Assignment 7: Butterfly Subdivision (10 Points)

In this assignment you will implement the Butterfly Subdivision algorithm shown in class. The provided code contains the half-edge data structure you will be using as your mesh data structure. Read README.txt carefully to find out how the template and the GUI work, how to build and run the code, and how to run the provided example executables.

1 Data Structure

Half-edge data structure defines objects of the following types: Vertex, Half-Edge, Face.

If you have a vertex \( v \), here’s what you can do with it:
- \( v.x() \), \( v.y() \), \( v.z() \) return the corresponding coordinates of the vertex
- \( v.he() \) returns a pointer to some half-edge that has \( v \) as its origin.

Having a half-edge \( h \), here’s what operations it supports:
- \( h.origin() \) returns the pointer to the origin of \( h \)
- \( h.face() \) returns the pointer to the face this half-edge bounds (face is to the left of half-edge)
- \( h.prev() \) returns the pointer to the previous half-edge in the face
- \( h.next() \) returns the pointer to the next half-edge in the face
- \( h.twin() \) returns the twin half-edge

Having a face \( f \), here’s what you can do with it:
- \( f.he() \) returns some half-edge belonging to that face.

2 Implementation of Butterfly 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, move on to the second one.

2.1 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. 2.1).
Figure 1: Topological subdivision. Every triangle is replaced by four new triangles.

For this stage, there’s no need to explicitly specify the positions of the new vertices or change the positions of the old ones. The only thing that bothers us is how to split the triangle in four, properly updating all the pointers in half-edges, vertices and faces. Here’s one suggestion how you can do it relatively easy. Create two functions, \textit{EdgeSplit}(he) and \textit{CutACorner}(f) with the following functionality:

\textit{EdgeSplit}(he) will take input half-edge and split edge into two:

Here’s the approximate pseudo-code for this function.

\textbf{function} \textit{SPLITEDGE}(HalfEdge he, Mesh M)
\begin{itemize}
  \item add new vertex \( v \) to \( M \)
  \item mark \( v \) as new
  \item add 2 new halfedges to \( M \)
  \item set affected next, prev, twin, origins
  \item mark he, he.twin and the new halfedges as already split
\end{itemize}
\textbf{end function}

So for each edge in the mesh, you first run \textit{EdgeSplit} function. After it’s all done, you create the additional faces using the \textit{CutACorner} function that cuts the first corner it finds (unless the face is already a triangle):

The approximate pseudocode 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 $next$, $prev$, $twin$, $origins$
end function

Once you have implemented those 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 $\triangleright$ Think 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 done.

2.2 Vertex Placement

Now we need to compute the positions of the vertices. The formulas are different for the old and new vertices. Luckily, in the topological subdivision we set up the “new” flag per every vertex exactly for this reason. This part should be relatively easy, for the formulas please refer to the lecture slides.

3 Submission

Submit your code, a README file containing your name and student number, and what you have implemented with the handin command that you should know from other courses:

    handin cs424 A7