CSE 5443 Lab 6
Half edge mesh decimation code
Autumn, 2021

Software overview

1. Each vertex, half edge and cell is in its own "DCMT" (decimate) base class, derived from base vertex, half edge and cell classes.

2. You can extend the vertex, half edge or cell classes as needed to add extra information to each vertex, half edge and cell.

3. A class `HALF_EDGE_MESH_DCMT_BASE` (or `HalfEdgeMeshDCMTBase`) derived from `HALF_EDGE_MESH_BASE` stores the mesh with added operations for computing mesh characterstics (edge lengths, cell angles,) and for applying collapse/split/join operations to mesh edges or cells.

4. A file `half_edge_mesh_coord.*` (or class `HalfEdgeMeshCoord` in java) contains routines for computing lengths and angles.

Decimation operations

1. CollapseEdge(halfEdge): Collapse edge halfEdge, mapping the edge to its midpoint.

   - Collapse edge can create non-manifold meshes, change mesh topology or even create illegal meshes.
   - Call routines IsIllegalEdgeCollapse(), FindTriangleHole(), IsolatedTriangle(), IsInTetrahedron(), to check for conditions that would make the edge collapse illegal or change mesh topology. (See section "Check decimation operations".)
   - Check also if the two edge endpoints are on the boundary but the (half) edge is not a boundary (half) edge. Collapsing such an edge will create a non-manifold vertex.
     Code for checking such a condition would look like:

     ```
     if (!halfEdge.IsBoundary() &&
         halfEdge.FromVertex().IsBoundary() &&
         halfEdge.ToVertex().IsBoundary()) {
     // Don't collapse an interior edge with boundary endpoints...
     ```

2. SplitEdge(halfEdgeA): Split the edge represented by halfEdgeA into two edges, adding a new vertex at the edge midpoint.

3. SplitCell(halfEdgeA, halfEdgeB): Split a cell into two cells by adding a diagonal between halfEdgeA.FromVertex() and halfEdgeB.FromVertex().

   - Splitting a cell across diagonal `(vA,vB)` will create a non-manifold mesh if the mesh already has edge `(vA,vB)`. Call `mesh.FindEdge(vA,vB)` to determine if a mesh has an edge `(vA,vB)`.

4. JoinTwoCells(halfEdge): Join two cells by deleting the half edge between them.

    - Joining two cells by deleting edge `(vA,vB)` can create illegal mesh structures.
    - Call `IsIllegalJoinCells(halfEdge)` to determine whether joining cells by deleting half edge `halfEdge` is illegal.

Check decimation operations

Decimation operations can change an oriented manifold mesh to a non-manifold mesh. It can even introduce "illegal" structures in a mesh. The following operations are to check that an operation will not create illegal structures or change the mesh topology. They should be called BEFORE applying the operation.

- The CollapseEdge(), SplitEdge(), SplitCell(), JoinTwoCells() operations check whether the operation is "legal" and will not perform the operation if it is illegal.
- The CollapseEdge(), SplitEdge(), SplitCell(), JoinTwoCells() DO NOT check whether an operation will create non-manifold structures or change the mesh topology. Your program needs to check and avoid such operations to avoid changing the mesh topology.

1. `HALF_EDGE_DCMT_MESH`::IsIllegalEdgeCollapse(halfEdge): Return true if collapsing edge halfEdge is illegal.

    - Collapsing edge `(vA,vB)` is illegal if some mesh cell contains both vA and vB but not edge `(vA,vB)`.

2. `HALF_EDGE_DCMT_MESH`::FindTriangleHole(halfEdge, ivC): Returns true and sets ivC if some vertex vC is connected to both endpoints of halfEdge, but no mesh triangle contains all three vertices.

    - In such a case, collapsing halfEdge would "close" the triangle hole, either creating a non-manifold or changing the mesh topology.

3. `HALF_EDGE_DCMT_MESH`::IsIsolatedTriangle(icell): Return true if cell icell is an "isolated triangle" whose three edges are all boundary edges. (Collapsing an edge of such a cell would destroy the triangle changing the number of connected components in the mesh.)

4. `HALF_EDGE_DCMT_MESH`::IsInTetrahedron(icell): Return true if cell icell is in a mesh tetrahedron. (Collapsing an edge of such a tetrahedron would change the tetrahedron into two duplicate triangles sharing the same edge.)

5. `HALF_EDGE_DCMT_MESH`::IsIllegalSplitCell(halfEdgeA, halfEdgeB): Return true if (`halfEdgeA.FromVertex()`, `halfEdgeB.FromVertex()`) is NOT a cell diagonal.

6. `HALF_EDGE_DCMT_MESH`::IsIllegalJoinCells(halfEdge): Return true if joining two cells by deleting the half edge is illegal.

- Deleting a half edge is illegal if some endpoint of the half edge has degree two, i.e. is only incident on two edges.
- Deleting half edge `(vA,vB)` is also illegal if the two cells joined by deleting `(vA,vB)` share some other vertex vC.

Computing edge lengths or angles

- Computing edge lengths requires computing a square root. Since two edge lengths can be compared without computing the square root, routines always compare and return the square of the edge lengths, not the edge lengths.
- Computing an angle requires taking the inverse cosine or sin. Since two angles can be compared without computing acos or asin, routines always compare and return the cosine of the angle, not the angles.
  - Note that if $A < A'$, then $\cos(A) > \cos(A')$. Thus the minimum angle A maximizes cos(A) and the maximum angle A minimizes cos(A).

1. `HALF_EDGE_MESH_DCMT_BASE`::ComputeMinMaxEdgeLengthSquared(): Return the square of the minimum and maximum length of any mesh edge. Return also half edges that have those minimum and maximum lengths.
2. `HALF_EDGE_MESH_DCMT_BASE`::ComputeMinCellEdgeLengthRatioSquared(): Return the minimum ratio of the squared lengths of the shortest to longest edge in any cell. Return a cell with that minimum ratio.
3. `HALF_EDGE_MESH_DCMT_BASE`::ComputeMinMaxAngle(): Return cosine of the minimum and maximum angles at any vertex in any cell. Return also half edges whose from vertices have those minimum and maximum angles.

Half edge (not half edge mesh) member functions:

4. `HALF_EDGE_DCMT_BASE`::ComputeLengthSquared(): Return the square of the length of the half edge.
5. `HALF_EDGE_DCMT_BASE`::ComputeCosAngleAtFromVertex(): Return cosine of angle (v0,v1,v2) where v1 is the from vertex of the half edge, and v0 and v2 are the vertices preceding and following v1, respectively, in the cell.

Cell member functions:

6. `CELL_DCMT_BASE`::ComputeMinMaxEdgeLengthSquared(): Return the square of the minimum and maximum lengths of the cell edges. Returns also the half edges in the cell with minimum and maximum length.
7. `CELL_DCMT_BASE`::ComputeCosMinMaxAngle(): Return cosine of minimum and maximum angle at any cell vertex. Returns also the half edges whose from vertices give the minimum and maximum angle.

Class VERTEX_DCMT_BASE

Class `VERTEX_DCMT_BASE` has the following public operations, in addition to the ones inherited from `VERTEX_BASE`.

1. bool IsIncidentOnMoreThanTwoEdge(): Returns true if the vertex is incident on more than two edges.

    - Note that a boundary vertex that is incident on two cells will have two "from" half edges, but be incident on three mesh edges. In that case, this routine will still return true.

Modifications to previous classes

1. `VERTEX_BASE::IsBoundary()`: Return true if vertex is on the boundary. Handles even cases of non-manifolds or inconsistently oriented mesh cells.

2. `HALF_EDGE_BASE`::MinIndexHalfEdgeAroundEdge(): Return the half edge around the same edge as the current half edge with minimum index. Useful for avoiding duplications in counting the number of edges.

3. `HALF_EDGE_BASE`::EndpointsStr(): Return a string containing the half edge endpoints. Useful for printing half edge information.

4. `HALF_EDGE_BASE`::IndexAndEndpointsStr(): Return a string containing the half edge index and endpoints. Useful for printing half edge information.

5. `CELL_BASE`::IsTriangle(): Return true if the cell has exactly three edges.

6. `HALF_EDGE_MESH_BASE`::CountNumVertices(): For C++ implementation only. Return the number of vertices. (Python and Java implementations already have NumVertices() methods.)

7. `HALF_EDGE_MESH_BASE`::CountNumIsolatedVertices(): Return the number of vertices that are not incident on any mesh cell.

8. `HALF_EDGE_MESH_BASE`::CountNumIsolatedVertices(): Return the number of boundary edges.

9. `HALF_EDGE_MESH_BASE`::CountNumCells(): For C++ implementation only. Return the number of cells. (Python and Java implementations already have NumCells() methods.)

10. `HALF_EDGE_MESH_BASE`::CountNumCellsOfSize(numv): Return the number of cells with numv vertices.

11. `HALF_EDGE_MESH_BASE`::CountNumCellsOfSizeGE(numv): Return the number of cells with numv or more vertices.

12. `HALF_EDGE_MESH_BASE`::CountNumCellsOfTriangles(): Return the number of mesh triangles.

13. `HALF_EDGE_MESH_BASE`::CountNumCellsOfQuads(): Return the number of mesh quadrilaterals.

14. `HALF_EDGE_MESH_BASE`::CountNumCellsOfPentagons(): Return the number of mesh pentagons.

15. `HALF_EDGE_MESH_BASE`::CheckVertexIndex(iv): Return true if iv is an index of some mesh vertex.