

## Implementing subdivision for Project 3

This is a description of a possible efficient implementation of the subdivision procedure. It will consist of the following steps:

1. Compute degrees of the vertices of the input mesh
2. Compute all the incident triangles of a vertex in the input mesh
3. Compute the adjacency table of the input mesh
4. Label the edges of the input mesh
5. Build the triangle table of the output (subdivided) mesh
6. Compute the vertex coordinates for the subdivided mesh

We would like to perform each of the six steps in an efficient manner (possibly, in linear time). This is because in practice one often needs to iterate subdivision several times to obtain a desirably smooth mesh. Each subdivision step causes the number of triangles to grow by a factor of four. This means that the size of the input mesh for the subdivision routine grows very fast (exponentially) with the number of subdivision steps.

### 1 Degrees of the vertices

First, allocate a vector of integers that will eventually hold the degrees of the vertices. The size of the vector is equal to the number of vertices. Then, for each triangle in the input mesh, increment the vector's entries corresponding to the vertices of that triangle.

In the end, after all the triangles are processed, each of the vector's entries will be incremented as many times as there are triangles incident upon the corresponding vertex. Thus, it will hold the number of incident triangles (degree of the vertex).

### 2 Incident triangles for all the vertices of the input mesh

The idea is similar to degree calculation. Associate a list of integers to each of the vertices. Initialize all lists to 'empty'. Then, add each triangle to the lists of all of its vertices. At the end of the day, the list of each vertex will hold the identifiers of all its incident triangles.

### 3 Adjacency table

A possible way to represent adjacency is by means of adjacency table as described in one of the previous lecture notes. The adjacency table lists three adjacent triangles for each of the triangles in the mesh. It is a table of integers of the same dimensions as the triangle table (3 columns,  $T$  rows, where  $T$  is the number of triangles). It is convenient to assume some link of the order the adjacent triangles are listed in the adjacency table to the order in which the vertices are listed in the triangle table. For example, one can assume that the ID of the triangle adjacent to the edge opposite to the  $i$ -th vertex of the  $j$ -th triangle is listed in  $j$ -th row and  $i$ -th column of the adjacency table ( $i = 0, 1, 2$ ,  $j = 0, 1, \dots, T - 1$ , assuming we are numbering starting from zero).

To compute the triangle adjacent across the edge joining a two vertices with labels  $a$  and  $b$  of the  $j$ -th triangle can be found by going over the list of incident triangles of either  $a$  or  $b$  and looking for a triangle with ID different from  $j$  that has both  $a$  and  $b$  as its vertices. To avoid scanning the longest lists, one might select the shorter of the two lists (i.e. go over the list of the vertex of smaller degree). With or without this heuristics, the algorithm works pretty well in practice since typically there are no vertices of very high degree in a ‘nice’ mesh.

**Problem for the theoretically inclined:** Does the heuristics described above lead to a linear time algorithm? Why?

### 4 Labelling the edges

Construct a table with the same structure as the adjacency table, but holding an identifier of the edge opposite to a vertex of a triangle instead of the identifier of the triangle adjacent across the edge opposite to a vertex. First, allocate the table and fill it with a negative number, e.g.  $-1$  (which will be interpreted as ‘not numbered yet’). Then, go over all triangles in the mesh and assign consecutive numbers to their edges (and fill these numbers into the table). To avoid numbering an edge twice:

1. We assign numbers only to the edges that are not numbered (‘-1!’).
2. Whenever we label an edge of a triangle, we find the other triangle that shares that edge and update one of the entries corresponding to that triangle. That other triangle can be found using the adjacency table

### 5 Triangle table of the output

At this point, we have all the information we need to label all vertices of the subdivided mesh. The vertices corresponding to the vertices of the original mesh (‘old’ vertices) can keep their IDs. If the original mesh has  $V$  vertices, we can set the ID of an edge vertex corresponding to the edge whose number is  $e$  to

$e + V$ . In this way, the ‘old’ vertices get IDs in the range  $0 \dots V - 1$  and the edge vertices - the IDs in the range  $V \dots E + V$  where  $E$  is the number of edges in the original mesh. By the way, since each triangle has 3 edges and each edge is shared by exactly 2 triangles,  $E = 3T/2$ .

So, we have the labelling of the vertices... Now we can simply go over the triangles of the original mesh and, for each of them, generate triples of vertices bounding each of the four triangles of the subdivided mesh. Be careful about to orient the ‘small’ triangles consistently with the orientation of the ‘large’ one. Since we are splitting each triangle into four, the number of output triangles will be  $4T$ .

## 6 Locations of vertices of the subdivided mesh

Use the adjacency/incidence information to compute the vertices that we need to use for the weighted (weights for the old vertices will depend on the degrees too) average, and compute those averages. You will need separate routines for ‘old’ and edge vertices, since the formulas are very different in each of the two cases. Use the averages to fill the vertex table of the output mesh. The number of output vertices will be  $V + E = V + 3T/2$ .