# Towards Optimal Decomposition Techniques

# For Ray Traced Terrain Visualisation

Paul Francis Michell

0059502

MSc Computing

Liverpool Hope University

# Plagiarism Declaration

**This page must be signed by the student, in the presence of the supervisor or award director. It must be bound in with the dissertation. By signing this page, the student is acknowledging that they have read and understood the statement.**

All sentences or passages in this dissertation from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any and all other materials that are not the work of the author of this dissertation (for example, but not limited to, computer programs and code samples, illustrations, diagrams, collected data, audio and video material) are specifically acknowledged by clear reference to the source. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and in the degree examination as a whole.

Name (Please use block capitals)

Signature

Date

Witnessed by (supervisor or award director):

Name (Please use block capitals)

Signature

Date

# Acknowledgements

Mr. T. Oliver, Forestry Commission North West, for supply of OS elevation data.

USGS for making US digital elevation models publicly available.

Dr. Mark Anderson for guidance and encouragement.

# Abstract

Terrain rendering is a complex but well understood process, which has largely been restricted to scientific visualisations, computer games, film and video. However, a new audience is developing within the GIS user community. Here various 3D presentation techniques are being used with ever larger spatial datasets. Also, detailed elevation models are now becoming widely available, allowing spatially referenced information to be combined with actual ground topology.

Ray tracing has been successfully used to render large outdoor scenes using realistic natural lighting. It has historically been viewed as a slow but high quality mechanism for 3D graphics, where a single image can typically take minutes, or sometimes hours, to generate. Where dynamic interaction with a 3D model is desired, an alternative method known as scan-line rasterisation is usually employed, often accelerated by graphics processing hardware. This technique for displaying terrain is by nature less realistic, but it is currently the only method employed by commercial GIS applications. As geo-spatial datasets increase in size the speed advantages of rasterisation diminish exponentially.

This study investigates the suitability of ray tracing as a viable alternative to hardware rasterisation for visualising large scale terrain datasets and establishes the computing resources required to produce images at interactive display rates. Wherever possible, the higher quality output methods available to ray tracing are retained, which adds a qualitative dimension to the case for adopting this technique.

Use of parallelism is explored and exploited to achieve interactivity on a single PC using a recently developed multi-core CPUs. A careful investigation is made of the domain and functional decompositions possible within the ray tracing paradigm. The results demonstrate that interactive display can be achieved on modern dual core CPUs at quarter VGA resolution. This is achieved without reliance on display hardware, or extensive pre-processing of the scene data. It also predicts from extrapolation of the results, that full DVD quality could be achieved using a typical office network of modern personal computers and that the performance will scale in a near linear manner with the number of available processors.

# Table of Contents

# Table of Figures

# Abbreviations

| | |
|---|---|
| 2D | Two Dimensional |
| 3D | Three Dimensional |
| API | Application Programming Interface |
| BRDF | Bidirectional Reflectance Distribution Function |
| BMP | Bit Mapped Picture (graphic file format) |
| CISC | Complex Instruction Set Computer |
| CPU | Central Processing Unit |
| DEM | Digital Elevation Model (a USGS defined terrain elevation data format) |
| DVD | Digital Versatile Disk (a disk format and video standard similar to VGA) |
| FPS | Frames Per Second |
| GADD | Geometric Atmospheric Density Distribution |
| GIS | Geographic Information Systems |
| HDR | High Dynamic Range |
| IBM | International Business Machines |
| IDE | Integrated Development Environment |
| JPEG | Joint Picture Engineering Group (graphic file format) |
| MIMD | Multiple Instruction Multiple Data |
| MISD | Multiple Instruction Single Data |
| NTF | National Transfer Format (OS spatial data encoding file format) |
| OpenGL | Open Graphics Library (programming interface specification) |
| OS | Ordnance Survey (UK governmental mapping agency) |
| PC | Personal Computer |
| PNG | Portable Networked Graphics (graphic file format) |
| RGB | Red, Green, Blue (triple value colour representation system) |
| SIMD | Single Instruction Multiple Data |
| SISD | Single Instruction Single Data |
| SSE | Streaming SIMD Extensions |
| TCP | Transmission Control Protocol |
| USGS | United States Geological Survey |
| VGA | Video Graphics Array (a standard display resolution of 640x480 pixels) |

Chapter 1

# Introduction

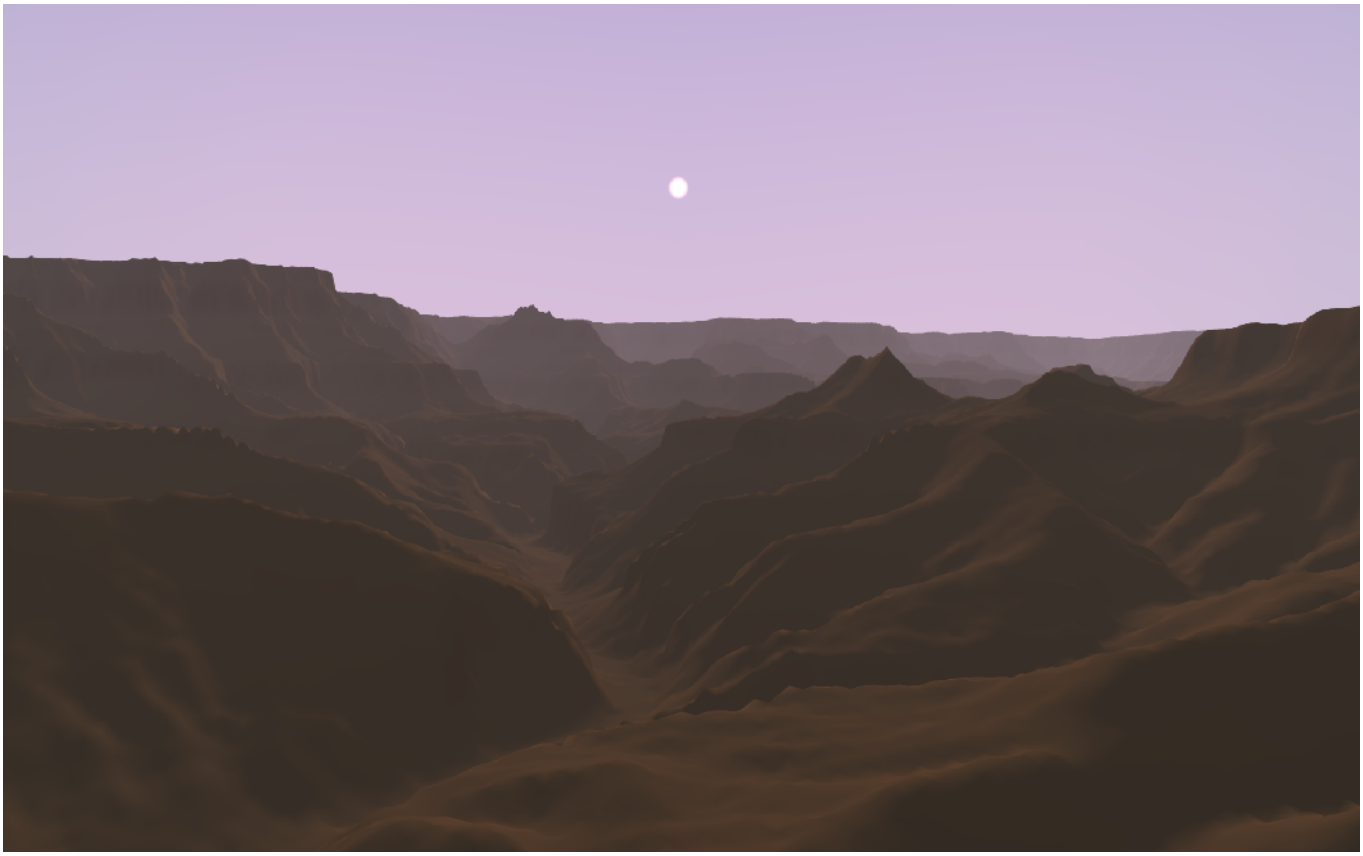


*Figure 1: Simulated Grand Canyon sunset rendered from USGS DEM data*

*"Some applications of computer graphics require a vivid illusion of reality."*

It is unlikely that Arthur Appel was able to conceive the full significance of this statement when he wrote it back in 1968 whilst working for the IBM Research Centre (Appel 1968). Neither could he have foreseen how quickly his theoretical insights would have translated into common-place technology. For thanks to the exponential development of computing power, sophisticated computer graphics systems are now available as cheap consumer devices. Indeed his vision of synthetic imagery was primarily directed at high end manufacturing and architectural visualisation. The now common uses in data presentation and, most of all, entertainment would have seemed inconceivable at a time when CPUs were rated in kilo Hertz and required elaborate programs to perform even basic multiplication or division.

There are two[1] fundamental methods for generating 3D images that have been developed by computing researchers. The first is referred to as scan-line rasterisation and the second ray tracing[2]. Rasterisation projects a scene's three dimensional geometry into two dimensional figures that are then drawn to the image. The order of drawing has to be carefully controlled so that the final picture contains the objects

[1] Assuming pure vector graphic displays can be ignored, as they are now no longer in general use.

[2] The actual term ray tracing has come to have a more precise meaning that will be discussed later. It is used here in the looser sense of any method that determines image pixel colouration by direct projection into a 3D scene.

correctly arranged by distance into the scene. Alternatively, an ancillary buffer may be employed, generally called the Z buffer, to mask drawing operations that should be concealed by elements already rendered. Ray tracing, by contrast, attempts to reconstruct the actual three dimensional ray[3] paths from the view point directly into the 3D model of the scene, for each image pixel.

It was quickly appreciated that whilst ray tracing had the potential to generate visually more sophisticated images, it had a far higher computational cost than an efficient scan-line algorithm. Consequently, almost all interactive 3D development has relied on rasterisation. Interestingly though, Appel (1968, 37) also observed:

> *"If techniques for the automatic determination of chiaroscuro with good resolution should prove to be competitive with line drawings, and this is a possibility, machine generated photographs might replace line drawings as the principle mode of communication in engineering and architecture."*

This is indeed the acid test: Can ray tracing now be competitively compared to rasterisation? Ray tracing has long been known to have alogarithmic time complexity with respect to scene size[4], whereas rasterisation has a linear time complexity (Rubin and Whitted 1980). This suggest that, however much slower ray tracing is, there must be a break-even point as the scene size increases. It has taken over thirty years for hardware advances to bring the computing world to the point where it is worth reconsidering this issue. In fact, it is perhaps worth re-stating the question as: have 3D scenes grown to a size that their data would be more efficiently visualised by ray tracing?

This study restricts itself to considering the potential uses of ray tracing in the field of GIS data terrain rendering. This is a rapidly growing field which is demanding ever greater sophistication in its visualisation techniques. To quantify the factors involved for this scenario an experimental software system was developed containing a parallel terrain ray tracer and an OpenGL based hardware rasteriser. With this, it is possible to directly compare the two methods of rendering for any given scene.

Whenever computing performance is being analysed the hardware platform must be considered as this will have an a priori influence upon execution times. CPU manufacturers have been struggling in recent years with two design issues. Firstly, CPUs with high clock rates generate a lot of heat. Effectively dissipating this heat is proving very difficult, especially in small form factor systems, such as laptops. Secondly, the increased processing power of these faster chips can rarely be matched by sustained communication with other parts of the system without drastically increasing the manufacturing cost. This leads to the general

[3] A ray in this context is the geometric representation of a 3D line segment using an origin point and a direction vector, often accompanied by other qualitative data required by the rendering algorithm.

[4] This is true assuming the scene is spatially partitioned using an algorithm that exhibits logarithmic spatial access, such as octrees, binary space partitions, or bounding volume hierarchies.

under-utilisation of the CPU in most desktop computers as they idle waiting for input. In response to these concerns, chip development has shifted towards adding multiple processing cores rather than just increasing clock speeds to prevent the stalling of Moore's Law[5].

Unfortunately though, multi-core processors can only convey their performance benefits to software that is designed from inception for parallel execution. Conventional rasterisation is not ideally suited for parallelism, whereas ray tracing is often cited as an example of an embarrassingly parallel[6] process (Allen and Wilkinson 2005, Günther et al. 2004). Thus if this hardware development trend continues, it will only serve to favour ray tracing in the medium to longer term. On the other hand, thanks to the popularity of 3D games, most modern consumer PCs contain dedicated hardware for rasterisation. So, at present, it enjoys a significant performance advantage, but only provided the scene data can be transferred at a rate consummate with the desired display frequency. These hardware issues will be discussed further and have considerable impact upon the experimental results.

Since parallel processing has the potential to tip the balance in the favour of ray tracing, it was essential to make this design method the foundation of the experimental renderer. Parallelism is, in essence, the subdivision of one large sequential task into a group of smaller concurrent tasks and thus achieves shorter execution times by being limited only by the length of the longest of the concurrent tasks. In short it is the computing application of the ancient maxim "divide and conquer!". Obviously, this does require the existence of multiple processing units capable of simultaneous operation. Therefore the usefulness of parallel processing on conventional single CPU PCs is somewhat moot, but as will be discussed later, it is still possible to make use of this existing processing power, in an office network context, by taking advantage of clustering methods.

There are two approaches for writing parallel programs, domain and functional decomposition. Domain decomposition, alternatively known as data partitioning, is concerned with the subdivision of the task's input data into smaller chunks which may then be processed simultaneously by separate but identical processing units. Functional decomposition, on the other hand, is the restructuring of the program's algorithm into loosely coupled procedures that exhibit some form of concurrency that may be exploited across multiple processing elements, rather in the manner of a factory production line. This is generally regarded as a much harder process to perform (Allen and Wilkinson 2005, 106) as it requires much more careful coordination between the tasks and the data they process. Both forms of decomposition can be implemented at different levels of granularity. Also, it is quite common for problems to exhibit less opportunity for functional than domain decomposition.

[5] That CPU processing power (actually circuit densities) will double every eighteen months to two years for a given manufacturing unit cost (Intel Corp. 2006a).

[6] A term invented by Geoffrey Fox (Fox, Messina and Williams 1994, chp. 7), in an early publication on parallel computing, to classify a problem that has a self-evident parallel implementation.

## Project Aims

The aim of this study is to investigate the application of ray tracing to terrain rendering by the development of an experimental rendering application. This is used to assess the specific objectives, but also to form the basis for ongoing research into this rapidly developing field. It additionally provides an opportunity to expand the authors software development skills and gain experience in cross-platform programming.

## Study Objectives

From an academic point of view there are two objectives. Firstly to provide a solid assessment of the real performance benefits achievable from applying parallelism to a terrain ray tracer. It is hoped that it will add to the growing body of evidence that contradicts Amdahl's law (Allen and Wilkinson 2005, 12; Günther et al. 2004, 9). The second objective is to assess the usefulness of functional decomposition within the realm of ray tracing, as this does not seem to have enjoyed much treatment in the academic literature.

Additionally, a professional objective exists. This is to assess the overall 'usefulness' of such a process to the commercial GIS world. This consideration should be twofold: is it a capable technique on current hardware and will it scale well to hardware that might be expected to be available in a year or two? In many ways the future question is the most important since it may well take that kind of time-scale to develop a production version of the renderer.

## Document Summary

- Chapter 2 details the previous relevant work carried out in the fields of ray tracing, terrain visualisation, parallel ray tracing and advance image generation.
- Chapter 3 describes the theoretical models adopted for use in the studies experimental software system.
- Chapter 4 covers the methodology employed to create the renderer and defines the metrics used to quantify the experimental results.
- Chapter 5 presents the performance data obtained from the experimental system.
- Chapter 6 Analyses the results and discusses the implications for the aims and objectives of the study. It also contains reflections on the methodology employed and future directions suggested by this work.

Chapter 2

# Previous Research

Four areas of prior academic work are pivotal to this study. Firstly, the classic papers on ray tracing were revisited. This was primarily to develop a clear foundation in the field, but also in recognition of the fact that it would not be possible to implement all of the latest techniques in the time-scale available for this project, so this material was most likely to inform the practical design of the test renderer. The second body of work concerns any previous investigations of parallel rendering as this is clearly germane. Thirdly, given the practical goal, previous research on terrain rendering and visualisation, especially ray tracing based methods would be very useful. Lastly, recent developments in photo-realistic synthetic image creation were reviewed to keep abreast of the state-of-the-art. A long term goal is to include as many of these new features as possible, as the qualitative advantages of ray tracing should not be forgotten in the quest to achieve interactive rendering.

## *Ray Tracing's Origins*

It is not now clear who first coined the term 'Ray Tracing'. It was certainly in use in computer papers as early as the 1960s. Indeed Turner Whitted who is usually cited as the father of modern ray tracing, himself cites several earlier works, most notably Appel (1968), who already used the term, but often without any precise intended meaning. The genesis of the idea seems to have occurred as a response to the limitations of scan line rasterisation[1]. It was realised that many of these limitations could be overcome by more closely modelling the actual optical processes of light transport. In fact an early motivating issue appears to be the search for a universal method for introducing plausible shadows to 3D images. Appel, whilst working for IBM, published several interesting papers on creating images by ray-path methods. Indeed, the main limiting factor on his work seems to be the computing capacity available to him.

## *Local Illumination Methods*

A key aspect of all 3D rendering techniques is the determination of the surface colouration of objects in a given scene. Initially, 3D computer graphics simply used Lambert's well known cosine law as the basis for surface shade determination. This was borrowed directly from optics theory and could be cheaply implemented in even a primitive shading system as the cosine is easily calculated by taking the inner vector product[2] of the surface normal with the lighting direction vector.

[1] It is beyond the scope of this study to discuss the scan-line process or its limitations in detail, but suffice to say the performance is bound by the scene size, since all drawing elements (or at least a given proportion of them) need to be considered by the renderer, the rasteriser is thus performance bound by the scene size. Additional issues surround handling transparency, reflections and shadows. This process is very well documented in computer graphics primers (Davis et al. 1999, 10-14; Watt 1989, 97-113). It should also be noted that it is the fundamental process employed by modern computer 3D display hardware and it is the mechanism behind the widely implemented OpenGL graphics programming library.

[2] Also known as the dot product. for brevity dot product will be used subsequently.

It was widely noted that naïve use of Lambert's Law created scenes where all objects were limited to perfectly diffuse reflectors[3]. Clearly the real world contained many more elaborate surfaces than this. Bui Tuong Phong (1973) in his Ph. D. thesis introduced an enhanced lighting equation which was quickly adopted as a useful standard by the graphics world. It added additional terms to the Lambert equation to allow for ambient light and specular highlights.

This provided the simulation of a far wider range of surfaces than was previously possible, without burdening the shading algorithm with too much additional computation. The Phong illumination model has been subsequently refined by Jim Blinn (1977) to optimise its computation and more closely mimic observed highlights on specular materials. This newer form is now known as the Blinn-Phong shading model and it is the calculation used within the OpenGL lighting specification.

There has been considerable furtherwork to advance the physical accuracy of illumination calculations. Most of these methods are derived from a theoretical understanding that specular surfaces are composed, at a microscopic level, of randomly arranged angular facets. The degree of roughness of these micro-facets directly determines the degree of diffusion of the resulting specular highlight. Several researchers have produced illumination models on this basis, the most widely used being the one proposed by Cook and Torrance (1982). This is the method usually employed in high-end rendering systems that seek to emulate the widest range of material types. The extra computational burden required does not make it suitable for interactive systems. For this reason, the Phong model and its variants are still the most widely used local illumination models today. It should be also be noted that any local lighting model may be used for scan-line rendering as well as ray tracing as they restrict themselves to calculating surface colouration without consideration of the wider scene geometry.

## *Early Modern Ray Tracers*

As for the development of ray tracing itself, it was Turner Whitted (1980) who was the first to propose a recursive algorithm as an elegant solution to determining the reflection and transmission ray paths through a scene. This process employed the Phong illumination model for surface lighting and in so doing, became the first system containing all of the archetypal features of a classic ray tracer.

A further improvement to this was later introduced by Carpenter, Cook and Porter (1984) called 'distributed ray tracing'. This traces more than one ray per pixel and more than one ray per surface interaction. These variations are perturbed slightly, and the results averaged. In this way it is possible, at the cost of proportionately longer render times, to add anti-aliasing, blurred reflections and refactions, depth of field, motion blur and soft shadows to a rendered image. The modern terminology used in computer graphics now appears to be that pre-Whitted, non-recursive processes are referred to as 'ray casting' and 'ray tracing' is

[3] Real examples of such dull surfaces would be matt paint or plain paper.

reserved specifically for the Whitted algorithm and its various offspring. Lastly it is worth noting that most research into ray tracing is initially published via SIGRAPH (the Special Interest Graphics Group) or announced in Ray Tracing News (Haines 2004).

## *Parallel Ray Tracing*

By contrast to the well established understanding of ray tracing, parallel rendering is still a relatively new phenomenon. There are only a few printed texts available concerning the practical implementation of a parallel rendering system, the key work being: 'Practical Parallel Rendering' (Chalmers et al. 2002). Nonetheless, there are many researchers actively investigating this field who are making their work available through specialist journals and via the internet. Two academic interest groups have formed around the topic: the 'Parallel Rendering Symposium' and the 'Euro-graphics Workshop on Parallel Graphics & Visualisation' (Chalmers et al. 1998, 20). These appear to be the current foci for activity in this field.

The first successful parallel renders made use of highly parallel, shared memory supercomputers. Indeed, their genesis stemmed from the goal of achieving real-time rendering (Slusallek and Wald 2001, 25). These systems were successful in generating VGA quality images at animation frame rates. However, their reliance on very expensive specialist hardware platforms restricts their wider usefulness.

More recently, similar results have been achieved on clusters of commodity PCs (Slusallek and Wald 2001). This has led to the establishment of the OpenRT project (Wald n.d.) which aims to define an API for ray tracing that is analogous to the rasterising API of OpenGL. Unfortunately, this is a closed source commercial endeavour and the proprietary nature of this interface may stifle its wider adoption. Recently an alternative open source project, called Manta, has been established (Bigler, Parker and Stephens 2006). This might prove to be a good common basis for ongoing research, although it is still too early to tell if it has succeeded in this respect.

So, within parallel renderers that have been developed, which methods have proved most successful? It should be noted that both domain and functional decompositions can be applied at different levels of granularity. In the case of domain decomposition, granularity refers to the size of the data chunks processed by individual tasks. Data partitioning has successfully been used at various levels of granularity in the parallel ray tracers investigated for this study. This should not be so surprising given the embarrassingly parallel nature of ray tracing. Furthermore, for shared memory systems where the scene data does not need any complex additional management, reports of near linear performance scaling are the norm (Chalmers et al. 1998, 190, 191 and 245). In one instance an unexpected super-linear performance was observed with the Kilauea renderer (Chalmers et al. 1998, 317). This seems to be the case for at least 20 processing units and for some of the parallel supercomputer systems this remains true for the total number of processors in the

system. There is not yet any consensus as to an optimal level of data granularity. Indeed, the lack of clear consensus suggests that the optimum is specific to the application data and not a fundamental of the algorithm.

Functional decomposition granularity is the extent of the algorithm encoded in the task. This is not as fluid as data granularity since the algorithmic structure of the problem places constraints on how the function may be divided. However, there are three broad levels of functional granularity that can be identified. Firstly there is instruction level parallelism. This is where individual CPU instructions are arranged for execution separately within a hardware MIMD, or MISD parallel system. On SIMD CPUs instructions can still perform certain tasks, such as array manipulations in parallel. Indeed, many SISD CPUs are now starting to add special instructions for performing SIMD operations, such as the Pentium SSE instructions (Intel 2000). This obviously requires re-writing the core algorithmic code and in the case of dedicated instructions, may require restructuring of the data so that it can be accessed in instruction ordered arrays.

There have been attempts to automate this process as part of compiler design, but this has proved to be an exceptionally complex task (Chalmers et al. 1998, 32), so at present, such optimisation is usually performed by hand. Despite this, several parallel renderers (Benthin, Dietrich et al. 2003; Bigler, Parker, and Stephens, 2006; Wald n.d.; Benthin, Slusallek et al. 2001, 156) include elements of instruction level parallelism and on PC cluster systems the SSE instructions find wide use in optimising vector arithmetic, allowing for the simultaneous processing of several (usually four) rays at once. This is usually referred to as a ray bundle. Unfortunately this does have the undesirable side effect of preventing the source code from being portable between different processor architectures.

It should also be mentioned that CISC processors are also starting to incorporate other forms of inherent instruction level parallelism, the benefits of which are gained without any special coding requirements. An example of this is the Pentium hyperthreading (Intel 2006b) facilities of Intel's latest chips.

The next level of functional decomposition is algorithmic decomposition, here as within structured programming, the process is rearranged into discrete, loosely coupled procedures which are then pipe-lined. This has not received much direct attention in ray tracing research. Perhaps there has been an implicit understanding that this would be less useful as the ratio of inter-task communication to computation performed would be very high for ray tracing. This is because the actual core code of the ray tracing algorithm is quite small, so there is not much to divide. The only example found was the Kilauea renderer (Chalmers et al. 2002, 249-327) which functionally decomposes shader execution fromother tasks such as acceleration grid parsing, or file access. This is not done for any stated performance gain, but to enable the creation of shaders in isolation from the complexities of the task scheduling mechanism (Chalmers et al. 2002, 282). Indeed, the ray tracing and shading tasks within a Kilauea process element execute strictly in serial, so if subsequent rays are required such as shadow rays, the shading task will stall until the

subordinate tracing tasks return. Parallelism is achieved by assigning multiple task stacks to a single processing element, thus ensuring continuous processor activity. It is unlikely that there is any actual performance benefit from this over a simpler data domain only approach.

The largest level of functional granularity is where the whole computational task executes concurrently with other application tasks. This is not directly applicable to the rendering subsystem, but is applicable to the host application. A good example would be the asynchronous screen update in a frameless rendering scenario (Slusallek and Wald 2001, 26). Here the code to update the screen can run as an independent process, reading the image buffer at intervals, while the ray tracer can run writing to the buffer without interruption or coordination. This type of mechanism is also useful in non-interactive rendering situations to provide visual feedback about the current task progress. finally it should be noted that the consensus view of researchers seems to be that domain decomposition offers far better performance gain than does algorithmic decomposition and this is certainly borne out by the amounts of research carried out in each area.

## *GIS Terrain Data*

The field of Geography has not been slow to appreciate the usefulness of computing resources. Many complex systems have been developed that display conventional cartography as 2D graphics with great sophistication. In deed, most of the worlds cartographic organisations now use software systems to produce their mapping products. Many also provide electronic products that often contain other geo-referenced data that could not be easily displayed in 2D cartography.

The representation of ground topology is one such product. In traditional mapping this would have to be represented as some form of special iconography, such as relief shading, gradient hatching, or contour lines. Using computers though, this information can be directly represented as a 3D render. Indeed additional information is being increasingly added to 3D plots and there is an increasing trend towards full realistic scene visualisation. This may be overlaid with details that may be real landscape features such as foliage and buildings, or other abstract information created by spatially referencing other classes of data.

There isn't sufficient space in this study to examine the development of digital cartography, but mention should be made of the particular data type that facilitates the use of 3D rendering: the digital elevation model. The original pioneers of DEMs were the US Geological Survey. They defined the original DEM file format for transfer of this type of geographic dataset. The USGS has a very public service orientated approach to their datasets and make them freely available over the internet (USGS 2004). Unfortunately the UK equivalent[4] from the Ordnance Survey (2007), is not so easily obtainable although in recent years they have been extending special licences for academic research and pan-governmental GIS usage. The actual data of a DEM consists of a 2D array of elevation values. These values are also accompanied by data about

[4] Land-Form PROFILE digital terrain model in NTF format.

the real-world location and intervals of the data points, sometimes referred to as the post spacing. To visualise this data the elevation values have to be converted into 3D points that are then used as the vertices of a mesh of triangles.

## *Terrain Rendering*

As for terrain visualisation research, most of it has been targeted at hardware rasterisation. Here optimisations are achieved by reducing the number of polygons that must be rendered by the graphics pipeline. This can be achieved either by simplifying the terrain tessellation, or by identifying sections of the geometry that are occluded and trimming these from the display list. Production systems often employ both mechanisms. On systems with good hardware acceleration, a terrain grid of two million triangles can be displayed at a rate of at least 30fps by using these methods. In reality, it is processing nothing like that number of triangles, perhaps ten to twenty thousand after simplification and culling.

In games programming, where much of this technology is employed, it is common practice to assess the actual 'polygon budget' (Wikipedia 1996) and design the game environments so that they stay within this size. There isn't sufficient scope to cover scan-line solutions beyond these cursory observations, however the Virtual Terrain Project (Discoe 2001) is an excellent resource for OpenGL based terrain rendering and a good portal for terrain visualisation in general. It has a rendering section where all of the major techniques are outlined and links to projects that employ them.

The ray tracing of height fields has also enjoyed some useful research, by virtue of landscape rendering being a very popular subject for the process[5]. Most of the early work was concerned with fidelity rather than speed, but now there is a good range of methods that provide efficient traversal of the terrain data (Musgrave 1988; Kaufman, Qu et al. 2003; Qu, Qiu et al. 2003; Henning and Stephenson 2004 and others). Since they all exploit some form of the spatial coherence of the height field's data, they all yield performance in logarithmic time complexity, thus showing excellent performance with increase in dataset size. In fact, Musgrave (1988, 9) noted, even in the 1980's, that the only significant limiting factor on terrain ray tracing, was available system memory.

In Chalmers (1998) review of parallel rendering, two instances of terrain rendering were reported. The first was by Parker who used a 256 CPU supercomputer to animate very large terrains in real-time. His findings not only support the linear scaling of performance, but demonstrate excellent sub-linear performance with scene size. In his tests increases in data size by a factor of 4 only decreased performance by 3% (Chalmers et al. 1998, 215). He also reports rendering scenes of 17 billion terrain quadrilaterals! The second use was by Slusallek to benchmark the RTRT renderer against various hardware rasterisers. He observed linear scaling and that for a 2 CPU system, the break-even scene size was at approximately one million triangles.

[5] As shown by the niche market for offline terrain scene renderers such as Vue d'Esprit (E-on Software 2007), Terragen (Planetside 1998) and MojoWorld (Pandromeda 1999). Pandromeda is actually a commercial outlet for Ken Musgrave's research activities.

It should be noted that ray tracing is not restricted to visualisation of planar polygons in the way that scan-line processes are. In fact, height field visualisation is actually an example of signal reconstruction and triangular tessellation is only one of a number of surface reconstruction strategies that may be applied. Henning and Stephenson (2004, 257) provide a useful summary of techniques commonly employed from voxel visualisation to fitting a bilinear patch. This is quite a complex issue and an area that would merit much further research as there is still room for improvement both in terms of performance and surface quality.

## *Atmospheric Shading Models*

For realistic landscape rendering purposes, atmospheric lighting phenomena are as important as terrain intersection. Humans rely on binocular vision to judge distance at close range, but this becomes ineffective once the parallax of a foreground object cannot be detected against its immediate background. At this point the brain entirely relies upon the subtle softening of contrast, the cooling or warming of tones and the intensity of shadows to make any judgement about distance and scale for a scene where the viewer has no prior local knowledge. This is a key psychological factor and its impact on scene perception should not be underestimated.

This perception was well understood by classical landscape painters and they developed a technique to model the phenomena which is called aerial perspective (Ebert, Musgrave et al. 1998, 361). In paintings, this is applied by the artist's judgement and observation, but for computer graphics, it is achieved by mathematically modelling the optical processes causing the effect. The fidelity of the model to the actual optical processes directly controls the visual effectiveness of the resulting image. Atmospheric models can vary quite considerably in complexity and precision. As with many other computer algorithms there is a dichotomy between accuracy and performance.

The simplest model is based upon the gross observation that objects grow fainter with distance. Indeed, this is a very common model and is supported in most rasterisation hardware. In OpenGL this rendering effect is known as fog (Davis, Heider et al. 1999, 242). It is quite common to find landscape rasterisers that use fog as a basic approximation for aerial perspective, or to simply hide the far clipping pane[6] (Nielsen 2003, 106). This fogging method is inadequate as an approximation for two main reasons: real aerial perspective varies logarithmically with altitude[7] and the fog colour is not constant but the sky colour in that direction (Nielsen 2003, 41). The latest generation of OpenGL hardware can now implement custom fogging by use of fragment shaders to perform these calculations, but it is still not commonplace and it does not integrate into the existing framework as elegantly as within a ray tracer.

[6] The OpenGL Z buffer can represent objects between two distances from the view point, referred to as the near and far clipping panes. Due to the finite range of the Z buffer depth values, the distance of the far clipping plane is constrained by the detail of the scene's foreground objects. If it has to be closer to the view point than the natural horizon, all terrain and objects beyond the far plane cannot be rendered. This causes a very gross rendering artefact where things appear or disappear depending on whether they have crossed the far clipping plane as the view point changes.

[7] As can be seen in the first image of Figure 10 where the top of Mt. Adams is clearer than the terrain at its feet.

Musgrave, Gritz and Worley (Ebert, Musgrave et al. 1998, 361-371) discuss the development of increasingly accurate, but computationally more expensive atmospheric models. In essence, aerial perspective, in fact the sky colour itself, are caused by the differential scattering of light at different wavelengths. Ultimate visual fidelity can only be achieved by careful analysis of all of the optical mechanisms at work in the atmosphereand performing volumetric integration on this model for the ray. Unfortunately this is too complex to be performed in real-time.

Nishita (Dobashi et al. 1996) proposed a thorough model for this process by isolating the contributing factors to both single and multiple scattering that occur due to atmospheric particles. Before it hits the atmosphere, sun light is effectively white. The more atmosphere it has to penetrate, the more light that gets scattered out of it's optical path. This is more accentuated for light waves at the blue end of the spectrum. This accounts for the sky background colour, as this is the scattered blue sun light. It also accounts for the reddening of the sun as it sets as it has to pass through a greater depth of atmosphere and thus loose more of its blue light. The physical processes involved are Rayleigh and Mie scattering. Rayleigh scattering accounts for the blue light scattering due to interaction with air molecules. Mie scattering is the effect of larger particles such as dust or pollution, which is responsible for visual haze and other discolourations.

This work has formed the core of most subsequent attempts at generating realistic skies. The typical solution involves some form of pre-computed sky-dome or 3D texture to store pre-computed atmospheric values, which are then interpolated at render time. Humphreys and Pharr provide a very thorough discussion of the practical implementation of a multiple scattering volume renderer with a concise summary of research in this field (2004, 803-817). Ertl, Falk and Schafhitzel (2007) havemanaged to perform real-time atmospheric shading including Mie and Rayleigh scattering as an OpenGL vertex shader. However, they did need to pre-compute a 3D scattering texture on a per-frame basis as they have also concluded that per-vertex scattering computations cannot yet be performed in real-time (Ertl, Falk and Schafhitzel 2007, 3).

## *Advanced Rendering Methods*

Lastly it is worth mentioning the most significant advances in 3D rendering since the establishment of ray tracing. Several methods have been developed that attempt to model the extremely complex problem of diffuse inter-reflections. In scenes with strong sources of direct lighting, the assumption of a constant ambient level may be acceptable, but in scenes that predominantly rely on indirect lighting, a constant ambient level creates unnaturally flat images. Duerer (2003) maintains an excellent resource called the 'Global Illumination Compendium'. This provides the key formulae for all of the advanced algorithms discussed here.

The Radiosity method was developed independently of ray tracing, and for some time was even regarded as a competing method. However, it is now normally employed in generating ambient lighting information, either for pre-calculated lighting for rasterisation[8], or as preprocessing step to conventional ray tracing, where the radiosity solution is sampled to supply the ambient term.

This method is simply the process of splitting all geometry into small patches, then for each patch in turn, the scene is rendered and the radiance received by the patch is recorded with the patch. This process is performed iteratively, so that patches illuminated in the prior cycle get to add their contribution to the next. Eventually, if sufficient cycles are performed, the scene reaches an equilibrium and a very accurate estimate of the diffuse inter-reflection is obtained. As may be imagined, this whole process is computationally, very expensive, from deciding how to subdivide the geometry into patches, to determining when to accept a solution as complete. There is considerable research around the parallisation of the radiosity method.

Path tracing, or forward ray tracing, is another attempt at directly modelling light diffusion. This works by tracing the actual path of all[9] light rays through a scene. All of their interactions with participating media, reflections and refractions are followed and, if the ray passes through the image plane, it is recorded. This is the reverse process of the Whitted method which back-tracks the rays required by the target image. If sufficient rays are generated by each light source, the results are very impressive. However, the render times are extremely long, even by comparison to radiosity.

There have been many attempts at speeding up path tracing, mostly by trying to predict useful ray paths, rather than generating millions of rays and only a small proportion of them usefully contributing to the final image. Monte Carlo integration techniques are commonly used to estimate surface irradiance with far fewer rays than purely stochastic path tracing (Kajiya 1986). Similaly there have been several techniques that try to identify key paths and explore these through the scene. This is often called bi-directional path tracing.

Photon mapping is another variant of path tracing developed by Henrik Jensen (2001) that traces the photons (rays with associated radiant energy) into the scene and records all surface interactions of the photons into a 3D data structure called the photon map. This map is generated as a pre-process step before a conventional ray tracing pass. At each surface intersection in the trace, the ambient term is determined by an estimation from statistically analysing the colour and strength of neighbouring photons in the photon map. The two key advantages of photon mapping are that it generally requires fewer photons to be traced than rays in stochastic path tracing and the photon map can be stored independently from the scene geometry which makes for much easier implementation. There are many variations and optimisations possible for photon mapping and this is an active field of research.

[8] This is very common in state-of-the-art computer games. The level models have a radiosity solution performed upon them and this is then used to shade the model at runtime. The visual effect is very impressive, but can be quickly flawed by any dynamic change in the scene not being reflected in the lighting.

[9] Obviously it is not practical to follow all light rays, so a statistically significant sample are used instead. Even so the number of ray paths that need to be traced is vast and only a small proportion actually contribute to the scene from the view point.

It is finally worth noting that all photo-realistic rendering methods areeffected by the limited dynamic range of display systems. A monitor is simply not capable ofcapturing all of the variations in brightness and contrast that could occur in a real world scene. In practice, these variations have to be compressed into the range that is available. This is similar to the exposure control of a camera selecting an optimum range for film to minimise under or over exposure.

## *Research Summary*

Initial work on computer graphics started in the 1960's and may of the key issues were identified then. However, it was not until the early 1980s that the first true ray tracing systems were developed. These generate computer images by mimicking the paths of light rays around a 3D scene, calculating the the optical effects they encounter along the way. The key theorists in this field were Phong and Whitted who between them developed the core of systems used today.

This work has been enhanced by many researchers to produce very high quality photo-realistic images. These techniques have been easily adapted to take advantage of hardware developments, especially in the field of parallel computing where ray tracing is an excellent example of a practical application for this technology.

Geographic systems have been developed that display conventional cartography as 2D graphics. Methods have evolved to represent 3D terrain data in mapping contexts and this is evolving into full 3D visualisation. To date, this visualisation has been achieved by other methods than ray tracing. However, the increasing size of GIS datasets and increases in the parallel processing power of modern computers are making ray tracing an increasingly attractive option.

Chapter 3

# Theoretical Framework

Having reviewed the published research concerning ray tracing and its parallisation, a framework needed to be established with which the experimental renderer could be implemented. The key theoretical models that are needed for this software system are described here. Note that to improve computational performance, it is preferable to avoid division operations as they are significantly slower than multiplications on most CPUs. For this reason, any division by a constant, in the following formulae, is replaced in the actual code by multiplication by the reciprocal.

## *Rendering Model*

The ray tracing engine developed for this study was aimed at producing a fast, yet well-featured, Whitted-style ray tracer using a modestly enhanced Phong illumination model. This approach has been adopted as something of a benchmark for ray tracers and was felt to provide a good balance between rendering time and image quality. The code has been structured to allow alternate rendering models to be integrated at a later stage, but this structure does not effect the core operation of the ray tracer and the details of its implementation will be discussed separately in the design methodology chapter.

### *The Local Illumination Equation*

The Phong illumination formula has been widely used in all fields of computer graphics and it is worth revisiting in detail further to understand its strengths and weaknesses. A Cornell box (Cornell University 1998) scene is used to illustrate the contributions provided by the separate parts of the equation. It is also useful to use a well known scene for comparison with other published rendering systems as often problems can be diagnosed visually rather than by inspecting code. As for the Phong equation[1] itself, it can be expressed as follows:

$$\mathbf{C} = \mathbf{k_a C_a} + \mathbf{k_e C_e} + \sum_{\mathbf{l}=1}^{\mathbf{n_l}} \mathbf{A_d} \left( \mathbf{k_d C_d} (\mathbf{L} \cdot \mathbf{N}) + \mathbf{k_s C_s} (\mathbf{V} \cdot \mathbf{R})^{\sigma} \right)$$

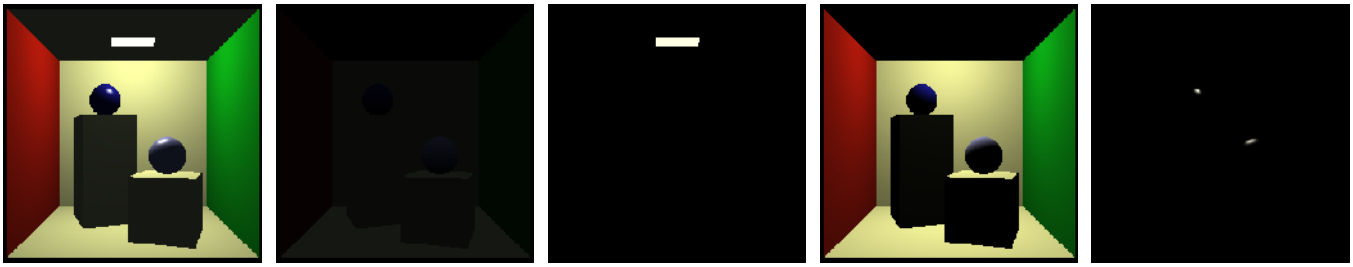
This may be understood as the ambient light level plus any emission from the surface and for each light source in the scene, terms for diffuse and specular reflection which are attenuated due to the light's distance $\mathbf{A_d}$. In this formula, all surfaces with a diffuse coefficient are assumed to be perfect Lambertian reflectors and so the reflection intensity is proportional to the cosine of the angle between the lighting direction **L** and

[1] The Phong illumination equation is quoted by numerous authors but often with subtle variations! The emission term is often omitted as is the attenuation term for the lights. Watt (1989, 52) also adds a view distance attenuation to the equation for visual cueing. It is also common to quote the modified Blinn-Phong equation which replaces **R.V** with **N.H**, where **H** is the halfway vector (Blinn 1977) but this is not strictly the Phong model. The form of the equation given here is the author's representation.

the surface normal **N**. This is calculated by taking the dot product of these two vectors[2]. The specular element is a convenient approximation for the observed highlights on glossy surfaces. It is not, however, an attempt to accurately model the real surface's micro-facet reflections. Again the use of a dot product provides a reasonably fast way of evaluating an angular quantity by avoiding costly trigonometric functions.

It should be noticed that by including the view direction vector **V**, this calculation is view dependent. The last vector term **R** is the direction of ideal reflection for the considered light source. The exponent in the formula $\sigma$ has no direct physical meaning, but controls the size of the highlight produced. Low values produce a dull, widely spread, highlight. Higher values, a smaller, brighter one.

The colour values $\mathbf{C}$, $\mathbf{C_a}$, $\mathbf{C_e}$, $\mathbf{C_d}$, $\mathbf{C_s}$ are all spectral quantities typically represented by n-tuples of discrete wavelength. In the case of an RGB based system such as this, they represent 3-tuple values. This equation must be evaluated on each of the colour channels in turn. Lastly the scalar constants $\mathbf{k_a}$, $\mathbf{k_e}$, $\mathbf{k_d}$, $\mathbf{k_s}$ are control values in the range of 0-1 that govern the relative proportions of the individual terms of the lighting equation. The Cornell Box scenes in Figure 2 are used illustrate the visual contribution of each of these terms to the final scene.



$$\mathbf{C} = \mathbf{k_a C_a} + \mathbf{k_e C_e} + \sum_{l=1}^{n_l} \mathbf{A_d k_d C_d (L \cdot N)} + \sum_{l=1}^{n_l} \mathbf{A_d k_s C_s (V \cdot R)}^{\sigma}$$

*Figure 2: The visual components of the Phong equation.*

The Phong equation is powerful and, as ray tracing computations go, relatively fast. It does however suffer from a few limitations. The first noticeable limitation is the completely flat nature of the ambient lighting. This is due to the ambient being determined by a constant value. Not only does this affect the believability of shadow regions, it also has the effect of reducing the dynamic range of the whole image.

There have been many suggestions as to how this can be improved. The rigorous methods involve some form of global illumination, but this is not viable for an interactive system. Ambient occlusion culling (Bülthoff and Langer 1999) would be useful for terrain scenes, but not for fully enclosed ones like the Cornell Box. Additionally, the requirement to extensively sample the occlusion level at all visible surface intersections makes this computationally expensive.

[2] Provided they are both normalised. This system maintains all vectors in normalised form.

Gooch et al. (1998) proposed a system for directional variation in the colour temperature of the ambient value from warm to cool as the surface normal diverges from the light direction to produce shadow detail. This works well for technical illustration images, but is not so suited to general scenes. It is also a per-light calculation which could become costly for complex scenes. However, the idea of varying the ambient light level by some simple function as a better approximation of the scene's diffuse inter-reflection is excellent and a simpler approach is implemented here. The ambient constant $\mathbf{k_a}$ can be optionally modulated by the vertical orientation of the surface. Since the surface normal's $Z$[3] coordinate is the cosine of this orientation, it may be directly employed to produce the ambient modulation $\mathbf{M_a}$:

$$\mathbf{M_a} = \mathbf{k_a}(1 + \mathbf{N_z})/2$$

Here the ambient level is at full intensity for surfaces with a zenithal normal and zero when antizenithal. This provides a better approximation to global illumination than a constant term and preserves the full dynamic range of the final rendered image. When this is substituted into the standard Phong equation, the modified rendering equation used by the experimental system becomes:

$$\mathbf{C} = \mathbf{k_a}\mathbf{C_a}(1 + \mathbf{N_z})/2 + \mathbf{k_e}\mathbf{C_e} + \sum_{\mathbf{l}=1}^{\mathbf{n_l}} \mathbf{A_d}(\mathbf{k_d}\mathbf{C_d}(\mathbf{L}\cdot\mathbf{N}) + \mathbf{k_s}\mathbf{C_s}(\mathbf{V}\cdot\mathbf{R})^{\sigma})$$

Figure 3 shows the visual difference between a constant ambient level and one using $\mathbf{N_z}$ modulation. The second image has a greater dynamic range than the first and the shape information is more discernible in shadow regions. From an interactive point of view, the main advantage of this method is that it re-uses a term that is already calculated by the system and so the computational impact of its use is minimal.

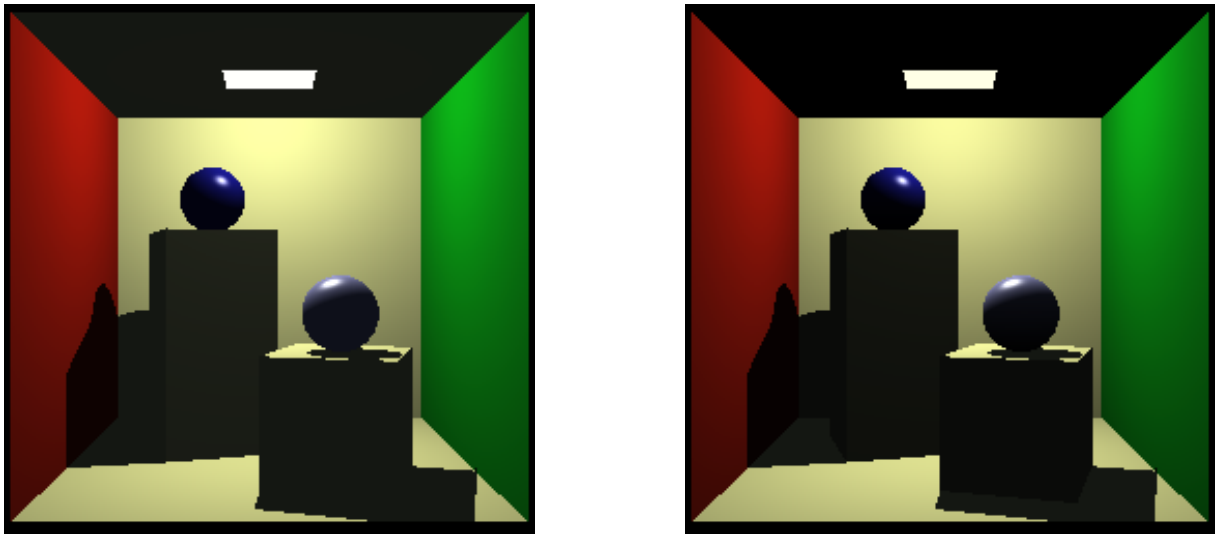


*Figure 3: Comparing constant and $N_z$ modulated ambient terms.*

[3] This system employs a right handed coordinate system with Z being the vertical axis (see Watt 1989, 2). It would be necessary to use $\mathbf{N_y}$ in other systems that use Y as the vertical axis.

The main limitation of Phong illumination can be seen by comparing Figure 2 with Figure 3: there are no shadows! Both renders in Figure 3 used recursive ray tracing to generate these shadows and this will be explained in the next section.

### *Ray Surface Interactions*

In a Whitted recursive ray tracer there are three processes that occur at each surface intersection: occlusion, reflection and refraction. Their results are combined to produce the final ray colour. The occlusion is calculated by a process of tracing a ray from the point of intersection to each light source in turn. If there are any intervening objects then the point lies in that light's shadow and it doesn't contribute to the result.

This is a trivial process for any light that may be approximated by a single point. Unfortunately, this is not true for most lights and their physical area must be considered. The most reliable method for rendering area lights is to generate multiple shadow rays which randomly sample a target point on the source.

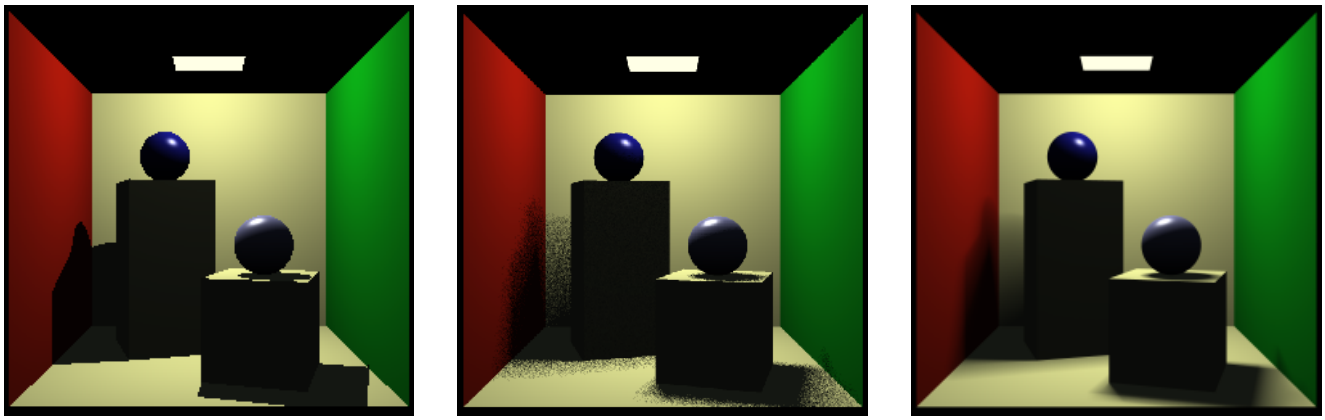


*Figure 4: Point source shadows compared with area shadows at 8 and 256 sample rays.*

Unfortunately, a large number of rays are required to avoid excessive noise in the final render and so it is not currently possible to employ this method in interactive rendering (see Figure 4). Fortunately, approximating the sun as a distant point light source, whilst generating rather harsh shadow margins, is not unreasonable for terrain visualisation and this is the technique used in the renderer.

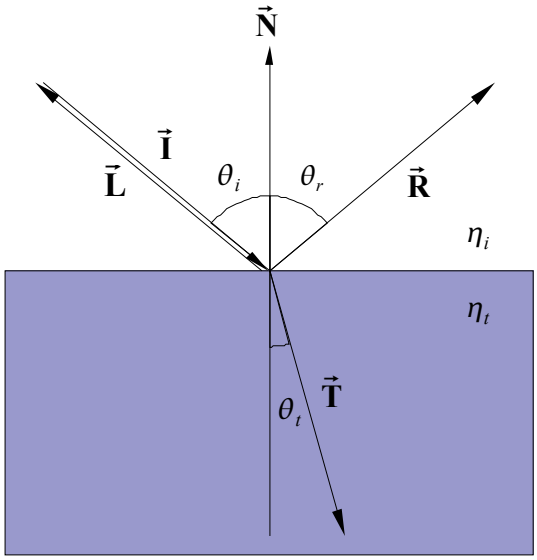



*Figure 5: Ray surface interaction diagram*

The most mathematically complex part of a ray tracer concerns the determination of ray paths for reflective and transmissive materials. The following section details the equations used. Where they are used unaltered from published texts, they are repeated with reference to their source. Where this system employs any modification to the source works, the derivation is shown.

The algebraic values illustrated in Figure 5 are defined as follows:

| | | | |
|---|---|---|---|
| $\vec{\mathbf{N}}$ | *The surface normal.* | $\theta_i$ | *The incidence angle.* |
| $\vec{\mathbf{I}}$ | *The incident light vector.* | $\theta_r$ | *The reflection angle.* |
| $\vec{\mathbf{L}}$ | *The light direction vector.* | $\theta_t$ | *The transmission (refraction) angle.* |
| $\vec{\mathbf{R}}$ | *The reflection direction vector.* | $\eta_i$ | *The incident medium refractive index.* |
| $\vec{\mathbf{T}}$ | *The transmission direction vector.* | $\eta_t$ | *The transmissive medium refractive index.* |
| | | $\eta$ | *The index of refraction.* |

Note that the vectors **L** and **I** are the negations of each other and it is important to insure the correct one is used in a given context. So algebraically:

$$\vec{\mathbf{L}} = -\vec{\mathbf{I}}$$

The following calculation (Watt 1989, 166) is used to determine the reflection vectors:

$$\vec{\mathbf{R}} = \vec{\mathbf{I}} + 2\,\vec{\mathbf{N}}\cos(\theta_i)$$

However $\theta_i$ need not be evaluated as:

$$\cos(\theta_i) = \vec{\mathbf{L}}.\vec{\mathbf{N}} \text{ or } \cos(\theta_i) = -\vec{\mathbf{I}}.\vec{\mathbf{N}}$$

Therefore the actual computation performed by the renderer:

$$\vec{\mathbf{R}} = \vec{\mathbf{I}} - 2(\vec{\mathbf{I}}.\vec{\mathbf{N}})\vec{\mathbf{N}}$$

The transmission vector is somewhat more complicated to determine. It requires an application of Snell's Law which defines the relationship between refractive indices and the angles of incidence and refraction. Snell's Law (Humphreys and Pharr 2004, 419) is taken as:

$$\eta_i \sin(\theta_i) = \eta_t \sin(\theta_t)$$

and the index of refraction is determined by:

$$\eta = \eta_t / \eta_i$$

Therefore:

$$\sin(\theta_t) = \eta_i / \eta_t \sin(\theta_i) = \sin(\theta_i) / \eta$$

Given the well known identity:

$$\sin(\theta)^2 = 1 - \cos(\theta)^2$$

Therefore:

$$\sin(\theta_t)^2 = \sin^2(\theta_i)/\eta^2$$

$$\cos(\theta_t)^2 = 1 - \sin^2(\theta_i)/\eta^2$$

$$\cos(\theta_t)^2 = 1 - (1 - \cos^2(\theta_i))/\eta^2$$

$$\cos(\theta_t)^2 = 1 - (1 - (-\vec{\mathbf{I}}.\vec{\mathbf{N}})^2)/\eta^2$$

$$\cos(\theta_t)^2 = 1 - (1 - (\vec{\mathbf{I}}.\vec{\mathbf{N}})^2)/\eta^2$$

$$\cos(\theta_t) = \sqrt{1 - (1 - (\vec{\mathbf{I}}.\vec{\mathbf{N}})^2)/\eta^2}$$

Since:

$$\cos(\theta_t) = -\vec{\mathbf{N}}.\vec{\mathbf{T}}$$

Therefore, the perpendicular component of $\vec{\mathbf{T}}$ is:

$$\vec{\mathbf{T}}_{\mathbf{perp}} = -\sqrt{1 - (1 - (\vec{\mathbf{I}}.\vec{\mathbf{N}})^2)/\eta^2}\,\vec{\mathbf{N}}$$

Since Snell's Law defines the ratio ofthe sines of the incidence and refraction angles and these are also the magnitudes of the lateral components of the direction vectors (Buss 2003, 240). This implies:

$$\vec{\mathbf{T}}_{\mathbf{lat}} = \frac{\eta_i \vec{\mathbf{I}}_{\mathbf{lat}}}{\eta_t}$$

Since $\vec{\mathbf{I}}_{\mathbf{lat}}$ (Buss 2003, 240) may be determined by:

$$\vec{\mathbf{I}}_{\mathbf{lat}} = \vec{\mathbf{I}} - (\vec{\mathbf{I}}.\vec{\mathbf{N}})\,\vec{\mathbf{N}}$$

Therefore:

$$\vec{\mathbf{T}}_{\mathbf{lat}} = 1/\eta(\vec{\mathbf{I}} - (\vec{\mathbf{I}}.\vec{\mathbf{N}})\,\vec{\mathbf{N}})$$

Finally this gives:

$$\vec{\mathbf{T}} = \vec{\mathbf{T}}_{\mathbf{lat}} + \vec{\mathbf{T}}_{\mathbf{perp}}$$

$$\vec{\mathbf{T}} = 1/\eta(\vec{\mathbf{I}} - (\vec{\mathbf{I}}.\vec{\mathbf{N}})\,\vec{\mathbf{N}}) - \sqrt{1 - (1 - (\vec{\mathbf{I}}.\vec{\mathbf{N}})^2)/\eta^2}\,\vec{\mathbf{N}}$$

As previously mentioned, it is desirable to avoid division in actual computational code, therefore the inverse index of refraction is used in this system and is defined as:

$$\zeta = 1/\eta = \eta_i/\eta_t$$

This then gives the final transmission equation used in the renderer as:

$$\vec{\mathbf{T}} = \zeta(\vec{\mathbf{I}} - (\vec{\mathbf{I}}.\vec{\mathbf{N}})\vec{\mathbf{N}}) - \sqrt{1 - \zeta^2(1 - (\vec{\mathbf{I}}.\vec{\mathbf{N}})^2)}\,\vec{\mathbf{N}}$$

As noted by Buss (2003,241) the expression under the square root must not be negative. Consequently, a test is performed to ensure:

$$\zeta^2(1 - (\vec{\mathbf{I}}.\vec{\mathbf{N}})^2) \leqslant 1$$

There is a physical significance to this threshold. If it is exceeded, then total internal reflection occurs. If this occurs in the renderer, a null transmission direction vector is returned and the reflection coefficient is set to one. Figure 6 shows a Cornell box where the spheres exhibit reflection and refraction.

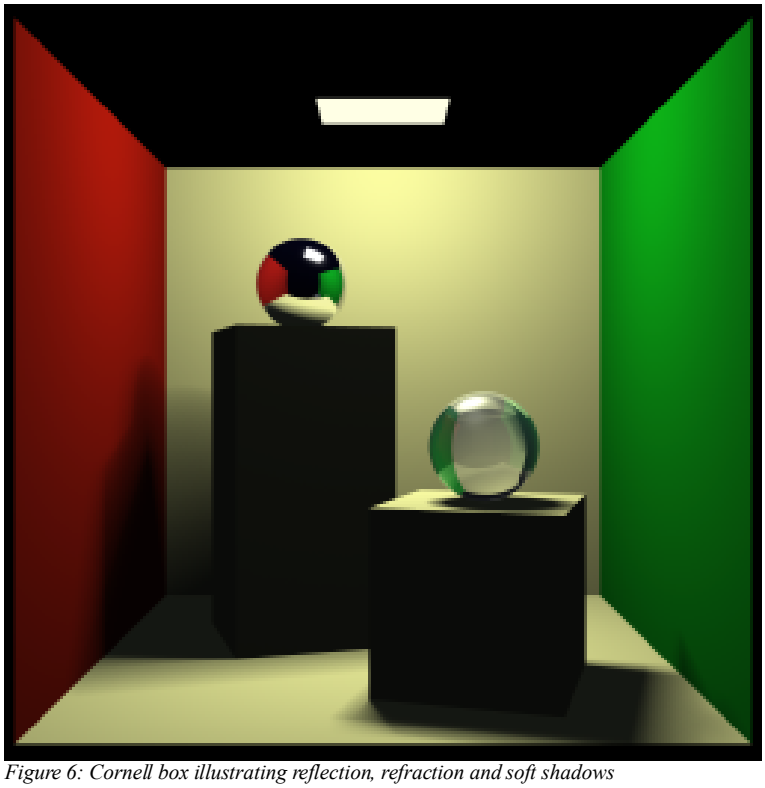


*Figure 6: Cornell box illustrating reflection, refraction and soft shadows*

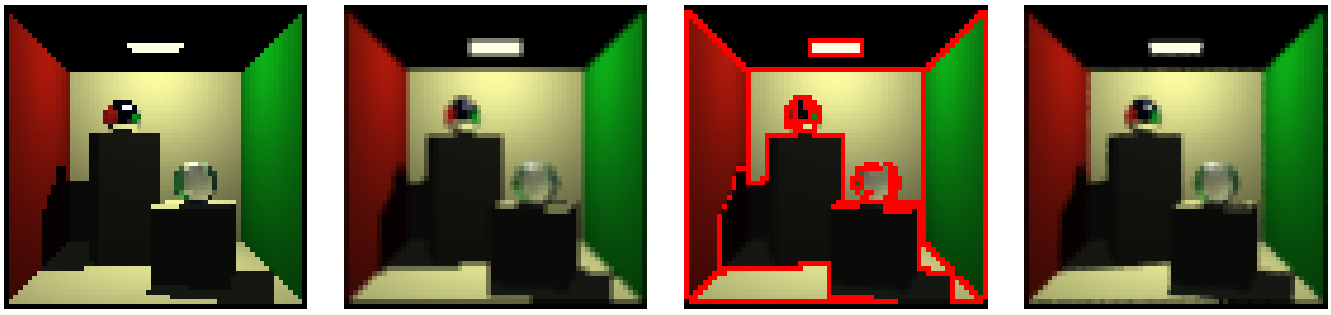
### *Anti-aliasing Techniques*

The quality of a ray traced image can be greatly improved by supersampling, a process of generating more than one ray per image pixel and averaging them. Figure 6 used 256 samples to generate a very high quality image. Contrast this with Figure 3 where no supersampling was performed at all. The key to effective supersampling is choosing a good distribution of a minimal number of sample rays to achieve acceptable image quality. The performance impact is directly proportional to the number of rays required.

Humphreys and Pharr (2004, 279-350) give extensive treatment to supersampling methods, most of which were implemented in the renderer. Unfortunately, due to the high computational cost of most methods, only vertex interpolation supersampling can be performed interactively.

This approach traces rays at the corners of each pixel. These corner values are then averaged to achieve a coarse form of supersampling. The performance advantage of this method comes from the fact that for an $\mathbf{N} \times \mathbf{M}$ image only $\mathbf{N} + \mathbf{M} - 1$ additional rays need to be cast. The main fault of this method is that it slightly blurs the final image.

For non-interactive rendering, adaptive supersampling was found to give the best time to quality ratio for most images. It works by performing an initial pass, similar to vertex interpolation, and where it finds a significant discontinuity between the values, it dynamically adds additional rays until a specified convergence is achieved. Figure 7 compares these supersampling methods, note that the red pixels in the adaptive diagnostic image indicate the pixels that are identified as requiring further sampling.



*No Supersampler* *Interpolated Vertex* *Adaptive Diagnostic* *Adaptive Supersampler*

*Figure 7: Small renders illustrating supersampling methods*

## *Water Rendering Model*

For dielectric materials (most translucent materials such as water and glass) both reflection and refraction occur and whilst terrain models can be treated as pure Lambertian diffuse reflectors, this is clearly not the case for water. Consequently, for a realistic simulation of large-scale water bodies, both phenomena must be employed. However, the relative contribution between the two of them is not constant. Water is almost entirely transparent when viewed from a perpendicular point of view, but from an oblique angle, the water surface becomes highly reflective.

This balance is determined by the Fresnel reflectance coefficient $\mathbf{f}$ and this is calculated by the formulae[4] taken from Humphreys and Pharr (2004, 419-420):

$$\mathbf{r}_{\mathbf{par}} = \frac{\eta_t \cos\theta_i - \eta_i \cos\theta_t}{\eta_t \cos\theta_i + \eta_i \cos\theta_t}$$

$$\mathbf{r}_{\mathbf{perp}} = \frac{\eta_i \cos\theta_i - \eta_t \cos\theta_t}{\eta_i \cos\theta_i + \eta_t \cos\theta_t}$$

$$\mathbf{f} = (\mathbf{r}_{\mathbf{par}}^2 + \mathbf{r}_{\mathbf{perp}}^2)/2$$

[4] These equations are actually a common simplification of Frenel's equations for unpolarised light.

Once the ray dependent Fresnel reflectance coefficient is found, it is combined with the surface material's reflection $\mathbf{r}_m$ and refraction $\mathbf{t}_m$ parameters to determine the final proportions of reflectance **r** and refractance **t** to be returned in the ray colour. Note that the system requires that the data observes:

$$\mathbf{r}_m + \mathbf{t}_m = 1$$

The final coefficients are determined by:

$$\mathbf{t} = \mathbf{t}_m(1 - \mathbf{f})$$

$$\mathbf{r} = \mathbf{r}_m + \mathbf{f}\,\mathbf{t}_m$$

There are two further requirements for effective water simulation. The first is to implement ray attenuation observing Beer's Law (Humphreys and Pharr 2004, 576). The second is to chaotically perturb the water surface to emulate ripples. Without these, water looks unnaturally clear and still. Beer's Law defines the transmittance $\boldsymbol{\tau}$ through a medium with a unit absorbence $\boldsymbol{\sigma}$ over a given distance **d** and is expressed as:

$$\tau = \mathbf{e}^{-\sigma \mathbf{d}}$$

This phenomenon is wavelength dependent and in an RGB system this must be applied per colour channel. To determine the resulting colour $\mathbf{C}_{rgb}$ given the material surface colour $\mathbf{S}_{rgb}$, the deep water colour $\mathbf{W}_{rgb}$, the absorbency coefficient $\boldsymbol{\Sigma}_{rgb}$ and the ray length **l,** the following three expressions are employed:

$$\mathbf{C}_r = \mathbf{S}_r \mathbf{W}_r (1 - \mathbf{e}^{-\Sigma_r \mathbf{l}}) \qquad \mathbf{C}_g = \mathbf{S}_g \mathbf{W}_g (1 - \mathbf{e}^{-\Sigma_g \mathbf{l}}) \qquad \mathbf{C}_b = \mathbf{S}_b \mathbf{W}_b (1 - \mathbf{e}^{-\Sigma_b \mathbf{l}})$$

The surface perturbation ripples, are achieved by adding small random values, taken from a standard homogeneous noise function (Perlin 1997), and applying them to the surface normal's X and Y components[5] and renormalizing the vector. Figure 8 is a high resolution anamorphic aspect ratio image that demonstrates all of the features of the renderer's water shader.

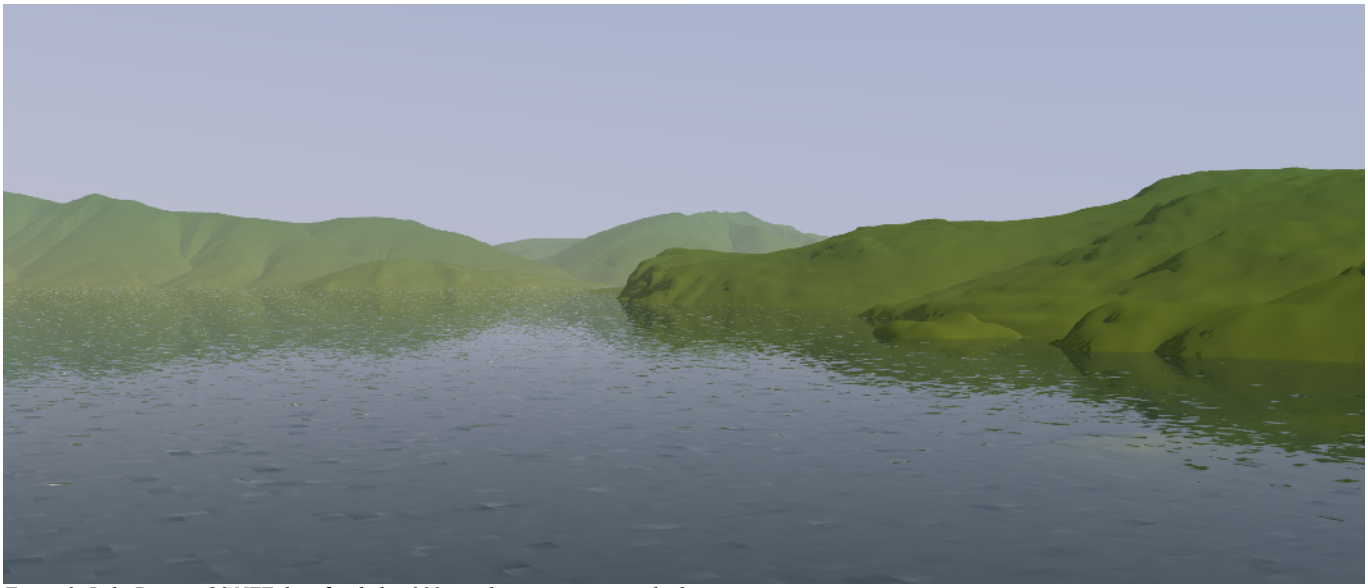


*Figure 8: Lake District OS NTF data flooded to 200m to demonstrate water shader*

[5] This process is only applicable to a surface such as water that is always a plane perpendicular to the cardinal Z axis.

## *Atmospheric Attenuation Model*

The key feature of ray tracing for large scale terrain visualisation, that really sets it apart from scan-line rasterisation, is the ability to efficiently simulate atmospheric absorption and scattering. Unfortunately, ray attenuation in air is much more complex that a simple treatment with Beer's Law and has to be dealt with by a completely separate process from the ray attenuation performed in the water model.

Given the aim of producing an interactive renderer, it was immediately apparent that any form of volume tracing was going to be far too computationally expensive. This would be desirable for high quality image production, but was beyond the scope of this project. So after reviewing the non-volumetric options one common feature was noticed: it was very common to find the sky colour for a given scene stored in a sky dome data structure and then, at render time, ray sky colours were interpolated from this structure.

The experimental renderer employed an altitude and azimuthal structured dome for ease of access. This does have the inefficiency of clustering too many points near the zenith. In practice, it is unusual to look straight up and so this region of the sky dome is rarely considered. It would probably be more efficient to store the data in a geodesic dome based structure, as this would evenly distribute the points over the surface. However this was not verified experimentally The sky colours of this dome are populated from empiric RGB[6] data statically compiled into the renderer. This is then spline interpolated from the sky dome vertices for the ray's direction vectors.

A second hemispherical data structure with its pole inclined to align with the solar position is also blended with the sky colour in an attempt to create solar glare. In practice, this had very little impact upon the render due to the renderer's lack of high dynamic range handling. This will be discussed later in the findings.

The sky dome was used to directly colour the background in the scene, but the remaining issue was how to implement aerial perspective without too high an impact on the render times. White linear fogging was implemented as a control test and so that its performance could be compared with OpenGL scenes performing the same operation.

The first actual sky model attempted was to adapt linear fog to take it's attenuating colour from the sky dome sampled in the ray direction in a similar manner to Nielsen (2003, 41) and perform linear blending over the scenes meteorological range. This worked quite well for short ranges, but had the unfortunate side effect of making everything beyond the meteorological range disappear. Musgrave describes this model as a homogeneous and isotropic GADD (Ebert, Musgrave et al. 1998, 363). He points out that the fault with it is that the real atmosphere does not have homogeneous density, but that the optical density of the atmosphere decreases exponentially with altitude above sea level.

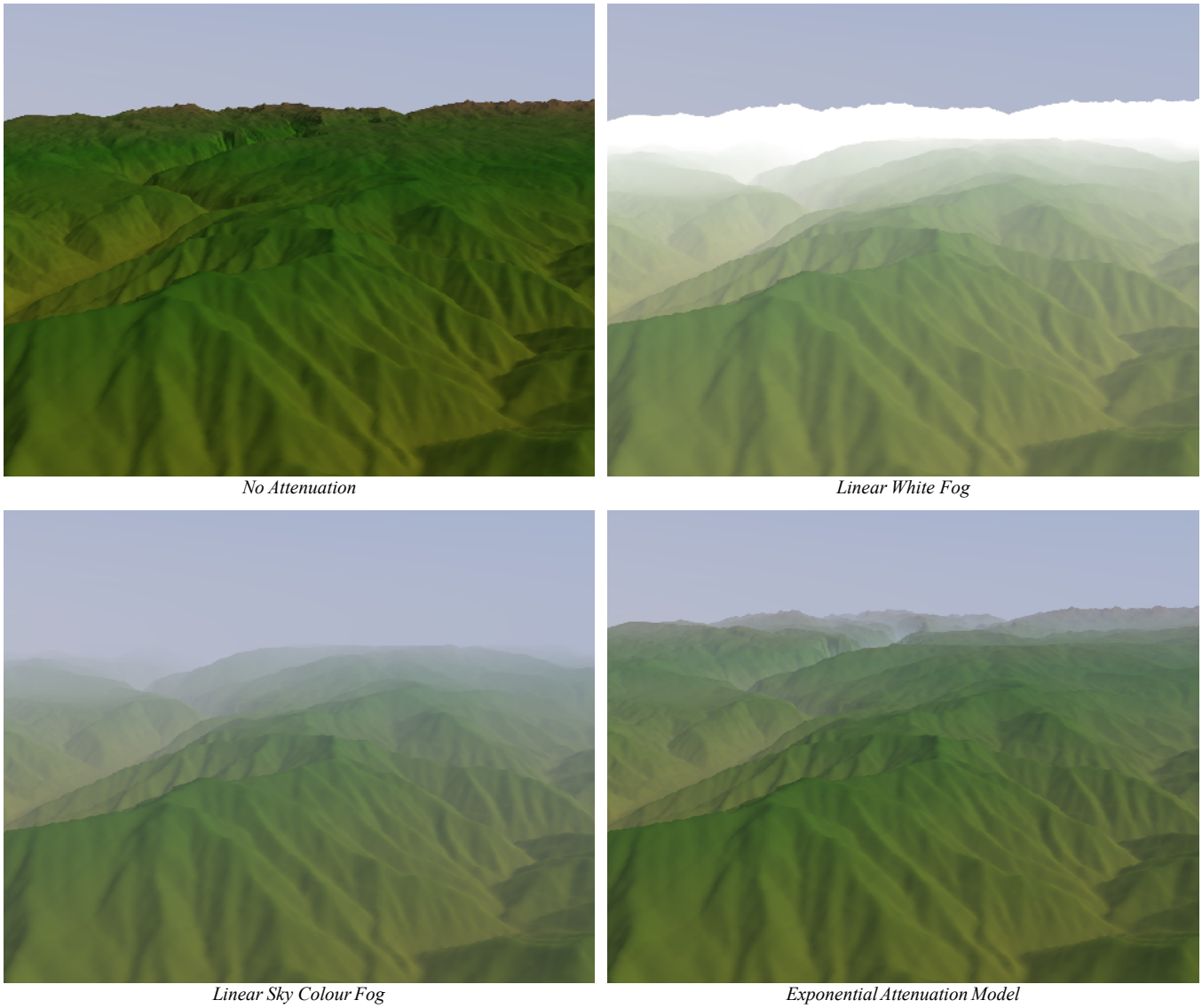
[6] Values were taken from high quality digital photographs of clear skys.

Musgrave (Ebert, Musgrave et al. 1998, 364-371) discusses progressively more complex GADDs, but in order to maintain interactivity, further enhancement was limited to simply adding the logarithmic optical density. This was handled using a formula derived from Nielsen's (2003, 75) approximation:

$$\tau = e^{-(h_v + h_t)/2s}$$

This equation models the logarithmic integral for calculating the optical depth $\tau$ where $h_v$ is the elevation of the view point $h_t$ the elevation of the target and $s$ the atmospheric scale height for the prevailing conditions.

The images of Figure 9 illustrate the progressive development of the atmospheric model used in the experimental renderer. Note the weakness of linear fogging that colours all terrain beyond the meteorological range white. A similar flaw exists with the linear sky model that causes the terrain to disappear.



*No Attenuation* *Linear White Fog*

*Linear Sky Colour Fog* *Exponential Attenuation Model*

*Figure 9: Yosemite USGS DEM renders comparing atmospheric models*

The final cost of employing the exponential altitude attenuation model was only one tenth of the overall rendering time as many of the arithmetic values requiredare already calculated as part of the ray tracing process. It has the visual property of local fidelity to nature, but it is not without faults.

The model actually assumes that the world is flat. This is due to there being no discrete integral fora spherical logarithmic atmospheric model. Such a system would require a numerical solution, which would be much slower to compute. The problem occurs with infinite horizons always appearing indistinct. This is visible if there is a sea plane in the image. As the ray intercept tends towards the infinite horizon they receive exceptional levels of attenuation. In reality, the horizon is at a finite distance and should be distinct on a clear day.

Figure 10 demonstrates a scene exhibiting the atmospheric modelsimulating dawn conditions over the Lake District. The solar altitude angle is 15°. The atmosphere has a meteorological range of 10km and a scale height of 1000m which is similar to low-lying water vapour.

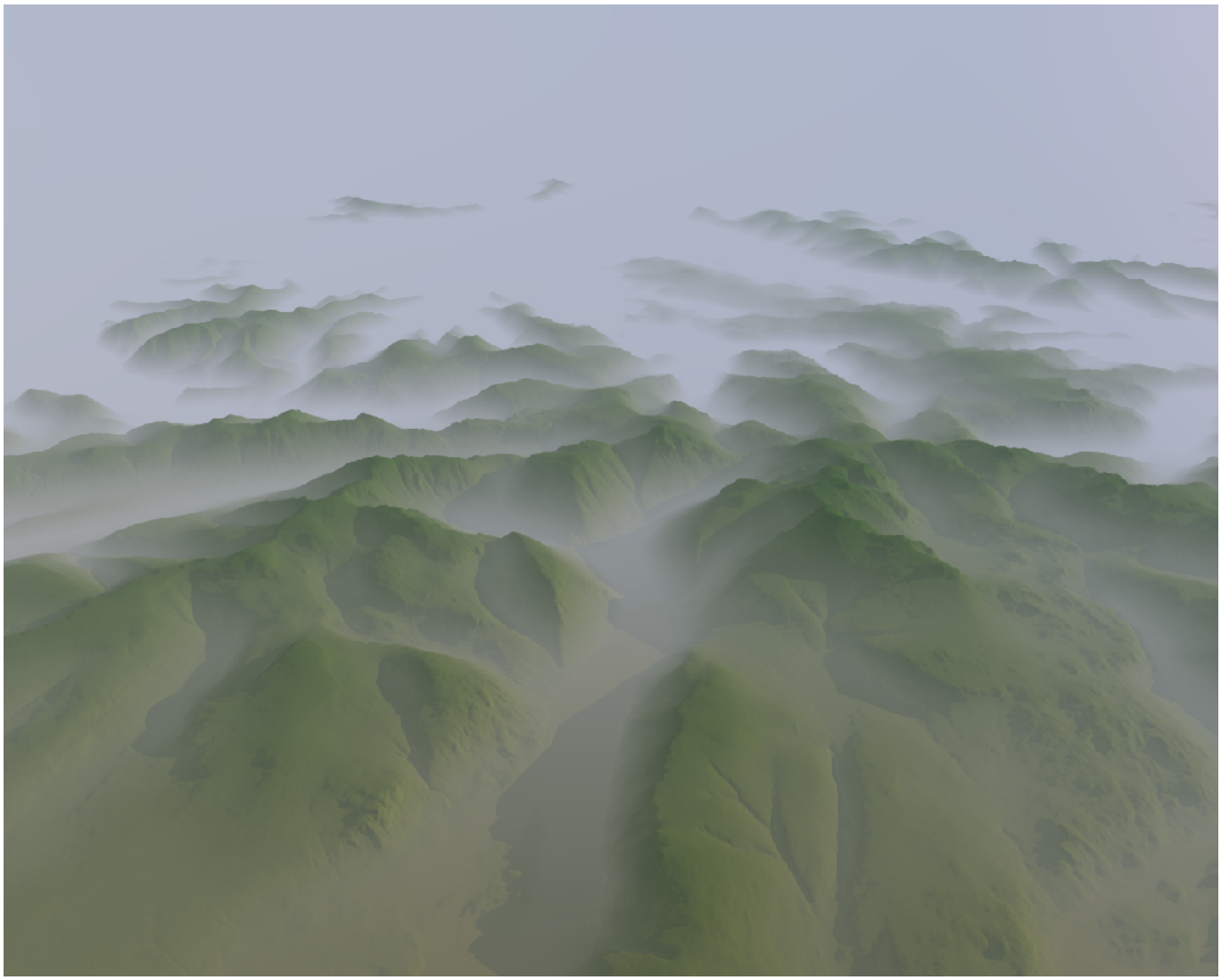


*Figure 10: Sunrise over the Lake District from OS NTF data*

## Terrain Grid Tracing Model

The terrain traversal method adopted was based upon the space partitioning work by Amanatides and Woo (1987) for tracking the ray path over the terrain grid. This was combined with a very efficient surface intersection test described by Kaufman, Qu et al. (2003).

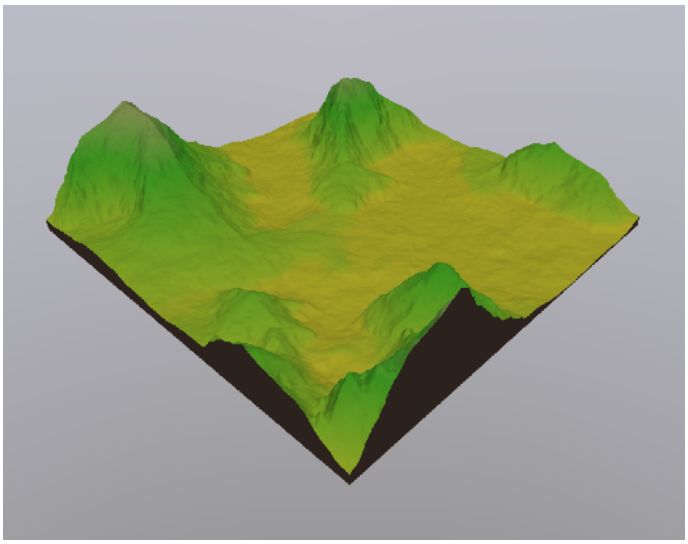


*Figure 11: Terrain tracing showing solid grid edge*

They also employed a multi-resolution, level of detail scheme to reduce the number of intersection tests required. This was not implemented here since it requires a significant pre-processing stage which would have to be performed per-frame unless the data being rendered could be guaranteed to remain constant. Also the multi-resolution maps for large terrain datasets would consume significant quantities of main memory.

The terrain intersection test was also simplified from the method of Kaufman, Qu et al. (2003) since their approach allowed the terrain to be rendered from the underside. This is not required for terrain rendering and an edge intersection below the surface level was deemed to hit the side of a solid wall (Figure 11) and so only hits form above needed testing for.

## Parallelism Model

One of the key academic goals of this project was to evaluate different models of parallelism for ray tracing. In order to be able to make base-line comparisons, the actual render module was designed to be invoked by a task manager in several ways. It could be executed by cooperative multitasking within the main program thread. It could also be invoked, pre-emptively, as a separate process. A third mode enabled multiple instances of the renderer to be executed as data-parallel tasks, operating on different scan-line blocks of a shared image buffer. This approach had the merit of ensuring that the exact same sequence of instructions was always being compared within the different parallelism models.

By virtue of using operating system threads, the same binary was execute on both single core CPUs using threaded concurrency and on multiple core CPUs with true parallel execution. In fact it was possible to utilise both concurrent and parallel execution on multiple cores, as the number of threads need not match the number of cores. The practical implications of this are discussed further in the findings. This software architecture is represented diagrammatically in Figure 12. This design enables investigation of the a priori domain decomposition, as well as the gross functional decomposition of separating the renderer from the host application.

The actual data partitioning applied for domain decomposition was an equal division of image scan-lines between the tasks. This had the merit of being very simple to implement, although as Chalmers discusses there are issues with tasks being unbalanced (Chalmers et al. 1998, 36). In the Cornell box test scenes, the sphere with refractive properties was noticeably more expensive per ray than other parts of the scene. For landscape scenes, large areas of water and rays grazing the horizon were also computational hot spots! Cost prediction algorithms where considered, but in practice, it was found that increasing the number of tasks to a level that ensured difficult portions of the image were spread between the tasks, was quite sufficient to keep both cores of a dual core CPU active. Given that the aim is to produce an interactive system, introducing any time intensive pre-processing was considered to be a poor design choice.

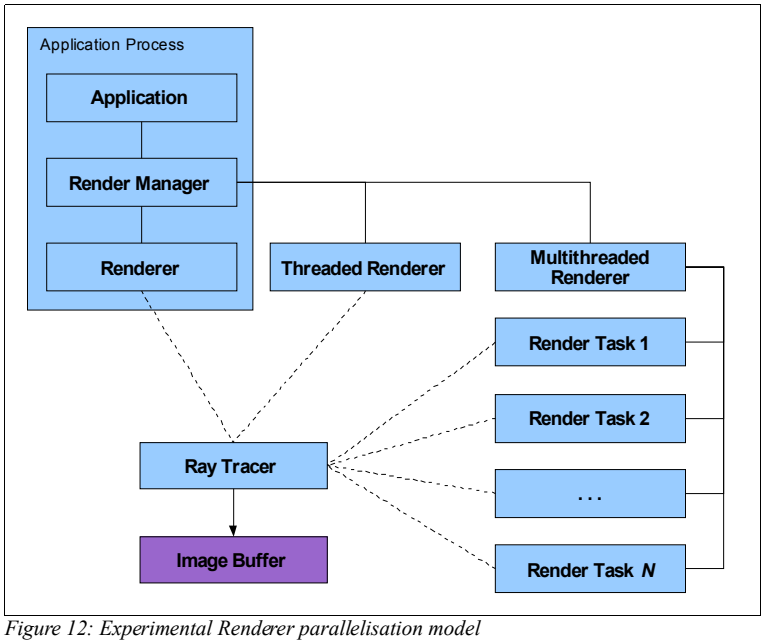


*Figure 12: Experimental Renderer parallelisation model*

Algorithmic functional decomposition was also investigated. The first method considered was splitting the ray path determination from surface shading. This split exists in the existing subroutine structure and was an obvious candidate to explore. However, when the rendering process was profiled, typically 90% or more of the actual task was spent testing for object intersections and not a balance between the two. Consequently, there would be no real performance gain from this decomposition as the overwhelming requirement was to split the scene database search, which in reality is a domain decomposition. Since this can already be efficiently achieved by executing multiple render tasks, there was no merit in pursuing this algorithmically.

A further finer grained decomposition was investigated out of academic interest, even though it would suffer from the same practical weakness as the previous method. This algorithmic decomposition was suggested by closer examination of the Phong equation, which could be conceptually expressed as:

$$\mathbf{C} = \mathbf{F}_a(\mathbf{i}) + \mathbf{F}_e(\mathbf{i}) + \mathbf{F}_d(\mathbf{i}) + \mathbf{F}_s(\mathbf{i})$$

Which is a series of functions that determine the ambient $\mathbf{F}_a$, emissive $\mathbf{F}_e$, diffuse $\mathbf{F}_d$ and specular $\mathbf{F}_s$ components for a given surface intercept $\mathbf{i}$. If the whole Whitted function is considered, then occlusion $\mathbf{F}_o$, reflection $\mathbf{F}_r$ and refraction $\mathbf{F}_t$ must also be added. The algorithmic decomposition then becomes:

$$\mathbf{C} = \mathbf{F}_a(\mathbf{i}) + \mathbf{F}_e(\mathbf{i}) + \sum_{\mathbf{l}=1}^{\mathbf{n}_l} \mathbf{F}_o(\mathbf{i}, \mathbf{l})(\mathbf{F}_d(\mathbf{i}, \mathbf{l}) + \mathbf{F}_s(\mathbf{i}, \mathbf{l})) + \mathbf{F}_r(\mathbf{i}) + \mathbf{F}_t(\mathbf{i})$$

The actual code of the ray tracer was restructured to follow this conceptual scheme, but in so doing a further weakness of this approach became apparent. Once the individual functions were created they each consisted of only a few lines of code and yet the shading process per ray, had expanded from a single function call per surface interaction, to four plus an additional three per light source! Even before threaded communication code was introduced, the communication over-head was considerable. Add to this the potential for pipeline stalls once running in parallel and the fact that at best it could only account for 10% of the total execution time, meant it was not worth pursuing as a means of enhancing performance.

It should be noted though that this process did have two beneficial side effects. Firstly the shading algorithmic decomposition enabled renders to be produced that displayed individual terms of the rendering equation (see Figure 2) and this had didactic value in conceptualising the behaviour of the equation. Secondly it enabled several coding errors to be uncovered that would have been much harder to notice buried inside a monolithic shading function.

On reflection, it should not be too surprising that algorithmic decomposition has limited value in performance tuning ray tracing, since it is a process that has undergone decades of careful honing for fast execution on serial processors. Algorithmic decompositions are known to be much harder to find on processes that have highly optimised algorithms. The historic goals of creating fast code with few branches and tight inner loops, is at variance with the process of isolating individual code functions and loosely coupling them as this inherently adds additional layers of communication to the whole process.

Chapter 4

# Methodology

There were many practical considerations to the actual process of creating the experimental renderer that need to be mentioned. This section will examine these along with a description of the methods for testing the performance attained by this experimental system.

## *Development Environment*

Before any coding could begin the choice of target operating system and development tools needed to be addressed. One increasingly clear concern is the problem of investing significant amounts of time into developing code that becomes locked into a particular operating system or chip architecture. A further important consideration is using a computer language that can achieve optimal performance from the underlying hardware. Given the performance related goals of this project, failing to generate good quality code might adversely influence the results.

Previous experience with native code compilers for C and Pascal, suggested they would be more suited than Java, C#, or other popular semi-compilers, even though the latter usually offer a much more comfortable development environment and are generally more productive in terms of code written per day. To avoid vendor lock-in, open source compilers were preferable, especially ones that are capable of producing portable code.

The author has had considerable experience of commercial programming in Delphi (CodeGear 2007) and was inclined to make direct use of this experience. However, Delphi has been struggling over the last few years to maintain it's status in commercial development and has been eclipsed by both C# and Java. Also Delphi makes no pretence to being cross-platform. Indeed its visual component library is a thin object orientated layer over the Windows API and the compiler can only produce Windows executables.

Given the meteoric rise of Linux in the professional computing world and its growing acceptance in wider user circles, it was felt that the ability to target this platform was important, not to mention the additional desire to keep the renderer code platform independent. Fortunately, there is an open source project called Free Pascal (Free Pascal Team 1993) that aims to produce a Delphi-like (but not strictly compatible) cross-platform compiler. This compiler has been showing considerable promise for several years. It is already targeting a good range of CPU architectures and has production quality support for Linux and Windows both for 32 and 64 bit Intel processors.

It's main limitation was that, until very recently, it did not have a professional quality IDE. However, just as this project was started, a promising development platform called Lazarus (Lazarus Team 2003) was reaching late Beta stage. The decision was taken to take the risk of using this, with the fall-back plan of reverting to using Delphi on Windows, if the code quality produced by Free Pascal/Lazarus proved to be too unreliable. This has proved a fortunate choice as both projects have gone from strength to strength over the intervening two years. It now boasts full visual development in an editor not unlike Delphi version 6; it can handle graphical debugging; visual user interface creation and trivial project recompilation on any supported platform to native code. Most impressive of all, Free Pascal[1] is capable of optimising, compiling and linking the 76,000 lines of the renderer program in under 6 seconds!

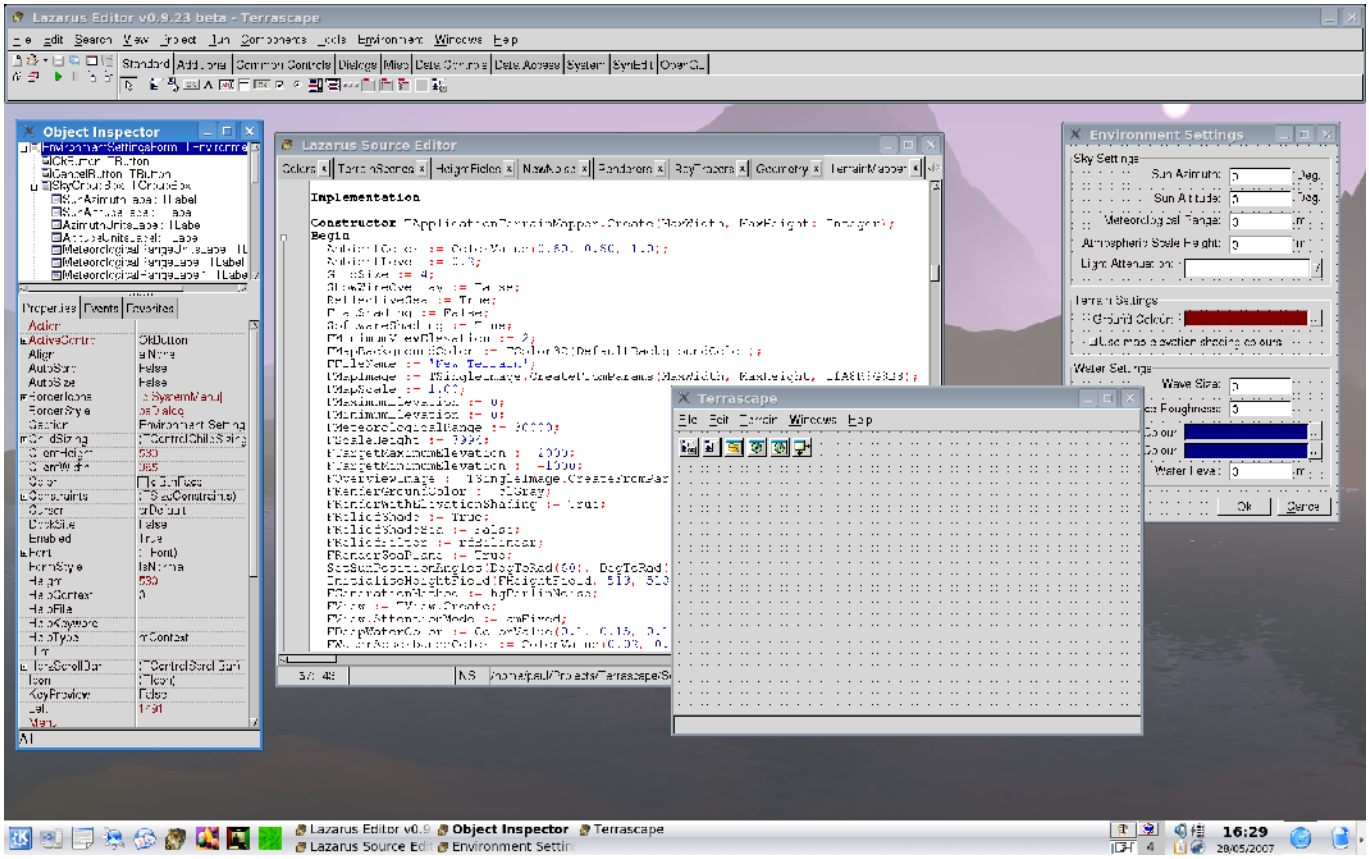



*Figure 13: Lazarus IDE running on Kubuntu desktop*

Choosing a Linux distribution for general desktop use as well as Lazarus based development was far from trivial and in fact, six different distributions were used through out the life time of this project. All have suffered from one limitation or another. Space does not permit a complete account of the issues involved, but the eventual winner was Kubuntu (Canonical 2006). This has a modern user interface, reliable software installation tools and is very stable in operation. This was installed in a dual boot configuration with Windows XP Media Edition so that direct comparisons could be made between executables created for either operating system.

[1] The version being used for Linux development for this project is 2.0.4.

## Parallelism Tools

When writing code for parallel execution with a compiler designed for creating serial applications, additional libraries are required to create multiple processes and coordinate their communication. The two most popular libraries for providing these facilities are PVM (PVM n.d.) and MPI (MPI 2007). PVM is the older and currently less supported of the two. Pascal translations of their header files are available for both systems and they can be used with Free Pascal.

On closer inspection of the renderer's requirements, it was realised that very little coordination would be needed between the processes and where it was required, critical sections could be used. Free Pascal has very thorough support for native OS threading, so it was decided that this would be used instead. In fact Free Pascal provides a higher level object orientated wrapper for the thread handling. This was directly used as the parent class for the task manager and the rendering task objects.

This proved to be a very efficient approach that had minimal inter-process communication overhead. It also removed the need for dependency upon an external library. The only limitation was that the thread library does not support remote threads and so there is no immediate scaling to network clustering. There is another Free Pascal library that encapsulates the TCP communication protocol. This could be combined with the threading unit to provide this facility and it is hoped that this will be implemented in future developments to this system.

## Development Process

Careful design was an important feature of this project, as was rigorous testing at each stage of progression. Low level code was developed using classic structured programming methods. Code modules were closely monitored to preserve loose coupling and to remove any unnecessary dependencies. Particular emphasis was placed upon achieving good platform independence. High level code took advantage of Free Pascal's object orientated extensions and implemented all conceptual entities as code objects.

Cross-platform compatibility was tested by the simple practice of daily alternating between Linux and Windows as the development platform. Lazarus itself was also undergoing considerable change throughout this period and it was updated daily using the Subversion (CollabNet 2006) code revision management system maintained by the Lazarus team.

The three main sources of complexity, the renderer, GIS data handling and parallel control code, were all developed and tested autonomously and then combined to produce the final test program. The renderer in particular, was first developed in isolation. It employed the simple Cornell Box test scene to verify its rendering fidelity by comparing the output with the official renders published on the Cornell University

Website (Cornell 1998). This scene's simplicity was also useful for visual inspection of the reflection and refraction code output. Whilst this was a slow process, it did ensure a good degree of code maturity in the base units of the final application.

## *Testing Procedure and Performance Analysis*

Benchmarking a computer process is a notoriously difficult activity! Both hardware and software configurations can have a dramatic impact upon the operation of any program. During code optimisation low level subroutines were profiled by counting CPU cycles. This is a useful relative measure of how efficiently a given section of code performs. However, this does not translate to overall performance, especially in a system executing parallel or even concurrent processes.

In the final analysis, the performance that is important is the performance as perceived by the application user. This has to account for interaction with the operating system and other processes, as this is how it would be used in practice. For this reason the benchmarking mechanism used was a simple stopwatch type algorithm using the system timer. While not accurate to the same precision as the clock frequency of the CPU it can still attain accuracy within hundreds of a second. External variabilities were reduced by having no other applications running during testing and switching off all network connections. The renderer also had a special operating mode which can switch off screen updates during frame rendering as this was found to be a very considerable source of variability. Lastly for any given test, the result recorded was an average of ten renders.

One key objective is to assess whether terrain ray tracing can now be performed at interactive frame rates. To make this assessment a definition for what constitutes interactivity must be established. Various dictionary definitions merely imply providing an immediate response. What is considered immediate is a psychological perception that varies from one person to another. Another related metric is the notion of real-time rendering. This is subtly different as it implies capability to provide smooth animation. Again what is smooth?

To answer this question it is worth looking at the standards employed in other fields. Film and video typically require 24fps although early films were hand cranked at 16fps (Davis et al. 1999, 21). Modern 3D computer games often strive for higher frame rates than this, although anything higher than the monitor refresh rate[2] is pointless as it cannot be noticed by the user. Early computer games refreshed the display at between 10-15fps. CAD 3D visualisation usually requires render times to be considerably less than a second for sufficient feedback to allow interactive changes in view point. So it would seem from these observations of usage that real-time rendering requires over 10fps and over 20fps to be considered of a standard suitable for modern animation. Interactive rendering is therefore less than real-time but over a psychological threshold in the region of a quarter of a second.

[2] Typically 50-120MHz.

Chapter 5

# Findings

Three tests were performed with the renderer. The first examined the optimal number of threads for a range of different CPUs. The second examined the relationship between resolution and rendering time and the last investigated the relationship between terrain data size and rendering time. The scene used for the first two of these tests was the fractal landscape shown in Figure 11.

## *Thread Configuration Analysis*

This analysis involved running the multi-threaded renderer on several systems with very different CPUs. The number of threads used ranged from 1 to 10 and then in 10's from 20 to 100. Each measurement was taken ten times and the average recorded. The output was at VGA resolution and the render included a water plane and shadows. Figure 14 displays the results.

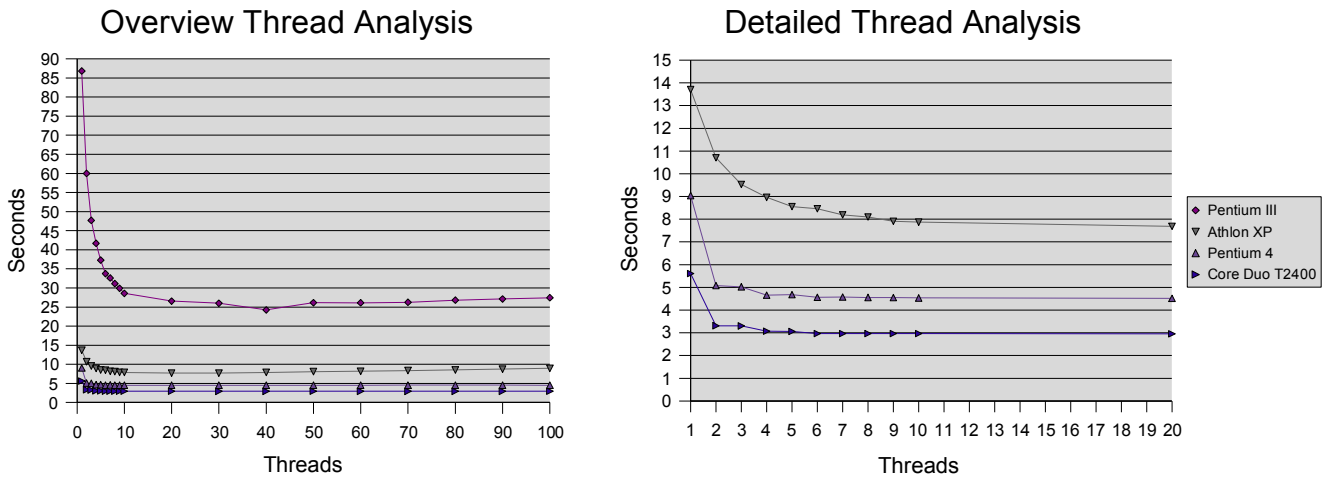



*Figure 14: Thread performance analysis graphs*

As expected the two newest systems produced the fastest results. It is particularly interesting that the lower speed Core Duo (1.83GHz) significantly out performed the much faster Pentium 4 (3.2GHz). This must be due both to the dual cores executing in parallel and also refinements in the chip design, since even with a single thread, it was still considerably faster. It is interesting to note that the performance curve of the Pentium 4 matches the shape of the Core Duo's very closely. This must be due to the hyper-threading abilities of the former which do clearly add significant processing power to additional threads. More surprising was the performance graph of the old Pentium III. This has no hyper-threading, but still achieved a large performance gain when utilising multiple threads.

It was observed that the graphs become almost flat after 30-50 threads on all CPU's. It would seem that the overhead for using threads is minimal and a large number does not cause much of a performance penalty. Consequently it would be best to always err on the side of too many rather than too few threads.

## Resolution Analysis

This analysis investigates the relationship between ray tracing and render size. A range of common screen resolutions from quarter VGA up to high definition television were selected and average render times plotted against their pixel counts. Four datasets are recorded. The first is for basic grid tracing only. The second includes shadow detection and the next also includes atmospheric modelling. The last model introduces a water plane into the scene as well.

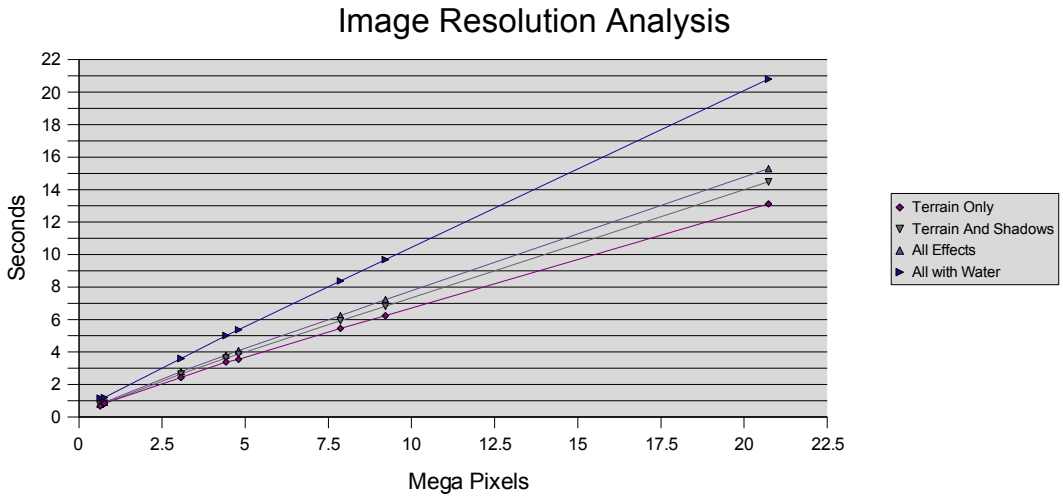



*Figure 15: Image resolution analysis graph*

Theses results corroborate very closely, the theoretical linear relationship between image size and ray tracing execution time. It should also be noted that the addition of the more advanced rendering features only has a proportionate impact on render times.

## Terrain Size Analysis

For this analysis, a range of synthetic terrain files were rendered at VGA resolution, both by ray tracing and by OpenGL[1]. The results were plotted against the number of samples in the terrain.

These results show the linear nature of OpenGL with scene size, contrasted with the logarithmic nature of ray tracing. The key result to note in this test is the crossover point between the datasets at around 1M quads. This suggests, that even limited to current systems, very large-scale model rendering is already better suited by ray tracing.

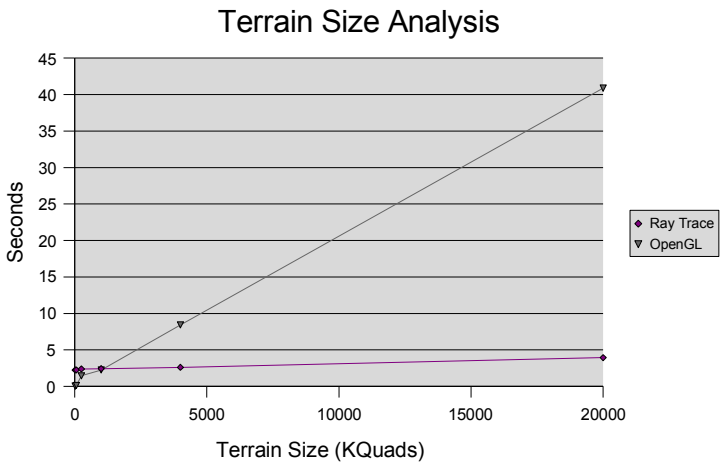



*Figure 16: Terrain size analysis graph*

[1] The OpenGL was hardware accelerated by an Nvidia GeForce Go 7800.

### *Other Observations*

All of the formal experiments were made using the multi-threaded renderer. It is worth mentioning that this, even with a single thread, out-performed the non-threaded renderer on all of the systems tested. This is perhaps an indication of the extent to which the OS interacts with the main loop of an application. Therefore any processor intensive task would probably benefit from being functionally decomposed from the host application in this manner.

Despite careful steps being taken to reduce background tasks, there was significant variation in the execution times of all individual tests. This is simply a fact of life on a modern OS and so when considering performance requirements for production systems, a suitable margin of error should be allowed for this contention. Not only are fluctuations caused by OS activity, some are caused by cache misses and uneven load balancing between threads. Cache use in a parallel system is very unpredictable and for large scenes there is probably not very much that can be done about this. Uneven loads, can to some extent, be ameliorated by running more threads with little penalty, given their relatively low overhead as established in the first analysis.

Chapter 6

# Conclusion

There are may conclusions that may be drawn from a study such as this, which has touched upon the research of several disciplines, in the hope of combining them for use in another. Some are reflections upon the work undertaken, others insights into future directions that might be explored.

## *Research Reflections*

The first consideration has to be was the experimental renderer adequate for the purpose it was designed? In the main, the answer has to be yes. It enabled the use of classic ray tracing methods to produce images of at least comparable quality to hardware rasterisation. It did so in conjunction with fast terrain grid tracing and thread based parallelism.

Nonetheless, there are still things that could be improved or added: There are still occasional rendering artefacts that are caused by problems with numeric precision in the ray tracing code; The choice of planar atmospheric model inhibits sharp horizons and not handling high dynamic range prevents the sky glow around the sun from being as bright as it should. Additionally, there is still scope for generally improving the quality of the code and further developing the algorithms that have been employed.

With regard to the analysis of the system. It would have been useful to have spent more time on developing tools to automatically perform benchmarking tasks, since running multiple tests by hand proved very labour intensive. With hindsight, it would also have been interesting to test a far wider range of rendering situations, especially to try and quantify the worst case conditions, such as rays that graze the horizon over very large models.

Given the goal of interactivity: has this been achieved? Well, sub-second render times are available for WQVGA and QVGA resolutions. This can feel stilted when the times start approaching one second, but it is generally usable for dynamic previewing for a high resolution image. It should be noted that HDTV quality images can be subsequently produced in well under 30 seconds for virtually any terrain size on the faster machines tested here.

Unfortunately in a commercial product, there are few users who would be content with such low image resolutions and a more realistic goal would be to aim for VGA or DVD quality. To achieve this, there would need to be a fourfold increase in performance. To reach this the program would need further enhancements. There is still scope for optimisation within the code, most notably by hand tuning the grid tracing routine as this accounts for 80% of execution time. Also the use of instruction level functional decomposition and further exploiting the spatial coherence of terrain models would yield significant improvements too.

Between these methods the current render times could be halved. If quad-core CPUs give the same performance improvement that dual-core chips do, then that could account for the rest of the power required.

It should be borne in mind, that for very large terrains, OpenGL isn't capable of interactively rendering the scene either. In fact it is more likely that ray tracing will achieve this before rasterisation does for the reasons previously discussed. Indeed this research closely agrees with Slusallek and Wald (2001, 29) who found that their software ray tracer running on a single CPU outperforms the best hardware rasterization engines for scenes with a complexity of roughly 1 million triangles. This proves to be very similar to the threshold in this system where a dataset size of 1024x1024 is the break-even.

## *Further Work*

Since data partitioning is embarrassingly parallel on shared memory systems (Slusallek and Wald 2001, 30) we should be able to expect linear, or near linear performance on multi-core CPU's. Quad core CPUs have only just become generally available at the time of writing and it is unfortunate that it was not possible to include one in this test. It would be the obvious first step of furthering this project, to see if they do indeed impart a further linear performance gain over dual core chips.

Related to this would be to explore the extension of the system into clustering. This is where multiple machines on a network could all contribute to the image creation by acting as rendering nodes. If each machine rendered one tenth of a VGA image, 10 machines could render the scene in one tenth of the time, plus any communication overhead. Even allowing for very poor communication this would complete in around half a second. It would be very interesting to see how far this approach could be perused before practical constraints came to the fore.

For use in a production role, there are several issues that would need to be addressed. For robustness it would be useful for tasks to able to communicate difficulties back to the task manger. For instance, time intensive image portions could be re-divide if execution time is becoming excessive. In heterogeneous environments, CPU's might not be capable of handling an equal share of the task, or they may be forced to terminate their task prematurely. Users would also require inclusion of additional GIS data for overlay on the terrain, for example: conventional 2D mapping, aerial photographs; or visualisation of 3D surface features such as buildings or vegetation. Arbitrary water bodies would also be required such as rivers and lakes. The current method of creating a universal water level would be insufficient for these purposes.

Considerable work is still required on developing the atmospheric model. Build a volumetric model to assess its computational requirements would be another fruitful avenue for extending this work into high quality rendering. It would also enable the renderer to produce local atmospheric phenomenon such as mist and clouds. Nielsen's (2003) produced an interesting study where he introduced a simplification of the

scattering functions for OpenGL vertex shaders, they might be suitable for adapting to ray tracing. It would also be interesting to compare the predicted atmospheric conditions to real observations on the ground and tune the model accordingly.

Given the goal of interactivity, it would be worth investigating dynamic resolution rendering. This would work by detecting situations where the user has triggered some adjustment to the image. The immediate response could then be provided by a first low resolution pass that could be displayed for direct feedback. If the interaction is continuous, then further low resolution frames could be generated until the user has achieved the desired static view. At which point the full quality render could proceed and be display as soon as it is complete. This would be a way of working around the immediate limits on interactivity for higher resolutions.

Achieving some form of intelligent load balancing would be desirable, rather than simply relying on generating more threads than processors in the hope of keeping them all occupied. It would be worth analysing if the maximum ray path length within the terrain grid's bounding volume might serve as a reasonable metric for computational load.

The key to optimising any ray tracer is in designing the best possible spatial partitioning. In the case of terrain rendering this is the grid tracing algorithm. The routine developed here, although fast, could still be improved by looking for ways to take advantage of terrain coherence.

The actual surface reconstruction could benefit from from a level of detail scheme. This would both enhance quality and improve performance. Near to the view point, it can be seen that certain terrain features drop below the nyquist limit of the terrain grid's data frequency. This is especially noticeable for arettes, or narrow gullies which take on a false saw tooth appearance. Fitting some form of spline patch might well yield a much more elegant surface in these contentious situations. Spline patches have been optimised for real-time use, as demonstrated by Benthin, Slusallek and Wald (2004). Similar has also been effectively demonstrated in the OpenRT for iso-surfaces (Marmitt, Kleer et al. 2004).

At present all quadrangles are subdivided into two triangles for the grid tracing and thus two intercepts are needed. A cursory analysis of surface curvature (Figure 17) for the Lake District data, indicates that less than a tenth of the quadrangles are actually non-

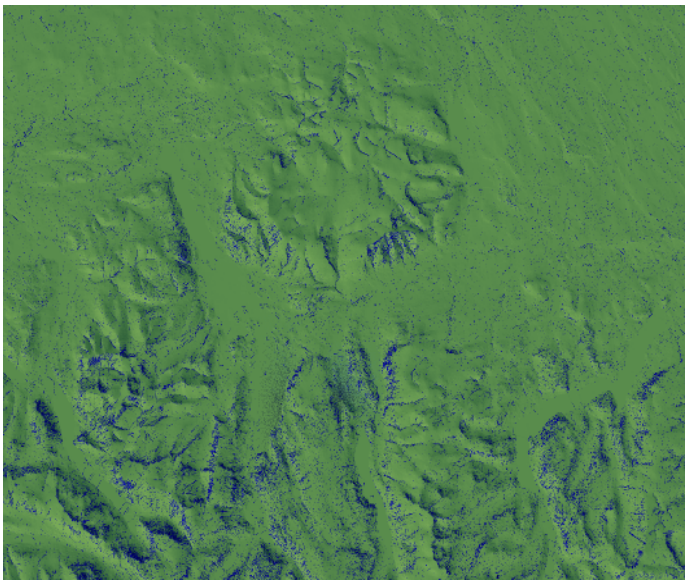


*Figure 17: Non-planar quadrangles in Lake District data*

planar[1]. This fact and the arithmetic simplicity of testing for a non-planar quadrangle would suggest an optimisation scheme which would remove the need two perform two triangle test in the majority of ray intercepts. Furthermore, if the quadrangle is sufficiently far away that it is less than an image pixel in size, a simple maximum height test can be used to determine intersection, rather than a full polygonal intersection test. This would be especially useful with larger terrain models.

One last avenue to explore might be to see if converting the renderer to C or C++ would yield any significant performance gain? It would be a very good test candidate to answer the question as to which produces the better code. It would also increase its portability by being written in C even if it were found to be no faster.

## *Final Summary*

This study aimed to quantify the performance improvement that could be obtained by applying parallelism to a basic ray tracing program running on typical consumer systems. By showing that such a program could just attain interactivity, albeit at low resolution, indicates that this process holds great potential for future software refinement and gain advantage from future hardware development. It is also hoped that these results might encourage the investigation of ray tracing's uses in other spheres of computer graphics, than those to which it has traditionally been restricted. It has been shown that it is already the fastest method for very large scenes and so it can be expected to be applied more and more widely as user dataset size continues to place pressure on conventional rasterising systems. Lastly, there is still much to be investigated if new uses are to be found for this challenging field of computer science, as Chalmers (Chalmers et al. 1998, 20) and his colleagues summarise:

> *"We believe that parallel photo-realistic graphics will continue to be a vibrant research topic in the future."*

[1] Assuming a 5% tolerance in the maximum elevation delta compared to the diagonal of the quadrangle. Quadrangles failing this tolerance are coloured blue on the relief map (Figure 17).

# References


Amanatides, J. and Woo, A. (1987) *A Fast Voxel Traversal Algorithm for Ray Tracing,* Eurographics '87, pp. 3-10

Allen, M. and Wilkinson, B. (2005) *Parallel Programming 2nd Ed.*, Pearson Educationc

Appel, A. (1968) *Some Techniques for Shading Machine Renderings of Solids*, AFIPS Spring Joint Computer Conf. pp. 37-45

Benthin, C., Slusallek, P. et al. (2001) *Interactive Rendering with Coherent Ray-Tracing*, in Computer Graphics Forum / Proceedings of the EUROGRAPHICS 2001, pp. 153-164

Benthin, C., Slusallek, P and Wald, I. (2004) *Interactive Ray Tracing of Free-Form Surfaces,* Proceedings of Afrigraph 2004, Stellenbosch (Cape Town), South Africa, November 3-5

Bigler, J., Parker, S. G. and Stephens, A. (2006) *Design for Parallel Interactive Ray Tracing Systems,* Proceedings of the IEEE Symposium on Interactive Ray Tracing, (to appear) [online] http://www.sci.utah.edu/%7Eabe/rt06/manta-rt06.pdf

Blinn, J. F. *(1977) Models of Light Reflection for Computer Synthesized Pictures*, SIGGRAPH 77, pp 192-198

Bülthoff, H. H. and Langer, M. S. (1999) *Perception of shape from shading on a cloudy day*. Technical Report No. 73, Max-Planck-Institut für biologische Kybernetik

Canonical Ltd. (2006) Kubuntu Desktop Linux [online] http://www.kubuntu.org/

Carpenter, L., Cook, R. and Porter, T. (1984) *Distributed raytracing*. SIGGRAPH, 18(3), pp. 137-145

Chalmers, A., Davis, T., Reinhard, E. Eds. (2002) *Practical Parallel Rendering*, A. K. Peters

Chalmers, A. G., Jansen, F. W. and Reinhard, E. (1998) *Overview of Parallel Photo-realistic Graphics*, Eurographics [Online] www.cs.bris.ac.uk/Publications/Papers/1000271.pdf

CodeGear (2007) Delphi 2007 for Win32 [online] http://www.codegear.com/products/delphi/win32

Cook, R. L. and Torrance, K. E. (1982) *A Reflectance Model for Computer Graphics*, ACM Transactions on Graphics, Vol. 1, pp. 7-24

Cornell University (1998) The Cornell Box [online] http://www.graphics.cornell.edu/online/box/

Davis, T., Heider, J. et al. (1999) *OpenGL Programming Guide 3rd Ed.*, Addison-Wesley

Discoe, B. (2001) *Terrain LOD: Runtime Regular-Grid Algorithms*, Virtual Terrain Project [online] http://www.vterrain.org/LOD/Papers/

Dobashi, Y., Kaneda, K., Nishita, T. and Yamashita, H. (1996) *Display method of the sky color taking into account multiple scattering. In Proceedings of Pacific Graphic*, pages 117–132

Duerer, A. (2003) *Global Illumination Compendium* [online] http://www.cs.kuleuven.ac.be/~phil/GI/TotalCompendium.pdf

E-on Software Inc. (2007) *Vue d'Esprit* [online] http://www.e-onsoftware.com/

Ebert, D. S., Kenton Musgrave, F. et al. (1998) *Texturing & Modeling A Procedural Approach 2nd Ed.*, AP Professional

Ertl, T., Falk, M. and Schafhitzel, T. (2007) *Real-Time Rendering of Planets with Atmospheres*, Journal of WSCG 07

Fox, G., Messina, P. and Williams, R. (1994). *Parallel computing works.* Morgan Kaufmann

Free Pascal Team (1993) Free Pascal [online] http://www.freepascal.org/

Gooch, A., Gooch, B., et al. (1998) *A non-photorealistic lighting model for automatic technical illustration.*In SIGGRAPH 98 Conference Proceedings, pages 447–452

Guibas, L. J. and Veach, E. (1997) *Metropolis Light Transport,* Computer Graphics, 31 (Annual Conference Series), pp.65–76

Günther, J., Slusallek, P. and Wald, I. (2004) *Realtime Caustics Using Distributed Photon Mapping*, Saarland University [online] http://graphics.cs.uni-sb.de/Publications/2004/RTPM_EGSR2004.pdf

Haines, E. Ed. (2004) *Ray Tracing News* [online] http://jedi.ks.uiuc.edu/~johns/raytracer/rtn/

Henning, C. and Stephenson, P. (2004) *Accelerating the Ray Tracing of Height Fields*, Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp. 254-258

Intel Corp. (2000) *Getting Started with SSE/SSE2 for the Intel Pentium 4 Processor* [online] http://cache-www.intel.com/cd/00/00/01/77/17741_getting_started.pdf

Intel Corp. (2006a) *Moore's Law* [online] http://www.intel.com/technology/mooreslaw/

Intel Corp. (2006b) *Hyper-Threading Technology* [online] http://www.intel.com/technology/hyperthread/

Jensen, H. W. (2001) *Realistic Image Synthesis Using Photon Mapping*, A. K. Peters

Kajiya, J. T. (1986) *The rendering equation*, ACM SIGGRAPH, pp. 143-150

Kaufman, A., Qu, H. et al. (2003) *Ray Tracing Height Fields*, Computer Graphics International, pp. 202-209

Lazarus Team (2003) Lazarus IDE [online] http://www.lazarus.freepascal.org/

Marmitt, G., Kleer, A., et al. (2004) *Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing*, Proceedings of Vision, Modeling, and Visualization (VMV) 2004, Stanford (CA), USA, November pp. 16-18Phong, B. T. (1973) *Illumination of Computer-Generated Images*, Department of Computer Science, University of Utah, UTEC-CSs-73-129.

MPI (n.d.) Message Passing Interface Forum [online] http://www.mpi-forum.org/

Musgrave, F. K. (1988) *Grid tracing: Fast ray tracing for height fields*, Technical Report YALEU/DCS/RR-639, Yale University, Dept. of Computer Science Research

Nielsen, R. S. (2003) *Real Time Rendering of Atmospheric Scattering Effects for Flight Simulators* [online] http://www.imm.dtu.dk/pubdb/views/edoc_download.php/2554/pdf/imm2554.pdf

Ordnance Survey (2007) Land-Form PROFILE [online] http://www.ordnancesurvey.co.uk/oswebsite/products/landformprofile/techinfo.html

Pandromeda Inc. (1999) *MojoWorld* [online] http://www.pandromeda.com/products/

Perlin, K. (1997) Original Noise Function [online] http://mrl.nyu.edu/~perlin/doc/oscar.html#noise

Planetside (1998) *Terragen* [online] http://www.planetside.co.uk/terragen/

PVM (2007) Parallel Virtual Machine [online] http://www.csm.ornl.gov/pvm/

Shirley, P (2000) *Realistic Ray Tracing*, A. K. Peters

Slusallek, P. and Wald, I. (2001) *State of the Art in Interactive Ray Tracing*, State of the Art Reports, EUROGRAPHICS 2001, pp. 21-42

USGS (2004) USGS Geographic Data Download [online] http://edc2.usgs.gov/geodata/index.php

Wald, I. (n.d.) *The Open Ray Tracer*, University of Saarbruecken [online] http://www.openrt.de/

Watt, A. (1989) *Fundamentals of Three-Dimensional Computer Graphics*, Addison Wesley Publishing

Wikipedia (1996) Low Poly [online] http://en.wikipedia.org/wiki/Low_poly

Whitted, T. (1980) *An Improved Illumination Model for Shaded Display*, Comm. ACM, Vol. 23, No. 6, pp. 343-349

# Bibliography

## *Ray Tracing, Rendering and Geometry*


Amanatides, J. and Woo, A. (1987) *A Fast Voxel Traversal Algorithm for Ray Tracing,* Eurographics '87, pp. 3-10

Angel, E. (2000) *Interactive Computer Graphics 2nd Ed.*, Addison-Wesley

Appel, A. (1968) *Some Techniques for Shading Machine Renderings of Solids*, AFIPS Spring Joint Computer Conf. pp. 37-45

Bala, K., Bekaert, P., Durtré, P. (2003) *Advanced Global Illumination*, A. K. Peters

Benthin, C., Dietrich, A. et al. (2003) *Interactive Distributed Ray Tracing on Commodity PC Clusters - State of the Art and Practical Applications*, Lecture Notes on Computer Science 2790 (Proceedings of EuroPar), pp. 499-508

Benthin, C., Purcell, T., Schmittler, J., Slusallek, P. and Wald, I. (2003) *Realtime Ray Tracing and its use for Interactive Global Illumination*, Eurographics State of the Art Reports

Benthin, C., Slusallek, P. et al. (2001) *Interactive Rendering with Coherent Ray-Tracing*, in Computer Graphics Forum / Proceedings of the EUROGRAPHICS 2001, pp. 153-164

Benthin, C., Slusallek, P and Wald, I. (2004) *Interactive Ray Tracing of Free-Form Surfaces*, Proceedings of Afrigraph 2004, Stellenbosch (Cape Town), South Africa, November 3-5

de Berg, M., van Kreveld, M., et al. (2000) *Computational Geometry 2nd Ed.*, Springer

Bigler, J., Parker, S. G. and Stephens, A. (2006) *Design for Parallel Interactive Ray Tracing Systems,* Proceedings of the IEEE Symposium on Interactive Ray Tracing, (to appear) [online] http://www.sci.utah.edu/%7Eabe/rt06/manta-rt06.pdf

Blinn, J. F. *(1977) Models of Light Reflection for Computer Synthesized Pictures*, SIGGRAPH 77, pp. 192-198

Bowyer, A. and Woodwark, J. R. (1983) *A Programmer's Geometry*, Butterworths

Bülthoff, H. H. and Langer, M. S. (1999) *Perception of shape from shading on a cloudy day*. Technical Report No. 73, Max-Planck-Institut für biologische Kybernetik

Buss, S. R. (2003) *3D Computer Graphics*, Cambridge University Press

Carpenter, L., Cook, R. and Porter, T. (1984) *Distributed raytracing*. SIGGRAPH, 18(3), pp. 137-145

Chalmers, A. G., Jansen, F. W. and Reinhard, E. (1998) *Overview of Parallel Photo-realistic Graphics*, Eurographics [Online] www.cs.bris.ac.uk/Publications/Papers/1000271.pdf

Cook, R. L. and Torrance, K. E. (1982) *A Reflectance Model for Computer Graphics,* ACM Transactions on Graphics, Vol. 1, pp. 7-24

Cornell University (1998) The Cornell Box [online] http://www.graphics.cornell.edu/online/box/

DeLoura, M. (2000) *Game Programming Gems*, Charles River Media

Dobashi, Y., Kaneda, K., Nishita, T. and Yamashita, H. (1996) *Display method of the sky color taking into account multiple scattering. In Proceedings of Pacific Graphic*, pages 117–132

Duerer, A. (2003) *Global Illumination Compendium* [online] http://www.cs.kuleuven.ac.be/~phil/GI/TotalCompendium.pdf

Ebert, D. S., Kenton Musgrave, F. et al. (1998) *Texturing & Modeling A Procedural Approach 2nd Ed.*, AP Professional

Ertl, T., Falk, M. and Schafhitzel, T. (2007) *Real-Time Rendering of Planets with Atmospheres*, Journal of WSCG 07

Fox, G., Messina, P. and Williams, R. (1994) *Parallel computing works.* Morgan Kaufmann

Gooch, A., Gooch, B., et al. (1998) *A non-photorealistic lighting model for automatic technical illustration.*In SIGGRAPH 98 Conference Proceedings, pp. 447–452

Guibas, L. J. and Veach, E. (1997) *Metropolis Light Transport,* Computer Graphics, 31 (Annual Conference Series), pp.65–76

Günther, J., Slusallek, P. and Wald, I. (2004) *Realtime Caustics Using Distributed Photon Mapping*, Saarland University [online] http://graphics.cs.uni-sb.de/Publications/2004/RTPM_EGSR2004.pdf

Haines, E. Ed. (2004) *Ray Tracing News* [online] http://jedi.ks.uiuc.edu/~johns/raytracer/rtn/

Heckbert, P. S. Ed. (1994) *Graphics Gems IV*, Academic Press

Henning, C. and Stephenson, P. (2004) *Accelerating the Ray Tracing of Height Fields*, Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp. 254-258

Hoffman, N. and Preetham, A. J. (2002) *Rendering Outdoor Light Scattering in Real Time*, ATI Technologies [online] http://ati.de/developer/dx9/ATI-LightScattering.pdf

Humphreys, G., Pharr, M. (2004) *Physically Based Rendering*, Morgan Kaufmann

Intel Corp. (2000) *Getting Started with SSE/SSE2 for the Intel Pentium 4 Processor* [online] http://cache-www.intel.com/cd/00/00/01/77/17741_getting_started.pdf

Intel Corp. (2006a) *Moore's Law* [online] http://www.intel.com/technology/mooreslaw/

Intel Corp. (2006b) *Hyper-Threading Technology* [online] http://www.intel.com/technology/hyperthread/

Jensen, H. W. (2001) *Realistic Image Synthesis Using Photon Mapping*, A. K. Peters

Kajiya, J. T. (1986) *The rendering equation*, ACM SIGGRAPH, pp. 143-150

Kaufman, A., Qu, H. et al. (2003) *Ray Tracing Height Fields*, Computer Graphics International, pp. 202-209

Kirk, D. (1992) *Graphics Gems III*, Academic Press

Lengyel, E. (2004) *Mathematics for 3D Game Programming & Computer Graphics 2nd Ed.*, Charles River Media

Marmitt, G., Kleer, A., et al. (2004) *Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing*, Proceedings of Vision, Modeling, and Visualization (VMV) 2004, Stanford (CA), USA, November pp. 16-18O'Rourke, J. (1998) *Computational Geometry in C 2nd Ed.*, Cambridge University Press

Mahovsky, J. and Wyvill, B. (2003) *Fast Ray-Axis Aligned Bounding Box Overlap Tests with Plucker Coordinates*, Journal of Graphics Tools, Vol. 9, No. 1, pp. 35-46

Musgrave, F. K. (1988) *Grid tracing: Fast ray tracing for height fields*, Technical Report YALEU/DCS/RR-639, Yale University, Dept. of Computer Science Research

Nielsen, R. S. (2003) *Real Time Rendering of Atmospheric Scattering Effects for Flight Simulators* [online] http://www.imm.dtu.dk/pubdb/views/edoc_download.php/2554/pdf/imm2554.pdf

Paeth, A. W. (1995) *Graphics Gems V*, Academic Press

Perlin, K. (1997) Original Noise Function [online] http://mrl.nyu.edu/~perlin/doc/oscar.html#noise

Premoze, S., Shirley, P. and Thompson, W. B. (1999) *Geospecific Rendering of Alpine Terrain* [online] http://www.cs.utah.edu/~shirley/papers/snowTerrain/

Phong, B. T. (1973) *Illumination of Computer-Generated Images*, Department of Computer Science, University of Utah, UTEC-CS pp. 73-129

Preetham, A. J. (1999) *A Practical Analytic Model for Daylight.* M.Sc. thesis, Department of Computer Science, University of Utah

Qu, H., Qiu, F. et al. (2003) *Ray Tracing Height Fields*, Computer Graphics International

Rubin, S. and Whitted, T. (1980) *A Three Dimensional Representation for Fast Rendering of Complex Scenes*, Computer Graphics, vol. 14, no. 3, pp. 110-116

Schafhitzel, T., Falk, M., Ertl, T. (2007) *Real-Time Rendering of Planets with Atmospheres*, Journal of WSCG

Shirley, P. (2000) *Realistic Ray Tracing*, A. K. Peters

Shirley, P., Morley, R. K. (2003) *Realistic Ray Tracing 2nd Ed.*, A. K. Peters

Slusallek, P. and Wald, I. (2001) *State of the Art in Interactive Ray Tracing*, State of the Art Reports, EUROGRAPHICS 2001, pp. 21-42

Veach, E. (1997) *Robust Monte Carlo Methods for Light Transport Simulation,* Ph.D. Dissertation, Stanford Universty [online] http://graphics.stanford.edu/papers/veach_thesis/thesis.pdf

Vince, J. (2001) *Essential Mathematics For Computer Graphics Fast*, Springer-Verlag

W3 Consortium (n.d.) *VRML Virtual Reality Modeling Language* [online] http://www.w3.org/MarkUp/VRML/

Wald, I. (n.d.) *The Open Ray Tracer, University of Saarbruecken* [online] http://www.openrt.de/

Watt, A. (1989) *Fundamentals of Three-Dimensional Computer Graphics*, Addison Wesley Publishing

Whitted, T. (1980) *An Improved Illumination Model for Shaded Display*, Comm. ACM, Vol. 23, No. 6, pp. 343-349

## *Parallel Theory and General Programming*

Allen, M. and Wilkinson, B. (2005) *Parallel Programming 2nd Ed.*, Pearson Education

Borland Software Corp. (2001) *Object Pascal Language Guide*, Borland Software

Chalmers, A., Davis, T., Reinhard, E. Eds. (2002) *Practical Parallel Rendering*, A. K. Peters

Coulouris, G., Dollimore, J. and Kindberg, T. (2001) *Distributed Systems Concepts And Design 3rd Ed.*, Pearson Education

James, M. (1989) *Foundations of Programming*, I/O Press

MPI (n.d.) Message Passing Interface Forum [online] http://www.mpi-forum.org/

PVM (2007) Parallel Virtual Machine [online] http://www.csm.ornl.gov/pvm/

Stephens, R. (1998) *Ready to Run Delphi Algorithms*, Wiley

## *Additional Background Information*

Angel, E. (2000) *Interactive Computer Graphics 2nd Ed.*, Addison Wesley

Albert, D. J., Isaacson, E. J., Morse, S. P. (1987) *The 80386/387 Architecture*, Wiley

Canonical Ltd. (2006) Kubuntu Desktop Linux [online] http://www.kubuntu.org/

CodeGear (2007) Delphi 2007 for Win32 [online] http://www.codegear.com/products/delphi/win32

CollabNet Inc. (2006) Subversion [online] http://subversion.tigris.org/

Davis, T., Heider, J. et al. (1999) *OpenGL Programming Guide 3rd Ed.*, Addison-Wesley

Discoe, B. (2001) *Terrain LOD: Runtime Regular-Grid Algorithms*, Virtual Terrain Project [online] http://www.vterrain.org/LOD/Papers/

E-on Software Inc. (2007) *Vue d'Esprit* [online] http://www.e-onsoftware.com/

Foley, J.D. and Van Dam, A. (1982) *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company

Free Pascal Team (1993) Free Pascal [online] http://www.freepascal.org/

Kerlow, I. V. (2004) *The Art of 3D Computer Animation and Effects 3rd Ed.*, John Wiley & Sons

Lazarus Team (2003) Lazarus IDE [online] http://www.lazarus.freepascal.org/

Ordnance Survey (2007) Land-Form PROFILE [online]
http://www.ordnancesurvey.co.uk/oswebsite/products/landformprofile/techinfo.html

Pandromeda Inc. (1999) *MojoWorld* [online] http://www.pandromeda.com/products/

Planetside (1998) *Terragen* [online] http://www.planetside.co.uk/terragen/

USGS (2004) USGS Geographic Data Download [online] http://edc2.usgs.gov/geodata/index.php

Wikipedia (1996) *Low Polygon Modeling* [online] http://en.wikipedia.org/wiki/Low_poly