```
I have developed/implemented loopsubdivision algorithm in javascript.
Below is the source code that have been changed/added in the code base of
step 3 of 1st assignment.


/**
 * Created  on 9/27/15.
 */

//addition of new functions in SoftEngine.js
Device.prototype.createNewMeshFromLoopSubdivision= function (mesh)
{
    var edgeCount= createEdgeMap(mesh);
    return loopSubDivision(mesh,edgeCount);
};
function loopSubDivision(cMesh,edgeCount){
    /**
     * implement loops subdivision refining step
     */
    var name=cMesh.name;
    var vertices=cMesh.Vertices;
    var verticesCount=cMesh.Vertices.length;
    var facesCount=cMesh.Faces.length;
    // var edgesCount=cMesh.edgesCount;//if I will get an idea to calculate an edge I can
 use it
    var newMesh=new SoftEngine.Mesh(name, verticesCount+edgeCount, 4*
facesCount);
    var newVertices=newMesh.Vertices;
    var newFacesIndex=0;

    var childMap=new Map();
    var currentEdgeArray=Array.from(edgeMap.values);
    for (var indexFaces = 0; indexFaces < cMesh.Faces.length; indexFaces++) {
        var currentFace = cMesh.Faces[indexFaces];
        var vertexA = currentFace.A;
        var vertexB = currentFace.B;
        var vertexC = currentFace.C;
        var newVertexAB,newVertexAC,newVertexBC;
        var keyAB=getKey(vertexA,vertexB);
        if(childMap.has(keyAB))
        {
            //edge is already visited
            newVertexAB=childMap.get(keyAB);
        }
        else{
            var newAB=refiningNewVertex(vertices,vertexA,vertexB);
            newVertexAB=verticesCount;
            newVertices[verticesCount++]=newAB;
            childMap.set(keyAB,newVertexAB);
        }
        var keyAC=getKey(vertexA,vertexC);
        if(childMap.has(keyAC))
        {
            //edge is already visited
            newVertexAC=childMap.get(keyAC);
        }
        else{
            var newAC=refiningNewVertex(vertices,vertexA,vertexC);
            newVertexAC=verticesCount;
```

```
58              newVertices[verticesCount++]=newAC;
59              childMap.set(keyAC,newVertexAC);
60          }
61          var keyBC=getKey(vertexB,vertexC);
62          if(childMap.has(keyBC))
63          {
64              //edge is already visited
65              newVertexBC=childMap.get(keyBC);
66          }
67          else{
68              var newBC=refiningNewVertex(vertices,vertexB,vertexC);
69              newVertexBC=verticesCount;
70              newVertices[verticesCount++]=newBC;
71              childMap.set(keyBC,newVertexBC);
72          }
73
74          //add new verteices in newVertices array
75          newMesh.Faces[newFacesIndex++]={
76              A: vertexA,
77              B: newVertexAB,
78              C: newVertexAC
79          };
80          newMesh.Faces[newFacesIndex++]={
81              A: newVertexAB,
82              B: vertexB,
83              C: newVertexBC
84          };
85          newMesh.Faces[newFacesIndex++]={
86              A: newVertexBC,
87              B: vertexC,
88              C: newVertexAC
89          };
90          newMesh.Faces[newFacesIndex++]={
91              A: newVertexAC,
92              B: newVertexAB,
93              C:newVertexBC
94          };
95
96      }
97      smoothening(newVertices,vertices);
98      return newMesh;
99
100 };
101
102 function getKey(v1,v2)
103 {
104     if(v1>v2)
105     {
106         var temp=v1;
107         v1=v2;
108         v2=temp;
109     }
110
111     var key=v1+','+v2;
112     return key;
113
114 };
115 function refiningNewVertex(vertices,v1,v2){
116
```

```
117     var sum=(vertices[v1].add(vertices[v2]));
118
119
120     var p1adjSet=edgeMap.get(v1);
121     var p2adjSet=edgeMap.get(v2);
122     //I noticed in the mesh only 2 opposite vertices are attached to each edge
123     var insectSet=intersectionSet(p1adjSet, p2adjSet);
124     var intersectArray=Array.from(insectSet);
125     if(intersectArray.length>2)
126        console.log(v1+','+v2+' '+intersectArray);
127     var intersectSum= new BABYLON.Vector3(0,0,0);
128     if(intersectArray.length==1)
129     {
130        var v3=intersectArray[0];
131        //boundary edge: need to find reflection of
132        var p4=reflectPoint(vertices[v3],vertices[v1],vertices[v2]);
133        intersectSum=p4.add(vertices[v3]);
134        //returning midpoint of the 2 points
135        //  return sum.scale(0.5);
136        console.log(v1+','+v2+' '+intersectArray);
137
138     }
139     else{
140        //mostly only 2 opposite vertices are attached to each edge
141        insectSet.forEach(function(value) {
142           intersectSum=intersectSum.add(vertices[value]);
143        });
144
145     }
146     sum=sum.scale(3);
147     sum=sum.add(intersectSum);
148     return sum.scale(1/8);
149  };
150
151  function smoothening(newVertices,oldvertices)
152  {
153     for(index=0;index<oldvertices.length;index++)
154        // newVertices[index]=oldvertices[index];
155        newVertices[index]=refiningCurrentVertex(oldvertices,index);
156
157  };
158  function refiningCurrentVertex(vertices,a){
159     var adjacencyVertices= edgeMap.get(a);
160     //console.log('a='+a);
161     //console.log(adjacencyVertices);
162     var n=adjacencyVertices.size;
163     //var B=3/(n*(n+2));//B is beta
164     var theta=Math.PI*(2/n);
165     var h=Math.cos(theta);
166     var m=(3+2*h);
167     var q=m*m/64;
168     var B=(0.625-q)/n;
169
170     var newA=vertices[a].scale(1-n*B);
171     var sum=new BABYLON.Vector3(0, 0, 0);
172     adjacencyVertices.forEach(function(adVertex) {
173        sum=sum.add(vertices[adVertex]);
174     });
175     sum=sum.scale(B);
```

```
176     return newA.add(sum);
177
178 };
179
180
181
182 // function refiningCurrentVertex(vertices,a){
183 //                 return vertices[a];
184 //             };
185 Device.prototype.LoadJSONFileAsync = function (fileName, callback) {
186     var jsonObject = {};
187     var xmlhttp = new XMLHttpRequest();
188     xmlhttp.open("GET", fileName, true);
189     var that = this;
190     xmlhttp.onreadystatechange = function () {
191         if (xmlhttp.readyState == 4 && (xmlhttp.status == 200 || xmlhttp.status == 0)) {
192             var allText = xmlhttp.responseText;
193             jsonObject = JSON.parse(allText);
194             callback(that.CreateMeshesFromJSON(jsonObject));
195         }
196     };
197     xmlhttp.send(null);
198 };
199
200 function setNewEdgeMap(edgeMap, a, b, c) {
201     var adjacencySet;
202     if(edgeMap.has(a))
203     {
204         adjacencySet=edgeMap.get(a);
205     }
206     else{
207         adjacencySet=new Set();
208         edgeMap.set(a,adjacencySet);
209     }
210     adjacencySet.add(b);
211     adjacencySet.add(c);
212 }
213 function createEdgeMap(cMesh){
214     var edgeCount=0;
215     edgeMap=new Map();
216     for (var indexFaces = 0; indexFaces < cMesh.Faces.length; indexFaces++)
217     {
218         var currentFace = cMesh.Faces[indexFaces];
219         var a=currentFace.A;
220         var b=currentFace.B;
221         var c=currentFace.C;
222         setNewEdgeMap(edgeMap, a, b, c);
223
224         setNewEdgeMap(edgeMap, b, c,a);
225
226         setNewEdgeMap(edgeMap,c,a, b) ;
227
228     }
229     for (var value of edgeMap.values()) {
230         // console.log(value);
231
232         edgeCount=edgeCount+value.size;
233
234     };
```

```
235     //console.log(edgeCount/2);
236     return edgeCount/2;//every vertex twice for an edge
237 };
238
239
240
241
242 //Code in main.js
243 //changed or modified in order to create new meshes
244
245 //Addition of following function for adding subdivision meshes on the +, - keypress
246
247 window.requestAnimationFrame = (function () {
248     return window.requestAnimationFrame ||
249         window.webkitRequestAnimationFrame ||
250         window.mozRequestAnimationFrame ||
251         function (callback) {
252             window.setTimeout(callback, 1000 / 60);
253         };
254 })();
255
256 var canvas;
257 var divCurrentFPS;
258 var divAverageFPS;
259 var device;
260 var meshes = [];
261 var mera;
262 var previousDate = Date.now();
263 var lastFPSValues = new Array(60);
264 var currentMesh;
265 var currentIndex=0;
266
267 document.addEventListener("DOMContentLoaded", init, false);
268
269 function init() {
270     divCurrentFPS = document.getElementById("currentFPS");
271     divAverageFPS = document.getElementById("averageFPS");
272
273     canvas = document.getElementById("frontBuffer");
274     mera = new SoftEngine.Camera();
275     device = new SoftEngine.Device(canvas);
276     mera.Position = new BABYLON.Vector3(0, 0, 10);
277     mera.Target = new BABYLON.Vector3(0, 0, 0);
278     device.LoadJSONFileAsync("monkey.babylon", loadJSONCompleted);
279 }
280
281 document.onkeydown = function(e) {
282
283     if(e.keyCode==38|| e.keyCode==108|| e.keyCode==171)
284     {
285
286         if(currentIndex<=4)
287         {
288             currentIndex++;
289             if(typeof meshes[currentIndex] === 'undefined') {
290                 meshes[currentIndex] = device.createNewMeshFromLoopSubdivision(
    meshes[currentIndex-1]);
291             }
292             currentMesh=meshes[currentIndex];
```

```
293         }
294         alert('up '+(currentIndex));
295 
296     }
297 
298 
299     if(e.keyCode==40||  e.keyCode==109||e.keyCode==189) {
300 
301         if(currentIndex>0)
302         {
303             currentIndex--;
304             currentMesh=meshes[currentIndex];
305         }
306         alert('down to'+(currentIndex));
307     }
308 
309 };
310 
311 
312 function loadJSONCompleted(meshesLoaded) {
313     meshes = meshesLoaded;
314     currentMesh=meshes[0];
315     // Calling the HTML5 rendering loop
316     requestAnimationFrame(drawingLoop);
317 }
318 
319 // Rendering loop handler
320 function drawingLoop() {
321     //calculation of fps
322     var now = Date.now();
323     var currentFPS = 1000 / (now - previousDate);
324     previousDate = now;
325 
326     divCurrentFPS.textContent = currentFPS.toFixed(2);
327 
328     if (lastFPSValues.length < 60) {
329         lastFPSValues.push(currentFPS);
330     } else {
331         lastFPSValues.shift();
332         lastFPSValues.push(currentFPS);
333         var totalValues = 0;
334         for (var i = 0; i < lastFPSValues.length; i++) {
335             totalValues += lastFPSValues[i];
336         }
337 
338         var averageFPS = totalValues / lastFPSValues.length;
339         divAverageFPS.textContent = averageFPS.toFixed(2);
340     }
341 
342     device.clear();
343 
344     // rotating slightly the mesh during each frame rendered
345     currentMesh.Rotation.x += 0.01;
346     currentMesh.Rotation.y += 0.01;
347 
348 
349     // Doing the various matrix operations
350     device.render(mera, currentMesh);
351     // Flushing the back buffer into the front buffer
```

```
352     device.present();
353
354     // Calling the HTML5 rendering loop recursively
355     requestAnimationFrame(drawingLoop);
356 }
357
358 //util methods created to support implementation of Loop Subdivision
359 //present in  a new file called util.js
360
361 /**
362  * Created on 9/26/15.
363  */
364
365 function intersectionSet(setA, setB){
366     // Iterate set entries with forEach
367     var iset=new Set();
368     setA.forEach(function(value) {
369         if(setB.has(value)){
370             iset.add(value);
371         }
372     });
373     return iset;
374
375 }
376
377 /**
378  * @brief Reflect point p along line through points p0 and p1
379  *
380  * @param p point to reflect
381  * @param p0 first point for reflection line
382  * @param p1 second point for reflection line
383  * @return object
384  */
385 function reflectPoint(p, p0, p1) {
386     var dx, dy, a, b, x, y;
387
388     dx = p1.x - p0.x;
389     dy = p1.y - p0.y;
390     a = (dx * dx - dy * dy) / (dx * dx + dy * dy);
391     b = 2 * dx * dy / (dx * dx + dy * dy);
392     x = Math.round(a * (p.x - p0.x) + b * (p.y - p0.y) + p0.x);
393     y = Math.round(b * (p.x - p0.x) - a * (p.y - p0.y) + p0.y);
394
395     return new BABYLON.Vector3(x, y, p0.z);
396 }
397
398
399 Problems Faced:
400 1. Selecting efficient datastructures well suited for Book keeping of edges and vertices
    was big difficulty.
401 2. Calculating exact number of vertices required in the new meshes.
402 3. Once I decided the edges to store in map deciding exact keys, and fiding any edge
    on the map with v1 and v2
403 I dont find the key in map with flipped v2 and v1 which appears in another face. So I
    implemented simple trick of making v1
404 the least index.
405 4. Indexing of the faces should be in definite order either clockwise or anticlockwise
    otherwise it disturbs rendering.
406 5. Finding 4th point opposite to an edge of a face was a difficulty so I decided to get
```

406 adjacency intersection of the adjacency sets vertices of edge containing vertices.
407
408
409 Things I learned:
410 1. To efficiently store edges and vertices **in** mesh objects.
411 2. To implement loops algorithm
412 3. To implement efficient rendering of meshes
413