Part II

Maximising the Quality and Performance of A Real-time Interactive Rendering System

Chapter 4

Benchmarking the Rendering Algorithms and Techniques

Chapter 4 presents the critical analysis and benchmarking of the previously discussed rendering algorithms and techniques as utilised by our interactive rendering engine. The empirical analysis presented in this chapter allows us to explore in the next chapter the practicality and performance benefits of a dynamically scalable interactive rendering engine in which GPU-CPU utilisation, as a secondary proof of concept approach, has been unified.

Outline:

- Benchmarking mechanism
- Evaluation criteria
- Algorithm comparison

4.1 Benchmarking Mechanism

Benchmarking entails running a computer program with the aim of assessing its performance. This action is normally hardware-centric and intended to measure the performance of numerous subsystems and/or execution routines. We use such a system to evaluate the previously discussed rendering subsystems. This benchmarking system basically functions as a plug-in to the previously discussed rendering engine where real-time performance data are streamed to a file-based database for post-processing and analysis. Figure 4.1 gives a visual representation of this system.

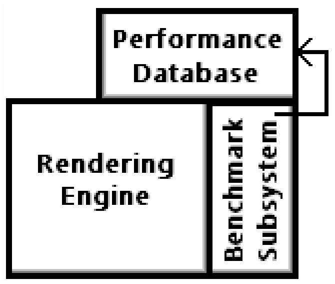


Figure 4.1 The rendering engine, benchmarking system and performance database.

Critical analysis was performed via scripted camera movement, object and light source additions. This was done not only to ensure consistent testing, but also to ease future validation and replication of results.

4.2 Rendering Subsystem Evaluation Criteria

The set of criteria used to evaluate the presented rendering techniques (such as cube mapping, post-processing effects and stencil shadow volumes) is now presented. The given evaluation criteria were selected with the aim of assessing the relationship between rendering quality and performance. This assessment provides the basis of the system presented in Chapter 5 in which the dynamic selection of algorithms as well as CPU-GPU process allocation (for cube mapping and physics processing) is used to control performance and quality. Table 4.1 lists the proposed evaluation criteria in the first column, indicating in parenthesis whether its focus is on quality, performance or both. The second column provides motivation for the criterion's inclusion.

Evaluation Criteria	**Motivation**
Scalability ***(performance)***	Evaluating the performance of an algorithm based on the intensifying complexity of the rendered scene allows for the identification of algorithmic limits and the maximum threshold for scene and model complexity. (Analyse the overall

	performance impact due to, for example, an increase in the number of light sources and the shadow casting model's polygonal complexity).
Rendering Accuracy and Detail *(quality)*	Determining whether, for example, a shadow is cropped and/or skewed properly, and accurately projected onto other models and surfaces, or whether a reflection is accurate allows for the evaluation of rendering quality.
CPU/GPU Utilisation *(performance/quality)*	Comparing, where applicable, a standard GPU-driven implementation to the same implementation being run on a CPU (with utilisation of modern multi-core architecture); allows for the evaluation of maximized parallelism versus conventionally GPU-based rendering.

Table 4.1 Evaluation criteria

4.3 Algorithm Comparison

This section compares the presented engine's core rendering elements. It also lists the observed results with specific emphasis on the most appropriate application areas.

To gather the necessary results, all algorithms were implemented for a number of scenes. In each case, unless otherwise stated, the scene was a relatively simple cubic environment featuring a single movable 3D model and a variable number of light sources. The 3D models utilised are those provided as samples by the Microsoft DirectX SDK (Figure 4.2). The test system had the following configuration:

NVIDIA GeForce^{TM} GTX 570 (1280MB GDDR5) (Video Card),
Intel Core i5 650 @ 3.2GHz+ (Dual Core Processor),
4.0GB (Memory),
1920x1080 (Screen Resolution).

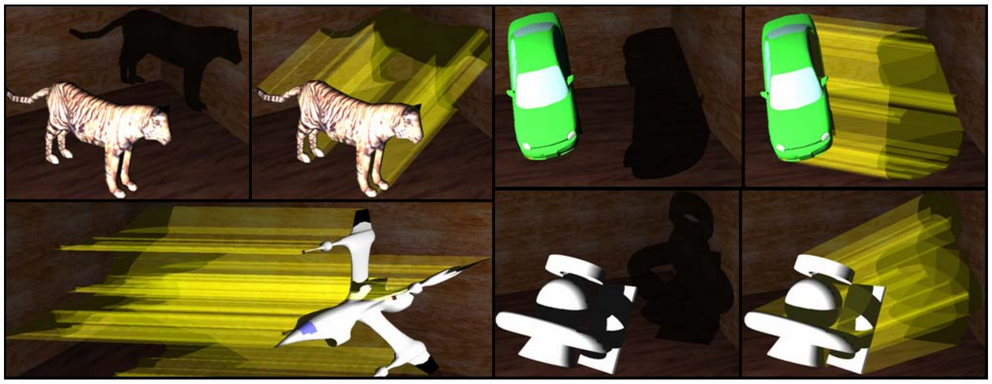


Figure 4.2 The 599-face 'tiger' mesh, 1628-face 'car' mesh, 4136-face 'shapes' mesh and a 9664-face 'battleship' mesh.

4.3.1 Shadows

The presented evaluation focuses on a number shadow rendering algorithms (discussed in section 3.6), specifically the stencil shadow volume algorithm, the shadow mapping algorithm and a number of hybrid approaches such as McCool's shadow volume reconstruction using depth maps, Chan and Durand's hybrid algorithm for the efficient rendering of hard-edged shadows, Thakur et al's elimination of various shadow volume testing phases and Rautenbach et al's shadow volumes and spatial subdivision approach. Please note that the mean performance of each algorithm is shown (performance data has been average over the four models due to individual behaviour showing identical patterns). Also, for the detailed critical analysis, please see the MSc dissertation, An Empirically Derived System for High-Speed Shadow Rendering (2008).

Starting out, it is important to note that Rautenbach et al's spatial subdivision algorithm was analysed in a statically lit environment. This results in its relatively high performance when compared to the other algorithms. Its performance was found to be comparable to the basic stencil shadow volume algorithm in situations where dynamic lighting is implemented. This octree-based algorithm will thus outperform all other algorithms where light sources are not added, moved or removed.

In Figure 4.3 the frame rates achieved via the implementation of spatial subdivision is compared to that obtained using the Heidmann algorithm. It is clear from the data that Rautenbach et al's approach results in significantly better performance than the original stencil shadow volume algorithm (40% better for one light source and 200% better for eight).

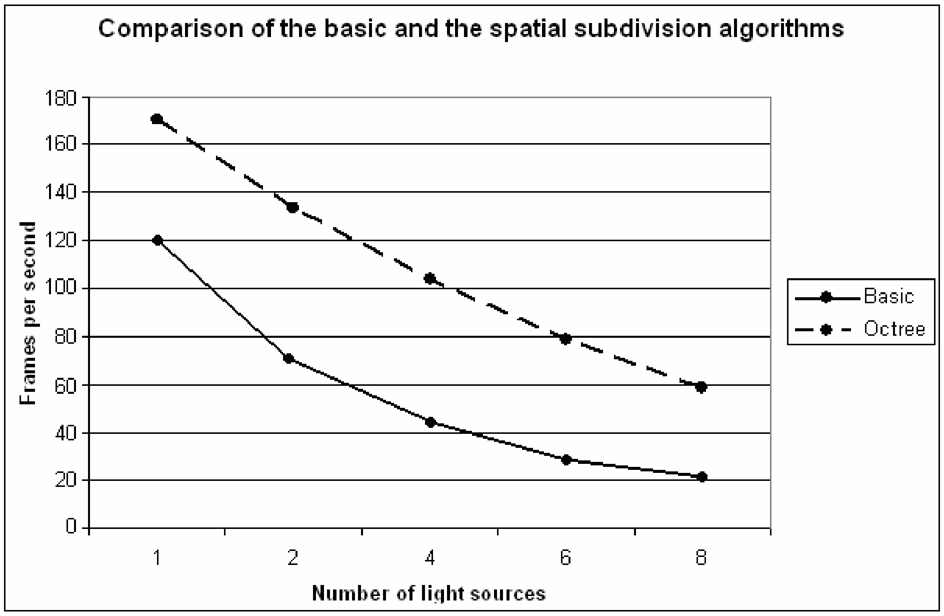


Figure 4.3 Comparison of Rautenbach et al's spatial subdivision approach with the depth-fail stencil shadow volume approach.

In Figure 4.4, the frame rates attained from using spatial subdivision combined with the utilisation of the SSE2 instruction set is compared to that obtained from using the Heidmann algorithm. Comparing a standard C/Direct3D implementation to the utilisation of Intel's SSE2 instruction set allows for the evaluation of maximised parallelism (as offered by these instruction sets) versus conventionally sequentially executed routines. Intel's SSE stands for Streaming Single Instruction, Multiple Data Extensions. It is based on the principle of carrying out multiple computations with a single instruction in parallel (Intel, 2002). The SEE instruction set (specifically SSE2 found on the Pentium 4+ architecture) also adds 64-bit floating point and 8/16/32-bit integer support.

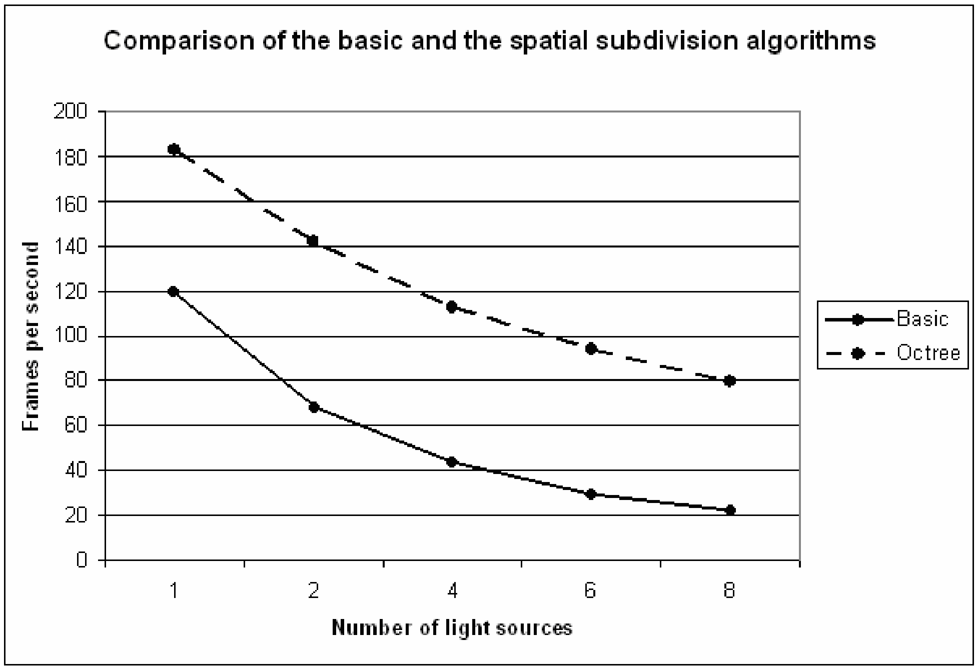


Figure 4.4 Comparison of Rautenbach et al's extended approach with the depth-fail stencil shadow volume approach.

From the results given in Figure 4.4, it is clear that Rautenbach et al's spatial subdivision approach combined with SSE offers the best performance for statically lit scenes. This algorithm does, however, require processing time for Octree-construction and the non-Octree enhanced shadow volume algorithms perform much better in situations where light sources are dynamically added, moved or removed. The presented rendering engine will use the spatial subdivision approach coupled with SSE2 utilisation for all environmental areas lit using static light sources.

From Figure 4.5, listing the mean performance comparison of all the shadow algorithms listed in section 3.6, it is clear that Chan and Durand's (2004) algorithm is the second best algorithm when rendering high-quality shadows with only a single dedicated light source. This algorithm shows significant performance degradation when more light sources are added. It does, however, outperform all the remaining shadow volume-based algorithms for up to eight light sources. The presented rendering engine will use Chan and Durand's algorithm for all scenes consisting of eight or less dynamic light sources when high-quality shadows are required.

Furthermore, the shadow mapping algorithm is observed to perform only slightly worse than Chan and Durand's (2004) algorithm (when rendering scenes consisting of just one light source). That said, the critical analysis implementation does render low-resolution shadow maps. However, increasing this shadow map resolution will have a net-negative impact on the scene's overall rendering performance. Shadow mapping (with average shadow resolution) is used for all scenes consisting of two or more dynamic light sources and where the shadow casting objects are located a significant distance from the point-of-view.

McCool's (2000) algorithm is the second best choice when dealing with scenes featuring one to eight light sources and when high-quality shadows are required. We won't be utilising this algorithm, rather opting for Chan and Durand's hybrid approach.

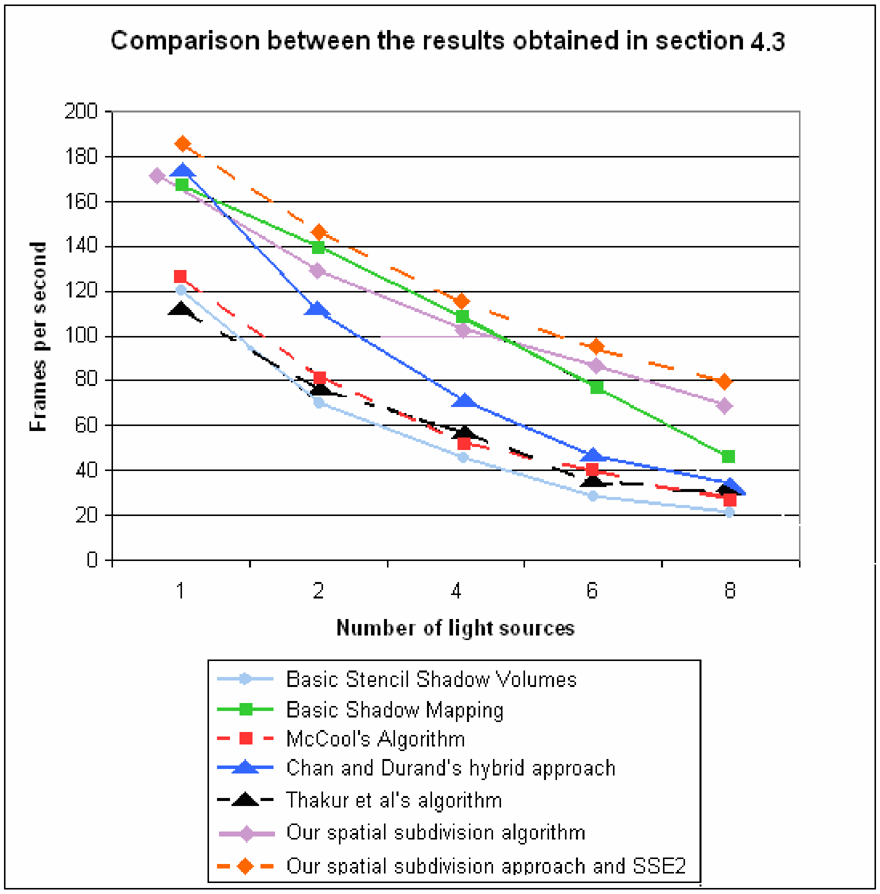


Figure 4.5 Comparison of all the shadow algorithms listed in Chapter 3 (1-8 light sources).

The previous comparison (given in Figure 4.5) only deals with a limited number of light sources. The choice between the most appropriate algorithms is, however, mostly superficial due to 200 frames per second and 60 frames per second displaying similar to the human eye. It is only when frame rates fall below 30 per second that we start to notice. Running the same simulations (but with the light source count ranging from nine to sixteen) shows a rapid decrease in the frames per second performance. Figure 4.6 shows these results.

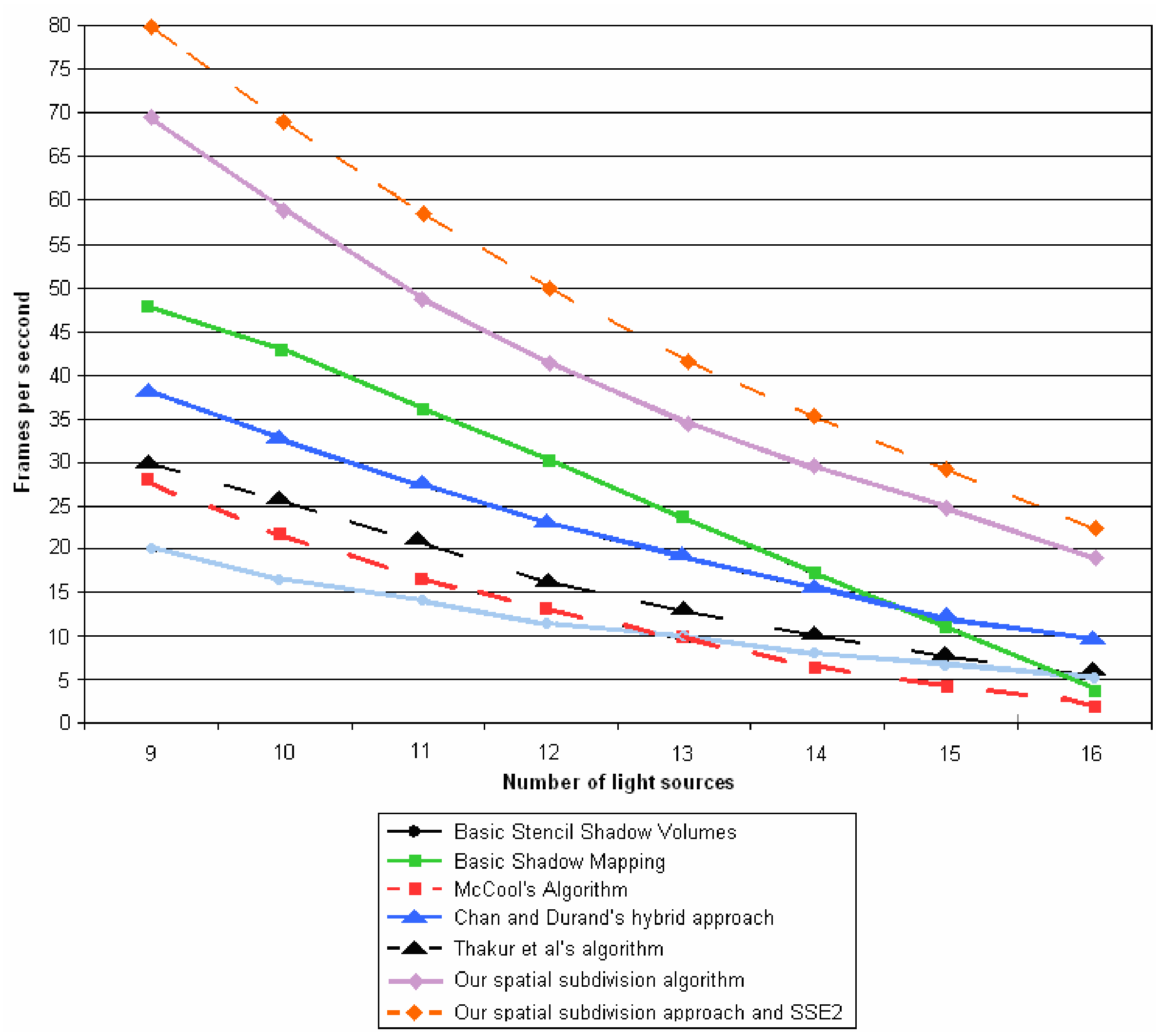


Figure 4.6 Comparison of all the previously listed algorithms (9-16 light sources) – please note; Rautenbach et al's spatial subdivision algorithm was analysed in a statically lit environment, thus resulting in its high performance (its performance is comparable to the basic stencil shadow volume algorithm in situations where dynamic lighting is implemented).

Considering Figure 4.6, Rautenbach et al's spatial subdivision approach coupled with the SSE2 instruction set is once again observed to outperform all the other algorithms. This algorithm is, as mentioned, only amenable to environments utilising static lighting –

making the comparison a bit bias. The presented rendering engine will, however, continue to use this algorithm for all environmental areas lit using static light sources.

The basic shadow mapping algorithm remains the best choice when dealing with dynamically lit environments, at least when working with fourteen or less light sources and when shadow rendering quality is not as important (see Figure 4.5 and 4.6). The presented rendering engine will use shadow mapping for all scenes consisting of more than two and less than fourteen dynamic light sources and where the shadow casting objects are located a significant distance from the point-of-view. Chan and Durant's algorithm will, however, prove a better choice for both close range and distant objects when rendering scenes consisting of fourteen or more dynamic light sources. Chan and Durrand's algorithm will also be used for all scenes consisting of nine or more dynamic light sources when high-quality shadows are required.

Shadow selection is based on the optimisation of the rendering frame rate and shadow quality. The presented rendering engine will thus select shadow generation algorithms by taking not only the scene's frames per second performance data into account but also by factoring in the viewer's position in relation to the shadow being rendered. The rendering accuracy and detail of distant shadows will thus carry less weight than those rendered relatively close to the viewer. Table 4.2 summarises the algorithms of choice based on the algorithmic comparison and scene conditions such as view distance, dynamic/static light conditions and number of light sources.

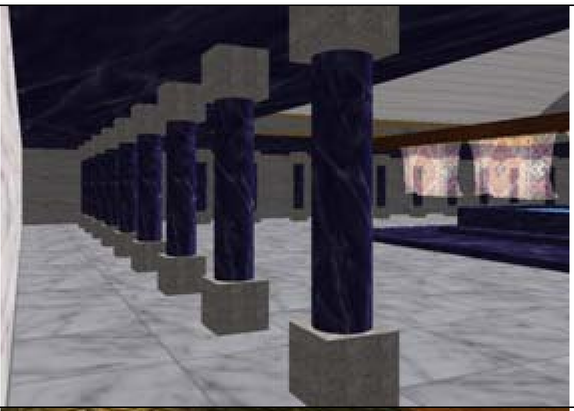
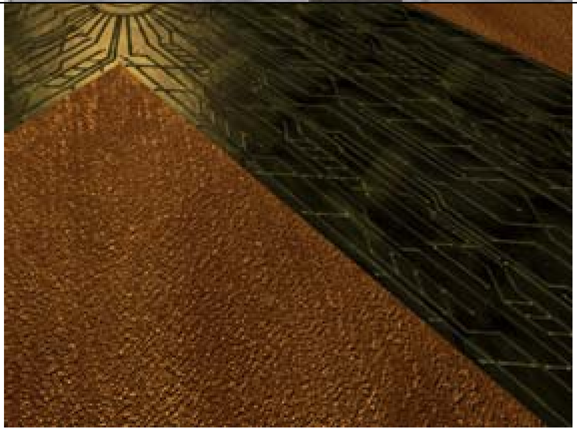
Most Appropriate Algorithm	**Conditions**
Rautenbach et al's (2008) spatial subdivision approach coupled with SSE2 utilisation.	All environmental areas lit using static light sources.
Chan and Durand's (2004) algorithm.	Scenes consisting of eight or less dynamic light sources when high-quality shadows are required and where shadow casting objects are located near the point-of-view.
Shadow mapping.	Scenes consisting of more than two and less than fourteen dynamic light sources and where the shadow casting objects are located a significant distance from the point-of-view.
	Chan and Durant's algorithm will, however, prove a better choice for both close range and distant objects when rendering scenes consisting of fourteen or more dynamic light sources. We will also use Chan and Durrand's algorithm for all scenes consisting of nine or more dynamic light sources when high-quality shadows are required.
McCool's (2000) and Thakur et	The second best choice when dealing with

al's (2003) algorithm.	scenes featuring one to eight light sources and when high-quality shadows are required. We won't be utilising this algorithm, rather opting for Chan and Durand's hybrid approach. The same goes for the classic stencil shadow volume algorithm and Thakur et al's (2003) algorithm.

Table 4.2 Algorithms of choice based on the presented critical analysis.

4.3.2 Shaders

The presented shader evaluation focuses on a number of shader implementations and lighting approaches (please refer to Appendix B and C, respectively, for a background discussion on shaders and lighting as well as various reflection models). These implementations, listed in Table 4.3, are organised into four shader effect quality groups based on each algorithm or technique's standalone processor utilisation and visual effect quality. Algorithms and rendering approaches are grouped in order of increasing complexity. For example, basic directional lighting is a much simpler (and thus less computationally intensive) approach than a lighting model that adds ambient occlusion to a scene.

Grouping/Description	**Rendered Scene Screenshot**
Low Shader Quality Consists of a simple light mapping shader implementation and a shader program enabling basic directional lighting (local illumination).	
Medium Shader Quality Extends the low shader quality grouping with the addition of a normal/bump mapping shader (as discussed in Appendix A, bump- or normal mapping is used for adding depth to pixels and thus creating a lighting-dependent bumpiness to a texture mapped), a shader used for the calculation and rendering of specular highlights (as discussed in Appendix C,	

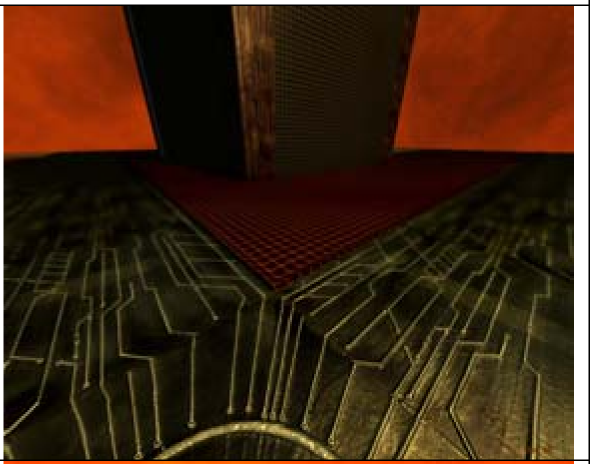
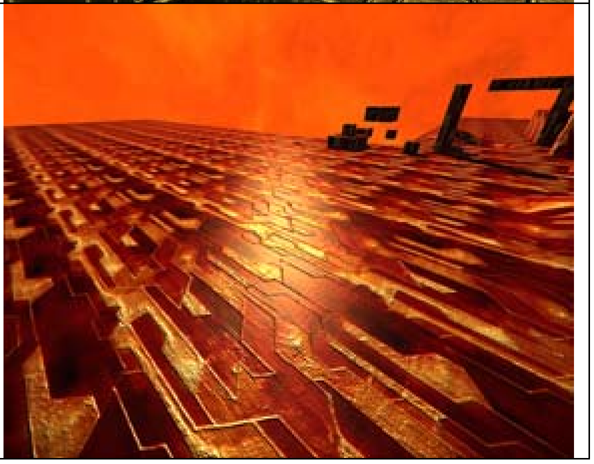
specular reflection is characterized by bright highlights on the surface of an object reflected in the direction of the view vector) and a shader enabling volumetric fog.	
High Shader Quality Similar to the Medium Shader Quality group but replaces light mapping with a detailed lighting model shader that adds a shader for ambient occlusion (as a way to enhance the ambient light term such that shadows and light emission from local features are included).	
Very High Shader Quality Extends the High Shader Quality selection by replacing the previous lighting model with High Dynamic Range Lighting and parallax mapping (an enhancement of bump/normal mapping; "bumpy" textures will have more apparent depth and will thus appear more realistic).	

Table 4.3 Shader effect quality groupings.

As discussed in section 3.5, high dynamic range lighting is the rendering of lighting using more than 256 colour shades for each of the primary colours. Thus, 16 to 32-bit colours per RGB channel are available for use (as opposed to the normal 8) – eliminating luminance and pixel intensity being clamped to a [0, 1] range. This allows the presented rendering engine the display of light sources over 100 000 times brighter than normally possible. HDR lighting results in the full visibility of both very dark and fully lit areas; unlike normal lighting, or *low dynamic range lighting*, where details are hidden in dark scenes when contrasted by a fully lit area. Using this form of lighting generally leads to a more vibrant looking scene. The inclusion of HDR Lighting (which, as shown, may greatly impact the performance of a scene) is controlled through the engine's shader quality scaling (with quality relying on the GPU's computational power).

Please see Figure 4.7 for the observed frames per second performance of each shader quality scaling group as the number of light source vary between 1 and 8. Figure 4.8 shows the same as the light sources increase from 9 to 16.

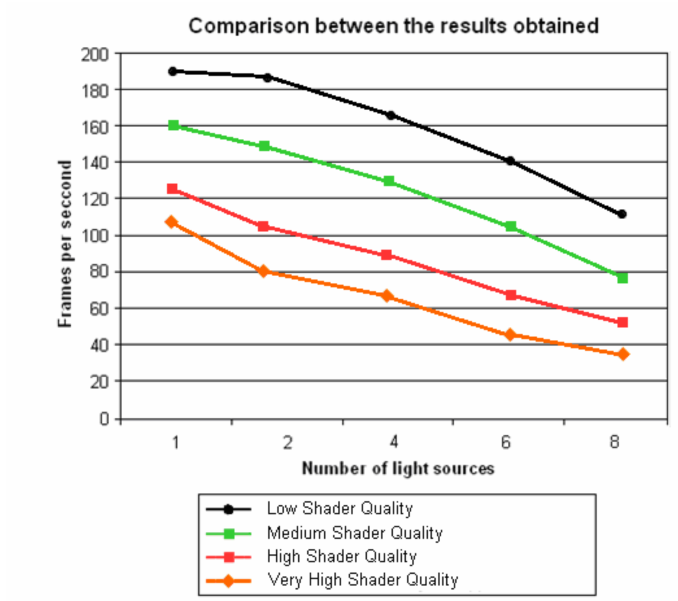


Figure 4.7 Comparison of all the previously listed quality scalings (1-8 light sources).

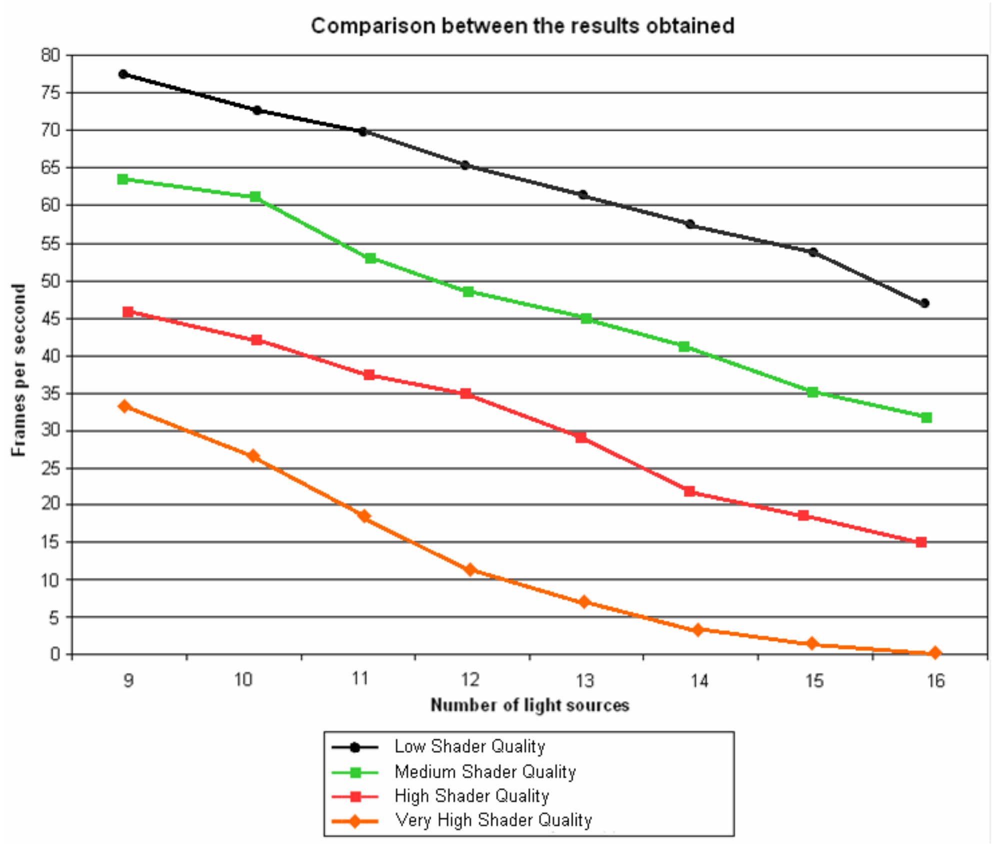


Figure 4.8 Comparison of all the previously listed quality scalings (9-16 light sources).

The first quality grouping is the best performing configuration and perfectly suited as a performance-orientated selection in situations where the GPU is being over-utilised or when faced with limited computational resources. As with the other shader groupings, this combination shows significant performance degradation when more light sources are added. It does, however, outperform all the remaining groups (the only cost being rendering quality).

The medium shader quality grouping performs only slightly worse than the first but given the visual quality benefits inherent to the utilisation of normal mapping, specular highlights and volumetric fog, it is clear that the first quality grouping should only be selected as a last resort effort to free up computational resources.

The third grouping performs relatively well when dealing with scenes featuring one to eight light sources and when high-quality special effects are required. The presented 3D engine will only utilise this shader grouping for scenes consisting of 14 light sources or less (as the FPS performance drops off significantly given further light source additions).

The final grouping gives similar performance figures and will only be utilised for scenes containing less than eight light sources and where the processing resources are available to facilitate HDR lighting and parallax mapping. That said, when there are few light sources, then, on the test system's hardware configuration, the FPS performance for even the very high shader quality grouping is well above the level at which the human eye can perceive any slowdowns. Running the same simulations with the light source count ranging from nine to sixteen shows a rapid decrease in the frames per second performance.

Shader selection is based on the optimisation of the rendering frame rate and rendering quality. The presented rendering engine will thus select the most appropriate shader grouping by taking not only the scene's frames per second performance data into account but also by factoring in the viewer's position in relation to the scene being rendered. The rendering accuracy and detail of distant objects (for instance, distant normal mapping calculations) will carry less weight than those rendered relatively close to the viewer. Table 4.4 summarises the most appropriate shader quality selections based on our algorithmic comparison and scene conditions such as view distance, dynamic/static light conditions and number of light sources.

Most Appropriate Selection	**Conditions**
Low Shader Quality.	The GPU is heavily overburdened (significant slowdowns or FPS drops are observed) and additional computational resources are required by other core rendering elements.
Medium Shader Quality.	The GPU is fully utilised (a GPU running at 100%)

	but no additional computational resources are required (no slowdowns or noticeable FPS drops are observed).
High Shader Quality.	The GPU is not fully utilised and the scene consists of one to eight light sources and high-quality special effects are required but Very High Shader Quality would overburden the GPU.
Very High Shader Quality.	The scene contains less than eight light sources and the computational resources are available to facilitate true HDR lighting and parallax mapping.

Table 4.4 Shader quality selections based on the presented critical analysis.

4.3.3 Local Illumination

As discussed in the previous chapter, the presented rendering engine, in its most basic form, allows for the use of local illumination which, unlike global illumination, only considers the interaction between a light source and object. Local illumination is implemented using the diffuse reflection model, resulting in a uniformly lit scene. An HLSL pixel shader is implemented to calculate the lighting effect on each pixel in our scene.

The evaluation focuses on two basic implementations (divided into two performance-impacting groups, Low and High). Table 4.5 lists these two lighting scaling approaches with Figures 4.9 and 4.10 giving the observed performance of each.

Grouping/Description	**Rendered Scene Screenshot**
Low Local Illumination Limits the number of light sources in an attempt to reduce GPU utilisation.	

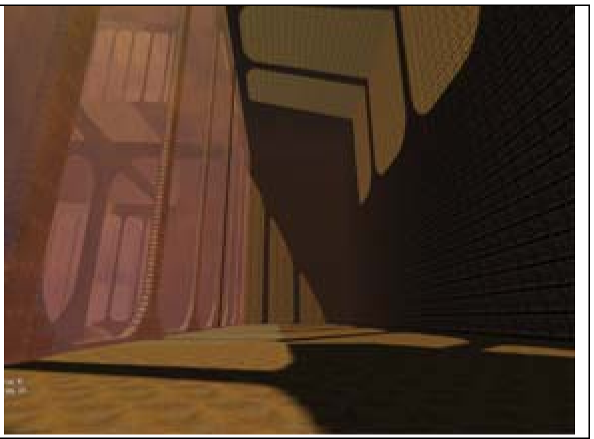
High Local Illumination Lifts the lighting limitation imposed by the low lighting group and occludes local light sources (a technique used to approximate the effect of environment lighting as an attempt to simulate the way light radiates in real life).	

Table 4.5 Lighting quality groupings.

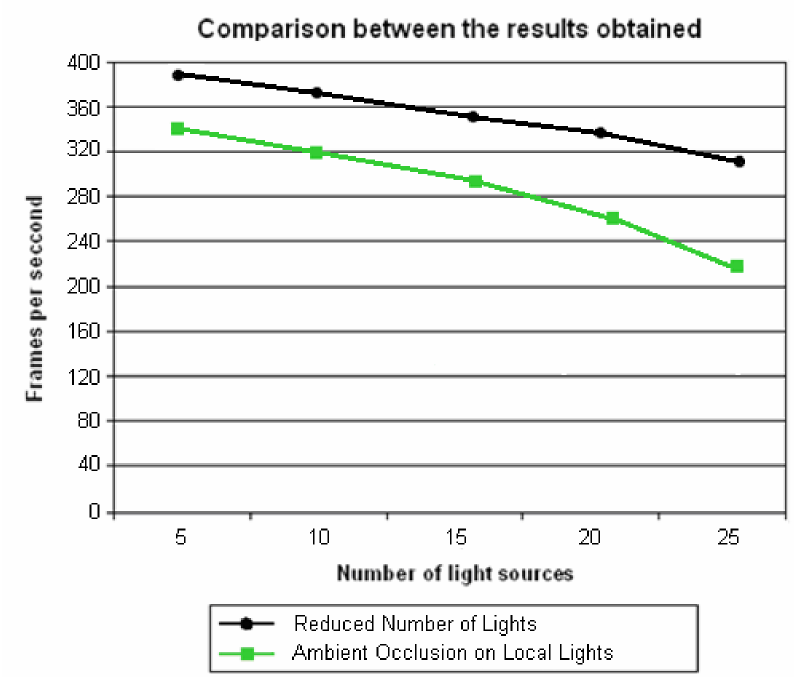


Figure 4.9 Comparison of all the previously listed quality scalings (5-25 light sources).

The first quality grouping, limiting the number of light sources, is perfectly suited as a performance-orientated selection in situations where the GPU is being over-utilised or when faced with limited computational resources (especially when rendering scenes consisting of 25 light sources or more). As with the other quality grouping, basic local illumination shows significant performance degradation as more light sources are added. The cost is not so much the scene's overall rendering quality as it is a limit on the scene's overall atmosphere and ambience (as can be observed by comparing the screenshots given in Table 4.5).

The high lighting quality grouping performs only slightly worse than the first but given the quality benefits inherent to the utilisation of ambient occlusion (as a way to enhance the ambient light term such that shadows and light emission from local features are included) and the relatively close FPS results when compared to basic local illumination, it is clear that the first quality grouping should only be selected as a last resort effort to free up computational resources. Thus, the presented 3D engine will use the high quality lighting grouping, unless the number of light sources increases above 50 (especially taking into account the overall performance impact of additional rendering algorithms and other GPU burdens such as physics processing).

Running the same simulations with the light source count ranging from 55 to 65 shows a rapid decrease in the rendering frame rate. Figure 4.10 shows these results.

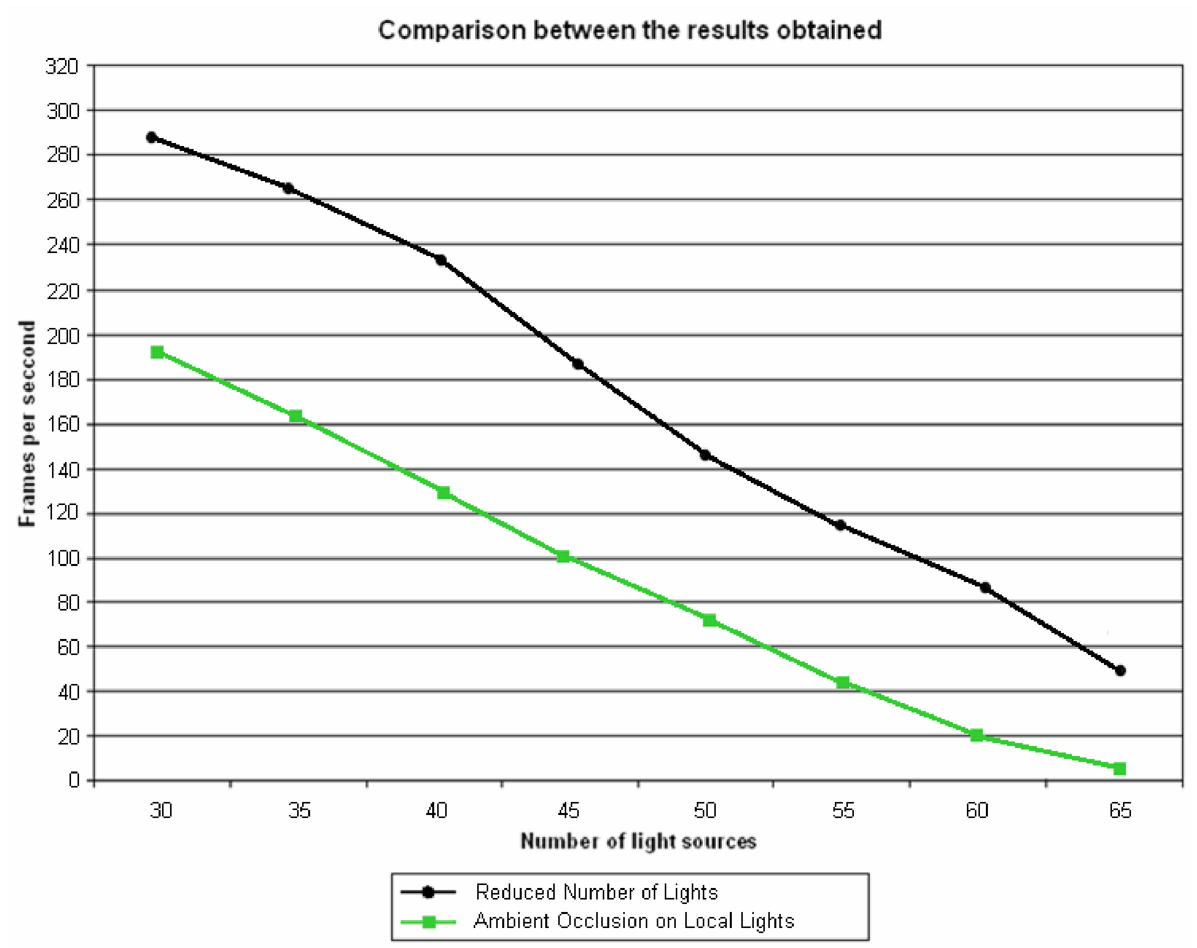


Figure 4.10 Comparison of all the previously listed quality scalings (30-65 light sources)

Lighting quality selection, as with shaders, is based on the optimisation of the rendering frame rate. Table 4.6 gives the most appropriate selection based on the presented algorithmic comparison.

Most Appropriate Selection	Conditions
Low Local Illumination.	The GPU is heavily overburdened and additional computational resources are required by other core rendering elements/the scene contains more than fifty light sources.
High Local Illumination.	The GPU is not fully utilised and the scene consists of fifty light sources or less and high-quality effects are required.

Table 4.6 Local illumination quality selections based on the presented critical analysis.

4.3.4 Reflection and Refraction

The presented rendering environment extends the basic local illumination lighting model by adding reflection and refraction effects to result in more realistic and lifelike images. When computation processing power is not available, the engine will utilise basic environmental mapping which allows us to simulate reflections by mapping real-time computed texture images to the surface of an object. Each texture image used for environmental mapping stores a “snapshot” image of the environment surrounding the mapped object. The engine further supports refractive environmental mapping, the Fresnel effect (Wloka, 2002) and chromatic dispersion resulting in an object’s colour being blended with reflections from its cube map (section 3.4). Thus, when the processing power is available, the presented renderer’s basic reflections can be extended to appear more lifelike.

The presented reflection quality evaluation focuses on a number of reflection and refraction implementations and approaches, specifically basic environmental mapping, CPU-based cube mapping, refractive environmental mapping and the extension of these reflection and refraction algorithms through the addition of the Fresnel effect and chromatic dispersion. Table 4.7 organises these implementation approaches into three reflection/refraction quality grouping: Low, Medium and High.

Grouping/Description	Rendered Scene Screenshot
Low Reflection Quality Supports only GPU-based environmental mapping	

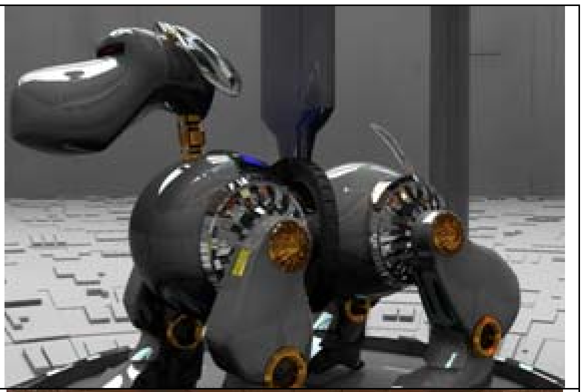
Medium Reflection Quality Moves all environmental mapping computations off to the CPU (thus freeing the GPU in the process).	
High Reflection Quality Replaces basic environmental- or cube mapping with processor-intensive refractive environmental mapping supporting chromatic dispersion and the Fresnel effect.	

Table 4.7 Reflection and refraction quality groupings.

Figures 4.11 and 4.12 give the observed performance of each reflection and refraction quality scaling group.

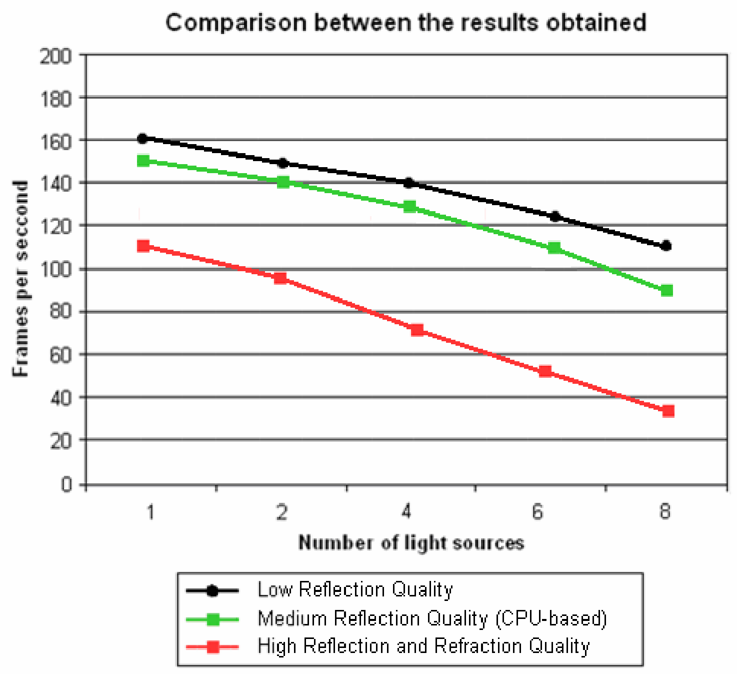


Figure 4.11 Comparison of all the previously listed quality scalings (1-8 light sources).

The first quality grouping, concerned with basic environmental mapping, is the best performing configuration and perfectly suited as a performance-orientated selection in situations where the GPU is being over-utilised or when faced with limited computational resources. As with the other groupings, this combination shows significant performance degradation when more light sources are added. It does, however, outperform all the remaining groups (the only cost being rendering quality).

The medium quality grouping performs only slightly worse than the first but given the fact that cube mapping is being performed on the CPU (thus freeing the GPU to perform other tasks) and that the frame rate is comparable to the same algorithm performed on the GPU, it is clear that CPU-based cube mapping is an excellent alternative to its only slightly better performing GPU-based counterpart.

The third grouping performs relatively well when dealing with scenes featuring one to eight light sources and when high-quality special effects are required. The presented renderer will only be utilising this grouping for scenes consisting of 7 light sources or less and where the processing resources are available to facilitate refractive environmental mapping, the Fresnel effect and chromatic dispersion. Running the same simulations (but with the light source count ranging from nine to sixteen) shows a rapid decrease in our frames per second performance. Figure 4.12 shows these results.

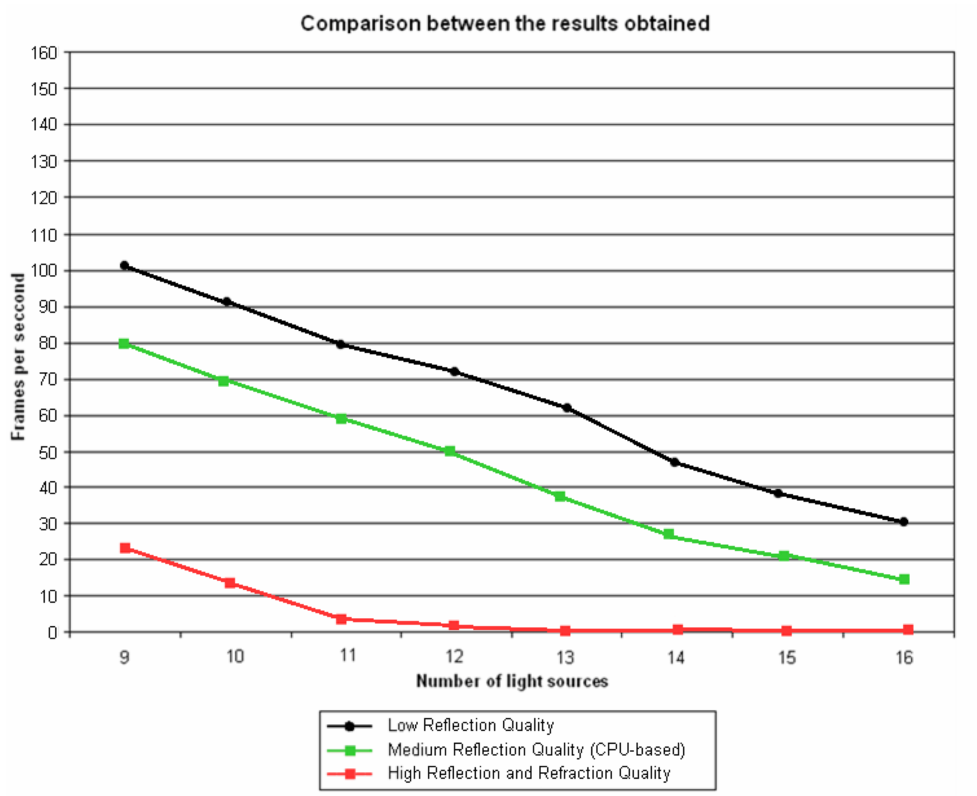


Figure 4.12 Comparison of all the previously listed quality scalings (9-16 light sources)

As with other shader implementations, reflection and refraction selection is based on the optimisation of the rendering frame rate and rendering quality. Also, rendering accuracy and reflection detail of distant objects (for instance, distant object reflections) carries less weight than those rendered relatively close to the viewer. Table 4.8 summarises the algorithms of choice based on the algorithmic comparison and scene conditions such as view distance, dynamic light conditions and number of light sources.

Most Appropriate Selection	**Conditions**
Low Reflection Quality.	The GPU is heavily overburdened, the CPU is fully utilised or cannot be utilised to lighten the GPU load and additional computational resources are required by other core rendering elements.
Medium Reflection Quality.	The GPU is fully utilised, additional computational resources are required and the CPU is not fully utilised and can be utilised to lighten the GPU load.
High Reflection and Refraction Quality.	The GPU is not fully utilised and the scene consists of one to seven light sources and high-quality special effects are required.

Table 4.8 Reflection and Refraction quality selections based on the presented critical analysis.

4.3.5 Physics

The presented rendering environment features not only basic physics simulations but also realistic object interaction based on Newton's Laws, a particle system inheriting from the physics system and realistic object interaction with all objects reacting based on the force exerted and environmental resistance. The engine's physics quality scaling (with quality relying on the GPU and/or CPU's computational power) is organised into performance-impacting selection categories ranging from Low to Very High. Specifically the categories which may be selected are either Off (very basic physics simulation), Low (75% Reduction in Physics Calculations), High (25% Reduction in Physics Calculations) or Very High (No Reduction in Physics Calculations).

The evaluation focuses on a number of physics calculations, specifically object acceleration, force, linear momentum, gravitational pull, projectile simulation through trajectory paths, friction and collision detection. The computational requirements for each of these are now presented with Figures 4.13 (a) and (b) giving the observed performance of each calculation group. However, Appendix E, in addition to section 3.7, can be consulted should background information be needed on the simulation of

Newtonian physics through the use of quantities such as mass, acceleration, velocity, friction, momentum, force, etc.

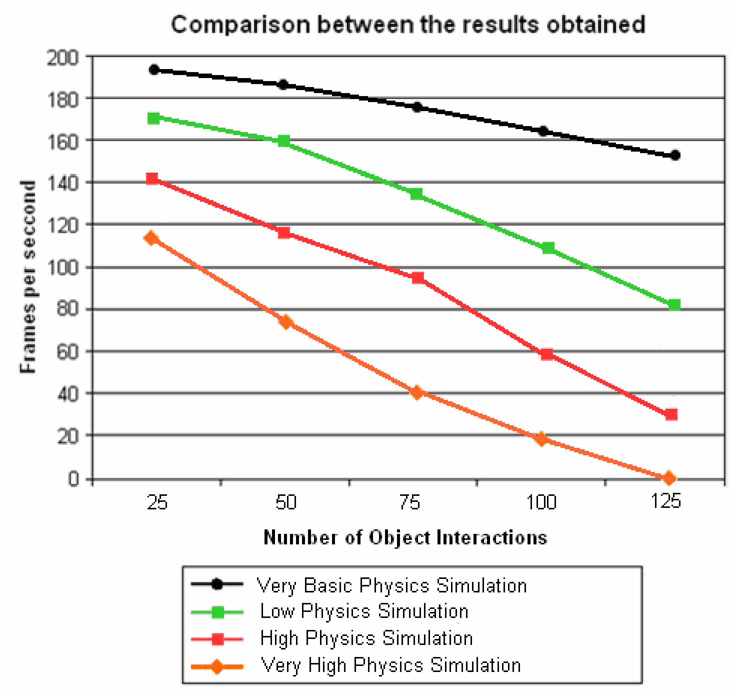


Figure 4.13 (a) Comparison of all the previously listed quality scalings – GPU (25-125 interacting objects).

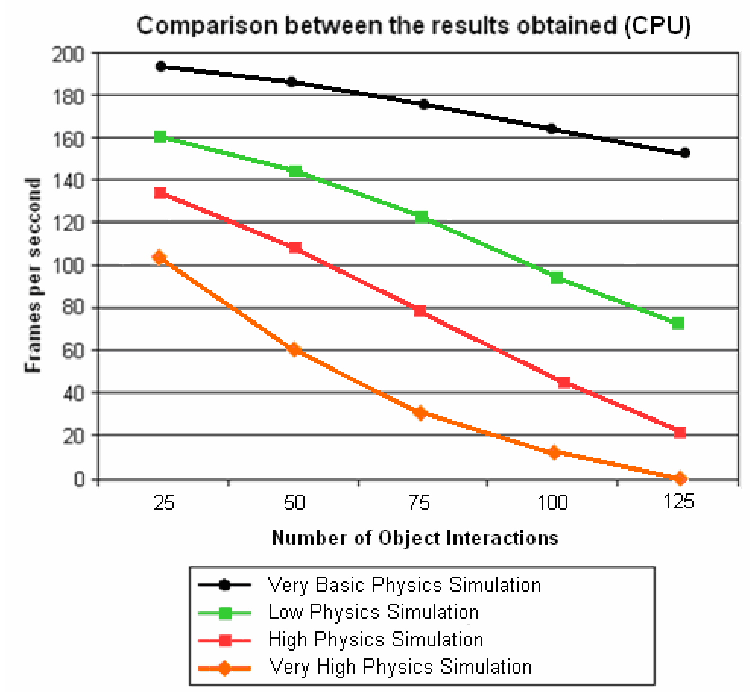


Figure 4.13 (b) Comparison of all the previously listed quality scalings – CPU (25-125 interacting objects).

Physics calculations performed on the GPU utilise NVIDIA's PhysX real-time physics engine (without any multithreading optimisation) while those performed on the CPU are based on the x87 floating point subset of the x86 architecture instruction set. CPU calculations fully utilise SSE multithreaded technology while PhysX calculations are not natively optimised for SSE or multithreading, resulting in situations where a PhysX implementation can be outperformed by well-coded multithreaded CPU physics implementations. There are thus two calculation subsets – the non-SSE PhysX code performed on the GPU and the multithreaded x87 code utilising SSE executed on the CPU. Simply running PhysX code (in software mode) on the CPU leads to significant performance drops (as thread control is to be handled by the developer).

The first quality selection, utilising only simple bounding boxes and basic edge detection and object interaction production rules, is the best performing configuration and perfectly suited as a performance-orientated selection in situations where the GPU and CPU are being over-utilised or when faced with limited computational resources. This selection shows a performance degradation as more objects are added. It does, not surprisingly, outperform all the remaining groups (the only cost being a lack in realism).

The low physics simulation selection performs, as expected, significantly worse than the first but given the quality benefits inherent to the utilisation of Newtonian physics (albeit with a reduction in computational accuracy), it is clear that the first quality selection should only be selected as a last resort effort to free up computational resources.

The third selection performs relatively well when dealing with scenes featuring up to 125 interacting objects and when highly accurate physics calculations are required. The presented rendering engine will only be utilising this grouping when highly accurate physics calculations are not possible. Very High Physics Simulations give proportionally lower performance figures and will only be utilised when the necessary computational resources are not required for graphics processing.

Physics selection is based on the optimisation of the rendering frame rate and rendering quality. The scene's frames per second performance data as well as the viewer's position in relation to the physics simulation are taken into account when selecting the most appropriate physics grouping. The accuracy of distant object simulations (for instance, two distant object colliding) will carry less weight than those rendered relatively close to the viewer. Table 4.9 summarises the groupings of choice based on our algorithmic comparison and scene conditions and number of objects.

Most Appropriate Selection	**Conditions**
Very Basic Physics Simulation.	The GPU is heavily overburdened, the CPU is fully utilised or cannot be utilised to lighten the GPU load and additional computational resources are required by other core rendering elements.
Low Physics Simulation.	The GPU is fully utilised, the CPU is fully utilised or cannot be utilised to lighten the GPU load but no additional computational resources are required.
High Physics Simulation.	The GPU or CPU is not fully utilised and the scene consists of one to 125 objects and high-quality physics are required but Very High Physics Simulation would overburden the GPU and/or GPU.
Very High Physics Simulation.	The necessary computational resources (CPU and/or GPU) are not required for graphics processing and the necessary physics calculations does not cause a noticeable drop in the perceivable smoothness of the scene being rendered.

Table 4.9 Physics quality selections based on the presented critical analysis.

4.3.6 Particle Effects

Particle effects, with the particle system inheriting from the physics system, allows for the simulation of natural phenomena such as fire, smoke, sparks, explosions, dust, trail effects, etc. As discussed in sections 3.7.2 and 3.7.3, the particle system is implemented using three stages, namely, the setup stage, the simulation stage and the rendering stage.

The setup stage involves specification of the particle system's spatial position and area of constraint – parameters controlled by the emitter. The emitter also controls the particle creation rate, that is, the rate at which new particles are injected into the system. Each particle has a specific time to live, after which it is destroyed.

The simulation stage takes care of particle rendering rates, particle spawning position (mostly randomised between some minimum and maximum coordinate range), particle properties (such as particle colour, velocity, etc) and positioning of the emitter. This stage also keeps track of each particle to check whether a specific particle has exceeded its lifetime. Each particle has an initial velocity and is translated based on some sort of physics model or simply by adding velocity to its current spatial position. Collision detection is also possible at this stage but rarely implemented.

Following the simulation state, each particle is rendered as either a coloured point, polygon or as a mesh.

The engine's particle system, based on the rules of physics, uses the following standard equations to calculate each particle's velocity and position:

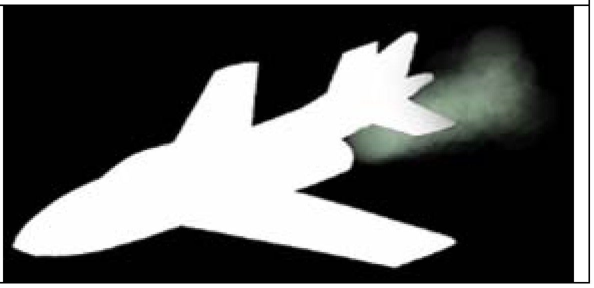
$$V_{new} = V_{old} + a \times t$$

$$Pos_{new} = Pos_{old} + (v_{old} \times t) + (\frac{1}{2} \times a \times t^2),$$

The above given equations factor in the initial motion of the particle, its trajectory and the overall effect of gravity where

- *Pos$_{new}$* is the particle's final position,
- *Pos$_{old}$* its initial position,
- *V$_{new}$* its final velocity,
- *V$_{old}$* its initial velocity,
- *a* the particle's acceleration and
- *t* the change in time.

Using these equations we start by initialising each particle's initial position and velocity. These values will be assigned to a particle when it is generated by the emitter. The quality scaling (with quality of explosions, dust, tread marks, beams, etc relying on the CPU and GPU's computational power) of the rendering engine's particle effects is, as with physics, organised into performance-impacting selections ranging from Low to Very High. Table 4.10 lists these particle effects scaling approaches with Figures 4.14 (a) and (b) giving the mean performance of each calculation group as executed on the CPU and GPU, respectively.

Grouping/Description	**Rendered Scene Screenshot**
Low Particle Simulation 75% reduction in effect quality.	

Medium Particle Simulation 50% reduction in effect quality.	
High Particle Simulation 25% reduction in effect quality.	
Very High Particle Simulation No reduction in effect quality.	

Table 4.10 Particle effects quality groupings.

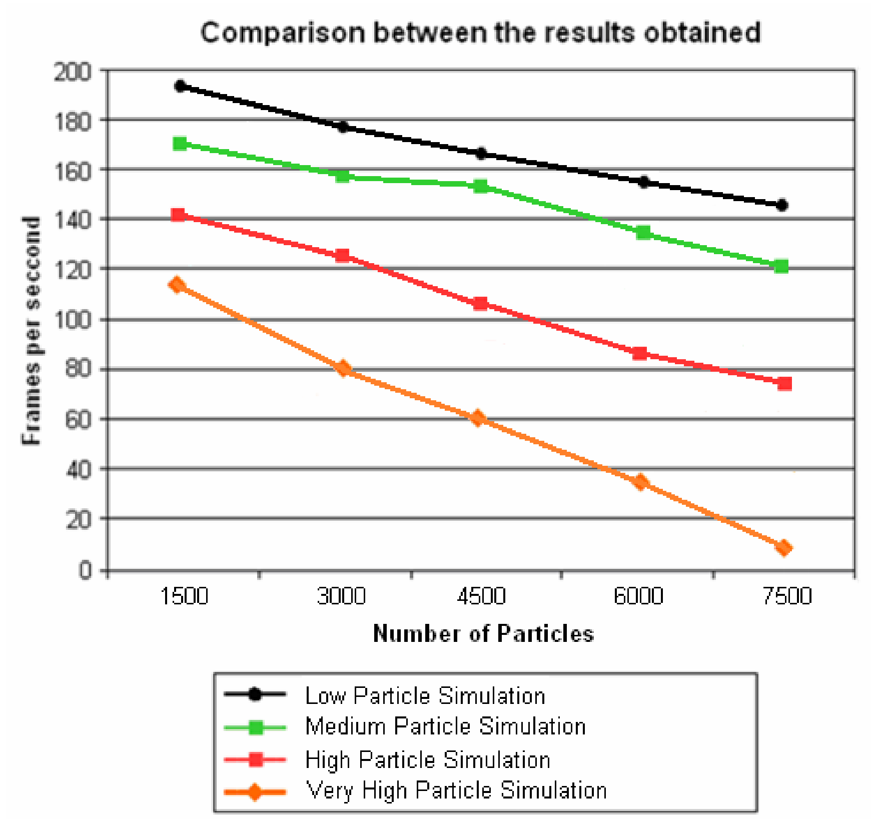


Figure 4.14 (a) Comparison of all the previously listed quality scalings – GPU

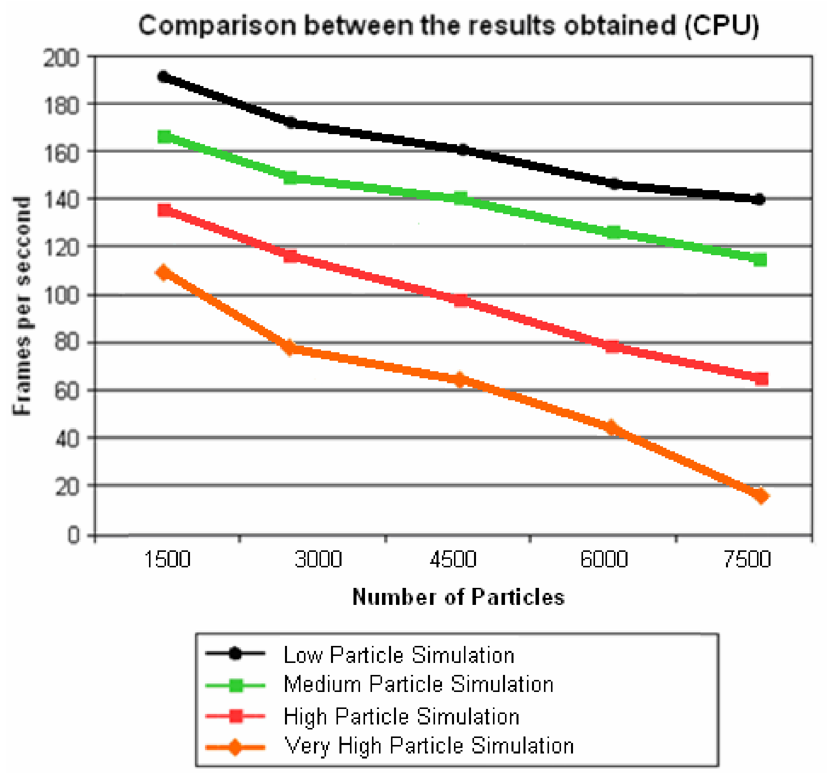


Figure 4.14 (b) Comparison of all the previously listed quality scalings – CPU.

As with physics, particle calculations – velocity and position – performed on the GPU utilises NVIDIA's PhysX real-time physics engine while those performed on the CPU are based on the x87 floating point subset of the x86 architecture instruction set.

The first quality selection, representing a 75% reduction in effect quality, is the best performing configuration and perfectly suited as a performance-orientated selection in situations where the GPU and CPU are being over-utilised or when faced with limited computational resources. As with the other performance selections, this combination shows a performance degradation as the number of particles increase. Unsurprisingly, it outperforms all the remaining selections (the only cost being a lack in visual quality and realism).

The medium particle simulation selection performs, as expected, slightly worse than the first but given the quality benefits inherent to the utilisation of more accurate Newtonian physics (albeit with a reduction in computational accuracy), it is clear that the first quality selection should only be selected as a last resort effort to free up computational resources.

The third selection performs relatively well when dealing with effects consisting of 1500 to 7500 particles and when highly accurate physics calculations are required. The presented 3D engine will only be utilising this selection when very high particle simulations are not possible. The final selection gives proportionally lower performance figures and will only be utilised when the necessary computational resources are not required for graphics processing.

Table 4.11 summarises the algorithms of choice based on our algorithmic comparison, scene conditions and number of particles.

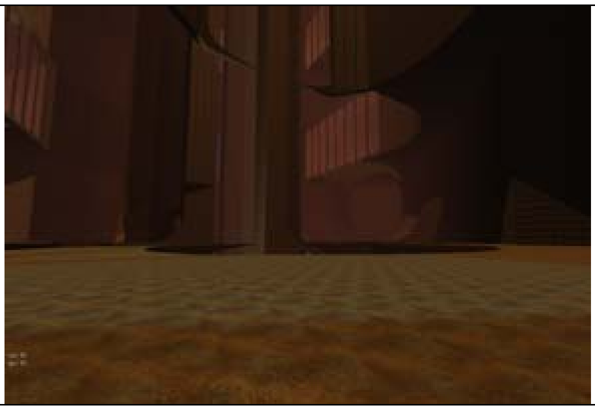
Most Appropriate Selection	**Conditions**
Low Particle Simulation.	The GPU is heavily overburdened, the CPU is fully utilised or cannot be utilised to lighten the GPU load and additional computational resources are required by other core rendering elements.
Medium Particle Simulation.	The GPU is fully utilised, the CPU is fully utilised or cannot be utilised to lighten the GPU load but no additional computational resources are required.
High Particle Simulation.	The GPU or CPU is not fully utilised and the effect consists of 1500 to 7500 particles and high-quality physics are required but Very High Particle Simulation would overburden the GPU and/or GPU.

Very High Particle Simulation.	The necessary computational resources (CPU and/or GPU) are not required for graphics processing and the necessary Newtonian calculations does not cause a noticeable drop in the perceivable smoothness of the scene being rendered.

Table 4.11 Effect quality selections based on our critical analysis.

4.3.7 Post-Processing

The presented rendering engine uses post-processing or quality-improvement image processing (through the use of pixel shaders) to add additional effects such as bloom lighting (the effect of producing light fringes around ultra-bright objects – an object with a bright light behind it will be "overlapped" by the light and thus appear more lifelike), motion blur (the streaking of rapid moving objects), ambient occlusion (the global effect of approximating the radiation of light by the casting of rays in every direction from an object's surface), depth of field (the variance in sharpness between the nearest and farthest objects in a scene), displacement mapping (as an alternative to normal mapping – used to displace surface points; giving surfaces great depth and detail) and halo effects (artificial glow added to light "emitting" objects such as light bulbs or a glowing red button). The engine's post-processing quality scaling relies on the GPU's computational power and consists of three quality groups: Low, Medium and High. Table 4.12 lists these scaling approaches with Figure 4.15 showing the mean performance of each post-processing quality scaling group.

Grouping/Description	**Rendered Scene Screenshot**
Low Post-Processing Adds Minimal Intensity Bloom Effects and Displacement Mapping to the rendered scene.	

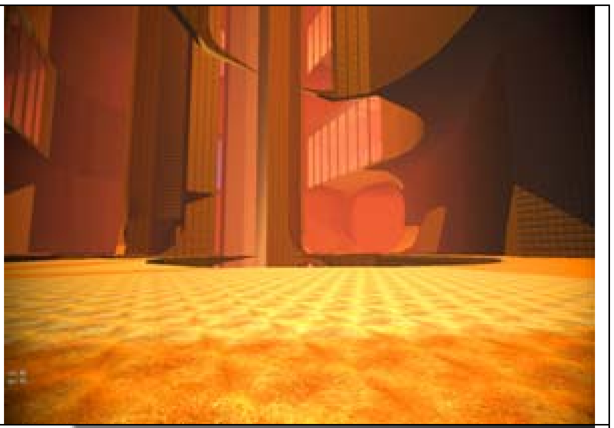
Medium Post-Processing Adds Ambient Occlusion (along with High Intensity Bloom Effects and Displacement Mapping).	
High Post-Processing Adds Depth of Field and Halo Effects to the rendered scene.	

Table 4.12 Post-Processing effects quality groupings.

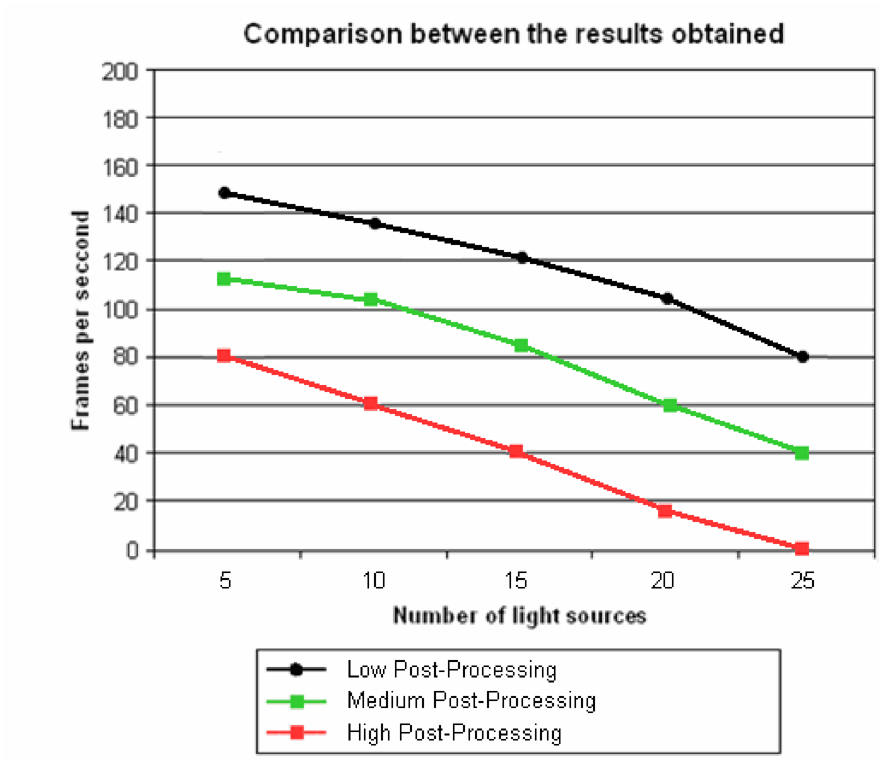


Figure 4.15 Comparison of all the previously listed quality scalings (5-25 light sources).

The first quality grouping, consisting of displacement mapping and bloom effects, is the best performing configuration and is well suited as a performance-orientated selection in situations where the GPU is being over-utilised or when faced with limited computational resources. As with the other groupings, this combination shows significant performance degradation as more light sources are added. It does, however, outperform all the remaining groups (the cost is not so much a loss in rendering detail as it is one where there are “less” special effects than when using the others; this selection simply excludes ambient occlusion, depth of field and halo effects).

The medium post-processing quality grouping performs somewhat worse than the first but given the global lighting benefits inherent to the utilisation of ambient occlusion, it is clear that the first quality grouping should only be selected as a last resort effort to free up computational resources.

The final grouping performs relatively well when dealing with scenes featuring one to ten light sources and when high-quality special effects are required. This grouping will be utilised for scenes consisting of 15 light sources or less and where the processing resources are available to facilitate ambient occlusion, depth of field, displacement mapping and halo and bloom effects.

The presented rendering engine selects the most appropriate grouping given the processing power available. Also, the rendering accuracy and detail of distant objects (for instance, distant displacement mapping or bloom calculations) carry less weight than those rendered relatively close to the viewer. Table 4.13 summarises the algorithms of choice based on our algorithmic comparison and scene conditions such as view distance, dynamic/static light conditions and number of light sources.

Most Appropriate Selection	**Conditions**
Low Post-Processing.	The GPU is heavily overburdened and additional computational resources are required by other core rendering elements.
Medium Post-Processing.	The GPU is fully utilised but no additional computational resources are required.
Very High Post-Processing.	The scene contains less than fifteen light sources and the computational resources are available to facilitate ambient occlusion, depth of field, displacement mapping and halo and bloom effects.

Table 4.13 Post-Processing quality selections based on our critical analysis.

4.4 Summary

In the chapter we started by presenting a benchmarking mechanism and a set of criteria for the evaluation of rendering algorithms and techniques. The given evaluation criteria were selected with the aim of assessing the relationship between rendering quality and performance – in turn allowing for, where applicable, the isolation of key algorithmic weaknesses and possible bottleneck areas.

Specific shadow algorithms benchmarked and analysed include: the basic stencil shadow volume algorithm, the basic hardware shadow mapping algorithm, McCool's shadow volume reconstruction using depth maps, Chan and Durand's hybrid algorithm for the efficient rendering of hard-edged shadows, Thakur el al's algorithm based on the elimination of various shadow volume testing phases and Rautenbach et al's algorithm based on shadow volumes, spatial subdivision and instruction set utilisation.

The subsequent shader evaluation, in turn, focused on a number of shader implementations and lighting approaches, after which two local illumination configurations were investigated. The first of these limiting the number of light sources in an attempt to reduce GPU utilisation with the second lifting this limitation while occluding local light sources (a technique used to approximate the effect of environment lighting as an attempt to simulate the way light radiates in real life). This evaluation was later extended with the inclusion of HDR lighting.

Next the evaluation focused on a number of reflection and refraction implementations and approaches, specifically: basic environmental mapping, CPU-based cube mapping, refractive environmental mapping and the extension of these reflection and refraction algorithms through the addition of the Fresnel effect and chromatic dispersion.

The chapter then shifted focus to the evaluation of a number of physics calculations such as object acceleration, force, linear momentum, gravitational pull, projectile simulation through trajectory paths, friction and collision detection followed by the benchmarking of our engine's dynamically allocated particle generator.

Chapter 4 concluded with the performance analysis of a number of post-processing shader implementations and lighting approaches, specifically displacement mapping, bloom effects, ambient occlusion, depth of field and halo effects.

Chapter 5

An Empirically Derived System for Distributed Rendering

Chapter 5 presents our empirically derived system for distributed rendering. This analysis highlights not only the performance benefits inherent to the utilisation of this system, but also the practicality of such an implementation.

In this chapter we will investigate:

- Dynamic algorithm selection
- Rules for selection of rendering algorithms
- Fuzzy rules for selection of the most appropriate rendering algorithm
- Construction of the algorithm selection mechanism
- Results obtained from our benchmarking environment

5.1 Introduction

The presented real-time rendering engine continuously analyses a dataset to determine the best solution to a given rendering problem – as in, the best algorithm or shader to use for a specific scene or a specific object in a scene. This selection system consists of an empirically ascertained dataset (containing the previously obtained algorithmic performance data), a collection of rules to analyse the data and information of various elements pertaining to the scene currently being rendered.

The rendering engine uses a selection engine to control the real-time selection of rendering algorithms and, when performing cube mapping or physics calculations, to effectively distribute processing between the CPU and GPU. The knowledge base of this engine, consisting of production rules, is derived from experimental results obtained through the critical analysis of numerous real-time rendering algorithms, as discussed in Chapter 4. These production rules are used by an inference engine which, in turn, is tasked with the selection of the most appropriate algorithm based on certain properties of the scene being rendered. For instance, the presented system could contain the following production rule:

> if there are a lot of light sources in a scene and the scene has a high geometric complexity, then enable a hybrid stencil shadow volume/shadow mapping algorithm.

The notions “a lot of light sources” and “high geometric complexity” are not quantitative facts. Fuzzy logic provides a solution to this problem by assigning quantitative values and/or ranges to these concepts (Salton, 1987). The concepts “a lot of light sources” and “high geometric complexity” can also be combined into the new one “overly complex”, resulting in a new production rule. The presented engine combines production rules with fuzzy logic to explicitly symbolise data. This is followed by the selection of the most appropriate rendering algorithm.

The next section presents this selection engine implementation in detail. Following this, a critical analysis of the empirically derived system for the high-speed rendering of complex 3D environments is performed. This analysis will convey not only the performance benefits inherent in the utilisation of this system, but also the practicality of such an implementation.

5.2 The Selection Engine and the Dynamic Selection and Allocation of Algorithms

The empirically derived system for high-speed rendering consists of a fuzzy logic based selection engine and several rendering algorithms and approaches. The selection engine controls, as mentioned, the selection and allocation of these algorithms by correlating the properties of the scene being rendered with the previously obtained algorithmic performance data.

The selection engine consists of the following modules:

- An inference engine.
- A fact database.
- A knowledge base.
- An explanation/debugging system.

The selection engine's knowledge base consists of experimental results obtained through the critical analysis of numerous real-time rendering algorithms. The inference engine is in turn used to select the most appropriate algorithm based on certain properties of the scene being rendered. The knowledge base is nothing more than a database of rules. These rules symbolise the stored knowledge. The fact database embodies the selection engine inputs (properties and performance statistics of the scene being rendered) which are subsequently used to make decisions and/or to take certain actions. The inference engine makes the actual decision by combining these selection engine rules and facts. The explanation system, implemented only in skeletal form should future developers require debugging information, generates information about the manner in which a decision was made. Figure 5.1 illustrates the architecture of the presented selection engine.

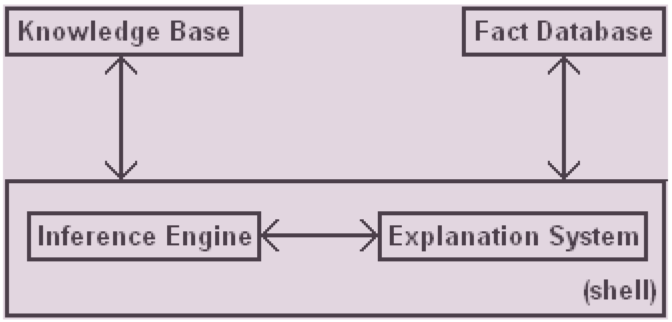


Fig 5.1 Architecture of the selection engine.

The selection engine's inference engine and explanation system are contained within a "shell" written for this study. The knowledge base and fact database are connected to this shell in a plugin-like fashion. The selection engine shell is used to define a generic algorithm selection system, with the selection engine's functionality controlled by the connected fact database and knowledge base.

The selection engine implementation utilises a forward chaining strategy to determine results from a collection of rules and facts. The process basically starts by reading the selection engine inputs from the fact database followed by a comparison between the read inputs and the rules within the rule database. Now, if an input fact matches all the antecedents of a rule, then the rule is triggered with its conclusion added to the fact database.

As mentioned in Chapter 4, the selection and/or allocation of algorithms is based on the continuous optimisation of the rendering frame rate and overall rendering quality. The implemented selection engine will thus select the most appropriate algorithms by taking not only the scene's frames per second performance data (dynamically changing as one moves through the scene) into account but also by factoring in the viewer's position in relation to the object or effect being rendered or calculated. The rendering accuracy and detail of distant objects or effects will thus carry less weight than those rendered relatively close to the viewer. The next section presents the dynamic selection and allocation (where applicable) of the algorithms and rendering approaches discussed in Chapter 3.

5.2.1 Shadows

The following rules can be defined for selecting the most appropriate shadow rendering algorithm (these rules are derived from the algorithmic comparison given in Chapter 4 – Table 4.4 summarises the shadow algorithms of choice based on this algorithmic comparison as well as scene conditions such as view distance, dynamic/static light conditions and number of light sources):

<u>Rule #1</u>
<u>If</u> the environment/sub-environment consists of only static light sources,
<u>Then</u> render all shadows via Rautenbach et al's spatial subdivision/SSE2 algorithm.

<u>Rule #2</u>
<u>If</u> the scene consists of eight or less dynamic light sources,
<u>And</u> high-quality shadows are required (shadow casting objects are located near the point-of-view),
<u>Then</u> render all shadows via Chan and Durand's algorithm.

<u>Rule #3</u>
<u>If</u> the scene consists of more than two and less than fourteen dynamic light sources,
<u>And</u> low-quality shadows are required (shadow casting objects are located a significant distance from the point-of-view),
<u>Then</u> render all shadows via the basic Shadow mapping algorithm.

<u>Rule #4</u>
<u>If</u> the scene consists of fourteen or more dynamic light sources,

And either low- or high-quality shadows are required (for both close range and distant objects),
Then render all shadows via Chan and Durand's algorithm.

Rule #5
If the scene consists of nine or more dynamic light sources,
And high-quality shadows are required (shadow casting objects are located near the point-of-view),
Then render all shadows via Chan and Durand's algorithm.

An interesting aspect of the presented selection engine implementation is its fuzzy logic-based nature. As a fuzzy logic based selection engine, it utilises a set of linguistic variables (related to the problem) and several membership functions. Fuzzy rules are derived from these variables as well as the knowledge base. These rules are applied by means of Mamdani fuzzy inference. Mamdani inference applies a set of fuzzy rules on a set of traditional precise inputs to obtain a precise output value (such as an action recommendation).

The presented fuzzy logic based selection engine will thus contain the following redefined, "fuzzified" rules for selection of the most appropriate shadow rendering algorithm (these rules are screen resolution independent, lower resolutions will simply imply faster overall graphics performance with the shadow generation phases remaining consistent):

Rule #1
If the environment/sub-environment consists of **stationary light sources**,
Then render all shadows via Rautenbach et al's spatial subdivision/SSE2 algorithm.

Rule #2
If the scene consists of **an average** number of dynamic light sources,
And high-quality shadows are required (shadow casting objects are located near the point-of-view),
Then render all shadows via Chan and Durand's algorithm.

Rule #3
If the scene consists of **few or and less than an above average** number of dynamic light sources,
And low-quality shadows are required (shadow casting objects are located a significant distance from the point-of-view),
Then render all shadows via the basic Shadow mapping algorithm.

Rule #4

If the scene consists of **many** dynamic light sources,
And either low- or high-quality shadows are required (for both close range and distant objects),
Then render all shadows via Chan and Durand's algorithm.

Rule #5
If the scene consists of **an average or greater than average** number of dynamic light sources,
And high-quality shadows are required (shadow casting objects are located near the point-of-view),
Then render all shadows via Chan and Durand's algorithm.

Mamdani inference is used to "fuzzify" all precise input values via the definition of fuzzy sets. Here we assume a representation of the number of light sources through the range [0, 20], the nature of a scene's light sources via the values 1 for dynamic and 0 for static and the distance from the viewer via the range [0, 90] (in world units). The presented implementation also defines the following linguistic variables: `Stationary, Dynamic, Average, Few` and `Many`. The system is thus based on Mamdani inference to apply a set of fuzzy rules on a set of traditional precise inputs to obtain a precise output value, specifically an action recommendation (the shadow algorithm to utilise).

The presented Mamdani implementation will thus load the critical analysis performance data, read the pre-programmed fuzzy sets and rules, associate the observed data with the fuzzy sets, run through each case for each and every fuzzy rule, calculate the rule-based fuzzy values, combine the calculated fuzzy values and finally calculate an exact value from the set of fuzzy values. For a thorough evaluation of these rules, please see the MSc dissertation, An Empirically Derived System for High-Speed Shadow Rendering (2008).

5.2.2 Shaders

As in the previous section, we can define several rules for the selection of the most appropriate shader quality scaling (section 4.3.2 presents the shader comparison, based on scene conditions such as view distance and the number of light sources, upon which these rules are based):

Rule #1
If the GPU is heavily overburdened,
And additional computational resources are required by other core rendering elements
Then render the scene using the Low Shader Quality grouping.

<u>Rule #2</u>
<u>If</u> the GPU is fully utilised
<u>And</u> no additional computational resources are required
<u>Then</u> render the scene using the Medium Shader Quality grouping.

<u>Rule #3</u>
<u>If</u> the GPU is not fully utilised,
<u>And</u> the scene consists of a few or less than a below average number of light sources,
<u>And</u> high-quality special effects are required,
<u>And</u> Very High Shader Quality would overburden the GPU,
<u>Then</u> render the scene using the High Shader Quality grouping.

<u>Rule #4</u>
<u>If</u> the scene contains a below average number of light sources
<u>And</u> the computational resources are available to facilitate true HDR lighting, translucent shadows, parallax mapping and volumetric materials,
<u>Then</u> render the scene using the Very High Shader Quality grouping.

5.2.3 Local Illumination

The following fuzzy rules, based on the comparison given in section 4.3.3, deal with the dynamic selection of the most appropriate local illumination quality approach:

<u>Rule #1</u>
<u>If</u> the GPU is heavily overburdened,
<u>And</u> additional computational resources are required by other core rendering elements
<u>Or</u> the scene contains a great number of light sources
<u>Then</u> render the scene using the Low Local Illumination grouping.

<u>Rule #2</u>
<u>If</u> the GPU is not fully utilised,
<u>And</u> the scene contains less than a very high number of light sources,
<u>And</u> high-quality special effects are required,
<u>Then</u> render the scene using the High Local Illumination grouping.

5.2.4 Reflection and Refraction

The most appropriate reflection and refraction quality approaches, as presented in section 4.3.4, can now selected in real-time using the following fuzzy rules:

Rule #1

If the GPU is heavily overburdened,
And the CPU is fully utilised
Or cannot be utilised to lighten the GPU load,
And additional computational resources are required by other core rendering elements
Then render the scene using the Low Reflection Quality grouping.

Rule #2

If the GPU is fully utilised,
And additional computational resources are required,
And the CPU is not fully utilised,
And can be utilised to lighten the GPU load,
Then render the scene using the Medium Reflection Quality grouping.

Rule #3

If the GPU is not fully utilised,
And the scene consists of a few or less than a below average number of light sources,
And high-quality special effects are required,
Then render the scene using the High Reflection and Refraction Quality grouping.

5.2.5 Physics

The following fuzzy rules control the selection of the most appropriate physics simulation approach based on the presented algorithmic comparison (section 4.3.5):

Rule #1

If the GPU is heavily overburdened,
And the CPU is fully utilised
Or cannot be utilised to lighten the GPU load,
And additional computational resources are required by other core rendering elements
Then implement physics using the Very Basic Physics Simulation grouping.

Rule #2

If the GPU is fully utilised Or If the CPU is fully utilised *Or* cannot be utilised to lighten the GPU load,
And no additional computational resources are required,
Then implement physics using the Low Physics Simulation grouping.

Rule #3
If the GPU is not fully utilised Or If the CPU is not fully utilised
And the scene consists of less than an above average number of objects,
And high-quality physics are required,
And Very High Physics Simulation would overburden the GPU and/or GPU,
Then implement physics using the High Physics Simulation grouping.

Rule #4
If the necessary computational resources (CPU and/or GPU) are not required for graphics processing,
And the necessary physics calculations does not cause a noticeable drop in the perceivable smoothness of the scene being rendered,
Then render the scene using the Very High Physics Simulation grouping.

5.2.6 Particle Effects

The most appropriate particle simulation selection (controlled using scene conditions and number of particles, as noted in section 4.3.6) is determined using the following set of rules:

Rule #1
If the GPU is heavily overburdened,
And the CPU is fully utilised
Or cannot be utilised to lighten the GPU load,
And additional computational resources are required by other core rendering elements
Then implement particle effects using the Low Particle Simulation grouping.

Rule #2
If the GPU is fully utilised
And the CPU is fully utilised
Or cannot be utilised to lighten the GPU load,
And no additional computational resources are required,
Then implement particle effects using the Medium Particle Simulation grouping.

Rule #3
If the GPU is not fully utilised Or If the CPU is not fully utilised
And the effect consists of a medium to high number of particles,
And high-quality physics are required,

And Very High Particle Simulation would overburden the GPU and/or GPU,
Then implement physics using the High Physics Simulation grouping.

Rule #4
If the necessary computational resources (CPU and/or GPU) are not required for graphics processing,
And the necessary Newtonian calculations does not cause a noticeable drop in the perceivable smoothness of the scene being rendered,
Then render the scene using the Very High Particle Simulation grouping.

5.2.7 Post-Processing

Table 4.13 summarises the algorithms of choice based on our algorithmic comparison and scene conditions such as view distance, dynamic/static light conditions and number of light sources. Further defining our selection engine, we can create the following fuzzy rules for selection of the most appropriate post-processing quality approach:

Rule #1
If the GPU is heavily overburdened,
And additional computational resources are required by other core rendering elements
Then render the scene using the Low Post-Processing Quality grouping.

Rule #2
If the GPU is fully utilised
And no additional computational resources are required
Then render the scene using the Medium Post-Processing Quality grouping.

Rule #3
If the GPU is not fully utilised,
And the computational resources are available to facilitate ambient occlusion, depth of field, displacement mapping and halo and bloom effects
And the scene consists of a less than average number of light sources,
Then render the scene using the Very High Post-Processing Quality grouping.

5.3 Construction of the Algorithm Selection Mechanism

The performance data gathered during the previously discussed critical analysis allows for the construction of a fuzzy logic-based selection and allocation system. This system, as mentioned, controls the real-time selection of rendering algorithms and quality groupings based on environmental conditions. The gathered data (algorithm, shader and rendering performance) is stored in a comma-delimited format with the rendering engine loading it into memory via the in-game loop (section 2.2) upon execution. Each implemented algorithm/rendering approach is, in turn, loaded into the engine via a dynamic link library. DLLs are based on Microsoft's shared library concept and can contain source code, data and resources. These libraries are generally loaded at runtime, a process referred to as run-time dynamic linking – thus allowing us to replace or change DLLs without recompiling the main executable. For example, the shadow rendering DLL contains the implementation details of the basic stencil shadow volume algorithm, the basic hardware shadow mapping algorithm, McCool's shadow volume reconstruction using depth maps, Eric Chan and Frédo Durand's hybrid algorithm for the efficient rendering of hard-edged shadows, Thakur et al's algorithm based on the elimination of various shadow volume testing phases and Rautenbach et al's algorithm based on shadow volumes, spatial subdivision and instruction set utilisation.

CPU utilisation monitoring is performed using Intel's CPUUsage class (Intel, 2010). This class, wrapping Microsoft's Performance Data Helper (PDH) API (used to collect performance data of various performance counters or system instances), provides an interface for the calculation of maximum, minimum and average CPU utilisation over a period of time. As stated by Intel (2010), CPU utilisation is a key metric for optimisation, performance analysis, and workload evaluation. However, the built-in Windows facilities for tracking CPU utilisation provide limited flexibility. The CPUUsage class attempts to alleviate this issue by providing a simple interface that can be used to programmatically track CPU percentage. The level of control provided by the CPUUsage class allows virtually unlimited CPU utilisation monitoring options for the application developer.

NVIDIA's PerfKit (and the PerfSDK API) is, in turn, used by the rendering engine to access the physical GPU hardware counters and GPU usage data in real-time. The NVPerfKit is actually a collection of performance monitoring, debugging and profiling utilities focused on accessing the low-level performance indicating components of the graphics driver and the GPU itself (assuming an NVIDIA GPU is being used). These low-level components are known as performance counters. They give information on the application's overall frames per second rendering, the video memory used in MB, the graphics driver's sleep time, the polygon count, etc (NVIDIA, 2011). Using the NVPerfKit, we are thus able to profile the rendering engine in terms of its GPU, driver and memory usage. A useful component of NVPerfKit is called PerfHUD, a real-time Direct3D and OpenGL application profiler that generates its output in the form of a heads-up display (shown in Figure 5.2).

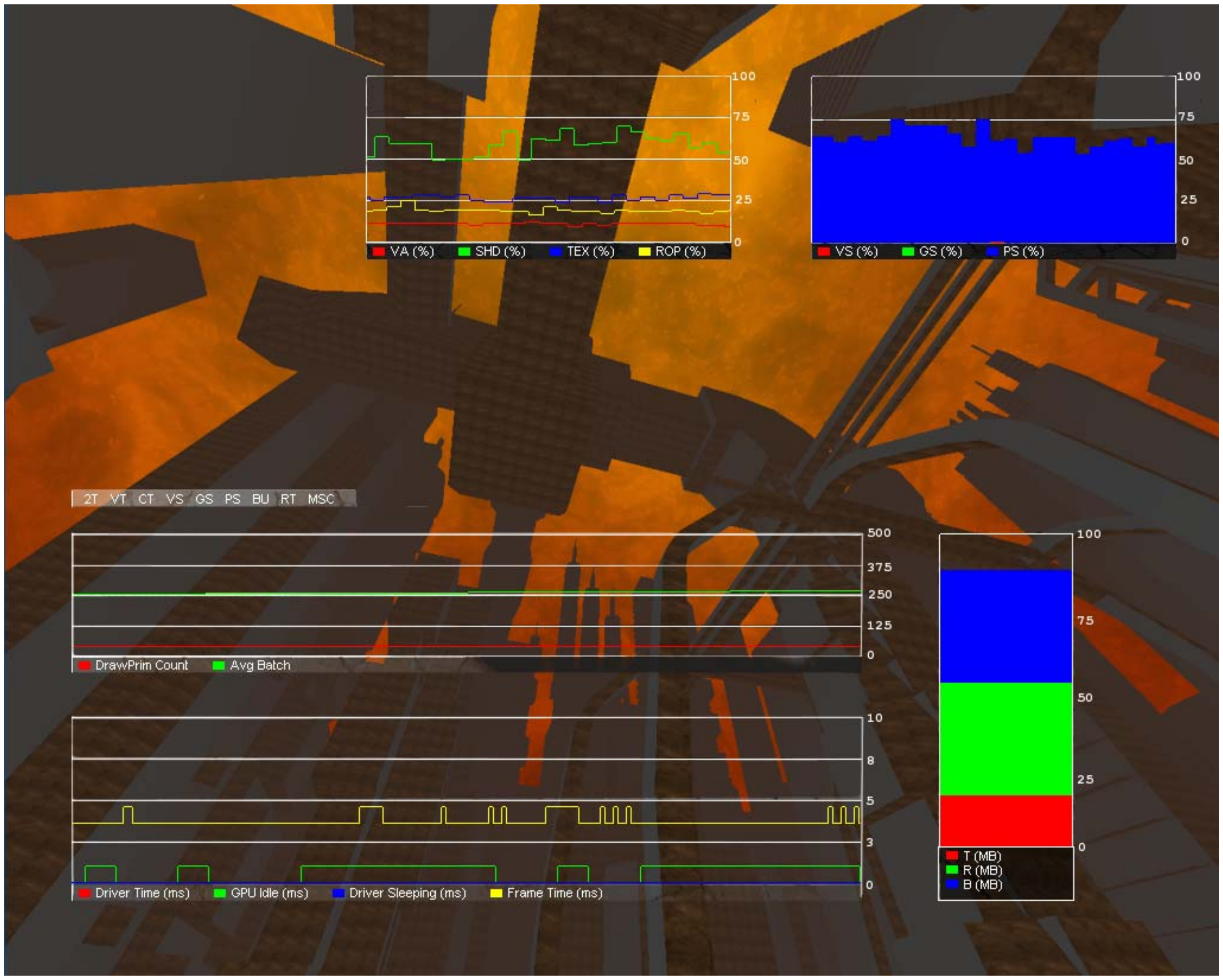


Figure 5.2 Nvidia's PerfHUD.

5.4 Results

By dynamically cycling through algorithms and quality groupings to compensate for performance-impacting changes in the presented rendering environment, we are able to bridge an existing gap between quality and high-speed rendering. The performance gains inherent in this system's use, when compared to traditional implementations, is subsequently highlighted.

We now describe the behaviour of the rendering engine when subjected to different scenarios, each performed independently of one another, that were designed to test its transition behaviour in respect of the various transition rules described above. The collective overall effect is a highly optimised rendering engine (the accompanying CD contains a high-definition video showcasing the rendering engine and the combined overall effect of dynamic quality selection and process allocation). Figure 5.3 shows a collage of the rendering engine in action.

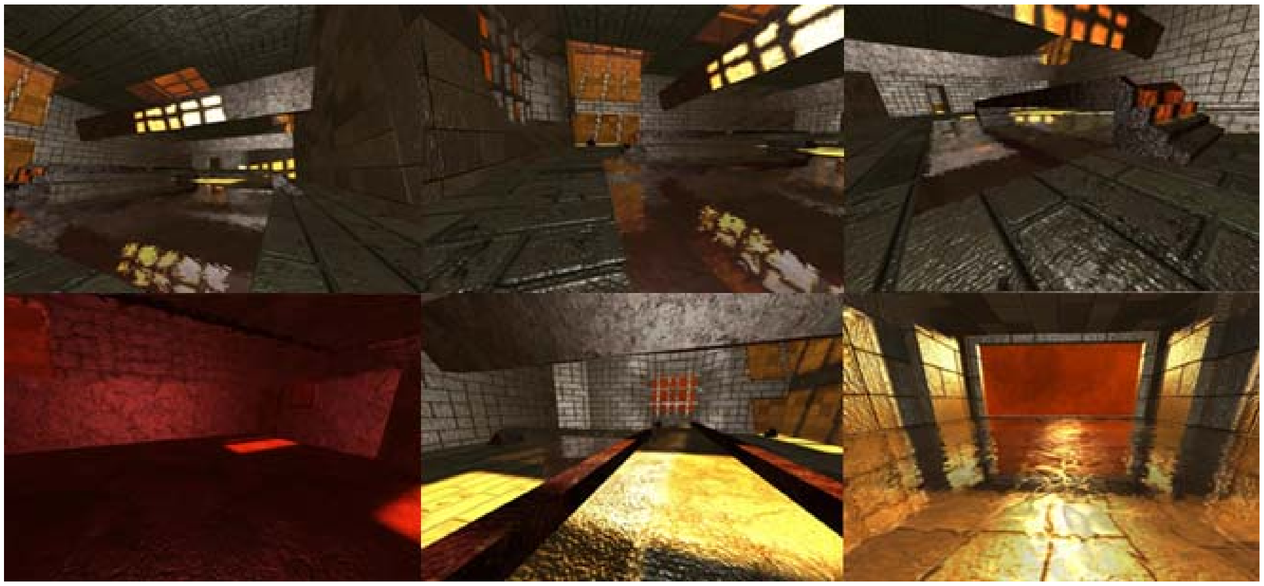


Figure 5.3 Various screenshots of the presented 3D engine.

Shadows

Starting with the engine's shadow quality scaling, as discussed in Rautenbach (2008), the presented benchmarking environment initially consisted of a number of static light sources. We then added a number of dynamic light sources (six) with shadow casting objects positioned relatively close to the viewer. This allowed us to analyse the transition from the spatial subdivision/SSE2 algorithm to Chan and Durand's algorithm.

Following this we increased the number of dynamic light sources to thirteen with the shadow casting objects translated to a significant distance from the point-of-view. All shadows previously rendered using Chan and Durand's algorithm were now rendered via shadow mapping.

Next we systematically increased the number of light sources to sixteen while leaving the shadow casting objects at their previous position – this caused a reselection of Chan and Durand's algorithm.

The shadow casting objects were subsequently translated back to their previous position (relatively close to the viewer) with the scene's lighting reset to nine dynamic light sources (with shadow casting objects located near the point-of-view) – Chan and Durand's algorithm was successfully selected.

Figure 5.4 shows the performance data obtained (for this specific instance) for up to eight light sources with Figure 5.5 showing the results obtained for nine to sixteen light sources.

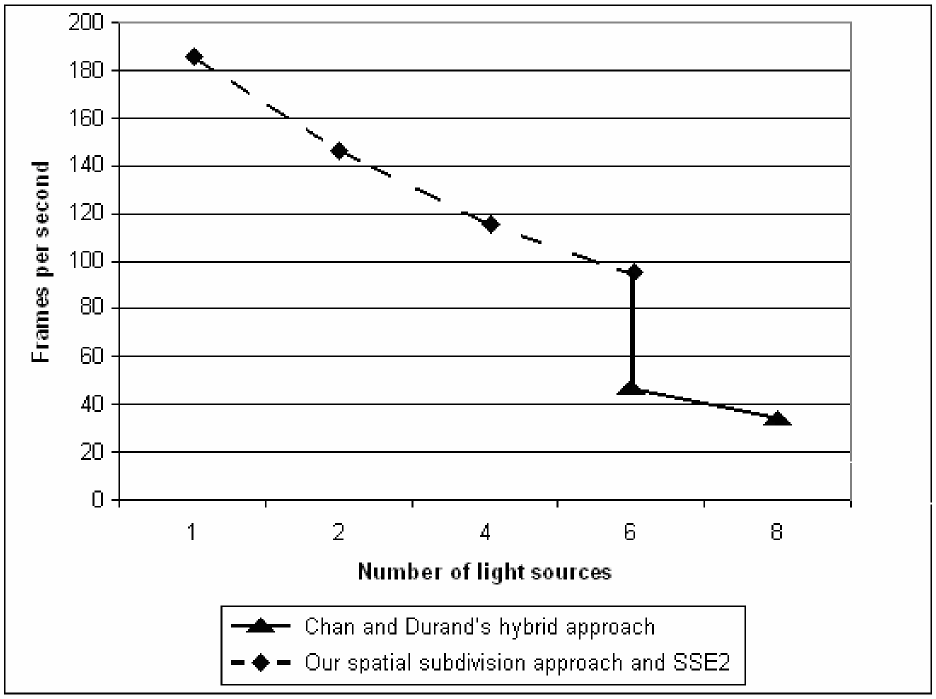


Figure 5.4 Shadow performance data for up to eight light sources.

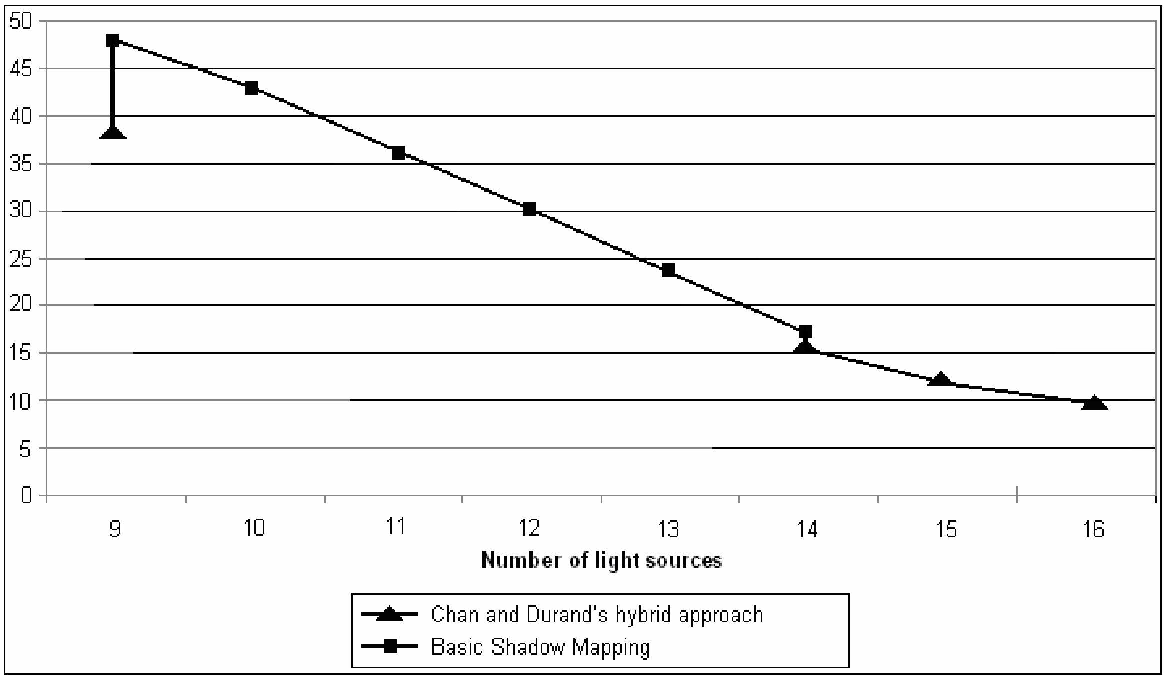


Figure 5.5 Shadow performance data for nine to sixteen light sources.

The experiment can be repeated in reverse order – that is, by starting with nine dynamic light sources (with shadow casting objects located near the point-of-view). Our benchmarking environment selected Chan and Durand's algorithm as its initial shadow rendering algorithm.

Next we systematically increased the number of light sources to sixteen while leaving the shadow casting objects at their previous position – Chan and Durand's algorithm was still the algorithm of choice and no alternative shadow rendering algorithms was selected.

Following this we decreased the number of dynamic light sources to thirteen with the shadow casting objects translated to a significant distance from the point-of-view. All shadows previously rendered using Chan and Durand's algorithm were now rendered via shadow mapping.

We now decreased the number of dynamic light sources to six with the shadow casting objects positioned relatively close to the viewer. This allowed us to analyse the transition from Chan and Durand's algorithm to the spatial subdivision/SSE2 algorithm.

Our final action was to set all the dynamic light sources to static. Figure 5.6 shows the performance data obtained for sixteen to nine light sources with Figure 5.7 showing the results obtained for eight to a single light source.

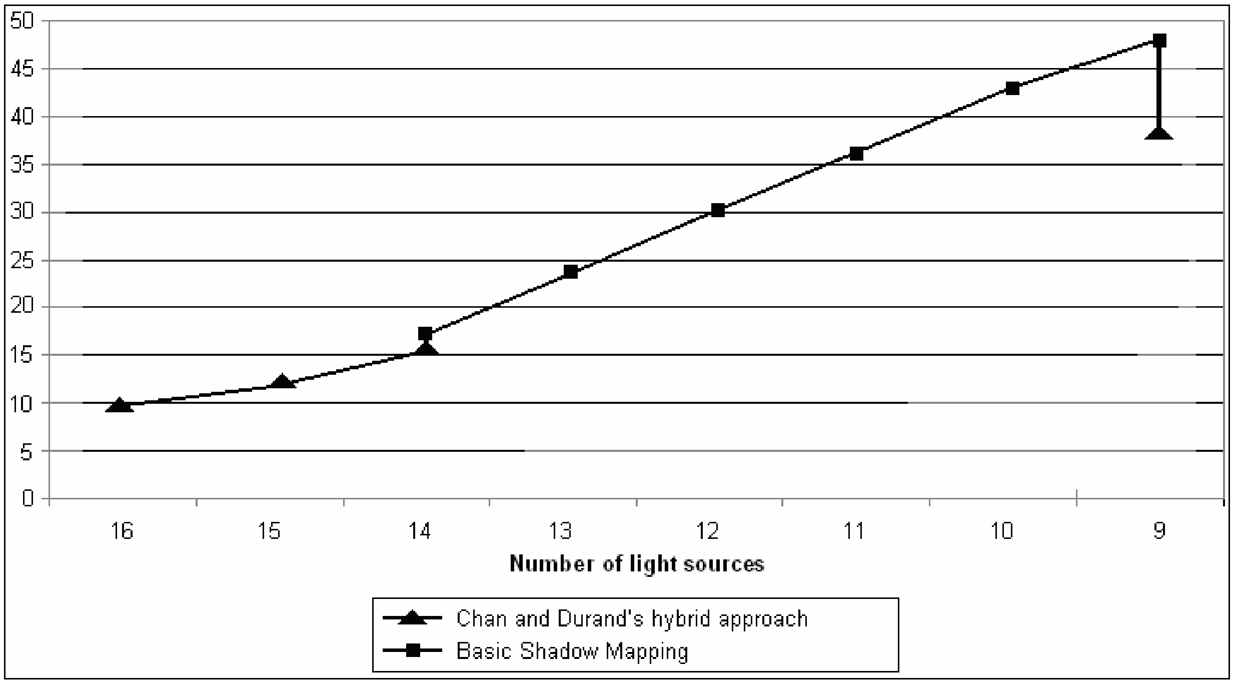


Figure 5.6 Shadow performance data for sixteen to nine light sources.

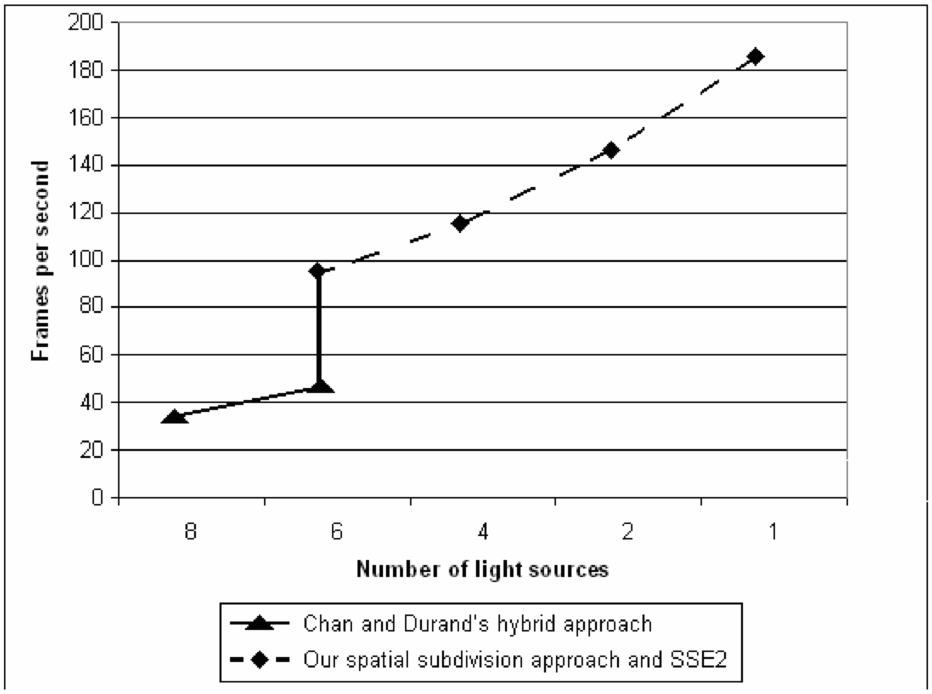


Figure 5.7 Shadow performance data for eight to a single light source.

Shaders

Similarly, for shader quality scaling, the presented benchmarking environment initially consisted of a single light source. We then added seven additional light sources with a number of objects positioned relatively close to the viewer. This allowed us to analyse the transition from the Very High Shader Quality grouping to the High Shader Quality Grouping.

Following this we increased the number of dynamic light sources to thirteen. The scene previously rendered using the High Shader Quality grouping were now rendered using simplified High Dynamic Range Lighting, normal maps, specular highlights and volumetric fog (the Medium Quality Grouping).

Next we systematically increased the number of light sources to sixteen. This caused a selection of the Low Shader Quality grouping.

The scene's lighting was now reset to seven dynamic light sources – the Very High Shader Quality grouping was successfully selected.

Figure 5.8 shows the performance data obtained (for this specific instance) for up to eight light sources with Figure 5.9 showing the results obtained for nine to sixteen light sources.

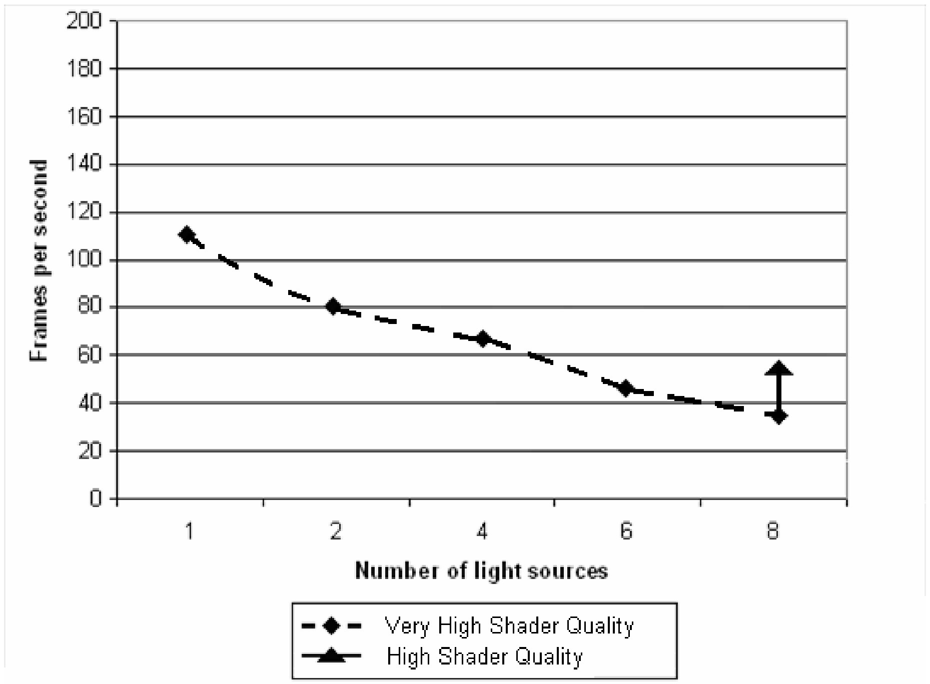


Figure 5.8 Shader performance data for up to eight light sources.

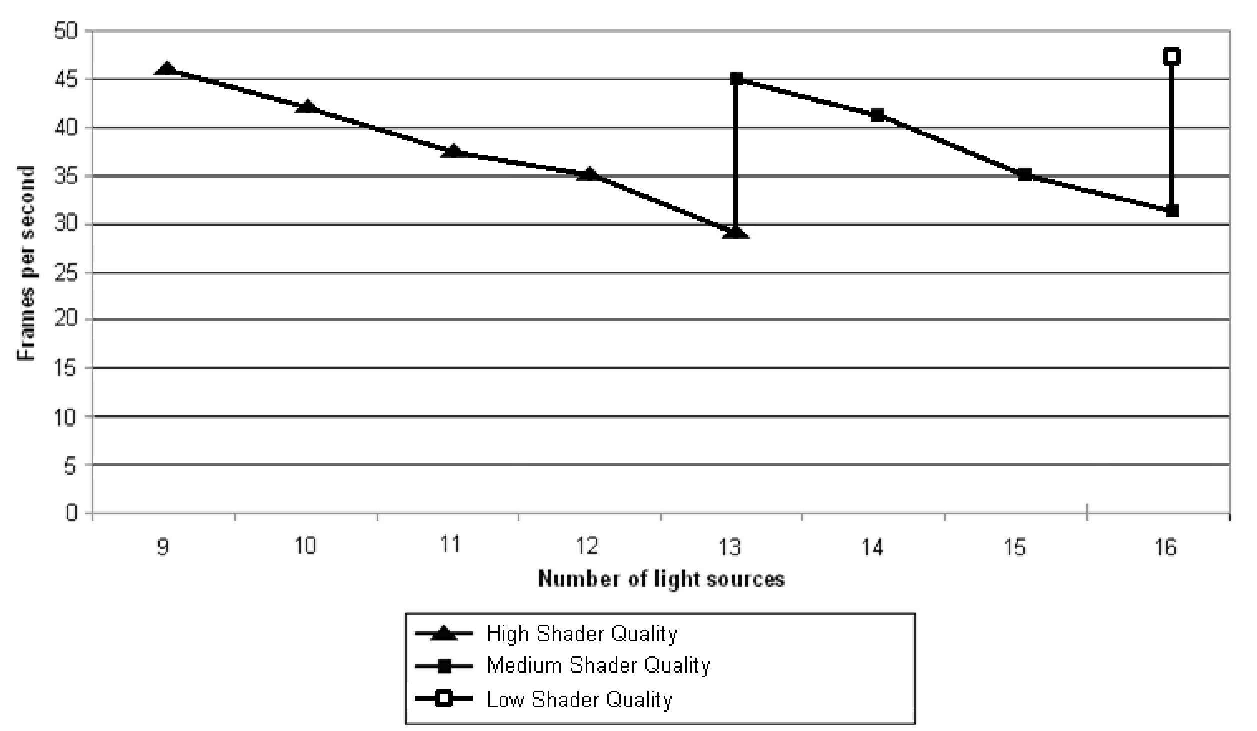


Figure 5.9 Shader performance data for nine to sixteen light sources.

Local Illumination

For local illumination quality scaling, the benchmarking environment (excluding all other special effects) consisted of five dynamic light sources. Fifty additional light sources were then progressively added (with a number of objects positioned relatively close to the viewer). This allowed the transition from the High Local Illumination Quality grouping to the Low Local Illumination Quality Grouping to be analysed.

The scene's lighting was now reset to twenty dynamic light sources – the High Local Illumination Quality grouping was successfully selected.

Figure 5.10 shows the performance data obtained (for this specific instance) for up to twenty-five light sources with Figure 5.11 showing the results obtained for thirty to sixty-five light sources.

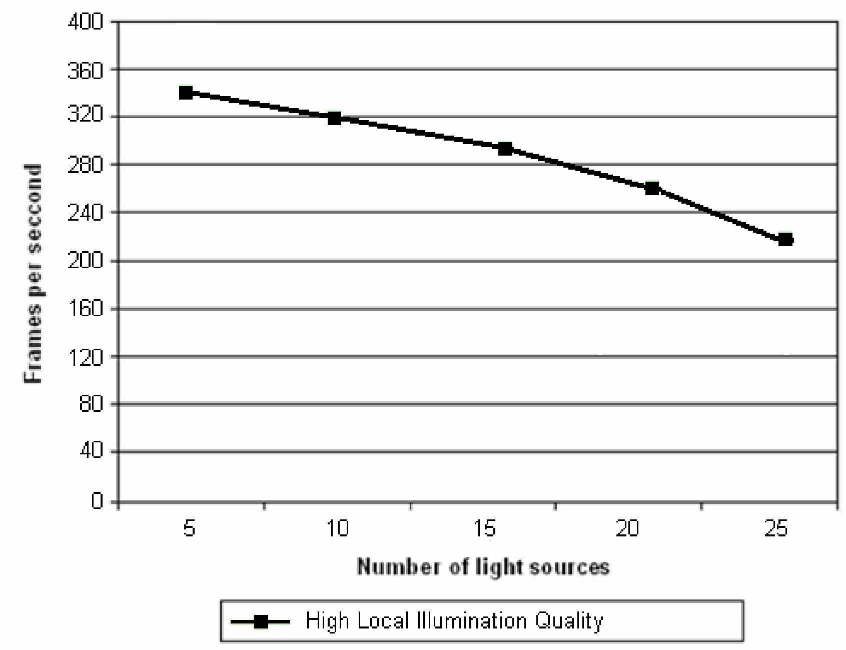


Figure 5.10 Shader performance data for up to twenty-five light sources.

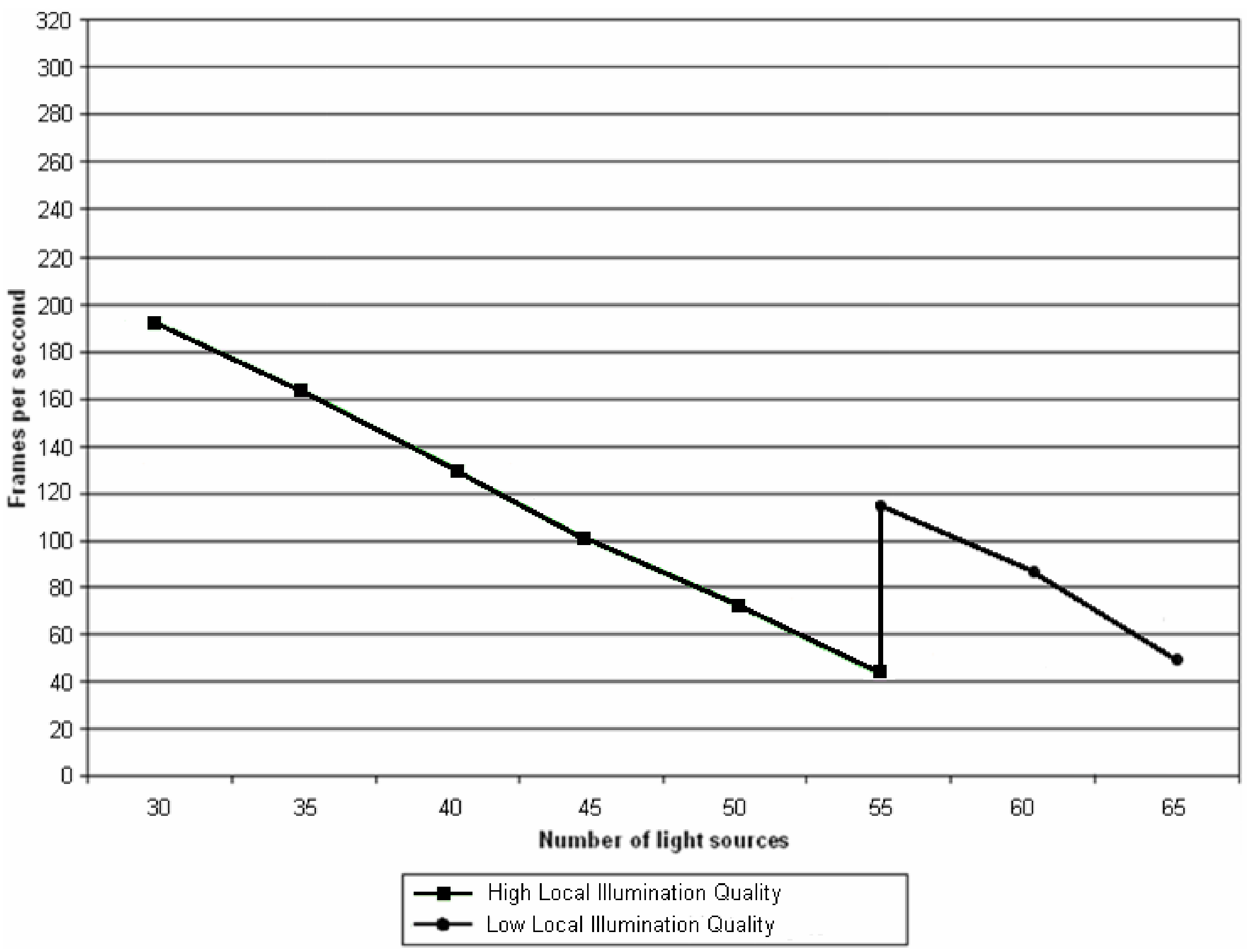


Figure 5.11 Shader performance data for thirty to sixty-five light sources.

Reflection and Refraction

Benchmarking the renderer's reflection and refraction quality scaling mechanism commenced with a test environment consisting of a single light source. Seven additional light sources, with a number of objects positioned relatively close to the viewer, were subsequently added. This allowed us to analyse the transition from the High Reflection and Refraction Quality grouping to the Medium Reflection and Refraction Quality Grouping.

Following this we increased the number of dynamic light sources to thirteen. The scene previously rendered using the Medium Reflection and Refraction Quality grouping were now rendered using the Low Reflection Quality Grouping.

The scene's lighting was now reset to seven dynamic light sources – the Very High Shader Quality grouping was successfully selected.

Figure 5.12 shows the performance data obtained (for this specific instance) for up to eight light sources with Figure 5.13 showing the results obtained for nine to sixteen light sources.

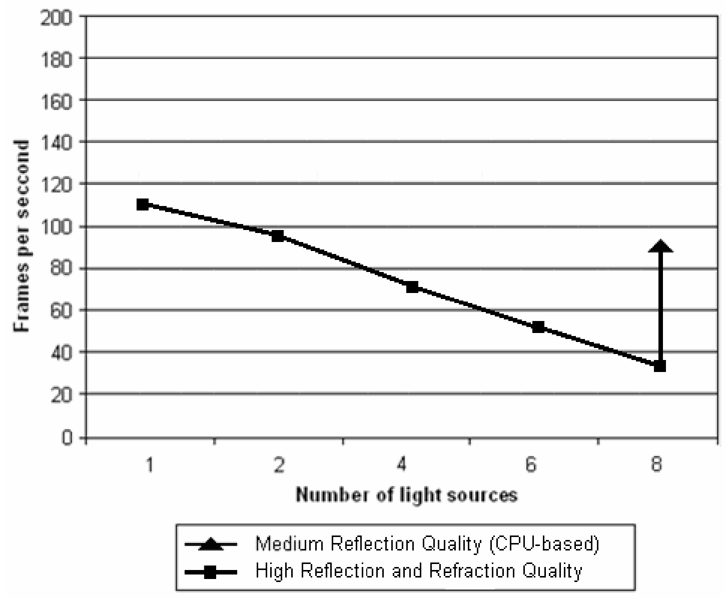


Figure 5.12 Reflection and Refraction performance data for up to eight light sources.

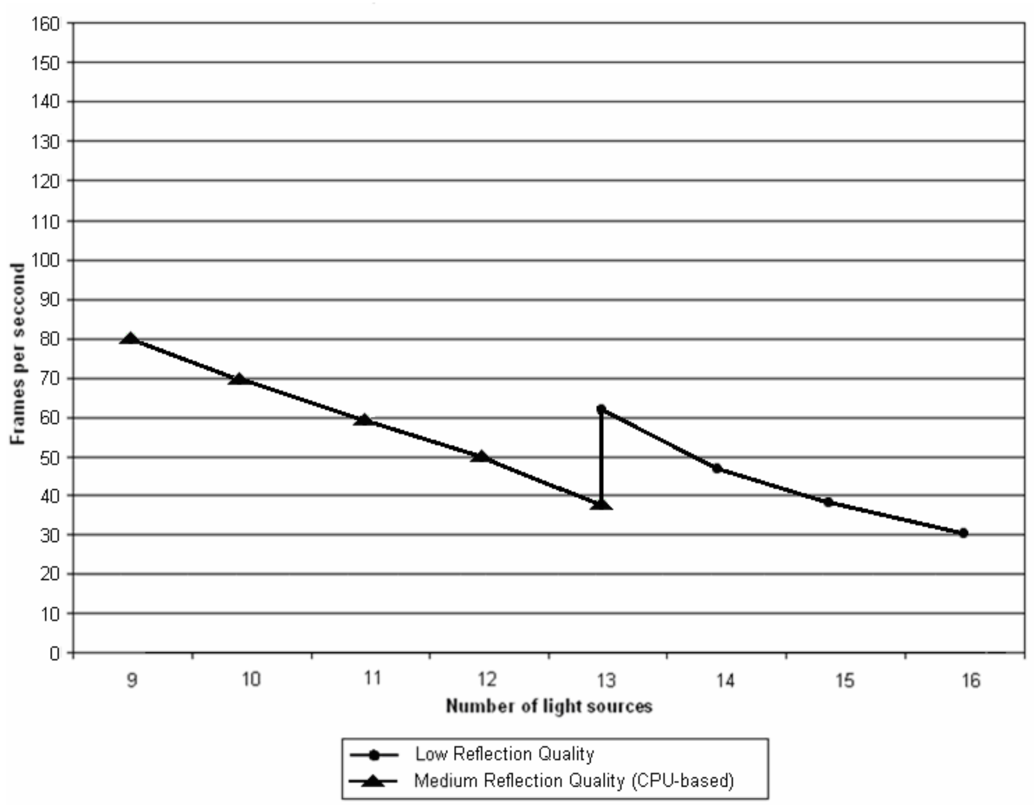


Figure 5.13 Reflection and Refraction performance data for nine to sixteen light sources.

Physics

Next, considering the renderer's physics quality scaling, our benchmarking environment consisted of a relatively simple cubic environment featuring 3D models and a simulated environment allowing for object interaction and collision.

We started with twenty-five objects then added fifty additional interacting objects. This allowed us to analyse the transition from the Very High Physics Simulation selection to the High Physics Simulation selection (with processes efficiently distributed between the CPU and GPU).

Following this we increased the number of objects to one hundred and twenty-five. The scene previously rendered using the High Physics Simulation selection were now rendered using the Low Physics Simulation selection.

The scene's object-count was now reset to fifty – Very High Physics Simulation was successfully selected.

Figure 5.14 shows the performance data obtained (for this specific instance) for up to one hundred and twenty-five objects.

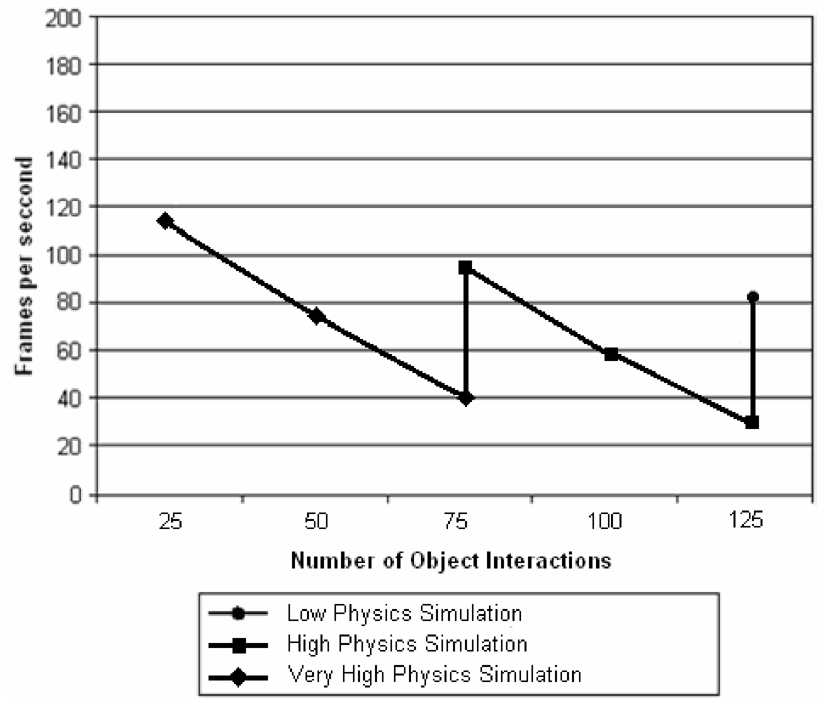


Figure 5.14 Physics performance data for up to 125 interacting objects.

Particles

As previously discussed, the particle system's evaluation focuses on a number of particle simulations (organised into performance-impacting selections ranging from Low to Very High). To gather the necessary results, we implemented our particle system for a basic scene – a relatively simple cubic environment featuring 3D models and a simulated environment with particle effects added to simulate explosions, dust, tread marks, beams, etc (the number of particles used per simulation range from 1500 to 7500).

The experiment started with one thousand five hundred particles. Four thousand five hundred additional particles were subsequently added. This allowed us to analyse the transition from the Very High Particle Simulation selection to the High Particle Simulation selection (with physics calculations efficiently distributed between the CPU and GPU).

Following this we increased the number of particles to one nine thousand. The scene previously rendered using High Particle Simulation were now rendered using Medium Particle Simulation.

The scene's particle-count was now reset to three thousand – Very High Particle Simulation was successfully selected.

Figure 5.15 shows the performance data obtained (for this specific instance) for up to nine thousand particles. Similar results are observed when repeating the experiment in reverse order.

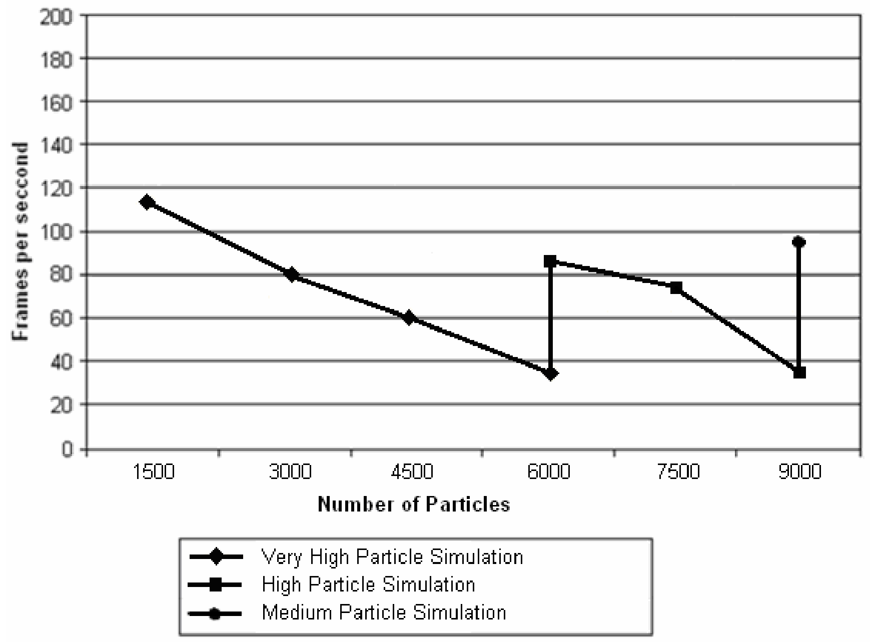


Figure 5.15 Particle performance data for up to 9000 particles.

Post-Processing

Post-Processing quality scaling benchmarking started with a basic 5 light source environment. Ten additional light sources were then added (with a number of objects positioned relatively close to the viewer). A transition from the High Post-Processing Quality grouping to the Medium Post-Processing Quality grouping was observed.

Following this we increased the number of dynamic light sources to twenty-five. The scene previously rendered using the Medium Post-Processing Quality grouping were now rendered using the Low Post-Processing Quality grouping.

The scene's lighting-count was now reset to seven dynamic light sources – the High Post-Processing Quality grouping was successfully selected.

Figure 5.16 shows the performance data obtained for up to twenty-five light sources. Similar results, as with all the other algorithms and approaches, are observed when repeating the experiment in reverse order.

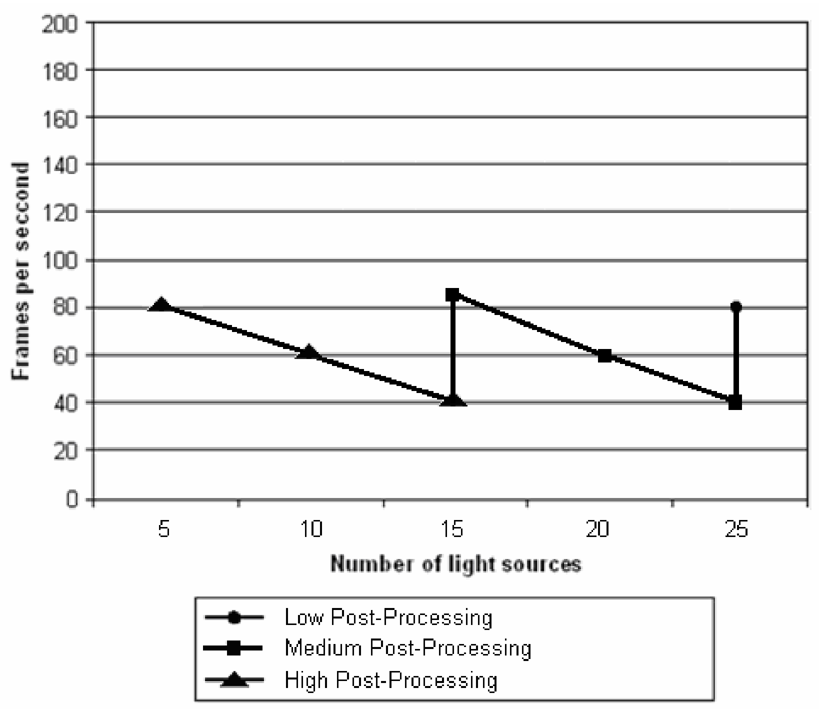


Figure 5.16 particle performance data for up to 25 light sources.

5.5 Summary

This chapter presented the general architecture of our empirically derived system for high-speed rendering – the dynamic process allocation and selection system being the main focus. Fuzzy logic-based reasoning for the explicit symbolisation of data was also looked at.

The final section summarised the results obtained by dynamically cycling through algorithms and quality groupings to compensate for performance-impacting changes in our rendering environment. These results illustrated the performance gains to be derived by the proposed system. The next chapter gives an overall summary of our work. It closes by discussing possible future work based on the presented research.

Chapter 6

Summary and Conclusion

Chapter 6 features an overall summary of our work. It closes by discussing possible future work based on the presented research.

In this chapter we will present:

- An overall summary of our work
- Concluding remarks and future work

6.1 Summary

The thesis presented a study performed through the implementation of a wide and representative range of rendering and physics algorithms (organised into performance-impacting groups). A platform supporting the swapping out of rendering algorithms and physics calculations as well as the transfer of specific tasks between the CPU/GPU was built. This platform enabled the detailed benchmarking of the various implemented algorithms which, in turn, allowed for the definition of a fuzzy-logic based expert system that was embedded into a real-time rendering engine. The rendering engine analyses the 3D environment being rendered and uses the benchmarked performance data that has been encapsulated in the fuzzy-logic based selection engine to determine the best solution to a given problem at any given moment. Whenever appropriate and for cube mapping and physics calculations, it augments the computational power of the parallel compute engine in modern GPUs with that of multi-core CPUs. This allowed for the rendering of complex geometric environments through the real-time swapping of rendering algorithms and, as proof of concept, through the effective distribution of specific processing tasks between the CPU and GPU.

The thesis was divided into two parts. Part I provided the background material deemed necessary to arrive at the final result. It started by looking at game engine architecture in general, highlighting the importance of software componentry, and the difference between game-engine code and game-specific code. Following this it focussed on a number of game engine architectures, specifically ad-hoc, modular and the directed acyclic graphs architecture (DAG).

Next it considered the first step invoked whenever a game is executed, namely initialisation. Initialisation was described as the stage responsible for resource and device acquisition, memory allocation, setup of the game's GUI, loading of art assets, etc. Following front-end initialisation, it discussed the exit state and the game loop for the uninterrupted execution of a game.

Following this, the thesis dealt with the general design and implementation of a generic game engine which serves as the core of the presented dynamically scalable interactive rendering engine.

The thesis then introduced our modular rendering engine as a scalable interactive testing environment and complete solution for the rendering of computationally intensive 3D environments. A detailed discussion of the presented interactive environment's core rendering elements was subsequently given. These elements were grouped into the following rendering or computation categories: shaders, local illumination, reflection and refraction, shadows, physics, particles and post-processing special effects. This was the end of Part I.

Part II of the thesis categorised the presented algorithms and rendering groupings based on the level-of-detail/rendering quality and the associated computational impact. It also focused on the critical analysis and detailed benchmarking of the presented rendering and simulation techniques – the data used by the presented fuzzy-based selection and allocation system.

Part II commenced with a discussion of the proposed benchmarking mechanism as well as a set of criteria for the evaluation of rendering algorithms and techniques. The given evaluation criteria were selected with the aim of assessing the relationship between rendering quality and performance – in turn allowing for, where applicable, the isolation of key algorithmic weaknesses and possible bottleneck areas.

Drawn from the MSc dissertation preceding this thesis (2008), the shadow algorithms benchmarked and analysed include: the basic stencil shadow volume algorithm, the basic hardware shadow mapping algorithm, McCool's shadow volume reconstruction using depth maps, Chan and Durand's hybrid algorithm for the efficient rendering of hard-edged shadows, Thakur el al's algorithm based on the elimination of various shadow volume testing phases and Rautenbach et al's algorithm based on shadow volumes, spatial subdivision and instruction set utilisation.

Shader evaluation subsequently focused on a number of shader implementations and lighting approaches. Following this, two local illumination configurations were investigated – the first of these limiting the number of light sources in an attempt to reduce GPU utilisation with the second lifting this limitation while occluding local light sources (a technique used to approximate the effect of environment lighting as an attempt to simulate the way light radiates in real life).

Next the evaluation focused on a number of reflection and refraction implementations and approaches, specifically: basic environmental mapping, CPU-based cube mapping, refractive environmental mapping and the extension of these reflection and refraction algorithms through the addition of the Fresnel effect and chromatic dispersion.

The thesis then shifted focus to the evaluation of a number of physics calculations such as object acceleration, force, linear momentum, gravitational pull, projectile simulation through trajectory paths, friction and collision detection followed by the benchmarking of the presented rendering engine's dynamically allocated particle generator.

The benchmarking exercise concluded with the performance analysis of a number of post-processing shader implementations and lighting approaches, specifically displacement mapping, bloom effects, ambient occlusion, depth of field and halo effects.

The thesis closed by presenting the general architecture of the proposed dynamically scalable interactive rendering engine – the dynamic process allocation and selection system being the main focus. It also looked at fuzzy logic-based reasoning for the explicit symbolisation of data. The results obtained by dynamically cycling through and offloading algorithms and quality groupings to compensate for performance-impacting changes in a rendering environment were subsequently given. These results illustrated the performance gains inherent to the proposed system's use.

6.2 Concluding Remarks and Future Work

The computer graphics industry has developed immensely during the past decade. Looking at the area of computer games one can easily see technological leaps being made on a yearly basis. However, most of the currently available rendering algorithms are only amenable to specific rendering conditions and/or situations.

A viable solution to GPU and, to a limited degree, CPU over- and/or underutilisation (depending on the scene being rendered) was to perform a critical analysis of numerous rendering algorithms with the aim of assessing the relationship between rendering quality and performance. Using this performance data gathered during the analysis of various algorithms, we were able to define a fuzzy logic-based selection engine to control the real-time selection of rendering algorithms and special effects groupings based on environmental conditions (as discussed in Chapter 4 and 5). This system ensures the following: nearby effects are always of high-quality (where computational resources are available), distant effects are, under certain conditions, rendered at a lower quality and the frames per second rendering performance is always maximised. Furthermore, as a secondary objective, we have shown that the unification of the parallel compute engine present in modern GPUs with that of multi-core CPUs to allow for the rendering of complex geometric environments is a viable solution for the management of scarce computational resources and that improved rendering quality and performance can be achieved through load-balancing between the CPU and GPU.

It is important to note that this engine and its selective utilisation of the CPU in an attempt to free up GPU resources and, in turn, to accelerate graphics performance is, in principle, also adaptable for use with 3D capable mobile devices (such as the iPhone, IPad and iPod Touch); it is expected to give these devices the ability to render special effects not previously possible by maximising the utilisation of both CPU and GPU. Further experimentation in this regard would seem appropriate.

We have also demonstrated that the use of a relatively simple fuzzy-logic based expert system can serve as a viable solution to the problem of selecting between and distributing competing algorithms in real-time. This resulted in the optimisation of GPU usage by ensuring that the quality of special effects is appropriately tuned.

This work is also, in some sense, similar to current research on software evolution in the context of MAUS (Mobile and Ubiquitous Systems) which investigates how on-the-fly architectural reconfigurations are needed for such systems as context changes due to their mobility (Autili et al, 2010). Our work can inform theirs in as much as it points to the utility of a fuzzy-logic based expert system to determine which changes to make as the context changes.

It is also important to note that, despite all the rendering algorithms and approaches available, a lot of work remains in the field. More algorithms could, as future work, be benchmarked and added to our selection engine's knowledge base. Special effects groupings could also be assigned collective weights based on the groups overall impact on rendering performance (for example, the post-processing effects group will have a bigger overall performance impact that the local illumination group). The implemented rendering engine is also highly expandable and alternate rendering solutions, whether GPU or CPU based, can be implemented and loaded into the engine as additional dynamic link libraries. Alternate algorithmic performance improvements can also be pursued.

Furthermore, utilising a selection system such as the one in this thesis will allow modern engines to not only do away with their performance setup screens (thus freeing users from the cumbersome task of fine-tuning the game's graphics performance) but will guarantee a rendering environment that is running at the most optimised level possible by not just lowering "drawing distance" or "texture quality" but by actually selecting the most appropriate rendering approach and shader implementation for the current scene being rendered.

Immersive rendering approaches used in conjunction with AI subsystems, game networking and logic, physics processing and other special effects (such as post-processing shader effects) are immensely processor intensive and can only be collectively implemented on high-end hardware. This thesis has illustrated that by cycling and distributing algorithms based on environmental conditions and by the exploitation of algorithmic strengths, that a vast array of high-quality real-time special effects and highly accurate calculations can become as common as texture mapping.