Andrea D. Volpe
BxBun83@aol.com
Mentor: H. Quynh Dinh
DMP Summer Project 2005
Vector Fields on Surfaces- Final Report

I. Description

We created different vector fields on the surfaces of different polygonal meshes, based on boundary conditions, and visualized the vector fields using a texture-based method that advects a noisy texture in the direction of the vector field. The boundary conditions consist of a number of sinks and sources. Each of these is set with a certain scalar value, while all other points are interpolated based on these, to create a scalar field from which a gradient vector field is created. Vectors in the field point away from a source and into a sink.

II. Background/Motivation

There are many different uses of vector fields on surfaces. Many of these have been studied and implemented. These applications include such texture synthesis, sketching illustrations, hair position, liquid flow, and more. [Dinh 2004] shows, for example, how texture mapping uses a vector field to advect properties from one model to another. For fluid simulation, [Zhang 2004] explains that the external force is a vector field. It does not need to correspond to actual physical phenomena and it can exist on synthetic 3D surfaces. It is interesting how texture synthesis uses vector fields. When performing texture synthesis on a rectangular grid the values of the texture are filled from left to right, top to bottom. On a surface, however, the vector field gives an order and direction to follow for synthesizing textures on surfaces [Turk 2001]. Our purpose was to create a program to experiment with different vector fields and modify the fields through control parameters, including sources, sinks, etc. in order to use them for applications of vector field on surfaces.

III. Related Work

The related works are methods for creating a vector field, visualizing vector fields, and applications using vector fields. Many methods have been developed to create a vector field. [Zhang 2004] created a vector field design system for planar domains and 3D surfaces, which allowed the user to create a vector field first using basis fields, then make geometric and topological changes, and finally move singularities for the desired result. According to [Zhang 2004] most existing vector field design systems generate gradient vector fields, as did [Dinh 2004], which we will be using, or incompressible vector fields.

Many applications of vector fields have been studied. [Zhang 2004] displays several of these including painterly rendering, pencil sketch illustration of smooth surfaces, and example-based texture synthesis. [Van Wijk 2003] mentions that visualizing a vector field plays a significant role in Computational Fluid Dynamics important in weather, climate, industrial processes, cooling, heating, etc.

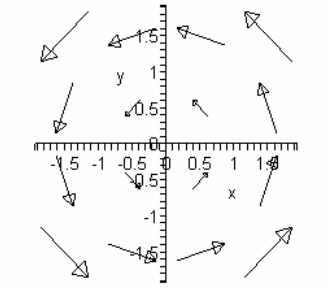
Many methods of visualization of vector fields have been developed. [Van Wijk 2002] mentions several of these. For example, arrow plots consist of arrows placed

discretely and sparsely on the surface, but it is difficult to visualize the continuous flow because of the discrete nature of the plot. Stream lines and advected particles are more informative than arrow plots but both have the problem that you cannot place particles or stream lines everywhere. You can, however, visualize the flow a bit better than arrow plots because the flow lines are continuous. Better methods for visualizing continuous vector fields are spot noise textures and line integral convolution. These methods can show a denser flow field. They are both, however, computationally expensive. The method we use, developed by [Van Wijk 2003] uses graphics hardware and methods, such as blending, to make the visualization of the flow field more efficient.

IV. Theory

Vector Field Creation

A Vector Field on two (or three) dimensional space, as defined by [Dawkins 2005] is a function $\vec{F}$ that assigns to each point (x, y) (or (x, y, z)) a two (or three) dimensional vector given by $\vec{F}(x,y)$ ($\vec{F}(x,y,z)$). An example is shown below.



Vector fields have been studied in computer graphics and applied to both two and three dimensions. One application is in the medical field, as shown by [Yezzi 2001], (*Tissue Gridding Using a PDE Approach)* in which tangent vectors are used to determine tissue thickness and therefore the clogging of blood vessels. The approach this paper takes to calculating vector fields was then used by [Dinh 2004] to transfer texture from one shape to the next in a shape transformation. This method creates a mapping between surfaces to transfer attributes from one surface to the next. To create the mapping, we begin with starting and ending contours, or three dimensional shapes. In two dimensions plus a time dimension, we begin with starting and ending contours, *a* and *b*, with start time at *a* equal to 0, end time at *b* equal to 1, and time interpolated between the two. We then create a scalar field from *a* to *b*. The gradient of this field becomes the vector field. [Dinh 2004] uses heat diffusion applied to the scalar field in order to distribute the vector field more uniformly. We apply the same concepts of finding a scalar field, diffusing it, and calculating its gradient vector field, except that we use starting and ending points instead of contours, which form our sources and sinks. Vectors in the field point away from a source and into a sink.

Vector Field Visualization

In order to display our vector field, we use [Van Wijk 2003]'s algorithm, as applied to curved surfaces. The algorithm begins by projecting a polygonal mesh to the frame buffer in the first time step. The image is then captured and stored for the next

time step. The next frame will look like the previous, except distorted in the direction of the vector flow a small distance, based on a set small time step. We are in actuality advecting the information along the vector field. To prevent the image from disappearing entirely, it is continuously blended with a noise image. To apply this to a surface, we need to distinguish between the surface and the background, using depth testing. We create a rectangle covering the entire screen with its depth set near the far clipping plane, texture mapped with the noise pattern, and only blend it with the screen if the depth of what is already on screen (the rendered model) is less than its own depth. That is, we are only blending the noise with the model drawn already, not the background. We then apply shading, in order to see the surface in three dimensions. On a curved surface, the shading, as well as keeping both the image and background in shades of grey help to create contrast and prevent blending of the background color with the foreground surface as the noise texture is advected.

V. Implementation

I started off with code from my mentor and a former student of hers. One program read in data from a file and stored it in a certain structure, in order to render the model with lighting (figure 1), with a texture, and then as a polygonal mesh. The second program read in data from a file, stored it in a different data structure, and with a set start and end point, calculated a vector for each vertex, therefore creating a vector field. I added the needed methods for creating the scalar field, the gradient vector field, and for diffusion, and associated global variables from the second file to the first, and changed any necessary data structures to match properly. This gave us a program that reads in data from a file, stores it in a structure, calculates a vector field, and renders the above mentioned figures.

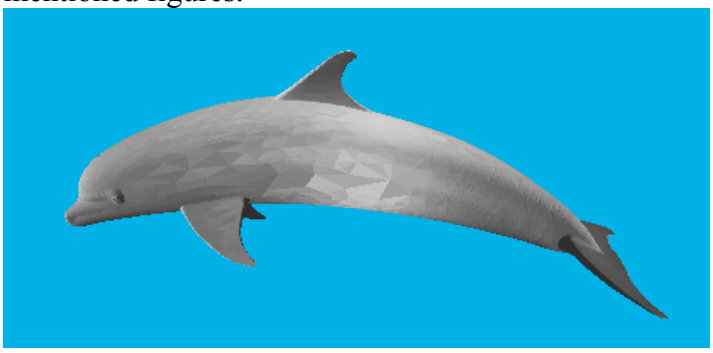


Figure 1: Original dolphin model, before vector field creation.

I then changed the code so that it would be able to use any number of start and end points to calculate the vector field. These “start” and “end” points are really sources and sinks. I calculated the scalar value at each vertex, in terms of its distance from each source and sink, and the values of the source and sink (i.e. [0, 1]). I drew each point’s vector in order to display the stationary vector field (figure 2).

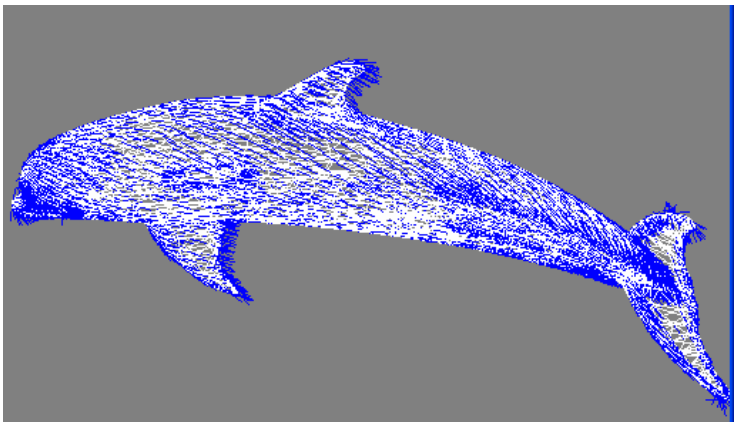


Figure 2: Polygonal mesh of dolphin model with vector field shown.

After this I applied the algorithm from [Van Wijk 2003] so that I could display the vector field as an animated noisy pattern. This had several steps:

1. Initialize the texture image F_t with a Background color, B. To do this, I cleared the color buffer to be grey and cleared the depth buffer.
2. Initialize the image F with B; That is, my entire screen is the image (which we will be treating like a texture) each particle of which I want to project along the vector field, to display its motion. So, I created a blank texture earlier, assigning it room in memory, where we will store our image.
3. Calculate the texture coordinates $t_i = (t_{ix}, t_{iy})$ for all vertices. This must be recalculated whenever the image rotates or moves in any way. We calculate these by using the equation $t_i = T(r_i - v_i\Delta t)$, where t_i represents the texture coordinates, T represents projection, r_i represents a point, v_i is the vector field at that point, and Δt is the time step for each blended image. This means that this equation is the projected position of the previous point on the path line through a vertex, r_i.
4. Render the mesh, texture mapped with F_t, without shading.
5. Blend in fresh noise, G. To do this, I used a sample function provided by [Van Wijk 2003]. It calculates the texture coordinates of a "noise" image for each time step. I then texture mapped this onto a rectangle that covered the entire screen, but whose depth was almost as far back as the viewport far clipping plane. I set the depth so that it would only render the rectangle where its depth was greater than what was on the screen, the image we want. I enabled blending based on the alpha set on the noise image. Then I captured this image and copied it to the original texture.
6. Store the result in F_t. I used an openGL command to copy the current image to the original blank texture we created.
7. Render the mesh shaded and blend it with the image. To do this, I first tried to render a shaded image and blend it with what was on the screen. This was not working well, so instead I rendered the shaded image, texture mapped with the texture I just created.

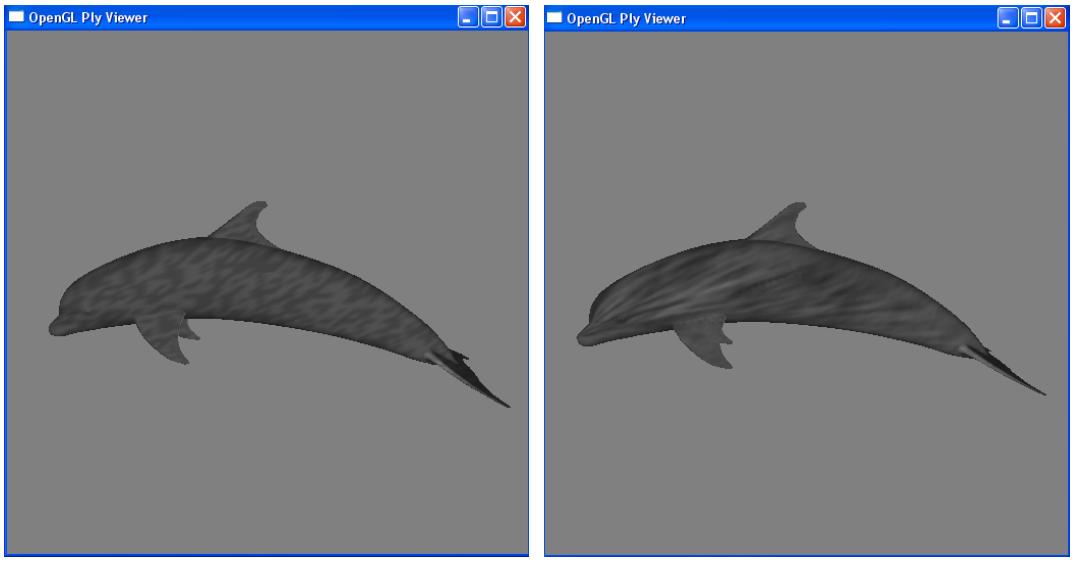


Figure 3: (Left) Dolphin model with noise texture. (Right) Dolphin model with noise advected in direction of the vector field.

VI. Experimentation and Results

1. I experimented with several different visualization effects. I changed lighting conditions by increasing ambient and diffuse light in order to make the scene brighter, despite the optimal grey colors. I experimented with changing the direction of the light, but it made little difference.

2. I then attempted to change the size and frequency of the noise pattern, which were set to 64 and 256 respectively. When the texture is as small as 2 by 2 it is just stripes in the image. At 4 a regular, blotchier patter begins. By 8 the blotches are blurrier and less regular. By the time I tested 16, it looked much the same as any size beyond that. I tried 64, 128, and 256 for the size of the noise pattern and there were only very slight differences. These results are displayed in figure 4.

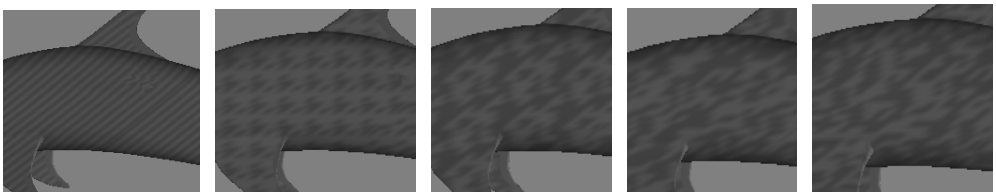


Figure 4: Result of noise texture after changing its size to: 2, 4, 8, 16, and 256.

As for the frequency, I tried powers of 2 from 6 to 10, and the higher the modulus number, the noisier the image got, as shown in figure 5.

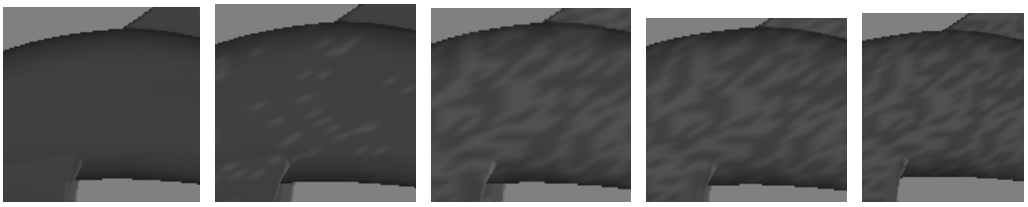


Figure 5: Result of noise texture after changing its frequency to 64, 128, 256, 512, and 1024.

3. I also tested different positions of the source and sink. In the figures below sources are marked by red circles and sinks by blue circles. I began by putting one source at the min z value, which was the right side of the tail fin, and one sink at the max z value, which was the tip of the dolphin's nose. This caused the vector field to go smoothly from the source, right side of the fin, to the sink, the nose, along the surface (figure 6).

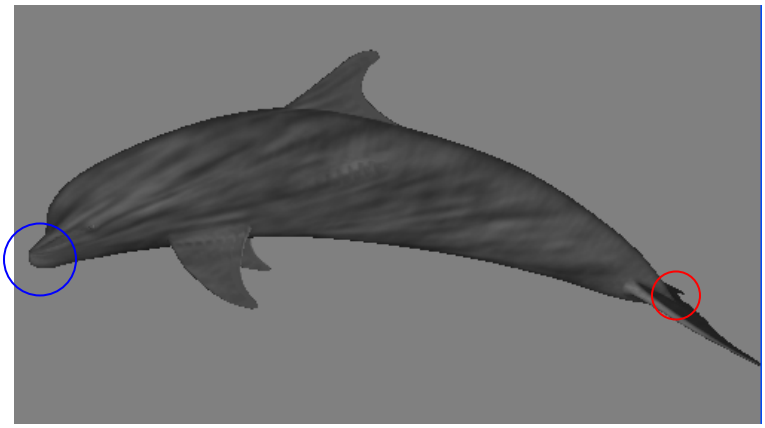


Figure 6: Noisy vector field on dolphin with source at its fin and sink at its nose.

If I put the sink in the middle of the figure, leaving the source at the tail fin, the vectors from either side flow along the surface toward the top center fin. There is no source or sink near the tip of the nose now, but due to the calculated vector field based on the source and sink, it is still pulled toward the sink (figure 7).

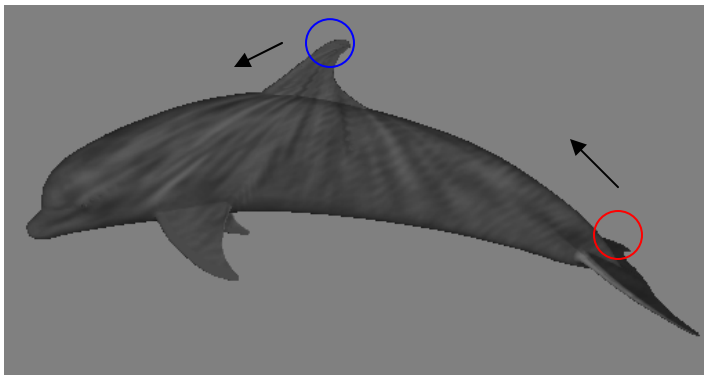


Figure 7: Noisy vector field on dolphin with source at its tail fin and sink at its top fin

If I had two sinks positioned with no source between them, the vectors seemed to be unsure which one to go towards and most ended up somewhere between the two. In figure 8 (Right), a divide seems to occur by the eye, to the left of which, the field points toward the left (where one sink is) and vectors to the right points to the other sink.

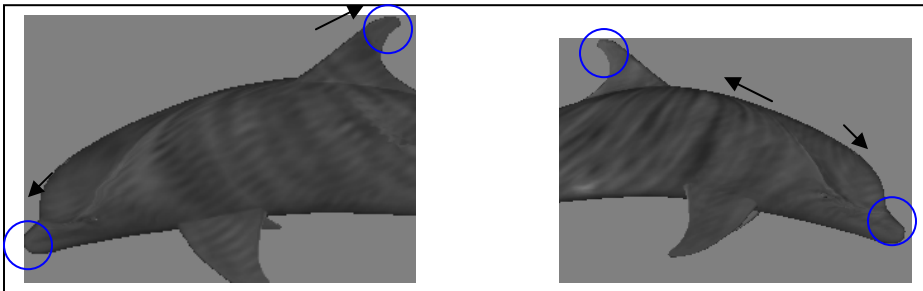


Figure 8: Example of vector field between two sinks

4. Number of sources and sinks

I tested the effects of several sources or sinks. A source from one point on the fin, as opposed to both is shown in figure 9. If the sources are both tips of the tail fin and the tip of the top fin, the flow on the back fin changed slightly, but the general flow from sources to sink remained the same.

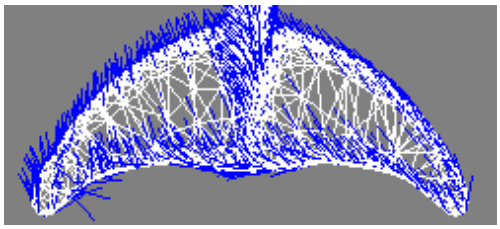
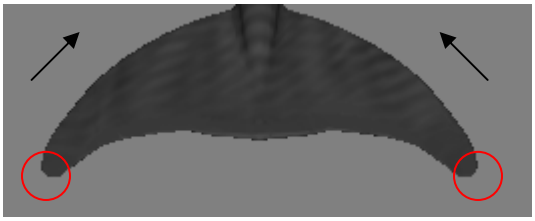
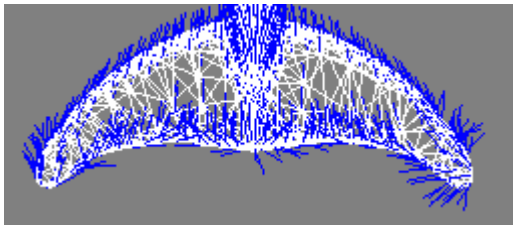


Figure 9: Tail fin of the dolphin displayed with one source at right tip of tail fin (Top) and with two sources at both tips of tail fins (Bottom)





If I put one source at the tail fin, and two sinks, the nose and top fin, it drastically changed the direction of the vector field, because not all vectors point toward the nose anymore.

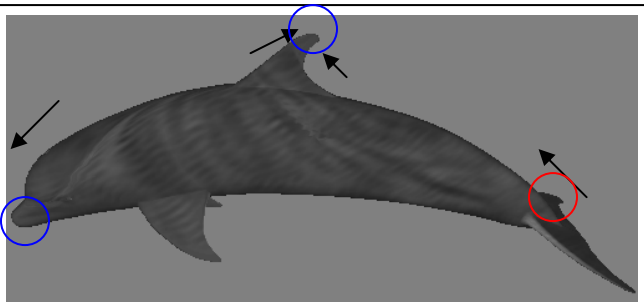


Figure 10: Dolphin model with one source at tail fin, one sink at top fin, and one sink at nose

5. I also experimented with changing the values of sources and sinks. Sources do not always have to be set with a scalar value of 0. Given three sources- each tip of the tail fin and the tip of the top fin- and one sink- the nose- I changed the value of one of the sources to .5. There is only a slight difference from when the sources were all 0 and sink was 1. Just near the top fin, the source that I set to .5, the vector field seems pulled slightly more upward.

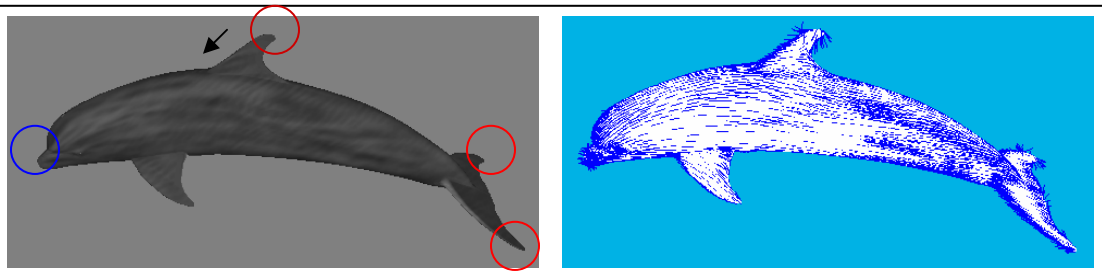


Figure 11: Dolphin model with value of source at top fin changed from 0 to .5

When the top fin and nose are sinks and the tail fin's tips are sources, with the top fin equal to .5, the results are similar.

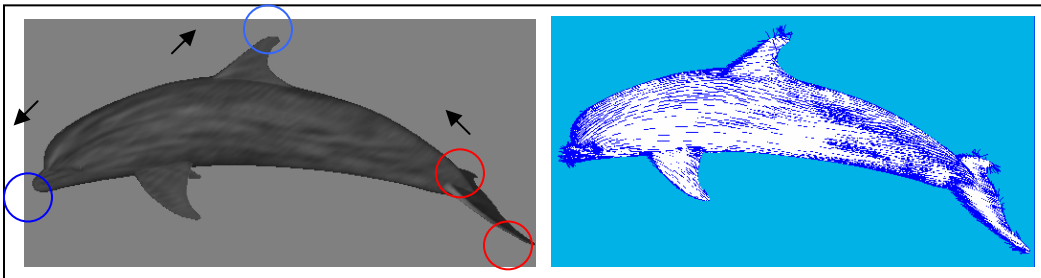


Figure 12: Dolphin model with value of sink at top of fin changed from 1 to .5

If the value of the sink at the nose is .5 and the top fin is 1, this causes a stronger pull toward the top fin and the pull from the nose to be practically non existent as shown in figure 13.

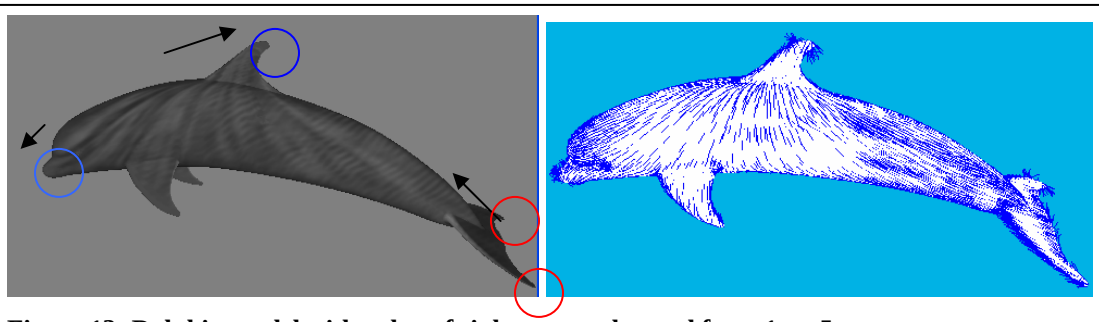


Figure 13: Dolphin model with value of sink at nose changed from 1 to .5

6. I began most of my work with the simpler dolphin model, but I tested the vector fields of several others as well. The sphere showed some interesting results (figure 14). The sink, when at the Z max, shown on the left, looked like it had a circular barrier in its center. When the min and max Y were used, instead of a singular sink, it seemed to break into two.

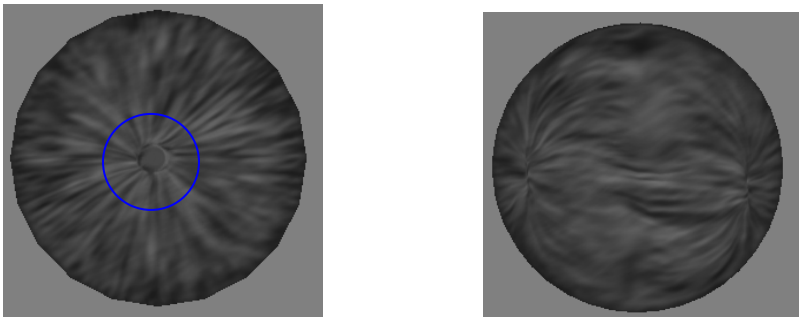


Figure 14: Sphere model with source and sink at min and max Z respectively (Left) and source and sink at min and max Y respectively (Right)

When using the min and max Z as the source and sink for the donut shape, the vector field seems to be smooth. However, when using X or Y min and max for the donut shape, a result similar to the X or Y used for the sphere occurred, as shown in the figure 15 (Right).

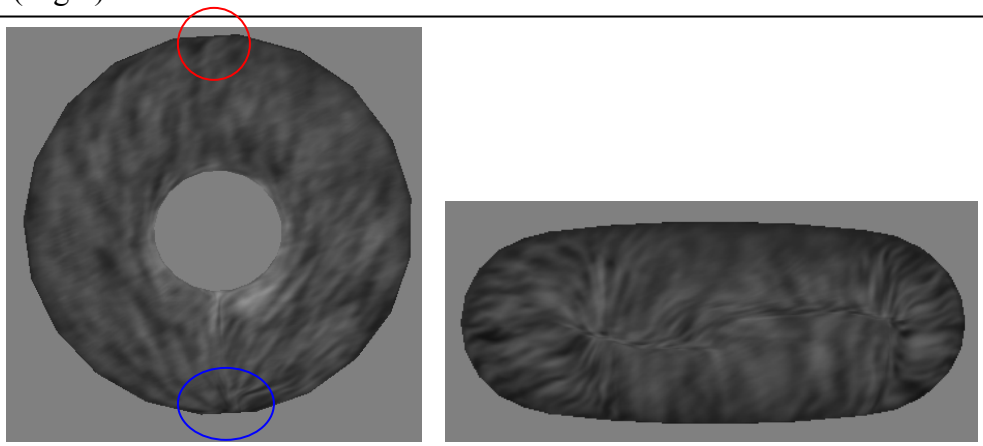


Figure 15: Donut model with min and max Z as source and sink (Left) and min and max X as source and sink (Right)

The elephant, with min and max X values as the source and sink, showed some interesting results around its ear. In figure 16 (Right) along the ear the field seems to be coming together as if there were a sink there, but then it pulls upward and around to the other side of the head where the sink truly is (Left.)

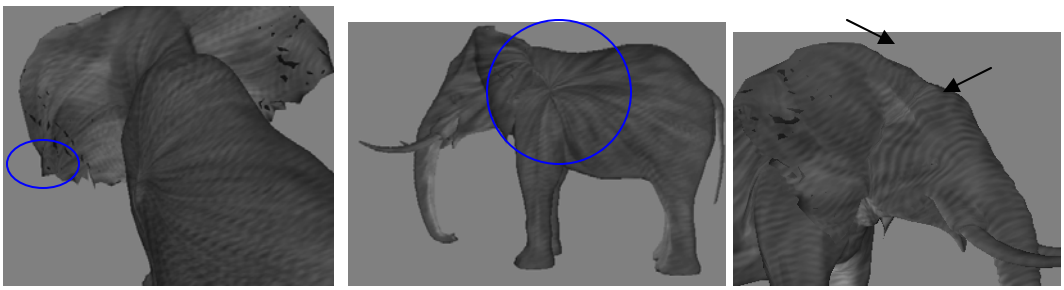


Figure 16: Elephant model with min and max X values as source and sink

Other, more complex models, like the seahorse seemed to flow relatively smoothly from source to sink. The vector field seems well distributed.

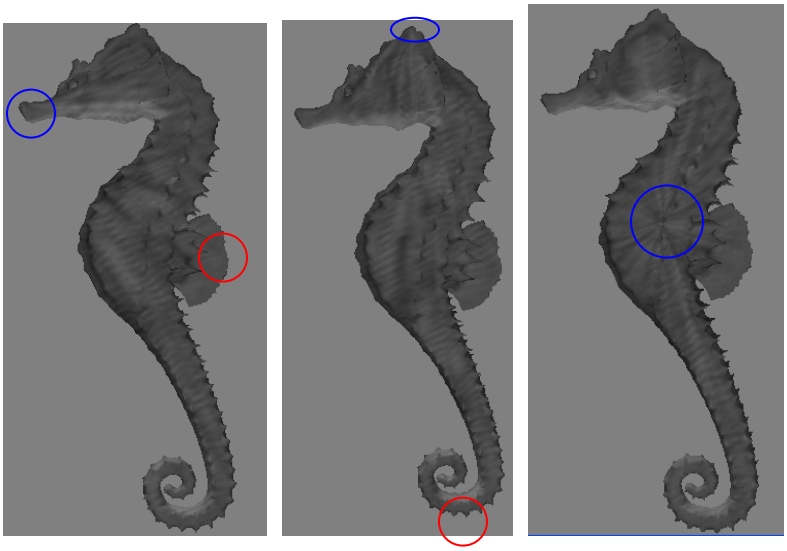


Figure 17: Seahorse model with min and max Z (Left), Y (Middle), and X (Right) as source and sink

The results shown by each of these experiments seem to show well distributed vector fields that work well on certain models, such as the dolphin. Certain oddities occurred, such as in certain spots of the sphere and donut shapes, or around the ears of the elephant. In future studies we might like to detect why these occur and what affects they might have on applications of the vector field on surfaces.

References:

Dawkins, Paul. *Paul's Online Math Tutorials and Notes*. © 2003-2005. http://tutorial.math.lamar.edu/.

Dinh, H. Q., Yezzi, A., and Turk, G. 2005. "Texture Transfer During Shape Transformation." *ACM Transactions on Graphics (TOG), Volume 24 Issue 2,* pp.289 – 310. http://www.cs.stevens.edu/~quynh/papers/mapping.pdf.

Hilaga, M., Shinagawa, Y., Kohmura, T., and Kunii, T. 2001. "Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes." *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 01),* pp.203 - 212. http://delivery.acm.org/10.1145/390000/383282/p203-hilaga.pdf?key1=383282&key2=4712989111&coll=GUIDE&dl=GUIDE&CFID=46896013&CFTOKEN=54155080.

Kempf, R. and Frazier, C. Ed. *OpenGL Reference Manuel Second Edition*. (c)1997 Silicon Graphics, Inc.

Turk, Greg. 2001. "Texture Synthesis on Surfaces." *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 01),* pp.347 – 354. http://www.cc.gatech.edu/~turk/my_papers/texture.pdf.

Van Wijk, J. J. 2002. "Image Based Flow Visualization." *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 02),* pp.745 – 754. http://www.win.tue.nl/~vanwijk/ibfv/ibfv.pdf.

Van Wijk, J. J. 2003. "Image Based Flow Visualization for Curved Surfaces." *IEEE Visualization 2003*, pp.123-130. http://ieeexplore.ieee.org/iel5/8844/27978/01250363.pdf?tp=&arnumber=1250363&isnumber=27978

Yezzi, A and Prince, J.L. 2001. "A PDE Approach for Measuring Tissue Thickness." *Computer Vision and Pattern Recognition, 2001. CVPR 2001*, Vol.1, 2001, pp.87 – 92.

Zhang, E., Mischaikow, K., and Turk, G. "Vector Field Design on Surfaces." *(Technical Report In Preparation) Georgia Institute of Technology.* ftp://ftp.cc.gatech.edu/pub/gvu/tr/2004/04-16.pdf.