Context-free spheres: A new method for rapid CPK image generation

Thomas C. Palmer

Advanced Scientific Computing Laboratory, Program Resources, Inc., National Cancer Institute – Frederick Cancer Research Facility, Frederick, MD 21701, USA

Frederick H. Hausheer

The Oncology Center, The John Hopkins University, Baltimore, MD 21205, and Advanced Scientific Computing Laboratory, Program Resources, Inc., National Cancer Institute – Frederick Cancer Research Facility, Frederick, MD 21701, USA

A new algorithm for rendering CPK images of molecules is presented. The algorithm is based on the observation that (given certain assumptions) the appearance of a sphere representing an atom is independent of the atom's position or orientation. For example, the size of a sphere's projection on the viewing plane is independent of its distance from the viewing plane. The shading of a sphere is dependent only on lighting parameters that are identical for each atom type. This algorithm takes advantage of this observation by precomputing a template *for each unique atom type and* stamping *these into the image with appropriate offsets in* X, Y *and* Z. *The implementation described herein enables generation of CPK images an order of magnitude faster than previous methods, with little sacrifice in image quality.*

Keywords: CPK modeling, depth buffer algorithms, raster graphics, interactive applications

Received 12 February 1988
Accepted 8 March 1988

Visualizing molecular structures is a critical process that is becoming increasingly important in a number of scientific disciplines. Many algorithms exist to generate a variety of images and provide representative "views" of molecular structures. Two major types of molecular visualizations have been popularized: space-filling CPK models and wire-frame models. The latter are useful in presenting a "see-through" model to the scientist and enable users to focus on intramolecular topology and relative atomic positions. In addition, existing computer graphics hardware lets users interact with wire-frame models consisting of up to a few thousand atoms.

CPK models aid in visualizing the spatial distribution and conformation of molecular structures. The surface of such structures is formed by the union of van der Waals contact surfaces, and enables enhanced perception and conceptualization of critical biologic interactions commonly occurring at this interface. CPK models also are used to study the effects of small perturbations in molecular structure, and they can facilitate the concise study of local and/or global alterations in conformation. Unfortunately, current graphics hardware does not permit interaction with CPK models larger than a few tens of atoms.

Several algorithms exist for generating CPK images of molecules. Atoms are represented by spheres in these images; thus, sphere intersections represent bonds between atoms. Sphere sizes are based on predetermined radii for van der Waals contact distances. The spheres are typically color-coded by atom type. Max[1] reviews several algorithms for generating CPK images.

Perhaps the best-known and most widely used method is Porter's scanline-oriented depth-buffer algorithm.[2] Two buffers are maintained during the processing of a scanline. The *shade buffer* contains color values and eventually holds the finished scanline; the *depth buffer* stores the depth values of currently visible elements of the shade buffer. Porter uses Bresenham's incremental circle generator[3] to find a sphere's visible intersection with a scanline: the front-facing semicircle perpendicular to the viewing plane. This method also gives the depth of the semicircle at each point along the scanline. To determine visibility, the depth is subsequently compared

to the current value in the depth buffer. If the sphere is visible at this point, the shade is computed and the depth buffer updated. Porter's algorithm also provides antialiasing of a sphere's perimeter edge and the edges caused by its intersection with other spheres. This method produces high-quality images but takes several minutes for molecules consisting of approximately 1 000 atoms.

Dynamic interaction with CPK models requires either special hardware or a sacrifice in image quality. Pique[4] built a system capable of displaying molecular systems of up to 50 atoms in near-real-time. Two approaches were used to achieve an update rate of approximately 8 per second. First, the algorithm was implemented on a powerful display processor directly coupled to the image memory. Second, the quality of the image was sacrificed, due to a reduction in resolution and the restriction of the light source to the viewpoint. Max's[5] parabolic approximation to spherical shading was also used.

Pixel-planes is an example of adapting specialized hardware to the sphere-rendering problem.[6] Pixel-planes is a raster device with a processor for every pixel. Its sphere-rendering algorithm uses Max's[5] parabolic approximation for determining both visibility and shading. It can display up to 13 000 spheres in real time.

The methods described above share an important calculation. The shading function is evaluated for *every visible pixel on a sphere*. Both Porter and Max recognized this computation as the bottleneck and optimized accordingly. Porter used Bresenham's incremental method to derive the shade, and, in his original implementation, this computation was coded in assembly language. Max's parabolic approximation is a quadratic polynomial that can be quickly evaluated by forward differences.

While these methods are efficient, they do not consider the following observation. Under two assumptions, *the size and shading of a sphere are independent of the sphere's position*. The assumptions are:

(1) *Orthographic parallel projection.* With parallel projection (as opposed to perspective projection) the size of a sphere as projected on the viewing plane is independent of its distance from the viewing plane.
(2) *Infinite light source.* Assuming that the light source is infinitely far away, the shading of a sphere is independent of its position with respect to the light source.

These assumptions restrict the image-generation environment but have little effect on the quality of images produced. Most CPK images are rendered under such conditions.

To take advantage of this observation, we precompute one *template* for each unique atom type. For each atom in the molecule to be imaged, we copy the template corresponding to its atom type into the image with appropriate offsets in X, Y and Z. This operation is called *stamping*. With this approach, the expensive shading calculations are performed only once for each unique atom type, which removes the current bottleneck in CPK image generation.

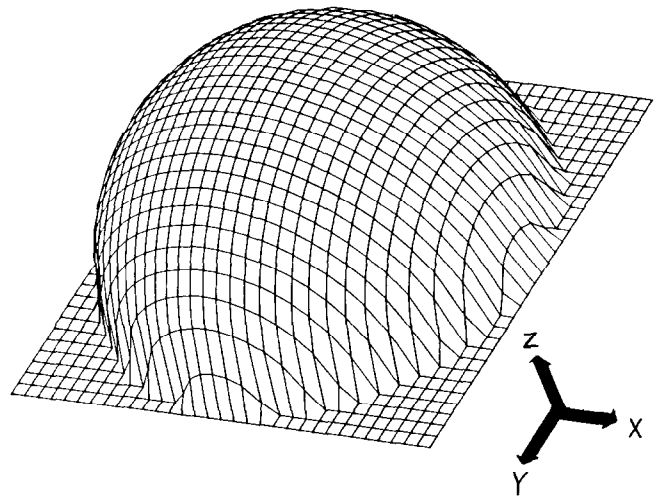

Figure 1. A template can be visualized as the forward-facing hemisphere of the van der Waals surface

We call this aspect of sphere rendering *context-free.** A context-free operation can be performed on objects without regard to their current state. We can stamp a sphere template into the image without regard for the sphere's position or orientation. Its appearance is only a function of atom type. However, CPK spheres are only partially context-free. The visible portion of a sphere can be reduced by intersections with other spheres or by being obscured by other spheres. Thus, CPK spheres are *context-sensitive* to depth. The depth of templates previously stamped into the image must be taken into account.

A somewhat similar technique has been used to generate van der Waals dot surfaces for display on interactive devices.[7] A set of spheres of the proper radius and dot density is precomputed. For each atom to be displayed, a copy of the proper "dot template" is made and translated in X, Y and Z by the atom's coordinates. Points interior to the surface are culled by comparison with the dot surfaces of bonded atoms.

ALGORITHM

The context-free stamping algorithm operates on two primary data structures: a *template* and the *image*.

It is useful to visualize the template data structure as the forward-facing hemisphere of the van der Waals surface of the atom (see Figure 1). The template is actually stored as a matrix of cells containing both shading and depth information. These cell components are called $template_S$ *and* $template_Z$, respectively. Figure

*The term is borrowed from the theory of computation and refers to a certain class of rules for building strings from other strings.

				18 9	23 11	23 11	18 9				
		18 9	30 15	36 18	39 19	39 19	36 18	30 15	18 9		
	18 9	33 16	41 20	46 23	48 24	48 24	46 23	41 20	33 16	18 9	
	30 15	41 20	48 24	52 26	54 27	54 27	52 26	48 24	41 20	30 15	
18 9	36 18	46 23	52 26	56 28	57 28	57 28	56 28	52 26	46 23	36 18	18 9
23 11	39 19	48 24	54 27	57 28	59 29	59 29	57 28	54 27	48 24	39 19	23 11
23 11	39 19	48 24	54 27	57 28	59 29	59 29	57 28	54 27	48 24	39 19	23 11
18 9	36 18	46 23	52 26	56 28	57 28	57 28	56 28	52 26	46 23	36 18	18 9
	30 15	41 20	48 24	52 26	54 27	54 27	52 26	48 24	41 20	30 15	
	18 9	33 16	41 20	46 23	48 24	48 24	46 23	41 20	33 16	18 9	
		18 9	30 15	36 18	39 19	39 19	36 18	30 15	18 9		
				18 9	23 11	23 11	18 9				

Figure 2. Template with a radius of six cells. Depth values (scaled by 10) appear above shade values in each cell. A head-on light source is assumed

2 shows a template with a radius of six cells. In a preprocessing step, a template of the correct atomic radius is computed for each unique atom type.

The $template_S$ component can be determined by evaluating any applicable shading function. Since templates are precomputed, the use of more realistic shading functions is possible without concern for run-time cost. For example, specular reflections (highlights) can be provided with no run-time penalty. The $template_Z$ component can be derived as a byproduct of the shading calculation. During the image-generation process, this depth information is used to solve the hidden-surface problem.

The other major data structure is the *image*. The image is a matrix of cells equal in size to the generated output image. The structure of image cells is exactly analogous to template cells. The $image_S$ component holds a color value of the visible image, and the $image_Z$ component stores a depth value of the full-screen depth buffer.

The initial stages of the stamping algorithm are identical to Porter's algorithm.[2] Atomic coordinates are read, transformed, mapped to display coordinates and then clipped. Beyond this point, the methods diverge. The stamping algorithm proceeds atom by atom (rather than scanline by scanline), stamping the corresponding template for each atom into the image.

The stamping operation can be described as an analog of *BitBlt*.[8] BitBlt ("bit-boundary block transfer") is a bitwise operation performed on two rectangular bitmaps. One bitmap is the *source* for the operation; the other is the *destination*. The operation is controlled by one of 16 possible Boolean functions. For example, if the Boolean function is AND, a bit will appear in the

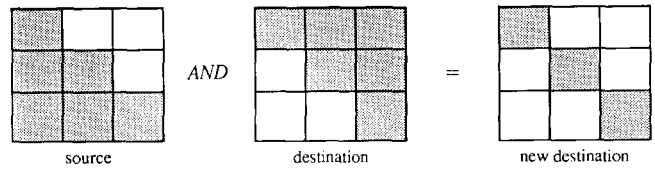

Figure 3. BitBlt: source AND destination = new destination

destination bitmap if and only if the corresponding source and destination bits are both set to 1. (See Figure 3.) BitBlt operations are used on workstations with bitmapped graphics and are often implemented in hardware.

SphereBlt also operates on rectangular source (template) and destination (image) arrays. The result is controlled by a function. However, the function is not Boolean, but rather the comparison operator ">": a $template_S$ value will appear in the image only if its corresponding $template_Z$ value plus the atom's depth is greater than the current $image_Z$ value at that location. The following program fragment (written in C) defines the operation to stamp an atom centered at X, Y and Z. The template radius is given by R.

```
for(x = 0; x < (2*R); x + +){
  for(y = 0; y < (2*R); y + +){
    if ((templateZ[x][y] + Z) > imageZ[X - R + x][Y - R + y]){
      imageS[X - R + x][Y - R + y] = templateS[x][y];
      imageZ[X - R + x][Y - R + y] = templateZ[x][y] + Z;
    }
  }
}
```

Values in the square but not the circle (the shaded areas of Figure 2) are implicitly ignored.

As can be seen, the SphereBlt operation is an adaptation of the classic *depth* (or Z) *buffer* algorithm. Determining visibility is based on the image space comparison of new and existing depth values. Computing each visible pixel involves only a comparison and two assignments. The CPK image-rendering bottleneck is thus shifted from the shading calculation to the memory accesses required to update the Z buffer and move pixel values into the frame buffer. Figure 4 shows an example of SphereBlt.

Note that there is no subpixel positioning; the center of the template copy must fall on a pixel center. Thus, atoms in the image may be slightly off-center from their true positions. The significance of this problem will be presented later.

IMPLEMENTATION

The speed of the SphereBlt algorithm permits interactive applications. Our current implementation is an interactive, menu-driven program called CpkTool. CpkTool is written in C and runs on a Sun Microsystems Sun-3/260c. Our configuration has 8 megabytes of main memory and Sun's optional CP2 Enhanced Graphics Processor.

The screen resolution of the color monitor is 1152 x 900 (81 pixels per inch). The frame buffer is 8 bits deep, offering 256 displayed colors from a palette of

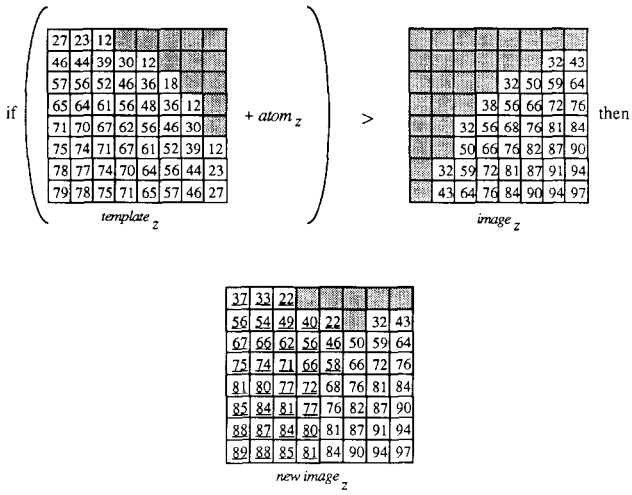

*Figure 4. SphereBlt: if (*template$_Z$ + atom$_Z$*)* > image$_Z$ *then* image$_Z$ = template$_Z$ + atom$_Z$ *(only the depth values are shown).* Atom$_Z$ *is assumed to be 10. New* image$_Z$ *values resulting from* template$_Z$ *are underlined*

16.7 million. Since we are limited in the number of colors that can be displayed at one time, the color map must be optimized; 31 shades are provided for each of 8 possible atom types. The image resolution of CpkTool is the largest square window that can be placed on the screen: 900 x 900.

CpkTool runs under the X Windows System,[9] or X, a device-independent network-transparent window system for bitmapped display workstations. Applications written for X can be run transparently on a variety of host machines and displays. While CpkTool would run faster if implemented with the Sun's native graphics routines, we anticipate porting the program to other local systems supporting X.

Before CpkTool can be run, the templates must be constructed. A utility program reads a set of shading parameters and builds a library of templates ranging in size from 5 to 80 cells in radius. The templates are stored in individual disk files. Note that if the shading parameters are changed (e.g., the light source moved), the library must be rebuilt.

The *templates$_S$* values range from 0 to 30. These values are determined by application of Lambert's cosine law, which relates the amount of reflected light at a point on a surface to the cosine of the angle between the sphere surface normal vector and a vector to the light source.[10] Specular reflections[11] are also supported. The *template$_S$* values will be mapped to color map addresses when the templates are read by CpkTool and assigned an atom type. The surface normal also provides the *template$_Z$* values. The Z component of the (unnormalized) normal vector equals the sphere depth at that point. This value is zero at the perimeter and equals the sphere's radius at the center of the template.

On startup, CpkTool selects a set of eight templates from the library based on the default window size ("zoom" factor).

The user can interact with the program in several ways. Among the more significant are:

(1) *Load a molecule.* Up to four molecules can be loaded at once. CpkTool reads atom types and coordinates from Protein Data Bank files.[12]

(2) *Delete a molecule.* This removes the selected molecule from the display.

(3) *Rotate or translate a molecule.* The selected molecule is rotated or translated independently of others. A transformation can also be applied to all molecules simultaneously.

(4) *Change the window size.* Zoom in or out. This command changes the size of spheres displayed; thus, a new set of templates must be read from the library.

(5) *Change the atom to color mapping function.* First, all atoms of the selected molecule are identically colored with a neutral shade. The user can then select and color individual atoms to highlight aspects of interest within the molecule. The user can select atoms by: (a) querying the database on the molecule's Protein Data Bank file (e.g., "color all phosphoryl oxygens red"), or (b) placing the Sun's mouse cursor over the atom of interest and pressing the mouse button.

The display must be redrawn whenever a command is issued that changes viewing parameters. Atoms are transformed, mapped to display coordinates, and then clipped. Atoms are sorted front to back by their Z center coordinate prior to stamping. Although the algorithm does not require this drawing order, it reduces the number of pixels written to the screen and, depending on the molecule and its orientation, can yield a substantial speed-up. After sorting, both *image$_S$* and *image$_Z$* are cleared and all atoms are stamped into the image as described above.

Note that if the user is selecting atoms of a molecule for highlighting (as in item 5 above), he need not redraw the entire molecule to display his selection. If *image$_Z$* is saved, the selected atom(s) can be redrawn over the existing image. Since a highlighted atom must overwrite itself, the SphereBlt function must be changed from ">" to ">= or =". This results in very rapid feedback to a particular color and atom selection.

The implementation described here is memory intensive. The main depth buffer (*image$_Z$*) consists of a 900 × 900 matrix of 2-byte values (1.62 megabytes). Frame buffer memory is used to store the visible image (*image$_S$*) on the Sun workstation. To approximate the amount of memory occupied by a set of eight templates, note that a window size of 50 × 50 Å (an effective image resolution of 18 pixels/Å) results in a template radius of 30 pixels for a carbon atom. There are 3 bytes per template cell: *template$_S$* is 1 byte and *template$_Z$* is 2 bytes. Assuming that the average template radius is 30 pixels, a set of eight templates would occupy 86.4 kilobytes. While this memory requirement appears substantial, the run-time usage is not a problem on a dedicated workstation with 8 megabytes of main memory.

CpkTool is currently being used for drug design, analysis of molecular structures and viewpoint selection for high-quality image generation.

Table 1. Worst-case stamping speeds

Window (Å)	Pixels/Å*	Carbon template radius	Atoms/second
60 × 60	15	25	30.9
50 × 50	18	30	23.5
40 × 40	22.5	38	15.4
30 × 30	30	51	9.1
20 × 20	45	77	4.2

*Assumes 900 × 900 image size.

Table 2. Speed depends on *Z* sort and molecular orientation

DNA orientation/sort	Atoms/second
Perpendicular to long axis	
Z sorted	50
not *Z* sorted	36
Parallel to long axis	
Z sorted	52
not *Z* sorted	47

Tim van Hook and Doug Schiff of Sun Microsystems implemented an algorithm similar to the one described above. Their implementation was designed to run on Sun's TAAC–1 Application Accelerator add-on board. The TAAC-1 provides a powerful processor operating on an additional 8 megabytes of memory. Their implementation was microcoded and can stamp up to 1000 spheres per second at a resolution of about 6 pixels/Å.

CONCLUSIONS

Table 1 shows the speed of the above implementation at various degrees of resolution. In this worst-case scenario, the values for atoms per second assume that *every* part of every sphere is visible. Stamping is substantially faster for molecular data (given sphere intersections and overlap). Table 2 illustrates these points and shows how the speed is dependent on the *Z* sort and the orientation of the molecule. The values were derived from 536 atoms in 15 base pairs of DNA at a resolution of 15 pixels/Å.

The image quality is quite good, as can be seen from Color Plates 1 and 2. Color Plate 1 is an image of cyclophosphamide (an anticancer drug), which was generated in 5 seconds at a resolution of 37.5 pixels/Å. In contrast, a standard implementation of Porter's algorithm required 45 seconds to generate this image. Color Plate 2 is an image of an anticancer drug metabolite (phosphoramide mustard) in alignment with one covalent attachment site in a 15 base pair fragment of DNA. This image was generated in 26 seconds at a resolution of 19.56 pixels/Å. Porter's algorithm took 267 seconds to render the molecule in Color Plate 2 at the same resolution.

Unlike images produced by Porter's algorithm, stamped sphere images suffer from aliasing artifacts (jagged edges) at both the sphere perimeters and the edges created by intersections with other spheres. The cause is inherent to the algorithm: Templates are derived from spheres sampled at display resolution. Antialiasing by postfiltering (subpixel averaging) requires that objects be sampled at a resolution higher than the final display resolution. However, as we will see below, there are antialiasing techniques that can produce high-quality images using stamped templates.

A more subtle aliasing effect is introduced by the lack of subpixel positioning for templates. Stamped spheres can be off from the true atom center by as much as $\sqrt{2}s/2$ Å,* where s is the effective resolution in angstroms per pixel. For $s = 0.0666$ Å/pixel, a template can be off by up to 0.047Å. This poses no problems for static images. However, if this algorithm is used to generate frames for an animated sequence, the aliasing would be notable. Atoms would appear to "jump" from pixel center to pixel center if a molecule is moved slowly enough. During a rotation, atoms near the rotation axis would not move smoothly compared to atoms farther away. This *temporal aliasing* can be disconcerting in an animated sequence.

There are several ways to improve the algorithm in terms of speed and image quality. A logical approach to speeding up the algorithm is to attack the next image-generation bottleneck. The limiting factor in stamping speed is memory bandwidth both to and from the *Z* buffer ($image_Z$) and to the frame buffer ($image_S$). A significant speed-up would result from the use of hardware *Z* buffer support and/or graphics processors tightly coupled to frame buffer memory. Note that the TAAC-1 implementation mentioned above takes this approach.

Since BitBlt operations are often implemented in hardware, an analogous consideration of hardware support for SphereBlt seems appropriate. Texas Instruments produces a graphics processor that has an enhanced BitBlt facility called *Pixblt*,[13] and in all probability this could be harnessed for SphereBlt.

The aliasing artifacts introduced by this algorithm can be mitigated by two techniques. Both methods use an interpolation scheme between a newly computed foreground color and the current background color. The atoms must be sorted and stamped from back to front so that the correct background is available for the interpolation. This leads to an increase in image-generation time (in addition to interpolation expense), since pixels that will eventually be overwritten are displayed.

For a sphere's perimeter edge, we can use a method developed for compositing antialiased digital images.[14,15] Each pixel is treated as covering a finite area and the amount of (pixel) area covered by objects is saved. This *alpha* value typically ranges from 0 to 1 and is stored along with color. During rendering, the

*This is the distance from the center of a pixel to any corner of the pixel.

alpha value is used as an interpolation factor between foreground and background colors. When the atom templates are precomputed, alpha values are computed for the pixels on the sphere perimeter by sampling these areas at a resolution higher than display resolution. The average of the pixel area samples can be used at run time to interpolate between the template edge color and the existing background color.

The edges caused by sphere intersections pose a different problem. Porter[2] developed a heuristic for blending the foreground with the background color whenever the difference between the corresponding depth values was less than some predetermined distance. This method can also be employed in the algorithm presented here, since background depth values are saved in $image_Z$.

A useful enhancement for CpkTool could include *adaptive refinement* methods.[16] Adaptive refinement techniques keep improving the image — provided that the viewing parameters are not altered. If these parameters are changed, the displayed model reverts to a more primitive form that can be displayed in real time. During transformations, molecules can be depicted as line drawings (as on a vector display) or as ball-and-stick drawings (using lines and reduced radius templates). When the user stops these transformations, the system progressively improves the image until the next transformation occurs.

We have presented a new algorithm for rapidly generating high-quality CPK images that permits facile interactive analysis of molecular structure. The CPK image-rendering bottleneck has been shifted from the shading calculation to the memory bandwidth problem of updating the Z buffer and moving pixel values into the frame buffer. The implementation we have described is approximately an order of magnitude faster than commonly used methods, and with future improvements, it will yield images of equal quality.

ACKNOWLEDGEMENTS

We thank Dr. Frederick P. Brooks, Jr., Helga Thorvaldsdottir, and Marc Levoy of the University of North Carolina at Chapel Hill for their invaluable support for this project. One of us (T.C.P.) received financial support under NIH Grant RR-02170-04 while a graduate research assistant at Chapel Hill.

F.H.H. acknowledges the support of an American Cancer Society Fellowship Award and a National Scientific Research Award.

We thank Doug Schiff of the NCAA Group of Sun Microsystems for information about the TAAC-1 implementation and for pointing out the BitBlt–SphereBlt analogy.

We also thank our colleagues at the Advanced Scientific Computing Laboratory for reviews and encouragement.

This research was sponsored, at least in part, by the National Cancer Institute, DHHS, under contract NO1-CO-23910 with Program Resources, Incorporated. The contents of this publication do not necessarily reflect the views or policies of the DHHS, nor does mention of trade names, commercial products or organizations imply endorsement by the US Government.

REFERENCES

1 Max, N. L. Computer representation of molecular surfaces. *J. Mol. Graph.*, 1984, **2**, 8–13
2 Porter, T. Spherical shading. *Comp. Graph.* 1978, **12**, 282–285
3 Bresenham, J. E. An incremental algorithm for digital display of circular arcs. *Comm. ACM*, 1977, **20**, 100–106
4 Pique, M. E. *Display of Space-Filling Molecular Models.* Technical Report TR83-004, UNC Computer Science Department, 1983
5 Max, N. L. ATOMLLL - atoms with shading and highlights. *Comp. Graph.*, 1978, **12**, 384
6 Fuchs, H. *et al.* Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. *Comp. Graph.* 1985, **19**, 111–120
7 Bash, P. A. *et al.* Van der Waals surfaces in molecular modeling: implementation with real-time computer graphics. *Science* 1983, **222**, 1325–1327
8 Foley, J. D., and Van Dam, A. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading, MA, 1984
9 Scheifler, R. and Gettys, J. The X Window System. *ACM Trans. Graph.*, 1986, **5**, 79–109
10 Rogers, D. F. *Procedural Elements for Computer Graphics.* McGraw-Hill, New York, 1985
11 Phong, B. T. *Illumination for Computer Generated Images.* Ph.D. thesis, University of Utah, Salt Lake City, 1973.
12 Bernstein, F. C. *et al.* Protein data bank – computer-based archival file for macromolecular structures. *J. Mol. Biol.*, 1977, **112**, 535–542
13 Asal, M. *et al.* The Texas Instruments 34010 graphics processor. *IEEE Comp. Graph. and Appl.*, October 1986, 24–39
14 Porter, T. and Duff, T. Composing digital images. *Comp. Graph.* 1984, **18**, 253–259
15 Carpenter, L. The A-buffer, an antialiased hidden surface method. *Comp. Graph.*, 1984, **18**, 103–108
16 Bergman, L. *et al.*, Image rendering by adaptive refinement. *Comp. Graph.*, 1986, **20**, 29–37