VOLUME RENDERING

Implementation and Execution details

CIS 560/401 – Computer Graphics Final Assignment
Submitted on 08th December 2008

Sunil Kamat
CGGT – Fall 2008

Table of Contents

1. Functions Implemented/Used
2. How To
3. Screenshots
4. References

1. Functions Implemented/Used

RayMarch()

Ray March algorithm is implemented in this function. No interaction with light is performed here. The steps followed to attain the desired result are as follows.

a. Shoot ray Xi at every pixel

b. As we march in steps in the negative Z direction, Xi value should be calculated to check if it has gone past the maximum Z value (in my case z=-4). This value can be increased as much as the far clipping plane but it increases the calculation time drastically. Also, delta S value can be lowered to increase the number of steps in the march to get finer results and decrease the amount of banding and contouring.

$$\mathbf{x}_i \mathrel{+}= \hat{\mathbf{n}}\, \Delta s$$

c. At each step we check if the ray intersects with any sphere.

d. When it intersects, we use the fBm function of perlin noise to displace boundary (explained further).

e. The return value of the previous step decides the value for deltaT to further use it for calculating transmissivity (T) and color (C). If the return value is less than or equal to zero, deltaT is considered as 1 else deltaT is calculated using an expression as shown below.

$$\Delta T = \exp\{-\kappa \Delta s\ \rho(\mathbf{x}_i)\}$$

f. Transmissivity value is further calculated using the following expression.

$$T \mathrel{*}= \Delta T$$

g. Finally color at each step is accumulated and a final color for that pixel is obtained. Here C is the accumulated color at each step for that pixel. K is the kappa value which is considered as 1 but can be further reduced to get finer results basically to increase the transparency but it would increase the render time. 'c' is the color of the sphere where the ray intersects.

$$\vec{C} \mathrel{+}= \frac{1-\Delta T}{\kappa}\ \vec{c}(\mathbf{x}_i)\ T$$

RayMarchWithLights()

This does everything that is done by the RayMarch() function but further adds interaction with light. As we had accumulated color at every step, in this case, the value of Q is calculated which is further used in the new color accumulation function as shown below.

$$\vec{C} += \frac{1 - \Delta T}{\kappa} \left(\vec{c}(\mathbf{x}_i) \odot \vec{F}\right) T \; Q(\mathbf{x}_c, \hat{\mathbf{n}}, \Delta si, \mathbf{x}_l)$$

NOTE: In case of ray marching with light interaction, the actual implementation as per logic must be `color+=((1 - deltaT)/t1->m_valueK) * cF * T * Q;`

But while experimenting with it, the background color was added to the accumulated color at the end as `color+=bgColor;`

This gave some interesting result, but is/may not be the actual way of doing the same. The results of this can be seen in the screenshot section of this document.

EvaluateQ()

This function calculates the value of Q which is further used in the color equation as shown above. Here, the density at the calculated point is used and summed up with the deltaS value and is used in the expression below to get the final Q value for the pixel in question. Q, D and NCap can be calculated as follows.

$$Q(\mathbf{x}_c, \hat{\mathbf{n}}, s, \mathbf{x}_l) = \exp\left\{-\kappa \int_0^D ds' \rho(\mathbf{x}_c + \hat{\mathbf{n}}s + \hat{\mathbf{N}}s')\right\}$$

$$D = |\mathbf{x}_l - \mathbf{x}_c - \hat{\mathbf{n}}s|$$

$$\hat{\mathbf{N}} = \frac{\mathbf{x}_l - \mathbf{x}_c - \hat{\mathbf{n}}s}{|\mathbf{x}_l - \mathbf{x}_c - \hat{\mathbf{n}}s|}$$

DisplaceBoundary()

The implicit function as shown below is implemented here. It uses the fBm function to displace the boundary. The amount of displacement can be manipulated by changing the values of the parameter “a” which is nothing but the thickness of the pyroclastic puff.

$$\left|\mathrm{fBm}\left(\frac{\mathbf{x} - \mathbf{x}_s}{|\mathbf{x} - \mathbf{x}_s|}\right)\right| + a - \frac{|\mathbf{x} - \mathbf{x}_s|^2}{r^2}$$

FBM()

FBM stands for fractal Brownian motion. The function implements the expression as shown below. 3-dimensional perlin noise function is used to add some noise as the ray passes through the sphere. The amplitude (A) and frequency (f) values in the equation are modifiable to get different results.

$$\mathrm{fBm}(\mathbf{x}) = \sum_{\ell=1}^{L} A^{\ell-1}\ \mathrm{Perlin}(\mathbf{x}\ f^{\ell})$$

noise3()

This is an important function which adds noise to the sphere. The noise3 function is nothing but the Perlin function in the above equation. The randomness to the noise is created using the mersenne twister random number generator. Ken Perlin's noise implementation was used for this but was further modified to insert the mersenne twister instead of drand().

SphereIntersect()

The bounding volume for the sphere is implemented here. This function speeds up the rendering process. Returns true if there is an intersection, else returns false. A sphere is used as a bounding volume in this case as it is more efficient.

VolumeRender()

Function uses all the calculated pixel position and color data and plots it. The plotted result is actually a pyroclastic puff.

DrawSphere()

Function uses the stored data collected from the text file and draws the sphere on the screen. OpenGl functions are used to plot the points on the screen. This function is just to check if the spheres get plotted at the required location before volume rendering it.

DataRead()

Simple function to read the data from the text file provided. The function takes care of errors related to non-existent file read, multiple spaces between two lines and comments in between the lines. Yes, one can add comments to the file. Comments have to start with the # (hash) symbol.

2. How To

Steps to follow.

a. Open and solution VS2008 and press F5 key.
b. Required data is populated from the text file at this step.
c. Press 1 to volume render the sphere. (No light interaction)
d. Rendered output is saved as BMP.

If light interaction is required then at step c above, press 2 instead of 1 to volume render with light interaction.

A BMP file is created and stored at the end of every render operation. The file can be found in the screenshot directory under the same project folder.

Different rendered output can be obtained by changing the following values. The changes in some of the following values will drastically increase the render time.

deltas
Found in the text file. Current value 0.2. Can be further reduced to get finer results. Helps in improving the ouput by reducing the banding and contouring. Try 0.02.

Amplitude and Frequency (A and f)
Variables can be found in the FBM function. Increase or decrease the value of A and f to get different and interesting results.

Thickness of the puff (a)
Variable can be found in DisplaceBoundary function. The thickness of the puff can be varied by changing the value of a.

Light Color (F)
Color of light can be modified changing the value of F vector. Can be found in RayMarchWithLights() function. Default is red light.

Background Color
Background color of the scene can be changed by changing the values in common.h file. The color of the end result varies and the puff interacts with the background color.

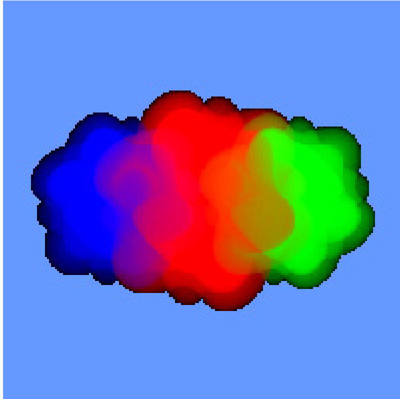
Position of light
Position of the light can be varied by changing the values in common.h file. Interesting results can be seen by changing the position of light.

Screen resolution
Resolution can be changed in common.h file which is currently set to 300X300.

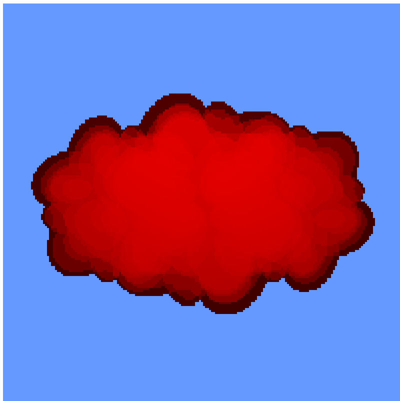
3. Screenshots

Many screenshots can be found under the screenshot folder. Some of them were taken while experimenting with the rendering functions and manipulating different variables. Some of them are as follows. Note that some of the other properties are also changed to get the desired result apart from what is shown below.

With A=3, f=1, a=2, deltas=0.02 (Ray March without lights)
Blue, Red and Green sphere used.

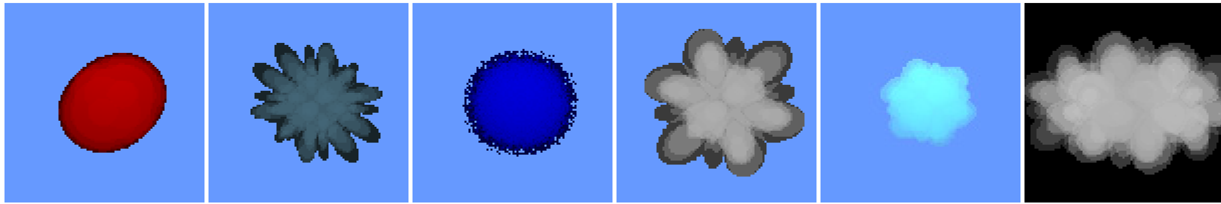


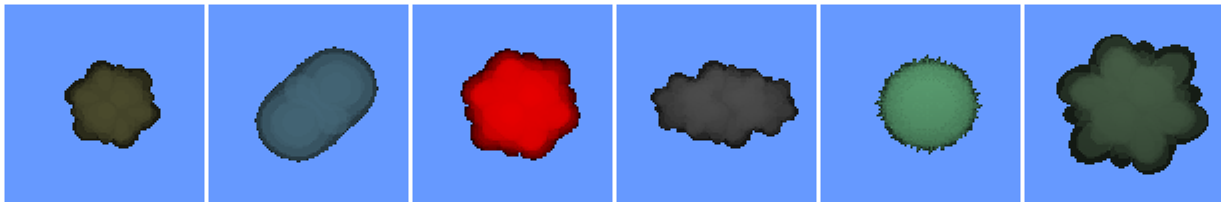
Three red spheres

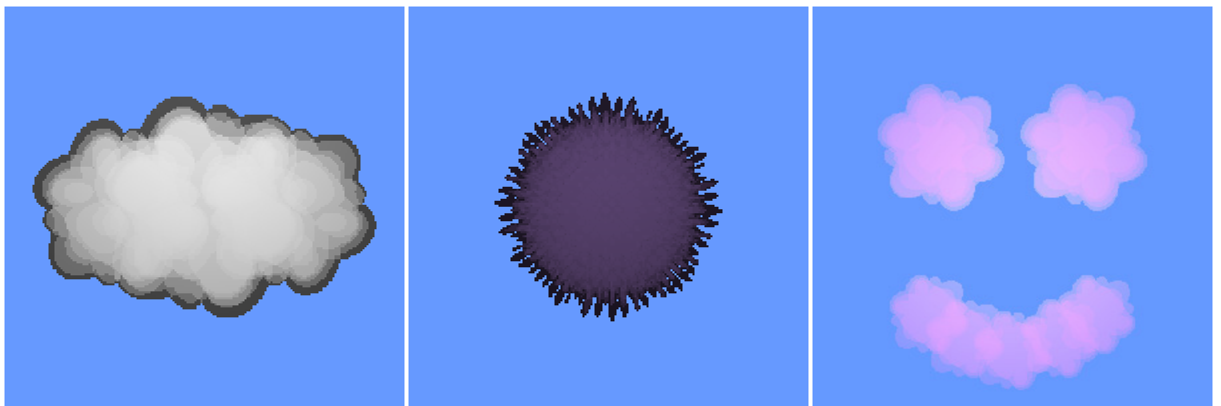


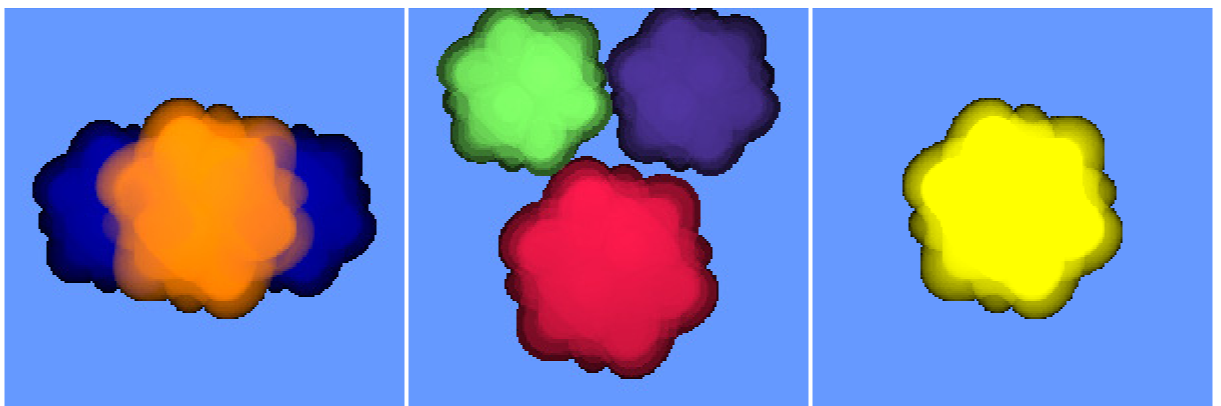
With A=3, f=1, a=2, deltas=0.2 (Ray March with lights)
Three red spheres used. Black border tinge here was removed as explained in section 1.

Some more screenshots with values of A, f, a, deltaS and K.

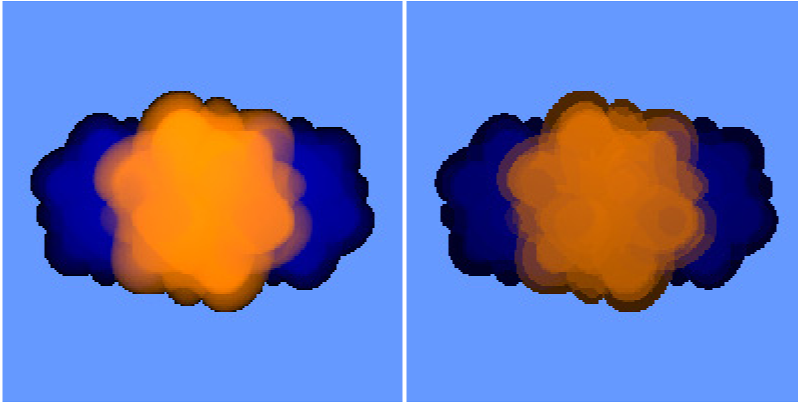








The difference by increasing the ray march steps can be clearly seen in these 2 pictures. First one has a deltaS value of 0.02 and second has 0.2. Banding and countering has decreased drastically but it takes a lot of time to render with smaller steps.



4. References

Main references

Tessendorf.pdf
Available at blackboard. Function images used in this writeup are also taken from this pdf.

http://tessendorf.org/papers_files/volumerendering.pdf
Concepts for Volume Rendering

Textures & Modeling: A Procedural Approach
By Ebert, Musgrave, Peachy, Perlin, & Worley

http://peter.grumpykitty.biz/perlin.html
Ken Perlin's implementation of perlin noise.

http://www.bedaux.net/mtrand/
Mersenne Twister pseudo-random number generator.

Others

http://www.cs.cmu.edu/~mzucker/code/perlin-noise-math-faq.html
http://www.programmersheaven.com/2/perlin
http://www.angelcode.com/dev/perlin/perlin.asp
http://jellyengine.blogspot.com/2008/04/perlin-noise-and-fractals.html
Wiki and Google