Visualizing 3D atmospheric data with spherical volume texture on virtual globes

Jianming Lianga,b, Jianhua Gonga,c, Wenhang Lia,c, Abdoul Nasser Ibrahim a,c

a State Key Laboratory of Remote Sensing Science, Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, P.O. Box 9718, Beijing 100101, China
b University of Chinese Academy of Sciences, Beijing 100049, China
c Zhejiang-CAS Application Center for Geoinformatics, Zhejiang 314100, China

ARTICLE INFO

Article history:
Received 19 November 2013
Received in revised form 24 March 2014
Accepted 26 March 2014
Available online 3 April 2014

Keywords:
Volumetric ray-casting
Virtual globe
Atmospheric volume data
Spherical volume texture

ABSTRACT

Volumetric ray-casting is a widely used algorithm in volume visualization, but implementing this algorithm to render atmospheric volume data that cover a large area on virtual globes constitutes a challenging problem. Because atmospheric data are usually georeferenced to a spherical coordinate system described by longitude, latitude and altitude, adaptations to the conventional volumetric ray-casting method are needed to accommodate spherical volume texture sampling. In this paper, we present a volumetric ray-casting framework to visualize atmospheric data that cover a broad but thin geographic area (because of the thinness of Earth’s atmosphere). Volume texture conforming to the spherical coordinate system of a virtual globe can be created directly from the spherical volume data to avoid oversampling, undersampling or a loss of accuracy due to reprojecting and resampling such data into a Cartesian coordinate system. Considering the insignificant physical thickness of the atmosphere of the Earth, the ray-casting method presented in this paper also allows for real-time vertical scaling (exaggeration of the altitudinal range) without the need to re-process the volume texture, enabling convenient visual observation of the altitudinal variations. The spherical volume ray-casting method is implemented in a deferred rendering framework to integrate the volume effects into a virtual globe composed of a variety of background geospatial data objects, such as terrain, imagery, vector shapes and 3D geometric models.

1. Introduction

Former US Vice President Al Gore put forth his visionary Digital Earth (DE) concept in 1998, defined as a computer-networked virtual representation of the physical Earth based on georeferenced heterogeneous data sources (Gore, 1999). Since the Gore speech, many DE prototypes and commercial applications have emerged and have continued to evolve over the last decade, even well beyond Gore’s vision (Goodchild et al., 2012). Pioneering DE systems such as Google Earth (GE) and NASA WorldWind (WW) have been developed to construct a realistic replica of the planet, complete with global satellite imagery, digital terrain models (DEM), vector data and three-dimensional (3D) cities. As we advance toward the year 2020, DE systems should become a dynamic framework for sharing information globally and improving our collective understanding of the complex relationships between society and the environment in which we live (Craglia et al., 2012) because the first principle of DE is unrestricted accessibility to DE by all of humankind whether they are individuals, government agencies, non-governmental organizations (NGOs), or for-profit and not-for-profit organizations (Ehlers et al., 2014).

Currently, DE is more focused on information about the Earth’s surface, from the atmosphere down to the biosphere. In a classical DE application, global terrain and imagery are the most fundamental data layers, for which the conventional out-of-core data management and level-of-detail (LOD) rendering schemes, e.g., P-BDAM (Cignoni et al., 2003), have been sufficiently optimized to support real-time exploration of a theoretically unlimited amount of data. However, neither GE nor WW provides any volume rendering functionalities, not to mention a practical mechanism for the interactive visualization of regional- or planetary-scale volumetric data, e.g., atmospheric modeling results. Instead, many GE users in academia rely heavily on Keyhole Markup Language (KML) to visualize, locate and navigate through their own geospatial data as two-dimensional (2D) overlays or 2.5D polygons (Bailey and Chen, 2011). For example, the progressive dispersion of
volcanic gas was visualized in GE by animating an array of KML polygons on a timeline (Wright et al., 2009). Volumetric data scanned by space-borne meteorological radar were represented as 2D vertical profiles using KML in GE. Such visualization techniques only present a partial view of the original volumetric data and may not be able to provide a holistic representation that allows viewers to easily understand the data as a whole.

Volumetric visualization has been broadly applied in academia and industry to facilitate visual analytics and simulate participating media in applications such as medical image rendering (Zhang et al., 2011), cloud simulation (Elek et al., 2012), and geoscience data visualization (Patel et al., 2010). There are many open-source volume visualization toolkits available online. For instance, the Visualization Toolkit (VTK) and Voreen are freely available volume rendering engines that incorporate many sophisticated algorithms. One of the earliest software systems to support fully interactive 3D visualization of time-varying volumetric weather simulations was Vis5D, which also provides an open-source API to enable developers of other systems to incorporate Vis5D’s functionality (Hibbard et al., 1994). The 2004 IEEE Visualization Contest encouraged participants to develop new tools to visualize the various types of behavior exhibited by a hurricane volume dataset. The following applications were among the three winning entries to the contest: (1) SimVis (Doleisch et al., 2004) was an interactive visual analysis software for multi-variate and time-dependent 3D simulation data on unstructured grids; (2) AtmosV (Doleisch et al., 2004) was an immersive prototype application derived from the Immersive Drilling Planner to visualize large multi-variate atmospheric datasets; (3) the OpenGL application (Greg and Christopher, 2004) used multithreading to maximize the interactivity of the visualization. Nevertheless, these visualization toolkits only provide a pure volumetric rendering environment that lacks the ability to integrate various geospatial data sources. In contrast to these general-purpose rendering toolkits, DE offers as one of its major benefits convenient viewer access to a variety of georeferenced data sources because DE plays the role of a knowledge engine (Krzysztof and Hitzler, 2012) rather than a pure rendering engine. Consequently, the implementation of volume visualization in DE must be conducted bearing in mind that such functionalities can easily be integrated with a regular virtual globe, set in an enriched geographic background that allows users to comprehend the context.

Graphic processing units (GPUs) have largely been used to accelerate volumetric rendering for interactive applications (Monte, 2013). Volumetric data in video memory are present in a GPU-compatible format, i.e., 3D texture or volume texture, which is supported in either DirectX or OpenGL to facilitate direct volumetric rendering. In fact, a volume texture is a collection of 2D textures in the form of slices. Volumetric ray-casting (Kruger and Westermann, 2003), shear-warp (Lacroute and Levoy, 1994) and splatting (Westover, 1991) are three classic direct volume rendering methods. Because volumes are determined on a pixel-by-pixel basis, ray-casting can generate high-quality images without any blurring or loss of detail (Dye et al., 2007). Volumetric ray-casting is perhaps the most researched method due to its superior rendering quality, easily understood fundamental principles and convenient GPU-based implementation.

A few attempts have been made to implement GPU-based volumetric ray-casting in virtual globe applications (Yang and Wu, 2010; Li et al., 2011, 2013). Yang and Wu (2010) created volume texture directly from electromagnetic data in a geographic coordinate system (GCS) (Bugayevskiy and Snyder, 1995) and presented a volume-ray casting method to visualize them on a virtual globe. Li et al. (2013) reprojected and resampled GCS-based atmospheric volume data into a Cartesian volume space for visualization in WW. However, this type of reprojection may lead to oversampling or undersampling of volumetric data whose voxels were originally described by longitude, latitude and altitude. Moreover, volume data in geosciences, such as atmospheric modeling results, may cover large geographic areas with a very small altitudinal range, calling for a ray-casting method that supports on-the-fly vertical exaggeration on virtual globes.

We herein present a GPU-based volumetric ray-casting framework to accomplish the following:

1. Direct visualization of GCS-based volumetric data (particularly atmospheric data in this study) on virtual globes, without resorting to data resampling and reprojection techniques, to provide a more faithful representation of the original data contents as they were recorded;
2. Real-time vertical scaling in GCS for more convenient observation of altitudinal variations;
3. Seamless integration of atmospheric volume data with other data contents on virtual globes to provide enriched geographic backgrounds for interactive volume analysis.

Fig. 1. Geographic setting of Hurricane Isabel.
Atmospheric volumetric data are composed of arrays of spherical voxels located by longitude, latitude and altitude. Spherical voxels have variable sizes along the parallel and meridian. Because longitude is not a uniform unit of measure, one degree of longitude gradually shortens from 111.321 km at the equator to 55.8 km at a latitude of 60° and finally to zero at the poles.

Unlike most commonly observed volumetric data, such as medical images, which are not georeferenced and can conveniently be visualized in an isolated environment without resorting to multi-source data integration, atmospheric volumetric data have the following characteristics:

1. Atmospheric volumetric data are composed of arrays of spherical voxels located by longitude, latitude and altitude. Spherical voxels have variable spatial sizes along the parallel and meridian. Because longitude is not a uniform unit of measure, one degree of longitude gradually shortens from 111.321 km at the equator to 55.8 km at a latitude of 60° and finally to zero at the poles.

2. Atmospheric volume data cover broad geographic areas, in some cases even the entire Earth, but represent a very narrow vertical range due to the thinness of Earth’s atmosphere. For example, Hurricane Isabel had a horizontal extent of approximately 2000 km, which was 100 times greater than its vertical extent of 19.8 km. However, there are 100 volume slices in the dataset along the vertical dimension, so it is necessary to scale up the altitudinal range to highlight the hurricane’s vertical features.

3. The spherical and Cartesian coordinate systems

A typical virtual globe platform makes use of two types of coordinate systems to describe 3D objects in different contexts, i.e., Cartesian coordinate systems (CCS) and GCS.

A CCS describes a position in 3D space by a triple set \( (X, Y, Z) \) (Fig. 2). The Cartesian origin \((0, 0, 0)\) of a virtual globe usually corresponds to the spherical or spheroidal center.

A spherical coordinate system describes a position in 3D space by a radius, a polar angle, and an azimuthal angle (Fig. 2). A GCS is a variant of the spherical coordinate system widely used in geosciences to describe the spatial locations of georeferenced data by longitude, latitude and altitude (Snyder, 1987).

In real-time rendering, 3D geometric primitives, e.g., triangles, have to be represented in Cartesian coordinates because the GPU rendering pipeline does not directly take in spherical coordinates. Nevertheless, GCS is more convenient for describing spherical locations and for helping human beings to imagine objects on the Earth. Therefore, many georeferenced data are natively registered to GCS and require on-the-fly transformation into Cartesian coordinates for rendering in a virtual globe; For example, georeferenced satellite images are clamped onto Cartesian triangular meshes as texture layers.

A recent study (Li et al., 2013) proposed a reprojection of GCS-based atmospheric volume data into a CCS so that each voxel has a uniform spatial size, unlike spherical voxels (Fig. 3): first, a CCS-aligned bounding volume is calculated and an average voxel size is determined to create a volume texture; second, the original voxels are transformed into a CCS and resampled into the volume texture, which can then be visualized by implementing the generic volume ray-casting algorithm in a virtual globe application.

However, such a direct implementation of the volume ray-casting algorithm on virtual globes may cause the following data-related problems:

1. Oversampling or undersampling. Because the spherical voxels have variable sizes due to the angular unit (Snyder, 1987), a uniform voxel resolution needs to be determined to reproject such volume data into a CCS.

2. Inefficient storage. A larger Cartesian bounding volume (Fig. 4) is needed to fully contain the spherical volume data for reprojection, resulting in many void spaces in the newly created volume texture.

3. Inconvenience in vertical scaling on virtual globes. After an atmospheric volume dataset is reprojected to a CCS, vertical scaling cannot be dynamically applied to zoom in on the altitudinal variations because the Cartesian voxels no longer conform to the spherical shells of the virtual globe (Fig. 4).

5. Ray-casting method for spherical volume texture on virtual globe

5.1. Spherical volume texture

In response to the above-mentioned problems related to the implementation of volume-ray casting on virtual globes, we introduce the notion of “spherical volume texture” (SVT), which is a replica of the GCS-based volume data in video memory.
For example, suppose we want to create a volume texture for a single time step of the Hurricane Isabel dataset: first, a volume texture of 512 × 512 × 100 is allocated in video memory, with each voxel representing a floating-point value; second, the spherical volume data are copied to the volume texture on a voxel-by-voxel basis. Like a generic volume texture, an SVT adheres to a normalized texture coordinate system, which means that in all three dimensions, the texture coordinates range from 0 to 1 (Fig. 5).

The spherical bounding volume of an SVT (Fig. 6) is composed of two spherical shells with four walls on the edges, with each voxel indexed by its longitude, latitude and altitude. The spatial size of a voxel cube tends to decrease toward the polar regions.

Next, we introduce the conventional volume ray-casting algorithm and discuss how to adapt the algorithm so that an SVT can be visualized directly on virtual globes.

5.2. The conventional volume ray-casting method

Volume ray-casting is an image-space rendering method that transforms a volume texture into a 2D image for display. To obtain a screen-space image, a ray is shot from every screen pixel through the volume to fetch samples and produce the final color through certain transfer functions. Before the actual ray-casting algorithm is executed, the entry and exit positions of each ray have to be obtained. A simple technique for finding the ray-volume intersections involves the following steps:

(1) Find a model-space bounding box for the volume data and tessellate it with triangle primitives for rendering.

(2) Prepare two 2D texture buffers to store the ray entry and exit positions.
(3) Enable back-face culling and draw the bounding box to render the front-face positions to the texture buffer for storing ray entry positions.
(4) Enable front-face culling and draw the bounding box to render the back-face positions to the texture buffer for storing ray exit positions.
After the ray entry and exit positions have been obtained in the form of two texture buffers, the core ray-casting algorithm can be executed in the following steps:

1. Ray construction. For each pixel of the final image, a ray is constructed from the entry position toward the exit position.
2. Texture sampling. From the entry to the exit position of a ray, equidistant texture samples are collected.
3. Shading and compositing. A color transfer function and a light transport model are used to calculate the final color for each ray based on an accumulation of the texture samples. The following equation describes a classic light transport model:

\[ I = \int_0^{t_{\text{max}}} I(t) e^{-\int_0^t \mu(t) \, ds} \, dt \]  

where \( t_{\text{max}} \) is the path distance from ray entry to exit, \( I \) is the final irradiance, \( s \) is the distance increment, \( t \) is the upper integration limit and \( \mu \) is the attenuation coefficient.

The conventional ray-casting method does not take into account the spherical curvature or the angular units of a spherical coordinate system and does not specify how to integrate with a virtual globe. Next, we will present an SVT-compatible ray-casting framework designed specifically for virtual globes.

5.3. Ray-casting for spherical volume texture

First, we have to find a spherical bounding box that fully contains the volume space so that we can determine the ray entry and exit positions. Yang and Wu (2010) transformed the eight corner points of the geographic bounding volume directly to construct a Cartesian bounding volume that ignores the curvature of the Earth. Such a Cartesian bounding volume may result in excessive void spaces over a large geographic domain, causing unnecessary samplings and calculations along the rays. We propose the use of spherical tessellation to create a bounding volume comprising two spherical shells and four vertical walls on the east, west, north and south edges (Fig. 7). Because a larger number of triangle primitives will put more stress on the GPU, this bounding volume has to be tessellated as simply as possible. We approximate a minimum bounding volume through spherical interpolation and tessellation (Fig. 7) in the following manner:

1. We determine the approximate parallel and meridian spacing for tessellation based on the spatial resolution of the data. Two spherical shells are tessellated with triangle strips based on the specified parallel and meridian spacing. One spherical shell is set at the minimum altitude, and the other is set at the maximum altitude.
(2) We extrude the four walls from the respective edges of the geographic extent by an amount equal to the altitudinal range.

We then allocate two separate floating-point texture buffers to store the ray entry and exit positions. We create a GPU vertex shader and fragment shader to render the spherical bounding volume. To write the face positions onto the textures, the positions are passed to the fragment shader as texture coordinates for rasterization. Back-face culling is enabled to obtain the ray entry positions (Fig. 8a), and front-face culling is enabled to obtain the ray exit positions (Fig. 8b). Fig. 8 shows the ray entry and exit positions in shaded colors:

Because each voxel of an SVT is spatially located by its longitudinal, latitudinal and altitude, we need to convert the Cartesian coordinates \((X, Y, Z)\) of a sample point on the cast ray into geographic coordinates using Eq. (2):

\[
\begin{align*}
\rho &= \sqrt{x^2 + y^2 + z^2} \\
\text{longi} &= \arctan\left(\frac{y}{x}\right) \\
\text{lati} &= \arcsin\left(\frac{z}{\rho}\right) \\
\text{alt} &= \rho - r
\end{align*}
\]

where \(r\) is Earth’s radius, \(\text{longi}\) is longitude, \(\text{lati}\) is latitude and \(\text{alt}\) is altitude. Because the altitude is obtained in real-time calculations, a vertical scale factor can be applied in transforming the spherical coordinates into texture coordinates using Eq. (3):

\[
\begin{align*}
u &= (\text{longi} - \text{longi}_{\text{min}})/(\text{longi}_{\text{max}} - \text{longi}_{\text{min}}) \\
v &= (\text{lati} - \text{lati}_{\text{min}})/(\text{lati}_{\text{max}} - \text{lati}_{\text{min}}) \\
s &= (\text{alt} - \text{alt}_{\text{min}})/[(\text{alt}_{\text{max}} - \text{alt}_{\text{min}}) \times \text{vscale}]
\end{align*}
\]

where \(u\), \(v\) and \(s\) are the normalized 3D texture coordinates, \(\text{vscale}\) is the vertical exaggeration factor, \(\text{longi}_{\text{max}}\) is the maximum longitude, \(\text{longi}_{\text{min}}\) is the minimum longitude, \(\text{lati}_{\text{max}}\) is the maximum latitude, \(\text{lati}_{\text{min}}\) is the minimum latitude, \(\text{alt}_{\text{max}}\) is the maximum altitude and \(\text{alt}_{\text{min}}\) is the minimum altitude.

As \(\text{vscale}\) changes, the vertices of the spherical bounding volume need to be re-calculated accordingly to provide correct ray-entry and ray-exit positions. Because both the ray-casting and the texture sampling are performed in real-time, a change in the variable \(\text{vscale}\) would be immediately reflected in the vertical component \(s\) of the 3D texture coordinates \((u, v, s)\). More specifically, in the execution stage of the ray-casting algorithm as described in Section 5.2, the 3D texture coordinates required for the texture sampling are calculated by Eq. (3). Assuming that the longitudinal range, latitudinal range and altitudinal range are known variables, what we need to do is transform the sampled ray position \((X, Y, Z)\) from the Cartesian space to the spherical space using Eq. (2). The resulting spherical position \((\text{longi}, \text{lati}, \text{alt})\) can then be transformed into the texture address \((u, v, s)\) using Eq. (3) for actual texture sampling.

5.4. Seamless integration to virtual globes

The conventional volume ray-casting algorithm does not provide a specific solution for how to merge the data with the virtual globe background scene. Because multi-source spatial data integration is one of the fundamental functionalities of virtual globes, fitting volume techniques into the virtual globe-rendering pipeline is vital to the applicability of the presented volume ray-casting framework.

Deferred rendering (Geldreich et al., 2004; Hargreaves and Harris, 2004) gained considerable popularity in 3D engine development between 2008 and 2010. In comparison to traditional forward rendering, which shades each 3D object individually, a typical deferred rendering pipeline renders all 3D objects from a scene into three separate texture buffers, i.e., the position buffer (or depth buffer), the normal buffer and the color buffer, and then performs shading at a later stage to yield the final visual (Fig. 9).

Most virtual globe engines, such as WW, still employ a forward rendering pipeline to render terrain, vector shapes and 3D models for immediate display, without considering possible blending with volume effects at a later stage. Therefore, we propose to leverage deferred rendering to seamlessly incorporate volume effects into a virtual globe in a compositing operation. Our deferred rendering pipeline is organized as shown below:

As indicated in the above illustration (Fig. 10), the main rendering pipeline consists of three render passes:

(1) In the first render pass, all of the virtual globe data objects, such as terrain, imagery, vector shapes and 3D geometric models, are rendered separately to a position buffer and a color buffer. The position buffer records the per-pixel positions, and the color buffer records the per-pixel colors. Unlike the regular virtual globe-rendering pipeline, the rendering results are not presented for immediate display but are saved for later compositing.

(2) In the second render pass, the ray entry and exit positions of the spherical volume are rendered to two separate textures.

(3) In the final render pass, the spherical volume is rendered using the ray-casting method presented herein and fused with the
virtual globe buffers to produce the final image. The position
and color buffers from the first render pass as well as the ray
entry and exit buffers from the second render pass are all
bound to a GLSL fragment shader. For each screen pixel, a ray is
shot through the spherical volume to accumulate texture
samples. The actual ray entry and exit positions are obtained
by comparing the original ray entry and exit positions against
the distance of the background pixel from the camera. The
compositing process is illustrated below in the form of a
pseudocode snippet:

```cpp
foreach (pixel on screen){
    vec3 ray_origin = getRayOrigin(uv);
    vec3 ray_entry = texture2D(tex_ray_entry, uv);
    vec3 ray_exit = texture2D(tex_ray_exit, uv);
    vec3 globe_pos = texture2D(tex_globe_pos, uv);
    vec3 globe_color = texture2D(tex_globe_color, uv);
    float origin2globe = length(ray_origin - globe_pos);
    float origin2entry = length(ray_origin - ray_entry);
    float origin2exit = length(ray_origin - ray_exit);
    // depth (distance to ray origin) comparison
    // case 1: volume is occluded, terminate ray-casting
    if(origin2globe < origin2entry){return globe_color;}
    // case 2: volume is partially occluded, change ray-exit
    // position if(origin2globe < origin2entry && origin2globe < origin2exit){ray_exit = globe_pos;}
    // case 3: volume is not occluded, keep ray-exit position
    else { }
    // volume ray-casting
    vec3 volume_color = cast_ray(SVT, origin2entry, origin2exit);
    vec3 final_color = composit(globe_color, volume_color);
    return final_color;
}
```

6. Analysis and discussion

The demonstrations were run on a machine with a 2.90-GHz
quad-core Intel Core i52310 CPU and 4 GB of RAM. The graphics
card is an NVIDIA GeForce GTS 450 with 1.0 GB of RAM. The
volume ray-casting framework presented in this paper was imple-
mented using C++, OpenSceneGraph and osgEarth. The core ray-
casting algorithm was written in a GLSL fragment shader.

6.1. Rendering of spherical volume data that cover a large geographic
area

To demonstrate that the method presented can effectively
visualize a GCS-based atmospheric dataset, we ignored the physi-
cal extent of the hurricane and overstretched the original volume
texture to cover a much larger geographic area (Fig. 11).

In the above figure, the images on the left side (Fig. 11a and c)
show that the volume texture covers a geographic area from 23N,
70E to 41.7N, 140E. The images on the right side (Fig. 11b and d)
represent an even larger geographic area, from 0N, 0E to 90N, 180E. Satisfactory rendering results are obtained for either case, suggesting the applicability of the presented method to the rendering of atmospheric volume data on a regional to global scale (in terms of geographic extent rather than data volume).

Arbitrary vertical scaling is an important feature of the method because most atmospheric datasets are relatively thin in the vertical dimension. The following figure (Fig. 12) illustrates how vertical scaling can magnify altitudinal features to help visual analytics.

The thickness of the atmosphere is no more than 1% of the Earth’s radius. Thus, when rendered on the globe without any altitudinal scaling, the Hurricane Isabel dataset appears similar to a sheer 2D image clamped on the Earth (Fig. 12a). Applying an altitudinal scale factor of 20 (Fig. 12b), we can clearly observe the vertical variations in comparison to no scaling at all (Fig. 12a).

### 6.2. Performance analyses

Performance testing is necessary to determine the rendering efficiency of the visualization framework as well as to verify the ray-casting method presented herein. Experiments should be conducted to evaluate the frames per second (FPS) for a computation-intensive visualization system under various constraint conditions (Yang et al., 2011).

Given a constant number of samplings for each ray, the rendering performance for the ray-casting method depends on the number of screen pixels being covered by the volume. A closer
viewing distance or a larger vertical scaling factor causes a larger portion of the screen area to be covered and therefore may lead to a gradual decrease in FPS. Thirteen different viewing distances were chosen to look at the volume center from 17,832 km to 2000 km away, with the vertical scaling factor varying between 1 and 25 (Figs. 13 and 14). A maximum of 128 texture samples

Fig. 13. Rendering performance against viewing distance and vertical scaling factor.

Fig. 14. Visualization at different viewing distances for a vertical scaling factor of 25.
were collected along each ray to be marched. An experiment under the above-mentioned constraint conditions was performed, and the FPS was plotted against the viewing distance and the vertical scaling factor, as shown in Fig. 13 below.

The experiment results (Figs. 13 and 14) found that rendering performance declines quickly with either decreasing viewing distance or increasing vertical scaling when the viewing distance is relatively large. As the viewing position draws closer, the rendering performance is more sensitive to the viewing distance than to the vertical scaling factor. Although the rendering performance was shown to be sufficient to support interactive exploration as suggested by the FPS (Fig. 13) under the given constraint conditions, it can be highly variable depending on the viewing distance and the vertical scaling factor.

6.3. Integration with multi-source data on a virtual globe

Integration with a variety of spatial data sources on virtual globes is a key feature of the presented volume ray-casting framework compared to the standard ray-casting method. To demonstrate this feature, we created a virtual globe scene composed of various spatial objects, e.g., geographic graticules, 3D models, 2D polygons and text labels.

Fig. 15 shows that the volume visualization of Hurricane Isabel is correctly fused with the geographic background on the virtual globe. The depth comparison in our ray-casting algorithm (Section 5.4) can help to achieve correct occlusion effects for the volume and other 3D objects. The alpha compositing produces a translucent effect so that we can still see the 3D objects behind the hurricane volume.

7. Conclusions

The increasing availability of volumetric data in geosciences has created a demand for DE to provide integrated volume visualization capabilities. However, mainstream DE systems such as GE, WW and osgEarth lack the necessary functionalities to visualize large-scale volume data. In response to such needs, we present a volume ray-casting framework that has been demonstrated to be suitable for integration into the open-source virtual globe osgEarth system without requiring any modifications to the rendering engine.

The applicability of the volume-ray casting framework presented in this paper has been tested using the Hurricane Isabel dataset, and the following conclusions have been drawn:

1. The volume ray-casting method can directly visualize atmospheric volume data on a large geographic area.
2. The volume ray-casting method can provide on-the-fly altitudinal scaling.
3. The deferred rendering pipeline can blend the volume effects with a virtual globe.

However, the presented method is still subject to many deficiencies that should be addressed in future work. Alpha blending involving multiple translucent 3D objects is difficult to achieve with deferred shading because the position and color buffers rendered from a virtual globe only provide the final results without recording the intermediate values. Furthermore, volume ray-casting is a costly rendering algorithm, the computation intensity of which increases with the projected screen area. Thus, a closer viewing distance or a larger vertical scaling factor may lead to a sharp drop in frame rates. In the future, other volume rendering methods should be tested and compared with volume ray-casting on virtual globes. Moreover, although the Hurricane Isabel data can easily fit into video memory in the case presented, a massive amount of volume data would still warrant further studies of out-of-core rendering techniques (Yang et al., 2011).

Acknowledgments

This research was supported by the Key Knowledge Innovative Project of the Chinese Academy of Sciences (KZCX2 EW 318), the National Key Technology R&D Program of China (2014ZX10003002), the National Natural Science Foundation of China (41371387), the National Natural Science Foundation of China (41201396), Jiashan Science and Technology Projects (2011A44) and the DRAGON 3 (Project ID 10668).

References
