

Mesh Simplification Using Quadric Error Metrics

Core Tasks

- Implement the QEM algorithm^[1] and Construct suitable data structures.
- Test the algorithm with simple meshes.

Extra

- Implement the evaluation metric to evaluate the approximations.
- Test the algorithm with complex meshes
- Experiment and build level of details hierarchy for the meshes.
- Try other strategies to compute the merged vertex when the matrix is not invertible.

Implementation Details

1. Programming Language and Libraries:

- Python 3.7
- Numpy, Scipy, GL
- Meshplot

2. Data structures & Class Design:

<i>Class Name</i>	<i>Variables</i>	<i>Functions</i>
Vertex	v: the (x,y,z) values of the vertex	add_face: push a neighbor face into neighborF
	neighborF: the neighbor faces around the vertex	neighbor_vertices: return the 1-ring neighbor vertices
	K: the vertex's K_p matrix	Update: update the K_p matrix and neighbor faces after we merged an edge
Edge	S: the start vertex	Compute_cost: compute the cost of this edge
	T: the terminal vertex	
	Cost: the cost of this edge	
CostHeap	Heap: a heap dictionary which use an edge as key, and the edge's cost as value	Push: push an edge to the heap
		Pop: pop out the edge with minimum cost from the heap
		Update: update the other edges in the heap after we popped out the minimum edge
		Delete: delete an edge from the heap

<i>Class Name</i>	<i>Variables</i>	<i>Functions</i>
QSlim	V_dict : holding all Vertex() of the current mesh.	Simplify : simplify a mesh with a given ratio using QEM algorithm
	E_dict : holding all Edge() of the current mesh.	Compute_loss : compute the loss between the current mesh and the original mesh
	HQ : a CostHeap() object	Plot_simplify : plot the simplified meshes
	Current_Ratio : the ratio between the current #of vertices and the original	Reload : reload the origin mesh from the file

QEM Algorithm Outline

1. Compute the Q_v matrices for all the initial vertices.
2. Select all valid vertex pairs.
3. Compute the optimal \bar{v} for each pair, $\Delta(\bar{v})$ becomes the cost of contracting that pair.
4. Place all the pairs in a heap keyed on cost with the minimum cost pair at the top.
5. Iteratively remove the pair (v_1, v_2) of least cost from the heap, contract this pair, and update the costs of all valid pairs involving the new vertex \bar{v} .

Evaluate the Loss of the Mesh Simplification

In order to evaluate the quality of the algorithm's output numerically without bias, we need a more rigorous error measurement rather than just sum up the edge costs. I randomly sampled two sets of vertices X_i and X_o on the input mesh M_i and the output mesh M_o respectively (we can do this by randomly sample barycentric coordinates on the faces). The paper^[1] introduced a metric which measures the average squared distance between the approximation and the original model. This is very similar to the E_{dist} energy term used by Hoppe et al. ^[2]:

$$E_i = \frac{1}{|X_n| + |X_i|} \left(\sum_{v \in X_n} d^2(v, M_i) + \sum_{v \in X_i} d^2(v, M_n) \right)$$

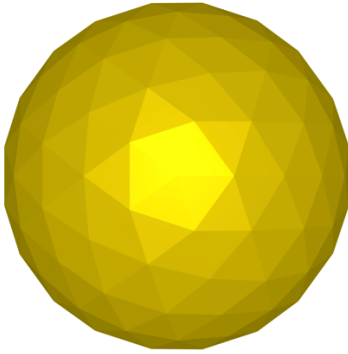

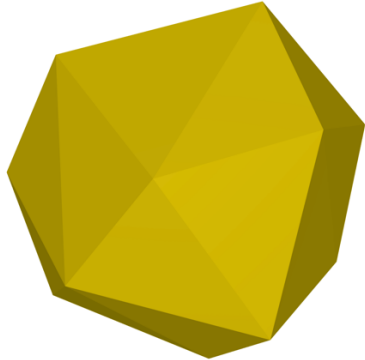
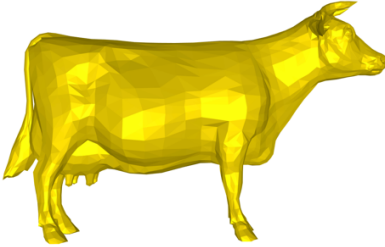
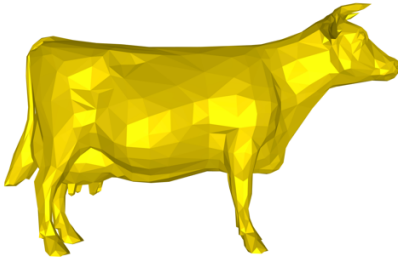
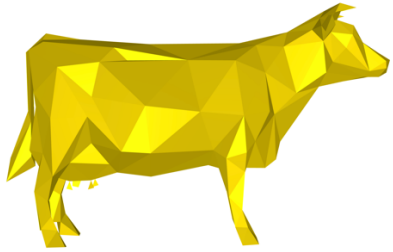
The distance $d(v, M) = \min_{f \in M} \|v - f\|$ is the minimum absolute distance from v to the closet face of M . We can compute this by taking the dot product of the face's plane parameters with v 's homogeneous coordinates. This metric evaluation is used for evaluation purpose only; it's not part of the algorithm (maybe because of the running cost). In my project, I randomly sampled 10000 vertices on each meshes.



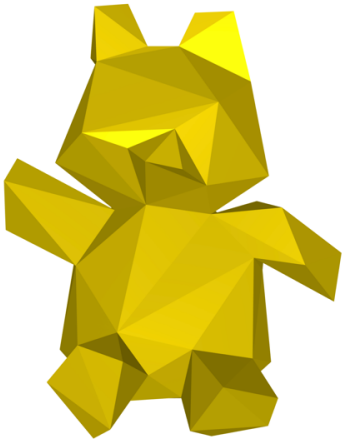



Drawback:

I noticed one drawback of this evaluation method. Because of the randomness of the vertices sampling, the E_i may not increasing as the mesh's resolution decreasing. Take the result of Stanford Dragon as example, when the vertices number decreased from 2500 to 500, the E_i **decreased** from 0.207 to 0.19

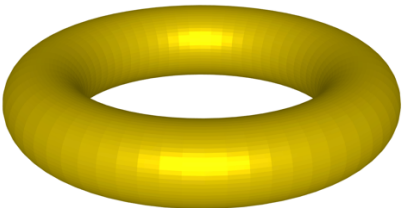
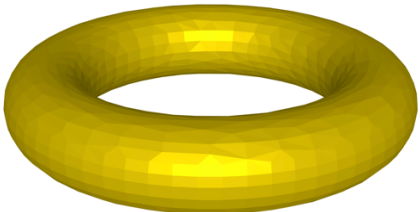

which is not theoretically correct. (**However**, this may be caused by the differences on implementation, the paper didn't mention too much details about the implementation).

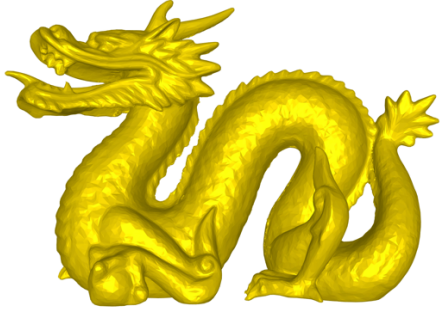

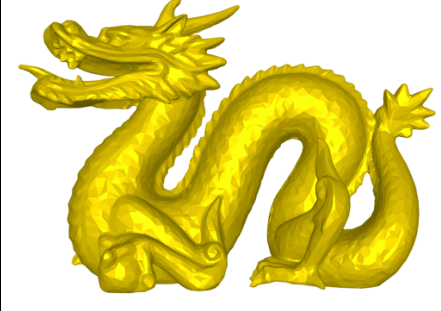



Simple Meshes

<i>Sphere</i>		
		
Origin Mesh V = 162, F = 320	$E_i = 2.61$ V = 81, F = 158	$E_i = 2.9$ V = 16, F = 28
<i>Cow</i>		
		
Origin Mesh V = 4583, F = 5804	$E_i = 0.103$ V = 1145, F = 2288	$E_i = 0.107$ V = 229, F = 448

Teddy		
		
Origin Mesh V = 1598, F = 3192	$E_i = 2.56$ V = 399, F = 794	$E_i = 3.18$ V = 79, F = 154
Bunny (Simplified version from graphics.stanford.edu)		
		
Origin Mesh V = 2503, F = 4968	$E_i = 0.0208$ V = 625, F = 1227	$E_i = 0.0222$ V = 125, F = 238

Complex Mesh

Torus		
		
Origin Mesh V = 4096, F = 8192	$E_i = 0.0126$ V = 1024, F = 2048	$E_i = 0.0166$ V = 204, F = 408

Stanford Dragon		
		
Origin Mesh V = 50004, F = 100000	$E_i = 0.178$ V = 25002, F = 49992	$E_i = 0.180$ V = 12501, F = 24792
		
$E_i = 0.186$ V = 5000, F = 9942	$E_i = 0.207$ V = 2500, F = 7376	$E_i = 0.19$ V = 500, F = 920

Level of Detail Hierarchy

Level of Detail hierarchy is important for the video game rendering process with high-resolution meshes such as *Stanford Dragon*. In order to build a LoD hierarchy, I have to define a variable to control the extent of the simplification. There are three basic attributes of a mesh can be used as a target value of simplification: numbers of vertices, faces and edges. Before I made the choice, I researched the relationship between these three attributes in triangle meshes.

Half Edge^[3]

One important new definition is half edge. Usually in the triangle meshes, one edge only has 2 side faces. So, we define a half edge by an edge and one of its side faces. And obviously one triangle is bounded by 3 different half edges. If the mesh has V vertices E edges and F triangle faces, the total number of half edges is:

$$2E = 3F$$

Euler's Formula^[3]

Another important theorem is Euler's formula for triangle meshes:

$$V - E + F = 2(1 - g)$$

Where $g \in \mathbb{N}$ is the genus of the mesh, genus is usually a very small number which is the number of ring shape handles in the mesh (e.g. Doughnuts shape has 1 genus). So approximately:

$$V - E + F \approx 0$$

Using the equation of half edge, if I substitute E by $\frac{3}{2}F$:

$$F \approx 2V$$

If I substitute F by $\frac{2}{3}E$:

$$E \approx 3V$$

Therefore V, E and F are approximately related. It's also a reason for OBJ files' design since the total amortized storage required per triangle will be reduced if we let the vertices shareable.

Then I can look at some other advantages of these variables. The edge is an unusual data in rendering process and storage, I exclude it first.

Number of Vertices

- Easy to acquire in each iteration of QEM algorithm.
- Related to the storage space of 3D mesh.
- Hard to expect the rendering cost.

Number of Faces

- Related to the run-time of rendering process directly
- But need to re-compute in each iteration of QEM algorithm

In order to reduce the run-time of QEM algorithm during the testing, I used number of vertices as the target value for LoD hierarchy.

Contraction Strategy

In the QEM algorithm, sometimes the Q Matrix may not be invertible then we cannot find a solution for the new vertex \bar{v} . In this situation, use the midpoint of the contracted edge as the replacement is the most convenient way. I also tried another strategy which is uniformly sampled 100 vertices on the contracted edge then choose the sample with the minimum cost.

Mesh	Num of V / F	Midpoint (E_i)	Optimal (E_i)	Error Reduction	Midpoint Runtime (s)	Optimal Runtime(s)	% Runtime Increasing
Cow	1145 / 2288	0.111	0.0995	10.4%	60.7	65.1	7.25%
	572 / 1142	0.116	0.1	13.8%	20.9	21.7	3.83%
	45 / 68	0.176	0.131	25.6%	16.7	17.5	4.80%
Bunny	1251 / 2476	0.0231	0.0219	5.2%	42.2	45.2	7.11%
	625 / 1228	0.0237	0.0214	9.7%	25.3	25.8	1.98%
	237 / 361	0.0242	0.0226	6.6%	12.4	12.9	4.03%

Notice: The later simplified meshes are generated from the previous one; Therefore, the runtime was decreasing as the number of vertices decreasing.

Based on the table above, I noticed that the error reductions of my sampling strategy aren't obviously. Maybe I should increase the number of sampling or try other methods. However, the extra run-time cost also not significant.

In my opinion, the fixed midpoint strategy generated good visual look for the simplified meshes already. Also, this strategy doesn't require too much runtime cost. So, it's a quality-time trade based on the platform and the users' requirements.

Since the new vertex selection only happened when we are pushing the edges to the heap or updating the edges in the heap after popping, one solution can be setting a threshold for the number of the updating edges:

```
T = 10
updateQueue = the edges waiting for updating
if len(updateQueue) > T:
    Strategy = FixedMidpoint ()
else:
    Strategy = Optimal()
for e in updateQueue:
    Heap.update(e, cost = Strategy(e))
```

What I learned:

1. What is mesh simplification, why mesh simplification important for some industry such as video game production (Level of Detail, hardware adaption).
2. How to implement QEM algorithm, how to choose the suitable data structure for geometric algorithm.
3. How to evaluate the mesh approximation (definition of E_i).
4. The design of OBJ file (shareable Vertices array, vertices normal, faces).
5. How to use python to processing and visualizing 3D mesh data (Conda libraries).
6. The Euler's formula and the relation between triangle mesh's # of V,F,E

Reference

- [1] Garland, M., & Heckbert, P. S. (1997, August). Surface simplification using quadric error metrics.
- [2] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In SIGGRAPH '93 Proc., pages 19–26, Aug. 1993.
- [3] http://www.pbr-book.org/3ed-2018/Shapes/Triangle_Meshes.html

Source of mesh models:

- OpenGL <https://github.com/McNopper/OpenGL>
- graphics.stanford.edu
- Clara.io: <https://clara.io/library>