CS554 Geometric Modeling

Project 3 Report

Tianle Yuan (933946576)

February 22, 2021

0. (Term project idea) Write one to two pages of the work you plan to implement as the term project idea. Include which paper(s) you plan to implement if you are implementing existing research. Be clear to include the problem statement and possible solution you are implementing.

The paper that I am going to implement named *Linear Rotation-invariant Coordinates for Meshes*.

The basic inspiration that I want to use this method is that I am right now doing a research about art stylization for Dunhuang wall painting. Right now, I am focusing on modeling a scarf, which frequently shows in the ancient mural art, and trying to make the scarf freely flying in the sky with a status of statically "dynamic". The art picture about the scarf details can seen in **Figure a**.

Figure a. **Scarf element in Dunhuang wall painting**

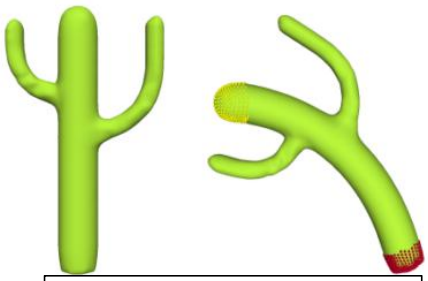


Figure b. Shape deformation by using mouse after set static point and drag point

By refer the solution of Dynamic Fluid Equations, the system 3D space right now is a velocity space, which means any triangle meshes in the space can assign a velocity to its vertices according to the unique space coordinates. Thus right now, the original model of my scarf can get moving by being affected by the "hand of velocity". But how to properly deform the flying scarf, I think I might can find some answers from this paper.

The reason that I chose this paper is that, when I want to deform a mesh in a 3D space, the method from the paper can provide a rigid motion for each part of the model meshes. As the **Figure b** shows, the algorithm from the paper can realize that when a certain part of the movement happens, the whole model can show the global dynamic.

In the Final project, I want to separate the implementation into 3 parts according to the guidance of the paper.

(1) Reconstruct the Discrete Frames (the matrix [$\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{N}$]) at each vertex by solving the

discrete surface equation shown as below, in which $\mathbf{b_1}$ equals to $\mathbf{T_1}$, $\mathbf{b_2}$ equals to $\mathbf{T_2}$. Here because we want all the meshes change together, those the equation system will be over-constrained, which means any change of any vertex on the edge will lead to a different results. I will solve the equation system by construct adjacent matrix for whole the vertices and using Eigen library for equation system solving.

$$\begin{aligned}
\mathbf{b}_1^j &= \left(\Gamma_{j,1}^{i,1}+1\right)\mathbf{b}_1^i+\Gamma_{j,1}^{i,2}\mathbf{b}_2^i+A_{j,1}^i\mathbf{N}^i \\
\mathbf{b}_2^j &= \Gamma_{j,2}^{i,1}\mathbf{b}_1^i+\left(\Gamma_{j,2}^{i,2}+1\right)\mathbf{b}_2^i+A_{j,2}^i\mathbf{N}^i \\
\mathbf{N}^j &= \Gamma_{j,3}^{i,1}\mathbf{b}_1^i+\Gamma_{j,3}^{i,2}\mathbf{b}_2^i+\left(A_{j,3}^i+1\right)\mathbf{N}^i.
\end{aligned}$$

(2) Reconstruct the geometry at each vertex from the reconstructed Discrete Frame and the discrete forms of the edge vectors ($\overrightarrow{eij} = c1\cdot T_1 + c2\cdot T_2 + c3\cdot N$). Then keep resolve the over-constrained equation system to get the new positions of the vertices of the whole mesh. Here I think I will use the least-squares method for the over-constrained system solving.

(3) Implement the effect of scaling and rotating. If it is possible, try to implement some editing effects like uniform scale and shear by resolve the extra equation system looks like below:

$$\begin{aligned}
\mathbf{b}_l^i &= A(\check{\mathbf{b}}_l^i), \qquad l=1,2 \\
\mathbf{N}^i &= A(\check{\mathbf{N}}^i),
\end{aligned}$$

The matrix A is the transformation constraints. In order to keep the original shape details of the model, I will need to go to check the compatibility of both the transformation constrains and positional constrains in the step (2) and (3), or the shape will not keep the rigid.

Solving the object deformation in a force field is hard. However, I think I can start from the simple step. If my system can control one handles of the shape, it will not be that hard to have more control points in the future.

1. (Silhouette drawing) Incorporate silhouette drawing into learnply.

For drawing the silhouette, what I did is try to traverse all the triangles (or vertices), then calculate the dot product for the judgement condition of the silhouette drawing. The algorithm can be easily shown in four steps:

a. Set the back ground and the shape surface in white color. Turn off all the lighting effect by disable GL_LIGHTING.

b. Traverse all the triangles or vertices, calculate the dot product value between the eye ray to the triangles or vertices and normal of triangles or vertices. Then save the dot value as a member of triangles or vertices class.

c. Traverse each edge of the shape mesh. Choose it’s neighbor triangles (2 triangles) or

neighbor vertices (2 vertices) for dot product judgement. If the two neighbor's dot result have different symbol, then we give the edge a flag of silhouette.

d. Traverse each edge and render all the edge whose silhouette flag is working.

a. Classify each face as being either forward or backward using the ray-normal dot product test. Then find all the edges that are part of silhouette. Draw the silhouette edges each time the viewpoint has changed. What problem do you encounter when you switch viewpoints?

For this one we use the triangle's normal as the dot product calculation's element. In order to show the changes of the silhouette edges each time when viewpoint has changed, I listed the render result from the side and front. The result of the Dragon model can be seen as below:

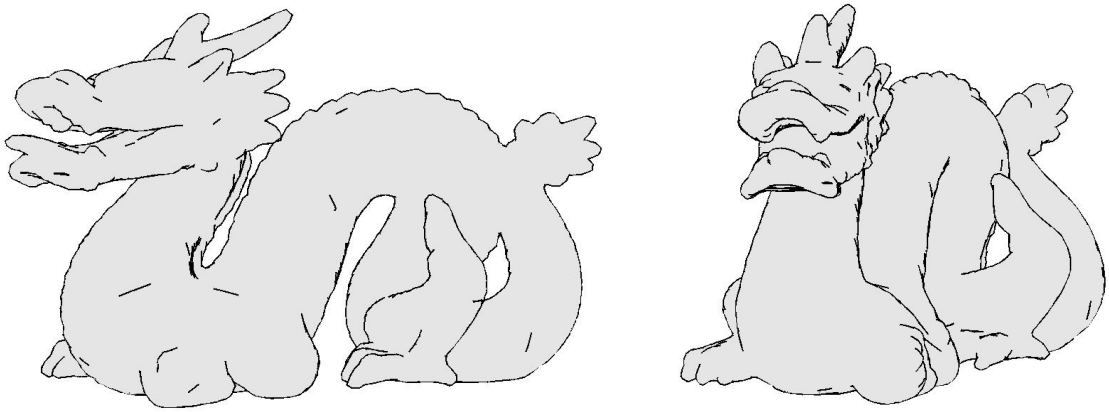
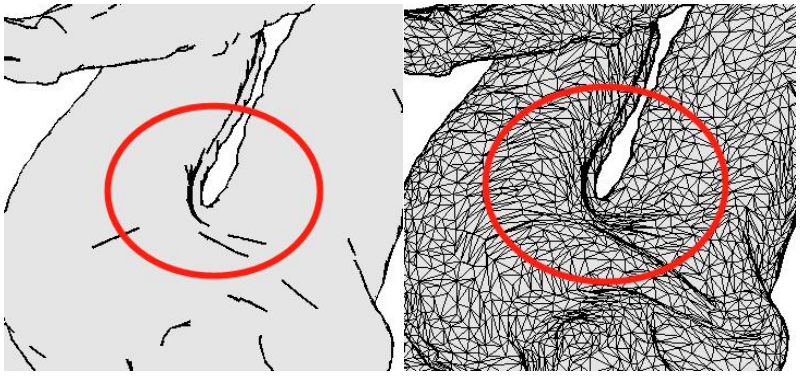


Figure 1. The face-based silhouette depiction
(Side vision left and Front vision right, line width = 1.4)

There are two problems were showing when the viewpoint has changed:

(1) In the line of sight, where the triangle mesh is dense, the probability of showing some overlapping lines would increase, which will lead to a dirty silhouette shape as the **Figure 2** shows:



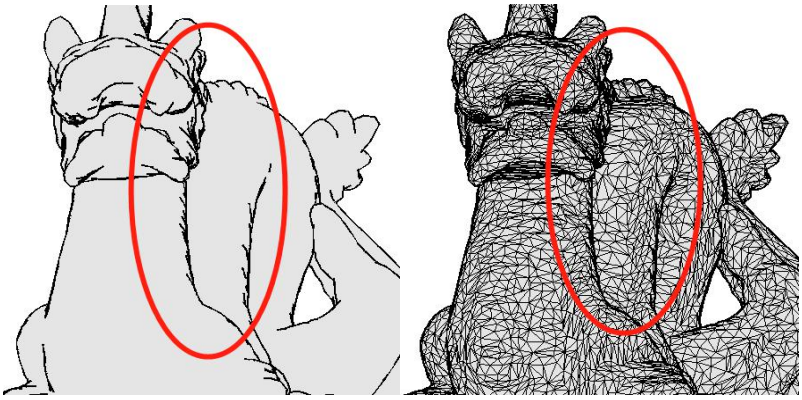


Figure 2. Dirty silhouette shapes

The possible reason shows this problem is that when the triangle meshes locating within sharp visual angles get dense, all of them will want to show the boundary of the positive and negative dot product. Unlike the continuous surface, the discontinuity of meshes will increase the chance of clear boundary overlapping. (Solve direction: Smoothing, see part c)

(2) The another problem is the uneven silhouette line width. The same points who are parallel to the line of sight, because the number of triangular mesh in the neighborhood is different, result in different overall silhouette width as the **Figure 3** shows.

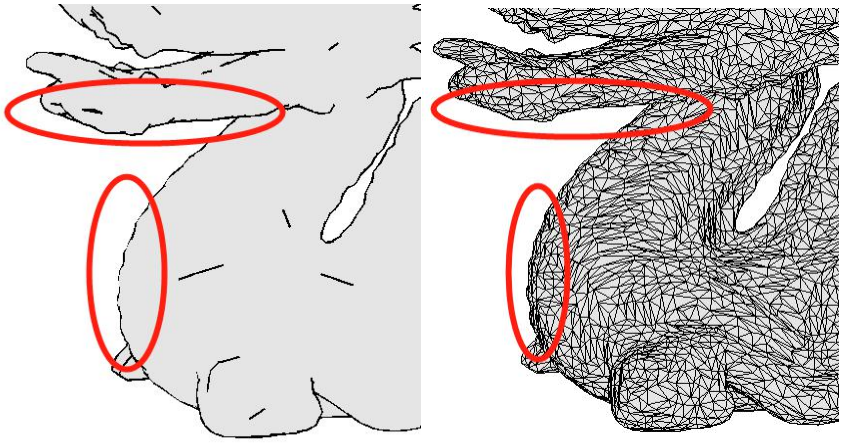


Figure 3. Different silhouette width

The possible reason shows this problem is that the density of each triangle's neighborhood is different. Thus, a possible way to solve this problem is try to increase the neighbor's number of triangles who have fewer neighborhood triangles. (Solve direction: Irregular Subdivision, see part c)

b. Implement a face-based silhouette depiction method. This time, perform the ray-normal dot product test at each vertex and extract silhouette inside each face that correspond to the zero levelset of the ray-normal dot product function. Compare this technique to the one in 3(a). Which one is more preferred, and why? Justify your answers both in words and

figures. Be comprehensive and thorough in your reasoning.

For this method, we firstly need to traverse all the vertices's normal and calculate the dot product with the eye ray. Then for each edge, we pick the critical point (equals to 0) of the positive and negative dot product result on the edge. Finally if we connect those points on the zero levelset, we can finally get the corresponding silhouette of the shape, whose result of the Dragon model can be seen as below:

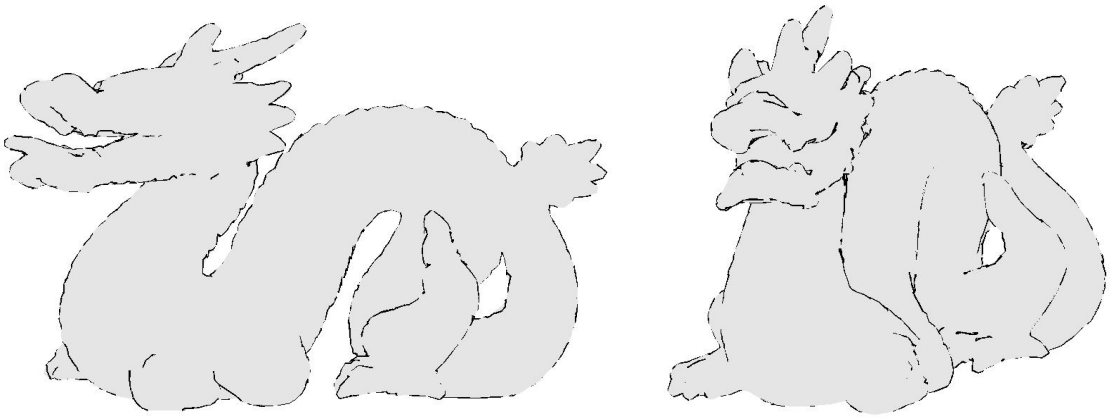


Figure 4. The vertex-based silhouette depiction
(Side vision left and Front vision right, line width = 1.4)

Before the comparison between this method with the face-based method, I want to point out that the two problems were showing when the viewpoint has changed:

(1) The first problem of the method is that, the lines are not continuous. The problem can be seen in the **Figure 5**.

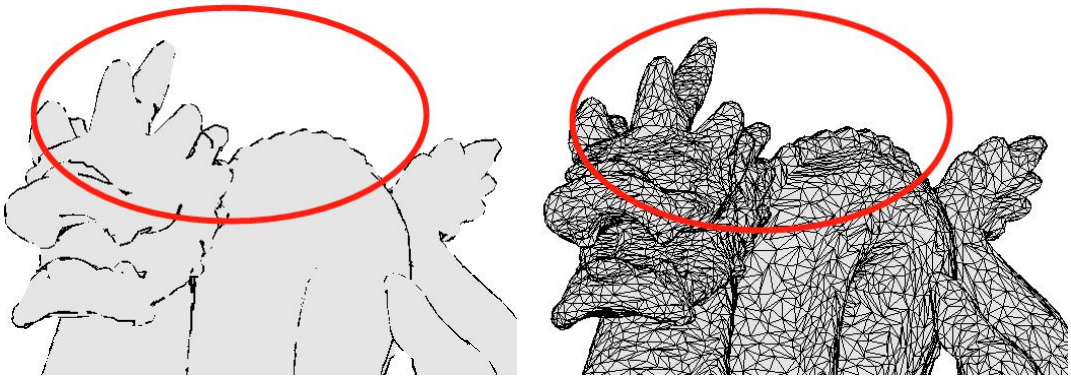


Figure 5. Discontinuous silhouettes

The possible reason for this problem is that, because this kind of silhouette is rendered on the zero level-set on the edge. However, different triangle's edges's level-set line are discontinuous, it will obviously shows the discontinuity of the silhouettes of the contour. (Solve direction: Irregular Subdivision)

(2) The second problem of the method is that, it do not show enough silhouettes of the model. As we can see in **Figure 6**, obviously, the inside part of the dragon do not have some details it should have. This problem might because there are only one zero level-set point on one edge of a triangle mesh, so it cannot be shown. (Solve direction: Irregular Subdivision)

Figure 6. Comparison between the vertex-based and face-based methods

Right now, I want to compare this vertex-based technique to the one in 3(a) (the face-based methods). The face based methods would be the one that I want to choose. There are four main reasons listed below:

(1) Considering the definition of the silhouette, it is not hard to see that the vertex-based method will lose some silhouette that it should have. Even doing the irregular subdivision can improve the final result as the blue circle of **Figure 7** showing. However, it cost too much for the further calculation for the tensor smoothing.

(2) Even if we can use irregular subdivision for solving some of the discontinuity of the vertex-based silhouette result, the optimize speed is too slow. Even there are some improvement of the contour (red circle in **Figure 7**), it still not efficient.

(3) The Vertex-based method itself has some instability. From the optimized result shown in **Figure 7** of the Vertex-based method, we can find, even the irregular subdivision method can fix some missing details, but the compensation is unstable. The fixed detail is a tiny circle (right side in the blue circle). From **Figure 8**, we can think two simple case that will lead to a circle silhouette. In the figure, blue lines is hour silhouette and the symbol "+" and "-" are the directions of vertices or faces' normal. The Vertex-based method can easily implement the case of circles without any restrictions. However, for the Face-based method, the right side case of the triangles is impossible. Although the surface derivation of the model is discontinuous, the surface it self is continuous, which means, no matter what case, a surface with it neighbour must have the same normal pointing tendency, or they will be not connected with each other. That is to say, the Face-based method it self is stable for mesh triangle changes when the topology property of the model doesn't change.

(4) Most of the problem listed in (a) part of the question 1 can almost be solved by applying the answer of the coming (c) part. The high optimization efficiency, is one of the most reason why I prefer the Face-based silhouette depiction method.

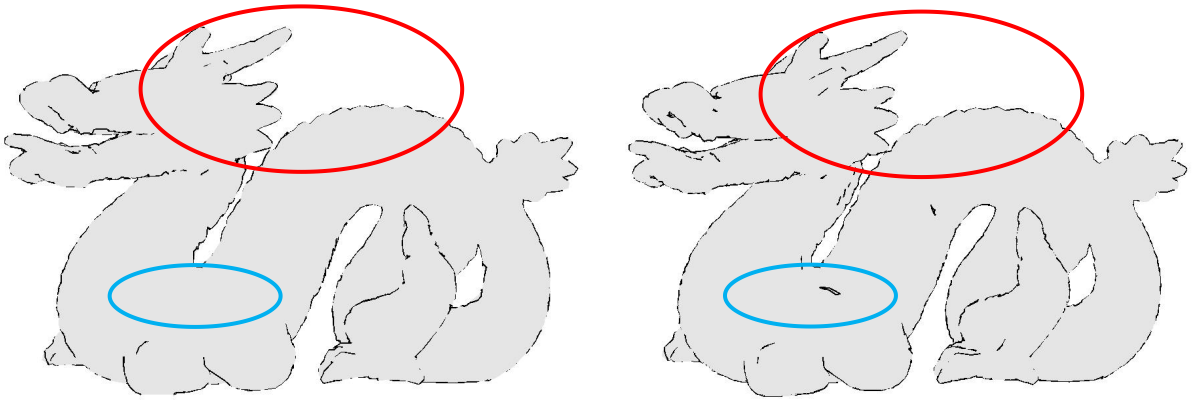


Figure 7. Vertex-based method's result before and after irregular subdivision (left Before, right After)

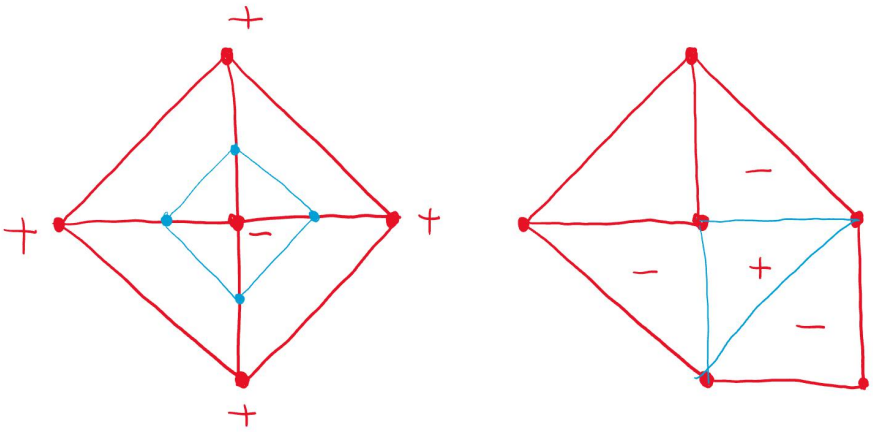


Figure 8. The circle situation of the silhouette (left Vertex-based method, right Face-based method)

c. Do the methods in (a) and (b) capture all the important features in the model? In what way can these methods be improved? Provide at least three aspects to this.

There are four plan can be used for the silhouette result optimization. Based on the analysis of the optimization for the two methods, here we would focus more on the ways that can improve the Face-based silhouette depiction method.

(1) *Change the width of the line*: The method can magnify some part of the silhouette who are too thin and also fix some discontinuity of the silhouette. I implemented this one by calling "`glEnable(GL_LINE_SMOOTH)`" and then set the line width by calling "`glLineWidth(width);`" The effect can be seen in **Figure 9**.

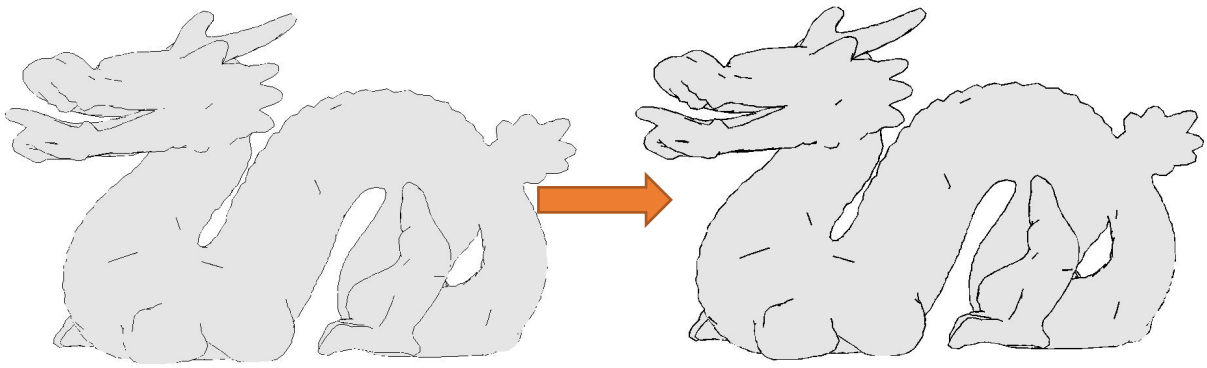


Figure 9. Line width changing
(left before width changing, right set width == 1.4)

By doing this method, the final result is much better. However, this method only focus on the global width of the silhouette, which means, the thin line will still be thin compared with the thick line. (As we have talked before, see the issue **(2)** of *Face-based method* in part **a** in **Figure 3**)

(2) *Smooth*: This method is trying to solve the issue **(1)** of *Face-based method* in part **a** that we have shown before (can see **Figure 2**). Basically, it is try to solve the overlapping lines caused by the discontinuous meshes surface. Thus, to solve this problem, we can do smoothing of the mesh surface, which can be seen in **Figure 10**.

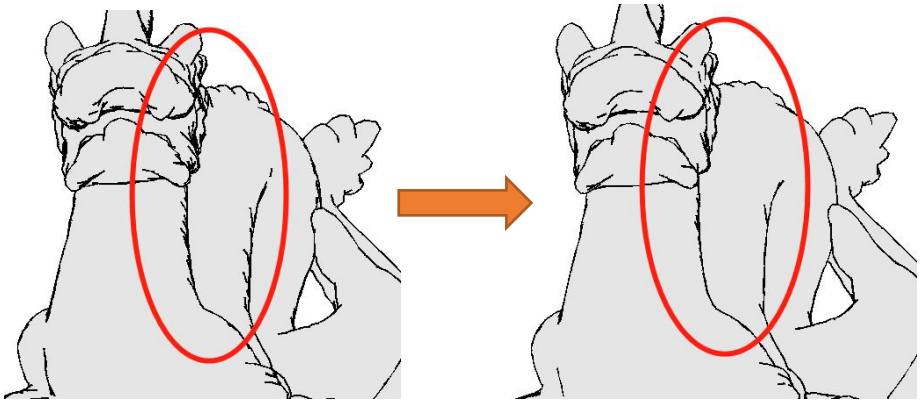


Figure 10. Smooth the triangle meshes
(Mean curvature flow scheme, iterating 5 times, dt =0.1)
(left Before, right After)

The result of this processing is really good and the cost is less. With small time step and small number of iterations can realize efficient Anti-noise. The overlapping silhouettes were quickly eliminated at very little cost to the loss of the model shape.

(3) *Irregular subdivision*: This method is trying to solve the problem of the uneven silhouette widths, which is the issue **(2)** of *Face-based method* in part **a** in **Figure 3**. Irregular subdivision increase the density of each triangle's neighborhood, decrease the parent triangle's

bad property and average the total shape's silhouette width. Compared with the effect that Irregular subdivision implement on *Vertex-based method* in **Figure 7**, *Face-based method* can realize width averaging faster, which can be seen in **Figure 11**.

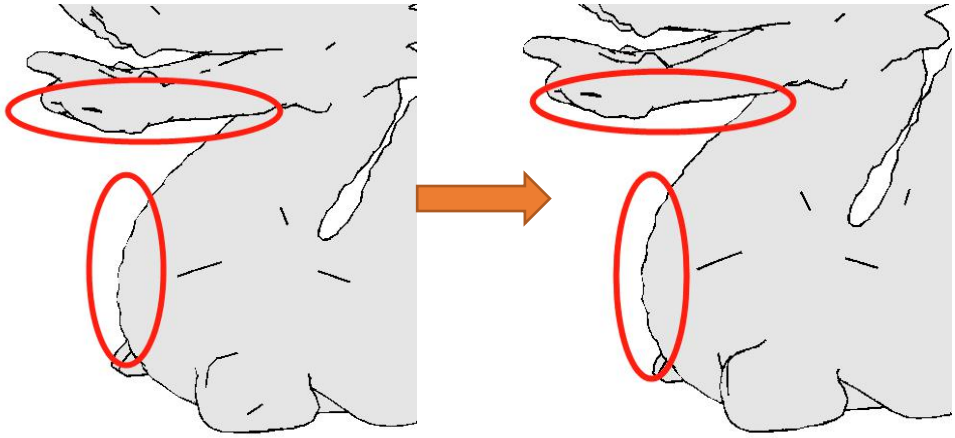


Figure 11. Irregular subdivision on the triangle meshes (left Before, right After)

The method actually is not that efficient. Even though the irregular subdivision speed is fast, the increased triangle mesh's number will increase the calculating speed of the later steps like streamline drawing and the tensor smoothing.

The reason why we don't use regular subdivision but irregular one is that regular subdivision will not divide the triangle with bad property into small triangles with good property. For example, there is a obtuse-angled triangle. If we do regular subdivision, the big obtuse triangle will be separated into several small obtuse triangles, which would keep the contour region with no silhouette still keep no silhouette.

(4) *Threshold*: This method is an additional plan for increase the width of the silhouette. The threshold is trying to loose the range of the dot product result. However, it is also the global method that edit the whole silhouette of the shape. Thus if a silhouette in a region has enough width, it will increase more noise which are not the needed as the **Figure 12** shows.

Figure 12. The problem of increasing the threshold of dot calculation (left Before, right After)

(5) *Some problem that the gl2.0 cannot solve*: There is a problem of the OpenGL displaying named Z-fighting. Right now because we draw all the mesh triangles by setting them as white color, if the edge, which is a part of the silhouette, locates behind the triangle it belongs to, the triangle will probably overlap the edge that should be rendering.

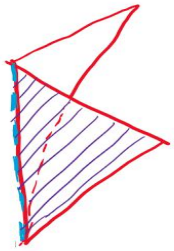


Figure 13. The problem of Z-fighting between silhouette edge and face

2. (Curvature estimation) Implement the discrete curvature estimation algorithm by Meyer et al. http://www.multires.caltech.edu/pubs/diffGeoOps.pdf. You will need to estimate and visualize the mean curvature, Gaussian curvature, and the curvature tensor. For the mean and Gaussian curvatures, design appropriate color coding schemes and explain them in the report. <u>(The remaining part of the question has been listed later)</u>

For the estimation of Mean and Gaussian curvatures, I used the calculation step as shown below:

(a) Traverse all the triangles surround a vertex, calculate the Voronoi region's area for each triangle. Then add all of those areas together and get the total Mixed area:

$\mathcal{A}_{\text{Mixed}} = 0$
For each triangle T from the 1-ring neighborhood of $\mathbf{x}$
  If T is non-obtuse, `// Voronoi safe`
    `// Add Voronoi formula (see Section 3.3)`
    $\mathcal{A}_{\text{Mixed}} +=$ Voronoi region of $\mathbf{x}$ in T
  Else `// Voronoi inappropriate`
    `// Add either area(T)/4 or area(T)/2`
    If the angle of T at $\mathbf{x}$ is obtuse
      $\mathcal{A}_{\text{Mixed}} +=$ area$(T)/2$
    Else
      $\mathcal{A}_{\text{Mixed}} +=$ area$(T)/4$

The calculation of the Voronoi region's area for a triangle can be expressed as below:

$$\mathcal{A}_{\text{Voronoi}} = \frac{1}{8} \sum_{j \in N_1(i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ \|\mathbf{x}_i - \mathbf{x}_j\|^2.$$

(b) Calculate Mean curvatures by multiple 0.5, the dot result between the $\mathbf{K}(x_i)$ and vertex normal. The formula can be shown below:

$$\mathbf{K}(\mathbf{x}_i) = \frac{1}{2\mathcal{A}_{\text{Mixed}}} \sum_{j \in N_1(i)} (cot\ \alpha_{ij} + cot\ \beta_{ij})\ \ (\mathbf{x}_i - \mathbf{x}_j)$$

(c) Calculate Gauss curvatures by following the formula as shown below:

$$\kappa_G(\mathbf{x}_i) = (2\pi - \sum_{j=1}^{\#f} \theta_j)/\mathcal{A}_{\text{Mixed}}$$

I designed a color coding schemes for both Mean and Gaussian curvatures. In the color scheme, the red color means the high positive value, the green color means the value with 0, and the blue color means the low negative value. Note: the value has not been normalized to the region between 0 and 1, thus the color result between the regular and irregular shape might be different because of the higher max/min value in irregular situation.

Here we show both regular (Torus) and irregular (Dragon) shape as a comparison:

Firstly, let's see the result of the Mean curvatures visualization in **Figure 14**:

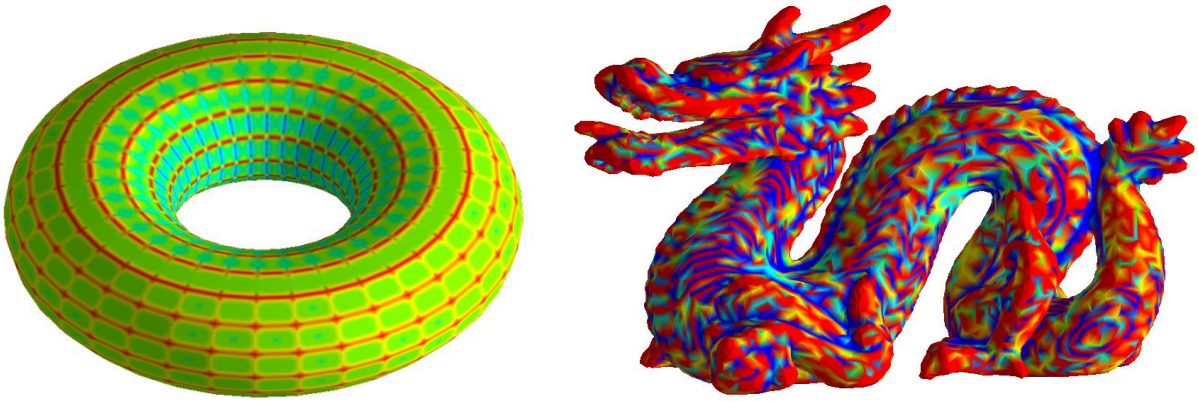


Figure 14. The Mean curvature visualization
(left regular shape, right irregular shape)

The result actually follows the definition of the Mean curvature. If we consider the maxim and the minim normal curvature, then the Mean curvature can be expressed as the $\mathbf{H = (k_1+k_2)/2}$. The formula tells us that the Mean curvature is the value of the mean normal curvature value in the neighborhood. Thus, under the law of Mean curvature, concave surfaces have negative values and convex surfaces have positive values. The saddle face or flat face's value is zero. Those two pictures satisfy the rule of Mean curvature visualization meaning. Specifically, because the mesh is not smooth enough, most of the Torus model are green color as they are almost flat faces and some are red and blue because discontinuity of the derivation leads to the local extreme values.

Secondly, let's see the result of the Gaussian curvature visualization in **Figure 15**:

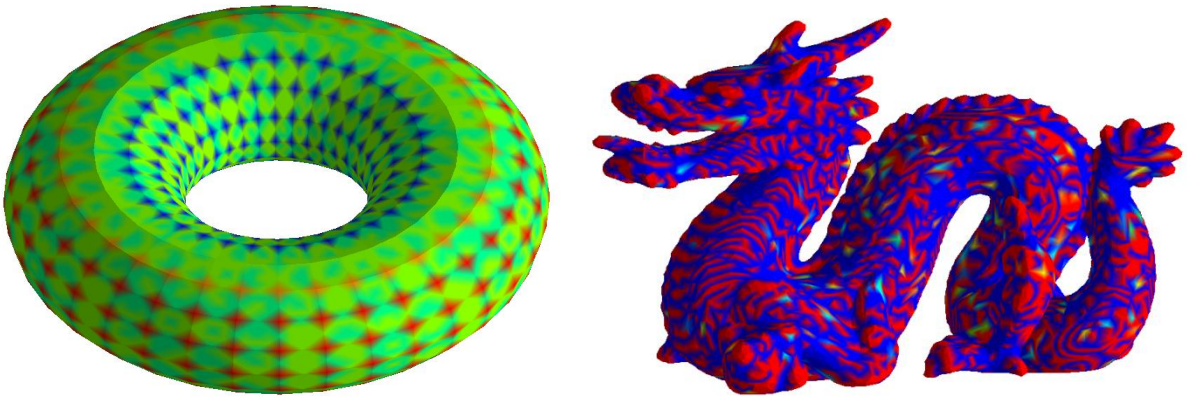


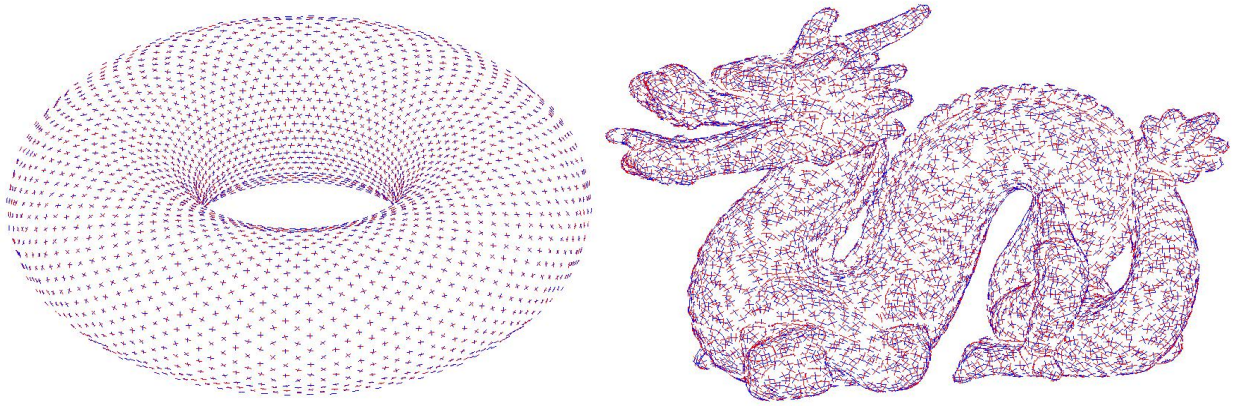
Figure 15. The Gaussian curvature visualization
(left regular shape, right irregular shape)

The result actually follows the definition of the Gaussian curvature. If we consider the maxim and the minim normal curvature, then the Gaussian curvature can be expressed as the $\mathbf{G = k_1 \cdot k_2}$. The formula tells us that the Gaussian curvature tends to choose the global extreme values. Thus, blue and the red only shows when the the vertices are either convex or concave than the points in the neighborhood.

For the curvature tensor, use crosses to show the principal curvature directions. For a stable estimation of the curvature tensor, you will need to smooth the initial estimation through the Laplacian smoothing process described in the class.

For the curvature tensor, I used the crosses to show the principal curvature directions. The basic idea of getting the principle curvature directions is trying to get the eigenvectors from the tensor matrix. The way of getting the tensor matrix is trying to solve the equation system derived from vertex tangent plane's tangents and its normal.

Here we both show the result of regular (Torus) shape and irregular (Dragon) shape. The red line means the major principle curvature directions and the blue line means the minor principle curvature directions as the **Figure 16** shows:



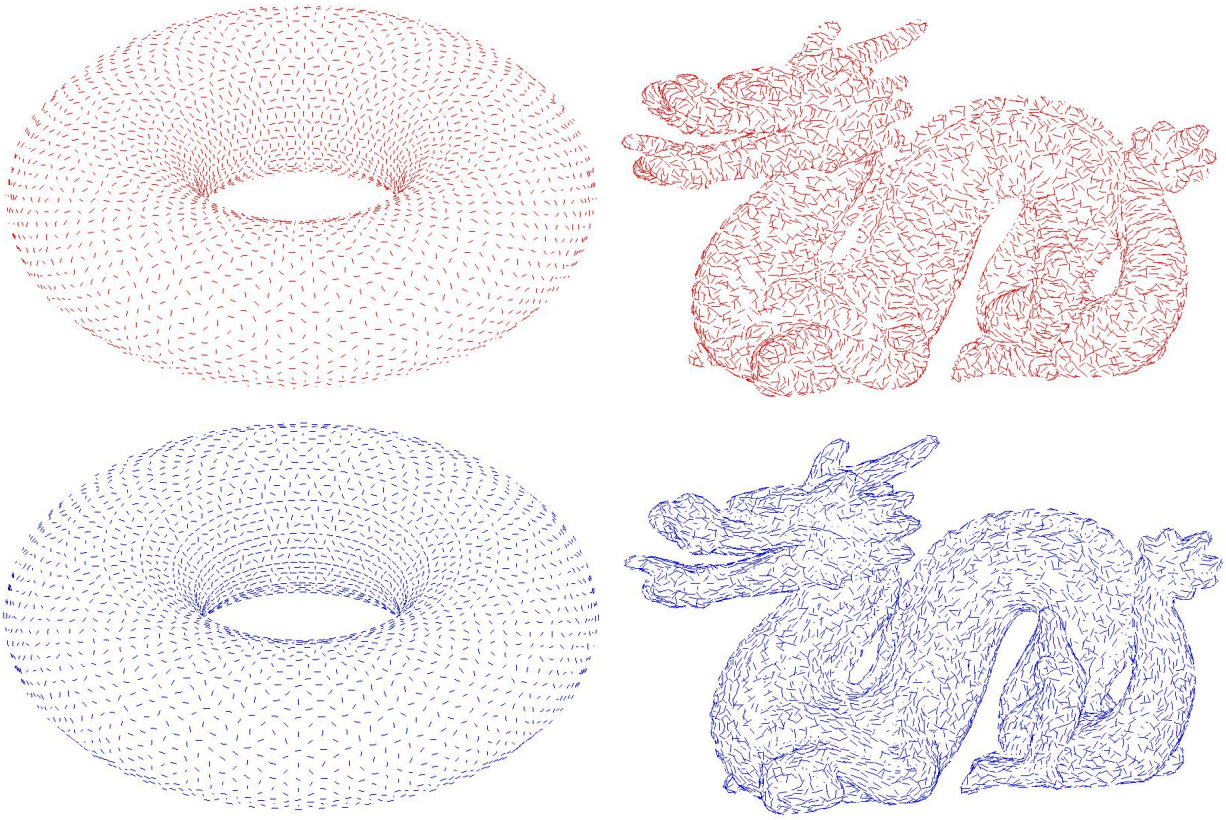


Figure 16. The Principle curvature directions visualization
(left regular shape, right irregular shape)
(First row the crosses,
Second row the major principle curvature directions ,
Third row minor principle curvature directions)

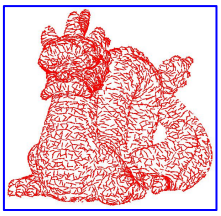
Right now, as we can see in **Figure 16**, the result of the principle curvature directions is kind of in a mess. Thus, for a stable estimation of the curvature tensor, we will need to smooth the initial estimation through the Laplacian smoothing process, which will be shown in the part **a**.

a. Compare the uniform, cord, mean curvature, and mean values weights for curvature tensor smoothing. Which scheme is more preferred? Justify your answers with theoretical and/or visual analysis.

Note: for simplify the pictures presenting and improve the visualization effect, we only show the major principle curvature directions (the red line segment) of Dragon, which means later analysis will use the half of each crosses rather than a full one. Also, we increase the width of the line to 1.4 so that we can easier see the change of directions. At the end of the part, we will show the full smoothed result for the preferred one.

To smooth the principle curvature directions of each vertices, we can think back to smooth each

vertices' tensor matrix. Here we transfer whole the vertices' tensor 2D matrices into 3D matrices so that we can do Laplacian smoothing process for each vertices. Thus, for different weight of averaging, there are 4 schemes we can choose. They are: Uniform, Cord, Mean Curvature (MC), and Mean Values (MV) weights. For the iterations, we use the Explicit method. The **Table 1-4** has shown the experiments:



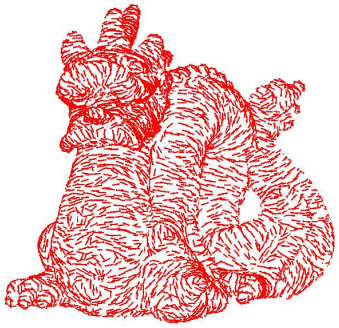
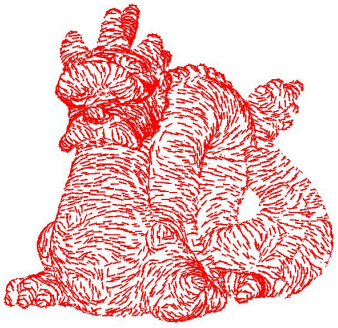
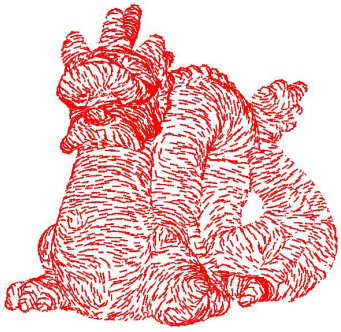
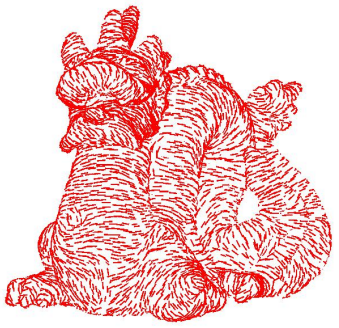
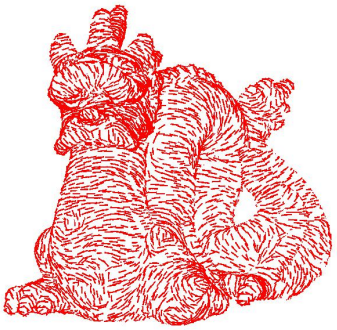
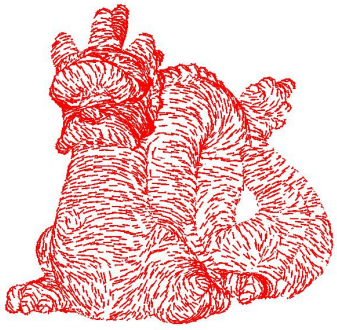
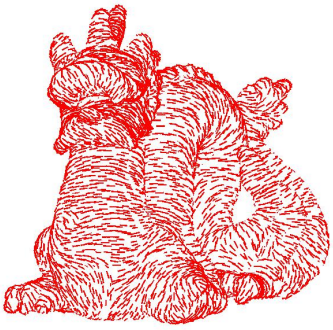
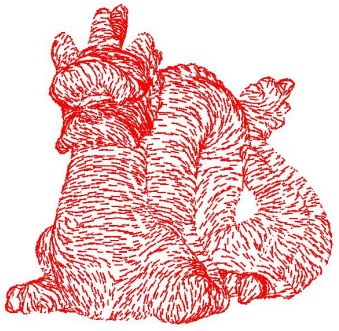
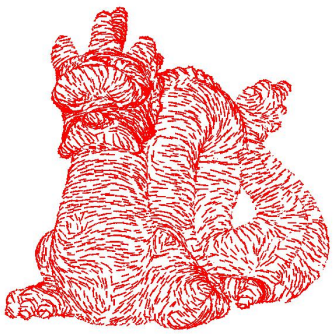
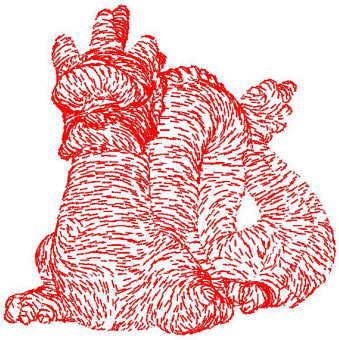
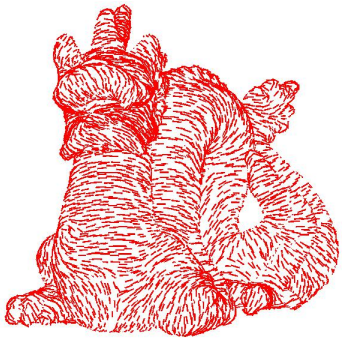
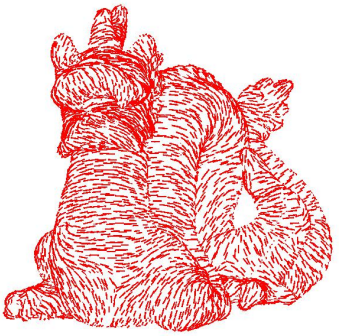
	5	10	15	20
.3	(+1.547s) 	(+1.495s) 	(+1.518s) 	(+1.502s) 
.6	(+1.532s) 	(+1.55s) 	(+1.463s) 	(+1.543s) 
.9	(+1.554s) 	(+1.486s) 	(+1.445s) 	(+1.487s) 

Table 1. Uniform scheme experiment

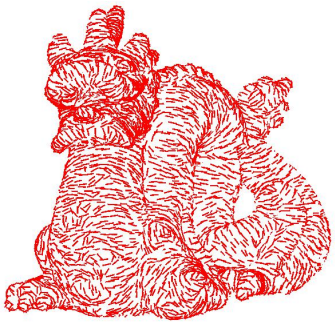
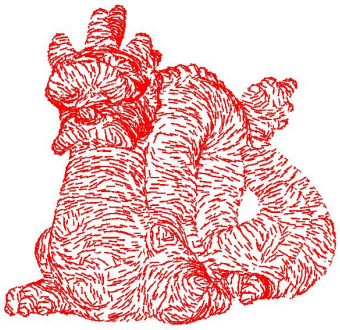
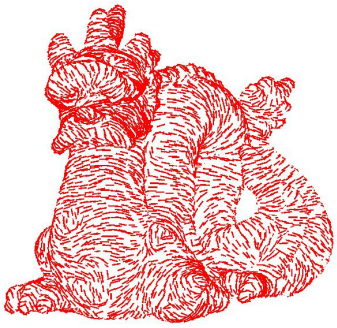
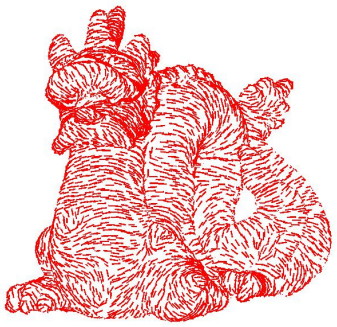
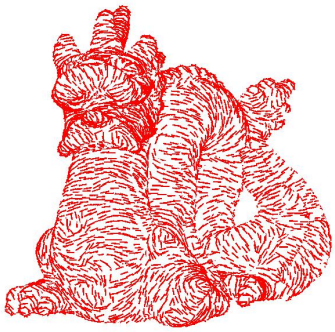
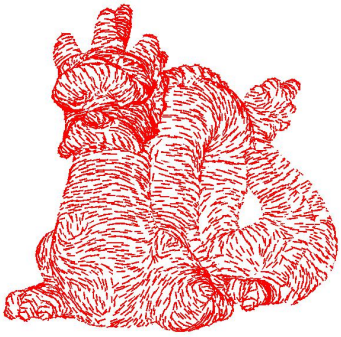
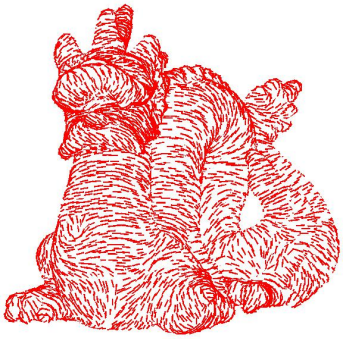
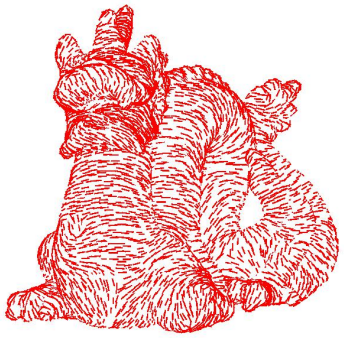
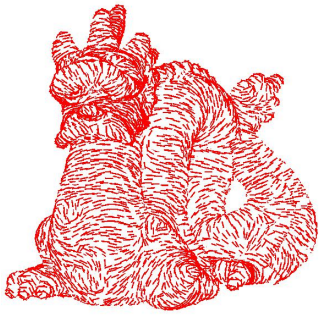
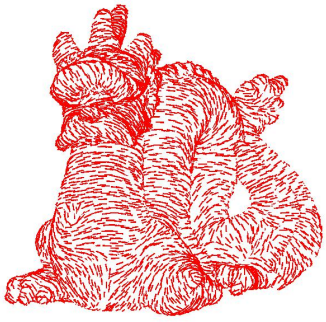
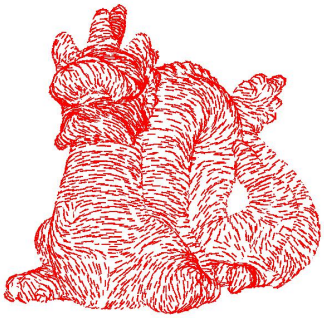
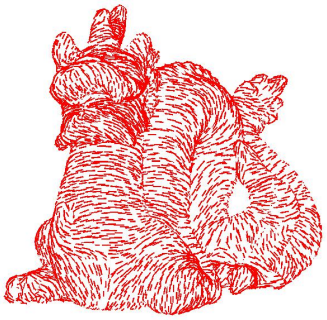
	5	10	15	20
.3	(+1.587s) 	(+1.612s) 	(+1.64s) 	(+1.639s) 
.6	(+1.507s) 	(+1.557s) 	(+2.156s) 	(+1.5s) 
.9	(+1.514s) 	(+1.574s) 	(+1.563s) 	(+1.518s) 

Table 2. Cord scheme experiment

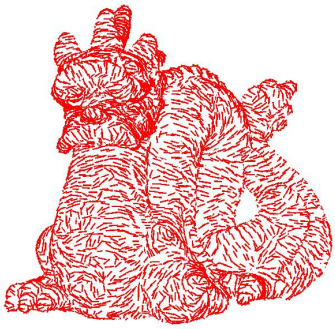
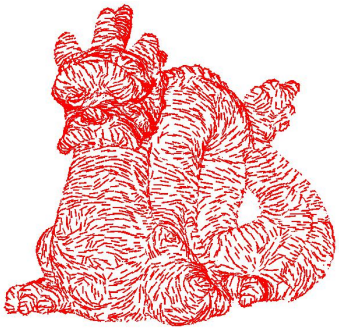
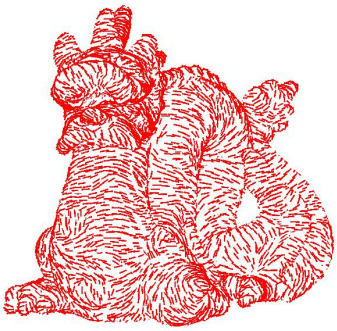
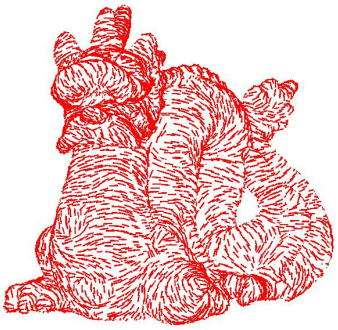
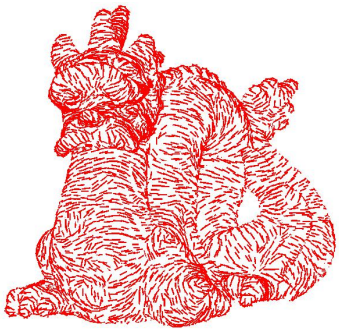
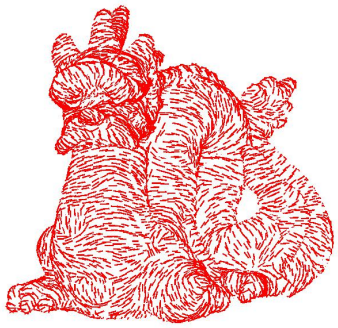
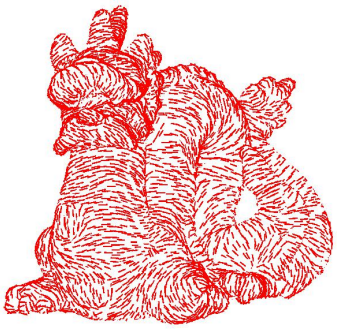
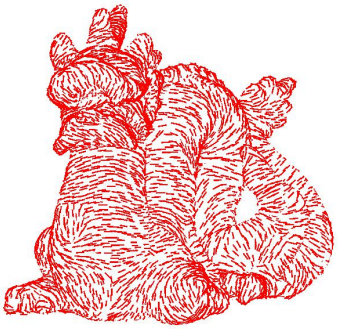
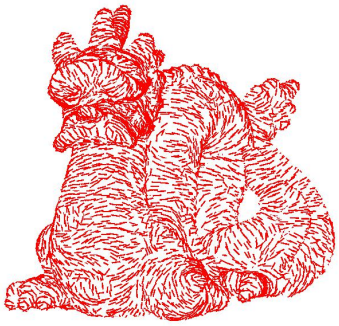
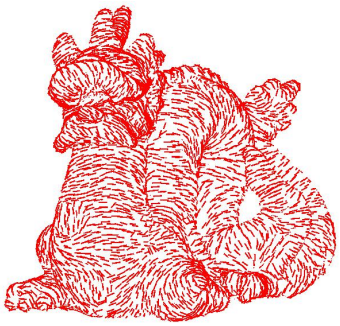
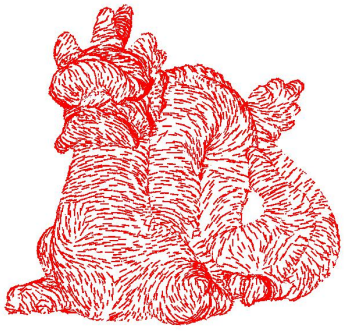
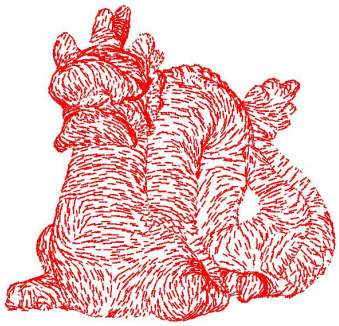
	5	10	15	20
.3	(+1.556s) 	(+1.533s) 	(+1.632s) 	(+1.554s) 
.6	(+1.559s) 	(+1.542s) 	(+1.701s) 	(+1.683s) 
.9	(+2.012s) 	(+2.047s) 	(+2.031s) 	(+2.012s) 

Table 3. Mean Curvature scheme experiment

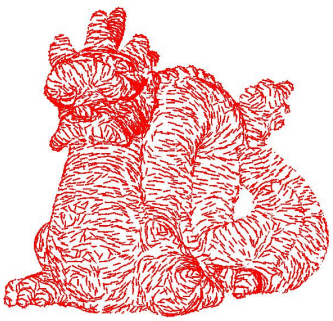
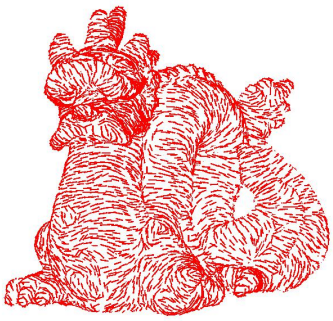
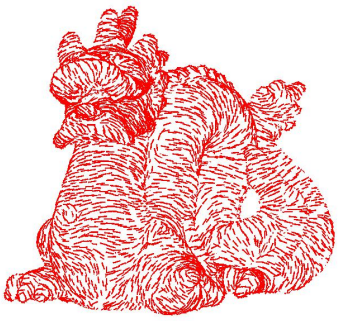
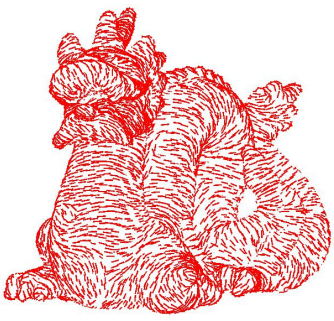
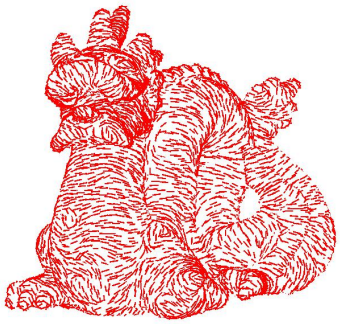
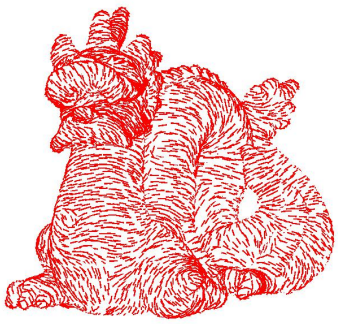
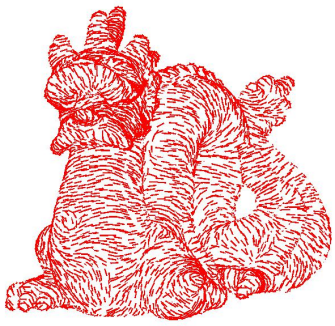
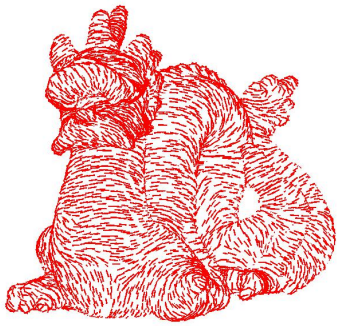
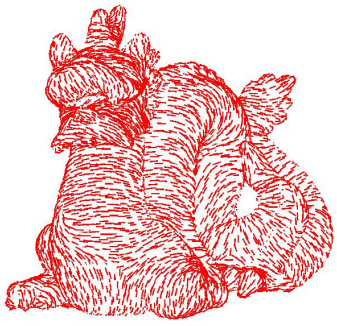
	5	10	15	20
.3	(+1.491s) 	(+1.506s) 	(+1.52s) 	(+1.507s) 
.6	(+1.515s) 	(+1.553s) 	(+1.498s) 	(+1.502s) 
.9	(+1.542s) 	(+1.538s) 	(+1.476s) 	(+1.482s) 

Table 4. Mean Value scheme experiment

From the experiment result, I would prefer to choose the Mean Value scheme as my choice. The reason can be concluded in 4 points based on **Table 1-4** and **Figure 17-20**:

(1) *Smooth efficiency*: this one is measured by the optimized result with it iteration times. We can choose the same place elements in the first row, fourth column from **Table 1-4** as the yellow square showing. The reason we choose the element at (1, 4) is that the small time step will keep the stability of each algorithm and also the 20 iteration times is fairly enough for tensor smoothing. It is not hard to find that, from the four yellow squares, the MV scheme can eliminate the most numbers of the “truculence”. Thus, MV relatively has the higher tensor smooth efficiency.

(2) *Time cost*: if we observe the average time of each schemes, we can find that the

operating speed can be queue as such a sequence: MV > Uniform > Cord > MC. Although the number of the experiment samples are not enough, it is not hard to feel the faster speed of MV and Uniform schemes when we run the program tensor smoothing. Thus, MV and Uniform can be the best choice of time saving.

(3) *Stability*: there are three aspects will effect the iteration stability: equation solver scheme, time step and iteration times. For the equation solver scheme, I used the explicit method based on the reason that it will use as much as possible old information and explicit method can quickly shows the ill nature of iterations when the time step is large. Thus we only need to think about time step and iteration times. Thus we can see the result after 20 times iteration under the 0.9 time step, as **Figure 17-20** showing. By watching the purple circle in **Figure 18** and the blue circles in **Figure 19**, we can see that: MC method is easy to show the instability once the time step is too large. For Cord method, even if the number of the irregular points is small, the instability still exist. Thus the Uniform and MV scheme can preserve the original smoothing characteristics of the tensor field.

(4) *Details*: right now, from the first three norms, we can conclude that the Uniform scheme and the MV scheme would be the best choice. However, this norm will guide us to choose the MV as the best tensor smoothing scheme. Compared with the same place in the region of **Figure 17**'s green circles, from **Figure 20**, we can easily find that the MV can handle the routing of tensor's eigenvectors directions more carefully and smoothly.

Figure 17. The Uniform scheme result (dt = 0.9, iterating 20 times)

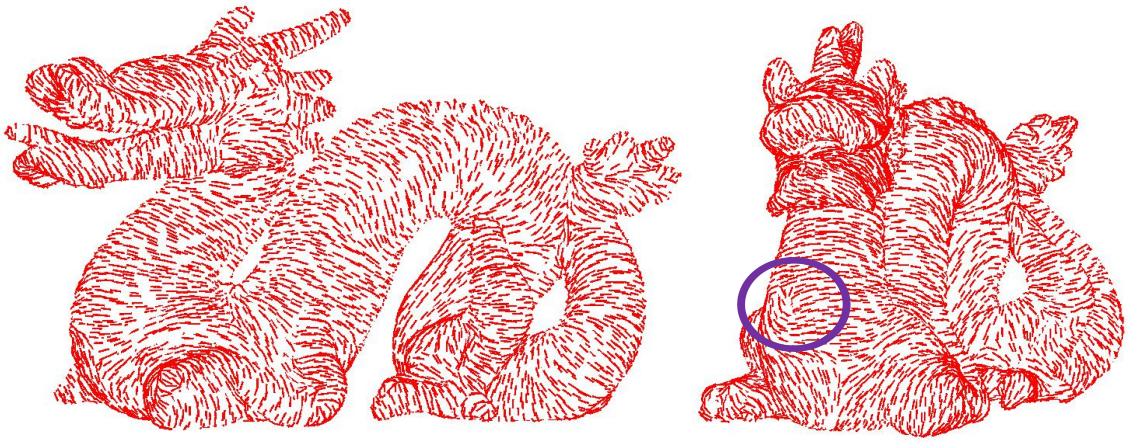


Figure 18. The Cord scheme result (dt = 0.9, iterating 20 times)

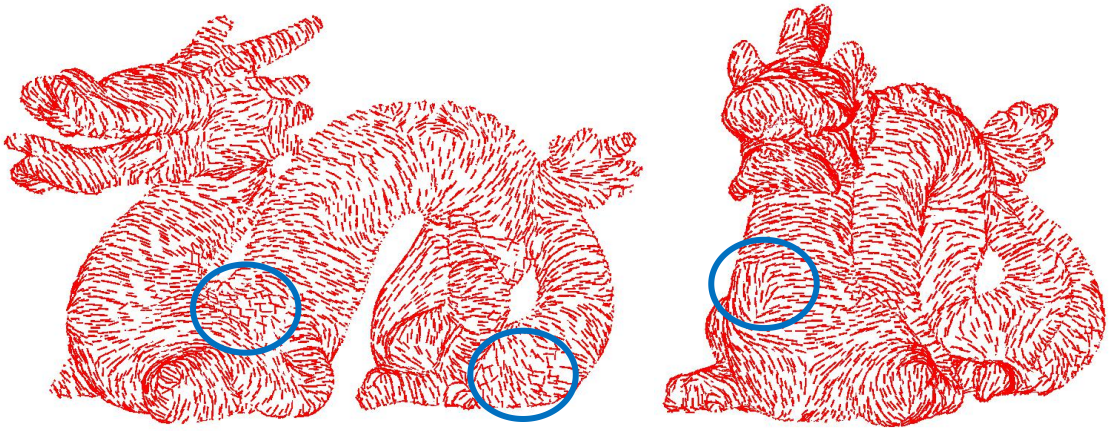


Figure 19. The Mean Curvature scheme result (dt = 0.9, iterating 20 times)

Figure 20. The Mean Value scheme result (dt = 0.9, iterating 20 times)

Based on all the analysis from the 4 norms and the experiment results shown in **Table 1-4** and **Figure 17-20**, we can choose MV as our best smoothing scheme without any hesitation. The final full result of MV scheme after tensor smoothing can be shown in **Figure 21**, which can be used to compare the original result from **Figure 16**:

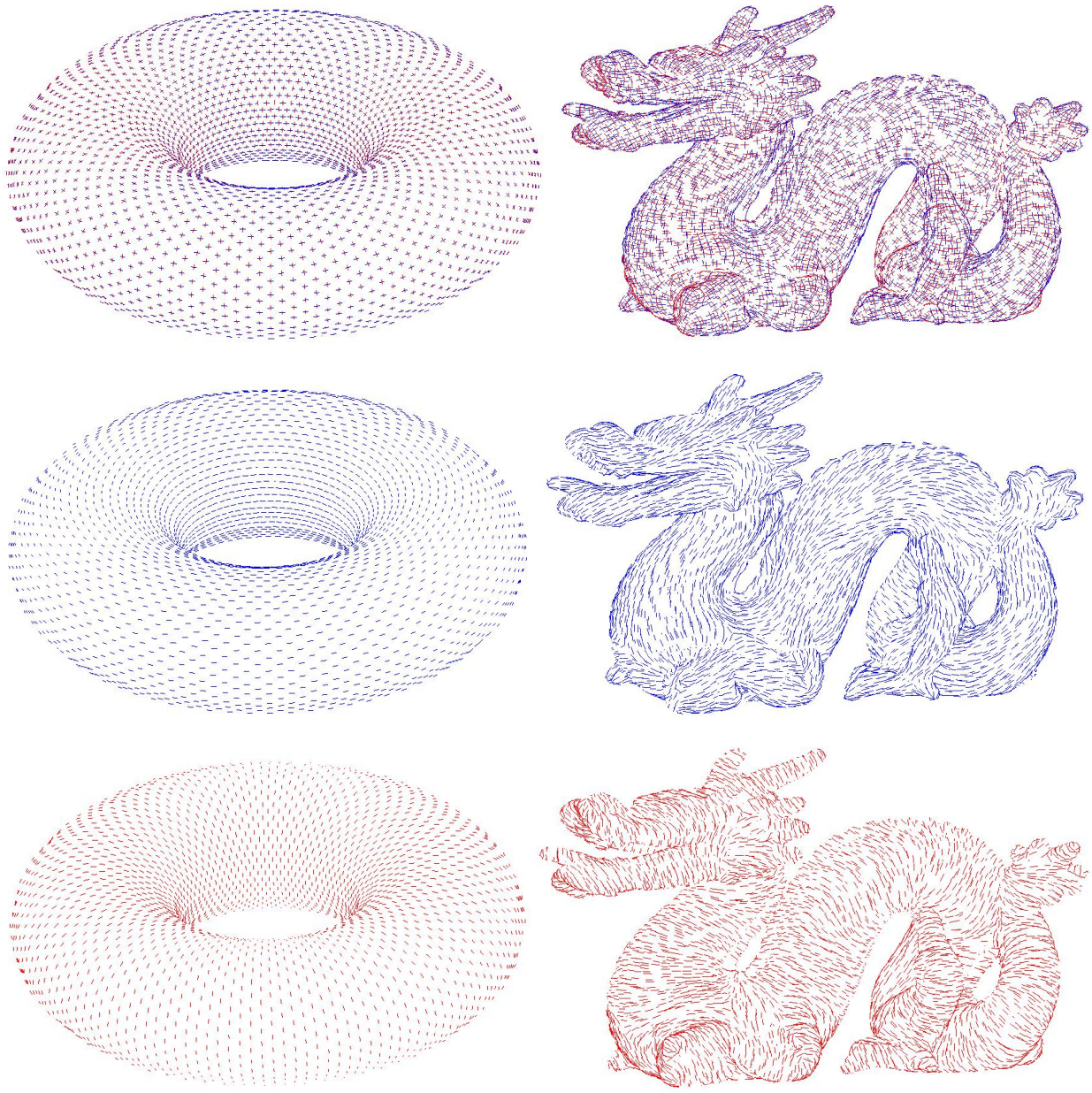


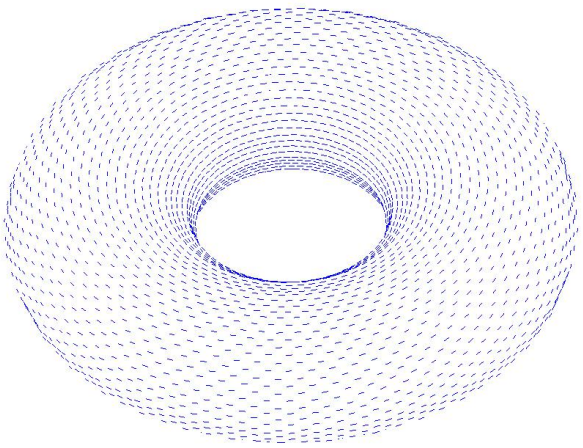
Figure 21. The Principle curvature directions visualization after tensor smoothing (left regular shape, right irregular shape, dt = 0.9, iterating 20 times)
(First row the crosses,
Second row the major principle curvature directions ,
Third row minor principle curvature directions)

b. Are you satisfied with the curvature estimation algorithm you have implemented even after smoothing? Did you find cases where it is not working well? Which parts of the algorithm do you think can be improved, and how do you propose to improve them? Make sure to draw conclusions only after having carefully examined all the test models from various viewpoints and tried different smoothing parameters.

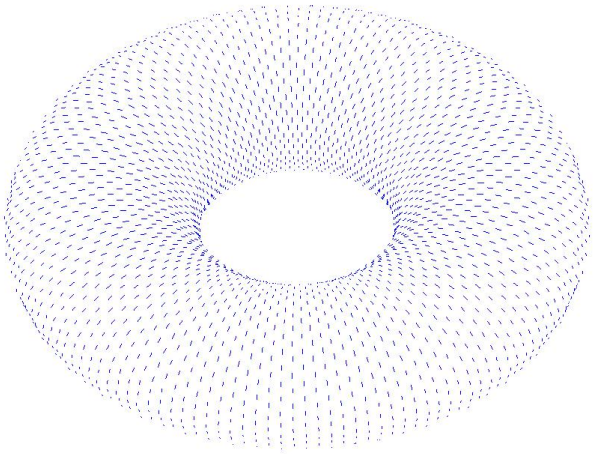
I think most part of the estimation algorithm, especially the estimation of the Mean Curvature and the Gaussian Curvature, is okay. For the three simple model: `Tetrahedron`, `Octahedron`, and `Icosahedron`, they are always stable when doing tensor smoothing, so they will not be discussed in this part.

However for the estimation of the curvature tensor (or the principle curvature), there are some problems. Same for easy statement, we only choose one principle curvature directions, at this time we choose minor principle curvature. Under the MV scheme with time step 0.9 we can see some **morbid iterating results** of some models as shown below:

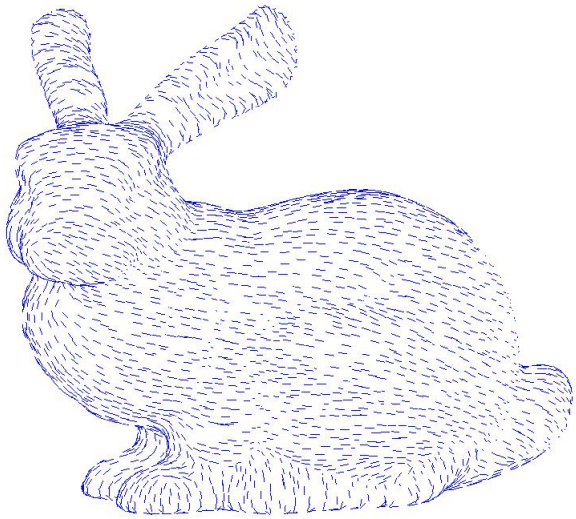
Problem 1. *Direction change*: when we use the Explicit method with MV scheme for tensor smoothing, almost all the models they can quickly get smoothed in 20 times. As the **Figure 22** shows, the tensor field can stay stable in the range of iteration time between at least 20 - 100 times. However, the vertex's tensor break the stable state when the iteration times get higher than 200 times. Finally when the iteration time get larger than 500 times, most of the principle curvature directions will change 90 degrees of angle, which is really bad because it means the minor principle curvature finally get into the major principle. This is the worst case. Once the system get unstable it goes the it's 100% reverse situation.



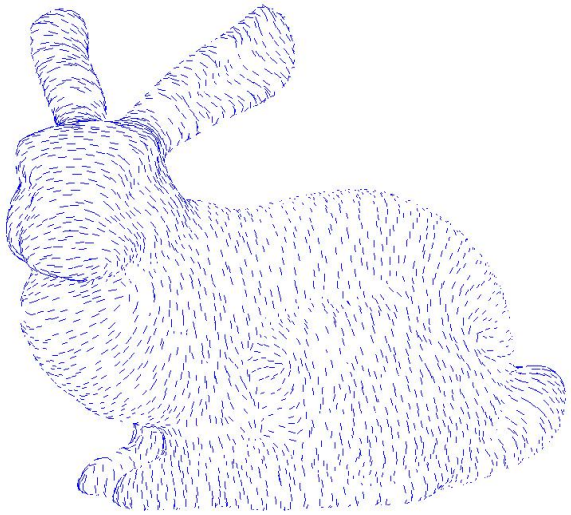
50*2 =100 times



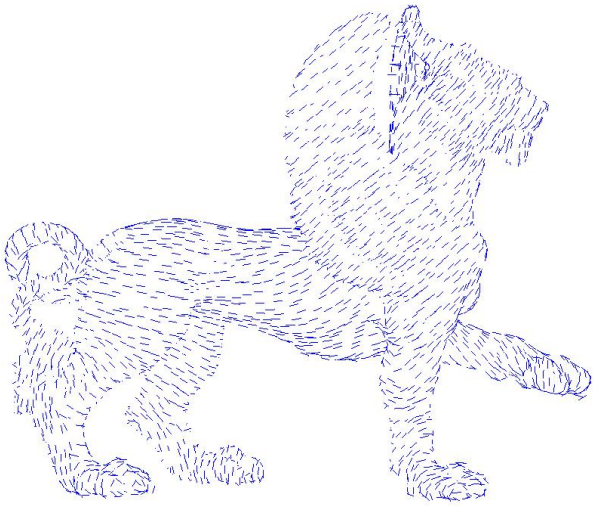
50* 10 = 500 times



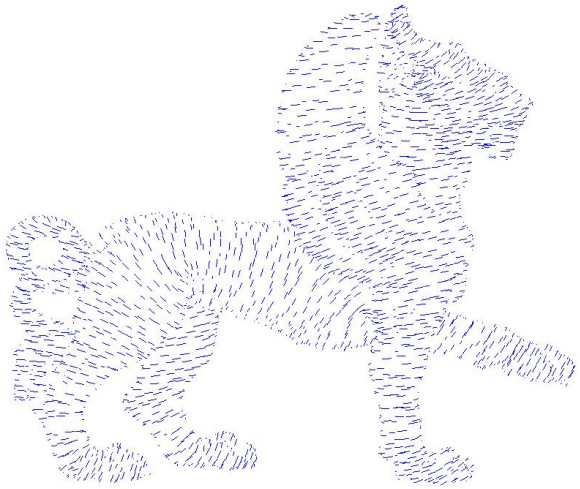
50*4 = 200 times



50*15 = 750 times

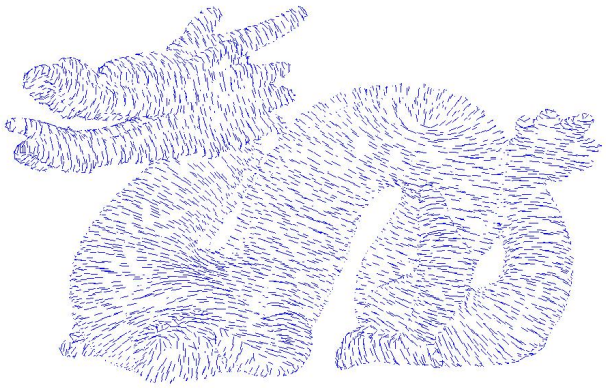


50*2 = 100 times

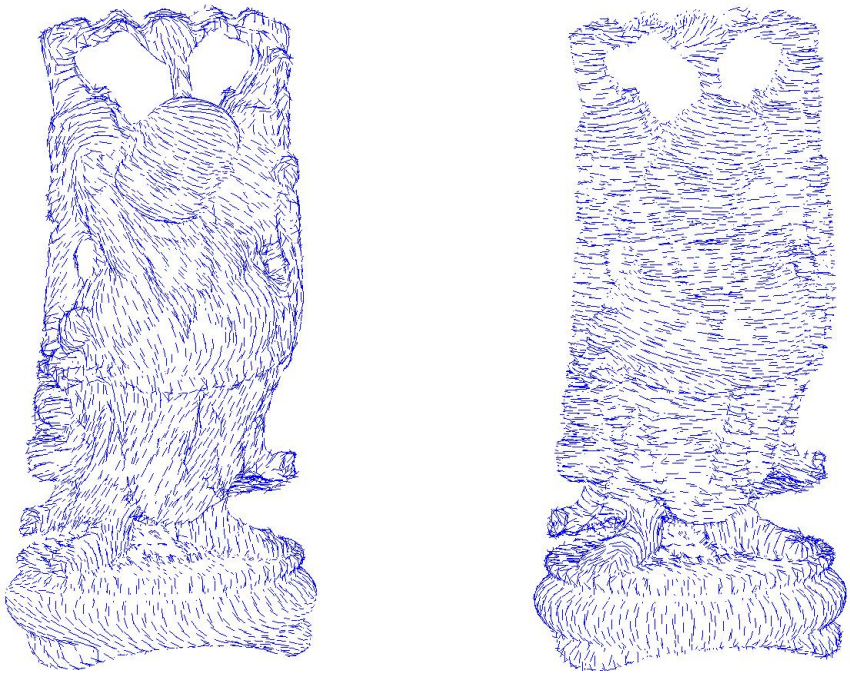


50*10 = 500 times

50*2 = 100 times



50*10 = 500 times

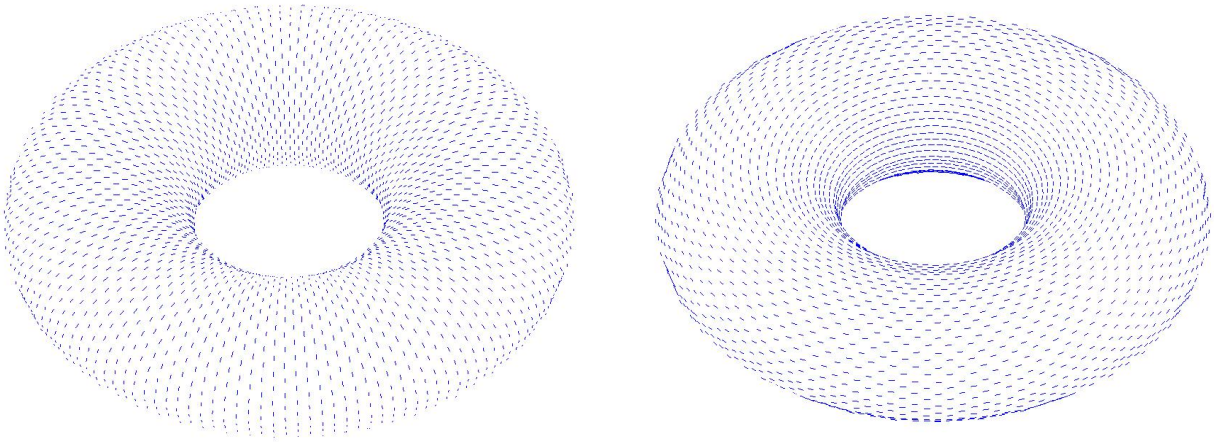


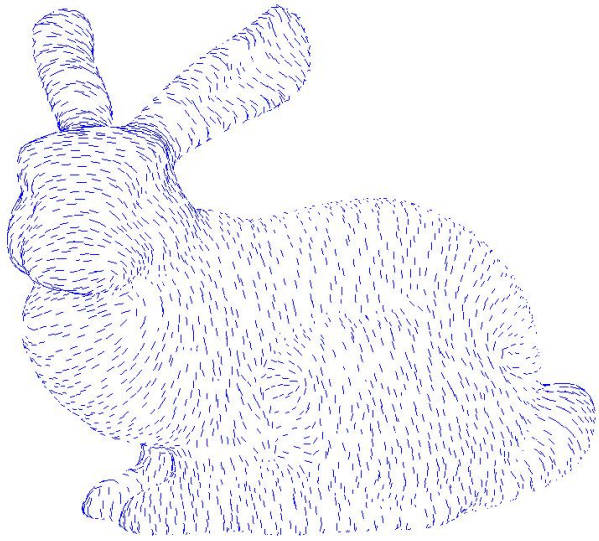
50*2 = 100 times　　50*20 = 1000 times

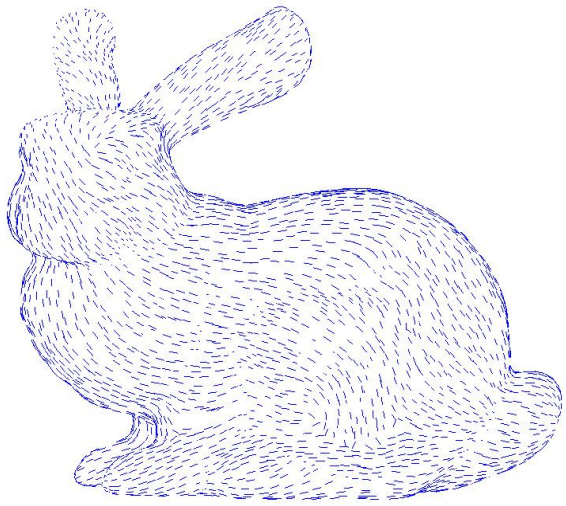
Figure 22. The Principle curvature direction morbid changing with high iteration time (left healthy result, right morbid result) (Torus, Bunny, Feline, Dragon, Happy models)

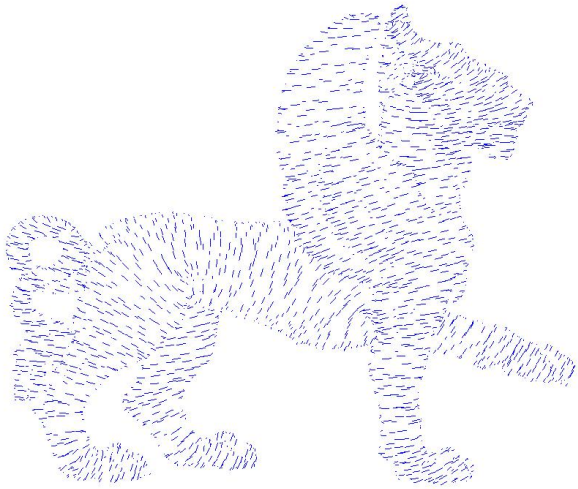
From **Figure 22**, besides the Sphere model and the simple models, we can see that most of them all shows the **morbid** iterating results. Thus, in order to solve this problem, there are three directions improve the algorithm.

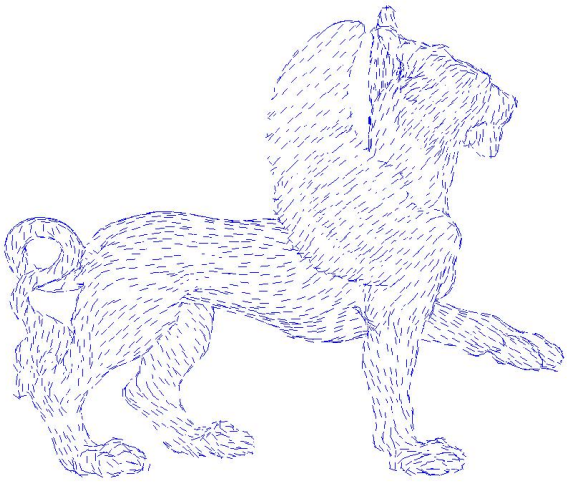
The first one is obviously decrease the time steps. Obviously, the Explicit method ask us to do this, because it cannot stand the long time step. So we change the time step from 0.9 to 0.3. As the **Figure 23** shows, under the same 500 iteration times, short time steps perfectly solved this problem.

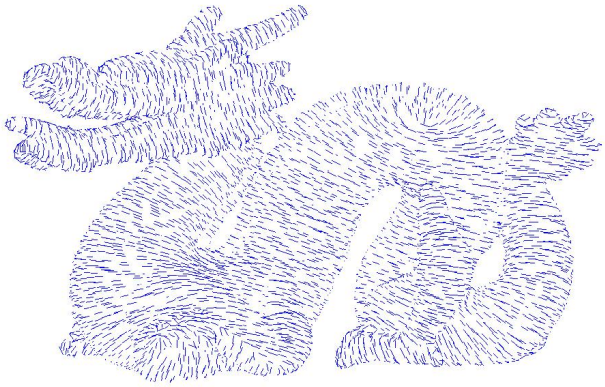


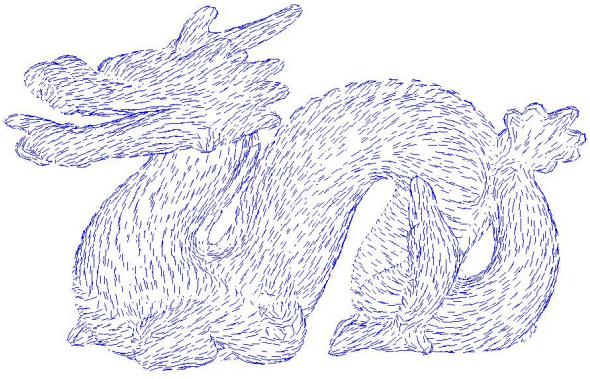












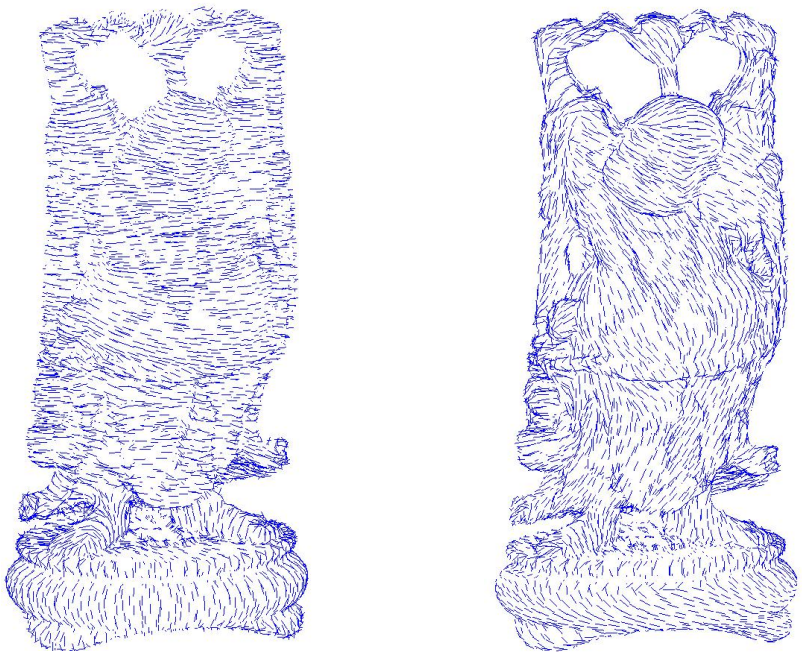


Figure 23. The improvement from the long time step (0.9s) (left) to the short time step (0.1s) (right) in 500 iterations

The second way to solve the unstable problem is trying to use the implicit method. Basically it is a way to allow our MV algorithm use the big time step. However, right now with 500 iteration times can be enough guarantee the final stable tensor smoothing result. So we will not repeat this method but just show some pseudo code as shown below:

```
Push back Matrix A
Vector b is the initial tensor
for (int i = 0; i < poly->nverts; i++)
    {
        for (int j = 0; j < n; j++)
            {
                p_sum += wij * (poly->vlist[j]->tensor3D - poly->vlist[i]->tensor3D);
                ...
                Set the data of coefficient triple of Matrix A (push_back())
                Set boundary variable's data to the right side b
                ...
            }
    }
X = Solver A(b);
Get the eigenvalue from matrix: translate*X*translate^T
for (int i = 0; i < poly->nverts; i++)
    poly->vlist[i]->tensor eigenvectors = eigenvecors;
```

The third way is try to use the local matrix operations for each vertex's tensor smoothing, rather than using the 3D tensor global matrix. Although, in my algorithm, I only do the transform from 3D back to 2D after the whole 3D matrix smoothing, the final 3D to 2D transform would still cost some data losing. The tensor smoothing process is very sensitive. Any singular direction will cause the unstable result. Thus, try the Parallel Transport technique would be the perfect way to eliminate data losing.

Problem 2. *Mesh Shape effect*: There is another problem for the model like sphere. Although, the final result of the sphere is stable as the first two pictures showing in **Figure 24**. No matter how many iteration time, the tensor smooth result always stay the same. However, we can obviously find that, the final principle directions almost distribute by following the shape of the model mesh as the mesh picture shown in **Figure 24**.

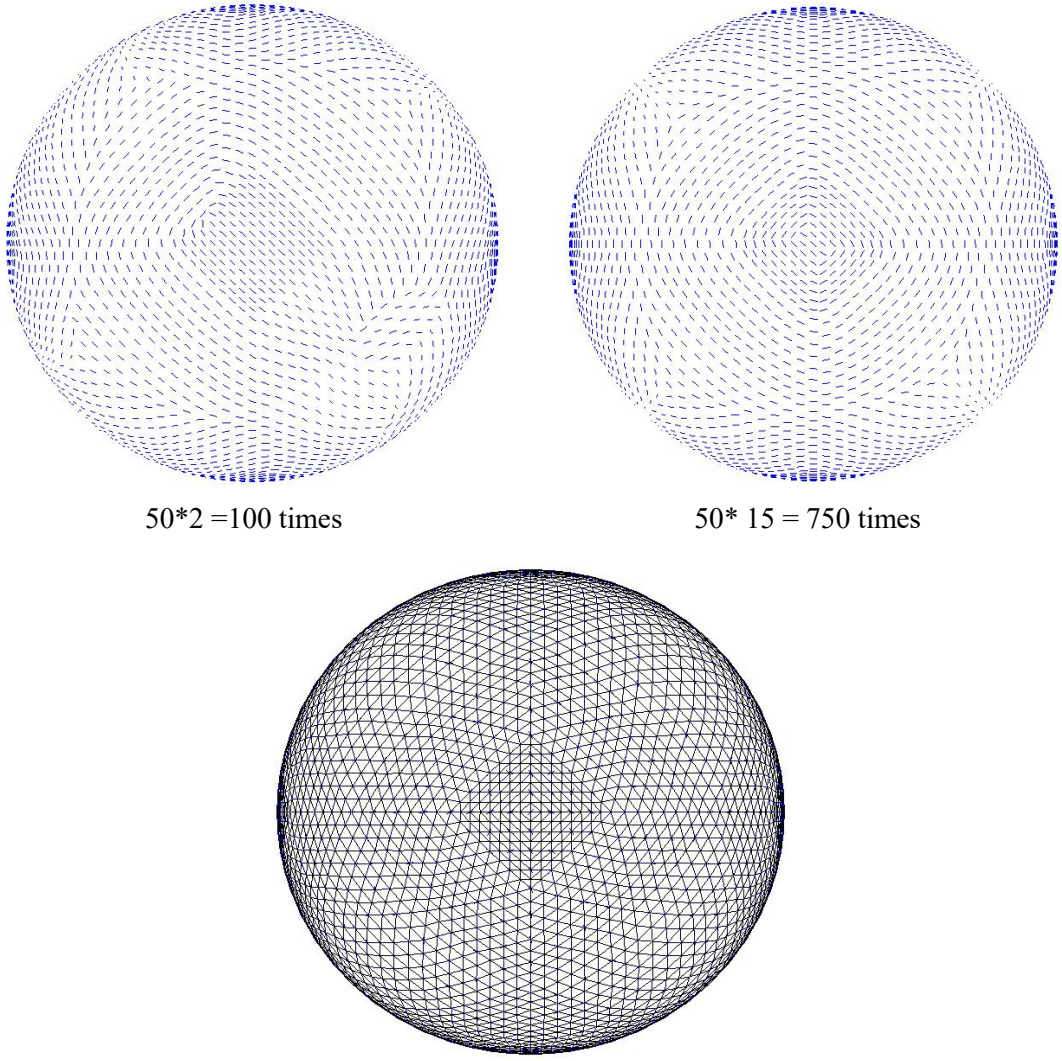
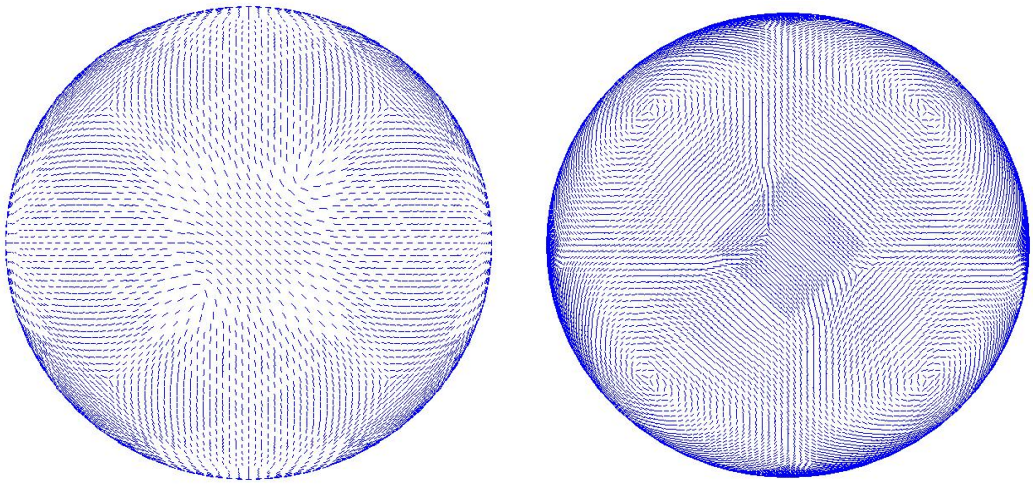


Figure 24. The stable tensor smoothing result of the Sphere model Compared with the original meshes

So basically, in order to get a smoother principle curvature direction field, we need to eliminate the effect of the model mesh. Thus, there are two plans for the improvement:

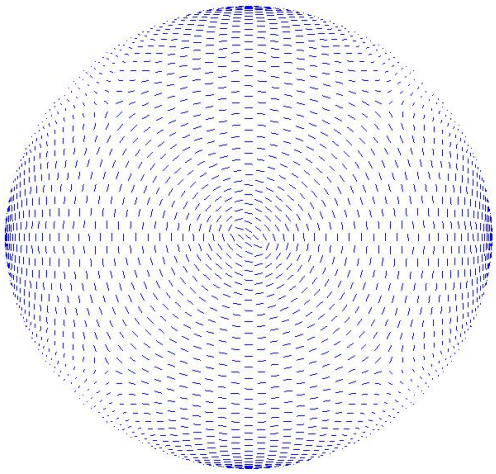
The first idea is before doing the tensor smoothing, we do subdivision first. As we can see from the **Figure 25**, even if the density of the principle curvature direction field increased, the final result still show some symmetrical textures related to the mesh macroscopically. So no matter irregular subdivision or the regular subdivision



Irregular regular

Figure 25. The stable tensor smoothing result after doing subdivision

The second idea is before doing the tensor smoothing, we do mesh smoothing first. As we have talked in the project 2 before, the Mean Value Coordinate scheme performs the best quality of the mesh smoothing result. Thus, here before doing the tensor smoothing, we try to implement MVC scheme for 20 times iteration with 0.9 time step.



Mesh Smoothing (20 times)

Figure 26. The stable tensor smoothing result (50* 15 = 750 times tensor smoothing) after doing 20 times MVC mesh smoothing

The final result as **Figure 26** shows, it eliminates the affection from the original model meshes. We can see from the picture that all the vector fields surrounded only with polar of the shape and there are no any textures from the original model mesh. This final result shows a good principle curvature direction field.

3. (Pen-and-ink: Graduate Student Only) Apply your curvature estimation algorithm to pen-and-ink sketching of a 3D model according to the algorithm described in the class. There will be a streamline starting from the center of each triangle and moving in either direction for five more triangles (11 triangles total per streamline). When crossing an edge, you need to compute the direction of the streamline in the new triangle T. Implement the following two ways and compare them.

Basically, for the method of part a and b, I have concluded 4 steps for implement them:

a. In the global 3D tensor dimension, for each triangle, interpolate it's vertices' 3D global tensors into the center of it, as the left picture of **Figure 27** shows. We get $T_{c\text{-}3D}$ by doing $(T_1+T_2+T_3)/3$. Then to get the principle direction on each triangle face, we change the 3D global tensor of each triangle into 2D local tensor $T_{c\text{-}2D}$ by doing $(T_1,\ T_2,\ Tris_{Normal})\cdot T_c\cdot(T_1,\ T_2,\ Tris_{Normal})^T$, in which T_1 can be each edge vectors in a triangle, and T_2 would be the cross product result between T_1 and the triangle's normal. The step can be seen from the right side picture of **Figure 27**. Finally, by getting the eigenvectors of $T_{c\text{-}2D}$, we can get the principle curvature directions for each triangle surface.

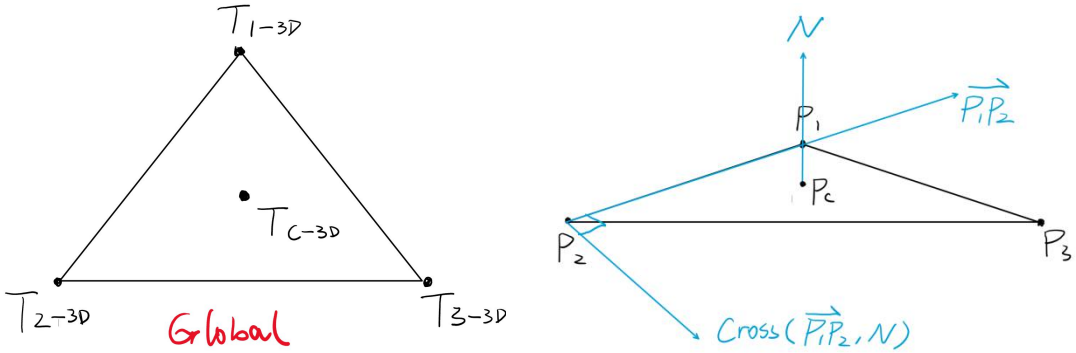


Figure 27. Triangle tensor interpolation, transform matrix construction

b. For each step's start point P, we try to decide which edge of the triangle will be intersected by the streamline start from the start point. Assume that there is an edge of a triangle. For the point $\vec{p}$ and the edge $\overline{ab}$, we do the judgement: $(\overrightarrow{pa}\times\vec{d})\bullet(\overrightarrow{pb}\times\vec{d})<0$, in which $\vec{d}$ is one of the principle directions in the triangle. If the start point P and the edge satisfy the judgement formula, then the streamline start from P will intersect to the edge $\overline{ab}$. The **Figure 28** shows the process.

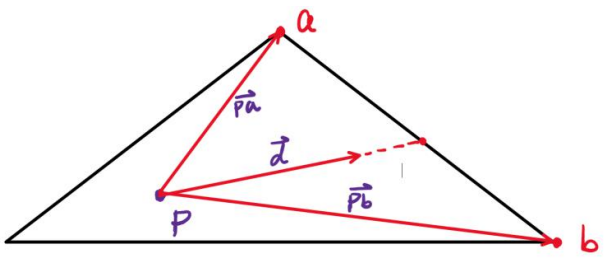


Figure 28. Intersection dot calculation

c. Right now we get whole the data that we want for calculating the intersection point. However, in the 3D space, it is a little bit hard to calculate the intersection point in 3D. I have shown my idea of the equation system in **Figure 29**.

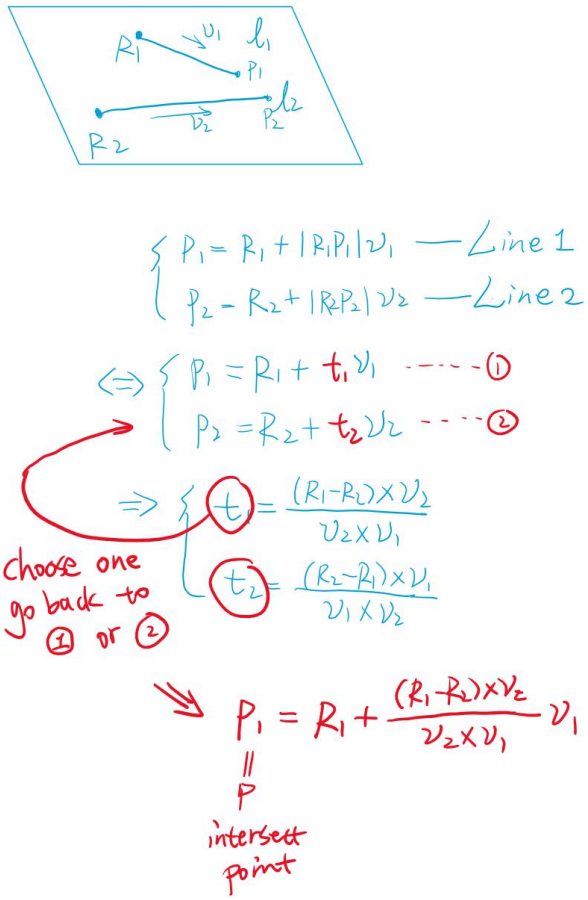


Figure 29. Equation system for the 3D intersection points

d. After having the technique of getting the intersection points, for each triangle, we save the center point of the triangle, 5 intersection points 3D coordinates in the major principle curvature directions and 5 intersection points 3D coordinates in the minor principle curvature directions. When doing rendering, we draw the streamlines by traversing all the mesh triangles.

a. Use the major or minor eigenvector direction based on the curvature tensor in T.

For the regular model such as Sphere and Torus, I do 200 times tensor smoothing. For the irregular model such as Bunny, Feline, Dragon, and Happy, I do 400 times tensor smoothing. The final result can be seen in **Figure 30**.

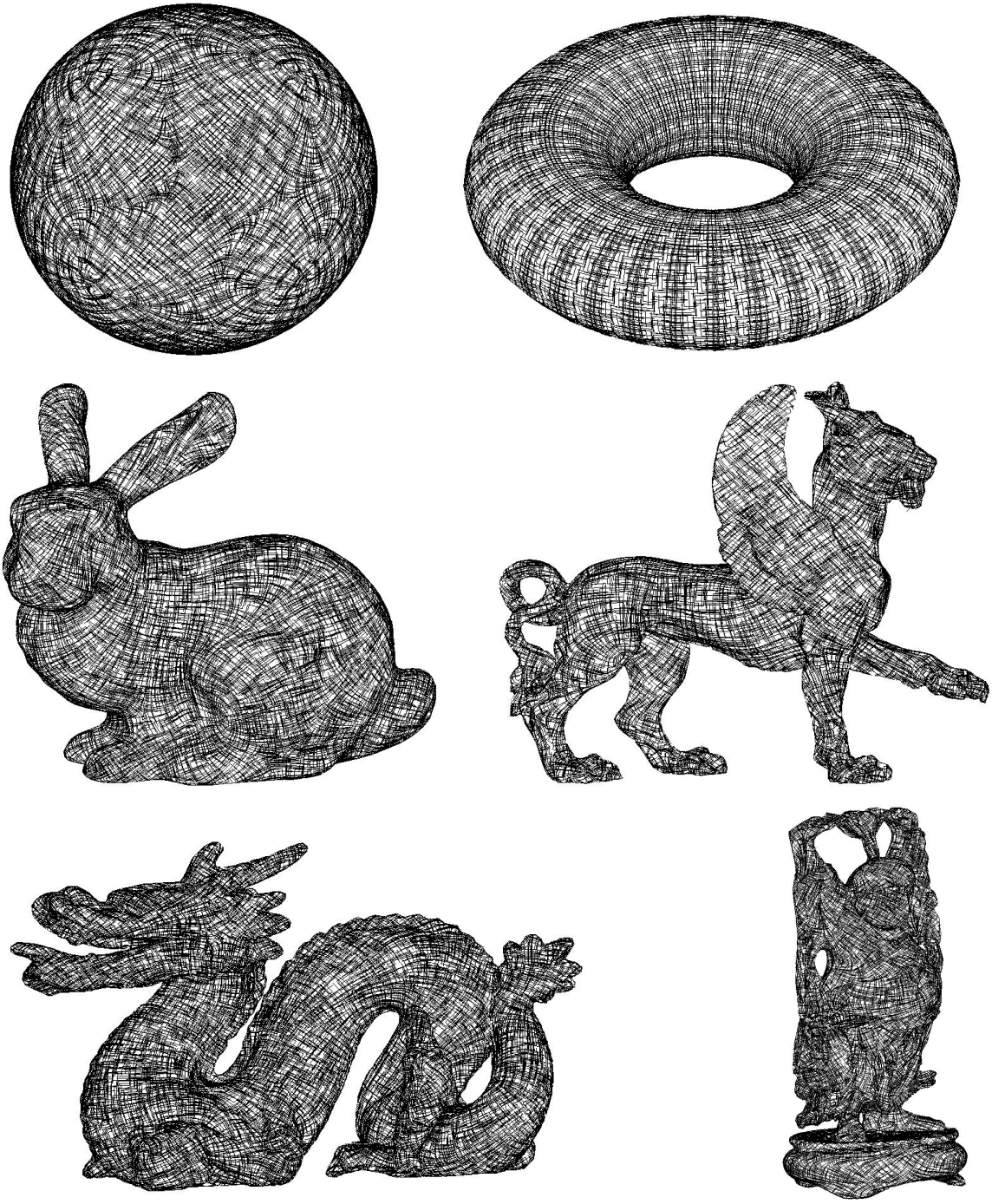


Figure 30. The Fold method results
(200-400 iteration times, MV scheme, 0.1 time step)

b. Unfold the triangle T to be in the same plane as T', then continue using the direction from the major or eigenvector direction from T.

For the regular model such as Sphere and Torus, I do 200 times tensor smoothing. For the irregular model such as Bunny, Feline, Dragon, and Happy, I do 400 times tensor smoothing. The final result can be seen in **Figure 31**.

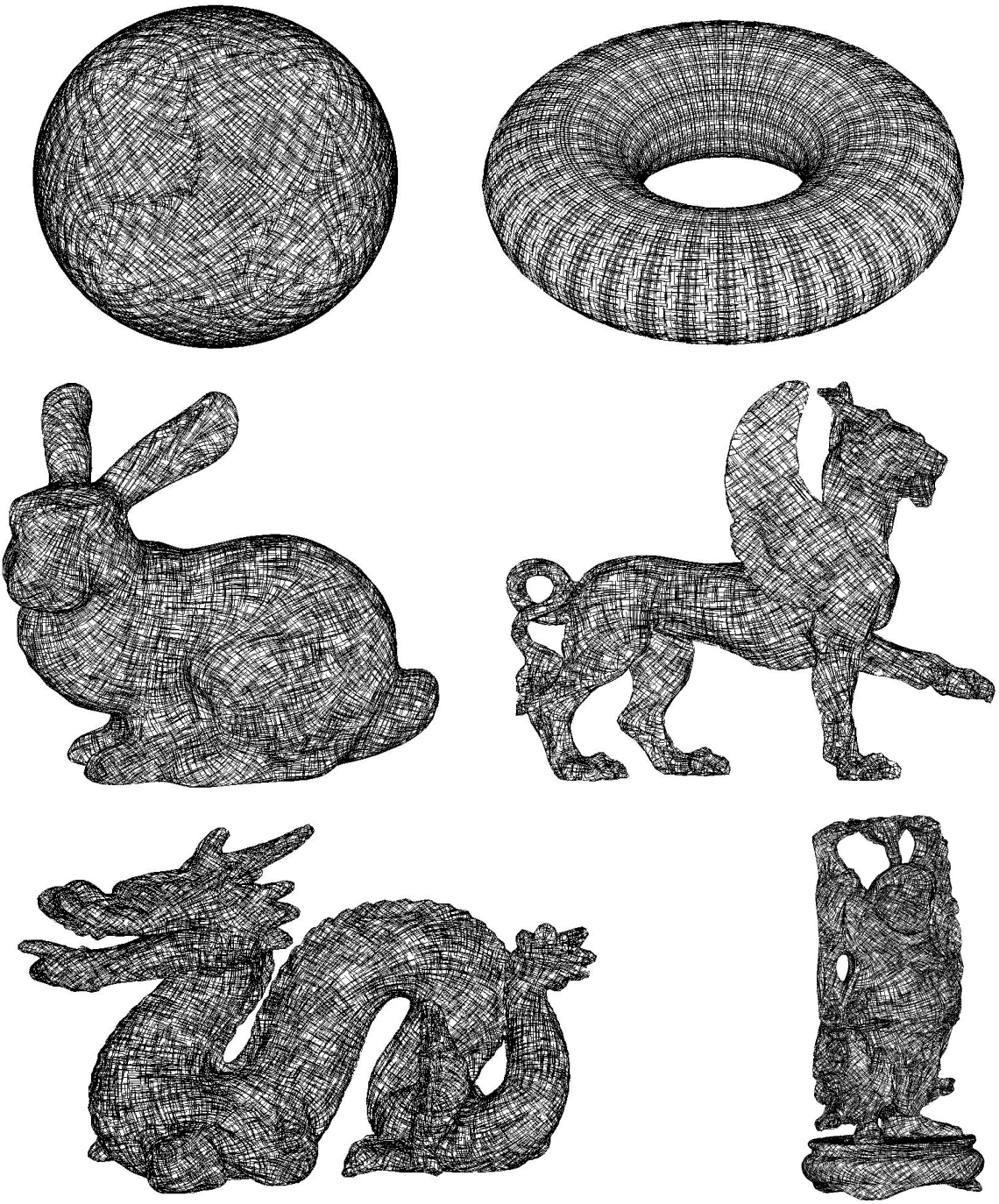


Figure 31. The Unfold method results
(200-400 iteration times, MV scheme, 0.1 time step)

Do the two methods give you comparable results? Are you satisfied with either approach? Discuss their relative strengths and weaknesses. What is a common problem with both approaches, and how would you fix the problem? In addition, are you satisfied with the final sketching results? If not, which parts of the pen-and-ink sketching pipeline do you plan to improve and how?

Yes, their character shows clear difference in some points. I would be more satisfied with the Fold method for the stream line drawing. I would take the Torus and the Bunny model as examples whose major direction details has been shown in **Figure 32**.

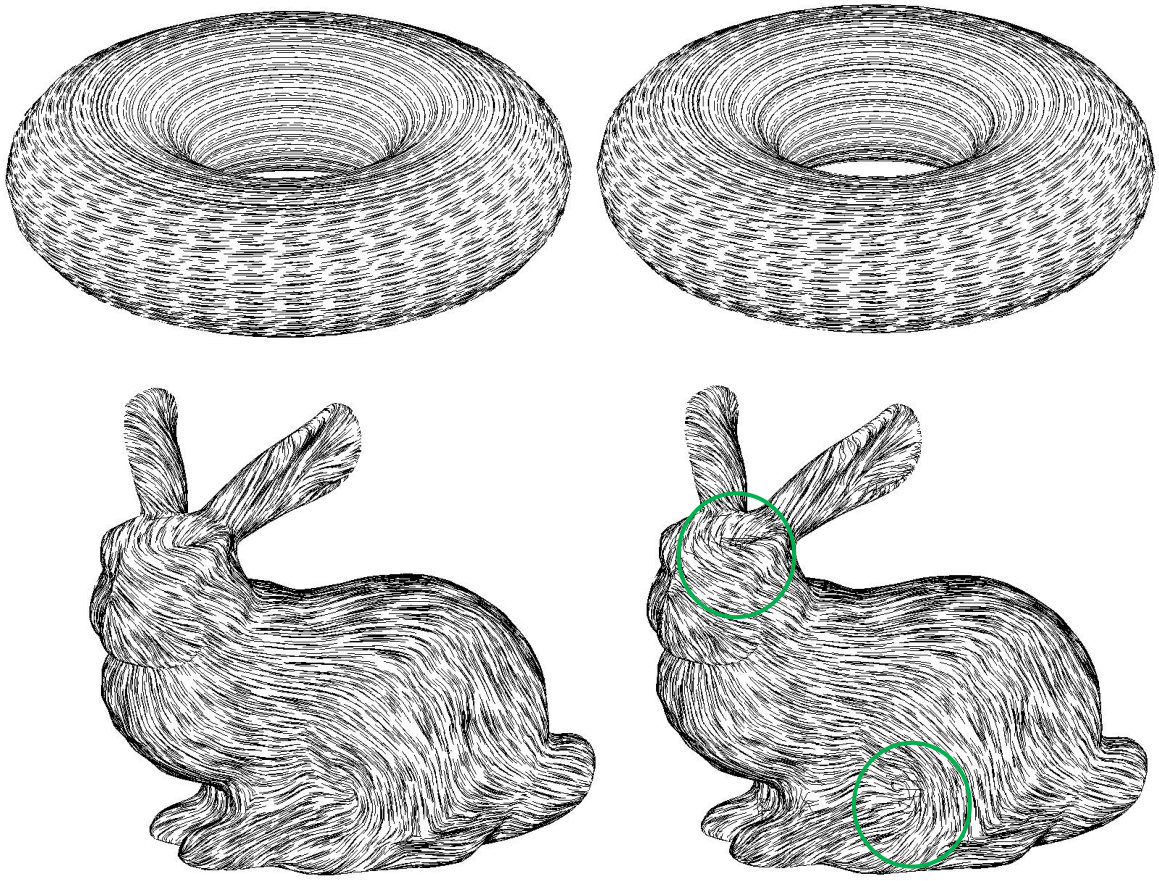


Figure 32. The Comparison details between Fold (left) and Unfold (right) methods (with the minor principle curvature directions)

From the pictures comparison, we can conclude relative strengths and weaknesses for both models:

Fold method:

Strengths: the stream line has a good fluid shape which can be seen left picture in the second row in **Figure 32**. It describes the shape of the model appropriately. This kind of visual character can be used for the simulation of animal's fury.

Weaknesses: the stream line is more sensitive to the shape. When the shape is not smooth enough, the final result will also flow the noise of the dirty shape.

Unfold method:

Strengths: the stream line highly follow the curvature direction of the initial triangles. Thus it can be a good implementation of doing surface texture coordinates setting, which would perform much better than the 2D heat function pair (F, G) coordinates in Project 2.

Weaknesses: the stream line can not do good management for the sharp curve on the surface. Compared with the fold method, the unfold method will show some mess-up lines in the region of sharp corner as the right figure in the second row in **Figure 32** shows. Also, because of the property of the Unfold method's "straightness", some of the lines will "stick" together, which can be seen by compare the pictures in the first row of **Figure 32**.

There is a common problem with both approaches. If we observe **Figure 30**, **31** and **32**, it is not hard to find that some times the principle curvature direction rotate in a wrong direction. This problem happens when the streamline turbulence shows, which can be seen in **Figure 33**.

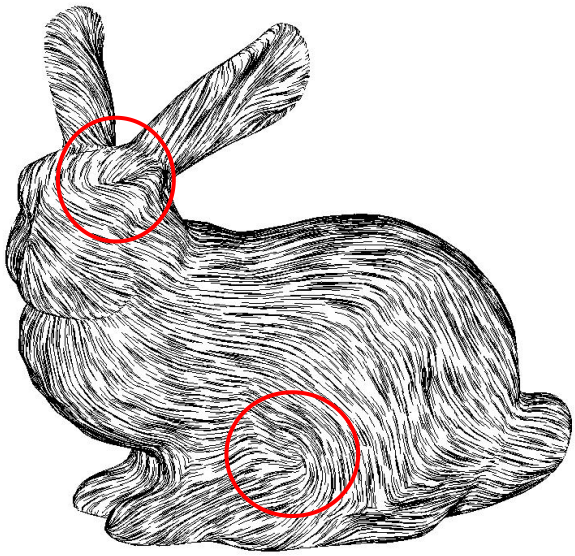


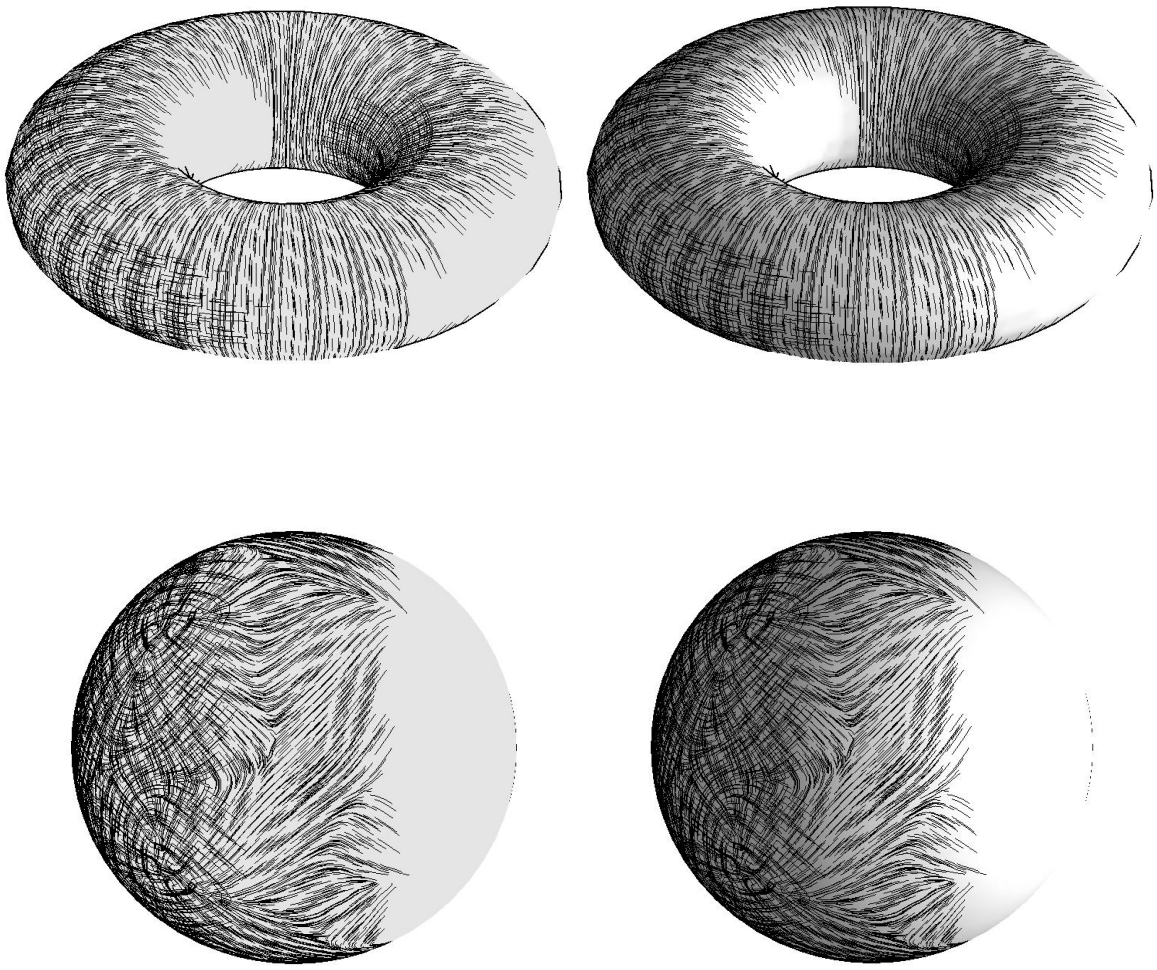
Figure 33. The Stream line turbulence

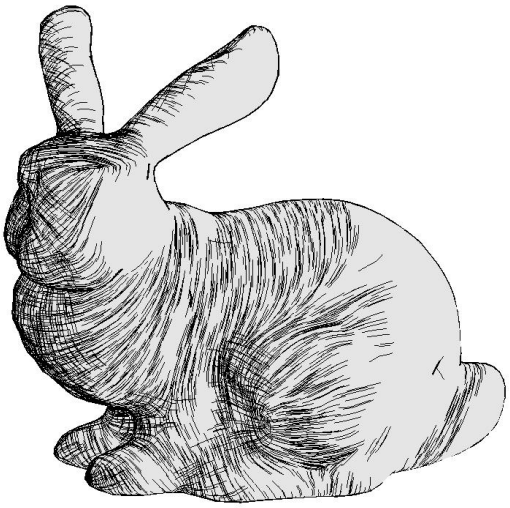
Basically, this problem comes from the tensor smoothing step. There are two solutions.

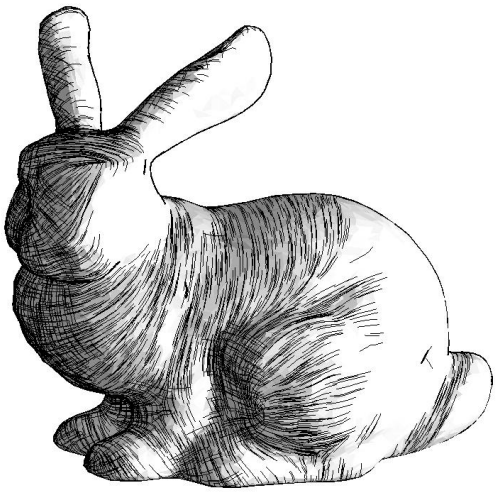
The first way is judge the eigenvector's direction with it's 1-distance neighbour's principle curvature directions. However, this method might increase the calculation time.

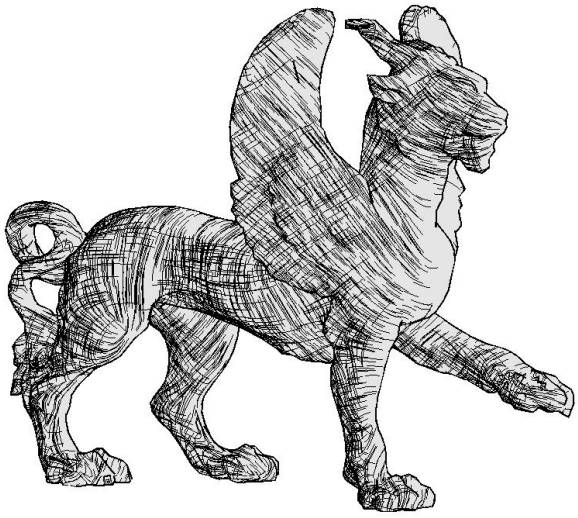
The another way to solve this problem, is trying to apply the implicit method. This method is really stable no matter talking about the numerical error or the stream turbulence elimination.

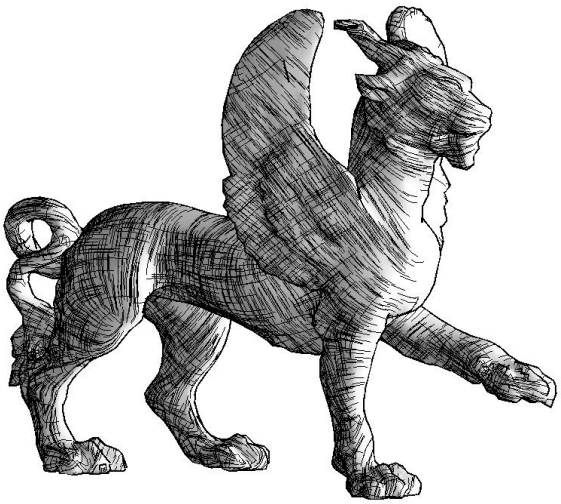
Finally, let us see the final sketch stylized result. After adding the silhouette we finished in question 1 and the streamlines in this question, I tried to set threshold by judging each triangle's dot product between its normal and the light source ray. The result in the range of [-1, 0.5] would be the shadow part, which I set showing both the minor and the major streamlines. The result in the range of (0.5, 0.2) would be the normal body part, which I set showing the major streamlines. The result in the range of [0.2, 1] would be the high light part, which I set no hatches for showing. The final result can be seen in **Figure 34**.

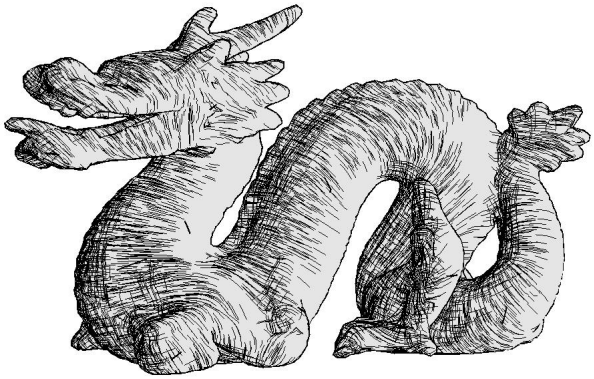












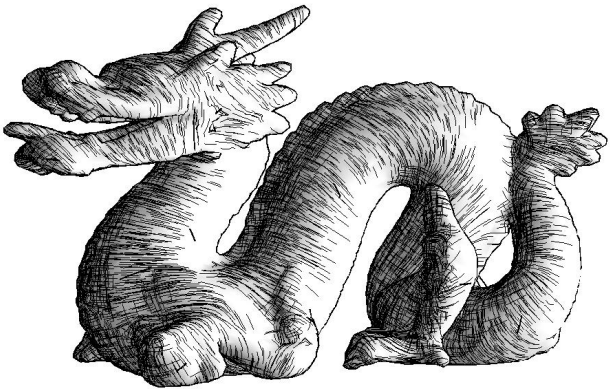


Figure 34. Final Sketch style results
(left with light off, right with light on)

By observing the final results that I get, I think there are two directions that the steam pipeline can be improved.

Firstly, if we magnify the final result, we can see lots of right angles in the shadow part of the final rendering, which can be seen in **Figure 35**.

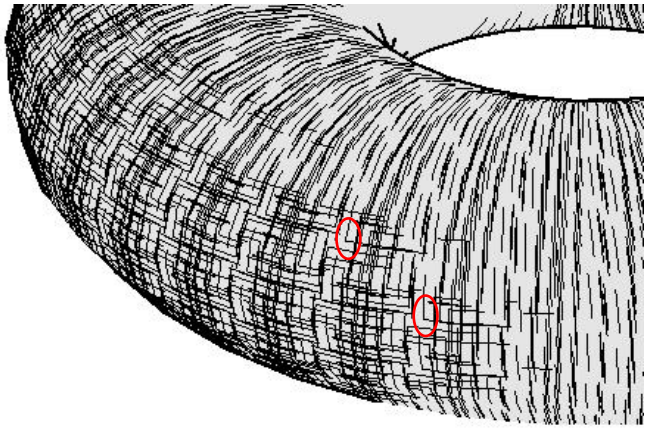


Figure 35. The problem of the right angles.

This issue comes from our setting of the start point of each stream line. We should not set one triangle to be the start point of both the minor stream line and the major stream line but set the triangle center as the center in the minor stream line and the major stream.

Secondly, from my method of doing sketching style I was focusing each triangles, which means once the triangle's dot product result do not satisfy the requirement of the threshold, the whole stream line from that triangle will be eliminated. This kind of removing will cost much texture losing as the **Figure 36** shows.

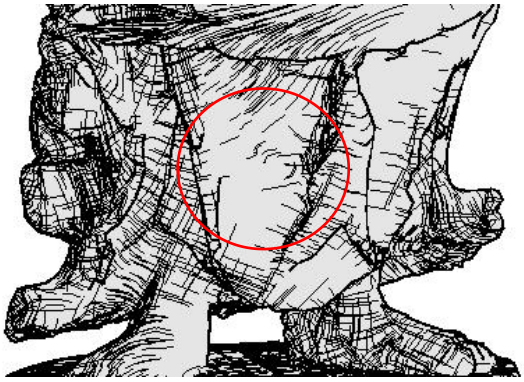


Figure 35. The Stream line losing

The way to solve this problem is trying to use shader for doing that. In the shader, we can compute the light ray for the dot product based on each pixels level. At that time, the result's textures will be more rich.

Keyboard guidance:
1=> Original shape with tensor field
2=> Regular subdivision
3=> Irregular subdivision
4=> Smooth
5=> Heat diffusion
6=> Original shape with white color
7=> Shape with 3D checkerboard color
or 3D checkerboard coordinates texture
9=> Silhouette
P=> Tensor Smoothing

s=> Swap smooth and flat
t=> Triangle mesh