3D Flow visualization using GPU-driven particle system

Oleg A. Potiy[*]
Computer Center
Rostov State University, Rostov-on-Don, Russia

Alexey A. Anikanov[†]
Computer Center
Rostov State University, Rostov-on-Don, Russia

Abstract

In this paper we would like to present a fast and effective technique for 3D unsteady flow visualization based on particle system animation. Particle advancement in flow is performed directly by means of a graphic processing unit(GPU).

The suggested method derives the advantages of texture visualization techniques - continuous cover of displayed domain and the most intuitive way of flow imagery. On the other hand, this technique allows to perform 3D flow animation with interactive frame rate and sufficient quality. Tracking particles injected in the velocity field do not require expensive, from the computation point of view, 3D volume rendering procedure, in contrast with texture 3D visualization techniques. Employing of GPU hardware capacities and stream computation approach makes possible to relieve the stress from a central processing unit.

Keywords: general-purpose computations on GPU, GPU based visualization, flow visualization, particle visualization, texture flow visualization

1 Introduction

The development of computational mathematical physics and fluid dynamics caused by computer power growth lasts three decades, gives us possibility to simulate complex behaviour of natural phenomenas and industrial effects without running expensive experiments. One of the most commonly used special cases of such modeling in ecology and hydraulic gas dynamics exploration is computation of 2D or 3D motion of continuum velocity field on equidistant or non-equidistant nodes net. The results of this numerical flow computation are large 2D or 3D arrays of vector or scalar data. The successful outcome of experiment depends much on the way of representing data and the conclusions the researcher could come to basing on image obtained. Thereupon, velocity field visualization becomes an essential part of many computational modeling tasks. The stage of data visualization should provide exhaustive information about flow structure, speed and other characteristics.

With the advent of techniques, employing graphic hardware capacities, it became possible to produce high-quality 2D flow visualization on a stadart PC-workstation in real-time mode. IBFV technique developed by Jack van Vijk [van Wijk 2002] suggested to produce image advection along streamlines of a velocity field by means of texture mapping on distorted quad polygonal net with vector field values assigned to mesh nodes. Further research in this area have lead to the development of 3D IBFV - visualization technique for 3D steady flow [Telea and van Wijk 2003]. Our contribution to the field of texture visualisation was the development of method for texture advection of unsteady flow [Anikanov and Potiy 2003] and its hardware GPU-based modification [Potiy and Anikanov 2004].

While visualizing planar 2D velocity fields texture techniques exhibit appropriate performance and give us detailed picture of flow, for 3D case there are several problems. Since 3D texture techniques deal with texture cube, it is necessary to use volume rendering algorithms [Hadwiger et al. 2002]. This fact makes implementation of such methods in two phases - texture advection itself, followed then by texture cube rendering. More over, texture visualization produces continuous images. In consequence of this generic feature, 3D texture visualisation cannot resolve small details of flow. Thus techniques for continuous texture advection of whole texture cube are not well suited for 3D flow animation in real-time scale. This fact becomes particularly apparent while visualizing unsteady velocity vector field.

Better solution, in our opinion, the use of discrete particles placed in the velocity field. Tracking particles along path lines produce flow animation [Berger 2000], [Stefan Guthe 2001]. Particles leave a gradually decaying in time trace, graphically demonstrating their path in flow. Discrete particle traces visualization allows to decrease haziness of the image obtained, while imaging 3D flow field. As a result this measure increases cognitive power of visualization.

One of the pioneer achievements in hardware-based particle visualization was Weiskopf's work [D. Weiskopf and Ertl 2001], which suggested to inject randomly spreaded particles in 3D texture cube and calculate their advection using programmable rasterization pipeline of a graphic card. The performance of this technique was about 4 frames per second. Final imaging was produced by volume rendering procedure, so specialized graphic station was required. Further development was the GPU-based modification of this concept [Weiskopf and Ertl 2004] with the performance showing 10 fps.

Latest achievements in graphic industry such as fully programmable rasterization pipeline, mostly stimulated by game manufacturers, allow to continue work on techniques effectively employing graphic card hardware capacities. Calculation organization based on direct GPU exploit gives possibility to relieve a stress from the central processing unit and, in that way, improve the overall performance of the visualization system.

We suggest a new technique for 3D flow visualization using particle system tracking, computed by means of graphic processing unit. The method combines the advantages of texture and particle visualization techniques, inheriting a paradigm of matter advection along stream lines and employing geometry objects for particle representation. We chose the following particle behaviour model - moving in flow a particle decays and eventually passes away. After that the particle regenerates in randomly chosen location of the visualizing domain. Such behaviour strategy allows not to track the whole particle trace, because tracing a streamline at once leads to inaccuracy accumulation. Using animation of streamlines short segments results in more dynamic and intuitive visualization in 3D case. Direct GPU utilization let us produce real-time interactive 3D visualization (from 10 to 50 frames per second, depending on particle

[*]e-mail: opotiy@mail.ru
[†]e-mail:anikanov@rsu.ru

amount and live period) in stationary and instationary flow cases on non-specialized graphic card. Besides that, the algorithm presented in this paper is a effective example of general-purpose GPU computauions

In the second section of the paper the detailed method overview - GPU-driven computation of particle advection, particle distribution in the visualized domain and the suggested way of particle tracks rasterization are presented. The third section contains technical details of technique implementation using OpenGL library. Section 4 comes with obtained results discussion and brief outline of possible modifications for better performance and cognition. Finally, section 5 summarizes the whole paper in conclusion.

2 GPU-driven particles visualization

Our method is based on using graphic processor for computation of 2D texture which colour components are treated as particle coordinates. This texture is rasterized into non-screen buffer by means of fragment program. The following reading of texture data from this buffer gives us stream lines vertex coordinates of moving particles along a path lines. The next stage of our technique is particle trace rasterization. The suggested way of calculations assumes that the researcher has a standard consumer PC-workstation with a graphic card installed on it, which supports non-screen rendering and fragment program execution environment within the bounds of the programmed rasterization pipeline support.

It should be pointed out, that the mentioned hardware capacities are available in all modern state-of-the-art graphic cards presented on the market. The proposed technique implementation uses OpenGL graphic library and its several extensions. Without losing any commonness, we can assume that field for visualization is defined in the unit cube domain.

2.1 Vertex GPU calculations

The most natural way of organizing general-purpose computations on a graphic processing unit is using planar texture maps as data array storage for a fragment program. A fragment program receives several texture maps as input data and returns its computation result as a pixel data in frame buffer. As a rule, single texture value(frequently called texel) is represented as one-,two-,tree- or four component value. The type and precision of components in this texture vector can be chosen by user as Glbyte, Glshort, Glint.

Recent development in a graphic hardware industry essentially extended this concept by adding support of four-component (RGBA) float point textures. Each texel in such texture objects is represented as 32-bits four floating point values. We consider RGB floating point texture vector as a particle coordinate storage (p_x, p_y, p_z) Forth alpha-component was chosen to represent information about life time of the corresponding particle.

In our algorithm 3D velocity field is specified by the planar RGB texture map, which texels are treated as vector field values $f_{xyz} = (f_x, f_y, f_z)$ assigned to nodes of a regular 3D net. It would be more natural to use 3D texture map with floating point components, but hardware support for such texture object exists only in recent graphic cards. These products are not available for us, so we should find an alternative solution. In this connection, the decomposition of a 3D texture map was performed - we splited texure cube into a number of 2D maps and arrange them on the surface of a large planar texture with an appropriate size. For example, packing texture cube with dimension 128^3 requires 2D texture map with size 4096×512 (fig. 1).

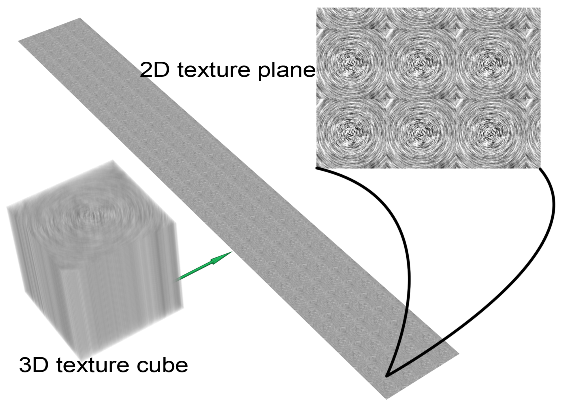

Figure 1: Converting 3D texture cube into 2D plane

The fragment program calculates new particle location, taking into account current coordinates $p_{xyz}(i) = (p_x(i), p_y(i), p_z(i))$ and extracting from packed velocity field vector value Tex_f at the location p. The input data for this fragment program on step i are tree texture objects - texture array with current particle coordinates $Tex_p(i)$, packed into 2D texture map velocity field Tex_f and one random texture $Tex_{rnd}(j), j = \overline{1,N}$ taken from the set of random locations used for "dead" particle coordinate renovation (fig. 2). The output of rasterization pipeline is the new texture array $Tex_p(i+1)$ which will be treated as $Tex_p(i)$ in future iteration. At the moment, particle advection through a flow is calculated with the help of the standard first-order integration method. However, the suggested way of computations makes possible an implementation of higher order integration techniques without adaptive choice of integration step. In this case, inaccuracy is equilibrated by small step, short particle life time and consequently small length of a curve, which particle circumscribes. So, an inaccuracy has no time to become big enough for having effect on visualization. It should be pointed out, that in many cases visualization used just for observation and subjective analysis. So, under these conditions we have no strict requirements for the numerical integration procedure.

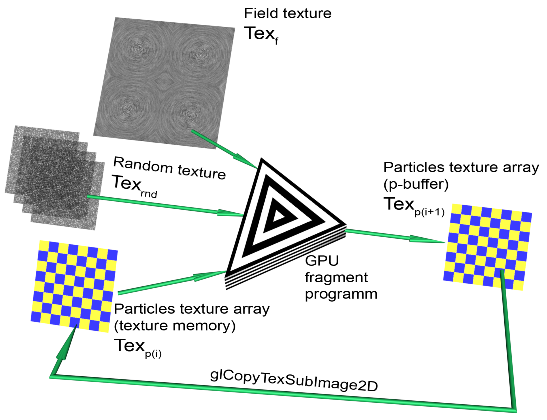

Figure 2: Particle advection pipeline

After texture coordinate array fragment program rasterization, frame buffer content is copied into a main texture memory and involves in the next algorithm iteration as current particle coordinates. Copying performs with the help of glCopySubTexture function call, which replaces a content of texture array $Tex_p(i)$ with new data $Tex_p(i+1)$.

2.2 Particle behaviour and visualizing

It is easy to see, that with continuous particle advancement in a flow, in the course of time all of them will be carried outside of visualized domain or accumulated inaccuracy will make the particle trajectory non-verisimilar. Illustration of this fact is continuous tracking of a single particle, placed in the rotation velocity field (fig. 3).

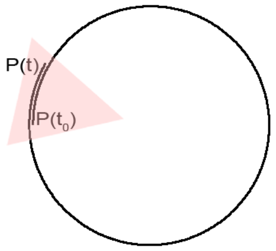

Figure 3: Tracing single particle in the rotation field.

To remedy this phenomena, the life time parameter $p_l(i)$ is assigned to each particle. Alpha-value channel of the texture array is used for this purpose. This life time parameter is incremented on each iteration. When $p_l(i)$ becomes equal to some limit value L this parameter is set to zero and particle considers as "dead" and newly "regenerated". Dead particle regenerates at the new location p_{rnd} inside the unit cube domain specified by preset set of random noise textures $Tex_{rnd}(j), j = \overline{1,N}$. On every method iteration, a random number k, $1 \leq k \leq N$ of a noise texture map $Tex_{rnd}(k)$ is involved with the process of dead particle regeneration in texture array $Tex_{p(i)}$. Besides that, a particle is confined in the unit cube domain. This measure preserves a particle from leaving visualized domain - if a particle comes out from the unit cube we mark it as "dead" by setting $p_l(i) = 0$.

In such a way, the fragment program calculates $p(i+1)$ using the next formula (1):

$$p(i+1) = (p_{xyz}(i+1), p_l(i+1)) = \begin{cases} (p_{xyz}(i) + f_{xyz}\Delta t, p_l(i)+1), \\ \quad if\ p_{xyz}(i+1) \in R^3(0,1) \\ \quad and\ p_l(i) < L \\ \\ (p_{rnd}, 0),\ p_{rnd} \in Tex_{rnd}(k), \\ \quad else \end{cases} \tag{1}$$

Such particle behaviour strategy allows to gain acceptable random distribution of dead particles in visualized domain, using pseudo-random number generators like rand(). Uniform cover of flow domain by particles makes possible visualization of all parts of a field. Particle life cycle mechanism, in one's turn, gives possibility to manage computer memory effectively without deleting dead particle. Fragment program just renovates its position inside the unit cube domain.

We chose the simplest way of the particle trace visualization as a semitransparent line segment set using GL_LINE primitive. Vertex texture array $Tex_p(i+1)$ is copied from non-screen buffer and used for stream lines rasterization. The life time parameter p_l is treated as an element index in a vertex array. Visualization of this array produce particle tracking as a piecewise polygonal representation. Using semitransparent line segments allows visually uniform flow imaging, just as using texture visualization techniques. However, in case of 3D field the resolving capacity of stream line short segments as polygonal primitives is much better, in contrast with volume rendering methods. Besides that, imaging stream line as a piecewise polygonal curve allows application of different techniques such as dependent texture mapping, colour coding and e.t.c. These measures will raise cognitive capacity of the visualization.

For purposes of stream line set imaging perspective projection is used. This gives better feel of 3D space, rather than parallel projection. In texture visualization techniques a parallel projection is mostly used, while application of perspective projection is difficult, because it is necessary to build spherical sections when visualizing 3D texture [Engel and Ertl 2002]. Since in our method flow is depicted using geometry objects we are not constrained in applying different types of projecting.

3 Technique implementation

While working over this technique we have been using NVIDIA GeForce FX 5900 graphic board on AGP 4.0 bus and OpenGL library extensions - NV_texture_rectangle, NV_fragment_program and NV_float_buffer [Kilgard May 19, 2004]. Using of NVIDIA-oriented OpenGL extensions slightly constricts the variety of hardware could be used, however our technique can be adopted to ATI graphic solutions with ease. We used the extension NV_texture_rectangle for fragment shader implemenataion, in view of the fact that this extension is optimized for NVIDIA platform. But it is possible to rewrite GPU fragment program code for running with pixel shading universal extension ARB_Fragment_Programm [ATI 2003], [Kilgard May 19, 2004].

As it was mentioned above, vertex texture array Tex_p is rasterized into non-screen frame buffer (p-buffer) in order to avoid slow screen rendering operations, so non-screen rendering allows to increase the performance of visualization system. Using specific p-buffer pixel format makes possible reading floating point pixel data from it.

Standard OpenGL texturing is limited in size to images with power-of-two dimensions and an optional 1-texel border. Using rigidly fixed dimensions in our case is not convenient, because this condition implies strong restrictions on a number of particles variation - 4096(64^2), 16384(128^2) or 65536(256^2). In contrast, NV_texture_rectangle extension adds support for a new texture object that implements 2D textures without requiring power-of-two dimensions. This extension make us free in adjusting smoothly the number of particles in a system, since we are able to choose arbitrary size for texture array Tex_p.

When visualizing stream lines, vertex buffer object was used in order to decrease a function call number for geometry objects generation. The alpha-channel of texture array object used to store life time of a particle treated as an element index in a stream line vertex array.

Employing the mentioned hardware capacities gives us possibility to produce interactive animation of nonstationary flows in a real time scale. The speed of the visualization is varied from 12 to 50 frames per second, depending upon the number and life time parameter of particles. Frame rate degrades as a consequence of growing particle life time since particle track length and amount of vertex data passed to a graphic card increase considerably. It should be pointed out, that even in arduous duty (about 40000 particles and 200 vertexes in the particle trace) our technique has a performance for about 17-25 frames per second, so we see some performance advantages in comparison with 3D texture visualization.

4 Results and future work

We have visualized several vector fields and the results are shown on (fig. 4). These images were obtained using 25600 particles with lifetime parameter value 100 at about 28-30 fps.

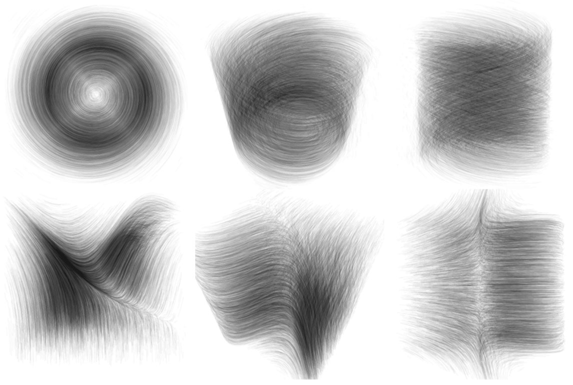

Figure 4: Example datasets visualization.

Further efforts concerning the development of this technique will be findings of the optimal approach to particles distribution in the unit cube. At the moment we observe the effect of stream lines higher concentration at locations where particles pass maximum length distance, while in other locations particle concentration degrades. Perhaps, an elimination of this phenomenon will require an essential revision of particle behaviour model.

The way of stream line imaging may be considerably improved by dependent texture mapping on a particle track. Besides that, polygonal particle trace representation allows application of a great number of techniques (stream lines coloring, dependent lighting and e.t.c.) for increasing cognitive qualities of visualization. In many respects this is stimulated by a flexible control of the programmed rasterization pipeline.

We should point out, that we have just started work on this technique. Code and algorithm detailed optimization will considerably increase the method performance.

5 Conclusion

In this paper we have presented a new fast and efficient technique for unsteady 3D flow visualization, based on vertex data computations by means of GPU-based texture rasterization. Such an approach gives us possibility to produce real time flow visualization using particle trace imaging. The suggested way of particle advection makes possible nonstationary velocity field interactive rendering. In the 3D case, polygonal stream line representation exhibits better results, from the point of view of performance and cognition, than continuous texture cube advection followed by volume rendering. Geometry objects are not constrained in projection type, so we used perspective projection. Obtained results encourage us to continue work on this technique.

References

ANIKANOV, A. A., AND POTIY, O. A. 2003. Texture advection for 3d flow visualization. In *Proceedings of GRAPHICON 2003*, MSU.

ATI, 2003. Ati opengl extension support.

BERGER, S. 2000. Color-table animation of fast oriented line integral convolution for vector field visualization. In *Proceedings of WSCG'2000*, WSCG.

D. WEISKOPF, M. H., AND ERTL, T. 2001. Hardware-accelerated visualization of time-varying 2d and 3d vector fields by texture advection via programmable per-pixel opefations. In *Proceedings of Vision, Modeling, and Visualization VMV '01 Conference*, Stuttgart, November 2001.

ENGEL, K., AND ERTL, T., 2002. Interactive high-quality volume rendering with flexible consumer graphics hardware. EUROGRAPHICS 2002.

HADWIGER, M., KNISS, J. M., ENGEL, K., REZK-SALAMA, C., AND LANDIS, H., 2002. High-quality volume graphics on consumer pc hardware. ACM SIGGRAPH 2002 Course #42 Notes, July.

KILGARD, M. J., May 19, 2004. Nvidia opengl extension specifications.

POTIY, O. A., AND ANIKANOV, A. A. 2004. Gpu based texture flow visualization. In *Proceedings of GRAPHICON 2004*, MSU.

STEFAN GUTHE, STEFAN GUMHOLD, W. S. 2001. Hardware-accelerated visualization of time-varying 3d and 3d vector fields by vexture advection via programmable per-pixel operations. In *Proceedings of Vision, Modeling, and Visualization VMV '01 Conference*, Stuttgart, November 2001.

TELEA, A., AND VAN WIJK, J. J. 2003. 3d ibfv: Hardware-accelerated 3d flow visualization. In *Proceedings of ACM SIGGRAPH 2003*, ACM.

VAN WIJK, J. J. 2002. Image based flow visualization. *ACM Transactions on Graphics 21*, 3, 745–754.

WEISKOPF, D., AND ERTL, T. 2004. Gpu-based 3d texture advection for the visualization of unsteady flow fields. In *Proceedings of WSCG 2004 Short Papers*, WSCG.