3D Flow Visualization Using Texture Advection

David Kao[1], Bing Zhang[2], Kwansik Kim[2], and Alex Pang[2]
[1] NASA Ames Research Center
[2] Computer Science Department, UCSC
davidkao@nas.nasa.gov, {bing, ksk, pang}@cse.ucsc.edu

Abstract

Texture advection is an effective tool for animating and investigating 2D flows. In this paper, we discuss how this technique can be extended to 3D flows. In particular, we examine the use of 3D and 4D textures on 3D synthetic and computational fluid dynamics flow fields. Animations of 3D flow fields using this technique can be found in www.cse.ucsc.edu/research/avis/texflow3d.html.

Key Words and Phrases: scientific visualization, volume rendering, 3D textures, vector field.

1 INTRODUCTION

This paper provides some initial results of using texture advection techniques for visualizing 3D flow fields. Animation is an indispensable tool for understanding the dynamical properties of velocity fields. One has several options on what and how to animate the field. For example, one can animate arrow plots, which have some limited effectiveness for 2D fields. But they quickly become too cluttered for 3D velocity fields. Alternatively, one can trace and animate particles in 2D and 3D, and for both steady and unsteady flow fields [4, 9]. While particle advection is an effective means of animating and showing the dynamics of the field, the particles are discrete and pointilistic. There are techniques that attempt to provide a surface-like property to these discrete particles [10, 12]. Rather than starting with particles, our approach in this paper is to investigate other representations.

Our goal is to produce an animation of the 3D flow field in such a manner that one might observe with cloud patterns, such as the large mass of swirling clouds in cyclonic events or how the jet trail dissipates as it is affected by the wind, etc. As such, we look at ways of representing 3D cloud-like textures that we can then let the flow field advect to generate our animation. In this initial investigation, we use simple fuzzy spheres as our "clouds" to see if the idea merits investigation. We also report on how these spherical cloud textures advect and deform over time using both a synthetic and a computational fluid dynamics (CFD) data set.

In the next section, we provide an overview of texture advection as a mechanism for 2D flow visualization. In particular, we describe how one can work around some difficulties such as providing continuous influx of new textures at inflow boundaries or source critical points. We then discuss how we generated our 3D and 4D spherical cloud textures. In Section 4, we discuss how the advected textures are volume rendered. Results of this technique on the two data sets are presented in Section 5. We finally summarize the work and discuss our future directions.

2 2D TEXTURE ADVECTION

Given a texture image and an underlying 2D flow field, the general approach of a 2D texture advection algorithm is to successively distort the input image over time such that it reveals the flow behavior when animated. We consider each pixel in the image as a massless particle and compute its path in the flow field using a second-order Runge-Kutta integration method. Suppose that there are n time steps in the animation. Then, at time step t_i, where $i < n$, each pixel is integrated backward by t_i time steps. We then replace the current texture at each pixel by the texture value of the pixel where the integration ends and save the "distorted" texture image at time t_i. To animate the flow, we simply playback the saved images from all n time steps.

This texture advection method is a simple and effective approach to animate the 2D flow. However, the method may not work well for some flows. Specifically, there are two cases where the method may fail to depict the flow behavior accurately: (a) if the flow has source critical points, where the flow appears to be repelling away from the critical point, and (b) if there are incoming flows from the grid boundary. In our earlier work [5], we have proposed several methods to deal with these two cases. Here we briefly describe the problems and give an overview of our approaches. The problem described here can occur in 3D flows as well.

In the case where the flow has one or more source critical points, some particles may reach one of the critical points, where the flow velocity is zero. The integration algorithm would then end prematurely because the velocity is zero. The question then is what texture value should be used? If we simply use whatever texture is at that critical point, then that texture value would expand over time in the animation. In the case of a source critical point, it is very likely that particles which are at an equal distance away from the critical point will reach the critical point at the same time. The textures on these particles would then

form a circular pattern. Hence, we would see the a constant texture expanding radially from the critical point. This is undesirable because the animation then would not reveal the flow direction accurately.

In the case where there are incoming flows from the boundary, some particles will reach the boundary during the integration. Analogously, what is the texture value to use? If we use whatever the texture is at the boundary, then we would see constant bands of texture coming from the boundary. This is also not a desirable outcome.

We have proposed two methods to deal with these two situations. The first method is based on 3D procedural textures [1]. During the integration, when the particle reaches either the boundary or the source critical point, where the integration ends prematurely, we step into a 3D texture volume by some depth to determine the new texture value. The depth is based on the number of time steps remaining when the particle integration ends and the texture color is a function of this depth value. We define the 3D procedural texture such that the color of the textures changes as one steps into the 3D texture volume. This provides an effect of continuously feeding texture from the boundary as well as source critical points.

Another method that we have proposed is streamline cycling [5] [2]. For a subset of pixels in the image, we consider each pixel as the seed of a streamline, where we integrate the particle forward and backward until it (a) reaches the boundary, (b) reaches a critical point, or (c) has exceeded the maximum length of a streamline. The latter case is necessary to prevent very long streamlines in circular flow regions. During the streamline computation, we save all the textures at each integration step. To animate the flow, we simply shift the textures along the streamline by a fixed amount at each time step of the animation. We found that it is not necessary to compute one streamline per pixel. It is sufficient to seed the streamlines every two or three pixels.

3 3D and 4D TEXTURES

We worked with both 3D and 4D procedural textures as input for advection of texture coordinates by the flow field. In this initial study, we describe how spherical textures are generated, how the spheres are spaced, and how they change over time. We also describe the texture advection process.

3.1 Spherical 3D and 4D Textures

Each 3D texture is populated with spheres. We initially use spheres to see how well the method works. We eventually plan to replace the spheres with more cloud-like shapes. Currently, each sphere is assigned a texture value of 255 at the enter. This value linearly drops off to 127 as we go out along the radius. Outside the sphere, the texture values are set to 0. This allows for fuzzy spheres. To introduce some variation, one can conceivably add noise to the values within the sphere, or change the range of values defining the sphere.

The spheres populating the 3D texture are randomly distributed throughout the volume using a Poisson distribution. That is, we specify a Poisson radius in addition to the sphere radius. The first frames of Figures 1 and 2 show the initial distribution of spheres in a 3D and 4D spherical textures respectively.

In our current implementation, we statically generate the 3D texture volumes. To allow the introduction of new spherical textures across inflow boundaries, we set the texture dimension to be twice the flow data volume along each dimension. This works for relatively short streamlines or animations. However, we will eventually run out of sphere textures to feed into the flow volume using this strategy.

In our earlier work with 2D flow visualizations using this texture advection technique, we noticed that in some flows, the advected textures may get crowded in some regions of the flow, while the textures get sparser in some other regions. One solution is to add an extra dimension to the texture space (i.e. 3D textures for 2D flow fields) so that new textures may pop out along the new dimension to overcome the sparseness problem. Likewise, old textures may disappear over time as they age in order to reduce the over crowding problem. Hence, we also consider 4D spherical textures in this work. One can simply treat the 4th dimension as time, where old spheres disappear over time, and new spheres appear over time.

While this idea seem logical, it is not very practical due to the large storage requirement for 4D textures. Instead, we use additional frames of 3D textures as a bank of additional texture to draw from. In the examples shown in this paper, we use a bank of four 3D texture frames, and cycle through these four volumes as needed. The next section discuss how these textures are used.

3.2 Advection of 3D and 4D Textures

We use the VisTech library [1, 8] to perform backward integration of streamlines. It handles such tasks as point location, so that we perform the advection in physical space, and it finds the corresponding computational cell that contains the data.

Here is the algorithm we used to advect a single 3D texture:

1. First, map the physical bounding box of the 3D flow field to the 3D texture volume.

2. For each voxel in the 3D texture volume, find the corresponding physical position in the flow volume. Introduce a seed at this point and initiate backward integration for a streamline of length n.

[1] Preprint is available from: www.cse.ucsc.edu/research/avis/texflow2d.html
[2] Preprint is available from: www.cse.ucsc.edu/research/avis/texflow.html

3. For each point along the streamline, obtain the corresponding texture coordinate and value (via trilinear interpolation). These are the values that will be played back at the seed location during the animation.

4. In the backward integration step above, the streamline may terminate in one of three ways:

   (a) If the streamline has a length of zero (e.g. for seed points at the boundary or at source critical points), then a texture value of zero is assigned to each frame in the animation for this point.

   (b) If the streamline reached a boundary or a source critical point after moving m steps, but still less than the total of n steps, then introduce $n - m$ sequential texture values from the texture volume. Note that we usually have a larger texture volume than the flow volume to allow for such buffer zone of texture values.

   (c) If the streamline was entirely within the texture volume, then the corresponding texture values along the streamline are assigned to the seed point in consecutive frames of the flow animation.

Figure 1 shows a few frames from using 3D spherical textures for a synthetic flow field where the flow goes around a vortex ring to produce a donut flow pattern. Note that no new spherical textures can be seen entering the flow volume. For that, we use 4D textures. The algorithm is essentially the same as above, except for the dimension of the texture volume, steps 1 and 4(b).

With 4D textures, each frame is 8x larger than the 3D counterpart (i.e. 2x more in each dimension). In step 1, the physical bounding box is placed in the middle of this expanded texture volume, with buffer texture completely surrounding the bounding box. Only the texture voxels that overlap the physical bounding box are seeded, and displayed. The buffer texture voxels surrounding the flow volume are used in step 4(b) in the following manner.

If the streamline reached a boundary or a source critical point after moving m steps, but still less than the total of n steps, then introduce $n - m$ sequential texture values from the surrounding buffer texture volume. If there is less than $n-m$ sequential texture values available in the current 3D texture frame, then we advance to the next 3D texture frame. If we run out of texture frames, we simply recycle the 4 frames over again. Figure 2 illustrates the effect of 4D textures on the same vortex ring data set.

4 VOLUME RENDERING

We use direct volume rendering (DVR) to render the sequence of 3D advected textures that have been advected by the 3D flow field. From the sequence of rendered volumes, we collate them into an animation. DVR is one of the most popular methods for visualizing 3D data sets. Some of the fundamental work can be found in Drebin et al. [3] and Levoy [7]. More recent work, particularly on speeding up the algorithm, includes hardware texture mapping technique [2, 11] and *shearwarp* [6]. For this project, we use a publicly available DVR software rendering package called UltraVis (www.hpl.hp.com/ultravis) from Hewlett Packard Laboratory. It is optimized for Intel Pentium III based PCs and achieves impressive interactive rendering rates on volume data of reasonable sizes. We chose this program because it is fast, freely available, does not depend on hardware and uses an accurate raycasting method. For animation purposes, we defined transfer functions so that the spherical texture patterns have fuzzy boundaries in order to simulate motion blur effects. We did not use any surface shading effects because they do not enhance the perception of the flow considerably but rather exaggerate the artifacts on the images due to the low resolution of the data. In order to improve depth preception, we also defined semi-transparent materials for the empty space between the spherical texture patterns.

5 RESULTS

We tested our 3D and 4D texture map with 3D flow field data. In this section, we describe some of the animations we generated. Additional and updated sequences are available from the web site listed in the abstract.

Figure 1 and 2 shows snapshot images from the animations of the *vortex ring* data. The vector field has 64^3 cells and is characterized by flow that goes around from the top of the volume, through the central axis and back out and over the top like a donut. In Figure 1, we use a single 3D texture volume that is 128^3. Hence, a 1-to-2 mapping between vector and texture space. From the animation sequence, we observe that the spheres get stretched as they are advected over time. The stretching results directly from the advection process, and simulates motion blur effects which can be enhanced by using different transfer functions during rendering. However, over-stretching would also hinder the desired effects. So, an appropriate balance must be struck.

In Figure 2, we use multiple 3D texture volumes each of them is 128^3. We notice that some of the spheres that were overly stretched are now gone, while some new spheres are introduced most notably near the boundaries. This type of texture seem to enhance the dynamic nature of the flow, and also reduce the problem of over crowding of textures and sparseness of textures in different regions of the flow over time.

In Figure 3, we apply the same techniques to a CFD data set that describes flow over and around an airfoil. There are (115, 157, 83) vector cells in this data set representing flow within a physical bounding box from (0, -0.33, 0) to (1.8125, 0.33, 1). This bounding box is mapped to the center of a texture volume that has (512, 200, 300) cells (i.e. mapped to the (256, 100, 150) interior texture cells). Four such texture volumes are used. For the wing geom-

etry, we assigned a pre-arranged texture value so that the wing can be rendered together with the flow. This helps to provide a frame of reference, but we still need to improve on smoothing the geometry.

6 SUMMARY AND FUTURE WORK

We have presented some initial results on how one might extend texture advection to 3D flow volumes. We have found that straight forward extensions of 2D techniques is possible but may not be the most efficient in terms of storage requirement. The results are still quite early but do show some promise. There are a number of avenues that we are currently exploring to improve this work. This includes: (1) make sure that proposed technique is robust enough to handle other types of 3D critical points and/or curves; (2) explore other types of 3D and 4D textures. In particular, our goal is to produce cloud like textures because most viewers can relate to their experience of watching clouds blown by the wind and inferring the wind pattern from their appearance. Rather than seeding or growing the cloud textures, we are also investigating an alternative subtractive process where we remove "matter" from a solid texture to produce cloud-like textures; (3) experiment with the rendering parameters to help produce a more fuzzy and wispy looking rendering of the advected cloud textures; (4) experiment with simple white noise textures in conjunction with texture advection from a single integration step; [3] (5) improve the space and time requirements of the algorithm; and (6) extend technique to time-varying 3D flow fields.

ACKNOWLEDGMENTS

We would like to thank Chris Henze for the synthetic 3D vortex ring flow field, Jennifer Dacles-Mariani and Greg Zilliac for the CFD wing data set, Suzana Djurcilov on alternative cloud-like 3D textures, and Gunter Knittel and HPL for making UltraVis available to the community. We would also like to thank the members of the Advanced Visualization and Interactive Systems laboratory at UC, Santa Cruz for their feedback and suggestions. This project is supported in part by NASA grants NCC2-5281 and NCC2-1260, LLNL Agreement No. B347879 under DOE Contract No. W-7405-ENG-48, NSF NPACI ACI-9619020, and NSF ACI-9908881.

References

[1] Stephen T. Bryson, David Kenwright, and Michael Gerald-Yamasaki. FEL: The Field Encapsulation Library. In R.D. Bergeron and A.E. Kaufman, editors, *Proceedings of Visualization 96*, pages 241–247. ACM, October 1996.

[2] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings 1994 Symposium on Volume Visualization*, pages 91–98, Washington, D.C., Oct 1994. IEEE/ACM.

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of SIGGRAPH 88*, pages 65–74, August 1988.

[4] A.J.S. Hin and F.H. Post. Visualization of turbulent flow with particles. In *Proceedings of Visualization 93*, pages 46–52. IEEE, 1993.

[5] David Kao and Alex Pang. On animating 2D velocity fields. In *SPIE Visual Data Exploration and Analysis VIII*, volume 4302, pages 41–48, 2001.

[6] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH 94*, pages 451–458, Orlando, FL, July 1994.

[7] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(5):29–37, May 1988.

[8] P. Moran, C. Henze, and D. Ellsworth. FEL 2.2 user guide. Technical Report NAS-00-002, NASA, 2000.

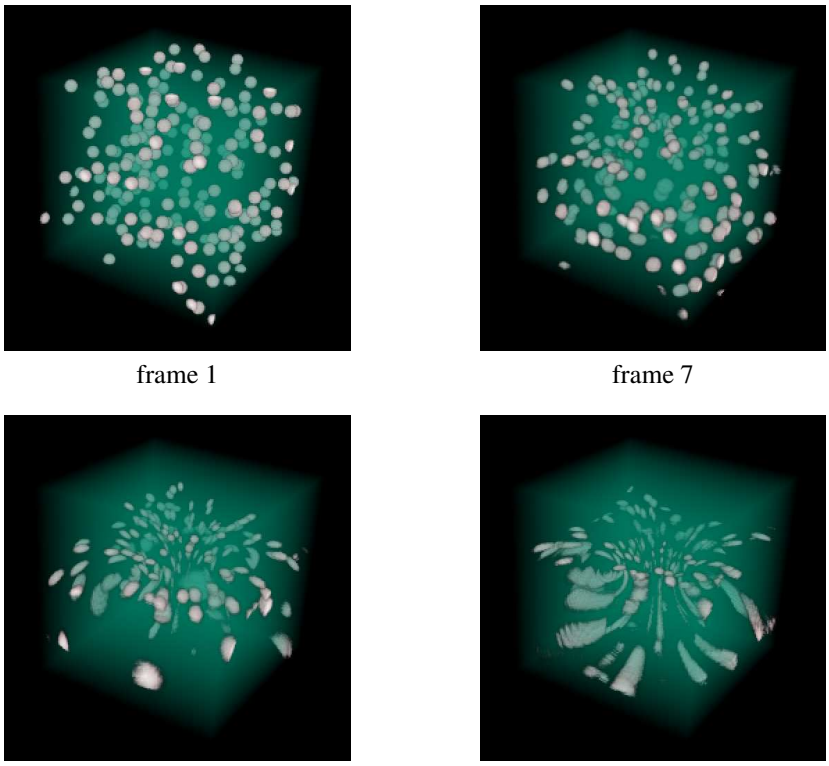
[9] Han-Wei Shen and David L. Kao. UFLIC: a line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of Visualization '97*, pages 317–322, 556. IEEE, October 1997.

[10] Johan Stolk and Jarke J. van Wijk. Surface-particles for 3D flow visualization. In *Proceedings Second Eurographics Workshop on Visualization in Scientific Computing*, 1991.

[11] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via 3D textures. In *ACM/IEEE Symposium on Volume Visualization*, pages 22–30, San Francisco, CA, October 1996.

[12] R. Wegenkittl, E. Groller, and W. Purgathofer. Animating flow fields: rendering of oriented line integral convolution. In *Computer Animation '97*, pages 15–21. IEEE Computer Society Press, June 1997.

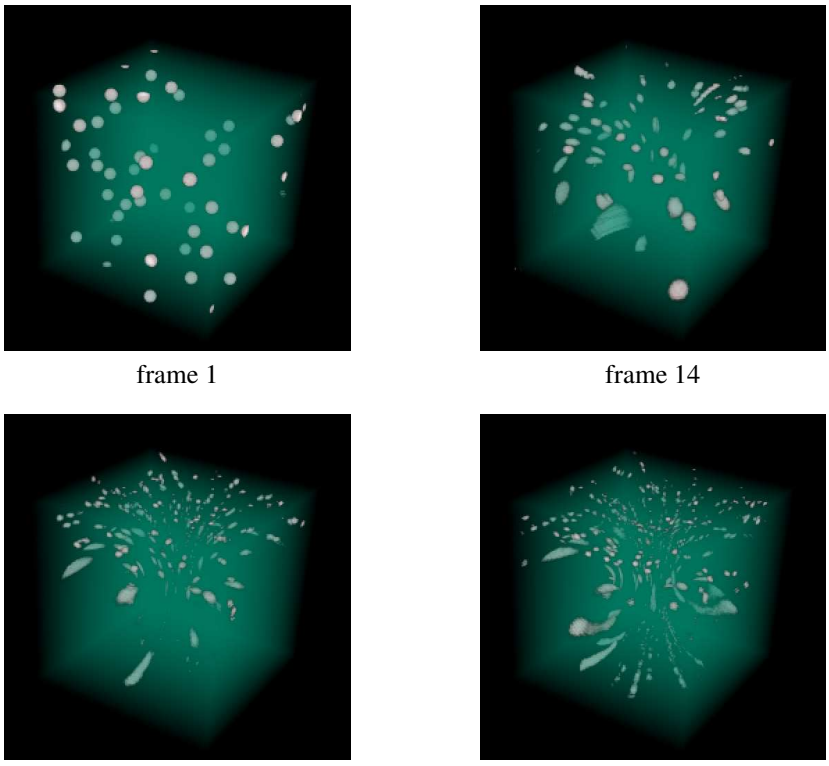
[3] Based on personal communication with Bruno Jobard. See www.csit.fsu.edu/~jobard/SwissMeteo/meteo.html.



frame 1 frame 7

frame 17 frame 30

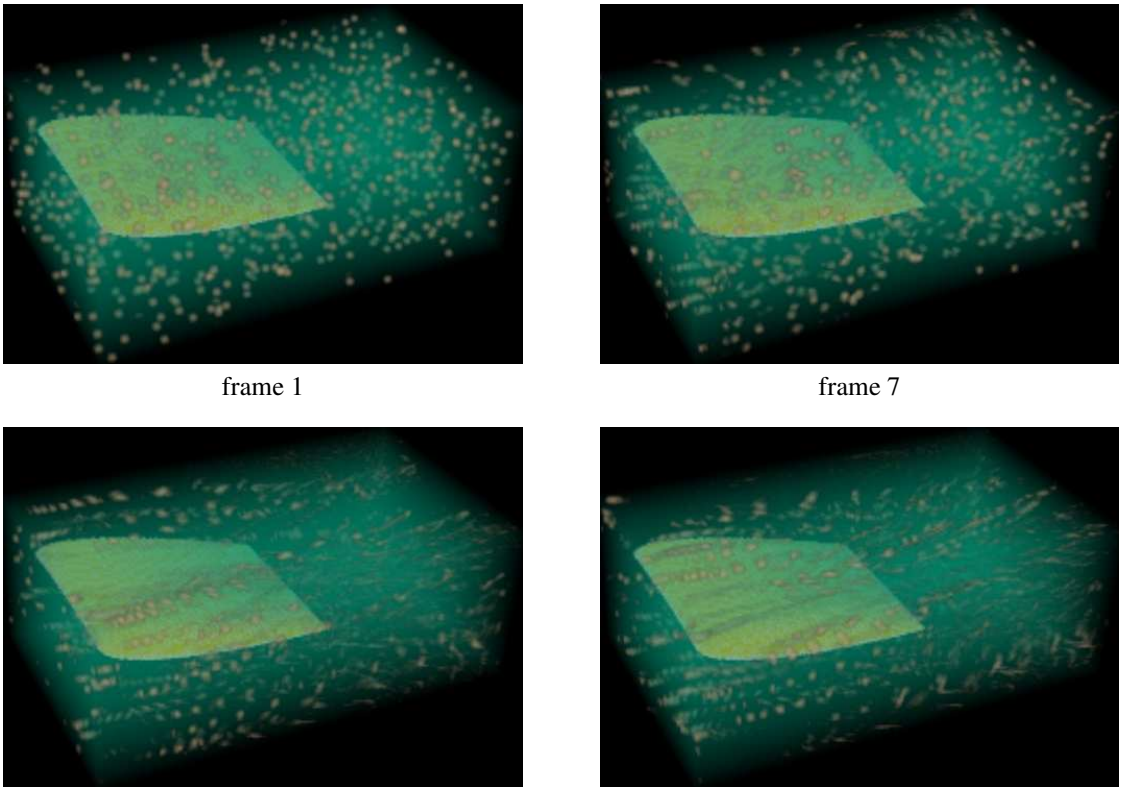
Figure 1. Images from the animation of the *vortex ring* data with single 3D texture map.



frame 1 frame 14

frame 27 frame 34

Figure 2. Images from the animation of the *vortex ring* data with 4 dimensional texture map.



frame 1 frame 7

frame 15 frame 30

Figure 3. Images from the animation of a CFD *wing* data using four 3D texture volumes.