CPSC 424 Assignment 7: Loop Subdivision

Term: September 2022, Instructor: Alla Sheffer, sheffa@cs.ubc.ca, http://www.ugrad.cs.ubc.ca/˜cs424

Due: Dec 07/2022 at 23:59

In this assignment you will implement the Loop Subdivision algorithm shown in class. The provided code contains the half-edge data structure you will be using as your mesh data structure.

I. Data Structure

The half-edge data structure defines objects of the following types: Vertex, HalfEdge, and Face.
If you have a vertex v, here's what you can do with it:

- $v.getPos()$ returns the corresponding coordinates of the vertex; $v.setPos(x, y, z)$ changes the corresponding coordinates of the vertex
- $v.getEdge()$ returns the half-edge with v as its origin; $v.setEdge(h)$ changes the half-edge with v as its origin
- $v.getId()$ returns the index of v

Having a half-edge h, here's what operations it supports:

- $h.getOrigin()$ returns the origin of h; $h.setOrigin(v)$ changes the origin of h
- $h.getFace()$ returns the face bounded by this half-edge; $h.setFace(f)$ changes the face bounded by thi half-edge
- $h.getPrev()$ returns the previous half-edge in the face; $h.setPrev(h)$ changes the previous half-edge
- $h.getNext()$ returns the next half-edge in the face; $h.setNext(h)$ changes the next half-edge
- $h.getTwin()$ returns the twin half-edge; $h.setTwin(h)$ changes the twin half-edge

Having a face f, here's what you can do with it:

- $f.getEdge()$ returns an arbitrary half-edge belonging to that face; $f.setEdge(h)$ changes the half-edge belonging to that face
- $f.vert(i)$ returns the i-th vertex of that face

II. Implementation of Loop Subdivision

Let's split one iteration of mesh subdivision into two main stages: Stage A (Topological Subdivision) and Stage B (Vertex Placement). Start with implementing the first stage and only when you are sure it works correctly should you move on to the second one.

A. Topological subdivision.

At this first stage, given the initial mesh, represented as a half-edge data structure, you should create a new half-edge data structure, so that for every triangle in the input mesh you create four new triangles in the output mesh (Fig. 1).

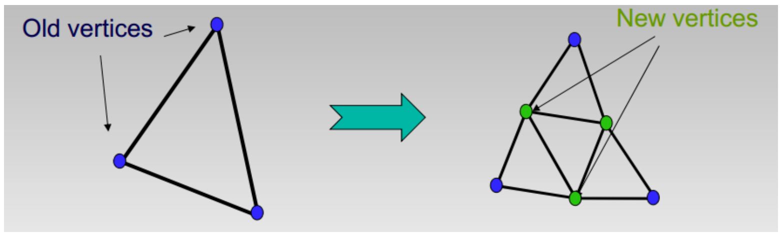
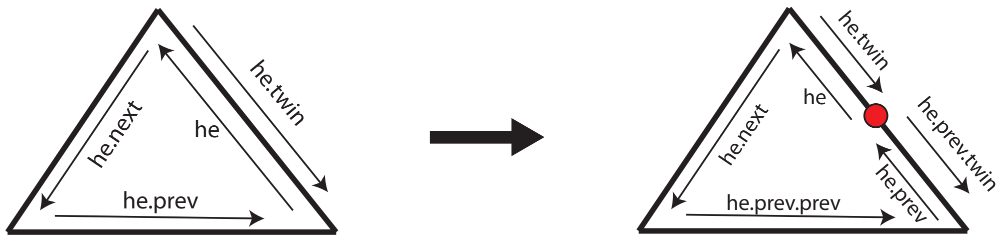


Figure 1: Topological subdivision. Every triangle is replaced by four new triangles.

For this stage,use the midpoints of the old edges as the positions of the new vertices (this will help you debug your code). Do not change the positions of the old ones. The key requirement here is to split the triangle in four, properly updating all half-edges, vertices and faces.

Here's one suggestion on how you can do it relatively easily. Create two functions, $EdgeSplit(he)$ and $CutACorner(f)$ with the following functionality:

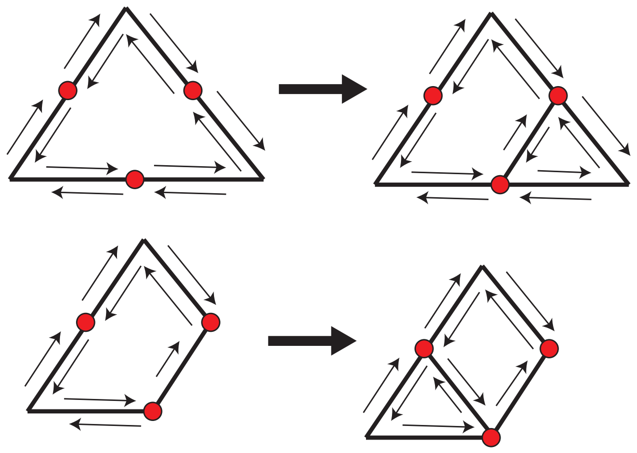
EdgeSplit(he) will take input half-edge and split edge into two:



Here's the approximate pseudo-code for this function:

```
function SPLITEDGE(HalfEdge he, Mesh M)
    add new vertex $v$ to $M$
    mark $v$ as new                                        ▷ store 'isNew' flag for every vertex
    add 2 new halfedges to $M$
    set affected $next, prev, twin, origins$
    mark $he, he.twin$ and the new halfedges as already split   ▷ store 'isSplit' flag for every halfedge
end function
```

For each edge in the mesh, first run the $EdgeSplit$ function. After its completion, create the additional faces using the $CutACorner$ function which cuts the first corner it finds (unless the face is already a triangle):



The approximate pseudo-code will be along the lines of:

```
function CUTACORNER(Face f, Mesh M)
    Add new halfedge $h$ to $M$
    Mark $h$ as already split
    Update $f.he()$ if necessary and add a new face to $M$
    Update all the necessary $next, prev, twin, origin$s
end function
```

Once you have implemented these functions, the overall algorithm for one iteration of subdivision will be more or less straightforward:

```
function SUBDIVIDELOOP(Mesh M)
    mark all vertices in $M$ as old
    mark all half-edges in $M$ as not split
    while there is an half-edge $he \in M$ that is not split yet do
        SPLITEDGE(he,M)
    end while
    while there is a non-triangle face $f \in M$ do        ▷ Think about how to check if face $f$ is a triangle or not
        CUTACORNER(f,M)
    end while
end function
```

Once this is done, your topological subdivision part is finished. Test it by applying the subdivision step over and over - check that the code does not crash, all triangles remain visible and are exactly where you expect them to be.

B.Vertex Placement.

Now we need to compute the correct positions of all vertices. The formulas are different for the old and new vertices. To avoid messing up the computations by changing the vertex positions in the middle of your calculations compute the new positions first. Before topological subdivision create an array for the new positions, compute the new vertex/old positions using the weights shown in class as well as the new edge/new positions, then after topological subdivision, set the positions of all vertices, both old and new, from your precomputations.

You can assume that the indices of the old vertices won't change if you add a new vertex. You can also assume that the index of a newly added vertex is *vertex_num*, where *vertex_num* is the size of the vertices before adding the new vertex. This means that if you iterate over every edge and add a new vertex you can be sure that their indices will be in the same order as the order of addition.