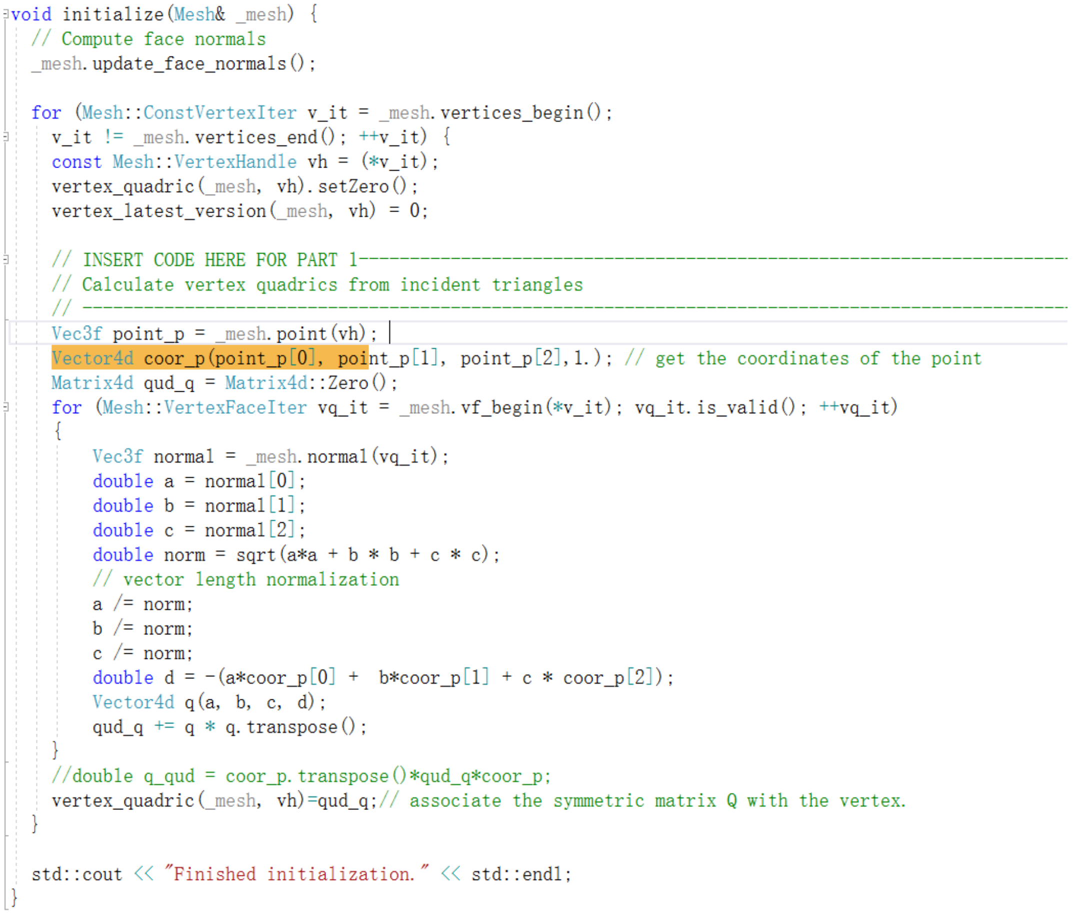
CS492 Homework4

XU YIN - 20205445

November 18, 2021

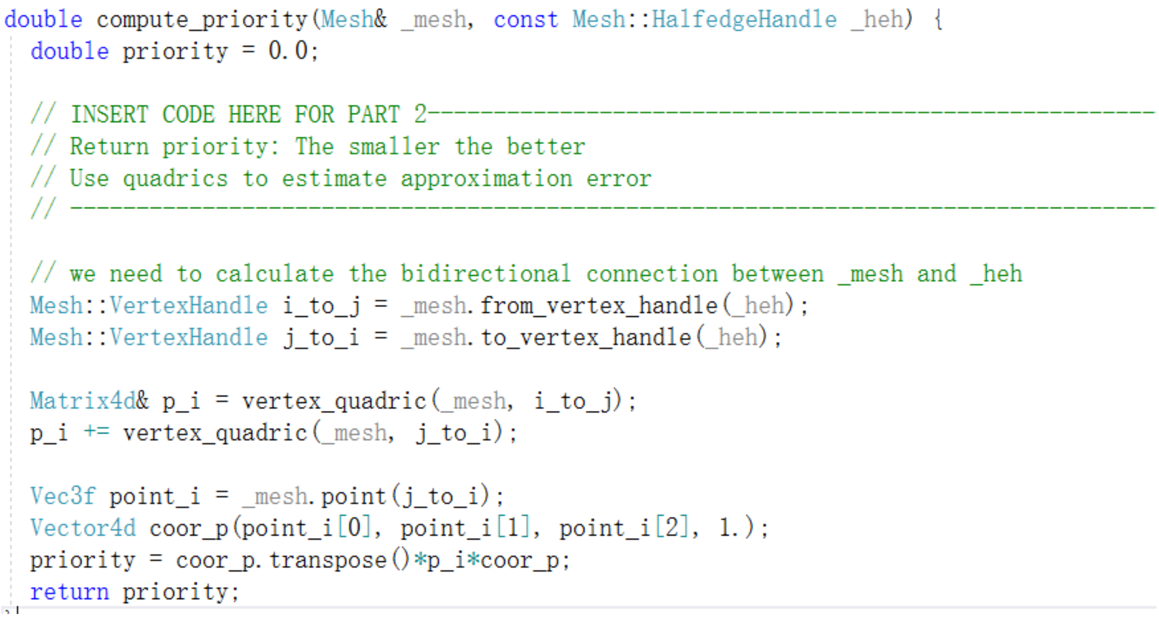
1 Complete the *initialize*() function (see Fig 1).



```
void initialize(Mesh& _mesh) {
  // Compute face normals
  _mesh.update_face_normals();

  for (Mesh::ConstVertexIter v_it = _mesh.vertices_begin();
    v_it != _mesh.vertices_end(); ++v_it) {
    const Mesh::VertexHandle vh = (*v_it);
    vertex_quadric(_mesh, vh).setZero();
    vertex_latest_version(_mesh, vh) = 0;

    // INSERT CODE HERE FOR PART 1-------------------------------------------------------------------
    // Calculate vertex quadrics from incident triangles
    // ---------------------------------------------------------------------------------------------
    Vec3f point_p = _mesh.point(vh);
    Vector4d coor_p(point_p[0], point_p[1], point_p[2],1.); // get the coordinates of the point
    Matrix4d qud_q = Matrix4d::Zero();
    for (Mesh::VertexFaceIter vq_it = _mesh.vf_begin(*v_it); vq_it.is_valid(); ++vq_it)
    {
        Vec3f normal = _mesh.normal(vq_it);
        double a = normal[0];
        double b = normal[1];
        double c = normal[2];
        double norm = sqrt(a*a + b * b + c * c);
        // vector length normalization
        a /= norm;
        b /= norm;
        c /= norm;
        double d = -(a*coor_p[0] +  b*coor_p[1] + c * coor_p[2]);
        Vector4d q(a, b, c, d);
        qud_q += q * q.transpose();
    }
    //double q_qud = coor_p.transpose()*qud_q*coor_p;
    vertex_quadric(_mesh, vh)=qud_q;// associate the symmetric matrix Q with the vertex.
  }

  std::cout << "Finished initialization." << std::endl;
}
```

Figure 1: The *initialize*() function.

2 Compute the priority of a halfedge (see Fig 2).



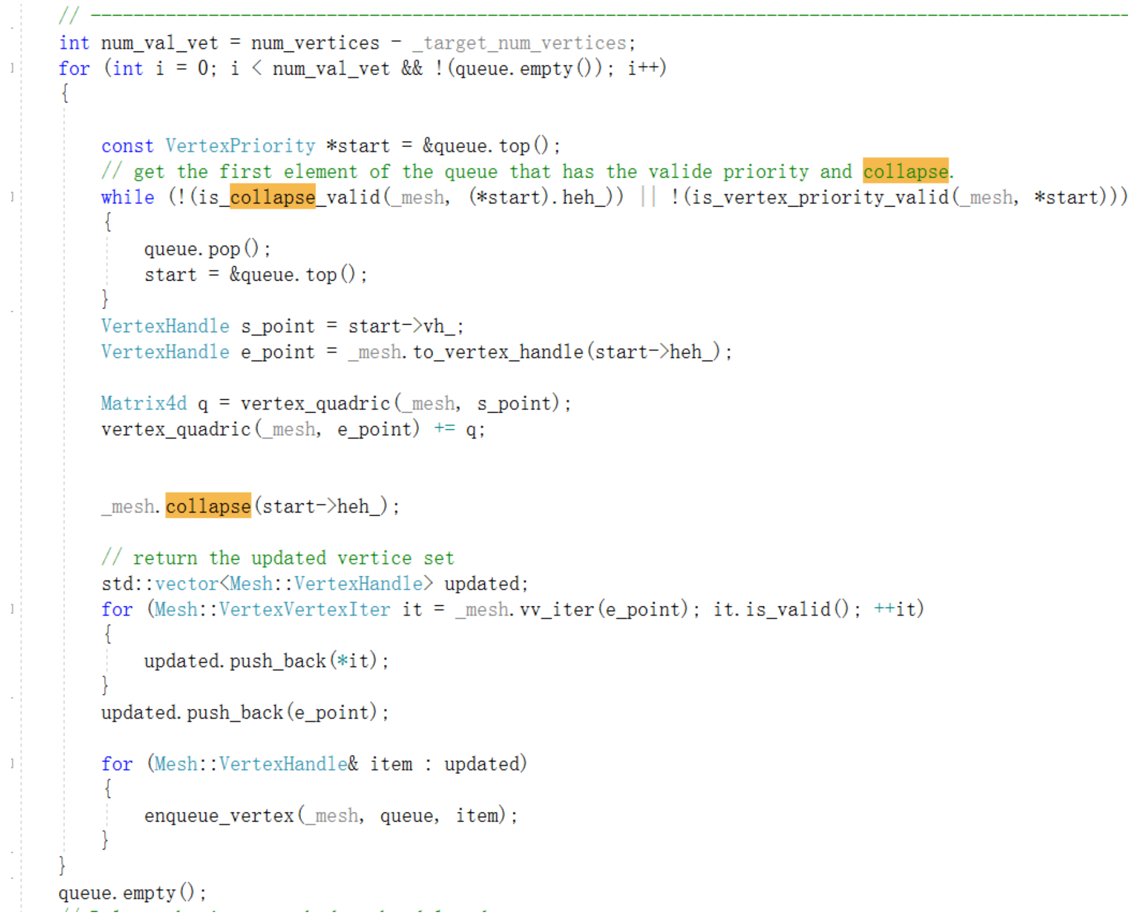
```
double compute_priority(Mesh& _mesh, const Mesh::HalfedgeHandle _heh) {
  double priority = 0.0;

  // INSERT CODE HERE FOR PART 2---------------------------------------------------------
  // Return priority: The smaller the better
  // Use quadrics to estimate approximation error
  // -------------------------------------------------------------------------------------

  // we need to calculate the bidirectional connection between _mesh and _heh
  Mesh::VertexHandle i_to_j = _mesh.from_vertex_handle(_heh);
  Mesh::VertexHandle j_to_i = _mesh.to_vertex_handle(_heh);

  Matrix4d& p_i = vertex_quadric(_mesh, i_to_j);
  p_i += vertex_quadric(_mesh, j_to_i);

  Vec3f point_i = _mesh.point(j_to_i);
  Vector4d coor_p(point_i[0], point_i[1], point_i[2], 1.);
  priority = coor_p.transpose()*p_i*coor_p;
  return priority;
```

Figure 2: Compute the priority of a halfedge.

3 Implement *decimate*() function (see Fig 4).

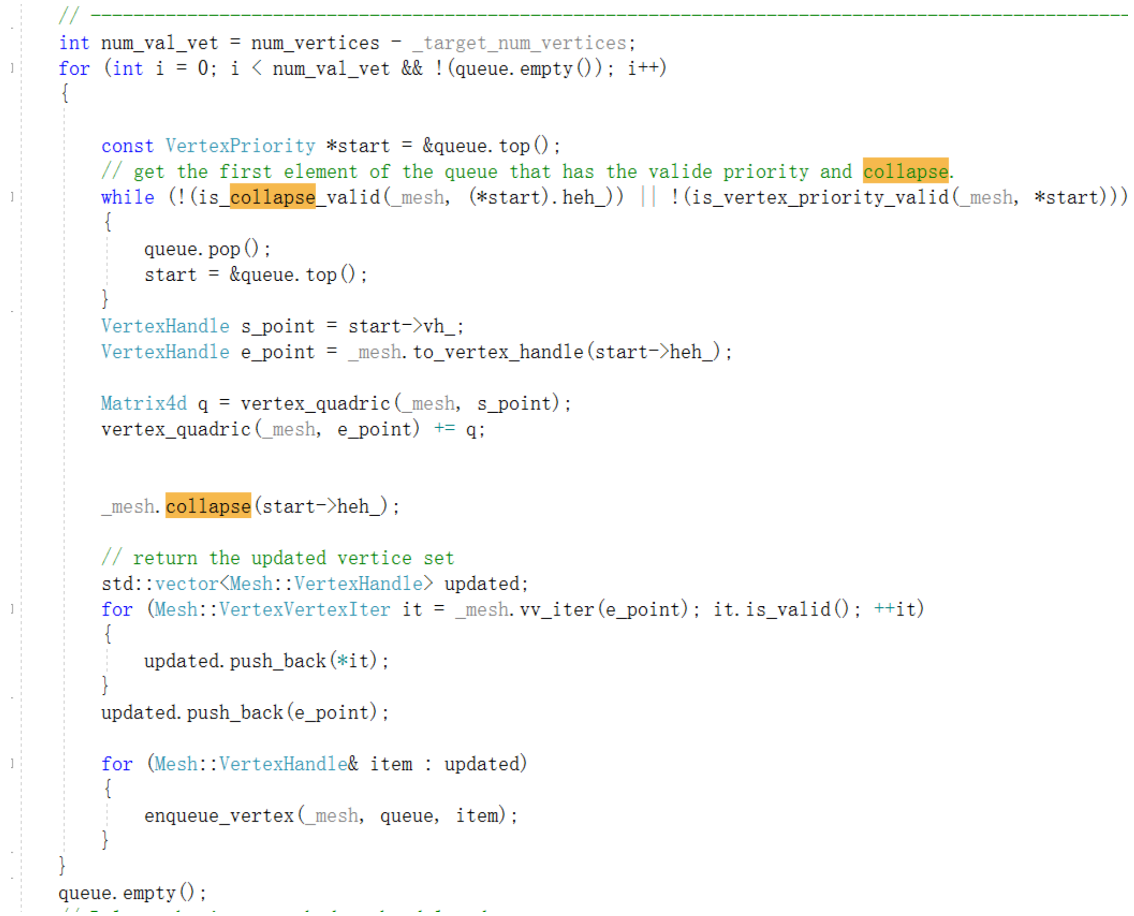
4 Results

5 Results



```
// ---------------------------------------------------------------------------------------------------
int num_val_vet = num_vertices - _target_num_vertices;
for (int i = 0; i < num_val_vet && !(queue.empty()); i++)
{

    const VertexPriority *start = &queue.top();
    // get the first element of the queue that has the valide priority and collapse.
    while (!(is_collapse_valid(_mesh, (*start).heh_)) || !(is_vertex_priority_valid(_mesh, *start)))
    {
        queue.pop();
        start = &queue.top();
    }
    VertexHandle s_point = start->vh_;
    VertexHandle e_point = _mesh.to_vertex_handle(start->heh_);

    Matrix4d q = vertex_quadric(_mesh, s_point);
    vertex_quadric(_mesh, e_point) += q;


    _mesh.collapse(start->heh_);

    // return the updated vertice set
    std::vector<Mesh::VertexHandle> updated;
    for (Mesh::VertexVertexIter it = _mesh.vv_iter(e_point); it.is_valid(); ++it)
    {
        updated.push_back(*it);
    }
    updated.push_back(e_point);

    for (Mesh::VertexHandle& item : updated)
    {
        enqueue_vertex(_mesh, queue, item);
    }
}
queue.empty();
```

Figure 3: Three steps to implement the surface decimation.



```
// ----------------------------------------------------------------------------------------------------
int num_val_vet = num_vertices - _target_num_vertices;
for (int i = 0; i < num_val_vet && !(queue.empty()); i++)
{

    const VertexPriority *start = &queue.top();
    // get the first element of the queue that has the valide priority and collapse.
    while (!(is_collapse_valid(_mesh, (*start).heh_)) || !(is_vertex_priority_valid(_mesh, *start)))
    {
        queue.pop();
        start = &queue.top();
    }
    VertexHandle s_point = start->vh_;
    VertexHandle e_point = _mesh.to_vertex_handle(start->heh_);

    Matrix4d q = vertex_quadric(_mesh, s_point);
    vertex_quadric(_mesh, e_point) += q;


    _mesh.collapse(start->heh_);

    // return the updated vertice set
    std::vector<Mesh::VertexHandle> updated;
    for (Mesh::VertexVertexIter it = _mesh.vv_iter(e_point); it.is_valid(); ++it)
    {
        updated.push_back(*it);
    }
    updated.push_back(e_point);

    for (Mesh::VertexHandle& item : updated)
    {
        enqueue_vertex(_mesh, queue, item);
    }
}
queue.empty();
```

Figure 4: Three steps to implement the surface decimation.

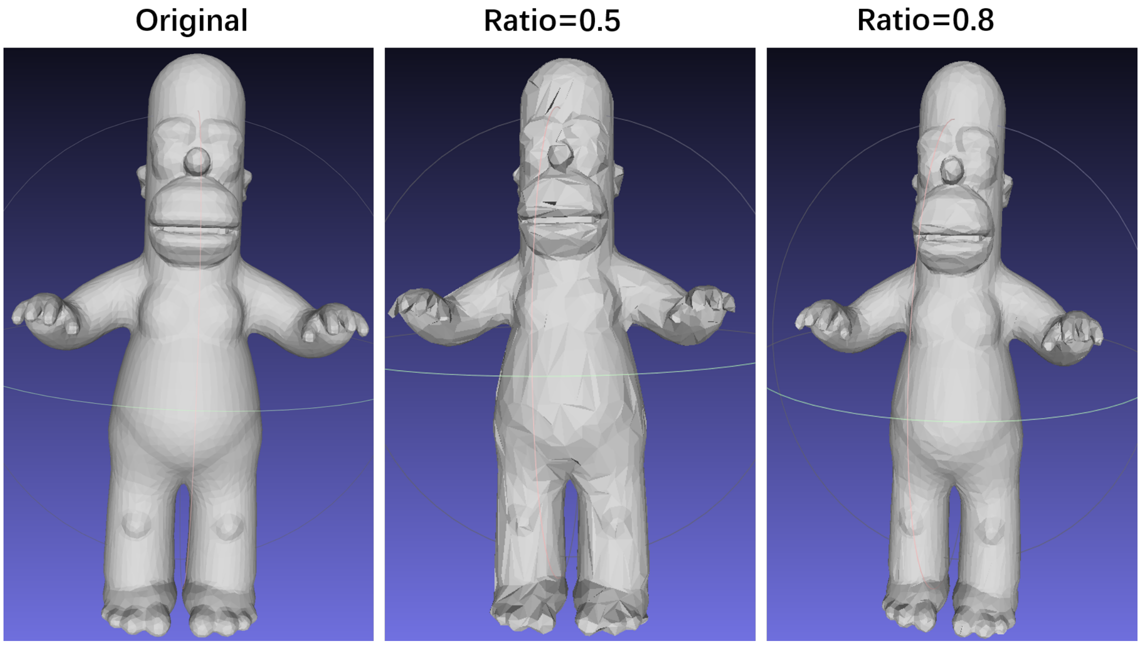


Figure 5: Demonstrations of the homer model (referring the cases of the original, ration=0.5,0.8).

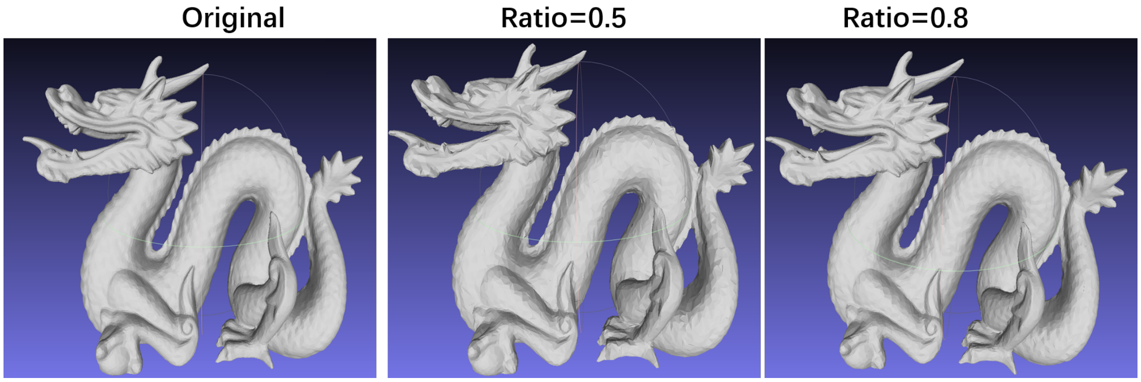


Figure 6: Demonstrations of the dragon model (referring the cases of the original, ration=0.5,0.8