CS5625 Programming Assignment 5: Subdivision Surfaces

Due: April 17 (Wednesday), 11:59 pm.
Work in groups of 2.

Instructor: Kavita Bala

1 Overview

The goal of this assignment is to implement subdivision surfaces. You will implement:

- Loop
- Catmull-Clark

For both subdivisions you should support boundary edges, crease edges, and extraordinary vertices.

2 What you need to do

The only files that you will need to edit are LoopSubdiv.java and CCSubdiv.jaja. For both Loop and Catmull-Clark, the constructor call should create a mesh that is subdivided once by the approriate algorithm.

We encourage you to tesh this with other, more complicated meshes. The code to do this is in the initializer of the scene controller.

Hitting the 'n' key subdivides the mesh, and hitting the 'm' key switches between Loop and Catmull-Clark.

3 What we provide

To perform subdivision, we need a more sophisticated mesh data structure. They come in many varieties including a half edge data structure or winged edge data structure etc. We have implemented an edge data structure inspired by the winged edge data structure. See below for a detailed specification of our edge data structure. We also provide the code that inputs a mesh (of triangles or quads) and creates the edge data structure that corresponds to that mesh.

The EdgeDS class has constructors that take a Trimesh or a Quadmesh and convert them into it's internal representation. You will have to write the code that uses EdgeDS and performs subdivision. Since multiple levels of subdivison are expected, but we want to see mesh subdivision incrementally, be sure to have a mechanism to extract a Trimesh or a Quadmesh after only one level of subdivision. Your code will follow this outline (this is for Loop; Catmull-Clark is similar):

```
// create edgeDS from old mesh
EdgeDS edgeDS = new EdgeDS(testMesh);
```

```
//subdivide it
LoopSubdiv loopSubdiv = new LoopSubdiv(edgeDS);

//create new mesh
Trimesh newMesh = (Trimesh)loopSubdiv.getNewMesh();
```

This is only a guideline. Look at the **SubdivSceneController** class. Feel free to change any code that you have been provided, or follow a different path for your implementation. But the expected functionality should be there, no matter which way you choose to proceed. Our released code does not do any subdivision. Instead it just returns the original mesh in the call above.

All the integer parameters (or new keyboard shortcuts) we need to change to test your programs must be cleanly isolated in one place.

4 EdgeDS

The EdgeDS holds the data structure needed for subdivision. The EdgeDS class has functionality to access edges, vertices, and faces. The EdgeDS can be created by supplying a Trimesh or a Quadmesh object. Each vertex, face, or edge is assigned a unique integer ID.

All the per-edge data, per-face data, per-vertex data are stored internally as TreeMaps. That is, a key value pair where the keys are vertexIDs/edgeIDs/polygonIDs and values are VertexData/EdgeData/PolygonData. Look at the individual EdgeData/PolygonData/VertexData classes below for more information on what data is stored per edge/vertex/polygon.

4.1 Useful Funcitons

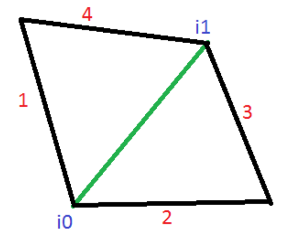
You will need the edgeID/polygonID/vertexID to lookup the corresponding edge/polygon/vertex data. To iterate over all edges/vertices/polygons, you will need all the edgeIDs, all the vertexIDs and all the polygonIDs. For this, use getEdgeIDs(), getVertexIDs(), and getPolygonIDs() respectively. Helper functions are provided to perform lookups using these IDs. Look at getEdgeData(), getVertexData() and getPolygonData().

4.1.1 Lookup Functions

The functions listed above do simple lookups. They either return all the keys of the Maps or a value for an individual key. There are two more functions provided, which do more involved lookups. These are:

- getOtherEdgesOfLeftFace(): Given an edgeID, it will return all the edges for the left polygon on that edge in clockwise order. You can walk this list one by one using a for loop and traverse all the edges of the left polygon. For simplicity, the edgeID that is passed in as an argument will not be included in the returned list. The first edge in the list is guaranteed to meet the vertex with id $i0$ of the input edgeID.
- getOtherEdgesOfRightFace(): Same as aove, but for the right polygon. *Note:* since boundary edges have only one polygon attached to them, calling this function on a boundary edge will return an empty list.

Consider the functions in the context of the mesh shown below.



Say we want to get all the edges of the polygon to the left of the green edge. The edgeIDs are shown in red. For the green edge, the vertices i0 and i1 are shown. In this instance, getOtherEdgesOfLeftFace() will return either [1,4] or [2,3], depending upon which polygon was deemed to be the left polygon.

4.2 EdgeData

This class has the per edge data. That is, the two vertices that it connects, a placeholder for the edge vertexID that will be created during subdivison and a list of polygons that share this edge. Note that the getOtherEdgesOfLeftFace() and getOtherEdgesOfRightFace() functions of EdgeDS class return a list of edges starting from i0.

4.3 VertexData

This class has the per vertex data as part of our edgeDS. That is, the vertex attributes (like position), a list of (vertex,edge) pairs that represent this vertex's connectivity and a placeholder for the new even vertex that will be created during subdivision

Note: If a vertex v0 is connected to vertex v1 via edge e0, then the pair (e0,v1) will be stored in v0 and the pair (e0,v0) will be stored in v1.

Some useful functions:

- getConnectedVertices(): This will return a list of all vertices that are connected with this vertex.
- getConnectedEdges(): This will return a list of all edges that are connected with this vertex.

4.4 PolygonData

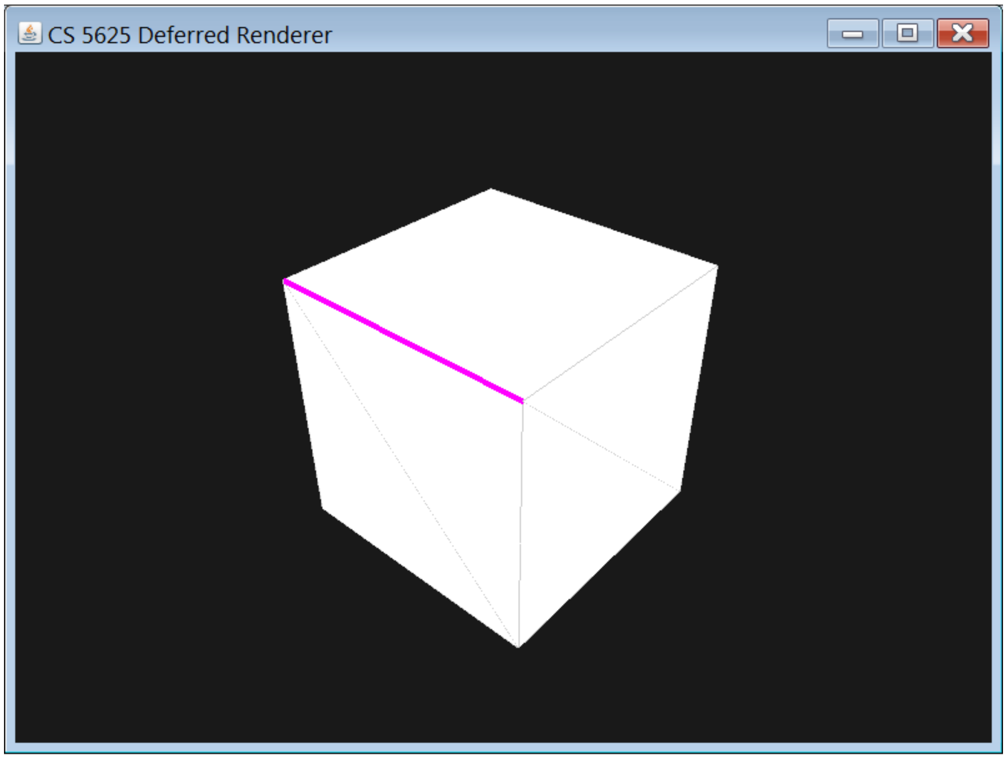
This class holds the per polygon data as part of our edgeDS. It has the list of vertices that define this polygon, and a list of edges that define it's boundary and a placeholder for the new face vertex that will be created during Catmull Clark.

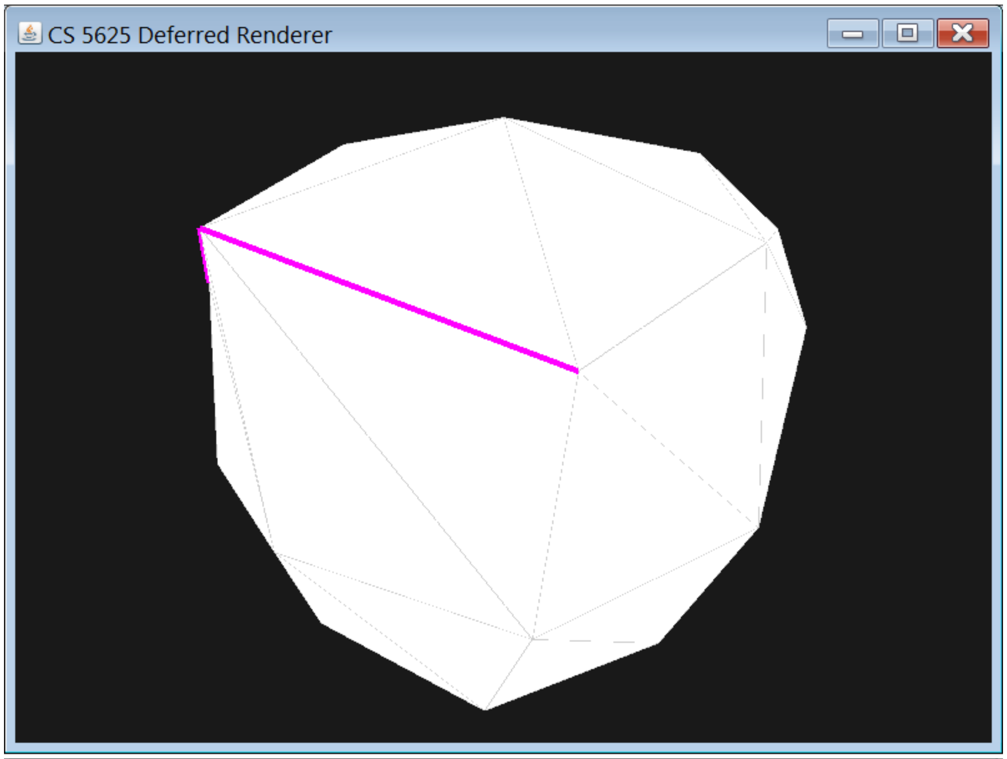
5 Testing your code

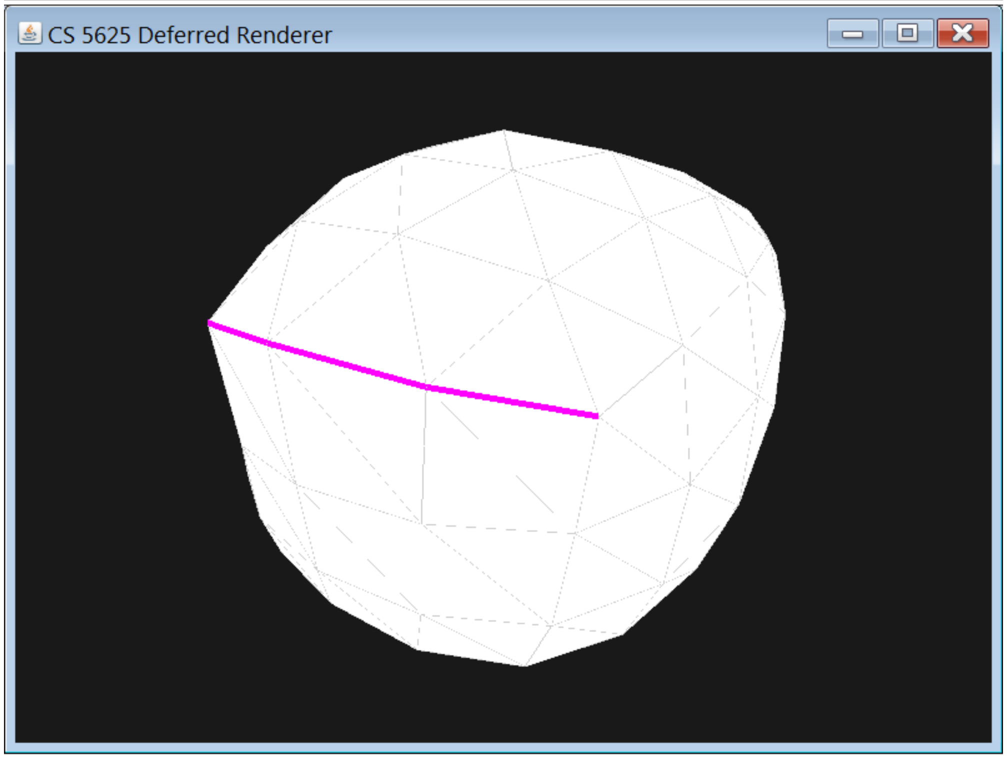
The defaultGeometry folder has some simple meshes. Try your subdivision code on those meshes first. None of them have creases. Also try with the example.obj file in the models directory. It has a cube and one of it's edges is a crease edge. Remember to make it easy for us to load new meshes while testing.

5.1 Example Output

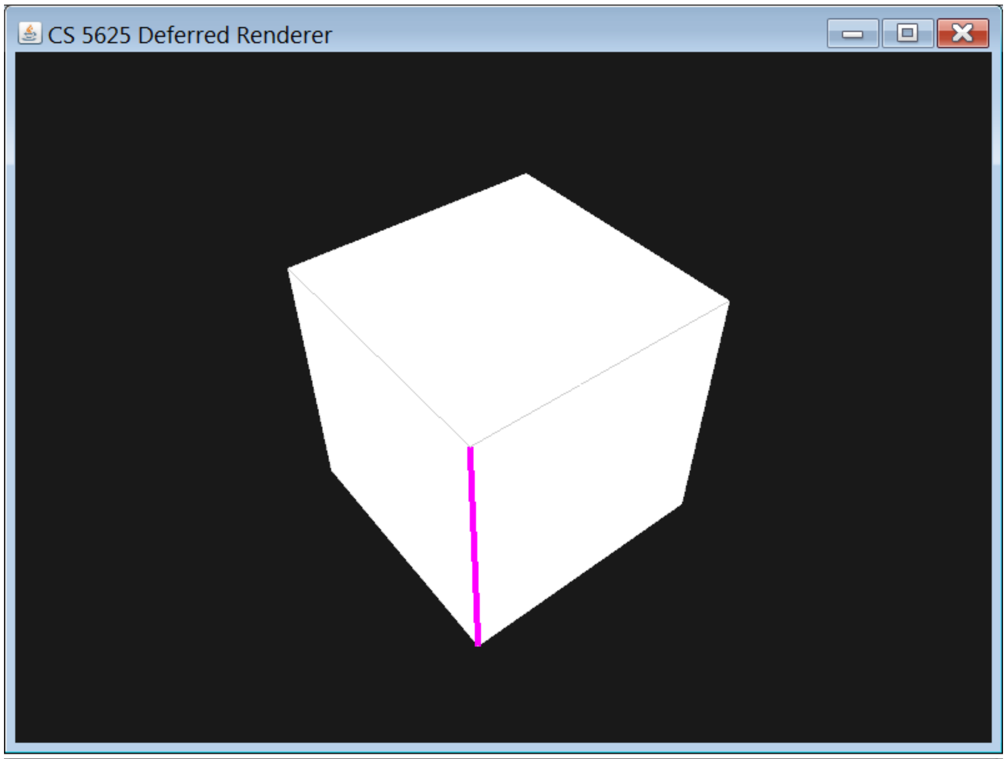
Here are two levels of Loop subdivision using the tetrahedron shape in default geometry.

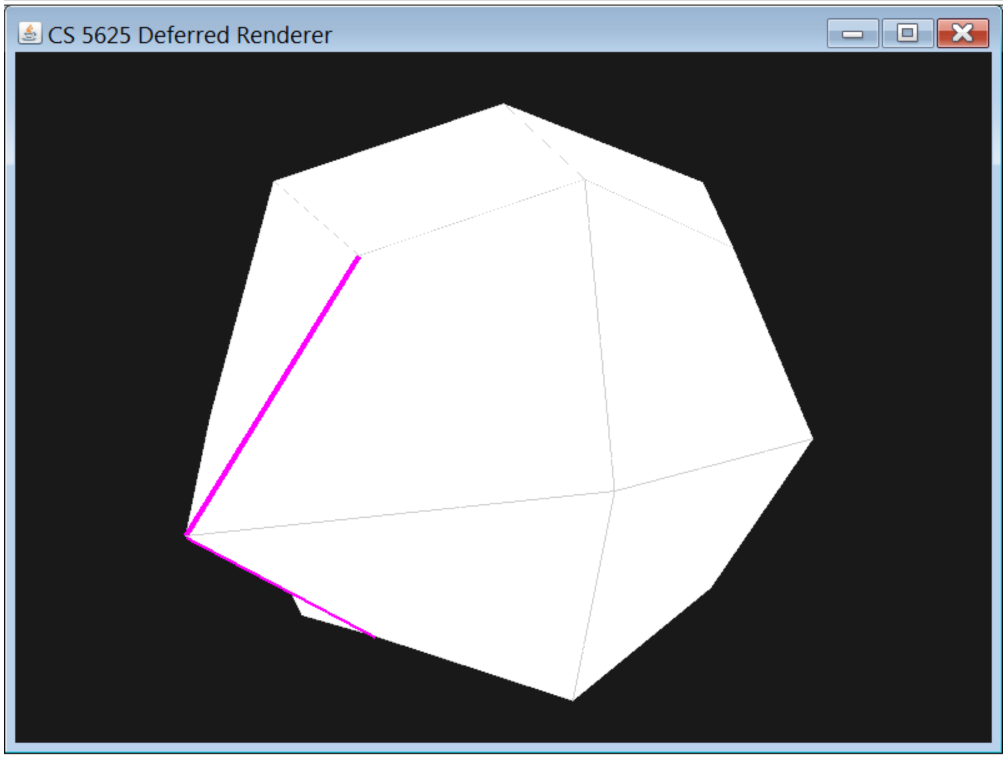


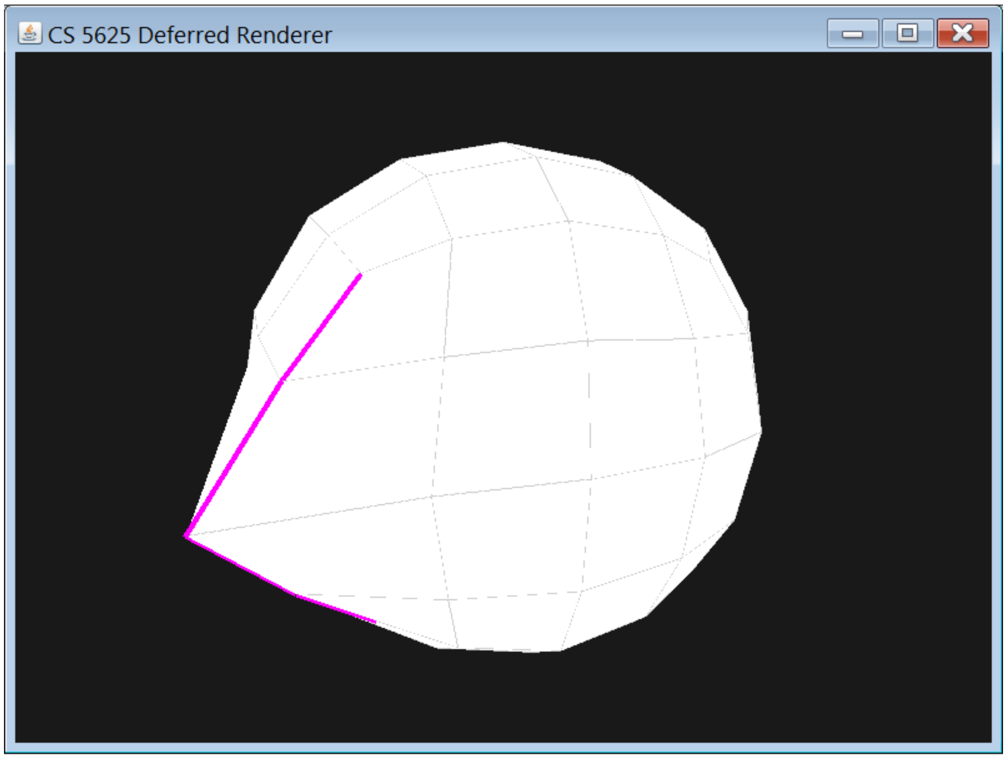




Here are two levels of Catmull-Clark subdivision using the CubeQuadMesh shape in default geometry.


CS 5625 Deferred Renderer


CS 5625 Deferred Renderer



6 What To Submit

You should submit your completed subdivision framework, along with a README file describing any implementation choices you made or difficulties you encountered. If you make any fancy new scenes, feel free to include those (and screenshots) as well. If you include extra keyboard shortcuts or hardcoded integer values, you must document them clearly in the CHANGME file.

Remember to keep the stuff in the project that is not part of the source code as well. This includes the bin/lib folder and the metadata the Eclipse needs to import the project.