GPU-Based Texture Flow Visualization

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

The current state-of-the-art graphic hardware from world leading 3D industry manufactures offers a way to configure and to control geometry and texture processing pipeline. As a result a possibility to use these video cards as a platform for scientific visualization has risen. The objective of this paper is to present current efforts on developing two texture visualization techniques for unsteady 2D and 3D vector fields, employing graphic processing unit (GPU) with OpenGL extension ARB_vertex_program. The program that implements suggested GPU-based 2D visualization procedure is presented. Methods presented in this work are a straight-forward extension of the Lagrangian-Eulerian advection technique. GPU exploit allows us to increase method performance and gain higher abstraction level in visualization applications.

Keywords: GPU, GPU based visualization, texture flow visualization

1 Introduction

For a long time quality texture visualization in real time scale has been performed on the high-end graphic stations. Main advantages of this technique are the continuity of image generated and the uniform cover of whole vector field domain to be visualized. Texture-based visualization methods are free from geometry-based technique characteristic restrictions and artifacts, such as chaotic line overlapping near field critical points and blank holes in case of low-speed flow. Furthermore, some of texture methods enable visualization of unsteady vector field.

One of the pioneer achievements in field of texture visualization was developing of the so-called spot noise technique by Jarke van Wijk [van Wijk 1991]. Further investigations in this branch of scientific visualization have led to LIC algorithm [Cabral and Leedom 1993] with many variations, including its propagation to a 3D case followed by accurate volumetric rendering. Once developed, LIC technique has widely spread in vector field visualization due to high resolution visions generated. However, this method requires high computational power systems to be effectively run on.

Since techniques employing graphic cards hardware capacities had been developed, it has become possible to perform quality texture visualization on a standard consumer PC-workstation in real time scale. One of the first successful implementations of this concept was Jack van Vijk work [van Wijk 2002] in which he presented fast image based texture visualization algorithm - IBFV. The main idea of this technique is the iterative image distortion, performed by texture mapping on deformed mesh by means of OpenGL API. Every mesh vertex is shifted according vector field value placed in this location. Among recent achievements in flow visualization is the development hardware-driven technique (3D IBFV) for visualization of stationary 3D vector field [Telea and van Wijk 2003] and software texture advection for 3D unsteady case [Anikanov and Potiy 2003]. Arising computational power and increasing programming flexibility of modern 3D graphic hardware allows to continue research in the field of hardware-driven flow visualization. Due to this new features offered by graphics hardware, many popular visualization algorithms are now being implemented efficiently on a GPU side.

The main goal of this work is the development of techniques with higher performance, especially in case of unsteady 3D field, and better cognition ability to provide user insight into structure and topology of complex formations. All methods presented in the paper are implemented using power of OpenGL API and its extensions - ARB_vertex_program and GL_EXT_texture3D.

In this paper GPU based methods for 2D and 3D texture visualization are presented. Developed techniques run directly on GPU. Such an approach makes possible to decrease CPU load in case of unsteady vector field and enables visualization applications to be implemented on higher abstraction level. A comparison with existed 3D flow visualization technique is given.

In section 2 the paper continues with brief overview of programmed geometry and texture object rasterization pipeline. Section 3 contains explanation of IBFV technique. Section 4 presents method for unsteady 2D vector field visualization and software application based on this GPU technique. Section 5 introduces the idea of GPU-based 3D flow visualization and gives comparison of this matter with existing 3D imaging technique. Finally, Section 6 draws the conclusion and outlines future work upon this problem.

2 Programmed Rasterization Pipeline

In this section we briefly discuss main stages of 3D rasterization pipeline and outline its programmed features our techniques are built upon.

There are three standard stages of graphic 3D pipeline, independently from technology used: geometry primitives defining, vertex and polygons processing and local illumination model estimation, polygon rasterization and texture mapping in combination with per-pixel fragment operations. These stages are then followed by number of fragment visibility tests - alpha tests, stencil buffer test and Z-buffer test.

First stage defines a geometry object as associated set of vertexes and faces that forms geometry model mesh (Fig.1, 1). Then goes second pipeline phase - model-view transformations and vertex illumination (Fig.1, 2). The hardware geometry engine computes linear transformations, such as translation, rotation, and projection. Local illumination models are also evaluated on a per-vertex basis at this stage of the pipeline. Finally fragment rasterization and per-pixel operations take place on 3 final stage of pipeline (Fig. 1, 3).

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

Rasterization is the conversion of geometric object into fragments. Note that although a fragment is closely related to a pixel, it may be discarded by one of several per-fragment tests, such as alpha or depth testing, before it becomes a pixel in the resulting image.

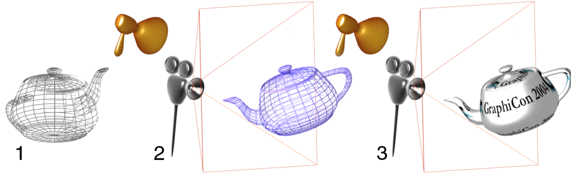

Figure 1: Three states of OpenGL graphic pipeline.

The current state-of-the-art 3D graphic platforms, such as nVIDIA GeForce4/FX [NVIDIA 2003] ATI Radeon 9600/9700/9800 [ATI 2003], allow developer to override built-in processing for second and third pipeline phases (Fig. 1, 2,3). During second pipeline stage, OpenGL extension ARB_vertex_program makes possible a dynamic calculation of per-vertex attributes, such as vertex position, normal, texture coordinates and e.t.c. In the same way, ARB_fragment_program extension enables user-defined fragment processing on third phase. Consequently, vertex program provides a means for texture advection by a dynamic computation of texture mapping coordinate using GPU. Coming into use, fragment programs permit flexible texture operations, such as dependent texture reads, to a greater extent, because of exploiting a per-pixel hardware abstraction level.

It must be noticed that the plenty of 3D graphic low and medium-level hardware enables vertex program execution on board, while fragment programs execution (frequently called fragment shaders) is a standard feature for only high-level graphic cards. However, driven by latest advances in 3D industry, these two features will probably be available to consumer in full measure.

3 Image Based Flow Visualization

This section follows with short overview of IBFV technique, since our methods derive from it.

The IBFV (Image Based Flow Visualization) procedure was suggested by Jarke van Wijk [van Wijk 2002]. Here we explain the main concept of this technique. Consider in the general case n-dimensional vector field (in further discussion we will lay $n = 2$ or 3):

$$V(x,t) \in R^n, x \in R^n, t \in R, \tag{1}$$

and differential equation, supplied with proper initial condition which describes dynamic system behaviour:

$$\frac{dp(t)}{dt} = V(p(t),t), p(0) = p_0 \tag{2}$$

First order numerical approximation of this equation is a well-known Eulerian formula for integrating with fixed time step Δt:

$$p(t+\Delta t) = p(t) + V(p(t),t)\Delta t. \tag{3}$$

Laying $t_k = \Delta t k$ we will have:

$$p_{k+1} = p_k + V(p_k,t_k)\Delta t. \tag{4}$$

Consider field of material values $F(x,t)$, where $x \in S$. Its physical meaning is a particle with material value $F(x,t)$ at location x and time moment t. As a rule, an abstract "material value" is treated as RGB triplet,weight or particle luminance and etc. We should note, that the material value of a particle is stable in time and, evidently, the next parity take place - $F(p_{k+1},t_{k+1}) = F(p_k,t_k)$ if $p_{k+1}, p_k \in S$. In the general case we have:

$$F(p_{k+1},t_{k+1}) = \begin{cases} F(p_k,t_k), if\, p_k \in S \\ 0, else \end{cases} \tag{5}$$

In the course of time, there will be a moment, when main part of particles p_k will have material value $F(x,t)$ equal to zero, since p_k will drive outside domain S and a so-called "wash-out" phenomena (in terms of particle washed by the flow) will take place. To prevent this phenomena we can take a convex combination of image $F(x,t)$ and noise signal $G(x,t)$:

$$F(p_{k+1},t_{k+1}) = (1-\alpha)F(p_k,t_k) + \alpha G(p_{k+1},t_{k+1}) \tag{6}$$

After N successive iterations will have:

$$F(p_n,t_n) = (1-\alpha)^n F(p_0,t_0) + \alpha \sum_{i=0}^{n-1} G(p_{k-i},t_{k-i}) \tag{7}$$

First term can be set to zero by taking absolutely black start image $F(p_0,t_0)$ in order to prevent its influence on final image:

$$F(p_n,t_n) = \alpha \sum_{i=0}^{n-1} G(p_{k-i},t_{k-i}) \tag{8}$$

According to this equation, the colour of particle p_n is a numerical approximation of integral convolution of noise image $G(x,t)$ along stream line of particle p with exponentially decaying kernel filter $\alpha(1-\alpha)^i$. Noise image should not contain high frequencies to gain smooth and laminar visualization.

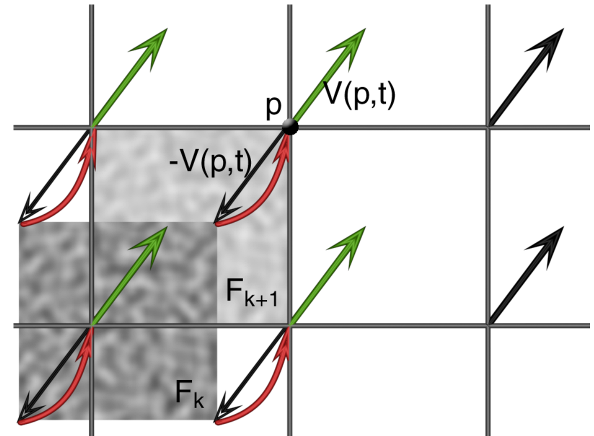

Figure 2: Texture mapping on node net.

Method implementation in OpenGL context represents an iterative process during which distorted mesh with texture F mapped on it is rasterized into the frame buffer. After that, using alpha blending mechanism, a convex combination of distorted texture F and noise image G_k, $k = \overline{1,N}$, is performed. The resulting image is temporarily stored and is involved in next procedure iteration. We have slightly modified this technique by fixing mesh and generating texture coordinates for mesh points, instead, in accordance with vector field values defined in mesh points. Mesh points with its p vector field values $V(p,t)$ are shown on Fig. 2.

One step of backward integration along stream line $p(t)$ leads to a point p_{k-1} with texture coordinates $p_{k-1} = p_k - v(p_k,t)\Delta t$. This

point is a previous in time position of particle in a flow. Further, after time Δt, p_{k-1} moves to p_k. Such an approach allows to deal with the problem from the position of Lagrangian-Eulerian advection [Jobard et al. 2002]. It is important to note, that this fact is true only for points placed in mesh vertexes. Texture coordinates of points inside mesh cell are calculated as bi-linear combination of texture coordinates from corner cell points.

4 GPU Based 2D IBFV

This is a straightforward implementation of IBFV procedure with OpenGL library extension ARB_vertex_program [ATI 2003], [NVIDIA 2003], which enables user-defined per-vertex operations to be executed on graphic board. In terms of this extension a vertex is characterised by 16 standard attributes. Among them are vertex coordinate, weight, normal, texture coordinate, per-vertex colour and etc. All mentioned attributes can be modified during vertex program execution cycle. This modification take place within GPU and does not affect central processing unit.

The main point of this GPU based technique lay in dynamic texture coordinate estimation for mesh point p. It is extremely suitable to pass vector field value $V(p,t)$ in a vertex program as a normal vector. So, the input data for vertex shader is mesh point coordinate p and a mesh point normal, treated as value $V(p,t)$. The output data is texture coordinate, calculated during work of GPU program:

```
ATTRIB inPos = vertex.attrib[0];//point coordinate - p
ATTRIB inNrm = vertex.normal;//point normal-vector value V(p,t)

PARAM mvp[4] = { state.matrix.mvp}; // model-view matrix
PARAM h = { ... }; // time step value

OUTPUT out = result.position; // result position
OUTPUT tex = result.texcoord; // calculated texture coordinate

DP4 out.x, inPos, mvp[0]; // model-view transformation
DP4 out.y, inPos, mvp[1];
DP4 out.z, inPos, mvp[2];
DP4 out.w, inPos, mvp[3];

MAD tex,h,inNrm,in; // calculating texture coordinate
```

After mesh with texture mapped on it has been rasterized, the content of the frame buffer is blended with noise texture G to form convex combination. The result is temporarily stored in new texture object and is involved in next iteration as image F. Performing texture coordinate calculation on GPU-side permits to decrease CPU load during unsteady vector field visualization. Temporal flow field changes, in contrast with IBFV, do not result in mesh or texture coordinate recalculation. This approach enables stream pipeline visualization in case of unsteady vector field.

Within the framework on this method the software application called **fVis** has been written. This program implements both standard IBFV and GPU-based IBFV. User can interactively control visualization by adjusting key parameters of IBFV - number of noise textures G_k, convex combination parameter α and absolute global flow speed. During fVis maintenance incorrect blending operations of ATI Radeon-family card has been revealed. Artifacts, appeared on alpha-blending phase with RGB-textures and small values of alpha parameter, results in nondeterministic colour painting. To remedy this support for a single-channel textures in *GL_LUMINANCE* format was added in fVis. Using *GL_LUMINANCE* textures allows to avoid chaotic texture painting.

fVis provides front advancing visualization using texture advection. Front moves from the so-called seed line according to flow direction and speed, simulating a liquid matter injection. The example of visualization is shown on Fig. 3.

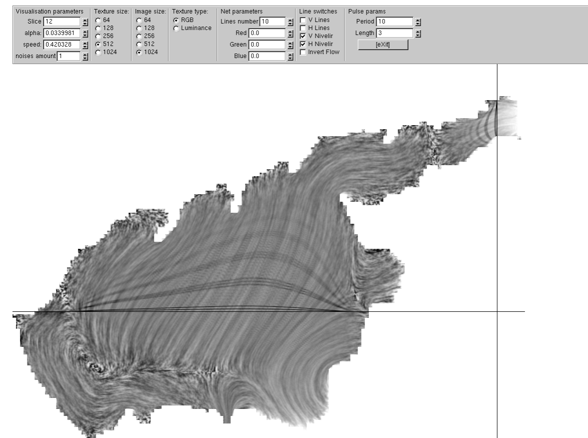

Figure 3: 2D GPU visualizations with front advancing.

5 GPU Based 3D IBFV

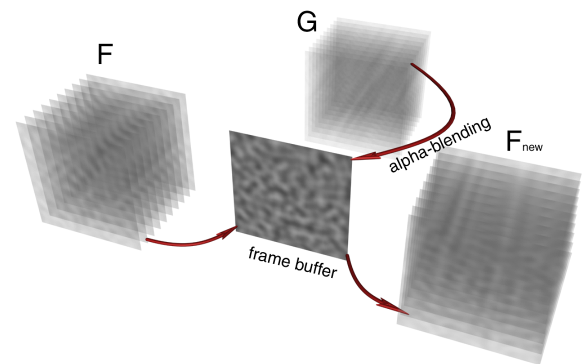

Figure 4: Stages of 3D GPU-based IBFV pipeline.

3D case requires textures with three dimensions, so, in contrast with 2D IBFV, we deal with texture cubes. An obvious solution for this problem is using OpenGL extension GL_EXT_texture3D, which implements 3D texture concept. Texture cube can be treated as 2D texture stack supplied by tri-linear interpolation - the texture value in cube is interpolated not only between 4 points of 2D texture slice, but among two adjacent slices. Imaging of final result is performed by volume rendering techniques [Hadwiger et al. 2002].

While generalizing our first method on three dimensional case we have faced the problem of α-blending image texture F with noise signal G. Current rasterization pipeline provides only planar frame buffer, thus making 3D α-blending impossible. To overcome this generic shortcoming we suggested performing 3D α-blending as series of independent 2D α-blendings. Treating 3D texture as a planar texture stack allows decomposition of texture cube on a number of 2D textures and performing α-blendings with each texture plane. Then the content of frame buffer is copied into appropriate stack position in a new 3D texture and this texture cube is involved

into next method iteration (Fig. 4). Consequently, each animation frame consists of two phases - texture stack advection and texture cube visualization.

Suggested way of graphic computations imply strong restrictions on GPU performance. Trilinear interpolation in cooperation with value fetching from 3D texture are very expensive operations, from the computation point of view. Using texture cubes requires enough room of memory on graphic boards. However, 3D texture exploiting leads to more accurate and precise visualization.

Vertex program for 3D flow visualization is the same as in 2D case. The only difference between them is the dimension of time step h parameter. In case of 3D IBFV - $h \in R^3$. The example of visualization is shown on Fig. 5.

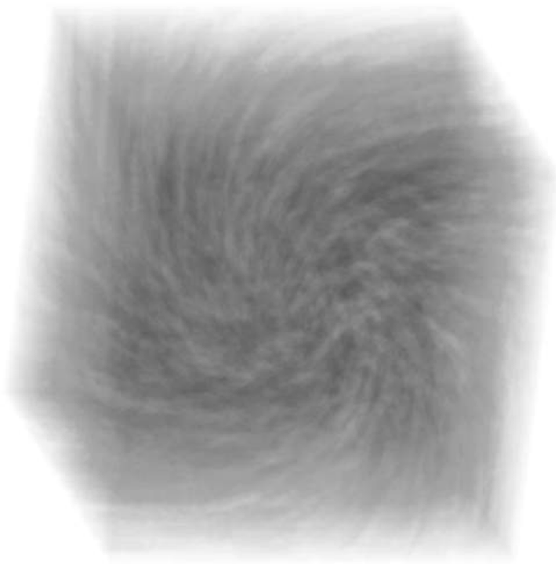

Figure 5: 3D GPU-based IBFV.

Method performance is approximately worse twice than standard 3D IBFV [Telea and van Wijk 2003]. This is caused by using 3D textures, instead of 2D texture stack in 3D IBFV. Cube textures provide tri-linear interpolation, which enables accurate volume rendering and texture mapping. GPU based 3D IBFV handles unsteady vector fields in contrast with 3D IBFV, thus making possible visualization of data stream.

6 Conclusion and Future Work

We have presented two techniques for unsteady 2D and 3D vector field visualization, exploiting graphic processing unit of a 3D accelerator. Work on problem of 2D GPU based method results in developing an application implementing this concept. During program maintenance two main visualization task classes were revealed - animated visualization of unsteady flow and front advancing simulation with static background showing stream lines. So, we suppose the case of 2D texture visualization is studied sufficient.

In case of 3D flow thoroughly code and algorithm optimization are required for better performance and cognition power. 3D IBFV implementation using OpenGL extension ARB_fragment_programm is the subject of special interest in this field. Orienting this technique on programmed rasterization phase we expect geometry reduction, thus making GPU based methods faster. Besides that, brightness and contrast correction of the resulting image is required [Engel and Ertl 2002]. This can be done, in the same way as texture advection, by the dependent texture reads technique employing fragment program.

We intend to continue our work on problem of GPU based visualization and create a program which will combine presented 2D and 3D GPU visualization techniques. This program will be used for flow visualization of field data obtained during numerical simulation of Azov Sea model at Laboratory of Computational Experiment on Supercomputer, Computer Center of Rostov State University.

Acknowledgements

We acknowledge the support of Ministry of Industry, Science and Technology of the Russian Federation under grant 37.011.11.0010, as well as the support of the President of the Russian Federation under grant MK1149.2003.01.

References

ANIKANOV, A. A., AND POTIY, O. A. 2003. Texture advection for 3d flow visualization. In *Proceedings of GRAPHICON 2003*, MSU, 1–6.

ATI, 2003. Ati opengl extension support.

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH 93,Computer Graphics Proceedings, Annual Conference Series*, ACM, vol. 4, 263–272.

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.

JOBARD, B., ERLEBACHER, G., AND HUSSAINI, M. Y. 2002. Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics 8*, 3, 211–222.

NVIDIA, 2003. Nvidia opengl extension specifications.

TELEA, A., AND VAN WIJK, J. J. 2003. 3d ibfv: Hardware-accelerated 3d flow visualization. In *Proceedings of ACM SIGGRAPH 2003*, ACM, 1–8.

VAN WIJK, J. J. 1991. Spot noise: Texture synthesis for data visualisation. *Computer Graphics 25*, 4, 309–318.

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