Computer Graphics 2021
Exercise 4: Linear Triangle Subdivision (Project 1)

Prof. Dr. Martin Hering-Bertram
Hochschule Bremen

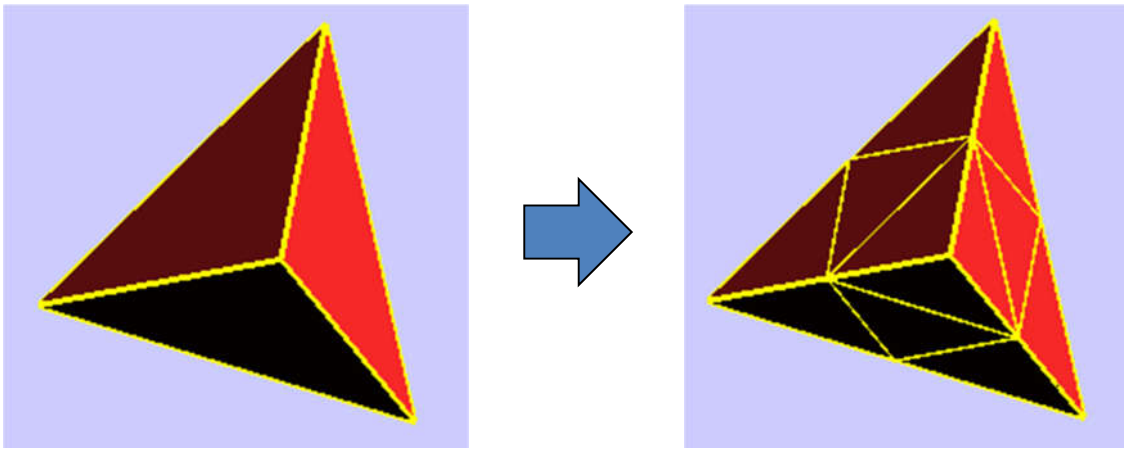

Figure 4.1: Linear triangle splitting.

Problem 1: Mesh Data Structure

(a) Extend your vertex and triangle classes by the following functions (use your own class and variable names):

```
class Vertex
{
public: // alternatively, use getters and setters
    float p[3]; // coordinates

    Vertex();
    Vertex( float point[3]);
    Vertex( float x, float y, float z);
    void Print(); // print coordinates on screen

};

Vertex operator+( Vertex a, Vertex b); // add points or vectors
Vertex operator-( Vertex a, Vertex b); // subtract points or vectors
Vertex operator*( float a, Vertex b);  // product between scalar and vector
float  operator*( Vertex a, Vertex b); // scalar product
Vertex operator%( Vertex a, Vertex b); // cross product

class Tri
{
public:
    int iv[3]; // vertex indices
    int it[3]; // adjacent triangle indices
    int ie[3]; // edge vertex indices
    Tri();
    Tri( int i[3]);
    Tri( int i, int j, int k);
    void Print();
};
```

The vertex operators can be included in the class as friend functions. Now, calculating the normal vector of a triangle t[i] is as simple as follows:

```
vector <Vertex> pts;   // point list
vector <int> valences; // valence list (no. of triangles for every point
vector <Tri> tris;     // triangle list
...
Tri t = tris[i];
Vertex a = pts[t.iv[0]];
Vertex b = pts[t.iv[1]];
Vertex c = pts[t.iv[2]];
Vertex nvec = (b-a)%(c-a);
```

You may implement a mesh class containing the above vectors. This way, your implementation can handle multiple meshes.

Problem 2: Connectivity and Linear Subdivision Algorithm

(a) Read in the file tetra.obj and print out all vertices and triangles as they are stored in your mesh class.

(b) Implement the **Connectivity Algorithm** for your mesh class, such that each triangle knows its triangle neighbors (see lecture notes on **Loop Subdivision**; you can assume that the surface does not have boundaries). You may also implement a function for validating the data structure and reporting errors. Print out the data structure and check its correctness.

(c) Implement a linear subdivision algorithm, as depicted in figure 4.1. Split every triangle into four introducing the edge vertices and set the corresponding indices in the triangle structure. Make sure that each vertex is inserted only once. For simplicity, **compute the edge midpoints**, rather than Loop's subdivision mask. Again, print the mesh and check for correctness.

(d) Render the mesh after one subdivision. Draw both, edges and triangles.

(e) Apply the subdivision a second time to verify that the data structure is valid.

Present your solution to this exercise in the laboratory on May 10th.

Computer Graphics 2021
Exercise 5: Loop Subdivision (Project 1)

Prof. Dr. Martin Hering-Bertram
Hochschule Bremen

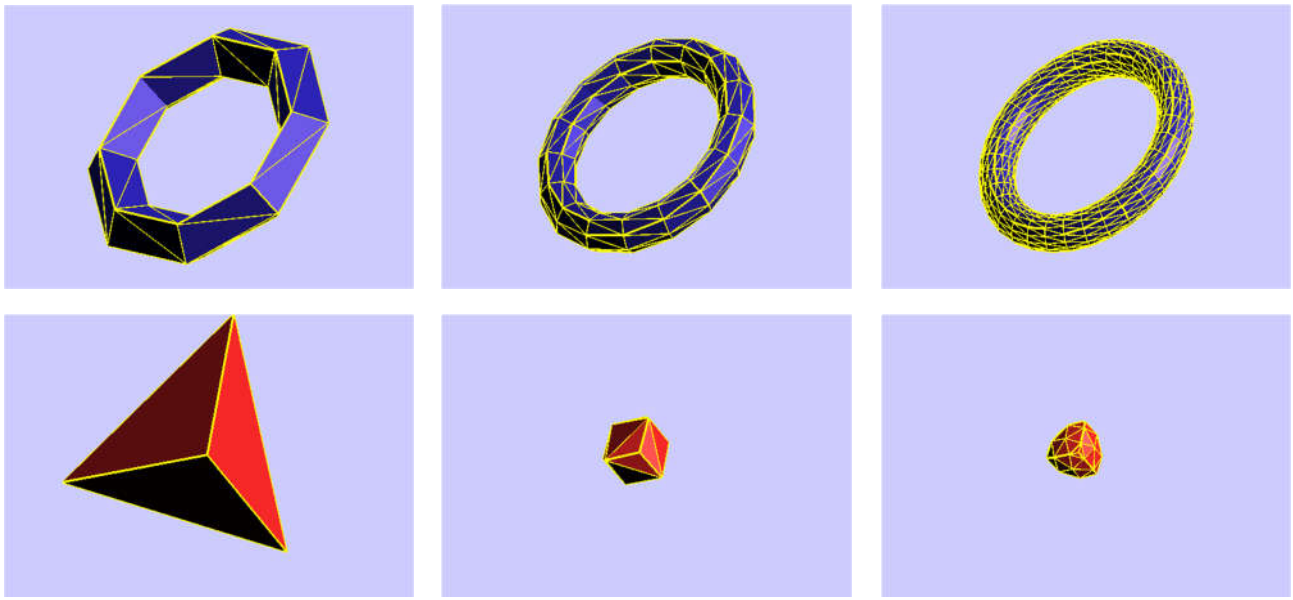

Figure 5.1: Loop Subdivision for triangle meshes.

Problem 1: Loop Subdivision

(a) Add the following functions to your Implementation:

```
void Vertex::operator*=( float a);
void Vertex::operator+=( Vertex a);
float beta_n( int n);
```

The first two are member functions of the vertex class and will simplify the computation of the subdivision masks.

beta_n calculates the values for β in the vertex mask for valence n (see lecture on Loop Subdivision), for example:

```
n: 3, beta: 0.1
n: 4, beta: 0.225
n: 5, beta: 0.327254
n: 6, beta: 0.4
n: 7, beta: 0.450921
n: 8, beta: 0.487132
```

Now, the first part of the vertex subdivision mask

```
    for each v∈V:
        v := β(v)*v
```

can be implemented as follows:

```
for( i=0; i< pts.size(); i++){ // multiply every vertex with beta
    int n = valences[i];       // n = valence of v_i
    float beta = beta_n( n);
    pts[i] *= beta;            // v_i *= beta(n)
}
```

(b) Replace the linear subdivision of exercise 4 by Loop subdivision. Test your algorithms first with only one subdivision step. Then, add a second and third subdivision. Print out the meshes and verify that the numbers of vertices and faces are correct. Examples are depicted in figure 5.1.

Problem 2: Documentation

Start preparing a LaTeX documentation of your project written in English, containing

- screenshots of your subdivision curves and surfaces
- a table with numbers of vertices and faces at the first three subdivision levels
- the most important formulae, e.g. subdivision masks set as equations
- example code fragments showing your implementation of the masks
- references for subdivision algorithms.

The documentation will be turned in, later. It should provide the most important features of your implementation. It should be as short as possible, easy to read and may contain the most important information for reproducibility.

Present your solution to this exercise in the laboratory and turn in your code for exercise 4+5 on May 17th.