High quality rendering of protein dynamics in space filling mode

Nick Matthews[a], Robert Easdon[a], Akio Kitao[b], Steven Hayward[a], Stephen Laycock[a,*]

[a]*School of Computing Sciences, University of East Anglia, Norwich Research Park, Norwich, UK, NR4 7TJ*
[b]*School of Life Science and Technology, Tokyo Institute of Technology, 2-12-1 Ookayama, M6-13, Meguro, Tokyo 152-8550, Japan*

Abstract

Producing high quality depictions of molecular structures has been an area of academic interest for years, with visualisation tools such as UCSF Chimera, Yasara and PyMol providing a huge number of different rendering modes and lighting effects. However, no visualisation program supports per-pixel lighting effects with shadows whilst rendering a molecular trajectory in space filling mode.

In this paper, a new approach to rendering high quality visualisations of molecular trajectories is presented. To enhance depth, ambient occlusion is included within the render. Shadows are also included to help the user perceive relative motions of parts of the protein as they move based on their trajectories. Our approach requires a regular grid to be constructed every time the molecular structure deforms allowing per-pixel lighting effects and ambient occlusion to be rendered every frame, at interactive refresh rates. Two different regular grids are investigated, a fixed grid and a memory efficient compact grid.

The algorithms used allow trajectories of proteins comprising of up to 300,000 atoms in size to be rendered at ninety frames per second on a desktop computer using the GPU for general purpose computations. Regular grid construction was found to only take up a small proportion of the total time to render a frame. It was found that despite being slower to construct, the memory efficient compact grid outperformed the theoretically faster fixed grid when the protein being rendered is large, owing to its more efficient memory access patterns. The techniques described could be implemented in other molecular rendering software.

[*]Corresponding author, s.laycock@uea.ac.uk

1. Introduction

Viewing biomolecular structures at an atomic resolution has given researchers insight into the relationship between protein structure, dynamics, function, and how they might interact with binding partners. A high quality visualisation that highlights surface features is useful as pockets and crevices are often where ligands bind. Graphical effects such as shadows and ambient occlusion can be used to make these features more obvious.

Ambient occlusion has been used within molecular rendering systems to enhance the topography of rendered structures, whilst development of protein viewing software that generates real time shadows has so far been neglected. The reason for this could be because ambient occlusion highlights the surface features within structures so well that cast shadows have been considered an expensive and almost unnecessary addition to the scene. Research has shown that shadows do provide insight into how an object is positioned, relative to other objects, in 3D space (Wanger et al., 1992). We believe the same effect will be useful to aid understanding in how parts of the protein move in relation to one another whilst rendering a trajectory.

Until recently, the computational power available on a desktop computer has been insufficient to generate ambient occlusion and cast shadows with per-pixel accuracy in real time. However, modern GPUs (Graphics Processing Units) provide, through use of general purpose GPU computing, large amounts of computational power that can be used to generate these effects.

In this paper we describe techniques for rendering high quality visualisations of biomolecular trajectories with per-pixel shadows and ambient occlusion. Our molecular shadow algorithm utilises ray casting to produce per-pixel shading on atomic structures. We also demonstrate effective use of a sphere based analytical ambient occlusion algorithm and apply it to a molecular trajectory. Both algorithms are reliant on a regular grid. By utilising the GPU, the regular grid used by these algorithms can be reconstructed per-frame, allowing full molecular trajectories to be rendered in real time whilst maintaining the quality of the render. GPU based grid construction is explored in detail, with two different grid algorithms described, implemented and compared. The techniques discussed in this paper are implemented in our software, *Protein Trajectory Viewer*, which is available to download from `http://www.haptimol.co.uk/files/proteintrajectoryviewer/setup_ptv.zip`

2. Previous work

In this section current molecular rendering applications are discussed, then we review existing techniques that could be used to generate the desired graphical effects.

2.1. Molecular Rendering

A large number of protein renderers are freely available. RasMol (Sayle and Milner-White, 1995) was one of the first protein viewer applications. It has been used widely since 1995 and has the ability to render a number of molecular visualisations, including ball and stick, spacefill and wireframe.

Since the publication of RasMol, a huge number of protein viewers have been developed and released to the community, all with different feature sets. A popular feature, included in VMD (Humphrey et al., 1996), PyMol (Schrödinger, LLC, 2015), and Avogadro (Hanwell et al., 2012) amongst others, is the ability to render high quality, often ray traced, depictions of proteins to file. This allows high quality renders to be produced, however not in real time.

Several molecular renderers have incorporated ambient occlusion, including Caver Analyst (Kozlikova et al., 2014), Yasara (Krieger and Vriend, 2014), MegaMol (Grottel et al., 2015) and Qutemol (Tarini et al., 2006) to improve the quality of their real-time renders.

Real-time shadows have also been used to great effect within molecular renderers. PyMol and RasMol both support rendering silhouette type shadows around the atoms, UCSF Chimera (Pettersen et al., 2004), Yasara (Krieger and Vriend, 2014) and QuteMol (Tarini et al., 2006) demonstrate a full shadow technique.

PyMol, Chimera, Caver, Yasara, VMD and MegaMol all have the ability to render molecular trajectories, however, to the best of our knowledge, no molecular renderer supports the rendering of trajectories with per-pixel shadows and ambient occlusion in real time on consumer grade hardware.

BallView (Moll et al., 2006) is a molecular tool that can be used to visualise proteins. BallView is capable of producing very high quality images because the RTFact CPU ray tracer has been included (Marsalek et al., 2010). BallView runs at interactive frame rates for small structures on consumer hardware: rendering Cranbin on our test workstation, equipped with a 4GHz Intel i7 and a GTX 980, BallView with ray tracing enabled achieves 8.5 FPS.

2.2. Rendering techniques

Generating realistic ambient occlusion and shadow effects has been a subject of research for many years, with many different approaches presented to the community.

2.2.1. Ambient Occlusion

Ambient occlusion shading is a popular way of enhancing the perception of depth of a three dimensional scene. The most straight forward and accurate way of calculating ambient occlusion is with ray tracing. After many rays have intersected the hemisphere centred on a point, a high quality approximation of the ambient occlusion is produced. The main limitation of this approach is the amount of computation required to produce a realistic result, although progress is being made in this area.

Ambient occlusion can be calculated in object-space or screen-space. The first method, object-space ambient occlusion, computes an occlusion value from the geometry within the scene. The alternate method, Screen Space Ambient Occlusion (SSAO), calculates ambient occlusion within camera space.

An early technique to render ambient occlusion in object space was presented by Pharr and Green (2004). The algorithm described uses a preprocessing step to calculate how much of the external environment can be seen from each point on a model, then uses this information to compute the visible lighting of that point. The preprocessing step is too expensive to consider integrating into the display loop.

Sattler et al. (2004) presented a hardware accelerated ambient occlusion algorithm that worked by constructing a visibility matrix. The method can be applied to dynamic scenes, however its limitation is it is applied per vertex rather than per pixel, meaning performance is dependant on the number of the vertices in the scene. Also, applying occlusion per vertex results in artefacts caused by under sampling, which can be seen as aliasing at the shadow's edges. A similar approach was used by Tarini et al. (2006) within QuteMol.

An alternate idea, first presented by Bunnell (2005) and later improved by Hoberock and Yuntao (2007), works by approximating an accessibility value of each element in the scene. The method does have limitations, namely, discontinuities may be visible within the ambient occlusion and often the resulting occlusion is biased toward darkness.

Another method, published by Kontkanen and Laine (2005), computes inter-object AO in real time. A grid surrounding each occluder is computed, then used by the fragment shader to calculate each object's occlusion upon a point. Although real-time performance is achieved, this method suffers from a long pre-computation time and has the potential to use a large amount of memory, especially if the number of object's within the scene is large.

Volume based ambient occlusion was presented by Papaioannou et al. (2010) and also used by Grottel et al. (2012). The methods work by storing occlusion information inside a volume. The method presented by Grottel et al. (2012) is used to great effect within the software "MegaMol." The method involves generating a volume which contains occlusion values for points in the scene. The occlusion values are calculated by considering the density of the scene around each point. The method presented by Grottel et al. (2012) supports rendering very large datasets at high frame rates. However, the quality of the final ambient occlusion effect is dependant on the resolution of the volume texture. The authors also note that artefacts can occur if the model being rendered does not align with the density volume.

Favera and Celes (2012) presented a cone-tracing based algorithm. The algorithm follows a comparable approach to that of ray tracing, however fewer cones are required per point than rays. An earlier cone-tracing algorithm was demonstrated by Crassin et al. (2011). A fast cone tracing implementation was developed by Staib et al. (2015), the algorithm presented is limited to non-overlapping spheres, because artefacts occur when the rendered spheres overlap, owing to the use of point-based rendering.

An analytical object-space ambient occlusion algorithm was presented by Skånberg et al. (2016). The algorithm described exploits the fact that all the geometry in the scene is spherical, and uses the solid angle formula to determine the fraction of any neighbouring spheres that are visible from a point. This is then used to compute the ambient occlusion contribution of the neighbouring sphere to the point in question. The technique used produces an ambient occlusion effect, however it is highlighted within the paper that there are limitations to this approach, mainly in the fact that a large search neighbourhood will result in poor performance, and a smaller one will result in a poor estimation of ambient occlusion.

Screen space ambient occlusion (SSAO) is an alternative method for calculating ambient occlusion. SSAO is generally less expensive than object space occlusion because the amount of work completed by the algorithm is dependent on the resolution of the render, rather than the number of objects in the scene. However, compared with object space occlusion, it suffers from some disadvantages, namely, the technique is view dependant, and is consequently unable to take into account geometry that is not rendered to the screen.

The first major development in SSAO was by Mittring (2007). The technique calculates a per-fragment occlusion value using the depth buffer. Each pixel calculates its own occlusion value by randomly sampling different points around itself and computes the amount of occlusion applied to it based on differences in depth. The algorithm was later improved by Filion and McNaughton (2008).

Other screen space ambient occlusion methods have been presented, including Horizon Based Ambient Occlusion (HBAO) presented by Bavoil et al. (2008) and improved by Bavoil and Sainz (2009) and Volumetric Obscurance (Loos and Sloan, 2010). Other improvements on the original technique have been presented, including work by Shanmugam and Arikan (2007) who present an approximation to ambient occlusion by separating near and far ambient occlusion, and Kajalin (2009) who built on Mitterings work.

A hybrid ambient occlusion algorithm that uses both SSAO and geometry based occlusion was presented by Reinbothe et al. (2009). The algorithm is stated to work in real time, and has the optimisation of only refreshing the AO maps for dynamic portions of the scene. However, as the number of dynamic triangles increases within the scene, it is likely that the frame rate would drop to unacceptably low levels.

2.2.2. Shadows

Shadowing occurs when an object blocks, or partially blocks, light from a source from falling on another surface. Ray tracing is the most elegant way to produce realistic shadows within a three dimensional scene although expensive. Alternative algorithms have been the subject of extensive research since the 1970s. The algorithms developed primarily fall into two broad categories: shadow volumes and shadow mapping.

Shadow volumes were first developed by Crow (1977). The main limitation of shadow volumes is that additional geometry is needed for the shadow volumes. This geometry is costly to generate and to render. Also, there are issues when the shadow caster does not have a mesh that accurately represents an object's shape - for example, when billboards are used. Finally, shadow volumes are not natively compatible with soft shadows, which are often seen in the real world. A recent survey conducted by Kolivand and Sunar (2013) reviews the shadow volume field, including algorithms that support soft shadows. They conclude that despite advances in the field, shadow volumes are still expensive to produce and recommend shadow mapping for use with real-time rendering.

Shadow mapping has proven to be the dominant method to generate real-time shadows in interactive applications, including the molecular graphics applications QuteMol and RasMol. The main disadvantage to shadow mapping is the accuracy of the generated shadow map is limited by its resolution, which is limited by the size of the texture. This inaccuracy can often be seen as aliasing at the edges of the shadows. A further disadvantage is that the scene has to be rendered once per light source, which is time consuming. Often, when scenes are lit with many light sources, the shadow mapping is limited to the main light source, typically the sun. Shadow mapping has been the subject of a large amount of research, with algorithms developed to tackle each of these problems.

Warping algorithms have been investigated as a way to tackle the problem of insufficient shadow map resolution near the eye (Stamminger and Drettakis, 2002), (Wimmer et al., 2004), Martin and Tan (2004). Another area of shadow map research built on the idea that points at different distances from the camera need different shadow map densities. Developed by Zhang et al. (2006) and later improved by Dimitrov (2007), the method works by splitting the frustum, allowing each shadow map to focus on a smaller area and therefore provide a better match between sampling frequencies in view space and texture space. These algorithms can produce an alias free shadow, however the shadow produced has a hard edge.

The first solution for producing a soft shadow effect with a shadow map was presented by Reeves et al. (1987). Multiple shadow map comparisons per pixel were taken and then averaged together, to give a percentage of the surface that is not in shadow. This technique was ported to the GPU by Bunnell and Pellacini (2004).

A further development in shadow mapping, called Variance Shadow Maps, was presented by Donnelly and Lauritzen (2006), but the method suffered from light bleeding. Further research has been undertaken to fix this issue by Lauritzen and Mccool (2007), however the solution is computationally expensive. A solution was presented by Annen et al. (2008) to produce real-time, all-frequency soft shadows with excellent results, however the algorithm uses a single shadow map, and suffers from incorrect shadows being generated for certain geometry. Despite the previous and ongoing research, ray tracing remains a more elegant method to generate pixel-perfect shadows within a scene.

As stated in Section 2.1 real-time ray tracing has been included in a molecular renderer before, however the frame rate achieved was low. Generating shadows with ray casting was first presented by Appel (1968) and following this there has been significant research into ray tracing incorporating shadows. Much of the research has been aimed at accelerating the ray tracing process, however few of the algorithms are quick enough to run in real time Woo and Poulin (2012). Ray tracing has been used in supercomputing based applications. BnsView, presented by Knoll et al. (2013) is a CPU based renderer designed for running on multi-core CPU architectures. The application supported a data set with 15 million atoms at interactive refresh rates, but the hardware used during testing cannot be considered consumer grade. Similarly, Stone et al. (2016) presents an immersive molecular visualisation tool that utilises remote GPU clusters to present a high definition render of a trajectory to a head mounted display.

In the following Section we describe how ray cast shadows can be combined with a computationally inexpensive soft shadow technique to generate pixel accurate shadows upon a molecular surface.

3. Methods

3.1. Rendering

A common way of rendering a sphere in a 3D scene is to create it out of a triangle mesh, however, this approach requires hundreds of triangles to generate a sphere that looks smooth to the human eye. This approach may work for smaller proteins but as the number of atoms being rendered increases, interactivity will quickly be lost, owing to the number of triangles required.

Ray casting is an approach originally presented by Gumhold (2003), and has since been improved by Sigg et al. (2006). The algorithm works by creating a billboard approximately the size of the sphere and casting rays through each of the fragments within it. If the ray collides with the sphere, the fragment is kept, otherwise it is discarded. Ray casting has been used to great effect within other molecular rendering software, including MegaMol (Grottel et al., 2015). It has also been used as part of the process to generate molecular surfaces (Krone et al., 2009), (Lindow et al., 2010).

3.2. Realistic Shadows

Realistic shadows are a useful tool to aid with the perception of a three dimensional environment. Shadows allow us to see geometry that is not visible, either because it is out of view or hidden behind another part of the scene. The ability to render accurate shadows whilst rendering a molecular trajectory will help enhance key changes in the structure during deformation.

A naive ray tracing method is to test each light ray cast against every object in the scene. This method quickly becomes impractical owing to the large number of intersection tests that have to be carried out every frame. Within *Protein Trajectory Viewer*'s ray casting algorithm, the number of tests performed is reduced by using a spatial data structure - a regular grid. The cast ray traverses the grid and only performs intersection tests on the atoms within each of the grid cells that the ray passes through, removing any intersection tests on objects that are nowhere near to the locality of the ray. This reduces the cost of the ray casting algorithm significantly, enough to allow it to be used in real time.

The regular grid construction algorithms are covered in depth in Section 3.4. The traversal algorithm used within *Protein Trajectory Viewer* is the 3D-DDA algorithm originally presented by Amanatides and Woo (1987). 3D-DDA is appropriate because the cost of traversal is small and should the ray "hit" something within its path, the algorithm terminates; it does not test the subsequent cells.

Calculating the shadows with ray casting results in shadows with a hard outline being generated. The cause of this is light within the scene being cast from a single point, making all points either in shadow or in full light. To generate soft shadows using ray casting, an area light source could be used. However, the number of rays cast per pixel will increase dramatically, increasing computation time when compared to a single light source.

An alternative method, presented by Parker et al. (1998), calculates approximate soft shadows from a point light source by using an enlarged sphere. Figure 1 shows how the method by Parker et al. (1998) works. The atom for which the shadow is being calculated is enlarged. Rays are then cast from each of the pixels in the scene toward the light source. If a ray cast from a pixel collides with the original atom, the pixel that cast it is in full shadow and the ray terminates. If the ray collides with the enlarged atom, it is partially in shadow. The level of shadow applied to the pixel is dependent on how close the ray is to intersecting the original sphere. The closer the ray passes to the sphere, the darker the pixel is. The ray does not terminate if it has only collided with the enlarged sphere. Any subsequent spheres that the ray intersects with contribute to the shadow intensity of the pixel. The resulting soft shadow effect can be seen in Figure 2.

3.3. Ambient Occlusion

Ambient occlusion is a popular technique used to enhance the depth perceptibility of a three dimensional render. The objective is to reduce the amount of ambient light shown in enclosed

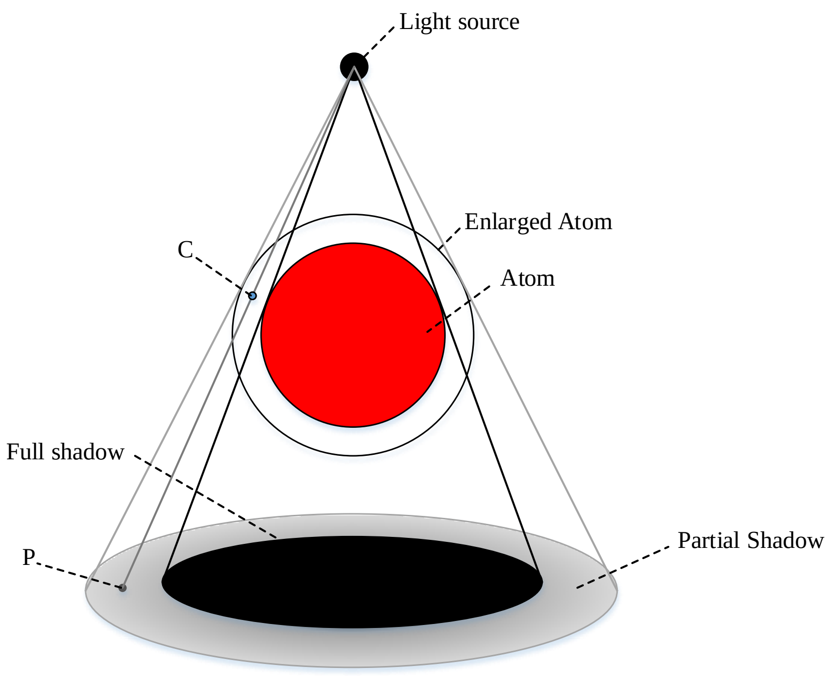

Figure 1: A diagram demonstrating how an enlarged sphere is placed over an atom to create a soft shadow effect. The shadow intensity at point P is half of the full shadow intensity, because its ray passes the atom halfway between the atom and enlarged atom.

portions of a scene. As reviewed in Section 2.1, ambient occlusion has been widely applied to renders of molecular structures. The technique discussed by Skånberg et al. (2016) has positive qualities we would like to implement within our trajectory viewer, specifically a smooth falloff of the occlusion and crisp details with no risk of sampling artefacts.

The algorithm presented by Skånberg et al. (2016) calculates the occlusion applied to a fragment by computing the solid angle of the fraction of neighbouring spheres that can be seen from the fragment the occlusion is being calculated for. We refer the reader to Skånberg et al. (2016) for a more detailed explanation of how the occlusion algorithm works. The ambient occlusion calculation within *Protein Trajectory Viewer* works in the same way, with the exception of how neighbouring atoms are found. Within Skånberg et al. (2016) neighbours are found within the fragment shader by using a grid data structure configured with a cell size equal to the desired cut off. Our approach uses the regular grid that is used to accelerate the shadow casting, and finds a list of neighbours on a per atom basis, rather than per fragment, within a pre-rendering step of the display loop. The advantage of this approach is increased flexibility when setting the desired cut-off distance. The disadvantage to our approach is an extra pre-render computation step is required.

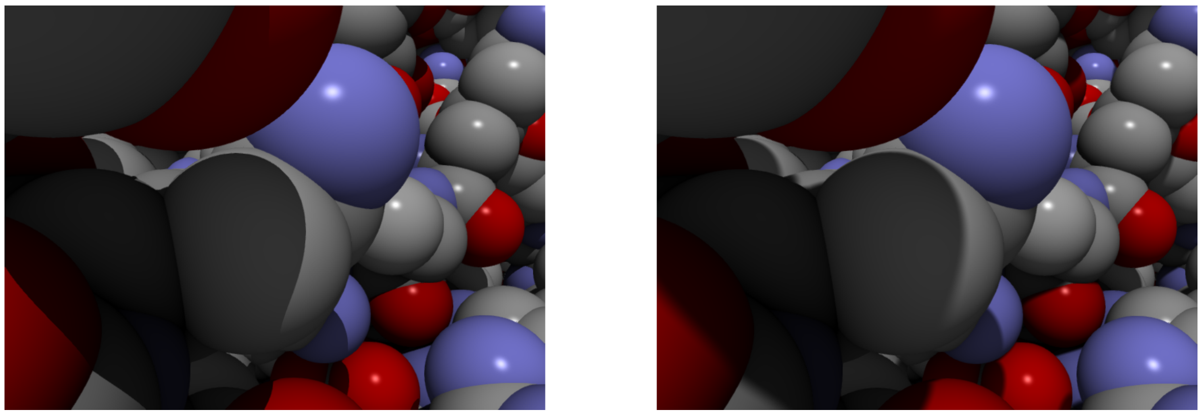

Figure 2: An image showing the difference between hard shadows (left) and soft shadows (right).

The final effect generated by the ambient occlusion is shown in Figure 3 (A). Note how the ambient occlusion darkens the small pockets within the scene. Also, one can see how the ambient occlusion (Figure 3 (A)) and shadows (3 (B)) complement each other. The ambient lighting highlights the small pockets in the structure, whilst the ray cast shadows highlight the large cavities. The final render is shown in Figure 3 (C).

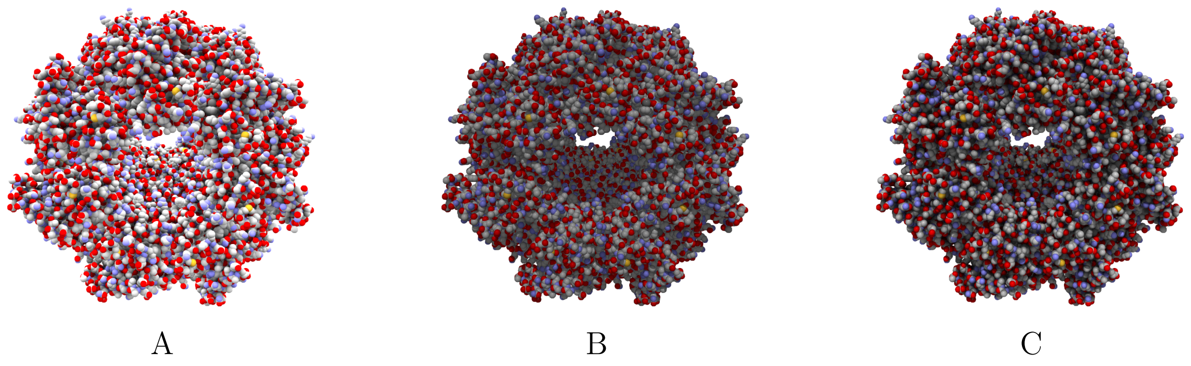

Figure 3: A - GroEL rendered with ambient occlusion only, no directional or diffuse light applied. B - GroEL rendered with directional lighting and shadows, but no ambient occlusion. C - GroEL with both shadows and ambient occlusion applied.

Note the increase in perceptible depth that the combined effect produces. As highlighted in the previous sections, to maintain this level of graphical clarity whilst rendering a trajectory, the occluder lists and regular grid have to be rebuilt after every trajectory time step. In Section 3.4 GPU based regular grid construction is investigated.

3.4. Acceleration Structure Construction

To maintain high quality visual effects whilst rendering the deformation of a protein, the grid used during the rendering process has to be rebuilt every time the rendered scene changes. To ensure an interactive frame rate (twenty four frames per second or higher), the grid reconstruction and rendering must complete in less than 41 milliseconds. For this reason, the data structure is reconstructed on the GPU. This has a number of advantages: firstly, modern GPUs can perform a large number of calculations every second. Secondly, reconstructing the grid on the GPU means very little data transfer has to occur between the host and device during rendering, avoiding costly data transfer between the CPU and GPU.

Depending on the intended use of the grid, two approaches to filling the grid can be taken - the grid can be "tight" or "loose" (Barbieri et al., 2013). In a tight grid each of the elements being inserted is placed into every cell that it touches. If you are inserting large triangles into a very fine grid, this can lead to many references pointing at the same triangle, resulting in many more insertions into the grid. This can be detrimental to the grid's overall performance depending on the application (Barbieri et al., 2013). However, taking a tight approach can provide substantial benefits when using the grid, especially if the grid is being used for a data comparison application; for example collision detection.

The loose grid approach only places one reference per object in the grid. The cell an object is placed in is implementation dependent, but any part of the object being inserted could be used, as long as the same point is used consistently throughout construction (Barbieri et al., 2013).

Our shadow ray casting algorithm relies on the grid to efficiently find all of the atoms the ray may collide with. A tight grid allows all of the atoms that are in the locality of the ray, even if the centre of the atom is in a neighbouring cell, to be tested efficiently. A loose grid would require the ray to test all of the atoms in the neighbouring cells for collisions, even those that are well out of the locality of the ray. As a result of this only tight grid construction algorithms are investigated.

The following sections describe two different tight grid construction algorithms, previously discussed by Green (2010), which are then compared in the results section of this paper. Both algorithms begin with the same first steps. First, a bounding box is calculated around the molecule. Then for each atom in A_i in the scene:

1. Find the cell C_{mid} that contains the centre of A_i.
2. Perform collision tests between the van der Waals radius of A_i and the surrounding cells.
3. Add a reference to the grid for all of the cells that A_i intersects with.

How step three is achieved is algorithm dependent. After the grid has been constructed, it is passed to OpenGL using a Shader Storage Buffer Object, which is written to directly from CUDA.

3.4.1. Fixed Grid

Regular grids can be represented in memory in a variety of ways. The most simple memory layout is depicted in Figure 4. This memory representation will be referred to as the “fixed grid” throughout the rest of this paper. Within a fixed grid, each of the cells has the same amount of memory allocated, which means this mode is the least efficient in regards to memory use. This allows each cell in the grid to be accessed directly, in constant time. The drawback to the fixed grid is how memory inefficient it is because the same amount of memory is allocated to every cell in the grid. Green (2010) presented a method for constructing a fixed grid on the GPU using atomics, and used it within a particle system.

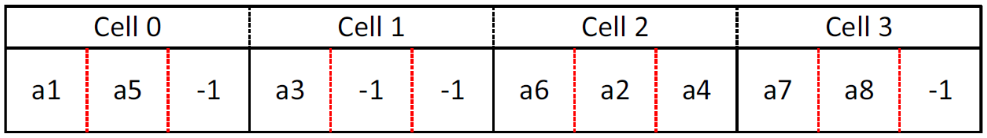

Figure 4: A representation of a one dimensional fixed grid containing eight elements (a1...a8). Note how each cell has the same amount of memory allocated. -1 denotes unused allocated memory within the grid.

The implementation of the fixed grid algorithm implemented within our software is similar to that presented by Green (2010). Simply, a large amount of memory is allocated for the grid, and the capacity of each cell is determined by dividing the total grid capacity by the number of cells required. This allows the structure to deform and providing the concentration of atoms does not become extreme, no new memory will need to be allocated. To populate the grid, a second array is used to keep track of the number of atoms in each cell. To insert an atom, the value in the tracker array is atomically incremented. This value is then used to calculate the exact memory location within the grid the atom will be stored at. The atom is then placed within the grid. This allows the grid to be constructed with a single pass of the data.

Within *Protein Trajectory Viewer* different amounts of memory are allocated for the fixed grid, depending on the number of atoms present within the loaded structure. If there are less than 1000 atoms within the structure, a small pool of 3.75 MB is used. The pool is increased to 37.5 MB if the number of atoms is over 1000, and 375 MB if there are over 10,000 atoms in the structure. The selected values were found through experimentation. They allow the majority of trajectories to be rendered without having to enlarge the memory pool whilst rendering. The memory efficient grid described in Section 3.4.2 was also configured to allow the number of cells within the grid to increase by a factor of five before having to allocate more memory.

3.4.2. Compact Grid

A more memory efficient grid solution was presented by Lagae and Dutré (2008). The compact grid, shown in Figure 5, is a data structure consisting of two arrays. The first array acts as a lookup table for the cell start indices within the second array. To access any individual cell of the grid, first the cell start location has to be read from the lookup array and then the data array accessed to retrieve the information. This data representation is more memory efficient and less expensive to enlarge if required, however during construction and use the grid is less operation efficient. Green (2010) suggested a method for creating a tight compact grid on the GPU, again utilising atomics. This compact grid algorithm requires two passes of the data; the first pass counts how many atoms are in each cell, the second pass uses this data to construct the grid. Compact grids can also be created by sorting the data and then finding the points in the sorted array where the cells change. However only loose grids can be constructed by using sorting and since *Protein Trajectory Viewer* requires a tight grid, owing to the shadow calculations, sorting based algorithms are not investigated further.

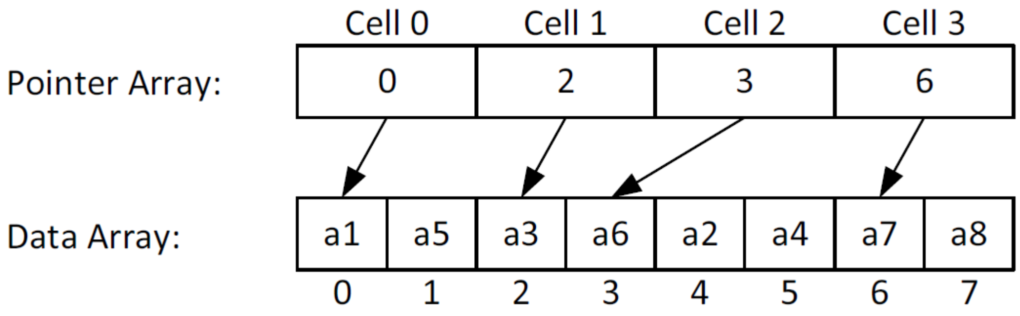

Figure 5: A representation of a compact grid containing eight elements (a1...a8). Note how the grid is closely packed, leaving no wasted space. Also the image shows the two step lookup method, demonstrating the potential inefficiency of the compact grid.

The four step construction algorithm of the grid was suggested by Green (2010). The steps are:

1. Calculate how many atoms are in each of the cells and store this value in a capacity array.
2. Perform an exclusive cumulative sum on the capacity array (Harris et al., 2007).
3. Use the array resulting from Step 2 as a lookup table for the start points of each cell.
4. Copy atom indices into the grid.

PDB Code	Number of Atoms
1CRN	327
1OMP	5737
5E0T	6,040
3E76	54,464
3JCU	75,994
1HTQ	97,872
FLAG	316,404

Table 1: A table showing the number of atoms that make up each of the tested proteins

To calculate how many atoms are in each cell in parallel, each atom is assigned one thread on the GPU. Each thread then calculates which cells the atom inhabits and atomically increments the count of the number of atoms that are resident in that specific cell. The count prior to incrementing is then stored for each atom, to allow the grid to be populated without any further atomic operations.

4. Results

Testing was carried out to determine the performance of the rendering and grid construction algorithms, and then the application as a whole is discussed. Firstly, the performance whilst rendering ray cast shadows is tested and the optimal settings for the regular grid found. Then the quality and performance of the ambient occlusion is analysed and discussed. We then assess the runtime of the pre-frame computation and discuss the performance of the application as a whole. Testing was performed on a desktop computer with an Intel i7 processor, 16 GB of RAM and an Nvidia GTX 980 with 4GB of VRAM.

Table 1 shows the number of atoms each of the proteins used in testing comprises of. All of the proteins, with the exception of FLAG and 1OMP, were acquired from the Protein Data Bank (Berman et al., 2000). 1HTQ contains ten models of glutamine synthetase from Mycobacterium Tuberculosis[1], which is treated as a trajectory during testing. 1OMP is a trajectory of a Maltodextrin binding protein (Sharff et al., 1992) and FLAG is a trajectory of a flagellar filament of Salmonella Typhimurium (Kitao et al., 2016). The other PDB files used within testing are static structures.

[1]Found at http://www.rcsb.org/pdb/explore.do?structureId=1htq

4.1. Graphical Effects

In this section we examine the performance and quality of the discussed graphical effects. All frame rate tests were performed at a resolution of 1920 x 1080, with proteins positioned to fill the window - see Figure 12 (B).

4.1.1. Shadows

To assess the effectiveness and performance of the shadow casting algorithm, the frame rate achieved whilst rendering shadows was measured using different cell sizes to determine which cell size would yield the best performance.

To measure the performance of the shadow algorithm, we rendered each of the test proteins and recorded the average frame rate achieved. This test was repeated with five different grid cell sizes. The formula nR_C is used to calculate the cell size, where R_C is the van der Waals radius of a carbon atom and n is an integer multiplier, greater than or equal to two. Using this formula with n set to a value of two or greater ensures that the majority of atoms are resident in a maximum of eight grid cells, important for the performance of the grid reconstruction algorithms used when a trajectory is being rendered. The results from this test can be seen in Figure 6. The test was performed with a compact grid owing to the excessive memory consumption but comparable performance of the fixed grid (See Section 4.2).

Figure 6 shows that for all of the tested proteins, the smallest grid cell size produces the highest frame rate. This is a result of the finer grid resulting in fewer intersection tests when the rays are cast within the lighting fragment shader. A grid cell size of $n = 2$ will be used for the remainder of testing because of this. Figure 6 also shows that the shadow casting algorithm comfortably achieves the target of 24 FPS for all of the structures tested.

We then generated a scene with a collection of atoms casting a shadow onto a flat structure to compare the results from our approach with RTFact within BallView (Marsalek et al., 2010). BallView was selected as a comparable application because it supports true ray cast shadows, as opposed to relying shadow mapping.

Figure 7 (A) highlights the effectiveness of the ray casting algorithm. The shadows indicate that two spheres are hidden behind those visible to the camera. Comparing the renders, the shadows generated in BallView (Figure 7 (B)) suffer from slightly jagged edges in places, and the hole in the centre of shadow is not very clear. Figure 7(A) shows the soft shadows included within *Protein Trajectory Viewer* work as designed, giving a smoother transition from dark to light than

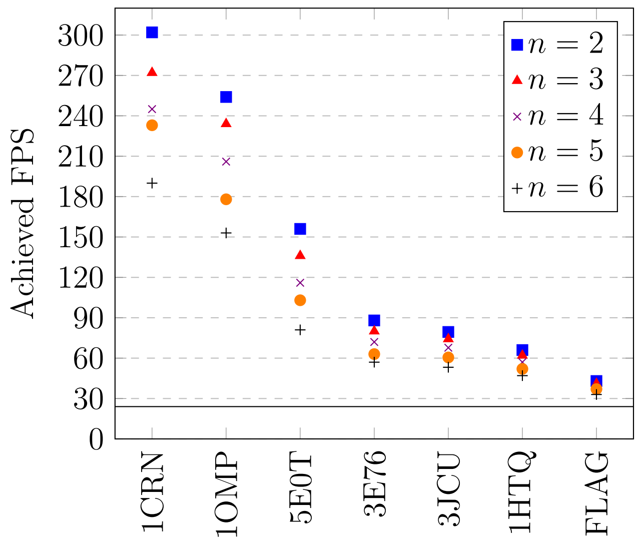

Figure 6: A graph showing the achieved frame rate when rendering seven proteins with ray cast shadows. Each of the proteins were positioned to fill the rendering window.

either of the other approaches. The same effect could be generated using the ray tracing approach with an area light source but this would be computationally expensive.

Protein Trajectory Viewer achieved 844.4 frames per second whilst rendering the displayed image, BallView managed 10.5 FPS. Within this test the images were generated at a resolution of approximately 1000 x 1000 pixels. The achieved frame rates show that the render produced by *Protein Trajectory Viewer* is considerably less costly than that generated by BallView.

We have shown that the shadows generated in *Protein Trajectory Viewer* are less costly than the real-time ray traced shadows in BallView and are better quality.

4.1.2. Ambient Occlusion

The quality of the analytical ambient occlusion technique implemented within *Protein Trajectory Viewer* is highly dependant on the cut off distance used to find occluders. The larger the cut-off distance, the better the image should look because more distant geometry will be included within the calculation.

The graph in Figure 8 shows how increasing the cut off distance effects the frame rate. As expected, the graph shows how testing more atoms for occlusion causes the frame rate to drop.

Figure 8 also shows that for all tested proteins, a cut-off of up to 10Å is achievable at an interactive frame rate on our hardware. It is unlikely that interactivity would be maintained whilst rendering a dynamic scene. This is investigated further in Section 4.2.

To assess the quality of the ambient occlusion technique, we compared the generated image

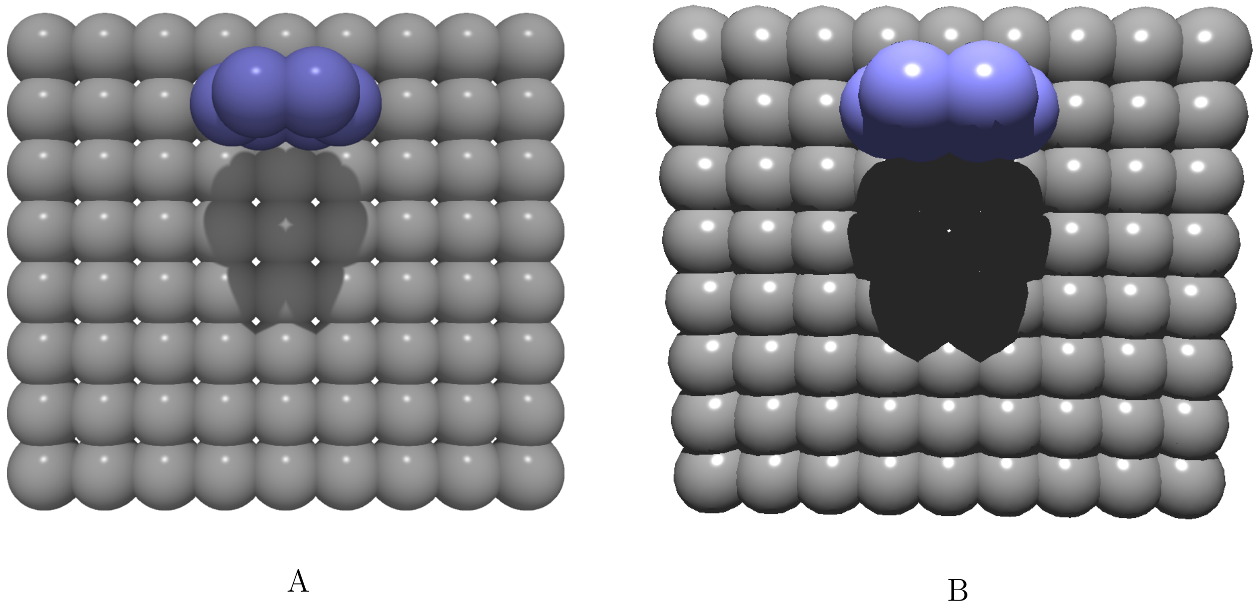

Figure 7: Figure shows the ray cast shadows generated by *Protein Trajectory Viewer* (A) and RTFact within BallView (C). The light was placed approximately above the camera within both of the scenes.

with a ground truth, generated using VMD and the Tachyon ray tracer (Stone, 1998). We also compare the generated render with the object space ambient occlusion method included in MegaMol (Grottel et al., 2015).

Figure 9 shows that the larger the cut-off applied, the more similar to the presented ground truth the generated image is, with image C providing the closest approximation of the three *Protein Trajectory Viewer* renders to the ray traced image D. When comparing the very deepest points of the images, the crevice in the centre of the image for example, this is effectively highlighted in images B and C, with some darkening of the area visible within A.

Comparing *Protein Trajectory Viewer* with MegaMol, shows that a large cut-off distance is needed to produce a similar ambient occlusion effect. However, when a larger cut off distance is used, the gradient of the occlusion applied by *Protein Trajectory Viewer* is smoother, with the outward facing parts of the atoms showing no occlusion. Within MegaMol, some occlusion appears to be present on the outward face of some of the atoms. However, the approach used by MegaMol is far less computationally expensive, achieving in excess of two thousand frames per second on our test computer, compared to one hundred and fourteen frames per second achieved when rendering image C.

From Figure 9 we can determine that the algorithm used demonstrates a reasonable approximation of per-pixel ambient occlusion, however the quality of the approximation is highly dependant

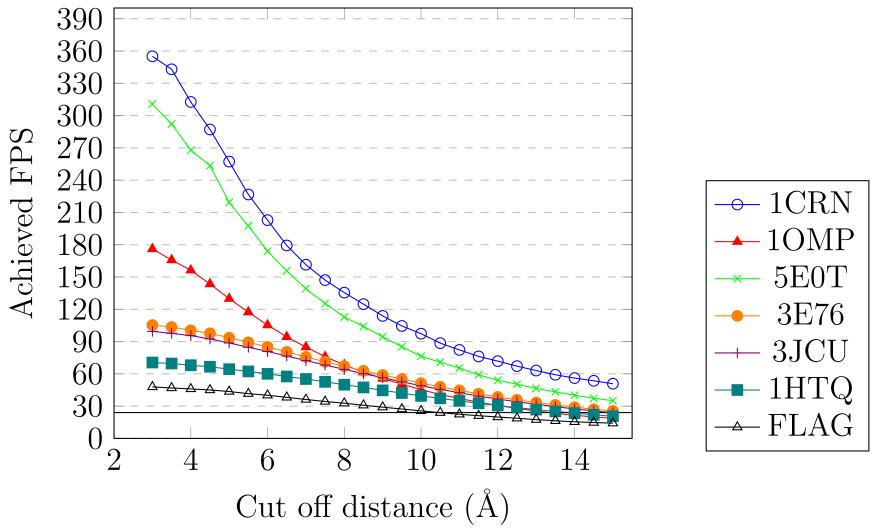

Figure 8: A graph showing the effect increasing the cut off distance used when searching for occluders has on the frame rate.

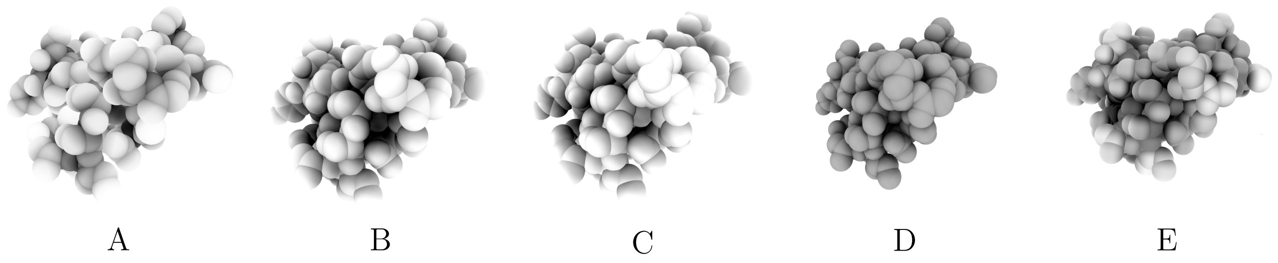

Figure 9: A) Protein Trajectory Viewer cut-off $2R_c$ (3.4Å), B) Protein Trajectory Viewer cut-off $4R_c$ (6.8Å), C) Protein Trajectory Viewer cut-off $6R_c$ (10.2Å), D) Tachyon within VMD, E) Megamol. The Ambient Occlusion volume size used within MegaMol was $10 \times 10 \times 9$.

on the cut-off distance used. As a result of this, we leave the occlusion cut-off distance as a variable to be controlled by the user, which can be adjusted to balance quality with performance.

4.2. Grid Performance

To render a trajectory, the regular grid used during the shadow computation and the lists of occluders used by the ambient occlusion algorithm will need to be refreshed. In this section the performance of these algorithms is discussed. Figure 10 shows a side by side comparison of the fixed and compact grid construction and occluder search run times. The cut off distances selected were chosen for the best quality ambient occlusion without having to increase the number of cells searched, when a cell size of $2R_c$ (3.4Å), the optimal cell size for the shadow casting, is used. A cut off of $2R_c$ (3.4Å) has to search at most one neighbouring cell in each direction, $4R_c$ (6.8Å) two cells and $6R_c$ (10.2Å) three cells.

The graph shows how the time taken to construct the regular grid and the time taken to use it

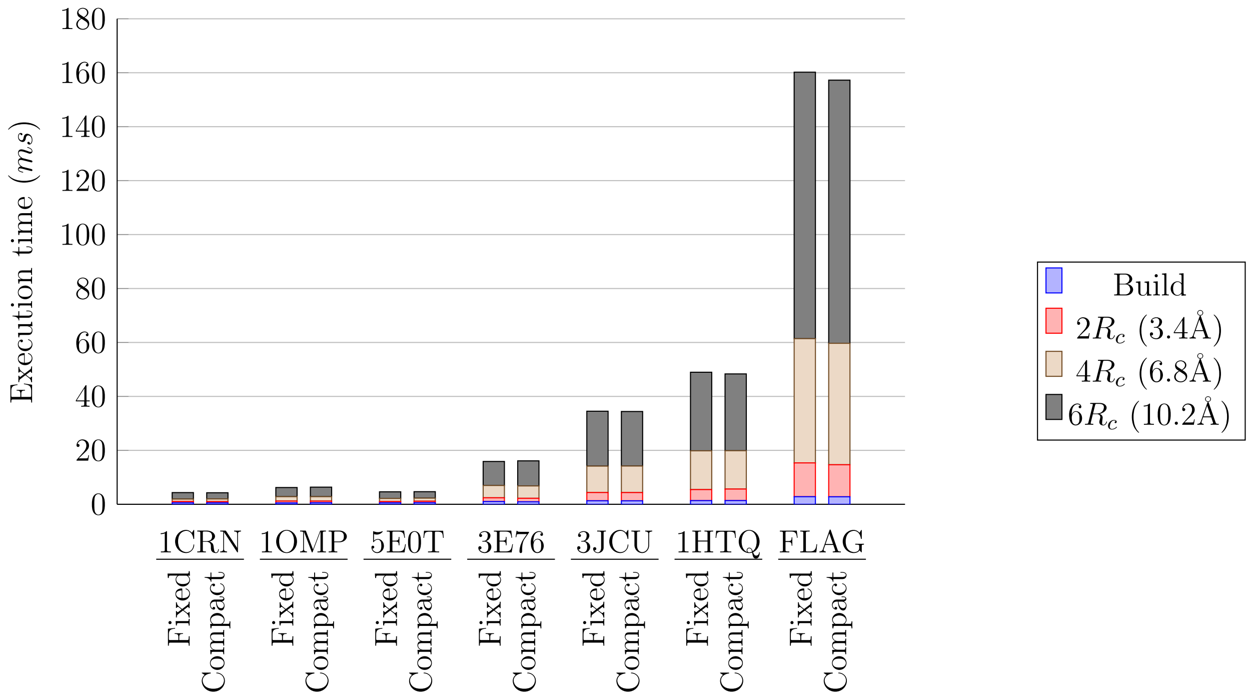

Figure 10: Graph showing the grid construction and occluder search times for both grids in milliseconds when the grid cell size is $n = 2$. Bars are cumulative.

to search for occluding atoms is apportioned. It is clear from the graph that the grid build stage makes up a smaller proportion of the total time than the occluder searching stage. The fixed grid completes the build stage more quickly than the compact grid; a result of only requiring a single pass of the data, compared to the two passes the compact grid requires. Despite this, the total execution time for both grids is very similar, typically with less than 5% of the total execution time separating them. The compact grid makes up the time lost during the build stage within the occluder search stage. The cause of this was found to largely be a result of more efficient memory access patterns within the occluder search kernel whilst using the compact grid. The overall performance of both grids is similar during this pre-rendering stage of the display loop.

Although the overall performance of the grids has been found to be comparable, the memory consumption of the fixed grid is much greater than that of the compact grid. The memory consumption of *Protein Trajectory Viewer* was measured using GPU-Z[2] whilst using both grid types as an acceleration structure. The results can be seen in Figure 11.

Figure 11 shows clearly that the fixed grid uses considerably more memory than the compact grid, for all of the proteins. This is as expected, as stated in Section 3.4.1, a fixed amount of

[2]https://www.techpowerup.com/gpuz/

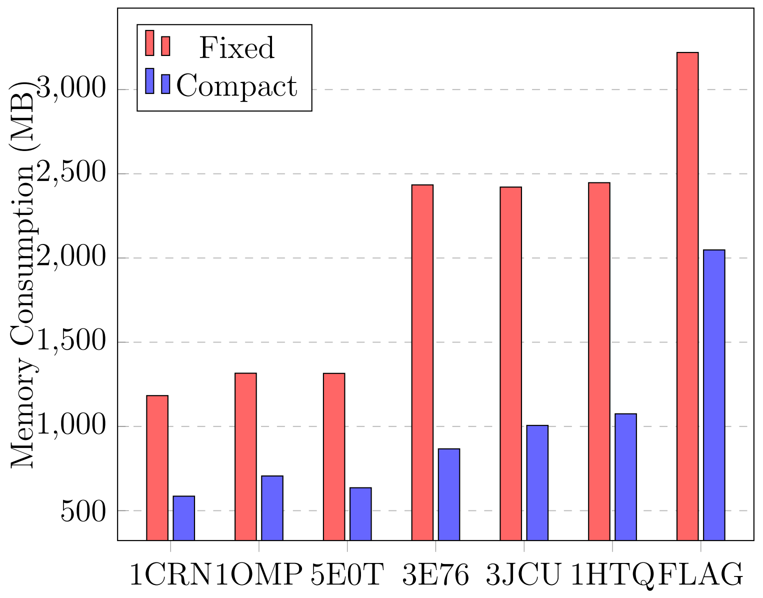

Figure 11: Memory consumption of each of the grids for each of the proteins tested. A cell size of $n = 2$ is used during testing. Memory is allocated to allow for protein expansion without having to reallocate memory during execution.

memory is allocated to the grid, and then the maximum capacity of each cell is dependent on the total number of cells in the grid. To allow large, distributed structures to be rendered correctly, the fixed grid memory pool has to be large. This is reflected in the overall graphical memory consumption of the application when using the fixed grid. If the required maximum capacity of the largest cell in the grid is known, the grid can be tuned to this, reducing the memory overhead, however this is impractical to compute whilst rendering a trajectory. This makes the compact grid the preferred structure for our use with our rendering algorithms.

4.3. Application Testing

In this section the performance of the complete application is tested. The rendering performance is measured by recording the number of frames rendered per second whilst rebuilding the regular grid every time a frame is rendered. This represents an extreme rendering case because it is uncommon to store enough trajectory frames to view the smooth transitions between one pose and the next, especially on very large structures, owing to current storage limitations. Because of this each frame of trajectory shows a marked different pose in its neighbouring frames. Consequently, trajectories are often played back more slowly, giving the user time to observe each frame before it advances. The slower the trajectory playback is advanced, the greater the average frame rate achieved will be, because the grid only has to be reconstructed when the trajectory frame is changed.

Figure 13 shows the frame rate achieved when rendering each of the seven proteins whilst reconstructing the grid and rebuilding each of the occluder lists every frame, with an ambient occlusion cut off distance of $4R_c$ (6.4Å). The test was performed at a moderate zoom level that allowed the entire protein to fit in the window - pictured in Figure 12 (A).

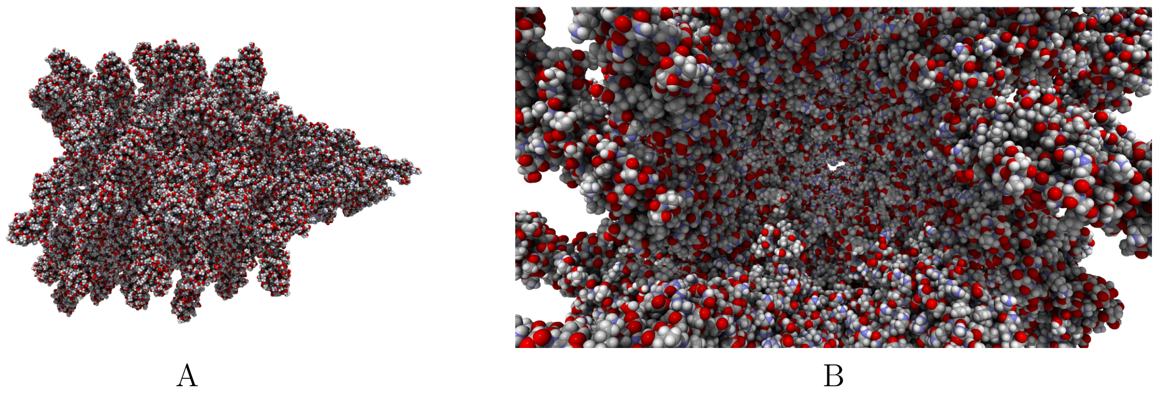

Figure 12: A - An image showing the zoom level used during the first part of the trajectory rendering and grid rebuild stress test. The larger the molecular structure rendered, the further back the camera was positioned, to keep the entire structure in view. B - An image showing the zoom level used during the second part of the trajectory rendering test. The zoom level was set so the protein filled the screen. The protein was set to spin on its y-axis during all rendering tests.

Figure 13 shows that for all the proteins except the two largest, 1HTQ and FLAG, interactive frame rates are maintained during the test. These results suggest that our algorithms will support interactive trajectory rendering for any structure comprising of at least seventy five thousand atoms, regardless of playback speed selected. A smaller ambient occlusion cut off distance could be used to achieve interactivity with larger proteins.

Comparing the fixed and compact grid's performance reveals that for the larger structures the compact grid outperforms the fixed grid. This concurs with the earlier findings - the compact grid is marginally more efficient in use, on the GPU, than the fixed grid. There is more variation between the two grids when rendering the smaller proteins. The fixed grid outperforms the compact grid whilst rendering 1CRN and 5E0T, whilst the compact grid produces a better frame rate when 1OMP and the larger proteins are being rendered. The cause of this is a result of the nature of the structure being rendered. 1OMP has a smaller bounding box relative to the number of atoms in the structure than 1CRN and 5E0T. This results in more adjacent atoms having to be tested per atom, both when building the occluder lists and performing the shadow ray tracing within OpenGL, increasing the number of accesses to the grid. This would suggest that the compact grid performs better when the grid cell memory accesses are concentrated into fewer cells.

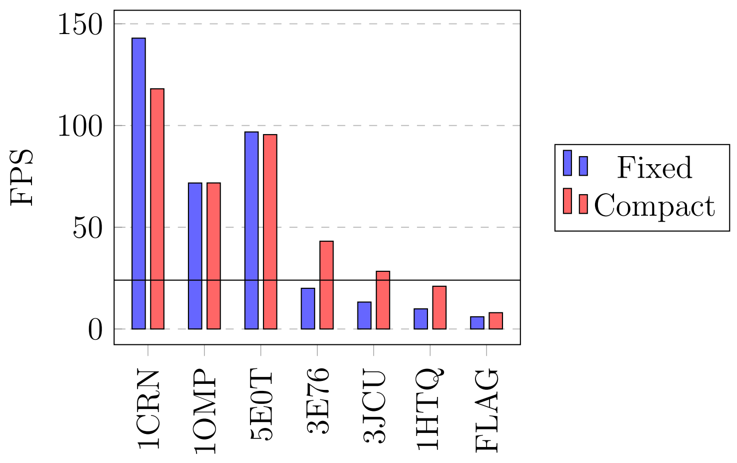

Figure 13: Frame rates whilst rendering each of the proteins with all visual effects enabled and performing grid reconstruction every frame. A cell size of $n = 2$ was used.

As mentioned earlier, the regular grid only has to be updated when the trajectory advances. Often this will be at a slower rate than the scene is refreshed. Using this information, the amount of computation that has to be performed every frame is reduced, resulting in an increase in the average frame rate, and enabling lower power video cards to render trajectories.

Whilst rendering the Flagella trajectory in this fashion, our application achieves an average frame rate of 87.9 frames per second at moderate zoom level and 32.17 frames per second whilst zoomed in when utilising the compact grid and an Ambient Occlusion cut off distance of 6.8Å. The trajectory was advanced once every two seconds. The grid reconstruction is noticeable as a slight pause in window interactivity occurs when zoomed in.

5. Conclusion

In conclusion, we have demonstrated a per pixel ray casting shadow algorithm that highlights hidden parts of the structure and has proven to be efficient enough to run in real time, at an interactive refresh rate. The ray cast shadows have been enhanced with a soft shadow effect to give a more natural shaded impression. Per pixel analytical ambient occlusion has also been demonstrated to be achievable in real time. We have also implemented a per-pixel ambient occlusion algorithm that enhances the surface features of molecular structures.

By utilising the GPU, it has been shown that the regular grid used by the per-pixel lighting can be reconstructed in very little time, in less than one millisecond in some cases. This allows the effects demonstrated in this paper to be applied to dynamic scenes.

Within the software *Protein Trajectory Viewer* the discussed grid construction and protein rendering techniques were implemented and tested. It was found that the methods are efficient enough that molecular trajectories of proteins comprising over three hundred thousand atoms can be rendered at forty frames per second on modern graphics hardware with per-pixel lighting.

Furthermore, it was shown that the methods discussed are fast enough to allow the scene to change at the refresh rate of the rendering window. This is important, because as storage mediums evolve and large amounts of storage become readily available, the ability to store smaller trajectory time steps will become reality. The smaller time steps will allow a smoother visualisation of a protein's trajectory to be shown. Our methods are fast enough to cope with this scenario.

It was found that despite the theoretical O(1) access time of the fixed grid, it did not always provide the best performance in practice. It was found that owing to its more efficient use of memory, the compact grid can provide better overall performance on the GPU when the grid is used intensively. Within our application, it was demonstrated that the smaller the cell size used, the better the application performed. It should be noted that if the grid was used less intensively than it is in our application, the fixed grid may be the better option, however it will always be less efficient in regards to memory use.

6. Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/M508020/1].

7. Bibliography

Amanatides, J. and Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10.

Annen, T., Dong, Z., Mertens, T., Bekaert, P., Seidel, H. P., and Kautz, J. (2008). Real-time, all-frequency shadows in dynamic scenes. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 34:1–34:8. ACM.

Appel, A. (1968). Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA. ACM.

Barbieri, D., Cardellini, V., and Filippone, S. (2013). Fast Uniform Grid Construction on GPGPUs Using Atomic Operations. In *PARCO*, pages 295–304.

Bavoil, L. and Sainz, M. (2009). Multi-layer Dual-resolution Screen-space Ambient Occlusion. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, pages 45:1–45:1, New York, NY, USA. ACM.

Bavoil, L., Sainz, M., and Dimitrov, R. (2008). Image-space Horizon-based Ambient Occlusion. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH '08, pages 22:1–22:1, New York, NY, USA. ACM.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242.

Bunnell, M. (2005). Dynamic ambient occlusion and indirect lighting. In *GPU gems 2*, volume 2, pages 223–233. Addison-Wesley Professional.

Bunnell, M. and Pellacini, F. (2004). Shadow map antialiasing. In Fernando, R., editor, *GPU Gems*, chapter 11. Pearson Higher Education.

Crassin, C., Neyret, F., Sainz, M., Green, S., and Eisemann, E. (2011). Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library.

Crow, F. C. (1977). Shadow Algorithms for Computer Graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, pages 242–248, New York, NY, USA. ACM.

Dimitrov, R. (2007). Cascaded shadow maps. *Developer Documentation, NVIDIA Corp.*

Donnelly, W. and Lauritzen, A. (2006). Variance Shadow Maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 161–165, New York, NY, USA. ACM.

Favera, E. C. D. and Celes, W. (2012). Ambient Occlusion Using Cone Tracing with Scene Voxelization. In *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 142–149.

Filion, D. and McNaughton, R. (2008). Effects & techniques. In *ACM SIGGRAPH 2008 Games*, pages 133–164. ACM.

Green, S. (2010). Particle Simulation using CUDA. *NVIDIA Whitepaper*.

Grottel, S., Krone, M., Mller, C., Reina, G., and Ertl, T. (2015). MegaMol; A Prototyping Framework for Particle-Based Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 21(2):201–214.

Grottel, S., Krone, M., Scharnowski, K., and Ertl, T. (2012). Object-space ambient occlusion for molecular dynamics. In *2012 IEEE Pacific Visualization Symposium*, pages 209–216.

Gumhold, S. (2003). Splatting illuminated ellipsoids with depth correction. In *VMV*, pages 245–252.

Hanwell, M. D., Curtis, D. E., Lonie, D. C., Vandermeersch, T., Zurek, E., and Hutchison, G. R. (2012). Avogadro: an advanced semantic chemical editor, visualization, and analysis platform. *Journal of Cheminformatics*, 4(1):17.

Harris, M., Sengupta, S., and Owens, J. D. (2007). Parallel prefix sum (scan) with CUDA. *GPU gems*, 3(39):851–876.

Hoberock, J. and Yuntao, J. (2007). High-quality ambient occlusion. In *GPU gems 3*, pages 257–273. Addison-Wesley Professional.

Humphrey, W., Dalke, A., and Schulten, K. (1996). VMD: visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, 27–28.

Kajalin, V. (2009). Screen space ambient occlusion. *ShaderX7: Advanced Rendering Techniques*, pages 413–424.

Kitao, A., Harada, R., Nishihara, Y., Tran, D. P., Simos, T. E., Kalogiratou, Z., and Monovasilis, T. (2016). Parallel cascade selection molecular dynamics for efficient conformational sampling and free energy calculation of proteins. *AIP Conference Proceedings*, 1790(1):020013.

Knoll, A., Wald, I., Navrtil, P. A., Papka, M. E., and Gaither, K. P. (2013). Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. pages 1–8. ACM Press.

Kolivand, H. and Sunar, M. S. (2013). Survey of Shadow Volume Algorithms in Computer Graphics. *IETE Technical Review*, 30(1):38–46.

Kontkanen, J. and Laine, S. (2005). Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48. ACM.

Kozlikova, B., Sebestova, E., Sustr, V., Brezovsky, J., Strnad, O., Daniel, L., Bednar, D., Pavelka, A., Manak, M., Bezdeka, M., Benes, P., Kotry, M., Gora, A., Damborsky, J., and Sochor, J. (2014). CAVER Analyst 1.0: graphic tool for interactive visualization and analysis of tunnels and channels in protein structures. *Bioinformatics*, 30(18):2684–2685.

Krieger, E. and Vriend, G. (2014). YASARA Viewmolecular graphics for all devicesfrom smartphones to workstations. *Bioinformatics*, 30(20):2981–2982.

Krone, M., Bidmon, K., and Ertl, T. (2009). Interactive Visualization of Molecular Surface Dynamics. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1391–1398.

Lagae, A. and Dutré, P. (2008). Compact, Fast and Robust Grids for Ray Tracing. *Computer Graphics Forum*, 27(4):1235–1244.

Lauritzen, A. and Mccool, M. (2007). Summed-area variance shadow maps. *GPU Gems*, pages 157–182.

Lindow, N., Baum, D., Prohaska, S., and Hege, H.-C. (2010). Accelerated Visualization of Dynamic Molecular Surfaces. *Computer Graphics Forum*, 29(3):943–952.

Loos, B. J. and Sloan, P.-P. (2010). Volumetric obscurance. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 151–156. ACM.

Marsalek, L., Dehof, A. K., Georgiev, I., Lenhof, H. P., Slusallek, P., and Hildebrandt, A. (2010). Real-Time Ray Tracing of Complex Molecular Scenes. In *2010 14th International Conference Information Visualisation*, pages 239–245.

Martin, T. and Tan, T. S. (2004). Anti-aliasing and Continuity with Trapezoidal Shadow Maps. *Rendering techniques*, 2004:15th.

Mittring, M. (2007). Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, pages 97–121. ACM.

Moll, A., Hildebrandt, A., Lenhof, H.-P., and Kohlbacher, O. (2006). BALLView: a tool for research and education in molecular modeling. *Bioinformatics*, 22(3):365–366.

Papaioannou, G., Menexi, M. L., and Papadopoulos, C. (2010). Real-time volume-based ambient occlusion. *IEEE transactions on visualization and computer graphics*, 16(5):752–762.

Parker, S., Shirley, P., and Smits, B. (1998). Single sample soft shadows. Technical report, Technical Report UUCS-98-019, Computer Science Department, University of Utah.

Pettersen, E. F., Goddard, T. D., Huang, C. C., Couch, G. S., Greenblatt, D. M., Meng, E. C., and Ferrin, T. E. (2004). UCSF Chimera–a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612.

Pharr, M. and Green, S. (2004). Ambient occlusion. In Fernando, R., editor, *GPU Gems*, chapter 17, pages 279–292. Pearson Higher Education.

Reeves, W. T., Salesin, D. H., and Cook, R. L. (1987). Rendering antialiased shadows with depth maps. In *ACM Siggraph Computer Graphics*, volume 21, pages 283–291. ACM.

Reinbothe, C. K., Boubekeur, T., and Alexa, M. (2009). Hybrid Ambient Occlusion. In *Eurographics (Areas Papers)*, pages 51–57.

Sattler, M., Sarlette, R., Zachmann, G., and Klein, R. (2004). Hardware-accelerated ambient occlusion computation. *Proceedings of Vision, Modeling, and Visualization 2004*, pages 331–338.

Sayle, R. A. and Milner-White, E. J. (1995). RASMOL: biomolecular graphics for all. *Trends in Biochemical Sciences*, 20(9):374.

Schrödinger, LLC (2015). The PyMOL molecular graphics system, version 1.8.

Shanmugam, P. and Arikan, O. (2007). Hardware Accelerated Ambient Occlusion Techniques on GPUs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 73–80, New York, NY, USA. ACM.

Sharff, A. J., Rodseth, L. E., Spurlino, J. C., and Quiocho, F. A. (1992). Crystallographic evidence of a large ligand-induced hinge-twist motion between the two domains of the maltodextrin binding protein involved in active transport and chemotaxis. *Biochemistry*, 31(44):10657–10663.

Sigg, C., Weyrich, T., Botsch, M., and Gross, M. H. (2006). GPU-based ray-casting of quadratic surfaces. In *SPBG*, pages 59–65.

Skånberg, R., Vázquez, P. P., Guallar, V., and Ropinski, T. (2016). Real-Time Molecular Visualization Supporting Diffuse Interreflections and Ambient Occlusion. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):718–727.

Staib, J., Grottel, S., and Gumhold, S. (2015). Visualization of Particle-based Data with Transparency and Ambient Occlusion. *Computer Graphics Forum*, 34(3):151–160.

Stamminger, M. and Drettakis, G. (2002). Perspective Shadow Maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 557–562, New York, NY, USA. ACM.

Stone, J. (1998). *An Efficient Library for Parallel Ray Tracing and Animation.* Master's thesis, Computer Science Department, University of Missouri-Rolla.

Stone, J. E., Sherman, W. R., and Schulten, K. (2016). Immersive Molecular Visualization with Omnidirectional Stereoscopic Ray Tracing and Remote Rendering. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1048–1057.

Tarini, M., Cignoni, P., and Montani, C. (2006). Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE transactions on visualization and computer graphics*, 12(5):1237–1244.

Wanger, L. R., Ferwerda, J. A., and Greenberg, D. P. (1992). Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications 12(3)*, pages 44–58.

Wimmer, M., Scherzer, D., and Purgathofer, W. (2004). Light space perspective shadow maps. *Rendering Techniques*, 2004:15th.

Woo, A. and Poulin, P. (2012). *Shadow Algorithms Data Miner.* CRC Press.

Zhang, F., Sun, H., Xu, L., and Lun, L. K. (2006). Parallel-split Shadow Maps for Large-scale Virtual Environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*, VRCIA '06, pages 311–318, New York, NY, USA. ACM.