Loop subdivision for triangle meshes

Reeder Ward, David Melamed and Majbrit Schöttner

May 14, 2021

Abstract

In this document, a self-implemented version of the subdivision for triangle meshes is explained in more detail.

1 Introduction

2 Subdivision element count

	no subdivision	subdivision 1	subdivision 2	subdivision 3
vertices	4	10	34	130
faces	4	16	64	256

As can be seen in the table, there is a regularity to the triangles by which they increase with each subdivision. The points do not show any regularity.

With each subdivision, four new triangles are assigned to a triangle, but a point can also have several triangles assigned to it.

3 Important Formulae

Chaikin's algorithm can be represented as a weighted sum of the points P_i^{k-1} of the previous iteration. Thus, Chaikin's algorithm can be represented as follows, where real coefficients $= a_{ijk}$:

$$\sum_{i=0}^{n_k-1} a_{ijk} P_i^{k-1} \tag{1}$$

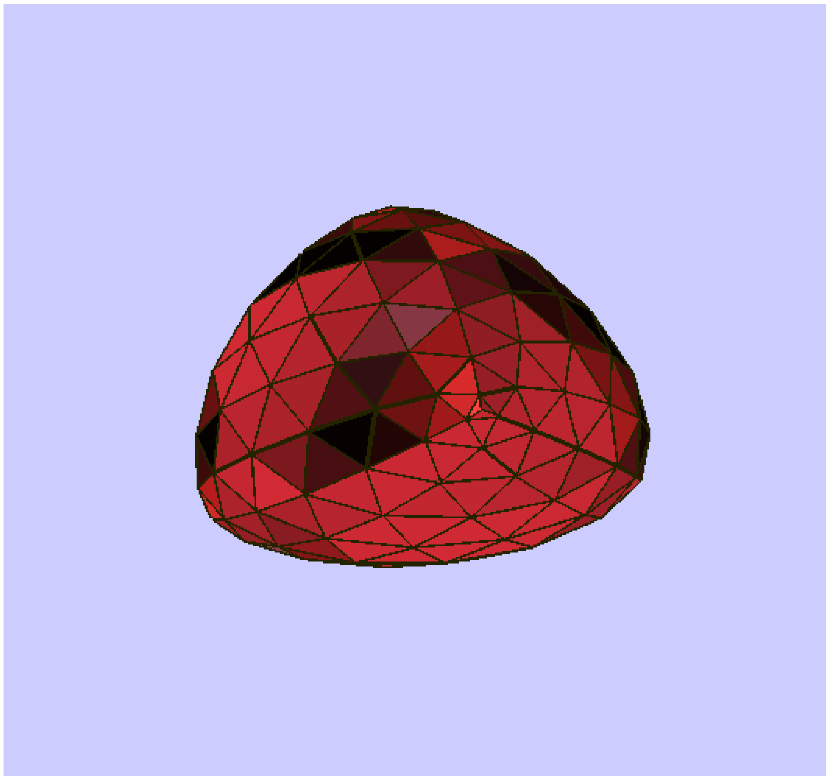

Figure 1: Something

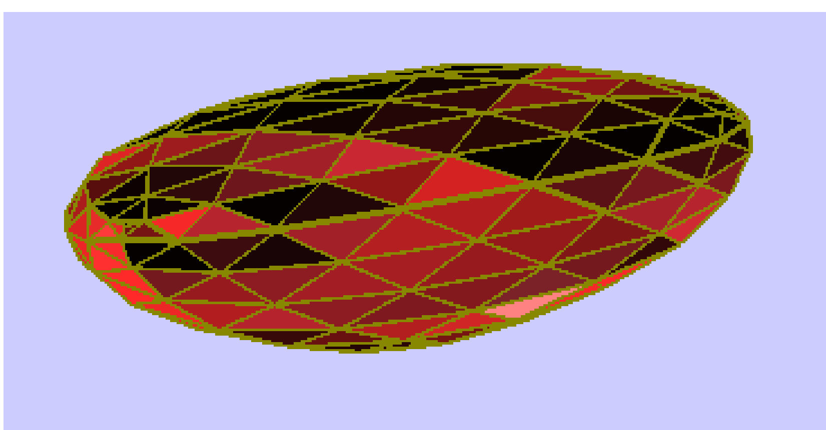

Figure 2: Something

3.0.1 Loop subdivision scheme:

Where e: edge point, v0 and v1: points spanning the edge, v1 and v2: third point of each of the two triangles involved, this is the equation for the **Edge Mask**:

$$e' = \frac{3}{8}(v_0 + v_1) + \frac{1}{8}(v_2 + v_3) \tag{2}$$

Vertex Mask:

$$v' = \beta(n)\mathbf{v} + \frac{1 - \beta(n)}{n} \sum_{i=0}^{n} \mathbf{e}'_{\mathbf{i}} \tag{3}$$

4 Code Snippets and Curves

Chaikin's Algorithm mask implementation:

```
// fill center of the matrix
int c = columns-2;
int l = (lines -3)*2;
int i=2;
int j=1;
while( i<l) {
    while( j<c) {
        matrix[i][j] = 0.75;
        matrix[i][j+1] = 0.25;
        matrix[i+1][j] = 0.25;
        matrix[i+1][j+1] = 0.75;
        j=j+1;
        break;
    }
    i=i+2;
}
// fill first two and last two lines of the matrix
matrix[0][0] = 1.0;
matrix[1][1] = 0.5;
matrix[1][0] = 0.5;
matrix[lines -1][columns-1] = 1.0;
matrix[lines -2][columns-1] = 0.5;
matrix[lines -2][columns-2] = 0.5;
```

Cubic interpolation mask implementation:

```
//fill center of the matrix
int c = columns-1;
int l = (lines -3)*2;
```

```
int i=2;
int j=0;
while( i<l) {
    while(  j<c) {
        matrix[i][j+1] = 1;
        matrix[i+1][j] = -0.0625;
        matrix[i+1][j+1] = 0.5625;
        matrix[i+1][j+2] = 0.5625;
        matrix[i+1][j+3] = -0.0625;
        j=j+1;
        break;
    }
    i=i+2;
}
//fill first two and last two lines of the matrix
matrix[0][0] = 1.0;
matrix[1][0] = 0.375;
matrix[1][1] = 0.75;
matrix[1][2] = -0.125;
matrix[lines-1][columns-1] = 1.0;
matrix[lines-2][columns-1] = 0.375;
matrix[lines-2][columns-2] = 0.75;
matrix[lines-2][columns-3] = -0.125;
matrix[lines-3][columns-2] = 1.0;
```

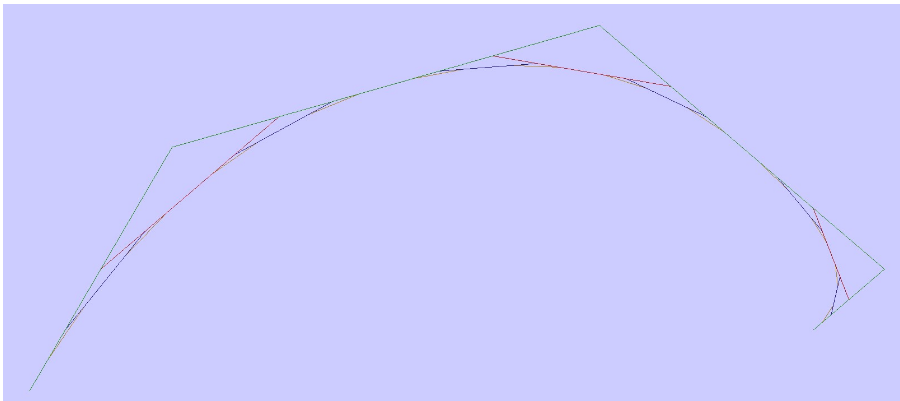

Figure 3: Chaikin's Algorithm curve.

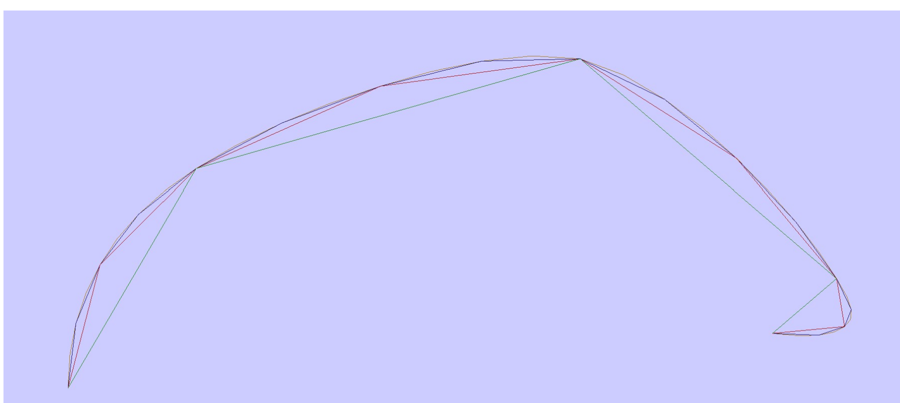

Figure 4: Interpolating cubic subdivision curve.