RasMol: A Program for Fast, Realistic Rendering of Molecular Structures with Shadows

Roger Sayle and Andrew Bissell

November 11, 1993

Abstract

This paper details the development of an interactive program for the visualisation of proteins and nucleic acids. The paper first reviews current techniques for displaying the three dimensional structures of molecules and methods for the determination of their cast shadows.

The paper then goes on to describe an efficient hybrid ray tracing algorithm for molecular graphics based upon uniform spatial subdivision. Results are then presented for implementations of this algorithm on both Transputer based multiprocessors and UNIX workstations under the X-Window System. Both versions are believed to have the fastest rendering times for shadowed union-of-spheres surfaces published to date. Finally details of work currently in progress and future directions are given.

Introduction

Computer graphics has long been a useful tool in the understanding of the three dimensional structure of molecules. The earliest drawings were no more than 'wireframes', with straight line segments representing the bonds between atoms. More recently the study of molecular surfaces has become important in order to understand the interactions between molecules, such as the effects of a drug or hormone on a receptor site. Modern protein biochemists are concerned with secondary and higher order structure where the areas of interest are the shape, orientation and accessibility of the active sites of proteins, rather than the individual bonds or atoms themselves. Visualisation of these features calls for more complex molecular surface representations with colouring and other effects playing a major role [12].

By far the most common raster representation of molecular structures is the space-filling model. This representation is based the plastic Corey-Pauling-Koltun (CPK) models used by chemists for representing small molecules. In this scheme each atom is depicted by a sphere with a radius equal to the atom's van der Waals radius, and a molecule is represented as the union of intersecting spheres. The contact between two molecules is shown by the contact between their space filling models because the preferred contact distance between two nonbonded atoms is the sum of their van der Waal radii.

As the demands placed on molecular graphics have grown, the algorithms used to render them have become more and more complex. To obtain a true impression of depth with modern raster displays, the displayed molecule must cast shadows and reflect highlights as one would expect of a real solid object, yet be rotated and deformed interactively. When the observer's position is coincident with the light source no shadows are seen. As the positions of the light source and the observer separate, shadows appear. It is not just the occlusion of hidden surfaces that create a perception of depth but also how shadows move across illuminated surfaces under rotation. Determining the shadows cast by a general surface is a very computationally expensive procedure, often requiring hours to generate a single frame. This is far from satisfying the requirement for real-time manipulation.

Most existing programs avoid this problem in a number of ways. The first and most obvious method is to ignore shadows altogether and render the molecule without them. The second is to assume that the observer is always coincident with the light source, implying that there are never any shadows visible to the viewer. This technique has the added advantage that the shading of a given atom consists of concentric circles of varying intensity, allowing the use of small lookup tables to avoid time spent calculating the shading on each atom. The last method is called *hither-and-yon* shading, which simply draws the molecules furthest from the viewer in a darker shade than those nearer, providing a slight feeling of depth.

Previous Work

Several algorithms have been proposed to calculate projected shadows, however, all of these methods greatly increase the amount of time required to render a molecule, especially when the number of atoms in the scene can be in the order of tens of thousands. These methods fall into two main categories; those that are based upon the use of shadow maps and those that perform ray tracing.

Shadow Maps

Shadow mapping is an extension of depth buffer algorithms to enable them to be rendered with shadowing [24]. These algorithm works by drawing two or more views of the scene. First the visible surfaces from the viewpoint of each of the light sources are determined using a depth buffer algorithm. Only the Z values and not the shading values need to be computed and stored. The scene is then rendered from the observer's viewpoint. At each visible pixel, a linear transformation is used to map the observed co-ordinate (x, y, z) into each light sources reference frame. This generates an co-ordinate (X, Y, Z) which is the position and depth of the pixel being drawn in the light's depth map. By comparing the appropriate entry in the depth map with the obtained depth, the pixel being tested may be determined to be visible or in shadow.

An good example of a shadow mapping implementation for molecular graphics is given by David Bacon's Raster3D program [1]. In his implementation, the depth buffer algorithm divides the screen up into a set of regular tiles, so that the objects visible in that tile may be stored as a sorted list. The advantage of using this division is that there are relatively few spheres associated with each tile.

Gwilliam and Max [9] also use a shadow mapping technique. Their method first generates resolution independent decompositions of the scene for the viewpoints of the observer and light sources and uses these decompositions to compute the final image. The decompositions consist of a collection of "trapezoids" with straight vertical sides and possibly curved top and bottom edges. Initially each sphere is approximated by two tapezoids which are then truncated and subdivided by intersections with other spheres. Although the Gwilliam-Max algorithm contains several restrictions on the environment and uses several approximations, it produces respectable images with cast shadows. The major advantage of the technique is that it is almost independent of the number of pixels in the final image.

The fastest implementation of a rendering program to shadow space-filling molecules previously described is Huang *et al.*'s Conic [10]. It also uses an efficient shadow mapping approach which does not constrain the observer and light sources to be positioned infinitely distant. This is done by using a scan conversion routine based on conic sections.

Ray Tracing

The use of ray tracing for generating photorealistic images was first introduced by Whitted [23]. The principle is that an observer views an object by means of light which travels in a straight line from its surface. Simple extension of this principle allows the 'global illumination' model that accounts for reflection of one object in another, refraction, transparency and shadow effects. Ray tracing algorithms find shadows by tracing a ray from each visible surface point towards the light source and testing whether the ray encounters any opaque surface before it gets there. The major bottleneck in the algorithm is the object intersection tests. In the naive algorithm every object in the scene has to be tested against each ray in order to determine that a visible pixel is not in shadow. Whitted determined that a brute force ray tracing routine spends 75%-95% of its effort determining intersections. An introduction to the principles of ray tracing is given in Glassner's book [7] and its application to molecular graphics is discussed by Palmer *et al.* [14].

The speeding up of ray tracing for realistic image synthesis has been an important research issue since its inception and to date a significant number of proposals have been put forward to improve upon the original naive algorithm. The three most popular acceleration techniques are based upon bounding volumes, regular spatial decomposition and adaptive spatial decomposition.

The use of bounding volume hierarchies to speed up exhaustive ray tracing was first described by Rubin and Whited [18]. By enclosing a group of objects within a large bounding volume (also called an 'extent' or 'closure') it is possible to eliminate many objects from further consideration with a single intersection check. Only if a ray intersects a 'parent volume', do the objects within it need to be tested for intersection. A hierarchy is then formed by recursive application of this principle, and in this way large numbers of objects may very rapidly be rejected from consideration. The application of this method to molecular graphics has been presented by Jones [11].

Spatial decomposition methods establish coherence another way. The entire 3D object space is divided into a number of small regions and for each the set of objects that intersect the region are found. Usually space subdivision divides the complete object space up into non-overlapping axis aligned rectangular prisms or cuboids called "voxels". Ray tracing proceeds by finding the voxel which contains the origin of the ray and iteratively determining the next voxel along the ray's path. At each step, each object contained in the voxel is tested for intersection and the algorithm stops at the first voxel in which an object is hit or when the ray leaves the scene. The algorithm used to calculate the next voxel a ray propagates to is called the *voxel traversal algorithm*.

Adaptive or nonuniform spatial decomposition techniques are those which discretize space into regions of varying size in order to conform to features of the environment. This variation in size allows more subdivision to be performed in densely populated regions and, conversely, large voxels to cover sparse or completely empty regions. Glassner [6] described the first use of the *octree* data structure in ray tracing to describe the connectivity between voxels. Octrees recursively divide voxels into eight octants until the voxel is sparsely populated or some number of subdivisions has occurred.

Uniform or regular spatial subdivision was first introduced by Fujimoto *et al.* [5]. In this approach object space is divided up into a 3D grid or lattice of voxels, which require no explicit data structures to describe their connectivity. Because the partitioning is completely independent of the scene being described, large numbers of voxels may be left empty which is much less efficient on storage. The major advantage of regular subdivision is in the efficient voxel traversal algorithms that have been developed for them. These algorithms such as Fujimoto's original three dimensional digital difference analyser (3DDDA) are based on raster line drawing algorithms and are far more efficient than the recursive methods of traversing octrees.

The RasMol algorithm

The RasMol program uses a hybrid rendering algorithm to achieve high speed display of molecular surfaces. The algorithm conceptually works in two stages; the first calculates the visible surfaces of the atoms in the scene using a scanline based algorithm and the second stage determines which of the visible pixels are in shadow using ray tracing.

Scan Conversion

The scan conversion algorithm used by the RasMol program uses a scanline z-buffer algorithm similar to the one described by Porter [16]. The choice of a scanline algorithm over the more usual full depth buffer method is primarily to reduce the memory overhead of the program. The use of a scanline algorithm is also advantageous for the shadowing and parallel implementations of the program as described below. By treating each atom as a single sphere primitive, rather than decomposing it into a surface of polygons, far fewer transformations and rendering calculations are performed per frame.

Before each frame is rendered, a *y-bucket* is generated by determining the highest scan line intersected by each atom and placing it in a list of atoms that begin on that scan line. This allows an active list of spheres that intersect the current line to be maintained as the frame is rendered. The contents of the appropriate bucket entry are appended to the active list before drawing each scan line and each atom is removed from the list once the last line on which it appears has been processed.

On each scan line it remains to solve the following equation for each atom:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r_0^2$$

where (x_0, y_0, z_0) are the coordinates of the sphere center, r_0 is the radius of the sphere and y is the y ordinate of the current scan line.

The RasMol program performs this calculation efficiently by maintaining a lookup table of integer values $\sqrt{x^2 - y^2}$ where both x and y are small positive integers. These values are stored in a triangular matrix, since $x \geq y$ otherwise the result is undefined. The representation of this data structure in C is an array of pointers to arrays of integers. Using C syntax, the table entry `LookUp[`x`][`y`]` contains the required value. An active atom is rendered by first calculating *drad*, the projected radius of the sphere on the current scan plane. This is given by the value `LookUp[`rad`][`dy`]`, where *rad* is the integer radius of the sphere and *dy* is the absolute value of $y - y_0$ (notice that $dy \leq rad$ for active atoms). Hence this atom is visible on the current scanline *drad* pixels to either side of x_0. For each value *dx* less than *drad*, the depth of the appropriate pixel is found by adding `LookUp[`$drad$`][`dx`]` to z_0.

This method is clearly faster than those implementations that require the calculation of square roots at each pixel, such as ray tracing and Pearl's "cpk" program [15]. It also has several advantages over Porter's algorithm [16] which uses Bresenham's incremental circle generator [2] to calculate the depth of each pixel on a sphere's visible intersection with a scanline. Firstly, the use of a moderate sized lookup table is faster than the incremental calculation and secondly Porter's method requires several iterations of Bresenham's algorithm for some pixels when the displayed spheres have large radii. The size of the table is quite small requiring only $n(n+1)/2$ entries, where n is the largest radius in the scene. The current RasMol implementation allocates 8kbytes to the table, allowing spheres with up to 125 pixel radius to be drawn. This is a huge memory saving over template based approaches which precompute a large array of depth values for each sphere size in the scene [13, 20].

Shadowing Algorithm

The calculation of projected shadows in RasMol is based upon ray tracing, or to be completely accurate shadow tracing. Performance analysis of pure ray tracing programs reveals that tracing the shadow rays requires far less time than the primary or initial rays and it is this difference in complexity that motivated the RasMol algorithm. The principle is to use a fast image space scan conversion algorithm to determine the visible surfaces in the scene and then shadow trace the results. This relatively overlooked approach was first described by Weghorst, Hooper and Greenberg [22].

RasMol implements the interface between these two algorithms by using an *item buffer* to record the visible object in addition to the depth at each pixel. The scan conversion acts as a first pass, and all shading and lighting model calculations are deferred until the second pass. The advantages with this approach are that unnecessary computation is avoided, the method is independent of the number of light sources and 'surface acne' [7] may be avoided by knowing which object is visible. The disadvantage is that values calculated during scan conversion must either be stored at great expense (one value per pixel) or reevaluated during shading. Because the RasMol program is intended primarily for space filling representations, recalculating values for the sphere primitives has very little overhead. The use of a scanline instead of a full depth buffer in the rendering phase also reduces the memory required by the item buffer.

Given the depth of a pixel on the screen, the first step in determining whether that pixel is in shadow or not, involves calculating the world space co-ordinates of that point. This is done by multiplying the image space co-ordinate vector by the inverse of the current transformation matrix. The current transformation matrix is a square 4×4 matrix that transforms the world space co-ordinates of the atomic centres into the final screen co-ordinates and depth of the atom on the display. This matrix permits the rotation, scaling and translation of the model. When the shadowing option is enabled, the RasMol program calculates the inverse transformation matrix using standard numerical methods, but while the molecule is being manipulated the inverse matrix is maintained by applying the inverse of the current transformation to the inverse transformation matrix. If the shadowing option is disabled the inverse matrix does not have to be kept up to date. This enables the inverse matrix to be maintained at very low overhead and not degrade the rendering times for scenes without shadows.

RasMol uses the uniform space decomposition approach to detect whether the visible pixels are in shadow or not. The decision to use a regular over an adaptive subdivision technique was determined by the application area. For CPK representations of macromolecules, the world database consists of a very large number of densely packed objects, all of approximately the same size. Hence the principle advantage of adaptive octree based approaches, the reduced storage requirement, is less effective because the scene is uniformly populated. For ray tracing more general scenes, it is likely that octree approaches are more efficient. The actual voxel traversal method implemented is based upon the algorithm described by Cleary and Wyvill [3]. Their algorithm requires a maximum of eight integer operations (additions or comparisons) to determine the next voxel on the ray's path in the current implementation.

One major advantage of the use of an object space based method is that the data structures need only be constructed once, as a preprocessing step, and require no further modifications between frames. Hence, as a molecule is rotated about it's axis, the shadow casting algorithm need only calculate the new position of the light sources relative to the original reference frame. Image space based shadow casting, such as shadow maps, require the positions of each of the atoms to be calculated relative to the light sources, which requires far more calculations as the number of atoms is generally larger than the number of light sources. This is the potential speed advantage of ray tracing approaches over shadow mapping.

Parallel Implementation

The RasMol program program was originally developed for execution on distributed memory multiprocessors, such as Transputer arrays or networks of workstations. To this end, several potential parallel versions of the basic algorithm were implemented based on both object and image space parallelism [8].

The mode of parallelism used by the program is referred to as image space parallelism. This means that the scene description is duplicated on each processor and different processors concurrently work on separate parts of the image. This can provide almost linear speedup as processors are added since the processors only communicate in order to output results and possibly to balance the load between processors.

The many object space parallelism methods presented in the literature were considered unsuitable for the application due to the architectures of the machines on which the program was intended to be executed. Object space parallelism is particularly well suited for large arrays of small processors, where distributing the scene database amongst the processors at the expense of interprocessor communication is a reasonable tradeoff. However, all the considered MIMD architectures had sufficient memory to duplicate the entire scene database on each processor and therefore the primary goal was to minimize communication between processors to achieve maximum performance/speed-up.

One of the main issues in parallel algorithm design is the balance of the workload between processors as one heavily loaded processor may drastically reduce the speed up of the whole system. The most common approach to image space load balancing is to preallocate equal numbers of pixels amongst the available processors, often referred to as *static* task allocation. The problem with such a scheme, is that using ray tracing different pixels may have different complexity because the number of intersection calculations required is not known in advance. If this is the case, some processors will finish long before others and have to wait idly until the system is ready to proceed to the next frame. One way to avoid this problem is to use *dynamic* task allocation where new tasks are allocated to processors once they have completed their current work. This leads to a much better sharing of the available work between processors at the expense of increased communication between processors. Implementations of these load balancing strategies for ray tracing on Transputer arrays have been discussed by various researchers [17, 21, 25].

Due to the details of the scan conversion phase of the RasMol algorithm, the parallel implementations use a scan line as the 'granularity' or quantum of work that is performed by a processor. The performance figures gained from several implementations indicated that the best performance was achieved using a *scattered* static task allocation policy. For a parallel machine with N processors, scattered preallocation gives every N^{th} scan line to each processor. Because complex scan lines (i.e. those that take longer to calculate) tend to be clustered together in an image, scattered allocation more evenly distributes the workload than 'block' allocation. Block allocation evenly divides the screen horizontally into N equal-height multiples of scan lines. The advantage of less interprocessor communication than dynamically allocating tasks has greater benefits than the disadvantage of suboptimal load balancing.

The implementation of the scan conversion algorithm may also be tailored to exploit the use of scattered static task allocation. Typically, parallel algorithms which are based on dynamic task allocation cannot be guaranteed to be given consecutive lines to calculate. Hence parallel scanline methods suffer from an overhead of maintaining the active list even over lines that they do not process. Realising in advance that the processor will only ever calculate every N^{th} line enables the y-buckets to be tailored to maintain lists of the atoms that commence on or before each N^{th} line. During rendering, atoms are removed from the active list if they do not appear on the next line rendered by the processor. In this way spheres that are less than N pixels in diameter may not even appear in some processor's y-buckets.

Results

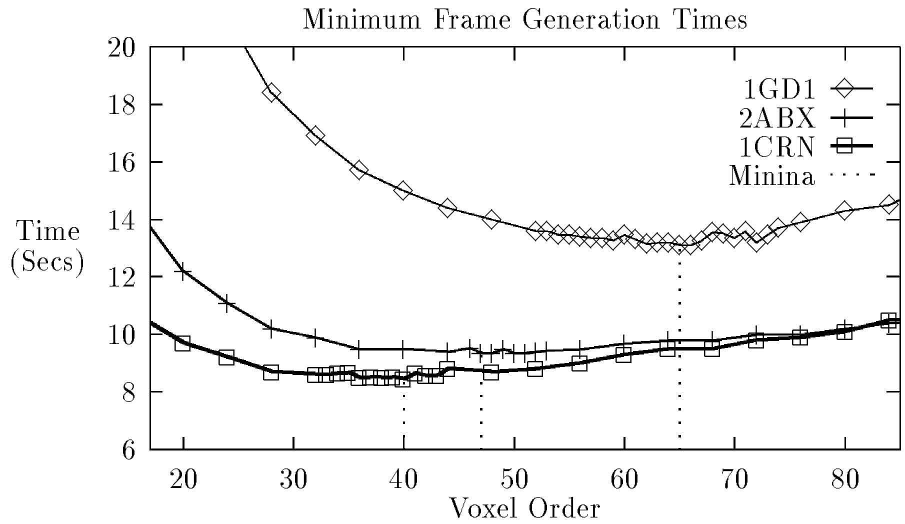
Uniprocessor Results

The key to RasMol's performance is the combination of an image space hidden surface algorithm and an object space accelerated shadow tracer. The shadow tracer uses a regular object space subdivision, allocating atoms to the appropriate voxels of a cubic lattice, to reduce the required number of ray-sphere intersections.

The order (number of voxels along each axis) of the voxel structure is critical to the performance of the shadow tracer. As voxel order increases the optimum performance will occur where the reducing cost of fewer ray-sphere intersections balances the increasing cost of stepping rays through more voxels.

To evaluate the optimum voxel order, measurements were taken for three proteins from the Brookhaven Protein Data Bank [4]: Crambin (1CRN), the smallest protein in the database with 327 atoms; Krait Venom Neurotoxin (2ABX), a typical protein of 1,118 atoms; and Glyceraldehyde 3 Phosphate Dehydrogenase (1GD1), one of the largest proteins in the database with 10,984 atoms.

RasMol was tested on a SUN SPARCStation 1+ in a 512×512 X11 window. Wall clock timings, taken at 4 unit intervals in voxel order, were averaged over 10 frames taken from equally spaced viewpoints on a circle around the z-axis. This approach implies that all coordinate transformation, database traversal, rendering and window update times are included. Thus the figures accurately reflect the performance that a user would see.



The results indicate that at low voxel orders performance is poor but improves rapidly with increasing voxel order. The curve becomes relatively flat over the mid-range (30 to 100) dropping to a minimum and then rising slowly. This range was sampled in unit voxel order steps, with run-time averaged over twenty frames, to obtain an exact minimum for each molecule. Large databases show degraded performance above voxel order 100 due to the size of the voxel array causing memory system paging.

Choosing a generic voxel order which will work for all molecules is relatively simple once we recognize that the central sections of all the curves are very nearly flat, and that the flat sections overlap on all the molecules. A generic voxel order of 50 has been chosen as this is not far from optimal for the largest and smallest molecules and is at a minimum for our chosen typical protein.

Another set of measurements characterised performance and pixel cover for different sized molecules. The pixel cover figure is the fraction of potential pixels in a window which were actually assigned when drawing the molecule.

Molecule Description	PDB Entry	Atoms	Pixel Cover	Time (sec)
Crambin	1CRN	327	35.5%	8.7
Cobra Venom	1CTX	541	28.6%	8.2
Deoxyribonucleic Acid	3ZNA	756	20.8%	7.0
Krait Venom Neurotoxin	2ABX	1118	33.5%	9.3
Ribonuclease A	5RSA	2229	38.5%	11.9
Human Hemoglobin	2HCO	2282	45.6%	13.4
Penicillopepsin	2APP	2366	39.2%	12.3
Human Hemoglobin V	2LHB	2620	36.6%	10.8
Human Immunoglobin	2FB4	3407	34.9%	11.8
GA3P Dehydrogenase	1GD1	10984	40.0%	13.7

The table shows run-time to be relatively independent of molecule size, although with a trend towards longer run-times for larger molecules. Calculating the average time the program takes to generate each assigned pixel we obtain figures ranging from 93.5μsec per pixel for 1CRN to 130.5μsec per pixel for 1GD1. Thus run-time per displayed pixel only rises by 1.4 times over a 33-fold increase in molecule size.

Measurements of performance in different sized windows were also taken. They are not presented in full here, but indicated run-time increased linearly with number of pixels for a given molecule. For example a shadowed frame of 1CRN is produced in 8.70 seconds at 512×512 resolution, and a non-shadowed frame in 1.35 seconds. At 256×256 the times are 2.22 and 0.36 seconds respectively. At 128×128 the times are 0.58 and 0.10 seconds respectively.

Together these results indicate that the run-time of RasMol is determined almost entirely by shadow tracing time (where the number of shadow rays are equal to the number of visible pixels in the image) and that run-time is very nearly linear in the number of pixels. This is very close to the ideal result of constant time ray tracing (for a given window size) predicted for uniform spatial subdivision techniques by Fujimoto *et al.* [5]. These are important results as they give us confidence in RasMol's ability to handle the increasingly large proteins whose structures are being determined.

A final set of measurements were taken for comparison with the results published by Huang *et al.* [10] which are the fastest figures published to date. We ran a 10 frame sequence for 5RSA at 1280×1024 resolution on a MIPS R2000 based Evans and Sutherland ESV 3+ workstation. Our run-time of 16.7 seconds per frame is over three times faster than their 51.4 CPU seconds. We therefore have some confidence in asserting that RasMol is the fastest uniprocessor shadowed molecule renderer to date.

Multiprocessor Results

Full results and analyses for an implementation of the algorithm on a Meiko InSUN board with four Inmos T800 Transputers are given elsewhere [19]. Important results which were drawn from this work indicated that very nearly linear speed-up of the sequential algorithm was possible when parallelising it using image parallelism with scattered static task allocation.

More recently the program has been ported to the Edinburgh Concurrent Supercomputer at the Edinburgh Parallel Computing Centre. This is a Meiko Computing Surface containing domains of T800 Transputers varying in size from 1 to 131 processors. Results from this implementation have confirmed that a near linear speed-up is possible, however actual observed performance is constrained by a sequential bottleneck in the system. The graphics boards in the system are limited in the rate at which they can accept and display in-coming pixels, and as such place a hard limit on performance of around one frame (800×500) every two seconds, which is reached at only 8 Transputers for 1CRN.

We are not aware of any other multiprocessor shadowed molecule renderer which is currently faster than RasMol. The potential exists, once the bottleneck is removed to achieve real-time, full frame performance. This has provided the motivation for ongoing work to implement the RasMol algorithms within a high performance graphics hardware architecture being developed by Andrew Bissell Technology.

Conclusion

In this paper, we present an algorithm designed specifically for fast rendering of union-of-spheres representations of large molecules such as proteins and nucleic acids. The method may also determine the shadows cast from an arbitrary number of light sources. Our current implementation of this algorithm has been incorporated into a general molecular graphics package and is several times faster than all other results published to date. Both 8 bit and 24 bit frame buffer versions of this package exist and are being used by several university biochemistry and molecular biology departments.

References

[1] David Bacon and Wayne F. Anderson. A fast algorithm for rendering

space-filling molecule pictures. *Journal of Molecular Graphics*, 6(4):219–220, December 1988.

[2] J. E. Bresenham. A linear algorithm for incremental display of circular arcs. *Communications of the ACM*, 20:100–106, 1977.

[3] J. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4:65–83, 1988.

[4] Frances C. Bernstein et al. The Protein Data Bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112:535–542, 1977.

[5] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 4:15–21, 1986.

[6] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4:15–22, October 1984.

[7] Andrew S. Glassner, editor. *An introduction to ray tracing*. Academic Press, 1989.

[8] Stuart Green. *Parallel Processing for Computer Graphics*. Pitman, 1991.

[9] Michael Gwilliam and Nelson Max. Atoms with shadows – an area-based algorithm for cast shadows on space-filling molecular models. *Journal of Molecular Graphics*, 7(1):54–59, March 1989.

[10] Conrad C. Huang, Eric F. Pettersen, Teri E. Klein, Thomas E. Ferrin, and Robert Langridge. Conic: A fast renderer for space-filling molecules with shadows. *Journal of Molecular Graphics*, 9(4):230–236, December 1991.

[11] David T. Jones. The application of fractal clustering to efficient molecular ray tracing on low-cost computers. *Journal of Molecular Graphics*, 9(4):249–253, December 1991.

[12] Nelson Max. Computer representation of molecular surfaces. *IEEE Computer Graphics and Applications*, pages 21–29, August 1983.

[13] Thomas C. Palmer and Frederick H. Hausheer. Context-free spheres: A new method for rapid CPK image generation. *Journal of Molecular Graphics*, 6(3):149–154, September 1988.

[14] Thomas C. Palmer, Frederick H. Hausheer, and Jeffrey D. Saxe. Applications of ray tracing in molecular graphics. *Journal of Molecular Graphics*, 7(3):160–164, September 1989.

[15] Laurence H. Pearl. Calculating CPK images on a UNIX workstation. *Journal of Molecular Graphics*, 6(2):109–111, June 1988.

[16] Thomas Porter. Spherical shading. In *Computer Graphics Vol. 12*, pages 282–285, 1978.

[17] Owen F. Ransen. The art of ray tracing. *BYTE*, pages 238–242, February 1990.

[18] S. Rubin and T. Whitted. A three-dimensional representation for fast rendering of complex scenes. In *Computer Graphics Vol. 14*, pages 110–116, 1980.

[19] Roger Sayle. Parallel algorithms for molecular graphics. B.Sc(Eng) Project Report, Department of Computing, Imperial College, June 1990.

[20] Chris Schafmeister. Fast algorithm for generating cpk images on graphics workstations. *Journal of Molecular Graphics*, 8(4):201–206, December 1990.

[21] Paul Walker. Ray tracing with an array of transputers. *BYTE*, pages 224–225, May 1985.

[22] H. Weghorst, G. Hooper, and D. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Computer Graphics*, 3:52–69, January 1984.

[23] J. T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23:343–349, 1980.

[24] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics*, 12(3), August 1978.

[25] J. R. Woodwark. A multiprocessor architecture for viewing solid models. *Displays Journal*, 5:97–103, 1984.