

Subdivision after user click mouse left button

Ang Lee coolpix@cc.gatech.edu

```
void Manager::RetriangulateMesh()
{
    int i, j, NumOfVerticesInCirclecount, subdivideCase;
    int InCirclecount = 0;
    int ccountBeforeTriangulation;
```

I start by determining which points located in the circle first. This can be done by going through the G-table comparing the distances between the vertices and the center of circle to the radius of the circle. After this process, all the indexes of the vertices located within the circle will be stored.

```
for (i = 0; i < vcount; i++)
{
    if ( (gx[i]-Mcenterx)*(gx[i]-Mcenterx) + (gy[i]-Mcentery)*(gy[i]-Mcentery) <= Mradius*Mradius )
    {
        gz[i] += 30; //Increase the height of the vertex by a constant
        glVertex2f(gx[i], gy[i]);
        InCircle[InCirclecount++] = i;
    }
}
```

Since to adjacent triangle share an edge, and if both this triangles are to be subdivided, they may use the same mid-edge point and which shouldn't be created twice. Hence, I created a Boolean array for the use of storing which mid-edge points have been created. First, set all the value in the array to be 'false', in other words, the point has not been created. The length of this array is the same as the corner table.

```
for (i=0; i < ccount; i++)
{
    MidpointAlreadyCreated[i] = 0;
}
```

During this process, the number of corner if change, so store the number before subdivision to be used in the for loop.

```
ccountBeforeTriangulation = ccount;
```

Go through the triangles. And for each triangle(which has three corners, $v[i]$, $v[i+1]$, $v[i+2]$) go through the table of which vertices located in the circle, to see if any of the three corners' vertex is on the list (i.e. in the circle). For one triangle, there are at most three matches. So, once all three been found, break. At the same time we would like to know along which edge a mid-edge vertex need to be created, so also record the case using a three-bit binary manner. Each bit correspond to an edge the that certain triangle.

```
for (i = 0; i < ccountBeforeTriangulation; i += 3)
{
    NumOfVerticesInCirclecount = 0;
    subdivideCase = 0;
    for (j = 0; j < InCirclecount; j++)
    {
        if (NumOfVerticesInCirclecount != 3)
        {
            if (v[i] == InCircle[j] )
            {
                NumOfVerticesInCirclecount++;
                subdivideCase++; //Binary 001
            }
            else if (v[i+1] == InCircle[j] )
            {
                NumOfVerticesInCirclecount++;
```

```

        subdivideCase+=2; //Binary 010
    }
    else if (v[i+2] == InCircle[j] )
    {
        NumOfVerticesInCirclecount++;
        subdivideCase+=4; //Binary 100
    }
}
}
}

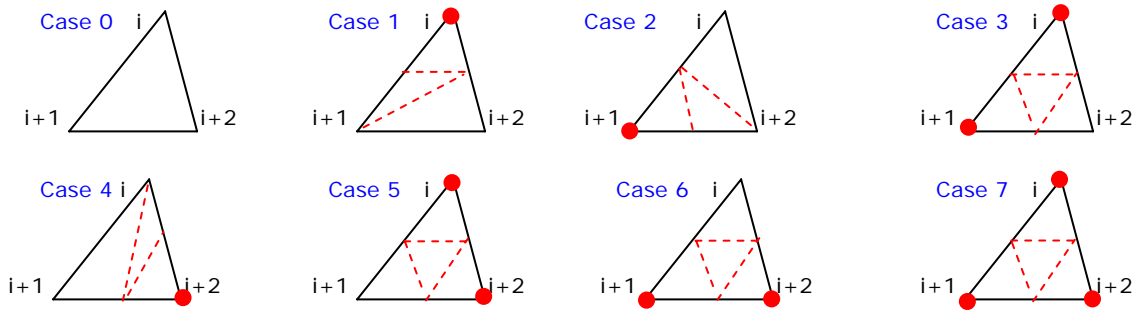
```

After we determined which case the current triangle belongs to, subdivide it. The function takes the case number and which triangle to be subdivided.

```

    SubdivideATriangle(subdivideCase, i);
} //end of for (i = 0; i < ccount; i += 3)
computeOtable();
}

```



```

void Manager::SubdivideATriangle(int subdcase, int i) // i is the first corner in the triangle
{

```

```

    int m1, m2, m3;

```

```

    //printf(" i = %i, v[i] = %i, case = %i\n", i, v[i], subdivideCase);
    switch (subdcase)
    {

```

```

        case 0 :

```

```

            break;

```

```

        case 1 :

```

```

            if (o[i+1]*p(i+2)*j != -1 && MidpointAlreadyCreated[o[i+1]]) // Mid-edge point exists. Go ahead
            to find it.

```

```

                m2 = GetMidPoint(v[i+2], v[ i ]);

```

'GetPoint' is a function takes two consecutive corners in a triangle and it returns the mid-edge point along the edge shared by those two consecutive corners.

```

            else // Mid-edge point doesn't exist. Create a mid-edge point.

```

```

            {

```

```

                m2 = vcount; vcount++;

```

```

                gx[m2] = (gx[v[i+2]] + gx[v[ i ]])*0.5;

```

```

                gy[m2] = (gy[v[i+2]] + gy[v[ i ]])*0.5;

```

```

                gz[m2] = (gz[v[i+2]] + gz[v[ i ]])*0.5;
            }

```

```

            if (o[i+2]*p(i)*j != -1 && MidpointAlreadyCreated[o[i+2]])

```

```

                m3 = GetMidPoint(v[ i ], v[i+1]);

```

```

            else

```

```

            {

```

```

                m3 = vcount; vcount++;

```

```

                gx[m3] = (gx[v[ i ] ] + gx[v[i+1]])*0.5;

```

```

                gy[m3] = (gy[v[ i ] ] + gy[v[i+1]])*0.5;

```

```

                gz[m3] = (gz[v[ i ] ] + gz[v[i+1]])*0.5;
            }

```

```

        }

```

```

v[ccount++] = m3;
v[ccount++] = v[i+1];
v[ccount++] = m2;

```

```

v[ccount++] = m2;
v[ccount++] = v[i+1];
v[ccount++] = v[i+2];

```

```

//v[i] = v[i];
v[i+1] = m3;
v[i+2] = m2;

```

Update the 'MidpointAlreadyCreated' array of the triangle.

```

MidpointAlreadyCreated[i+1] = MidpointAlreadyCreated[i+2] = 1;
break;

```

case 2 :

```

if (o[i/*p(i+1)*/] != -1 && MidpointAlreadyCreated[o[i]])
    m1 = GetMidPoint(v[i+1], v[i+2]);

```

else

```
{
```

```

    m1 = vcount;vcount++;
    gx[m1] = (gx[v[i+1]] + gx[v[i+2]])*0.5;
    gy[m1] = (gy[v[i+1]] + gy[v[i+2]])*0.5;
    gz[m1] = (gz[v[i+1]] + gz[v[i+2]])*0.5;

```

```
}
```

```

if (o[i+2/*p(i)*/] != -1 && MidpointAlreadyCreated[o[i+2]])

```

```

    m3 = GetMidPoint(v[ i ], v[i+1]);

```

else

```
{
```

```

    m3 = vcount;vcount++;
    gx[m3] = (gx[v[ i ]] + gx[v[i+1]])*0.5;
    gy[m3] = (gy[v[ i ]] + gy[v[i+1]])*0.5;
    gz[m3] = (gz[v[ i ]] + gz[v[i+1]])*0.5;

```

```
}
```

```
v[ccount++] = m3;
```

```
v[ccount++] = v[i+1];
```

```
v[ccount++] = m1;
```

```
v[ccount++] = m3;
```

```
v[ccount++] = m1;
```

```
v[ccount++] = v[i+2];
```

```
//v[i] = v[i];
```

```
v[i+1] = m3;
```

```
v[i+2] = v[i+2];
```

```
MidpointAlreadyCreated[i] = MidpointAlreadyCreated[i+2] = 1;
```

```
break;
```

case 4 :

```

if (o[i/*p(i+1)*/] != -1 && MidpointAlreadyCreated[o[i]])

```

```

    m1 = GetMidPoint(v[i+1], v[i+2]);

```

else

```
{
```

```

    m1 = vcount;vcount++;
    gx[m1] = (gx[v[i+1]] + gx[v[i+2]])*0.5;
    gy[m1] = (gy[v[i+1]] + gy[v[i+2]])*0.5;
    gz[m1] = (gz[v[i+1]] + gz[v[i+2]])*0.5;

```

```
}
```

```

if (o[i+1/*p(i+2)*/] != -1 && MidpointAlreadyCreated[o[i+1]])

```

```

    m2 = GetMidPoint(v[i+2], v[ i ]);

```

else

```
{
```

```

    m2 = vcount;vcount++;

```

```

        gx[m2] = (gx[v[i+2]] + gx[v[ i ]])*0.5;
        gy[m2] = (gy[v[i+2]] + gy[v[ i ]])*0.5;
        gz[m2] = (gz[v[i+2]] + gz[v[ i ]])*0.5;
    }
    v[ccount++] = m2;
    v[ccount++] = m1;
    v[ccount++] = v[i+2];

    v[ccount++] = v[i];
    v[ccount++] = m1;
    v[ccount++] = m2;

    //v[i] = v[i];
    v[i+1] = v[i+1];
    v[i+2] = m1;
    MidpointAlreadyCreated[i] = MidpointAlreadyCreated[i+1] = 1;
    break;
case 3 :
case 5 :
case 6 :
case 7 :
    if (o[i/*p(i+1)*/] != -1 && MidpointAlreadyCreated[o[i]])
        m1 = GetMidPoint(v[i+1], v[i+2]);
    else
    {
        m1 = vcount;vcount++;
        gx[m1] = (gx[v[i+1]] + gx[v[i+2]])*0.5;
        gy[m1] = (gy[v[i+1]] + gy[v[i+2]])*0.5;
        gz[m1] = (gz[v[i+1]] + gz[v[i+2]])*0.5;
    }

    if (o[i+1/*p(i+2)*/] != -1 && MidpointAlreadyCreated[o[i+1]])
        m2 = GetMidPoint(v[i+2], v[ i ]);
    else
    {
        m2 = vcount;vcount++;
        gx[m2] = (gx[v[i+2]] + gx[v[ i ]])*0.5;
        gy[m2] = (gy[v[i+2]] + gy[v[ i ]])*0.5;
        gz[m2] = (gz[v[i+2]] + gz[v[ i ]])*0.5;
    }

    if (o[i+2/*p(i)*/] != -1 && MidpointAlreadyCreated[o[i+2]])
        m3 = GetMidPoint(v[ i ], v[i+1]);
    else
    {
        m3 = vcount;vcount++;
        gx[m3] = (gx[v[ i ] ] + gx[v[i+1]])*0.5;
        gy[m3] = (gy[v[ i ] ] + gy[v[i+1]])*0.5;
        gz[m3] = (gz[v[ i ] ] + gz[v[i+1]])*0.5;
    }
    v[ccount++] = m3;
    v[ccount++] = v[i+1];
    v[ccount++] = m1;

    v[ccount++] = m2;
    v[ccount++] = m1;
    v[ccount++] = v[i+2];

    v[ccount++] = m3;
    v[ccount++] = m1;
    v[ccount++] = m2;

    //v[i] = v[i];
    v[i+1] = m3;

```

```
v[i+2] = m2;

printf("ccount = %i\n", ccount);
printf("vcount = %i\n", vcount);

MidpointAlreadyCreated[i] = MidpointAlreadyCreated[i+1] = MidpointAlreadyCreated[i+2] = 1;

break;
}
//end of switch(subdcase)
}
```