Real Time Ray-Tracing for Interactive Visual Simulation

Simon G. Skinner
XPI Simulation Ltd.
Surbiton, Surrey, UK
Simon.Skinner@xpisimulation.com

Craig J. Hutchinson
XPI Simulation Ltd.
Surbiton, Surrey, UK
Craig.Hutchinson@xpisimulation.com

ABSTRACT

The recent development of highly powerful programmable graphics cards has made it possible to use Ray Tracing graphics techniques for interactive simulation applications.

Previously, Ray Tracing (RT) techniques have been used for animated film production and non-real time applications where images may be generated in minutes or hours. The same techniques can now be used to generate images in 1/30th or 1/60th second for use where the image must change in response to operator input – for example in ground vehicle or flight simulators.

Use of Ray Tracing techniques potentially allows the same scene to be rendered across a variety of wavebands (optical, thermal, electromagnetic) using broadly similar algorithms, reducing the number of different databases needed and promoting correlation, interoperability and reuse as well as higher image quality. Another potential benefit is reducing the workload, time and most importantly the cost incurred for synthetic natural environment database production, especially in the advanced texturing techniques now used to produce high quality 'serious' games.

This paper describes work funded by the UK Ministry of Defence, and undertaken by XPI Simulation Ltd. and Lockheed Martin UK Insys Ltd., to evaluate the feasibility and implementation of ray tracing techniques for improving the quality and usability of simulation graphics. The paper describes the problems encountered in achieving real time rates for simulation and training applications – for example the performance hits incurred handling moving 3D models with articulated parts, along with some solutions to these issues that have been found.

The paper describes the future potential of real time ray tracing techniques given the steadily improving cost / performance ratio of PC graphics hardware, and the effect that implementation of RT techniques will have on the high performance graphics used in simulation and training visualization.

ABOUT THE AUTHORS

Simon G. Skinner BSc, CENG, MIET is the managing director and CEO of XPI Simulation Ltd. He graduated with a Bachelors in Electronic Engineering in 1986 and spent several years designing real-time software for massively parallel computing systems. More recently he has been involved with research work for Synthetic Natural Environments, data representation and common data formats for simulation and training applications. He is the UK industry representative on the NATO MSG-071 'Missionland' project researching the use of large geo-typical data sets for multinational training exercises.

Craig J. Hutchinson BSc is a Software Engineer of XPI Simulation Ltd. He graduated with a Bachelors of Science in Computer Games Programming in 2008 and has been involved in software development and research at XPI for the last 2 years.

Real Time Ray-Tracing for Interactive Visual Simulation

Simon G. Skinner
XPI Simulation Ltd.
Surbiton, Surrey, UK
Simon.Skinner@xpisimulation.com

Craig Hutchinson
XPI Simulation Ltd.
Surbiton, Surrey, UK
Craig.Hutchinson@xpisimulation.com

RASTER VERSUS RAY TRACING SUMMARY

Raster and Ray Tracing are two techniques for generating 2D images on a screen or projection device based on a 3D internal representation of a scene composed of many polygons. The following paragraphs provide a simplified description of the techniques which are described in much more detail in many other places; a good reference on ray tracing is Glassner, 1989, and on rasterization techniques Möller, 2008.

Simplified Raster technique description

Raster techniques (see Figure 1) involve the projection through a rectangular viewport and consequent transformation of the vertices of a 3-dimensional model into 2-D coordinates; the polygons are then rendered in 2D space ("rasterized") along with an appropriate texture. Depth (or Z) buffering is used to ensure that objects closer to the viewer cover objects that are further away. Optimizations include "culling" which ensures that only polygons which are within the viewport are transformed and drawn on the screen.

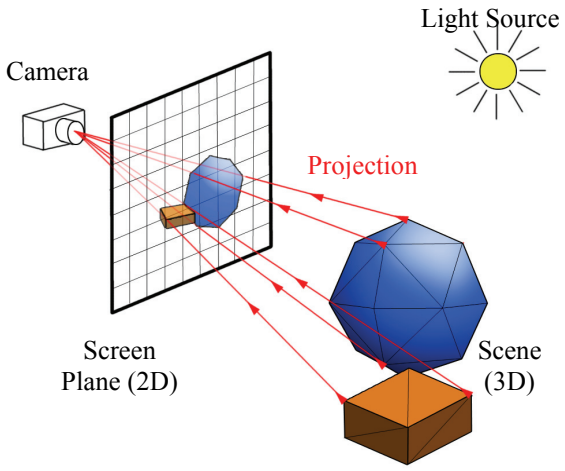

Figure 1 - Rasterization

Software libraries such as OpenGL or DirectX provide a method of abstracting the rasterization process. On top of these libraries, simulation graphics engines impose further abstraction so that the typical user is normally dealing with the movement of discrete objects such as tanks and aircraft within a defined geographic area, with various other parameters to control lighting, visibility and the appearance of weather. Virtually all visual systems for simulation and training use the rasterization process as it is fast and effective, and graphics cards and processors have evolved to significantly accelerate the process. With current technology, rasterization produces relatively dense and complex images within 1/60th or 1/30th second.

Simplified Ray Tracing technique description

Ray Tracing techniques involve casting a series of rays from the observer through a viewport (see Figure 2). The screen pixel generated is calculated based on the sum of the effects on the ray as it travels to hit light sources, this might be direct effects but also indirect reflections, refractions, shadows and other interactions including absorption which would model atmospheric effects on the light ray. The complexity of the calculations can be restricted by limiting the hierarchy of secondary and subsequent rays within the scene.

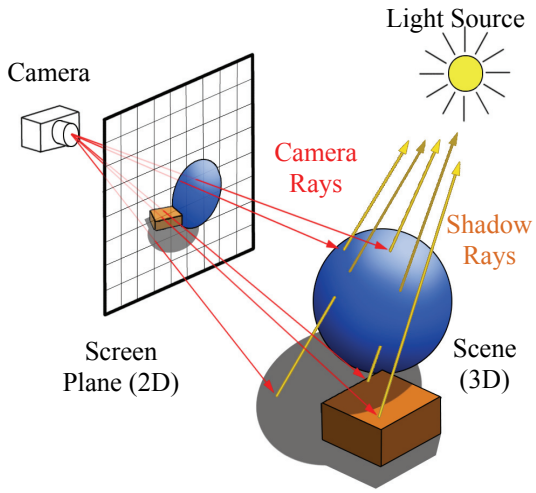

Figure 2 - Ray-tracing

It should be noted that the technique is not restricted to the visual light domain but may be used at other electromagnetic radiation frequency domains such as radar and non-visible light (e.g. infra-red), in this case the 'light' sources are replaced with radar or heat sources. The calculations on reflectivity, absorption and refraction must then take account of the different properties of the surfaces and items being modelled.

DEFINITION OF "REAL TIME"

The frame rate of image generation is a key statistic to determine the performance of a system to be used in interactive training and simulation applications. In addition the frame rate generally drives the latency of a visual system. In many simulations, the latency of the visual system is a key measure of usability. For example, to achieve Level D FAA standard, the complete simulator must have a latency of less than 150ms (Federal Aviation Authority). In many cases the visual system running at 60 frames a second would contribute 60 to 80ms of this time – with about 50ms within the image generator and the remainder in the projection or display system

Different applications require different frame rates. Historically frame rates above 15 Hz have been considered real time, using modern technology frame rates of 30Hz are considered satisfactory for many applications, with 60 Hz or even higher (e.g. 100Hz) being necessary for the most highly accurate man-in-the-loop simulation systems.

Ray-Tracing in Real-Time

Ray-tracing is one of the earliest forms of rendering techniques but it has, up to now, not been used in real-time simulation and training applications. This is due to the huge number of computations required to render even a simple scene at low resolution. At 1024x768 screen resolution, over 1.5 million rays are required to render just a simple shadowed scene. Until recent years computers were not fast enough to calculate all these rays in fewer than 66 milliseconds to obtain a 15Hz update rate. Previous work (Muuss, 1995),
has made attempts to reach 'real-time' performance with a target of 5 frames per second; the actual achieved rate of one frame every 30 seconds was seen on a 28 GFLOP machine (in 1995, this was state-of-the-art computer hardware with a cost of $6 million).
Ray tracing computation has the following characteristics:

- Computations with single precision floating point accuracy
- Algorithms are inherently highly parallel

This means that modern fast parallel computer systems will have better performance than older systems.

Nevertheless, until recently, ray tracing techniques within modelling and simulation have been restricted to applications like generating highly accurate sensor signatures of military vehicles.

GPU VERSUS CPU

Over the last decade the Graphics Processing Unit (GPU) has become a standard part of almost every computer used at home and at work. The GPU can process far greater quantities of data than that of a single CPU of similar price (see Figure 3). Different memory configurations, processor speeds and algorithms will of course give different real-world results; however there is no doubt that the recent

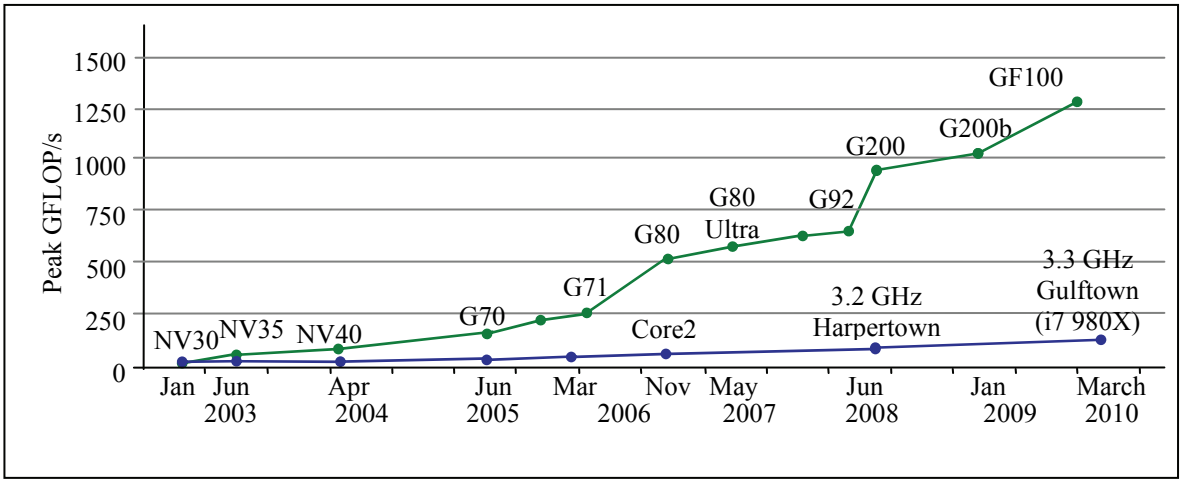

Figure 3 - GPU (NVIDIA) vs. CPU (Intel) peak single precision floating point performance

increase in peak performance of GPU's is much higher than the increase in peak performance of CPU's.

The design of the GPU has been engineered to perform the same or similar rendering tasks as fast as possible using a fixed 'pipeline'. For years this was sufficient for most raster graphics but in later years the requirement to make the graphics processor do what the software developers want has taken priority.

The end result of this hardware development is a highly parallel, extremely powerful, general purpose compute engine with applications in High Performance Compute (HPC) applications, and uses beyond conventional computer graphics.

Software Techniques

Many special graphics techniques and effects can now be programmed through the use of "shaders" – graphics programming code blocks, and now other means such as the Compute Unified Device Architecture (CUDA) programming language for NVIDIA GPUs.

To ensure the GPU was a feasible solution we implemented a ray-tracer using the CUDA architecture. Results of this were very positive and GPU ray-tracing was concluded to be possible with currently available hardware and software APIs. Some limitations were also identified associated with the differing hardware design of the GPU which was shown to be an issue. These include issues such as branching and memory limitations. With clever design these issues can be minimised.

A CPU ray-tracing prototype was also implemented and presented. This implementation, although fairly optimal, did not perform well when compared to what a GPU could manage. Initial results a year ago showed the GPU implementation was more than an order of magnitude faster than the CPU implementation; at this point a decision was taken to concentrate all further efforts on the GPU.

NVIDIA OptiX

During the first half of the project NVIDIA announced a ray 'casting' library called 'NVIRT' – later renamed to OptiX. This package accelerates the ray-tree traversal tests. The library itself does not implement ray-tracing and can be used for any means such as collision detection or in our case ray-tracing. By utilizing the package we were able to benefit from the in-depth knowledge of the hardware developers and focus our limited time on creating accurate and fast images. The API also allows us to leverage newer GPUs as they arise without changes to our renderer.

T45 DECK MARSHALING SIMULATOR

To put the developed work into practice, the next stage was to implement a prototype ray tracing system for a real training and simulation application.

The training system chosen is the T45 Deck Marshalling Simulator. The Type 45 Destroyer (T45) is a category of ship which has recently been brought into service with the Royal Navy. The simulator system is designed to allow training of the Flight Deck Officer (FDO) and Flight Deck Controller (FDC) who control the landing, refuelling and re-arming of rotary-wing aircraft which can land on the rear deck of the destroyer. This system was chosen because:

- The training simulator has recently been delivered and thus it is easier to substitute new technology than would be the case for an older system – we have current working knowledge of the source code and databases
- Ray Tracing potentially provides image quality improvements which benefit the training task
- There are some technical challenges to overcome - including highly articulated characters
- The system does not require 60 Hz update rates, rates from 15Hz to 30Hz are acceptable

The following sections describe the work carried out to produce a prototype ray tracing visual system for the simulator. A screenshot of the existing system with a conventional raster graphics engine (XPI's "Tempest" OpenGL renderer) is shown in Figure 4 with the ray traced equivalent in Figure 5.

Figure 4 - Existing raster simulation screenshot

Note that the ray traced scene has completely correct dynamic shadows for all objects in the scene, something extremely difficult to achieve in conventional raster engines (in the raster example, just the helicopter has an accurate shadow).

Figure 5 - Prototype ray traced simulation screenshot

Database compatibility

Many military simulations use the OpenFlight format (Presagis Inc., 2010) for object model databases due to its focus in targeting real-time as well as implementing a scheme for articulations and switches. It is a standard interchange format for UK military simulator systems and use is widespread throughout the international simulation industry. The current models (T45 ship, Merlin rotary wing aircraft) are in this format so the ray-tracer needed to be fully compatible with the features it includes.

It was found that some of the standard fields within the OpenFlight model dealing with transparent surfaces were not filled in during the original modeling process, these needed modification to deal with the reflective and refractive surfaces in the model – the mirrors and cockpit glass (see Figure 6). This is an issue of backward compatibility for future systems, and proper representation may require additions or changes to the existing OpenFlight format.

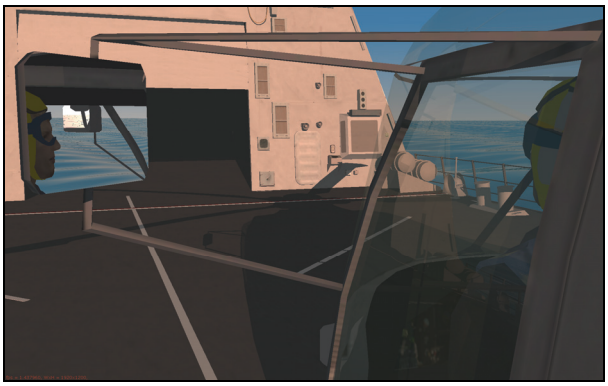

Figure 6 - T45 and Merlin modelled in Creator showing glass and mirror reflections

Moving Models

One of the most important requirements for a fast ray tracer is the need for a quick method to test for intersections between a ray and objects in the scene. A simplistic approach of testing every ray for intersection with every polygon in the scene would be far too slow. Therefore, it is necessary to index the polygons in a way that allows the program to quickly find those polygons which might lie in the path of a ray. The structure generated to do this is called an 'Acceleration structure'. The computation required has not been near real-time until recently, so moving objects in ray-tracing has always been difficult.

The NVIDIA OptiX toolset can be utilized to perform this functionality and we are able to maintain interactive rates with multiple moving vehicles. The performance loss is still significant however unlike raster which incurs no such penalty.

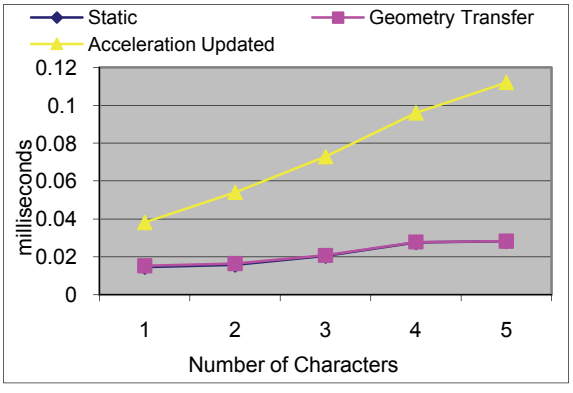

Figure 7 - Character animation timings

To make things worse, characters (e.g. humans) require significantly more processing due to the acceleration structure needing recalculation at the polygon level as

the physical geometry deforms (Figure 7). This process severely hinders performance in ray-tracing and is very much an area in which further development is required.

Articulations

A key part of the T45 deck marshalling simulator takes advantage of model articulations and switches to show visual cues and movement. In their simplest form, articulations are simply sub-models which can move separately, but relative, to the main model itself. Examples of this are the helicopter rotor blades which need to rotate around the pivot point.

To support articulations, the internal model structure was extended to separate the models into their articulated parts, each of which are given a separate transform node. However unlike a raster engine, deep hierarchies are best avoided to reduce the number of ray-space transformations required and to better optimize the acceleration structures (see Figure 8).

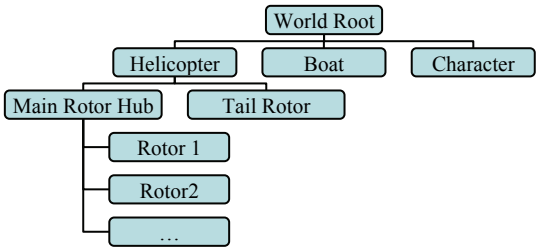

Figure 8 - Scene hierarchy containing 3 levels of transform (before compacting)

Instead a fairly flat hierarchy should be created with as few levels as possible (see Figure 9). This simplifies rendering and potentially improves performance whilst maintaining the same functionality as before.

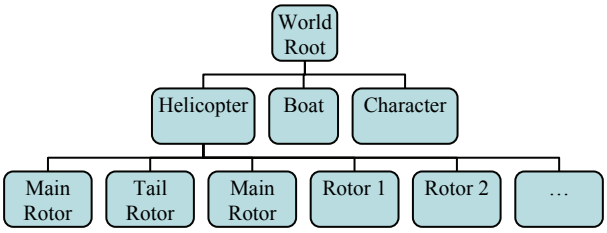

Figure 9 - Scene hierarchy containing 2 levels of transform (after compacting)

It is important to note that the flattened hierarchy is generated at the time of model load, so there is no requirement for a modeller to change existing OpenFlight models – a key requirement to ensure interoperability and model reuse.

Switches

Switches require the addition and removal of geometry from a group. This occurs at the model level and would need acceleration structures to be updated at runtime or for there to be increased memory usage from the need to store and cache all possible switch combinations. Supporting switches with the current software causes slowdowns in the rendering but it is expected that we will be able to perform asynchronous updates to the acceleration structures in the near future.

Lights

OpenFlight supports light sources within models so the ray-tracer therefore needed support for them. As discussed before ray-tracing is 'physical' in nature and lighting of any type is possible given the computing power available. The implementation we have created is naïve and performs no light culling but results have shown the improved interactions given by ray-traced lights over previous lighting types do improve the visual quality (see Figure 10).

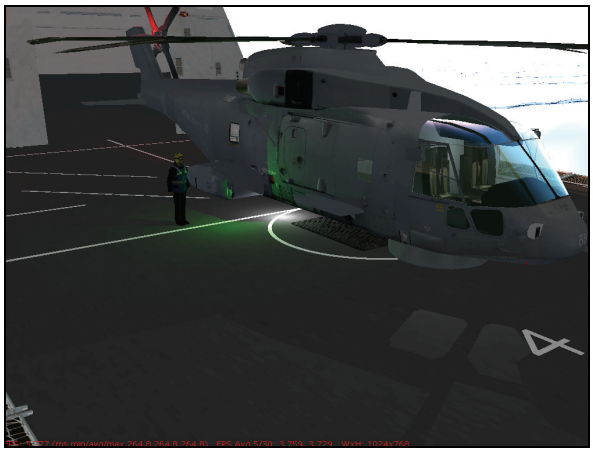

Figure 10 - Multiple lights and shadows

Sea representation

Sea is normally represented in one of two ways in raster graphics. The first method is to use a polygon mesh that is conformed to some kind of mathematical equations, such as multiple sine waves which then give an effect of 3D waves. On top of this mesh is placed a combination of textures and effects to represent the smaller waves and sea surface. One of the problems with this technique is that it is not possible to achieve the same level of polygonal detail across the whole sea surface, necessitating the use of some form of Level of Detail (LOD) system to reduce the load in the distance.

More recent raster implementations overcome this by 'tessellation' in which geometry is created mathematically on the graphics hardware itself.

The second, more common, technique renders the sea surface as a flat polygon with a complex "shader" (a programmable render software fragment) producing the visual effect of the waves and disturbances. This often looks better when viewed from above the water and when the sea is relatively calm, but the effect breaks down when the eye-point is close to the water.

Ray tracing is not limited to just rendering polygons and can represent any mathematical shape as a 'parametric' equation. This gives us the benefit of extremely low memory footprint and removes any necessity to update acceleration structures or geometry. The shapes can therefore have near infinite detail or fractal characteristics unlike raster which exhibits polygon facets when viewed closely.

In some systems the sea has been created by blending multiple sine waves to create the undulations in the surface. A minimum of 4 waves intersecting at angles were seen to be required to form a realistic looking surface. To approach this problem the sine wave must be 'parameterised' as a mathematical equation, however a sine wave is not a polynomial equation (of finite length) and is not easily solved.

It was however possible to approximate the peak and troughs of the Sine wave using a parabola, which is of finite length, as shown in Figure 11.

This sort of mathematical function can be extended to a number of shapes and waves. It was beyond the scope of the project to investigate sea state models and a straightforward Fourier series was used due to its simplicity.

This method produced sea that can go all the way to the horizon and near infinity without increased memory requirements. The only underlying problems experienced are those due to the numerical limitations of the computer.

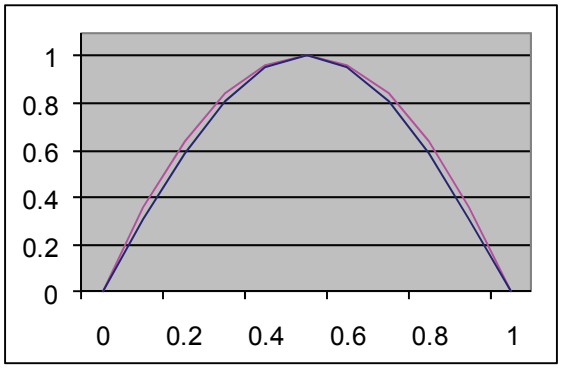

Figure 11 - Sine wave (Pink) Parabola (Blue)

Figure 12 shows the result achieved with the simple sea algorithm implemented for the ray tracing prototype.

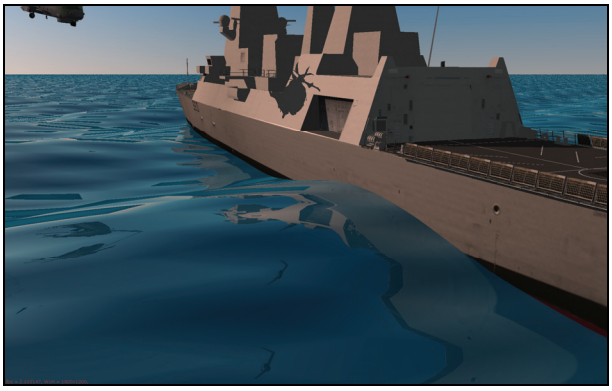

Figure 12 - Simple parametric sea

Performance achieved

During the project, performance gains were impressive; this was due to a combination of the following factors:

- Hardware performance improvement
- OptiX and CUDA improvements
- Ray tracing algorithm developments

Countering this improvement, the introduction of more features as the software develops negatively affects performance as would be expected. Figure 13 shows the net performance improvements over a 9 month period.

Overall, the ray-tracing prototype on the T45 simulator described above achieves between 10Hz and 30Hz image update rates depending on view position, using a standard PC with single card GPU (Tesla C2050 with 3GB RAM). The performance from the prototype is close to being acceptable for the application and is also cost-effective (under $3000 hardware cost).

OTHER FEATURES

Image Warping

Unlike raster graphics, ray-tracing is not constrained to planar rendering to a 2D surface such as a flat screen monitor. The unconstrained images that ray-tracing can produce are able to produce warped and high field-of-view images, such as that shown in Figure 14. This may include such features as fish-eye lenses – now appearing in certain military vehicle observation cameras, and spherical or cylindrical warping for large scale simulators.

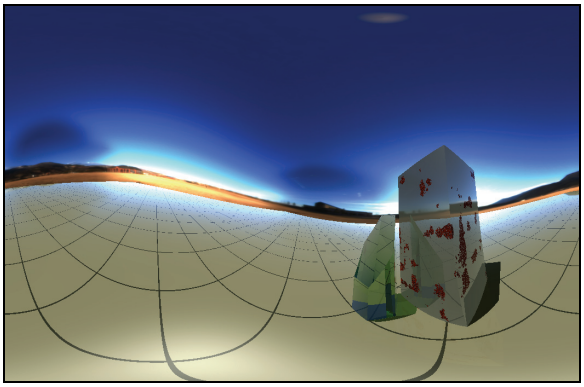

Figure 14 - Spherical/Omni-directional Camera showing a full 360 degrees render

Raster graphics has always needed to perform these effects via multiple renders with the use of post processing of the imagery for warping and blending. This often incurs an extra latency of a video frame and depending on the solution may result in reduced quality images as pixels are stretched to cover multiple samples in the output image. Ray-tracing can overcome all constraints present in these situations.

Anti-Aliasing

Current Raster rendering techniques have various efficient anti-aliasing techniques which make it possible to greatly improve the visual quality of images without a large performance hit. Almost all simulation systems now have anti-aliasing enabled because of its use in removing distracting 'aliasing' artefacts particularly in moving scenes.

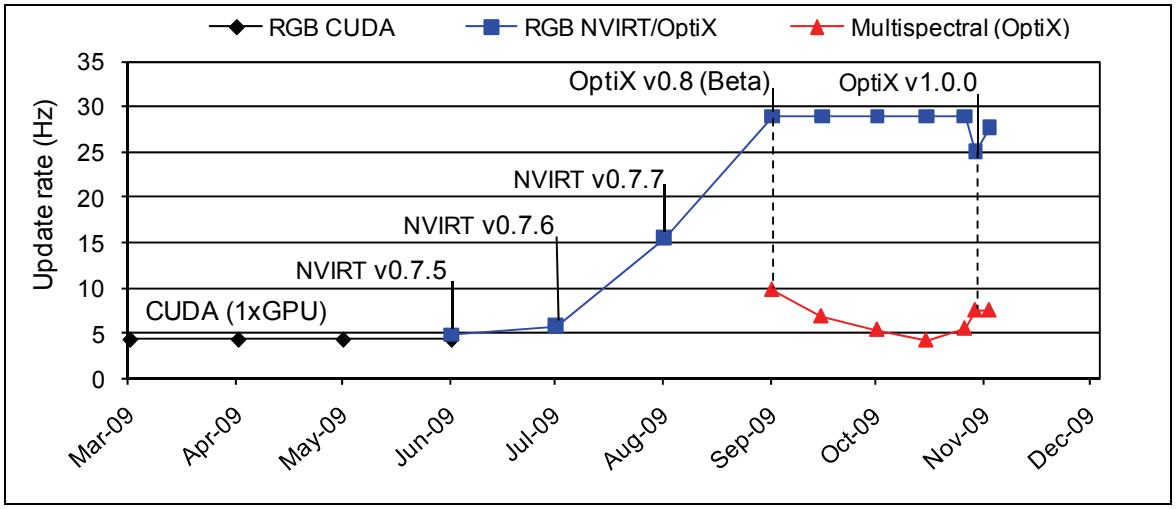

Figure 13 - Performance over time; dual NVIDIA Quadro 5800 @ 1024x768

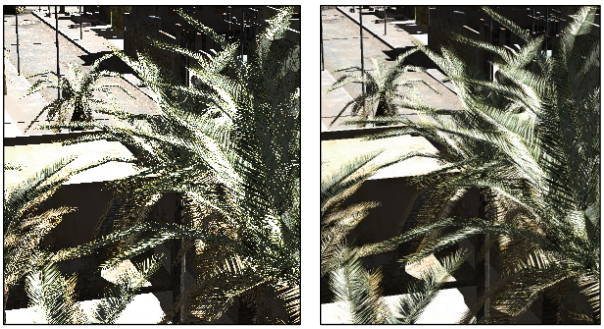

Figure 15 - Non anti-aliased image (left) and super sample anti-aliasing (right)

In the early days of raster rendering, Anti-Aliasing (AA) was only possible using super–sampling, rendering the image at a much higher resolution and then sampling the image down to the correct resolution. This technique was used to produce Figure 15 using our ray-tracer.

More recent GPUs use a number of more advanced techniques to anti-alias only where the benefit is greatest, for example at the edge of polygons. With these techniques and performing the AA directly in the hardware the effect of AA on performance is greatly reduced. Leveraging this highly efficient anti-aliasing could be very beneficial but due to some complications we have not yet implemented such an approach. With OpenGL 3.x and DirectX11 it is potentially viable we can leverage this highly efficient AA using Hybrid rendering.

HYBRID RENDERING

Both raster and ray tracing rendering techniques have benefits relating to efficiency and image quality. Potentially a hybrid approach could 'bring together' the best of both into a unified rendering technique.

The graphics hardware in current systems does, and will continue to have, raster acceleration for a while to come. It made sense in our research to utilize this hardware to make best use of the existing COTS graphics hardware.

With ray tracing, the scene is being sampled by millions of rays that can represent, to a greater or lesser degree, the real physics of light or other electro-magnetic radiation. This is in contrast to the (incumbent) raster technique, which relies on rendering approximations and using database artists to adjust the appearance and rendering performance. Ray-tracing can therefore produce the physical results introduced by light 'rays' in the real world, these include reflections, refractions, ands shadows. More effects are possible – an example is "global illumination" which produces far improved lighting fidelity.

Ray-tracing natively supports many lighting effects and could potentially support any effect experienced in the real world by the simple inclusion of more rays and the processing power required. Most lighting effects have no need for expensive one-off artist modifications to the database models as we have presented.

Techniques

The preferred hybrid approach would involve a true joining of the two rendering approaches, in which the rasterizer and the ray-tracer are highly integrated. This means it would be possible for the raster pipeline to trigger a ray-trace call highly efficiently.

Within the current architectures available there are only a few potential ways the two techniques can interoperate – at the moment multiple passes are required.

Implementation

The technique implemented allowed us to keep all existing ray-tracing code. The only change was to allow ray-tracing to commence based on the distance, normal, and texture data acquired from the raster engine in an effective manner.

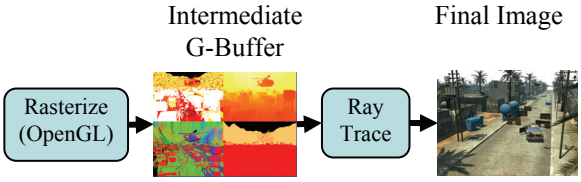

Figure 16 – Hybrid rendering method

First, data is calculated in the Raster pass (pass 1) and written into Textures which are passed onto the ray-tracer (pass 2). The intermediate data buffer containing information needed by pass 2 is called a G-Buffer (see Figure 16). Raster shaders were written using the GL Shader Language (GLSL) to generate the G-Buffer. GLSL is a standard part of OpenGL and allowed us to access a large amount of information from the raster pipeline which the ray-tracer would have previously calculated less efficiently on its own.

Once the G-Buffer data is gathered it is then the job of the ray-tracer to initiate ray-tracing, as if it were a normal render. This means all the effects we obtain in

ray-tracing will work in our hybrid, and fully ray-traced implementations work seamlessly.

Performance gains

The benefits were found to outweigh most issues incurred, as for a test database performance varied between views to give 50-95% better performance than pure ray tracing alone (See Table 1). This is a fairly significant benefit; on a single Tesla C2050 card (the latest technology which is about 2.5 times faster than a Quadro 5800 used for testing) frame rates of 40-65Hz are being achieved for the static scene.

Table 1. RT vs. Hybrid relative performance (Quadro 5800 4GB @ 1920x1200)

Posn. 1	Posn. 2	Posn. 4	Posn. 7	Posn. 8
188.21%	**194.51%**	151.85%	178.98%	152.72%

The current implementation however does not scale across multiple GPUs due to bandwidth limitations. Ray-tracing itself is inherently scalable and the need for this inter-card communication should be removed by performing alternate frame-rendering similar to NVIDIA SLI technology or by reducing the data transfer required through compression and optimizations.

CONCLUSIONS

The research work described in this paper has demonstrated that:

- Using Ray Tracing techniques it will soon be feasible to achieve the update rates needed for interactive simulation and training applications
- It is possible to use COTS PC architectures and graphics cards to create image generation systems which produce higher quality images including better shadows and lighting. These improvements will likely reduce the amount of manual database editing needed, reducing the cost of generating high quality visual simulations.
- Hybrid approaches (using raster and ray tracing techniques) offer performance gains beyond just using ray tracing with no loss in quality

The research has proved successful in producing a prototype application based on existing database models and applications; this has implications for improving legacy systems and reusing existing work.

Other work carried out but not included in this paper has shown that the techniques described can be applied not only to the visual spectrum but in other areas of the electromagnetic domain such as infra-red or at radar frequencies. This will increase the possibility of having a single common correlated dataset, thus improving simulation interoperability and reuse.

FURTHER WORK

The research work is expected to continue examining the following areas:

- Improving performance with concentration on achieving algorithm optimization, improved memory usage and improved image quality
- More features such as volumetric atmospheric effects, battlefield smoke and other weapons effects
- Increased use of parametric surfaces and data abstractions to reduce memory requirements
- Use in simulations across multiple EM wavebands and sensors including infra-red, NVG and radar.

It is also planned to exploit the research and replace current raster based image generators with ray tracing variants in some existing simulator and training devices.

ACKNOWLEDGEMENTS

This work was sponsored by the Defence Technology Innovation Centre, part of the UK Ministry of Defence.

The authors acknowledge the valuable contribution to this paper of Colin A. Stroud and Trevor Ward of Lockheed Martin UK Insys Ltd. who were partners on the project.

REFERENCES

Federal Aviation Authority. *Federal Acquisition Regulations 14 CFR 60*. In *General Simulator Requirements, Appendices A & C.*

Glassner, A. S. (1989). *An Introduction to Ray Tracing.* Morgan Kaufmann Publishers.

Muuss, M. J. (1995). Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Models. *BRL CAD Symposium.* Aberdeen Proving Ground, MD.

Presagis Inc. (2010, 6 22). *OpenFlight*. Retrieved 6 22, 2010, from Presagis: http://www.presagis.com/products_services/standards/openflight/#

Möller, Tomas, et. al. (2008). *Real-time Rendering.* AK Peters Ltd.