# Ambient Occlusion and Shadows for Molecular Graphics

Robert Easdon

A thesis submitted for the degree of
Master of Science
at the University of East Anglia
September 2013

# Ambient Occlusion and Shadows for Molecular Graphics

Robert Easdon

# Abstract

Computer based visualisations of molecules have been produced as early as the 1950s to aid researchers in their understanding of biomolecular structures. An important consideration for Molecular Graphics software is the ability to visualise the 3D structure of the molecule in a clear manner.

Recent advancements in computer graphics have led to improved rendering capabilities of the visualisation tools. The capabilities of current shading languages allow the inclusion of advanced graphic effects such as ambient occlusion and shadows that greatly improve the comprehension of the 3D shapes of the molecules.

This thesis focuses on finding improved solutions to the real time rendering of Molecular Graphics on modern day computers. The methods of calculating ambient occlusion and both hard and soft shadows are examined and implemented to give the user a more complete experience when navigating large molecular structures.

# Acknowledgements

I would like to thank both my primary supervisor Dr. Stephen Laycock and my secondary supervisor Dr. Steven Hayward for their support and advice throughout my degree, helping advance both my computer graphics knowledge, as well as my understanding of the biological aspects of this research. Further thanks in particular to Dr. Stephen Laycock for his consistent encouragement and guidance, which has helped me greatly over the past two years.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Computer generated images of molecules first appeared in the 1950s [AW59]. Up until this point, molecules were hand drawn or modelled by physical objects, for study in structural biology and biochemistry. These models were limited by practicalities such as complexity of construction as well as the large space they consumed. Molecules can consist of thousands of individual atoms, making it hard to study areas of the model when they are occluded by other atoms. Although not known at the time, computers would hold the answer to the problem of displaying the atomic structure of molecules. Early computing hardware, devices to display the images, as well as devices for interactive input were all limitations for computer graphics engineers at the time. However, as computing power increased and new graphics hardware was developed, real time rates could be achieved to allow interaction with a molecular structure on personal computers. Exploration of molecules no longer required physical models or expensive and large mainframe systems. Researchers were able to load newly developed graphics programs onto their own devices, allowing for more efficient work.

Appreciating the 3D form of a molecule from a 2D image on a display is often difficult. Following advancements in graphics algorithms and hardware, improved lighting and stereoscopic techniques are commonly employed in molecular visualisation software. These techniques are useful to aid biologists in understanding the 3D

structure of proteins by enhancing the simulation of depth in the image. One such technique, popularised by the games industry and now seen in molecular visualisations, is ambient occlusion [ZIK98].

Ambient occlusion is a shading method used in computer graphics which helps to improve upon local illumination models by taking into account attenuation of light due to occlusion. It attempts to approximate the way light radiates in real life, producing effects such as contact shadows, unattainable using standard lighting methods. Although a very basic approximation to full global illumination, it can be calculated much faster whilst still enhancing the visual representation of the scene.

Ambient occlusion is commonly calculated in offline rendering software by casting rays in every direction from a point on the surface. Rays that do not intersect geometry in the scene and reach the background or "sky" increase the brightness of the surface, whereas rays that do intersect with other objects in the scene contribute no illumination. This results in points surrounded by large amounts of geometry rendered darker than points with little geometry on the visible hemisphere. Attempts to incorporate ambient occlusion into real time applications have seen a number of techniques developed, such as image space approximations of the algorithm introduced by Mittring [Mit07], to reduce calculation time of the effect. Although, as with many real time approximations, these techniques tend to suffer from inaccuracies and are of a lower quality than a ray traced approach.

Whilst ambient occlusion deals with light with no specific direction or source, it may also be desirable to handle light that has a clear origin, such as a torch or the sun. In real time graphical applications today, Shadow Mapping is a popular choice to generate shadows in a scene.

Shadow Mapping has been used both in pre-rendered scenes and real time scenes in many graphics applications, including molecular graphics programs. Shadows are

created by testing whether a pixel is visible from the light source, by comparing it to a z-buffer from the light source's view, stored in the form of a texture. The ease of implementation combined with the lack of knowledge needed about the scene's geometry has made this technique a popular choice for real time graphics applications, including molecular visualisation programs. However, additional techniques are required to correct visible aliasing at shadow edges where projected shadow map texels cover a noticeably large area in screen space.

To give users the best experience when visualising molecular structures, ambient occlusion and shadowing methods are both included into molecular graphics packages to enhance user interaction with the structure. However, there are challenges in creating real time graphics applications of large biomolecules on a variety of computer hardware. The rendering algorithms need to maintain a frame rate of 30 Hertz to ensure the simulation remains interactive. This is not a simple task considering molecules can contain many thousands of atoms.

## 1.1 Motivations and Research Objectives

Many software programs exist which attempt to convey the 3D form of biomolecules. However, few manage to provide the sense of depth that is imperative to help navigate large molecular structures and understand how their geometry relates to their function. Using complex lighting techniques, such as ambient occlusion, to achieve a high quality visual representation is vital in aiding a user's perception of a molecule, although producing these results is commonly limited to images generated offline. However, speed of rendering is of the highest importance in real time applications, with modern day computers giving the expectation of a fast and smooth interactive frame rate.

With these motivations in mind, the aim of this thesis is to improve upon clarity

of the existing HaptiMOL iSAS software and bring an easier understanding of the 3D shape and structure of large molecules or complex proteins, whilst still producing a real time visualisation. This will be achieved through implementing advanced, yet efficient, lighting techniques including ambient occlusion, and hard and soft shadows, that will be key to improving the realism of the molecular visualisation. The techniques developed will be incorporated into HaptiMOL iSAS, improving upon its traditional rendering methods. HaptiMOL iSAS is a software package that provides haptic rendering with molecular graphics to give a deeper appreciation of the shape of biomolecules by augmenting the sense of sight with touch [SHL09]. By using a spherical probe that is the same size as a water molecule, rolling it over the biomolecule is equivalent to exploration of the solvent accessible surface.

## 1.2 Contributions

The main contributions of this thesis include; the investigation of the visualisation problems associated with rendering large structures encountered in biomolecular study. The technical comparison of popular ambient occlusion and shadow techniques and how they perform when used in molecular graphics. Ray tracing on the GPU with the use of the OpenGL graphics library and the GLSL shading language is investigated as a method to create these effects, focussing on the space-filling molecular representation. Knowing we are rendering spheres to represent atoms allows us to make several optimisations to allow real time performance using this technique. Finally, the integration of the most promising results into existing software that has been developed for the study of arbitrary size biomolecules.

## 1.3 Thesis Outline

Chapter 2 gives a comprehensive review of first the history behind the graphical visualisation of molecules, and then moves on to current methods being employed to solve the problems associated with it. Chapter 3 looks at efficient ways of rendering space-filling representations of molecules, and how clipping planes can be implemented to remove atoms from view whilst still maintaining a realistic visual representation of the molecule. In Chapter 4, a variety of ambient occlusion methods are compared based on performance and visual quality. Shadows play a key role in 3D perception in graphics, therefore Chapter 5 covers shadow techniques, including a ray traced approach on the GPU. The thesis concludes in Chapter 6, including a section on further work that can be done to extend on the research described in previous chapters.

## 1.4 Terminology

### 1.4.1 HaptiMOL iSAS

Software developed on throughout the thesis. Further explained in Section 3.2. Haptic **MOL**elcular **i**nteractive **S**olvent **A**ccessible **S**urface.

### 1.4.2 Molecular Graphics (MG)

Molecular Graphics (MG) is the term used to describe the study of molecules and their properties through graphical representation.

### 1.4.3 OpenGL Shading Language (GLSL)

OpenGL Shading Language (GLSL) is a high-level shading language to give developers more direct control of the graphics pipeline. It allows for increased flexibility in the rendering pipeline at the vertex and fragment level.

### 1.4.4 Screen Space Ambient Occlusion (SSAO)

Screen space ambient occlusion (SSAO) is a rendering technique for approximating the computer graphics ambient occlusion effect in image space.

# Chapter 2

# Literature Review

## 2.1 Introduction

Before attempting to find solutions to the real time rendering of Molecular Graphics on modern day computers, it is important to firstly understand where the problems came from and what methods have already been employed to solve them. Therefore this chapter focuses on reviewing the literature relevant to Molecular Graphics, and in particular two graphical techniques, ambient occlusion and shadows, that are the focus of this thesis. In Section 2.2, early examples of Molecular Graphics on computers are looked at to help give an overview of the importance of Molecular Graphics. Current and popular molecular visualisation systems are then analysed in Section 2.3, as well as the modern rendering techniques that have been developed for these applications. The final two sections look at graphics techniques that link directly to this thesis. Section 2.4 discusses Ambient Occlusion, methods to apply this lighting technique to real time applications, and how it can be used with good effect to improve the comprehension of the 3D shapes of the molecules. Finally, Section 2.5 investigates the current methods of implementing shadows into a 3D scene, and looks at why they are so important to a person's perception of the environment.

## 2.2 Early Molecular Graphics on Computers

Molecular Graphics is the field of visualising molecules and their properties on graphical display devices [IUP]. There are many uses for visualising molecules, with these models playing an important role in the study of proteins and nucleic acids, and allowing hypotheses to be tested. Molecular Graphics assisted drug design is another extremely important area of research aided by computers to speed up the development process.

Before the time of computers, molecules were modelled using physical objects, perhaps the best known is the DNA model built from rods and sheets by Watson and Crick [WC$^+$53]. Physical models allow easy interactivity, are tactile, and their visual aspects can not be easily reproduced on computers. However, computer rendering of molecules is now the standard method to present graphical molecular data. This is due to the practicalities of building physical models, with large molecules possibly taking weeks to build. Computer visualisations are also very good at representing computed properties such as atomic charge and electrostatic potential, needed for applications such as drug design. Haptic devices are sometimes employed to reproduce the missing tactile element found in physical models, and has been applied to molecular graphics in three main areas. The first is the teaching of structural biology, where a user can explore all areas of geometry with an extra sense of touch [SWS$^+$03]. A second application is particle steering with the haptic device used to manipulate a molecular structure [BKB$^+$07]. Finally haptics are used to explore the docking of proteins and ligands [SB08]. Combined with 3D stereoscopic displays and realistic lighting techniques giving the illusion of depth, haptics can improve visualisations for molecular graphics researchers.

It was in 1966 at Massachusetts Institute of Technology (MIT) that computers were first used to graphically render molecular data to a CRT display. Using Project

MAC (Multi-Access Computer), an early mainframe computer [FC66], a model building program was designed to work with protein structures [Lev66]. The program allowed the animation and interaction of a molecular structure, with a 3D effect achieved by rotating the wireframe model constantly on the screen. Figure 2.1 shows an overview of the molecular modelling system, with the visual display known as the 'KLUGE' in the bottom left of the image.

Hardware constraints limited the system, offering only a small amount of interaction to the user with a trackball-like device to control the rotation speed of the molecule on the screen. Storage and processing power at the time limited the viewing of structures to simplified versions of more complex molecules. The lack of colour also limited the ability to distinguish between atom types. Figure 2.2 provides a close up view of the display, as well as the input device that controls the molecule direction and speed of rotation.

Levinthal noticed the large potential of using computers to display and interact with large molecular structures, however it was then too early to know if computing could advance in power and decrease in price enough for this to be used on a large scale. Although the importance of this creation was not fully realised at the time, this program would pave the way for future developments in molecular graphics.

Also in the 1960s, Carroll K. Johnson of Oak Ridge National Laboratory released ORTEP (Oak Ridge Thermal Ellipsoid Plot Program), a program to produce stereoscopic drawings of molecular and crystal structures with a pen-plotter (a vector graphics printing device) [Joh65]. It was able to produce ball-and-stick illustrations of crystal structures at a quality high enough for publication uses. An appealing feature of this application was the ability to produce stereoscopic pairs of images. This technique generates a 3D view of an image by creating a single image for each eye. This stereo rendering aids greatly in the visualisation of complex arrangements

Figure 2.1: Overview of the molecular modelling system, with the KLUGE in the lower left corner, and a space-filling model by the display. Photo courtesy of Martin Zwick.


Figure 2.2: A close up view of the CRT screen, KLUGE, with the globe device that controls the direction and speed of a structure's rotation. Photo courtesy of Martin Zwick.

of atoms, such as proteins. Figure 2.3 shows a pair of images generated by ORTEP. Later revisions to ORTEP added features such as colouring and additional output formats, however, it remained a static modelling package. Interaction with a molecule is achieved through the command line, updating the image based on certain inputs, meaning it could not provide a true interactive experience.

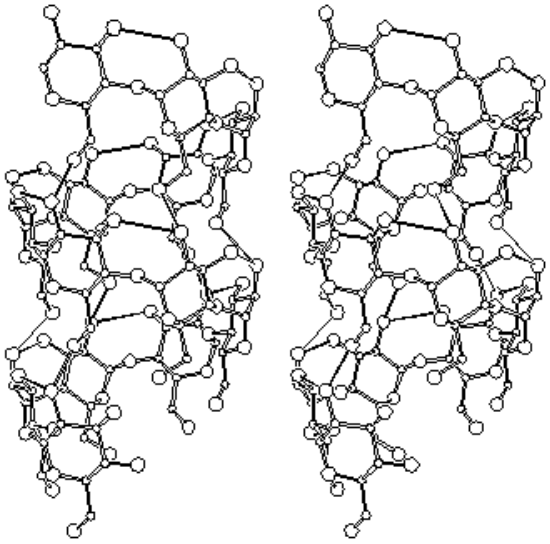


Figure 2.3: A stereoscopic pair of images depicting the amylose polymer generated by ORTEP [Joh65].

By the 1990s, technological developments in hardware allowed for high quality visualisations of molecules on personal computers. The availability of the personal computer (PC) removed the need for expensive and large mainframe systems, requiring full-time staff to maintain. Researchers could load molecular graphics programs onto their general purpose device to explore molecules of interest whenever desired, allowing more effective and efficient work.

RasMol, developed by Roger Sayle [SB92] in 1992, was an important development in the molecular graphics field at the time. Before RasMol, visualisation programs

required expensive dedicated workstations to run, meaning they were not easily accessible to researchers due to the large costs involved. RasMol quickly became a widely used tool for creating images for research publications, and for education purposes. The source code was made open source, and was continued to be revised years after its release by many contributors to support different types of computers.

Sayle's original interest during his undergraduate degree was in the problem of depth perception of solid objects in computer graphics. This interest was then applied to molecular graphics in particular, leading to the creation of a shadowing program that would generate shadows for hard sphere molecular representations. RasMol's rendering algorithm worked by first calculating the visible surfaces using a scanline z-buffer algorithm similar to the one described by Porter [Por78], then calculating the shadowed pixels using ray tracing [Whi80]. These shadows provided an enhanced 3D effect that could help in the discovery of pockets in the molecular structure, which are of interest to researchers when looking at how the geometry of the molecule relates to its function.

RasMol's efficient rendering code has been used in a number of molecular visualisations programs, including MDL's Chime [CHI]. Chime, made available in 1996, takes around 16,000 lines of lines of code from Sayle's original source, with many thousands more added by MDL to add new features whilst improving the user experience. Chime differs from RasMol in that Chime is a web browser plug-in that runs inside the browser, with visualisations happening directly on a page of a website. Websites providing the ability to run this plug-in would display molecules provided by the author of the site, however dedicated websites and interfaces provided support for showing any molecule desired, such as Protein Explorer.

Protein Explorer is ideal for beginners, with the goal of the program to make the power of Chime available to scientists, educators, and students alike [Pro]. It is a

RasMol-like interface to help load and navigate structures, providing the full power of Chime, without requiring advanced technical knowledge and a large amount of time to use. It can make the visual exploration of the protein much easier for novice users as well as more convenient for specialist users. Chime proves more effective than RasMol for presenting structural information, mainly because the plug-in can be accompanied by text and keys within the same web page. Buttons in the text can control Chime's functions through the JavaScript web language, allowing tasks to be automated, and providing an excellent educational resource. One negative feature of Chime is the code base has not been released publicly, which would have allowed external code contributions for new functionality to keep the program relevant.

A more modern open-source molecular viewer came in the form of Jmol in the early 2000s [Jmo]. It is a cross-platform program, that can be run as a standalone Java application on the desktop, or as an applet that can be integrated into web pages of all major browsers. This offered an alternative to the Chime plug-in which was no longer in development. With a basic understanding of web technologies such as HTML, Jmol can be easily incorporated into a website to display molecular graphics visualisations to a wide audience.

Similar to RasMol, Jmol uses a custom rendering engine optimised specifically for drawing core geometric primitives that are used in molecular representations. Instead of using universal graphics APIs such as DirectX or OpenGL, building a custom rendering engine allows more efficient rendering as the engine does not need to be generic for all graphics. This also means Jmol can perform high-performance 3D rendering with no hardware requirements and no need for 3D acceleration plug-ins, making it a highly adopted tool even today.

While Jmol was an improvement to existing rendering systems at the time such as RasMol, there was a growing desire for the use of advanced rendering techniques

to improve the aesthetics and perception of a molecule. Rapid advancements in dedicated graphics hardware would allow these effects to be achieved at real time rates. The next section will look at modern systems that employ these techniques that allow for enhanced high quality publication images to be generated, as well as interactive study of molecular structures.

## 2.3 Modern Molecular Graphics Systems and Techniques

Modern molecular graphics applications commonly split the rendering of molecular data into two areas: real time and offline rendering. Offline rendering is generally used to produce high quality images of structures used for publication, or combined to create scripted animations. Performance is a second priority to image quality with pre-rendering, allowing more complex algorithms to be used to improve the image. Of course, user interaction will be impaired, however it is typical for molecular graphics programs to provide the option for a simplified interactive display, as well as an offline solution allowing the user to create high quality images and movies.

Visual Molecular Dynamics (VMD) [HDS96] is a molecular modelling and visualisation system, primarily designed for viewing and analysing the results of molecular dynamics simulations. A common theme in molecular graphics applications, the program uses OpenGL [Grob] to render real time visualisations, and incorporates external rendering tools such as POV-Ray [Pov] and RenderMan [Stu], to create high quality ray traced images for publication use. Another system able to create presentation quality images, as well as analyse molecules in real time is PyMol [Del02]. It is seen as the current industry standard for molecular graphics software. Again OpenGL is used to generate interactive visualisations, with PyMol implementing Shadow Mapping for shadow generation. Offline renders can be created using PyMol's own ray

tracing engine, which has made it widely used for molecular images in much of the scientific literature.

Ray tracing is a graphics technique that takes a different approach to rendering a 3D scene than the traditional rasterisation technique. The rasterisation algorithm takes a stream of vertices, transforms them into corresponding 2D points on the viewer's monitor, and finally the transformed 2D triangles are filled in as appropriate. Ray tracing, however, is a technique for generating an image by tracing a ray through pixels in an image plane and simulating the effects of its encounters with the objects in the scene. The technique is capable of producing a very high degree of realism, far higher than the popular rasterisation method, but this comes at a greater computational cost. Therefore it is not common to see the technique employed for real time graphics applications. However, due to ray tracing's ability to simulate a wide variety of optical effects, such as reflection, refraction and scattering, it has become a widely used tool for use in film special effects as well as molecular graphics renders, where the image can be rendered offline.

The first ray tracing algorithm used for rendering was presented by Arthur Appel in 1968 [App68], although what was presented is now generally considered ray casting, as no secondary rays such as shadow rays were used. The idea behind the algorithm is for each pixel, rays are cast from the eye to find the closest object in the scene intersecting the ray. Using the material properties and the effect of the lights in the scene, this algorithm can determine the shading of this object. The simplifying assumption is made that if a surface faces a light, the light will reach that surface and not be blocked or in shadow. The important advantage Appel showed was what ray casting offered over older scan line algorithms in its ability to easily deal with non-planar surfaces and solids, such as cones and spheres. If a mathematical surface can be intersected by a ray, it can be rendered using ray casting. More complex

objects can be created by using solid modelling techniques and easily rendered.

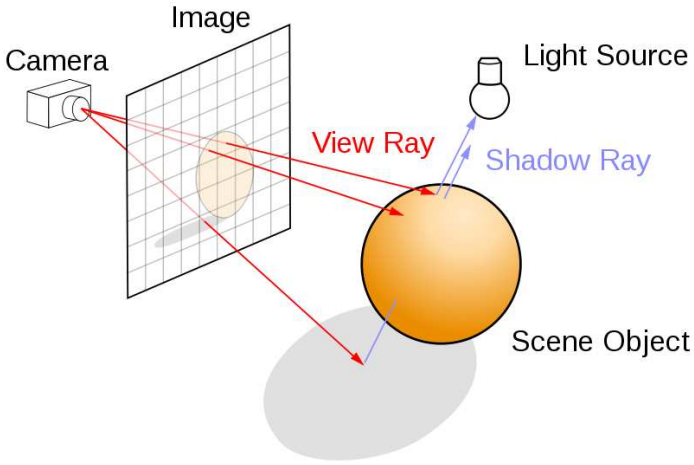



Figure 2.4: A diagram of the ray tracing algorithm. View rays are sent from the camera to find the closest intersection point. Rays to test for shadows or other illumination effects are then sent out from this point.

The next important development came from Turner Whitted in 1980 [Whi80]. Whilst Appel cast rays from the eye into the scene until they hit an object, the rays were traced no further. Whitted continued the process. When a ray hit a surface, it could generate up to three new types of rays: reflection, refraction, and shadow. A reflected ray continues on in the mirror-reflection direction from a shiny surface. It is then intersected with objects in the scene and the closest object it intersects is what will be seen in the reflection. Refraction rays travelling through transparent material work similarly, with the addition that a refractive ray could be entering or exiting a material. The shadow ray is used to test if a surface is visible to a light. A ray is traced between this intersection point and the light. If any opaque object is found in between the surface and the light, the surface is in shadow and so the light does not contribute to its shade. This new layer of ray calculations added more realism to ray traced images, however the time to calculate all ray paths in the scene meant it could take many minutes or hours to produce just a single frame.

With the advancement in graphics hardware, with fixed function pipelines replaced with programmable vertex and fragment processors, the graphics pipeline evolved into a general programmable stream processor capable of more than simple triangle rendering. Purcell et al. showed how ray tracing could be mapped to graphics hardware in an efficient way using textures to store the scene structure, and a uniform grid to reduce intersection tests [PBMH02]. They note that the uniform grid enables constant-time access to the grid cells and takes advantage of coherence using the blocked memory system of the GPU. However, it is not an optimal acceleration structure for scenes with non-uniform distributions of geometry.

Relative performances of acceleration structures in ray tracing have been studied widely. Havran [Hav00] compares a number of acceleration structures across a variety of scenes and finds that the k-d tree is the best general purpose acceleration structure for CPU ray tracers. Foley et al. [FS05] applied a k-d tree acceleration structure to GPU ray tracing and showed how a k-d tree yields far better performance than a uniform grid for complex scenes.

The first implementation of a real time ray tracer was credited at the 2005 SIGGRAPH computer graphics conference as the REMRT/RT tools developed for the BRL-CAD solid modelling system [Muu87]. The BRL-CAD ray tracer is the first known implementation of a parallel network distributed ray tracing system that achieved several frames per second in rendering performance. Since then, there have been considerable efforts and research towards implementing ray tracing in real time speeds for a variety of purposes. This research has been extended into the field of molecular graphics by Marsalek et al. [MDG+10], who worked on the ray tracing component of BALLView.

BALLView is the Biochemical Algorithms Library's standalone molecular modelling and visualisation application. It offers standard visualisation models for atoms,

bonds, and surfaces as well as grid based visualisation. At CEBIT 2009, BALLView was prominently presented as the first complete integration of real time ray tracing technology into a molecular viewer and modelling tool, using RTfact: a real time ray tracing library. For best results however, a very high specification computer is required to produce an interactive frame rate.

A drawback of conventional ray tracing is the unrealistically sharp nature of reflections and shadows, due to the use of single rays to sample these domains. Distributed ray tracing, also called distribution ray tracing and stochastic ray tracing, proposed by Cook et al. [CPC84] takes a different approach. It uses multiple rays to sample area lights, glossy and diffuse reflection together with other effects, and allows for the rendering of "soft" phenomena such as soft shadows.

Shadows in traditional ray tracing are discrete, limited by the use of a single shadow ray per pixel. It produces shadows with sharp edges, known as "hard" shadows. Shadows in the real world transition from fully shadowed to partially shadowed in a gradual fashion. This is due to the finite area of real light sources, and scattering of light from other surfaces. By modelling light sources with a finite area, and using a set of rays that are cast in a random fashion about the projected area of the light source, the amount of light transmitted from the source to the surface can be approximated by the ratio of rays that hit the light source to the total number of rays cast. Of course, many rays are needed to produce smooth results, and can therefore lead to lengthy render times.

An alternative approach, path tracing, proposed by Kajiya [Kaj86], allows for interactive previewing of a scene. Instead of expensive but accurate estimates of the radiance from surface interactions seen in distributed ray tracing, path tracing combines a large number of cheap but individually inaccurate samples to produce a good estimate. It can produce very fast but noisy initial results. It then refines them

over time as the number of samples increases.

As the power of CPUs and GPUs increased, so did the interest in ray tracing and path tracing algorithms for real time applications. This was aided by the maturing of GPGPU programming toolkits such as CUDA [Cor] and OpenCL [Groa] and GPU ray tracing SDKs such as OptiX [PBD+10].

Ray tracing is just one of many methods to calculate illumination in a scene. Photon mapping, developed by Jensen [Jen96], is a two-pass global illumination algorithm that approximately solves the rendering equation. Rays from the light source and rays from the camera are traced independently until some termination criterion is met, then they are connected in a second step to produce a radiance value. In the first rendering pass, packets of light, or photons, are cast into the scene from a light source. When a photon hits a surface, the intersection point and incoming direction are stored in a cache called a photon map. In the second rendering pass, the scene is rendered by a Monte Carlo ray tracer, with the photon map used to estimate the radiance of each pixel to produce the final image. Figure 2.5 illustrates the algorithm. The advantage of photon mapping is that it is easier to solve global illumination problems that are traditionally hard to solve from the observer's standpoint, for example: intersecting a small light source with random bounces, or simulating the refraction of light through a transparent substance such as glass or water.

Radiosity is another popular algorithm to calculate global illumination in computer graphics. Introduced by Goral et al. [GTGB84], it divides the scene into smaller surfaces, or patches. All the patches are considered to be diffuse reflectors, emitters or a combination of both. A view factor is computed for each pair of patches, which defines how well each patch can see each other. This indicates the amount of radiant light energy that can be transferred between the patches. The smaller the patches are the more accurate the result will be, at the cost of longer computation time. Figure

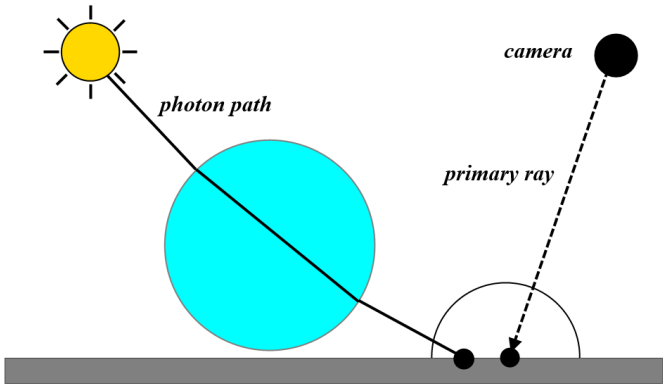



Figure 2.5: Due to the random paths of the eye rays in the second pass, they are unlikely land on the exact spots stored in the photon map. The solution is to average several photons hits within a circumference about the eye ray's intersection point.

2.6 illustrates the radiosity method. The downside with radiosity is that unlike rendering methods such as path tracing, which handle all types of light paths, radiosity only accounts for paths which leave a light source and are reflected diffusely before hitting the eye. Therefore radiosity based methods are generally not used to solve the complete rendering equation.

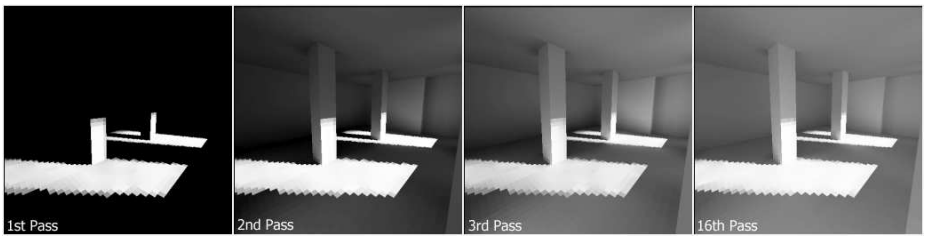



Figure 2.6: The radiosity algorithom applied to a scene. Each pass adds additional illumination to the scene. Image by Hugo Elias.

Real time rendering means ensuring each rendered frame is computed fast enough to allow the viewer to interact with the virtual environment. A variety of tools and libraries are available to help produce graphics in real time, helping to take advantage of modern graphics hardware. New graphics techniques have allowed higher quality

visual results to be achieved, while still maintaining a 30 Hertz frame rate, generally considered the minimum for interactive applications.

QuteMol is an open-source molecular visualisation program that focusses on applying modern rendering techniques to achieve a comprehensive array of visual effects [TCM06]. It makes heavy use of modern graphics card features including programmable shaders and frame and vertex buffers, to help achieve these effects at real time rates.

The program uses two popular molecular representations for its visualisations: space-fill, where atoms are represented as spheres, and ball-and-stick, where bonds between atoms are also displayed. An efficient method to render these models is vital considering molecules can contain many thousands of atoms. QuteMol uses a technique called "impostors" to greatly reduce the amount of geometry needed to represent atoms. An impostor is a term used to describe the use of 2D geometry to subtly replace a real 3D object in a scene. The impostor will always face the camera, and therefore appears to look natural from any angle. QuteMol's impostors are procedurally calculated in the fragment shader, meaning they look perfect regardless of how enlarged they are on the screen. The reduction in triangles needed to represent an object translates into significant performance gains, especially for large molecular structures, meaning more rendering time can be spent on complex lighting effects.

Efficient methods used by Botsch and Kobbelt [BK03], and Guennebaud et al. [GP+03] need just one vertex call to generate an impostor. Further improvements by Sigg et al. [SWBG06] introduce methods to compute a tight screen-space bounding box of the quadric in the vertex shader to allow for perspective warping seen in tessellated geometry. For each fragment generated during the rasterisation of this box, the ray-quadric intersection is computed in the fragment shader to classify pixels belonging to the shape. To avoid lighting calculations for fragments which are subsequently

overdrawn (a drawback of this technique), deferred shading [DWS+88] is employed to reduce lighting calculations needed to one per pixel.

QuteMol offers a range of non-photorealistic effects with the goal of producing clearer, more informative images of molecules. There are many of these effects, inspired by artistic styles such as painting or drawing, that can be found in feature films as well as video games. QuteMol shows how some of these effects, particularly edge highlighting, can also be used for scientific visualisation to enhance an image. For example, a thin border around the edge of atoms can be helpful in detecting atom intersections (top left of Figure 2.7). Simple border detection is achieved by identifying polygons that are facing away from the viewer, and colouring them a darker shade. Depth-aware borders attempt to bring more information about the structure in an initiative way, with line thickness increasing at atom borders as the distance between the atoms increases. This effect is show in the top right image of Figure 2.7. QuteMol also features a depth-aware halo effect, where halos irradiate from each atom, again the strength of the effect indicating depth distance. They are similar to the depth-aware border effect, but it maps z-distance over transparency rather than line thickness. This is illustrated in the bottom left image of Figure 2.7. Unlike photorealistic techniques, these effects are generally fast to compute, taking advantage of fragment shaders to perform fast per-pixel effects.

As well as offering non-photorealistic effects, QuteMol uses many of the common shading techniques, including Phong shading and self-shadowing using shadow maps. However, it also includes more advanced methods, namely ambient occlusion, that is considered a challenge for real time rendering. Ambient occlusion, explored further in Section 2.4, proves to be particularly useful in improving image clarity. In QuteMol, the occlusion factors are computed on the GPU, stored in a specially formatted texture to allow use with the impostors technique, then used at run-time to

cover atoms' surfaces. Figure 2.8 shows a comparison of the lighting methods available on a space-filling model. The left image is rendered using the basic lighting equation, with the lack of depth perception clearly being an issue. The center image shows how self-shadowing, similar to RasMol's rendering algorithm, can improve depth cueing. However the right image shows how ambient occlusion can provide significant improvements to depth perception. The lower image of Figure 2.8 shows a mix of non-photorealistic techniques as well as simple global illumination models to greatly improve the perception of the model, and provide helpful images for research purposes.

This thesis focusses on two graphical techniques that have already been shown to improve realism and depth perception in molecular graphics applications: ambient occlusion and cast shadows. There has been a vast amount of research into both by the graphics community, and we will now look at the development of both in greater detail.

## 2.4 Ambient Occlusion

Ambient occlusion is a technique that approximates the amount of indirect light reaching a point on the surface of an object [ZIK98]. The amount of light reaching the surface is based on how much light is being occluded by other objects in the world. Unlike local lighting effects where only light coming directly from a light source is taken into account, ambient occlusion is a global method, taking into account other geometry in a scene. It can therefore produce effects otherwise unattainable to standard lighting methods, such as contact shadows. Although it is a very basic approximation to full global illumination, it can be calculated far faster, and is still able to enhance the visual quality of the scene. This has made it a popular and well researched technique, used in graphics applications such as video games and molecular

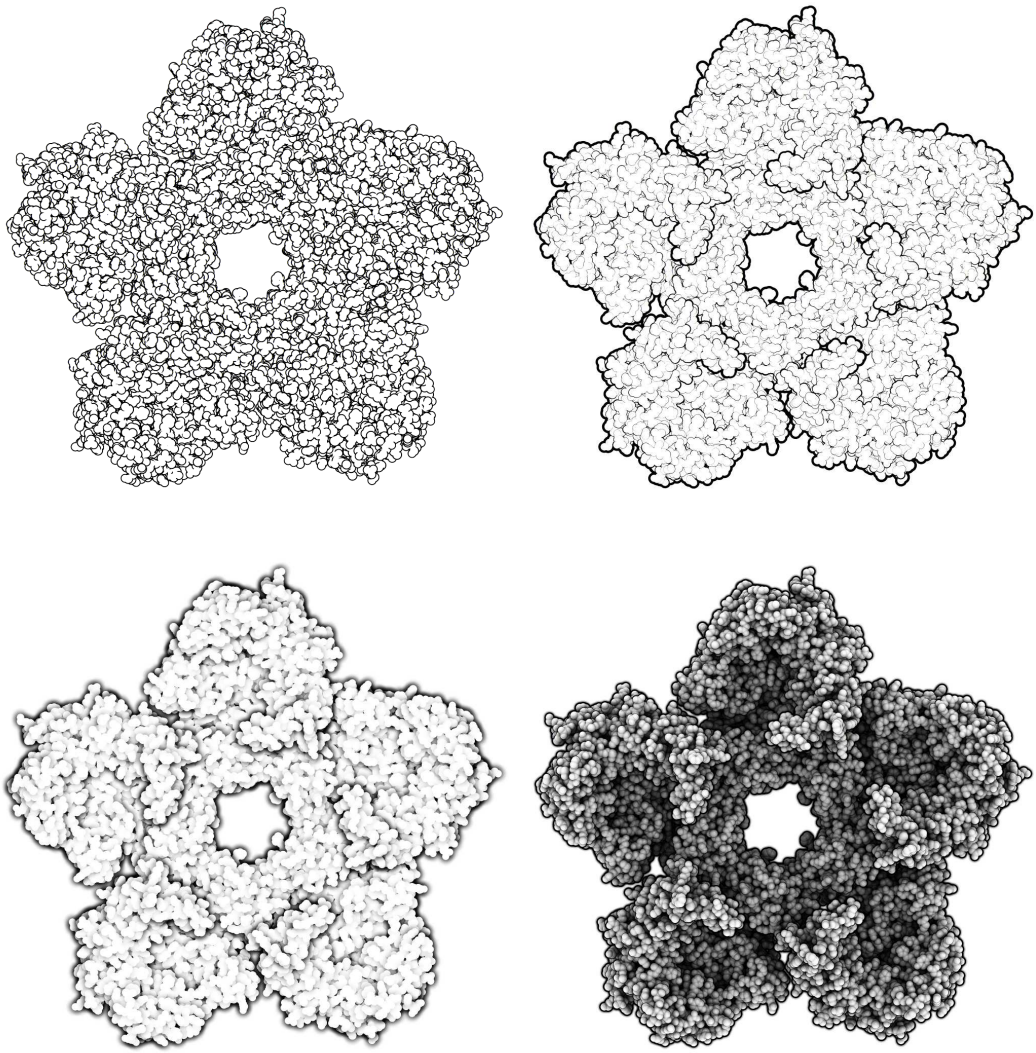


Figure 2.7: Four images showing QuteMol's array of post-processing techniques to enhance depth perception. The top left image shows thin borders around each atom to help in detecting atom intersections. The top right image shows depth aware borders with line thickness increasing based on distance. The bottom left image show a similar effect, but uses halos with varying transparency to achieve the same result. Finally the bottom right image shows these effects combined with ambient occlusion to greatly aid the users perception of the molecule.

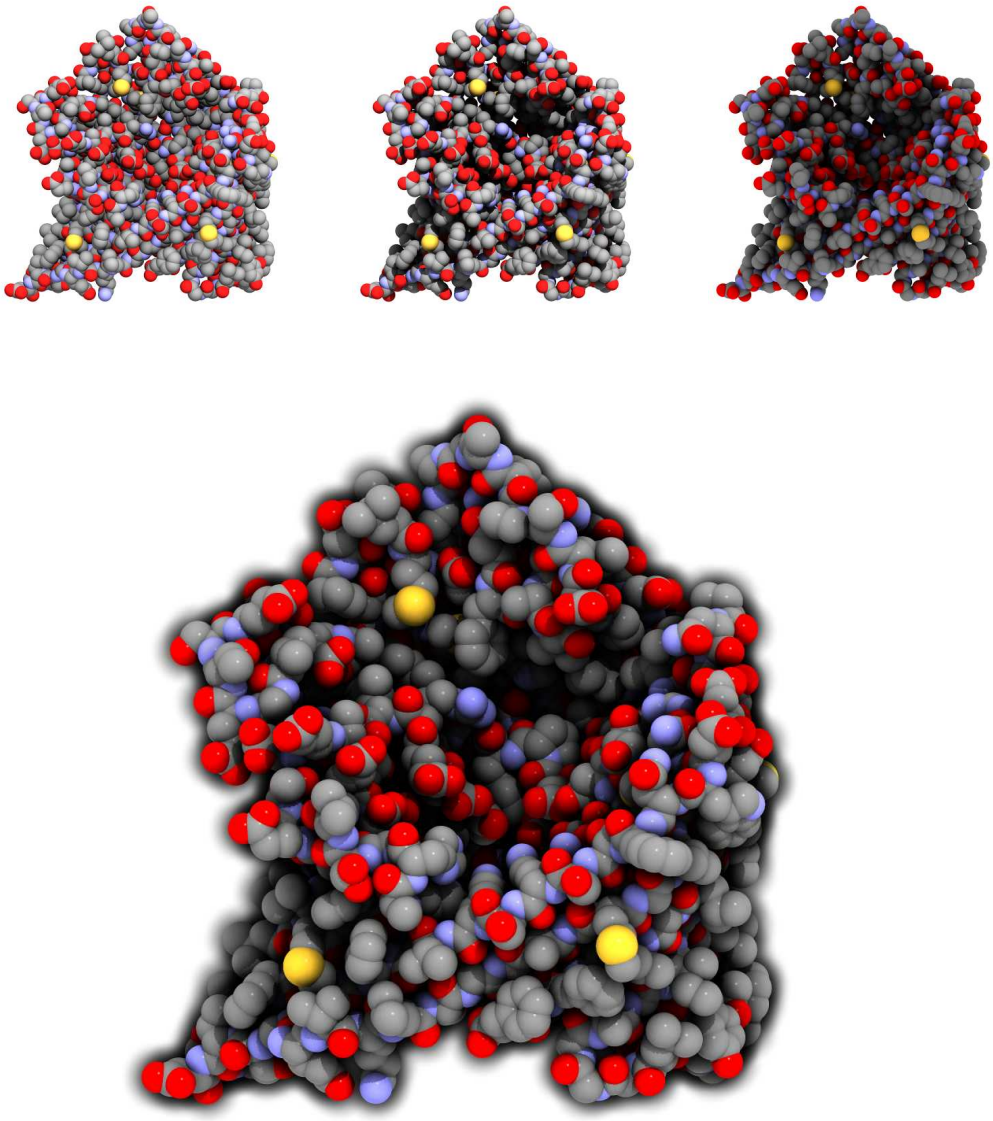


Figure 2.8: Three illustrations comparing the same molecule rendered using QuteMols standard rendering (left), self shadowing (middle) and ambient occlusion rendering modes (right). The right image uses ambient occlusion to darken areas of the molecule that light cannot access, therefore giving a better perception of depth and creating a better 3D appearance to the molecule. The bottom image combines all of what QuteMol offers to provide the user many cues to the structure of the molecule.

visualisation programs.

Ambient occlusion helps to give perceptual clues to the spatial proximity of geometry in a scene. Langer and Bülthoff [LB00] found that the depth discrimination under diffuse lighting is superior to that predicted by standard direct illumination. They found combining static ambient and dynamic lighting using a simple multiplication gives perceptually plausible results at high frame rates.

The occlusion $A_{\bar{p}}$ at a point $\bar{p}$ on a surface with normal $\hat{\mathbf{n}}$ can be computed by integrating the visibility function over the hemisphere $\Omega$ with respect to projected solid angle:

$$A_{\bar{p}} = \frac{1}{\pi} \int_{\Omega} V_{\bar{p},\hat{\omega}} \, (\hat{\mathbf{n}} \cdot \hat{\omega}) \, d\omega \tag{2.4.1}$$

where $V_{\bar{p},\hat{\omega}}$ is the visibility function at $\bar{p}$, defined to be zero if $\bar{p}$ is occluded in the direction $\hat{\omega}$ and one otherwise, and $d\omega$ is the infinitesimal solid angle step of the integration variable $\hat{\omega}$. The classical ambient term [Pho75] is commonly combined with $1 - A_{\bar{p}}$, improving the visual quality greatly over its traditional constant value.

A vast number of techniques have been developed that can be used to approximate this integral. The most simple and accurate method is the use of the Monte Carlo method by casting rays from the point $\bar{p}$ in a hemisphere around the surface normal, and testing for intersections with other geometry in the scene. After a large number of rays have been cast, we get an accurate approximation of the occlusion value at this point.

One of the first applications of ambient occlusion in production rendering was for spatial effects in motion pictures [Lan02], using commercialised software [Chr02]. This software uses pre-rendered occlusion maps accessed when the scene needs to be rendered. Ray traced ambient occlusion is also available for offline renders in molecular graphics software, such as PyMol, for high quality publication images. While it

is simple to implement these techniques for offline rendering applications, doing so for real time dynamic scenes has proven to be difficult. We will look at methods that attempt to speed up occlusion calculations to enable real time performance, noting if they have been used in molecular graphics applications.

### 2.4.1 Geometry Based Methods

Techniques to calculate ambient occlusion can generally be categorised into two distinct areas: geometry based methods and image based methods. Geometry based techniques, the focus of this section, are methods where occlusion calculations use the full information of the 3D geometry in the scene. Computing the occlusion at this stage avoids view dependant errors seen in image-space methods, although performance can be impacted for scenes with large complexity.

Pharr and Green [PG04] use techniques developed for offline rendering by Landis [Lan02], pre-processing the scene to compute the accessibility at each vertex. The accessibility value is then interpolated for each pixel in the fragment shader, giving a smooth occlusion effect with minimal run-time overhead. As these values are pre-calculated, the method will not produce the correct lighting results for dynamic meshes, limiting this technique to static scenes. The pre-processing of the accessibility values is performed using ray tracing, which can lead to lengthy pre-processing time for larger scenes. For a scene containing 150,000 triangles, the pre-process took approximately four minutes, using 512 rays per triangle to compute the accessibility values.

Sarletu and Klein [SK04] present a similar technique for calculating self-occlusion of static meshes. Their work differs by computing the accessibility by rendering the object from a number of directions positioned uniformly over the bounding sphere of the object. The vertex accessibility from these positions is found using occlusion

queries to take advantage of graphics hardware. The downside to this technique is the need to render the object many times to generate occlusion uniformly, however this method is faster than the software ray tracer in the previous method. This method has been implemented into QuteMol's molecular visualiser as a fast way to compute occlusion for static molecules, with successful results.

To enable occlusion for dynamic scenes, Bunnel [Bun05] proposes a disc-based approach to approximate faces in a mesh. This method again requires a large pre-computation step to calculate the disc elements from the polygon data. For each vertex, shadow values from all emitters are gathered to determine the ambient occlusion. Figure 2.9 illustrates this method. Gathering the visibility from all emitters can cause areas that are too dark, because elements that are in shadow can themselves cast shadows. A second pass is used to correct this overshadowing by multiplying each form factor by the emitter element's accessibility from the last pass, meaning elements that are in shadow will cast fewer shadows on other elements. As the visibility values are estimated per-vertex, highly tessellated meshes are needed to produce higher quality detailed occlusion. Although this method is able to produce dynamic occlusion in real time, performance is again limited by the geometric complexity of the scene.

Similar themed approaches from Ren et al. [RWS+06] and Shanmugam and Arikan [SA07] use spheres instead of discs as proxy occluders. Both these approaches approximate complex geometry with spheres to compute a directional visibility function. As proxy geometry is far simpler than the original, this method will miss high frequency occlusion effects that can help define small details. This method also tends to suffer in performance when there are a large number of objects. Shanmugam and Arikan split up the ambient occlusion calculations into two different parts: a high frequency approximation for nearby surface detail in image space, and a low frequency

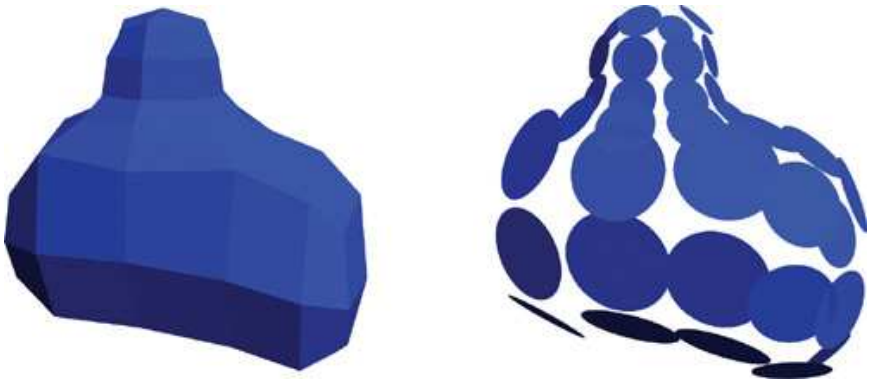


Figure 2.9: Disk-based ambient occlusion technique by Bunnel [Bun05]. Disc elements are pre-calculated from mesh data, and used to analytically calculate ambient occlusion values per vertex at run time.

approximation using spherical occluders for a soft occlusion effect caused by distant objects. Defining occlusion as two distinct areas has been employed successfully in work by Reinbothe et al. [RBA09], where rays are cast against a voxelized scene for low frequency effects, then high-frequency features are highlighted in a screen space occlusion pass.

A novel method by Kontkanen and Laine [KL05], Ambient Occlusion Fields, computes inter-object occlusion in real time. For each occluding object, they pre-compute a 3D grid in the surrounding space that encodes an approximation of the occlusion caused by the object. This volumetric information is then used in the fragment shader at run-time to quickly determine an occlusion value on the receiving object. While this method achieves real time performance, it has similar issues to other methods we have seen, specifically long pre-computation time and large amounts of memory from the usage of a large number of textures.

We have seen a number of the popular geometry based ambient occlusion methods. These methods each have their advantages in fulfilling specific roles, although a common theme is the limitations in performance from geometric complexity, as well as pre-processing being required for many of these methods. We will now assess the

second major category of ambient occlusion techniques that calculate occlusion in screen space, which have been a popular topic of research due to their simplicity and support for arbitrary scene complexity.

### 2.4.2 Image-Space Techniques

Screen Space Ambient Occlusion (SSAO) is a rendering technique to approximate the ambient occlusion factor by analysing the scene's depth buffer. It was developed by Mittring [Mit07] and was used for the first time in a video game in the 2007 PC game Crysis [Fra]. The algorithm runs in a fragment shader for each pixel on the screen, sampling the depth values around the current pixel and computing the amount of occlusion from each of the sampled points based on differences in depth. The resulting approximation is relatively successful in adding an ambient occlusion effect to a scene in real time, and has made it a very popular choice in a number of interactive applications such as video games.

Figure 2.10: Resulting image from Mittring's implementation of SSAO [Mit07]. Incorrect shading of geometry is clearly visible due to over occlusion, resulting in an overly dark image. This is corrected in later implementations.

Mittring found that by comparing depths of surrounding pixels, it was possible to compute a darkening factor to create silhouettes around objects. By limiting this

effect to nearby receivers, a local ambient occlusion effect could be produced. To create an image with high visual quality, a large number of samples would be needed per pixel, which would not be acceptable for a real time application. To reduce the number of samples, they suggested randomly distributing sample positions around the origin in a sphere. The high-frequency noise that is produced is then blurred, using information from the depth buffer to preserve edge details. This method allows a reduction in the number of depth samples per pixel, eight being used in this algorithm, while maintaining performance and visual quality.

There are a number of key advantages SSAO has over the ambient occlusion methods we have previously discussed. Firstly, as the occlusion factor is calculated in screen space, the process can be done entirely on the GPU taking advantage of its large processing power. The process is also completely dynamic, requiring no pre-computation, meaning no loading times and no requirements for system memory. This allows the technique to be integrated very easily into a modern pipeline. Finally, the method is completely independent of scene complexity, ideal for scenes with high numbers of polygons where other methods would struggle to achieve real time performance.

Of course, there are notable disadvantages to this method that stop it becoming the preferred choice in certain applications. The technique is still a relatively expensive process after considering the need for depth aware blurring stages to smooth the output before lighting calculations happen. The effect is view dependant, not able to take into account geometry that is not rendered on the screen, such as geometry clipped by the frustum or discarded due to back-face culling. The technique is also only effective for local occlusion, with a large sampling radius proving too costly due to missed cache reads in the GPU hardware.

Whilst Mittring was the first to introduce SSAO, there have since been a vast

number adaptations of the original algorithm to tackle some of the disadvantages listed above. Filion and McNaughton [FM08] introduce a method inspired by the work from Crytek, with key changes made to improve both image quality and performance.

A problem with Crytek's original technique is self-occlusion. Using random offset vectors generated in a sphere around the test point leads to some samples penetrating through the surface of the object itself. Filion and McNaughton solve this issue by generating vectors around a hemisphere centred around the normal at that point on the screen. This approach does not produce incorrect shadowing that occurs in Crytek's technique, seen in Figure 2.10, but does require an extra buffer to store the image space normals. However, as normals are commonly stored as well as depth in a deferred rendering pipeline, this is generally of little concern. They also found the main bottleneck of their algorithm to be texture sampling, noting how increasing the sampling area size reduces the GPU's texture cache effectiveness yielding poorer performance. To combat this, they down sample the depth buffer to a quarter of the original size, improving texture cache performance significantly at the cost of a loss in detail.

Horizon Based Ambient Occlusion (HBAO), introduced by Bavoil et al. [BSD08], takes another approach to estimating occlusion values. This technique uses the depth buffer as a height map, calculating the amount of occluding geometry for each point by searching for the horizon. The horizon is defined by the average slope in the height field around this point, with heavily occluded points having a steep horizon angle. Sample directions around the point of interest are chosen in image space and sample points are created along those directions, producing a circular sampling area around the pixel. This method produces higher quality occlusion effects than standard SSAO, but is more expensive to compute.

Another novel technique described in their research includes the use of an angle

bias, which is used to remove artifacts that sometimes appear when geometry is rendered at low tessellation. The bias allows occlusion near the tangent plane of the point's hemisphere to be ignored. This geometry could provide false occlusion values, which would previously have been difficult to exclude using older methods.

Like Filion and McNaughton, they source a half resolution depth buffer for the occlusion calculations. They then render the occlusion pass at half resolution as well, whilst blurring at full resolution to avoid bleeding across edges. They apply the cross bilateral filter, developed by Kopf et al. [KCLU07], separately in the X and Y directions. Although it is a non-separable technique, using this method greatly reduces the performance cost for a small drop in blurring quality.

Bavoil and Sainz [BS09] build on the horizon-based occlusion algorithm, attempting to solve view dependant issues seen in other SSAO algorithms. They use multiple depth layers, with a technique called depth peeling, in an attempt to remove view dependant artifacts caused by the use of a single depth layer. The technique takes samples from all the depth layers for each test point. By taking the largest occlusion value from all the layers, they greatly reduce the issue of missing information behind the Z-Buffer in the vast majority of cases, providing there are enough depth layers.

Bavoil and Sainz also introduce a method to solve missing occlusion caused at the edges of the screen, due to the lack of depth information outside the view frustum. They suggest enlarging the depth images and the field of view, using a simple constant value, although this will increase time to render the depth image by a small amount. To help increase the performance of the technique, they also suggest a max footprint parameter, discarding samples outside this area, giving a performance boost due to better caching.

Volumetric Obscurance published by Loos and Sloan [LS10], improves upon the SSAO technique by making better use of each depth buffer sample. Instead of treating

samples as points, with a binary comparison between the depth buffer and the sampled depth, each sample is treated as a line sample. This is used to estimate occlusion as proportional to the volumetric occlusion of a point. However, the performance of this technique is not competitive with other methods we have seen.

As we can see, there has been a vast amount of research into screen space methods to calculate ambient occlusion. This is because of its desirable traits such as predictable rendering time, ease of integration, and being completely dynamic. The separation of rendering time from geometric complexity makes it desirable for real time applications with dynamic scenes, and therefore has become a very common technique employed in video games, but has yet to be integrated into any popular molecular graphics programs.

## 2.5 Shadows

Shadows occur when an object partially or fully blocks light falling on another surface. Unlike ambient occlusion, where light is considered to have no specific direction or source, we commonly tend to think of shadows that are cast from a clear origin, with a consistent direction, such as the sun or a torch light. Shadows are important in computer graphics to help create a realistic image of a scene. They provide the user with important visual cues about the 3D structure of objects, and their relative positions to each other, to enhance a user's depth perception of the world.

Depth perception is defined as the visual ability to perceive the relative distance of objects in one's visual field. It arises from a variety of depth cues, commonly split into two categories: binocular cues, providing depth information when viewing a scene with both eyes, and monocular cues, providing depth information when viewing a scene with one eye. Binocular cues include binocular disparity and convergence.

Monocular cues include occlusion, perspective, size, reference frames, as well as lighting effects including the shadows that are cast by objects, providing an effective cue for the brain to determine the shape of objects and their position in space. Figure 2.11 illustrates how powerful shadows can be in providing depth cues in a scene.

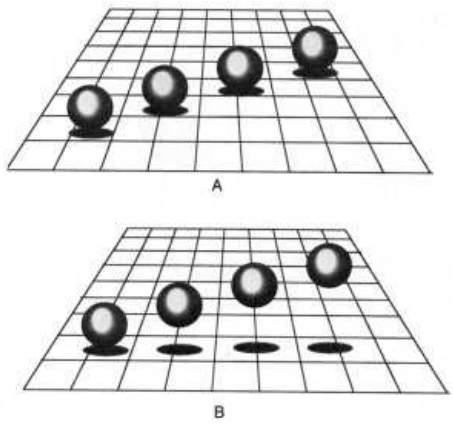



Figure 2.11: An example of how shadows are able to change our perception of an objects position in space. Without shadows, the balls would appear to be in the same positions. However, with shadows, the balls are clearly in different positions in the two images.

There have been numerous studies into the role shadows play in helping us understand the 3D world. Wanger et al. [WFG92] assessed the influence of pictorial cues on the perceived spatial relations in computer generated images. They found that hard shadows provided the dominant cue for spatial and scaling tasks compared with other common visual cues such as including projection, motion and frames of reference. This work was further developed by Wanger [Wan92], to assess the effect of shadow quality on the perception of space. Their results indicated that less physically accurate hard edged shadows may be preferable in tasks requiring accurate perception of an object's shape.

When an object moves, the apparent relative motion against a stationary object gives hints about their relative distance. Kersten et al. [KMK94] analysed how cast

shadow motion provides information for the inference in object motion, and measures the human observers' use of this information. They demonstrate that simply adjusting the motion of a shadow is enough to dramatically change the apparent trajectories of the shadow casting object. Mamassian el al. [MKK98] also found a shadow's motion to have a profound effect on the spatial understanding. They also show how shadows can be informative about the shape of the object, the shape of the object receiving the shadow, and the spatial arrangement between the two.

Clearly these psychophysical experiments strongly show that shadows play a vital role in computer graphics to produce an enhanced visual representation of the scene. Since the survey of shadow algorithms by Woo et al. [WPF90], there has been dramatic development in both graphics hardware and technology allowing shadows to become part of real time applications, including molecular graphics. However, recreating realistic shadows is still a difficult task, and still generates much research.

Ray tracing is perhaps the easiest method to comprehend when adding shadows to a scene. A shadow ray is traced from each surface point toward each light. If any opaque object lies between these two points, then the surface is in shadow. Shadows created using this technique were first seen in work by Whitted [Whi80], but whilst graphics hardware has advanced rapidly, creating ray traced shadows at real time speeds is still a challenge today. We will now turn to investigate popular rendering techniques employed in modern applications to recreate real time shadows in a graphical environment.

### 2.5.1 Shadow Volumes

Shadow volumes, proposed by Frank Crow [Cro77], is a method that calculates the geometry of the area occluded from a light source to determine shadowed areas. The basic process to calculate the shadow volume is to first find all silhouette edges of the

mesh. These are the edges that separate front facing and back facing faces. These edges are then extended in the direction away from the light source to a point at infinity. The volume is then capped at the front or back depending on the implementation. If a point lies within this volume, then it is in shadow of the light. Figure 2.12 illustrates the shadow volumes of a basic scene.

Shadow volumes were a popular method for real time shadowing, and while shadow maps have become the more popular choice, shadow volumes still have their advantages. The main being they are accurate to the nearest pixel unlike shadow maps, that suffer aliasing due to limitations in texture resolution.

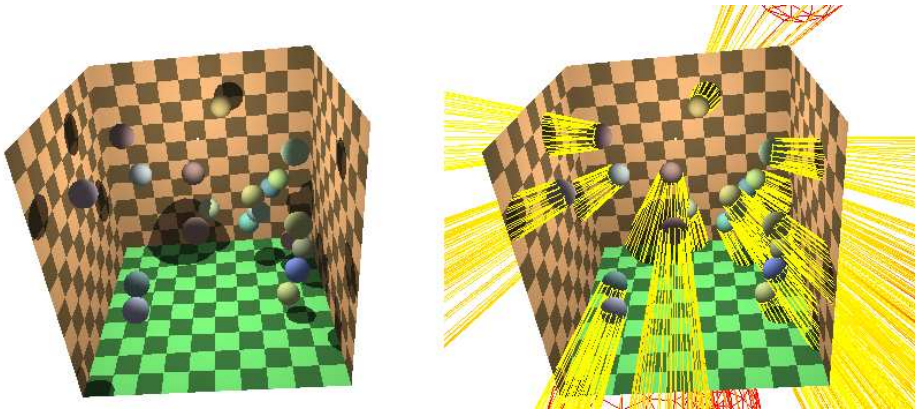


Figure 2.12: An illustration of the shadow volume technique. The yellow lines indicate the triangle meshes that represent the shadow volumes. If a point lies inside a mesh, then it is said to be in shadow.

The original algorithm was adapted by Heidmann [Hei91] to use the stencil buffer to compute the per-pixel count for the point in volume test. Front facing areas of the shadow volume increment the stencil buffer, and back facing areas decrement it. After rendering all the volume, the stencil buffer holds a mask that indicates areas of shadow. However, their method does produce incorrect results when the viewport intersects with the shadow volume. Everitt and Kilgard [EK03] resolved this by moving the far clipping plane to infinity, producing a more robust and hardware

accelerated technique.

There are, however, issues with the shadow volume technique that are still present today. The additional geometry needed for the volumes is costly not only for the CPU but also in fill rate time, as these polygons tend to cover large portions of the visible scene. These issues are compounded for more complex shadow casters. There are also cases where shadow casters do not have a mesh that accurately represents their shape, such as billboards. As shadow volumes are based on the object's mesh, the shadow of the object will not be represented correctly. Finally, shadow volumes can not produce realistic soft shadow effects seen in real-world environments, which are important for creating a realistic visualisation. A technique called Shadow Mapping takes a different approach to creating shadows in computer graphics, which we will now look at in detail.

### 2.5.2 Shadow Mapping

Shadow mapping is one of the popular techniques to add shadows to a 3D scene. Introduced by Lance Williams [Wil78], it has been used extensively in real time and offline graphics, including many popular molecular graphics applications such as QuteMol and RasMol. The technique requires at least two stages. The scene is firstly rendered from the light's point of view, with the depth buffer saved to a texture. The scene is then rendered again from the camera's viewpoint, with each pixel tested against the depth map of the light, after being projected into the right coordinates. If the z-value is greater than the stored value in the depth map, then the object can be considered behind the occluding object, and therefore can be drawn in shadow. Figure 2.13 gives a visual breakdown of this process.

There are many reasons as to why shadow mapping has become a very popular technique for shadow generation in graphics, especially for real time applications.

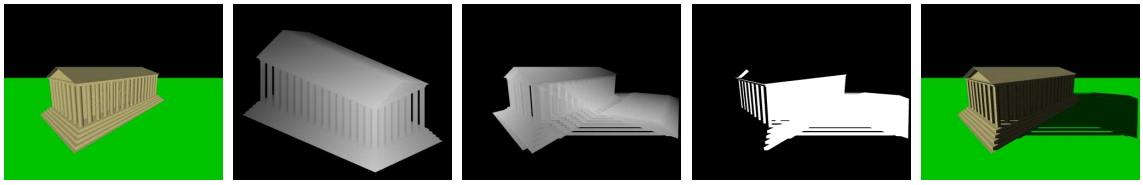


Figure 2.13: The shadow mapping process from left to right: the scene from the camera's viewpoint, the scene from the light's viewpoint, the light's shadow map projected onto the scene's depth, areas where pixels fail the depth test, and the final rendered scene with shadows.

They are very easy to implement, and need only a single texture to hold depth information for each light. There is no processing or knowledge of the scene's geometry, as shadow calculations happen in image space. This means the technique will work automatically for geometry created or altered on the GPU. Their cost is also less sensitive to geometric complexity than other techniques, which is vital in today's applications with ever increasing polygon counts.

There are disadvantages to the shadow mapping technique however. They are generally less accurate than shadow volumes, with the accuracy of the shadow map limited by its resolution. This drop in quality, especially for small shadow maps, is usually visible as aliasing at shadow edges where the projected shadow-map texels cover a noticeably large area in screen space. A desirable way to overcome this would be to increase the shadow map size, but given memory and computational constraints, is not always viable. The scene geometry must also be rendered once per light, and for point lights many more times to capture its omnidirectional depth. For scenes with many lights, shadows normally have to be restricted to the main light source in the scene, normally the sun. There has been much research to improve many of the problems listed here, and we will look at the more notable techniques in more detail.

A particular area of shadow map research looks at warping algorithms to tackle the problem of insufficient shadow map resolution in regions near to the eye. A technique

called Perspective Shadow Maps by Stamminger and Drettakis [SD02] attempts to wrap the light frustum to exactly coincide with the view frustum by applying standard shadow mapping in post-perspective space of the camera. This approach improves shadow quality in some cases, however encounters difficulties in dynamic environments. Light Space Perspective Shadow Maps by Wimmer et al. [WSP04] wrap the camera frustum in a way that leaves the directions of light sources unchanged. A new light frustum is built that is parallel to the shadow map, sized to include the camera frustum and potential shadow casters. This method is more stable than the Perspective method, but does not use the shadow map fully. Trapezoidal Shadow Maps by Martin and Tan [MT04] build a bounding trapezoid of the camera frustum instead of a frustum in the Light Space method to improve shadow map usage.

Another area of shadow mapping research looks at splitting the view frustum into multiple depth layers. A separate shadow map is then generated for each of these layers. Introduced by Zhang et al. [ZSXL06] and further developed by Dimitrov [Dim07], the motive behind this idea is the observation that points at different distances from the view need different shadow map sampling densities. By splitting up the frustum, each shadow map can focus in a smaller area and therefore provide a better match between sampling frequencies in view space and texture space.

The work described above looks at how best to choose shadow map samples to reduce artefacts, however, even aliasing free these methods still produce hard shadow effects. In order to better recreate real world soft shadow effects, several solutions have been developed. Percentage Closer Filtering by Reeves et al. [RSC87] is a method where multiple shadow map comparisons are made per pixel and are then averaged together, calculating the percentage of the surface that is not in shadow. The original algorithm used random samples over the region to be shaded. Bunnell and Pellacini [BP04] optimised this technique for modern GPU hardware, using a 4x4

sample region to reduce aliasing.

Variance shadow maps by Donnelly and Lauritzen [DL06] try to address the problem of efficiently filtering shadow maps. They note that standard shadow maps can only store the depth of a single point, and present a method to improve on this by representing a distribution of depths for each pixel. By doing this, the shadow map can be represented in a manner that can be filtered linearly, giving the ability to take advantage of graphics hardware's built-in capabilities such as mipmapping and anisotropic filtering. However, this technique has the potential issue of light bleeding, meaning areas of light occur that should be fully in shadow. Attempts have been made to fix this issue by Lauritzen [Lau07], and VSM certainly remains a promising direction for future research.

In this section we have looked at a variety of popular shadowing techniques and their developments over time. While ray tracing is used to create realistic and accurate shadows in offline renders, shadow mapping has become the dominant method for recreating shadow effects in real time applications. A combination of Cascaded Shadow Maps and Percentage Closer Filtering seems to be the current method of choice to produce shadows for current generation real time graphics applications, as they have been highly optimised for the current generation of GPUs.

## 2.6 Conclusion

The overarching goal of Molecular Graphics is to provide a scientist an enhanced way to study molecules and their properties through graphical representation. Graphical systems have been developed to provide the ability to manipulate a molecule by rotating its geometry and hiding atoms that obscure others. Advanced lighting methods have been adopted to provide a better sense of depth, with ambient occlusion and shadows being two of the more popular methods due to their simplicity in

comparison to a full global illumination model. In the next chapter, we will look at efficient methods to render molecules in the space-filling representation, to improve both performance and appearance in our HaptiMOL iSAS software.

# Chapter 3

# Techniques for the Rendering of Molecules in Space-Filling Representation

## 3.1 Introduction

In chemistry a space-filling model, also known as calotte model, is a type of 3D molecular model where the atoms are represented by spheres. The radius of the sphere is proportional to the radius of the atom, commonly defined by the van der Waals Radii [Bon64]. The center-to-center distances of the atoms are proportional to the distances between the atomic nuclei.

The colouring of atom types is important to help distinguish individual atoms in the molecule. Most molecular graphics programs use the CPK colour system, named after the designers of the original space-filling models created by Corey and Pauling in the 1950s [RBC53] and later revised by Koltun [Kol65]. This system provides a consistent standard for atomic colour for atoms of different chemical elements.

Our HaptiMOL iSAS software uses the space-filling representation, and CPK colour system, for its molecular visualisation. Knowing we are only working with spheres allows us to make several optimisations to the rendering process, whilst also enhancing the visual appearance of the molecules.

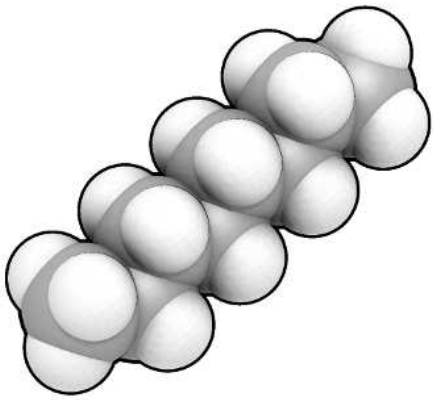


Figure 3.1: A space-filling model of n-octane $C_8H_{18}$

## 3.2 HaptiMOL iSAS

HaptiMOL iSAS is software that provides haptic rendering with molecular graphics to give a deeper appreciation of the shape of biomolecules by augmenting a user's sense of sight with touch. By using a spherical probe that is the same size as a water molecule, the process of exploration has the benefit of being able to determine regions on the molecular surface that are accessible to the solvent. This gives insight into how awkward it is for a water molecule to gain access to or escape from channels and cavities, indicating possible entropic bottlenecks.

The software uses the OpenGL 2.0 API to render the molecular structure, using display lists to render triangle meshes representing the spherical atoms. Phong shading is used for lighting, with two light sources illuminating the molecule. One light source is located at the centre of the probe, giving the user a better understanding of its position in relation to other atoms in the molecule. In this thesis, we aim to improve upon these rendering techniques currently used to display molecules to the user. This section focuses on how to render spheres that represent atoms efficiently.

Later sections look at adding advanced lighting effects to the software, including ambient occlusion and shadows, in the hope of further aiding the depth perception of the molecule to improve navigation of a biomolecule.

## 3.3 Ray Casting using 2D Billboards

A common way for molecular visualisation programs to render a sphere in a 3D scene is to generate a triangle mesh. This mesh would likely need hundreds of triangles to produce a smooth enough shape to appear spherical to the human eye. In some cases it may be viable to generate a sphere with an arbitrary number of triangles, and thus improve the approximation, but this will always be an approximation. Per-pixel lighting can also be used to help disguise a low polygon mesh with the smooth lighting effects it provides, however the individual triangles in the mesh may still be visible where the spheres intersect. In the general case, memory usage and rendering time have to be taken into account, therefore we would need to limit the number of triangles rendered to maintain an interactive frame rate, consequently hurting the visual representation.

A sphere can be defined as a set of points in space that are an equal distance from a common center point. A technique that is employed by some molecular visualisation programs such as QuteMol to approximate this definition is Impostors. This is where the geometric shape is simply a place-holder providing a way to invoke the fragment shader over a certain region of the screen.

Given a 2D square, we can invoke a fragment shader over this region, culling away fragments that are further away from the center of the square than the radius of the sphere. This square will be in the same position and have the same dimensions as the actual circle, and it will always face the camera. In the fragment shader, we then compute the position and normal of each point along the sphere's surface. By doing

this, we can map each point on the square to a point on the sphere we are trying to render. This technique produces spheres that appear perfectly round regardless of how close the sphere is to the camera. This also helps to drastically reduce the number of triangles per sphere from hundreds to just two. However, the problem with this technique is that the sphere will always appear perfectly circular to the viewer, no matter the viewing angle. The problem is illustrated in 3.2.

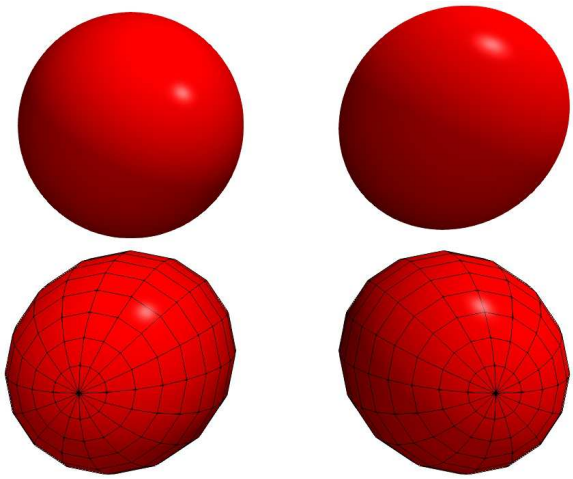


Figure 3.2: A comparison between traditional geometry (two bottom images), ray casting (top right), and creating a circle in window space (top left).

The technique described above, whilst producing seemingly perfect spheres, is simply creating a circle in window space. This will not reproduce the elongation effects seen on the real geometry in Figure 3.2. It is a good approximation if the spheres appear small on the screen. However, if the spheres are large, or close to the camera, then inaccuracies can be clearly seen with this method. We will instead use the ray casting technique to get the position and normal of a sphere for each fragment. This will also prove important in later chapters, where ray tracing is used for ambient occlusion and shadows. It will ensure the shadows cast by the spheres will match exactly the geometry being rendered.

Ray casting, first coined by Scott Roth in 1982 [Rot82], is a technique for generating an image by casting a ray through each pixel in an image plane, then intersecting these rays with objects in the scene. Once the nearest intersection point has been found, the final pixel colour is determined based on the incoming light and material properties of the object. Instead of performing the ray casting algorithm for each pixel of the screen, we perform it on each billboard. This removes the need for a acceleration structure to ensure the first object hit by the ray can be found quickly.

We can compute the intersections between rays and spheres using their parametric equations.

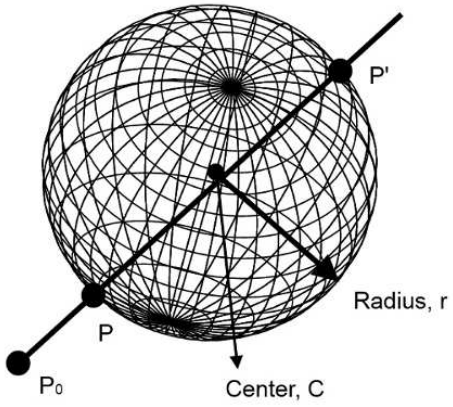



Figure 3.3: A ray with position $P_0$ intersecting a sphere with center $C$ and radius $r$ at points $P$ and $P'$.

$$\mathbf{P} = \mathbf{P}_0 + t\hat{\mathbf{v}} \tag{3.3.1}$$

$$|\mathbf{P} - \mathbf{C}|^2 - r^2 = 0 \tag{3.3.2}$$

We can substitute the ray equation into the sphere equation:

$$|(\mathbf{P}_0 + t\hat{\mathbf{v}}) - \mathbf{C}|^2 - r^2 = 0 \tag{3.3.3}$$

This can then be written in the form of a quadratic equation:

$$\hat{\mathbf{v}}^2 t^2 + 2\hat{\mathbf{v}} \cdot (\mathbf{P}_0 - \mathbf{C})t + |\mathbf{P}_0 - \mathbf{C}|^2 - r^2 = 0 \tag{3.3.4}$$

The equation can be solved using the quadratic formula. Equation 3.3.5 is the general form of the quadratic equation. We can the substitute in the values in order to solve:

$$at^2 + bt - c = 0 \tag{3.3.5}$$

$$a = |\hat{\mathbf{v}}|^2 \tag{3.3.6}$$

$$b = 2\hat{\mathbf{v}} \cdot (\mathbf{P}_0 - \mathbf{C}) \tag{3.3.7}$$

$$c = |\mathbf{P}_0 - \mathbf{C}|^2 - r^2 \tag{3.3.8}$$

Solving this quadratic equation for each fragment of the billboard will produce a sphere that is identical to one made up from traditional geometry, but will be pixel perfect regardless of its position to the camera. We can compare this technique with the more traditional triangle mesh that was previously being used to represent atoms in the HaptiMOL iSAS software. The time to render a single frame was measured, with no lighting effects used, and camera positioned to fit the whole structure on screen. The results are shown in Figure 3.4.

It is clear from Figure 3.4 that even the most poorly tessellated triangle mesh can not reproduce the performance of the billboard technique. This is especially apparent with larger molecules with many tens of thousands of atoms. Frame rates drop below what is considered interactive when using triangle meshes, but with billboards can remain interactive to the user.

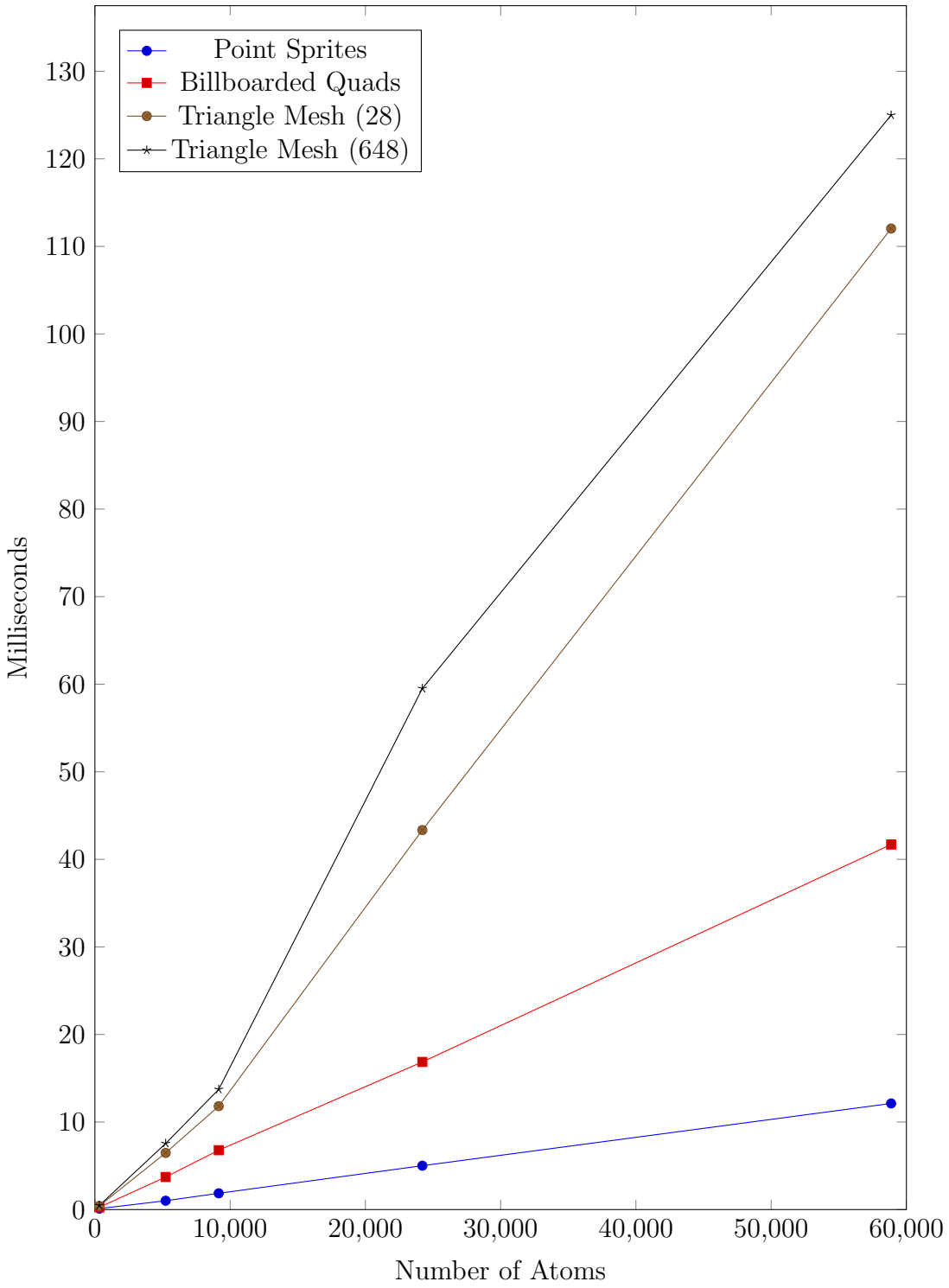



Figure 3.4: This graph compares the performance of Point Sprites, Billboards aligned on the GPU, and triangle meshes when rendering molecules with varying amount of atoms, from hundreds to tens of thousands.

### 3.3.1 Point Sprites

In order to create the camera aligned billboards described in section 3.3, we can calculate how to position the four points to align them to face the camera on the CPU. We then send these points to the GPU to render them. However, OpenGL provides an alternative way produce view aligned geometry on the GPU, using Point Sprites. This saves doing billboard calculations in the main application and reduces the number of vertices that have to be sent to the vertex program.

The comparison of the two rendering techniques, also shown in Figure 3.4, shows the advantage Point Sprites have over billboarded quads. This is due to the billboarding calculations taking place on the GPU for the Point Sprites, as well as less data needing to be transferred through the pipeline. However, a large drawback to Point Sprites arises when they are clipped from the screen. OpenGL discards the entire Point Sprite when the vertex is outside of the view frustum, which leads to a noticeable disappearance of atoms around the edges of the screen. This leads to the conclusion that both Billboard and Point Sprite rendering methods could be included into the software, with Point Sprites being used on lower end graphics cards where an interactive frame rate takes precedence over visual appearance.

## 3.4 Culling Molecules with a Clipping Plane at an Arbitrary Angle

The iSAS software allows the user to interact with a molecule with a probe sphere. As the user moves the probe into the pockets and channels of the molecule, the user can lose sight of it. If we are able to clip the structure away as the probe moves inside, we could always see the probe in relation to the protein structure. Another motivation for clipping planes includes being able to see channels that may have been impossible to detect without culling parts of the structure away. Other software packages either

do not provide clipping functionality, or do not employ a nice clipping strategy, and so the visual results can appear unnatural.

For our software, it would be desirable for a clipping plane to be fixed to the probe sphere, meaning as the user moves the probe into channels and pockets, spheres between the probe and the camera are culled away. This would ensure the user's view of the probe sphere is never restricted. It would be beneficial if the user has the option to cull the spheres entirely if they intersect the plane, or alternatively display the partially culled sphere as if it had been cut. This may be advantageous in cases where the user wants to know exactly how far the probe is in relation to other atoms in the molecule. This approach will also produce a more visually correct culling effect, where instead of whole atoms unnaturally popping out of view when they intersect the plane, the atoms gradually get smaller as the plane passes though them.

The improvements in the rendering of atoms in Section 3.3 show how 2D ray traced billboards can be used in place of traditional triangle meshes. This makes the implementation of rendering partially cut spheres easier, as we can use the intersection points of the ray-sphere intersection to our advantage in order to change the shape of the atom.

When a ray intersects a sphere, it has two points of intersection, where it enters and exits the sphere (unless the ray is tangent to the sphere, meaning there is just one intersection point). As a billboard will always face the camera, it is common when rendering spheres using this technique to only calculate the intersection point nearest to the camera, as we know the second intersection point will always be occluded. This approach will work fine when the spheres do not need to be culled, however the 3D effect breaks if fragments are discarded that are behind the plane, seen in the left image of Figure 3.5.

A solution to rendering spheres that have been cut by a plane at an angle can

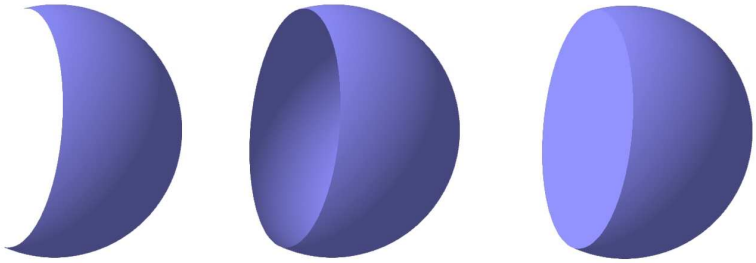


Figure 3.5: These images show spheres being cut by a clipping plane. The left hand image is only rendering the initial intersection point of the ray-sphere intersection. The center image is rendering both intersection points of the ray-sphere intersection. The right hand image shows how the sphere can be altered from appearing like an empty shell, to a more natural solid look.

be seen in Figure 3.6. As described above, a ray-sphere intersection produces two intersection points, $P_{min}$ and $P_{max}$, where $P_{min}$ defines the first intersection, closest to the camera, and $P_{max}$ defines the second intersection. If both $P_{min}$ and $P_{max}$ are behind the clipping plane, the fragment can be discarded, seen as the red section of Figure 3.6. If $P_{min}$ is behind the plane, but $P_{max}$ is in front, then we can render the fragment at position $P_{max}$, depicted as the green section of Figure 3.6. If both $P_{min}$ and $P_{max}$ are in front of the plane, we simply render $P_{min}$ shown as the blue section of Figure 3.6.

The result of calculating both $P_{min}$ and $P_{max}$ can be seen in the center image of Figure 3.5. However the appearance is still not what would be expected from a sphere cut in two. Most people intuitively imagine the atom modelled as "full" solid sphere rather than as a thin empty shell. To make the sphere appear solid, as in the right image of Figure 3.5, we simply calculate the distance from $P_{max}$ to the plane, in the direction of the camera, and move $P_{max}$ onto the plane accordingly. We also set the normal of the fragment to that opposite of the plane, to produce the correct lighting results. The result of this technique on a full molecule can be seen in Figure 3.8.

It will be important to ensure fully dynamic ambient occlusion and shadows when

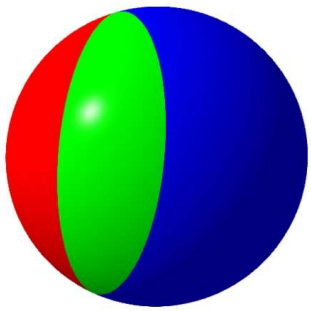


Figure 3.6: A diagram to show how a plane intersects with a 2D billboard representing a sphere. The red area is where the fragments can be discarded completely. The green area shows where the plane intersects $P_{min}$ but not $P_{max}$. The blue area of the sphere remains unchanged.

the atoms are culled, as these atoms no longer produce occlusion or cast shadows on atoms still visible. This will aid in producing a more visually believable scene, where lighting effects change dynamically in response to the user moving the clipping plane. Work on both dynamic ambient occlusion and shadows will be covered in later chapters. These lighting effects would not be possible in many other software packages such as QuteMol, where ambient occlusion effects are baked into textures in a pre-processing stage.

The results of this technique provide real time user interaction between a clipping plane and a molecule. A realistic visualisation is maintained by modelling the atoms to appear solid when culled by the clipping plane. This helps to produce a more immersive and useful molecular visualisation for the user.

### 3.4.1 Multiple Clipping Planes

As well as a clipping plane fixed to the probe sphere, it may also be desirable for the user to hide other sections of a molecule that they may not need to view. The algorithm above can support multiple clipping planes by moving $P_{max}$ onto the closest

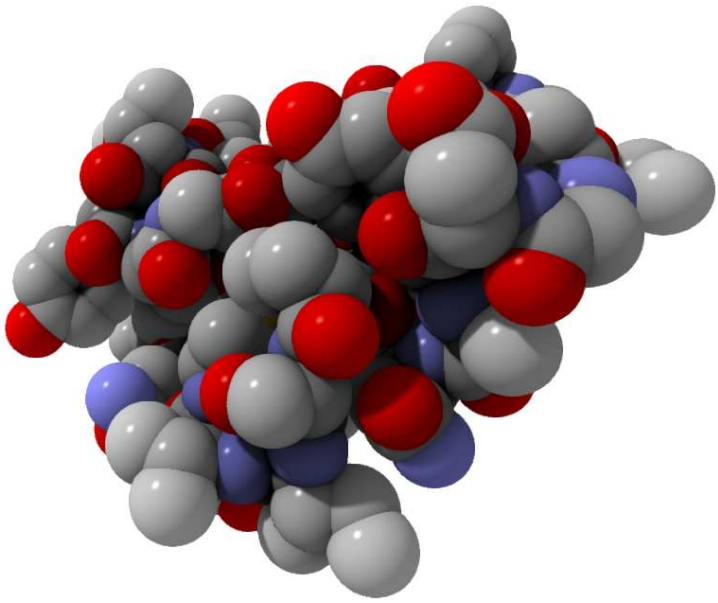


Figure 3.7: This is the full structure of a molecule that has not been clipped by the clipping plane.

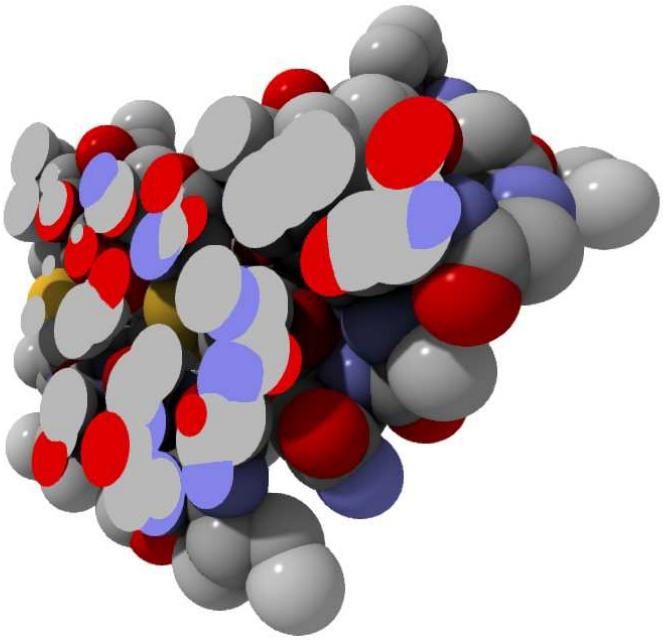


Figure 3.8: Here the protein has been culled by a clipping plane. It is possible to see pockets and channels that were previously hidden when viewing the full structure.

plane. The result of two clipping planes on an atom is shown in Figure 3.9.

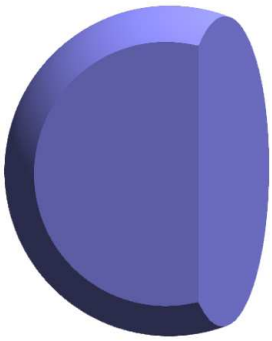


Figure 3.9: Here an atom has been culled by two clipping planes, from the front, and from the right.

The performance impact of adding a dynamic clipping plane to the scene is small. For each atom, we perform a simple sphere-plane test on the CPU to determine if the atom is fully behind the plane. These atoms do not need to be rendered, therefore the number of draw calls can be reduced and this means we keep the more complex fragment shader computations to a minimum. If the atom partially intersects the clipping plane, a vertex attribute is set and passed to the fragment shader to indicate further processing. For fragments that intersect the clipping plane, we perform two point-plane tests, for both $P_{min}$ and $P_{max}$, to determine which pixels we need to discard or alter. Atoms ahead of the clipping plane are rendered normally without these tests, again reducing the amount of fragment operations. However, the real bottleneck in performance tends to be the number of OpenGL draw calls. In practise, this means that adding more clipping planes to the scene will likely reduce the draw call count, and therefore boost performance.

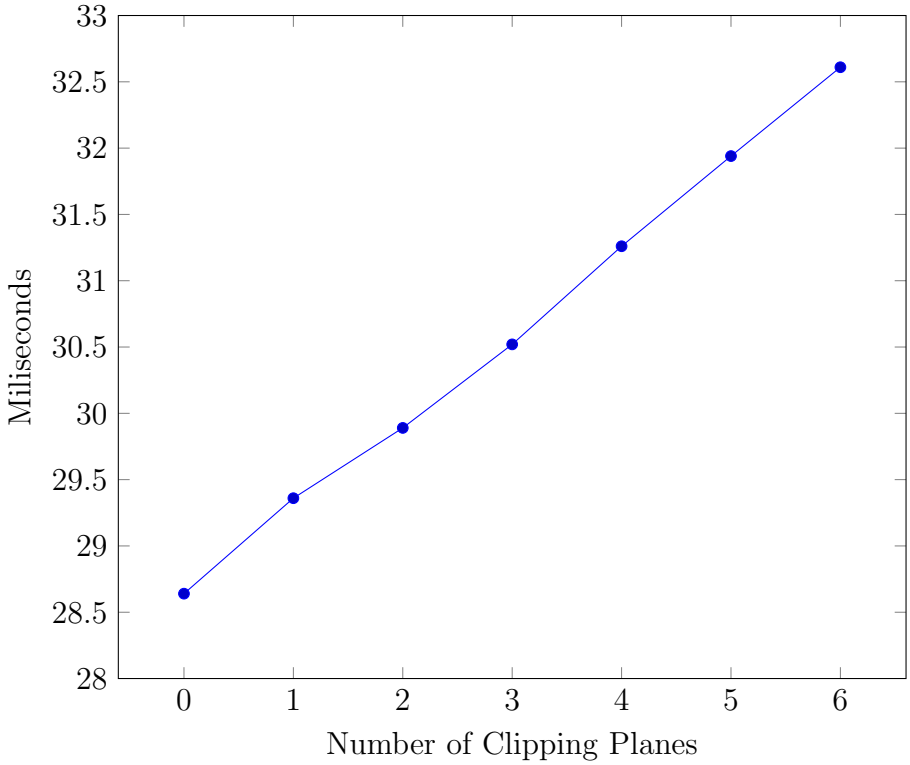



Figure 3.10: This graph shows the performance impact on the GPU of adding clipping planes to the scene. The clipping planes were positioned around the 6 faces of the navigation cube (introduced in the next section) on a molecule containing 58870 atoms. 327 atoms were left unclipped in the navigation cube after all clipping planes were added. This performance test was done without culling atoms on the CPU to highlight the GPU performance impact of the algorithm described in Section 3.4.

## 3.5 Clipping Molecules to a Navigation Cube

HaptiMOL uses a navigation cube to visualise the boundaries of the safe working area of the haptic device being used. Moving the device out from the edges of the cube will translate the molecule so that the desired area of the protein can be safely studied. The navigation cube is illustrated in Figure 3.11.

In the previous section, we looked at how clipping planes can be used to remove areas of the molecule to ensure the user is always able to see the probe sphere. Through experimentation, we noted how the inclusion of a simple sphere-plane test per atom

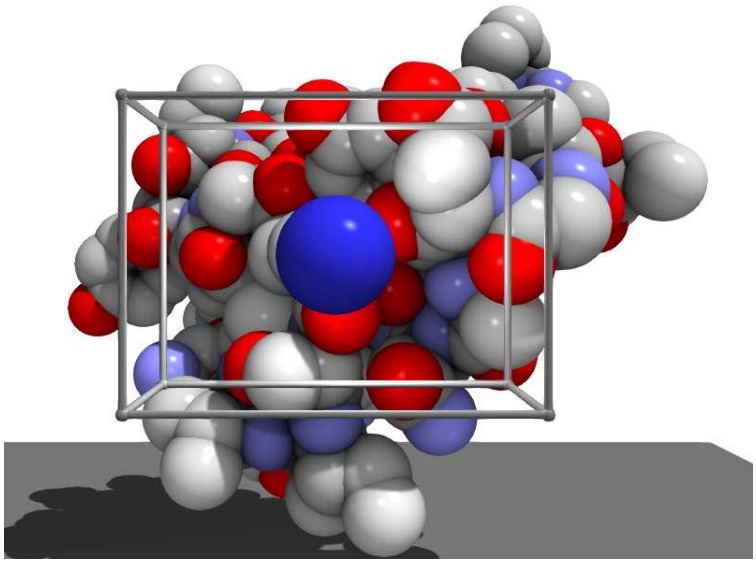


Figure 3.11: The Navigation Cube is used to explore a protein. The dimensions of the cube are based on the usable workspace of the device.

can help to improve performance by reducing draw calls. The performance benefits are especially noticeable when rendering large molecules with tens of thousands of atoms, where vertex processing tends to become the bottleneck of the graphics pipeline. Figure 3.4 shows how, even using the efficient billboarding techniques to render the atoms, we still reach a point when the time to render a frame drops out of what is considered the minimum for real time performance.

For systems unable to render the entirety of a molecule in real time, we have developed a method to only render atoms inside a scalable navigation cube. This provides the user an easy way to reduce the number of atoms rendered to suit their needs, without having to position clipping planes manually, and without having to perform sphere-plane tests per atom. Once the user has navigated to the area of the molecule that they are interested in exploring, this method can be employed to remove atoms outside the area of interest, and therefore boost performance to real time levels for interaction with the molecule.

To render only atoms inside the selected area, we use a uniform grid to allow us to quickly determine the specified area of the molecule the user has navigated to. We talk more about the uniform grid in Chapter 5, including its generation and choice on grid cell density. Figure 3.12 gives a 2D illustration of how the grid cells are chosen. Each of the navigation cube corners are transformed into the grid's local space. For each corner, we find their grid index, indicated by the green cells. The minimum and maximum index of the bounding cells are then updated accordingly, shown as the blue cells. Once these cells have been determined, we then traverse through each cell, testing spheres contained in each cell with the navigation cube. Figure 3.13 shows this technique implemented in our software.

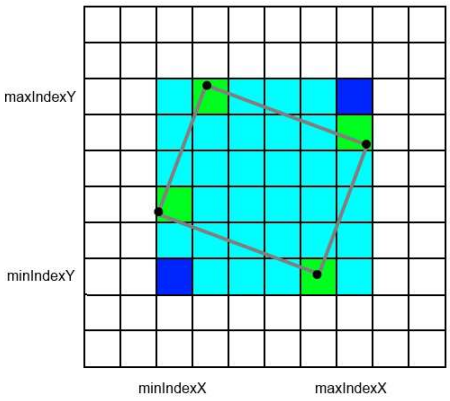



Figure 3.12: This figure illustrates the calculation of the bounding cells of the navigation cube, finding both the minimum and maximum grid index.

It is important to determine the performance overhead of this technique. Table 3.1 shows the clipping method applied to a number of molecules. We first measured the rendering time of a small molecule of 327 atoms, without the clipping method employed. We then used the clipping method on a number of much larger molecules, moving the navigation cube to select a 327 atom section of each. We noted only a very small increase in rendering time, showing this method can be applied effectively on all molecule sizes in our target range.

We also show how we now have an effective method to render sections of very large molecules in real time. 3IYN, an molecule containing almost 100,000 atoms, takes 60 milliseconds render time per frame using the billboard technique. The user can now enable this technique to achieve real time rates that they could get when rendering smaller size molecules.

Table 3.1: Here we compare a the rendering time of a small molecule, 1CRN, with a number of larger molecules clipped to the same amount of atoms using this method.

| Name | Amount | Frame Time (ms) | Overhead (ms) |
|---|---|---|---|
| 1CRN | 327 | 0.32 | 0.0 |
| 1BKS | 5231 | 0.36 | 0.04 |
| 3ETS | 9152 | 0.36 | 0.04 |
| 1JB0 | 24198 | 0.36 | 0.04 |
| 1AON | 58870 | 0.37 | 0.05 |
| 3IYN | 99595 | 0.38 | 0.06 |

## 3.6 Conclusion

In this chapter, we cover a variety of techniques to render atoms being represented by spheres. The use of billboards as a platform to perform ray casting provides an enhanced visual benefit over polygonal meshes by ensuring the spheres are accurate to the nearest pixel, whilst sustaining a far greater frame rate than even the most poorly tessellated triangle meshes. We also show how billboards can be produced cheaply using point sprites, to further reduce processing time in the creation of camera facing geometry.

Whilst this billboarding technique provides many advantages over traditional ray casting methods, it can only be used for the primary ray. Further rays to create shadows, reflections or refraction effects will still need to rely on the traditional methods described in the following chapters.

The clipping plane is a vital tool for exploring large data sets, ensuring the user

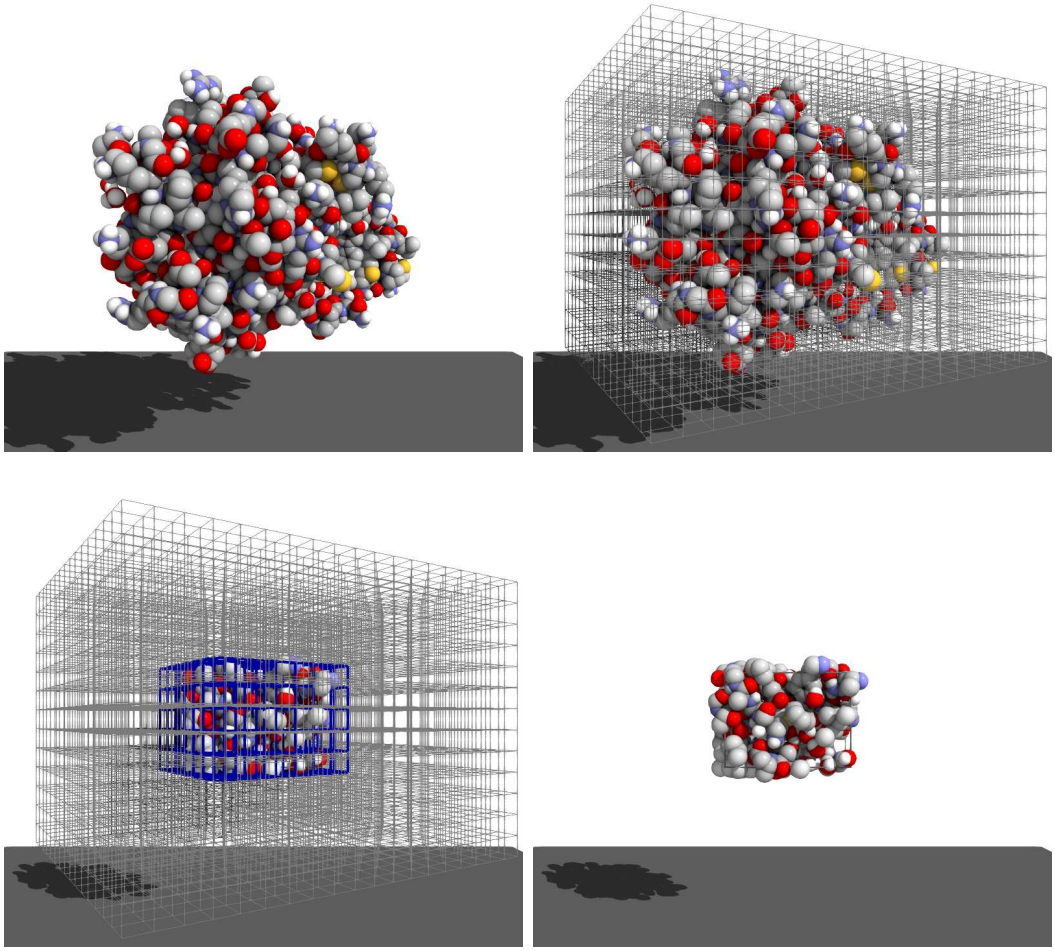


Figure 3.13: These four images show how clipping to the navigation cube is achieved. The top left image shows the full molecule, 1ADG, and the top right image includes the uniform grid of the molecule. The bottom left image highlights the bounding cells of the navigation cube, whilst the bottom right image shows the final result of clipping to this cube.

can always see the probe sphere, whilst also finding pockets and channels in the structure that may be hidden from view. The use of ray casting has allowed for spheres to appear partially culled, giving the user a precise visual cue to the location of the clipping plane in relation to the atoms. Additional clipping planes can easily be implemented with minimal performance impact.

Finally, we have shown a simple technique to render small portions of molecules using a uniform grid to quickly determine areas to cull. This is especially useful for interacting with larger molecules, that are hard to render in real time, even with the billboarding method. This technique will also prove useful for limiting complex lighting calculations to a manageable area, discussed further in Chapter 5.

The next chapters will look at lighting techniques, including realistic ambient occlusion effects to help improve the perception of depth in the scene, and ray traced soft shadows.

# Chapter 4

# Ambient Occlusion for Molecules in Space-Filling Representation

## 4.1 Introduction

In graphics, it is often difficult to appreciate the 3D form of a scene from a 2D image. In our HaptiMOL iSAS software especially, it becomes increasingly necessary when visualising molecules with many thousands of atoms to ensure the user has a good visual understanding of the structure. This will benefit researchers looking at how the geometry of the molecule relates to its function, and also aid users of our software to spot pockets and channels the probe can access. To aid in the depth perception of a molecule, advanced lighting and stereoscopic techniques are sometimes employed in molecular visualisation software. These techniques, now available due to the advancements by modern graphics algorithms and hardware, are used to help achieve the simulation of depth that aid in helping biologists understand the 3D structure of proteins.

The ambient occlusion shading model is perhaps the most popular lighting technique to help the perception of the 3D scene. Experiments have shown that depth perception under uniform diffuse lighting conditions are superior to a direct lighting

model [LB00]. In this chapter, we will discuss a number of ambient occlusion implementations with the goal of improving the users perception of depth in our software.

## 4.2 Ray Traced Ambient Occlusion

The most simple, yet most expensive technique to calculate ambient occlusion is to cast rays from a point, and to test for an intersection with other geometry in the scene. After many hundreds of samples generated randomly over the point's hemisphere, an accurate approximation of the ambient occlusion is produced. Figure 4.1 shows molecule 1CRN rendered using this method, using the ray tracer developed in the next chapter. With 100 rays cast per pixel, the noise pattern from the randomly cast rays is reduced to an acceptable level, however the image takes over 3 seconds to generate. We use cosine-weighted sampling, where samples are weighted by the cosine of the angle between the ray and the normal. This generates samples weighted more heavily towards the normal, and reduces noise for the same amount of computation. Of course, for a real time application this is still not a viable solution. We will investigate the possibility of reducing the number of samples, as well as other methods to try to reduce the rendering time to real time rates.

### 4.2.1 Single Sample Ray Traced Ambient Occlusion

It is common for ambient occlusion methods such as SSAO to use a much lower number of samples per pixel, with an additional blurring pass taking into account depth discontinuities to remove the high frequency noise produced as a consequence. This solution allows a reduction in the number of depth samples per pixel to around 16 while maintaining an acceptable quality.

However, 16 samples per pixel will still be expensive if ray traced. Figure 4.2 shows how a single ray traced sample can produce an ambient occlusion approximation,

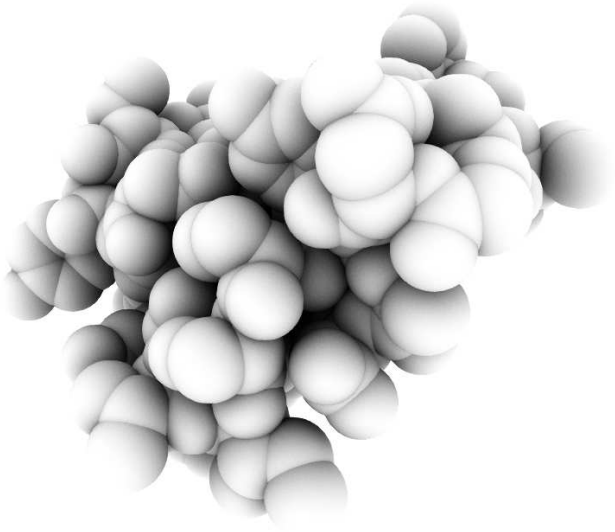
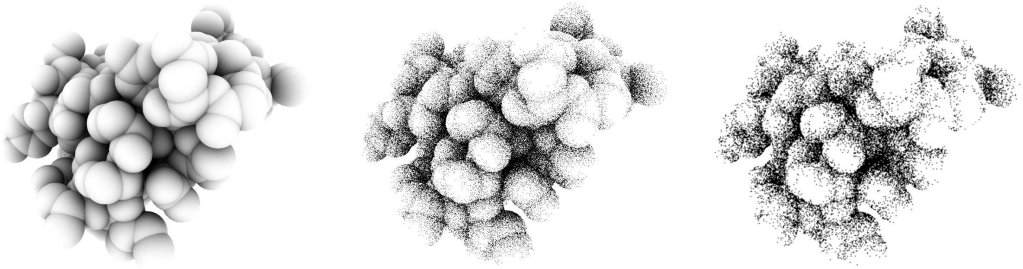


Figure 4.1: Ray traced ambient occlusion using 100 random samples per pixel. The image took around 3 seconds to generate. This image will be used as a reference throughout this chapter.

albeit very noisy. Figure 4.2b is calculated at full screen resolution, whilst Figure 4.2c is calculated at half resolution. The performance of each of these can be seen in Table 4.1.



(a) 100 samples at full resolution.

(b) 1 sample at full resolution.

(c) 1 sample at half resolution.

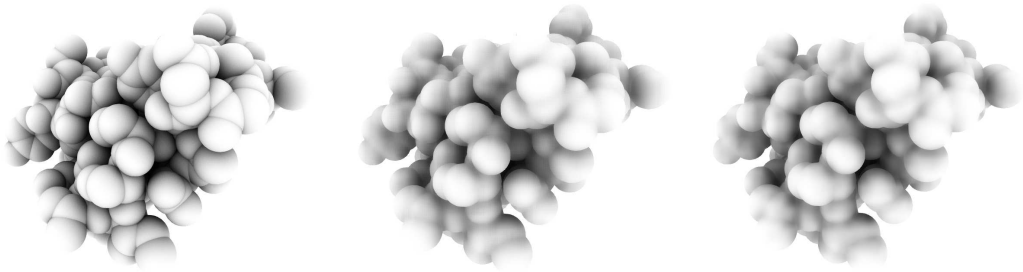
Figure 4.2: Ray traced ambient occlusion of the 1CRN molecule, with varying samples and resolutions.

A smart blur filter can then be used to remove the noise, whilst retaining sharp edges. The blur pass gives the impression of using multiple samples for a fraction of the cost. A common smart blurring technique used today is a cross bilateral

Table 4.1: Ray traced ambient occlusion at full and half resolution.

| Name | Amount | Full Resolution (ms) | Half Resolution (ms) | Change (%) |
|---|---|---|---|---|
| 1CRN | 327 | 21.67 | 7.04 | -67.51 |
| 1BKS | 5231 | 47.28 | 15.51 | -67.19 |
| 3ETS | 9152 | 56.89 | 17.30 | -69.59 |
| 1JB0 | 24198 | 58.33 | 16.68 | -71.40 |
| 1AON | 58870 | 64.07 | 20.50 | -68.00 |
| 3IYN | 99595 | 71.90 | 23.64 | -67.12 |

filter. This is a depth dependant Gaussian blur that diffuses the ambient occlusion whilst avoiding blurring across edges. This is achieved by using lower weights for samples with a large difference in depth from the current pixel. Figure 4.3 shows the effectiveness of the blurring stage to remove noise from the image.



(a) 100 samples at full resolution.

(b) 1 sample at full resolution with bilateral blur.

(c) 1 sample at half resolution with bilateral blur.

Figure 4.3: Ray traced ambient occlusion of the 1CRN molecule, after applying a bilateral filter to remove the noise caused by the reduced number of samples.

The results of using a single ray traced ambient occlusion sample per pixel are promising. The more global ambient occlusion effect that ray tracing provides is desirable, especially to help gain an understanding of a large molecular structure. With a smart blurring stage, it is possible to produce smooth results from an original noisy source image.

There are notable issues with this method. Flickering can be seen caused by the randomly generated ray direction for each pixel. This is exaggerated when calculating the ambient occlusion at a lower resolution to the screen size. A working solution

is to ensure the blur radius is large, and to apply a secondary blur pass to further smooth the flickering effects. The consequence of the heavy blurring is local ambient occlusion detail is lost, and in Section 4.5 we look at ways to tackle this.

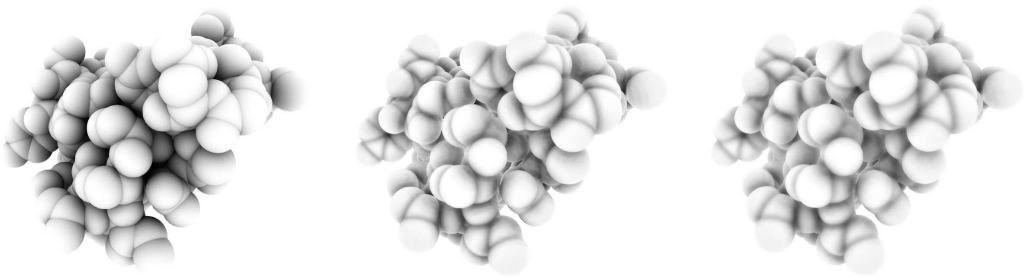
## 4.3 Screen Space Ambient Occlusion

Screen space ambient occlusion (SSAO) is a rendering method to approximate ambient occlusion in real time. It has become the standard approach for computing ambient occlusion in real time applications such as video games. First introduced by Mittring [Mit07], the technique uses the depth buffer as an approximation of the geometry in the scene. The occlusion factor is then calculated in screen space, meaning the process can be done entirely on the GPU, The process is dynamic, requires no pre-computation, and is independent of scene complexity. These features are desirable for our HaptiMOL iSAS software, where molecules can contain many thousands of individual atoms.

Whilst calculated completely on the GPU, screen space ambient occlusion is still a relatively expensive process. It is common to see SSAO calculated at half screen resolution, and with perhaps only 16 samples per pixel. A depth aware blur such as a cross bilateral filter will then be used to smooth the image before using it in lighting calculations. Figure 4.4 shows SSAO calculated at different resolutions after a blur pass has been performed.

Compared with other ambient occlusion solutions, SSAO has a set of features that make it desirable for molecular graphics software. The independence from scene complexity makes it ideal for molecular structures of all sizes. The fully dynamic nature makes it excellent for scenes with moving objects such as the probe sphere in our software. It is also easily integrated into a modern graphics pipeline.

The limitation of SSAO, however, is the distances at which the method produces



(a) 100 samples at full resolution.

(b) 16 samples at full resolution with bilateral blur.

(c) 16 samples at half resolution with bilateral blur.

Figure 4.4: Screen space ambient occlusion of the 1CRN molecule, with bilateral blur.

results that can be interpreted as occlusion. An extreme case can be seen in the differences between Figures 4.10 and 4.12, where atoms are too far away for the SSAO technique to capture, giving a false impression of the structure of the molecule. Increasing the sampling radius to try and capture a larger area of objects decreases performance significantly, and leads to fine ambient occlusion detail being lost. SSAO is a useful technique for solving occlusion at relatively small distances, meaning another solution needs to be found if occlusion at higher distances is desired.

## 4.4 Analytically Calculating the Ambient Occlusion of Spheres

The ambient occlusion equation is most easily approximated by sending out test rays from a point on a surface of an object, and finding if the rays intersect with other objects in the scene. The ambient occlusion value will then be dependant on how far away the objects are, and how many intersections are found. To produce sufficiently accurate results using this method, many hundreds of rays need to be tested per pixel, which rules out this method for real time applications.

Work by Iñigo Quílez [Qui06] shows how it is possible to analytically calculate the occlusion of a sphere to a point on an arbitrary surface. This is especially useful for our iSAS software, which uses spheres to represent atoms of the molecule. We will

therefore investigate the possibility of using this method for a space-filling molecular representation, with the goal to create high quality and dynamic occlusion effects.

The equation below describes the blocking factor for a sphere with radius $r$ that is at a distance $d$ from this point. The blocking factor, $bl(r, d)$, is the area that the sphere projects into the hemisphere divided by the area of the hemisphere.

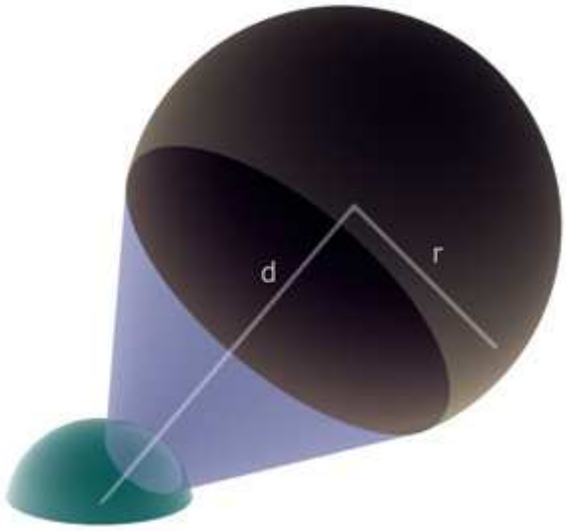



Figure 4.5: The ambient occlusion of a sphere to a point on a surface can be calculated knowing the radius of the sphere and the distance from the sphere's center to the surface.

$$bl(r, d) = 1 - \sqrt{1 - (r/d)^2} \tag{4.4.1}$$

The expression is simply a function of $r/d$. This is logical, as making a sphere bigger means it will have to move further to have the same projected shadow or occlusion factor. A sphere $x$ times bigger will project the same shadow if it is $x$ times further away. The effect can be seen in Figure 4.6.

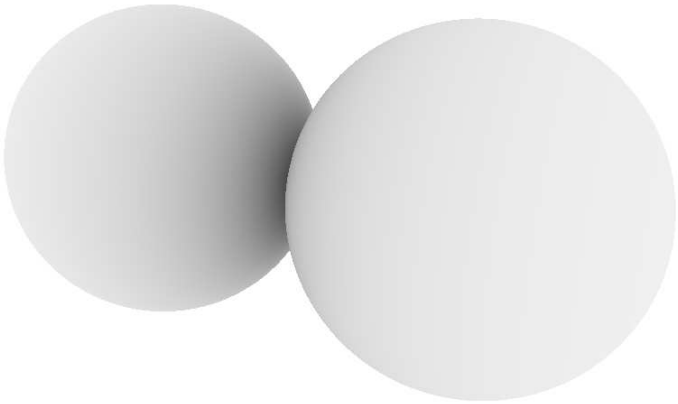


Figure 4.6: This is the result of calculating the ambient occlusion of two white ray-traced spheres, using the equation in 4.4.1.

### 4.4.1 Implementation Decisions

To integrate this technique into our software, for each atom, we store the atoms nearest to it. As the atoms in the molecule do not need to move, we can calculate an atoms nearest neighbours in an initialisation stage as the molecule is loaded. The sphere IDs of these neighbouring atoms are then stored in a texture to be accessed in the lighting stage of our graphics pipeline. These IDs point to a second pre-calculated texture that stores the position and radius of each atom. A final texture is used to store a flag for each atom indicating if the atom has been culled by a clipping plane. This will ensure the lighting is dynamic as atoms are removed from view. The occlusion of the movable probe sphere is calculated on all nearby pixels, in a similar fashion to a point light detailed in the next chapter.

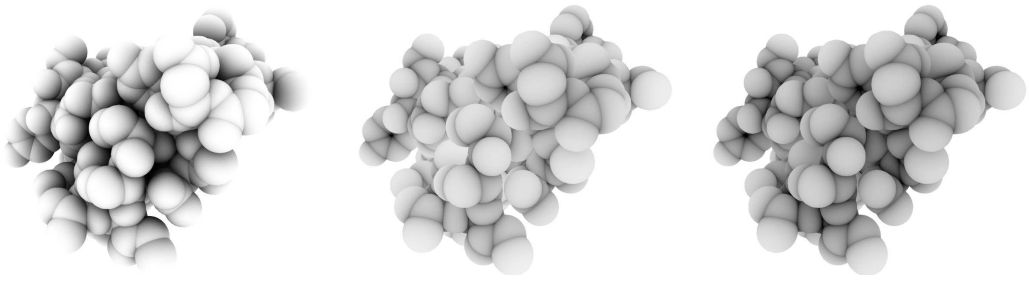
Table 4.2: AO data layout.

| Red | Green | Blue | Alpha | Texture Format |
|---|---|---|---|---|
| Culled Flag | | | | GL_R8 |
| Position | | | Radius | GL_RGBA32F |
| Atom Occluders | | | | GL_R32F |

The amount of occluding atoms to store for each atom is based both on performance and storage limitations. OpenGL 2.1 API implementations require as a minimum a 2048x2048 texture size to be supported. Based on this minimum size, we can store up to 32 occluding atoms per atom whilst still supporting molecules well over 100,000 atoms in size. Table 4.3 shows the performance of this ambient occlusion method with a varying number of occluding atoms and over a range of molecules.

Table 4.3: Analytical ambient occlusion with varying number of occluding atoms.

| Name | Amount | 8 atoms (ms) | 16 atoms (ms) | 32 atoms (ms) |
|---|---|---|---|---|
| 1CRN | 327 | 2.41 | 4.19 | 6.66 |
| 1BKS | 5231 | 2.44 | 4.31 | 7.02 |
| 3ETS | 9152 | 2.72 | 4.83 | 7.61 |
| 1JB0 | 24198 | 2.74 | 4.90 | 7.97 |
| 1AON | 58870 | 2.48 | 4.41 | 7.68 |
| 3IYN | 99595 | 2.39 | 4.36 | 6.58 |



(a) 100 samples at full resolution. (b) Analytical AO using 8 nearest spheres. (c) Analytical AO using 32 nearest spheres.

Figure 4.7: Analytical ambient occlusion of the 1CRN molecule.

Analytically calculating the ambient occlusion of spheres has many appeals. The formula to calculate the occlusion factor is very simple, and performs well at a per pixel basis. The technique produces occlusion with no artifacts, and therefore fits well with our pixel perfect spheres from the previous chapter. This also means no additional blur stage is required. However, this method does require additional preprocessing and memory overhead, although a lower occluder count could be used to

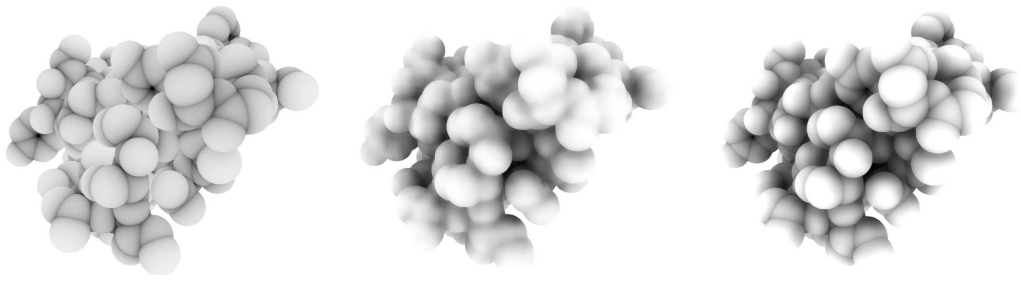
tackle this if needed. Due to the occluder limit, this technique is only useful for occlusion in localised areas as it is unable to capture a more global occlusion value.

## 4.5 Hybrid Ambient Occlusion

There has been interesting work investigating hybrid ambient occlusion techniques, such as Shanmugam and Arikan [SA07]. They look at separating the ambient occlusion problem into detailed high-frequency occlusion, and distant low-frequency occlusion domains, demonstrating good results.

In the previous sections we have presented a number of techniques to calculate the ambient occlusion of molecular structures. They each have their own unique characteristics, and each have advantages and disadvantages over each other. Taking two of these methods and using them together in a hybrid solution could provide an ambient occlusion solution that no technique on their own could provide.

The single sample ray traced ambient occlusion seen in Section 4.2.1 calculates occlusion that is too far for other techniques to produce, but lacks fine detail as a result of heavy blurring. The analytical solution from Section 4.4 is ideal for calculating occlusion at smaller distances, but lacks a more global reach. Combining these two methods with a simple multiplication can produce results much closer to the reference ray traced image that either could on their own. Figure 4.8 illustrates this method.



(a) Analytical AO with 8 occluding spheres.

(b) 1 ray traced sample at half resolution, with blur.

(c) Multiplication of (a) and (b).

Figure 4.8: Combination of two ambient occlusion techniques.

## 4.6 Comparison of Ambient Occlusion Techniques

The previous sections have detailed a number of possible solutions to calculate ambient occlusion for space-filling molecules. Choosing the best solution for our HaptiMOL iSAS software will need to be based on a number of areas. Performance is the key factor for the software. The application needs to run in real time to allow the user to navigate the structure with a probe sphere. Aiding the depth perception is an important secondary factor to take into account. A good visual understanding on the molecule will be of great benefit to the user, helping them move the probe with more confidence and efficiency. Visual quality also needs to be factored in, with a smooth natural appearance important as to not distract the user.

Figure 4.9 shows the relative performances of each ambient occlusion method that has been implemented, and Table 4.4 shows their advantages and disadvantages. It is clear that both the SSAO and analytical methods have a significant performance advantage over the ray traced and hybrid solutions. The performance of the ray traced solutions is also affected by the number of atoms in the molecule. The SSAO and analytical methods are not affected by the molecule size. This predictable constant performance is very appealing for software that visualises molecules that vary by a large amount in size.

Table 4.4: A comparison of ambient occlusion techniques we have presented.

| Technique | Advantages | Disadvantages |
|---|---|---|
| Ray Traced 100 samples | High accuracy, no noise | Not real time |
| Ray Traced 1 sample | Real time, captures global occlusion | Noisy, needs strong blurring |
| SSAO | Real time, constant rendering time | View dependant, local occlusion only |
| Analytical | Real time, high quality | Local occlusion only |
| Hybrid | Captures both near and far occlusion | Not realistic for real time use yet |

The choice between SSAO and the analytical technique comes down to visual quality, as performance wise they are similar, and both provide localised occlusion. SSAO requires calculation at half resolution, with a blurring pass to achieve an acceptable level of performance. The analytical solution produces a pixel perfect effect, with no artifacts or flickering due to randomised sampling methods. For our software the analytical method is the best solution, with constant fast performance, and high quality occlusion.

For users with access to higher end PCs, the hybrid solution may be a viable option, and could be considered for future updates in our software.

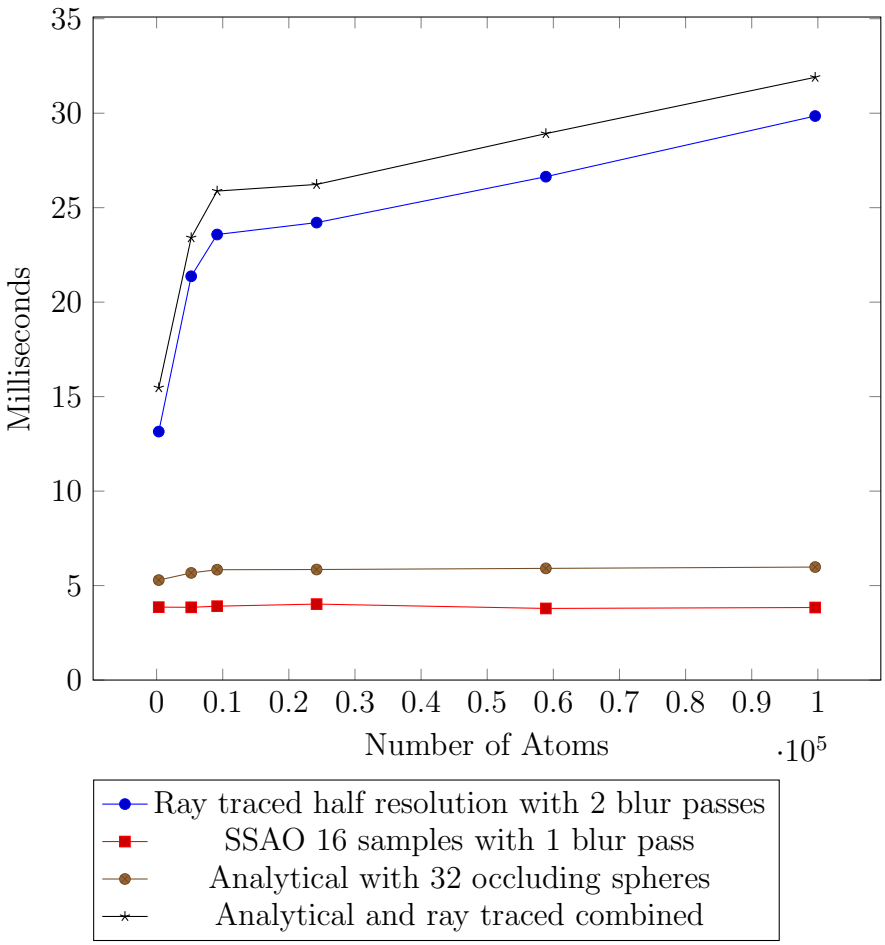



Figure 4.9: This graph shows the performance of calculating ambient occlusion using a number of techniques.

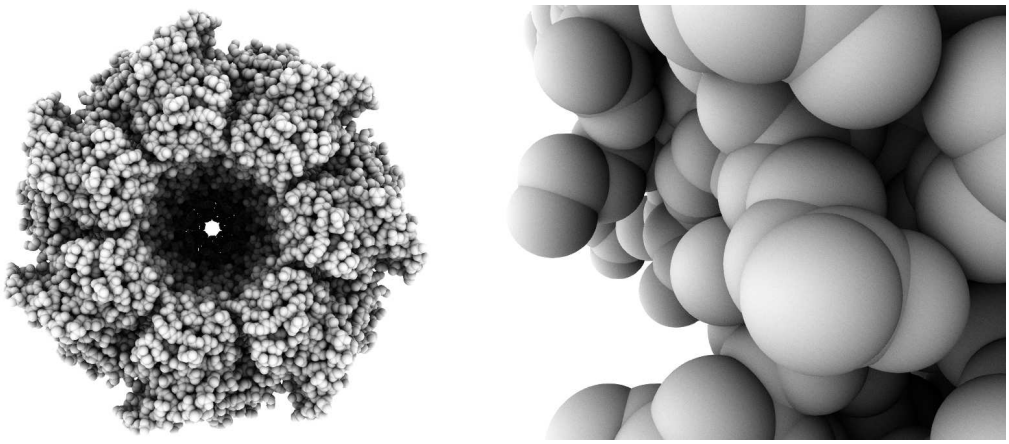


Figure 4.10: 100 samples at full resolution.

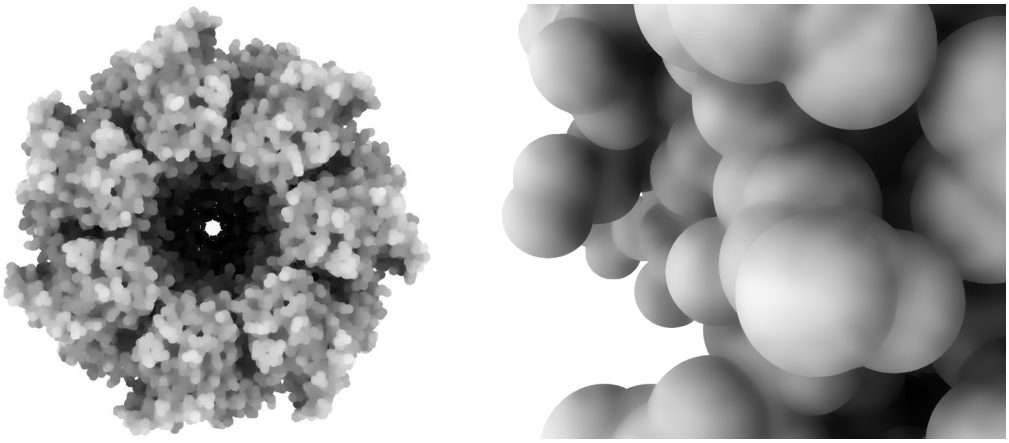


Figure 4.11: 1 sample at half resolution with blur.

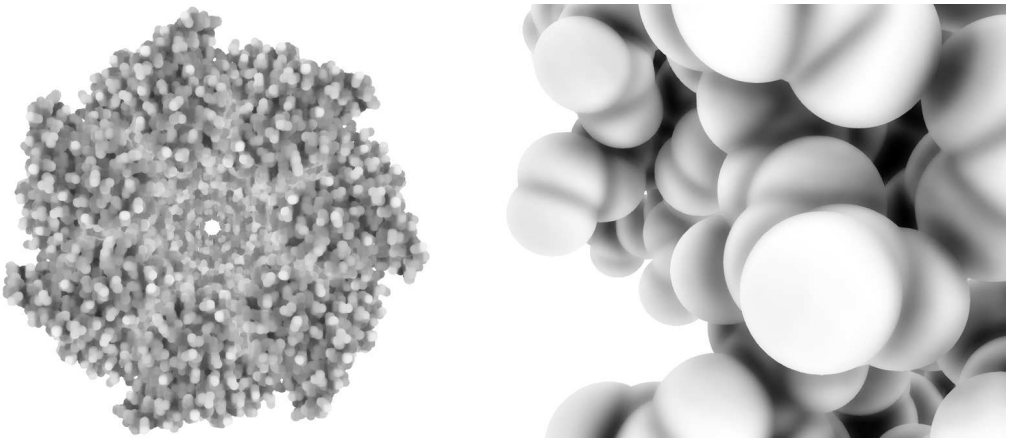


Figure 4.12: SSAO at half resolution with blur.

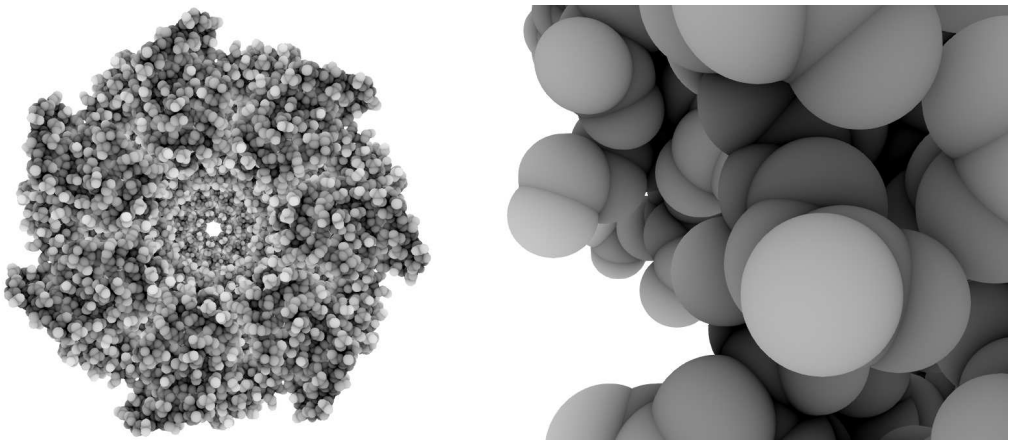


Figure 4.13: Analytical technique with 32 occluding atoms.

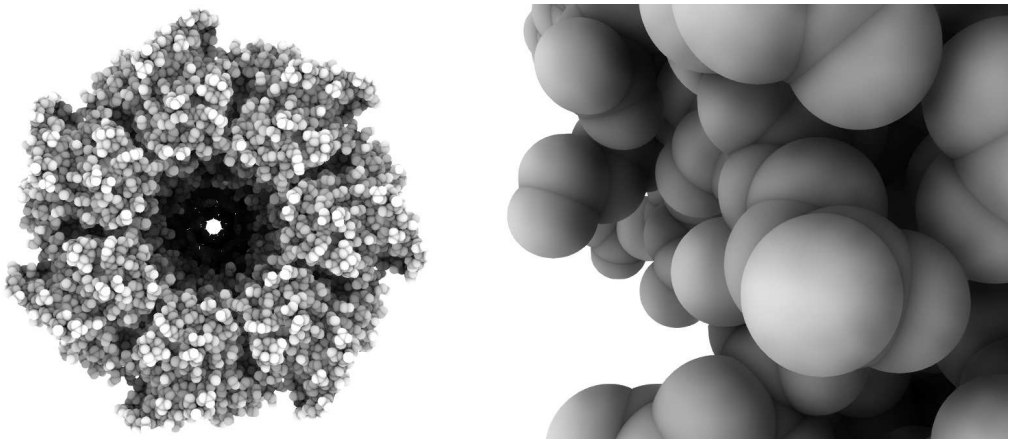


Figure 4.14: Hybrid solution with ray tracing and analytical methods.

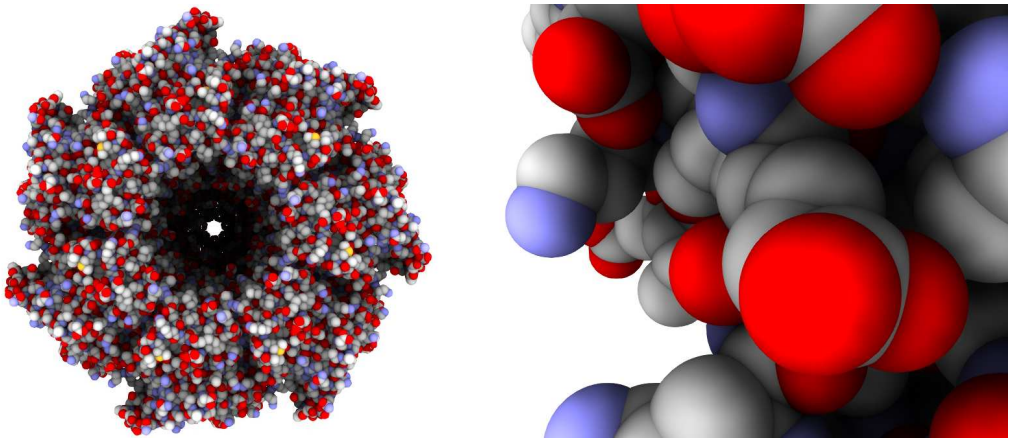


Figure 4.15: Hybrid solution with diffuse colours.

## 4.7 Deferred Rendering

In the previous chapter ray tracing was used to produce pixel perfect spheres onto billboards giving a large visual improvement to the scene. However, when this technique is combined with relatively complex lighting calculations, performance can be an issue. This is largely due to the need to set the fragment depth manually in the fragment shader. If the driver knows that the output fragment depth is the same as the one generated, it can perform a simple depth test before executing the fragment shader, known as an early depth test. This means that it can discard fragments before computing potentially complex fragment shaders. Indeed, most modern hardware has complicated early z culling hardware that can discard multiple fragments with a single test. However, as soon as we set the fragment depth manually, further along in the pipeline, these optimisations can not be performed.

The lighting calculations performed per pixel on the molecules, whilst not inherently expensive on their own, can become very expensive to compute when combined with the large overdraw of pixels as a consequence of no early depth testing. A technique called deferred rendering can be employed in this case, to replace the traditional forward rendering approach.

Forward rendering is a straightforward approach where we apply a set of transformations on the vertices of every object in the vertex shader followed by a lighting calculation per pixel in the fragment shader. Since each pixel of every object gets only a single fragment shader invocation we have to provide the fragment shader with information on all light sources and take all of them into account when calculating the light effect per pixel. This is a simple approach but it has its downsides. If the scene is highly complex with many objects and a large depth complexity (same screen pixel covered by several objects) we get a lot of wasted GPU cycles.

Deferred rendering is a popular technique which targets the specific problem above.

The key point behind deferred rendering is the decoupling of the geometry calculations and the lighting calculations. Instead of taking each object from the vertex buffer into its final resting place in the framebuffer, we separate the processing into two major passes. In the first pass we run the usual vertex shader, but instead of sending the processed attributes (such as positions, normals, colours) into the fragment shader for lighting calculations, we forward them into what is known as the Geometry Buffer or G-Buffer. This is a logical grouping of several 2D textures that are written to all at once using a capability of OpenGL called Multiple Render Targets (MRT). Since we are writing the attributes in the fragment shader, the values that end up in the G-Buffer are the result of the interpolation performed by the rasterizer on the vertex attributes. This stage is called the Geometry Pass. Every object is processed in this pass, and because of the depth test, when the geometry pass is complete the textures in the G-Buffer are populated by the interpolated attributes of the closest pixels to the camera. This means that all the irrelevant pixels that have failed the depth test have been dropped and what is left in the G-Buffer are only the pixels for which lighting must be calculated. This will solve the issue of overdraw that is caused when rendering our geometry.

Using deferred rendering over the traditional forward rendering provides other benefits. Lighting as a 2D post process means we can apply potentially thousands of lights efficiently, combining different types such as point and spot, and mixing shadowed and unshadowed lights, without many different combinations in of lighting shaders as previously required. It also allows us to spatially optimise lighting and complex shading, meaning we only apply light to the pixels which are actually in the light's area of effect. The information stored in the G-Buffer can also be used for other post processing effects, that may also be useful to improve the viewers' perception of the 3D structure of the molecule.

Anti-aliasing is an important consideration for graphics applications. Aliasing occurs as monitors display an image in the form of uniformly coloured pixels, with geometry in the scene not aligned to these pixels appearing jagged. FXAA has become one of the most popular techniques to combat aliasing. Created by Timothy Lottes [Lot11], FXAA runs as a single full screen pixel shader post process, that can be easily be integrated into the deferred rendering pipeline. On our development system, this shader takes 1.5 milliseconds to run on a 1280x720 image at default settings. This is a very acceptable cost for the excellent edge smoothness it produces. The only downside to this technique in the form of a slight loss of detail in textures will not effect our software.

There are notable downsides to deferred rendering that need to be discussed. Deferred rendering has an inherent overhead in terms of memory use, due to the storage requirements of the G-Buffer. It also has potentially reduced material flexibility. Unlike in forward rendering, it is not possible to change the shader which computes lighting to make changes to one material, as we have to do everything en-masse in a 2D post process. Another problem is that any transparent objects in the scene can not be handled in the deferred renderer. These objects need to be dealt with in a second, forward rendered pass meaning we still need a working forward renderer.

In our case however for the HaptiMOL software, the disadvantages were not enough of a concern to outweigh the vast benefits deferred rendering provides. There is little to no use of the alpha channel in the application. The material properties are very simple for the molecules, meaning we are not disadvantaged from a loss of material flexibility. Finally, the concern about memory use is still valid, but with careful decisions on what data we store in the G-Buffer, as well as texture formats for the render targets, memory use can be kept to a minimum. Table 4.5 breaks down the G-Buffer layout of our pipeline, with the total GPU memory usage of the G-Buffer

at around 25MB at 1280x720 resolution.

Table 4.5: G-Buffer data layout.

| Red | Green | Blue | Alpha | Texture Format |
|---|---|---|---|---|
| Diffuse Colour | | | | GL_RGBA8 |
| Position | | | Atom ID | GL_RGBA32F |
| Normal | | | Material ID | GL_RGBA16F |

## 4.8 Conclusion

In this chapter we hope to have shown how ambient occlusion can greatly improve visual depth perception of molecules, aiding the user to navigate the structure with more efficiency. We have explored a variety of methods all able to run in a real time environment.

Analytically calculating the ambient occlusion of spheres provides the best overall solution for our software. We show how this technique can perform similar to other modern ambient occlusion techniques such as Screen Space Ambient Occlusion, and compares favourably when comparing visual quality, benefiting from the lack of blurring required to produce a smooth outcome. Ray traced results are shown to be promising, and will become a viable solution as users of our software adopt powerful modern graphics cards.

Ambient occlusion works well as a basic approximation of indirect illumination in this application. However the effect is lost when direct lights are added to the scene without shadows. Producing shadows for our molecular visualisation software will be the focus of the next chapter.

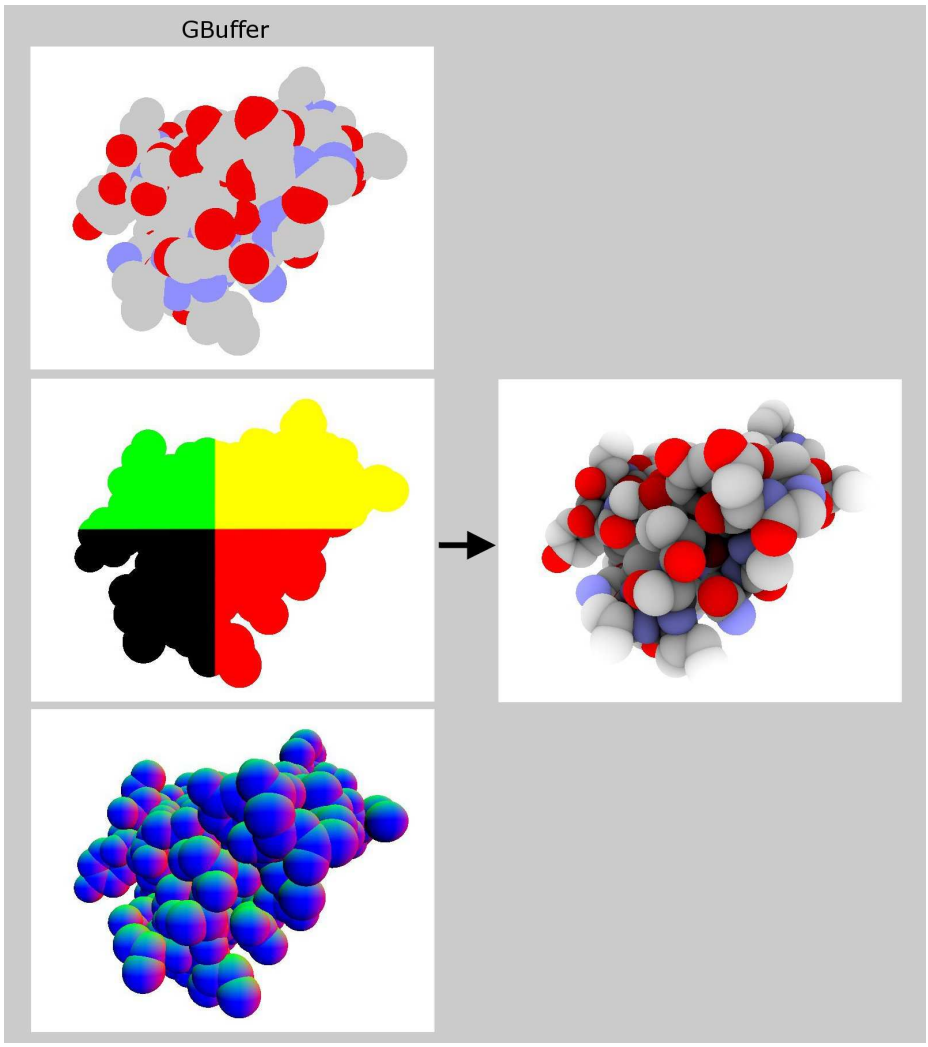



Figure 4.16: The scene is rendered into a G-Buffer, storing material properties (left top), positions (left middle) and normals (left bottom) of the scene. The information in these textures is then used to produce the final image, seen on the right.

# Chapter 5

# Real Time Shadows for Molecules in Space-Filling Representation

## 5.1 Introduction

Shadows are vital for the human perception of a 3D environment. Research such as work from Mamassian et al. [MKK98] and Wanger et al. [WFG92] explore, through psychophysical experiments, the role shadows play in helping understand an object's relative position and size in a graphical scene. Work by Kersten et al. [KMK94] shows how the movement of the shadows play an important role in producing apparent motion in depth. Shadows can also reveal geometry hidden from view, or geometry that is outside the field of view. The ability to achieve accurate, dynamic and real time shadows in our HaptiMOL iSAS software would greatly enhance the users' ability to navigate a molecular structure.

However, reproducing realistic shadows in a real time graphics application has been and remains a difficult task. In Section 3.3 we described a technique to produce ray cast spheres using camera aligned billboards. In this chapter we will look to continue exploring the ray tracing theme to produce an elegant solution for dynamic shadows, whilst still maintaining an interactive frame rate.

## 5.2 GPU Ray Tracing with GLSL

Ray tracing offers a much more elegant and accurate solution for our problem domain compared to other popular shadow techniques such as shadow mapping. A scene needs to be rendered once per directional light source, and six times for each omnidirectional light source using shadow mapping. Using ray tracing, shadows are a simple extension to the scene, especially for simple geometry such as spheres. A shadow ray is used to test if a surface is visible to a light. When a primary ray hits a surface, a ray is traced between this intersection point and the light. If any opaque object is found in between the surface and the light, the surface is in shadow and so the light does not contribute to its shade.

Ray tracing is generally an expensive process due to the high number of intersection tests per frame. A brute force approach to ray tracing is to test each ray with all objects in the scene. This becomes infeasible to compute in real time as the number of objects in the scene increases. The primary approach to accelerate ray tracing is to reduce the total number of intersection tests. This usually involves traversing some form of spatial data structure to find objects nearby the ray.

Currently the HaptiMOL iSAS software uses OpenGL and OpenGL Shading Language (GLSL) for rendering a molecular structure with Phong shading and no shadows. GLSL is a high-level shading language created to give developers more direct control of the graphics pipeline at the vertex and fragment level. It is desirable to carry on using these technologies for ease of integration, as well as compatibility with older hardware. Shader Model 3.0 introduced the ability to perform GPGPU in the shaders, and work by Purcell et al. [PBMH02] and Christen [Chr05] have shown how it is possible to perform ray tracing techniques on the GPU with shaders, using a uniform grid to reduce intersection tests.

The uniform grid is one of the easier acceleration structures to implement on a

GPU. First introduced by Fujimoto and Iwata [FI85], the grid evenly divides the scene into same size cells, with each cell containing references to all objects that intersect it. The structure works best when the grid cells closely match all the objects in size. This can be achieved in our case as the dimensions of the atoms in a molecule vary only by a relatively small amount. Atoms sizes defined by Bondi [Bon64] range from Hydrogen with an atomic radius of 1.2 angstroms, to potassium with a radius of 2.75 angstroms. The structure also maps well to GPU memory by using textures as storage, and the algorithm to traverse the structure is simple enough to implement in a pixel shader, meaning it is still a popular choice of acceleration structure on the GPU. For these reasons it makes a good choice of acceleration structure for our software.

### 5.2.1 Grid Creation

The creation of the uniform grid will need to be performed each time a user loads a molecule into the program. The creation needs to be fast enough to ensure there is minimal delay between loading and the molecule being rendered. The method of the uniform grid's creation is shown below. For each Sphere $S_i$:

1. Find the grid cell containing the center of Sphere, $S_i$

2. Perform collision tests between Sphere, $S_i$, and all the cells directly next to the initial cell from step 1, with cells represented by Axis Aligned Bounding Boxes (AABB)

3. Add a reference to Sphere, $S_i$, in the grid texture for the initial cell, as well as cells where Sphere-AABB test returns true

One of the most important choices for the uniform grid is the cell size. It is generally the case that the cell size will be large enough to contain the largest object.

This ensures the number of cells an object overlaps is no more than eight and limits the cell intersection tests, important for the initialisation speed. However, performance during the grid traversal stage is of most importance, as this needs to happen in real time. The decision of cell size is further discussed in Section 5.2.4.

### 5.2.2 Grid Traversal

A simple way to traverse a uniform grid was first introduced by Amanatides and Woo [AW87], using a 3D-DDA algorithm. This algorithm only requires a small amount of operations to guide the ray between neighbouring cells with the cost of traversal being linear to the number of cells visited. Another feature of the algorithm is a valid hit will occlude hits in the following cells as voxels are visited in order of the ray passing through the grid, seen in Figure 5.1.

### 5.2.3 Grid Storage

To access the uniform grid structure as well as the geometry data in the pixel shader, the data is mapped to texture memory. Previous work on GPU ray tracing from Purcell et al. [PBMH02], Christen [Chr05] and Nguyen [Ngu07], use a method where each cell in the grid index structure contains a pointer to a grid cell list. The grid cell list contains atom IDs that point to the actual geometry data. This method can potentially handle molecules with hundreds of thousands of atoms in size, which will ensure support for a large proportion of molecules currently recorded.

As these data textures are generated on CPU, the current implementation is limited to rendering static scenes only. This is acceptable in our software as the atoms in a molecule do not move. However, clipping planes can be placed in the scene stopping atoms behind the plane from being rendered. Solutions to ensure dynamic lighting effects for the molecule are explored in Section 5.4.

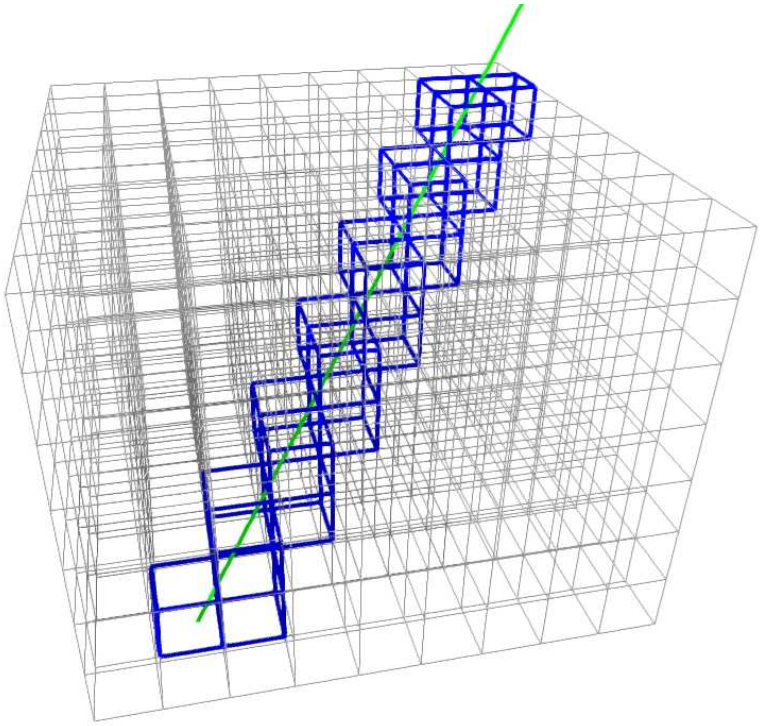


Figure 5.1: A representation of the 3D-DDA algorithm. The green line represents the ray, and the highlighted blue cells are cells the ray intersects.

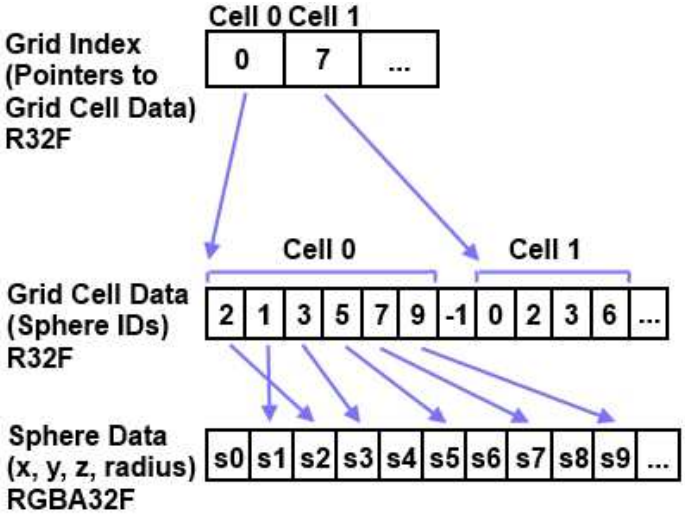



Figure 5.2: This is the structure of the data that is mapped to texture memory. Based on a texture size of 2048x2048 pixels, this structure ensures we can perform ray tracing calculations on molecules hundreds of thousands of atoms in size.

**Algorithm 1** Shadow calculation algorithm in light pass

```
1: for all pixels containing geometry do
2:     calculate N · L
3:     if N · L > 0 then
4:         if Sphere ID > 0 then
5:             calculate grid index of ray origin
6:             if grid index is outside grid then
7:                 project ray origin onto grid
8:             end if
9:             while inside grid do
10:                for all spheres in grid cell do
11:                    if ray intersects sphere then
12:                        pixel is in shadow
13:                    end if
14:                end for
15:            end while
16:        end if
17:    end if
18: end for
```

### 5.2.4 Performance

Choosing a suitable cell size for the uniform grid is vital to achieve real time performance for shadow calculations. A smaller cell size should help reduce the number of intersection tests needed per cell. A larger cell size will help reduce memory required to store the grid data, as the number of empty grid cells, as well as the likelihood of spheres being in multiple cells is reduced. If memory usage is a concern, the grid cell size can be increased at the cost of a likely decrease in performance.

Figure 5.3 compares the performance of a variety of grid cell sizes for a test molecule, from a single cell brute force approach, to cells that are the smallest possible size, meaning they are able to fully contain the largest atom in the molecule.

The most common molecules explored using HaptiMOL generally contain thousands or tens of thousands of atoms. GroEL, containing nearly sixty thousand atoms, is considered a large molecule. By default we use the minimum cell size available, as

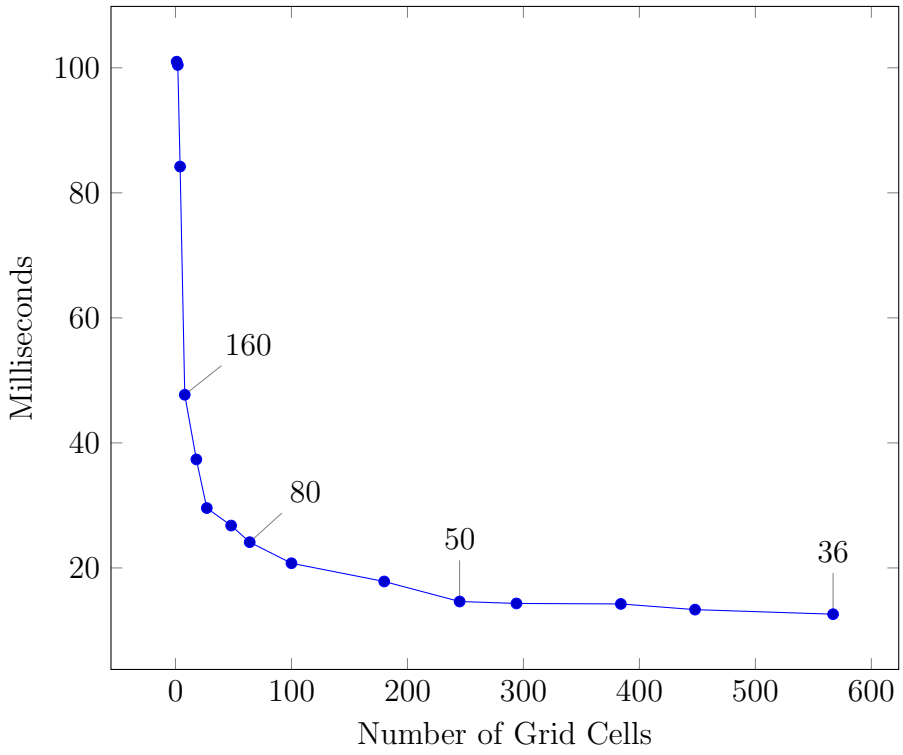



Figure 5.3: This graph shows the performance of calculating shadows for a single directional light source on a molecule 1CRN containing 327 atoms. As the grid cell density is increased, the calculation time drops dramatically. Cell sizes are indicated above selected points. The units are based on the van der Walls radii, with an Oxygen atom radius, for example, having a value of 15.2.

for the vast majority molecules we have the memory available to achieve maximum performance. This size is a good starting point to reduce memory requirements, as atoms can be in no more than eight cells in the worst case. However, as we can see, there is room for more memory savings if required.

In Figure 5.3, we can see for low levels of subdivision, the shadow calculations are very expensive. However, with only a small increase in subdivision the performance can increase dramatically. These results closely resemble those of Amanatides and Woo [AW87], although, with the minimum cell size set to contain the largest atom in the molecule, we do not see the reduced performance of having too many grid cells they encounter.

The most notable feature of the performance results is that we see from roughly halfway between the minimum and maximum amount of cells, the performance increase becomes negligible. For 1CRN, the time to calculate shadows with 567 grid cells takes 12.60 milliseconds, whilst for 294 grid cells, this time increases only slightly to 14.32 milliseconds. Almost halving the number of cells with less than a 2 millisecond drop in performance is significant, especially if this trend is repeated for molecules with tens of thousands of grid cells. For very large molecules or molecules with unique structures that make the uniform grid inefficient, with lots of empty cells, finding this optimal zone between memory usage and performance is appealing, allowing smaller textures to store the grid to be used.

To help investigate why the performance gains drop off so dramatically, Figure 5.4 shows the relationship between the number of grid cells and the average number of atoms per cell for 1CRN.

We can see a nice correlation between Figures 5.3 and 5.4. This shows a direct link in performance and the number of atoms per cell. Traversing the grid is relatively cheap, whilst the intersection tests needed per cell is where we incur the most cost. What we also learn is that increasing the number of cells beyond a certain point does little to reduce the average number of atoms per cell, which explains the reduced performance gains for higher grid densities. For this particular molecule, a cell size of 50 seems to produce the best performance to memory ratio. It would be desirable to know if these findings can be used for other molecules of varying sizes.

Looking at the performance results of this new selection of molecules, we see a similar pattern as before. A cell size in the region of 50 seems to produce the best memory to performance ratio. It is important to note, however, for larger molecules especially, the performance difference between the minimum cell size and our optimal suggestion is still significant when considering real time performance. With 1AON,

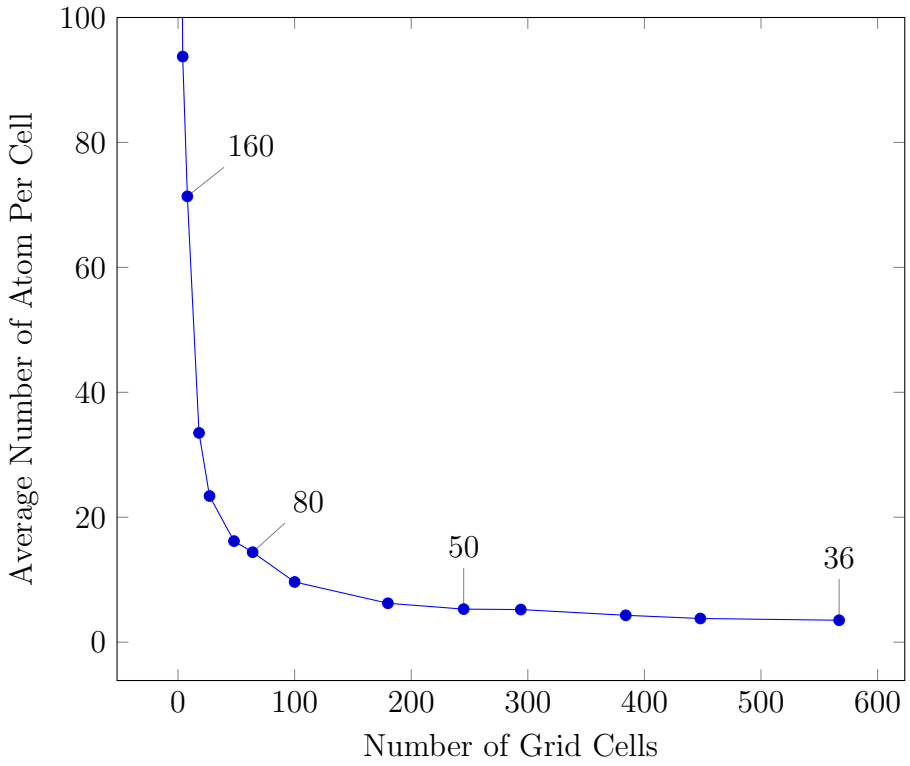
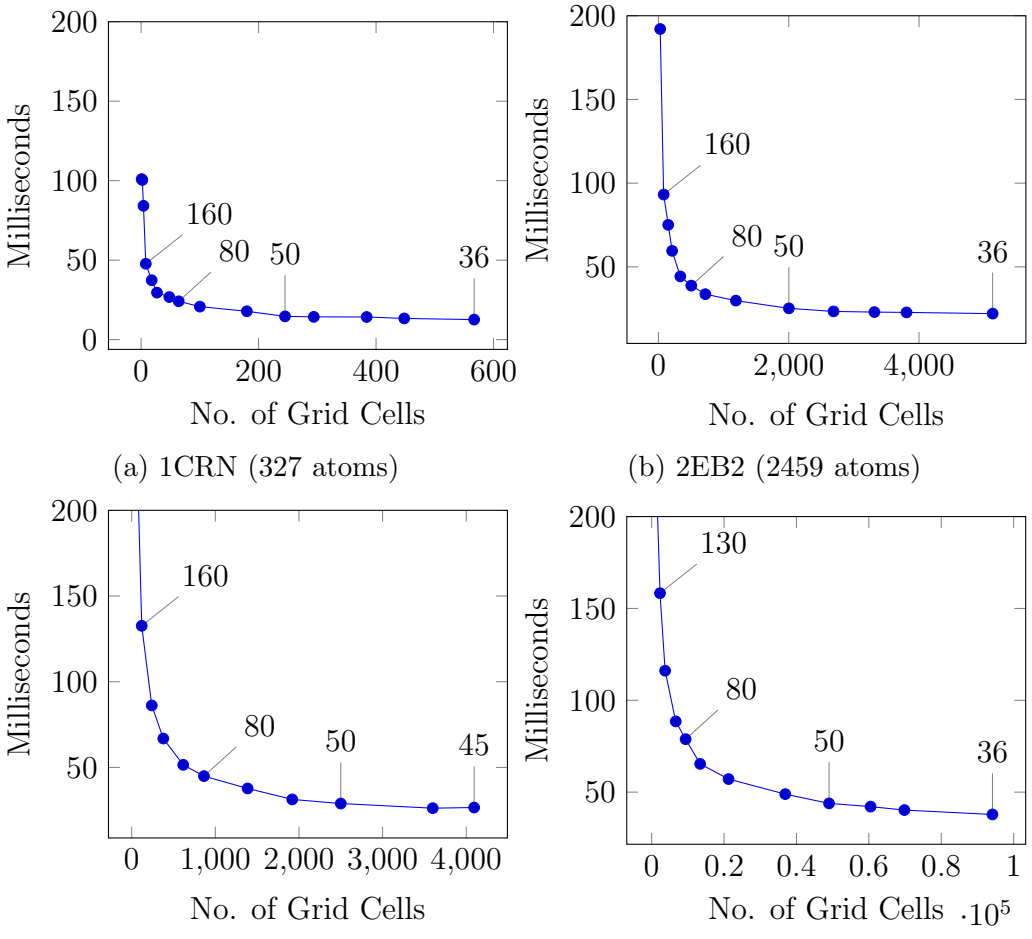



Figure 5.4: Here the relationship between the average number of atoms per cell and the number of grid cells is shown. There is clearly a link between the number of atoms per cell, and shadow calculation time. Cell sizes are shown above selected points.

for example, this difference in around 10 milliseconds, which is a large amount of time considering real time targets 33 milliseconds rendering time.

Our ultimate goal is to achieve real time shadows to improve the users' perception of the scene. We have shown we can achieve real time rates for small and medium sizes molecules. However, for large molecules, ray traced shadows drop below these rates on our development system. In Chapter 3 we looked at rendering only a small portion of a molecule, clipped to a navigation cube to reduce draw calls and therefore rendering time. In Section 5.4, we will look at building on this technique to achieve real time and dynamic shadows on larger molecules such as GroEL.




(a) 1CRN (327 atoms) (b) 2EB2 (2459 atoms)

(c) 1BKS (5231 atoms) (d) 1AON (58870 atoms)

Figure 5.5: Here we compare four molecules to look for similar patterns in performance in relation to the number of grid cells. Being able to choose an optimal cell size (indicated above selected points) when initialising the grid can lead to significant memory savings with a relatively small increase in shadow calculation time.

## 5.3 Aiding User Interaction with a Shadow Casting Probe Light

The HaptiMOL iSAS software includes a point light positioned at the center of the probe sphere. The attenuation of the light is set to illuminate atoms near to the probe, giving the user a better understanding of its position relative to other atoms in the molecule. Currently however, the light does not cast shadows, meaning areas of the molecule that are occluded from the light are still being illuminated. A shadow

casting probe light would give the user a better understanding of where the probe sphere is positioned relative to the molecular structure.

Creating shadows for a point light using the Shadow Mapping technique requires the scene to be rendered six times to cover areas the light can potentially illuminate. Ray tracing greatly simplifies the calculation of point light shadows, needing a single shadow ray per pixel. Algorithm 1 describes ray tracing for directional lights, where the ray needs to traverse the grid until it exits, as the light source is effectively outside the grid. For point lights, the algorithm can be modified slightly to reduce the number of voxels traversed. This is achieved by adding an extra condition to stop the traversal once the ray reaches the light's grid cell, as we know everything after the lights grid cell can not occlude the pixel we are testing. Figure 5.6 illustrates this method.

Another small change that can be made is limiting the area of shadow calculation to pixels in the illuminated area of the point light. A pre-set cut-off is defined manually in the light pass shader to ensure expensive shadow calculations are performed only when necessary. An example of the reduced area of pixels to perform processing on is shown in Figure 5.7.

## 5.4 Dynamic Shadows for Navigation Cube Exploration

One of our aims for our software is to achieve fully dynamic shadows when atoms are culled, either by a clipping plane or by the navigation cube introduced in Chapter 3. This will aid in producing a better interactive experience, where lighting effects change in response to users' actions.

In the previous chapter, we described a simple way to achieve a dynamic ambient occlusion effect by using an additional texture to store a culled flag for each atom. We can use the same technique here to ensure atoms that have been culled from view

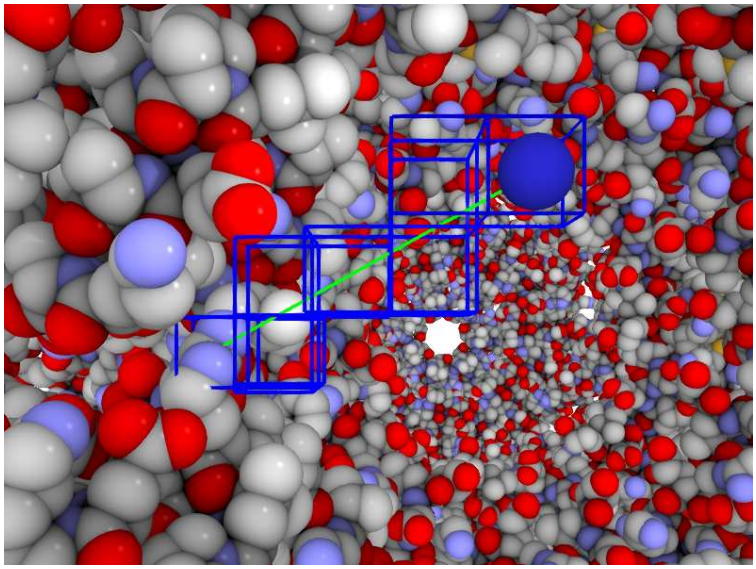


Figure 5.6: This figure shows how the grid is traversed for a point light. Once the light's grid cell has been reached, we can stop the traversal.

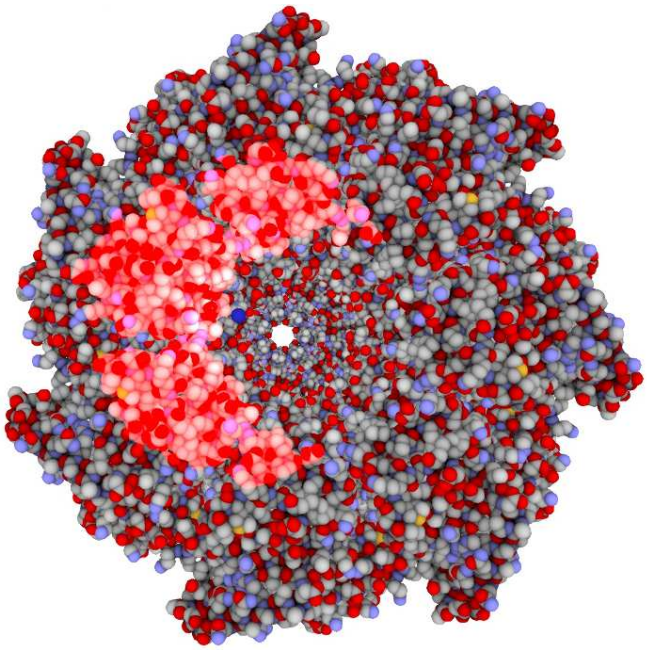


Figure 5.7: The red area highlighted on the molecule shows the pixels that the point light illuminates. Limiting shadow calculations to only these pixels ensures no unnecessary processing, therefore improving performance.

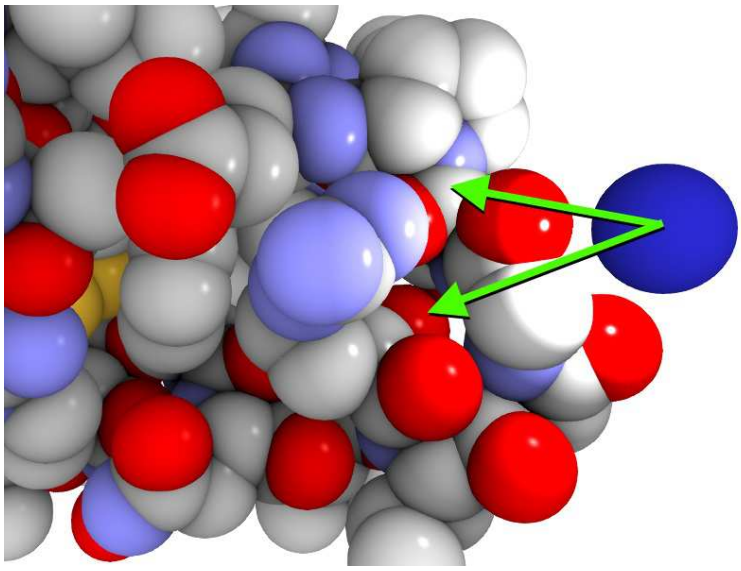


Figure 5.8: This image shows the probe sphere (dark blue) with a point light at its center. The attenuation of the light is set to illuminate atoms in the molecule that are in close proximity, giving the user a better sense of depth when navigating the molecular structure. However, without shadows, there are areas that receive light even thought they are occluded by other atoms (highlighted by the arrows).

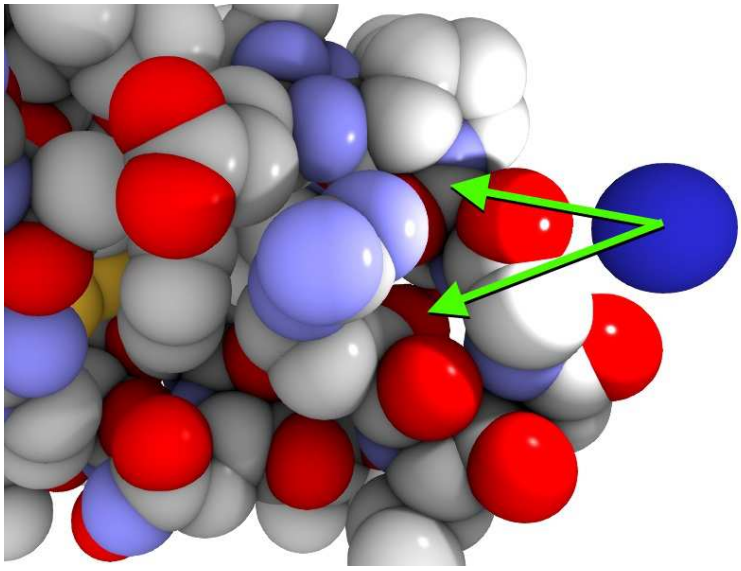


Figure 5.9: Here is the same image, but with shadows enabled. The position of the probe sphere relative to other atoms is made clearer, allowing the user to navigate the structure with more confidence.

can no longer cast shadows on atoms still visible in the scene. We can make a small but important change to Algorithm 1 to reduce unnecessary traversal of the shadow ray through the uniform grid. Figure 3.12 shows how we calculate the bounding cells for the navigation cube. We can send the minimum and maximum indexes of these cells to the light shader, and are then are able to traverse the grid until we are outside of these bounding cells, instead of outside of the entire grid as was previously needed. Figure 5.10 illustrates this change.

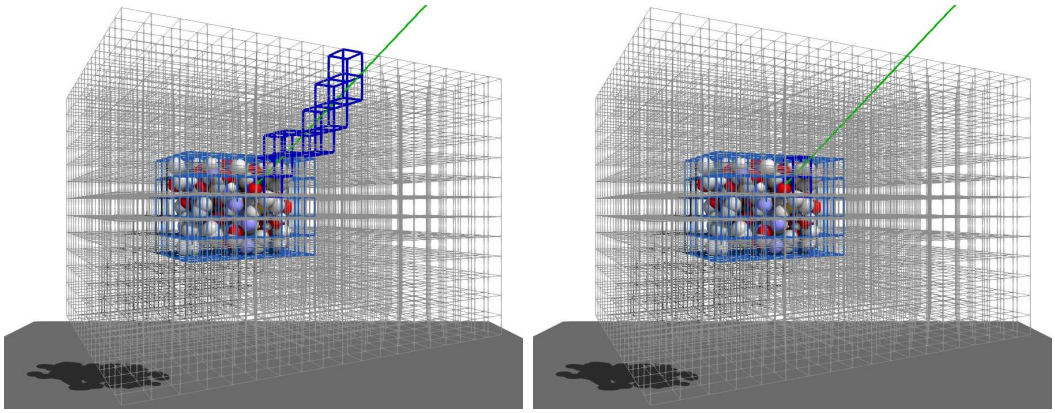


Figure 5.10: The light blue cells indicate the bounding cells of the navigation cube. A shadow ray (the green line), traverses the grid until it is outside of the workspace, instead of traversing the entire grid.

The bounding cells of the navigation cube are an efficient way of culling large portions of a molecule, as traversal of a shadow ray can be stopped when it is outside of these cells. However, we still need to test for atoms that may be in the bounding cells but are not in the navigation cube. Table 5.1 shows the performance overhead of the dynamic shadows, with 6 milliseconds being the average on a variety of molecules. This decrease in performance comes from the need to check if an atom has been culled, meaning an additional texture read and branch in the light pass shader.

What can be seen is that we have an effective way to produce dynamic ray traced shadows on molecules with tens of thousands of atoms by reducing atoms rendered

Table 5.1: Here we compare a the shadow calculation time of a small molecule, 1CRN, with a number of larger molecules clipped to the same amount of atoms using this method.

| Name | Amount | Shadow Calculation Time (ms) | Overhead (ms) |
|---|---|---|---|
| 1CRN | 327 | 12.68 | 0.0 |
| 1BKS | 5231 | 18.77 | 6.09 |
| 3ETS | 9152 | 18.53 | 5.85 |
| 1JB0 | 24198 | 19.21 | 6.53 |
| 1AON | 58870 | 18.84 | 6.16 |
| 3IYN | 99595 | 19.23 | 6.55 |

to those in the navigation cube. Molecules that would be impossible to render in real time on our system, can now be explored at interactive rates. Reducing shadow calculation time also allows for the possibility of adding both a directional light source, as well as a point light source located at the center of the probe sphere, to the scene.

However, by limiting the atoms rendered to those inside the navigation cube, the user can not appreciate the full structure of the molecule. Our solution is to render the backbone structure of the molecule, using only simple lighting effects. The backbone structure illustrates the main chain of the structure which many biologists use when visualising a molecule. The main chain often adopts substructures called secondary structures which are mainly helices and sheets, and their arrangement in proteins makes their structure easy to identify. These can not be seen in space-filling models, as they are occluded by the dense packing of the atoms. Figure 5.11 shows how the backbone structure is rendered alongside the space-filling representation, with the complex lighting calculations only being performed on a much smaller scale. This helps boost performance whilst still allowing ray traced shadows and ambient occlusion effects for areas of the molecule the user is interacting with.

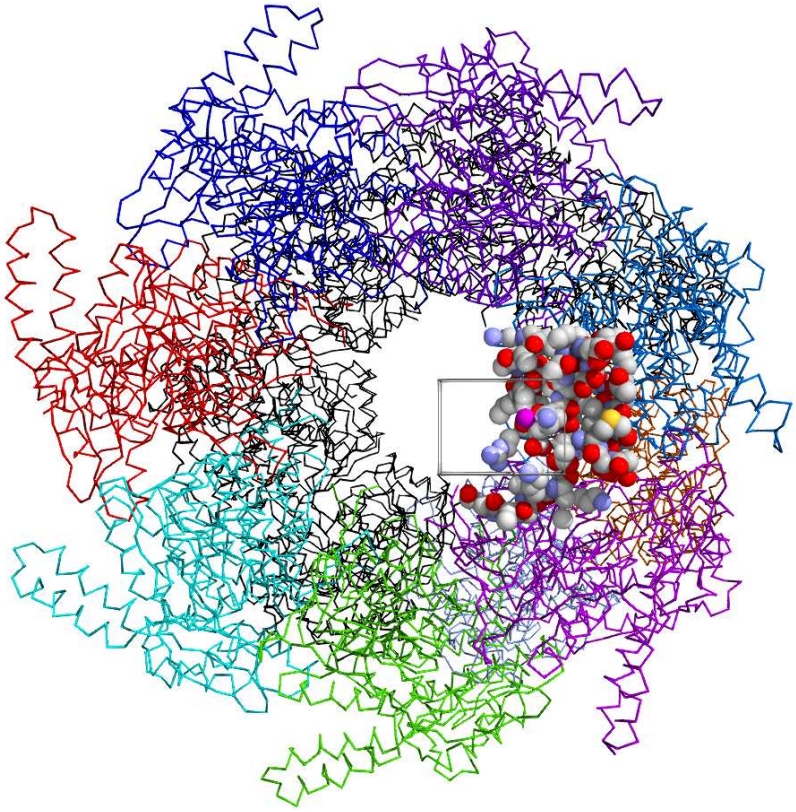


Figure 5.11: The backbone structure of the molecule is rendered to give the user a better understanding of the molecular structure. Space-filling atoms, and therefore complex lighting and shadow calculations, are limited to those atoms inside the navigation cube.

## 5.5 Soft Shadows for Space-Filling Molecules using Ray Tracing

One of the limitations of ray tracing is the hard edges it computes for shadows. This is because the light source is usually defined by a single point, meaning the surface is either fully lit or fully shadowed. In reality, a surface can have a partial view of a light, where only a fraction of the light source is visible from that point. Traditionally we need to cast multiple rays from these area light sources in order to model how they interact with the scene. This technique will produce an accurate soft shadow effect if enough rays are used, however this dramatically increases computation time

relative to hard shadow computation.

Knowing we are only ray tracing with spheres provides the opportunity to calculate plausible soft shadows with little extra computation. Parker et al. [PSS98] introduced an algorithm that creates a shadow of a soft-edged object from a point light source, rather than creating a correct soft-edged shadow from an area light source. We can integrate this technique into our previous ray tracing work with minimal extra code, providing a simple way to create visually plausible soft shadows with little extra computation. Although these soft shadows are approximate, they are robust and have penumbra widths that behave in a believable way. The main benefit of this technique is that only one shadow ray is needed per pixel.

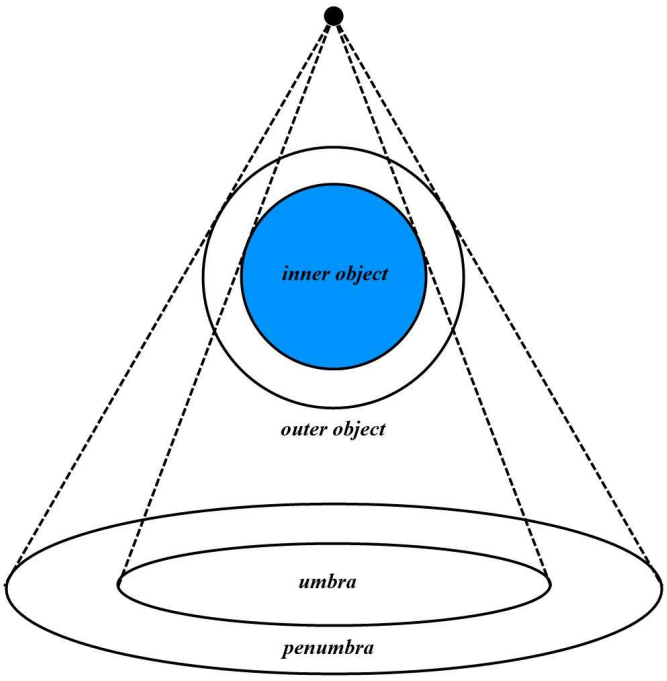



Figure 5.12: The inner sphere casts the umbra of the shadow. An outer sphere creates a larger shadow, with the area between the two spheres creating the penumbra.

As seen in Figure 5.12, the penumbra is the shadow of the semi-opaque (outer) object that is not also shadowed by the opaque (inner) object. The transparency

of the outer object increases from no transparency next to the inner object, to full transparency at the boundary of the outer object

The intensity gradient of the penumbra needs to look natural. This can be achieved by computing a variable $s$ that begins on the surface of the inner object, and increases with distance to the surface of the outer object. Parker el al. suggest a linear gradient between the two boundaries, however a Hermite curve provided a smoother transition when tested.

There is also a need for the algorithm to remain simple and robust for multiple objects. For example, if a point is in the penumbra region for two objects, we can make a composite attenuation factor based on the individual factors $s1$ and $s2$. We can take the minimum value between $s1$ and $s2$, which will yield continuous intensity transitions and visually pleasing results for the shadow of multiple objects.

For a shadow ray $\mathbf{p} = \mathbf{o} + t\hat{\mathbf{v}}$ toward a light with position $\mathbf{l}$, and a sphere with center $\mathbf{c}$ and radius $R$, we need to decide whether we are in the penumbra region, and if so, what is the value of $s$, the fractional distance between umbra and penumbra boundaries. We first compute the distance $t_0$ to the point on the shadow ray closest to $\mathbf{c}$:

$$t_0 = (\mathbf{c} - \mathbf{o}) \cdot \hat{\mathbf{v}} \tag{5.5.1}$$

If $t_0$ is negative, then $s$ is set to 1, as there is no collision with the sphere. We then compute the value of the minimum distance $d$ from the center of the sphere $\mathbf{c}$ to the ray:

$$d = \|t_0\hat{\mathbf{v}} - \mathbf{c} + \mathbf{o}\| \tag{5.5.2}$$

If this distance is between $R$ and $Rb$ then we calculate $s$ as:

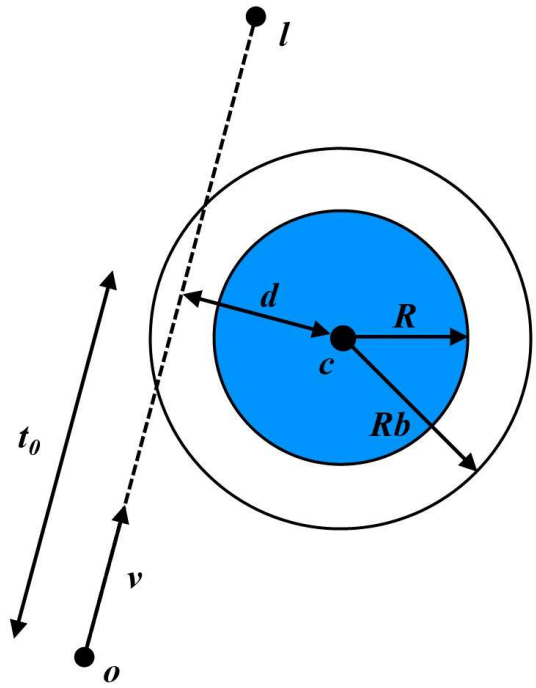



Figure 5.13: To produce the soft shadow effect, we surround our sphere with center $c$ and radius $R$, with a larger sphere with radius $Rb$. We then scale the shadow value based on the distance $d$ from the shadow ray to the center of the sphere.

$$s = (d - R)/b \tag{5.5.3}$$

If $d$ is less than $R$, $s$ is set to 0 as the ray is colliding with the inner sphere. If $d$ is greater than $Rb$, $s$ is set to 1, as the ray is not colliding with either the inner or outer sphere. The value of $s$ is multiplied by the specular and diffuse terms to produce the soft shadow effect in the scene.

Table 5.2 compares the performance of this soft shadow technique with hard shadows. For small molecules with hundreds of atoms, we can see around a 20 percent increase in rendering time. This increases to around 30 percent for molecules with tens of thousands of atoms. This relatively small hit in performance is offset by the realistic and more natural shadows that help the spatial perception of the molecule.

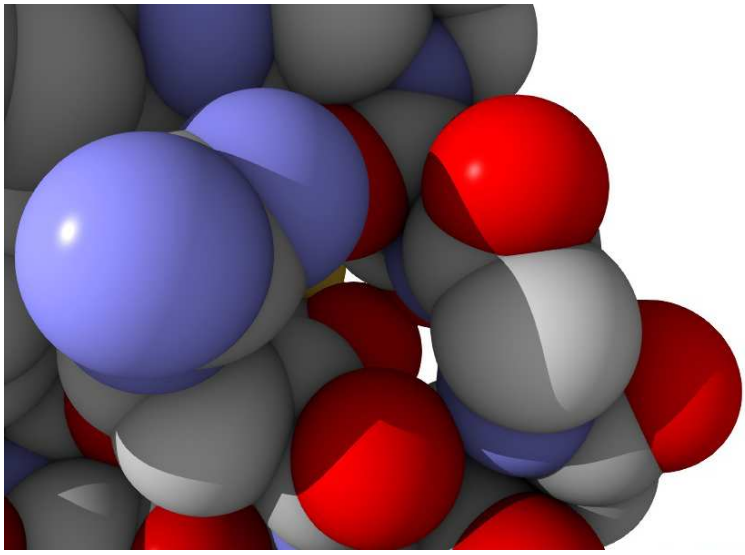


Figure 5.14: Here is an example of traditional hard shadows, calculated by casting a single shadow ray towards a point light source. If the shadow ray collides with any spheres in the molecule, then both the diffuse and specular terms are set to zero, leaving just the ambient term to light the pixel.

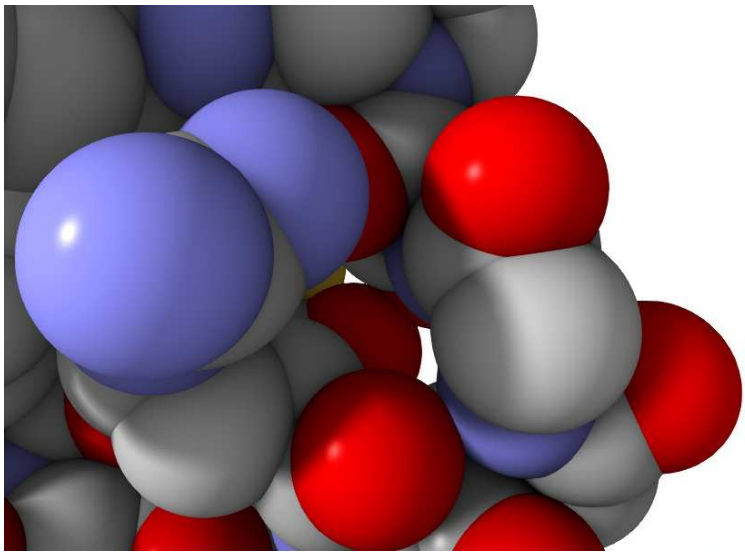


Figure 5.15: Here the hard shadows have been replaced with the soft shadows using the technique described above. The smooth change between shadowed and unshadowed areas can be seen, aiding the spatial perception of the scene.


Figure 5.16: Here we see our new rendering techniques integrated into the HaptiMOL iSAS software. The user interacts with the molecule by controlling a probe sphere using a PHANTOM Omni haptic device.

Table 5.2: A comparison of the performance between hard and soft shadows.

| Name | Amount | Hard Shadows (ms) | Soft Shadows (ms) | Difference (%) |
|---|---|---|---|---|
| 1CRN | 327 | 12.72 | 15.54 | 19.95 |
| 1BKS | 5231 | 26.68 | 33.77 | 23.45 |
| 3ETS | 9152 | 33.53 | 43.69 | 24.52 |
| 1JB0 | 24198 | 38.29 | 49.33 | 25.19 |
| 1AON | 58870 | 38.04 | 50.14 | 27.44 |
| 3IYN | 99595 | 35.93 | 47.14 | 26.98 |

## 5.6 Future Advances in the Rendering of Space-Filling Molecules using Path Tracing

Path tracing, proposed by Kajiya [Kaj86], is a computer graphics technique similar to ray tracing, in that rays are cast from a camera into the scene. Path tracing differs by generating light transport paths using a chain of rays. Rays are cast recursively along these paths to estimate the transported radiance. These rays are traced recursively until hitting a light source, with the rendering equation being evaluated along the path. There is no distinction made between illumination emitted from a light source and illumination reflected from a surface. The ray directions are governed by a bidirectional reflectance distribution function (BRDF) that defines how light is reflected at the surface of an object. Path tracing naturally simulates global illumination effects such as indirect lighting, soft shadows and caustics, that have to be specifically added to other rendering methods. Some of these effects are visible in Figure 5.17, including colour bleeding and reflections. When combined with physically accurate models of both objects and materials, path tracing can produce photorealistic images, and is commonly used to generate reference images for other rendering algorithms due to its unbiased nature.

To avoid artifacts caused by the random sampling of light paths, a large amount of samples are needed per pixel to produce a noise free image. Biased path tracers accept the loss of a small amount of accuracy in return for far fewer samples needed

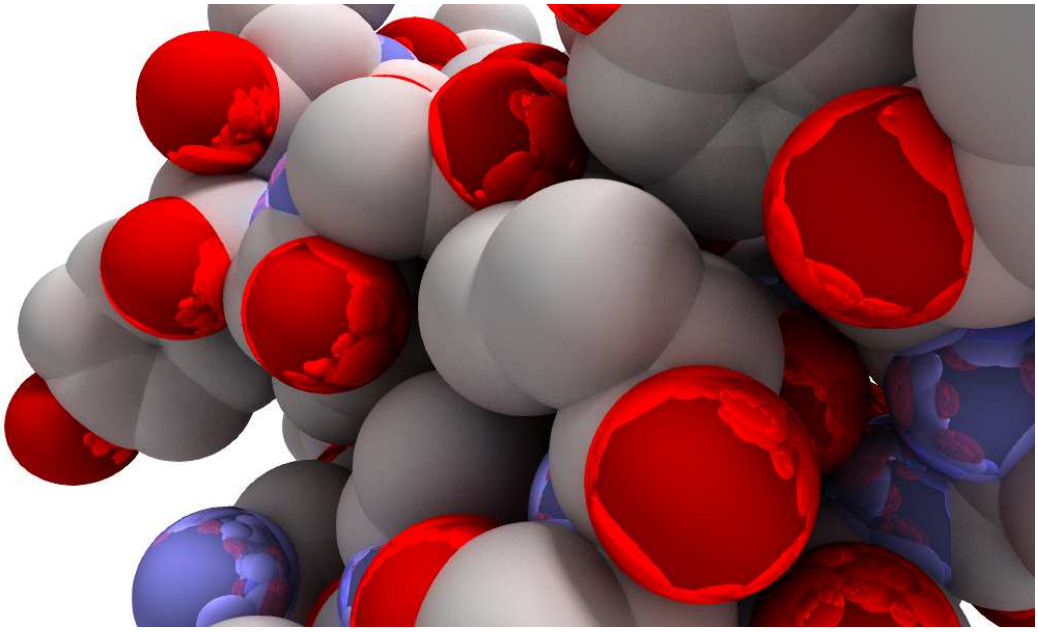


Figure 5.17: Here we see a molecule rendered using path tracing. The light path was allowed to bounce up to three times in the scene. Atoms were given different material properties, helping to quickly distinguish between element types. Reflections can be seen on the red oxygen atoms and blue nitrogen atoms.

per pixel to generate a noise free image. There are many techniques employed to help reduce the samples needed per pixel, and therefore rendering time of the image.

In unbiased path tracing, a path's radiance is considered zero if it does not hit a light source. However, it is unlikely to hit a light source using randomly generated ray directions, especially if the light source is small. As lights in scene are likely to effect surface radiance the most, lights can be sampled directly at each intersection point along the path.

Rays in an unbiased path tracer are generated uniformly over a surface. However, it is far more efficient to generate rays in the directions the luminance would have been greater. Importance sampling is a technique commonly used to cast less rays while still converging to a correct luminance value. If the density of rays cast in certain directions matches the strength of contributions in those directions, the result

will be the same, but far fewer rays were cast.

In an unbiased path tracer, light paths bounce infinitely in the environment until a light source is hit, or they leave the scene. However, as every bounce has less energy, a light path that has bounced many times will have little impact of the surface radiance. Limiting the number of light path bounces, commonly to three or four per pixel, helps reduce rendering time dramatically with little impact on the final image. Figure 5.18 illustrates the basic principles of path tracing.

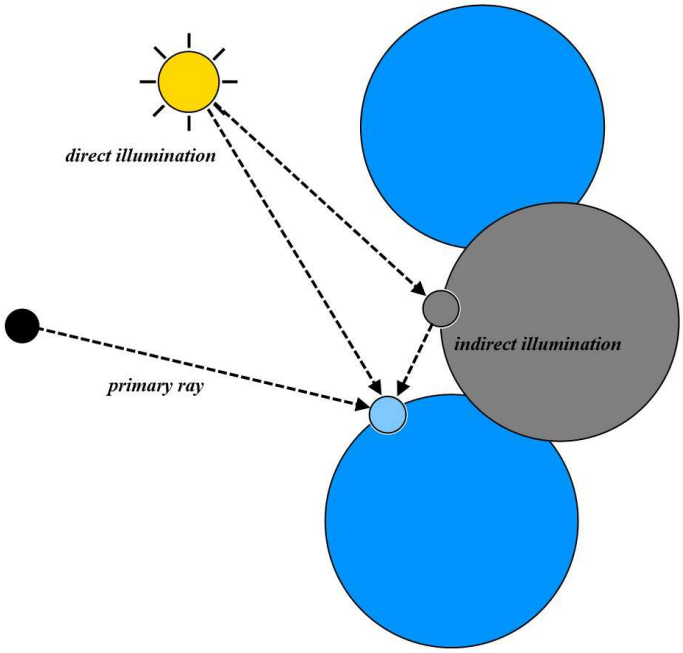



Figure 5.18: This diagram illustrated how the colour of a pixel is determined using path tracing. The light source is directly sampled at each intersection point. Indirect illumination is gathered along the light path, which is based on the material BRDF.

Recent advances in graphics hardware have started to allow straight forward and intuitive algorithms such as path tracing to be considered for real time applications. Our ray tracer, detailed in Section 5.2, can be easily extended to allow for multiple bounces of light rays per pixel. The two main light sources in our scene, a directional light and a probe light, can be sampled directly when calculating the direct lighting

effects. Algorithm 2 describes the simple path tracing method used in our GLSL lighting shader.

**Algorithm 2** Path tracing algorithm

```
1: for all pixels containing geometry do
2:     for number of bounces do
3:         get minimum intersection point
4:         if no hit then
5:             return total accumulated colour
6:         end if
7:         calculate direct lighting
8:         colour accumulation *= current surface colour
9:         total accumulated colour += colour accumulation * direct lighting
10:        set ray origin to intersection point
11:        set ray direction based on material BRDF
12:    end for
13:    pixel colour = mix(accumulated colour, previous frame's colour, number of samples)
14: end for
```

We measured the rendering speeds of path tracing on a number of molecules, using our development machine from previous performance tests, as well as a desktop PC with a Nvidia GeForce GTX 580 graphics card. For our implementation, around 100 samples per pixel are needed to reduce noise to an acceptable level, however real time rates with this number of samples are clearly not yet achievable even when using a high end GPU. Progressive rendering, a method for incremental computation of the image, can allow for the scene to remain interactive, gathering additional samples as the camera stops moving. Even this can not be achieved on our development machine, taking 83 milliseconds to calculate one sample on a small molecule. However, using the GTX 580 we can achieve a minimum of one sample per pixel on all the molecules tested, while achieving at least 30 frames per second. This highlights the speed of progression seen in modern graphics cards.

Table 5.3: Path tracing performance with 100 samples per pixel, 2 light sources, and the light path allowed to bounce up to 3 times in the scene. Render times are in seconds.

| Name | Amount | ATI Mobility Radeon HD 5650 | GeForce GTX 580 |
|---|---|---|---|
| 1CRN | 327 | 8.33 | 1.53 |
| 1BKS | 5231 | 19.62 | 1.75 |
| 3ETS | 9152 | 24.77 | 2.13 |
| 1JB0 | 24198 | 26.85 | 2.34 |
| 1AON | 58870 | 31.89 | 2.86 |

## 5.7 Conclusion

In this chapter we hope to have shown how shadows play an important role in improving the visual depth perception of molecules, aiding the user in navigating the structure with more efficiency. Shadows were achieved using ray tracing techniques implemented on the GPU with GLSL, including a uniform grid acceleration structure to help achieve real time frame rates. Giving the probe sphere the ability to cast shadows onto the molecular structure allows the user to gain a better understanding to the position of the probe relative to other atoms. Clipping a molecule to the navigation cube plays an important role in exploring large molecules at an interactive frame rate. We have looked at methods to ensure shadow effects change dynamically in response to the user moving the cube, creating a more visually believable scene. Additional improvements to the shadows to give them a more natural soft feel were also incorporated with only a small increase in processing cost. We finally looked at path tracing as a future rendering method in molecular graphics, noting how graphics hardware is progressing towards real time rates for simple scenes using this method.

# Chapter 6

# Conclusions

## 6.1 Discussion and Conclusions

The results of this thesis have been to improve the rendering methods for the HaptiMOL iSAS software. The software enables users to interact with the solvent accessible surface of biomolecules, by probing the surface with a sphere. These rendering improvements provide a number of benefits to users when exploring and navigating a molecular structure.

Using a billboarded approach to render spheres that represent atoms in the molecule gave the most significant performance benefit to the software. With this technique, we can reduce the number of vertices that represent the atom down from hundreds in the original software, to just four using this method. Even with this technique, however, frame rates drop below real time rates on our development machine when rendering molecules above 50,000 atoms. The use of point sprites has been shown as a possible way to render a sphere to the screen with just a single vertex sent through the graphics pipeline, with significant performance gains.

There are other notable advantages to using this billboard technique over a traditional triangle mesh. A ray-sphere intersection test is performed per pixel onto the billboard, giving a noticeable visual benefit by ensuring a sphere's representation is

accurate to the nearest pixel. This is desirable as the haptic feedback provided by the device will exactly match the visual representation on the screen.

The software features a clipping plane to aid the user in exploring large data sets. In its old form, the software supported a single clipping plane attached to the probe sphere, helping the user maintain a clear view of the probe, whilst aiding in finding pockets and channels in the structure that may have been hidden from view. Experimentation has shown how these planes can also be used to help reduce rendering time by reducing draw calls, especially for large molecules, where the system is limited by the number of vertices in the scene. Our improvements to the clipping plane implementation include allowing atoms to appear partially culled, giving the user a precise visual cue to the location of the plane in relation to the atoms. The clipping plane can also be placed at any angle chosen by the user, with additional clipping planes supported if needed.

While the performance and appearance of the molecule's geometry is a very important aspect of the software, we were also interested in modernising the lighting techniques currently employed in the system. Ambient occlusion, although only an approximation of full global illumination, has been show with great effect by Tarini et al. [TCM06] to enhance a users perception of a molecular structure.

Their chosen method was to pre-compute the ambient occlusion, storing the values in a texture map for each atom. However, our software allows a user to interact with a molecule, through moving a probe sphere, as well as removing areas of the molecule with clipping planes. We needed a ambient occlusion solution that could handle these dynamic scenes, to enable the user to receive real time visual feedback of the lighting effects based on their input with the device.

We have spent time investigating a number of real time solutions for calculating

ambient occlusion. Each of these methods has their own advantages and disadvantages, and our goal was to find a solution that would best fit our requirements. One choice was to use a popular technique called screen space ambient occlusion. This method is ideal for large scenes due to its independence from scene complexity, although calculating occlusion in screen space leads to local and view dependant results. We also investigated ray tracing as a tool to capture the occlusion. This method provides a better way to capture a more global occlusion effect, however it comes at a large performance cost. We looked at reducing the number of samples, whilst incorporating a blurring stage to reduce this cost to a more acceptable level. Although we do see a notable loss of localised detail due to the lower sample count and blurring filter. A more unique and specialised method we considered was to analytically calculate the ambient occlusion of spheres, demonstrated by Iñigo Quílez [Qui06]. This proved to be a very effective way at producing high quality local ambient occlusion effects with relatively small performance costs, although unable to capture global occlusion effectively. We finally looked at using a combination of these methods, allowing us to take advantages from different techniques to produce an outcome that a single technique on their own could not achieve. A full comparison can be seen in Section 4.6, with Figure 4.9 giving a visual comparison of all the techniques.

Shadows are perhaps the most important of visual cues to perceive both depth and location of objects in a 3D scene. We have looked at using shadows to aid the user navigate the structure with more efficiency.

In Chapter 3, we looked at efficient methods to produce ray traced spheres using billboards. In Chapter 5, we looked at the possibility at continuing the ray tracing theme, to produce real time ray traced shadows on the GPU. We had to consider memory requirements, and an existing structure proposed by Purcell et al. [PBMH02] was chosen, meaning we could support molecules well over 100,000 atoms in size. We

also looked at acceleration structures that would map well to GLSL, with a uniform grid proving the easiest to implement. The results are promising, able to maintain real time rates on our development machine. There are limitations to using a uniform grid as an acceleration structure, namely a large amount of empty grid cells are produced as the cell sizes decrease. We therefore suggest a cell size that balances performance with memory requirements.

We also looked at adding shadows to another existing light source in the software, located at the center of the probe sphere, illuminating areas of the atoms in close proximity to the probe. Allowing this light source to produce shadows would give the user a better understanding to the position of the probe relative to other atoms. Techniques to optimise the shadow calculations of a point light were developed, such as performing calculations only on certain pixels, and reducing grid traversals.

A technique for calculating soft shadows using a single shadow ray was implemented, with a small impact on rendering performance. To help reduce shadow calculation times on larger molecules, we added an option to render only a small section of a molecule. This allows a user to still use these advanced lighting techniques on molecules that would otherwise not be rendered at interactive rates. We also added the option to render a the backbone trace in areas outside the navigation cube, helping the user visualise the full structure.

Given current rates of GPU development, it will not be long before accurate global illumination techniques, such as path tracing, become possible to achieve in real time. We extended our ray tracing system to one that can bounce multiple rays per pixel. With this technique, we could achieve effects such as reflection and refraction, as well as colour bleeding. Atoms were given different material properties to help distinguish between them, as well as aid in depth perception. The rendering speed using a modern high end GPU proved to be vastly superior to the medium range laptop we

were developing on previously, taking a large step towards real time performance. However, a new generation of graphics cards will be required for interactive usage of this method.

Overall, we have investigated and implemented rendering methods that will benefit users of our software. The system built allows graphics effects to be tuned to the users taste, as well as turned off completely. This will allow the software to run on a large variety of computers, both low and high end. The GLSL shaders are built dynamically, ensuring the minimal and optimal code is run each frame, reducing costly operations such as branching. The effects are high quality and real time, both goals set from the outset.

## 6.2 Future Work

Ambient occlusion and shadows are just two of many graphical effects that enhance depth perception. Whilst our approach was to reproduce a realistic visualisation of the molecule, we could have also included techniques such as depth of field. This is the distance between the nearest and farthest objects in a scene that appear acceptably sharp in an image. This could be used based on the location of the probe sphere, blurring objects a certain distance from the sphere, allowing the user to focus on just a small area of the molecule. Other post processing techniques that may prove useful could be depth cuing techniques such as depth aware contour lines of haloing effects proposed by Tarini et al. [TCM06].

Even using point sprites, large molecules become hard to render in real time on lower specification PCs. Occlusion culling could be looked at as a way to reduce the number of draw calls needed to render the structure.

HaptiMOL ENM is software which allows forces to be applied to atoms of an elastic

network model, using a haptic feedback device. It used a ball and stick representation to visualise the molecule. Our ray tracing techniques may be able to be implemented into with software, adding the ability to produce ray traced cylinders that represent the stick.

We have shown how path tracing can be applied to molecular graphics to provide global illumination in the scene. While we could not achieve a real time rate of 30 Hertz using this method, a combination of future hardware, as well as filtering and importance sampling techniques to reduce the number of samples required per pixel, will soon allow this method to be used in interactive molecular graphics programs.

# Bibliography


[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS Spring Joint Computing Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 37–45. Thomson Book Company, Washington D.C., 1968.

[AW59] Berni J Alder and TE Wainwright. Studies in molecular dynamics. i. general method. *The Journal of Chemical Physics*, 31:459, 1959.

[AW87] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *In Eurographics 87*, pages 3–10, 1987.

[BK03] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern gpus. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 335–343. IEEE, 2003.

[BKB+07] Cagatay Basdogan, Alper Kiraz, Ibrahim Bukusoglu, Aydın Varol, Sultan Doğanay, Cécile Pacoret, Richard Bowman, Graham Gibson, Sinan Haliyo, David Carberry, et al. Haptic guidance for improved task performance in steering microparticles with optical tweezers. *Opt. Express*, 15(18):11616–11621, 2007.

[Bon64] A. Bondi. van der waals volumes and radii. *The Journal of Physical Chemistry*, 68(3):441–451, 1964.

[BP04] Michael Bunnell and Fabio Pellacini. Shadow map antialiasing. *GPU Gems: Programming Tech Tricks for Real-Time Graphics*, 2004.

[BS09] Louis Bavoil and Miguel Sainz. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009: Talks*, page 45. ACM, 2009.

[BSD08] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH '08, pages 22:1–22:1, New York, NY, USA, 2008. ACM.

[Bun05] Michael Bunnel. Dynamic ambient occlusion and indirect lighting. In Matt Pharr and Randima Fernando, editors, *GPU Gems 2*, pages 223–233. Addison Wesley, 2005.

[CHI] Chime-mdl discovery framework. http://iop.vast.ac.vn/theor/conferences/smp/1st/kaminuma/ChemDraw/chemscape.html [Accessed on 24.01.14].

[Chr02] Per H Christensen. Note# 35: Ambient occlusion, image-based illumination, and global illumination. *PhotoRealistic RenderMan Application Notes*, 2002.

[Chr05] Martin Christen. Ray tracing on gpu. 2005.

[Cor] NVIDIA Corporation. Cuda. http://www.nvidia.com/object/cuda_home_new.html [Accessed on 24.01.14].

[CPC84] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 137–145. ACM, 1984.

[Cro77] Franklin C Crow. Shadow algorithms for computer graphics. In *ACM SIGGRAPH Computer Graphics*, volume 11, pages 242–248. ACM, 1977.

[Del02] Warren Lyford Delano. The pymol molecular graphics system, 2002. http://www.pymol.org [Accessed on 24.01.14].

[Dim07] Rouslan Dimitrov. Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*, 2007.

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165. ACM, 2006.

[DWS+88] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *ACM SIGGRAPH Computer Graphics*, volume 22, pages 21–30. ACM, 1988.

[EK03] Cass Everitt and Mark J Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. *arXiv preprint cs/0301002*, 2003.

[FC66] Robert M Fano and Fernando J Corbató. Time-sharing on computers. *Scientific American*, 215:128–140, 1966.

[FI85] Akira Fujimoto and Kansei Iwata. Accelerated ray tracing. In TosiyasuL. Kunii, editor, *Computer Graphics*, pages 41–65. Springer Japan, 1985.

[FM08] Dominic Filion and Rob McNaughton. Effects & techniques. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 133–164, New York, NY, USA, 2008. ACM.

[Fra] Crytek Frankfurt. Crysis. http://www.crysis.com/ [Accessed on 24.01.14].

[FS05] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22. ACM, 2005.

[GP+03] Gael Guennebaud, Mathias Paulin, et al. Efficient screen space approach for hardware accelerated surfel rendering. In *VMV*, volume 20003, pages 1–10, 2003.

[Groa] Khronos Group. Opencl. http://www.khronos.org/opencl/ [Accessed on 24.01.14].

[Grob] Khronos Group. Opengl. http://www.khronos.org/opengl/ [Accessed on 24.01.14].

[GTGB84] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 213–222. ACM, 1984.

[Hav00] Vlastimil Havran. *Heuristic ray shooting algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University, 2000.

[HDS96] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[Hei91] Tim Heidmann. Real shadows, real time. *Iris Universe*, 18:28–31, 1991.

[IUP] Iupac compendium of chemical terminology, electronic version. http://goldbook.iupac.org/MT06970.html [Accessed on 24.01.14].

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques 96*, pages 21–30. Springer, 1996.

[Jmo] Jmol: an open-source java viewer for chemical structures in 3d. http://www.jmol.org/ [Accessed on 24.01.14].

[Joh65] CK Johnson. Ortep, a fortran thermal-ellipsoid plot program for crystal structure illustrations. 1965.

[Kaj86] James T Kajiya. The rendering equation. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[KCLU07] Johannes Kopf, Michael F Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Trans. Graph.*, 26(3):96, 2007.

[KL05] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48. ACM, 2005.

[KMK94] Daniel Kersten, Pascal Mamassian, and David C. Knill. Moving cast shadows and the perception of relative depth. *Max-Planck-Institute for Biological Cybernetics Technical Report*, 1994.

[Kol65] Walter L. Koltun. Precision space-filling atomic models. *Biopolymers*, 3(6):665–679, 1965.

[Lan02] Hayden Landis. Production-ready global illumination. In *Siggraph Course Notes*, volume 16, 2002.

[Lau07] Andrew Lauritzen. Summed-area variance shadow maps. *GPU Gems*, 3:157–182, 2007.

[LB00] M. S. Langer and H. H. Bülthoff. Depth discrimination from shading under diffuse lighting. *Perception*, 29:49–660, 2000.

[Lev66] Cyrus Levinthal. *Molecular model-building by computer*. WH Freeman and Company, 1966.

[Lot11] Timothy Lottes. Fxaa. *NVIDIA white paper*, 2011.

[LS10] Bradford James Loos and Peter-Pike J. Sloan. Volumetric obscurance. In Daniel G. Aliaga, Manuel M. Oliveira, Amitabh Varshney, and Chris Wyman, editors, *SI3D*, pages 151–156. ACM, 2010.

[MDG+10] Lukas Marsalek, Anna Katharina Dehof, Iliyan Georgiev, Hans-Peter Lenhof, Philipp Slusallek, and Andreas Hildebrandt. Real-time ray tracing of complex molecular scenes. In Ebad Banissi, Stefan Bertschi, Remo Aslak Burkhard, John Counsell, Mohammad Dastbaz, Martin J. Eppler, Camilla Forsell, Georges G. Grinstein, Jimmy Johansson, Mikael Jern, Farzad Khosrowshahi, Francis T. Marchese, Carsten Maple, Richard

Laing, Urska Cvek, Marjan Trutschl, Muhammad Sarfraz, Liz J. Stuart, Anna Ursyn, and Theodor G. Wyeld, editors, *IV*, pages 239–245. IEEE Computer Society, 2010.

[Mit07] Martin Mittring. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121, New York, NY, USA, 2007. ACM.

[MKK98] Pascal Mamassian, David C. Knill, and Daniel Kersten. The perception of cast shadows. *Trends in Cognitive Sciences*, 2(8):288 – 295, 1998.

[MT04] Tobias Martin and Tiow Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Rendering Techniques*, pages 153–160, 2004.

[Muu87] Michael J. Muuss. Rt and remrt: Shared memory parallel and network distributed ray-tracing programs. In *Fourth Computer Graphics Workshop Proceedings*, pages 86–98. USENIX, 1987.

[Ngu07] Hubert Nguyen. *Gpu gems 3*. Addison-Wesley Professional, first edition, 2007.

[PBD+10] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *ACM Transactions on Graphics (TOG)*, 29(4):66, 2010.

[PBMH02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 703–712, New York, NY, USA, 2002. ACM.

[PG04] Matt Pharr and Simon Green. Ambient occlusion. In Randima Fernando, editor, *GPU Gems*, pages 279–292. Addison Wesley, 2004.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[Por78] Thomas K Porter. Spherical shading. In *ACM SIGGRAPH Computer Graphics*, volume 12, pages 282–285. ACM, 1978.

[Pov] Persistence of vision pty. ltd. (2004) persistence of vision raytracer (version 3.6) [computer software]. Retrieved from http://www.povray.org/download/ [Accessed on 24.01.14].

[Pro] Protein explorer homepage. http://www.umass.edu/microbio/chime /pe_beta/pe/protexpl/ [Accessed on 24.01.14].

[PSS98] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. *Most*, pages 1–6, 1998.

[Qui06] Inigo Quilez. Sphere ambient occlussion, 2006. http://www.iquilezles.org/www/articles/sphereao/sphereao.htm [Accessed on 24.01.14].

[RBA09] Christoph K Reinbothe, Tamy Boubekeur, and Marc Alexa. Hybrid ambient occlusion. In *Eurographics 2009-Areas Papers*, pages 51–57. The Eurographics Association, 2009.

[RBC53] Linus Pauling Robert B. Corey. Molecular models of amino acids, peptides, and proteins, 1953.

[Rot82] Scott D Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2):109 – 144, 1982.

[RSC87] William T Reeves, David H Salesin, and Robert L Cook. Rendering antialiased shadows with depth maps. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 283–291. ACM, 1987.

[RWS+06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 977–986. ACM, 2006.

[SA07] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *In I3D 07: Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM*. Press, 2007.

[SB92] Roger Sayle and Andrew Bissell. Rasmol: A program for fast, realistic rendering of molecular structures with shadows. In *Proceedings of the 10th Eurographics UK*, volume 92, pages 7–9, 1992.

[SB08] Erk Subasi and Cagatay Basdogan. A new haptic interaction and visualization approach for rigid molecular docking in virtual environments. *Presence: Teleoperators and Virtual Environments*, 17(1):73–90, 2008.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 557–562. ACM, 2002.

[SHL09] Matthew Stocks, Steven Hayward, and Stephen Laycock. Interacting with the biomolecular solvent accessible surface via a haptic feedback device. *BMC structural biology*, 9(1):69, 2009.

[SK04] M Sarletu and G Klein. Hardware-accelerated ambient occlusion computation. In *Vision, modeling, and visualization 2004: proceedings, November 16-18, 2004, Standford, USA*, pages 331–338, 2004.

[Stu] Pixar Animation Studios. Renderman. http://renderman.pixar.com/ [Accessed on 24.01.14].

[SWBG06] Christian Sigg, Tim Weyrich, Mario Botsch, and Markus Gross. Gpu-based ray-casting of quadratic surfaces. In *Proceedings of the 3rd Eurographics/IEEE VGTC conference on Point-Based Graphics*, pages 59–65. Eurographics Association, 2006.

[SWS+03] Ganesh Sankaranarayanan, Suzanne Weghorst, Michel Sanner, Alexandre Gillet, and Arthur Olson. Role of haptics in teaching structural molecular biology. In *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings. 11th Symposium on*, pages 363–366. IEEE, 2003.

[TCM06] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

[Wan92] Leonard Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, I3D '92, pages 39–42, New York, NY, USA, 1992. ACM.

[WC+53] James D Watson, Francis HC Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[WFG92] Leonard R Wanger, James Ferwerda, and Donald P Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, 1992.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pages 270–274. ACM, 1978.

[WPF90] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *Computer Graphics and Applications, IEEE*, 10(6):13–32, 1990.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques*, pages 143–151. Eurographics Association, 2004.

[ZIK98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. An ambient light illumination model. In *Rendering Techniques*, pages 45–56, 1998.

[ZSXL06] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 311–318. ACM, 2006.