Loop subdivision for triangle meshes

Reeder Ward, David Melamed and Majbrit Schöttner

Mai 2021

1 Introduction

	subdivision 1	subdivision 2	subdivision 3
vertices	...	...	...
faces	...	...	...

2 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:

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

Cubic interpolation:

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{2}$$

2.0.1 Loop subdivision scheme:

Edge-Mask:

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

Vertex Mask:

$$v' = \alpha(n)\mathbf{v} + \frac{1-\alpha(n)}{n} \sum_{i=1}^{n} \mathbf{v_i} \tag{4}$$

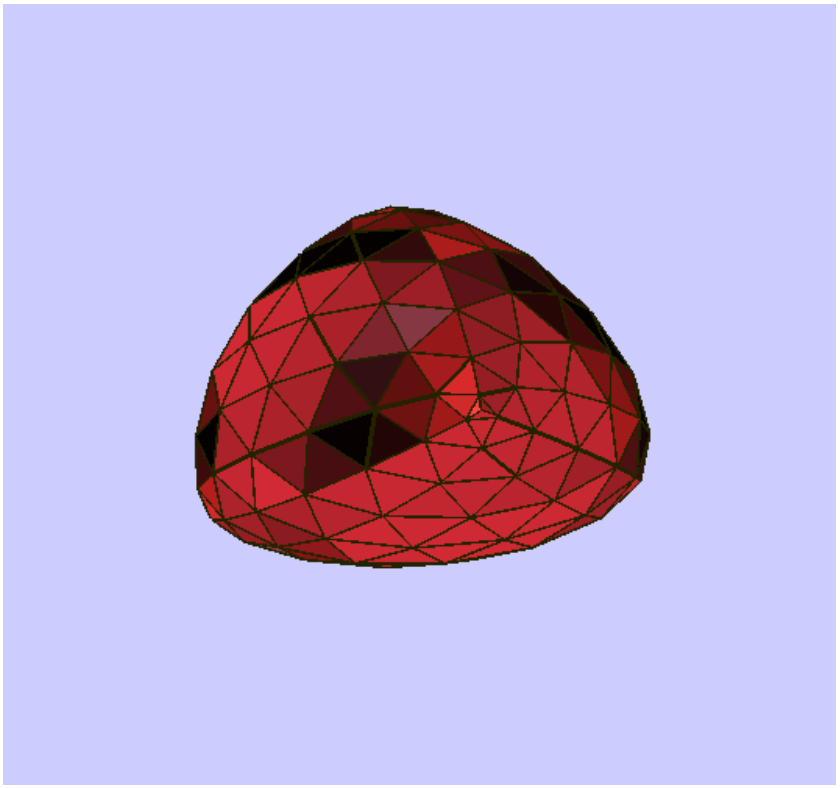

Figure 1: Something

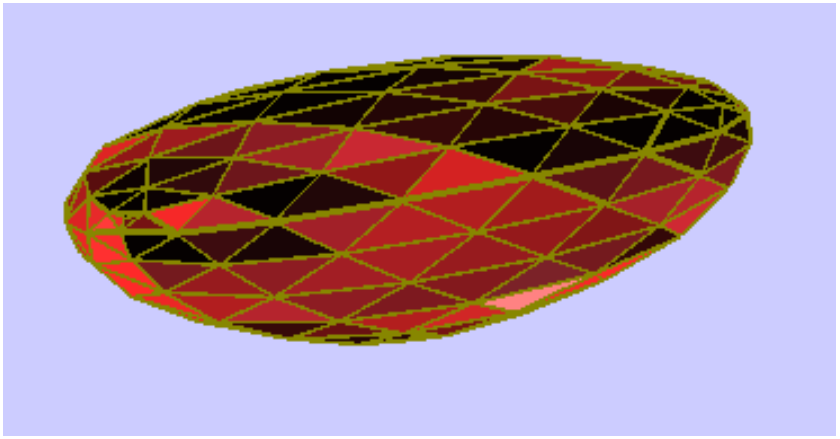

Figure 2: Something

3 Code Snippets

```
//calculate with dynamic size of matrices
//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;

//---matrix multiplication---
float xproduct = 0;
float yproduct = 0;
float zproduct = 0;
for( int i=0; i<lines; i++) {
    xproduct = 0;
    yproduct = 0;
    zproduct = 0;
    for( int j=0; j<columns; j++) {
        //calculate x,y and z of one new point
        xproduct += matrix[i][j]*xold[j];
        yproduct += matrix[i][j]*yold[j];
        zproduct += matrix[i][j]*zold[j];
    }
    //insert new coordinates in vectors
    xnew.push_back(xproduct);
    ynew.push_back(yproduct);
    znew.push_back(zproduct);
}
xyznew.push_back(xnew);
xyznew.push_back(ynew);
xyznew.push_back(znew);
```

Figure 3: Implementation of Chaikin's Algorithm.

```
//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;
```

```
//---matrix multiplication---
float xproduct = 0;
float yproduct = 0;
float zproduct = 0;
for( int i=0; i<lines; i++) {
    xproduct = 0;
    yproduct = 0;
    zproduct = 0;
    for( int j=0; j<columns; j++) {
        //calculate x,y and z of one new point
        xproduct += matrix[i][j]*xold[j];
        yproduct += matrix[i][j]*yold[j];
        zproduct += matrix[i][j]*zold[j];
    }
    //insert new coordinates in vectors
    xnew.push_back(xproduct);
    ynew.push_back(yproduct);
    znew.push_back(zproduct);
}
xyznew.push_back(xnew);
xyznew.push_back(ynew);
xyznew.push_back(znew);
```

Figure 4: Implementation of interpolating cubic subdivision.