

EARTHSCAPE, A MULTI-PURPOSE INTERACTIVE 3D GLOBE VIEWER FOR HYBRID DATA VISUALIZATION AND ANALYSIS

A. Sarthou^{a,*}, S. Mas^a, M. Jacquin^a, N. Moreno^a, A. Salamon^a

^aExelis Visual Information Solutions, Toulouse, France – (jean-arthur.sarthou, stephane.mas, marc.jacquin, nicolas.moreno, arnaud.salamon)^a@exelisinc.com

Commission II, WG II/6

KEY WORDS: 3D Viewer, Globe View, Dynamic Footprint, Touch Screen, Shader Programming, Remote Computation, Full Motion Video, Hybrid visualization

ABSTRACT:

The hybrid visualization and interaction tool EarthScape is presented here. The software is able to display simultaneously LiDAR point clouds, draped videos with moving footprint, volume scientific data (using volume rendering, isosurface and slice plane), raster data such as still satellite images, vector data and 3D models such as buildings or vehicles. The application runs on touch screen devices such as tablets. The software is based on open source libraries, such as OpenSceneGraph, osgEarth and OpenCV, and shader programming is used to implement volume rendering of scientific data. The next goal of EarthScape is to perform data analysis using ENVI Services Engine, a cloud data analysis solution. EarthScape is also designed to be a client of Jagwire which provides multisource geo-referenced video fluxes. When all these components will be included, EarthScape will be a multi-purpose platform that will provide at the same time data analysis, hybrid visualization and complex interactions. The software is available on demand for free at france@exelisvis.com.

1. INTRODUCTION

Visualizing data over a virtual globe is now a common thing since the first release of Google Earth in 2005. A lot of different virtual globe viewers exist nowadays, NASA World Wind for example, and some are devoted to the visualization of scientific data (meteorological, environmental, geological or climatic data...) such as SonarScope-3DViewer, created by Altran and IFREMER, which is dedicated to oceanic visualization. More focused on GIS data, ArcGIS Pro from ESRI includes various visualisation and analysis tools. Geosoft has developed (but no more supports) Dapple which was based on World Wind and specialized in geological data visualization. EverView is also based on World Wind and dedicated to geological information and is used to deal with biological studies (Romañach 2014). In (Angleraud 2014), the author presents magHD, a visualization software who aims at using distributed storage to deal with massive amount of 3D data. The author emphasises the advantages of real time manipulation of data since the interpretation of moving images by the brain allows new features to be detected by the user. In the situational awareness domain, Luciad proposes a set of software that allows aircraft positions and sight of view, FMV videos draped on terrain or vector information such as buildings and routes to be displayed.

We propose to gather some of these aspects (GIS, scientific visualization, situational awareness) in EarthScape, a new hybrid visualization and interaction tool. EarthScape is based on osgEarth, an open source terrain rendering SDK capable of reading traditional georeferenced data such as KML files. Although many operations can be achieved with KML files (Ballagh 2011), advanced visualization features (volume rendering, point cloud) are implemented to propose an extensive visualization tool.

The architecture of the software and its core libraries are presented first. Then, we describe the different supported

sources of data. In Section 4, the visualization of volume scientific data using shader programming is presented. Section 5 deals with the touch screen capabilities of the software and some HMI aspects. In section 6, future connections with process and data services products of Exelis (ENVI Services Engine and Jagwire) is discussed. In the last section, conclusions and perspectives are drawn.

2. SOFTWARE ARCHITECTURE

2.1 Architecture overview

The viewer is principally developed in C++. The OpenSceneGraph SDK is the core of EarthScape. The main GIS features are handled with the osgEarth library while the HMI widgets are managed with Qt. Additional libraries are used for more specific purposes, such as OpenCV for the management of the video fluxes. One of the advantages of the open source approach concerns the security aspects. While an internet connection can be required to download third-party data such as raster images, it is mandatory for some users to be sure that their own data will not be transmitted to anyone. Such a security is not totally guaranteed (officially or officiously) with closed source solutions such as Google Earth.

As EarthScape is developed in C++, it cannot be run on a web browser and is not a light client. The interest of using such a fast language at the expense of the ease of installation is to allow heavy computations to be performed by the client. While the data processing is supposed to be performed by a remote server using ESE, EarthScape can compute algorithms such as path finding or physical simulations. A compiled language strategy also gives a wider choice in third party components (such as OpenSceneGraph and osgEarth).

2.2 OpenSceneGraph

OpenSceneGraph (<http://www.openscenegraph.org>) is an open-source 3D graphics toolkit written in C++ and OpenGL. It

simplifies the use of OpenGL and proposes a graph based scene management.

2.3 osgEarth

osgEarth (<http://osgearth.org>) is an open source terrain rendering SDK developed by Pelican Mapping and based on OpenSceneGraph. It deals with tiled imagery, elevation and vector data from local disk or remote servers like OGC, WMS or TMS. As for OpenSceneGraph, osgEarth is a cross-platform library.

There is almost no technical documentation of osgEarth. Fortunately the whole source code is available and questions can be asked to the developers in a user forum. Some parts of osgEarth are quite tricky. It took for instance a lot of time to be able to display multiple draped vector layers in the right order with good performances.

Concerning the tile management, osgEarth completely manages this process and caches the remote data. However, the tile update of cache data is quite slow compared to some other solutions such as Google Earth. It is also possible to implement specific tile servicing processes.

2.4 OpenCV

OpenCV (<http://opencv.org>) is an open source computer vision SDK supported by Willow Garage. Mainly used for robotics applications, it proposes several tools to handle image and video acquisitions and processes. It is a cross-platform library. It is used in EarthScape to read video fluxes.

2.5 HMI

The Qt framework (<http://qt-project.org>), with its advanced widgets and its designer tool, is used to implement HMI widgets and manage the main window (move, resize...) and all widgets animations. Figure 1 shows the application screen with a resource list and a scientific data panel opened on a tablet device. OpenSceneGraph and osgEarth SDK contain classes to facilitate the integration of a 3D view in a Qt widget.

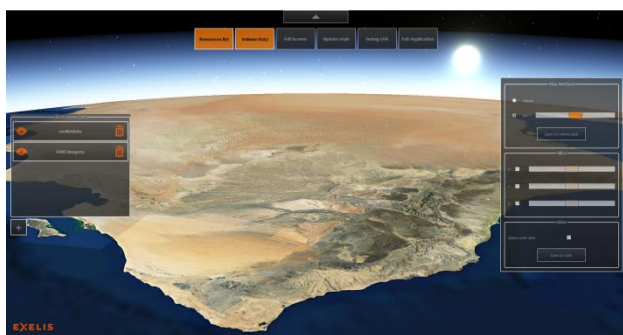


Figure 1 - EarthScape HMI with Qt

2.6 GLSL

Shaders are programs running on the graphic card and handling the way each screen pixel is rendered. The shader uses meshes information, light positions or texture data to compute the color of each pixel. With OpenGL, custom shaders can be implemented in GLSL (OpenGL Shading Language), a C-like programming language. New shaders have been developed for the current project to handle customized real-time rendering of volume data.

2.7 GPS sensor

The viewer may retrieve positions from various localization sources including Windows 8 compliant onboard GPSes and external (USB or Bluetooth) NMEA compliant GPSes.

The viewer expects a socket based service to request at its own frequency and returning an XML response containing mainly Lat/Lon and altitude position and other information like horizontal and vertical speeds, timestamp or heading if available. Once the position is known, osgEarth permits to attach a graphic object to the position and link the camera to a relative position of this moving object: above, inside, aside, below.

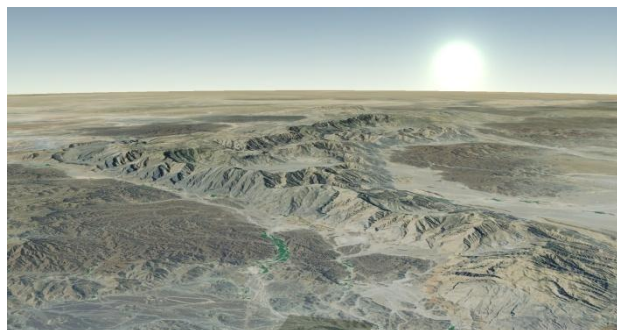
3. DATA VISUALIZATION

3.1 Earth globe content

The native terrain rendering engine of osgEarth is used to display realistic earth globe content, including DEM, satellite imagery and sky effects. Associated GDAL plug-in allows the connexion to OGC servers (ex: WMS for satellite imagery) and the import of several geographic files formats (ex: Geotiff for DEM). Figure 2 shows such a rendering. The osgEarth cache and tile serving principle allows multithread based pyramidal loading of data content during a smooth zoom changing interaction.

3.2 Data management

OpenSceneGraph and osgEarth scene nodes are used to build graphical content on the globe. The application then manages the scene as a list of “displayable resources”. Each “displayable resource” contains a reference to the graphical scene node, plus additional information dedicated to the interactions with it (name to be displayed in the HMI, current visibility, default viewpoint on the data...). The data loading from the local drive is handled outside the “displayable resource” and is currently synchronous. Further work will consist in implementing asynchronous data loading.



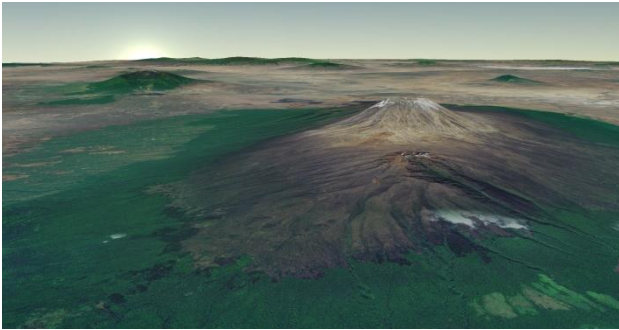


Figure 2 - Realistic earth globe visualisation

3.3 Vector data

Vector layers are accessed through the OGR part of GDAL which supports most of the usual vector formats including shapefile, MapInfo, ESRI, up to OGC WFS and TFS (Tiled Feature). Various draping techniques enable taking in account altitude values if available for on ground positioning. Symbol representation of vector data, even if not as rich as for some other map engines, allows decent and effective rendering. The osgEarth labelling of features has been enhanced and may be combined with icons on a SQL like queries filtering. Figure 3 shows vector data of buildings associated with DEM and imagery.



Figure 3 - Buildings draped on DEM

3.4 Database access

The GDAL plugin provides a direct access to PostGreSQL/PostGIS data tables. Using PostGreSQL/PostGIS instead of shapefiles for vector data has many advantages like combining vector data through views and SQL relational queries using PostGIS vector operators and functions. Performances are even much better than using shapefiles thanks to GIST indexing and PostGreSQL binary cursors. And finally, PostGreSQL/PostGIS might be the best solution for storing dynamic data as the refresh of vector data source has been enhanced.

3.5 Full Motion Video

The application can load and display Full Motion Videos (FMV). Generally acquired by UAVs, FMV files combine image frames and localisation metadata (MISB format in our example). For now, the localisation metadata is extracted from the video file using ENVI and its FMV module, then exported to ASCII readable file. The application uses OpenCV to load and browse the video frames, and OpenSceneGraph texture mechanism to drape them on the terrain. Figure 4 shows a UAV acquisition localised on the map using the MISB metadata.

Figure 5 shows this same video but with a fake localisation on the Alps. Concerning the computational cost, this function does not affect the overall good framerate of the application.



Figure 4 – Geo-referenced UAV acquisition



Figure 5 - Video draping on terrain

3.6 LIDAR data

Geo-referenced LiDAR data is read from binary files generated by ENVI LiDAR from classical LiDAR formats (LAS). The resulting point cloud is directly rendered by OpenGL. An example (the city of Avon, NY, USA) is shown in Figure 6. No optimisation such as adaptive point display is performed for now for LiDAR data. For the current example, the 5 millions-points set is displayed and manipulated in real-time on a usual laptop computer. Using an open source library such as Point Cloud Library could be an option to manage LiDAR point sets of high weight.

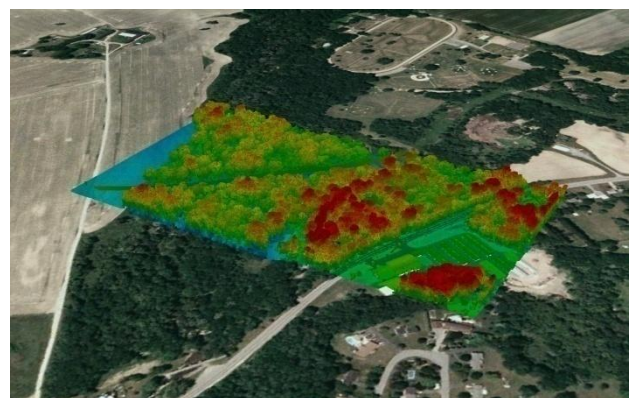


Figure 6 - LiDAR data over geo-referenced images (Avon, NY, USA)

3.7 Volume Data

Volume data are extracted from NetCDF files using IDL. For now, only scalar data such as temperature is considered. The way these data are displayed is described in Section 4.

3.8 Differed time and animations

The application can import trajectory files containing time stamped 6D poses (time, longitude, latitude, altitude, yaw, pitch, and roll). Models of standard mesh format can be imported and animated along the trajectory. This feature allows the representation of any type of vehicle trajectory. Figure 7 gives an example of a train trajectory representation. Camera navigation modes have been implemented to enable the overview of the full trajectory, the focus on the animated item, or the view from inside the item (pilot view).

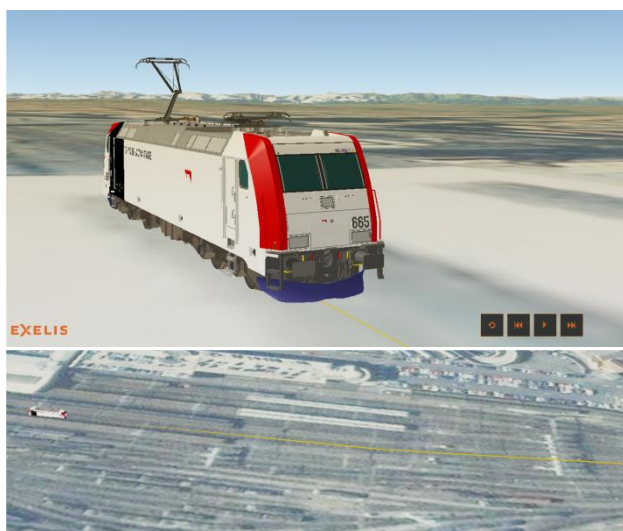


Figure 7 - Train model and trajectory

A central component is responsible for maintaining the current virtual time of the scene. It is updated at each 3D rendering. As shown in Figure 8, any entity interested in the virtual time observes this time server and takes its changes into account. For now, any 3D scene animation and the video files playback are synchronized like this. This architecture would easily allow the addition of time triggered items such as a physical simulator, or HMIs to display mobile metadata. The application proposes a media player widget which triggers play, pause and jumps on the virtual time server. The user can thus navigate in the virtual time.

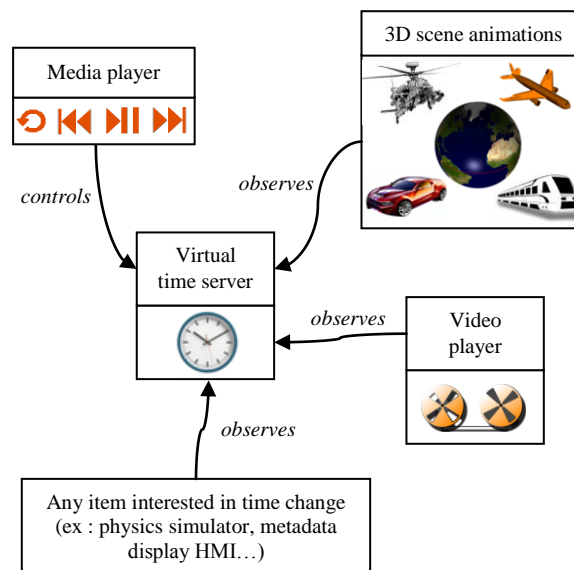


Figure 8 - Time management architecture

4. VOLUME RENDERING

4.1 Shader programming with GLSL

GLSL, the default OpenGL shading language, is used in EarthScape to create *ad hoc* shaders. There are different kinds of shaders taking place at different levels of the graphical pipeline. In the present work, two kinds of custom shaders are created. The first one is the vertex shader. It runs once for each vertex of the 3D scene and determines the on-screen position of the vertex. The second kind of shader is the fragment shader which is run once for each pixel of the screen. It computes the color of the pixel according to various data such as lights positions or surfaces properties.

The shaders are compiled and run by the GPU of the graphic card instead of the CPU. Data from the main program managed by the CPU to be used with shaders have first to be transferred from the RAM or the hard-drive to the graphic card memory. One of the main advantages of the shader approach is that each shader instance (which computes a pixel color or a vertex position) is executed by one GPU core. As many GPU cores (tens to thousands depending on the graphic card) are present on a single GPU, using shaders offers a simple and efficient way to perform parallel computing. Shaders give an easy access to a great parallel computational power so visualization and extraction algorithms can be implemented directly on shaders. In EarthScape, isosurfaces, slices or dense rendering of 3D volumes are computed using the GPU and the volume data in the graphic card memory. Such visualization methods are efficient and necessary when navigating through volume data as the analysis of volume information on screen is not as trivial as for 2D data.

OpenSceneGraph allows to easily assign a specific shader for each 3D geometry. Our home-made shaders are used on box geometries containing volume data while the standard OpenSceneGraph shader is used for the other geometries.

4.2 Managing scientific data

When a scientific data set has to be displayed, a range of scientific values is generally associated with a color ramp. For this purpose, a function F_{sc} has to be defined for each scientific value V_s to translate it to a displayed color data V_c .

As OpenGL and GLSL provide a fast and easy access to the texture memory of the graphic card, the data to display is stored as a 2D or 3D texture. Textures are encoded in RGBA format, and each channel can store an integer or a float value which offers a limited range of values. This range can be smaller than the range of the scientific data to display. A first solution consists in computing the color $V_c = F_{sc}(V_s)$ of each value of the scientific data set on the CPU and then to store the new color dataset in the texture memory. However, it means that all V_c have to be computed from V_s on the CPU. In case of real-time modification of F_{sc} , this procedure could be required to be performed for each frame. As new data set has to be transferred to the GPU, this method is not optimal in term of performances.

Another approach used here is to transfer the function F_{sc} and the scientific data to the GPU and to compute $V_c = F_{sc}(V_s)$ on it at each frame. To be stored in texture memory (which provides a limited type of data), each scientific scalar value of the scientific dataset is converted to a vector of four 8-bit integers to be stored in a RGBA texture. The maximum Max_s and minimum Min_s values of the scientific data set are considered and each scalar value between Min_s and Max_s is mapped between $[0, 0, 0, 0]$ and $[255, 255, 255, 255]$.

4.3 Volume rendering

The volume of data can be visualized as a translucent block coloured by the data. The rendering is performed with a volume ray-casting algorithm (Kruger, 2003). The data is contained in a box, and we consider a pinhole camera model. The camera is located at point C . We consider a virtual grid between the camera and the data to render. Each element of the grid is a pixel of the screen. Rays are cast from C through the center of each pixel of the grid. Some rays hit the volume box and goes through it step by step (small steps increase the rendering quality but also the computational time), passing through many voxels and accumulating information. For each of these rays, the corresponding pixels are coloured according to the information gathered along the ray. Figure 12 shows such a rendering of temperature data.

4.4 Isosurface rendering

We consider the isovalue α that defines the isosurface. For each ray progressing inside the volume data box (corresponding to one pixel of the screen), the value of the scientific data α_n at the location of the step number n of the ray is checked and compared to α_{n-1} , the data value at the previous step. If $\alpha_n < \alpha < \alpha_{n-1}$ or $\alpha_{n-1} < \alpha < \alpha_n$, then the ray has crossed the isosurface. Then, the resulting opacity of the corresponding pixel on the screen is set to 1. The computation of the color of the pixel is performed according to the light sources positions and intensities, and to the local isosurface normal. This latter is obtained by calculating the normalized gradient of the volume data. For the present work, a Phong shading (ambient, diffuse and specular lights are considered) is used (Phong, 1975). The Figure 9 shows an isosurface rendering for a volume data.

The classic way to render an isosurface is to compute on the CPU the mesh of the isosurface from the data set and to send it

to the GPU to display it. Compared to the present algorithm, this approach has two drawbacks. The creation of the isosurface mesh requires a quite complex algorithm, and in case of real-time modification of the isovalue, a new mesh of the isosurface has to be generated and transferred many times to the GPU.

In the present method, no isosurface mesh is generated and each pixel computes its color from the raw volume data set and the lights properties only. This method is fast and allows real-time update of the displayed isosurface when the user changes the isovalue. It is a good demonstration of the shader capability to do more than simple display operations.



Figure 9 – Isosurface rendering of volume data

4.5 Slice plane rendering

Let us consider $a \in [0,1]$ the relative position of the plane in the box along a given axis. For each ray traversing the box, its relative positions in the considered axis p_n is compared to its position p_{n-1} at the previous step. If $p_n < a < p_{n-1}$ or $p_{n-1} < a < p_n$, the ray has crossed the slice plane, so the color of the pixel can be determined by interpolating the data between p_{n-1} and p_n . A rendering example is shown in Figure 10. As for the iso-surface algorithm, the present shader algorithm is very simple to implement once the ray-tracing method of the volume rendering algorithm is available. It also allows real-time interaction of the user on the slice planes positions.

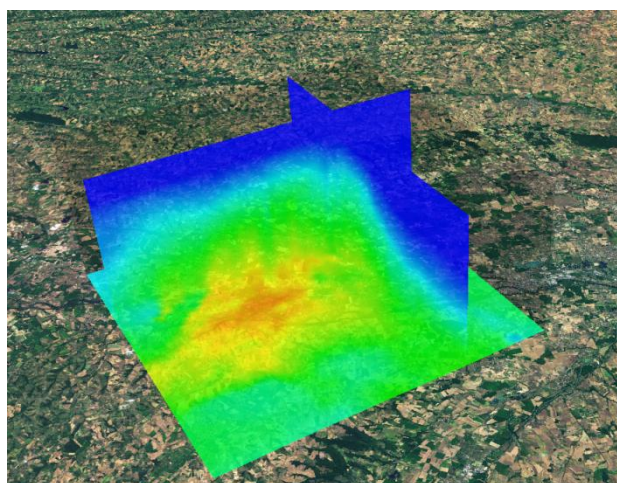


Figure 10 – Rendering of slice planes of volume data

4.6 Geo-referenced volume data rendering

The ray-casting algorithms for volume rendering are generally used on boxes as it simplifies the rendering algorithm. In the case of georeferenced data (e.g. the air temperature), the information is available in a significant range of latitude, longitude and altitude. Hence, the data is present in a volume that cannot be approximated by a box. It is possible to include the scientific data in a bigger box that contains it (Li 2013) but some part of the box has no use and it generates unnecessary calculations and memory use. In (Liang 2014), Liang *et al* use a ray-casting algorithm on a curved box by converting the position of each step of each ray from Cartesian to the unit curvilinear local frame of the curved box. The Figure 11 shows the shape of a curved box used as base volume for volume rendering. This method is used in EarthScape.

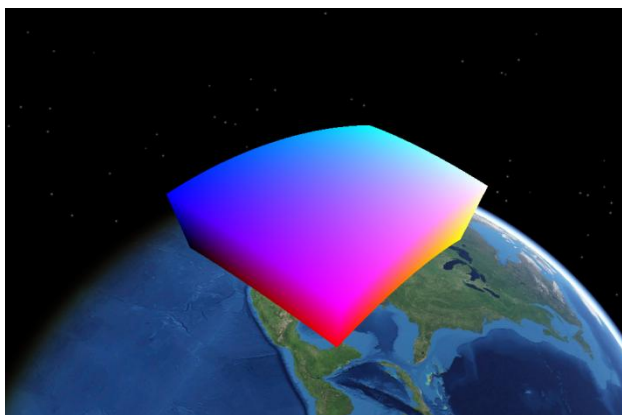


Figure 11 - Curved box used for volume rendering

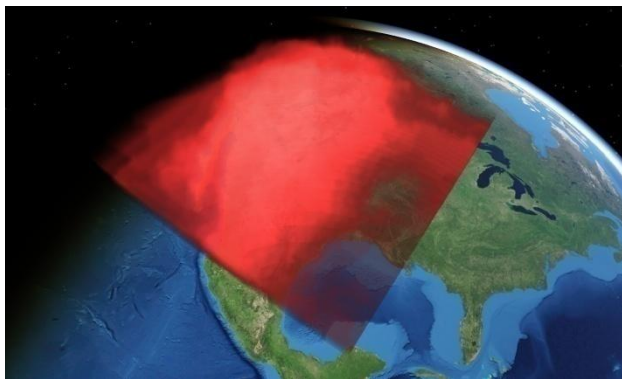


Figure 12 - Volume rendering of air temperature values over North America

4.7 Performances

The volume rendering is the most resource demanding part of EarthScape. The computational power required to render a frame with a volume data first depends on the on-screen size of the data. The second parameter which has a direct influence is the discretization step of the ray. A smaller step induces more visual quality by producing a more smoothed rendering (otherwise the volume seems to be a stack of layers instead of a real volume) but also requires more steps and increases the computational cost of the shader. For a good compromise between speed and quality, the number of steps for the rays is set to 100. With this configuration on a computer with an i7-2700QM CPU with a 2.40 GHz clock and a nVidia Quadro

1000M, a framerate of 20 frames per second is obtained in HD (1920×1080) resolution.

5. HMI AND TOUCH SCREEN CAPABILITIES

5.1 HMI widgets

osgEarth proposes classes dedicated to HMI prototyping such as containers, horizontal/vertical boxes, images, frames, labels and buttons. Those widgets have been initially used for Head-Up Display HMI in the application. The advantage was that it required no additional HMI library, and that widgets could be handled with the animation engine of OpenSceneGraph. However, it quickly appeared that those widgets are quite basic and troublesome with touch screen interactions. Indeed, touch filters had to be implemented to dissociate real to fake touch events. Qt library was thus benchmarked. The result is that Qt widgets are not subject to parasitic touch events and that Qt designer offers a gentle way to build more complex HMIs. Hence, Qt is now used in EarthScape.

5.2 Touch interactions with the globe

The application also offers touch screen interactions with the globe. The osgEarth native earth manipulator was used for standard navigation around the globe. This manipulator proposes several goodies such as node tethering (to keep focus on a moving node), or automatic displacements (to reach a chosen viewpoint with a smooth animation). Touch interactions were also implemented to perform specific actions on the globe (move or rotate projected textures, add annotations on the globe).

Touch screen interaction can eventually be problematic with osgEarth. In the current version of EarthScape, some patches that filter the user actions have been added to the touch screen management to tackle those issues. Some development teams which use Qt and osgEarth are also using a third-party library such as Gesture Works to perform touch interactions.

6. CONNEXION TO ENVI SERVICES ENGINE AND JAGWIRE

6.1 ENVI Services Engine

We could qualify EarthScape as a semi-light client (or semi-heavy). It is not a web client and it does not analyse data. To perform data analysis, we are currently working at connecting EarthScape to ENVI Services Engine (ESE) (Jacquin, 2015). The principle is to select an area on the globe and to use ESE to require an analysis (a *task*) performed by a remote server. ESE tasks can include IDL, ENVI or ENVI LiDAR functionalities.

ESE exposes tasks capabilities to the client through JSON format. It appears to be simple and efficient for parsing and converting tasks input parameters in order to include them in EarthScape widgets. ESE tasks results are also exposed as JSON to the client. Whenever results are tabular, vector or raster data, they may be served with the help of a map engine providing OGC standard services like WFS, WCS or WMS. GeoServer or Mapserver may be considered to offer the compliant services to integrate the effective result into EarthScape.

6.2 Jagwire

Jagwire, developed by Exelis, can be considered as a new catalog for storing any kind of geographic data extended to

FMV (Full Motion Video) and WAMI (Wide Area Motion Imagery) assets as well as LiDAR data or still imagery. Jagwire proposes simple dedicated viewers for each type of data but provides Web interfaces fluxes to access assets from any client. Using the FMV and WAMI capabilities of EarthScape, our aim is to be a client of Jagwire to display multiple FMV with real-time footprint evolution.

7. CONCLUSIONS AND PERSPECTIVES

A global visualization and interaction tool has been presented. EarthScape can deal with many different data types at a same time. Vector and raster display of geo-referenced data of multiple formats is provided by the OpenSceneGraph and osgEarth libraries. Dedicated shaders have been developed to display scientific data allowing volume, isosurface and slice plane visualizations. It shows the usefulness of shaders to easily perform a lot of operations such as feature extractions.

EarthScape allows users to perform multiple interactions with the data and the computational power of the heavy-client approach will permit to implement complex algorithms such as interactive physical simulation or path finding. For Exelis, one of the goals of EarthScape is to have a visualization and data gathering platform from which custom applications will be developed.

Future development on visualization will be devoted to include the display of the sensor footprint from the position and attitude of the sensor. It will probably be performed on GPU by a dedicated shader as the computation of the intersection between the terrain and the sensor sight needs a lot of resources to be computed in real-time. The next step will be to perform data processing by sending requests to a remote ESE. Concerning Jagwire, the aim is to use EarthScape as a heavy-client capable of processing Jagwire provided data using ESE. The connection with ESE will be a very important asset as it will opens EarthScape to a virtually infinite number of data analysis processes. It will be necessary to develop new visualization tools to handle as many ESE outputs as possible.

A mid-term objective will also be to add more data formats including NetCDF, LiDAR .las, and Jpeg2000 formats.

With all these components, EarthScape will be a multi-purpose platform to analyse, merge and visualise hybrid data and perform complex interactions in a unified environment.

The current binary of the software is available on demand for free at france@exelisvis.com.

REFERENCES

- Angleraud, C., 2014. magHD: a new approach to multi-dimensional data storage, analysis, display and exploitation. *IOP Conf. Ser.: Earth Environ. Sci.* 20, 012035.
- Ballagh, L., Raup, B., Duerr, R., Khalsa, S., J., Helm, C., Fowler, D., Gupte, A., 2011. Representing scientific data sets in KML: Methods and challenges. *Computers & Geosciences* 37, pp. 57–64.
- Jacquin, M., 2015. Geoanalytics on-demande paradigm shift ENVI services engine. ISPRS Archives *GeoBigData – ISPRS Geospatial Week 2015*.
- Kruger, J., Westermann, R., 2003. Acceleration techniques for GPU-based volume rendering. In: *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, Washington, DC, USA, pp. 287–292.
- Li, J., Jiang, Y., Yang, C., Huang, Q., M., 2013. Visualizing 3D/4D environmental data using many-core graphics processing units (GPUs) and multi-core central processing units (CPUs). *Computers & Geosciences*, 59, pp. 78–89.
- Liang, J., Gong, J., Li, W., Ibrahim, A. N., 2014. Visualizing 3D atmospheric data with spherical volume texture on virtual globes. *Computers & Geosciences*, 64, pp. 81–91.
- Phong, B., T., 1975. Illumination for computer generated pictures. *Communications of ACM* 18, 6, pp. 311–317
- Romañacha, S., McKelvy, M., Conzelmann, C., Suirb, K., 2014. A visualization tool to support decision making in environmental and biological planning. *Environmental Modelling & Software* 62, pp. 227–229.