A COMPRESSION SCHEME FOR VOLUMETRIC ANIMATIONS OF RUNNING WATER*

Bedrich Benes
Vaclav Tesinsky
Tecnologico de Monterrey, Campus Ciudad de Mexico
bedrich.benes@itesm.mx
FEL CVUT Prague

Abstract Fluid animations are becoming a standard tool for computer animators. Simulation of turbulent gases, running water, eroded surfaces, or splashing waves are common, but still demanding because they are usually calculated in a voxel space. This brings new requirements to the tools that are used for such animations. The data structures are enormous but providing a good space and time coherency. We present a compression scheme that can be used for storing, accessing, and viewing such animations interactively. Key-frames are compressed by the RLE algorithm and in-betweens as difference frames. To display the scene we convert the level of water and the terrain surface to triangle meshes by the marching cubes algorithm. With this lossless technique we reach compression factor up to 1:100. Scenes can be decompressed fast, can be displayed, and manipulated interactively.

1. Introduction

Computer animations of running water and turbulent gases are in the focus of the computer graphics community for a long time. The interest has moved from ad-hoc techniques to the physically correct solution of the Navier-Stokes equations that provide a complete simulation of the motion of a liquid. The landmark paper in this area is the work of Foster and Metaxas, 1996. They introduced a simplified, but still physically acceptable, solution of the Navier-Stokes equations and demonstrated its usability for simulation of running water. The later works focused the practical aspects of the animations Foster and Fedkiw, 2001 and the photorealistic animations Enright et al., 2001.

*This is a slightly extended version of the paper.

The common property of these techniques is that they work in a voxel space, albeit they use particles to display the level of water. Previewing, manipulating, storing, etc. of such scenes is difficult and causes a huge demand to the software and the hardware. Techniques that facilitate the work are necessary.

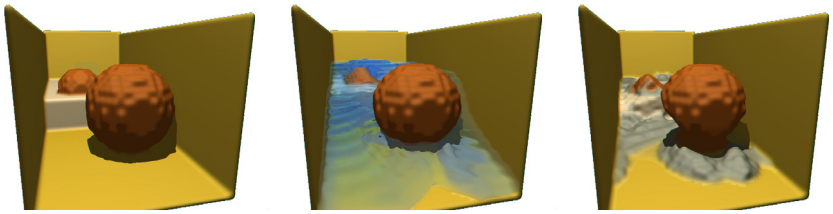

Figure 1. Results of erosion simulation of an artificial river. The left two images show the erosion process, the last frame displays only the deposited material

We present a compression scheme that is specially designed for volumetric animations of running water. Animations are stored in a form that facilitates fast displaying and scene manipulations. The user can see interesting details, preview some areas in time, etc. The simulation can be stopped and rerun from any point. We use this scheme to simulate hydraulic erosion. These simulations are computationally very demanding so the results should be saved in some reasonable way for an analysis.

Our scheme is based on key framing. Reference scenes are compressed by the Run Length Encoding (RLE). The in-betweens are saved using differential scheme allowing for an efficient scrolling forward and backward in the animation. Since we want to display the water and the terrain, the animations must display the surfaces. We detect them by the marching-cubes algorithm and convert them into the boundary representation. The surfaces are stored as triangle meshes that are efficiently displayed by a graphics hardware. This kind of displaying results in a scene preview. The photorealistic rendering, such as in Figure 1, is provided by a specialized raytracer that is launched as an external application.

This scheme facilitates not only displaying the simulation results but also their reuse. We store the complete information about the simulation; the pressure field, the velocity field, states of the cells, etc. This allows us to rerun previously stopped animation, going back and changing some parameters, etc.

2. Previous Work

An application for displaying volumetric medical data was introduced by Avila et al., 1994. The `VolVis` system become quite famous in the area of medical data displaying namely for its integration of various displaying algorithms and techniques. The system supports different three-dimensional input data that

can be displayed as rough data, compression domain rendering, volumetric ray-tracing, and irregular grid rendering among others.

cker Chiueh et al., 1997 present an integrated compression and visualization scheme that displays compressed volumetric scenes without actually decompressing them. The compression is performed in the Fourier domain and therefore its primer application area is a static scene. This technique is suitable for displaying medical data rather than dynamic scenes obtained by algorithms of fluid dynamics.

A hardware assisted rendering approach was introduced by Lum et al., 2001. They use the hardware support for texture displaying to render a time varying volumetric data. The two different compression schemes are used: the palette based encoding and the temporal encoding. The latter is based on the DCT.

A wavelet-based compression scheme for time varying volumetric data was described by Guthe and Straßer, 2001. They use lossy compression scheme with coding frames similar to the MPEG compression.

Rosa et al., 2003 presented an approach that is closely related to ours. They show a system for interactive displaying and manipulating time-varying data of thermal flow simulations. The main difference to our paper is that to compress a single scene they exploit the hardware support for indexing three-dimensional textures. The scene-to-scene coherency is coded by the DCT. The advantage of the method is that they use a special kind of coding that can be displayed by trilinear filtering that makes the method really efficient.

3. Compression Scheme

The application generates huge amounts of voxel data. For example one simulation of a voxels space in the resolution 300^3 with 700 frames occupies, in the uncompressed form, more than 300GB of the disk space.

The compression scheme is designed to fulfill two goals. First, the compression and decompression must be lossless. We want to use it not only for storing an animation, but also for rendering, and, the most important, for rerunning the simulation from a desired point. The second goal we need is the ability to scroll fast through the data. We want to move forward and backward in time as fast as possible and we want to display the data correspondingly. We exploit two important properties of the volumetric data. The data has high spatial coherency and time coherency. The first is used for coding the key-frames (we should say "key-scenes"), the latter is used for storing the in-betweens. This scheme is depicted in Figure 2.

A property of the volumetric data is that the majority of the values are clamped to the interval $[0, 1]$ so we can easily store them as the fixed-point representation. We multiply the data by `0xFFFF` and store them as 16-bits inte-

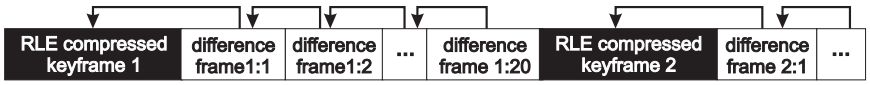

Figure 2. Compressed key frames and difference frames are stored

gers. In this way only the data that is used for direct displaying is stored. High precision is required for the data that is used for rerunning the simulation.

Key-frames exploit the high spatial coherency of the volumetric data. The are just few types of material, water, and air that are distributed in the more or less continuous areas. The most complicated are the boundaries of different environments; the deposited material, and the level of water. The spatial data coherency makes the data a candidate for the Run Length Encoding (RLE).

Each sequences of the same value is coded as a pair $[n, v]$, where n is the number of repetitions and v is the value itself. If there is a sequence of highly varying values they are stored in the uncompressed form. The value zero is used to indicate the begin and the end of an uncompressed sequence. For example the sequence 000001234999999 will be coded as 5001234069.

To perform the compression/decompression efficiently the voxel structure is taken as a linear array i.e., in the way it is stored in the memory that corresponds to scanning sequentially the rows of the first layer, then skipping into the second one, etc. Testing this compression scheme on various scenes gives an average compression factor 1:30.

The key-frame is stored every twenty frames. With the increasing distance of the key-frames also increases the compression factor but storing few key-frames makes the scrolling difficult. We have found the twenty frames as the good compromise between the compression quality and the interactivity.

In-betweens exploit the time coherency of the sequences. The two successive scenes do not vary very much. The scenes in-between the keyframes are stored as the difference frames relatively to the previous frame (see Figure 2).

The compression is asymmetrical i.e., the compression takes longer time than the decompression. When compressing we take the previous frame, denoted by A and the new frame B in the same form as in the previous case, i.e., as a long sequence of values. All equal values in the second array are simply skipped. We store the pair $[-, n]$ where $-$ indicates that we are skipping the data and the n is the number of bytes to skip. If there is a difference between the scenes we store the exact value.

We have measured the compression factor of the in-betweens of different scenes and the average value is around 1:400. The overall compression factor of the animations is around 1:100.

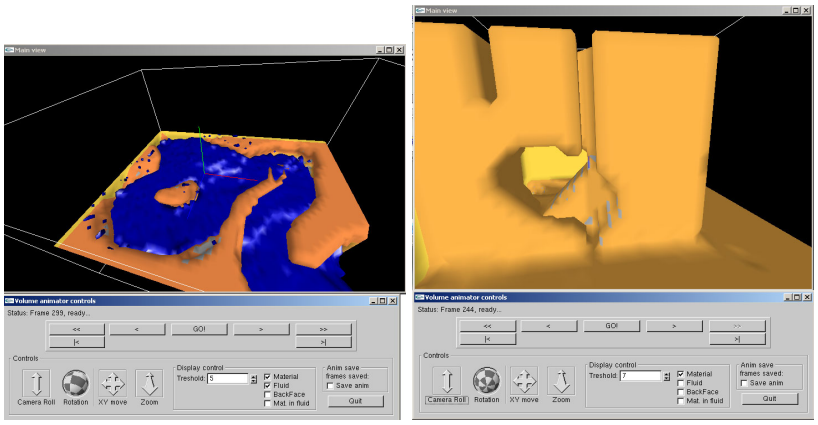

Figure 3. The simulator

4. Implementation and Results

We have developed an application that we use for displaying and manipulating the volumetric scenes that result from our erosion simulation algorithm. The application is written in $\mathcal{C}$ under OpenGL. The application allows zooming, scrolling the animation forward and backward by key-frames, saving the preview, manipulating parameters of displaying, changing the transparency, disabling and enabling materials, and spawning the simulator or the raytracer. The application window snapshots are in Figure 3.

The system first decodes the first key-frame and the scene is stored in the memory. The actual scene can be decompressed from a key-frame or an in-between. In the latter case, the scene must be calculated from the closest previous frame and the actual difference frame. When the scene is calculated the additional data structure representing the surfaces boundaries is also generated. We use the marching cubes algorithm that generates the mesh of triangles (polygon soup). The scenes are small, we store it as OpenGL display lists and this makes any interactive manipulation immediate.

One objection is that at the moment we want to see an animation we have to do a lot of work for each frame. The scene must be decoded, uncompressed, and the additional data structure must be generated. Keeping this in mind, we have experimented with storing the meshes together with the volumetric data, keeping the meshes in a LRU cache, etc. At the end we have found that the solution of generating the data every time it is necessary is satisfactory. For greater scenes, it would be definitively good to store the mesh in the volumetric file. For small scenes, the biggest scene we use is 400^3 voxels, this solution was fast enough.

We have tested the entire system on a 3GHz IBM PC equipped with the NVIDIA GeForce 3 graphics card, 1.5GB of main memory, and SCSI hard discs. The response of the system was always immediate. Please note that there is an additional video showing the experiments and the interaction with the system.

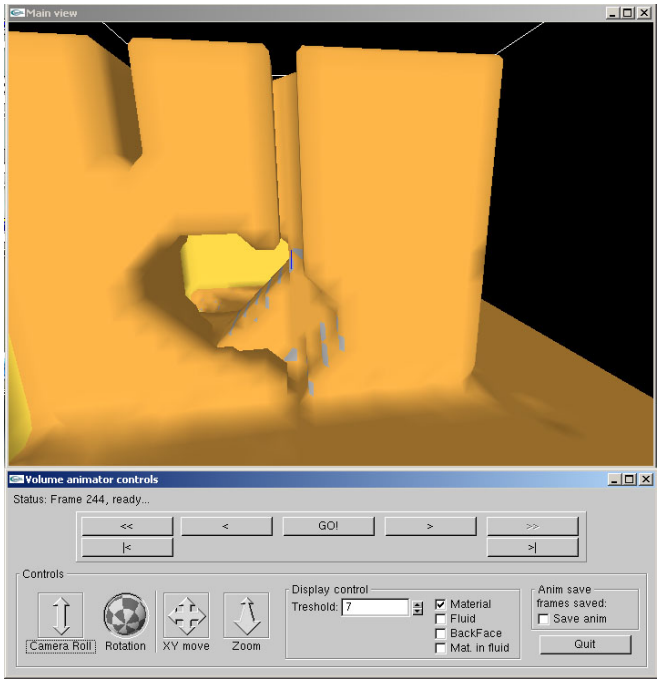

Figure 4. Another data displayed by the simulator

5. Conclusions

A compression scheme for storing volumetric data is presented. The main purpose is to display and reuse the volumetric data produced by the fluid and the erosion simulation. We use key-framing and difference frames coding. The scene that corresponds to a key-frame is compressed using RLE and the in-betweens are compressed by storing only the difference values.

We exploit the spatial and the time coherency. The data does not vary very much in space and this helps us to reach compression factor of about 1:30 for key-frames. The changes between the frames are also low and this allows us to compress the in-betweens with the compression about 1:400. The overall animation compression factor is 1:100.

To display a scene we convert the data into the sets of triangles by the marching cubes algorithm. The meshes are not stored together with the volumetric data, because the speed of decompression is high enough to give a good response of the entire system. We were exploring scenes with size around 0.5-1GB in the compressed form that corresponds to scenes with the uncompressed size up to 100GB.

There are many things that could be done as the future work. The main objection is that this compression scheme is well-suited for one kind application. Could we find some general-purpose compression scheme that would compress any fluid-dynamics simulation? What could be the limitations? What would happened with the compression if the material boundaries in the scene are wilder?

Another potential issue is the marching cubes algorithm. The resulting scenes are good for preview but some kind photorealistic rendering must be run to get visually plausible scenes. There are techniques exploiting scene-to-scene coherency for displaying volumetric data by ray casting Shen and Johnson, 1994. It would be interesting to integrate both approaches.

We also do not benefit from the fact that the up-to-date graphics hardware usually has a good support for three-dimensional textures. It would be interesting to explore approaches that are using it Lum et al., 2001 and integrate them with our method.

Another possible future work is to use hardware-assisted rendering to display the volumetric data directly on the computer's GPU.

Acknowledgments

We would like to express our gratitude to the students from the Czech Technical University in Prague that were in charge with some parts and the first tries of the project. Our thanks belongs to Jan Hornyš who is implementing the raytracer and to René Trbušek who has implemented the first version of the fluid simulator and showed us the way we should not go.

References

Avila, Ricardo, He, Taosong, Hong, Lichan, Kaufman, Arie, Pfister, Hanspeter, Silva, Claudio, Sobierajski, Lisa, and Wang, Sidney (1994). Volvis: a diversified volume visualization system. In *Proceedings of the conference on Visualization '94*, pages 31–38. IEEE Computer Society Press.

cker Chiueh, Tzi, kai Yang, Chuan, He, Taosong, Pfister, Hanspeter, and Kaufman, Arie (1997). Integrated volume compression and visualization. In *Proceedings of the 8th conference on Visualization '97*, pages 329–ff. IEEE Computer Society Press.

Enright, Douglas, Marchner, Stephen, and Fedkiw, Ronald (2001). Animation and rendering complex water surfaces. In Hughes, John F., editor, *Proceedings of SIGGRAPH 2002*, Com-

puter Graphics Proceedings, Annual Conference Series, pages 736–744. ACM, ACM Press / ACM SIGGRAPH.

Foster, Nick and Fedkiw, Ronald (2001). Practical animation of liquids. In Fiume, Eugene, editor, *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 23–30. ACM, ACM Press / ACM SIGGRAPH.

Foster, Nick and Metaxas, Dimitri (1996). Realistic animation of liquids. In *Proceedings of Graphical Models and Image Proceedings*, volume 58(5), pages 471–483.

Guthe, Stefan and Straßer, Wolfgang (2001). Real-time decompression and visualization of animated volume data. In *Proceedings of the conference on Visualization '01*, pages 349–356. IEEE Computer Society.

Lum, Eric B., Ma, Kwan Liu, and Clyne, John (2001). Texture hardware assisted rendering of time-varying volume data. In *Proceedings of the conference on Visualization '01*, pages 263–270. IEEE Computer Society.

Rosa, Gabriel G., Lum, Eric B., Ma, Kwan-Liu, and Ono, Kenji (2003). An interactive volume visualization system for transient flow analysis. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 137–144. ACM Press.

Shen, Han-Wei and Johnson, Christopher R. (1994). Differential volume rendering: a fast volume visualization technique for flow animation. In *Proceedings of the conference on Visualization '94*, pages 180–187. IEEE Computer Society Press.