Ray Casting for Iso-surface in Volumetric Data

By

Do Hoang Ngoc Anh

FINAL REPORT

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Technology (Hons)
(Information Technology)

DECEMBER 2005

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Ray Casting for Iso-surface in Volumetric Data

By

Do Hoang Ngoc Anh

Dissertation submitted to the
Information Technology Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(INFORMATION TECHNOLOGY)

Approved by,

[illegible]

(Mr. Nordin Zakaria)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

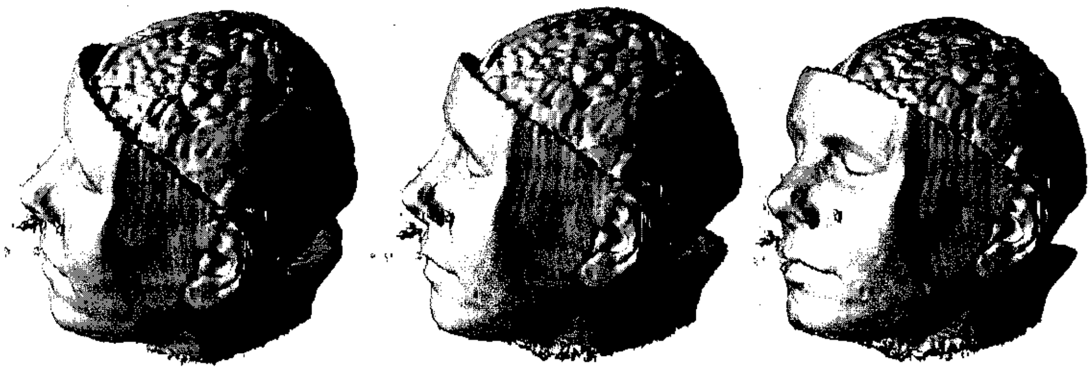
DECEMBER 2005

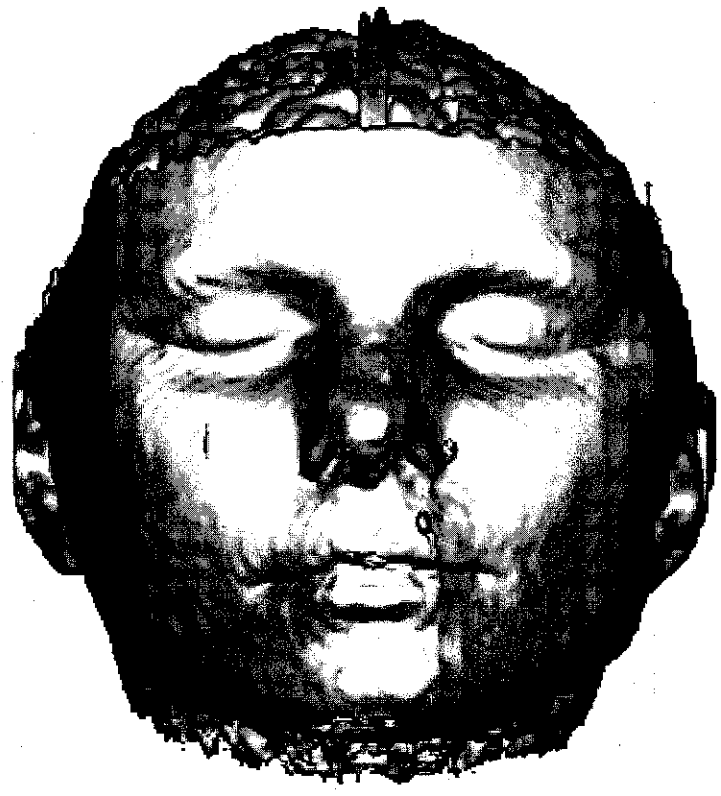
CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

DO HOANG NGOC ANH





Ray Casting for Iso-surface in Volumetric Data

(dataset: BrainSmall - Volpack - Stanford)

ABSTRACT

Volume data visualization is an active field of research and development. It can be applied in many areas such as medical, oil and gas exploration, etc... Although volume visualization is highly computational cost, there is a vision of real time volumetric visualization systems based on interactive ray tracing. Over the years, many rendering algorithms have been created and enhanced. The focus of this project is to develop a simple ray casting program for volumetric data. The program will be able to render specific volume data using a single processor in a reasonable amount of time. It is open to improve for implementation on multiprocessors. The thesis will compare some existing algorithms for ray casting in terms of image quality, computing time, complexity and so forth. The thesis includes a proposal of new multisampling algorithm, which significantly reduces rendering time while producing similar quality of image with existing algorithms.

ACKNOWLEDGEMENTS

I would like to present my respect and thankfulness to my supervisor, Mr. Nordin Zakaria who shows his supports not only in professional knowledge, academic aspect but also in my character development. I would like to thank him for the chance of working in real project, which may be a topic for my further study.

I would like to thank Dr. Ahmad Kamil Mahmood for his encouragement during the last stages of this project.

I would like to thank Dr. Abas Md Said and the board of management for the vision about visualization.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND NOMENCLATURES

Voxel - Unit of three-dimensional array of data that contains any number of fields of data, such as tissue density or temperature.

Illumination - the transport of luminous flux from light sources between points via direct and indirect paths

Lighting - the process of computing the luminous intensity reflected from a specified 3-D point

Shading - the process of assigning a colors to a pixels

CHAPTER 1

INTRODUCTION

Currently, researches on interactive volume ray tracing maybe categorized into 3 main groups. These categories are namely parallel software-based implementation, single graphic unit implementation, and multiple hardware graphic units in parallel. Researcher in the first group tends to use parallel programming to accelerate its software. Ray casting, shear warp and splating are techniques explored by this group. This group takes advantage of PCs, resulting in a flexible and cheap multiprocessor solution. The second group uses single graphics adapters to perform rendering. There are some advantages of the method in this group over the first group such as the ability to exploit texture mapping hardware. The last group uses hardware graphics units in parallel [5].

This thesis is a research project, which aims to study on ray casting of volumetric data. The thesis focuses on iso-surface rendering using ray casting algorithm. Objective of the thesis is to compare some existing rendering algorithms and to find some ideas on how to improve them. The report is organized into the following sections:

1. **Literature survey**. The project will provide an overview of ray tracing concepts, which are necessary for any ray tracing program. The in-depth survey will focus on existing algorithms that aim at real time interactive program.

2. **Implementation of selected algorithms**. Some of techniques and algorithms will be implemented. The final program will be able to render volume data. Different algorithms will be implemented for comparison purpose.

3. **Analysis and comparison**. Selected algorithms will be analyzed and compared according to their accuracy, computing time, computational cost, complexity and so forth. The comparison may be beyond implemented algorithms.

4. **Enhancements and areas of research**. The project will suggest some ideas to enhance existing algorithms. Since those algorithms are well developed, chances of success of those ideas are few. However, they maybe new ways of seeing the old things.

CHAPTER 2

LITERATURE REVIEW AND THEORY

"If I have seen farther than others, it is because I was standing on the shoulders of giants". Albert Einstein

This chapter first provides an overview about ray tracing concepts, and then drills on particular theories that are implemented by the thesis.

2.1 Basic knowledge about ray tracing

2.1.1 Ray

Theoretically, ray, or half line, is a 3D parametric line with a half-open interval, usually $[0,+\infty)$ [1]. However, in practice, the interval of a ray is usually set as $[\varepsilon, +\infty)$ due to the fact that the origin of the ray is usually placed on the surface. The interval thus is ranged from ε to ∞ in order to avoid self intersection [2].

A ray can be written as:

$R(t) = O + D * t$

or

$x(t) = Origin.x + Direction.x * t$

$y(t) = Origin.y + Direction.y * t$

$z(t) = Origin.z + Direction.z * t$

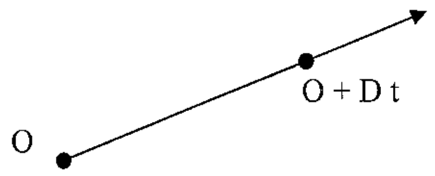


Figure 1: Ray presentation

where Origin.x, Origin.y, Origin.z are the coordination of origin of the ray in the coordination system, Direction.x , Direction.y ,Direction.z are the value of direction vector upon perspective axis. t is the time value, which lays between interval tmin and interval tmax: $\varepsilon \leq t < +\infty$. In practice, we use 2 float values tmin and tmax to set the valid range for a ray.

2.1.2 Ray casting

The idea of ray casting is simple. Given a view point (eye), given some primitives, given a view plane (screen). The ray tracer will generate rays, which have origin located at eye, go through every single pixel on the screen to hit onto primitives. The colors of screen pixels are determined by the intersections between rays and primitives.

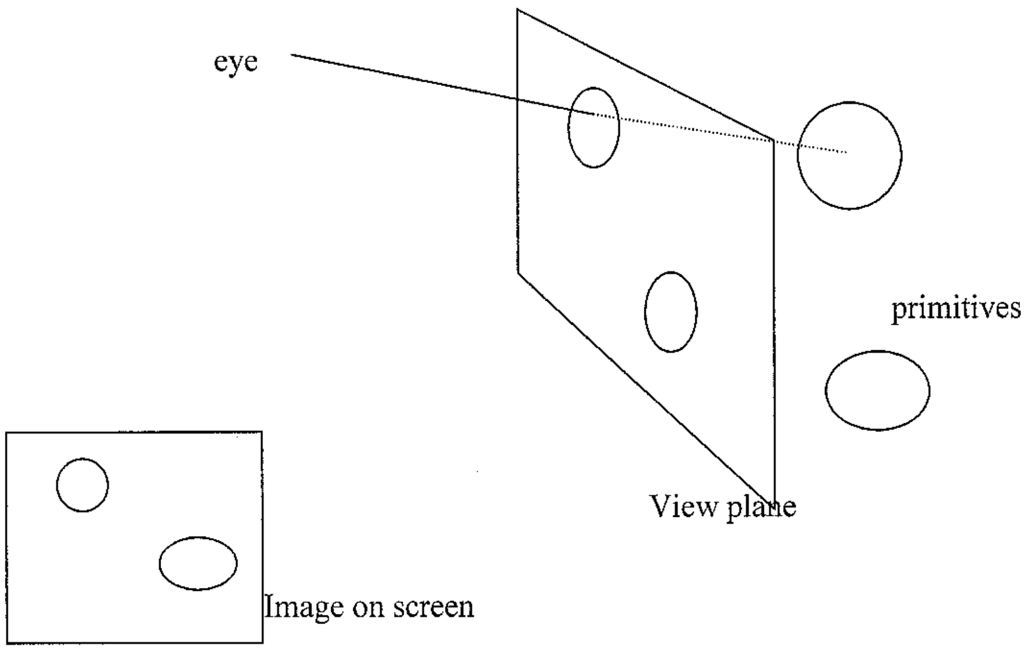


Figure 2: Ray casting basis

There are 2 main tasks: to determine the intersections between rays and objects (hit points), and to determine what color that point is [1]. The term "primitive" stands for both object and iso-surface.

- **Intersections**

There are 3 different kinds of finding intersection.

- *To find closest hit to the eye.* It is the most fundamental task in ray tracing program. The hit point must satisfy the condition that it is the nearest point to the eye among all hit points. It usually requires an additional mechanism to verify the condition.

- *To find any hit.* It is to check as if the ray hit primitive at any point in between tmin and tmax. General speaking, this case is simpler than the first case and most of ray tracing engines have special algorithm for it.

- *To find all hits.* There is not many ray tracers take care of this case; it requires advance algorithm [2].

Following example discusses on a simple ray-plane intersection [1]. The object is a plane whose implicit equation : $(p-a) * n = 0$. The ray is described as $p(t) = O + t * d$. Thus, the ray hit the plane at the value of t where $(O + t * d - a) * n = 0$; Or $t = \frac{(a-o)n}{d.n}$

- **Coloring intersection point ?**

Coloring a point, or shading, is to find the color of the point with given light sources and reflecting primitives. Generally, there are 2 contributors to the color of a point, local illumination and global illumination.

- *Local illumination* deals with the light come directly from light sources. It ignores the effects of other primitives, such as reflected light from primitives or occluded light from light sources. Common techniques to calculate local illumination are flat shading, Gouraud shading and Phong shading. *Flat shading* assigns the same color for a flat polygon, which can be enhanced by calculating the color at edges. *Gouraud shading* is the most common technique, used by OpenGL, which calculates color in linear approach along scan-lines. Its limit is the inaccuracy in illustrating specular components. *Phong Shading* calculates color at every pixel of the objects, provides better results compared to Flat and Gouraud shading, but its computational cost is more expensive.

- *Global Illumination* deals with the light come directly and indirectly from light source. In contrast to local illumination, it also calculates the reflected light from other objects. Ray tracing is used for global illumination. The idea of ray tracing is firstly, to cast a ray and find the closest intersection point, and then to recursively cast secondary rays from intersection points to get global effect on the hit point.

Results of illumination depend on properties of light source and the surface where rays hit objects. Those properties are color, position and direction, directional attenuation of the light source; color, position, orientation and micro structure, absorption capacity of the surface [3].

2.1.3 Enhancement – multi-sampling technique

Single-sample ray tracer, which is tracer that estimates color of hit points by sampling with one ray, is fast and good for interactive application [2].

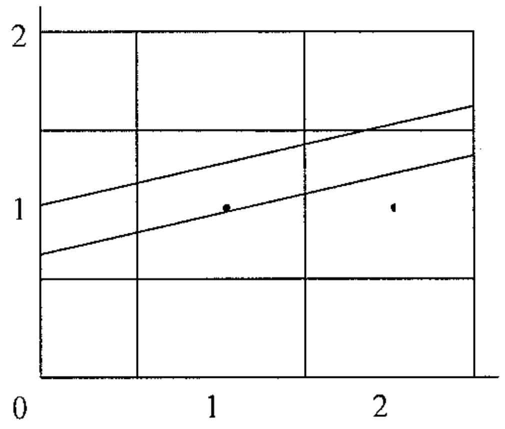


Figure 3: Aliasing with single sample

The problem of single-sampling ray tracing is it creates jaggies in the image. Above diagram illustrates the problem. Single sample ray in point (2,1) does not hit the object (a leaning bar). The image thus does not display the bar consistently.

To have better quality picture, there is a need of advance technique, such as multi-sample ray tracing. Multi-sampling ray tracing means to sample the scene with more than 1 ray per screen-pixel. The arrangement of sampling ray is random, which is different from super-sampling or regular sampling. Super sampling is still potential of generating jaggies. Following diagram illustrates a randomizing schema that use grid to partition the square unit into smaller grids. [4]

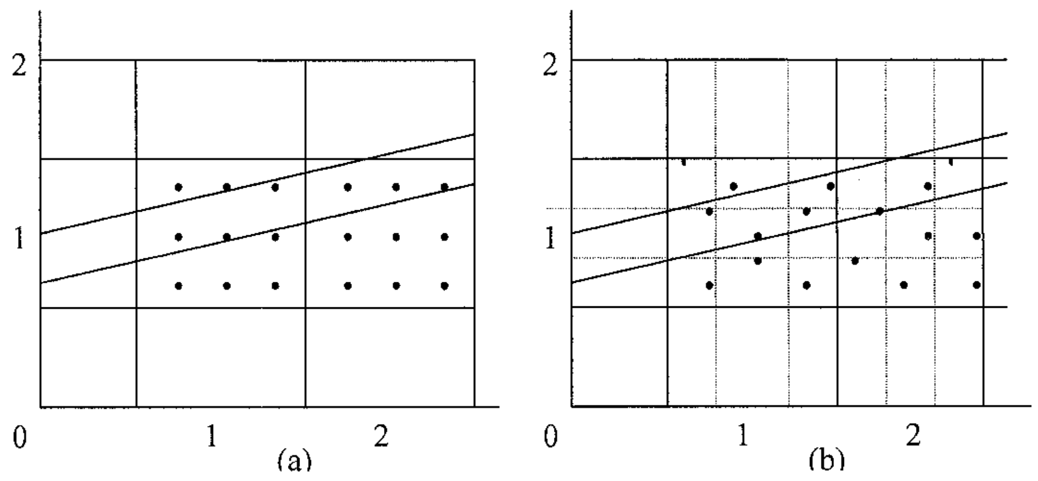


Figure 4: Multisampling (a) Super sampling (b) Random sampling

2.1.4 Primitives

Basically, scene is a list of geometric primitives to be rendered. In order to render it, primitive intersections must be computable [2]. The common approach to find intersection is, firstly, to develop geometric expression of the primitives, and then, to solve the polynomial of equation where ray and primitives are equal.

There are many types of primitives; they are objects like plane, triangle, sphere...; iso-surfaces in volumetric; algebraic surface... Triangle is one of the most common and important primitives since all objects can be displayed by triangles.

2.1.5 Volumetric data

Volumetric data is a set of samples (x, y, z, v), representing the value v of some property of the data, at a 3D location (x, y, z) [7]. The volumetric data is usually stored in a three dimensional array. The samples are aligned to the Cartesian axes and are equally spaced in a given dimension [8].

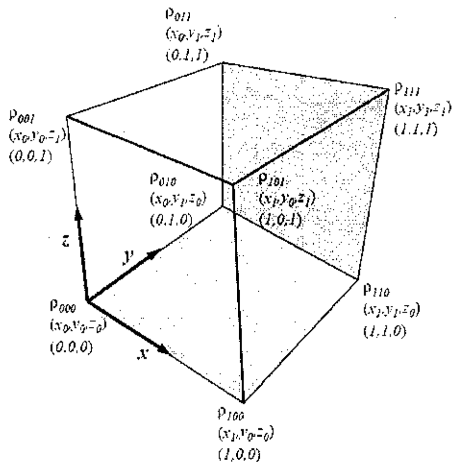


Figure 5: Voxel

A cell, or voxel, of volumetric data is illustrated by above diagram, where value of sample points are p000 at (x0,y0,z0), p001 at (x0,y0,z1), etc... In trilinear volume, where value of point along 3 axes are linear, the value of one point p inside the voxel at coordination (x,y,z) is calculated by

P(x,y,z) =

$$\frac{x_1-x}{x_1-x_0}\frac{y_1-y}{y_1-y_0}\frac{z_1-z}{z_1-z_0}p_{000}+\frac{x_1-x}{x_1-x_0}\frac{y_1-y}{y_1-y_0}\frac{z-z_0}{z_1-z_0}p_{001}$$
$$+\frac{x_1-x}{x_1-x_0}\frac{y-y_0}{y_1-y_0}\frac{z_1-z}{z_1-z_0}p_{010}+\frac{x_1-x}{x_1-x_0}\frac{y-y_0}{y_1-y_0}\frac{z-z_0}{z_1-z_0}p_{011}$$
$$+\frac{x-x_0}{x_1-x_0}\frac{y_1-y}{y_1-y_0}\frac{z_1-z}{z_1-z_0}p_{100}+\frac{x-x_0}{x_1-x_0}\frac{y_1-y}{y_1-y_0}\frac{z-z_0}{z_1-z_0}p_{101}$$
$$+\frac{x-x_0}{x_1-x_0}\frac{y-y_0}{y_1-y_0}\frac{z_1-z}{z_1-z_0}p_{110}+\frac{x-x_0}{x_1-x_0}\frac{y-y_0}{y_1-y_0}\frac{z-z_0}{z_1-z_0}p_{111}$$

(pixel value triliniear equation)

Iso-surface is the surface that consists of points that have the same value within a volumetric data. To render an iso-surface in volume, which is the main purpose of this project, there are 2 main techniques, indirect rendering and direct rendering.

- *Indirect rendering* is the technique that first generates a model according to raw data and later renders the model. A common technique of indirect rendering is Marching Cube [6]. The algorithm requires a mapping dictionary of voxel corner value and the shape of iso-surface associated with that voxel. In fact, Marching Cube is faster than direct rendering. Indirect rendering technique has a pitfall of inaccuracy. Since it requires a middle step of generating models, it misses some data (for full resolution, it takes very long time to generate the model).

- *Direct rendering* is the technique that renders the iso-surface directly from raw data without creating any model. Direct rendering technique has the pitfall of computational cost since it requires calculating for every single point on the screen. However, there are many ways to reduce rendering time such as parallel programming.

2.1.6 Ray acceleration methods

One of active areas of researches is ray tracing acceleration. Existing methods maybe categorized into 3 groups, fast intersections, fewer rays, and generalized rays.
The aim of the first group is to find intersections in shorter time. Fast intersection methods can be subdivided into 2 groups, which are faster ray-object intersection and fewer ray-object intersections. The second group tries to reduce number of ray and the last group uses special rays with different shapes. [19]

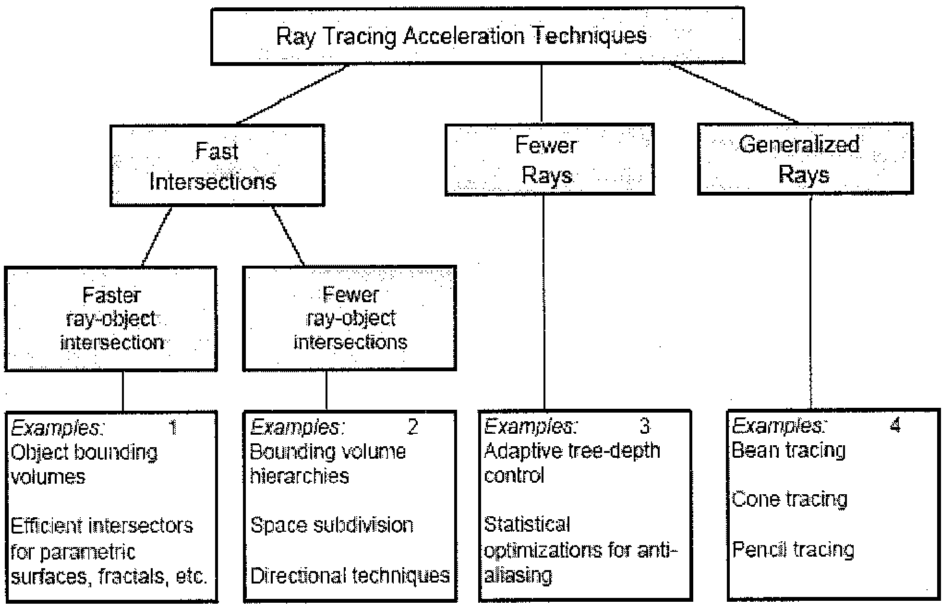

Ray Tracing Acceleration Techniques
Fast Intersections
Fewer Rays
Generalized Rays
Faster ray-object intersection
Fewer ray-object intersections
Examples: 1
Object bounding volumes
Efficient intersectors for parametric surfaces, fractals, etc.
Examples: 2
Bounding volume hierarchies
Space subdivision
Directional techniques
Examples: 3
Adaptive tree-depth control
Statistical optimizations for anti-aliasing
Examples: 4
Bean tracing
Cone tracing
Pencil tracing

Figure 6: Ray acceleration classification

2.2 Algorithms, analyses and comparisons

This chapter, which is the main part of this report, describes and compares some recent ray casting algorithms, which was deployed in volumetric data rendering. There are some comparisons that go beyond implemented parts. The project conducted researches on intersection, gradient estimation and ray traversal techniques.

2.2.1 Intersection

This section is a comparison of Schwarze's approach [10], simple middle value approximation, linear approximation and exact linear approach [11].

- **Trilinear interpolation: cube polynomial**

This method was developed by Schwarze[10], it results in the most elegant intersection point. The algorithm bases on Cardano solution for general cubic equation. Pitfall of this algorithm is time consuming since it uses arccos function.
The detail of this algorithm is described as follow.

Pixel value trilinear equation after substituting with $u = \frac{x - x_0}{x_1 - x_0}$, $v = \frac{y - y_0}{y_1 - y_0}$, $w = \frac{z - z_0}{z_1 - z_0}$ became

$$\begin{aligned} p(u,v,w) = {} & (1-u)(1-v)(1-w)\,p000 + (1-u)(1-v)(w)\,p001 + (1-u)(v)(1-w)\,p010 + \\ & (1-u)(v)(w)\,p011 + (u)(1-v)(1-w)\,p100 + (u)(1-v)(w)\,p101 + \\ & (u)(v)(1-w)\,p110 + (u)(v)(w)\,p111 \end{aligned}$$

Substituting $1 - u = u_0$, $1 - v = v_0$, $1 - w = w_0$,

$u = u_1$, $v = v_1$, $w = w_1$

$$p(u,v,w) = \sum_{i,j,k=0,1} u_i v_j w_k p_{ijk}$$

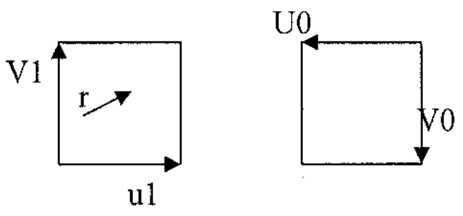


Figure 7: Interpreting ray in different coordination system

A ray p(t) = a + tb intersects with the iso surface at a point p(t) = p_{iso}. This equation can be converted into cubic polynomial by displayed p(t) = a + tb on 2 coordination systems

$$a_0 = (u_0^a, v_0^a, w_0^a) = \left(\frac{x_1 - x_a}{x_1 - x_0}, \frac{y_1 - y_a}{y_1 - y_0}, \frac{z_1 - z_a}{z_1 - z_0}\right); b_0 = (u_0^b, v_0^b, w_0^b) = \left(\frac{-x_b}{x_1 - x_0}, \frac{-y_b}{y_1 - y_0}, \frac{-z_b}{z_1 - z_0}\right);$$

$$a_1 = (u_1^a, v_1^a, w_1^a) = \left(\frac{x_a - x_0}{x_1 - x_0}, \frac{y_a - y_0}{y_1 - y_0}, \frac{z_a - z_0}{z_1 - z_0}\right); \; b_1 = (u_1^b, v_1^b, w_1^b) = \left(\frac{x_b}{x_1 - x_0}, \frac{y_b}{y_1 - y_0}, \frac{z_b}{z_1 - z_0}\right)$$

p(t) = p_{iso}became

$$p_{iso} = \sum_{i,j,k=0,1} (u_i^a + tu_i^b)(v_j^a + tv_j^b)(w_k^a + tw_k^b)p_{ijk}$$

$$p_{iso} = \sum_{i,j,k=0,1} (u_i^a v_j^a + tu_i^b v_j^a + tu_i^b v_j^a + t^2 u_i^b v_j^b)(w_k^a + tw_k^b)p_{ijk}$$

$$p_{iso} = \sum_{i,j,k=0,1} \begin{pmatrix} u_i^a v_j^a w_k^a + tu_i^b v_j^a w_k^a + tu_i^b v_j^a w_k^a + t^2 u_i^b v_j^b w_k^a \\ + tu_i^a v_j^a w_k^b + t^2 u_i^b v_j^a w_k^b + t^2 u_i^b v_j^a w_k^b + t^3 u_i^b v_j^b w_k^b \end{pmatrix} p_{ijk}$$

Set

$$A = \sum_{i,j,k=0,1} (u_i^b v_j^b w_k^b)p_{ijk}$$

$$B = \sum_{i,j,k=0,1} (u_i^b v_j^b w_k^a + u_i^b v_j^a w_k^b + u_i^b v_j^a w_k^b)p_{ijk}$$

$$C = \sum_{i,j,k=0,1} (u_i^b v_j^a w_k^a + u_i^b v_j^a w_k^a + u_i^a v_j^a w_k^b)p_{ijk}$$

$$D = -p_{iso} + \sum_{i,j,k=0,1} (u_i^a v_j^a w_k^a)p_{ijk}$$

Derived cubic polynomial $At^3 + Bt^2 + Ct + D = 0$ can be solved using Cardano's method.

Given F(x)=x^3+ px + q. (p $\neq$0)

If x is a solution of F(x) =0, then there exist u, v that x = u + v and 3uv = - p.

Substituting x = u+ v onto the equation $(u + v)^3 + p(u + v) + q = 0$

$$u^3 + v^3 + (3uv + p)(u + v) + q = 0$$

Substituting $3uv = - p$ yields $u^3 + v^3 = - q$

Hence, u^3 and v^3 are roots of the quadratic equation $s^2 + qs - (p/3)^3 = 0$

The solutions of this equation are:

$u^3 = (-q/2) + [(q/2)^2 + (p/3)^3]^{1/2}$

$v^3 = (-q/2) - [(q/2)^2 + (p/3)^3]^{1/2}$

If $(q/2)^2 + (p/3)^3$ is negative, it will be *"casus irreducibilis"*, in which results complex numbers. Triginometrical method is the only way to find real root.

Setting $\cos(\varphi) = -0.5q(-p/3)^{-2/3}$, where $\varphi \in (0,\pi)$; $r = 2 (-p/3)^{1/2}$ results $x_1 = r \cos(\varphi /3)$, $x_2 = r \cos(\varphi + 2 \pi)/3$, $x_3 = r \cos(\varphi+4\pi)/3$

This project uses Schwarze's algorithm to implement Cardano's method.[9] This algorithm naturally initiates many disadvantages. Firstly, it requires separate algorithm for degraded cases of F(x) =0, e.g. A = 0. The degraded case requires quadratic equation solution. Secondly, to find the value of φ, it requires using arccos function, which is expensive in term of computational cost. Besides, the algorithm is unstable, in term of numerical accuracy [11]. However, the algorithm produces highly accurate intersection if ε is small enough. In a recent research, a value of 1.e -30 was used [8].

- **Middle point approximation**

This algorithm is considered as the simplest algorithm of calculating intersection. When visiting a voxel with iso-surface inside, the ray finds intersection t as the average value of tIn and tOut. tIn and tOut are the value of tHit when the ray enters the voxel and exits from it, respectively.

The computing time is fast, but quality image is poor with blocky effect. It is inaccurate.

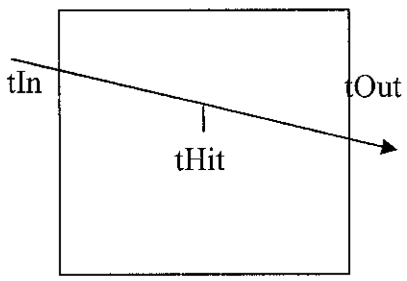


Figure 8: Middle point interpolation

- **Linear interpolation approximation**

Linear interpolation is more advanced than simple middle point interpolation but it is still simple. In order to apply it, there must be an assumption that value along the ray from entry point to exit point is linear.

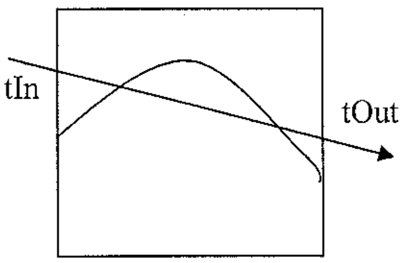


Figure 9: Linear interpolation miss-case

The algorithm at first, checks if density value of entry point and exit point are both greater or both smaller than density value of iso-surface. This condition means there will not be a hit in between 2 points. It is a potential error as the iso-surface may bend over the ray.

Hit point is calculated by

$$t_{hit} = t_{In} + \left(t_{Out} - t_{In}\right)\frac{p_{iso} - p_{in}}{p_{out} - p_{in}}$$

The algorithm is fast, simple but inaccurate.

- **Exact linear interpolation**

The algorithm applies repeated linear interpolation [12] on the nearest interval that has iso-surface. The mathematical base of the algorithm is the cubic polynomial condition of having root.

Given $f(t) = At^3 + Bt^2 + Ct + D$; intersection points are roots of equation f(t)=0

$$f'(t) = 3At^2 + 2Bt + C$$

Extrema values of f(t) are at the value of f'(t) =0. By applying quadratic equation on f'(t)=0, values of e_0 and e_1 will be found.

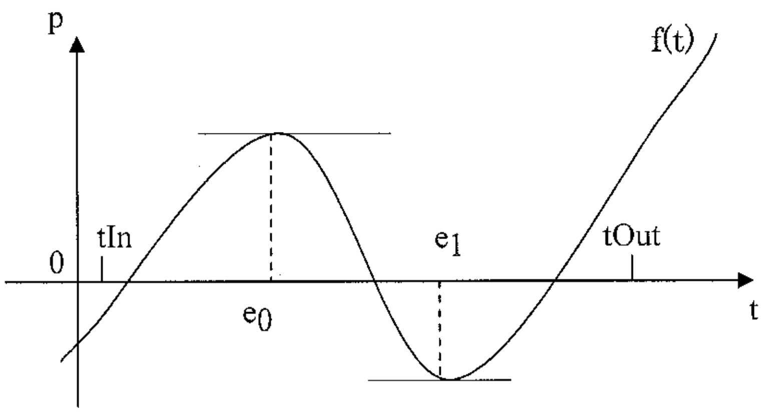


Figure 10: Exact linear interpolation

Above diagram illustrates a cubic polynomial with 3 roots in between tIn and tOut. In this case, at first, it calculates extrema values of f(t), which are e_0 and e_1. Then it determines an interval to further examine within 3 intervals $[t_{in}, e_0]$, $[e_0, e_1]$, $[e_1, t_{out}]$, which is $[t_{in}, e_0]$ since f(tIn) < 0, f(e0) >0 AND it is the nearest interval. Finally, it applies repeated linear interpolation [12] on that interval to find hit point. Pseudo code for this algorithm can be found in appendix 1.

This algorithm produces quite accurate intersection without complex calculations like Cardano's method. It is useful for closest hit intersection since it concerns to the nearest interval only.

2.2.2 Gradient Vector (normal vector)

This section discusses on normal vector estimation, or gradient estimation, at the intersection point of an iso-surface. Normal vector is essential for shading. The section analyzes 2 most common techniques, i.e. exact normal and rectilinear gradient. These techniques were implemented in the program. This section also mentions to smooth exact algorithm.

- **Exact Normal**

Given a trilinear volumetric data, normal vector at a point (x,y,z) is derived as $\vec{N} = \Delta\vec{p} = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}, \frac{\partial p}{\partial z}\right)$. Below derivation is to find $\vec{N}_z$. Similar processes can be apply for $\vec{N}_x$ and $\vec{N}_y$.

Since density value at a point (u,v,w) is given by $p(u,v,w) = \sum_{i,j,k=0,1} u_i v_j w_k p_{ijk}$

Density value at a point (u,v,w+Δw) is given by $p(u,v,w+\Delta w) = \sum_{i,j,k=0,1} u_i v_j wd_k p_{ijk}$

where $wd_1 = \frac{z+\Delta z - z_0}{z_1 - z_0}$ and $wd_0 = \frac{z_1 - z - \Delta z}{z_1 - z_0}$ or $wd_k = w_k + (-1)^{k+1}\frac{\Delta z}{z_1 - z_0}$

$$\Delta p = p(u,v,w+\Delta w) - p(u,v,w) = \sum_{i,j,k=0,1} u_i v_j (-1)^{k+1}\frac{\Delta z}{z_1 - z_0} p_{ijk}$$

$$\vec{N}_z = \frac{\Delta p}{\Delta z} = \sum_{i,j,k=0,1} (-1)^{k+1}\frac{u_i v_j}{z_1 - z_0} p_{ijk}$$

$$\vec{N}_x = \frac{\Delta p}{\Delta x} = \sum_{i,j,k=0,1} (-1)^{i+1}\frac{v_j w_k}{x_1 - x_0} p_{ijk}$$

$$\vec{N}_y = \frac{\Delta p}{\Delta y} = \sum_{i,j,k=0,1} (-1)^{j+1}\frac{u_i w_k}{y_1 - y_0} p_{ijk}$$

Exact gradient presents the exact shape of the surface. However, for the visualization purpose, exact gradient is not useful since the overall image is not smooth. Due to the fact that at the side of each voxel, there are cliffs and edges, surfaces do not smoothly match between voxels.

- **Smooth exact (slicing voxel)**

Smooth exact estimation uses a virtual voxel: the point to calculate normal vector is in the center, size of the voxel is the same as other voxels, planes are parallel with other voxels, density value at vertice is calculated by trilinear interpolation [14].

Smooth exact estimation algorithm introduces a smooth surface; however, there is the potential of loss data since it reduces the sharpness of edges.

- **Rectilinear gradient**

It is the most common technique to estimate the normal vector. The algorithm is as follow: In preprocess, it calculates normal vectors at grid points (vertices) and stores them. In rendering process, to find normal vectors at arbitrary points, it derives them by triliniear interpolation equation from stored grid point normal vectors.
The algorithm is fast, produces acceptable surface with less lost and smooth enough for visual effects.

2.2.3 Ray traversal

This section discusses on some algorithms of ray traversal. These algorithms includes simple stepping ray, and fast ray traversal by Amanatides and Woo[17].
Basically, ray traversal means to find which voxel that the ray enters at a value of t. Recall that a ray is represented as R(t) = O + D * t. Where t can be considered as time of traversal, O is the origin and D is the direction vector of the ray R.

- **Stepping ray**

This method is named by the author of this thesis, according to the way that the ray travels. This algorithm performs the traversal of the ray by increasing t with a value tStep. With a value t, the associated voxel can be computed by floor functions.

$$\text{Xvoxel} = \text{floor}\left(\frac{O_x + D_x * t}{voxelSize_x}\right)$$

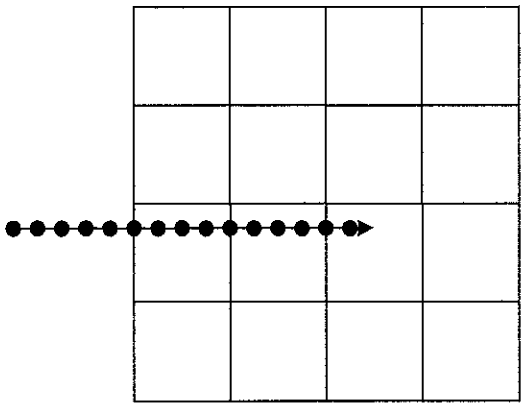


Figure 11: Stepping ray

It is simple to understand and implement, however, it contains potential of error. Since the ray traversal depends heavily on value of tStep, change in this value will lead to change on result.

As illustrated by below diagram, the ray may miss a voxel if the value of tStep is large enough and the direction of the ray is not parallel with any axis. Theoretically, the ray always miss some voxels since with every value of tStep, there is position of ray that makes interval AB smaller than tStep.

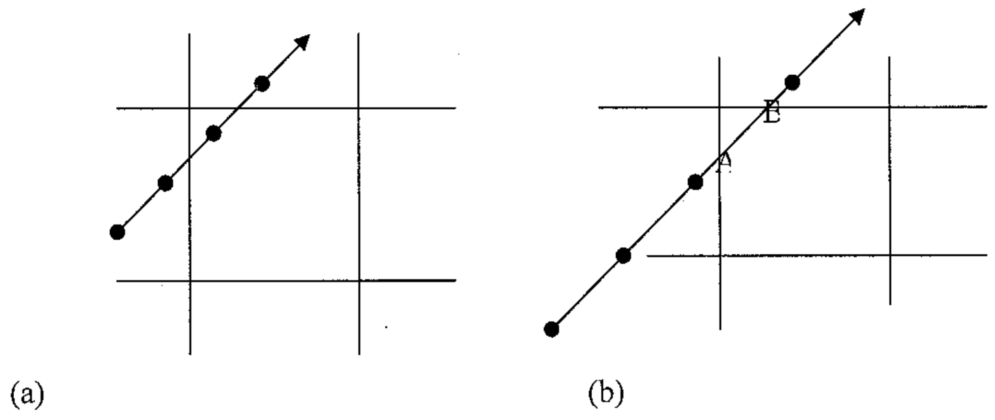


Figure 12: Stepping ray problem (a) Common situation (b) Miss-case

In practice, the algorithm produces picture with some dots on the iso-surface. Besides, this algorithm requires many calculations when increasing the value of t with tStep, which is small.

- **Ray traversal by Amanatides and Woo** [17]

In 1987, Amanatides and Woo developed a fast algorithm for ray traversal that nowadays many applications still use without changes. Below is the version of the algorithm in 2 D grid.

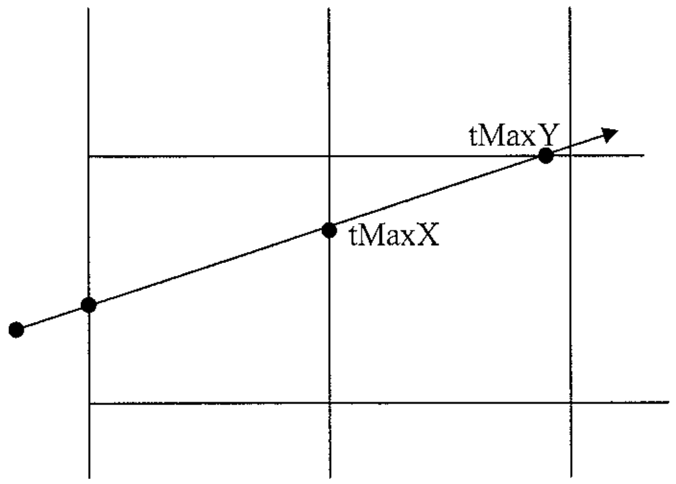


Figure 13: Amanatides and Woo's ray traversal algorithm

The algorithm use integer variable X and Y to store the voxel coordinates; tDeltaX and tDeltaY to indicate how long it takes the ray to travel the width and the height of a voxel; xStep and yStep to indicate the sign of ray direction, which is 1 if it is positive, -1 if it is negative. The algorithm has 2 two phases: initialization and incremental traversal.

- *The initialization phase* is to identify the voxel that contains the ray origin or the first voxel that the ray hits the volume data. X and Y are set with this voxel coordinates. Next, it finds the value of tMaxX and tMaxY as the first value of t at which the ray crosses the first vertical and horizontal voxel boundaries. The minimum of these two values will indicate how much we can travel along the ray and still remain in the current voxel.

- *The incremental phase* is a simple loop:

```
loop{
      if(tMaxX < tMaxY)
      {
            tMaxX= tMaxX + tDeltaX;
            X= X + stepX;
```

```
        }
        else
        {
                tMaxY= tMaxY + tDeltaY;
                Y= Y + stepY;
        }
        VisitNextVoxel(X,Y);
}
```

The incremental phase basically finds which boundary that the ray will hit first, and to increase coordinates of the voxel accordingly.

In the program of this thesis, the algorithm is slightly changed. It uses 2 more variables oldX and oldY to store the most recent visited voxel and further examine that voxel. The purpose of visit old voxel is to get tIn and tOut without redundantly comparing tMaxX and tMaxY.

- **Ray traversal in partition space**

In this project, recursive algorithm was used to travel a ray through partition space. When the ray hits a parent node, the algorithm finds a local ray that enters the node and applies Amanatides and Woo's algorithm on that ray. There is another way to implement it; one can, instead of casting local ray, change the initialization according to the size of child-node. However, the computational cost is almost the same since both algorithms have to find the coordinates of first hit child node.

It is difficult to say if the algorithm is good or bad. Basically, the effectiveness of an algorithm depends on nature of dataset. It leads to a phenomenon called "teapot in stadium" [17], where the scene consists of a huge empty space with only one small object at center.

2.2.4 Acceleration by space partition

This section discusses on the effectiveness of different space hierarchy on the speed of the program. There are 2 hierarchies were used, which are nested grid and octree. Theoretically, octree is the most effective hierarchy, however, in practice with this data set, it show some disadvantages.

- **Nested Grid**

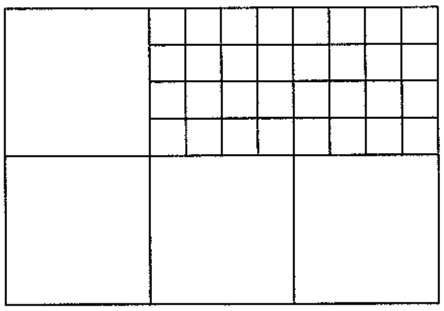


Figure 14: Nested grid

This hierarchy at first divides the scene with a large grid. Neither number of column nor row is necessary to be power of 2. Secondly, it divides each grid which contains object into smaller grid, and so forth. In this project, nested grid shows more advantages than octree.

- **Octree**

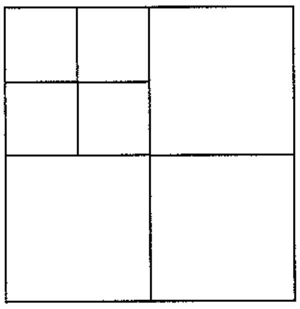


Figure 15: Octree hierarchy

Octree has the same construction as nested grid; the different is each node has exactly 4 children. In this project, the volumetric data is partitioned by octree until each node contains only 1 voxel. It takes about 7 levels. The experiment is to show the effect of number of level.

2.3 Proposal of new multisampling schema

"Eureka"- Archimede

Multisampling is a method to reduce alias effect. Increasing number of ray leads to increasing computing time. This section proposes an anti-aliasing sampling algorithm with the acceleration of ray traversal. Its aim is to reduce aliasing with less time. Basis of the algorithm is to cast sampling rays directly to the object/voxel, which was detected by a primary ray. The algorithm saves time of ray traversal for sampling rays.

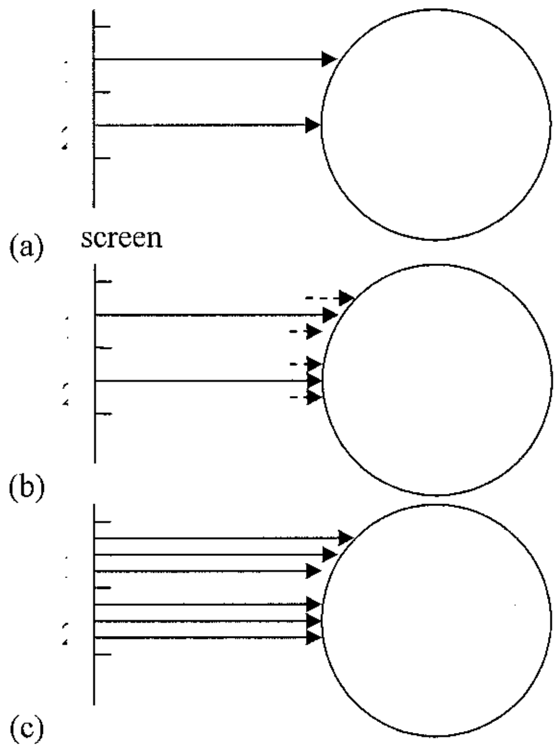


Figure 16: Proposal multisampling algorithm (a) Cast primary rays to find object (b) Cast sampling rays directly to objects (c) Existing sampling techniques require sampling rays travel from screen.

The new algorithm has 3 phases. For each pixel on screen, do the follows:

- Cast primary ray to find objects/voxels that the ray intersects with. Store those objects/voxels into memory.

- Cast sampling rays directly to appropriate stored objects/voxels.
- Apply integral methods to determine the color of screen pixel.

The algorithm was implemented with a laptop Centrino Mobile 1.5GHz. Sample set is BrainSmall, 128 * 128 * 84, 1:1:1, example dataset associated with Volpack package, Standford University. Supersampling, 9 samples per pixel, image 512*512.

Single ray	Normal multi-sampling	New multi-sampling
2 secs	14 secs	9 secs

Table 1: Rendering time of different sampling algorithm

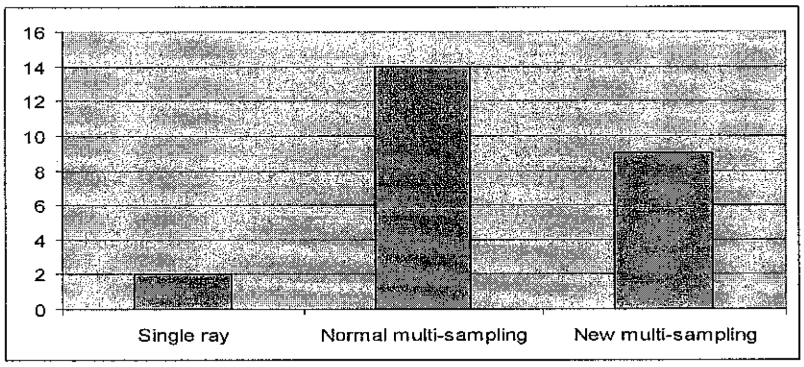


Figure 17: Rendering time of different sampling algorithm

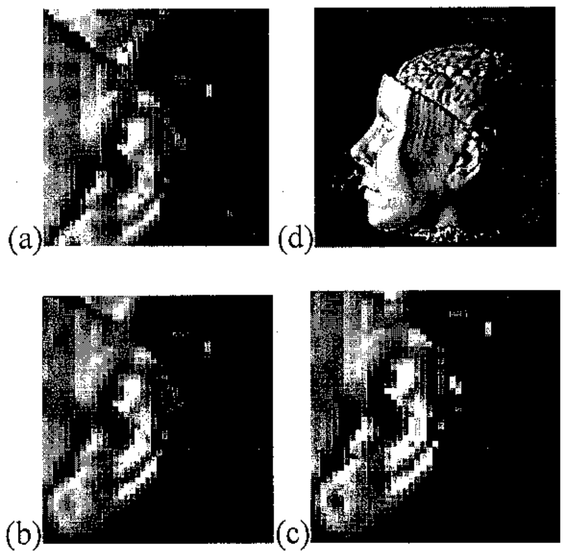


Figure 18: **Anti aliasing in volumetric data** (a) New algorithm (b)Normal algorithm (c) Single sample (d) Higher resolution produce the same result for 3 algorithms

This algorithm fastens the process of sampling because sampling rays do not have to travel from screen to object. It does not increase the resolution because no new object is discovered. Small objects still can be missed.

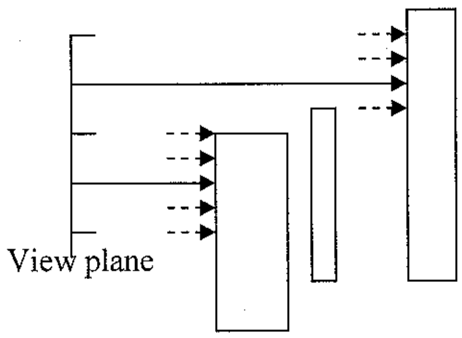


Figure 19: Miss-case

Although the rendering engine is not yet optimized, dramatic decrease in rendering time proves the advantage of new algorithm. New algorithm is faster because it eliminates traversal time for sampling rays.

CHAPTER 3

METHODOLOGY/PROJECT WORK

This chapter describes about the architecture and module of the program. In general, the program implements algorithms that are described in section 2.3 in the previous chapter.

3.1 Functions

The program is a ray casting program, was created with the purpose of implementing and comparing specific algorithms. These algorithms include space partition, ray traversal, ray-isosurface intersection, and gradient estimation. The program does not implement global illumination.

Recall that the interval value of t should not include 0, however, since there is no secondary rays were generated, the interval value is assigned as $[0,+\infty)$

In general, the program has following functions

- Data input: to capture dataset BrainSmall into a 3D array of float. The module can be modified to capture other file formats.
- Preprocess : to analyze captured volume data and generate pre-requisite data such as normal vector at grid point
- Ray casting: to cast a ray through each pixel on screen into volume data. There are 3 different sub modules that perform 3 different algorithms.
- Intersection interpolation: to calculate hit point of each ray on iso-surface.
- Gradient estimation: to calculate normal vector at hit point.
- Color determination: to assign a color on a pixel on the screen. The color is calculated with Phong shading.
- Image Output: to generate the final image. The format of the image file is ppm.

3.2 Program flow

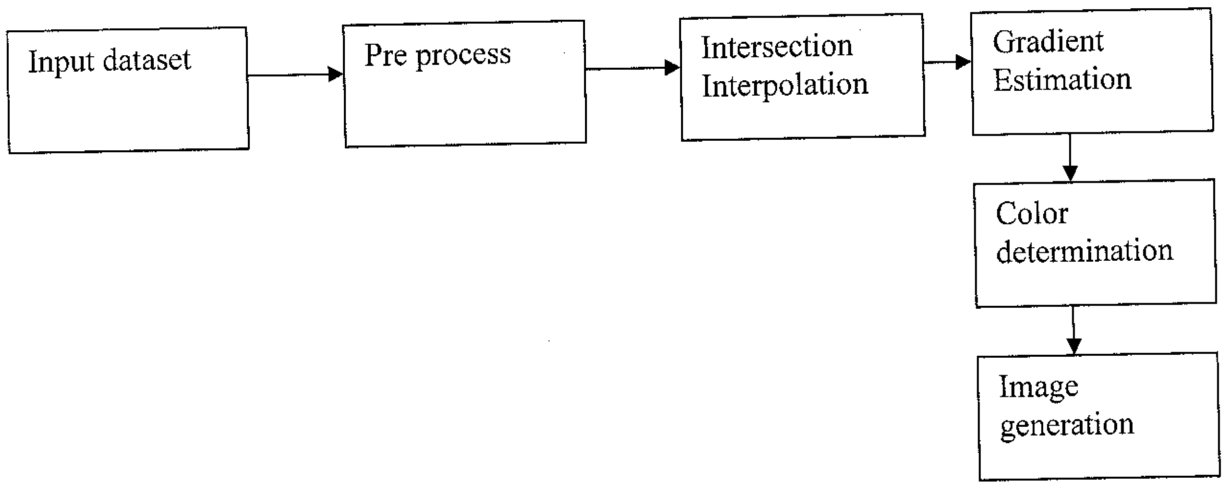


Figure 20: Program flow

3.3 Psedo code

The main program psedo code

```
FOR EACH pixel O ON screen
        Cast a ray R that originated from O
        IF the ray hit the surface
                Calculate gradient vector at hit-point
                Calculate hit-point color
                Color the screen pixel
        ELSE
                Color the screen pixel with default value.
        END IF

END FOR
```

3.4 Modules

The program of the project has a structure that is described in the book "Realistic ray tracing" refer to Peter Shirley and Keith Morley (2003). The program has been developed by starting with simple, standard source code, and then modifying to meet objectives. Listed source code below is only a portion of the whole program.

Box.h and Box.cpp : to declare and implement properties and functions of box (voxel). These files contain source code from Amy Williams, Steve Barrus, R. Keith Morley, and Peter Shirley "An Efficient and Robust Ray-Box Intersection Algorithm" Journal of graphics tools, 10(1):49-54, 2005; source code from Jochen Schwarze. "Cubic and quartic roots". In Andrew Glassner, editor, Graphics Gems, pages 404–407. Academic Press, San Diego, 1990.

cell.h and cell.cpp : to declare and implement properties and functions of cell, used to implement space partition hierarchy.

denfile.h and denfile.cpp : to read volume data file. The files are parts of Volpack package publication. Retrieved from the Stanford Computer Graphics Laboratory's by August 15, 2005.

Web page http://www-graphics.stanford.edu/software/volpack/#Distribution or ftp://www-graphics.stanford.edu/pub/volpack/.

image.cpp image.h: to generate image, the files are parts of source code from the course Advanced Computer Graphics II, Spring 2005 by Steven G. Parker. Retrieved from University of Utah by August 1, 2005.
Webpage http://www.cs.utah.edu/classes/cs6620/

ray.h: to implement ray as in paper from Amy Williams, Steve Barrus, R. Keith Morley, and Peter Shirley "An Efficient and Robust Ray-Box Intersection Algorithm" Journal of graphics tools, 10(1):49-54, 2005

rgb.h, rng.h, vector3.h: to implement color, vector; are parts of source code from Peter Shirley, R. Keith Morley. "Realistic Ray Tracing" second editor. A.K. Peters Publisher 2003.

3.5 Argument

Compiled program is supported with several arguments, which allow users select different algorithms. The scene, includes view point, light and dataset, is static for each compiled program.

-r : ray traversal

-o : octree hierarchy

-s : stepping ray

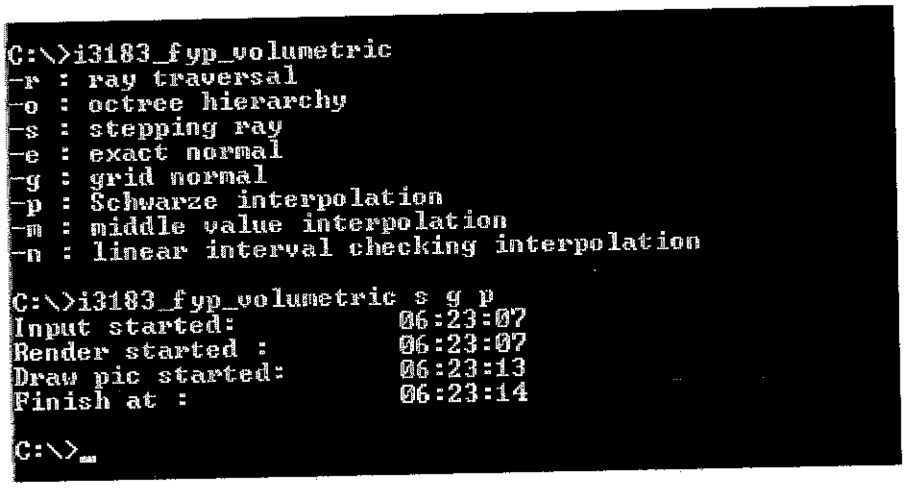
-e : exact normal

-g : grid normal

-p : Schwarze interpolation

-m : middle value interpolation

-n : linear interval checking interpolation



```
C:\>i3183_fyp_volumetric
-r : ray traversal
-o : octree hierarchy
-s : stepping ray
-e : exact normal
-g : grid normal
-p : Schwarze interpolation
-m : middle value interpolation
-n : linear interval checking interpolation

C:\>i3183_fyp_volumetric s g p
Input started:          06:23:07
Render started :        06:23:07
Draw pic started:       06:23:13
Finish at :             06:23:14

C:\>_
```

Figure 21: In-line command arguments

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results and comparisons

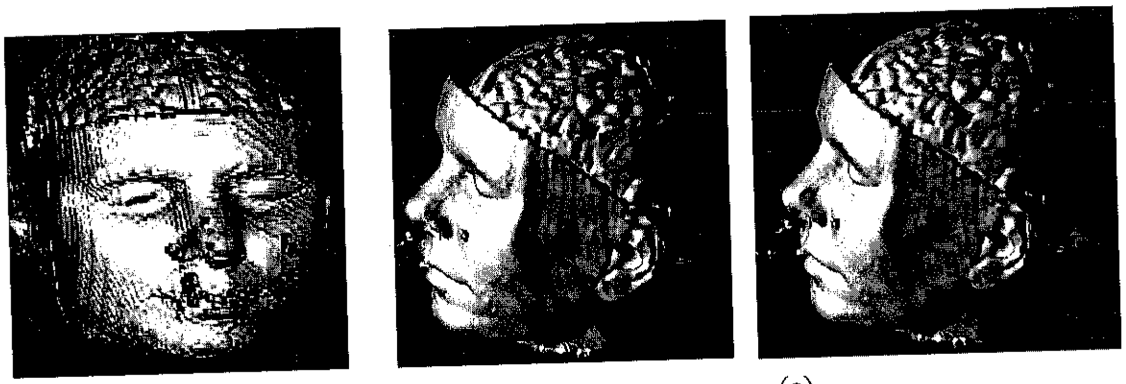
- **Changes in ε make significant changes in image.**

(a) ε = 1.e-6 (b) ε = 1.e-4

Figure 22: Schwarze approach : unstable accuracy.

- **Different interpolation algorithms**

The data sample is BrainSmall, 128 * 128 * 84, 1:1:1, example dataset associated with Volpack package, Standford University[20]



(a) (b) (c)

Figure 23: Different interpolation algorithms (a)Linear Interpolation (b) Schwarze's approach and exac linear interpolation (c) Linear Interpolation

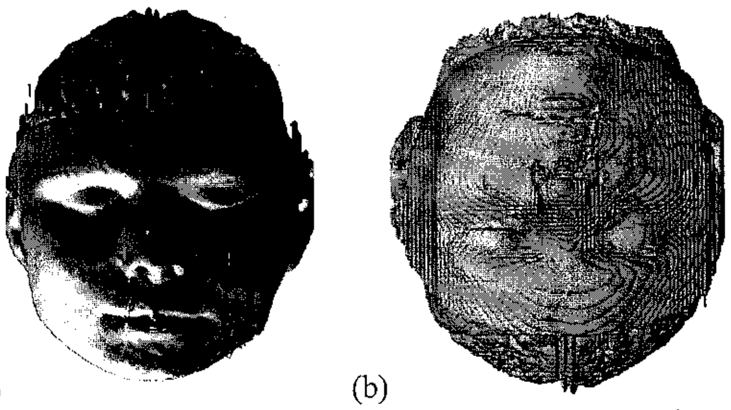


Figure 24 : Different gradient estimation algorithms (a) grid normal (b) exact normal

- **Different acceleration techniques**

Following results are rendering time of different algorithm on a laptop, 1.5GHz. Image 512*512

  - Stepping ray: 3 mins
  - Octree : 14 secs (7 levels of depth)
  - Uniform grid : 10 secs (3 levels of depth)

- **Proposal algorithm**

The algorithm was implemented with a laptop 1.5GHz, image 512 * 512, super sampling, 9 rays per pixel, 25 level depth, unit is second. Rendering engine is modified from Parker's course program [15]. The algorithm is applied on the first casting of the rays. Experiment shows that the new algorithm takes advantage when the scene has many objects.

No of Object	**1**	**7**	**13**	**19**	**25**	**31**	**61**
Single Ray	5	5	14	24	32	37	76
Normal Super Sampling	39	96	152	232	283	344	700
New Super Sampling	55	91	141	183	212	258	501

Table 2 : Rendering time experimental result with object scene

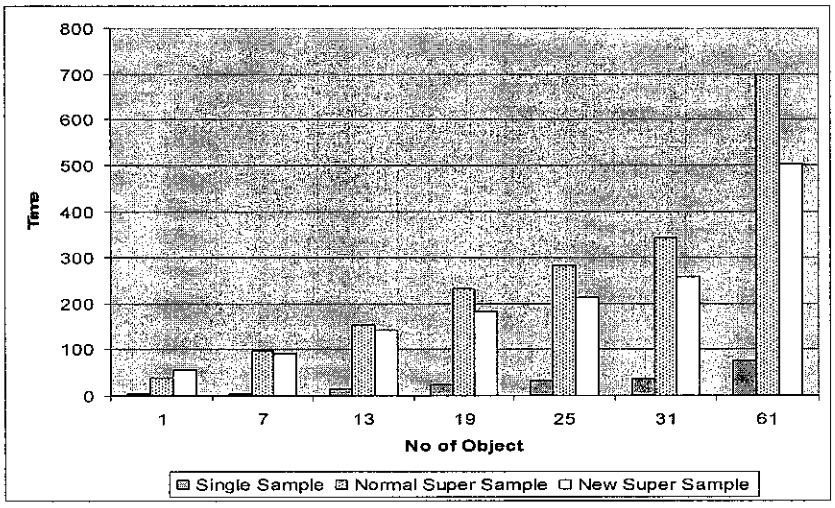


Figure 25 : Rendering time

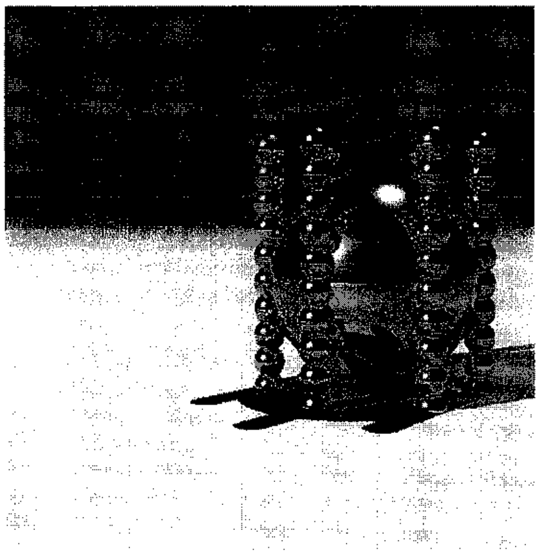


Figure 27: Single sample ray tracing

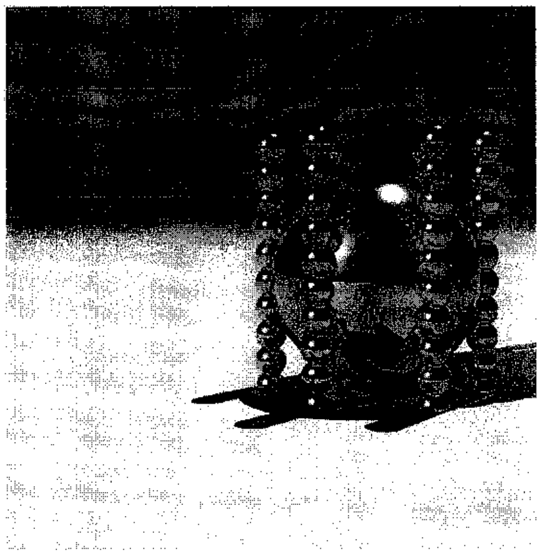


Figure 26: Normal super sampling algorithm and new super sampling algorithm produce the same image

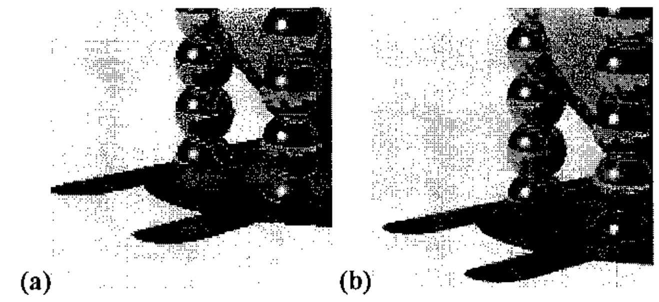


Figure 28: Comparison between single and multi sampling

4.2 Discussion - Future work and research areas

"In the beginner's mind there are many possibilities, but in the expert's there are few."

Shunryo Suzuki-Roshi

This section mentions about a few ideas that may be further research.

Topic	**Description**	**Field of study**	**Potential problem**
Space partition according to dataset pattern	To develop an algorithm/schema that can determine technique of space partition according to dataset pattern		
Course of viewpoint	To develop sets of courses of view points, to accelerate interactive rendering by mapping between view point and to-be-examinated voxels. Those courses have to fit with specific purpose of viewing and satisfy users' needs. New courses will be added into the set on the spot when users navigate the volume	Machine learning, data mining	
Local ray algorithm	To find a more efficient algorithm for casting local ray in space partition		
New global illumination algorithm	to develop a formula for gradient interpolation at a point (x,y,z) in a scene Object[i], LightSource[i]. To simplify the reflecting light from other objects, i.e. triangles become points. Given object at specific coordination with (x,y,z) contributes approximated illumination. Contributed illumination from each object is the accumulation of its triangles. Occlusion contributes negative value.	Photon mapping	Shape of light source will not be easy to handle

Table 3 : Proposal topics for further research

CHAPTER 5

CONCLUSION AND RECOMMENDATION

The thesis consists of 4 main sections: literature review, implementation of selected algorithms, analysis and comparison, and enhancements and areas of research. The thesis proposes a new algorithm to accelerate multisampling rendering and suggests a few topics for further research.

Literature review (section 2.1) summarizes basic knowledge of ray tracing, from definition of ray, primitive to fundamental concepts of volumetric rendering, ray acceleration.

Implementation of selected algorithms (section 2.2) is deeper research on 4 areas of ray tracing. These groups are intersection interpolation, gradient estimation, ray traversal and ray acceleration. Research on each group consists of analysis of a few algorithms and programming implementation of them. Different algorithms produce different results, they are then compared (section 4.1) for a match between theory and practice.

Areas of research (section 4.2) describes some ideas with that the author came up. These ideas may be not good enough for possible researches; however, they also may be new ways to see old things. The thesis includes a proposal of new algorithm for ray acceleration in multisampling (section 2.3). The new algorithm eliminates traversal time of sampling rays from view plane to objects, thus reduces rendering time. Miss-case with small objects is the limitation of the algorithm.

REFERENCE

[1] Peter Shirley, R. Keith Morley. (2003) *Realistic Ray Tracing* second edition. A.K. Peters Publisher 2003. ISBN 1-56881-198-5

[2] Ingo Wald (2004) *Realtime Ray Tracing and Interactive Global Illumination.* Ph.D Thesis. Saarland University. Retrieved from University of Utah by September 14, 2005. Website: http://www.sci.utah.edu/~wald/PhD/wald_phd.pdf

[3] Pascal Vuylsteker (2003). *Lecture Note.* Retrieve from The Australian National University by August 16, 2005.
Website : http://cs.anu.edu.au/student/comp4610/plan.html

[4] Alexander Keller (2002) *Monte Carlo and Beyond.* Course note. ETH Zürick 2002. Retrived from Universität Ulm by October 3, 2005.
Website: http://graphics.uni-ulm.de/BeyondMonteCarlo.pdf

[5] Joe Kniss, Patrick McCormick, Allen McPherson, James, Ahrens, Jamie Painter, Alan Keahey, Charles Hansen (2001) *Interactive Texture-Based Volume Rendering for Large Data Sets* IEEE 2001. Retrieved from University of Utah by October 20, 2005.
Website : www.cs.utah.edu/~jmk/TRex_CGA.pdf

[6] William E. Lorensen and Harvey E. Cline.(1987) *Marching cubes:A high resolution 3d surface construction algorithm.* Conference Proceedings . ACM Siggraph 1987. Retrieved from Israel Institute of Technology by October 20, 2005
Website: www.cs.technion.ac.il/~u_shani/cs236807-S2/lecturesstudents/Marching Cubes.pdf

[7] Arie E. Kaufman. *Volume Visualization: Principles and Advances.* Retrieved from Mitsubishi Electric Research Laboratories by June 14, 2005.

Website:http://www.merl.com/people/pfister/courses/Bonn2000/Papers/KaufmanVolume Visualization.pdf

[8] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, Peter-Pike Sloan (1999) *Interactive Ray Tracing for Isosurface Rendering.* Retrieved from University of Utah by August 16, 2005.
Website: http://www.cs.utah.edu/sci/publications/ieee_tvcg99/tvcg99.pdf

[9] Milan Ondrus (2004) *Algorithm in K.* Retrieved by November 14, 2005.
Website http://homepage.hispeed.ch/milano. 2004

[10] Jochen Schwarze(1990) *Cubic and quartic roots.* In Andrew Glassner, editor, Graphics Gems, pages 404–407. Academic Press, San Diego, 1990.

[11] Ingo Wald, Heiko Friedrich, Philipp Slusallek, Gerd Marmitt, Andreas Kleer(2004) *Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing.*VMV 2004. Retrieved from Universität Des Saarlandes by August 16, 2005.
Website: graphics.cs.uni-sb.de/VolumeRT/ Publications/IsoIsec/IsoIsec_VMV2004.pdf

[12] A. Neubauer, L. Mroz, H. Hauser, R. Wegenkittl (2002).*Cell-based first-hit ray casting.* Proceedings of the Symposium on Data Visualisation 2002. Retrieved from VRVis by September 3, 2005.
Website: medvis.vrvis.at/fileadmin/publications/TVCG_printed.pdf

[13] Torsten Möller, Raghu Machiraju, Klaus Mueller, Roni Yagel (1997) *A Comparison Of Normal Estimation Schemes.* IEEE. 1997. Retrieved from State University at Stony Brook by Octorber 12, 2005.
Website: www.cs.sunysb.edu/~mueller/ papers/vis97_NormalSchemes.pdf

[14] Paul Agron (2002). *Isosurface Viewing in Volume Rendering.* Course paper. Retrieved from State University at Stony Brook by Octorber 12, 2005.

Website: http://www.cs.sunysb.edu/~pagron/report.pdf

[15] Steven G. Parker (2005) *Lecture note* CS6620, Advanced Computer Graphics II. Spring 2005. Retrieved from University of Utah by August 1, 2005
Website: http://www.cs.utah.edu/classes/cs6620/

[16] John Amanatides, Andrew Woo (1987) *A fast voxel traversal algorithm for ray tracing*. Eurographics '87. 1987. Retrieved from York University by August 18, 2005.
Website www.cs.yorku.ca/~amana/research/grid.pdf

[17] A. S. Glassner. (1984) *Space subdivision for fast ray tracing*. IEEE Computer Graphics and Applications, pages 15, October 1984.

[18] Amy Williams, Steve Barrus, R. Keith Morley, Peter Shirley. (2005) *An Efficient and Robust Ray-Box Intersection Algorithm*. Journal of graphics tools, 10(1):49-54, 2005

[19] Greg Humphreys (2003). *Ray Tracing Acceleration Techniques*. Lecture note. Image Synthesis. Retrieved from University of Virginia by October 11, 2005.
Website:www.cs.virginia.edu/~gfx/Courses/2003/ImageSynthesis/scribed_notes/03_acceleration.pdf

[20] Philippe Lacroute (1994) *The VolPack Volume Rendering Library*. Retrieved from Stanford University by August12, 2005
Website: graphics.stanford.edu/software/volpack/

APPENDIX

Appendix 1

Pseudo code for interpolation algorithms

Ingo Wald, Heiko Friedrich, Philipp Slusallek, Gerd Marmitt, Andreas Kleer *Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing* (2004).

Pseudo code for linear intersection

```
pIn := f(tIn);
pOut := f(tOut);
if sign(pIn − pIso) = sign(pOut − pIso) then
    return NO HIT
end if
return tHit := tin + (tout − tin)((pIso−pIn)/(pOut−pIn));
```

Pseudo code for exact repeated linear intersection

```
t0 = tin; t1 = tout; f0 = f(t0); f1 = f(t1)
if f0 has real roots then e0 = smaller root of f0
    if t0 ≤ e0 ≤ t1 then
        if sign(f(e0)) = sign(f0) then
            t0 := e0; f0 := f(e0)
        else
            t1 := e0; f1 := f(e0)
        end if
    end if
    e1 = second root of f0
    if t0 ≤ e0 ≤ t1 then
        if sign(f(e1)) = sign(f0) then
            t0 := e1; f0 := f(e1)
        else
            t1 := e1; f1 := f(e1)
        end if
    end if
end if

if sign(f0) = sign(f1) then
    return NO HIT;
end if
```

```
for i=1..N do
    t := t0 + (t1 − t0) *((pIso - f0)/(f1−f0))
    if sign(f(R(t))) = sign(f0) then
        t0 := t; f0 = f(R(t))
    else
        t1 := t; f1 = f(R(t))
    end if
end for

return tHit := t0 + (t1 − t0) *((pIso - f0)/(f1−f0))
```

Appendix 2

How did love end?

1. Stars.

A student who is working on a ray tracing program was having a walk with his girlfriend. It was a romance night with sparkling stars on the black sky. They were watching in silence until they saw a pair of very bright stars which are close together. The girl said "Honey! Look at them, the left one is you, and the right one is me. I am always by your side." The boy looked at them for a while and replied "My love! We are looking upward, the direction is positive; I think the rays miss the front-top-corners of the voxels."

2. A new algorithm

The girl tried a new makeup style. She was very happy while the boy kept gazing at her face. She said "Honey! I will keep this style since you really love it". The boy replied "I still see some sharp edges on the surface. I think you need to find another algorithm for your normal vectors"

3. And the consequence.

The boy finished his project and went to meet the girl after a long time busy with coding. He was in shock to see her with another guy. And he was horrified to hear his ex-girlfriend's saying "Honey! Your rays skip everything. I do love you, but your bounding box was too tight for a place for me." (The End)

"*Do love your work, don't miss your love*" - Do, Anh