Paul Mallard & Pauline Dos Santos E4FI group 2

Question 6 : Write the pseudo-code for the Catmull-Clark subdivision algorithm.

Based on the Catmull-Clark subdivision algorithm we need to be careful too :

1. Add a face point at the centroid of each face
2. Add an edge point for each edge, averaged from:
    - the two endpoints of the edge
    - and the two face points on either side of the edge
3. Reposition old vertices using :

$$\upsilon_{new} = \frac{F}{n} + \frac{2R}{n} + \frac{P\,(n-3)}{n} = \frac{F + 2R + P\,(n-3)}{n}$$

    Where :
    - F : average of adjacent face points
    - R : average of midpoints
    - P : original position
    - n : valence of vertex

4. Rebuild the mesh :
    - new face points
    - new edge points
    - repositioned original vertices

```
void myMesh::subdivisionCatmullClark() {
   std::vector<myFace*> newFaces;
   std::vector<myHalfedge*> newHalfedges;
   std::vector<myVertex*> newVertices;
   std::map<myFace*, myVertex*> facePoint;
   std::map<myHalfedge*, myVertex*> edgePoint;
   std::map<myVertex*, myPoint3D> newPos;

   for (auto* f : faces) {
      myPoint3D centroid(0, 0, 0);
      int count = 0;
      auto* start = f->adjacent_halfedge;
      auto* h = start;
      do {
         centroid += *h->source->point;
         count++;
         h = h->next;
      } while (h != start);

      myPoint3D center = centroid / float(count);
      myVertex* fp = new myVertex();
```

```
    fp->point = new myPoint3D(center);
    facePoint[f] = fp;
    newVertices.push_back(fp);
}


std::set<myHalfedge*> processedEdges;
for (auto* e : halfedges) {
    if (!e->twin || processedEdges.count(e)) continue;

    myPoint3D mid = (*e->source->point + *e->next->source->point) * 0.5f;
    myPoint3D f1 = *facePoint[e->adjacent_face]->point;
    myPoint3D f2 = *facePoint[e->twin->adjacent_face]->point;
    myPoint3D avg = (mid + (f1 + f2) * 0.5f) * 0.5f;

    myVertex* halfedgePoint = new myVertex();
    halfedgePoint->point = new myPoint3D(avg);
    edgePoint[e] = edgePoint[e->twin] = halfedgePoint;
    newVertices.push_back(halfedgePoint);

    processedEdges.insert(e);
    processedEdges.insert(e->twin);
}

for (auto* v : vertices) {
    std::vector<myHalfedge*> adjacentHalfedges;
    auto* start = v->originof;
    auto* h = start;
    do {
        adjacentHalfedges.push_back(h);
        if (!h->twin) break;
        h = h->twin->next;
    } while (h!= start);

    int n = adjacentHalfedges.size();
    if (n == 0) continue;

    myPoint3D F(0,0,0), R(0,0,0);
    for (auto* e : adjacentHalfedges) {
        F += *facePoint[e->adjacent_face]->point;
        R += (*e->source->point + *e->next->source->point) * 0.5f;
    }
    F /= float(n);
    R /= float(n);
    myPoint3D P = *v->point;

    newPos[v] = (F + R * 2.0f + P * (float(n) - 3.0f) ) / float(n);
}
```

```
for (auto* f : faces) {
    auto* start = f->adjacent_halfedge;
    auto* h = start;
    do {
        myVertex*  h1= h->source;
        myVertex*  ep1= edgePoint[h];
        myVertex*  facePt= facePoint[f];
        myVertex*  ep0= edgePoint[h->prev];

        myFace* newF = new myFace();
        std::vector<myHalfedge*> he(4);
        for (int i = 0; i < 4; ++i) he[i] = new myHalfedge();

        he[0]->source = h1;
        he[1]->source = ep1;
        he[2]->source = facePt;
        he[3]->source = ep0;

        for (int i = 0; i < 4; ++i) {
            he[i]->adjacent_face = newF;
            he[i]->next = he[(i+1)%4];
            he[i]->prev = he[(i+3)%4];
            he[i]->source->originof = he[i];
        }

        newF->adjacent_halfedge = he[0];
        newFaces.push_back(newF);
        for (auto* h : he) newHalfedges.push_back(h);

        h = h->next;
    } while (h!= start);
}

std::map<std::pair<myVertex*, myVertex*>, myHalfedge*> edgeMap;
for (auto* e : newHalfedges) {
    auto key = std::make_pair(e->source, e->next->source);
    auto rev = std::make_pair(e->next->source, e->source);
    if (edgeMap.count(rev)) {
        e->twin = edgeMap[rev];
        edgeMap[rev]->twin = e;
    } else {
        edgeMap[key] = e;
    }
}

faces = std::move(newFaces);
halfedges = std::move(newHalfedges);
```

```
    vertices.insert(vertices.end(), newVertices.begin(), newVertices.end());

    for (auto& v : newPos) {
        *v.first->point = v.second;
    }

    checkMesh();
}
```