On the Visualization of Time-Varying Structured Grids Using a 3D Warp Texture

Yuan Chen Jonathan Cohen Subodh Kumar

Johns Hopkins University

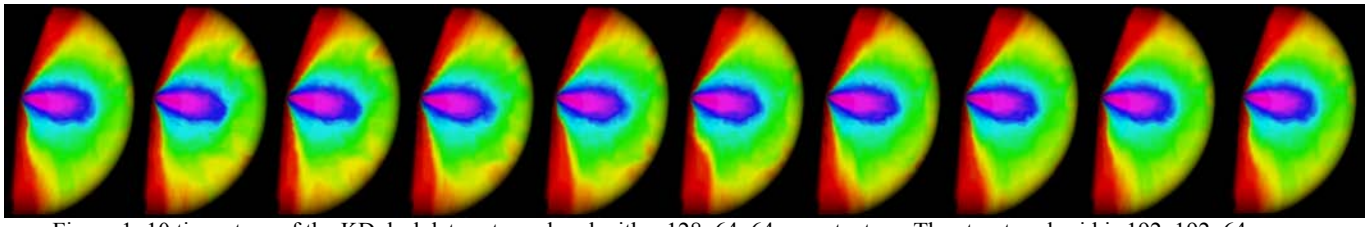

Figure 1: 10 time steps of the KDphrd data set, rendered with a 128x64x64 warp texture. The structured grid is 192x192x64.

Abstract

We present a novel scheme to interactively visualize time-varying scalar fields defined on a structured grid. The underlying approach is to maximize the use of current graphics hardware by using 3D texture mapping. This approach commonly suffers from an expensive voxelization of each time-step as well as from large size of the voxel array approximating each step.

Hence, in our scheme, instead of explicitly voxelizing each scalar field, we directly store each time-step as a three dimensional texture in its native form. We create the function that warps a voxel grid into the given structured grid. At rendering time, we reconstruct the function at each pixel using hardware-based trilinear interpolation. The resulting coordinates allow us to compute the scalar value at this pixel using a second texture lookup.

For fixed grids, the function remains constant across time-steps and only the scalar field table needs to be re-loaded as a texture. Our new approach achieves excellent performance with relatively low texture memory requirements and low approximation error.

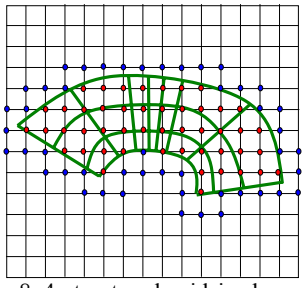

Figure 2: An 8x4 structured grid is shown in green in the Euclidean domain. The R^3 grid is shown in gray. Red samples are centers of interior voxels, blue samples that of boundary voxels, and unmarked samples that of exterior voxels.

1 INTRODUCTION

A common representation for volumetric data in the form of 3D scalar fields is the structured grid. The structured grid is topologically a uniform, Cartesian grid, but is geometrically warped into R^3, a Euclidean space, for visualization. For example, curvilinear grids specify a warped position for each data point and assume some interpolation function for locations between the data points. Although structured grids may be rendered using customized ray tracing and cell projection techniques as well as using resampling, it is quite common to discard much of the structure by tetrahedralizing the grid cells and rendering as an unstructured grid. This is exemplified by the fact that many papers describing unstructured grid rendering actually use test data that originated as structured grids (such as NASA's blunt fin and delta wing models).

Resampling the scalar data at each time-step into a regular R^3 volume not only dramatically increases the total volume size and the processing time, but also potentially introduces unnecessary resampling artifacts. We propose a novel approach that still combines regular sampling with 3D texture-based volume rendering, but with an important difference from standard resampling approaches – we do not resample the scalar field. Our approach leverages the regular topological structure of the grid to provide a natural mapping to graphics hardware. We realize that the structured data in fact lies on a rectilinear lattice in some warped space, just not in R^3 (see Figure 2 for a 2D example). Let us denote this space by triple $\langle s, t, r \rangle$. The scalar value is sampled on a lattice: at regular intervals in s, t, and r. Indeed, a structured grid is often specified by a three dimensional array of scalar values along with an unwarping function, or a table, that specifies the R^3 location of each lattice point. Thus the time varying scalar data may be directly stored as a 3D texture in that warped space – call it the grid texture. Of course, this texture is not regular in R^3 and hence may not be directly rendered as textured slices. Instead, we use a regular R^3 grid to sample the inverse of the structured grid's unwarping function. The resulting warping texture effectively serves as a voxel-based parameterization of R^3. Now we can find the scalar value at any point in R^3, denoted by $\langle u, v, w \rangle$, by first looking up the warp texture $(s, t, r) = \mathbf{W}(u, v, w)$ to find its 3D warped coordinate, and then looking up the scalar value from the grid texture: $S(s, t, r)$.

This indirect texturing approach has some inherent advantages over a standard resampling algorithm. First, for Eulerian grids, which have a constant warping function over all time steps, we create a single warping texture, **W**, to describe the parameterization of space, and re-use it for all the time-varying scalar values. Second, the scalar data itself remains as compact as in its original form, and thus requires minimal bandwidth to load to texture memory and a minimal footprint to store there. Third, the warping texture often requires less resolution to maintain good quality rendering than does a direct resampling of the scalar texture. This is due to the fact that warping functions are often largely smooth and well-approximated by the tri-linear interpolation performed on the hardware. Scalars tend to have higher frequency content and many discontinuities. Furthermore, by avoiding directly resampling the scalar field, we eliminate one resampling of this data, delaying its resampling to the stage of rendering individual pixels. Any error induced by sampling the warping function has the effect of distorting the data in space rather than changing the actual scalar values portrayed. Our technique is thus effective not

only to reduce the data size for time-varying scalar fields, but also for static scalar fields as compared to direct resampling.

2 WARP TEXTURE GENERATION

The essential data component for our rendering algorithm is the warp texture, **W**, a voxel-based approximation to the inverse of the structured grid's unwarping function, $\mathbf{W}_i$. Thus, each value in the warp texture describes a mapping back into the grid (scalar) texture domain and may be applied as texture coordinates to look up interpolated values from the original scalar data.

The generation of the warp texture is primarily a sampling process. The warp texture is defined to cover the bounding box of the structured grid unwarped into the R^3 domain. (We refer to the cells of a regular rectilinear R^3 grid also as voxels.) Given some R^3 grid resolution, a voxel may be classified as interior, boundary, or exterior, according to their relationship with the warped cells of the structured grid, as shown in Figure 2.

To compute the values of **W** at the centers of the interior voxels, we perform a point location query of each voxel sample (its center) within the cells of the structured grid. Such a query should report which cell, if any, contains the sample location, and where the sample is within that cell (i.e. its $<s,t,r>$ coordinate). If there is such a cell, then the voxel is an interior voxel, and the matching grid coordinates are stored at that voxel (such as the red samples in Figure 2). At the centers of these interior voxels, the warping error is effectively zero.

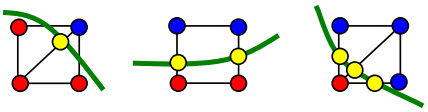

Figure 3: Three cases of boundary matching (in 2D). The blue boundary voxels are computed using extrapolation so as to match the warping function at the yellow points along the structured grid boundary.

For boundary voxels, we assign the warping value using an extrapolation process. This extrapolation ensures an exact warp at some place along the boundary of the structured grid, as shown in Figure 3. These boundary voxels are mapped to somewhere outside the valid [0,1] domain of the structured grid, and at rendering time all pixels which map outside the domain are rendered as transparent black.

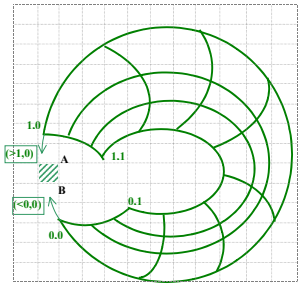

Figure 4: It is possible for a voxel to be both above and below the structured grid (mapped into R^3), resulting in extraneous grid being generated. This problem is avoided using a stencil texture.

In general, it is not always possible to assign consistent warping values everywhere outside the domain. As illustrated in Figure 4, it is possible for some voxels outside the domain in the positive direction to neighbor voxels outside the domain in the negative direction. Trilinear interpolation in this case would generate values within the domain for pixels within these voxels. We employ a 3D stencil texture to eliminate such ambiguities. This approach suffices so long as the warping texture resolution is high enough to allow at least one exterior voxel between nearby boundaries.

3 ERROR COMPUTATION

Apart from numerical errors, the main source of error in our technique is the approximate reconstruction of the warp-function from the samples at voxel centers. In order to measure this error, we compute the difference of the interpolant from the actual warp function at each point of R^3. Since, in general, we only know the samples of the unwarp function at the structured grid lattice points, it is sufficient to find the error at locations that these lattice points map to in R^3. For each such point, p_g, we are given its mapping into R^3, p_e. We then evaluate the warping texture at $W(p_e)$ using tri-linear interpolation of neighboring voxel centers. The error incurred at p_g, hence, is $|W(p_e) - p_g|$. We can then compute statistics such as the maximum, or root mean square error over all grid sample points.

4 TIME-VARYING PLAYBACK

By design, managing the playback of multiple time steps is straightforward using our rendering algorithm. Because structured grids adaptively sample R^3, the grid resolution is often small enough that we can fit many time steps in video memory. In this case, we just load all the time steps for the desired sequence and advance through them by rebinding only the scalar grid texture each frame (or every k frames, to slow down playback speed with respect to some faster camera motion).

If the desired time steps are too large to fit in video memory at once, we can still swap textures once per time step and the frame rate may be limited by the memory bandwidth to the graphics card. We can also employ some (possibly lossy) texture compression scheme.

5 IMPLEMENTATION

We have implemented the warp texture generation algorithm as well as the rendering algorithm on a Linux PC equipped with 1.6 GHz AMD Athlon CPU, 1 GB RAM, and an NVIDIA GeforceFX 5800 Ultra with 128 MB VRAM. 3D texture-based volume rendering is performed with viewport-aligned slicing. A custom fragment program uses the R^3 location as an index into the warp and stencil texture. The results of the warp texture lookups are scaled and biased, then used to perform the scalar texture lookup. The scalar texture, which has been quantized to one byte per scalar, is then used to index a final transfer function texture, which maps the scalar value to RGBA according to a user-specified transfer function. If the stencil value equals 0, or the scalar texture coordinates are out of range, the fragment is instead colored as transparent black. For all our tests the texture sizes are small enough that the rendering remains highly interactive (over 15 fps). Playbacks of time-steps are similarly interactive.

6 CONCLUSIONS

We have discussed a new 3D texture-based algorithm for visualization of time-varying structured grids. By factoring a static warping function out from the time-varying scalar data, we eliminate the need to resample the scalar data for each time step and maintain a relatively low space requirement for storing and transmitting the data.

ACKNOWLEDGMENTS

We would like to thank Julian Krolik and Shigenobu Hirose for providing the astronomical data which motivates this work, and Siddharth Sonrexa and Joshua Leven for their work on some of the implementation modules. This work was supported in part by NSF ITR Grant AST-0313031.