Flow visualization in science and mathematics

Nelson Max¹, Carlos Correa, Chris Muelder, Shi Yan, Cheng-Kai Chen, and Kwan-Liu Ma
Department of Computer Science, University of California, Davis
1 Shields Ave., Davis California, 95616, USA

E-mail: max@cs.ucdavis.edu

Abstract. We present several methods for visualizing motion, vector fields, and flows, including polygonal surface advection, visibility driven transfer functions, feature extraction and tracking, and motion frequency analysis and enhancement. They are applied to chaotic attractors, turbulent vortices, supernovae, and seismic data.

1. Introduction
At the SciDAC Institute of Ultra-Scale Visualization, we are developing tools to visualize very large data sets, as well as basic visualization paradigms, algorithms, and methods. Here we report on several different visualization methods for 3D motion and flow fields. Visualizing 3D flow is difficult because representations tend to obscure each other, so we attempt to optimize visibility. We discuss several different representations, including advecting polygonal surfaces, contour surface volume rendering, and deforming textures. For interactive performance, we have implemented most of the techniques using hardware Graphics Processing Units (GPUs). Sections 2, 3, 4, and 5 below discuss, respectively, visualizing strange attractors, visibility driven transfer functions, feature tracking, and motion frequency analysis and enhancement.

2. Visualizing strange attractors
A strange attractor is an attractor (a limit set for trajectories) in a flow field whose trajectories show sensitive dependence on initial conditions. Since the initial discovery of the Lorenz attractor [1] in 1963, several other interesting examples from 3D steady flows have been studied, including the Rossler attractor [2], the Duffing attractor, and the Van der Pol attractor. More details on these attractors can be found in [3] and [4]. They look like smooth surfaces along two directions, but have a fractal behavior like a Cantor set in the other direction normal to these surfaces. It is possible to visualize these attractors by taking a random cloud of initial points, and then tracking them in the flow until they approach very close to the limit attractor. However, in looking at clouds of points, it is difficult to see the layered fractal structure. Therefore we have developed polygon-based visualizations, where we advect an initial triangular mesh towards the attractor. For more details of the methods described below, see Yan and Max [5].

¹ To whom any correspondence should be addressed.
2.1 Polygonal approximation

We start from an initial triangular mesh approximation, and let the flow advect the vertices towards the limit attractor, adaptively subdividing and retiling as necessary to track the change in surface shape. We use fourth-order Runge-Kutta integration to advect the vertices.

The Duffing and Van der Pol attractors represent 1D Newton-law motions with periodic forcing. To turn them into 3D flows, we wrap the relevant disk in the 2D position/velocity plane around a circle representing the forcing phase, to give a solid torus. A triangle mesh for the boundary of this torus then serves as the initial polygonal approximation. Figure 1 shows this torus approaching the Duffing attractor.

Figure 1. Six stages in the approach of an initial torus to the Duffing attractor.

For the Lorenz and Rossler attractors, we cannot use such a closed initial surface. One alternative would be to start with a disk somewhere near the attractor. But these two attractors loop around holes, and an initial large disk gets pulled and stretched by the flow near the holes because it has the wrong topology, while a smaller disk takes too long to spread out over the limit attractor. Therefore, we start with a random collection of points, move them for many integration steps until they approach the attractor, and then triangulate the resulting point cloud using the Cocone program of Dey et al. [6]. Figure 2a shows such a triangulation. Note that there is a gap where this program fails, as two sheets of the attractor approach too close to each other, and near this gap, there are incorrect triangles joining these two sheets. Figure 2b shows a modified surface where an annular sector near this gap has been removed, and then a collection of new pink triangles connecting one of the sheets across the gap has been added. Pink triangles have also been added to fill in some concave regions along the edges of the sheets. Figure 3 shows several stages in the advection of this surface, as it spreads over itself to create a fractal structure of multiple sheets.

Figure 2. a) A triangulation of points on the Lorenz attractor. b) The triangulation after an annular sector has been removed, and replaced by a sheet of pink triangles, and other pink triangles have been added to fill in concave regions.
2.2 Triangle subdivision
As the surface is advected by the flow, triangles will get stretched and bent, until they no longer approximate the desired surface. We deal with bent triangles by adaptively subdividing the edges of the mesh. We track the position of the midpoint of each edge in the original triangulation, and if this midpoint moves too far away from the edge, so that the angle between the segments joining it to the edge endpoints strays too far from $180^\circ$ degrees, we subdivide the edge and the two triangles adjacent to it, and begin tracking the midpoints of any newly created edges. For steady flows like those defining these attractors, we can initialize these new midpoints at points inside triangles of the initial mesh.

2.3 Surface retiling
As the surface advection and subdivision proceeds, we soon get too many vertices to advect, and too many triangles for interactive display. Thus we periodically retile the surface, using a method similar to that of Turk [7]. We randomly insert new points on the surface, with probability density varying with the curvature of the surface approximation, by taking the probability of inserting a random point into a triangle proportional to a weighted sum of the triangle area and the average curvature estimates for its three vertices. To create a good spacing for these new points, we apply a repelling force between neighboring points on the same surface sheet, with a smaller force for points in high curvature triangles. For surfaces with borders, like those in figures 2 and 3, we add forces to move the new points away from the border, and then later add extra new border points along a Catmull-Rom spline interpolating the old border vertices. We do five iterations of the following two steps: a) move the non-border points in the direction of the sum of all the forces on them, and b) project them to the plane of their triangles. If they project outside their triangle, the neighboring triangles are recursively rotated up into this plane until the new point can be placed into one of them. All the triangles are retriangulated to include the new points as vertices. The new points are then moved to lie on a weighted least squares quadratic polynomial fit to the nearby old vertices on the same surface sheet, with the weights decreasing with distance from the new point. Then the old vertices are removed one by one, together with any triangles that use them, and the resulting holes are retriangulated. Finally, we iteratively consider all quadrilaterals formed by two triangles adjacent across an edge. If removing that edge and replacing it by the other diagonal of the quadrilateral would improve the triangulation, we perform this “edge flip”.

Our strange attractors consist of many parallel closely spaced curved sheets, and if they are triangulated inconsistently, the sheets will appear to intersect. Therefore, every time we add a new random point $P$ as described above, we look for intersections, with other nearby triangulated sheets, of the line through $P$ normal to the surface. If any intersection points are found, they are paired to the point $P$ with an attracting force that will tend to keep the two points aligned as they both move with the repulsion forces, and thus keep the triangulations aligned. Figure 4 shows a constant-forcing-phase cross section of the Duffing attractor, revealing the fractal structure of the closely spaced sheets.
3. Visibility Driven Transfer Functions

Scalar field volume visualization is also useful for flow visualization, to show scalar quantities like velocity magnitude or vorticity magnitude. Transfer functions are used to map the scalar quantity to color and opacity, which can then be integrated along the viewing rays, as explained in Max [8]. One popular paradigm for designing such transfer functions is to concentrate the opacity in specific narrow regions of the scalar value range, to give the appearance of concentric contour surfaces of different colors. Here, we use a sum of Gaussians opacity transfer function $A$ of the form

$$A(x) = \sum \alpha_i g_{\mu_i, \sigma_i}(x)$$

where $x$ is the scalar data value, $g_{\mu, \sigma}(x)$ is the Gaussian with mean $\mu$ and standard deviation $\sigma$, and the $\alpha_i$ are weighting factors. Figure 5a shows an example of such a visualization, of vorticity magnitude in a pseudo-spectral simulation of coherent vortex structures from [9]. Note that the inner yellow and red contour surfaces are visible only where the excessively opaque purple surface has been sliced away by the boundary of the data volume. In order to see these inner contours better, the purple surface must be made less opaque by changing its weight in the above equation. In figure 5b, the user has adjusted the opacity weights $\alpha_i$ and now the red and yellow contours are now more visible.

We have developed a system for automatically adjusting the $\alpha$ and $\sigma$ parameters, to optimize the visibility of important ranges in the scalar function. The user first specifies the desired importance by another transfer function $O(x)$. The parameters are optimized to minimize a weighted average of the following two functions, summed over all the voxels $p$ in the volume, where $x(p)$ is the scalar value at the voxel $p$. The first function $E_S$ tries to match the transfer function to the user's importance function:

$$E_S(x(p)) = [A(x(p)) - O(x(p))]^2.$$  

The second function $E_V$ is used to maximize the visibility of the important voxels, and thus uses a minus sign in the function to be minimized:

$$E_V(x(p)) = -O(x(p))T(p)$$
where

\[ T(p) = \exp\left[-\int A(x(r(t))) \, dt\right] \]

is the transparency along the ray \( r(t) \) from the viewpoint to the voxel \( p \), parametrized by arclength \( t \). Thus, after summation over all the voxels \( p \), the function to be minimized is

\[ \sum_p \left[ \beta_1 E_S(x(p)) + \beta_2 E_V(x(p)) \right] \]

where \( \beta_1 \) and \( \beta_2 \) are the weights for the matching and visibility terms, respectively.

It seems that it would be a difficult problem to evaluate this sum, since the visibility term involves an integral for every voxel. However, this term can be approximated by the same method used for producing the volume rendered image in GPU hardware. The volume is sliced in front to back order by multiple sampling planes parallel to the viewing screen, with spacing \( dt \), 3D texturing hardware is used to access and interpolate the voxel data and the transfer functions, and the accumulated transparency integral is incrementally adjusted per viewing-ray sample by multiplication by \( 1 - A(x(r(t))) \, dt \), which is the first order Taylor approximation to \( \exp[-A(x(r(t))) \, dt] \). The GPU can then multiply this accumulated transparency by \( O(x(p)) \) and add this term to the output buffer for each ray through a pixel, in order to compute the visibility term to be optimized, in the same way that it multiplies the transparency by the color transfer function and accumulates the contribution to the ray color for creating a volume rendered image. We used the conjugate gradient method to optimize the opacity transfer function, and the GPU hardware to compute it during the optimization.

Figure 6 shows a visualization of the entropy in a supernova simulation with the highest entropies violet and blue and the lowest ones red. The optimized opacity transfer function reveals more of the important features. Note that this is a view dependent optimization, and may produce different results for different viewing angles. For more details see Correa and Ma [10].

![Figure 5](image_url)

**Figure 5.** Vorticity magnitude with: \( a) \) initial transfer function; \( b) \) user-adjusted transfer function.
Figure 6. Entropy in supernova simulation with: a) initial transfer functions; b) automatically optimized transfer functions.

4. Feature Tracking

One can visualize vortices as the regions where the vorticity magnitude, or some other scalar quantity computed from the velocity field, exceeds a threshold. To understand evolving vortices, we need to track them in time. Silver and Wang [11] developed a method that matches the regions in one time step to those in the next, using their volumes of overlap. This requires detecting the voxels that belong to each region, and to each pair of potentially overlapping regions. Although they did this efficiently using an octree data structure, their algorithm still needs to detect the voxels in the interior of the regions. We have developed a feature extraction and tracking algorithm that is based on adjusting the set of boundary voxels, those whose scalar values exceed the threshold, but which have adjacent neighbors whose values do not. Since it operates only near the boundaries of the tracked regions, it can be more efficient, especially if only one or a few selected regions are to be tracked, or if the motion is slow, so that only a few voxels need to be adjusted.

Our extraction and tracking method has two steps, prediction and adjustment. First, the motion of the region's center of gravity in previous frames is used to predict a translation of the previous frame's region boundary onto the current frame. Then region growing and shrinking adjusts the boundary to the new region or regions that it overlaps.

For the motion prediction, we use either a constant prediction, which does not translate the previous frame's region at all, a linear prediction which translates it by the motion of the center of gravity between the previous two frames, or a quadratic prediction, which fits a quadratic polynomial to the center of gravity of the previous three frames, and uses it to predict the translation. In order to compute the center of gravity incrementally, we maintain a count of the number of voxels in the region, and the sum of their coordinate vectors, and adjust the count and sum vector as voxels are added to or removed from the region. We also maintain voxel markings as inside, surface, or outside the region, and translate these markings by the predicted motion when moving to the next time step.

To incrementally adjust the vector sum, count, markings, and the set of boundary voxels for the next time step, we use a queue of voxels that are to be checked, which is initialized to the set of the predicted boundary voxels. While the queue is not empty, we remove a voxel $V$ from it. If $V$ satisfies the threshold condition for the new region, we add it to that region, and then for each neighboring voxel $N$ that belongs to the region, if $N$ is yet in the queue, we add it to the queue, and if any neighbors of $V$ do not belong to the region, we add $V$ to the boundary voxel set. If $V$ does not belong to the
region, we remove it from the region, and enqueue any neighbors that are currently marked as in the region. This adjustment automatically handles topological changes from region bifurcation, joining, and disappearance, but if all regions are to be tracked, methods which check the data values at all voxels are necessary to periodically check for the birth of new regions. The pseudocode below expresses this algorithm.

```
Initialize the queue with the predicted surface
while (the queue not empty)
    pop a voxel V off of the queue
    if (V’s value is above the threshold value)
        mark V as in the region
        if (V is on the new surface)
            save V as part of the new surface
        for each neighbor N of V
            if (N is not marked as inside the region)
                if (N is not in the queue)
                    enqueue N and increment the center of gravity data
            else  // V’s value is not above the threshold value
                mark V as not in the region and decrement the center of gravity data
        for each neighbor N of V
            if (N is marked as inside the region)
                if (N is not in the queue) enqueue N

Figure 7 shows several vortices from the simulation from [9] shown in figure 5, identified by color and tracked over different time steps. For more details, see Muelder and Ma [12].

5. Motion Frequency Analysis and Enhancement
When studying time series of displacements from measurements or simulations of seismic waves and their effects on structures, it can be useful to analyse and visualize energy and displacement in separate temporal frequency bands. Here, we used fourth order Butterworth filters in the frequency domain to define high-pass and low-pass filtered data, and visualize the two components separately. The low-pass component represents the permanent displacement, and the high-pass component represents the transient shaking.
Figure 8. Earthquake displacement: a) no enhancement, b) enhancement of high-pass component.

Figure 9. Earthquake data of figure 8, with the high-pass component visualized by a deforming tube texture.

We visualized data from a finite element simulation of the Humboldt Bay Middle Channel Bridge and the nearby river channel and banks, from Zhang et al. [13]. In figure 8a, the low-pass data is shown in orange, and the high-pass data is shown in blue. Each component was given its own opacity. In figure 8b, the high frequency component opacity was increased by a factor of 2.2, making it more visible. The data was given on a curvilinear Lagrangian grid of hexahedra, which, for visualization
purposes, were divided into tetrahedra, so that the GPU ray tracing method of Weiler et al. [14] could be applied. For more details, see Chen et al. [15].

In figure 9, the high frequency component is indicated by a deforming texture of horizontal tubes. Instead of accessing actual 3D texture data, the tube opacity was evaluated analytically as a function of 3D position, modified by subtracting the time-varying displacement vector at that position, so as to enhance the motion of the tubes.

6. Conclusions and future work
We have discussed a number of different techniques, each developed to visualize different aspects of motion and flow, for various specific goals and applications. We are now applying our polygonal surface advection method to other simulation data, and have found that is has problems with turbulent flows. Edge lengths can become zero, and the advected surface can become too convoluted to compute and render interactively. We hope to develop methods of more aggressively simplifying the surface, and also to use point-based rendering, so that we do not need to maintain the connectedness topology for a polygonal mesh. The visibility driven transfer functions are view dependent, and could cause disorientation during interactive viewpoint changes, and we could develop viewpoint independent ones by optimizing over a representative collection of views.

Acknowledgements
This research was supported in part by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777 and DE-FG02-08ER54956, and in part by the Japan Society for the Promotion of Science.

References
[1] Lorenz E N, 1963 Deterministic nonperiodic flows J. Atmospheric Sc. 20 130 - 41
[2] Rossler O E 1976 An equation for continuous chaos Phys. Lett. 1976 A 57 379 - 80
[3] Abraham R H and Shaw C D 1963 Dynamics: the geometry of behavior, Part two: chaotic behavior Ariel Press
[4] Strogatz S H 1994 Nonlinear dynamics and chaos Westview Press
[5] Yan S and Max N 2009 Polygonal surface visualization of strange attractors, Tech. report CSE-2009-11. University of California, Davis
[6] Dey T K and Giesen J 2001 Detecting undersampling in surface reconstruction Proc. ACM Symp. Computational Geometry 257 - 63
[7] Turk G 1992 Re-tiling polygonal surfaces Computer graphics 26(2) 55 - 64
[8] Max N 1995 Optical models for volume rendering IEEE Trans. Visualization and Computer Graphics 1(2) 99 - 108
[9] Chen S-Y and Shan X 1992 High resolution turbulent simulation using the connection machine-2 Computers in Phys. 6(6) 643 - 6
[10] Correa C and Ma K-L 2009 Visibility driven transfer functions IEEE Pacific Visualization Symposium Beijing 177 - 84
[11] Silver D and Wang X 1997 Tracking and visualizing turbulent 3D features IEEE Trans. Visualization and Computer Graphics 3(2) 129 - 141
[12] Muelder C and MA K-L 2009 Interactive feature extraction and tracking by utilizing region coherency IEEE Pacific Visualization Symposium Beijing 17 - 24
[13] Zhang Y, Acero G, Conte J, Yang Z and Elgamal A 2004 Seismic reliability assessment of a bridge ground system Proc. 13th World Conf. on Earthquake Engineering Vancouver, B. C. Canada
[14] Weiler M, Kraus M, Merz M, and Ertl T 2003 Hardware-based ray casting for tetrahedral meshes Proc. IEEE Visualization 333 - 40
[15] Chen C-K, Correa C, and Ma K-L 2009 Frequency enhancements for visualizing 3D data Tech. report CSE-2009-4. University of California, Davis