Fast space-filling molecular graphics using dynamic partitioning among parallel processors

Bradley J. Gertner,* Robert M. Whitnell and Kent R. Wilson

Department of Chemistry, University of California, San Diego, La Jolla, CA, USA
**Department of Chemistry and Biochemistry, University of Colorado, Boulder, CO, USA*

*We present a novel algorithm for the efficient generation of high-quality space-filling molecular graphics that is particularly appropriate for the creation of the large number of images needed in the animation of molecular dynamics. Each atom of the molecule is represented by a sphere of an appropriate radius, and the image of the sphere is constructed pixel-by-pixel using a generalization of the lighting model proposed by Porter (*Comp. Graphics *1978,* **12**, *282). The edges of the spheres are antialiased, and intersections between spheres are handled through a simple blending algorithm that provides very smooth edges. We have implemented this algorithm on a multiprocessor computer using a procedure that dynamically repartitions the effort among the processors based on the CPU time used by each processor to create the previous image. This dynamic reallocation among processors automatically maximizes efficiency in the face of both the changing nature of the image from frame to frame and the shifting demands of the other programs running simultaneously on the same processors. We present data showing the efficiency of this multiprocessing algorithm as the number of processors is increased. The combination of the graphics and multiprocessor algorithms allows the fast generation of many high-quality images.*

Keywords: molecular dynamics, space-filling molecular models, multiprocessor algorithms, parallel processing, spherical primitives, raster graphics

INTRODUCTION

The recent availability of high-powered, relatively inexpensive graphics workstations has led to an increased interest in the use of these workstations in the field of scientific visualization. Our research in the area of molecular dynamics of chemical reactions in solution has benefitted greatly from a program we have developed that displays, with substantial interactive control, the classical molecular dynamics trajectories that are generated from our numerical calculations.[1-3] There is already a sizable literature on the graphical display of molecular structure or dynamics, as perusal of any issue of the *Journal of Molecular Graphics* will readily demonstrate. A review of the field to 1983 is given by Max[4] while a sampling of more recent work that is relevant, and in some cases antecedent to the present work, is given in References 5–14.

However, there is an important difference between the quality of the images required for a user's interactive display (because substantial information can be garnered from relatively crude images) and the more sophisticated graphics desired for the presentation and publication of images. The presentation graphics we are concerned with here require considerably more CPU time to generate a single image, and for the animation of molecular dynamics, a large number of these images is required. (A five-minute 16-mm film, either mono or stereo,[15] requires the generation and exposure of 7200 individual frames.) One can easily see that any savings in CPU time for the generation of a given image will lead to a significant decrease in the real time necessary to produce a film.

We have also been interested in developing graphics algorithms that take advantage of the capabilities of our four-processor Silicon Graphics IRIS 4D/240GTX graphics workstation. The most important features of this computer for our purposes are its 24-bit RGB graphics and its ability to perform parallel processing among some or all of the processors. The implementation of parallel processing on this machine allows each processor to have independent access to common global data and to create local data structures that are inaccessible from the other processors. This loose interleaving of the processors permits each processor to complete its task independently and to return the results of its task to the global structures.

Silicon Graphics supplies an excellent library of graphics subroutines that we have used in fast, but lower resolution, polygon-based graphics for our interactive molecular display programs. Because these routines take advantage of the specialized graphics hardware in the workstation, they can be very fast. However, we have found that for our applications, these standard routines cannot create images of sufficiently high quality that can take full advantage of the

Address reprint requests to Prof. Wilson at the Department of Chemistry 0339, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA, 92093-0339, USA.
Received 11 July 1990; accepted 9 November 1990

resolution of 16-mm film (although the company-supplied hardware and software are continually being improved).

The problem of achieving the very high-quality images we need is simplified by the type of images we will be constructing. We have been primarily interested in displaying our molecular dynamics calculations using space-filling (CPK) models of the molecules in the system. These models can be depicted as sets of intersecting spheres. We have therefore developed a high-resolution, pixel-wise raster algorithm based on the spherical primitives developed by Porter.[16] This algorithm does not incorporate interactions among spheres in the sense of the "global illumination model" described by Palmer et al.[12] However, by carefully considering intersections between the spheres and the edges of the spheres in our images (i.e., where the spheres abut the background), we can achieve much higher quality images than the simple calculation of illumination based on surface normals would give.

Our algorithm does result in very high-quality molecular images. But that is only half the goal. To achieve fast generation of these images, our graphics algorithm, in addition to being very efficient, can easily be divided among a number of processors, each of which independently creates a section of the image. We describe the implementation of this multiprocessor algorithm as well as an efficient method for dividing work among the processors based on the work done in creating previous images. The allocation of processor resources thus automatically adjusts to changes in the images from frame to frame and the shifting usage of the processors by other jobs running concurrently. We have found that the overhead (in terms of CPU time) needed to set up the multiprocessor calculations is small compared to the CPU time each processor uses in creating its share of the final image. We thereby achieve near-theoretical efficiency in the use of multiple processors as compared to the generation of the same image on a single processor. We present in detail the results of our benchmarks on both single and multiple processors.

CREATION OF A SINGLE IMAGE

In this section, we will describe how to construct a single image given a set of atoms with positions (X_i, Y_i, Z_i) and radii R_i. Each atom can be assigned a material as defined by a set of ambient, diffuse and specular colors. The first part of this section describes our implementation of a set of standard modeling transformations that determine the orientation of the system. We next discuss the lighting calculations that result in the color to be assigned to a single pixel given the position of a light source and the material characteristics of the object at that pixel. The construction of a single spherical primitive is then described in detail. Finally, the efficient simultaneous interleaving of several spherical primitives to create the final image is outlined.

Modeling transformations

The system we wish to describe consists of a set of atoms with an arbitrary orientation in space. Therefore, we need to be able to reorient the system with respect to the viewer. This can be achieved with three nondegenerate rotations about the center of the system, which we define for our convenience as

$$\begin{aligned} X_0 &= (X_{max} + X_{min})/2 \equiv 0 \\ Y_0 &= (Y_{max} + Y_{min})/2 \equiv 0 \\ Z_0 &= (Z_{max} + Z_{min})/2 \equiv 0 \end{aligned} \tag{1}$$

where X_{max} and X_{min} are the boundaries of the system in the X-direction (similarly for Y_{max} and Y_{min} and for Z_{max} and Z_{min}). We also allow ourselves to rescale the entire system. Initially, the system size is scaled so that the maximum distance across the box containing the system, either in the X-, Y- or Z-direction is equal to 1. The radii of the spheres are scaled accordingly, so that the sizes are proportional to the appropriate van der Waals radii.

For images that we wish to view with a perspective projection, our system is then processed by applying a perspective transformation[17] to all of the coordinates and radii of the atoms. Starting with the primed coordinates, the transformation for each atom i is,

$$\begin{aligned} Z_i &= Z_i' \\ X_i &= X_i' \cot(\phi/2)/(d - Z_i') \\ Y_i &= Y_i' \cot(\phi/2)/(d - Z_i') \\ R_i &= R_i' \cot(\phi/2)/(d - Z_i') \end{aligned} \tag{2}$$

where ϕ is the angle of the field of view (a measure of how rapidly objects get smaller) and d is the distance from the viewer's eye to the center of the system. It is important to note that, for efficiency, only the radius of the atom is rescaled according to the perspective transformation. The actual spherical representation of the atom is drawn orthographically.

For stereo images,[18] only the transformation of the X-coordinates needs to be modified from that in Equation (2). The new transformation has the form

$$\begin{aligned} X_i^{left} &= (X_i' + p) \cot(\phi/2)/(d - Z_i') \\ X_i^{right} &= (X_i' - p) \cot(\phi/2)/(d - Z_i') \end{aligned} \tag{3}$$

where p is the parallax shift, a measure of how far apart the eyes are. Note that because the parallax shift is applied before the perspective transformation ($X_i' \pm p$, not $X_i \pm p$), the atoms have already been rotated, eliminating the need to apply separate rotations for each eye.

For images drawn orthographically, the perspective transformation is not applied. Stereo is not implemented for orthographic images because they are missing the enhanced depth cues provided by the perspective transformation.[18]

The results of these modeling transformations are then used in conjunction with the lighting model of the next section to generate the image. The last step is to rescale the image to the appropriate screen coordinates to display it on the graphics monitor.

Lighting model

For computational efficiency, our lighting model[17,19,20] consists of one arbitrarily oriented light source at infinity and a viewer at infinity along the positive Z-axis. The image of

each sphere is individually ray-traced in the sense that we ignore all shadowing and all reflections of other spheres.

The color (R, G, B) of each pixel of an atom i with surface normal $\hat{N}$ consists of the following terms:

(1) An ambient light term,

$$\mathbf{I}_i^{amb} = (R_i^{amb}, G_i^{amb}, B_i^{amb}) = \mathbf{C}_i^{amb} \quad (4)$$

which is constant

(2) A diffuse reflection term,

$$\mathbf{I}_i^{diff} = \mathbf{C}_i^{diff}\hat{S} \cdot \hat{N} \quad (5)$$

which depends on the surface normal, the direction of the light source $\hat{S}$, and the constant $\mathbf{C}_i^{diff} = (R_i^{diff}, G_i^{diff}, B_i^{diff})$

(3) A specular reflection term,

$$\mathbf{I}_i^{spec} = \mathbf{C}_i^{spec}(\hat{T} \cdot \hat{N})^{s_i} \quad (6)$$

which depends not only on the surface normal and the light source, but also on the position of the viewer $\hat{V}$ and the specularity of the atom s_i. Here

$$\hat{T} = \frac{\hat{V} + \hat{S}}{|\hat{V} + \hat{S}|} \quad (7)$$

which is the direction midway between the viewer and the light source. This term also depends on the constant $\mathbf{C}_i^{spec}$.

Note that the constants $\mathbf{C}_i^{amb}$, $\mathbf{C}_i^{diff}$ and $\mathbf{C}_i^{spec}$ can be defined independently for each sphere. The resultant color of the pixel $\mathbf{I}_i$ is

$$\mathbf{I}_i = \max(\mathbf{I}_i^{amb} + \mathbf{I}_i^{diff}, \mathbf{I}_i^{spec}) \quad (8)$$

where the maximum is taken over each component of the color vector. We choose $\mathbf{C}_i^{amb}$ and $\mathbf{C}_i^{diff}$ so that the sum $\mathbf{I}_i^{amb} + \mathbf{I}_i^{diff}$ can never be greater than the maximum intensity allowed by the computer on which the image is being created. These general definitions allow us flexibility with the range of intensities available.

Spherical primitive

In this section we describe the production of a spherical primitive that is used to represent a single atom. We will later use this algorithm simultaneously on several spheres. The logic of interleaving several procedures for individual spheres is based on the work of Porter,[16] Ferrin[21] and Feldmann,[22] and will be described in the following section.

The spherical primitive consists of the depth, with respect to the viewer (who is on the positive Z-axis), and the associated surface normal of every pixel covered, entirely or partially, by the sphere. Because we will want to antialias the edge of the sphere, we also need to know what fraction of each pixel is covered by the sphere.[20] This information can then be used in conjunction with the lighting model to determine every color of a single sphere.

To reiterate, we need to answer the following questions about every pixel to determine its relationship with the sphere:

(1) Is the pixel covered by the sphere at all?
(2) If so, is it covered partially or entirely by the sphere?
(3) If it is partially covered, what fraction of it is covered (for antialiasing)?
(4) What is its z-value (for hidden-surface removal)?

(In addition, we also need the surface normal of the sphere at the pixel so that we can perform the lighting calculations.)

For convenience, we choose the spherical primitive to be centered on the pixel $\langle 0, 0 \rangle$, with a radius of R, which is rounded to the nearest integral number of pixels. (Note that this is the radius after the modeling transformations of Equations (1–3) have been performed.)

If the pixel $\langle x, y \rangle$ satisfies

$$x^2 + y^2 < (R + 1/2)^2 \quad (9)$$

then it is covered by the sphere.

If the pixel also satisfies

$$x^2 + y^2 < (R - 1/2)^2 \quad (10)$$

then it is entirely covered by the sphere.

Otherwise, the pixel is only partially covered by the sphere, and we need to determine what fraction f of the pixel is covered. At the upper limit (outer edge, Equation (9)) the pixel would not be covered at all, and thus $f \rightarrow 0$. The pixel color is then that of the object behind the sphere. At the lower limit (inner edge, Equation (10)) the pixel would be covered completely, so $f \rightarrow 1$ and the pixel color is that of the sphere. Therefore we can linearly interpolate between the two limits to determine the fractional contribution to the color of the pixel from the sphere:

$$f = 1/2 + dR \quad (11)$$

where

$$dR = R - \sqrt{x^2 + y^2} \quad (12)$$

To determine the z-value of the pixel, we must first determine if

$$R^2 - x^2 - y^2 > 0 \quad (13)$$

If this is the case, the z-value of the center of the pixel is

$$z = \sqrt{R^2 - x^2 - y^2}. \quad (14)$$

Otherwise, we choose $z = 0$. This is because the center of the pixel is not covered by the sphere. We therefore use the z-value of the closest point of the sphere to the center of this pixel.

Implementation of the spherical primitive

Because of the discrete nature of the pixels, our implementation of the spherical primitive advantageously performs all calculations in fixed-point arithmetic. (The Silicon Graphics computer on which we have implemented this algorithm is a 32-bit machine. This places a limit on the maximum color resolution. Higher resolutions can be obtained from 64-bit machines, for example. The implementation of the algorithm we present here does not explicitly depend on the number of available bits except where noted.) This discretization also allows us to work our way incrementally through the array of pixels covered by the sphere in a systematic way. In a fashion similar to that of Porter,[16] the sphere is drawn one scanline at a time, from top to bottom. Because the sphere is centered on a pixel (by definition), we can cut

the work in half by using the left-right symmetry of the sphere. To use the full (eightfold) symmetry of the sphere requires a prohibitively large amount of storage to perform the hidden-surface removal after all the spheres have been drawn, and also prohibits the use of multiprocessing if hidden-surface removal is performed as the spheres are drawn. Thus, we restrict ourselves to the twofold symmetry of the hemispheres.

The scanline implementation systematically answers questions 1–4 by checking to see if the criteria are satisfied. To answer question 1, we start at the top of the sphere, work outward for the scanline on which the top of the sphere lies and then downward to subsequent scanlines. The center of the top scanline of the sphere is located at the pixel $\langle 0, R\rangle$, and the center is, in general, at pixel $x = 0$ for every scanline. Because we have left-right symmetry and only need to consider positive values of x, working outward means stepping outward in the x-direction from $x = 0$, one pixel at a time, until we no longer satisfy Equation (9). To answer questions 2–3 we find all of the partially covered pixels by stepping inward until we no longer satisfy Equation (10). At the same time, we can answer question 4 by calculating the depth z of each pixel. This information can then be combined with the lighting model of the previous section to produce the colors of every pixel affected by the sphere on this scanline. In fact, as discussed below, we will not generate the color of the pixels until we have all the depth and coverage information for all of the spheres that overlap the scanline. We are able to perform hidden-surface removal then because we have equivalent information for all of the spheres that affect this scanline.

After we finish a scanline, we step down to the next scanline and repeat the process. To reduce the amount of computation, however, we can use the information obtained from the previous line. If we are working on the top half of the sphere, we know that the number of pixels that satisfy Equation (9) can only increase, so we start with the previous maximum x-value and work outward. If we are processing the bottom half, we can only step inward, and starting with the previous x-value is again a good idea.

More precisely, the algorithm is as follows, starting at pixel $\langle 0, R\rangle$:

(1) Working outward, add to the list of covered pixels all pixels that satisfy Equation (9). This is accomplished by monitoring the quantity

$$\Delta(x, y) = (R + 1/2)^2 - x^2 - y^2 \tag{15}$$

which is initially $\Delta(0, R) = R + 1/4$. We can determine whether the pixel beside (x, y) should be accepted by checking $\Delta(x + 1, y)$

$$\begin{aligned}\Delta(x + 1, y) &= (R + 1/2)^2 - (x + 1)^2 - y^2 \\ &= \Delta(x, y) - 2x - 1 \\ &= \Delta(x, y) + Inc(x)\end{aligned} \tag{16}$$

If this quantity is positive, the pixel is covered by the sphere and we need to check the next pixel. To move down to the next scanline at the same x-coordinate, we just need

$$\begin{aligned}\Delta(x, y - 1) &= (R + 1/2)^2 - x^2 - (y - 1)^2 \\ &= \Delta(x, y) + 2y - 1 \\ &= \Delta(x, y) + Dec(y)\end{aligned} \tag{17}$$

Note that the functions $Inc(x)$ and $Dec(x)$ allow us to monitor the edge of the sphere without ever performing a multiply or a divide, just bitshifts, additions and elementary comparisons. For the upper half of the sphere we will always attempt steps outward, $x \rightarrow x + 1$. For the lower half we need only step inward, and then only when $\Delta(x, y) < 0$. This procedure defines the outer boundary of the sphere; i.e., it answers question 1.

(2) To answer question 2 we must work inward, $x \rightarrow x - 1$, toward the center of the defining circle at every scanline. If the quantity

$$\begin{aligned}\Delta_R(x, y) &\equiv R^2 - x^2 - y^2 \\ &= \Delta(x, y) - R - 1/4\end{aligned} \tag{18}$$

is less than zero; i.e.,

$$R^2 < x^2 + y^2 < (R + 1/2)^2 \tag{19}$$

the pixel is still only partially covered by the sphere. In fact, we know that $z = 0$ (question 4). We also know that $-1/2 < dR < 0$, so $f < 1/2$ (questions 2–3). Once we have stepped in far enough so that $\Delta_R(x, y) \geq 0$ is satisfied, we need to check for the inner edge, Equation (10). If we have

$$(R - 1/2)^2 < x^2 + y^2 < R^2 \tag{20}$$

we know that $0 < dR < 1/2$, so that $f > 1/2$, as well as $z > 0$.

(3) To determine dR with greater precision and thereby answer question 3, we use an iteration scheme based on Newton's method of root finding. This scheme avoids the use of a square-root function and the associated floating-point arithmetic. If the nth iteration of this method gives dR_n, then

$$dR_{n+1} = dR_n - g(dR_n)/g'(dR_n) \tag{21}$$

with

$$\begin{aligned}g(dR) &= (R - dR)^2 - x^2 - y^2 \\ &= R^2 - x^2 - y^2 - 2RdR + (dR)^2 \\ &= \Delta_R(x, y) - 2RdR + (dR)^2\end{aligned} \tag{22}$$

and

$$g'(dR) = -2R + 2dR \tag{23}$$

Therefore, the $(n + 1)^{st}$ iteration of dR is

$$dR_{n+1} = \frac{(\Delta_R(x, y) - (dR_n)^2)}{2(R - dR_n)} \tag{24}$$

If the pixel satisfies Equation (19), our best initial guess would be $dR_0 = -1/4$. On the other hand, if the pixel satisfies Equation (20), our best guess is $dR_0 = 1/4$. Applying two iterations of the above scheme gives us more than nine bits of accuracy for all values of the radius R, with two integer divides. (Note that if more bits of accuracy are needed, another iteration of this algorithm may be required.) This gives us the fractional coverage, $f = 1/2 + dR$, and answers question 3.

(4) To calculate the z-value of the pixel we need to combine the ideas of the previous methods. First we step forward, $z \rightarrow z + 1, \ldots,$ while

$$x^2 + y^2 + z^2 < R^2 \tag{25}$$

and then we determine the fractional component of the depth with an iteration scheme like that used for dR. To step forward we define the quantity

$$\Delta_z(x, y, z) \equiv R^2 - x^2 - y^2 - z^2 \tag{26}$$

which allows us to monitor whether we want to increment z; that is, we accept a step forward, $z \rightarrow z + 1$, if we satisfy

$$\begin{aligned}\Delta_z(x, y, z + 1) &= R^2 - x^2 - y^2 \\ &\quad - (z + 1)^2 \\ &= \Delta_z(x, y, z) + Inc(z) \\ &\geq 0\end{aligned} \tag{27}$$

Note that the initial value of Δ_z is

$$\Delta_z(x, y, 0) = \Delta_R(x, y) \tag{28}$$

so that once $z > 0$ we can initialize Δ_z from Equation (18). After we have determined the integer portion $Int(z)$ of z, we can apply Newton's method to determine the fractional portion dz, starting with $dz_0 = 1/2$:

$$dz_{n+1} = \frac{(\Delta_z(x, y, z) + (dZ_n)^2)}{2(Int(z) + dz_n)} \tag{29}$$

Two iterations achieves the desired accuracy, thus

$$z = Int(z) + dz_2 \tag{30}$$

The final step is to determine the surface normal $\hat{N}$ and the absolute depth of the pixel for hidden-surface removal. The depth is just the Z-value of the center of the sphere Z_i plus the z-value calculated in Equation (30):

$$Z_i(x, y) = Z_i + z(x, y) \tag{31}$$

The surface normal is just

$$\hat{N} = (x/R, y/R, z/R) \tag{32}$$

for the right hemisphere, and

$$\hat{N} = (-x/R, y/R, z/R) \tag{33}$$

for the left hemisphere, and this can be coded efficiently by first calculating $1/R$ at the beginning of the construction of the spherical primitive. Once we have this information, the pixel color can be calculated according to Equation (8) using lookup tables for the values of $(\hat{T} \cdot \hat{N})^{s_i}$. These tables are regenerated only when the angle of illumination is changed.

Simultaneous generation of spheres

If our images consisted of a single sphere, the algorithm above would be sufficient to generate the image. However, we are interested in the generation of images containing many spheres and these spheres will be allowed to overlap or intersect. This section will be concerned with the efficient simultaneous generation of these spheres.

Two problems are to be dealt with here. First, we need to define how the pixels on the edge of each sphere are to be blended (antialiased) with whatever color is behind it, whether that color is due to another sphere or to the background. Second, spheres that intersect will have interior pixels that must be blended to smoothly join the two spherical primitives. The blending process makes this algorithm a true three-dimensional (3D) algorithm rather than the so-called two-and-a-half dimensional ($2\frac{1}{2}$D) methods[5] that define the color of a pixel solely by whatever object overlapping that pixel is closest to the viewer.

The generation of each image is made more efficient by some preprocessing.[21] This is accomplished by first sorting the list of spheres into the order in which they will initially appear (from top to bottom) on the screen. The spheres must therefore be sorted by $Y_i + R_i$ such that the largest value is first. This sorting allows us to determine when an atom becomes "active," that is, it allows us to determine which scanline contains the top of the sphere.

We can then work our way from top to bottom through the scanlines. As each atom becomes active, it is added to a list of spheres that is already sorted from front to back, i.e., by the Z-value of the center of the sphere. This sorting minimizes the number of color calculations to be performed on each pixel because of hidden-surface removal. Finally, as the algorithm passes the bottom of a sphere the sphere is removed from the active list. Thus, on any given scanline, we have a list of only those spheres that might appear on that scanline, and that list is sorted so that we know which spheres are in front of the other spheres. Thus, for any given pixel, it is easy to determine which sphere is closest to the viewer.

We now turn to the antialiasing of the edges of the spheres. From the spherical primitive algorithm, the fractional coverage of the pixel by the frontmost sphere is known. Furthermore, after processing the spheres that cover this pixel, the color of the object immediately behind that sphere is known as well. (Note that this object may be another sphere or the background.) Thus, the color intensity $\mathbf{I}$ of the pixel can be defined in terms of the fractional coverage f, the color $\mathbf{I}_1$ of the forward sphere and the color $\mathbf{I}_2$ of the rear object by

$$\mathbf{I} = f\mathbf{I}_1 + (1 - f)\mathbf{I}_2 \tag{34}$$

Note that we perform this antialiasing only if $f < 1$ for the frontmost sphere covering a given pixel.

The problem of blending interior pixels must be handled somewhat differently because it is nontrivial to calculate the fractional coverage of two interior pixels from spheres of different radii.[23] Instead, we linearly interpolate the two colors[16,21] as a function of the associated depths (z-values) of the surface of the sphere. We define a constant ϵ such that if one surface is more than ϵ in front of the other surface, then the color of the pixel is taken to be that of the front surface. Otherwise, we define the color $\mathbf{I}$ of the pixel in terms of the colors $\mathbf{I}_1$ and $\mathbf{I}_2$ of the two spheres by

$$\mathbf{I} = \frac{(\epsilon + \delta z)\mathbf{I}_1 + (\epsilon - \delta z)\mathbf{I}_2}{2\epsilon} \tag{35}$$

where

$$\delta z = z_1 - z_2 \tag{36}$$

and z_1 and z_2 are the depths of the two surfaces. Our choice of ϵ is 1.

Because we calculate the depth of each sphere associated with a given pixel in two stages—first the integer part, and then the fractional part (Equations (29) and (30))—we can increase the efficiency of our algorithm as follows. (For a

very similar procedure, see Pearl.[24]) If one sphere is more than ϵ behind another at a given pixel and the sphere in front covers the pixel entirely, the information about the back sphere will not be used at that pixel. Thus by first determining if the information about a particular pixel is needed, we can save a substantial amount of computational effort. Because the spheres have been previously sorted front-to-back according to the positions of their centers, the pixel will be covered by the first sphere in the list most of the time. (This points out another difference between the algorithm presented here and $2\frac{1}{2}$D algorithms, where the first sphere in the sorted list would always cover the pixel being considered.) If we wish to check the approximate depth of each sphere covering the pixel (other than the first sphere in the list, whose depth is always calculated exactly) to determine if it is within ϵ of the front of the image, we need only add ϵ to the integer part of the depth of the back sphere to place an upper limit on the exact depth. The preprocessing of the approximate depths results in not having to evaluate Equation (29) for most of the hidden pixels. In practice, this pre-evaluation results in a 40% savings in CPU time.

The blending and antialiasing described in this section result in very natural and smooth images even in the difficult case of intersecting spheres. We wish to reiterate that the algorithms presented in this section are completely independent of the resolution of the display system, the number of bits for each color of each pixel, or the number of bits used in the intermediate calculation. However, the quality of the image increases considerably with an increase in any of those parameters.

As the old saying goes, "Seeing is believing." In the Color Plates we present examples of various types of images that can be generated readily. Color Plate 1 displays a representative organic molecule, 2-mercaptonicotinamide. It demonstrates the ability of this algorithm to display high-quality intersecting spheres (as in the aromatic ring), as well as more isolated spheres (such as that of the sulfur atom). Color Plate 2 shows in more detail the quality of intersections that can be achieved with this algorithm. This image is of two intersecting spheres with radii corresponding to those of the cyanide ion. Color Plates 3 and 4 show single time steps of a molecular dynamics trajectory for a $Cl^- + CH_3Cl$ S_N2 reaction in water.[3] We highlight only those water molecules that have the greatest effect on the reactants at any given time. The latter type of images present a particular difficulty because successive images of an animation can be quite different. In the next section, we will describe how to efficiently deal with this situation.

MULTIPROCESSING AND DYNAMIC PARTITIONING

The task of filling the screen with the images described in the previous section is, to say the least, computationally intensive. Each image consists, in general, of several hundred scanlines with hundreds of thousands of individually calculated pixels. The algorithm presented in the preceding section lends itself to parallel processing because the spheres are local objects in the sense that the pixels generated on each scanline are independent of those on any other scanline. This scanline independence allows us to divide the task of creating the image into several subtasks, each consisting of the generation of a set of adjoining scanlines. It is possible to divide the overall task in other ways (such as quadrants); however the left-right symmetry of the spheres could not then be fully utilized as described in the previous section.

To effectively utilize the multiprocessing environment, we want to keep every processor busy. If one processor is doing all the work while the others are idle, we are defeating the purpose of multiprocessing. In the absence of external demands on CPU time, it might be possible to devise a scheme that predicts the optimal division of work among processors before any image is generated. However, in the multi-user environment in which we generate graphics, the load on any given processor is constantly changing thereby making explicit prediction schemes very difficult to implement. Because of this load variability, we choose to implement a very simple, but robust, partitioning scheme. In this respect, the following algorithm differs from those implemented for dedicated graphics processors.[6] Finally, our partitioning algorithm must take into account that the overall images we wish to generate are not necessarily symmetric, as Color Plates 1–4 show. Thus, the algorithm we present here will prove to be very effective for variably loaded processors as well as for asymmetric images.

Dynamic partitioning

If each successive image is only slightly different than its predecessor, then knowledge of the amount of CPU time each processor used in generating image n should be useful in predicting the optimal distribution of processor loads for image $n + 1$. Consider a system with N processors, each of which requires t_i CPU seconds to complete its subtask. Then the real time T_n used to create image n is given by

$$T_n = \max_{1 \le i \le N} \{t_i^n\} \tag{37}$$

(Ideally, all of the t_i^n will be equal so that each processor completes its subtask at exactly the same time.) Furthermore, if each processor is required to draw l_i scanlines in the image n for a total of $L_n = \sum_{i=1}^{N} l_i^n$ scanlines, then the effective rate of scanline production is $K_n = L_n/T_n$. Given this knowledge, we wish to develop an algorithm that minimizes T_{n+1}.

The total number of scanlines L_n is fixed by the image being drawn, but the total time T_n is determined by the distribution of work among the processors. To minimize the total elapsed time, we must maximize the effective rate of scanline production. The simplest optimization scheme is to assume that the rate of scanline production for each processor is approximately independent of the number of scanlines drawn for small variations in this number. Because we know that the optimal time is the same for each processor, we know that the maximum rate of scanline production is given by

$$K_n^{\max} = L_n/T_{n\ \mathrm{opt}} \tag{38}$$

But the optimization assumption tells us that the rate of scanline production for each processor is

$$k_i^n = l_i^n/t_i^n = l_{i\ \mathrm{opt}}^n/T_{n\ \mathrm{opt}} \tag{39}$$

Because the subtasks are run in parallel, the optimal rate is given by

$$K_n^{\max} = \sum_i l_i^n/t_i^n \equiv \sum_i k_i^n \tag{40}$$

On the next image, the number of scanlines drawn by each processor will then be

$$l_{i\,\mathrm{opt}}^{n+1} = L \frac{l_i^n/t_i^n}{\sum l_i^n/t_i^n} = L \frac{k_i^n}{K_n^{\max}} \tag{41}$$

Thus, if in the current image n a processor takes longer to draw its scanlines than would be predicted by the rate $K_n^{\max}$, in the next image that processor will be allocated a smaller number of scanlines to draw. To compensate for the fact that successive images can occasionally vary considerably and for the existence of random external sources of CPU usage, we do not implement Equation (41) directly. Instead, we allow the current distribution of scanlines among the processors to relax to that given by Equation (41) by using a damping factor γ. If we consider Equation (41) to be the optimal distribution for image $n + 1$, then the damped distribution we actually use is given by

$$l_{i\,\mathrm{d}}^{n+1} = l_i^n + [l_{i\,\mathrm{opt}}^{n+1} - l_i^n]/\gamma \tag{42}$$

We use a value of 2.5 for γ in our implementation of this algorithm. Another factor that must be considered is that the total number of scanlines changes from one image to the next. To handle this variability we simply rescale the damped distribution appropriately:

$$l_i^{n+1} = l_{i\,\mathrm{d}}^{n+1} \frac{L^{n+1}}{L^n} \tag{43}$$

To complete the implementation of this algorithm, we must have an initial distribution of scanlines. In fact, the algorithm is relatively insensitive to this initial choice. Many of the images we generate in the course of our molecular dynamics studies are roughly spherical and the initial distribution of scanlines we use is chosen to reflect that shape. However, tests that we have done where we begin with an equal distribution of scanlines among the processors (and therefore an unequal distribution of effort) show that the partitioning algorithm settles down within four or five images to a scanline distribution much more in accord with the spherical nature of the images, as Table 1 shows. There we list the percentage of scanlines in a given frame that each processor is responsible for calculating. We present these percentages for two different images: the water system of Color Plates 3–4 (but displaying all water molecules) and an enlarged image that fills the entire 1000-scanline area used in our display. In our tests, we begin with a set of partitions that differ considerably from the optimum for each of these images. However, the results presented in Table 1 show that within 4–5 frames, the repartitioning algorithm leads to a variation of less than 1% in the size of the partitions for each of the processors. Thus, the robustness of the algorithm is evident in this fast and stable response to a large perturbation. The perturbations due to other processes occupying one or more processors are also handled well by this dynamic partitioning, thereby resulting in a considerable decrease in the real time needed to create a large number of images even when there are external random loads being placed on some or all of the processors.

Table 1. Test of repartitioning algorithm

Image	Processor	Frame number 0	1	2	3	4
Small Water	1	0.241	0.273	0.291	0.292	0.293
	2	0.245	0.192	0.186	0.187	0.187
	3	0.250	0.200	0.189	0.182	0.181
	4	0.264	0.334	0.334	0.339	0.339
Large Water	1	0.295	0.265	0.250	0.245	0.243
	2	0.185	0.220	0.237	0.243	0.245
	3	0.183	0.226	0.238	0.244	0.247
	4	0.337	0.289	0.275	0.268	0.266

TIMING RESULTS

In this section, we present the results of our CPU time benchmarks on a four-processor Silicon Graphics IRIS 4D/240GTX for several of the images displayed in the color plates. We wish to determine both the efficiency of the graphics algorithm presented and the effect of the dynamic partitioning method. To achieve this, we have gathered timing statistics for three different sets of images:

(1) 256 water molecules, as well as the atoms of a model S_N2 reaction[1,3,25]
(2) 100 argon atoms and the atoms of a Cl + Cl_2 reaction[2,26]
(3) the energy flow dynamics[3] of which Color Plates 3–4 are two representative images. (In fact, Color Plate 3 represents the beginning of a dynamics run at 500 fs before the reaction reaches the transition state, while Color Plate 4 depicts the system at the transition state.)

Each set of images was generated from our molecular dynamics display program and represents a set of time steps for a molecular dynamics trajectory. For all three cases, the image varied to some degree from frame to frame. In the water and argon images, this variation was fairly small. In the energy flow case, on the other hand, the variation was quite large, with only three spheres being displayed at the beginning of the set of images and over 200 being displayed by the last frame. We therefore have a range of images that are useful in testing the robustness of both the imaging and multiple processor algorithms.

Table 2 presents a numerical description of these images. The numbers given here are averages over 100 frames of an actual molecular dynamics trajectory. Table 2 gives the average number of spheres in each image, the average total

Table 2. Image parameters

Image	Spheres	Scanlines	Pixels drawn	Pixels blended
Water	771	651	303704	19164
Argon	103	606	273029	9689
Energy flow	12	454	167181	3378

number of scanlines that any sphere in the image overlaps, the average number of pixels drawn and the average number of pixels blended. The last number gives some indication of how many spheres intersect in the image. We note here that the argon image consists mainly of nonintersecting but overlapping spheres, while the water image contains spheres that intersect substantially as well as overlap.

Table 3 presents timing data with and without overhead for these three sets of images. Overhead is defined here as the time it takes to preprocess all the spheres—i.e., modeling transformations, material definitions, and sorting—plus the time it takes to display the pixels on the screen once the spheres are processed. In our implementation of the graphics and multiprocessing algorithms, this overhead is performed solely by the master processor. However, the initializations needed for multiprocessing on our Silicon Graphics computer that cannot be performed solely by the master processor (and must therefore also be done by each of the slave processors) are not included in our definition of the overhead.

In Table 3, the average time needed to construct each frame of the image is given in seconds. These times are also given as a function of the number of processors used by the multiprocessing algorithm. From this timing data, an "effective processor" measure can be calculated (with one real processor on a quiescent machine being defined as 1 effective processor) and these results are noted in the table as well.

For images that remain relatively constant from frame to frame, the partitioning algorithm is very efficient. This can be seen from Table 3 in that the number of effective processors is very close to the theoretical efficiency when the overhead is not taken into account. Even when the overhead is included in the timings, the number of effective processors is still quite large.

For the energy flow dynamics, as Color Plates 3–4 show, the image can change substantially from frame to frame presenting a large challenge to any prediction scheme for partitioning. This relative lack of continuity is reflected in the moderate decrease in the number of effective processors for this image. However, even in this very difficult case the ability to obtain more effective processors when all available processors are used testifies to the robustness of the algorithms presented here.

We have demonstrated in this work that it is possible to obtain high-quality images of molecular dynamics very efficiently. Thus, it is possible to generate "pixel-perfect" high-resolution images for an animated film in a few hours. We also wish to stress that the algorithms presented here, although implemented on a Silicon Graphics multiple processor computer, are not specific to a particular machine. We also do not use specialized transformation and rendering hardware which, while very useful for lower quality images, do not produce the pixel-perfect representations described here. Thus, as the number of processors or the number of bits assigned to a particular color increases, very minor modifications to the programs that implement these algorithms will be sufficient. Eventually, as processors become even faster and machines with more processors running in parallel become available, the production of high-quality animated films of molecular dynamics will be limited only by the time needed to expose the image and advance the film rather than the production of the image itself. The data of Table 3 show that we have already come close to that point with the present algorithms and technology.

ACKNOWLEDGMENTS

We would like to thank Ilan Benjamin for useful discussions, Sherman George and Dan Brewton for valuable assistance in creating animated films and Bill Keirstead for assistance in creating the color plates. Gertner acknowledges support from a National Science Foundation Graduate Fellowship.

REFERENCES

1 Gertner, B.J., Wilson, K.R. and Hynes, J.T. Nonequilibrium solvation effects on reaction rates for model S_N2 reactions in water. *J. Chem. Phys.* 1989, **90**, 3537
2 Benjamin, I., Gertner, B.J., Tang, N.J. and Wilson, K.R. Energy flow in an atom exchange chemical reaction in solution. *J. Am. Chem. Soc.* 1990, **112**, 524

Table 3. Timing results of multiprocessor algorithm

Image	Processors	Time (sec) w/overhead	Effective processor w/overhead	Time (sec) w/o overhead	Effective processor w/o overhead
Water	1	11.12	1.00	10.57	1.00
	2	6.04	1.84	5.48	1.93
	3	4.25	2.62	3.68	2.87
	4	3.33	3.34	2.76	3.83
Argon	1	6.70	1.00	6.27	1.00
	2	3.60	1.86	3.15	1.99
	3	2.59	2.59	2.13	2.94
	4	2.04	3.28	1.59	3.94
Energy flow	1	3.72	1.00	3.48	1.00
	2	2.09	1.78	1.86	1.87
	3	1.51	2.46	1.28	2.72
	4	1.22	3.05	0.96	3.62

3 Gertner, B.J., Whitnell, R.M., Wilson, K.R. and Hynes, J.T. Activation to the transition state: Reactant and solvent energy flow for a model S_N2 reaction in water. *J. Am. Chem. Soc.* 1991, **113**, 74
4 Max, N.L. Computer representation of molecular surfaces. *J. Mol. Graphics* 1984, **2**, 8
5 Pique, M.E. Fast 3D display of space-filling molecular models. Technical report 83-004, Department of Computer Science, University of North Carolina, Chapel Hill, NC, USA, 1983
6 Hubbard, R.E. and Fincham, D. Shaded molecular surface graphics on a highly parallel computer. *J. Mol. Graphics* 1985, **3**, 12
7 Johnson, B.A. MSURF: A rapid and general program for the representation of molecular surfaces. *J. Mol. Graphics* 1987, **5**, 167
8 Goodsell, D.S. RMS: Programs for generating raster molecular surfaces. *J. Mol. Graphics* 1988, **6**, 41
9 Palmer, T.C. Context-free spheres: A new method for rapid CPK image generation. *J. Mol. Graphics* 1988, **6**, 149
10 Goodsell, D.S., Mian, I.S. and Olson, A.J. Rendering volumetric data in molecular systems. *J. Mol. Graphics* 1989, **7**, 41
11 Gwilliam, M. and Max, N. Atoms with shadows—An area-based algorithm for cast shadows on space-filling molecular models. *J. Mol. Graphics* 1989, **7**, 54
12 Palmer, T.C., Hausheer, F.H. and Saxe, J.D. Applications of ray tracing in molecular graphics. *J. Mol. Graphics* 1989, **7**, 160
13 Klein, T.E., Huang, C.C., Pettersen, E.F., Couch, G.S., Ferrin, T.E. and Langridge, R. A real-time malleable molecular surface. *J. Mol. Graphics* 1990, **7**, 16
14 Lauher, J.W. Chem-Ray: A molecular graphics program featuring an umbra and penumbra shadowing routine. *J. Mol. Graphics* 1990, **8**, 34
15 Bartlett, N., Erickson, G.A., Mackay, D.H.J. and Wilson, K.R. Easy projection of stereo movies. *J. Mol. Graphics* 1986, **4**, 190
16 Porter, T. Spherical shading. *Comp. Graphics* 1978, **12**, 282
17 *Graphics Library Programming Guide* Silicon Graphics, Inc., Mountain View, CA, USA, 1989
18 Lipton, L. *Foundations of the Stereoscopic Cinema* Van Nostrand Reinhold, New York 1982
19 Newman, W.M. and Sproull, R.F. *Principles of Interactive Computer Graphics* McGraw–Hill, New York, 1973
20 Hearn, D. and Baker, M.P. *Computer Graphics* Prentice-Hall, Englewood Cliffs, 1986
21 Ferrin, T.E. Computer subroutine SHADE, written in the C programming language, University of California, San Francisco, USA, 1985
22 Feldmann, R.J., Bing, D.H., Furie, B.C. and Furie, B. Interactive computer surface graphics approach to study of the active site of bovine trypsin. *Proc. Nat. Acad. Sci. USA* 1978, **75**, 5409
23 Smith, G.M. and Gund, P. Computer-generated space-filling molecular models. *J. Chem. Info. Comp. Sci.* 1978, **18**, 207
24 Pearl, L.H. Calculation of CPK images on a UNIX workstation. *J. Mol. Graphics* 1988, **6**, 109
25 Bergsma, J.P., Gertner, B.J., Wilson, K.R. and Hynes, J.T. Molecular dynamics of a model S_N2 reaction in water. *J. Chem. Phys.* 1987, **86**, 1356
26 Bergsma, J.P., Reimers, J.R., Wilson, K.R. and Hynes, J.T. Molecular dynamics of the $A + BC$ reaction in rare gas solution. *J. Chem. Phys.* 1986, **85**, 5625