CG Project Report
interactive Texture Based Isosurface Rendering

Kaoxin Yao*
Shanghaitech University
Shanghai, China
yaokx@shanghaitech.edu.cn

Siyuan Zhang*
Shanghaitech University
Shanghai, China
zhangsy3@shanghaitech.edu.cn

ABSTRACT

Abstract.

ACM Reference Format:
Kaoxin Yao and Siyuan Zhang*. 2022. CG Project Report interactive Texture Based Isosurface Rendering.

1 INTRODUCTION

introduction In this project, we need to render the isosurface by using the volume rendering method. In this task, our team compared two acceleration structures, octree and texture, and finally chose texture acceleration to achieve **real-time** rendering. By using a simple and convenient UI interface, we were able to adjust the color mapping of the transfer function, change the sampling step, change the viewing angle, and switch between two pre-intergrated tranfer functions.

In the subsequent subheadings of INTRODUCTION, I will describe some of the fundamentals of the content we use.

1.1 Isosurface

An isosurface is a three-dimensional analog of an isoline. It is a surface that represents points of a constant val within a volume of space. in other words, it is a level set of a continuous function whose domain is 3D-space.

1.2 Octree Accelerate Structure

An octree is a tree-like data structure used to describe a three-dimensional space. Each node of the octree represents a volume element of a cube, and each node has eight child nodes. As a result, we can combine the eight child nodes into a single parent node to complete the tree structure.[6]. According to the report[3] as shown in **??**, the Octree-accelerated Isosurface rendering has a very good acceleration ratio, although it does not achieve real-time results.

1.3 Pre-integrated transfer function

The classic transfer function based on interpolation contains two techniques, which are introduced in a popular way, namely post-intergrated transfer function and pre-intergrated transfer function. The former is the same as interpolating across the data's original value and then converting that use the transfer function. The latter is comparable to transforming the original value of data into processed information and then performing interpolation through the processed information for the processed information. In general, the pre-intergrated transfer function will produce a smoother outcome and avoid aliasing.

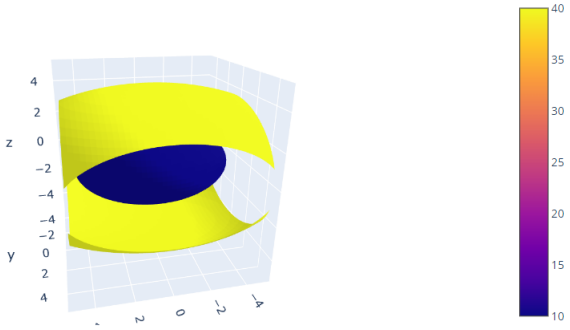

Figure 1: a case of an Iso-surface rendering

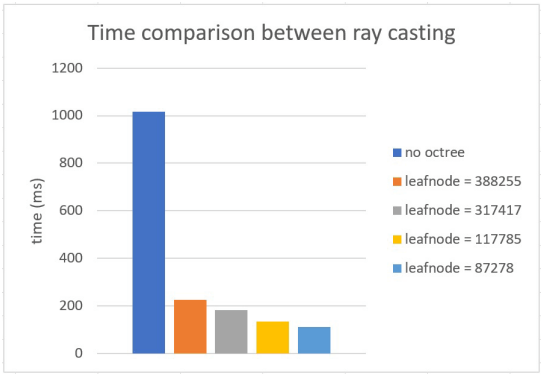

Figure 2: A Comparison of using Octree or not[3]

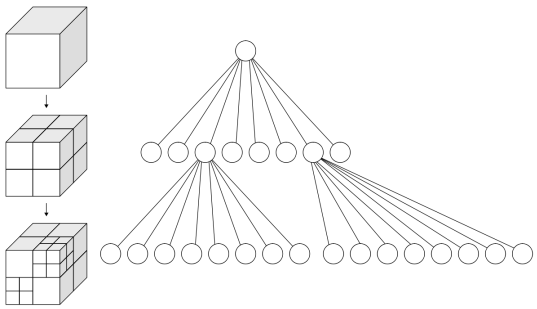

Figure 3: Octree Schematic diagram

*All two authors contributed equally to this research.

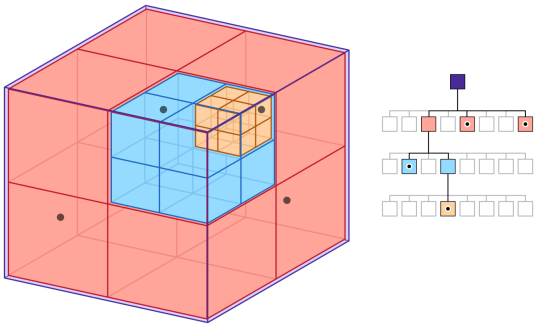

Figure 4: Octree in Volume

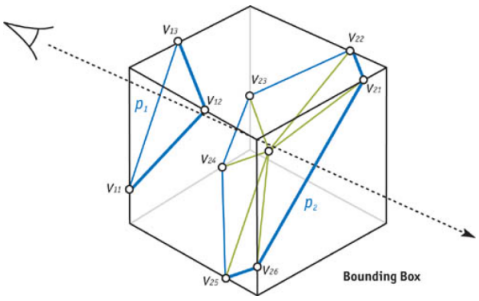

Figure 5: Volume Rendering

1.4 Volume Rendering

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.A direct volume render requires every sample value to be mapped to opacity and a color. This has already done by our "transfer function"
.

The technique of volume ray casting can be derived directly from the rendering equation. A ray is generated for each desired image pixel. Using a simple camera model, the ray starts at the center of projection of the camera and passes through the image pixel on the imaginary image plane floating in between the camera and the volume to be rendered. Then the ray is sampled at regular or adaptive intervals throughout the volume. The data is interpolated at each sample point, the transfer function applied to form an RGBA sample, the sample is composited onto the accumulated RGBA of the ray, and the process repeated until the ray exits the volume. The RGBA color is converted to an RGB color and deposited in the corresponding image pixel. The process is repeated for every pixel on the screen to form the completed image.

The discrete volume rendering equation and the Schematic diagram are showed below.

$$C = \sum_{i=1}^{n} C_i \prod_{j=1}^{i-1} (1 - A_j)$$

$$A = 1 - \prod_{j=1}^{i-1} (1 - A_j)$$

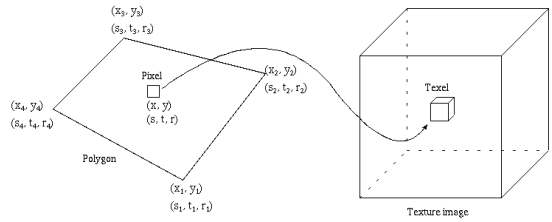

Figure 6: Calculating color and opacity of a pixel inside a texture-mapped polygon using a 3D texture

2 METHODOLOGY

methodology In the subsequent part of the method, I will describe how we completed the project according to our code logic and the sequence of our experiments

2.1 Octree Accelerate Structure

We'll start by reading the data. We can read the information directly because our data is a set of isovalues of size x*y*z. Because the distribution of isovalue is sparse, we can use octree instead of uniformly distributed grids to make the calculation of ray intersections simpler and faster in the following steps. Combining eight grids with the same isovalue into one large grid is more intuitive. In order to prepare for subsequent ray intersections, we process the data into an octree using a recursive approach.

2.2 Texture Accelerate Structure

In addition to octree, we can use color preprocessing as texture to further accelerate, because opengl has an excellent texture interpolation function, which can also make each point through gpu interpolation, to achieve the effect of parallelism, and has a better acceleration effect than octree. After comparing the speed, this project chose the texture acceleration method as the core acceleration algorithm, and achieved the effect of real-time acceleration.[1]

2.2.1 3D Texture Mapping. A specific hardware approach has been devised to enable interactive production of view-orthogonal slices. This is an extension of texture-mapping to three-dimensional textures, which is suitably referred to as "3D texturing."

The figure above shows how 2D texture-mapping interpolates two additional coordinates (s, t) throughout the interior of a polygon. Three additional coordinates (s, t, r) are interpolated in 3D texture-mapping. These three coordinates are utilized as indices into a three-dimensional image, the 3D texture, to define a pixel's color and opacity, as seen in Figure 6. Tri-linear interpolation is used to recreate texture values.

Because of the GPU's great texture mapping optimization and the fact that we performed the interpolation at the same time as the sample Texture, this calculation is incredibly quick on the GPU.

We primarily employ this strategy as well.

2.3 Define the transfer function

Since we already have the structure related to the isovalue, in order to make our results more visualized, we need to define our transfer function, that is, for each cube, we map the known isovalue to

the four channels of RGBA. Here, we use a simple UI to manually change the color values corresponding to different isovalues, and map different isovalue layers to the same color. For the A channel, We need to make the transparency selection by associating the value of the point we sampled with the difference between the isovalue and the value of the point, because we only want to render the content of the surface where the isovalue is located. Conversely, the higher the transparency, the smaller the probability that the point is on the surface.

Volumetric data can be treated directly with 3D textures. Rather than constructing a set of two-dimensional slices in a pre-processing step, the volumetric data is downloaded straight into the graphics hardware and used to determine color and opacity values for each pixel covered by a rendered primitive.

2.4 Different type of Pre-intergated transfer function

Then we need to perform volume rendering work, we need to obtain the RGBA of each volume, and since we cannot completely sample the known data every time, this results in not only the need to complete the value in the data for the color , the interpolation of value to isovalue and the conversion of transparency also need to complete. For the conversion of value to color, it is divided into two types: pre-intergrated and post-intergrated. In this task, post-intergrated is equivalent to first interpolating the value of the volume (trilinear interpolation), and then passing the interpolated value through The transfer function converts to RGBA, which may cause the color to be not smooth enough, so we prefer to use the pre-intergrated transfer function, which can first obtain RGBA based on known information, and then interpolate to get better and smoother results.

In order to achieve the pre-intergrated function, we initially directly used the establishment of 1D dimensional CDF table as a pre-calculated content, according to our design of the transfer function, we can pre-calculate the RGBA corresponding to each isovalue, in finding the color, we can first find the previous CDF value and the current CDF value by integrating the RGBA on the current isovalue as the color information of the volume. Therefore, in order to further reduce the error, we also have all isovalue directly projected into a 2D PDF table, directly through the two isovalue values to find more accurate color information.

2.5 Volume Rendering for Result

Now that we have a very detailed octree structure with four channels of RGBA, we can use volume rendering to get the final RGB information for each ray. We can directly use the origin and direction of the viewing angle as the light, the light enters the structure, obtains the intersection point with the bounding box, and then obtains a smoother color value according to the preprocessed RGB information and the trilinear interpolation of the cube,the processed alpha information is used as the information for visibility processing, and iteratively passes through volume rendering until the visibility is 1(which means the afterwards things is no longer useful) or all volumes are traversed, and finally the color result is displayed on the screen.

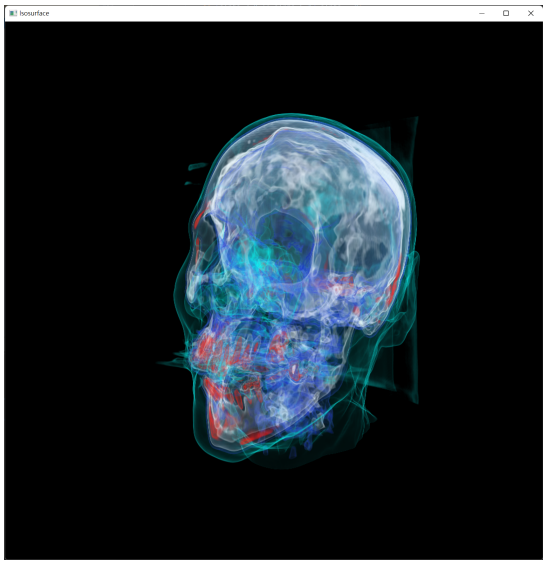

Figure 7: an Isosurface Rendering Result of a CT Head

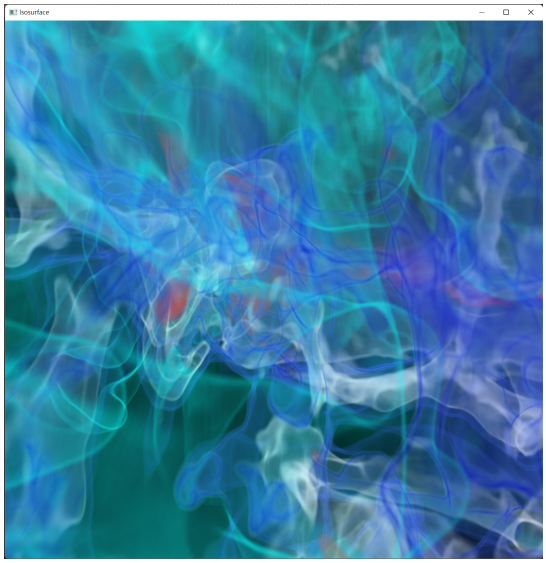

Figure 8: a Real Close Look of a CT Head

2.6 Simple and vivid UI interface

Considering that the design of isovalue and transfer function is diverse, we use a simple UI interface to help dynamically adjust different isovalues and transfer functions corresponding to different isovalues. We simply do it through the mapping relationship between isovalue and color at this point. In addition to the most important sides, we can also adjust the volume rendering step size, camera position, zoom in and zoom out and other details by pressing the buttons to better show our process of isosurface volume render.

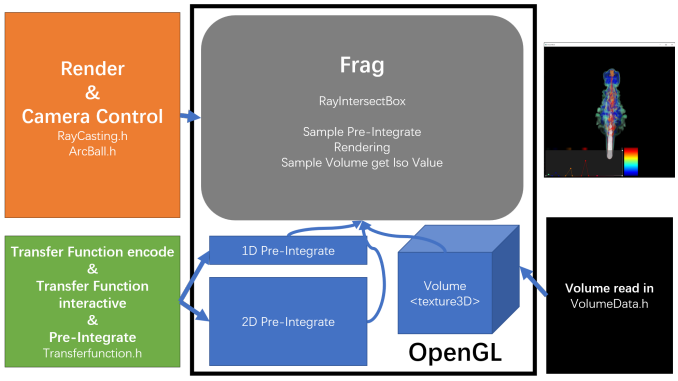

Figure 11: code arch

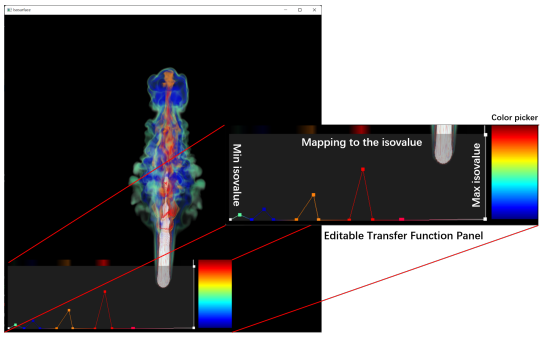

Figure 9: a Simple UI Interface to Change Transfer Function

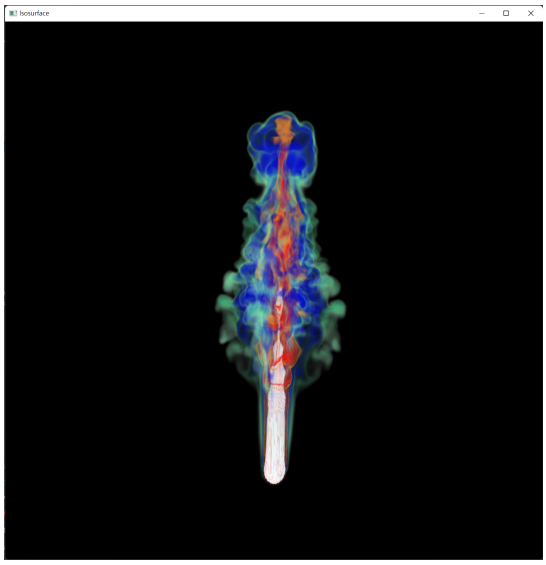

Figure 10: a Isosurface Rendering Result of a Flame

3 CODE , 3RDLIB AND REFERENCE

Our code is constructed as figure 11 .

We use archball[5] for control, jet.h for color picker color creation, and other OpenGL-related external libraries. In addition, when writing the code, we referred to the following open source code.[2][4]

REFERENCES

[1] [n. d.]. https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems_ch https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems_ch39.html. (Accessed on 01/16/2022).
[2] [n. d.]. nnagururu/vdrill at 7e17e81aa88b9331b505f4955318ef384e59652a. https://github.com/nnagururu/vdrill/. (Accessed on 01/16/2022).
[3] [n. d.]. SHTU_CS171/presentation.pdf at main · MeliaLiu/SHTU_CS171. https://github.com/MeliaLiu/SHTU_CS171/blob/main/project/Presentation/presentation.pdf. (Accessed on 01/16/2022).
[4] [n. d.]. toolchainX/Volume_Rendering_Using_GLSL. https://github.com/toolchainX/Volume_Rendering_Using_GLSL. (Accessed on 01/16/2022).
[5] [n. d.]. TuvokBasics/ArcBall.h at master · BlueBrain/TuvokBasics. https://github.com/BlueBrain/TuvokBasics/blob/master/ArcBall.h. (Accessed on 01/16/2022).
[6] Aaron Knoll, Ingo Wald, Steven G. Parker, and Charles D. Hansen. 2006. Interactive Isosurface Ray Tracing of Large Octree Volumes. *2006 IEEE Symposium on Interactive Ray Tracing* (2006), 115–124.