UM EECS 487
Lab 9: Subdivision Surface

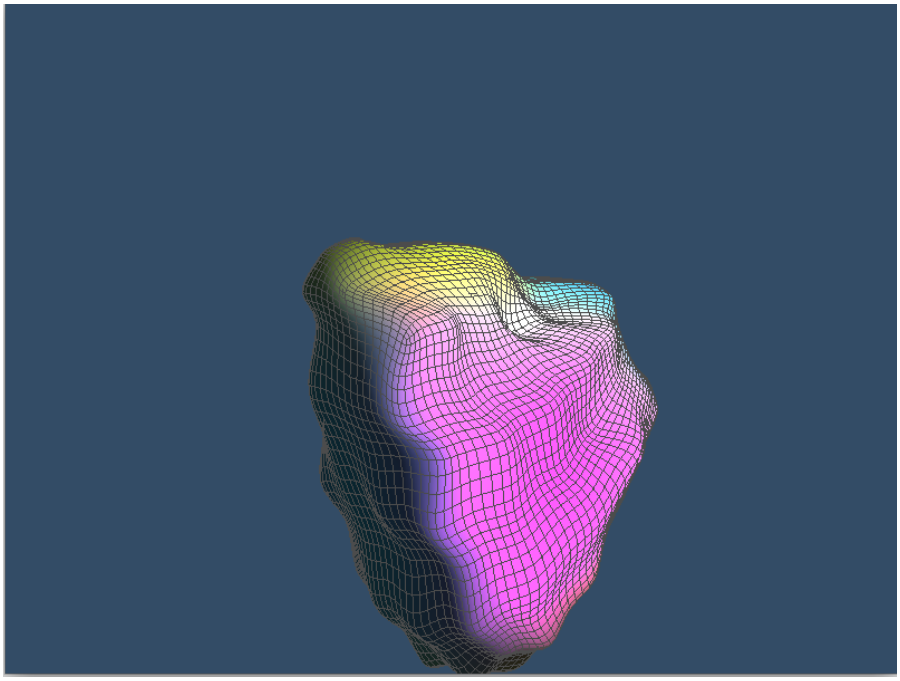

FIGURE 1. A delectable cotton candy-like blob made from subdividing a random base mesh.

1. Out of the Box

Make and run the executable `subdivision`. A window will appear that shows a pastel cube hovering peacefully. Try out the key bindings below.

Key Binding	Function
x/X or j/k	Rotate around the x-axis
y/Y or s/w	Rotate around the y-axis
z/Z or h/l	Rotate around the z-axis
m	Toggle the mesh on/off
d	Subdivide the mesh
r	Randomly displace the vertex positions
I or SPACE	Reset the mesh
q or ESC	Quit the application

You can use the above keys to rotate the mesh. The rotation used is $\mathbf{v}' = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z\mathbf{v}$, so you can expect to find gimbal lock, and other non-intuitive controls (for certain orientations).

What Does It Do? Try pressing 'd' a few times to subdivide the mesh; it just increases the tessellation of every face without affecting the shape of the geometry at all. The basic algorithm is as follows.

create array *vertices*.
create array *quads*.
for each face f in the mesh **do**
  add the face point $\mathbf{f}$ to *vertices*.
  for each edge e **do**
    add the midpoint $\mathbf{m} = \frac{1}{2}(\mathbf{v}_0 + \mathbf{v}_1)$ of e to *vertices* where $\mathbf{v}_0$ and $\mathbf{v}_1$ are the endpoints of the edge.
  end for
  for each vertex $\mathbf{v}$ **do**
    add $\mathbf{v}$ to *vertices*.
  end for
  for each vertex $\mathbf{v}$ **do**
    add a quad to *quads* with vertices: the old face point $\mathbf{f}$, the vertex to the "left" of $\mathbf{v}$, the corner $\mathbf{v}$, and the vertex to the "right" of $\mathbf{v}$.
  end for
end for
update the mesh to use *vertices* and *quads*.

It takes each face (quad) and divides it up into four pieces using the face point and each edge's midpoint. This is sufficient for an artist who may want more control points in the mesh to start adding finer detail; however, it would be nice if it could smooth out the mesh a little as well. Thus after creating a mesh, the artist can smooth it a few times and voilà, a smooth, hi-resolution mesh is created!

2. Catmull-Clark Subdivision

In order to smooth out the mesh, the face points that are created could be moved, the new edge midpoints could be moved, and the original edge endpoints could be moved. Different combinations of these with different *movement*-procedures can result in many different types of smoothing. We will use the Catmull-Clark algorithm as described in the lecture on subdivision surfaces in this lab.

create array *vertices*.
create array *quads*.
for each face f in the mesh **do**
  add the face point $\mathbf{f}$ to *vertices*.
  for each edge e **do**
    add the edge point $\mathbf{e} = \frac{1}{4}(\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{f} + \mathbf{f}')$ to *vertices* where $\mathbf{v}_0$ and $\mathbf{v}_1$ are the endpoints of the edge, $\mathbf{f}$ is the center of the current face, and $\mathbf{f}'$ is the center of the other face in the mesh sharing edge e. Thus the edge point $\mathbf{e}$ is the average of the edge endpoints and the face points of faces containing the edge.
  end for
  for each vertex $\mathbf{v}$ **do**
    add $\mathbf{v}' = \frac{\overline{\mathbf{f}}+2\overline{\mathbf{m}}+(n-3)\mathbf{v}}{n}$ to *vertices* where $\mathbf{v}$ is the current vertex, $\overline{\mathbf{f}}$ is the average of all faces containing $\mathbf{v}$, and $\overline{\mathbf{m}}$ is the average of the midpoints of all edges containing $\mathbf{v}$. These are the **actual** midpoints, not the edge points, $\mathbf{e}$'s, computed above. n is the valence of $\mathbf{v}$.
  end for

for each vertex $\mathbf{v}'$ **do**
add a quad to *quads* with vertices: the old face point $\mathbf{f}$, the vertex to the "left" of $\mathbf{v}'$, $\mathbf{v}'$, and the vertex to the "right" of $\mathbf{v}'$.
end for
end for
update the mesh to use *vertices* and *quads*.

See the lecture notes on subdivision surfaces for more information.

The goal of this lab will be to convert the simple subdividing algorithm listed in § 1, into the Catmull-Clark algorithm listed above. All modifications will take place in `mesh.cpp`. **It is highly recommended that you look through the provided Mesh class before you begin!**

The New Edge Centers. The first step is to correct the edge points that are created. For the code given, these are $\mathbf{m} = \frac{1}{2}(\mathbf{v}_0 + \mathbf{v}_1)$ where $\mathbf{v}_0$ and $\mathbf{v}_1$ are the edge endpoints. Search for `// TASK 0 //` to find where this is done and change it to be $\mathbf{e} = \frac{1}{4}(\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{f} + \mathbf{f}')$ where $\mathbf{v}_0$ and $\mathbf{v}_1$ are the endpoints of the edge, $\mathbf{f}$ is the center of the current face, and $\mathbf{f}'$ is the center of the other face in the mesh sharing edge e. Thus the edge point $\mathbf{e}$ is the average of the edge endpoints and the face points of faces containing the edge.

When you complete this part try running the program. When you subdivide the mesh the face points and edge midpoints will be set correctly. However, the edge endpoints will not move. This should create a "spiky ball."

The New Edge-Endpoint Positions. At this point the mesh smooths nicely, except for those pesky corners. A few lines below TASK 0, there is a line that updates the edge endpoints by calling `newVertexForSmooth(Vertex &v)`. The next part is to implement this function. It takes a vertex $\mathbf{v}$ and calculates its new position as $\mathbf{v}' = \frac{\overline{\mathbf{f}}+2\overline{\mathbf{m}}+(n-3)\mathbf{v}}{n}$ where $\mathbf{v}$ is the current vertex, $\overline{\mathbf{f}}$ is the average of all faces containing $\mathbf{v}$, and $\overline{\mathbf{m}}$ is the average of the midpoints of all edges containing $\mathbf{v}$. These are the **actual** midpoints, not the edge points $\mathbf{e}$'s computed in TASK 0.

Average the Containing Faces. Look for `// TASK 1 //` and for now, just set $\mathbf{v}' = \overline{\mathbf{f}}$, the average of the face points.

Now go to `// TASK 2//`. It is inside the function that averages the face points for faces containing the given vertex. The function `facesWithVertex(XVec3f &v, unsigned int &count, unsigned int *buffer, unsigned int bufferSize)` is used to get all of the edges containing the vertex `v`; `count` will be the number of faces found (it will absolutely be 2!); `buffer` represents a buffer which will be filled with indices of the faces. So if `buffer`, after the call, contains 7 and 49, that means that `_quads[7]` is a face containing the vertex in question (and the same for 49).

Average the face points found (you should use `centerOfQuad`) and return the result. Be sure to average position, normal, and color.

If you run the program now, you should see the corners get pulled in really far (because averaging the face points is far too extreme on its own).

Average the Containing Edges Go to `// TASK 3 //` and change it so that $\mathbf{v}' = \overline{\mathbf{m}}$, the average of the edge midpoints.

Now go to `// TASK 4 //`. It is inside the function that averages the edge midpoints for the edges containing the given vertex. The function `edgesWithVertex(XVec3f &v, unsigned int &count, unsigned int *buffer, unsigned int bufferSize)` is used to get all the faces containing the vertex `v`. It acts much the same as its face equivalent, except that it fills the buffers with pairs, and `count` is the number of **pairs**, not elements. So if `buffer` contains, after the function is called, (5, 11, 12, 19, 43, 18) then the edge made of `_vertices[5]` and `_vertices[11]` contains the point in question. In this example `count` will be 3 and **not** 6.

Average the midpoints of the edges found and return the result. Be sure to average position, normal, and color.

If you run the program now, you should see the corners get pulled in significantly less than for the faces, but after performing a few subdivisions, you will notice that this is not a pleasing result either.

Weight the Pieces. Go to `// TASK 5 //` and change it so that $\mathbf{v}' = \frac{\bar{\mathbf{f}}+2\overline{\mathbf{m}}+(n-3)\mathbf{v}}{n}$. Run the program and observe the smoothing! It may help to think of what this formula means for triangles ($n = 2$) and for quads ($n = 3$).

The New Normals. You may notice that the final result produces pleasantly smooth meshes. However in some locations the normals may give the appearance of a sharp *edge*. After performing a subdivision average the normals so that the mesh is smooth in shape and "lighting."

You should be able to loop over all faces, calculate the normal, add it to each vertex, and then normalize at the end, as in PA2. However, the algorithm does not produce planar faces (why?), so it is difficult to find a face-normal. This can be easily remedied by imagining the quad as two triangles. Compute the normal of each triangle and then add that to the two vertices opposite from the dividing diagonal. Then, do the same with the other diagonal and the other two vertices.