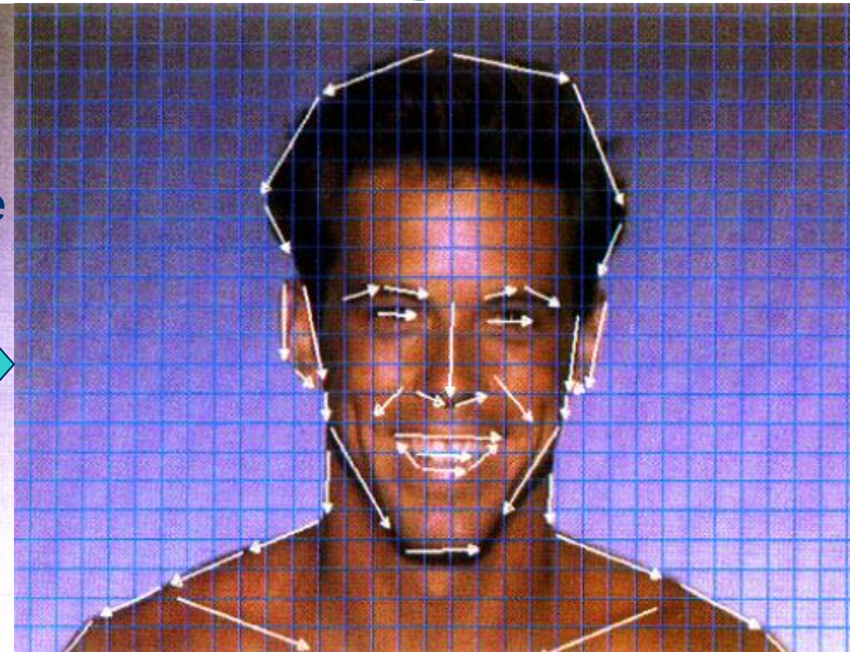
Beier/Neely

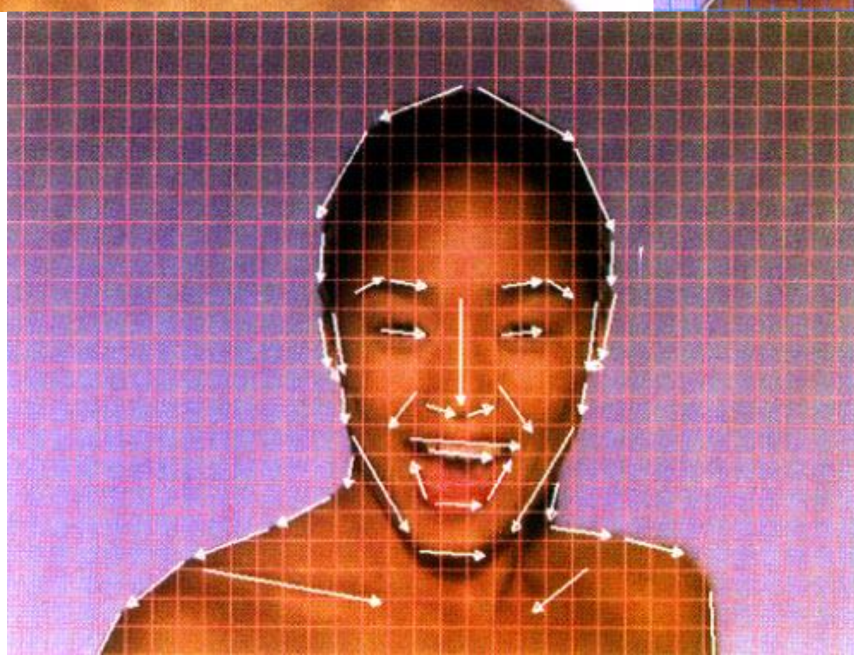


Feature-based Morphing Example

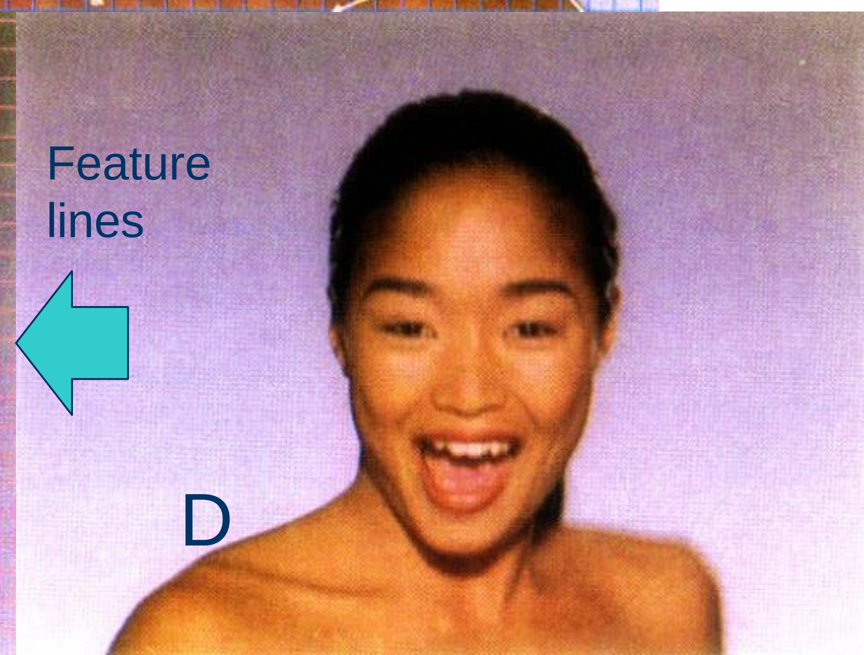


Feature lines

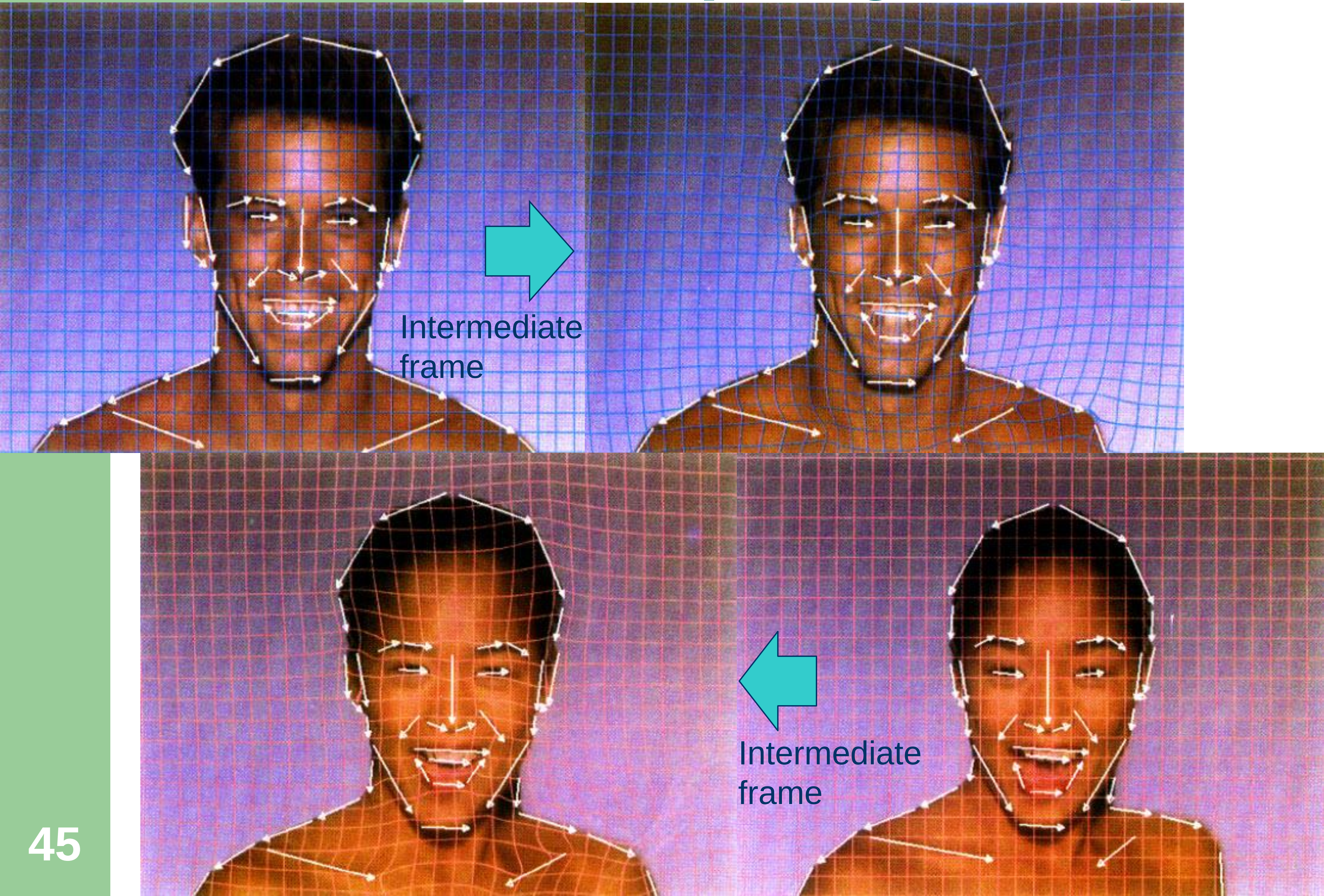




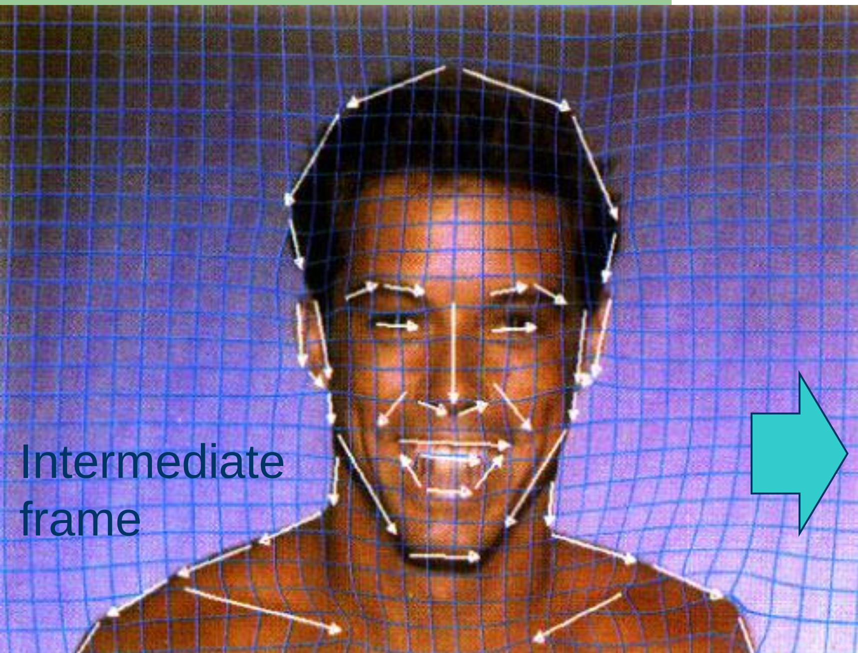
Feature lines

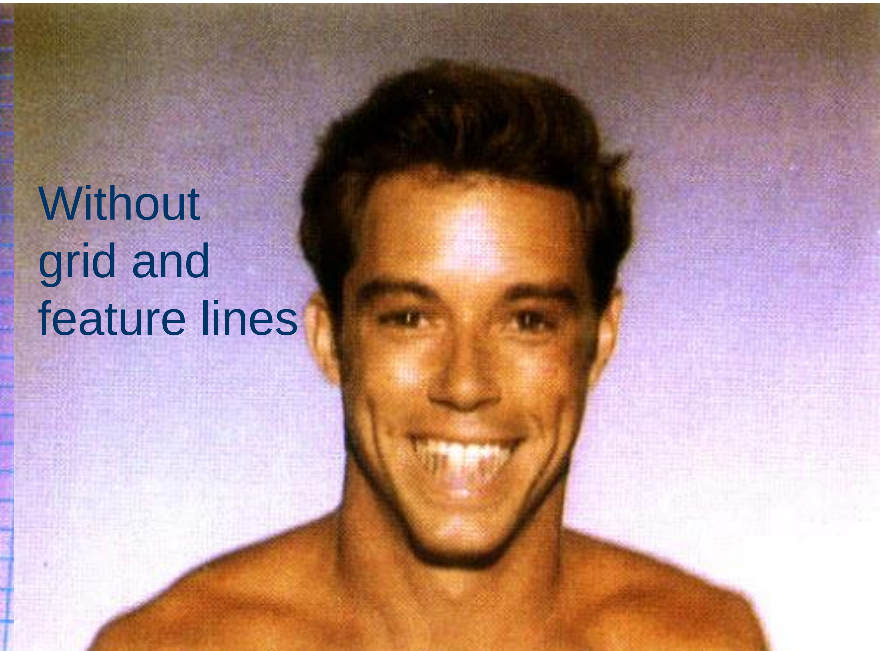


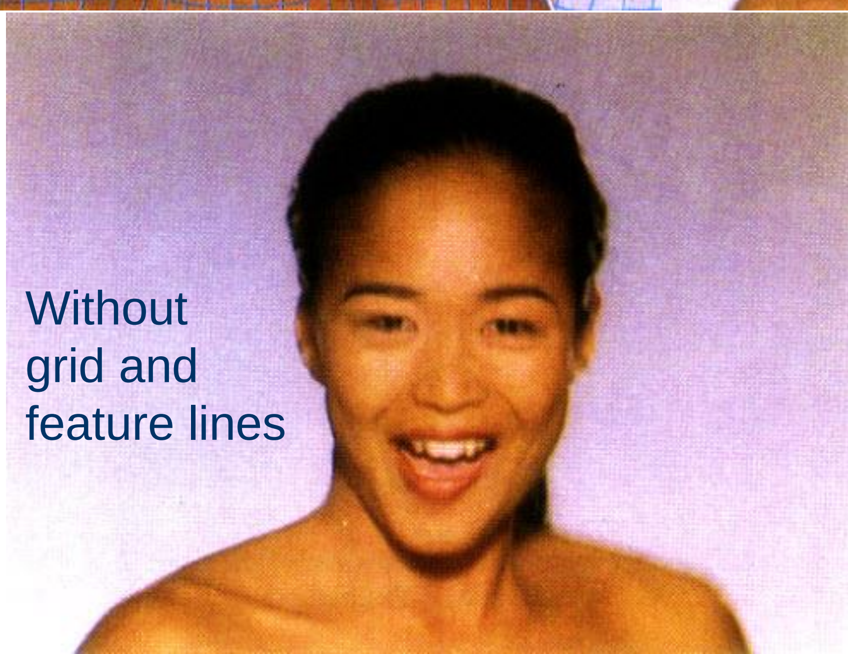
Feature-based Morphing Example

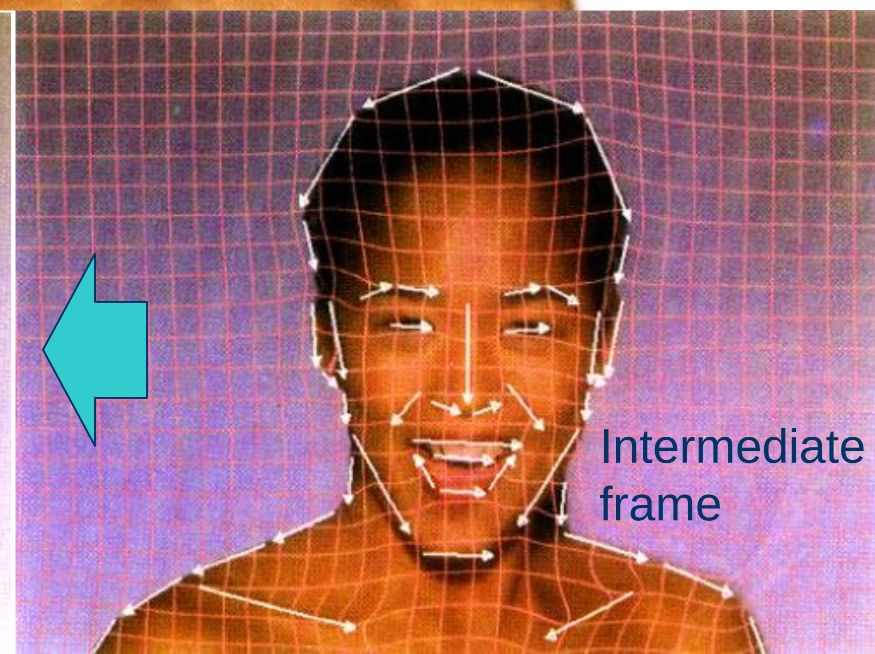


Feature-based Morphing Example

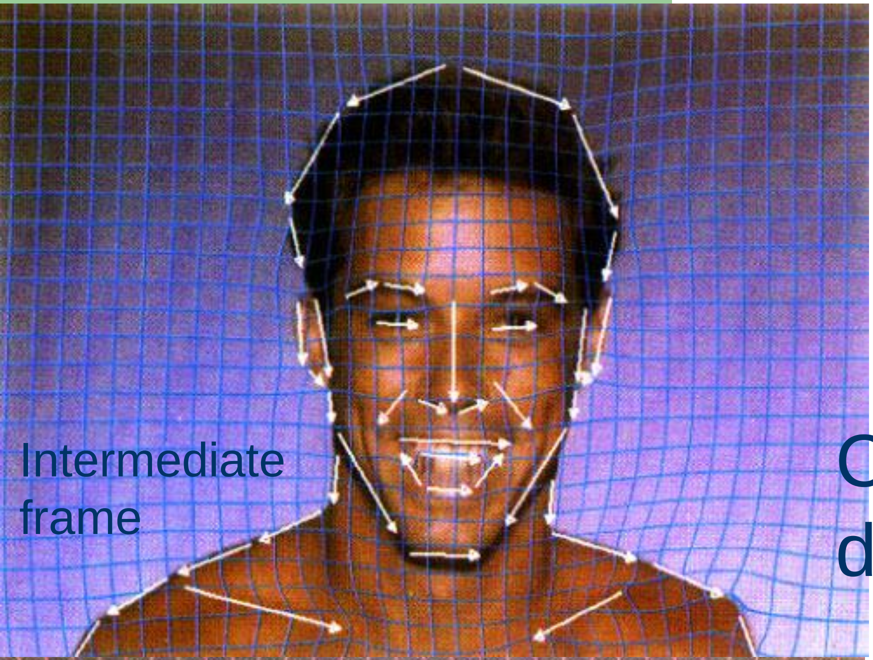




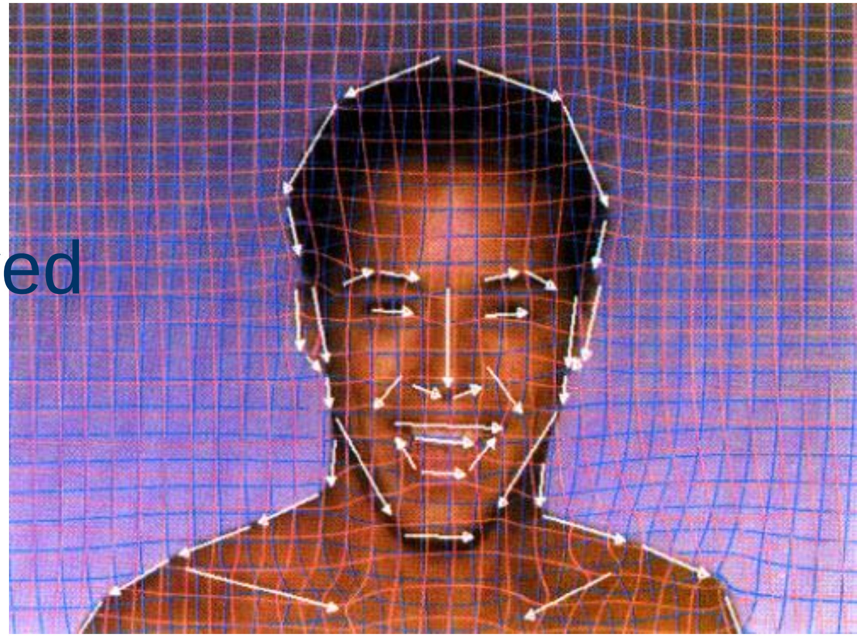




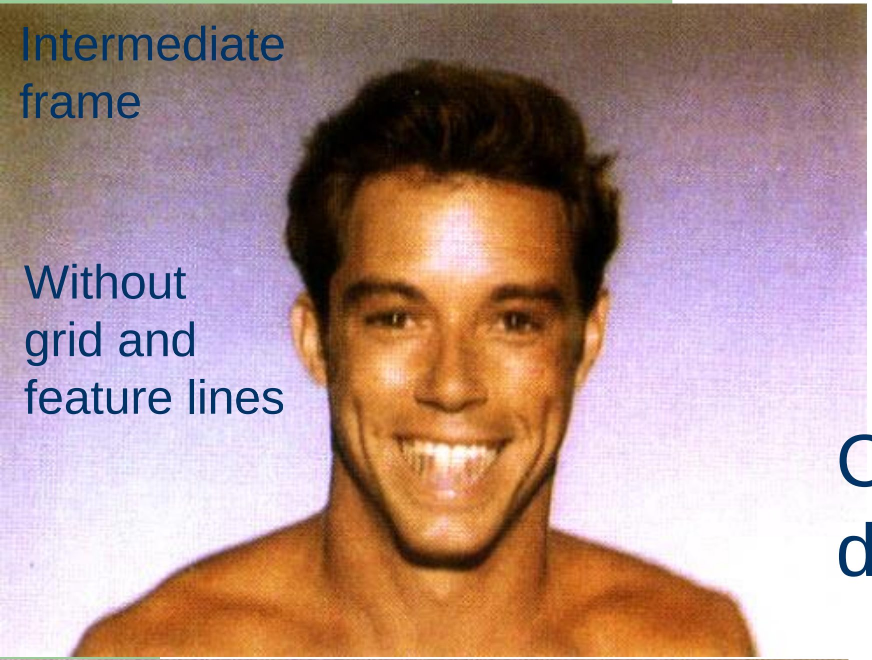
Feature-based Morphing Example

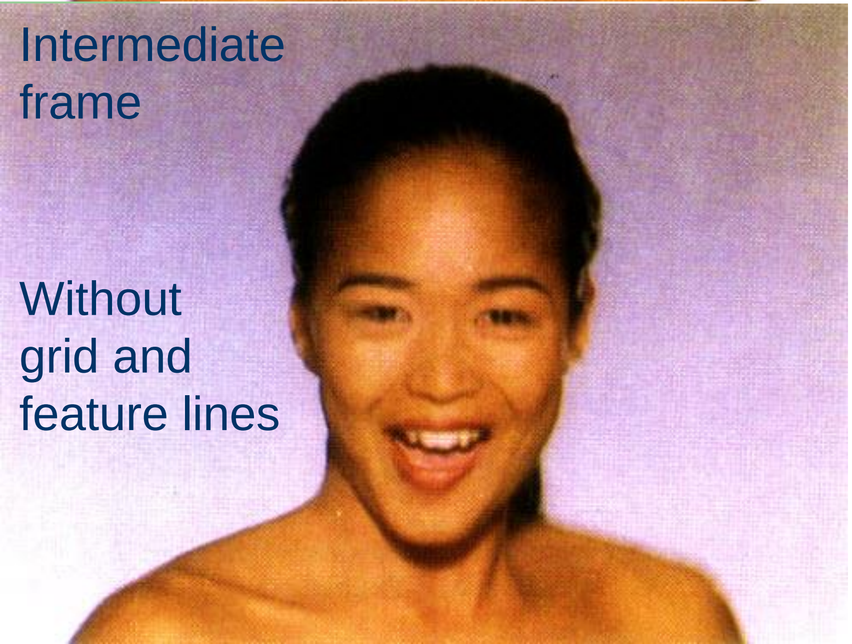


Cross-dissolved

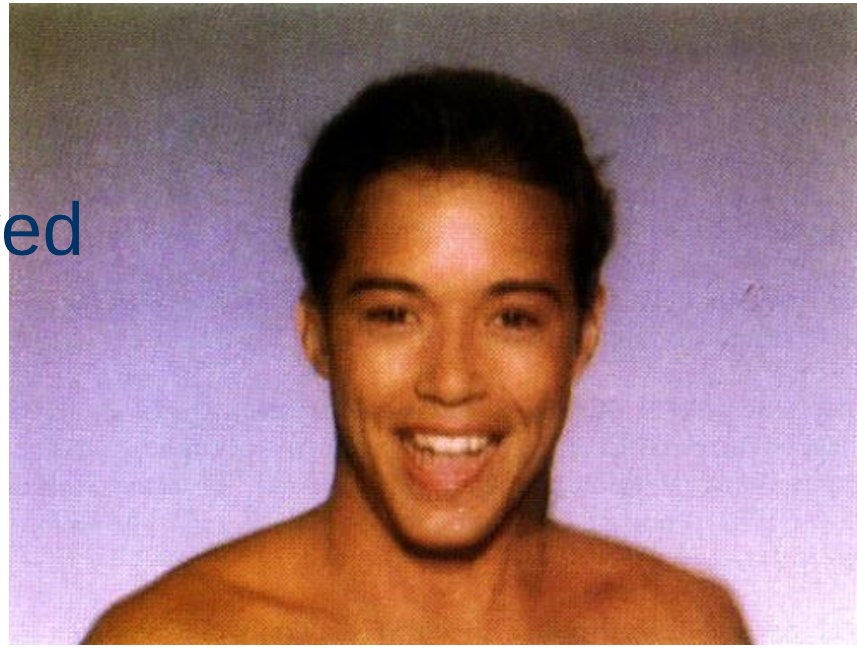


Feature-based Morphing Example





Cross-dissolved



End of Interpolation V