DOI: 10.1111/j.1467-8659.2010.01817.x

Pacific Graphics 2010
P. Alliez, K. Bala, and K. Zhou
(Guest Editors)

Volume 29 (2010), Number 7

Polygonal Surface Advection applied to Strange Attractors

S. Yan[1], N. Max [1], and K.-L. Ma[1]

[1]University of California, Davis

Abstract

Strange attractors of 3D vector field flows sometimes have a fractal geometric structure in one dimension, and smooth surface behavior in the other two. General flow visualization methods show the flow dynamics well, but not the fractal structure. Here we approximate the attractor by polygonal surfaces, which reveal the fractal geometry. We start with a polygonal approximation which neglects the fractal dimension, and then deform it by the flow to create multiple sheets of the fractal structure. We use adaptive subdivision, mesh decimation, and retiling methods to preserve the quality of the polygonal surface in the face of extreme stretching, bending, and creasing caused by the flow. A GPU implementation provides efficient visualization, which we also apply to other turbulent flows.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

Advected streak or time surfaces have long been used to visualize flow. However, they must be adaptively subdivided, simplified, and/or retiled if they are to faithfully track turbulent motion. We have developed methods to do this, which we apply to strange attractors. Since the motion towards these attractors exhibits strong stretching, bending, and folding, these examples serve as good test cases for our adaptive surface advection algorithms. In addition, we show visualization of turbulent flows found in real-world applications.

An n-dimensional vector field V defines a flow $F(t,x)$ by the differential equation

$$\frac{dF(t,x)}{dt} = V(t) \qquad (1)$$

with initial conditions $F(0,x) = x$. This is a system of n first order ordinary differential equations (ODEs) in n variables.

For such a system of ODEs, a set A is called an attractor for an open set U if a) for every x in U, the distance from $F(t,x)$ to A approaches 0 as t approaches infinity, b) if x is in A, then $F(t,x)$ is in A for all t, and c) A has no smaller subset with these properties. Thus, a sink point for a linear vector field is an attractor. If there is sensitive dependence on initial conditions on A, so that trajectories starting at arbitrarily close initial conditions eventually diverge, we get a so called strange attractor, with a fractal structure.

In 1932, while trying to model convection in the atmosphere, Edward Lorenz [Lor63] discovered such a 3D vector field with sensitive dependence on initial conditions. It has an attractor of two spiral lobes, as shown in figures 1 and 2, with a trajectory jumping unpredictably from one to the other. Guckunheimer and Williams [GW79] analyzed the geometric nature of a simplified version of this attractor in terms of two and one dimensional ways of abstracting the flow, and Tucker [Tuc02] used interval arithmetic to show that the original Lorentz attractor in equation 2 has the same properties as the simplified version. Paiva *et al*. [PdFS06] visualize 2D strange attractors using affine arithmetic, which gives tighter bounds than interval arithmetic. Rössler [Rös76] discovered a simpler system of three ODEs, whose attractor has only one spiral lobe. In both of these systems, the flow spreads or folds the attractor A over on top of itself infinitely many times, like phyllo or croissant dough, so that the attractor has a fractal Cantor set structure in one dimension, and a smooth surface structure in the other two. A good visual explanation of this phenomenon is given in Abraham and Shaw [AS83], a more technical but still readable explanation is given in Strogatz [Str94], and a more thorough discussion of methods for analyzing and categorizing attractors is given in Gilmore and Lefranc [GL02].

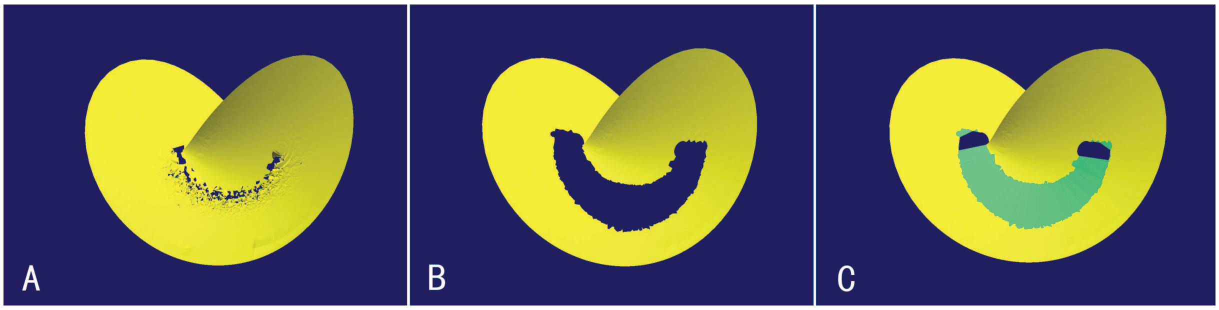

Figure 1: *A:The Lorenz attractor, as triangulated by Cocone. B:The Lorenz attractor, from points triangulated by Cocone, with an annular region removed. C:Lorenz attractor of Figure 1B with added triangles in green.*

We also consider two other attractors from the Van der Pol and Duffing equations, describing forced oscillators.

Most visualization systems for these strange attractors track a collection of isolated points as they move in the flow, and visualize either their positions or their streamlines. A notable exception is the work of Saupe and Tvedt [ST94], which uses volume rendering to visualize the invariant measure determined by the Lorenz attractor, by tracking the positions of a collection of points, and using their local density to estimate the invariant measure. A related approach is taken by Dellnitz *et al*. [DHJR97], who approximate the attractor by a k-D tree hierarchy of boxes, and use Monte-Carlo integration and an eigenvector solution to determine an invariant measure that is constant on each box. Krauskopf *et al*. [KOD*05] survey multiple numerical methods for finding the 2D stable and unstable manifolds of critical points, illustrating them on the stable manifold of the critical point at the origin in the Lorenz flow. Here we use polygonal surface rendering to visualize the fractal behavior of the attractor itself. This presents special challenges, since the flows involved stretch and fold the surface into multiple layers which are very close to each other. We solve these challenges using adaptive subdivision, retiling, and a new method for creating aligned triangulations on nearby layers.

2. The Lorenz and Rössler Attractors

The flow for the original 1963 Lorenz attractor is given by the equations

$$\begin{aligned}\frac{dx}{dt} &= 10y-10x\\ \frac{dy}{dt} &= 28x-xz-y\\ \frac{dz}{dt} &= xy-\frac{8}{3}z\end{aligned}\qquad(2)$$

with two non-linear terms, xz and xy, and for the Rössler attractor, the equations are

$$\begin{aligned}\frac{dx}{dt} &= -y-z\\ \frac{dy}{dt} &= x+0.2y\\ \frac{dz}{dt} &= 0.2+xz-5.7z\end{aligned}\qquad(3)$$

with only one non-linear term xz.

Our initial approach was to model the "template" or "branched manifold" (see [GL02] [BW83]) of these two attractors directly, from a collection of tracked points. We did this by taking a large collection of random points in a cube enclosing the attractor, and moving them by the flow for many time steps, using 4^{th} order Runge-Kutta integration (Press *et al*. [PTVF92]), until they approach the attractor. For the Lorenz attractor shown in Figure 1, we added points along the unstable manifold of the critical point at the origin, to get a smooth outer edge. Next we applied the Cocone program of Dey *et al*. [DG01] to create an initial triangulation. The triangles were incorrect where surface sheets approached each other, because triangles were formed joining different sheets of the surface, as shown in Figure 1A. Therefore we interactively chose an annular sector of triangles to remove from the surface, as in Figure 1B, leaving gaps where two surfaces exist on one side, and merge to become one on the other side. The gaps were partially closed by identifying two arcs on the boundary where the removal cut the surface, and joining them by a collection of long thin triangles, shown green in Figure 1C.

The shape in Figure 1C, even if colored by a uniform color, does not give a good visualization of the fractal nature of the surface, because the multiple adjacent sheets are not shown. If this polygonal surfaces is used as the initial condition for further flow, and adaptively subdivided and retiled as the flow progresses, as described in section 4, multiple layers can be formed. Lorenz [Lor63] has shown that the distance between layers decreases by a factor of .00007 each time the flow circles around one of the two holes in Figure 1C, so there is no hope of visualizing the fractal structure directly.

However, if we artificially separate the layers, as shown in Figure 2, we can see how they spiral around the holes.

The Rössler attractor folds its surface repeatedly on top of itself, but the folded part does not completely cover the original surface. This leads to a banded invariant measure of the density of the tracked random dots shown in Figure 3A, and of the fractal. Polygonal surface rendering can help visualize this density. Figure 3B shows the an opaque triangulated surface after multiple further advection steps in the flow. Figure 3C shows the same surface, rendered with partially transparent polygons. The colors from the regions overlapped by different numbers of semi-transparent layers reveal and explain the density variation of the dots in Figure 3A.

3. The Duffing and Van der Pol attractors

The Duffing attractor describes a periodically forced particle moving in a double well potential. We used the differential equation from [Kan]:

$$\frac{d^2x}{dt^2} + 0.2\frac{dx}{dt} - x + x^3 = 0.3cos(t). \tag{4}$$

If we make $y = \frac{dx}{dt}$ a new variable, we get a system of two first order differential equations

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= -0.2y + x - x^3 + 0.3cos(t). \end{aligned} \tag{5}$$

Then to get a 3D attractor, we let t be the third variable. Since the right hand sides of these equations are periodic with period 2π in the phase t, we can let t vary around a circle of radius r (we used $r = 1.7$) in the (u, v) plane in 3D (u, v, w) space, and wrap a part of the 2D plane orthogonal to this circle around it, to get a solid torus containing the attractor. The map from (x, y, t) to (u, v, w) is given by

$$\begin{aligned} u &= (r+y)\ cos(t) \\ v &= (r+y)\ sin(t) \\ w &= x \end{aligned} \tag{6}$$

We can use a closed torus as an initial smooth surface, which, if placed appropriately, will approach the attractor. Figure 12D shows early stages of this motion.

The Van der Pol equations are,

$$\begin{aligned} \frac{dx}{dt} &= y + 0.245sin(t) \\ \frac{dy}{dt} &= 9(0.11 - x^2)y - 0.72x \end{aligned} \tag{7}$$

and can be similarly represented and visualized in 3D.

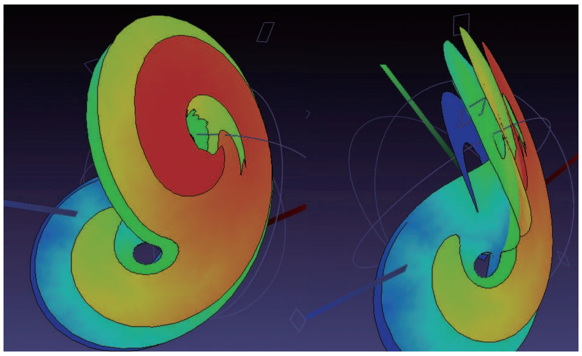
Figure 2: *Artificially separated layers of the Lorenz attractor.*

4. Polygonal surface advection

4.1. Related work

Dynamic re-meshing, intensively studied in many areas, generally falls into two categories, parameterization-based and mesh adaption. While a parameterization based method maps the original mesh to a parameter space and create a new mesh by re-sampling, the mesh adaption method directly adjusts the original topology with edge operations and vertex positioning. For example Surazhsky *et al*. [SAG03] refine the original mesh with a desired number of primitives, using weighted centroidal Voronoi tessellation to replace vertices. Vorstaz *et al*. [VRS03] use local parameterization and relax edge connectivity to create a regular mesh. Alliez *et al*. [ACSD*03] proposed the idea of anisotropic sampling using lines of curves to better preserve features with fewer primitives.

Re-meshing based streak surfaces and time surfaces are common tools in flow field visualization. However, efficient implementation of these methods for real-time uses is a challenge due to the fact that mesh adjustment is computation intensive. To deal with this problem, Funck *et al*. [vFWTS08] have proposed the smoke surface idea. Although this method is based on geometric surfaces, re-meshing is avoided by applying transparency to the area with high distortion. While the smoke surface approach is efficient for real time applications, it fails to display the long term behavior of turbulent data sets due to the quick fade out of the smoke surface. Later, Krishnan *et al*. [KGJ09] implemented on parallel CPUs an adaptive edge subdivision method for streak surfaces. Their method uses an efficient but simple length test to subdivide or contract edges, which is less sensitive than ours, and did not give us good results on our test cases.

A good solution to re-meshing is to break geometry primitives into isolated elements. Computation on each is hence separate and suitable for a GPU implementation. Schafhitzel *et al*. [STWE07] therefore developed the idea of using point clouds to represent streak surfaces. However in that paper,

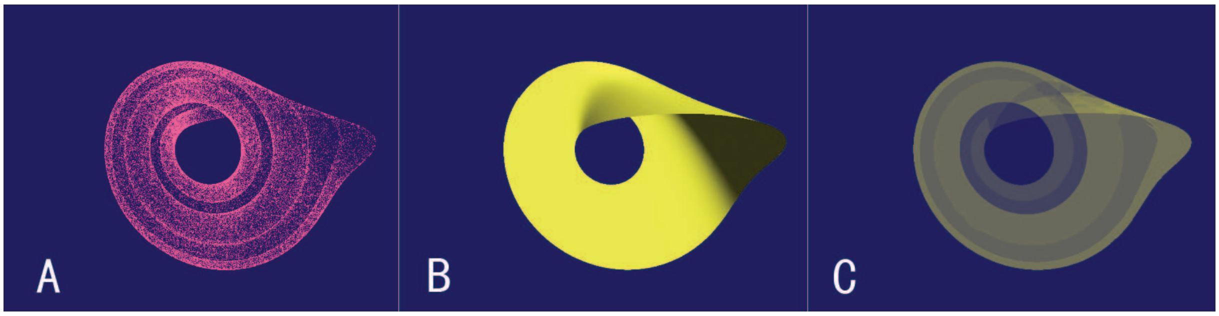

Figure 3: *The Rössler attractor, with dots, opaque surfaces, and transparent surfaces.*

a simple connectivity between time lines and between adjacent points on a same time line is still maintained. Extra points are inserted into this implicit connectivity if the local point density is not enough to cover the surface.

Buerger *et al*. [BFTW09] developed a real-time streak surface visualization method, isolating geometry primitives by using separate patches. Topology change on each patch is carried out by a parallel GPU program. Subdivided patches become new isolated primitives in the next advection round. Artifacts, such as the cracks between patches, are dealt with by using a point-based rendering method.

Mesh adaption and re-triangulation operations are hard to do efficiently on a GPU. No matter which edge picking strategy is used, the topology adjustment method itself, for example edge contraction, is a serial process due to potential conflicts when removing neighboring primitives. Therefore, we chose to implement two independent decimation methods on the CPU and only execute them infrequently. For greatest speed, standard quadric matrix simplification is used, because it is a good balance between accuracy, speed and simplicity, while for alleviating self-intersection problems, a more time consuming mesh retiling method is used.

4.2. Overview of our method

Our algorithm is a GPU subdivision based animation, with a CPU decimation method executed when needed. The input of our method is a starting mesh, which we deform according to the flow. The proposed method consists of three major steps: moving vertices according to the flow, adaptively subdividing the object to approximate the flow-deformed shape, and simplifying the object with a better triangulation when the object becomes too complex.

A key innovation is our method for creating and maintaining aligned vertices and triangulations on nearby layers of the surface during our simplification process, so that it is less likely that they intersect. Gumhold *et al.* [GBK03] have developed a related method of surface simplification that avoids intersections of nearby layers, and have applied it to simplify human models together with the clothes around them. Their method works by finding the time during a continuous geomorph for an edge collapse when an intersection first occurs, and trying to avoid this intersection by finding an alternate position for the vertex to which the edge collapses. However, the intersection situation we met is different than theirs, as in our case, the mesh is not static and intersections happen when the mesh deforms under the flow. Although the intersection free simplification prevents intersection during the simplification procedure, it does not prevent intersection as the animation proceeds. Applying intersection check for every frame is expensive for an interactive animation. On the other hand, our method produces surfaces with matched triangulations on nearby layers, which are then less likely to intersect again later as they are deformed by the flow, so it does not need to be applied very often. It is tailored for the applications to attractors and to other surfaces advected by turbulent flows, rather than to general triangle mesh simplification.

During each animation step, we update the vertex positions by the standard 4^{th} order Runge-Kutta integration (Press *et al.* [PTVF92]). To evaluate the curvature of each edge, we also update the position of the center point of each edge. When the vertices of the seeding object move by the flow field, the object is stretched and twisted. The original vertices of the object may not be dense enough to represent the twisted shape. Therefore we use a subdivision method to increases the number of points in areas with small details.

The subdivision process used in this paper occurs on each edge. We use thresholds to check the length and bending degree of an edge, to see if an edge split is necessary. Normally, a re-meshing method checks only the length of the stretched edges. An edge split is made if the length surpasses a certain threshold. We discovered that checking only the edge length causes discontinuities and jagged creases if the starting triangulation is not fine enough. But checking only the bending degree as the edge splitting criterion tends to create long thin triangles. Therefore in this paper, we use both the edge bending degree and edge length to determine if an edge requires a splitting.

To check the edge bending degree, we examine the per-

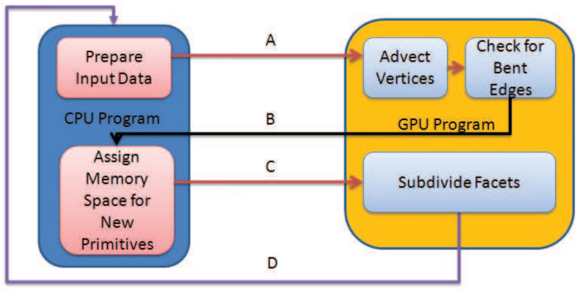

Figure 4: *The workflow of the subdivision method. A: input data including vertices, edges, and edge midpoints. B: Advected vertices, midpoints and edge bending information. C: new memory space for subdivided edges. D: Subdivided facets.*

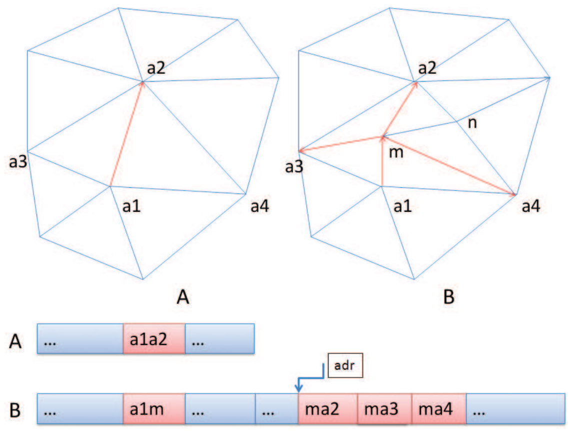

Figure 5: *The storage change of a split edge. A is before subdivision, B is after subdivision. The original edge a_1a_2 has generated four new edges a_1m, ma_2, ma_3 and ma_4. The bottom table shows the storage of the edge a_1a_2 in the memory before and after the subdivision.*

pendicular distance h of the tracked midpoint from the straight edge. If the bending exceeds our threshold, we then make a cut on the edge. This degree of bending b is given by the equation

$$b = \frac{h}{l}\sqrt{l} = \frac{h}{\sqrt{l}}, \tag{8}$$

where l is the length of the edge. Since, we want the subdivision to happen less frequently on short edges, we added an extra factor of the square root of the edge length to h/l, to decrease the bending measure as the edge becomes shorter. This is useful to prevent overly detailed subdivision, especially when the triangles have become smaller than a pixel. Once an edge is split, triangles are then subdivided according to the number of the split edges.

4.3. The GPU implementation

The GPU has proved very suitable for subdivision algorithms. DirectX 11 even supports tessellation on a hardware level with the new Hull shader and the Domain shader. However, the output of these shaders is a highly detailed model, to be used with displacement maps for better rendering results. This does not require the triangle connectivity that is important in our method. Therefore, we used CUDA instead for the implementation of the adaptive subdivision, and have achieved it at interactive speeds.

The workflow of our subdivision method has three major computational steps done in the GPU, and two simpler clerical steps done in the CPU, as shown in Figure 4. The inputs of the algorithm are vertices, edges, edge midpoints and facets. The GPU program first advects the position of vertices and midpoints and identifies bent edges. Then the CPU program assigns new memory spaces for new primitives that will be created during the subdivision step. Finally the GPU creates new primitives for subdivided facets and edges. This section covers the data structure and the implementation details.

Some data structures for representing a triangle mesh, for example the half-edge or winged-edge structures, are difficult to implement efficiently on a GPU due to the complex connectivity they maintain. Our data structure maintains a much simpler connectivity, using three types of primitives. A point is a 3-tuple of the position coordinates. An edge is a 2-tuple of the indices of its end points. A triangle is a 3-tuple of the signed indices (defined below) of its edges. Four arrays are used to store these primitives: two point arrays for the vertices and for the edge midpoints, one edge array, and one triangle array. We define the direction of an edge a_1a_2 by the vector of $a_2 - a_1$. The three edge indices of a triangle are in counter clockwise order. An edge is shared between two adjacent facets. The direction of the edge within each facet is important for subdivision and rendering. We add a sign to the edge's index to determine the edge's direction.

Before the algorithm begins, the program creates the four arrays for the initial object and ships them to the GPU global memory (Figure 4, A). Vertices as well as edge midpoints are first advected in parallel by the flow equations. The result of the first step is kept in the GPU memory and another GPU program reads it to check the bending degree and length of each edge. An output memory space is allocated by the CPU program in advance for this step that holds the new midpoint position of each split edge. If splitting does not occur, the corresponding memory position will be filled with a specific value that is easy to distinguish, for example, a position that is known to be outside the flow domain. This result is then read back by the CPU program (Figure 4, B) to assign memory for new primitives due to subdivision, including new vertices, edges and triangles. The reason that we implement this step in the CPU is that creating a continuous and dense memory space is a serial operation that is difficult and inefficient on a GPU. Although some bandwidth is sacrificed, most of this data, for example the new vertex positions, needs to be read back eventually for display and mesh decimation.

For each bent edge, we need to assign new memory space for the storage of the newly created middle points and sub-edges. The CPU program which travels through the returned data assigns memory that can hold four new edges for each subdivided edge. (Besides, the two sub-edges, two others are needed when the two adjacent facets need to be subdivided to create new triangles later. New memory space is right after the end of the old edge memory array.) Suppose the tracked edge midpoint is m. As shown in Figure 5, if the edge a_1a_2 has been split, the four new edges caused by this edge cut are a_1m, ma_2, ma_3 and ma_4. We reuse the address of edge a_1a_2 to store the sub-edge a_1m. Three new addresses are allocated to store the other three, starting at adr. The sub-edge ma_2 is always at adr. And $adr+1$ stores the new edge ma_3 at the left side of the original edge while $adr+2$ stores the new edge ma_4 at the right side of the original edge. Although the storage space for the new edges ma_3 and ma_4 is allocated, the space is empty and will be filled during the next stage. The storage for edge mn will be allocated when edge a_2a_4 is subdivided. Similarly, for each subdivided facet, a piece of memory is also assigned for the storage of new sub-facets. The newly created memory space is then read back to a GPU program again (Figure 4, C). The last GPU program subdivides each triangle by creating new edges and facets and writes to the corresponding memory address. The output after this step is a new set of the four arrays. This information is used for display and decimation and becomes the input of the next round (Figure 4, D).

4.4. Mesh decimation

Most polygon streak surface approaches, for example that of Buerger *et al*. [BFTW09], only split edges when they get too long. Methods with this kind of subdivision often do not report the need for mesh decimation, as the geometry grows at a relative slow rate. However, due to jagged creases and other artifacts caused by these methods, we use edge bending degree as a supplementary criterion. But as a consequence the geometry data increase very rapidly. Much of the geometry is unnecessarily detailed, especially, for example, when a relatively flat surface deforms parallel to itself. This is a noticeable issue in the Lorenz data set. Therefore, as a part of the algorithm, we apply mesh decimation on the geometry data when it has become too large.

A good mesh decimation method should decrease the geometry amount while preserving the object shape, and should improve the triangulation. For efficiency, direct modification of mesh topology is a good approach.

The quadric matrix decimation method by Garland *et al*. [GH97] is a popular mesh simplification method. It creates a good triangulation of the initial model with a specified number of geometry primitives. However, a parallel implementation of such a method is difficult due to the fact that it requires a global sort on the errors of removing primitives and parallel edge contraction has potential conflicts contracting neighboring edges. Many researchers have proposed out-of-core implementations of simplification by breaking down the model into portions and carrying out the simplification in a local area (DeCoro *et al*. [DT07]). This idea sacrifices image quality for speed and tends to create non-manifolds more easily. Therefore, we chose to implement the decimation on the CPU. Although quadric matrix decimation takes a few seconds in our tests, it doesn't need to be called very often, and thus doesn't compromise the performance very much. For details about this method, please refer to Garland *et al*. [GH97]. We applied it to the Lorentz attractor, with results shown in Figure 12A.

4.5. The polygon retiling method with attraction forces

One issue we face while visualizing the strange attractors with polygons is that the fractal structure tends to have intersecting sheets due to the small distance and inconsistent triangulations of two nearby layers. If only the advection and a conventional decimation method is applied, self-intersection artifacts can be seen at a very early stage. To alleviate this problem, we want to create consistent triangulations at a stage when multiple fractal sheets are formed but the self-intersection has not yet occurred.

Therefore, as an alternative decimation method, we also implement a polygon retiling method similar to Turk's [Tur92]. The major difference is that in our method, points on nearby sheets may have attraction forces that maintain a consistent triangulation of these sheets.

The workflow of our polygon retiling method contains three major steps. The first step randomly samples a set of new points inside triangles of the initial mesh based on probability, such that more points are sampled in areas with high curvature and in triangles of larger size, as shown in Figure 6A. Each sample point has repelling forces from every other sample point and may have attraction forces also if close points exist on nearby sheets. The strengths as well as the repelling radii of the forces are different according to the curvature of the triangle containing the point. For high curvature triangles, points have a smaller repelling force, so more points will flock there after repulsion. The second step moves the points on the surface mesh according to the forces. We fold the motion path of each point to force it to fall onto the mesh surface. This motion procedure repeats several times to create a physically stable state where the distances between the new points are almost equal (Figure 6B). Once they are evenly distributed, we project each point to a local least square surface fit to the neighboring original vertices. Surface re-triangulation with only the new points as vertices

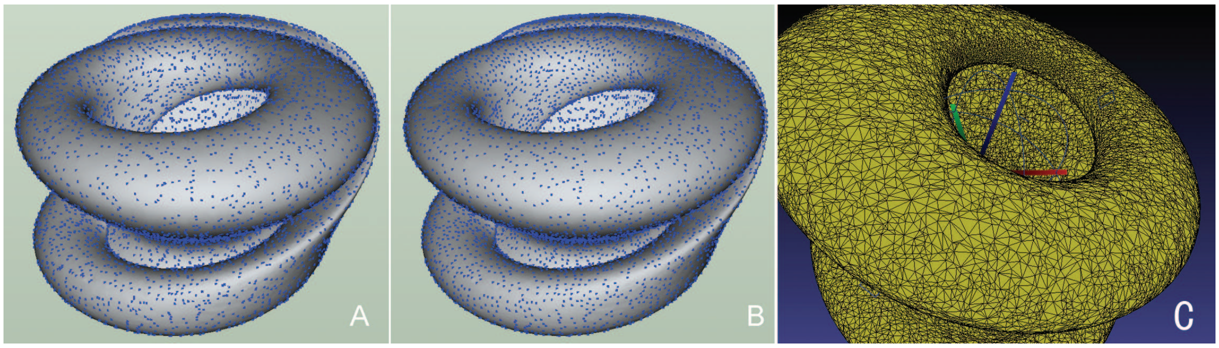

Figure 6: *A: Random points sampled on the Duffing data set based on curvature and triangle size (More points are sampled on the creases). B: Evenly distributed points after repulsion. C: Retiling result with the new points.*

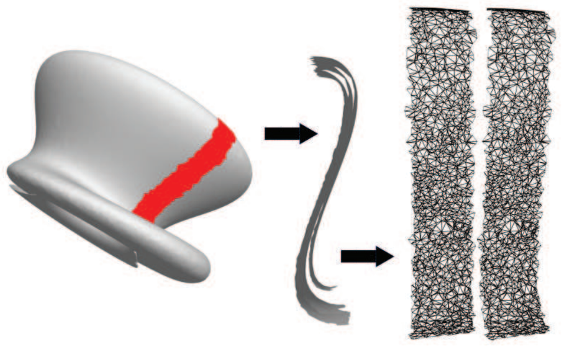

Figure 7: *The attraction forces help to preserve nearby sheets of the Duffing attractor.*

and edge flipping after that will produce a better triangulation (Figure 6C). Below, we will mostly focus on attraction forces. For details about the retiling method, please refer to Turk [Tur92].

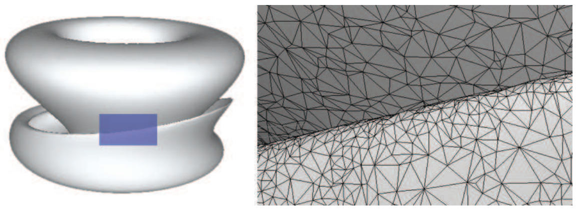

Figure 8: *A crease area after retiling.*

Sharp creases tend to generate dents after retiling. We therefore increase sampling on crease areas. And if the crease becomes too sharp, as in the Rössler data set, we start to consider the crease as a sharp feature. The sharp edge will push nearby points away to create a clear and simple mesh topology. Also, the re-triangulation method will not remove any point on the sharp edge and the edge flipping method will not flip such an edge. Figure 8 shows a crease in the Duffing data set after retiling.

4.6. Attraction force

Because these attractors have infinitely many parallel sheets in a Cantor set arrangement, these sheets eventually become inseparable due to the limited floating point precision. We still want our method to be able to visualize as many layers as possible. However, if the advection or the retiling method create different triangulations on two nearby curved sheets, the two layers may intersect each other, the way two nearby concentric regular n-gon circle approximations will intersect each other if one of them is rotated by π/n. Without this rotation we say that the two approximations are aligned, because corresponding edges are parallel and do not intersect.

Before the self-intersection happens, we apply attraction force on the mesh to create aligned vertices and matching triangulations. Point positions on nearby sheets correspond closely and remain close for a time during the flow. Any subdivision and decimation that happens later on will also maintain matching triangulation topology.

We tried to align the triangulations on nearby sheets by assigning nearby points on two different close sheets an attraction force. Our initial hope was that the attraction forces could align points on different sheets consistently, and because the close sheets are similar in shape, we could have consistent triangulation. However, experiments showed that the attraction force alone doesn't help to make the triangulation consistent on different layers, because a point may be attracted by many points on the other layer. Also, due to the random sampling of points, the numbers of points on different sheets are inconsistent, and it is hard to create consistent triangulations with different numbers of points. So instead of simply applying attraction forces, we also pair points from different sheets, and apply the attraction force only between those paired points.

The point pairs are generated during the sampling stage.

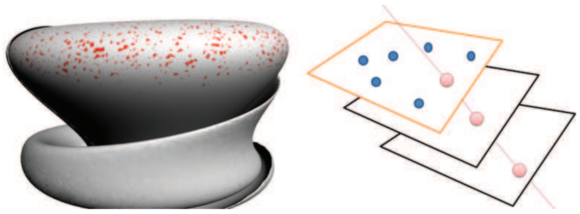

Figure 9: *Left. Self-intersection artifacts caused by inconsistent triangulation on different sheets. Red areas are parts of the second layer that penetrate the first layer. Right. Sampling point pairs on close sheets. Pink points along the ray are paired sampled points on different sheets. Attraction force will be applied between them. Blue points are topology neighbors that have only repelling forces to the pink point on the yellow sheet.*

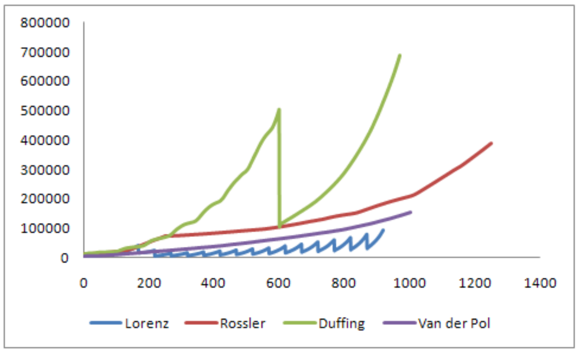

Figure 10: *Vertex count per animation frame. Sudden decreases are from decimation.*

As in [Tur92], we randomly pick a triangle based on a probability proportional to the curvature. But after creating the random point in it, we also shoot out a ray from the point in the direction of the point's normal to test for intersection with other triangles of the nearby sheets. If an intersection point is close enough to the sampled point, we also consider it as a sampled point and pair it with the original sampled point (Figure 9, right). Therefore, for a part of the surface with no close neighboring sheets, the improved sampling method acts as before, but for close sheets, the sampling method can create equal numbers of points on both layers.

Figure 7 shows the result of applying attraction forces on close sheets on the Duffing attractor. We cut out a slice of the model to have a better visualization of the Cantor set structure perpendicular to the sheets, and the effect of the attraction forces. As shown in Figure 7, the triangulations on two close sheets are almost identical. Our idea is based on the assumption that geometrically close points stay together for a while under the flow field and are thus likely to cause identical subdivision of nearby sheets later on. While we cannot prove that this process will eliminate any self intersections, nor predict how long after this retiling the advecting surfaces will remain intersection-free, since nearby points eventually diverge, the method has performed well in practice. Figure 12D represents frame 1000 of our animation sequence of the Duffing attractor, 400 frames after the retiling took place. We colored the front and back surfaces differently, and adjacent layers have opposite orientation, so any intersections would be clearly visible, and none are seen. In contrast, Figure 9A depicts the artifacts at frame 630 when only the normal retiling without attractive forces is applied.

5. Results

Our method is implemented with the OpenGL API and CUDA. Tests are carried out on a 1.86GHz Intel Core 2 processor, with an Nvidia 8800 GTX graphics card. The output images are 1280x1024 in resolution. We have tested our method on the four strange attractors, as well as on two turbulent physical flows.

Table 1 shows the performance on the four strange attractors. Under certain conditions, for example, using a relatively large threshold for splitting edges, we can achieve a real-time frame rate. Figure 10 shows how the number of vertices grows during the animations. For the Lorenz data set, we apply quadric matrix decimation every 50 frames to ensure a better triangulation, since otherwise the in-plane stretching and bending can cause triangle orientations to flip. For the Duffing data set, the retiling method was applied once at frame 600.

Table 1: *Performance. Ad: Advection time, Dt: Decimation time, Dtimes: Decimation times per 1000 frames, Rt: Rendering time.*

Data set	Ad (ms)	Dt	Dtimes	Rt (ms)
Lorenz	18.03	825 (ms)	18.75	6.72
Rössler	33.94	n/a	0	18.43
Duffing	84.49	6.58(min)	1	28.75
Van der Pol	36.90	n/a	0	16.56

Figures 12A-12C and Figure 12E show animation sequences of the four attractors. In these figures, the structure of the attractor is shown by the semi-transparent surfaces. For the Duffing data set and the Van der Pol data set, we use a torus as the initial surface. Figure 12C and Figure 12E show the motion of the torus approaching the attractors. Figure 12D rotates the last frame of the Duffing animation against a fixed sectioning plane, visualizing the periodic change of the 2-D attractor. Our attraction force approach successfully renders the close sheets without intersection.

Figure 11 gives two examples of using the same technique on turbulent physical flow fields. Colors are used to indi-

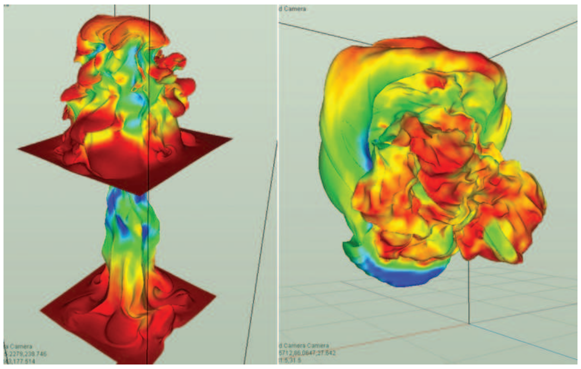

Figure 11: *Two examples of using the polygon visualization method on normal turbulent flow fields. Left: the Solar Plume data set. Right: the Super Nova data set.*

cate the local speed. Our adaptive triangulation successfully tracks the complicated shapes created by the turbulence.

6. Conclusion

The contribution of this paper is a polygonal surface advection method to visualize flow fields with an efficient GPU implementation. The algorithm is an adaptive subdivision process together with an infrequently applied CPU polygon decimation method accounting for nearby layers. We have tested it mainly on the strange attractors and achieved interactive or even real time performance. However we hope that this method will be generic and useful for many other flow field data sets. The attractor data sets are very extreme cases, in that they stretch the surface severely and form sharp creases and nearby parallel sheets. Therefore we believe that the issues we have solved for the strange attractors will help us to expand this polygonal surface advection method to other data sets. As a future research focus, we will also investigate automatic seeding methods for our polygon surface advection, so that a proper starting surface can be posed to reveal important features of the flow field.

7. Acknowledgements

This research was supported by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777 and DE-FG02-08ER54956, and by the Japan Society for the Promotion of Science. We thank the anonymous reviewers for suggestions which greatly improved the paper.

References

[ACSD*03] Alliez P., Cohen-Steiner D., Devillers O., Lévy B., Desbrun M.: Anisotropic polygonal remeshing, 2003.

[AS83] Abraham R. H., Shaw C. D.: *Dynamics: The Geometry of Behavior: Part two: Chaotic Behavior*. Ariel Press, Santa Cruz, 1983.

[BFTW09] Buerger K., Ferstl F., Theisel H., Westermann R.: Interactive streak surface visualization on the gpu. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1259–1266.

[BW83] Birman J. S., Williams R. F.: Knotted periodic orbits in dynamical systems–i: Lorenz's equation. *Topology 22*, 1 (1983), 47 – 82.

[DG01] Dey T. K., Giesen J.: Detecting undersampling in surface reconstruction. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry* (New York, NY, USA, 2001), ACM, pp. 257–263.

[DHJR97] Dellnitz M., Hohmann A., Junge O., Rumpf M.: Exploring invariant sets and invariant measures. *Chaos 7* (June 1997), 221–228.

[DT07] DeCoro C., Tatarchuk N.: Real-time mesh simplification using the gpu. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 161–166.

[GBK03] Gumhold S., Borodin P., Klein R.: Intersection free simplification. In *In The 4th Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics* (2003), pp. 11–16.

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York NY USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

[GL02] Gilmore R., Lefranc M.: *Nonlinear Dynamics and Chaos*. John Wiley & Sons, Inc., 2002.

[GW79] Guckenheimer J., Williams R. F.: Structural stability of lorenz attractors. *Publications Mathematiques de L'IHES 50*, 1 (1979), 59 – 72.

[Kan] Kanamaru T.: Duffing oscillator. http://www.scholarpedia.org/article/Duffing_oscillator.

[KGJ09] Krishnan H., Garth C., Joy K.: Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1267–1274.

[KOD*05] Krauskopf B., Osinga H. M., Doedel E. J., Henderson M. E., Guckenheimer J., Vladimirsky A., Dellnitz M., Junge O.: A survey of methods for computing (un)stable manifolds of vector fields. *I. J. Bifurcation and Chaos 15*, 3 (2005), 763–791.

[Lor63] Lorenz E. N.: Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences 20*, 2 (1963), 130–141.

[PdFS06] Paiva A., de Figueiredo L. H., Stolfi J.: Robust visualization of strange attractors using affine arithmetic. *Computers and Graphics 30*, 6 (2006), 1020–1026.

[PTVF92] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.

[Rös76] Rössler O. E.: An equation for continuous chaos. *Physics Letters A 57*, 5 (1976), 397 – 398.

[SAG03] Surazhsky V., Alliez P., Gotsman C.: *Isotropic Remeshing of Surfaces: a Local Parameterization Approach*. Research Report RR-4967, INRIA, 10 2003.

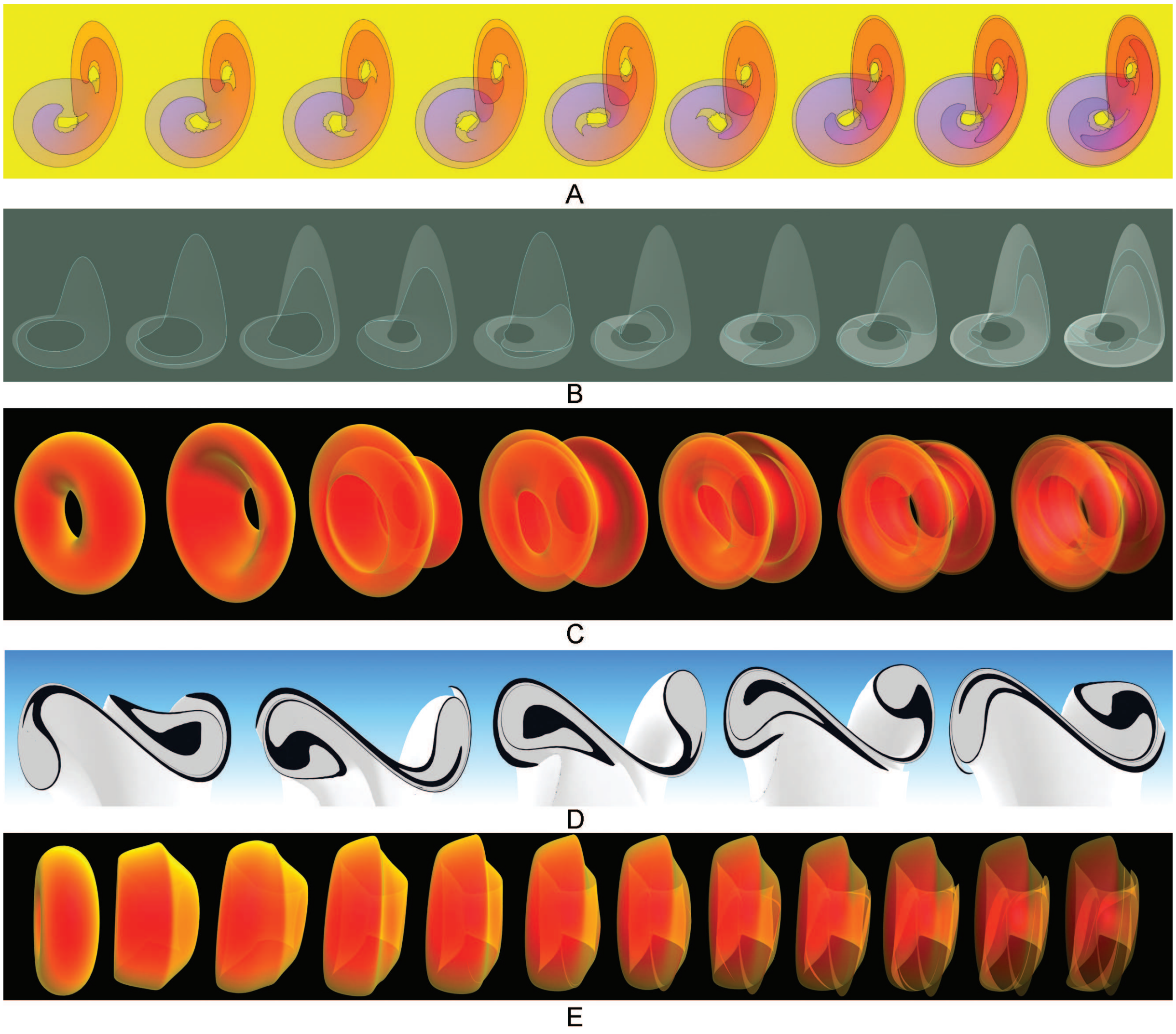

Figure 12: *A: An animation sequence of the Lorenz data set. B: An animation sequence of the Rössler data set. C: A torus approaching the Duffing attractor. D: The Duffing attractor, rotating against a sectioning plane. The opposite sides of the surface have different colors, in order to reveal any intersections. E: An animation sequence showing a torus approaching the Van der Pol attractor.*

[ST94] SAUPE D., TVEDT W.: *Volume rendering strange attractors*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994.

[Str94] STROGATZ S. H.: *Nonlinear Dynamics and Chaos*. Westview Press, 1994.

[STWE07] SCHAFHITZEL T., TEJADA E., WEISKOPF D., ERTL T.: Point-based stream surfaces and path surfaces. In *GI '07: Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), ACM, pp. 289–296.

[Tuc02] TUCKER W.: A rigorous ode solver and Smale's 14th problem. *Foundations of Computational Mathematics 2*, 1 (2002), 53 – 117.

[Tur92] TURK G.: Re-tiling polygonal surfaces. *SIGGRAPH Comput. Graph. 26*, 2 (1992), 55–64.

[vFWTS08] VON FUNCK W., WEINKAUF T., THEISEL H., SEIDEL H.-P.: Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2008) 14*, 6 (November - December 2008), 1396–1403.

[VRS03] VORSATZ J., RÖSSL C., SEIDEL H.-P.: Dynamic remeshing and applications. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications* (New York, NY, USA, 2003), ACM, pp. 167–175.