Rapid Development of Scientific Virtual Reality Applications

Scott Sorensen, Zachary Ladin, Chandra Kambhamettu
University of Delaware
Newark DE, USA
Email: sorensen@udel.edu

***Abstract*—Visualizing and working with large scale scientific image data can benefit greatly from advances in consumer Virtual Reality and game development tools, but presently only limited applications have leveraged these. We present a set of techniques for mapping scientific images and data manipulation tasks into virtual reality applications with an emphasis on short development time and high quality user experience. Existing techniques often require large teams to develop applications that suffer in quality or require extensive development time. Using off the shelf Virtual Reality hardware and freely available development tools we present a set of techniques for making immersive and interactive data visualization and manipulation applications. We demonstrate these techniques with a series of Virtual Reality apps to produce high quality immersive applications in incredibly short periods of time. These applications show large datasets and extracted high level information geared towards image understanding. We illustrate that these techniques can be integrated into existing data acquisition, image processing and machine learning work flows. With these techniques we hope to encourage future development of scientific applications for emerging Virtual Reality platforms.**

1. Introduction

Recent consumer Virtual Reality (VR) systems have enabled development of many new VR applications, allowing for these applications to reach a wider user base than ever before. New head mounted displays (HMD) are high resolution, lightweight, and have been built for mass consumption by gamers. Game developers have put considerable effort into 3D navigation, user interfaces, and interaction and have made good headway in VR development. The technology developed for immersive VR gaming in 3D environments has far ranging applications outside of video games, and in this work we will demonstrate applications of scientific data visualization using gaming hardware and game development software.

Large scale datasets across scientific disciplines are also becoming ubiquitous, and visualizing and interacting with data is increasingly desirable. While data visualization is well established, applications using VR are still in their infancy. We will demonstrate a series of techniques for processing and visualizing scientific data in VR that allow for rapid development of high quality applications. The techniques presented leverage existing game development software and hardware. We demonstrate these techniques with a series of scientific applications using image data from traditional camera systems as well as continental scale and global scale geospatial data.

2. Background

With the release of the Oculus Rift and HTC Vive in Spring of this year, 2016 has been been dubbed the year of VR by gaming and news outlets [1], [2]. The hardware is still expensive by consumer standards, but it is widely available, and for the first time, there is a viable platform for developing VR software with mass market capabilities. While the hardware has been marketed as a a gaming platform, there is an increasing push for non-gaming applications on multiple platforms. Oculus has a variety of applications and ”experiences” on their platform store, and the newly launched Viveport is an app store for VR content that does not fit the mold of a game.

Many of these applications and many of the games developed for virtual reality platforms have been produced using game development engines like Unity, or the Unreal Engine 4. These engines provide a framework for content development and creation. The rise of 3D games has led to a plethora of advances in real time graphics capabilities including techniques for ambient occlusion [3] and near real time motion capture [4]. Using game engines allows for developers to take advantage of many developments and optimization efforts without re-implementing from scratch. Furthermore, game engines handle movement, physics, and common schemes for interaction in 3D. In this work, we will demonstrate a workflow for developing scientific visualization VR apps with the Unreal Engine 4, and discuss the role this sort of development can play in prototyping.

3. Methods

In this section we discuss the techniques used to develop Virtual Reality applications for scientific visualization. This work does not aim to be a tutorial or instruction book on game development, but instead focuses on elements of development that make scientific applications unique.

978-1-5090-3284-6/16/$31.00

These techniques are focused on generating mesh models and corresponding texture that can be easily imported and utilized by game engines, which in turn, can be rapidly converted into virtual reality applications.

Game engines have utilized 3D mesh objects since the 1990s, and modern games consist of hundreds of thousands of polygons onscreen at a given time. The realtime graphics pipeline for 3D games is well suited for 3D meshes with UV texture parameterization with associated materials and textures. By inserting purpose-built 3D meshes and associated output textures we can create dynamic VR models, and allow for natural observation and more intuitive physical interaction.

3.1. Reconstruction

Reconstruction is one of the main areas of study in computer vision, with many different techniques such as stereo, structure from motion, and many others. Visualizing these models using conventional means is undesirable because 3D viewing applications can be difficult to use, and each has a unique, and sometimes unintuitive user interface. By comparison, the use of VR headsets allows for not only an immersive stereo view of the scene, but motion paralax and natural movement which provides an intuitive sense of scale and ease of use. Creating simple VR applications with reconstructed models is straightforward using the Unreal engine. Any reconstructed point cloud simply needs to be converted to a mesh and imported into the engine as either an FBX or OBJ file, and then it can be directly imported and added to the 3D viewport. The Meshlab utility [5] provides an excellent set of tools for surface fitting 3D points and means of converting between file formats, as well as many other tools for 3D modeling.

In Fig. 1, we show a screenshot from a VR app that contains models created using a low cost Microsoft Kinect, and a more sophisticated lidar setup. The Kinect SDK provides a way to directly export the mesh files. The Lidar system outputs point clouds, and we have used Poisson reconstruction [6] to fit a mesh. Both scans were simplified using quadratic edge collapse decimation[7] to reduce to polygon count to maintain high frame rate in VR. Image-based reconstruction techniques can be used for texture mapping, resulting in more realistic models in VR. Many commercially available photogrammetry applications will export fully texturemapped meshes, and these can be directly imported into the engine. When developing reconstruction techniques in house, texture mapping can be easily achieved by projecting 3D points onto an image frame to compute UV parameters. Fig. 2 shows an application using models created using both commercial SFM applications (Autodesk 123D Catch) and in house stereo systems.

3.2. Mesh Generation

While reconstructions are a natural fit for VR applications, There are many other types of images that translate nicely into VR. While any 2D image can be represented

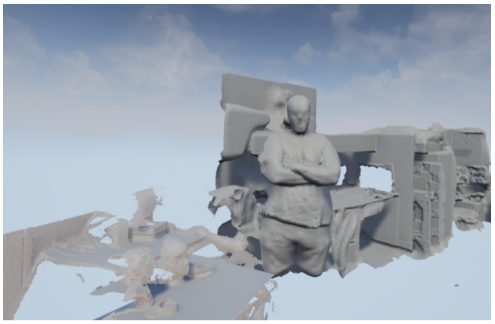

Figure 1: A screenshot of a VR application with textureless models creted using a Microsoft Kinect and a lidar scan.

Figure 2: Screenshot from a VR application with models created using SFM and stereo, as well as atmospheric particle effects.

on a simple rectangular mesh, there is little benefit to viewing these in VR over traditional displays. We advocate a technique for mesh generation to create geometric canvases on which image data of many types can be reprojected and visualized. To create these meshes, we will create parametric meshes programmatically and generate vertices, faces, vertex normals, and texture parameters based on image data. This technique has wide ranging applications, but in this section we will illustrate our work using geospatial data accquired from NASA's Moderate Resolution Imaging Spectroradiometer, or MODIS [8]. There are two satelites with MODIS instruments aboard that collectively image the entire globe every 1-2 days. MODIS captures many spectal bands that pertain to atmospheric, surface, and oceanographic properties of the planet, and here we will focus on two, namely surface temperature and an index of vegetation reflection.

MODIS data, like most remotely sensed data is georeferenced, meaning there is a mapping between pixel coordinates in the image to real geographic position, and we use this mapping to generate meshes and for texture mapping. For geospatial data, projection is an important variable, and we will discuss two that we have used for VR. We have developed a VR globe and a 3D analog of Web Mercator. We will discuss generation of the globe first as it is simpler, and then extend the techniques for 3D Web Mercator illustrating our technique with a map of the conterminous United States.

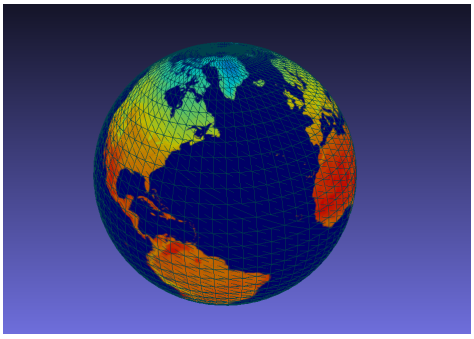

Figure 3: Colormapped MODIS derived mean monthly surface temperature imagery applied to the generated globe.

To generate a globe we will model the planet as a sphere which we will programmatically generate. We uniformly sample $0 \leq \theta \leq 2\pi$ and $\frac{-\pi}{2} \leq \phi \leq \frac{\pi}{2}$ in steps of δ. The choice of δ allows us to tune the polygon count of the resulting mesh, for higher quality or to maintain higher framerate in VR. Vertices are the set of all points $P = [\theta, \phi, 1]$ in polar coordinates. The UV coordinates for each vertex are normalized by

$$[U, V] = [1 - \frac{x}{2\pi}, 1 - \frac{y + \frac{\pi}{2}}{\pi}] \quad (1)$$

Since this is a unit sphere, the vertex normal is the vertex location for each point. We explicitly index faces by creating a set of upper and lower triangles from the uniformly sampled points. This completes the globe model, and we can apply any geospatial images as texture, as long as they are projected in web Mercator for the entire globe. Fig. 3 shows the model texture mapped with MODIS imagery. This model was made using $\delta = 5°$, and contains fewer than 3000 vertices, which is of satisfactory resolution and minimally intensive for computation.

To generate 3D web Mercator maps we have used topographic data from the National Geophysical Data Center[9]. We first generate a binary mask for the area in the conterminous, by rasterizing shapefiles. This allows us to generate vertices within the bounds of the USA. To generate the mesh we again uniformly sample points, but this time we sample $Lat_{min} \leq Lat \leq Lat_{max}$ and $Lon_{min} \leq Lon \leq Lon_{max}$ where Lat_{max} is the maximum latitude, Lon_{min} is the minimum longitude etc. These values are obtained by padding a small area (one degree) around the points in the shapefile. Points are uniformly sampled in steps of δ degrees.

For each point, we find the height by extracting the value from the topographic map, and a binary value indicating whether the point is within the bounds of the USA. Since the units of our map are in degrees, we must scale the height, which is in meters, and for this work we have chosen to scale the values by 0.001. Normals are computed across the whole set of vertices using the local neighborhood of points[10]. We construct faces by again indexing triangular faces over the grid of points, but only including faces where all 3 vertices are valid points within the US. Texture mapping uses the exact same normalization scheme as the globe, because this allows us to use the same geoscale data for multiple models in the same application. Fig. 4 shows a model generated with approximately 36,000 vertices.

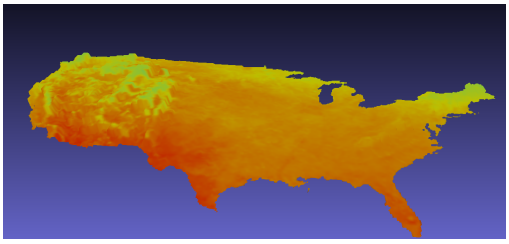

Figure 4: The generated model of the conterminous USA with overlaid MODIS data

We have additionally used this technique to generate meshes for image data from small camera systems. A mesh can be generated by raytracing points in the image to corresponding scene geometry. We have used this for camera systems mounted on a ship which observe a large area around the vessel. The mesh is generated by a set of rays originating at the camera, and intersecting with sea level. This allows images from the camera system to be reprojected onto the mesh, preserving scale at sea level. While this requires prior knowledge of the scene, it is well suited for fixed cameras such as those in security systems, which could be calibrated and meshes designed in advanced.

3.3. Application Development

After creating the mesh models we save them as OBJ files and import them directly into the Unreal Engine. By dragging the imported assets into the 3D viewport, positioning, and scaling them appropriately, we are ready for basic VR visualization, by clicking VR Preview. We will want to set up textures and material overlays, but for 3D models with no texture information, we are already done! The Unreal Engine supports complex dynamic materials, and while it is beyond the scope of this work to go in depth here, we will discuss some specific use cases with geospatial data. With the meshes we have created we can apply any geospatial images with the proper projection, and we can even show time series data by using media textures.

The Unreal Engine's media textures allow for videos to be applied to materials. Our applications with MODIS surface temperature band and vegetation reflectance indices show the progression of seasons and climate trends. Videos can be played across a network and via streams, opening up possibilites for real time applications with live image data. Furthermore, any image stream can be used with the same texture mapping, meaning any image processing step can be used so long as it preserves image geometry. Detection and classification results can be overlaid directly on images and displayed graphically with no issue. While the Unreal Engine offers many options for advanced shaders in the material properties, often with image data, a simple emmisive color is sufficient, with no base or specular component. The intent of designing visualizations is to present the image data

in a clear way, and this has the added benefit of reducing the cost of computing scene illumination.

3.4. Interaction

VR presents new methods for not only observing data, but new means of interacting and manipulating data in 3D. The Unreal Engine supports a wide array of peripherals, and common locomotion schemes. Standard keyboard and controller inputs work with no need to configure anything. These schemes are relatively intuitive and users with any experience with games can operate the controls with minimal instruction. Motion controls allow for even more natural interaction and manipulation of 3D data. With motion controls like those of the HTC Vive, users can physically grab and move meshes in the VR application just as they would grab a real object. This allows users to do things like view a region on the globe by spinning it, or looking closely at a specific part of the map by bringing it close to their face. This control scheme is now supported by a free plugin in the Unreal Engine, making it easy to implement.

4. Example applications

We have developed many VR applications with these techniques, ranging from visualizations of simple textureless meshes, to interactive motion controlled visualizations with multiple media textures playing back simultaneously. The Unreal Engine implements many features that would be time consuming for researchers to implement on their own. We use VR as an integral part of prototyping and visualization, and have developed applications with a wide range of data from thermal images collected in the Arctic, to 3D data from MRI and stereo images accquired in vivo by a stereo laparoscope. We have discussed geospatial image data as an example, and we have constructed applications with these meshes that allow the user to observe the full timeseries recorded by MODIS sensors. We have colormapped the surface temperature and vegetation index bands for the period of 2000 to 2013.

In our application, one month plays back in a single second, and a user can walk around, grab, hold, and even throw the meshes. The walls show the current date for the playback, and there is a legend showing what the colormapping means in real units. Fig. 5 shows an app with two globe meshes with both bands. This app is great for visualizing seasonal change, and you can see the effect of this on vegetation simultaneously. Fig. 6 shows the 3D web Mercator mesh with surface temperature, and it clearly shows the effect of elevation on temperature as the area in the Rocky mountains is significantly cooler.

5. Conclusion

In this work we have presented a scheme for quickly developing scientific Virtual Reality visualization applications. Using gaming hardware and game development engines, it is possible to rapidly build high quality applications for a variety of VR platforms with minimal development time. We have demonstrated an approach to generating 3D meshes from image data, and have illustrated this with geospatial data. We have used these techniques for the development of many applications in our lab, and have begun to use VR as part of our workflow of prototyping and development for many projects. As the cost of entry decreases with VR, the demand for scientific and other non-gaming VR apps will increase, and it is our hopes that these techniques will help those developing apps to meet that demand.

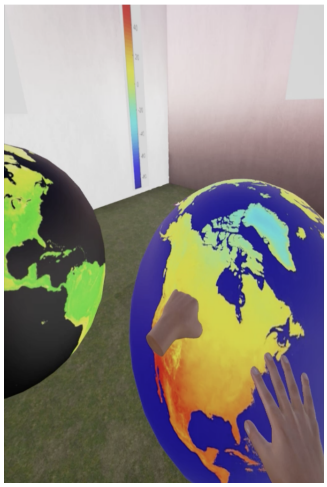

Figure 5: A screenshot of the VR app with two globes showing timeseries of MODIS data.

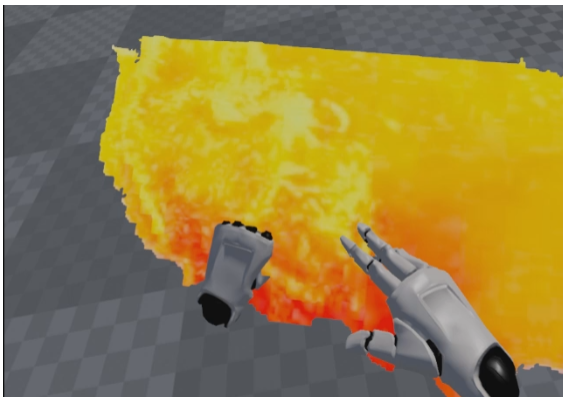

Figure 6: A screenshot of the VR app with a 3D web Mercator map.

References

[1] Chris Morris, "Is 2016 the year of virtual reality?," *Fortune*, vol. Tech, dec 2015.

[2] "The year of vr," *GameInformer*, dec 2015.

[3] Daniel Wright, "Dynamic occlusion with signed distance fields," in *ACM SIGGRAPH*, 2015, vol. Advances in Real-Time Rendering in Games.

[4] Steve Caulkin Vladmir Mastilovic Tameem Antoniades, Kim Libreri, "From previs to final in five minutes: A breakthrough in live performance capture," in *ACM SIGGRAPH*, 2016, vol. REAL-TIME LIVE!

[5] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia, "Meshlab: an open-source 3d mesh processing system," April 2008.

[6] Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe, "Poisson surface reconstruction.," in *Symposium on Geometry Processing*, Alla Sheffer and Konrad Polthier, Eds. 2006, vol. 256 of *ACM International Conference Proceeding Series*, pp. 61–70, Eurographics Association.

[7] Hugues Hoppe, "New quadric metric for simplifying meshes with appearance attributes," in *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, Washington, DC, USA, 1999, VISUALIZATION '99, pp. –, IEEE Computer Society.

[8] NASA, "Modis l2. version 6. nasa eosdis land processes daac, usgs earth resources observation and science (eros) center, sioux falls, south dakota, (https://lpdaac.usgs.gov), accessed 08 15, 2016, at http://reverb.earthdata.nasa.gov," Online, 2009.

[9] Boulder Colorado NOAA, National Geophysical Data Center, "Data announcement 88-mgg-02, digital relief of the surface of the earth," online, may 1988.

[10] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle, *Surface reconstruction from unorganized points*, vol. 26, ACM, 1992.