Polyhedral Approximation of a 3-D Mesh

by

Kartik Vermun (11CS30015)

Kumar Saurav (11CS30016)

Department Of Computer Science and Engineering
Indian Institute Of Technology, Kharagpur
March 2016

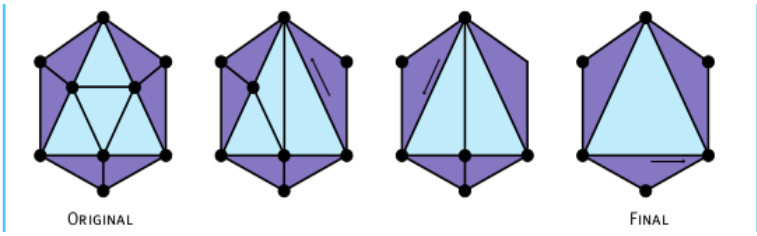

Figure 1: Edge Collapsing

0.1 Algorithm Explanation

The input for the problem was given to be a 3D mesh as a obj file and a distance metric. This distance metric functions as a cutoff to avoid approximating faces whose distance from a specific point is greater than the metric. We have used the distance metric in a slightly stricter sense as expained below.

0.1.1 EDGE COLLAPSING

The heart of our approximating algorithm is edge collapsing. We associate with each edge(u,v) a cost function which indicates the cost of moving u to v, i.e. making edge(u,v) a single point v. The cost function is described below:

$$cost(u,v) = \mid u - v \mid \times max_{f \in Tu}\{\ min_{n \in Tuv}\{\ 1 - f.normal \times n.normal \div 2\}\ \} \quad (1)$$

where Tu is the set of traingles that have vertex u and Tuv is the set of triangles that have vertex u and v.

The cost function helps to approximate small details first. Note also that fewer polygons are needed to represent nearly coplanar surfaces while areas of high curvature need more polygons. Based on these heuristics, we define the cost of collapsing an edge as the length of the edge multiplied by a curvature term. The curvature term for collapsing an edge uv is determined by comparing dot products of face normals in order to find the triangle adjacent to u that faces furthest away from the other triangles that are along uv. Equation 1 shows the edge cost formula in more formal notation.

You can see that this algorithm balances curvature and size when determining which edge to collapse. Note that the cost of collapsing vertex u to v may be different than the cost of collapsing v to u. Furthermore, the formula is effective for collapsing edges along a ridge. Although the ridge may be a sharp angle, it won't matter if it's running orthogonal to the edge. Figure 2 illustrates this concept. Clearly, vertex B, sitting in the middle of a flat region, can be collapsed to A or C. Corner vertex C should be left alone. It would be bad to move vertex

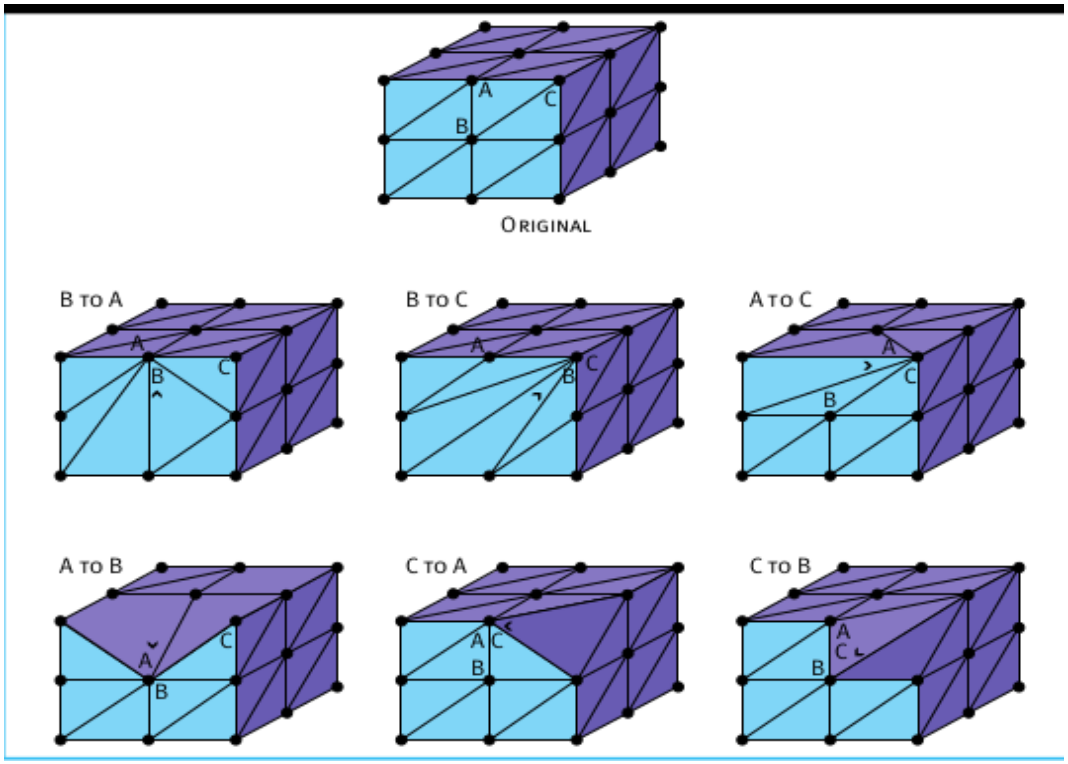

Figure 2: Edge Collapsing

A, sitting along the top ridge, onto interior vertex B. However, A could be moved (along the ridge) onto C without affecting the overall shape of the model.
We use the distance metric d as a cutoff while running the algorithm. We continue the approximation as long as there is an $edge(u, v)$ for which $\mid u - v \mid \leq d$. This is the "stricter restriction" as discussed previously , the reason being if the distance between 2 vertices will be less than d then the distance between the approximated face and the point choosen will also be less than d .

0.1.2 Algorithm

Data: Obj file with distance metric
Result: Approximated object viewed as svg
Initialization-Compute edge cost for each edge;
while *there is an edge whose magnitude is less that* d **do**
    $edge(u, v) = GetMinCostEdge()$;
    $CollapseUtoV()$;
    for *each neighbor* s *of* u **do**
        $UpdateFaces(s)$;
        $UpdateCost(s)$;
    end
end

Algorithm 1: Polyhedral Approximation

0.1.3 Time Complexity

Let the algorithm run for T iterations. The min cost edge can be extracted in $\log N$ time where N is the number of vertices.
The collapsing of vertex from u to v also happens in $\log N$ time.
Each point can have $\mathcal{O}(N)$ neighbors. Updating faces for each neighbor can require $\mathcal{O}(N)$ time. Updating cost for each neighbor can again be a $\mathcal{O}(N)$ time procedure. Hence the time complexity for the algorithm becomes $\mathcal{O}(T * N^2)$

0.1.4 Proof of Correctness

The algorithm iteratively reduces the faces by choosing minimum cost edges , which will cause minimal changes to the occur in the mesh. As each edge collapses the faces adjacent to both vertices of the edge are removed and they get approximated by elongation of the neighboring faces adjacent to the collapsed vertex.

This continues as long there is a valid edge whose cost is less than d.

0.1.5 Test Results

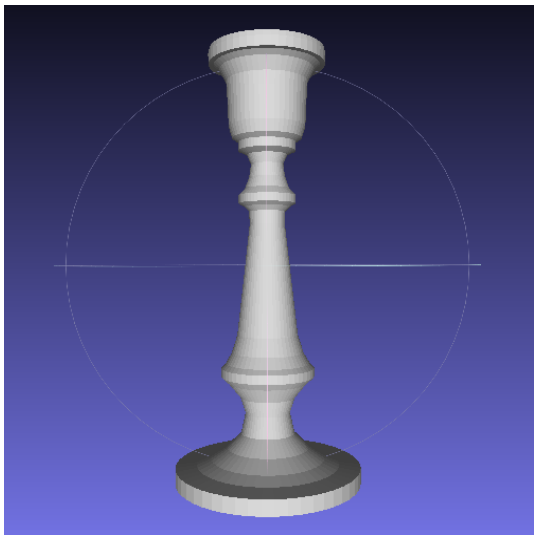

Figure 3: Original Image : 2101 vertices 4150 faces

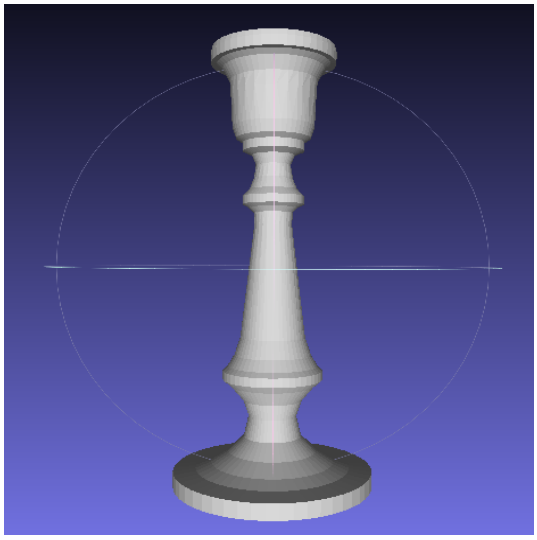

Figure 4: Approximated: 2001 vertices 3994 faces

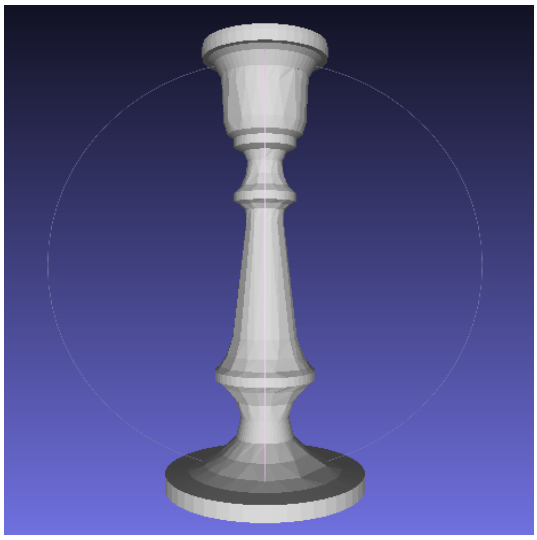

Figure 5: Approximated: 1601 vertices 3194 faces

Figure 6: Approximated: 1001 vertices 2194 faces

Figure 7: Approximated: 601 vertives 1194 faces