4.4 3D Shape Interpolation
- changing one 3D object into another

Two categories:

- surface based approach (2nd case)
- volume based approach (1st 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:

- **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? (the other direction?)

**Answer**: NO

- Donut and teacup have the same topology (one hole, one component)
Shape Representation: a review

A representation of a 3D object:
Shape Representation: a review

An alternative:

Face Table → Edge Index Table

Edge Table → Vertex Table

Edge Index Table
Shape Representation: a review

An alternative:

Face Table

<table>
<thead>
<tr>
<th># of edges</th>
<th># of edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

Edge Table

<table>
<thead>
<tr>
<th>Index Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Vertex Table

<table>
<thead>
<tr>
<th>E0</th>
<th>V0</th>
<th>V1</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shape Representation: a review

An alternative:

Face Table

<table>
<thead>
<tr>
<th># of edges</th>
<th># of edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

Edge Table

<table>
<thead>
<tr>
<th>Index Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Vertex Table

<table>
<thead>
<tr>
<th>E0</th>
<th>V0</th>
<th>V1</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shape Representation: a review

An alternative:

Edge Table

Vertex Table

Edge Index Table

CS Dept, Univ of Kentucky
Shape Representation: a review

An alternative:

Vertex Table

<table>
<thead>
<tr>
<th>V_0</th>
<th>X_0</th>
<th>Y_0</th>
<th>Z_0</th>
<th># of degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_1</td>
<td>X_1</td>
<td>Y_1</td>
<td>Z_1</td>
<td># of degree</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Edge Index Table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>3</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Edge Table

CS Dept, Univ of Kentucky
Winged-Edge Data Structure: a review

- To avoid using variable-length data structures
- Hide the implementation behind a class interface
Winged-Edge Data Structure: a review

Example: representing a tetrahedron

Edge Table

<table>
<thead>
<tr>
<th>Edge Name</th>
<th>Vertices</th>
<th>Faces</th>
<th>Clockwise</th>
<th>Counter-Clockwise</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>from</td>
<td>to</td>
<td>left</td>
<td>right</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>3</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>f</td>
<td>3</td>
<td>1</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>4</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>4</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>4</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>
Winged-Edge Data Structure: a review

Example: representing a tetrahedron

Vertex-Edge Table

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
</tr>
</tbody>
</table>

Face-Edge Table

<table>
<thead>
<tr>
<th>Face</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>d</td>
</tr>
<tr>
<td>B</td>
<td>e</td>
</tr>
<tr>
<td>C</td>
<td>d</td>
</tr>
<tr>
<td>D</td>
<td>b</td>
</tr>
</tbody>
</table>
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 $v$?

  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 $v$?

Edge Table (to find $D$ and $e$)

Edge Table (to find $C$ and $d$)

Edge Table (to find $A$ and $c$, stop)
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

UNION of the projected vertex-edge topologies
- costly process
- projected edges are intersected and merged into one topology

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

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)

Cutting path
New edge
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

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)

Cutting path

New vertex
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

\[ A_t = (1-t)A_s + tA_D \]
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

Auxiliary grid

Second pass

$y$-coordinate of $A_I$

$(A = (x, y))$

$x$-coordinate of $A_S$
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 (cont.):

3. Destination image is also warped according to the intermediate grid in a two-pass process
Generation of intermediate images (cont.):

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

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

\( \alpha \) is a function of the current frame number and the range of the frame numbers over which the morph is to take place.
Generation of intermediate images (cont.):

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:

\[ Q = Q_1 + uS + vT \]

\[ P_1P_2 \text{: feature line in an intermediate image} \]
\[ Q_1Q_2 \text{: feature line in source image} \]

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

S

D

Beier/Neely
Feature-based Morphing Example

Feature lines

S

Feature lines

D
Feature-based Morphing Example
Feature-based Morphing Example

Intermediate frame

Without grid and feature lines

Intermediate frame

Without grid and feature lines

Without grid and feature lines

Intermediate frame
Feature-based Morphing Example

Intermediate frame

Cross-dissolved

Intermediate frame

CS Dept, UK
Feature-based Morphing Example

Without grid and feature lines

Cross-dissolved

Intermediate frame

Intermediate frame

Without grid and feature lines

CS Dept, UK
End of Interpolation V