Tetrahedral Meshes

The Tetrahedron

Orientation

Tet(A, B, C, D) and Tet(A, B, D, C) are different orientations.

Orientation for triangle meshes:
Corners are stored counter-clockwise as seen from the “outside” of the mesh.

Orientation for tetrahedron:
Orientation of Tet(A, B, C, D) = sign(AB \cdot (AC \times AD))
(AB \cdot (AC \times AD)) is the mixed product or determinant of |(AB)(AC)(AD)|.
B sees triangle(A, C, D) as counter-clockwise.

Defining Opposite Corners
Corner table for Tetrahedral Meshes
(Just like triangle corner table)

\[
\begin{array}{c|ccc}
V_0 & x_0 & y_0 & z_0 \\
V_1 & . & . & . \\
V_2 & . & . & . \\
V_3 & . & . & . \\
V_4 & . & . & . \\
\end{array}
\quad \begin{array}{c|ccc}
V_0 & o_0 \\
V_1 & o_1 \\
V_2 & o_2 \\
V_3 & o_3 \\
\end{array}
\]

Tet0

Tet1

Not all corners will have opposites:
(Corners on the borders of the object)

Traversing a tetrahedral mesh

To traverse a tet-mesh, you walk along the "floor" from tetrahedron to tetrahedron across a "gate" face.

Your code needs to resolve the twist along the gate in order to traverse the mesh. "Keep your head up" by finding same edges on joining tetrahedra.

Tetrahedral Mesh Compression (Edgebreaker)

C case: the same
S case: encode the ID of the vertex that is hit
Vertices are ID’ed, starting from 0 and spiraling outward around the gate triangle. The idea is that the vertex is going to be near the gate; so few bits will be used to tell the decoder which vertex this S tetrahedron touches. Add a cost of 1+log(n) for this case.
(see Gumhold and Straßer)
Also should be noted that some gates are bounding triangles.

**Avoiding Recursion when Traversing Tet-meshes Edgebreaker-style**

With a little extra effort, you can analyze a tetrahedron locally to decide to recurse 0, 1, or 2 times.

**Prediction**

For triangles: (Parallelogram prediction)

\[ V = B + C - A \]

For tetrahedron: (Average of parallelogram prediction for each face except gate)

\[ V = \frac{D+B+C}{3} + \left( \frac{D+B+C}{3} - A \right) \]

**Usage of Tet-meshes**

- **FEM**: Finite element meshes
  - Usually cubes or tetrahedra
    - **Cubes**
      - Usually regular grid
      - Harder to build
      - Can be converted to tetrahedral, but in many different ways
    - **Tetrahedra**
      - More flexible

**Cubes to Tetrahedra**

You can divide cubes into 6 congruent tetrahedra, or 4 congruent tetrahedra and one larger tetrahedron in the middle.