Volumetric Visualization

Full Paper

James H. Money
Idaho National Laboratory
Idaho Falls, Idaho
james.money@inl.gov

Landon S. Woolley
Idaho National Laboratory
Idaho Falls, Idaho
landon.woolley@inl.gov

Figure 1: Computer Assisted Virtual Environment

ABSTRACT

This paper will describe a volume visualization tool capable of rendering large scale simulated and tomographic datasets in real time on commodity hardware. The ability to render and manipulate immense amounts of data in a three-dimensional environment is an instrument that could benefit the medical field, biological studies, engineering education, and physics research. The software is open source which allows other programmers to add additional functionality where needed. The Unity game engine allows the software to be published on twenty nine platforms [4]. This work will serve as an essential foundation for telecollaborative dataset visualization and analysis at the Idaho National Laboratory.

CCS CONCEPTS

•Human-centered computing → Visualization toolkits; *Scientific visualization; Information visualization;* **•Computing methodologies → Parallel algorithms; Ray tracing;**

KEYWORDS

dynamic memory allocation, volumetric rendering, scientific visualization, fragment shader, rasterization, ray casting, ray marching, clipping planes

ACM Reference format:
James H. Money and Landon S. Woolley. 2017. Volumetric Visualization. In *Proceedings of PEARC17, Idaho Falls, ID, USA, December 1, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3093338.3093344

PEARC17, Idaho Falls, ID, USA

DOI: http://dx.doi.org/10.1145/3093338.3093344

1 INTRODUCTION

During an MRI scan of a human brain, doctors are only able to view data slice by slice with limited three dimensional capability. Implementing volumetric rendering can solve this problem. By rendering the data in real time and combining each slice, doctors can study their data in a three dimensional environment. Viewing the data this way produces a more natural feel and can lead to additional discoveries that would be far more difficult to achieve using a traditional two dimensional plane. Unfortunately, manipulating large amounts of data can be quite taxing for a computers CPU and GPU causing the frames per second to drop dramatically while the processor attempts to handle the vast data. With that being said there are various ways to improve the frame rate and optimize performance.

2 BACKGROUND

The volumetric visualization tool was a project born in the Center for Advanced Energy Studies (CAES) and was matured by James Money and his team [3]. In scientific visualization and computer graphics, volume rendering is a set of techniques used to display a 2D projection of a 3D discretely sampled data set, typically a 3D scalar field [2]. A typical 3D data set is a group of 2D slice images acquired by a CT, MRI, or MicroCT scanner [6]. Volumetric rendering allows researchers in any field of study to visualize their data in a three dimensional environment [7]. Problems such as low frames per second and distorted data are no longer present thanks to dynamic memory allocation and the use of complex algorithms [1].

3 PROJECT OVERVIEW

This project has three key sections that make up the volumetric visualization. First is volume rendering, which is accomplished by the use of a fragment shader. A fragment shader will process a fragment generated by the rasterization into a set of colors and a single depth value [5]. Second are the functions that perform the ray marching and ray casting. These functions perform data visualization. The information that is collected will determine the red, green, blue, and alpha values of each pixels. Third are the tools that are used to manipulate and understand the data.

4 VOLUMETRIC RENDERING

By attaching a fragment shader to a cube, Unity is able to render the volumetric data inside the cube itself. However, depending on the dimensions of the data that has been loaded, the data can appear to be skewed, stretched, or completely unrecognizable. This problem is easily remedied. By creating a metadata file that is associated with each .RAW volumetric data file, Unity can load each file without the necessity of manual adjustments. Doing so ensures that our data will never be distorted. The metadata file holds the specified dimensions for each render as well as various other options such as rotation, step iteration value, and alpha value.

4.1 V-Sync

One simple way to increase the performance is by setting Unity's V-Sync option to "none". V-Sync is primarily used in video games to solve screen tearing. With V-Sync enabled Unity attempts to run the software at the same refresh as the monitor. However, doing so strains the GPU and since there is no visible screen tearing with the volumetric data that setting has been disabled for increased performance.

4.2 Ray Casting

Ray casting is the corner stone for volume visualization. It works by creating a ray for every pixel on the users display. Each ray has a point of origin, direction, and length. For volume visualization the ray is not limited to a specific distance in case the data size exceeds expectations. A ray cast will only stop once it has reached an object. However, to achieve a three dimensional volume with varying transparencies the ray must continue through the object once it has successfully made contact. To solve this problem the fragment shader implements a method called ray marching.

4.3 Ray Marching

Ray marching allows the ray to hit an object and proceed through it, and continuously return data. From the data that is returned Unity is able to determine the color each pixel should be. The ray cast and ray march work together to gather information, see figure two below.

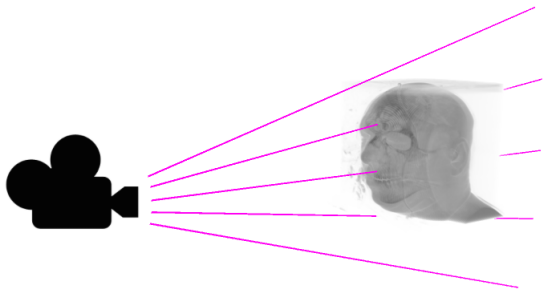

Figure 2: This is a demonstration of how the ray cast and ray march functions cast a ray through the volumetric data.

The use of ray marching allows Unity to adjust the step iteration value. This value directly corresponds with how many steps the ray takes as it passes through the data. The higher the steps iteration the greater the image quality becomes. This come at tremendous cost to the GPU. As the steps increase, the time it takes to load each frame increases exponentially, but there are a few different methods that can be implemented to save the time it takes Unity to process each frame.

4.4 Ray March Control

Using the ray casting approach can lead to sudden drops in frame rate due to the shader attempting to load every piece of the cube all at once. This makes all the data available within GPU memory, which is unnecessary. It is impossible to see every side of the data at the same time which causes processing power to be wasted on the parts of the volumetric data that goes unseen.

On a 1080p display Unity is shooting 2,073,600 rays, all of which are collecting data and return simultaneously, bringing in an enormous amount of information. By stopping the raymarch after it has reached a part of the volumetric data that has an opacity value of one Unity can stop the raymarch without any change in the image on the users display, effectively improving efficiency. The algorithm to stop ray marching when a pixel is solid can be seen below.

Algorithm 1: If the pixel that the ray is marching through is solid then stop the ray march.

```
// Solid pixel skipping
if (ray_pos.x > 1 ||
    ray_pos.y > 1 ||
    ray_pos.z > 1)
    break;
```

The algorithm that determines the amount of steps it takes to go through the data cube is displayed in the algorithm below.

Algorithm 2: This function declares a variable rayStep which determines how many steps are taken to get through the volumetric data. The higher the steps the more detail is rendered.

```
float3 rayDir;
rayDir = pFar - pNear;
float3 rayStep;
rayStep = normalize(rayDir)*sqrt(3)/_Steps;
for(int k = 0; k < _Steps; k++)
{
  float4 voxel_col = get_data(rayPos);
  voxel_col.a = _NormPerStep*length(rayStep)*
      pow(voxel_col.a,_Alpha);
  rayPos += rayStep;
}
```

4.5 Detail Restriction

The most detail this software should ever be processing is what the users display can show. By restricting the scale of the data to the scale of the display it is possible to stop unnecessary processing from occurring and therefore increase the performance. Improving the frame rate is a key aspect of the volumetric rendering software. The processing power of a laptop or virtual reality headset can be very limited and any unnecessary computing would be inefficient. Dynamic memory allocation plays an essential role in rendering volumetric data. It permits the software to dump any and all useless data which considerably improves the time it takes to render each individual frame. The algorithm for adjusting the volumetric data resolution is seen in algorithm three below.

Algorithm 3: The texFilling variable is a float3 that determines the amount of data that is to be sampled.

```
float3 posTex;
float data;
posTex = float3(pos[_Axis[0] - 1],
                pos[_Axis[1] - 1],
                pos[_Axis[2] - 1]);
posTex = (posTex-0.5) * texFilling + 0.5;
data = tex3Dlod(_Data, float4(posTex,0)).a;
```

4.6 Empty Space Skipping

There is no need to raymarch through the empty space around the volumetric data, otherwise it would be wasting precious computing power by marching through empty space. Fortunately, the raycast has been programmed to ignore the empty space. That, however, does not address the concern of a ray passing through an area of the object where there is no color to render. A noticeable amount of processing power can be saved by telling the ray cast shader to cease ray marching through a ray that is not making contact with an object. The algorithm for empty space skipping can be seen below.

Algorithm 4: When the ray march makes contact with a completely transparent pixel it will return 0.

```
// Empty space skipping
if (ray_pos.x = 0 ||
    ray_pos.y = 0 ||
    ray_pos.z = 0)
    break;
```

5 ANALYTICAL TOOLS

To promote further discoveries from the data, features such as a transfer function and a clipping plane tool have been added to the volume renderer. These tools allow for more comprehensive and thorough research. Below is an example of the transfer function and clipping planes in action. The transfer function allows the user to see the different layers by assigning them different colors and the transfer function allows the user to slice the data in half, see figure three below.

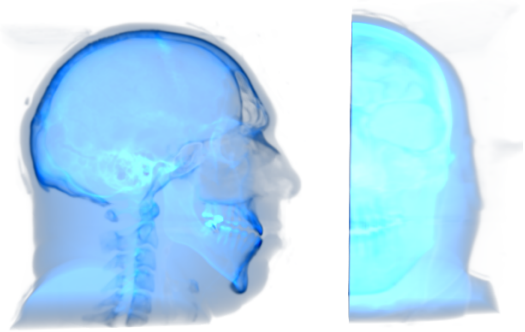

Figure 3: In this figure the clipping plane cuts along the Y-axis. This gives the user a clear view inside the skull and the transfer function allows the user to distinguish the different densities of the data.

5.1 Transfer function

The transfer function uses the density data received from the ray march to determine the color and alpha values of each pixel. The ability to adjust the alpha levels of the volumetric data allows the user to see through specific portions of the volumetric data and receive a clearer view of the information that the user is intent on studying.

5.2 Clipping plane

Clipping planes allow for a cross-sectional view of the data volume. By modifying the axis that the plane is cutting through the user is able to achieve an unobstructed view of the inside of the volumetric data. It works by stopping all rendering until a curtain distance. The algorithm for the clipping plane tool can be seen in algorithm five.

6 CONCLUSION

The software is able to render volumetric data in real time on common hardware without drastic drops in frame rate. The use of ray casting and ray marching allow for pixel perfect rendering. Various adjustments to Unity's settings as well as minor tweaks to the fragment shader have increased the amount of frames that can be rendered per second.

Algorithm 5: The clipping plane has variables for each of the three axis which allow the user to slice through their data in any direction.

```
float _SliceAxisXMin, _SliceAxisXMax;
float _SliceAxisYMin, _SliceAxisYMax;
float _SliceAxisZMin, _SliceAxisZMax;
float4 get_data(float3 pos) {
  float3 posTex = float3(pos[_Axis[0]-1],
         pos[_Axis[1]-1],
         pos[_Axis[2]-1]);
  posTex = (posTex-0.5) * _TexFilling + 0.5;
  float data;
  data = tex3Dlod(_Data, float4(posTex,0)).a;
  // slice and threshold
  data *= step(_SliceAxisXMin, posTex.x);
  data *= step(_SliceAxisYMin, posTex.y);
  data *= step(_SliceAxisZMin, posTex.z);
  data *= step(posTex.x, _SliceAxisXMax);
  data *= step(posTex.y, _SliceAxisYMax);
  data *= step(posTex.z, _SliceAxisZMax);
  data *= step(_DataMin, data);
  data *= step(data, _DataMax);
}
```

REFERENCES

[1] Kyle Hayward. 15 January 2009. Volume Rendering 101. (15 January 2009). http://graphicsrunner.blogspot.ca/2009/01/volume-rendering-101.html

[2] James Money and Thomas Szewczyk. 14 July 2017. Scientific and Intelligence Exascale Visualization Analysis System Theory of Computation. (14 July 2017). https://www.osti.gov/scitech/biblio/1375756

[3] James Money and Thomas Szewczyk. 9 July 2017. An Experiment in Generalized Structured Frameworks for Visualization and Analysis. (9 July 2017). https://dl.acm.org/citation.cfm?id=3093344

[4] Unity. 1 January 2015. Unity - Multiplatform. (1 January 2015). https://unity3d.com/unity/features/multiplatform

[5] Unity. 27 March 2017. OpenGL - Fragment Shaders. (27 March 2017). https://www.khronos.org/opengl/wiki/Fragment_Shader

[6] VRUI. 14 July 2004. Interactive Volume Visualization. (14 July 2004). http://idav.ucdavis.edu/~okreylos/ResDev/VolVis/index.html

[7] Alan Zucconi. 24 November 2017. Volumetric Rendering - Raymarching. (24 November 2017). http://www.alanzucconi.com/2016/07/01/raymarching