Flat Angle Terminator

Guan Zimu 2021/4/13

1.Motivation

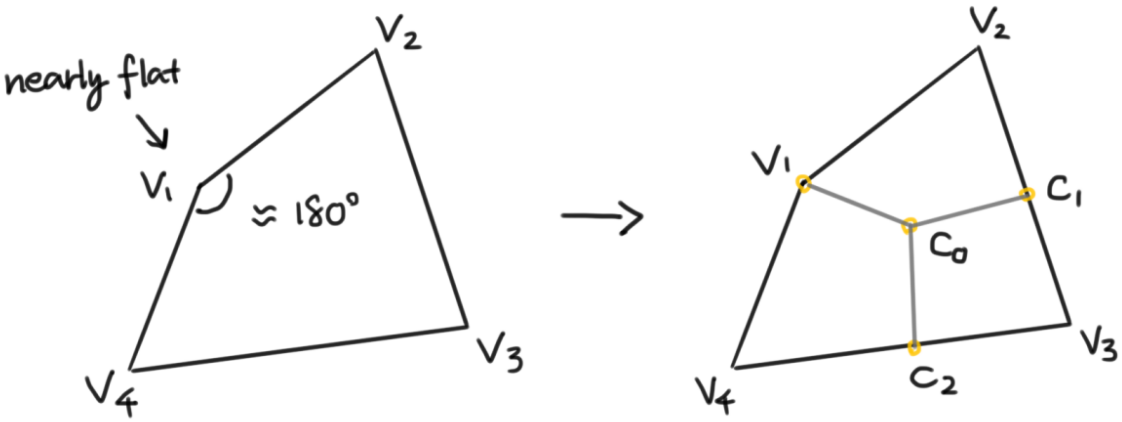
In quadrilateral mesh, quad cells with nearly flat interior angles would generate nearly singular matrix or lead to divergence in the subsequent computation, which are not good for simulation. Therefore, we need some algorithm to remove such low quantity quad units.

2.Algorithm

I basically implement the rough algorithm from the reference document for practicing and getting familiar with geometry processing. This algorithm is incomplete, limitation would be shown later.

2.1 Source Cell Split

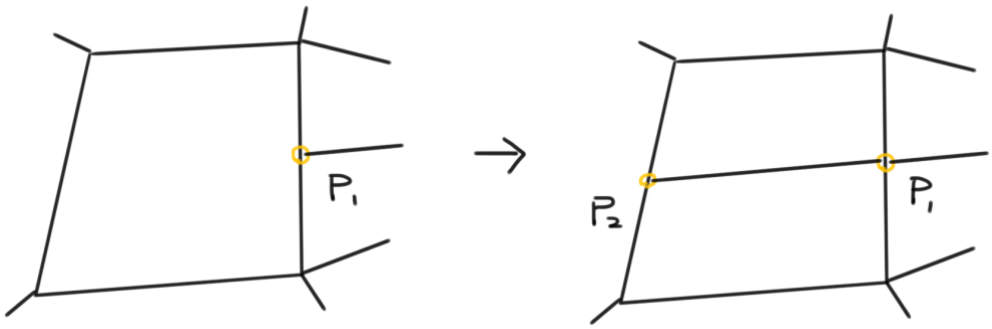
We first find the cells having nearly flat angles by traversing all cells in the mesh. Then we split the cell as follows:



where $\mathbf{c_0} = (\mathbf{v_1} + \mathbf{v_2} + \mathbf{v_3} + \mathbf{v_4})/4$ is the center of the quad cell,
$\mathbf{c_1} = (\mathbf{v_2} + \mathbf{v_3})/2, \mathbf{c_2} = (\mathbf{v_3} + \mathbf{v_4})/2$ are the centers of opposite edges of the flat angle respectively.

2.2 Split Propagation

New vertices are created in the source cell split step. We need to keep split neighbor cell to keep the completeness of the quad mesh. Basically a cell would be split like follows:



where $\mathbf{p_2}$ is the center of the opposite edge of the already split point $\mathbf{p1}$. Thus the split procedure would propagate until it meets a already split edge or boundary edge of the mesh.

Considering one edge can belongs to more than 2 cells, meanwhile, the split procedure would propagate quite deep, we use a BFS to implement split propagation. Basically we use cells as basic nodes in the queue. Besides, two hash maps of edge-information & cell-information pairs are added to reduce the seeking time.

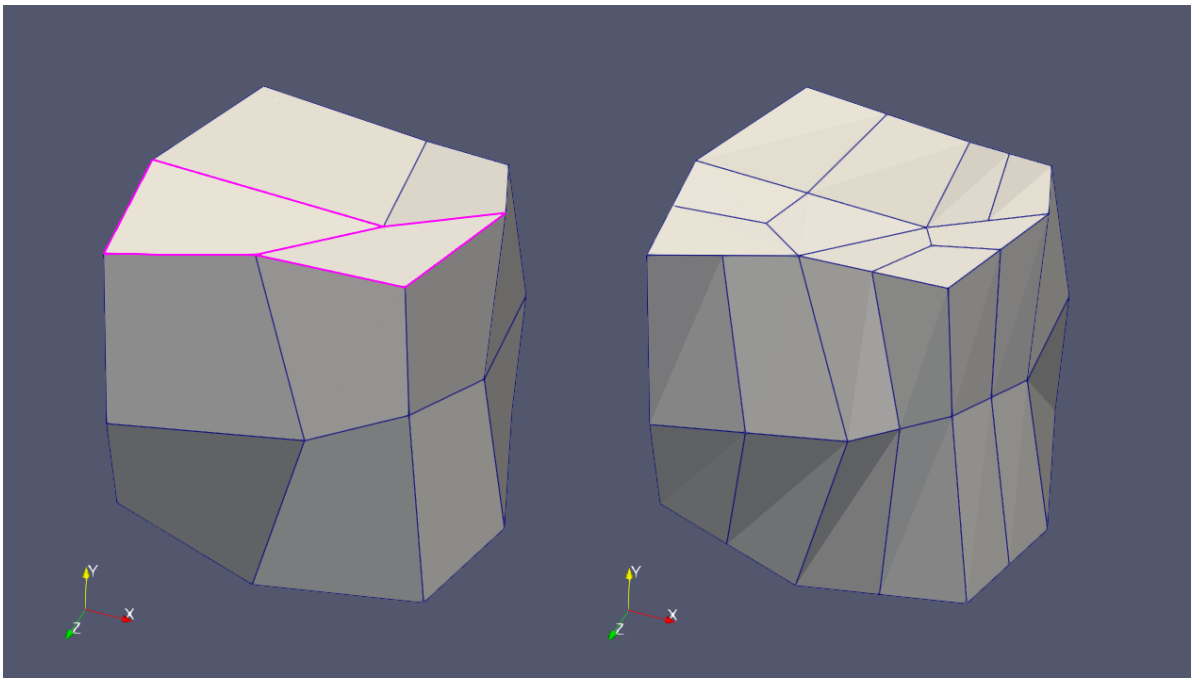
In practice, split is not as simple as the image above. There are many special cases, for example, a cell can be split from two directions; a split procedure may meet with source cell, or boundary edge, and so on. Most of special cases are covered in the implement code, and there are detailed comments in the code classifying different cases. I would not go into details in this report.

Because we use two hash maps to store edge-cell relationship and other information, seek time is just $O(1)$. The branch of searching tree would not be too much, in most cases it is just 2. As a consequence, split propagation procedure would not exceed $O(N)$, where N is the cell number of the mesh. Consider the traverse step, the total time complexity is $O(N)$.

3. Results

cube

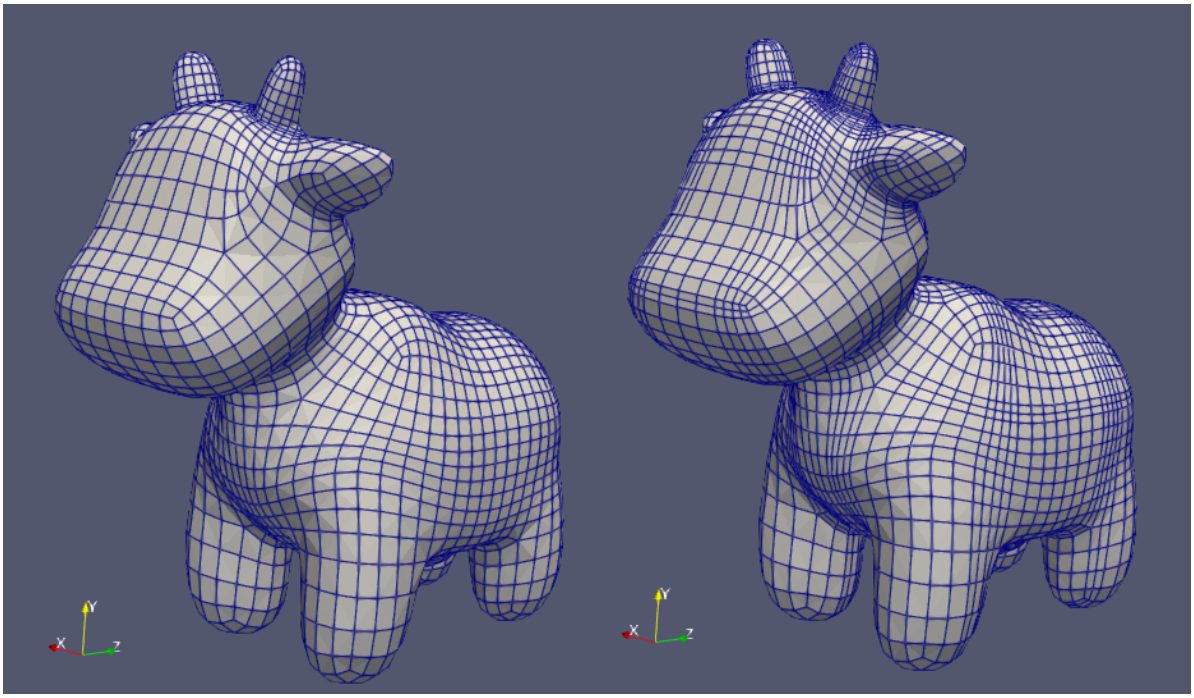
The quad cells highlighted by pink lines are split into 3 pieces respectively.



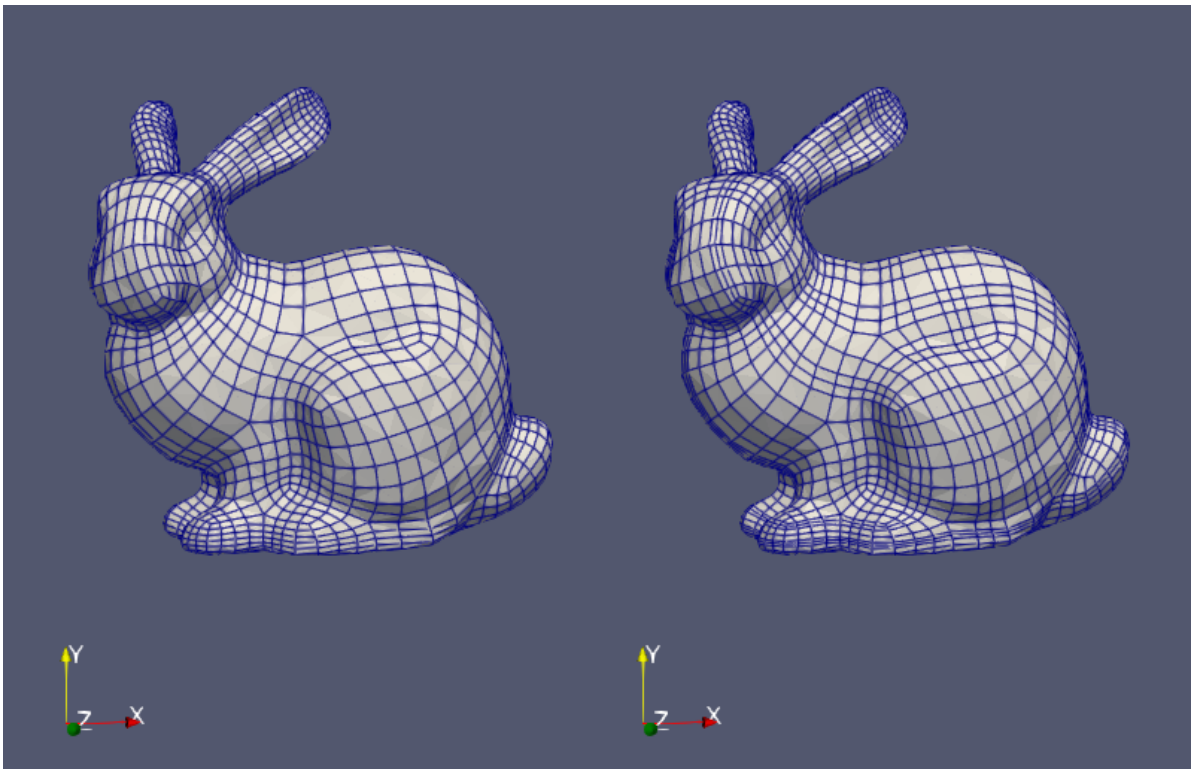
armadillo

The quad cell highlighted by pink lines is split into 3 pieces.

spot

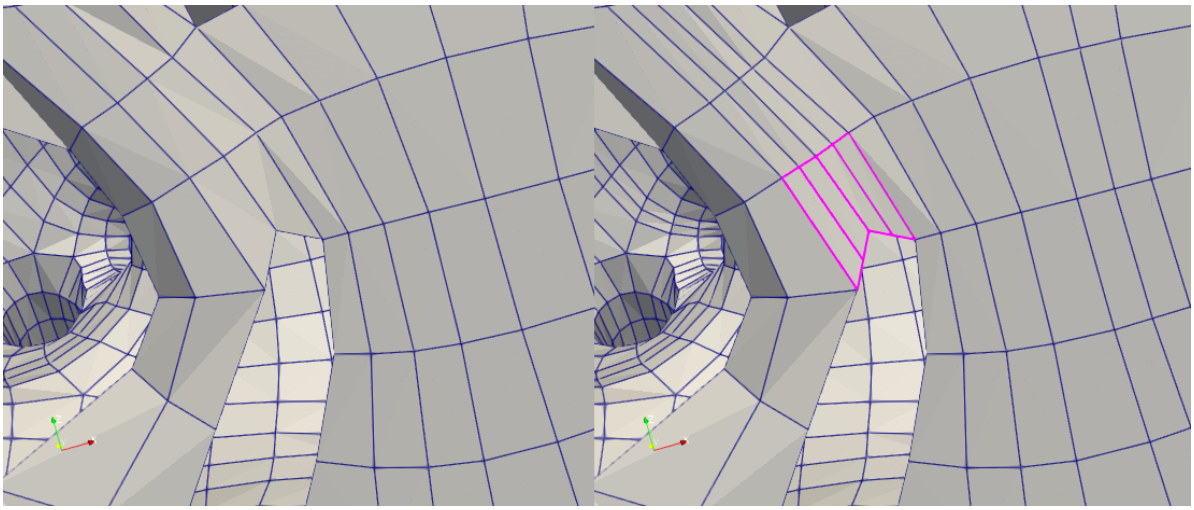


bunny



edge

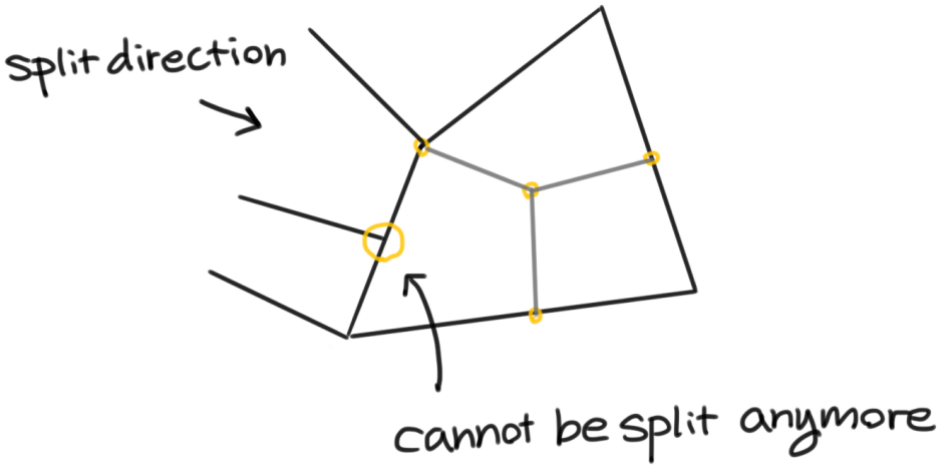
The algorithm also works well on boundary edge.



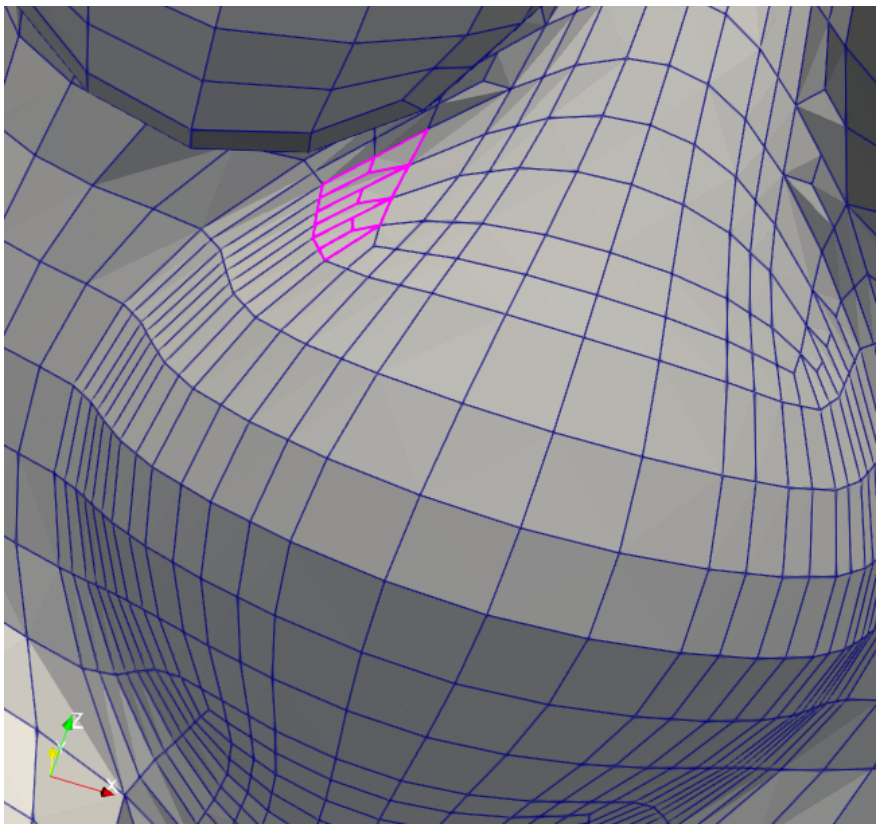
4. Limitations

The algorithm has a big limitation: it cannot guarantee whether there would not still be nearly flat angles in the split cells. Sometimes the number of flat angles could be even bigger than it of the original mesh if the model is complex.

Another limitation is that when the split propagation meets the un-split edge of the source cell, it cannot split anymore, which would leave a unconnected vertex on the edge.



Below is an example of the limitation. We can both see two limitations on the image. The split cells highlighted by pink line still have flat interior angles; besides, because they are adjacent, some split procedure cannot propagate which leave unconnected vertexes on the adjacent edges.



5. Code

The code is well commented and has basic IO interface.

https://github.com/TaKeTube/Geometry/tree/main/FaltAngleTerminator

It can read .vtk and .obj quad mesh and output a .vtk processed quad mesh.

A standard command is like this:

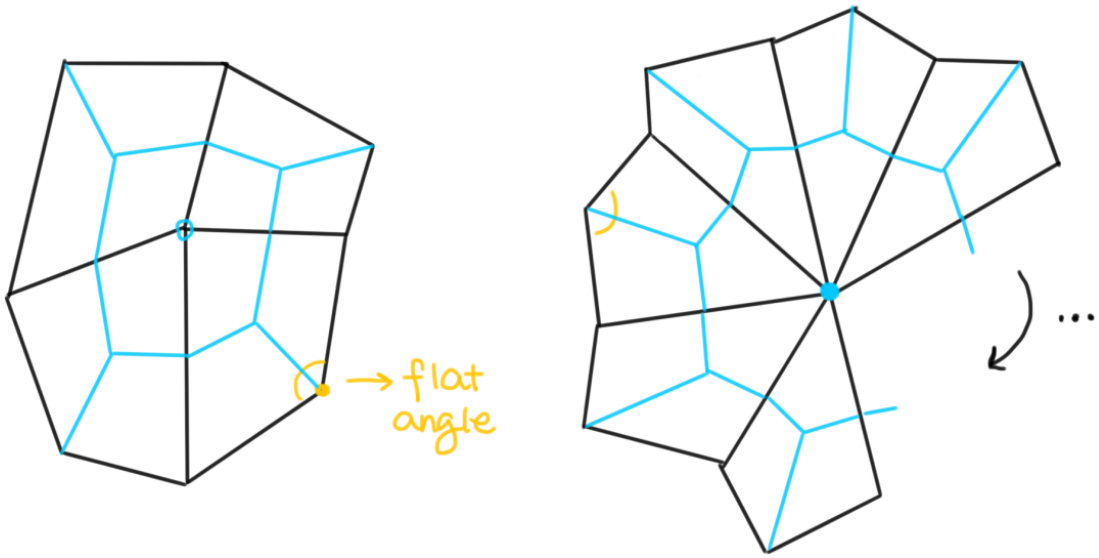
```
./FlatAngleTerminator.exe -input "../data/cube.obj" -output "processed_cube.vtk"
```

6. Other Methods (not implement yet)

Here are some other thoughts I have to remove the flat angle.

6.1 Another split procedure

The propagation method would hugely change the topology structure of the mesh, it is not locally at all. Therefore, another split procedure can be applied to keep the change local.



as shown in the image, we choose the opposite vertex of the flat angle vertex then apply the split to the cells around this point. This method could keep the change local. However, as we can see in the right image, this method could not guarantee there would not still be nearly flat angles in the split cells as well.

6.2 Optimize a metric locally

I'm thinking about a method that basically adjust the position of the vertexes, which avoid the split.

- First, we find the nearby area of the bad cell.
- Then we interpolate a local smooth surface M. (or not smooth, generally)
- We now find a **metric** $d(v_1, \cdots, v_n)$ or generally a scalar score function

$$d : \mathbb{R}^{n\times 3} \to \mathbb{R}$$

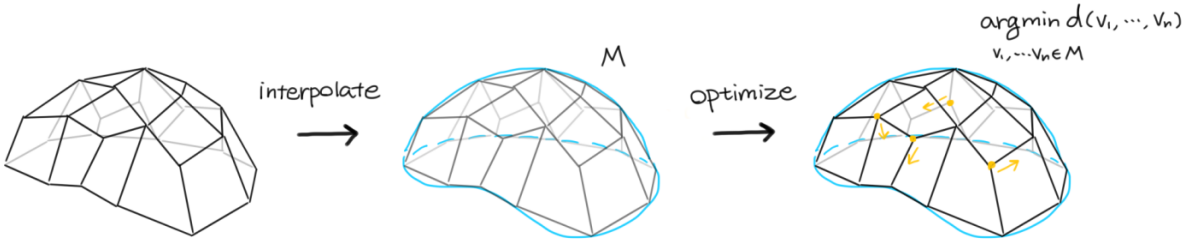
where n is the number of vertexes in this area, to measure the angle differences of interior angles of all quad cells in the area with respect to right angles. This function should be sensitive to each individual cell's angle. For example, even though the total difference is small, a huge difference on one cell and very small differences on other cells may not get a good score.

- we restrict all points on the interpolate surface. Therefore, the problem becomes **optimizing a metric** $d(v_1, \cdots, v_n)$ **on a bounded 2-dimensional manifold** M **(or surface, generally)**

$$\underset{v_1,\cdots,v_2\in M}{\arg\min}\ d(v_1, \cdots, v_2) \equiv \underset{\mathbf{v}\in M^n}{\arg\min}\, d(\mathbf{v})$$

which can be solved by Lagrange Multiplier Method using gradient descend or other effective methods:

$$\mathcal{L}(\mathbf{v}, \lambda) = d(\mathbf{v}) - \lambda M(\mathbf{v})$$



The limitation of this method is that we may get a local minimal, depending on how we choose the score function.

Another limitation is the curse of dimensionality. For n points, the degree of freedom is $2n$ (because they are on 2-d manifold), which is a challenge for computation.

Besides, this method may only works on manifold mesh depending on how we interpolate the surface.