DOI: 10.1111/cgf.12135
Eurographics Conference on Visualization (EuroVis) 2013
B. Preim, P. Rheingans, and H. Theisel
(Guest Editors)

Volume 32 (2013), Number 3

Interactive Ray Casting of Geodesic Grids

Jinrong Xie[1], Hongfeng Yu[2] and Kwan-Liu Ma[1]

[1]University of California-Davis, USA
[2]University of Nebraska-Lincoln, USA

Abstract

Geodesic grids are commonly used to model the surface of a sphere and are widely applied in numerical simulations of geoscience applications. These applications range from biodiversity, to climate change and to ocean circulation. Direct volume rendering of scalar fields defined on a geodesic grid facilitates scientists in visually understanding their large scale data. Previous solutions requiring to first transform the geodesic grid into another grid structure (e.g., hexahedral or tetrahedral grid) for using graphics hardware are not acceptable for large data, because such approaches incur significant computing and storage overhead. In this paper, we present a new method for efficient ray casting of geodesic girds by leveraging the power of Graphics Processing Units (GPUs). A geodesic grid can be directly fetched from storage or streamed from simulations to the rendering stage without the need of any intermediate grid transformation. We have designed and implemented a new analytic scheme to efficiently perform value interpolation for ray integration and gradient calculations for lighting. This scheme offers a more cost-effective rendering solution over the existing direct rendering approach. We demonstrate the effectiveness of our rendering solution using real-world geoscience data.

1 Introduction

Climatological and environmental studies are critical for understanding and predicting important natural phenomena, such as glacier shrinking and sea-level rising. Climate research has been internationally recognized as one of the most crucial missions with the largest investment for computational and mathematical analysis. For example, the U.S. Department of Energy aims to enable exascale computing capabilities to model climate at a resolution of roughly 10 meters over the entire Earth [Lev10]. By leveraging the unprecedented power of supercomputer, scientists can simulate the highest-resolution climate/Earth system models, leading to more reliable numerical weather prediction.

1.1 Spherical Geodesic Grids

Geodesic grids, first introduced by Williamson [Wil68] and Sadourny et al. [SAM68] for meteorological applications in 1968, have been rediscovered widely to model the Earth's surface for various applications of climate modeling in the recent years [RRH*02, AC07, WWF09, CBX12]. The advantage of geodesic grids is particularly evident by their suitability for large-scale numerical simulations [MLP*02, MUS07]. The isotropic structure of geodesic grids can completely avoid the well-known Pole problem and the expensive filtering operations present in conventional latitude-longitude grids (e.g., curvilinear hexahedral grids for spherical shells). The hexagonal-triangular duality of geodesic grids allows scientists to design discrete operators and upgrade scales to higher resolutions in a more flexible manner. In addition, the data structure of geodesic grids is extremely well suited for scientists to parallelize numerical solvers and achieve high efficiency on large distributed memory parallel computers.

The construction of a geodesic grid typically begins with an icosahedron inside a sphere, as shown in Figure 1 (a1). We can then subdivide the icosahedron to generate the refined grid by iteratively applying a bisection operation on the triangles [RRH*02], as shown in Figure 1 (a2) and (a3). This subdivision process can be repeated until the desired resolution is obtained. Each grid point has 6 nearest neighbors, except for the original 12 icosahedral points where each point only has 5 neighbors. These 12 special points are referred to as *pentagonal points*. The triangles define a Delaunay grid, and we can further join the centroids of neighboring triangles to construct Voronoi polygons. As shown in Figure 1 (b), the dashed lines represent the Delaunay triangles, and the solid lines represent the Voronoi polygons. The Voronoi polygons are pentagons with respect to the pentagonal points, and are hexagons with respect to the other points. For a triangular

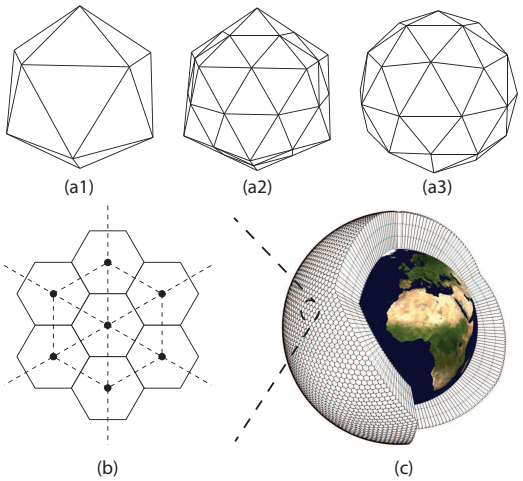


Figure 1: *(a1-a3): The first iteration of subdivision to construct a geodesic grid. (a1) shows the initial icosahedron with 12 vertices, 30 edges, and 20 equal spherical triangles. (a2) shows that each triangle is subdivided through bisection. (a3) shows the new vertices are popped out onto the sphere. (b) shows the duality of Voronoi polygon and Delaunay triangulation. (c) shows the structure of a 3D spherical geodesic grid.*

grid, each grid cell has some neighbors across the cell walls (referred to as *wall neighbors*) whereas others across the cell vertices (referred to as *vertex neighbors*). The different types of neighbors incur unavoidable asymmetries for the finite-difference operators on triangular grids. Fortunately, for the grid constructed from Voronoi polygons, each cell's neighbors are all wall neighbors, thus allowing the finite-difference operators to treat all neighboring cells in the same way and become as symmetrical and isotropic as possible. Therefore, a geodesic grid defined on a spherical surface is composed of Voronoi polygons, which contains the information of each polygon's vertices, edges, and center point. The data values are stored at the center points of polygonal cells. To model ocean or atmosphere in 3D, the spherical polygon mesh is scaled and duplicated to construct a set of spherical layers perpendicular to the Earth's surface, and each layer corresponds to a different distance value to the Earth's surface, as shown in Figure 1 (c).

1.2 Visualization of Geodesic Grid Data

Visualization has proven to be an effective tool to help scientists study their large and complex data. Although a variety of sophisticated visualization algorithms have been designed for structured and unstructured grids, comprehensive work on visualization of geodesic grids is surprisingly less discussed in the literature, possibly due to the relatively short history of large-scale applications of this grid type in practice. With the fast growth of geodesic grid applications, new techniques and tools are highly desirable to enable effective analysis and visualization of high-resolution data generated from large climate simulations.

However, visualization of large geodesic grid data imposes some unique challenges. First, the data structure of geodesic grids is constructed using a recursive refinement procedure on a spherical surface, which presents very different geometry properties from other existing unstructured grids. Therefore, it is usually not applicable to use conventional tools to visualize geodesic grid data directly without any remeshing operations. Second, even though it is possible to transform geodesic grids into more generally supported grids, such as tetrahedral grids, for visualization, this approach often incurs significant computing and storage overhead, and becomes infeasible to process large data from current tera/petascale and future exascale simulations.

We introduce a viable solution to generate high-quality visualization of geodesic grid data in full extent at interactive rates. This is achieved by minimizing the data movement and preprocessing before carrying out visualization operation. Our solution has the potential to support in-situ visualization with simulations on geodesic grids. It will enable scientists to monitor simulations and possibly capture important features during the simulation runs. Moreover, exascale computers are expected to be mostly heterogeneous systems with specialized processors, deep memory hierarchies, and high levels of concurrency [SDM10]. We also need to consider the easy applicability of our solution to such platforms.

To the best of our knowledge, we present the first design and implementation of interactive ray casting of geodesic grid volume data with almost no overhead. Our work makes the following contributions:

- By carefully studying the properties of geodesic grids, we present a novel scheme to march rays in an efficient way without reconstructing full connectivity information in 3D, and thus avoid memory and computing overheads.
- We derive an analytic solution for the interpolation of scalar values along a ray within a geodesic grid cell, and achieve high quality rendering with adaptive sampling.
- We represent geodesic grids in GPU memory in a form that best matches the original storage format and minimize the data transformation overhead.

We evaluate our approach with two data sets of real-world climate simulations. Our approach can generate high-quality visualization of full resolution geodesic grid data, and allow scientists to navigate in great details at an interactive or nearly interactive rate. Moreover, our approach directly takes original data as input, allowing for sharing data structures with simulations and performing visualization during simulation time on supercomputers.

2 Related Work

Volume rendering of unstructured grids are commonly based on either cell-projection or ray-casting. Shirley et al. [ST90]

introduced one of the first cell-projection algorithms, Projected Tetrahedra, in 1990. Since it needs visibility ordering of the cells, this algorithm can be very expensive for a large amount of cells. A set of optimization techniques have been developed to accelerate sorting [Wil92, SBM94, KE01] or avoid sorting [WKME03b].

Ray-casting is another popular method for rendering unstructured grids [Gar90]. It can avoid the costly visibility sorting; however, the mesh connectivity information is usually needed for mesh traversal. Various approaches have been presented to accelerate ray-casting of tetrahedral grids using graphics hardware [WKME03a, WMKE04, MRB*08], with a particular focus on the representations of connectivity information for memory efficiency.

Cell-projection and ray-casting methods have been extended to render more general polyhedral grids. For example, Bennett et al. [BCM*01] presented a parallel implementation of cell-projection for meshes with arbitrary polyhedra. Muigg et al. [MHDH07] presented a hybrid scheme for rendering unstructured grids based on importance classification. Muigg et al. [MHDG11] further introduced a new unstructured-grid representation which can minimize redundant connectivity information and improve grid traversal performance in graphics hardware.

Great efforts have been made to generate high-quality volume rendering of unstructured grids. Max et al. [MHC90] showed that with barycentric interpolation for signal reconstruction inside a tetrahedron, integration using sampling at the cell faces can generate the exact rendering results. Röttger et al. [RKE00] applied this idea to Projected Tetrahedra, and developed the pre-integrated volume rendering method to render isosurfaces using 3D texture hardware. Röttger et al. [RE02] further enhanced the pre-integration technique using 2D texture hardware and increased the performance of volume rendering. Moreland et al. [MA04] presented partial pre-integration to approximate the volume rendering integral with computational feasibility for tetrahedral grids. Lum et al. [LWM04] improved the pre-integration technique by speeding up lookup table generation and minimizing lighting artifacts. However, Marchesin et al. [MdV09] showed that pre-integration is less suitable for volume rendering of AMR data with hexahedral cells. They derived an analytical function for signal reconstruction inside a hexahedron, and used the gradient of the function for shading. Apart from rectilinear grids, Finkbeiner et al. [FEVM10] demonstrated interactive volume rendering on the body centered cubic lattice using box splines.

3 Ray-casting Framework

Although visualization of general tetrahedral grids and polyhedral grids are well covered in the literature, there is a lack of comprehensive visualization work on geodesic grids. A simple approach to visualizing geodesic grids is to first decompose a cell into a set of tetrahedrons, and then to render the tetrahedral grids using conventional tools as the ones discussed in Section 2. However, this simple approach often incurs significant computing and storage overhead for large data. While it is also possible to treat geodesic grids as polyhedral grids, this scheme either requires visibility ordering of all cells or constructing 3D connectivity information for mesh traversal, and requires considerable overhead as well. Alternatively, our approach directly takes geodesic grids as input, performs mesh traversal without constructing full 3D connectivity information, and thus can significantly reduce memory footprint and improve computing efficiency. Similar to Marchesin et al. [MdV09], we also develop an analytic solution to compute ray integral within grid cells and generate high-quality rendering of geodesic grid data.

3.1 Grid Representation

A viable grid representation is essential for efficient mesh traversal. A typical representation organizes mesh connectivity information with a focus on memory efficiency, which is the key for the overall ray-casting performance. For a general geodesic grid data set, because each layer has the same Voronoi polygonal mesh structure as the outer spherical surface, scientists only save the 2D connectivity information of the outer surface in data. Intuitively, we can simply construct the 3D cells of hexagonal or pentagonal frustum by connecting the corresponding vertices on every two neighboring spherical layers. However, this approach needs considerable computing and memory consumption.

Our design is based on two important properties of geodesic grids. First, a geodesic grid has the duality of Voronoi-Delaunay, and data values are stored at the center points of hexagonal or pentagonal cells, as discussed in Section 1.1. To compute the data value at any other spatial location, first we need to identify the triangular frustum that contains the location, and use interpolation to compute the needed value according to the values stored at the frustum vertices. We observe that although triangular grid connectivity information is not directly provided, the Voronoi-Delaunay duality allows us to construct and identify triangular frusta from the Voronoi polygonal grids without explicitly reconstructing the full triangular grids. Second, a geodesic grid is also radially symmetric. This means if the side edges of an arbitrary frustum cell are extended along the direction of radius, they all converge on the sphere center. Given this property, we can implicitly construct an intermediate 3D frustum cell during mesh traversal according to the 2D mesh structure of the outer surface and the distance values of layers.

Given these two properties, we represent a 2D Voronoi polygonal grid into six tables. In our representation, a polygonal cell is named a *cell*, a vertex of a cell is named a *corner*, the center of a cell is called a *center*, and a wall of a cell is called an *edge*.

- **center** table: Each entry of the table contains the longitude and latitude values of a cell center.

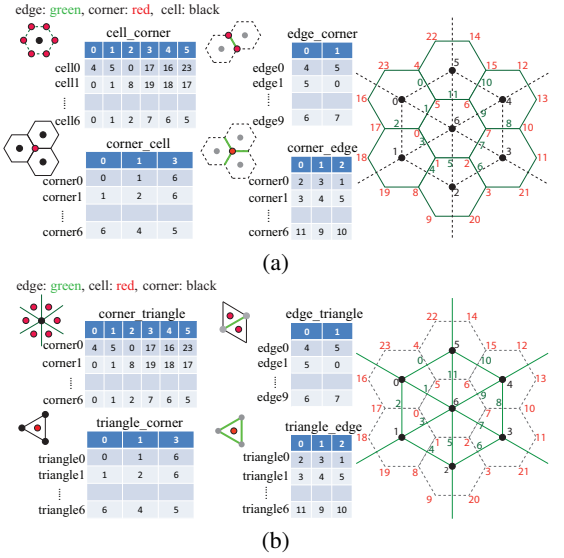


Figure 2: *(a): The* cell_corner, corner_edge, corner_cell, *and* edge_corner *tables used to represent a 2D Voronoi polygonal grid of climate simulations. (b): The representation of dual triangular mesh of (a). We note that the names of tables and table entries are modified for an illustration purpose in (b), and they are not changed in the real implementation. Thus, from (a) to (b), the content of the four tables keeps intact, which implies that no explicit grid transformation is required.*

- **corner** table: Each entry of the table contains the longitude and latitude values of a corner.
- **cell_corner** table: Each entry of the table contains six indices of cell corners in the **corner** table with respect to a cell. If two indices share a same value, this entry indicates a pentagonal cell.
- **corner_cell** table: Each entry of the table contains three indices of the cells in the **cell_corner** table with respect to a corner, and these three cells share the corner.
- **edge_corner** table: Each entry of the table contains two indices of the corners in the **corner_cell** table with respect to an edge, and these two corners share the edge.
- **corner_edge** table: Each entry of the table contains three indices of the edges in the **edge_corner** table with respect to a corner, and these three edges share the corner.

Figure 2 (a) illustrates the tables of several cells. This simple representation matches the storage pattern of raw geodesic grid data [RRH*02, PKS*11], and thus facilitates fetching and organizing a geodesic grid into GPU memory with minimal computing, memory and implementation overhead. With this representation, we can implicitly obtain the dual Delaunay triangulation to meet the data interpolation requirement and perform ray traversal in an efficient and effective manner. First, as shown in Figure 2 (b), we note that each triangle is uniquely identified by a hexagonal cell corner, which means the ith hexagonal cell corner uniquely corresponds to the ith triangle. Second, each triangle's vertices are uniquely identified by the centers of three neighboring hexagonal cells, and these three cells share the corner identifying the triangle. For example, the three indices in **corner_cell**$[i]$ entry can be treated as three vertex indices of the ith triangle. Third, querying **corner_edge**$[i]$ gives the three edge indices of the ith triangle. Then from these three edge indices, we can quickly identify the neighboring triangles by querying the **edge_corner** table.

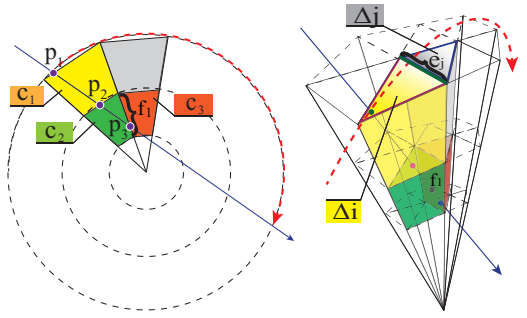


Figure 3: *Grid traversal. (a) Side View. The yellow, green, and red frusta are constructed on the fly with marching the ray through the grid. The intersection of the ray and a temporary frustum is used to identify the next neighboring triangle on the spherical surface. (b) Perspective View. The red dashed curve is the projection of the ray on the spherical surface. The triangles $\triangle_i$ and $\triangle_j$ are identified based on the surface connectivity information, and are used to construct the corresponding frustum at the different layers.*

3.2 Grid Traversal

Algorithm 1 illustrates the overall framework of traversing the grid and performing ray integration frustum by frustum. As shown in Figure 3 (a), given a ray r, we first find its first intersection point, $\mathbf{p}_1$, with the grid. We use the conventional image-based technique [WKME03a] to find the first intersected triangle $\triangle_i$ on the spherical surface. The corresponding implementation will be described in Section 3.6. Then by taking advantage of the radially symmetric property, we can easily construct a 3D triangular frustum on the fly, as the yellow frustum c_1 shown in Figure 3 (a). Next we compute the intersection point using the function *RayIntersectFace* that is a trivial ray-triangle intersection test. We assume that the intersection point is $\mathbf{p}_2$, where the ray leaves the frustum c_1, on the bottom face of c_1. In this case, the ray will enter into the green frustum c_2 in the next layer, and c_2 is still constructed on the fly based on the triangle $\triangle_i$. We compute the next intersection point $\mathbf{p}_3$, and assume $\mathbf{p}_3$ is on a side face f_1 of c_2. We note that a side face of any frustum uniquely corresponds to an edge of the triangle that is used to construct this frustum. We assume that in this case, f_1

corresponds to the edge e_j of the triangle $\triangle_i$, as shown in Figure 3 (b). Then, by querying the **edge_corner** table, we can quickly find the neighboring triangle $\triangle_j$ on the spherical surface, and construct the next frustum c_3, as the red one shown in Figure 3. Algorithm 2 provides the detailed process for finding the next neighboring triangle.

We repeat this traversal process until the ray leaves the domain. We note that this process only uses the 2D connectivity information on the outer spherical surface, and only construct the 3D triangular frustum intersected with a ray on the fly. Each ray corresponds to a projected curve on the spherical surface, as the red dashed curve shown in Figure 3 (b). A temporary frustum is essentially used to find the intersection edges of surface triangles with the projected ray on the spherical surface, thus enabling grid traversal without explicit 3D connectivity information.

Algorithm 1 Ray-casting geodesic grids

```
for each pixel on the projected visible surface parallel do
    Spawn a ray r from the camera
    Obtain the current triangle index △ hit by r
    Initialize the current layer index l as the outer layer index 0
    Initialize r's exit face f on c as a dummy value −1
    λ_min = ∞, λ_max = −1
    while r is within the domain of the grid do
        Construct a frustum c according to △ and l
        for each face j of c do
            // loop through all face of the frustum
            λ = RayIntersectFace(r, c, j, l)
            if λ_min > λ then
                λ_min = λ
            end if
            if λ_max < λ then
                λ_max = λ, f = j
            end if
        end for
        VolumeIntegration(λ_min, λ_max, c)
        NextTriangle(c, f, l, △)
    end while
end for
```

3.3 Interpolation

The geodesic grid in our study is composed of polygonal cells, and the scalar values are cell centered. The signal reconstruction for such grids can be handled using conventional piecewise-constant sampling techniques. However, if smoothed results are preferred, the scalar value at any spatial location inside a cell can be interpolated with the vertex-centered data [MHDG11]. In our study, given the Voronoi-Delaunay duality, we have the dual Delaunay triangulation where the scalar values are defined on the dual grid vertices.

Algorithm 2 $NextTriangle(frustum, exitFaceID, curLayerID, curTriangleID)$

```
maxLayerID = the maximum number of layers in the grid
if exitFaceID is the bottom face of frustum then
    if curLayerID ≥ maxLayerID then
        // hit the deepest layer
        terminate the traversal of the current ray
    else
        curLayerID+=1
    end if
else if exitFaceID is the top face of frustum then
    if curLayerID ≤ 0 then
        // hit the outer layer
        terminate the traversal of the current ray
    else
        curLayerID-=1
    end if
else
    // Ray hits a side face
    Find the edge index edgeID according to exitFaceID in frustum
    if curTriangleID ≠ edge_corner[edgeID][0] then
        curTriangleID = edge_corner[edgeID][0]
    else
        curTriangleID = edge_corner[edgeID][1]
    end if
end if
```

Therefore, the scalar and gradient values can be computed within triangular frustum cells.

We use barycentric interpolation for computing scalar values within a frustum as shown in Figure 4. Let the point $\mathbf{s} = (x, y, z)$ be the sample point whose scalar value needs to be interpolated. In the frustum containing $\mathbf{s}$, we denote the vertices of top spherical triangle by **A**, **B**, **C**, and the vertices of bottom spherical triangle by **D**, **E**, **F**. Given the radially symmetric property of geodesic grids, we generate an interpolation ray shooting from the sphere center **o** through $\mathbf{s}$. This interpolation ray intersects with the top and bottom triangles at the points $\mathbf{r}_1$ and $\mathbf{r}_2$, respectively.

In the first step, we use barycentric interpolation to compute the scalar value at $\mathbf{r}_1$ on the top triangle $\triangle ABC$. It is trivial to compute the barycentric coordinates $(1-u-v, u, v)$ of $\mathbf{r}_1$ in terms of the top triangle vertices. The radially symmetric property implies that the barycentric coordinates of $\mathbf{r}_2$ on the bottom triangles $\triangle DEF$ has exactly the same values as that of $\mathbf{r}_1$. Therefore, the location and scalar values of $\mathbf{r}_1$ and $\mathbf{r}_2$ can be expressed by the location and scalar values of the triangle vertices using barycentric coordinates.

In the next step, we linear interpolate the scalar value of the sampling point $\mathbf{s}$ along the line segment $\mathbf{r}_1\mathbf{r}_2$. We can express the scalar value sv of $\mathbf{s}$ in terms of (x, y, z) as:

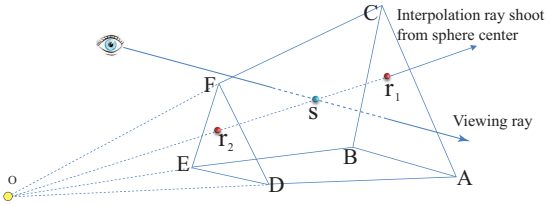


Figure 4: *Barycentric interpolation of the scalar value within a triangular frustum. We first interpolate the scalar values at the points* $\mathbf{r}_1$ *and* $\mathbf{r}_2$ *on the top and bottom triangles, respectively, and then use linear interpolation to compute the scalar value at the point* $\mathbf{s}$ *on the line segment* $\mathbf{r}_1\mathbf{r}_2$.

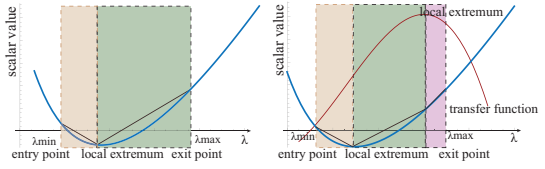


Figure 5: *The plot of a reconstructed scalar value using our analytic function. (a) The analytic function is shown in blue, and has one local minimum. The cell along a ray is split into two intervals colored in light orange and green. The analytic function is monotonic within each interval, and its piecewise linear approximation is shown in black. (b) A transfer function is shown in red, and introduces an additional local extremum point. The cell is further split into three monotonic intervals accordingly.*

$$sv = \frac{a_4x^2 + a_5y^2 + a_6z^2 + y(a_7 + a_8z) + x(a_9 + a_{10}y + a_{11}z)}{a_1x - a_2y + a_3z} \tag{1}$$

where a_i, $i \in \{1,...,11\}$ are the coefficients depending on the coordinates of the six vertices of current frustum and their scalar values. In addition, if we denote the eye position by $\mathbf{e}$ and the viewing ray direction by $\mathbf{d}$, then the point $\mathbf{s}$ can be expressed as:

$$\mathbf{s} = \mathbf{e} + \mathbf{d} * \lambda \tag{2}$$

Substitute (x,y,z) in Equation 1 using Equation 2:

$$sv = (b_3 + b_2\lambda + b_1\lambda^2)/(b_5 + b_4\lambda) \tag{3}$$

where b_i, $i \in \{1,...,5\}$ are the coefficients that depend on the coordinates and the scalar values of the neighborhood vertices. This concludes our analytic interpolation scheme.

3.4 Sampling and Integration

A conventional way to compute ray integration is by sampling over a viewing ray with certain step sizes. Usually a homogeneous sampling can miss small structures within a high frequency scalar field. An alternative solution is to build a pre-integration table to capture possible small features. However, given the number of scalar values and the degrees of our analytic function in Equation 3, it is infeasible to build a pre-integration table to cover the entire interval of large data. Similar to Marchesin et al. [MdV09], we develop a scheme for the integration of a reconstructed scalar value. For our geodesic grid rendering, the scalar function might have several local extremums depending on the specific value of b_i. Let's take one extremum for instance, as shown in Figure 5 (a). We can split a ray within a frustum by the ray entry point, local minimum point, and the ray exit point, to obtain a set of monotonic intervals. For each interval, we use piecewise linear approximation for integration [MdV09]. In this way we can cover the entire interval of scalar value and capture all small structures. In addition, more extreme points may be presented over a ray if we take the transfer function into consideration, as shown in Figure 5 (b). In this case, we still can capture fine features by further splitting a ray at the additional extreme points, and apply the same scheme to generate high-quality volume rendering.

3.5 Gradient Estimation

We can possibly compute the gradient at a sampling point by taking the partial derivative of Equation 1 in terms of x, y and z, and obtain the gradient expression $(\frac{\partial sv}{\partial x}, \frac{\partial sv}{\partial y}, \frac{\partial sv}{\partial z})$. However, we note that if the gradient estimation only relies on the six vertices of a frustum, the gradient values are not necessarily continuous at the frustum boundaries.

To smooth the gradients across frusta, a common strategy is to use the information of the neighboring frusta [CHM11]. We use a similar strategy in our study. Because each vertex is shared by twelve frusta (except for the outer most and inner most surface points and pentagonal points), any given vertex must have twelve different gradient values with respect to the frusta incident to this vertex. First, the twelve gradients are evaluated independently using our derived analytic gradient estimation within each neighboring frustum. Second, the average of these twelve gradients is assigned to the given vertex. After obtaining the averaged gradients for all vertices of a frustum, we then use them to interpolate the gradient at any sampling point along the ray within the frustum. In this way we can significantly improve the shading effect and enhance the perception of shape and depth.

3.6 Implementation

We have implemented our ray-casting method on the GPU using OpenGL and CUDA. Given any view angle, we first render the visible surface. Thanks to our data representation being well suited to the duality of Voronoi-Delaunay, we can render the triangular visible surface using GLSL directly from the original Voronoi polygonal grid, as described in Section 3.2. In the fragment shader, we encode cell indices

	Low res. GCRM	High res. GCRM
resolution	220km	28km
# cells	10242	655362
# corners	20480	1310720
# edges	30720	1966080
# layers	61	99

Table 1: *The GCRM data sets for performance testing in our study. Both data sets contain multiple variables including vorticity, temperature, heat flux, rain water tendency, etc.*

variable	1 CPU core	1 GPU	2 GPUs
heat flux	37.199s	0.494s	0.272s
temperature	45.591s	0.783s	0.468s
rain water tendency	56.247s	0.755s	0.384s
vorticity	59.397s	0.689s	0.460s

Table 2: *Performance (in second) comparison for rendering one frame. The grid has 655362 cells and 99 layers. The image resolution is 512×512. We choose to render four different variables.*

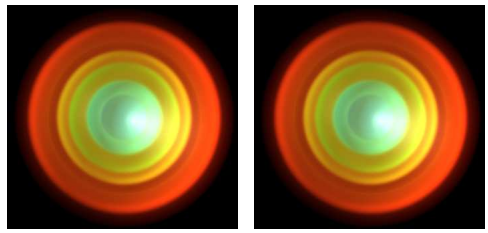
into the fragment color values. With this image-based technique, we can easily identify the outer triangle from which each viewing ray enters into the grid.

We then map each ray to a CUDA thread to perform grid traversal in parallel. The tables of grid representation are organized in CUDA global memory in the same way as the one shown in Figure 2. Each ray starts from the entry triangle whose index is obtained from the fragment of visible surface. We construct the frusta on the fly according to the triangle indices and the current layer index. In each frustum, we analytically compute scalar values, estimate the gradient, and perform ray integration. In addition, we also compute the ray exit point to construct the next frustum for the ray to enter, until the ray leaves the domain.

Our implementation is attractive in that the kernel of our parallel volume rendering CUDA source code can be shared, built and executed on both CPUs and GPUs, thus reducing implementation overhead. Specifically, we exploit the similarity between C/C++ code and CUDA code, and put the volume rendering core in a header file shared by both CPU and GPU implementations. This enables us to run our code on both GPU-enabled machines and the platform, such as supercomputers, without GPU supports. Thus, our implementation is highly compatible to heterogeneous systems.

4 Results and Discussion

We evaluate our approach from several aspects, including the performance and the effectiveness of our analytic interpolation and gradient estimation. We also compare our approach with the conventional tetrahedron based rendering results.



(a) (b)

Figure 6: *Ray-casting of a synthetic spherical scalar field defined on a real geodesic grid with 10242 cells and 61 layers. (a) shows the result using our analytic approach with lighting enabled, and (b) shows the ground truth image. The image difference between (a) and (b) is negligible.*

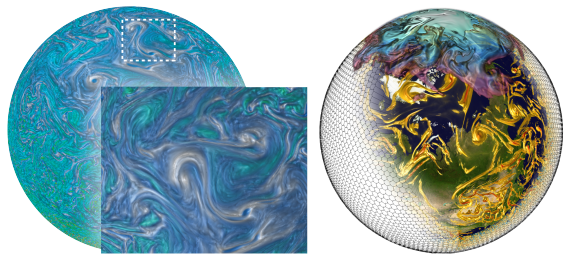


Figure 7: *Ray-casting with lighting effects using our analytic ray integration and gradient estimation on the high resolution data set. The left images shows the vorticity variable and the close-up view of a selected region. The right image shows the combination of grid illustration, volume rendering of vorticity (yellow tubes) and temperature (north cap). Our approach can succinctly reveal great details from large data with enhanced depth and shape cues.*

4.1 Rendering Performance

The experimental study has been conducted on a Intel Core 2 Quad CPU at 2.4 GHz with 24 GB memory and dual NVIDIA Geforce 580 GTX GPUs with 1.5 GB memory per GPU. We have implemented our approach on both CPUs and GPUs (Section 3.6). The performance evaluation is conducted on two global cloud resolving model (GCRM) data sets with low and high resolutions. The detailed information of the two GCRM data sets is listed in Table 1.

Table 2 shows the performance difference between CPU, single GPU and dual GPUs rendering on the high resolution geodesic grid. For the CPU rendering, we only use one single CPU core and perform rendering in a sequential fashion. When using dual GPUs, we duplicate the grid data on each GPU, and partition workload in image space. We can see that our single GPU accelerated volume rendering achieves approximately a 100× speedup from the CPU rendering, and the dual GPUs rendering achieves roughly a 2× speedup from the single GPU rendering.

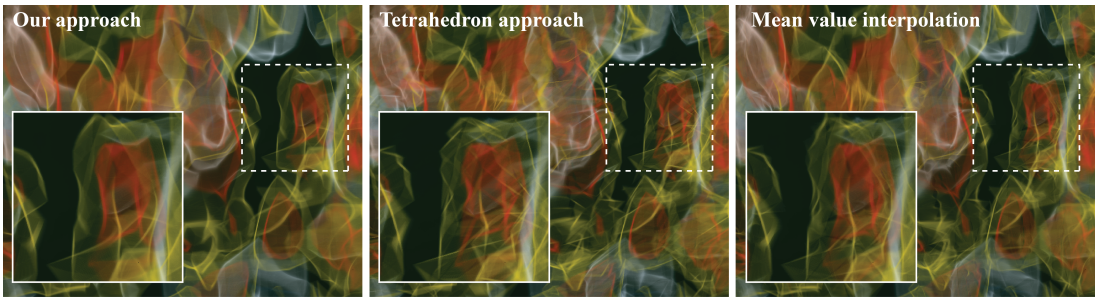


Figure 8: *Rendering quality comparison of our approach (left) with the conventional tetrahedron based method (center) and the mean value interpolation method (right) in a close-up views. We render the vorticity variable in the high resolution grid. The ratio of sample step size to the globe radius is set as* 10^{-4} *to make sure there are enough sampling points within each cell. The boundary artifacts at the high frequency regions are perceivable from the rendering of tetrahedral mesh due to less vertex information for interpolation. Meanwhile, even if the rendering quality of mean value interpolation method is slightly better than the tetrahedron method, certain jaggy effect can still be perceived. This is due to the fact that mean value interpolation has proved to be smooth in the interior of a cell except at the vertices [HF06]. Therefore, in terms of image quality, our approach outperforms both methods.*

4.2 Rendering Quality

To verify the effectiveness of our analytic scheme for ray integration and gradient estimation, we first use a synthetic spherical scalar field generated on a real geodesic grid. Figure 6 (a) shows the result using our approach. We also generate a ground truth image in a way that at a sampling point along a ray the scalar value is precisely determined as the Euclidean distance between the sphere center and the point, and the gradient is precisely determined by the vector from the sphere center to the point, as shown in Figure 6 (b). We note that our approach can generate high-quality rendering and provide nearly indistinguishable results from the ground truth. The difference only becomes perceivable when the grid resolution is extremely low, which is normally impractical for simulations.

Figure 7 demonstrates the quality of ray integration and lighting effects using our analytic scheme on the real climate data set. We note that our approach can achieve high-quality rendering with smooth shading. In particular, the fine turbulent features embedded in the high resolution data are succinctly revealed with the enhanced depth and shape cues provided by our gradient estimation method.

We also provide both qualitative and quantitative comparisons of our approach with two state-of-the-art approaches. The first approach implicitly tetrahedralizes each frustum on the fly and uses the ray-casting method on tetrahedral mesh [WKME03a] to render the transformed data. The second approach employs mean value interpolation [MHDH07, MHDG11] to compute scalar values at sampling points within a frustum. Figure 8 demonstrates the rendering quality comparison of our ray-casting approach with these two methods in a close-up view. We observe that our approach can generate high-quality rendering superior to both methods, especially at the regions with high frequency.

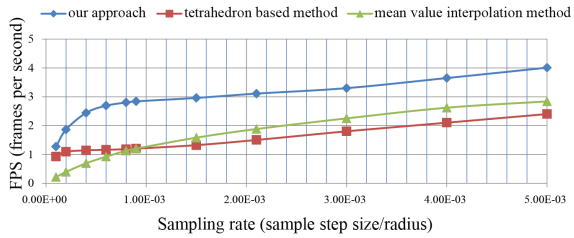


Figure 9: *Performance measures and comparison with the conventional tetrahedron based method and the mean value interpolation method. The performance data is obtained by rendering images of Figure 8 with different sampling rate that is defined as the ratio of sample step size to the globe radius. Our approach outperforms both methods.*

Our ray-casting also has a lower complexity in terms of the total cell number and sampling point number, and can achieve a higher frame rate than the conventional methods, as denoted by the performance comparison in Figure 9. Our approach achieves an average of 98.6% performance gain over the tetrahedron based method. This is mainly because the tetrahedron based method needs visibility sorting of three tetrahedrons within a frustum. Our approach also achieves an average of 172% improvement over the mean value interpolation method. This is mainly because it requires an inverse trigonometric function and several vector operations to compute the angles over eight triangular faces of a frustum for mean value coordinates [MHDH07].

Figure 10 shows a set of high-quality and high-resolution rendering of full-resolution data by our approach, which allows scientists to capture the fine details from their large climate data on geodesic grids.

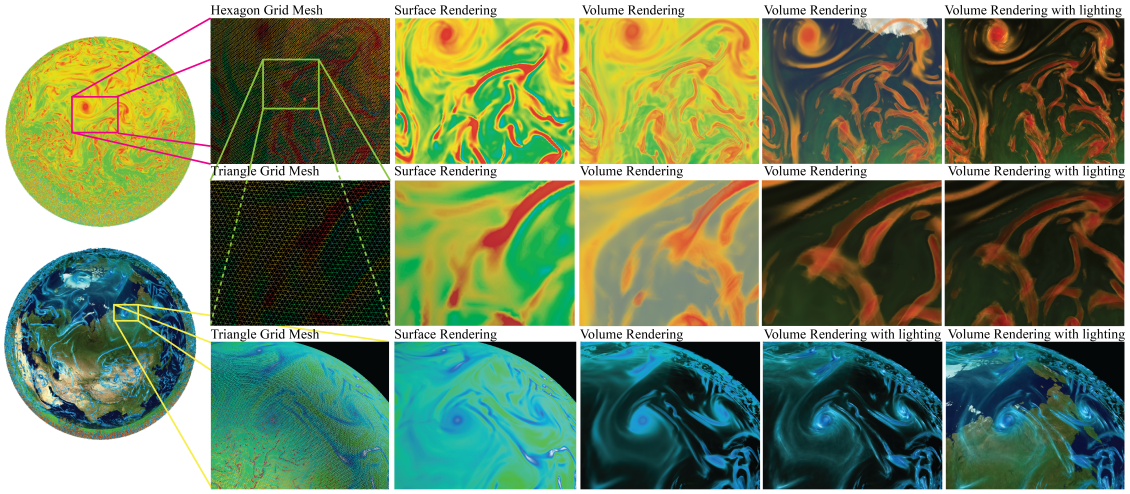


Figure 10: *The left most two images show the volume rendering of the whole global atmospheric vorticity variable using different visualization parameters. The top two rows of images incrementally show the close-up views of a particular region of interest. From left to right, we show the mesh grid, the surface rendering, the volume rendering with different transfer functions, and the images with lighting disabled and enabled. The bottom row shows a close-up view from a different geolocation, superimposed with geophysical information.*

5 Conclusion and Future work

We have presented a direct volume rendering approach for geodesic grid data. Without loss of generality, we apply the approach to hexagonal grids, which can be directly fetched and rendered without any intermediate grid transformation. As a result, we can significantly reduce computing and memory cost. For the higher resolution grid in our current study, our approach directly takes the grid of the outer most surface with 1.3 million grid points as input. However, if we explicitly convert the geodesic grid to the tetrahedral or polygonal grids over 99 layers, it results in an input of 128.7 million grid corners and thus makes the memory and preprocessing cost unaffordable for large data. The data structure in our approach has the same complexity as the original simulation grids, and is potentially scalable with the increasing scale of simulations. Furthermore, our current design well matches the simulation data storage patterns, which facilities a direct integration with simulation codes and offers possibly viable in-situ visualization solutions for large climate simulations.

We have also derived a new analytic scheme for interpolation, gradient calculation, and ray integration. The accuracy of our analytic scalar and gradient interpolation is comparable to those using central difference numerical computations in simulations. The rendered results feature better image quality, less memory overhead, and higher performance compared with conventional methods.

There are several promising directions for future work. First, we would like to improve the gradient calculation using higher order analytic functions. We approach this with caution as such improvements may incur more memory access and computing cycles. It is an interesting topic on how to make a trade off between image quality and computation complexity for large data. Secondly, we want to optimize the performance of multiple GPUs ray-casting scheme, and address the bottleneck of data transfer between GPU memory and main memory. Our current grid is generated after 7 recursive subdivisions of an icosahedron, such that each mesh cell covers an Earth surface area of 27 km in diameter. Currently, our collaborators are generating higher-resolution grids using finer subdivision. For example, in the near future, a grid will have a resolution of 4 km in diameter and 256 layers. Given this scale, the size of each variable will reach to 4.5 terabytes. No single machine can presently handle such a large scale data set with ease. We will seek a multi-resolution solution on PC clusters and in-situ solution on HPC supercomputers to effectively process the large data sets. This will enable scientists to study their climate data at full extent and responsively deliver more accurate global weather or climate forecasts. This paper presents important foundation work for carrying out these further research tasks.

Acknowledgments

This research has been sponsored in part by the National Science Foundation through grants OCI-0905008, OCI-0850566 and NSF OCI-0749227, and also the Department of Energy through grants DE-FC02-06ER25777 and DE-FC02-12ER26072, program manager Lucy Nowell. The GCRM data sets are obtained through Karen Schuchardt at Pacific Northwest National Laboratory.

References

[AC07] ABRUGIA G., CARFORA M. F.: Semi-Lagrangian advection on a spherical geodesic grid. *International Journal for Numerical Methods in Fluids 55* (2007), 127–142. 1

[BCM*01] BENNETT J., COOK R., MAX N., MAY D., WILLIAMS P.: Parallelizing a High Accuracy Hardware-Assisted Volume Renderer for Meshes with Arbitrary Polyhedral. In *Proceedings of PVG* (2001), pp. 150–156. 3

[CBX12] CHEN C., BIN J., XIAO F.: A Global Multimoment Constrained Finite-Volume Scheme for Advection Transport on the Hexagonal Geodesic Grid. *Monthly Weather Review*, 140 (2012), 941–955. 1

[CHM11] CORREA C. D., HERO R., MA K.-L.: A Comparison of Gradient Estimation Methods for Volume Rendering on Unstructured Meshes. *IEEE Transactions on Visualization and Computer Graphics 17*, 3 (2011), 305–319. 6

[FEVM10] FINKBEINER B., ENTEZARI A., VILLE D. V. D., MÖLLER T.: Efficient Volume Rendering on the Body Centered Cubic Lattice Using Box Splines. *Computers & Graphics 34*, 4 (2010), 409–423. 3

[Gar90] GARRITY M. P.: Raytracing Irregular Volume Data. In *Proceedings of VVS* (1990), pp. 35–40. 3

[HF06] HORMANN K., FLOATER M.: Mean Value Coordinates for Arbitrary Planar Polygons. *ACM Transactions on Graphics (TOG) 25*, 4 (2006), 1424–1441. 8

[KE01] KRAUS M., ERTL T.: Cell-Projection of Cyclic Meshes. In *Proceedings of IEEE Visualization* (2001), pp. 215–559. 3

[Lev10] LEVY D.: Exascale Supercomputing Advances Climate Research. *SciDAC Review*, 16 (2010), 20–31. 1

[LWM04] LUM E. B., WILSON B., MA K.-L.: High-Quality Lighting and Efficient Pre-Integration for Volume Rendering. In *Proceedings of Joint EUROGRAPHICS-IEEE TCVG Symposium on Visualization* (2004), pp. 25–34. 3

[MA04] MORELAND K., ANGEL E.: A Fast High Accuracy Volume Renderer for Unstructured Data. In *Proceedings of the IEEE Symposium on Volume Visualization and Graphics* (2004), pp. 9–16. 3

[MdV09] MARCHESIN S., DE VERDIÈRE G. C.: High-Quality, Semi-Analytical Volume Rendering for AMR Data. *IEEE TVCG 15*, 6 (2009), 1611–1618. 3, 6

[MHC90] MAX N., HANRAHAN P., CRAWFIS R.: Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. *Computer Graphics 24*, 5 (1990), 27–33. 3

[MHDG11] MUIGG P., HADWIGER M., DOLEISCH H., GRÖLLER E.: Interactive Volume Visualization of General Polyhedral Grids. *IEEE TVCG 17*, 12 (2011), 2115–2124. 3, 5, 8

[MHDH07] MUIGG P., HADWIGER M., DOLEISCH H., HAUSER H.: Scalable Hybrid Unstructured and Structured Grid Raycasting. *IEEE TVCG 13*, 6 (2007), 1592–1599. 3, 8

[MLP*02] MAJEWSKI D., LIERMANN D., PROHL P., RITTER B., BUCHHOLD M., HANISCH T., PAUL G., WERGEN W.: The Operational Global Icosahedral-Hexagonal Gridpoint Model GME: Description and High-Resolution Tests. *Monthly Weather Review*, 130 (2002), 319–338. 1

[MRB*08] MAXIMO A., RIBEIRO S., BENTES C., OLIVEIRA A., FARIAS R.: Memory Efficient GPU-Based Ray Casting for Unstructured Volume Rendering. In *Proceedings of Volume Graphics* (2008), pp. 155–162. 3

[MUS07] MITTAL R., UPADHYAYA H., SHARMA O.: On Near-Diffusion-Free Advection over Spherical Geodesic Grids. *Monthly Weather Review 135*, 12 (2007), 4214–4225. 1

[PKS*11] PALMERA B., KOONTZA A., SCHUCHARDTA K., HEIKESB R., RANDALLB D.: Efficient Data IO for a Parallel Global Cloud Resolving Model. *Environmental Modelling & Software 26*, 12 (2011), 1725–1735. 4

[RE02] RÖTTGER S., ERTL T.: A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids. In *Proceedings of the IEEE Symposium on Volume Visualization and Graphics* (2002), pp. 23–28. 3

[RKE00] RÖTTGER S., KRAUS M., ERTL T.: Hardware-Accelerated Volume and Isosurface Rendering Based On Cell Projection. In *Proceedings of IEEE Visualization* (2000), pp. 109–116. 3

[RRH*02] RANDALL D. A., RINGLER T. D., HEIKES R. P., JONES P., BAUMGARDNER J.: Climate Modeling with Spherical Geodesic Grids. *Computing in Science & Engineering 4*, 5 (2002), 32–41. 1, 4

[SAM68] SADOURNY R., ARAKAWA A., MLNTZ Y.: Integration of the Nondivergent Barotropic Vorticity Equation with an Icosahedral-Hexagonal Grid for the Sphere. *Monthly Weather Review*, 96 (1968), 351–356. 1

[SBM94] STEIN C. M., BECKER B. G., MAX N.: Sorting and Hardware Assisted Rendering for Volume Visualization. In *Proceedings of VolVis* (1994), pp. 83–89. 3

[SDM10] SHALF J., DOSANJH S., MORRISON J.: Exascale Computing Technology Challenges. In *Proceedings of VecPar* (2010), pp. 1–25. 2

[ST90] SHIRLEY P., TUCHMAN A.: A Polygonal Approximation to Direct Scalar Volume Rendering. In *Proceedings of VVS* (1990), pp. 63–70. 2

[Wil68] WILLIAMSON D. L.: Integration of the barotropic vorticity equation on a spherical geodesic grid. *Tellus*, 20 (1968), 642–653. 1

[Wil92] WILLIAMS P. L.: Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics 11*, 2 (1992), 103–126. 3

[WKME03a] WEILER M., KRAUS M., MERZ M., ERTL T.: Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proceedings of IEEE Visualization* (2003), pp. 333–340. 3, 4, 8

[WKME03b] WEILER M., KRAUS M., MERZ M., ERTL T.: Hardware-Based View-Independent Cell Projection. *IEEE TVCG 9*, 2 (2003), 163–175. 3

[WMKE04] WEILER M., MALLÓN P. N., KRAUS M., ERTL T.: Texture-Encoded Tetrahedral Strips. In *Proceedings of VolVis* (2004), pp. 71–78. 3

[WWF09] WELLER H., WELLER H. G., FOURNIER A.: Voronoi, Delaunay, and Block-Structured Mesh Refinement for Solution of the Shallow-Water Equations on the Sphere. *Monthly Weather Review*, 137 (2009), 4208–4224. 1