Jan. 14, 2006

Continuation of Compression Slides from Feb. 9, 2006

Note:
- *The notes accompany the slides*
- *The in-between slides that I didn't take notes on are either data/results or self-explanatory*

Review of a few slides starting with **Euler formula for Simple Meshes (15)**
- Strips and corridor representations
- Review of problem with "warts"
- Not always efficient for storage because vertices stored multiple times

3Tb encoding of VST and TST suffices (22)
- Send vertices in same order
- Decoder will rename the vertices
- But when a decoder requests a vertex, how will encoder know which vertex to send? Encoder and decoder's vertex numberings can be different
    - since server has original, and knows in what order he is sending them, server can do one of the following:
        - renumber original vertices to match the decoder's
        - simply send dictionary to decoder
- To number the vertices on the VST, "walk around" and number each new vertex with the next number
- Compression is guaranteed 3 bits/Triangle or 3Tb
- Binary tree can be encoded with 2b/Triangle
    - Normal binary tree can be encoded with only 1b/node, because each parent is guaranteed to have 2 children(left, right)
    - However, T-tree needs 2 bits, because a parent is allowed to have either 1 or 2 children. If there is only 1 child, then we need to encode if it's either left or right.
- VST can also be encoded with only 2 bits
    - The order of the children is important
    - Each parent can have multiple children
    - To encode:
        - First bit: if parent or not
        - Second bit: if child, then if last child or not

Run-length encoding of TST and VST (24)
- Many triangles have a unique, single child
- We can encode the number of children in a chain of unique children.
    - Example 1: Parent -> 1 child -> 1 child -> 1 child is a chain of unique children and has a length of 3
    - Example 2: Parent -> 1 child -> 1 child -> 2 children is a chain of only length 2, because the second child has 2 children.
- Works fairly well
- About 1 byte/triangle
- This is good because it combines geometry and connectivity

Edgebreaker is a state machine (32)

- Idea: Enter through 'X' and walk to neighboring triangles and mark in 'Red' the triangles and adjacent vertices we traverse through
- Different cases:
    - Case 1: left and right triangles unmarked and opposite vertex also unmarked.
        - Color center triangle and opposite vertex
        - C = center
    - Case 2: left triangle is marked already
        - Color center triangle and label it L
        - L = left
    - Case 3: right triangle is marked already
        - Color center triangle and label is R
        - R = right
    - Case 4: left and right triangles unmarked, but opposite vertex is already marked
        - This is different from Case 1, because it disconnects the topology.
        - We must go both left and right to keep everything connected
        - Color center triangle and label it S
        - Place left triangle onto a stack so that we can come back to it later
        - S = stack
    - Case 5: Both left and right triangles have already been visited
        - We are at the end of this path.
        - Color center triangle and label it E
        - If there exists any stack of triangles from Case 4, we will visit them now
        - Otherwise, we are done
        - E = end
- The number of C's is actually the number of vertices
- This also means that half of the triangles are of type C, because there are twice as many triangles as there are vertices.
- This will always work if the mesh is simple

EB re-numbering of vertices (34)

- Thick lines are part of the vertex spanning tree

Corner table: data structure for T-meshes (35)

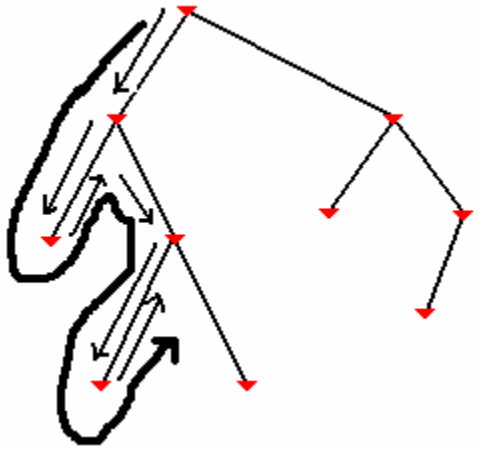
- Stores the connectivity of mesh
- For each triangle, store the vertex ID, and specify the opposite corner.
    - In the example, we look at corner 2.
    - The corresponding vertex ID is 3.
    - The opposite corner is 5, because we see that corner 5 is on vertex ID 4, which is "on the other side" of vertex ID 3 when we reflect over the two triangles' 2 similar points.
    - Therefore, corner 2 is also an opposite corner for corner 5.

Computing adjacency from incidence (36)

- In our table, the opposite corners are not necessarily stored
- So how do we obtain O?
  - First make triplets of (vertex, vertex, corner) for each corner.
  - Since each triangle has 3 corners, each triangle will generate 3 triplets.
  - Next, sort the triplets by the triplet value. Can use bin sorting. Since we know how many bins we'll need, sorting can be done very fast.
  - After sorting, pair up consecutive triplets. It makes sense that the first two values (the vertex, vertex) of the pairs should be identical. The 3rd values (corner) will then be opposite corners.

Wrap & Zip EB decompression (with Szymczak) (39)

- We can use our CLERS encoded chain to reconstruct the triangles as vertices are sent from the server
- However, we do not know if two corresponding points are actually the same point or separate points.
- We start by constructing the triangles without any welding of vertices.
- In addition to the CLERS labels, we also represent each triangle with an arrow.
  - Notice only C has arrow in counterclockwise direction
- After we reconstructed the triangles, we only "zip" from L's or E's.
  - E, zip up the two edges and continue in the direction of the arrow
  - L, zip up the edge and continue.
- Why is only C's arrow in counterclockwise direction?



- Looking at the T-tree, we will only label a triangle C when we traverse DOWN, and whenever we traverse UP, the triangle will be labeled something else.
- This is because C introduces new vertices, so it goes down.

Manifold meshes may have handles (46)

- Shells = “components”
- The formula **T = 2V + 4(H – S)** is useful because we are able to find the values of all the variables.
    - We already have values V and T
    - We can compute S
        - Use corner table and color in triangles
        - Each time we come to an end, that counts as 1 component.
        - If there exists an empty triangle, start coloring process again.

Project 2 discussion

- Want to represent a partition of space
- Will use images of very few colors (hard images)
- The images will therefore have solid, painted regions
- We want to find the edges bounding the different regions

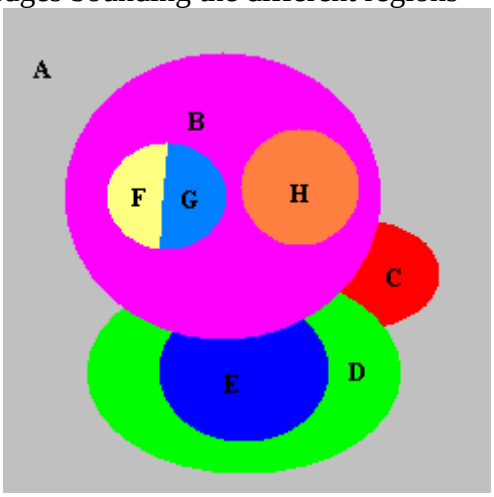


Figure 1: Example Image

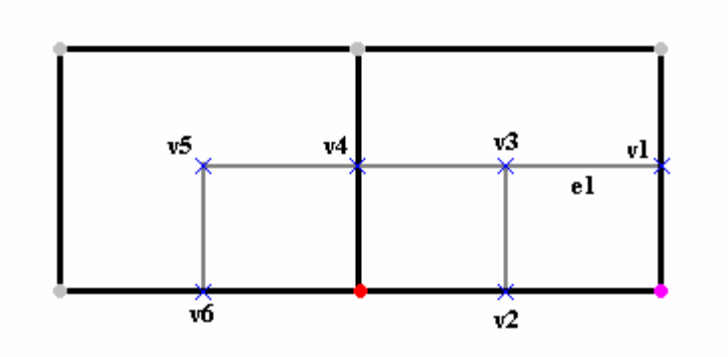


Figure 2: Vertices representation

- Colored dots in Figure 2 correspond to color region in Figure 1
- In Figure 2, we place an 'X' in the middle of a thick edge if the two adjacent vertices on that edge are different in color.
- We also place an 'X' in the middle of each square if any of the edges of that square are marked with an 'X'
- We then connect adjacent 'X's and label each of these as e's, for edges.
- These edges the boundaries between different regions
- Dump the edges to a file
- Store the x, y values of the vertices

Design a representation/data structure to store the edges

- One example is to use two tables.
    - First table: For each edge, store starting vertex, opposite edge
    - Second table: Store length of loops