Docktool: A Dynamically Interactive Raster Graphics Visualization for the Molecular Docking Problem

TR87-013

1987

Thomas Cheek Palmer

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

Docktool: A Dynamically Interactive Raster Graphics Visualization for the Molecular Docking Problem

by

Thomas Cheek Palmer

A Thesis submitted to the faculty of The University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science.

Chapel Hill

1987

Approved by Frederick P. Brooks

Advisor: Dr. Frederick P. Brooks

[illegible]

Reader: Dr. Michael Cory

[illegible]

Reader: Dr. Henry Fuchs

THOMAS CHEEK PALMER. Docktool: A Dynamically Interactive Raster Graphics Visualization for the Molecular Docking Problem (Under the direction of Dr. FREDERICK P. BROOKS, JR.)

ABSTRACT

This thesis describes an experimental raster graphics visualization that assists in solving the molecular biologist's ***docking problem***. The docking problem can be defined as the attempt to determine whether a given conformation of a small molecule will *fit* into the active site of a macromolecule. The visualization is designed to allow as much perceptual power as can be accomplished on a standard color workstation while allowing interactive docking. This thesis limits itself to geometric fit of objects whose shapes do not change as they are docked.

Perceptual problems unique to current visualizations of molecular surfaces are discussed in terms of structural perception and depth perception.

The macromolecule is represented as a surface of space-filling spheres. The system provides unique interaction capabilities that are based on the data structure used to store this surface. For example, the user can clip through the surface to reveal hidden structure or make the surface transparent to view the stick-figure bonds underneath. This data structure is a three-dimensional ***depth buffer***. The interactive speed results from solving the hidden-surface problem in a preprocessing step and storing the output in the depth buffer.

The small molecule is represented as a stick-figure color coded by atom type. This simple depiction permits real-time transformations. In addition, the depth buffer provides a lookup table which we can use to determine if the small molecule is *bumping* the macromolecule. Bump checking is performed in real-time with both visual and sound feedback.

To My Parents

ACKNOWLEDGEMENTS

I wish to thank Dr. Frederick P. Brooks Jr. for continuous support, inspiration, and for creating an exciting and stimulating environment in which to work and learn.

I would also like to thank the other members of my committee, Dr. Michael Cory and Dr. Henry Fuchs, for valuable suggestions, comments, and corrections.

I thank Michael Pique and Richard Holloway who provided pieces of code, Doug Schiff for pointing me towards enhanced spheres, Lee Westover and Doug Schiff for reviewing early drafts of this thesis, and Helga Thorvaldsdottir for support and encouragement.

I would also like to thank our collaborators at Burroughs Wellcome Company: Michael Cory, James Bentley, and Lee Kuyper. They provided data and tried early versions of the system.

I also thank Sally for her support, encouragement, and patience.

This work was supported under NIH Grant RR-02170-04.

Table Of Contents

Chapter 1
Introduction

This paper describes an interactive computer graphics system using a raster visualization to assist in solving the molecular biologist's ***docking problem.*** The docking problem consists of determining whether a given geometric, electrostatic, and hydrophilic/hydrophobic conformation of a small molecule will ***fit*** into the active site of a macromolecule. Fit is usually determined by energy minimization techniques. This thesis limits itself to geometric fit of objects whose shapes do not change as they are docked.

Chapter 2 discusses docking and the various problems associated with current hardware and techniques.

Chapter 3 presents the DOCKTOOL strategy for visualizing and interacting with macromolecule and small molecule.

Chapter 4 is a description of the major algorithms used in the DOCKTOOL system.

Chapter 5 discusses algorithmic complexity, problems with the visualization, and suggested improvements.

Chapter 2
The Docking Problem

The docking problem is of vast importance to both molecular biologists and molecular computer graphics specialists. Molecular biologists are interested because of the tremendous scientific and practical benefits of understanding how small molecules bind to large molecules and how this interaction alters the behavior of the large molecule. Molecular graphics specialists are interested in the complex perceptual and computational problems involved. In the following chapters I will refer to the small molecule as the *ligand* and to the large molecule as the *macromolecule.*

2.1 Current Approaches

Current methods of docking using interactive computer graphics typically employ a color vector display device and represent the ligand and macromolecule as stick-figure bonds color coded by atom type. The macromolecule may also be augmented by a molecular surface depicting the active site. A molecular surface is typically displayed as a *dot surface* [Langridge *et al.*, 1981]. A set of dots corresponds to either the van der Waals surface or the *solvent-accessible surface* [Connolly, 1981] of the molecule. Both ligand and macromolecule can be independently rotated and translated. Some docking systems permit dynamic conformational changes to the ligand.

In one technique, the ligand (a real or hypothetical compound), is "flown" into the active site region and a suitable orientation is found with respect to the active site molecular surface. The orientation is found based on the shape of ligand and active site and the chemist's past experience with both. Chemical complementarity is also used to help match the ligand to the site. Functional groups that can interact chemically and have the correct shape are matched. The chemist then iteratively adjusts the conformation of the ligand and runs an energy-minimization algorithm on the complex. When he is satisfied, the ligand coordinates, position, and orientation are written out.

Another docking technique is *superposition.* A chemical group has a known position and orientation within the active site. The chemist constructs a hypothetical compound using the known group as a building platform and frame of reference. He then proceeds as before: iteratively adjusting the ligand conformation and energy-minimizing [Kuyper, 1987].

2.2 Perceptual Problems

Humans use perceptual cues to form, and then either accept or reject, perceptual hypotheses about objects within their environments. These hypotheses are evaluated within a context of learned and innate responses to stimuli. The more experience one has in perceiving an environment, the more accurately his perceptions will reflect the physical reality of that environment. Hypotheses are also more likely to be correct if perception is *intermodal*, *i.e.*, relying on the collaboration of multiple cues and senses [West, 1986].

Chemists experienced with standard computer graphics visualizations of molecules bring a crucial perceptual context to the docking problem. In addition, some visualizations have real-world counterparts to provide context, *e.g.*, the color coding correspondence of atoms in both plastic CPK models (first used by Corey, Pauling, and Koltun), and computer generated CPK images (spheres representing the van der Waals radius of atoms).

Three-dimensional perception in computer graphics is a difficult problem even with this context. The perceptual conflict arising from the use of two-dimensional displays is a major stumbling block. The shading of computer generated objects tells us we are seeing a three-dimensional image. However, both eyes receive the same image and, as we move our heads, the image remains the same. If we use traditional display hardware we must provide as many visual cues as possible to overcome this conflict.

From the computer scientist's point of view, the docking problem is a perception problem. The chemist needs *structural perception* to remind him of constraints arising from size, shape, and type. He requires *depth perception* to determine the relative positions and orientations of the ligand molecule and the macromolecule.

Structural perception and depth perception are interdependent. In particular, size cannot be determined without knowledge of distance (in the absence of other cues). In the sections below, cues are classified as being either structural cues or depth cues. In reality, the distinction can be vague.

2.2.1 Structural Perception

Structural perception is the identification of the size, shape, and type of objects in the visual field.

General Cues to Structural Perception. Structural perception is strongly influenced by learned and innate responses. In familiar environments we rarely need visual clues to form percepts. However, when faced with unknown objects in unfamiliar environments we must rely on visual cues.

The human visual system is particularly good at perceiving contours created by abrupt changes in color or brightness. The perception of a contour against a background is a powerful indication of shape. We must perceive a contour before we can perceive form [McBurney and Collings, 1977]. A related cue is *occlusion*, or the covering of one object by another. This can provide structural information about both foreground and background objects.

Shadows and shading provide cues to structure and type. The shape of an object's shadow as it falls across another object can reveal much about both. Color, luminance, and reflectance are crucial in determining surface properties of objects. A shiny gold ring is perceived as being hard, while a field of ripe wheat is not.

Structural Perception in Molecular Graphics. In molecular graphics, structural perception is the identification of the size, shape, and type of various chemical units such as atoms, groups, residues, and molecules. In chemistry, "shape" can be thought of as the strength, charge, and extent of electron clouds surrounding the atomic nucleus.

Occlusion provides clues to shape. Two interpenetrating CPK spheres hide parts of each surface and indicate that the atoms are bonded. However, in the absence of depth cues the image may be ambiguous. The atoms represented by the spheres may appear bonded only because our line of sight nearly coincides with a line through the two sphere centers, and not because they are close in the model. †

The shading attributes of objects are typically used to color-code chemical units by type. Color can also be used to carry other chemical data such as temperature or secondary structure type. *Relative fading*, or the selective fading of objects relative to others, can give cues to structure or type by emphasizing certain displayed elements [Lipscomb, 1981].

A good cue to position and shape unique to interactive molecular graphics is to provide *bump-check* feedback to the user. This feedback gives an indication of where two non-bonded atoms have mutually interpenetrating van der Waals surfaces.

† An interesting analogy is found in astronomy. Pairs of stars that interact gravitationally are called double stars. Pairs merely aligned with earth but far apart in space are also termed double; they are more commonly called *optical* double stars.

2.2.2 Depth Perception

Depth perception is the identification of the distance of objects in the visual field from each other and from the observer.

General Cues to Depth. Depth cues fall into three categories: muscular cues, binocular disparity, and monocular cues.

There are two sources of muscular cues. *Convergence* cues come from the rectus muscles that control the convergence of the eyes as we look at objects at various distances. *Accommodation* cues result from the ciliary muscles that control the shape of the lens as we focus. These cues can provide reliable depth information for objects closer than thirty feet [West, 1986].

The interocular distance gives rise to depth perception through binocular disparity. We build a three-dimensional image in our brains from the disparate images provided by our two eyes. The perception results from the cortical fusing of the two images. The exact mechanism is poorly understood; nevertheless binocular disparity is a very powerful cue to depth.

Monocular cues to depth are varied, but most fall into three categories: perspective, interactive, and shading. Perspective cues include the relative size of objects, linear perspective (*e.g.*, the apparent convergence of parallel lines), and texture gradients. Interactive cues include head motion parallax, the kinetic depth effect, and occlusion. Shadows, color, and lighting fall into the shading category.

Depth Perception in Molecular Graphics. We are constrained by graphics display devices in our attempts to provide depth perception to a user. The lack of a true three-dimensional display device rules out muscular cues and head motion parallax.† Perspective projections are rarely used in molecular graphics. Wright found that perspective was confusing in images of stick-figure molecules. Bonds far away seem shorter, not distant. He explained this by noting the general lack of parallel lines in molecular stick-figure images [Wright, 1972]. This rules out the monocular perspective cues in images containing stick-figure bonds.

Various mechanisms exist to present each eye with a different, slightly sheared image and thereby provide cues via binocular disparity. The effect is powerful and widely used in molecular graphics. See [Lipscomb, 1981] for various methods.

† A *varifocal* mirror can provide a true three-dimensional image. See [Traub, 1967], [Rawson, 1969], or [Fuchs *et al.*, 1982].

Shading cues to depth can be realized on traditional raster display devices. Intensity depth cueing can be a powerful indication of depth. Darker features with less contrast are perceived as being more distant than brighter, sharper features. Transparency can indicate depth on raster displays by modulating foreground color with background color. Another cue is the direction of the light source. We are used to seeing objects lit from above. We perceive reversed depth when objects are lit from below (in the absence of other cues).

Interactive cues to depth (with the exception of head motion parallax), can be provided using standard display devices. The ***kinetic depth effect*** [Wallach and O'Connell, 1953] provides depth perception through motion, especially rotational motion. This effect is analogous to head motion parallax (although weaker) in that an action by the observer causes a perceptual change that is associated with the action. It is interesting to note that while rotation of a sufficiently complex object always provides depth information, the perceived depth and direction of rotation are ambiguous in the absence of other cues, particularly perspective projection and intensity depth cueing [Braunstein, 1976]. Unless the viewer has some knowledge about the structure of objects he is viewing, perspective without intensity depth cueing does not necessarily eliminate rotational ambiguity.

Another interactive cue to depth is the motion of a clipping plane along the viewing axis of the display. Objects closer to the user disappear first as the plane is moved farther away. A typical technique with vector displays is to use near and far planes and clip the image to a small slab around the area of interest [Langridge *et al.*, 1981]. However, this may be motivated more by the desire to reduce visual clutter than the need for depth cues.

As solid objects move relative to one another, closer objects occlude those behind. Occlusion is also effective in static images as a cue to depth. Seeing a continuous contour superimposed on a broken one is a powerful indication that the continuous contour is closer.

It is also possible to increase depth perception by removing cues to flatness, thereby shifting the tide of the perceptual battle in favor of depth. The most significant technique is to shield the display. This reduces the perceptual conflict arising from the three-dimensional environment surrounding the two-dimensional display. Closing one eye eliminates the strong binocular disparity cue that tells us we are looking at a flat screen. If we restrict head motion we deny the user the ability to use head motion parallax to explore the image.

2.2.3 Perceptual Problems With Vector Graphics Visualizations

Vector display devices are in wide use by the molecular graphics community. They can provide real-time transformations of several thousand vectors, intensity depth-cueing, and color. Excellent stereo mechanisms are also available. However, there are a number of perceptual problems with vector graphics visualizations of molecular surfaces.

Molecular surfaces represented with dot surfaces have a serious flaw: no structure is hidden. This visualization is rarely encountered in the real world and makes it difficult for a user to maintain a structural model in mind. Depth cues are provided by intensity depth cueing and stereo mechanisms, but many users clip the image to a thin slab around the area of interest. This results in a gain in local depth perception at the loss of global structural perception.

2.3 Computational Problems

2.3.1 Transformations

Vector display devices capable of real-time transformations of thousands of vectors have been in existence for years. Raster devices with Bresenham's line drawing algorithm encorporated in hardware are capable of transforming thousands of vectors in real-time.

On the other hand, real-time transformations of spheres on a raster display are very difficult. However, under restricted circumstances or with special hardware, real-time transformations are possible. Michael Pique was able to approach real-time update rates for up to fifty spheres by taking advantage of head-on lighting and a fast microprocessor attached to a frame buffer [Pique, 1983]. ***Pixel-planes*** is a processor-per-pixel raster device capable of displaying over 13,000 spheres in real-time. [Fuchs *et al.*, 1985].

2.3.2 Energy Calculations

Energy calculations are an important part of a docking system. Unfortunately, good models of the potential energy of molecules do not exist. Current methods are iterative and extremely CPU intensive. Minimization algorithms are essentially $O(n^2)$ in complexity, where n is the number of atoms.

A straight distance check basically implements a van der Waals bump check. Although it is still $O(n^2)$, the distance check is much simpler and has been implemented in hardware. Evans and Sutherland produces an *Energy Co-processor* board that computes atomic distances (amoung other things). Evans and Sutherland claims that 50,000 atom pairs can be processed at a rate of ten per second.

Chapter 3
DOCKTOOL Strategy

3.1 Overview

The DOCKTOOL strategy is to apply raster graphics algorithms and hardware to the docking problem. With this decision, we gain the ability to supply the user with additional perceptual cues unique to raster images. The cost is the restriction of some object transformations.

DOCKTOOL uses shaded CPK spheres to represent the macromolecule. The spheres give perceptual cues via shading, occlusion, and intensity depth cueing. We lose the ability to transform the macromolecule in real time, but gain some interesting capabilities based on the data structure used to store the spheres.

At a minimum, a useful docking tool must provide rotations and translations of the ligand molecule. The ligand is represented with a visualization common to vector displays: bonds as short lines color-coded by atom type. The ligand is superimposed on the macromolecule's spheres. This ligand visualization allows perceptual cues via the kinetic depth effect, intensity depth-cueing, bump checking, and occlusion (when ligand bonds are hidden by the macromolecule). Figure 3.1 shows Trimethoprim (the ligand) superimposed on *E. Coli* Dihydrofolate Reductase (the macromolecule).

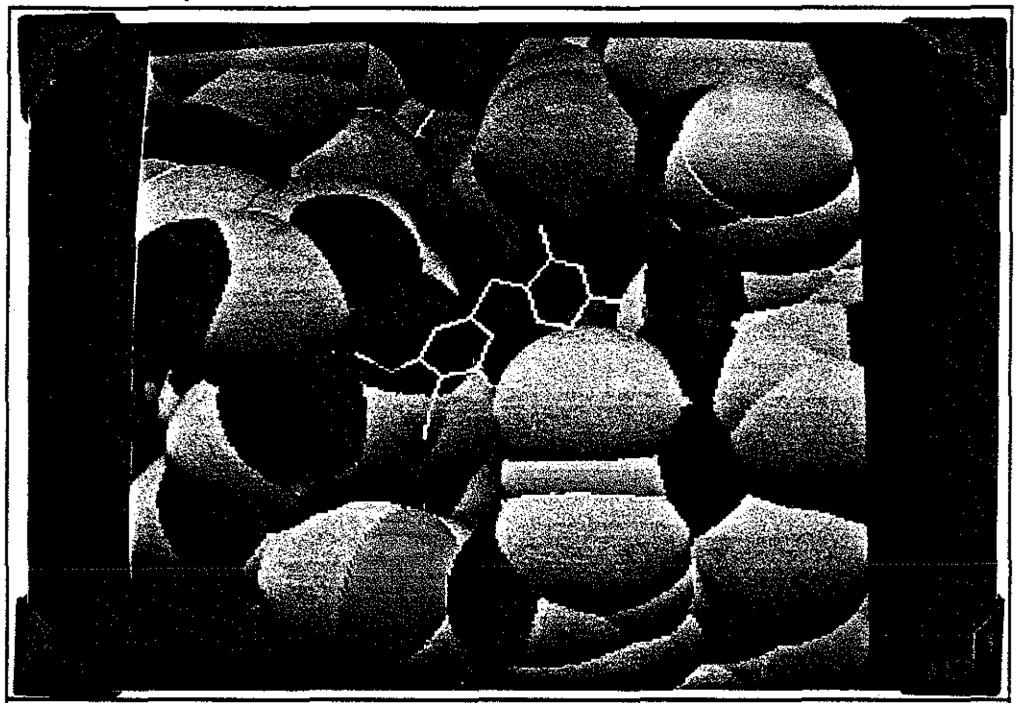

Figure 3.1: Docktool

The purpose of DOCKTOOL is to provide an environment for experimentation with a raster visualization for the docking problem. It is not within its domain to provide a fully functional docking system. Several desirable features are not implemented. DOCKTOOL is a rigid-body docking system, *i.e.*, the ligand molecule cannot be changed during a session. No energy feedback is given, either via dynamically minimized ligand conformation or numerical energy data.

3.2 The Macromolecule

One common method of storing three-dimensional data is the *voxel.* In this technique, space is partitioned in X, Y, and Z. The resulting cubic volumes are called voxels (a contraction of volume element) and a data value is associated with each. Sources of voxel data include medical imaging studies and electron density of molecules from X-ray crystallography. In each case the value is a measure of density and there is no underlying structure explicit in the set of voxels.

Another three-dimensional data structure is the *depth buffer.* A depth buffer is similar to voxel-based structures but differs in one important way. The depth buffer's counterpart to a voxel is a depth element or a *dexel* [Van Hook, 1986]. While a voxel has attributes based on its postion in space, a dexel has attributes based on the object that created it.

If we imagine a hollow sphere stored in a voxel-based structure, we see that the set of voxels that intersect the sphere surface is a very small percentage of the total. A depth buffer storing the same sphere would store a small subset of the voxel-based structure: the set of non-empty voxels. Therefore, if we are generating the three-dimensional data structure from modelling primitives (and we are concerned about storage restrictions), a depth buffer is preferable to a voxel structure. DOCKTOOL uses a depth buffer to store the macromolecular surface.

The depth buffer consists of a two-dimensional matrix of *depth vectors* in the X, Y plane of the screen. Therefore, each depth vector is associated with a unique X, Y screen location. The set of dexels occuring at a given X, Y location make up the depth vector at X, Y in the depth buffer.

A dexel inherits an X, Y location based on the depth vector that contains it. However, when we took a subset of the voxel-based structure to create our depth buffer, we lost Z information that was implicit in the voxel's position. To remedy this, each dexel must store the Z value of the object that created it.

The use of a depth buffer rather than a voxel-based structure is a time-space trade-off. A typical DOCKTOOL depth buffer is about 1% of the space required for a voxel-based structure. On the other hand, a voxel can be accessed by a simple table lookup using its X, Y, and Z coordinates. Dexels require a lookup in X, Y (to find the depth vector) and a search along the depth vector for the appropriate Z value.

Algorithms operating on depth buffers provide unique interactive capabilities. (The algorithms are detailed in Chapter 4.) We can clip through the molecular surface to reveal hidden structure. The molecular surface can be made either transparent or completely invisible to show the bonds underneath. Ligand atoms can be checked against the depth buffer to provide real-time bump checking. Interactive speed is achieved by solving the hidden-surface problem in a preprocessing step and storing the results in the depth buffer.

These capabilities come at some cost. Depth buffer creation is time-consuming and must be performed in the preprocessing step prior to interactive use. Since the viewpoint must be chosen prior to preprocessing, interactive macromolecular rotations are ruled out. We can, however, initially orient the macromolecule to provide a good view of the active site.

DOCKTOOL uses *enhanced radius* CPK spheres. Each sphere has 1.57Å added to its van der Waals radius. This value is approximately equal to the van der Waals radius of nitrogen and lies halfway between carbon and oxygen. Enhanced spheres are used for three reasons. First, when dexels interior to other spheres (from bonded atoms) are removed, the remaining dexels form an enclosing shell representing the molecular surface. This reduces both depth buffer size and visual clutter. Second, it allows us to model the molecular surface with spheres rather than more complicated modelling primitives (such as bicubic patches). Third, as will be explained in Chapter 4, the enhanced surface works in concert with the stick-figure ligand to provide real-time bump checking.

3.3 The Ligand

The ligand molecule is represented as a stick-figure. Each line in the stick-figure represents the bond between two bonded atoms. Each half of the line is color-coded by the type of atom at that end of the bond. This simple depiction permits real-time rotations and translations superimposed on the macromolecule.

The enhanced surface of the macromolecule provides the van der Waals surface for the ligand. The result is that, in bump checking, the effective surface of the ligand is equivalent to a CPK surface of 1.57Å radius atoms making up the ligand. Recall that the degree of macromolecular enhancement is 1.57Å. See Figure 3.2. The dotted spheres represent the true van der Waals radii of the two atoms.

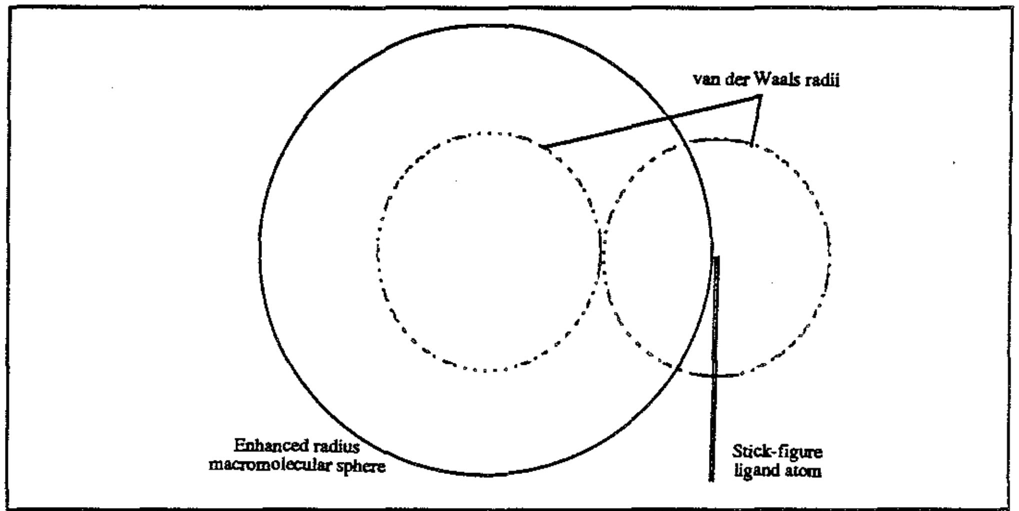

Figure 3.2: Enhanced Spheres.

3.4 Previous Work

Peter Atherton describes an image processing system for CAD applications based on a three-dimensional depth buffer [Atherton, 1981]. His specification permits clipping, transparency, translucency and shadows. He describes an implementation and discusses a realization based on a fast graphics engine with access to a large frame buffer.

Michael Connolly describes depth buffer algorithms for molecular modelling [Connolly, 1985]. His goal was to be able to preprocess molecular visualizations into a depth buffer and then change any of several parameters to quickly generate a new image.

Tim Van Hook uses a frame buffer algorithm utilizing linked lists of surfaces to implement real time display of numerically controlled milling operations [Van Hook, 1986]. Van Hook coined the term *dexel.*

B. Lee and F. M. Richards were the first to use enhanced van der Waals radius atoms [Lee and Richards, 1971]. They studied the *static accessibility* of a protein's surface and internal cavities to solvent molecules. The term static is used because "no account has been taken of potential flexibility or movement of groups in the structure." Their enhancement factor was 1.4Å.

C. D. Barry was the first to use enhanced radius spheres in conjunction with a stick figure for studying docking problems. [Barry, 1980].

Chapter 4
Algorithms

4.1 System Overview

The DOCKTOOL system consists of two main pieces of software and several utility programs. The *Preprocessor* uses a scanline, *Z*-buffer algorithm to solve the hidden surface problem and build the three-dimensional depth buffer of macromolecular surface and bonds. *DockTool* is the interactive program that provides molecular surface clipping and transparency, ligand transformations, and bump checking. It consists of a host process and a graphics satellite process. The host code handles files and devices and manages the user interface. The satellite portion implements *frame-buffer algorithms*, ligand transformations, and bump checking. These algorithms are termed frame-buffer algorithms since they operate on the depth buffer loaded into frame-buffer memory (Figure 4.1).

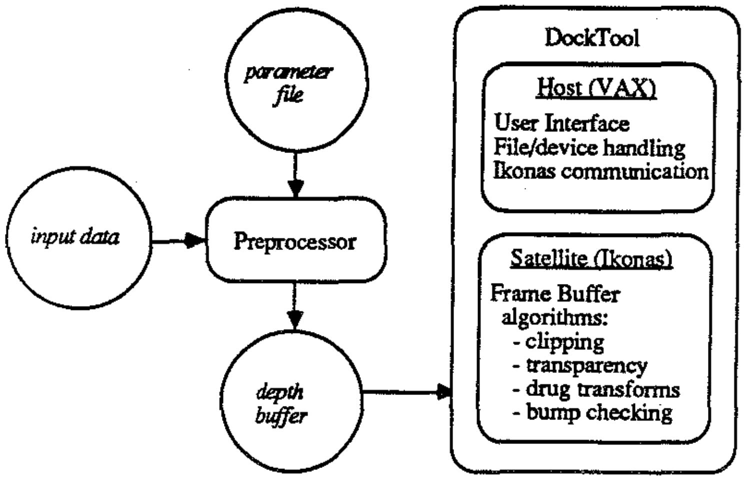

Figure 4.1: System Overview.

The *DockTool* host system is a VAX-11/780 running the 4.3 BSD UNIX operating system. The graphics satellite display is the Adage-Ikonas RDS-3000. The satellite code is written in GIA2 [Bishop, 1982], a microcode compiler with C-like syntax. The microcode runs on the AMD 2900 based microprocessor in the Ikonas. The AMD 2900 instruction cycle time is 200 nanoseconds.

4.2 The Preprocessor

The *Preprocessor* is an extension of Porter's spherical shading algorithm for CPK images of atoms [Porter, 1978]. The extensions include handling of both front and back atomic hemispheres, stick-figure bonds, and a variable-position infinite light source.

In Porter's algorithm, atoms are first sorted to scanline order and then sorted from back to front. He then uses Bresenham's circle algorithm [Bresenham, 1977] twice: once to trace the atom's circular silhouette in the X-Y plane and once at each scanline to trace the front facing semi-circular arc of each atom in the X-Z plane. The semi-circular arcs are scanned into a frame buffer, overwriting previous values if the depth of the new value is less than the current depth.

To produce depth buffers we must save, rather than overwrite, previously computed shades and Z values. A growing depth vector is maintained at each element of the scanline. For a pixel of a semi-circular arc that evaluates to element X in scanline Y, we load that pixel's attributes into a dexel and insert the dexel in increasing Z order into the depth vector at X.

To create a three-dimensional molecular surface, we must also include the back-facing hemisphere of each atom in the depth buffer. This is accomplished by reflecting each front-facing semi-circular arc around the atom's Z center. The reversed arc is then scanned into the depth buffer. This technique is crucial to bump checking. Each front surface dexel must have a matching back surface dexel, as will be seen below.

The *Preprocessor* reads a parameter file on startup that controls its operation. Parameters can be supplied that control light direction, viewpoint, display window, depth cueing, and degree of sphere radius enhancement. The User's Manual in Appendix A describes these parameters in detail.

Sphere shading is assigned based on atom type and a simple illumination model incorporating Lambert's cosine law. Intensity depth-cueing is computed with linear attenuation as described in [Rogers, 1985].

The *Preprocessor* also renders the macromolecule's bonds. The bonds are represented as stick-figures color coded by atom type. A modified three-dimensional Bresenham line drawing algorithm is used to render the lines into the depth buffer and compute the dexel attributes for the bonds [Kaufman and Shimony, 1986]. The voxel-based algorithm assumes that the X, Y, and Z resolution is identical. That is not the case in the *Preprocessor.* X and Y are scaled from 0 to 255 and Z is scaled from 0 to 32,767. A bond parallel (or nearly parallel) to the Z axis could generate thousands of dexels if we treated Z as scaled the same as X and Y. We take a sample of the potentially generated voxels; with sample size based on the relative scaling between X and Z.

After all pertinent atoms and bonds have been processed for a scanline, each depth vector is post-processed to cull "unneeded" dexels. An unneeded dexel is one lying between matching front and back surface dexels. Figure 4.2 provides a graphic view as the depth vector at X_i is being post-processed.

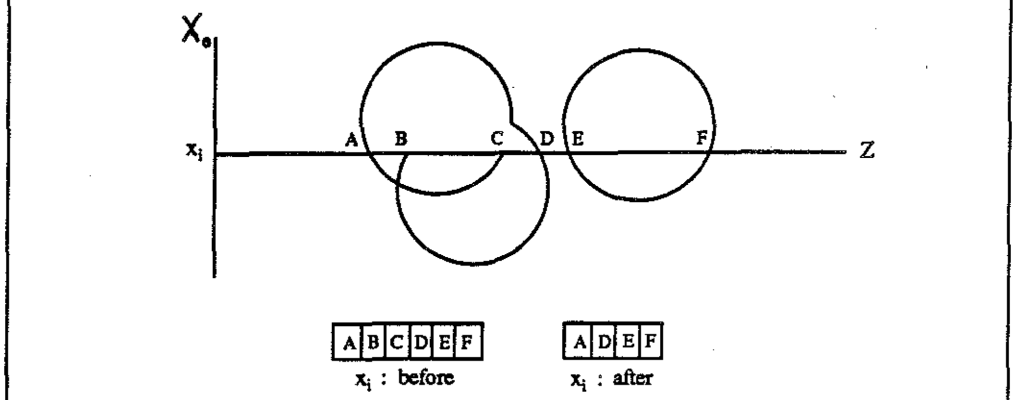

Figure 4.2: Depth Vector Post-Processing.

Depth vectors from X_0 to X_{i-1} have already been post-processed. Dexel B will be marked for removal since it lies between dexels A and C. Dexel C will then be marked for removal since it lies between dexels B and D. At that point, dexels A and D become the matching front-back pair. (In Figure 4.2, front dexels are to the left.) After post-processing, the depth vector is written to a binary output file (minus those marked for removal). Each depth vector is preceded by a count of the dexels in the vector.

The X, Y, Z resolution is 256 by 256 by 32,768. Dexels are packed into 24 bits to match frame buffer pixel size. The dexel partitioning is shown in Figure 4.3.

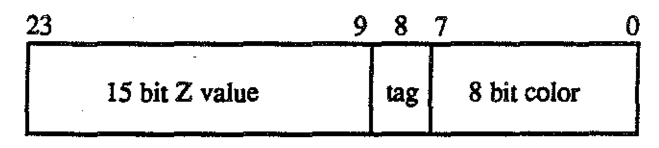

Figure 4.3: Dexel Partitioning.

The Z values are stored in the most significant bits to permit comparisons without masking other attributes. The tag is used by frame-buffer algorithms to distinguish between objects; the current implementation distinguishes bond dexels from surface dexels. Other schemes are possible: *e.g.* active site versus non-active site atoms or main chain versus non-main chain atoms.

Eight bits of color provide only 256 possible values. Fortunately, since organic molecules are primarily composed of carbon, nitrogen, and oxygen, CPK images require few distinct colors. Often three color ranges suffice: green for carbon, blue for nitrogen, and red for oxygen. To get the most out of 8 bits, *DockTool* provides 32 shades for each of seven atomic types and a back surface color. The high order three bits select an atom type (or back surface), and the low order five bits determine the shade.

4.3 Frame Buffer Algorithms

Figure 4.4 shows the layout of Ikonas frame buffer memory and the location of *DockTool* data structures.

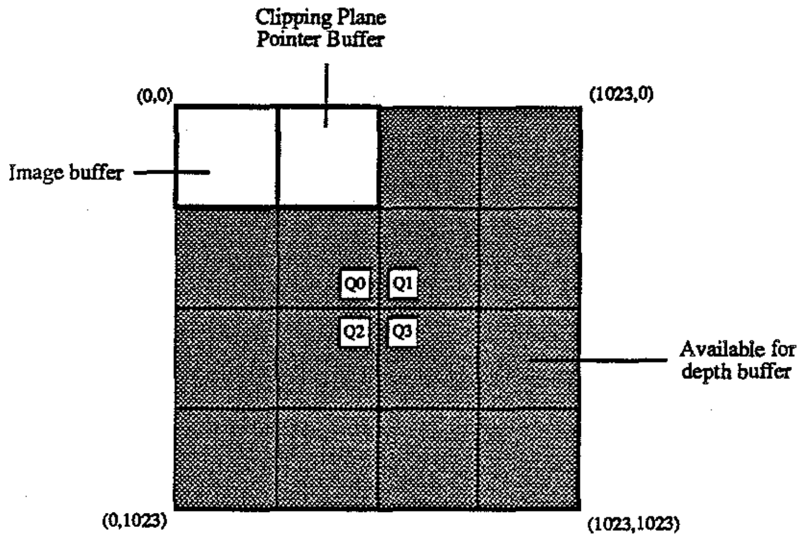

Figure 4.4: Frame Buffer Allocation.

The Ikonas frame buffer contains 1,048,576 pixels. The *image buffer* and the *clipping plane pointer buffer* (CPPB) are both 256 by 256 and together contain 131,072 pixels. This leaves 917,504 pixels for the depth buffer. The depth buffer contains 65,536 depth vectors. This provides for an average of 14 dexels per depth vector.

Addressing considerations limit the length of a depth vector: pixel addresses are not consecutive across the boundaries between quadrants 0 and 1 (Q0 and Q1 in Figure 4.4) and across quadrants 2 and 3 (Q2 and Q3). Since frame-buffer algorithms assume that dexels within a depth vector have consecutive addresses, depth vectors are limited to 512 dexels. In practice, each depth vector must be preceded and followed by a null marker. Therefore, the maximum number of dexels in a vector is 510.

The CPPB provides the necessary run-time access to the depth buffer. The value at X, Y is either null or a pointer to the depth vector at X, Y in the depth buffer.

Double buffering is required for smooth raster image updates. In order to save valuable frame buffer space, double buffering is implemented within the image buffer. Since shades are only 8 bits, we can partition the 24-bit-deep buffer into three sets. Each set consists of 8 consecutive bit planes. Two of the sets are used to implement double buffering. The next image is rendered into one set while the image currently being displayed (from the other set) is protected by a write mask. We then use the Ikonas crossbar to switch the displayed sets.

4.3.1 Clipping

Macromolecular surface clipping permits the user to cut away obstructing portions of the molecule. Clipping is accomplished by comparing a user selectable Z value against the Z values in every depth vector. The dexel chosen for display from each depth vector has the smallest Z value of all those dexels whose Z is greater than the user selected Z. The user selects a Z value by manipulating a slider.

The host process polls this slider to get new positions for the clipping plane. This value is transmitted to the Ikonas and the new image is computed as follows.

For each X, Y in the image buffer:

1. Get the dexel pointer from X, Y in the CPPB
2. Search the depth vector (in the appropriate direction) until a dexel Z value is found that is the smallest of all those greater than the new Z clip value at X, Y (or until we find the null marker)
3. Copy the 8-bit shade (or background color) to the current write plane of the image buffer
4. Update the CPPB at X, Y to point to the new visible dexel.

Figure 4.5 provides a graphic depiction of the algorithm.

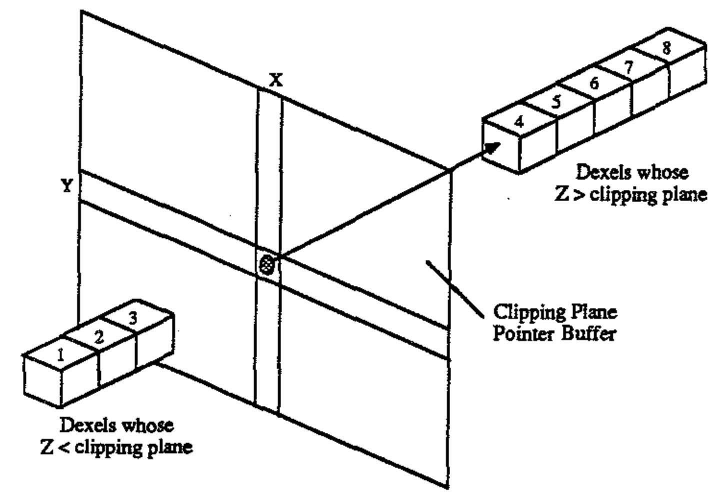

Figure 4.5: Clipping Algorithm.

The CPPB pointer points to the currently visible dexel at X, Y (dexel 4). If the new clipping plane Z is greater than the Z at dexel 4, we will search in the positive Z direction (towards dexel 5). Otherwise, the search will proceed backwards towards dexel 3. Recall that the dexels are actually contiguous within a depth vector, *e.g.*, dexel 3 is accessed by decrementing the pointer to dexel 4.

By updating the CPPB to always point to the currently visible dexel at X, Y we reduce the search time through depth vectors during subsequent image updates. This savings assumes a "locality of motion" of the clipping plane. Large movements may still require many long searches to find the visible dexel.

There is no reason why our clipping plane Z value must be the same at every X, Y position. We can use the slider value as a parameter and compute the Z value for each depth vector as a function of X, Y, and parameter value. I identify two basic types of these procedural clipping planes: ***functional*** and ***tabular***.

Any function of X and Y can be used to define a clipping "plane". For example, a paraboloid pointing away from the viewer would appear to clip a larger and larger hole as the slider is moved forward. I've implemented five functional clipping planes. Four are sloped planes: sloped left and right and parallel to the Y axis and sloped up and down and parallel to the X axis. The fifth is a "punch" clipping plane. The plane is reduced to a small square; only the area within the square is clipped as the slider is moved. Both the punch size and center can be interactively changed by the user.

We can also define a clipping plane by any tabular function of X and Y. Given X and Y we can look up the table Z value and add the slider Z value to it. No tabular clipping planes are currently implemented. It should be possible to use a 256 by 256 section of the frame buffer as the table. Although this monopolizes a large part of the frame buffer, we would need only one table; another could be loaded from the host and overwrite the previous table.

4.3.2 Object Selection

The dexel tag allows frame-buffer algorithms to identify dexels as belonging to particular objects. The current implementation uses tags for two purposes: cutting surface or bonds on or off, and making the surface transparent.

Given the clipping algorithm described above, implementing on/off objects is easy. As we search the depth vectors for visible surfaces we merely skip those dexels containing the tag number of the off object. The one-bit tag allows at most one object to be off at any time.

Transparency is also easy. A transparent color is calculated only if a bond dexel is immediately behind a surface dexel. In addition, the computed color intensity is independent of the distance between the surface and the bond. Given these simplifications, the transparent color is also easy to calculate. If the molecular surface is visible (*i.e.* not clipped) and transparent, we look one dexel further in the depth vector. If that dexel is a bond, the transparent color is merely the bond shade reduced in intensity. No attempt is made to incorporate the surface color; the multiplications required to produce a weighted sum are expensive.

4.3.3 Ligand Transformations

Ligand rotations and translations are controlled by a joystick and three knobs. These devices are polled by the host process for updates. If a change has been made, the host builds a fresh transformation matrix and sends it to the Ikonas. The Ikonas transforms the ligand atomic coordinates and uses a three-dimensional Bresenham line-drawing algorithm to render the stick-figure ligand [Kaufman and Shimony, 1986].

A two-dimensional line drawing algorithm would be sufficient to render the stick figure superimposed on the macromolecule. However, we need transformed Z coordinates for three purposes. First, intensity reduction for depth cueing is based on Z value. Second, the Z values are required for bump checking. Third, we need Z values for every pixel drawn so that we can clip the ligand if it is behind the macromolecule. This is accomplished by a Z comparison with the currently visible dexel at the X, Y position of the pixel to be drawn.

4.3.4 Bump Checking

The depth buffer contains all the information needed for bump checking. After the ligand coordinates have been transformed by the latest update, each atom is checked. The X, Y coordinates of the atom are used to access the depth vector via the CPPB. Once the depth vector has been located, we search backwards to find the first dexel (since the CPPB pointer may point to any dexel in the vector). We then search forward through the depth vector, counting front and back surface dexels, until we encounter a surface dexel whose Z is greater than the atom's Z. If the count is even at this point, then the atom does not lie in the interior of the molecular surface, otherwise it does.

Figure 4.6 shows a slice of the depth buffer in the X-Z plane at Y_s.

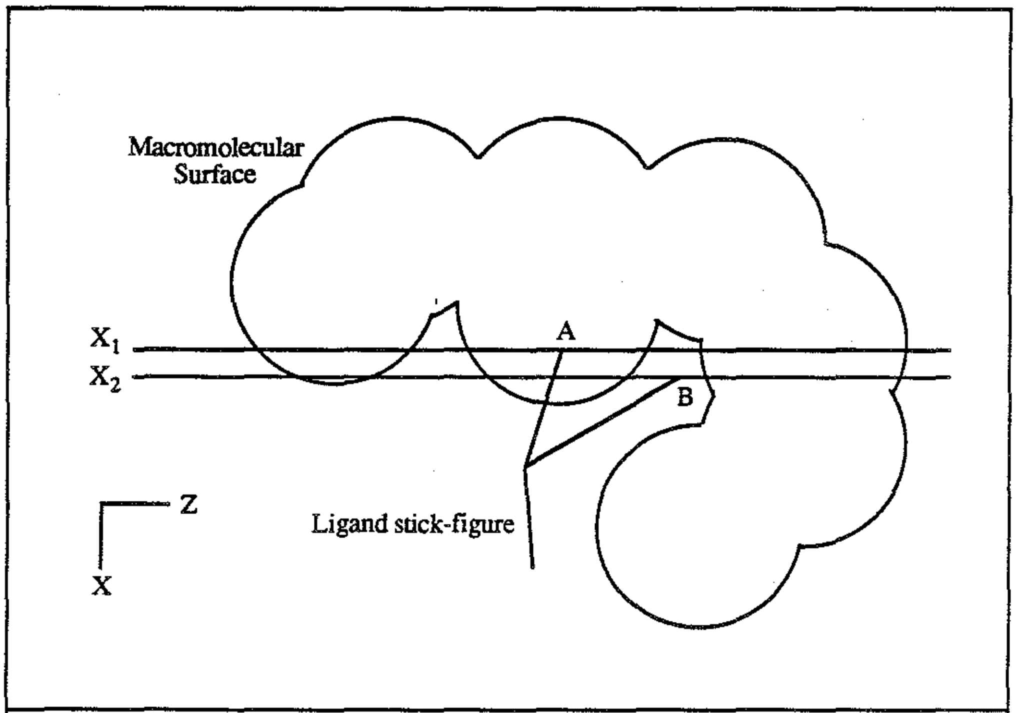

Figure 4.6: Bump Checking Algorithm.

The coordinates of ligand atoms A and B are X_1, Y_s and X_2, Y_s, respectively. As we count towards increasing Z (left to right) in the depth vector at X_1, Y_s, we encounter three surfaces before a surface dexel has a Z value greater than atom A. Therefore, A is within the molecular surface and bumping. Searching the depth vector at X_2, Y_s we count four surfaces prior to atom B. Therefore B is not bumping.

This algorithm makes two assumptions. First, every depth vector must begin with a front surface dexel. Second, each front-surface dexel must have a matching back-surface dexel. The *Preprocessor* assures that both assumptions are correct. All atoms are translated in Z so that the atom with the smallest Z center coordinate will be greater than the minimum Z window boundary. This assures that no front surface dexels are clipped by the windowing operation. Rendering the back atomic hemisphere as the mirror image of the front assures that each front surface dexel has a matching partner in the rear. Hence, all lines through the molecular surface parallel to the Z axis pass through an even number of dexels.

4.4 Other Algorithms

Frame Buffer Loader. Prior to interactive use, the depth buffer must be loaded to the Ikonas frame buffer. This task is performed by the *Loader*. The *Loader* performs a function analogous to an operating system program loader. It allocates space for depth vectors, loads the dexels, and initializes the clipping plane pointer buffer.

The *Loader* consists of a host program running on the VAX and microcode running on the Adage-Ikonas. The host program reads the binary depth-buffer file and

- buffers depth vector counts and dexels in a manner allowing efficient transmission and allocation on the Adage-Ikonas
- transmits the data

The microcode program receives the data and

- allocates space for the depth vectors
- loads the depth vectors
- stores a pointer to the head of each depth vector in the CPPB

Chapter 5
Conclusions

5.1 Algorithmic Complexity

Graphics algorithms are notoriously difficult to analyze. The analysis of an interactive system not only depends on data, but also on mode of use. However, it is possible to make some statements regarding the complexity of *DockTool* algorithms.

First, here are some statistics from the depth buffer generated from 426 atoms in the active site of *E. Coli* Dihydrofolate Reductase.

Total dexels:	*193,131*
Total (non-empty) depth vectors:	*62,734*
Screen coverage (by non-empty depth vectors):	*95.72%*
Maximum depth vector length:	*101 dexels*
Minimum depth vector length:	*2 dexels*
Average depth vector length:	*3.08 dexels*

5.1.1 Clipping

The clipping algorithm is $O(n^2)$, where n is the length of one dimension of the image buffer (currently 256). We can expand this to

$$O(n^2(A+2)) \tag{1}$$

based on frame buffer memory accesses. A frame buffer read or write is the single most expensive operation in frame buffer algorithms on the Ikonas. One frame buffer access takes 700 ns; an access to Ikonas program memory requires only 200 ns. The A in equation (1) is the average number of dexel accesses (per depth vector) required to find the visible surface. The constant 2 embodies the CPPB accesses: one to read the depth vector pointer and one to write the updated depth vector pointer. The range of A is zero to the maximum depth vector length and, in practice, averages about one.

For on/off objects and transparency A is slightly larger. If an object is off we may have to skip a dexel belonging to the off surface and look further in the depth vector. If we encounter a transparent surface dexel in our search, we must look one dexel further to determine if it is a bond dexel.

Note that the speed of the clipping algorithm is totally independent of the quality of the image. Sophisticated rendering and anti-aliasing algorithms can be used to create depth buffers with no degradation in run-time speed. If anti-aliasing is used, some heuristic must be employed to determine the Z value of anti-aliased dexels. However, this is a *Preprocessor* decision and effects *DockTool* run-time speed only slightly: spheres may be somewhat larger after anti-aliasing.

The speed is dependent on image complexity. It depends on the ratio of zero length to non-zero length depth vectors (*i.e.* X, Y complexity or the percentage of the image buffer covered by image). The constant term of equation (1) collapses to one for zero length depth vectors. Speed also depends on Z image complexity. Large jumps in the clipping plane position are slower for long depth vectors, *i.e.* A in equation (1) is relatively large.

The actual results are promising. With standard clipping the system provides about one to two updates per second with the Dihydrofolate Reductase depth buffer described above. On/off objects and transparency slow it down only slightly.

5.1.2 Bump Checking

Most of the bump checking work is performed by the *Preprocessor*. At *DockTool* run-time, bump checking involves only a lookup and a search for each ligand atom. The algorithm is $O(m)$, where m is the number of atoms in the ligand molecule. Based on frame buffer accesses, we can expand this to

$$O(m(B+1)) \tag{2}$$

The constant B embodies the search time through the depth vector. Recall that we must first find the beginning of the vector before we can search forward. The 1 in equation (2) is the CPPB lookup to find the correct depth vector. Since the number of atoms in a ligand molecule is usually small (less than fifty), the bump checking algorithm is very fast.

Let's analyze the bump check resolution for a macromolecular carbon atom enhanced by 1.57Å and scaled with a "typical" window size. The atomic radius after enhancement is 3.31Å. The radius of the rendered sphere in the image buffer is 32 pixels. This gives 0.1034Å per pixel or about 10 pixels to the Angstrom. Therefore, the bump check error in X, Y is less than 0.1034Å. The radius of the rendered sphere in Z is 8336 "Z units". This gives us 0.0002Å per Z unit and the same value as an upper bound on the Z direction bump check error. Clearly, the X, Y resolution leads to the dominant error.

Bump checking errors are also introduced by the approximation of the ligand van der Waals surface by stick-figure and macromolecular enhanced spheres. A carbon ligand atom is approximated by the enhancement, 1.57Å. Its normal radius is 1.74Å. This gives a possible error of up to 0.17Å, as seen in Figure 5.1.

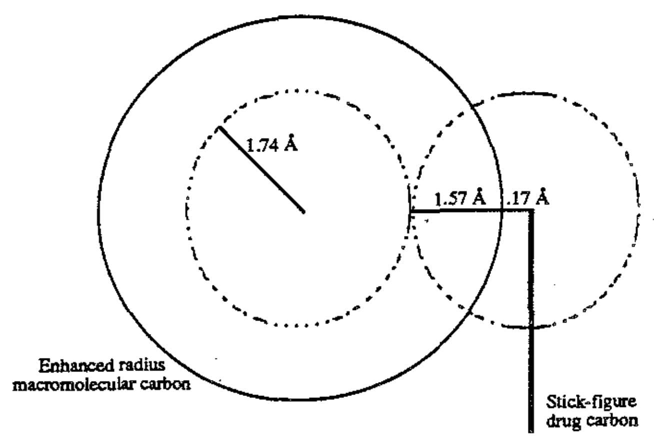

Figure 5.1: Bump Checking Error.

The dotted spheres represent the true van der Waals radii of the two carbons. As can be seen, the stick-figure carbon will bump the macromolecular carbon at a point where the actual surfaces are interpenetrating by 0.17Å.

5.2 Problems and Suggested Improvements

The raster visualization of the molecular surface seems to provide better structural perception than a vector display dot surface. The occluding surfaces work to reduce unneeded visual complexity. Clipping can also be used to reduce visual clutter. At the same time, the raster surface provides more "samples" of the molecular surface.

Unfortunately, depth perception is inadequate. Intensity depth cueing provides only a rough gauge of depth. Bump checking and occlusion provide greater precision but give depth information only at points of contact.

The lack of standard transformations for the macromolecule (in particular rotation), is a serious drawback. It is difficult to maintain three-dimensional structural perception of the macromolecule. A stereo display mechanism would certainly help. DOCKTOOL can be upgraded to make use of stereo display technology.

A means of displaying orthogonal views might improve depth perception without resorting to stereo. A 90° Y rotated view could either be displayed at all times on half the image buffer or at the user's discretion on the whole screen [Cory, 1987]. This technique has not been explored.

Perception is somewhat degraded by aliasing artifacts exacerbated by low resolution. As indicated above, anti-aliasing techniques can be used by the *Preprocessor* (once heuristics for determining the Z value of a dexel have been chosen). The resolution could also be raised. However, as can be deduced from equation (1), clipping performance falls off rapidly as the width of the image buffer is increased. In addition to a larger image buffer and CPPB, the higher resolution depth buffer will also require more space.

On the other hand, mode of use might permit higher resolution. The user may not change the clipping plane often once he has found a good position. In that case, only fast ligand update algorithms are required. Bump checking is independent of resolution and the line drawing algorithm for the ligand would be slowed only slightly. Increasing the resolution would also improve the accuracy of bump checking.

Other possible enhancements:

- The use of Connolly's analytic molecular surface [Connolly, 1983] rather than spheres. This surface should give better bump checking accuracy and possibly better structural perception.
- Allow the ligand conformation to be interactively changed.
- Tie the ligand to an energy minimization algorithm providing dynamic coordinate updates.
- Give more two-dimensional cues to depth:

  - Texture gradients; *e.g.* with a bump texture more distant objects would have smaller, less distinct bumps

  - Intensity depth cueing via "bluish haze" and reduced contrast. This would mimic the usual perception of looking at distant objects on a hazy day.

Chapter 6
User's Manual

This chapter is a reference manual for the DOCKTOOL system. Specifically, it covers depth buffer creation, CPK image rendering, and the interactive *DockTool* program.

The following notation is used in this chapter:

- Commands to be typed at the terminal to the operating system are printed in `typewriter type`.
- Commands to DOCKTOOL programs are printed in **boldface**.
- Alternate spellings for a command are listed in parentheses.
- Arguments are printed in *italics* and may be one of the following types:
  - integer - a number without a decimal point
  - float - a number with a decimal point
  - filename.xxx - any acceptable system filename with suggested suffix ".xxx"
  - color - three float values giving red, green, and blue color components; normalized to the range 0-1.
- Arguments in brackets ([ ]) indicate a choice of those listed.
- An argument followed by braces containing an integer ($\{n\}$) indicates that the argument is to repeated n times.
- Arguments separated by "|" and within brackets ([*argtype*1 | *argtype*2]) indicate a choice of argument types.

All DOCKTOOL files can be accessed from a common parent directory. Currently, that directory is grumpy:/grip4/palmer/docktool. Hereafter, this root is referred to as /dtroot.

The DOCKTOOL system uses file name suffixes to differentiate file types. The following conventions are used:

.par - *sprep* or *srend* parameter input file

.a - atomic coordinate input data

.b - atomic connectivity file (for bonds)

.db - *sprep* depth buffer output file

.edb - encoded depth buffer file

.ag - depth buffer grabbed from Ikonas frame buffer with Glassner's agrab

.cpk - *srend* raster output file in Pique's ikload format

6.1 Creating a Depth Buffer File

Several steps are involved in producing a depth buffer file. The first part of the pipeline is shown in Figure 6.1.

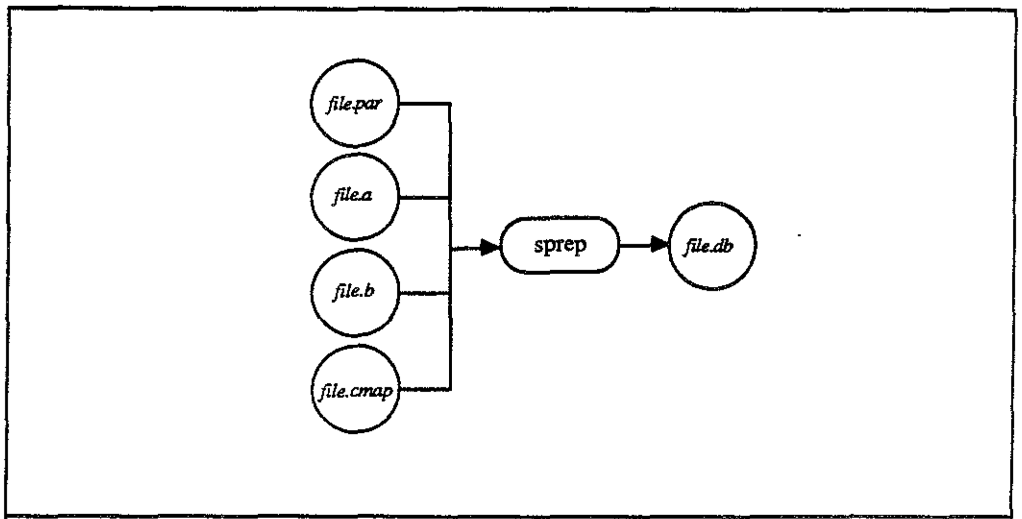

Figure 6.1: Preprocessor Pipeline: Part 1

The *Preprocessor* described in section 4.2 is a parameter driven batch process. The source code of the *Preprocessor* actually compiles into two different programs:

sprep - 8-bit color *DockTool* preprocessor as discussed in Chapter 4. *Sprep* currently renders at 256 by 256 resolution only.

srend - 24-bit color CPK image renderer. *Srend* currently renders at 512 by 512 resolution only. Use of *srend* to generate CPK images will be discussed shortly. Attributes shared by *sprep* and *srend* (*e.g.*, .par file command syntax), are discussed here.

The following are acceptable commands in a .par file. All commands are accepted by both *sprep* and *srend*. However, some commands are ignored by one or the other; these are marked with (*sprep* only) or (*srend* only).

CENTER [*integer* | *float*] {3}

The three numbers define the center of rotation of the molecule. The units are in angstroms in the world coordinate system.

ROTATE [*integer* | *float*] {9}

The nine values represent a 3 by 3 rotation matrix to be applied to all atomic coordinates. The elements are given in row-major order. The center of rotation is specified with the **CENTER** command.

WINDOW [*integer* | *float*] {4}

The four values represent the minimum X, maximum X, minimum Y, and maximum Y window parameters. The units are angstroms in the world coordinate system. (Note: The Z window parameters are set automatically.)

ATOMFILE *filename.a*

The input atomic coordinates are read from *filename.a*. The file must consist of a list of atomic attributes, one atom per line. Each line has the following format:

$$X \quad Y \quad Z \quad R \quad C \quad atype$$

The first three values give the X, Y, and Z position (in world coordinates) of the atom's center. The fourth value gives the atomic radius. If this value is zero, the radius will be set to a default value based on atom type. The fifth value is a color table number and is ignored. The *atype* defines the atomic type and is chosen from the set

$$\{C, O, N, X, H, S, P, U, I, Z, F\}$$

yielding carbon, oxygen, nitrogen, xenon, hydrogen, sulfer, phosphorus, copper, iodine, zinc, and flourine, respectively. (Note: This is the standard UNC .a format.)

BONDS *filename.b*

Filename.b is a file giving atomic connectivity information. Each line in the file is a pair of integers separated by a space. The integers are line numbers in *filename.a* and imply that the atoms on those lines are bonded. The line count starts from zero. For example, the line containing "0 1" implies that the first two atoms in *filename.a* are bonded. Atomic connectivity files are created by the DOCKTOOL utility program *mkbonds*.

COLORMAP *filename.cmap*

Filename.cmap is a colormap file that allows the default colormap entries to be overridden. The format of a line of a colormap file is as follows:

atype color

where *atype* is chosen from the set given under the **ATOMFILE** command and *color* is a color argument as described above (*srend* only).

PREP *filename.db*

Filename.db is a depth buffer output file produced by *sprep*. Parameter files read by *sprep* MUST use the **PREP** command to specify an output file (*sprep* only).

RASTER *filename.cpk*

Filename.cpk is a raster output file produced by *srend*. It is formated for display by Pique's ikload Ikonas frame buffer loader. If the **RASTER** command is missing from the .par file, the default output is the Ikonas frame buffer (*srend* only).

LIGHT *float* {3}

The three numbers describe a vector in the world coordinate system that represents a light direction. The light source is infinitely far away.

LIGHTCLR *color*

The *color* specifies a light color to be used in the shading model (*srend* only).

BACKGROUND *color*

The *color* specifies a background color for CPK images (*srend* only).

PHONG

Use the Phong shading model in computing surface colors (*srend* only).

IA [*integer* | *float*]

Set the incident ambient light intensity to the argument value.

IL [*integer* | *float*]

Set the incident intensity from the light source to the argument value.

KA [*integer* | *float*]

Set the ambient diffuse reflection constant to the argument value.

KD [*integer* | *float*]

Set the diffuse reflection constant to the argument value.

KS [*integer* | *float*]

Set the Phong reflectance curve constant to the argument value. (*srend* only).

KN [*integer* | *float*]

Set the Phong cosine power to the argument value. (*srend* only).

K [*integer* | *float*]

Set the depth cueing distance constant to the argument value.

DEPTHCUE

This flag enables intensity depth cueing. Objects farther away from the viewer appear darker. (See the **K** command.)

ENHANCED *float*

Enhanced atomic radius spheres are used. If the *float* argument is missing, the default value (1.57Å) is added to each atomic radius. Otherwise, *float* as used as the enhancement factor.

INHERIT

If this flag is present, hydrogen atoms inherit the color of the immediately preceding atom in the atomic coordinate input file (*i.e.*, *filename.a*).

NOSPHERES

If this flag is present, CPK spheres will not be rendered. This is used to render atomic bonds only (*srend* only).

STATS

Compile and print various depth buffer statistics (*sprep* only).

VERBOSE

Print various informational messages during execution.

Two files are required to build a depth buffer: an atomic coordinate file (filename.a) and a parameter file (filename.par). If bonds are to be included in the depth buffer, filename.b will also be needed. To create filename.b from filename.a run *mkbonds* prior to *sprep*. (Note: All programs in the pipeline are found in /dtroot/bin.)

```
mkbonds filename.a > filename.b
```

The next step in creating a depth buffer file is to run *sprep*. Type

```
sprep filename.par
```

Sprep reads filename.par, processes the commands found there, and writes filename.db (or the file name specified in the **PREP** command in filename.par).

Subsequent stages in the preprocessor pipeline are shown in Figure 6.2.

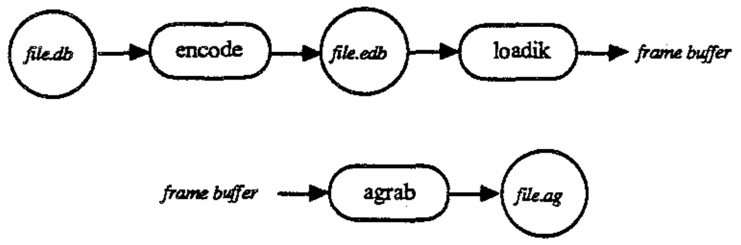

Figure 6.1: Preprocessor Pipeline: Part 2

The next step is to filter filename.db through *encode*. *Encode* compresses the depth buffer yielding reduced file size and increased efficiency in frame buffer loading. Type

```
encode filename.db > filename.edb
```

After encoding we are ready to load the depth buffer to the Ikonas frame buffer. The program to do this is *loadik*. Type

```
loadik filename.edb
```

Loading the frame buffer is very time consuming for large depth buffers. Fortunately, it need only be done once and can then be grabbed with a frame buffer "grabber". The current system makes use of Glassner's *agrab* and *aput* to save and reload depth buffers. To save a depth buffer after loading with *loadik*, type

```
/usr/ikonas/agrab -f -w 0 0 1023 1023 filename.ag
```

Fortunately, the depth buffer creation process is easily mechanized. The file /dtroot/data/MakeDB is a makefile that automates the pipeline. For example, it suffices to type

```
make -f MakeDB filename.edb
```

to generate the encoded depth buffer starting from only filename.a and filename.par. See /dtroot/data/MakeDB and make(1) for more information. /dtroot/data also contains example .par, .a, .b, and .cmap files.

Srend is used to generate 24-bit color CPK images at 512 by 512 resolution. Type

```
srend filename.par
```

Srend will read filename.par, process the commands found there, and produce a CPK image. If a **RASTER** command was given in filename.par, then the specified file will be used for output. Otherwise, *srend* writes directly to the Ikonas frame buffer. See Figure 6.3.

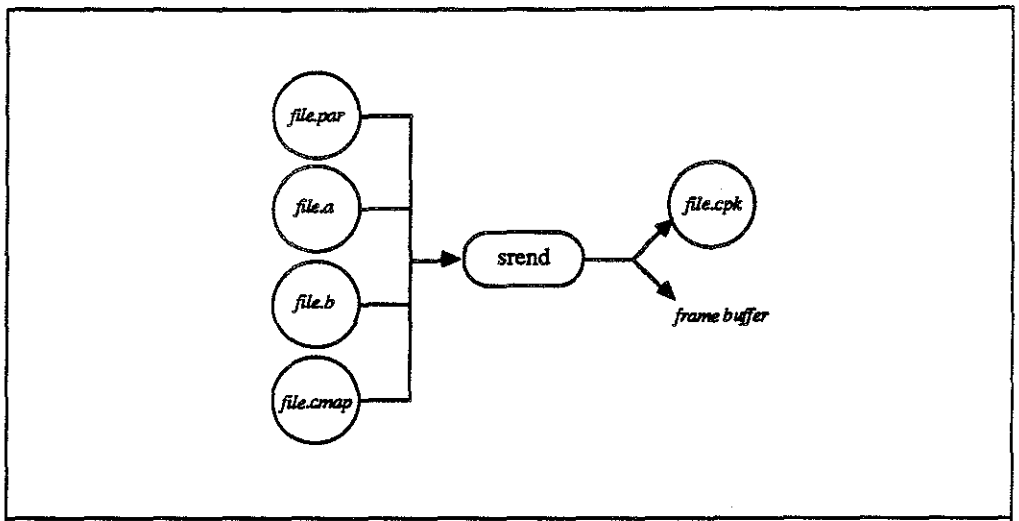

Figure 6.3: Srend Pipeline

The file /dtroot/data/MakeDB also knows how to make .cpk files. Try

```
make -f MakeDB filename.cpk
```

6.2 Running DockTool

The interactive program *DockTool* (described in section 4.3), is located in /dtroot/dock/docktool. Invoke with

```
docktool [ -l -a -q ] [ -d ligand.a ] filename.edb
```

Only one option from the first set is allowed. If none is given, -l is assumed. The -l option specifies that filename.edb is to be loaded into the frame buffer via *loadik*. If the -a option is used instead, docktool will look for filename.ag (in the same directory as filename.edb) and load it via /usr/ikonas/aput. The -q (quick) option tells docktool to assume that the depth buffer is already loaded into frame buffer memory.

The -d option tells docktool to use ligand.a as the atomic coordinate file for the stick-figure ligand. Docktool MUST be able to find ligand.b in the same directory as ligand.a. If ligand.b does not exist, *mkbonds* can be used to generate it (as described above).

After docktool has completed initialization, two windows will appear on your terminal screen separated by a dashed line. The top window is the *Information Window*. System messages will appear here. The bottom window is the *Command Window*. User commands are typed here at the prompt. These commands are described below.

surf [*on off std transparent*]

Change the state of display of the macromolecular surface:

on	display the molecular surface
off	make the surface invisible (also effectively issues a "bonds on" command since both surface and bonds cannot be off at the same time)
std	display the molecular surface as opaque
transparent	display the molecular surface as transparent

bonds [*on off*]

Change the state of display of the macromolecular bonds:

on	display the molecular bonds
off	make the bonds invisible (also effectively issues a "surf on" command since both surface and bonds cannot be off at the same time)

beep [*on off*]

Enable or disable bump check sound feedback.

bumpcheck [*on off*]
(**bc**)

Enable or disable bump checking.

help
(**?**)

Activate the online help facility.

plane [*std lin rin uin din vin vout punch*]

Change the clipping plane:

std	standard plane parallel to screen
lin	parallel to Y axis and tilted in at the left
rin	parallel to Y axis and tilted in at the right
uin	parallel to X axis and tilted in at the top
din	parallel to X axis and tilted in at the bottom
vin	parallel to Y axis; a "V" pointing in

vout	parallel to Y axis; a "V" pointing out
punch	clipped area is a square subset of the screen

Two commands, **pc** and **pr**, are provided for controlling the position and size of the *punch* clipping plane.

pc *integer* {2}

The two values represent a location in the screen coordinate system. The punch center is set to this point. (0,0) is the upper left corner; (255,255) is the lower right.

pr *integer*

Set the punch "radius" to the argument value. The value actually represents half the length of the punch side in screen coordinate units.

quit
(q)

Quit docktool.

Appendix A: Docktool Source Code

The directory grumpy:/grip4/palmer/docktool/spheres
contains the souree code for sprep and srend:

```
        main.c
        parscan.h
        parscan.l
        prep.h
        prep.c
        render.h
        render.c
        spheres.c
        bonds.c
        clip.c
        clip.h
        write.c
        write.h
        common.h
        config.h
        standard.h
        aux.c
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        main.c:  contains:

    >>>>    main:

                                        Thomas C. Palmer
                                        8-jan-87
*/

/* $Header: main.c,v 1.4 87/03/18 16:43:16 palmer Exp $ */
static char rcsid[] = "$Header: main.c,v 1.4 87/03/18 16:43:16 palmer Exp $";

#include "standard.h"

main(argc, argv)
int argc;
char *argv[];
{
#ifdef DOREND
    static char prgmname[] = "srend";
#else
    static char prgmname[] = "sprep";
#endif

  /* process command line arguments */
    while (argc > 1 && argv[1][0] == '-') {
        switch (argv[1][1]) {
          default:
            fprintf(stderr, "%s: unknown arg %s\n",prgmname,argv[1]);
            exit(1);
        }
        argc--;
        argv++;
    }

    if (argc != 1)      /* Reopen stdin for yylex */
        if (freopen(argv[1], "r", stdin) == NULL) {
            fprintf(stderr, "%s: can't open %s\n",prgmname,argv[1]);
            exit(1);
        }

  /* Render or preprocess spheres */
    prep();
    scan();
    exit(0);
}


 /* end main.c */
```

```
/*
 *      parscan.h:  Header file for parscan.
 */

#include "config.h"
#include "standard.h"

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::::: */

#define ERR_TKN      260
#define CENTER_TKN   261
#define XFORM_TKN    262
#define AFILE_TKN    263
#define WIND_TKN     264
#define VERBOS_TKN   265
#define ENHW_TKN     266
#define ENHWO_TKN    267
#define BGC_TKN      268
#define OFILE_TKN    269
#define IA_TKN       270
#define IL_TKN       271
#define KA_TKN       272
#define KD_TKN       273
#define KS_TKN       274
#define K_TKN        275
#define KN_TKN       276
#define LIGHT_TKN    277
#define LCLR_TKN     278
#define PHONG_TKN    279
#define YONCLIP_TKN 280
#define HITHCLIP_TKN 281
#define XINCLIP_TKN 282
#define YINCLIP_TKN 283
#define CFILE_TKN    284
#define PFILE_TKN    285
#define BFILE_TKN    286
#define NOSPH_TKN    287
#define STATS_TKN    288
#define INHER_TKN    289
#define DPCUE_TKN    290


/* :::  Externals  ::::::::::::::::::::::::::::::::::::::::::::::::  */

extern int yylex();
#ifndef SUN
char yytext[200];
#endif


  /* end parscan.h */
```

```
%{
/*
    Lexical analysis for parameter file scanning.
*/

#include "parscan.h"
#define NOTYET(str) fprintf(stderr,"%s not implemented.\n",str)

%}
%a 3000
%e 1200
%o 3500

D    [0-9]
I    [-+]?{D}+
F    [-+]?{D}*"."{D}*
W    ([" "|\t|\n]+)
S    [a-zA-Z]+
FN    [a-zA-Z0-9\.\/_]+
%%
CENTER({W}({I}|{F}){3,3})        return(CENTER_TKN);
ROTATE({W}({I}|{F}){9,9})        return(XFORM_TKN);
WINDOW({W}({I}|{F}){4,4})        return(WIND_TKN);
ATOMFILE({W}{FN})                return(AFILE_TKN);
VERBOSE                          return(VERBOS_TKN);
STATS                            return(STATS_TKN);
ENHANCED                         return(ENHWO_TKN);       /* without amount */
ENHANCED({W}{F})                 return(ENHW_TKN);        /* with amount */
BACKGROUND({W}{F}{W}{F}{W}{F})   return(BGC_TKN);
LIGHTCLR({W}{F}{W}{F}{W}{F})     return(LCLR_TKN);
RASTER({W}{FN})                  return(OFILE_TKN);
PREP({W}{FN})                    return(PFILE_TKN);
LIGHT({W}{F}{W}{F}{W}{F})        return(LIGHT_TKN);
IA({W}({I}|{F}))                 return(IA_TKN);
IL({W}({I}|{F}))                 return(IL_TKN);
KA({W}({I}|{F}))                 return(KA_TKN);
KD({W}({I}|{F}))                 return(KD_TKN);
KS({W}({I}|{F}))                 return(KS_TKN);
K({W}({I}|{F}))                  return(K_TKN);
KN({W}{I})                       return(KN_TKN);
PHONG                            return(PHONG_TKN);
CLIP{W}YON({W}{F})               return(YONCLIP_TKN);
CLIP{W}HITHER({W}{F})            return(HITHCLIP_TKN);
CLIP{W}XIN({W}{F})({W}{F})       return(XINCLIP_TKN);
CLIP{W}YIN({W}{F})({W}{F})       return(YINCLIP_TKN);
COLORMAP({W}{FN})                return(CFILE_TKN);
BONDS({W}{FN})                   return(BFILE_TKN);
NOSPHERES                        return(NOSPH_TKN);
DEPTHCUE                         return(DPCUE_TKN);
INHERIT                          return(INHER_TKN);
DRUG                             NOTYET("DRUG");
TRANPARENCY                      NOTYET("TRANPARENCY");
ANTIALIAS                        NOTYET("ANTIALIAS");
^"#"[^\n]*\n                     ;
[ \t\n]                          ;
.                                return(ERR_TKN);
```

```
/*
 *      prep.h:  Header file for prep.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#define ATOMSCALE 100.0         /* Scale up atoms */
#define RSCALE      1.0         /* Radius scale */
#define PROBERADIUS 1.5         /* For enhanced spheres */
#define FUDGEFACTOR 2.0         /* Auto z windowing fudge factor */

#define DEFAULT_XY_WIN  30.0    /* Default window */
#define DEFAULT_Z_WIN   30.0

#define DEFAULT_IA 0.5          /* incident ambient light intensity */
#define DEFAULT_IL 1.0          /* incident intensity from light source */
#define DEFAULT_KA 0.15         /* ambient diffuse reflection constant */
#define DEFAULT_KD 0.6          /* diffuse reflection constant */
#define DEFAULT_KS 0.2          /* Phong reflectance curve constant */
#define DEFAULT_KN 15           /* Phong cosine power */
#define DEFAULT_K  1            /* distance constant (depth cueing) */


/* :::  Globals  :::::::::::::::::::::::::::::::::::::::::::::::  */

POINT center;           /* rotation center */
FLOAT matrix[3][3];     /* rotation matrix */
POINT win_min;          /* window lower left front corner */
POINT win_max;          /* window upper right back corner */

char afile[80];         /* Name of input atom file */
char bfile[80];         /* Name of input bonds file */
char cmapfile[80];      /* Name of input colormap file */

bool verbose;           /* Be verbose */
bool enhanced;          /* Enhance sphere radii by proberadius */
FLOAT proberadius;      /* Amount of enhancement */

NCOLOR cmap[] = {       /* colormap */
    { 0.10, 0.90, 0.30 },   /* carbon */
    { 0.90, 0.10, 0.20 },   /* oxygen */
    { 0.08, 0.30, 0.90 },   /* nitrogen */
    { 1.00, 0.85, 0.00 },   /* sulfer */
    { 0.92, 1.00, 0.81 },   /* phosphorus */
    { 0.88, 0.38, 0.10 },   /* copper */
    { 0.53, 0.71, 0.97 },   /* zinc */
    { 0.82, 0.76, 0.54 },   /* hydrogen */
    { 0.50, 0.50, 0.50 },   /* xenon */
    { 0.50, 0.50, 0.50 },   /* iodine */
    { 0.50, 0.50, 0.50 },   /* flourine */
    { 0.50, 0.50, 0.50 },   /* unknown (carbon) */
};


/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::  */

#define swappoints(p1,p2) {       \
    register int t;               \
    t=p1.x;  p1.x=p2.x; p2.x=t;  \
    t=p1.y;  p1.y=p2.y; p2.y=t;  \
    t=p1.z;  p1.z=p2.z; p2.z=t;  \
}


  /* end prep.h */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

       prep.c:  contains:

    >>>>    prepinit:
    >>>>    prep:
    >>>>    readparfile:
    >>>>    printparams:
    >>>>    readatomfile:
    >>>>    readbondfile:
    >>>>    readcmapfile:
    >>>>    rotatoms:
    >>>>    winatoms:
    >>>>    clipatoms:
    >>>>    sortatoms:
    >>>>    s_compare:
    >>>>    sortbonds:
    >>>>    b_compare:
                                          Thomas C. Palmer
                                          8-jan-87
*/

/* $Header: prep.c,v 1.8 87/04/06 19:52:50 palmer Exp $ */
static char rcsid[] = "$Header: prep.c,v 1.8 87/04/06 19:52:50 palmer Exp $";

#include "standard.h"
#include "config.h"
#include "common.h"
#include "render.h"
#include "prep.h"
#include "parscan.h"
#include "clip.h"

#ifdef SUN
char yytext[200];
#endif

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  prepinit:   Initialize variables
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                  8-jan-87
 *
 */

prepinit() {
    register int i;

    verbose = FALSE;    enhanced = FALSE;
    depthcue = FALSE;   inherit = FALSE;

#if (defined STATS) && (defined DOPREP)
    stats = FALSE;
    totaldexels = 0;  totaldv = 0;
    maxdvl = 0;  mindvl = MAXDEPTHVECSIZE+1;
#endif

    center.x = center.y = center.z = 0.0;

    matrix[0][0] = matrix[1][1] = matrix[2][2] = 1.0;
    matrix[0][1] = matrix[0][2] = matrix[1][0] = 0.0;
    matrix[1][2] = matrix[2][0] = matrix[2][1] = 0.0;

    win_min.x = win_min.y = -DEFAULT_XY_WIN * ATOMSCALE;
    win_max.x = win_max.y =  DEFAULT_XY_WIN * ATOMSCALE;
    win_min.z = MAXZ;  win_max.z = -MAXZ;

    strcpy(afile, "");     /* Input atoms file name */
    strcpy(bfile, "");     /* Input bonds file name */
    strcpy(cmapfile, "");  /* Input colormap file name */
#ifdef DOREND
    ofptr = NULL;          /* Output raster file pointer */
    nospheres = FALSE;     /* Do render spheres */
#endif

    /* Set default shading parameters */
    light.x = 0.0;  light.y = 0.0;  light.z = 1.0;      /* head on lighting */
    lclr.r = lclr.g = lclr.b = 1.0;                     /* white light */
    bgc.r =   bgc.g =   bgc.b = 0;                      /* black background */
    phong = FALSE;                                      /* no Phong shading */
    Ia = DEFAULT_IA;    Il = DEFAULT_IL;
    Ka = DEFAULT_KA;    Kd = DEFAULT_KD;
    Ks = DEFAULT_KS;    Kn = DEFAULT_KN;
    K =  DEFAULT_K;

  /* Initialize array of clipping functions to NULL */
    clipnum = 0;
    for(i=0; i<MAXCLIP; i++)
        clipfunction[i].ptr = NULL;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  prep:       Read the .par file and process any .a/.b files
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 8-jan-87
 *
 */

prep() {

    prepinit();
    readparfile();
    if(verbose) printparams();
    readcmapfile();
    readatomfile();

  /* Rotate, scale to window, clip and sort atoms */
    rotatoms();
    winatoms();
    clipatoms();
    readbondfile();  /* read bonds after atoms are transformed */
    sortbonds();
    sortatoms();
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readparfile:       Read the .par file
 *
 *  Errors:         unknown item in parameter file
 *
 *  Notes:          Lex is used to supply recognized tokens.
 *
 *  Revisions:     Initial   TCP                 8-jan-87
 *
 */

readparfile() {

    NCOLOR tmpclr;
    char tmp[80];

    while(TRUE) {
        switch(yylex()) {
          case CENTER_TKN:
            sscanf(yytext, "%*s %f %f %f", &center.x,&center.y,&center.z);
            center.x *= ATOMSCALE;
            center.y *= ATOMSCALE;
            center.z *= ATOMSCALE;
            break;
          case WIND_TKN:
            sscanf(yytext, "%*s %f %f %f %f", &win_min.x,&win_max.x,
                   &win_min.y,&win_max.y);
            win_min.x *= ATOMSCALE;  win_max.x *= ATOMSCALE;
            win_min.y *= ATOMSCALE;  win_max.y *= ATOMSCALE;
            break;
          case AFILE_TKN:
            sscanf(yytext, "%*s %s", afile);
            if(verbose) {
                fprintf(stdout,"atoms input file: %s\n",afile);
                fflush(stdout);
            }
            break;
          case BFILE_TKN:
            sscanf(yytext, "%*s %s", bfile);
            if(verbose) {
                fprintf(stdout,"bonds input file: %s\n",bfile);
                fflush(stdout);
            }
            break;
          case OFILE_TKN:
#ifdef DOREND
            sscanf(yytext, "%*s %s", tmp);
            ofptr = fileopen(tmp, "w");
            if(verbose) {
                fprintf(stdout,"raster output file: %s\n",tmp);
                fflush(stdout);
            }
#else
            fprintf(stderr,"Warning: Raster output not in this version.\n");
#endif
            break;
          case XFORM_TKN:
            sscanf(yytext, "%*s %f %f %f %f %f %f %f %f %f",
                   &matrix[0][0],&matrix[0][1],&matrix[0][2],
                   &matrix[1][0],&matrix[1][1],&matrix[1][2],
                   &matrix[2][0],&matrix[2][1],&matrix[2][2]);
            break;
          case VERBOS_TKN:  verbose =  TRUE;    break;
          case DPCUE_TKN:   depthcue = TRUE;    break;
          case INHER_TKN:   inherit =  TRUE;    break;
          case STATS_TKN:
#if (defined STATS) && (defined DOPREP)
```

```
            stats = TRUE;
#else
            fprintf(stderr,"Warning: stats not available in this version.\n");
#endif
            break;
          case ENHWO_TKN:
            enhanced = TRUE;
            proberadius = PROBERADIUS;
            break;
          case ENHW_TKN:
            enhanced = TRUE;
            sscanf(yytext, "%*s %f", &proberadius);
            break;
          case BGC_TKN:
            sscanf(yytext, "%*s %f %f %f", &tmpclr.r, &tmpclr.g, &tmpclr.b);
            scaleclr(tmpclr,bgc);
            break;
          case LCLR_TKN:
            sscanf(yytext, "%*s %f %f %f", &lclr.r, &lclr.g, &lclr.b);
            break;
          case LIGHT_TKN:
            sscanf(yytext, "%*s %f %f %f", &light.x,&light.y,&light.z);
            light.z = -light.z;
            normalize(light);
            break;
          case PHONG_TKN:   phong = TRUE;    break;
          case IA_TKN:      sscanf(yytext, "%*s %f", &Ia);    break;
          case IL_TKN:      sscanf(yytext, "%*s %f", &Il);    break;
          case KA_TKN:      sscanf(yytext, "%*s %f", &Ka);    break;
          case KD_TKN:      sscanf(yytext, "%*s %f", &Kd);    break;
          case KS_TKN:      sscanf(yytext, "%*s %f", &Ks);    break;
          case K_TKN:       sscanf(yytext, "%*s %f", &K);     break;
          case KN_TKN:      sscanf(yytext, "%*s %d", &Kn);    break;
          case YONCLIP_TKN:
            sscanf(yytext, "%*s %*s %f", &clipfunction[clipnum].world_z);
            clipfunction[clipnum].world_z *= -ATOMSCALE; /* left handed */
            clipfunction[clipnum].ptr = clip_yon;
            if(++clipnum == MAXCLIP) {
                fprintf(stderr,"Error: max clipping functions (%d) exceeded.\n"
                        ,MAXCLIP);
                exit(1);
            }
            break;
          case HITHCLIP_TKN:
            sscanf(yytext, "%*s %*s %f", &clipfunction[clipnum].world_z);
            clipfunction[clipnum].world_z *= -ATOMSCALE; /* left handed */
            clipfunction[clipnum].ptr = clip_hither;
            if(++clipnum == MAXCLIP) {
                fprintf(stderr,"Error: max clipping functions (%d) exceeded.\n"
                        ,MAXCLIP);
                exit(1);
            }
            break;
          case XINCLIP_TKN:
            sscanf(yytext, "%*s %*s %f %f",
                   &clipfunction[clipnum].slope,&clipfunction[clipnum].world_z);
            clipfunction[clipnum].world_z *= -ATOMSCALE; /* left handed */
            clipfunction[clipnum].ptr = clip_xin;
            if(++clipnum == MAXCLIP) {
                fprintf(stderr,"Error: max clipping functions (%d) exceeded.\n"
                        ,MAXCLIP);
                exit(1);
            }
            break;
          case YINCLIP_TKN:
            sscanf(yytext, "%*s %*s %f %f",
                   &clipfunction[clipnum].slope,&clipfunction[clipnum].world_z);
            clipfunction[clipnum].world_z *= -ATOMSCALE; /* left handed */
            clipfunction[clipnum].ptr = clip_yin;
            if(++clipnum == MAXCLIP) {
                fprintf(stderr,"Error: max clipping functions (%d) exceeded.\n"
                        ,MAXCLIP);
                exit(1);
            }
            break;
          case CFILE_TKN:
            sscanf(yytext, "%*s %s", cmapfile);
            if(verbose) {
                fprintf(stdout,"input colormap file: %s\n", cmapfile);
                fflush(stdout);
            }
            break;
          case PFILE_TKN:
#ifdef DOPREP
            sscanf(yytext, "%*s %s", tmp);
            pfptr = fileopen(tmp, "w");
            if(verbose) {
                fprintf(stdout,"depth buffer output file: %s\n", tmp);
                fflush(stdout);
            }
#else
            fprintf(stderr,"Warning: DockTool prep not in this version.\n");
#endif
            break;
          case NOSPH_TKN:
#ifdef DOREND
            nospheres = TRUE;
            if(verbose) {
                fprintf(stdout,"No spheres being rendered.\n");
                fflush(stdout);
            }
#else
            fprintf(stderr,
                    "Warning: Spheres always rendered into depth buffer.\n");
#endif
            break;
          case ERR_TKN:     printf("Error in parameter file: %s\n", yytext);
            break;
          case NULL:        return;  /* EOF */
          default:          printf("Unknown case returned by yylex.\n");
        }
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  printparams:        Print various system parameters.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                  20-mar-87
 *
 */

printparams() {

#ifdef DOPREP
    fprintf(stdout,"Stats: %s\n", (stats ? "TRUE" : "FALSE"));
#endif
#ifdef DOREND
    fprintf(stdout,"No spheres: %s\n", (nospheres ? "TRUE" : "FALSE"));
#endif
    fprintf(stdout,"Enhanced spheres: %s\n", (enhanced ? "TRUE" : "FALSE"));
    fprintf(stdout,"Hydrogens inherit: %s\n", (inherit ? "TRUE" : "FALSE"));
    fprintf(stdout,"Phong shading: %s\n", (phong ? "TRUE" : "FALSE"));
    fprintf(stdout,"Depth Cueing: %s\n", (depthcue ? "TRUE" : "FALSE"));
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readatomfile:       Read the .a file of atomic coordinates
 *
 *  Errors:     unable to open .a file
 *              empty .a file
 *              MAXATOMS exceeded
 *
 *  Notes:         Adapted from Mike Pique's "atoss".
 *
 *  Revisions:     Initial   TCP                  8-jan-87
 *
 */

readatomfile() {
    register int natoms;
    register FILE *fptr;
    char linebuf[100];
    FLOAT x,y,z,rad;
    char atype[6];
    int lastcindex = 0;

    if(STREQ(afile,"")) {
        fprintf(stderr, "Error: no atom file given in parameter file.\n");
        exit(1);
    }

    fptr = fileopen(afile,"r");
    natoms = 0;
    while((fgets(linebuf,sizeof(linebuf),fptr)) != NULL) {
        sscanf(linebuf,"%f %f %f %f %*s %s",&x,&y,&z,&rad,atype);
        if(rad == 0.0) {  /* determine radius from atom type */
            switch(atype[0]) {
              case 'C': case 'c': rad = 1.74; break;   /* carbon */
              case 'O': case 'o': rad = 1.40; break;   /* oxygen */
              case 'N': case 'n': rad = 1.54; break;   /* nitrogen */
              case 'X': case 'x': rad = 2.40; break;   /* xenon */
              case 'H': case 'h': rad = 1.20; break;   /* hydrogen */
              case 'S': case 's': rad = 1.80; break;   /* sulfer */
              case 'P': case 'p': rad = 1.70; break;   /* phosphorus */
              case 'U': case 'u': rad = 1.40; break;   /* copper */
              case 'I': case 'i': rad = 1.40; break;   /* iodine */
              case 'Z': case 'z': rad = 1.40; break;   /* zinc */
              case 'F': case 'f': rad = 2.20; break;   /* flourine */
              default: rad = 1.74;   /* carbon */
            }
        }
        switch(atype[0]) {
          case 'C': case 'c': atoms[natoms].cindex = 0; break; /* carbon */
          case 'O': case 'o': atoms[natoms].cindex = 1; break; /* oxygen */
          case 'N': case 'n': atoms[natoms].cindex = 2; break; /* nitrogen */
          case 'S': case 's': atoms[natoms].cindex = 3; break; /* sulfer */
          case 'P': case 'p': atoms[natoms].cindex = 4; break; /* phosphorus */
          case 'U': case 'u': atoms[natoms].cindex = 5; break; /* copper */
          case 'Z': case 'z': atoms[natoms].cindex = 6; break; /* zinc */
          case 'H': case 'h': if(inherit)                      /* hydrogen */
                                    atoms[natoms].cindex = lastcindex;
                              else atoms[natoms].cindex = 7;
             break;
          case 'X': case 'x': atoms[natoms].cindex = 8; break; /* xenon */
          case 'I': case 'i': atoms[natoms].cindex = 9; break; /* iodine */
          case 'F': case 'f': atoms[natoms].cindex = 10; break; /* flourine */
          default:            atoms[natoms].cindex = 11; break; /* (carbon) */
        }
        atoms[natoms].x = x * ATOMSCALE;
        atoms[natoms].y = y * ATOMSCALE;
        atoms[natoms].z = z * ATOMSCALE;
        if(enhanced) rad += proberadius;
        atoms[natoms].rad = rad * ATOMSCALE;
        if(inherit) lastcindex = atoms[natoms].cindex;

        if(natoms++ == MAXATOMS) {
            fprintf(stderr, "Error: Max atoms (%d) exceeded.\n", MAXATOMS);
            exit(1);
        }
    }

    if(natoms == 0) {
        fprintf(stderr, "Error: zero atoms read.\n");
        exit(1);
    }

    nAtoms = natoms;
    if (verbose) {
        fprintf(stdout,"%d atoms\n", nAtoms);
        fflush(stdout);
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readbondfile:      Read .b connectivity file
 *
 *  Errors:     unable to open .b file
 *              MAXBONDS exceeded
 *
 *  Notes:      Instead of making each bond into two half bonds it
 *              would be faster to call it one, give it two colors,
 *              and have the updatebonds routine change colors halfway
 *              through.
 *
 *  Revisions:     Initial   TCP               8-jan-87
 *
 */

readbondfile() {
    register int nbonds;
    register FILE *fptr;
    char linebuf[100];
    int a1,a2;

    if(STREQ(bfile,"")) { dobonds = FALSE; return; }

#ifdef DOREND
    if((clipfunction[0].ptr == NULL) && (!nospheres)) {
        dobonds = FALSE;  /* no need to do bonds; wouldn't be visible */
        return;
    }
    else dobonds = TRUE;
#else
    dobonds = TRUE;
#endif

    fptr = fileopen(bfile,"r");
    nbonds = 0;
    while((fgets(linebuf,sizeof(linebuf),fptr)) != NULL) {
        sscanf(linebuf,"%d %d",&a1,&a2);
        sticks[nbonds].p1.x = x9bit(atoms[a1].x);
        sticks[nbonds].p1.y = y9bit(atoms[a1].y);
        sticks[nbonds].p1.z = atoms[a1].z;
        sticks[nbonds+1].p1.x = x9bit(atoms[a2].x);
        sticks[nbonds+1].p1.y = y9bit(atoms[a2].y);
        sticks[nbonds+1].p1.z = atoms[a2].z;

        sticks[nbonds].p2.x = (x9bit(atoms[a1].x) + x9bit(atoms[a2].x))>>1;
        sticks[nbonds].p2.y = (y9bit(atoms[a1].y) + y9bit(atoms[a2].y))>>1;
        sticks[nbonds].p2.z = (atoms[a1].z + atoms[a2].z)>>1;
        sticks[nbonds+1].p2.x = sticks[nbonds].p2.x;
        sticks[nbonds+1].p2.y = sticks[nbonds].p2.y;
        sticks[nbonds+1].p2.z = sticks[nbonds].p2.z;

        sticks[nbonds].cindex = atoms[a1].cindex;
        sticks[nbonds+1].cindex = atoms[a2].cindex;

      /* Swap so that p1 contains the smallest y coordinate (screen) */
        if(sticks[nbonds].p1.y > sticks[nbonds].p2.y)
            swappoints(sticks[nbonds].p1,sticks[nbonds].p2);
        if(sticks[nbonds+1].p1.y > sticks[nbonds+1].p2.y)
            swappoints(sticks[nbonds+1].p1,sticks[nbonds+1].p2);

        nbonds += 2;
        if(nbonds >= MAXBONDS) {
            fprintf(stderr, "Error: Max bonds (%d) exceeded.\n", MAXBONDS);
            exit(1);
        }
    }
    if(nbonds == 0) { fprintf(stderr, "Error: zero bonds read.\n"); exit(1); }
    nBonds = nbonds;
    if (verbose) {
        fprintf(stdout, "%d bonds\n", (nBonds/2));
        fflush(stdout);
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readcmapfile:   Override default colormap
 *
 *  Errors:     Can't open colormap file
 *
 *  Notes:      Format of a colormap file (starting in col 1):
 *                      Atype r g b
 *              where atype is the atom type and r,g,b are normalized.
 *              Comments are denoted by '#' in column 1.
 *
 *  Revisions:     Initial   TCP             14-jan-87
 *
 */

readcmapfile() {

    FILE *fptr;
    NCOLOR clr;
    register int i,cnt=0;
    char linebuf[80],atype[40];

    if(STREQ(cmapfile,""))  /* no map given in parameter file */
        return;

    fptr = fileopen(cmapfile,"r");
    while((fgets(linebuf,sizeof(linebuf),fptr)) != NULL) {
        if((linebuf[0] == '#') || (linebuf[0] == '\n')) continue;
        sscanf(linebuf,"%s %f %f %f",atype,&clr.r,&clr.g,&clr.b);
        switch(atype[0]) {
          case 'C': case 'c': i = 0;  break;   /* carbon */
          case 'O': case 'o': i = 1;  break;   /* oxygen */
          case 'N': case 'n': i = 2;  break;   /* nitrogen */
          case 'X': case 'x': i = 3;  break;   /* xenon */
          case 'H': case 'h': i = 4;  break;   /* hydrogen */
          case 'S': case 's': i = 5;  break;   /* sulfer */
          case 'P': case 'p': i = 6;  break;   /* phosphorus */
          case 'U': case 'u': i = 7;  break;   /* copper */
          case 'I': case 'i': i = 8;  break;   /* iodine */
          case 'Z': case 'z': i = 9;  break;   /* zinc */
          case 'F': case 'f': i = 10; break;   /* flourine */
          default:            i = 11; break;   /* unknown */
        }
        cmap[i].r = clr.r;  cmap[i].g = clr.g;  cmap[i].b = clr.b;
        if(++cnt == MAXCOLORS) {
            fprintf(stderr,"Error: max color table entries (%d) exceeded.\n",
                    MAXCOLORS);
            exit(1);
        }
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  rotatoms:   Rotate the atomic coordinates about center
 *
 *  Notes:          Adapted from Mike Pique's "atoss.c"
 *
 *  Revisions:      Initial   TCP                8-jan-87
 *
 */

rotatoms() {

    POINT t1, t2;
    register int i;

    for(i=0; i<nAtoms; i++) {
        t1.x = atoms[i].x - center.x;
        t1.y = atoms[i].y - center.y;
        t1.z = atoms[i].z - center.z;
        t2.x = matrix[0][0]*t1.x + matrix[0][1]*t1.y + matrix[0][2]*t1.z;
        t2.y = matrix[1][0]*t1.x + matrix[1][1]*t1.y + matrix[1][2]*t1.z;
        t2.z = matrix[2][0]*t1.x + matrix[2][1]*t1.y + matrix[2][2]*t1.z;
        atoms[i].x = t2.x + center.x;
        atoms[i].y = t2.y + center.y;
        atoms[i].z = t2.z + center.z;

        /* Check for max/min Z bounds */
        if((atoms[i].z - atoms[i].rad) < win_min.z)
            win_min.z = atoms[i].z - atoms[i].rad;
        if((atoms[i].z + atoms[i].rad) > win_max.z)
            win_max.z = atoms[i].z + atoms[i].rad;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  winatoms:   Apply the windowing specified in the .par file
 *              Note that z is handled with auto-windowing.
 *
 *  Notes:          Adapted from Mike Pique's "atoss.c"
 *
 *  Revisions:      Initial   TCP                8-jan-87
 *
 */

winatoms() {

    register int i;
    FLOAT scale;
    FLOAT mx,my;
    FLOAT winsize,fudge;

    winsize = win_max.x - win_min.x;  /* assumed square in x,y and z */
    scale = (((long)XPIXELS) * UNITY)/winsize;

  /* Compute Z windowing separately.  Win_min.z has  */
  /* been set to the minimum Z for this input file.  */
  /* We'll use this value to translate all Z values. */
  /* Add a fudge factor. (Taking proberadius into account.) */
    if(enhanced) fudge = FUDGEFACTOR + proberadius;  else fudge = FUDGEFACTOR;
    win_max.z += (fudge * ATOMSCALE);

#ifdef DOPREP
  /* write window information to header of depth buffer output file */
    writeheader(winsize);
#endif

  /* Compute Il normalized distance for depth cueing */
    depthwindow = Il/(win_max.z * scale);

    mx = (win_min.x + win_max.x)/2.0;
    my = (win_min.y + win_max.y)/2.0;

    for(i=0; i<nAtoms; i++) {
        atoms[i].x = (atoms[i].x - mx) * scale;
        atoms[i].y = (atoms[i].y - my) * scale;
#ifdef DOREND
        atoms[i].z = (atoms[i].z * scale) - (win_max.z * scale);
#else
        atoms[i].z = -((atoms[i].z * scale) - (win_max.z * scale));
#endif
        atoms[i].rad = atoms[i].rad * scale;
    }

  /* scale installed clipping plane slope and z values */
    for(i=0; i<clipnum; i++) {
        clipfunction[i].slope *= scale;
        clipfunction[i].z *= scale;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clipatoms:        Check atomic coordinates against x,y screen bounds
 *
 *  Notes:        Adapted from Mike Pique's "atoss.c"
 *
 *  Revisions:     Initial   TCP                 8-jan-87
 *
 */

clipatoms() {

    register int i, bound, cnt=0;

    bound = XPIXELS*UNITY/2;

    for(i=0; i<nAtoms; i++) {
      /* Check against x,y screen bounds */
        if ((atoms[i].x + atoms[i].rad) < -bound ||
            (atoms[i].x - atoms[i].rad) >  bound ||
            (atoms[i].y + atoms[i].rad) < -bound ||
            (atoms[i].y - atoms[i].rad) >  bound) {
            cnt++;
            atoms[i].cindex = CLIPPED;  /* Mark as clipped */
        }
    }

    if(verbose) {
        fprintf(stdout, "%d atoms clipped\n",cnt);
        fflush(stdout);
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  sortatoms:  Sort atoms by Y and radius
 *
 *  Notes:      Adapted from Mike Pique's "atoss".
 *
 *  Revisions:     Initial   TCP                 8-jan-87
 *
 */

sortatoms() {

    int s_compare();

    qsort(atoms, nAtoms, sizeof(INFORMAT), s_compare);
}
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  s_compare:  Qsort comparison routine
 *
 *  In:         p,q - pointers to structures to compare
 *
 *  Out:        -1,0,1 - indication of order (see qsort manual)
 *
 *  Notes:      Adapted from Mike Pique's "atoss".
 *
 *  Revisions:     Initial   TCP              8-jan-87
 *
 */

s_compare(p,q)
register INFORMAT *p, *q;
{
    if (y9bit(p->y+p->rad) <  y9bit(q->y+q->rad)) return(-1);
    if (y9bit(p->y+p->rad) == y9bit(q->y+q->rad)) return(0);
    return(1);
}
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  sortbonds:  Sort bonds by Y
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP              22-jan-87
 *
 */

sortbonds() {

    int b_compare();

    if(nBonds <= 1) return;
    qsort(sticks, nBonds, sizeof(INSTICKS), b_compare);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  b_compare:  Qsort comparison routine
 *
 *  In:         p,q - pointers to structures to compare
 *
 *  Out:        -1,0,1 - indication of order (see qsort manual)
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 8-jan-87
 *
 */

b_compare(p,q)
register INSTICKS *p, *q;
{
    if (p->p1.y <  q->p1.y) return(-1);
    if (p->p1.y == q->p1.y) return(0);
    return(1);
}

 /* end prep.c */
```

```
/*
 *      render.h:  Header file for render.
 */

/* :::  Include Files  ::::::::::::::::::::::::::::::::::::::::::::: */

#include <math.h>


/* :::  Defined constants  :::::::::::::::::::::::::::::::::::::::: */

#ifdef DOREND                      /* Various constants from Porter's XPT */
#define XPIXELS  512
#define YPIXELS  512
#define UNITY     64
#define HALFUNITY 32
#define logUNITY   6
#else
#define XPIXELS  256
#define YPIXELS  256
#define UNITY    128
#define HALFUNITY 64
#define logUNITY   7
#endif

#ifdef BACKTOFRONT      /* needed for anti-aliasing/transparency/clipping */
#define ISAFTER      <
#define LASTINORDER (-32768)
#else                   /* faster */
#define ISAFTER      >
#define LASTINORDER (32768)
#endif

#define MAXATOMS  1300
#define MAXBONDS  3000
#define MAXCOLORS 12
#define INFINITY (32767)
#define MAXZ     (16384)

#define CLIPPED     -1
#define SPHERE_DONE -1

#define BOND_Y_DONE   -2
#define BOND_DONE   -1

#define X_SWITCH 1   /* 3-D Bresenham interchange flags */
#define Y_SWITCH 2
#define Z_SWITCH 3

#ifdef DOPREP
#define BACKSURFRAMP 7
#define RAMPSIZE     32
#define BOND_RAMPVAL 28
#endif


/* :::  Typedefs  ::::::::::::::::::::::::::::::::::::::::::::::  */

typedef struct {
    FLOAT x,y,z;
} POINT;

typedef struct {
    long x,y,z;
} INTPOINT;

typedef struct {
    long  x,y,z;         /* atom center */
    short  rad;          /* atom radius */
    int cindex;          /* colormap index */
} INFORMAT;

typedef struct {
    INTPOINT p1,p2;      /* top/bottom point */
    int cindex;          /* color table index */
} INSTICKS;

typedef struct {         /* Standard 0-255 RGB color */
    uchar r,g,b,fill;
} COLOR;

typedef struct {         /* Normalized RGB color */
    FLOAT r,g,b;
} NCOLOR;

typedef struct sphere {
    int xcen;            /* x at center */
    int zcen;            /* z at center */
    int rad;             /* radius */
    int xrp;             /* x at right perimeter */
    int xlp;             /* x at left perimeter */
    int yp;              /* y at perimeter */
    int drp;             /* delta right */
    int dlp;             /* delta left */
    int cindex;          /* colormap index */
    struct sphere *next; /* ptr to next sphere */
} SPHERE;

typedef struct bond {
    int x,y,z;                   /* current coordinates */
    int dy,dz;                   /* Bresenham's error terms */
    int xsign,ysign,zsign;       /* iteration amounts */
    int yincl,yinc2;             /* y delta increments */
    int zincl,zinc2;             /* z delta increments */
    int cnt;                     /* iterations remaining */
    int interc;                  /* deltas interchanged? */
    int cindex;                  /* bond color (color table index) */
    struct bond *next;           /* next in linked list */
} BOND;


/* :::  Globals  ::::::::::::::::::::::::::::::::::::::::::::::::  */

INFORMAT atoms[MAXATOMS];  /* Input atom list */
INSTICKS sticks[MAXBONDS]; /* Input bonds list */
SPHERE spheres[MAXATOMS];  /* Rendering atom list */
BOND  bonds[MAXBONDS];     /* Rendering bond list */

SPHERE activesphere;    /* head of linked list of active spheres */
SPHERE availsphere;     /* head of linked list of available spheres */
SPHERE newsphere;       /* head of linked list of newly active spheres */

BOND activebond;        /* head of linked list of active bonds */
BOND availbond;         /* head of linked list of available bonds */
BOND newbond;           /* head of linked list of newly active bonds */

int nAtoms;             /* number of input atoms */
int nBonds;             /* number of input bonds */

bool dobonds;           /* TRUE: render bonds */
bool depthcue;          /* TRUE: do depth cueing */
bool inherit;           /* TRUE: hydrogens inherit previous atom color */
#ifdef DOREND
bool nospheres;         /* TRUE: don't render spheres */
#endif

#ifdef DOREND
COLOR scanbuf[XPIXELS];   /* scan conversion buffer */
```

```
long  scandepth[XPIXELS]; /* z depth buffer */
#endif
int   overlap[XPIXELS];   /* portion of border overlapping a sphere */

COLOR bgc;              /* Background color */
NCOLOR lclr;            /* light color */

#ifdef DOREND
FILE *ofptr;            /* Output raster file pointer */
#else
FILE *pfptr;            /* Output depth buffer file pointer */
#endif

POINT light;            /* point light source */
bool  phong;            /* Flag: if TRUE do Phong shading */
FLOAT ambient;          /* ambient light intensity */
FLOAT Ia;               /* incident ambient light intensity */
FLOAT Il;               /* incident intensity from light source */
FLOAT Ka;               /* ambient diffuse reflection constant */
FLOAT Kd;               /* diffuse reflection constant */
FLOAT Ks;               /* Phong reflectance curve constant */
FLOAT K;                /* distance constant (depth cueing) */
int   Kn;               /* Phong cosine power */
FLOAT depthwindow;      /* Il over normalized Z distance (depth cueing) */

#if (defined STATS) && (defined DOPREP)
bool stats;             /* TRUE: write depth buffer statistics */
long maxdvl,mindvl;     /* max/min depth vector lengths */
long totaldexels;       /* total dexels in depth buffer */
long totaldv;           /* total non-empty depth vectors in depth buffer */
#endif

extern NCOLOR cmap[];   /* color table */


/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::  */

/*
 * We need macros to convert from the 16 bit coordinate system
 * of the original data to the 9 bit coordinate system of the
 * frame buffer. 'x9bit' maps x from [-16K,16K] to [0,511].  'y9bit'
 * maps y from [-16K,16K] to [511,0]. (Adapted from Tom Porter's XPT.)
 */

#ifdef DOREND
#define x9bit(a)  ((((a)+MAXZ)>>logUNITY) )
#define y9bit(a)  (511 - (((a)+MAXZ)>>logUNITY))
#else
#define x9bit(a)  ((((a)+MAXZ)>>logUNITY) )
#define y9bit(a)  (255 - (((a)+MAXZ)>>logUNITY))
#endif

/*
 * We need to be able to round a 16 bit value to the center
 * (in 16bit space) of what will be the pixel in the frame buffer.
 * (Adapted from Tom Porter's XPT.)
 */
#define round(a)  (((a) & (-UNITY)) | (UNITY/2))

#define interval(x,d,y) \
        ((x)<(d)?0 : (d)<(y)? UNITY : (UNITY*((x)-(d)))/((x)-(y)) )

#define sq(a)   ((a)*(a))
#define swap(a,b) { register int t; t=a; a=b; b=t; }

#define scalarmult(t,s,v)  { t.x=(s)*v.x; t.y=(s)*v.y; t.z=(s)*v.z; }
#define vectoradd(t,v1,v2) { t.x=v1.x+v2.x; t.y=v1.y+v2.y; t.z=v1.z+v2.z; }
#define normalize(v) {  \
    FLOAT norm;         \
    norm = sqrt(sq(v.x) + sq(v.y) + sq(v.z));  \
    v.x /= norm;        \
    v.y /= norm;        \
    v.z /= norm;        \
}

#define dotprod(a,b)    ((a.x*b.x) + (a.y*b.y) + (a.z*b.z))

#define scaleclr(nc,c) {          \
    c.r=(uchar)(255.0*nc.r);    \
    c.g=(uchar)(255.0*nc.g);    \
    c.b=(uchar)(255.0*nc.b);    \
 }


/* :::  Externals  ::::::::::::::::::::::::::::::::::::::::::::  */

extern COLOR shade();


  /* end render.h */
```

```
#endif
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        render.c:  contains:

    >>>>    scan:
    >>>>    bufinit:

                                        Thomas C. Palmer
                                        10-jan-87
*/

/* $Header: render.c,v 1.7 87/04/06 19:53:02 palmer Exp $ */
static char rcsid[] = "$Header: render.c,v 1.7 87/04/06 19:53:02 palmer Exp $";

#include "common.h"
#include "standard.h"
#include "config.h"
#include "render.h"
#include "write.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  scan:       Scan spheres and bonds into scanline buffer and write
 *
 *  Notes:      Adapted from Tom Porter's XPT.
 *
 *  Revisions:     Initial   TCP                10-jan-87
 *
 */

scan() {

    register int scanline;

#ifdef DOREND
    if(!nospheres)
#endif
        linkspheres();     /* Initialize sphere linked lists */
    if (dobonds)
        linkbonds();   /* Initialize bonds linked lists */
    writeinit();       /* Initialize writes */
    shadeinit();       /* Initialize shading */

  /* Get spheres/bonds already active on top scanline */
#ifdef DOREND
    if(!nospheres)
#endif
        getspheres(YPIXELS-1,nAtoms);
    if (dobonds) getbonds(YPIXELS-1,nBonds);

    for(scanline=0; scanline<YPIXELS; scanline++) {
        bufinit();                   /* Initialize scanline buffers */
#ifdef DOREND
        if(!nospheres)
#endif
            getspheres(scanline,nAtoms); /* Add any new spheres to new list */
#ifdef DOREND
        if(!nospheres)
#endif
            scanspheres(scanline);       /* scan convert active spheres */
        if (dobonds) {               /* same for bonds */
            getbonds(scanline,nBonds);
            scanbonds(scanline);
        }
        writescan(scanline);         /* write the scanline */
    }
#if (defined STATS) && (defined DOPREP)
    if(stats) writestats();
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  bufinit:    Initialize scanline buffers to default values.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               10-jan-87
 *
 */

bufinit() {
    register int scanpix;

#ifdef DOPREP
    for(scanpix=0; scanpix<XPIXELS; scanpix++) {
        scandexel[scanpix].next = NULL;
        scandexel[scanpix].dexel = 0;     /* dexel count reset to zero */
    }
#else
    for(scanpix=0; scanpix<XPIXELS; scanpix++) {
        scanbuf[scanpix] = bgc;
        scandepth[scanpix] = INFINITY;
    }
#endif
}


 /* end render.c */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

      sphere.c:  contains:

    >>>>    getspheres:
    >>>>    loadsphere:
    >>>>    scanspheres:
    >>>>    updatesphere:
    >>>>    line:
    >>>>    dot:
    >>>>    shade:
    >>>>    delta:
    >>>>    shadeinit:
    >>>>    linkspheres:

    The majority of these routines were adapted from Tom Porter's XPT.

                                        Thomas C. Palmer
                                        10-jan-87
*/

/* $Header: spheres.c,v 1.5 87/04/06 19:53:08 palmer Exp $ */
static char rcsid[] = "$Header: spheres.c,v 1.5 87/04/06 19:53:08 palmer Exp $";

#include "common.h"
#include "standard.h"
#include "config.h"
#include "render.h"
#include "write.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  getspheres:
 *
 *  In:         scanline - current scanline
 *              natoms   - total input atoms
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                10-jan-87
 *
 */

getspheres(scanline,natoms)
register int scanline, natoms;
{
    static int next=0;  /* Next atom to be moved from atoms array to avail */
    int top;

    if(next == natoms) return;  /* eof */

    while(TRUE) {
        if(atoms[next].cindex == CLIPPED) {
            if(++next == natoms) return;
            continue;
        }
        top = y9bit(atoms[next].y + atoms[next].rad);
        if(top < 0) top = YPIXELS-1;
        if(top != scanline) return;
        loadsphere(scanline, atoms[next]);
        if(++next == natoms) return;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  loadsphere: Add the atom to the active sphere list
 *
 *  In:         scanline - current scanline
 *              atom     - atom data to add to new list
 *
 *  Errors:     Maximum active spheres exceeded
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                10-jan-87
 *
 */

loadsphere(scanline, atom)
register int scanline;
INFORMAT atom;
{
    register SPHERE *p, *q;
    int peak, scanpeak, x, y;

#ifdef DOPREP  /* Clip atoms whose Z center ISAFTER LASTINORDER */
    if(atom.z ISAFTER LASTINORDER) {
        fprintf(stdout,"Warning: Atom completely clipped in Z.\n");
        fflush(stdout);
        return; /* DON'T sort into newsphere list */
    }
#endif

    p = availsphere.next;
    if(p == &availsphere) {
        fprintf(stderr,"Error: max number of active spheres exceeded.\n");
        exit(1);
    }

    p->rad = atom.rad;
    p->xcen = atom.x;
#ifdef DOREND
  /* Make left-handed here for srend */
    p->zcen = -atom.z;
#else
  /* Don't make left-handed here (handled previously in winatoms) */
    p->zcen = atom.z;
#endif
    p->cindex = atom.cindex;

    peak = atom.y + atom.rad;
    x = round(atom.x) - atom.x;
    y = round(peak) - atom.y;
    p->xlp = x;
    p->xrp = x;
    p->yp = y;
    p->dlp = delta(x-HALFUNITY, y-HALFUNITY, atom.rad);
    p->drp = delta(x+HALFUNITY, y-HALFUNITY, atom.rad);

    scanpeak = y9bit(peak);
    if(scanline == YPIXELS-1) {    /* already active on top scanline */
        do {
            if(updatesphere(p,scanline,FALSE) == SPHERE_DONE) return;
        } while(++scanpeak != 0);
    }
    else
        if (scanpeak != scanline) {  /* sorting failure */
            fprintf(stderr, "Sorting failure?  Current line: %d.\n", scanline);
            fprintf(stderr, "Atom %d %d %d.\n",atom.x,atom.y,atom.z);
            exit(1);
        }
    availsphere.next = p->next;

  /* Sort into newsphere list by Z */
    q = &newsphere;
    while(p->zcen ISAFTER q->next->zcen)
        q = q->next;

    p->next = q->next;
    q->next = p;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  scanspheres: Scan convert the currently active spheres.
 *
 *  In:         scanline - current scanline
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                10-jan-87
 *
 */

scanspheres(scanline)
int scanline;
{
    register SPHERE *q, *qq, *pp;

    q = &activesphere;    pp = newsphere.next;
    while((qq = q->next) != &activesphere || pp != &newsphere) {
        if(qq->zcen ISAFTER pp->zcen) { /* merge new sphere into active list */
            q->next = pp;
            pp = pp->next;
            q->next->next = qq;
            qq = q->next;
        }
        if(updatesphere(qq,scanline,TRUE) == SPHERE_DONE) {
            q->next = qq->next;           /* remove from active list */
            qq->next = availsphere.next;
            availsphere.next = qq;
        }
        else q = q->next;
    }
    newsphere.next = &newsphere;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  updatesphere:       Use Bresenham's circle algorithm to scan
 *                      the silhouette of the sphere in the X-Y plane.
 *
 *  In:         p        - ptr to sphere
 *              scanline - current scanline
 *              doscan   - flag: if TRUE scan into scanline buffer
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               10-jan-87
 *
 */

/* Globals used by updatesphere, line and dot */
int firstleft, firstright, lastleft, lastright;
int leftedge, rightedge, thisedge;

updatesphere(p,scanline,doscan)
register SPHERE *p;
register int scanline;
bool doscan;
{
    register int rptr, lptr;
    int stat = TRUE;

    rptr = x9bit(p->xrp);
    lptr = x9bit(p->xlp);

    if(p->yp > 0) {    /* y at perimeter is positive; top half */
        firstright = p->xrp;
        while(TRUE) {  /* move right perimeter to right */
            overlap[rptr] = interval(2*p->xrp, p->drp, -2*p->yp);
            if(p->drp >= 0) break;
            rptr++;
            p->xrp += UNITY;
            p->drp += 2*p->xrp;
        }
        lastright = p->xrp;
        lastleft =  p->xlp;
        while(TRUE) {  /* move left perimeter to left */
            overlap[lptr] = interval(-2*p->xlp, p->dlp, -2*p->yp);
            if(p->dlp >= 0) break;
            lptr--;
            p->xlp -= UNITY;
            p->dlp -= 2*p->xlp;
        }
        firstleft = p->xlp;
    }
    else {              /* bottom half */
        if((p->yp+UNITY) > 0) {   /* midline */
            p->drp = delta(p->xrp-HALFUNITY, p->yp-HALFUNITY, p->rad);
            p->dlp = delta(p->xlp+HALFUNITY, p->yp-HALFUNITY, p->rad);
        }
        lastright = p->xrp;
        while(TRUE) {  /* move right perimeter to left */
            overlap[rptr] = interval(-2*p->yp, p->drp, -2*p->xrp);
            if(p->drp < 0) break;
            if((p->xrp-UNITY) < 0) break;
            rptr--;
            p->xrp -= UNITY;
            p->drp -= 2*p->xrp;
        }
        firstright = p->xrp;
        firstleft = p->xlp;
        while(TRUE) {  /* move left perimeter to right */
            overlap[lptr] = interval(-2*p->yp, p->dlp, 2*p->xlp);
            if(p->dlp < 0) break;
            if((p->xlp+UNITY) >= 0) {  /* sphere done */
                stat = SPHERE_DONE;
                break;
            }
            lptr++;
            p->xlp += UNITY;
            p->dlp += 2*p->xlp;
        }
        lastleft = p->xlp;
    }

    if (doscan) line(p,scanline);
    p->yp -= UNITY;
    p->drp -= 2*p->yp;
    p->dlp -= 2*p->yp;
    return(stat);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  line:       Use Bresenham's circle algorithm to trace the
 *              semi-circular arc of the sphere in the X-Z plane.
 *
 *  In:         p        - ptr to sphere
 *              scanline - current scanline
 *  Notes:
 *
 *  Revisions:     Initial   TCP                11-jan-87
 *
 */

line(p, scanline)
register SPHERE *p;
register int scanline;
{
    register int x, z, del, r, scanpix;

    x = firstleft;
    z = 0;
    r =  lastright - HALFUNITY + overlap[x9bit(lastright)];
    r -= firstleft + HALFUNITY - overlap[x9bit(firstleft)];
    r >>= 1;
    scanpix =   x9bit(p->xcen + firstleft);
    leftedge =  x9bit(lastleft);
    rightedge = x9bit(firstright);
    thisedge =  x9bit(firstleft);

    if(firstleft == firstright) {  /* too small to see */
        scanpix = dot(p,0,scanpix,scanline,x);
        return;
    }

    del = delta(x+HALFUNITY, z+HALFUNITY, r);
    while(x < 0) {  /* left side */
        while(del < 0) {  /* move forward */
            z += UNITY;
            del += 2*z;
        }
        scanpix = dot(p,z-HALFUNITY+interval(2*z,del,2*x),scanpix,scanline,x);
        x += UNITY;
        del += 2*x;
    }
    if (z == 0) return;
    del = delta(x+HALFUNITY, z-HALFUNITY, r);
    while(TRUE) {  /* right side */
        scanpix = dot(p,z-HALFUNITY+interval(2*x,del,-2*z),scanpix,scanline,x);
        while(del >= 0) {  /* move backward */
            z -= UNITY;
            if(z == 0) return;
            del -= 2*z;
        }
        x += UNITY;
        del += 2*x;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  dot:        Determine a pixel/dexel shade and either write
 *              the pixel or add the dexel to the depth buffer.
 *
 *  In:         p       - ptr to sphere
 *              z       - z value of pixel/dexel
 *              scanpix - index within scanline
 *              scanline - current scanline
 *
 *  Out:        next index within scanline
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                11-jan-87
 *
 */

dot(p,z,scanpix,scanline,x)
register SPHERE *p;
int z, scanpix, scanline, x;
{
#ifdef DOPREP
    COLOR clr;
#endif
    register long depth1,depth2;

    if(scanpix < 0 || scanpix >= XPIXELS) {  /* clip to screen */
        thisedge++;
        return(scanpix+1);
    }

    depth1 = p->zcen - z;

#ifdef DOPREP                     /* link into depth vector */
    depth2 = p->zcen + z;
    if(depth1 > depth2) swap(depth1,depth2);
    clr = shade(p,z,scanpix,FRONT,NULL);
    linkdexel(p,clr.r,STAG,depth1,scanpix,FRONT);
    clr = shade(p,z,scanpix,BACK,NULL);
    linkdexel(p,clr.r,STAG,depth2,scanpix,BACK);
#else
    if(depth1 <= scandepth[scanpix]) {
        if(!clipped(depth1,scanpix,scanline)) {
            scandepth[scanpix] = depth1;
            scanbuf[scanpix] = shade(p,z,x,FRONT,NULL);
        }
        /* else scanbuf[scanpix] = shade(p,z,x,FRONT,CLIPPED); */
    }
#endif
    return(scanpix+1);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  shade:      srend:  return a 24 bit color
 *              sprep:  return an 8 bit color in the red byte
 *
 *  In:         p       - ptr to sphere
 *              z       - z value of pixel/dexel
 *              x       - index within scanline
 *              surf    - flag: front or back sphere surface (sprep)
 *              clipit  - flag: TRUE if pixel is clipped (srend)
 *
 *
 *  Out:        pixel/dexel color
 *
 *  Revisions:     Initial   TCP              13-jan-87
 *
 */

COLOR shade(p,z,x,surf,clipit)
register SPHERE *p;
int z,x,clipit;
bool surf;
{
    COLOR cshade;
    POINT normal;
    FLOAT dotp;
    FLOAT tmp;
    FLOAT Il_over_dist;
#ifdef DOREND
    NCOLOR nshade;  /* normalized colors */
    POINT tmpv1,tmpv2;
    static POINT eye = {0.0, 0.0, -1.0};
#endif
#ifdef DOPREP
    int rampoffset;
#endif

#ifdef DOREND
    nshade.r = cmap[p->cindex].r * ambient;
    nshade.g = cmap[p->cindex].g * ambient;
    nshade.b = cmap[p->cindex].b * ambient;

    if(clipit == CLIPPED) {  /* just use ambient for clipped pixels */
        /* scaleclr(nshade,cshade);    /* Scale color */
        cshade.r = 25;  cshade.g = 25;  cshade.b = 25;
        return(cshade);
    }
#endif

    if(surf == FRONT)
         { normal.x =  x;  normal.y =   p->yp;   normal.z = z; }
    else { normal.x = -x;  normal.y = -(p->yp);  normal.z = z; }

    normalize(normal);
    dotp = dotprod(normal,light);

  /* Depth cue kludge.  Needs work. depthwindow set in winatoms (prep.c) */
  /* Currently depends on Il being equal to one. */
    if(depthcue) {
        tmp = ((-p->zcen + z) + K) * depthwindow;
        if(tmp == 0)  Il_over_dist = Il; else Il_over_dist = Il/tmp;
        Il_over_dist = -Il_over_dist + 1;
    }
    else Il_over_dist = Il;

#ifdef DOPREP
    if(surf != BACK) {  /* Don't depth cue back surface */
        dotp *= Il_over_dist;
    }
    if(dotp <= 0.0)
        cshade.r = 0;
    else cshade.r = (uchar) (dotp * ((FLOAT)RAMPSIZE));

    if(surf == BACK) {
        rampoffset = BACKSURFRAMP;
       /* leave 2 values in cmap for background colors */
        if(cshade.r > 29) cshade.r = 29;
    }
    else rampoffset = p->cindex;

  /* Clip to upper limit */
    if(cshade.r >= RAMPSIZE) {
        cshade.r = RAMPSIZE-1;
    }

    cshade.r += (rampoffset*RAMPSIZE);
    return(cshade);
#else
    if(dotp > 0.0) {
        dotp *= Kd * Il_over_dist;
        nshade.r += cmap[p->cindex].r * dotp;
        nshade.g += cmap[p->cindex].g * dotp;
        nshade.b += cmap[p->cindex].b * dotp;
    }

    if(phong) {  /* Do Phong shading */
        scalarmult(tmpv1,-2.0,normal);
        vectoradd(tmpv2,light,tmpv1);
        normalize(tmpv2);
        dotp = dotprod(eye,tmpv2);
        if(dotp > 0.0) {
            dotp = Ks * pow(dotp, (FLOAT)Kn) * Il_over_dist;
            nshade.r += lclr.r * dotp;
            nshade.g += lclr.g * dotp;
            nshade.b += lclr.b * dotp;
        }
    }

  /* Clip colors */
    if(nshade.r > 1.0) nshade.r = 1.0;  if(nshade.r < 0.0) nshade.r = 0.0;
    if(nshade.g > 1.0) nshade.g = 1.0;  if(nshade.g < 0.0) nshade.g = 0.0;
    if(nshade.b > 1.0) nshade.b = 1.0;  if(nshade.b < 0.0) nshade.b = 0.0;

    scaleclr(nshade,cshade);    /* Scale color */

    return(cshade);
#endif
}
```

```c
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  delta:      Compute Bresenham circle delta value
 *
 *  In:         x - sphere x
 *              y - sphere y
 *              r - sphere r
 *
 *  Out:            delta value
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                10-jan-87
 *
 */

delta(x,y,r)
int x,y,r;
{
    long nx,ny,nr;

    nx = x;  ny = y;  nr = r;
    return((int)((sq(nx) + sq(ny) - sq(nr))/UNITY));
}
```

```c
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  shadeinit:  Initialize shading parameters
 *
 *  Revisions:     Initial   TCP                13-jan-87
 *
 */

shadeinit() {

    ambient = Ia * Ka;
    if(ambient > 1.0) ambient = 1.0;  if(ambient < 0.0) ambient = 0.0;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  linkspheres: Initialize linked lists and link the sphere
 *               array into an circular available list.
 *
 *  Notes:       Adapted from Tom Porter's XPT.
 *
 *  Revisions:      Initial   TCP               10-jan-87
 *
 */

linkspheres() {
    register int i;
    register SPHERE *p, *q;

    activesphere.next = &activesphere;
    newsphere.next = &newsphere;
    activesphere.zcen = newsphere.zcen = LASTINORDER;

    p = &availsphere;
    for(i=0; i<MAXATOMS; i++) {
        q = &spheres[i];
        p->next = q;
        p = q;
    }
    p->next = &availsphere;
}


 /* end sphere.c */
```

```c
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        bonds.c:  contains:

    >>>>    getbonds:
    >>>>    loadbonds:
    >>>>    scanbonds:
    >>>>    updatebonds:
    >>>>    sign:
    >>>>    linkbonds:
    >>>>    plot:
                                        Thomas C. Palmer
                                        22-jan-87
*/

/* $Header: bonds.c,v 1.5 87/04/06 19:52:19 palmer Exp $ */
static char rcsid[] = "$Header: bonds.c,v 1.5 87/04/06 19:52:19 palmer Exp $";

#include "common.h"
#include "standard.h"
#include "config.h"
#include "render.h"
#include "write.h"

/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  getbonds:   Get any bonds active on the current scanline and
 *              load to current bonds list.
 *
 *  In:         scanline - current scanline
 *              nsticks   - total input sticks
 *
 *  Notes:      Adapted from Tom Porter's XPT.
 *
 *  Revisions:     Initial   TCP                22-jan-87
 *
 */

getbonds(scanline,nsticks)
register int scanline, nsticks;
{
    static int next=0;  /* Next atom to be moved from atoms array to avail */
    int top;

    if(next == nsticks) return;  /* eof */

    while(TRUE) {
        if(sticks[next].cindex == CLIPPED) {
            if(++next == nsticks) return;
            continue;
        }
        top = sticks[next].p1.y;
        if(top < 0) top = YPIXELS-1;
        if(top != scanline) return;
        loadbonds(scanline, sticks[next]);
        if(++next == nsticks) return;
    }
}
```

```c
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  loadbonds:
 *
 *  In:         scanline - current scanline
 *              stick     - stick data to add to new list
 *
 *  Errors:     Maximum active bonds exceeded
 *
 *  Notes:      Adapted from Tom Porter's XPT.
 *
 *  Revisions:     Initial   TCP                22-jan-87
 *
 */

#ifdef DOPREP
#define ZINC_SCALE 15      /* Scale Z increments in Bresenham calc */
#else
#define ZINC_SCALE 5
#endif

loadbonds(scanline, stick)
register int scanline;
INSTICKS stick;
{
    register BOND *p, *q;
    int scanpeak, delta_x,delta_y,delta_z;

  /* Clip bonds whose Z value ISAFTER LASTINORDER */
    if(stick.p1.z ISAFTER LASTINORDER) {
        fprintf(stdout,"Warning: Bond completely clipped in Z.\n");
        fflush(stdout);
        return; /* DON'T sort into newbond list */
    }

    p = availbond.next;
    if(p == &availbond) {
        fprintf(stderr,"Error: max number of active bonds exceeded.\n");
        exit(1);
    }

  /* Initialize bond structure for Bresenham calculations */
    delta_x = abs(stick.p2.x - stick.p1.x);
    delta_y = abs(stick.p2.y - stick.p1.y);
    delta_z = (abs(stick.p2.z - stick.p1.z)) / ZINC_SCALE;

    if((delta_x > delta_y) && (delta_y > delta_z)) {
        p->interc = X_SWITCH;
    }
    else if((delta_x >= delta_z) && (delta_z >= delta_y)) {
        p->interc = X_SWITCH;
    }
    else if((delta_y >= delta_x) && (delta_x >= delta_z)) {
        p->interc = Y_SWITCH;
        swap(delta_x,delta_y);
    }
    else if((delta_y >= delta_z) && (delta_z >= delta_x)) {
        p->interc = Y_SWITCH;
        swap(delta_x,delta_y);
    }
    else if((delta_z >= delta_x) && (delta_x >= delta_y)) {
        p->interc = Z_SWITCH;
        swap(delta_x,delta_z);
    }
    else {
        p->interc = Z_SWITCH;
        swap(delta_x,delta_z);
    }
  /* Initialization for x */
    p->x = stick.p1.x;  p->cnt = delta_x;
    p->xsign = sign(stick.p2.x-stick.p1.x);

  /* Initialization for y */
    p->y = stick.p1.y;
    p->ysign = sign(stick.p2.y-stick.p1.y);
    p->dy = (2 * delta_y) - delta_x;
    p->yinc1 = 2 * delta_y;
    p->yinc2 = 2 * (delta_y - delta_x);

  /* Initialization for z */
#ifdef DOREND
  /* Make left-handed here for srend */
    p->z = -stick.p1.z;
#else
  /* Don't make left-handed here (handled previously in winatoms) */
    p->z = stick.p1.z;
#endif
    p->zsign = ZINC_SCALE * sign((-stick.p2.z) - p->z);
    p->dz = (2 * delta_z) - delta_x;
    p->zinc1 = 2 * delta_z;
    p->zinc2 = 2 * (delta_z - delta_x);
    p->cindex = stick.cindex;

    scanpeak = p->y;
    if(scanline == YPIXELS-1) {     /* already active on top scanline */
        do {
            if(updatebonds(p,scanline) == BOND_DONE) return;
        } while(++scanpeak != 0);
    }
    else
        if (p->y != scanline) {  /* sorting failure */
            fprintf(stderr, "Sorting failure?  Current line: %d.\n", scanline);
            fprintf(stderr, "Stick top: %d %d %d.\n",
                    stick.p1.x,stick.p1.y,stick.p1.z);
            exit(1);
        }

    availbond.next = p->next;

    /* Sort into newbond list by Z */
    q = &newbond;
    while(p->z ISAFTER q->next->z)
        q = q->next;

    p->next = q->next;
    q->next = p;
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  scanbonds:  Scan the active bonds list into the scanline buffer
 *
 *  In:         scanline - current scanline
 *
 *  Notes:      Adapted from Tom Porter's XPT.
 *
 *  Revisions:     Initial   TCP               22-jan-87
 *
 */

scanbonds(scanline)
int scanline;
{
    register BOND *q, *qq, *pp;

    q = &activebond;    pp = newbond.next;
    while((qq = q->next) != &activebond || pp != &newbond) {
        if(qq->z ISAFTER pp->z) { /* merge new bonds into active list */
            q->next = pp;
            pp = pp->next;
            q->next->next = qq;
            qq = q->next;
        }
        if(updatebonds(qq,scanline) == BOND_DONE) {
            q->next = qq->next;          /* remove from active list */
            qq->next = availbond.next;
            availbond.next = qq;
        }
        else q = q->next;
    }
    newbond.next = &newbond;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  updatebonds:        Use a modified 3-D Bresenham line routine
 *                      to scan the bond into the scanline buffer
 *
 *  In:         p        - ptr to bond
 *              scanline - current scanline
 *              doscan   - flag: if TRUE scan into scanline buffer
 *
 *  Notes:      Adapted from "3D Scan-conversion Algorithms for Voxel
 *              Based Graphics", Kaufman and Shimony, 1986 Workshop on
 *              Interactive 3D Graphics, UNC Chapel Hill
 *
 *  Revisions:     Initial   TCP                22-jan-87
 *
 */

updatebonds(p,scanline)
register BOND *p;
register int scanline;
{
    int stat=FALSE;

    plot(p,scanline);  p->cnt--;
    while(p->cnt >= 0) {

        switch(p->interc) {
          case X_SWITCH:  p->x += p->xsign;  break;
          case Y_SWITCH:  p->y += p->ysign;  stat = TRUE; break;
          case Z_SWITCH:  p->z += p->zsign;  break;
        }

        if(p->dy < 0) p->dy += p->yinc1;
        else {
            p->dy += p->yinc2;
            switch(p->interc) {
              case Y_SWITCH:
                p->x += p->xsign;  break;
              default:
                p->y += p->ysign;  stat = TRUE;
            }

        }
        if(p->dz < 0) p->dz += p->zinc1;
        else {
            p->dz += p->zinc2;
            switch(p->interc) {
              case Z_SWITCH:
                p->x += p->xsign;  break;
              default:
                p->z += p->zsign;
            }
        }
        if(stat) return(BOND_Y_DONE);
        plot(p,scanline);  p->cnt--;
    }
    return(BOND_DONE);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  sign:       Sign function for Bresenham line initialization.
 *
 *  In:         a - argument
 *
 *  Out:        {-1,0,1} if a is {<,=,>} 0
 *
 *  Revisions:     Initial   TCP                22-jan-87
 *
 */

sign(a)
register int a;
{
    if (a > 0) return(1);
    if (a < 0) return(-1);
    return(0);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  linkbonds: Initialize linked lists and link the bond
 *             array into an circular available list.
 *
 *  Notes:     Adapted from Tom Porter's XPT.
 *
 *  Revisions:    Initial   TCP              10-jan-87
 *
 */

linkbonds() {
    register int i;
    register BOND *p, *q;

    activebond.next = &activebond;
    newbond.next = &newbond;
    activebond.z = newbond.z = LASTINORDER;

    p = &availbond;
    for(i=0; i<MAXBONDS; i++) {
        q = &bonds[i];
        p->next = q;
        p = q;
    }
    p->next = &availbond;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  plot:       Write the dexel to the depth buffer scanline or
 *              the scanline buffer
 *
 *  In:         p        - ptr to bond
 *              scanline - current scanline
 *
 *  Notes:
 *
 *  Revisions:    Initial   TCP              25-jan-87
 *
 */

plot(p,scanline)
register BOND *p;
int scanline;
{
    uchar cmapaddr;
    FLOAT I1_over_dist,tmp;
#ifdef DOREND
    COLOR clr;
#endif

  /* Depth cue kludge.  Needs work. depthwindow set in winatoms (prep.c) */
  /* Currently depends on I1 being equal to one. */
    if(depthcue) {
        tmp = ((-p->z) + K) * depthwindow;
        if(tmp == 0)  I1_over_dist = I1; else I1_over_dist = I1/tmp;
        I1_over_dist = -I1_over_dist + 1;
    }
    else I1_over_dist = I1;

#ifdef DOPREP                    /* link into depth vector */
    if(I1_over_dist > 1.0) I1_over_dist = 1.0;
    if(I1_over_dist < 0.0) I1_over_dist = 0.0;
    tmp = RAMPSIZE * I1_over_dist;
    cmapaddr = ((int) tmp) + (p->cindex*RAMPSIZE);
    linkdexel(NULL,cmapaddr,BTAG,p->z,p->x,STICK);
#else
    if(p->z <= scandepth[p->x]) {
        scandepth[p->x] = p->z;
        if(!clipped(p->z,p->x,scanline)) {
            scaleclr(cmap[p->cindex],clr);
            scanbuf[p->x] = clr;
        }
    }
#endif
}

 /* end bonds.c */
```

```
/*
 *      clip.h:  Header file for clip.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#define MAXCLIP 5       /* Maximum allowed clipping planes */


/* :::  Typedefs  ::::::::::::::::::::::::::::::::::::::::::::::  */

typedef struct {
    int (* ptr)();      /* clipping function pointer */
    FLOAT world_z;      /* Z value in world coordinates */
    int z;              /* Z value for clipping function (screen coords) */
    FLOAT slope;        /* clipping plane slope */
} CLIP_PROC;


/* :::  Globals  :::::::::::::::::::::::::::::::::::::::::::::::  */

CLIP_PROC clipfunction[MAXCLIP];  /* array of clipping function ptrs */
int clipnum;                      /* current number of installed functions */

/* Clipping functions */
int clip_yon();
int clip_hither();
int clip_xin();
int clip_yin();


   /* end clip.h */
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        clip.c:  contains:

    >>>>    clipped:
    >>>>    clip_yon:
    >>>>    clip_hither:
    >>>>    clip_xin:
    >>>>    clip_yin:
                                        Thomas C. Palmer
                                        13-jan-87
*/

/* $Header: clip.c,v 1.2 87/03/18 16:43:10 palmer Exp $ */
static char rcsid[] = "$Header: clip.c,v 1.2 87/03/18 16:43:10 palmer Exp $";

#include "standard.h"
#include "config.h"
#include "render.h"
#include "clip.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clipped:    Determine if the pixel/dexel is clipped by any of
 *              the installed clipping planes
 *
 *  In:         depth    - depth of pixel/dexel
 *              scanpix  - position within scanline
 *              scanline - current scanline
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                13-jan-87
 *
 */

clipped(depth,scanpix,scanline)
long depth;
int scanpix,scanline;
{
    register int i=0;

    while(clipfunction[i].ptr != NULL && i < MAXCLIP) {
        if(((*(clipfunction[i].ptr))(depth,clipfunction[i].z,clipfunction[i].slope,scanpix,scanline)) == CLIPPED)
            return(CLIPPED);
        i++;
    }

    return(NULL);
}
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clip_yon:   Implements a yon clipping plane
 *
 *  In:         depth - depth of pixel/dexel
 *              zclip - z depth of clipping plane center
 *              slope - slope of clipping plane
 *              scanpix - position within scanline
 *              scanline - current scanline
 *
 *  Out:        flag indicating clip status
 *
 *  Revisions:     Initial   TCP                13-jan-87
 *
 */

clip_yon(depth,zclip,slope,scanpix,scanline)
long depth;
FLOAT slope;
int zclip,scanpix,scanline;
{
    if(depth > zclip)
        return(CLIPPED);
    else return(NULL);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clip_hither:   Implements a hither clipping plane
 *
 *  In:          depth - depth of pixel/dexel
 *               zclip - z depth of clipping plane center
 *               slope - slope of clipping plane
 *               scanpix - position within scanline
 *               scanline - current scanline
 *
 *  Out:         flag indicating clip status
 *
 *  Revisions:      Initial    TCP                 13-jan-87
 *
 */

clip_hither(depth,zclip,slope,scanpix,scanline)
long depth;
FLOAT slope;
int zclip,scanpix,scanline;
{
    if(depth < zclip)
        return(CLIPPED);
    else return(NULL);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clip_xin:    Implements a clipping plane parallel to the y-axis
 *               but sloped.
 *
 *  In:          depth - depth of pixel/dexel
 *               zclip - z depth of clipping plane center
 *               slope - slope of clipping plane
 *               scanpix - position within scanline
 *               scanline - current scanline
 *
 *  Out:         flag indicating clip status
 *
 *  Revisions:      Initial    TCP                 13-jan-87
 *
 */

clip_xin(depth,zclip,slope,scanpix,scanline)
long depth;
FLOAT slope;
int zclip,scanpix,scanline;
{
    scanpix -= XPIXELS/2;

    if(depth < ((scanpix*slope)+zclip))
        return(CLIPPED);
    else return(NULL);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clip_yin:   Implements a clipping plane parallel to the x-axis
 *              but sloped.
 *
 *  In:         depth - depth of pixel/dexel
 *              zclip - z depth of clipping plane center
 *              slope - slope of clipping plane
 *              scanpix - position within scanline
 *              scanline - current scanline
 *
 *  Out:        flag indicating clip status
 *
 *  Revisions:     Initial   TCP                13-jan-87
 *
 */

clip_yin(depth,zclip,slope,scanpix,scanline)
long depth;
FLOAT slope;
int zclip,scanpix,scanline;
{
    scanline -= YPIXELS/2;

    if(depth < ((scanline*slope)+zclip))
        return(CLIPPED);
    else return(NULL);
}


 /* end clip.c */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writeheader:        Write the windowing info header for DockTool.
 *                      DockTool needs to know the depth buffer window
 *                      to scale the drug molecule properly.
 *
 *  In:         winsize - window size
 *
 *  Revisions:     Initial   TCP                  2-mar-87
 *
 */

#ifdef DOPREP
writeheader(winsize)
FLOAT winsize;
{
    int buf;

    buf = (int) winsize;
    write(fileno(pfptr),&buf,sizeof(buf));
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writescan:  Write a scanline worth of pixels or depth vectors
 *
 *  In:         scanline - current scanline
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                11-jan-87
 *
 */

writescan(scanline)
int scanline;
{
#ifdef DOREND
    uchar buf[3][XPIXELS];
    register int i;
#endif

#ifdef DOPREP  /* write to depth buffer file */
    writescandexel();
    flushbuf();  /* flush remaining buffer */
    return;
#endif

#ifdef DOREND
    if(ofptr != NULL) {   /* write to raster file (ikload format) */
        for(i=0; i<XPIXELS; i++) {
            buf[0][i] = scanbuf[i].r;
            buf[1][i] = scanbuf[i].g;
            buf[2][i] = scanbuf[i].b;
        }
        write(fileno(ofptr), buf, sizeof(buf));
    }
    else {                /* write to Ikonas frame buffer */
#ifdef IKONAS
        ikpwrite(scanbuf,sizeof(scanbuf),0,scanline);
#else
        fprintf(stderr,"Ikonas frame buffer writes not in this version.\n");
        exit(1);
#endif
    }
#endif
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writescandexel:     Postprocess and write a scanline of depth vectors
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                17-jan-87
 *
 */

#ifdef DOPREP
writescandexel() {
    register int scanpix;
    DEXELNODE *tptr, *freeptr;

    for(scanpix=0; scanpix<XPIXELS; scanpix++) {
        cleanvector(&scandexel[scanpix]);
        writecount(scandexel[scanpix].dexel);
        tptr = scandexel[scanpix].next;
        freeptr = tptr;
        while(tptr != NULL) {
            if(tptr->dexel != INTERIOR)
                writedexel(tptr);
            tptr = tptr->next;
            free(freeptr);  freeptr = tptr;
        }
    }
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writedexel:         Write a dexel
 *
 *  In:         nptr - ptr to dexel node
 *
 *  Revisions:     Initial   TCP                17-jan-87
 *
 */

#ifdef DOPREP
writedexel(nptr)
DEXELNODE *nptr;
{
    writebuf[curbufsize++] = nptr->dexel;
    if(curbufsize == WRITEBUFSIZE) flushbuf();
}
#endif
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  flushbuf:   Flush the output buffer
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                17-jan-87
 *
 */

#ifdef DOPREP
flushbuf() {

    write(fileno(pfptr),writebuf,(sizeof(int)*curbufsize));
    curbufsize = 0;
}
#endif
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  cleanvector:       Post-process the depth vector to remove
 *                     dexels lying between matching front and
 *                     back dexels.
 *
 *  In:         head - ptr to the head of the depth vector
 *
 *  Errors:         Front surface has no matching back surface.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                17-jan-87
 *
 */

#ifdef DOPREP
cleanvector(head)
DEXELNODE *head;
{
    DEXELNODE *fptr,*bptr;
    int dumpvector=FALSE;

    if(head->next == NULL) return;
    fptr = head->next;
    do {
        bptr = fptr->next;
        while(bptr->sptr != fptr->sptr) {  /* find matching back dexel */
            if(bptr == NULL) {
                fprintf(stderr,"Error: can't find matching back surf.\n");
                exit(1);
            }
            if ((bptr->dexel != INTERIOR) && (bptr->front != STICK)) {
                bptr->dexel = INTERIOR;  /* mark for deletion */
                head->dexel--;           /* reduce dexel count */
            }
            bptr = bptr->next;
        }

        fptr = fptr->next;  /* find next front dexel */
        while((fptr != NULL) && (fptr->front != FRONT))
            fptr = fptr->next;

    } while(fptr != NULL);
}
#endif DOPREP
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writecount:     Write the number of dexels in a depth
 *                  vector to the depth buffer.
 *
 *  In:             cnt - size of depth vector
 *
 *  Errors:         Negative depth vector size.
 *                  Max depth vector size exceeded.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP              17-jan-87
 *
 */

#ifdef DOPREP
writecount(cnt)
int cnt;
{
#ifdef DEBUG
    if(cnt < 0) {
        fprintf(stderr, "Error: Negative depth vector size!\n");
        exit(1);
    }
#endif
#ifdef STATS
    if(stats) {
        totaldexels += cnt;
        if(cnt != 0) totaldv++;
        if(cnt > maxdvl) maxdvl = cnt;
        if((cnt < mindvl) && (cnt != 0)) mindvl = cnt;
    }
#endif
    if(cnt > MAXDEPTHVECSIZE) {
        fprintf(stderr, "Max depth vector size (%d) exceeded.\n",
                MAXDEPTHVECSIZE);
        exit(1);
    }

    writebuf[curbufsize++] = cnt;
    if(curbufsize == WRITEBUFSIZE) flushbuf();
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  linkdexel:  Load the dexel data and link it into a depth vector.
 *
 *  In:         p       - ptr to sphere
 *              cnum    - Either a colormap index (bonds) or Z value (spheres)
 *              tag     - surface tag: front or back
 *              depth   - Z depth
 *              scanpix - index within scanline
 *              surf    - flag: front or back surface
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP              17-jan-87
 *
 */

#ifdef DOPREP
linkdexel(p,cnum,tag,depth,scanpix,surf)
SPHERE *p;
int cnum;
uchar tag;
int depth,scanpix;
bool surf;
{
    DEXELNODE *nptr, *tptr;
    bool zclipped = FALSE;

  /* Clip to scanline */
    if((scanpix < 0) || (scanpix >= XPIXELS)) return;

  /* Clip to Z bounds */
    if(depth < 0) fprintf(stderr,"Warning: dexel depth < 0 (%d)\n", depth);
    if(depth > 32767) depth = 32767;

    nptr = newdexel(p,cnum,tag,depth,surf);

  /* use head dexel to count dexels in depth vector */
    scandexel[scanpix].dexel++;

  /* Search for insertion point */
    tptr = &scandexel[scanpix];
    while((tptr->next != NULL) && ((tptr->next)->z <= depth))
        tptr = tptr->next;

  /* link dexel into depth vector */
    nptr->next = tptr->next;
    tptr->next = nptr;
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  newdexel:   Allocate and load a new dexel node
 *
 *  In:         p       - ptr to sphere
 *              cmapaddr- color map address
 *              tag     - surface tag: front or back
 *              depth   - Z depth
 *              surf    - flag: front or back surface
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                17-jan-87
 *
 */

#ifdef DOPREP
DEXELNODE *newdexel(p,cmapaddr,tag,depth,surf)
SPHERE *p;
uchar cmapaddr;
uchar tag;
int depth;
bool surf;
{
    DEXELNODE *nptr;

    nptr = (DEXELNODE *) malloc(sizeof(DEXELNODE));
    if(nptr == NULL) {
        fprintf(stderr,"Error: malloc failed!\n");
        exit(1);
    }
    nptr->sptr = p;
    nptr->front = surf;
    nptr->next = NULL;
    nptr->z = depth;
    nptr->dexel = packdexel(cmapaddr,tag,(depth));
    return(nptr);
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writestats:        Write various compiled statistics
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                18-mar-87
 *
 */

#if (defined STATS) && (defined DOPREP)
writestats() {
    FLOAT avgdexels,coverage;

    avgdexels = ((FLOAT)(totaldexels))/((FLOAT)(totaldv));
    coverage = ((FLOAT)(totaldv))/((FLOAT)(XPIXELS*XPIXELS));

    fprintf(stdout,"\n\nDepth buffer statistics:\n\n");
    fprintf(stdout,"Total dexels: %d\n", totaldexels);
    fprintf(stdout,"Total (non-empty) depth vectors: %d\n", totaldv);
    fprintf(stdout,"Screen coverage (by non-empty depth vectors): %.2f%%\n",
            (coverage*100));
    fprintf(stdout,"Maximum depth vector length: %d dexels\n", maxdvl);
    fprintf(stdout,"Minimum depth vector length: %d dexels\n", mindvl);
    fprintf(stdout,"Average depth vector length: %.2f\n\n", avgdexels);
}
#endif


 /* end write.c */
```

```
/*
 *      write.h:  Header file for write.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#define WRITEBUFSIZE 4096

#define INTERIOR     0xeffffff   /* Dexel marked as interior to molecule */
#define ZCLIPPED     0xdffffff   /* Dexel marked as Z clipped */

#define BACK     0
#define FRONT    1
#define STICK    2

#define STAG 0x1    /* sphere surface dexel tag */
#define BTAG 0x2    /* bond dexel tag */


/* :::  Typedefs  ::::::::::::::::::::::::::::::::::::::::::::::  */

#ifdef DOPREP
typedef struct dexelnode {
    int z;                      /* for comparison */
    bool front;                 /* Flag: TRUE if dexel is the front of sphere*/
    int dexel;                  /* packed dexel attributes */
    SPHERE *sptr;               /* ptr to associated sphere */
    struct dexelnode *next;     /* ptr to next dexel in depth vector */
} DEXELNODE;
#endif


/* :::  Globals  ::::::::::::::::::::::::::::::::::::::::::::::::  */

#ifdef DOPREP
DEXELNODE scandexel[XPIXELS];   /* heads of depth vectors for a scanline */
int writebuf[WRITEBUFSIZE];     /* buffer writes of depth vectors */
int curbufsize;                 /* current buffer size */
#endif


/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::::  */

#define packdexel(c,t,z)  ((c)|(t<<8)|((z&0xfffc)<<9))


/* :::  Externals  ::::::::::::::::::::::::::::::::::::::::::::::  */

#ifdef DOPREP
DEXELNODE *newdexel();
#endif

   /* end write.h */
```

```
/*
 *      common.h:  Header file for DockTool.
 */

/* :::  Defined constants  :::::::::::::::::::::::::::::::::::::::: */

#define IKBUFSIZE       256
#define MAXDEPTHVECSIZE 255


  /* end common.h */
```

```
/*
 *      config.h:  Header file for sprep.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#ifndef DOPREP
#define IKONAS                    /* Enable Ikonas output */
#endif
/*#define BACKTOFRONT*/           /* rendering order: */
                         /* BTF is slow but needed for transparency/clipping */

/*#define SUN */

#ifdef SUN
#undef IKONAS
#define FLOAT float
#else
#define FLOAT double
#endif

#define STATS

#define uchar unsigned char
#define bool  unsigned char


   /* end config.h */
```

```
/*
 *      standard.h
 */

/* :::  Include Files  ::::::::::::::::::::::::::::::::::::::::::: */

#include <stdio.h>


/* :::  Defined constants  :::::::::::::::::::::::::::::::::::::: */

#define TRUE  1
#define FALSE 0

/* :::  Typedef  :::::::::::::::::::::::::::::::::::::::::::::::: */

#define boolean unsigned char


/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::: */

#define STREQ(s1,s2)  (strcmp(s1,s2)==0)
#define MAX(x,y)   ((x)>(y) ? (x):(y))
#define MIN(x,y)   ((x)<(y) ? (x):(y))
#define ABS(x)     ((x)<0 ? (-(x)):(x))


/* :::  Externals  :::::::::::::::::::::::::::::::::::::::::::::: */

FILE *fileopen();


  /* end standard.h */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        aux.c:  contains:

    >>>>    fileopen:

                                            Thomas C. Palmer
                                            8-jan-87
*/

/* $Header: aux.c,v 1.1 87/01/14 13:54:02 palmer Exp $ */
static char rcsid[] = "$Header: aux.c,v 1.1 87/01/14 13:54:02 palmer Exp $";

#include "standard.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  fileopen:   Open a file
 *
 *  In:         fname - name of file to open
 *              mode -  open mode
 *
 *  Out:        file pointer
 *
 *  Errors:     unable to open file with given mode
 *
 *  Notes:      Adapted from K&P: "The Unix Programming Environment"
 *
 *  Revisions:     Initial   TCP               20-jul-86
 *
 */

FILE *fileopen(fname, mode)
char *fname, *mode;
{
    FILE *fptr;

    if ((fptr = fopen(fname, mode)) == NULL) {
        fprintf(stderr, "Can't open %s (mode: %s).  Bye.\n",fname,mode);
        exit(1);
    }
    return(fptr);
}


 /* end aux.c */
```

The directory grumpy:/grip4/palmer/docktool/util
contains the source code for encode and mkbonds:

```
        encode.c
        encode.h
        mkbonds.c
        mkbonds.h
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        encode.c:  contains:

    >>>>    main:
    >>>>    encode:
    >>>>    writebuffer:
    >>>>    flushbuffer:
    >>>>    openbinfile:

                                            Thomas C. Palmer
                                            5-nov-86
*/

/* $Header: encode.c,v 1.1 86/11/10 19:58:14 palmer Exp $ */
static char rcsid[] = "$Header: encode.c,v 1.1 86/11/10 19:58:14 palmer Exp $";

#include <stdio.h>
#include "encode.h"

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fptr, *fileopen();

  /* process command line arguments */
    while (argc > 1 && argv[1][0] == '-') {
        switch (argv[1][1]) {
            default:
                fprintf(stderr, "encode: unknown arg %s\n",argv[1]);
                exit(1);
        }
        argc--;
        argv++;
    }

    if (argc == 1)      /* use stdin */
        fptr = stdin;
        else fptr = fileopen(argv[1], "r");

    encode(fptr);
    exit(0);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  encode:     Encode runs of zero length depth vectors
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               5-nov-86
 *
 */

encode(fptr)
FILE *fptr;
{
    int head;
    int bufindex, runlength = 0, wordsread;
    int readbuf[READBUFSIZE];

  /* skip the header line containing windowing information */
    read(fileno(fptr),&head,sizeof(head));
#ifdef SUN
    head = swapbytes(head);
#endif
    write(fileno(stdout), &head, sizeof(head)); /* write the header */
    fflush(stdout);

    while((wordsread=
            (read(fileno(fptr),readbuf,(sizeof(int)*READBUFSIZE)))) > 0) {
        wordsread /= sizeof(int);  bufindex = 0;
        while(bufindex < wordsread) {

          /* Run length encode zero-length depth vectors */
            while(readbuf[bufindex] == 0) {
                runlength++;
                if (++bufindex == wordsread) break;
            }

            if(bufindex == wordsread) break;

            if(readbuf[bufindex] != 0) {
                if(runlength > 0) writebuffer(ENCODE(runlength));
                runlength = 0;
            }

          /* Write non zero-length depth vectors */
            while(readbuf[bufindex] != 0) {
                writebuffer(readbuf[bufindex]);
                if (++bufindex == wordsread) break;
            }
        }
    }

    if(runlength > 0)
        writebuffer(ENCODE(runlength));
    flushbuffer();
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  writebuffer:       Add the item to the buffer and flush if full.
 *
 *  In:          item - item to add
 *
 *  Revisions:     Initial    TCP                  6-nov-86
 *
 */

writebuffer(item)
int item;
{
    writebuf[curbufsize++] = item;
    if(curbufsize == WRITEBUFSIZE)
        flushbuffer();
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  flushbuffer:       Flush write buffer.
 *
 *  Revisions:     Initial    TCP                  6-nov-86
 *
 */

flushbuffer() {
#ifdef SUN
    register int i;
#endif

#ifdef SUN    /* Need to reverse integer byte ordering for Vax binary reads */
    for(i=0; i<curbufsize; i++) {
        writebuf[i] = swapbytes(writebuf[i]);
    }
#endif

    write(fileno(stdout), writebuf, (sizeof(int) * curbufsize));
    curbufsize = 0;
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  swapbytes:  Reverse integer byte ordering for VAX binary reads.
 *              Versions using this run on a Sun but write binary
 *              files to be read on a VAX
 *
 *              in bytes:  0 1 2 3      out bytes:  3 2 1 0
 *
 *  In:         val - integer to reverse
 *
 *  Out:        reversed integer
 *
 *  Revisions:      Initial   TCP                25-feb-87
 *
 */

#ifdef SUN
swapbytes(val)
int val;
{
    int rval;
    char *vptr, *rptr;

    vptr = (char *) &val;
    rptr = (char *) &rval;
    rptr += sizeof(int) - 1;

    *rptr-- = *vptr++;
    *rptr-- = *vptr++;
    *rptr-- = *vptr++;
    *rptr = *vptr;

    return(rval);
}
#endif


 /* end encode.c */
```

```
/*
 *      encode.h:  Header file for encode.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::::: */

/* #define SUN          /* Sun version */

#define READBUFSIZE  1024
#define WRITEBUFSIZE 1024


/* :::  Globals  :::::::::::::::::::::::::::::::::::::::::::::::::  */

int writebuf[WRITEBUFSIZE];
static int curbufsize = 0;

/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::::  */

#define SETBIT31 0x80000000
#define ENCODE(r) ((r) | SETBIT31)


  /* end encode.h */
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        mkbonds.c:  contains:

    >>>>    main:
    >>>>    dobonds:

    Mkbonds was adapted from code supplied by Rich Holloway.

                                        Thomas C. Palmer
                                        24-jan-87
*/

/* $Header$ */
static char rcsid[] = "$Header$";

#include "/grip4/palmer/docktool/spheres/standard.h"
#include "mkbonds.h"

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fptr;

  /* process command line arguments */
    while (argc > 1 && argv[1][0] == '-') {
        switch (argv[1][1]) {
            default:
                fprintf(stderr, "mkbonds: unknown arg %s\n",argv[1]);
                exit(1);
        }
        argc--;
        argv++;
    }

    if (argc == 1)      /* use stdin */
        fptr = stdin;
        else fptr = fileopen(argv[1], "r");

    dobonds(fptr);
    exit(0);
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  dobonds:    Print atom indices if atoms are closer than
 *              BOND_DISTANCE (under the assumption that they're bonded).
 *
 *  In:         fptr - input file pointer
 *
 *  Out:        bonded atom indices
 *
 *  Errors:     Max input atoms exceeded
 *
 *  Revisions:     Initial   TCP               24-jan-87
 *
 */

dobonds(fptr)
FILE *fptr;
{
    register int i=0,j,natoms;
    char linebuf[100];
    double dist_sq,bond_sq;
    POINT atoms[MAXATOMS];

  /* Load atomic coordinates to atoms array */
    while((fgets(linebuf,sizeof(linebuf),fptr)) != NULL) {
        sscanf(linebuf,"%f %f %f",&atoms[i].x,&atoms[i].y,&atoms[i].z);
        if(++i == MAXATOMS) {
            fprintf(stderr,"Error: Max atoms (%d) exceeded.\n",MAXATOMS);
            exit(1);
        }
    }

    bond_sq = sq(BOND_DISTANCE);
    natoms = i;
    for(i=0; i<natoms; i++) {
        for(j=i+1; j<natoms; j++) {
            dist_sq = (sq(atoms[i].x-atoms[j].x) +
                       sq(atoms[i].y-atoms[j].y) +
                       sq(atoms[i].z-atoms[j].z));
            if(dist_sq <= bond_sq) {
                fprintf(stdout,"%d %d\n",i,j);
                fflush(stdout);
            }
        }
    }
}

 /* end mkbonds.c */
```

```
/*
 *      mkbonds.h:  Header file for mkbonds.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::::: */

#define BOND_DISTANCE 1.8
#define MAXATOMS 2000


/* :::  Typedefs  :::::::::::::::::::::::::::::::::::::::::::::::::  */

typedef struct {
    double x,y,z;
} POINT;


/* :::  Macros  :::::::::::::::::::::::::::::::::::::::::::::::::::  */

#define sq(a)  ((a)*(a))


  /* end mkbonds.h */
```

The directory grumpy:/grip4/palmer/docktool/loadik
contains the source code for loadik:

```
        loadik.h
        loadik.c
        defconstants.h
        gik.h
```

```
/*
 *      loadik.h:  Header file for loadik.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::::: */

#define KRFILE "/grip4/palmer/docktool/loadik/loadik.kr"


/* :::  Globals  ::::::::::::::::::::::::::::::::::::::::::::::::::::  */

int Go;
int Goaddr, ndvecsaddr, nodebufaddr, numnodesaddr, runlengthaddr;
int numnodes[MAXBUFDEPTHVEC];
int nodebuf[MAXDEPTHVECSIZE * MAXBUFDEPTHVEC];


/* :::  Macros  :::::::::::::::::::::::::::::::::::::::::::::::::::::  */

#define BIT31OFF  0x7fffffff
#define DECODE(a) ((a) & BIT31OFF)

#define RUN_ENCODED(a)  ((a) < 0 ? TRUE : FALSE)


  /* end loadik.h */
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        loadik.c:  contains:

    >>>>    main:
    >>>>    load_ll:
    >>>>    loadinit:
    >>>>    flushzerobuf:
    >>>>    flushnodebuf:
                                        Thomas C. Palmer
                                        28-oct-86
*/

/* $Header: loadik.c,v 1.2 87/03/02 12:30:58 palmer Exp $ */
static char rcsid[] = "$Header: loadik.c,v 1.2 87/03/02 12:30:58 palmer Exp $";

#include <ikdefs.h>
#include <sys/file.h>
#include "/unc/palmer/src/lib/stdlib.h"
#include "defconstants.h"
#include "loadik.h"

 /* Global file descriptor for binary file: */
int     ikllfd;         /* Pixel linked list file */

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fptr;

  /* set defaults */

  /* process command line arguments */
    while (argc > 1 && argv[1][0] == '-') {
        switch (argv[1][1]) {
            case 's':   /* -s option */
                break;
            default:
                fprintf(stderr, "loadik: unknown arg %s\n",argv[1]);
                exit(1);
        }
        argc--;
        argv++;
    }


    /* Initialize */
    if (argc == 1)      /* use stdin */
        fptr = stdin;
    else fptr = fileopen(argv[1], "r");

    load_ll(fptr);
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  load_ll:    Load the depth buffer to the Ikonas
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               23-sept-86
 *
 */

load_ll(fptr)
FILE *fptr;
{
    int head;
    int bytesread;
    int runlength, totalnodes, ndvecs;
    boolean needtoflush;

    loadinit();

  /* skip the header line containing windowing information */
    read(fileno(fptr),&head,sizeof(head));

  /* Read the size of the depth vector, read the depth vector */
  /* and then send to the Ikonas for relocating and loading.  */

    while(TRUE) {
        ndvecs = 0;  totalnodes = 0;  needtoflush = FALSE;
        while(!needtoflush) {

            if((read(fileno(fptr), &numnodes[0], sizeof(int))) <= 0) {
                flushnodebuf(totalnodes,ndvecs);
                close(fileno(fptr));    return;
            }

            if(RUN_ENCODED(numnodes[0]))
                flushzerobuf(DECODE(numnodes[0]));

            else {    /* Read a run of nonzero length depth vectors */
                if(numnodes[ndvecs] > MAXDEPTHVECSIZE) {
                    fprintf(stderr, "Error: Bad depth vector size (%d).\n",
                            numnodes[ndvecs]);
                    fprintf(stderr, "Possible preprocessor error?\n");
                    exit(1);
                }

                do {
                    read(fileno(fptr),&nodebuf[totalnodes],
                                   (sizeof(int)*numnodes[ndvecs]));
                    totalnodes += numnodes[ndvecs];  ndvecs++;

                  /* Check need to flush node buffer conditions */
                    if ((ndvecs == MAXBUFDEPTHVEC)  ||
                        (totalnodes > ((MAXBUFDEPTHVEC-1)*MAXDEPTHVECSIZE))) {
                        needtoflush = TRUE;  break;
                    }

                    if((read(fileno(fptr),&numnodes[ndvecs],sizeof(int))) <= 0) {
                        flushnodebuf(totalnodes,ndvecs);
                        close(fileno(fptr));    return;
                    }

                    if(numnodes[ndvecs] > MAXDEPTHVECSIZE) {
                        fprintf(stderr, "Error: Bad depth vector size (%d).\n",
                                numnodes[ndvecs]);
                        fprintf(stderr, "Possible preprocessor error?\n");
                        exit(1);
                    }
                /* Check for possible runlength encoded zeros */
                  if (RUN_ENCODED(numnodes[ndvecs])) {
                      flushnodebuf(totalnodes,ndvecs);
                      flushzerobuf(DECODE(numnodes[ndvecs]));
                      totalnodes = 0;
                      needtoflush = TRUE;
                  }

              } while(!needtoflush);
          }
      }

      flushnodebuf(totalnodes,ndvecs);
  }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  loadinit:   Initialize the Ikonas for loading.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                4-nov-86
 *
 */

loadinit() {

    ikbegin(KRFILE);
    Goaddr         = iklookup("Go");
    ndvecsaddr     = iklookup("ndvecs");
    nodebufaddr    = iklookup("nodebuf");
    numnodesaddr   = iklookup("numnodes");
    runlengthaddr  = iklookup("runlength");
    ikrmmap();
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  flushzerobuf:       Write the buffer of run length encoded
 *                      zeros to the Ikonas.
 *
 *  In:         runlength - length of the run of zeros
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                4-nov-86
 *
 */

flushzerobuf(runlength)
int runlength;
{
    ikwrite(&runlength, sizeof(int), runlengthaddr);

    Go = 1;  ikwrite(&Go, sizeof(Go), Goaddr);
    iksync(Goaddr, 0);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  flushnodebuf:       Write the buffer of depth vectors to the Ikonas.
 *
 *  In:         totalnodes - total number of dexels in this buffer
 *              ndvecs     - number of depth vectors
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                4-nov-86
 *
 */

flushnodebuf(totalnodes,ndvecs)
int totalnodes,ndvecs;
{
    static int zero = 0;

    if(totalnodes == 0) return;
    ikwrite(numnodes, (sizeof(int)*ndvecs), numnodesaddr);
    ikwrite(&ndvecs, sizeof(int), ndvecsaddr);
    ikwrite(&zero, sizeof(int), runlengthaddr);
    ikwrite(nodebuf, (sizeof(int)*totalnodes), nodebufaddr);

    Go = 1;  ikwrite(&Go, sizeof(Go), Goaddr);
    iksync(Goaddr, 0);
}

 /* end loadik.c */
```

```
/*
 *      defconstants.h:  Header file for docktool.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

/* MAXBUFSIZE should be as close to 16,000 as possible.  If over, the */
/* Ikonas scratchpad will fill up. */
#define MAXDEPTHVECSIZE  255   /* Max allocation for depth vectors in fb */
#define MAXBUFDEPTHVEC    62
#define MAXBUFSIZE     15810   /* MUST be (MAXBUFDEPTHVEC * MAXDEPTHVECSIZE) */

#define DEPTHNODESIZE      1    /* Ikonas linked list node size */
#define IKBUFSIZE        256    /* size of Ikonas image buffer */

/* Using NZBITS bits of Z in Ikonas linked lists.  Note that we don't use  */
/* the maximum possible number representable in NZBITS bits.  We need some */
/* unique color/Z values for the Ikonas BMP code to use (IKNULL, EMPTY).   */
#define NZBITS 14

#if      (NZBITS==16)
#define IKZMAX  (0xfffe)        /* 2^16 - 2 */
#else if (NZBITS==14)
#define IKZMAX  (0x3ffe)        /* 2^14 - 2 */
#endif


  /* end defconstants.h */
```

```
/*
 *      gik.h:  Header file for gia2 source files.
 *
 *      This file is included by:
 *
 *              - loadik.g  <loadik>
 *              - ikclip.g  <docktool>
 */

/* :::  Defined constants  :::::::::::::::::::::::::::::::::::::::: */

#define TRUE  1
#define FALSE 0

#define IKNULL  0xffffff        /* Ikonas NULL pointer */
#define EMPTY   0xfffffe        /* Pointer buffer: empty depth vector */

/* Starting Ikonas x,y for clipping plane pointer buffer */
#define PBXSTART 256
#define PBYSTART 0

#define NUMBUFS 4


/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::::  */

/* Handshake with Vax process */
#define waitforupdate() {      \
    Go = FALSE;                \
    while(!Go) ;               \
}

/* IK24ADDR: build a BMP usable fb (LORES marika) address from x,y. */
/* Note: If x > 511 the 21st bit must be set to yield a true 24 bit */
/* Ikonas marika address. */

#define FB24ADDR(x,y)  ((((y)<<10)|(x))<<1)
#define BIT21 (0x200000)
#define IK24ADDR(x,y,a)  {        \
    a = FB24ADDR(x,y);            \
    if (x>511) { a |= BIT21; }    \
}


  /* end gik.h */
```

The directory grumpy:/grip4/palmer/docktool/dock
contains the source code for docktool:

```
        main.c
        parse.h
        parse.c
        readdrug.h
        readdrug.c
        ikclip.h
        ikclip.c
        gikclip.h
        gikdrug.h
        ikclip.g
        dev.h
        matcom.h
        dev.c
        kb.h
        kb.c
        screen.h
        screen.c
        docktool.h
        cmdtype.h
        config.h
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        main.c:  contains:

    >>>>    main:
    >>>>    setup:
    >>>>    shutdown:
    >>>>    Interrupt_trap: Trap interrupts, quit gracefully.
    >>>>    help:

                                        Thomas C. Palmer
                                        28-oct-86
*/

/* $Header: main.c,v 1.4 87/03/26 21:51:27 palmer Exp $ */
static char rcsid[] = "$Header: main.c,v 1.4 87/03/26 21:51:27 palmer Exp $";

#include <signal.h>
#include "/grip4/palmer/docktool/spheres/standard.h"
#include "config.h"
#include "docktool.h"

main(argc, argv)
int argc;
char *argv[];
{
    char drugfile[MAXSTR];      /* drug file */

    int starttype;
    int Interrupt_trap();               /* interrupt trap routine */
    signal(SIGINT,  Interrupt_trap);    /* Trap interrupt */

  /* set defaults */
    starttype = FALSE;
    strcpy(drugfile,"");

  /* process command line arguments */
    while (argc > 1 && argv[1][0] == '-') {
        switch (argv[1][1]) {
          case 'l':   /* Call loadik to load encoded depth buffer file */
            starttype = LOADIKSTART;
            break;
          case 'a':   /* Call aput to load "agrabbed" depth buffer file */
            starttype = APUTSTART;
            break;
          case 'q':   /* Quick start: just read windowing data from file.edb */
            starttype = QUICKSTART;
            break;
          case 'd':   /* Read drug atomic coordinate file */
            strcpy(drugfile,argv[2]);
            argv++;
            break;
          case '-':   /* Give help */
            help();
            break;
          default:
            fprintf(stderr, "docktool: unknown arg %s\n",argv[1]);
            exit(1);
        }
        argc--;
        argv++;
    }

   /* initialize */
     setup(starttype,argv[1],drugfile);

     dopoll();
 }
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  setup:      Initialize DockTool
 *
 *  In:
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               28-oct-86
 *
 */

setup(starttype,edbfile,drugfile)
int starttype;
char *edbfile, *drugfile;
{
    int nBonds;

#ifdef DBX
    postinfo("\nWarning: This is a dbx version.  No docktool\n");
    postinfo("         keyboard commands are accepted.\n\n");
#else
    initscreen();  /* Initialize screen management system */
    postinfo("Wait for the prompt to appear below.\n\n");
    initkb();
#endif

  /* Load pointer buffer and linked list to Ikonas frame buffer */
    ikdataload(starttype,edbfile);

    ikbmpload();                /* Load Ikonas BMP code */
    iksetcmap();                /* Set up Ikonas color map */
    nBonds = readbonds(drugfile); /* Read atomic bond data */
    iksendbonds(nBonds);        /* Send bond data to Ikonas */
    initdevices();              /* Initialize devices */
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  shutdown:   Print a message and exit.
 *
 *  In:         msg  - final message
 *              code - exit code
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               13-nov-86
 *
 */

shutdown(msg, code)
char *msg;
int code;
{
    ikshutdown();
#ifndef DBX
    closescreen();
    kbunset();
#endif
    fprintf(stderr, "%s", msg);
    exit(code);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  Interrupt_trap:     Trap keyboard interrupts; quit gracefully
 */

int Interrupt_trap()
{
    signal(SIGINT,  Interrupt_trap); /* Trap another interrupt */
    shutdown("Interrupted.  Bye.\n", -2);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  help:
 *
 *  Revisions:     Initial   TCP                28-oct-86
 *
 */

help() {
    fprintf(stdout,"usage:\n\n");
    fprintf(stdout,"\tdocktool [-l -a -q] [-d file.a] file.edb\n\n");
    fprintf(stdout,"options:\n\n");
    fprintf(stdout,"\t-l            Load file.edb via loadik\n");
    fprintf(stdout,"\t-a            Find and load file.ag via aput\n");
    fprintf(stdout,"\t-q            Quick start: just read header from file.edb\n");
    fprintf(stdout,"\t-d file.a     Use file.a as the drug molecule\n");
    exit(0);
}


 /* end main.c */
```

```
/*
 *      parse.h:  Header file for parse.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::::: */

#define MAXCMD 160


/* :::  Typedefs  ::::::::::::::::::::::::::::::::::::::::::::::::::  */

typedef struct {
    char symbol[MAXSTR];
    int  referent;
} SYMBOLTABLE;


/* :::  Globals  :::::::::::::::::::::::::::::::::::::::::::::::::::  */

boolean gottacmd;
char cmdstr[MAXCMD];

int curtransp;          /* Current transparent object */
int curtransl;          /* Current translucent object */

#define NUMPLANES 13    /* Size of planes symbol table */
SYMBOLTABLE planestab[] = {
        "STD",          STD,
        "STANDARD",     STD,
        "LEFTIN",       LEFTIN,
        "LIN",          LEFTIN,
        "RIGHTIN",      RIGHTIN,
        "RIN",          RIGHTIN,
        "UPIN",         UPIN,
        "UIN",          UPIN,
        "DOWNIN",       DOWNIN,
        "DIN",          DOWNIN,
        "PUNCH",        PUNCH,
        "VIN",          VIN,
        "VOUT",         VOUT,
 };

   /* end parse.h */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        parse.c:  contains:

    >>>>    dopoll:
    >>>>    doparse:
    >>>>    queue_cmd:
    >>>>    newplane:
                                        Thomas C. Palmer
                                        11-nov-86
*/

/* $Header: parse.c,v 1.4 87/03/26 21:51:31 palmer Exp $ */
static char rcsid[] = "$Header: parse.c,v 1.4 87/03/26 21:51:31 palmer Exp $";

#include "/grip4/palmer/docktool/spheres/standard.h"
/* #include "/unc/palmer/src/lib/stdlib.h" */
#include "config.h"    /* system configuration flags */
#include "docktool.h"
#include "cmdtype.h"
#include "parse.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  dopoll:     Poll the A-D devices and the keyboard for user input.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 28-oct-86
 *
 */

dopoll() {
    int zclip;

#ifndef DBX
    prompt();
#endif

    while(TRUE) {

        /* Poll the devices for Z clip, translate, and rotate updates */
        switch(polldevices(&zclip)) {
          case NEW_ZCLIP:
            iksend(ZUPDATE, zclip, NULL);
            break;
          case NEW_MATRIX:
            ikmatrixsend(xform);
            break;
          case NOCHANGE:
            break;
        }

        /* Poll the keyboard for user commands */
#ifndef DBX
        pollkeyboard();
        if(gottacmd)
            doparse();
#endif
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  doparse:    Parse a user command.
 *
 *  Errors:     Unknown command
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 13-nov-86
 *
 */

doparse() {
    char token1[MAXSTR],
         token2[MAXSTR],
         token3[MAXSTR],
         msgbuf[MAXSTR];
    short ntokens;
    static int tagstate = NONE;
    int tmp;

    gottacmd = FALSE;

  /* Process the next command. */
    ntokens = sscanf(cmdstr, "%s %s %s", token1, token2, token3);

    switch(ntokens) {
      case -1:
        break;  /* empty input */
      case 1:   /* Commands with one argument */
        if (STREQ(token1, "help") || STREQ(cmdstr, "?")) {
            posthelp();
            break;
        }
        if (STREQ(token1, "q") || (STREQ(cmdstr, "quit"))) {
            shutdown("",1);
            break;
        }
        else {
            sprintf(msgbuf, "unknown command: %s.\n", cmdstr);
            postinfo(msgbuf);
            break;
        }

      case 2:   /* Commands with two arguments */
        if (STREQ(token1, "help") || STREQ(cmdstr, "?")) {
            if (STREQ(token2, "planes"))
                postplaneshelp();
            break;
        }
        if (STREQ(token1, "surf")) {
#ifdef TRANSP
            if(STREQ(token2,"on")) {
                if(tagstate == SURFTAG) {
                    iksend(TRANSPARENT,NONE,NULL);
                    tagstate = NONE;
                }
            }
            else if(STREQ(token2,"off")) {
                iksend(TRANSPARENT,SURFTAG,NULL);
                tagstate = SURFTAG;
            }
#ifdef TRANSL
            else if(STREQ(token2,"tp") || (STREQ(token2,"transparent"))) {
                iksend(TRANSLUCENT,SURFTAG,NULL);
            }
            else if(STREQ(token2,"std") || (STREQ(token2,"standard"))) {
                iksend(TRANSLUCENT,NONE,NULL);
            }
#else
            postinfo("Transparent surface not available in this version.\n");
#endif
#else
            postinfo("Surface on/off not available in this version.\n");
#endif
            break;
        }
        if (STREQ(token1, "bonds")) {
#ifdef TRANSP
            if(STREQ(token2,"on")) {
                if(tagstate == BONDTAG) {
                    iksend(TRANSPARENT,NONE,NULL);
                    tagstate = NONE;
                }
            }
            else if(STREQ(token2,"off")) {
                iksend(TRANSPARENT,BONDTAG,NULL);
                tagstate = BONDTAG;
            }
#else
            postinfo("Bonds on/off not available in this version.\n");
#endif
            break;
        }
        if (STREQ(token1, "plane")) {
#ifdef CLIPPLANES
            newplane(token2);
#else
            postinfo("Procedural planes not available in this version.\n");
#endif
            break;
        }
        if (STREQ(token1, "punchrad") || STREQ(token1, "pr")) {
#ifdef CLIPPLANES
            iksend(PUNCHRAD,atoi(token2),NULL);
#else
            postinfo("Procedural planes not available in this version.\n");
#endif
            break;
        }
        if (STREQ(token1, "bumpcheck") || STREQ(token1, "bc")) {
#ifdef BUMPCHECK
            if (STREQ(token2, "on"))
                iksend(BUMPC,TRUE,NULL);
            else if(STREQ(token2, "off"))
                iksend(BUMPC,FALSE,NULL);
#else
            postinfo("Bump checking not available in this version.\n");
#endif
            break;
        }
        if (STREQ(token1, "beep")) {
#ifdef BUMPCHECK
            if (STREQ(token2, "on"))
                iksend(BEEP,TRUE,NULL);
            else if(STREQ(token2, "off"))
                iksend(BEEP,FALSE,NULL);
#else
            postinfo("Bump checking not available in this version.\n");
#endif
            break;
        }
        if (STREQ(token1, "depthcue") || STREQ(token1, "dc")) {
            iksend(DEPTHCUE,atoi(token2),NULL);
            break;
        }
      case 3:   /* Commands with three arguments */
        if (STREQ(token1, "punchcen") || STREQ(token1, "pc")) {
#ifdef CLIPPLANES
            iksend(PUNCHCENTER,atoi(token2),atoi(token3));
#else
            postinfo("Procedural planes not available in this version.\n");
#endif
            break;
        }
      default:
        sprintf(msgbuf, "unknown command: %s.\n", cmdstr);
        postinfo(msgbuf);
    }
    prompt();
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  queue_cmd:      Add a text string to the command queue and
 *                  signal that we have a command
 *
 *  In:         cmd - text string
 *
 *  Notes:          Called from do_char in kb.c.
 *
 *  Revisions:     Initial   TCP                13-nov-86
 *
 */

queue_cmd(cmd)
char *cmd;
{
    gottacmd = TRUE;
    strcpy(cmdstr,cmd);
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  newplane:   Look up symbolic name for clipping planes
 *
 *  In:         symname - symbolic name
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                18-nov-86
 *
 */

newplane(symname)
char *symname;
{
    register int i;

  /* Look up symbolic name */
    strtoupper(symname);
    for(i=0; i<NUMPLANES; i++) {
        if (STREQ(symname, planestab[i].symbol)) {
            iksend(PROCPLANE,planestab[i].referent,NULL);
            return;
        }
    }

  /* Symbol not found */
    postinfo("Can't find symbolic plane name.\n");
}


 /* end parse.c */
```

```
/*
 *      readdrug.h:  Header file for readdrug.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#define MAXBONDS  500
#define MAXATOMS  500

#define UNITY    128
#define logUNITY   7
#define TRANSBONDS  -128

#define ATOMSCALE   100.0
#define INFINITY    100000.0

#define RAMPSIZE     32         /* Colormap ramp constants */
#define NUMRAMPS      8
#define BACKGRDRAMP   7         /* background ramp number */


/* :::  Typedefs  :::::::::::::::::::::::::::::::::::::::::::::::  */

typedef struct {
    int x,y,z;
} INTPOINT;

typedef struct {
    FLOAT x,y,z;
} POINT;

typedef struct {
    int  x,y,z;         /* atom center */
    short  rad;         /* atom radius */
    int cindex;         /* colormap index */
} INFORMAT;

typedef struct {
    INTPOINT p1,p2;     /* top/bottom point in screen coordinate system */
    int cindex1;        /* color table index of first half */
    int cindex2;        /* color table index of second half */
} INSTICKS;


/* :::  Globals  ::::::::::::::::::::::::::::::::::::::::::::::::  */

INFORMAT  atoms[MAXATOMS];  /* Input atom list */
INSTICKS sticks[MAXBONDS];  /* Input bonds list */

FLOAT winsize;          /* window size of macromolecule; used to window drug */


/* :::  Macros  :::::::::::::::::::::::::::::::::::::::::::::::::  */

#define x9bit(a)  ((((a)+16384)>>logUNITY) )
#define y9bit(a)  (255 - (((a)+16384)>>logUNITY))

   /* end readdrug.h */
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        readdrug.c:  contains:

    >>>>    readheader:
    >>>>    readbonds:
    >>>>    readatomfile:
    >>>>    readbondfile:
    >>>>    transbonds:
    >>>>    winatoms:

                                        Thomas C. Palmer
                                        12-feb-87
*/

/* $Header: readdrug.c,v 1.4 87/03/26 21:51:38 palmer Exp $ */
static char rcsid[] = "$Header: readdrug.c,v 1.4 87/03/26 21:51:38 palmer Exp $";

#include "/grip4/palmer/docktool/spheres/standard.h"
/* #include "/unc/palmer/src/lib/stdlib.h" */
#include "config.h"
#include "/grip4/palmer/docktool/loadik/defconstants.h"
#include "readdrug.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readheader: Read the header store with the encoded depth
 *              buffer and use to window the drug molecule
 *
 *  In:         edbfile - encoded depth buffer file name
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                2-mar-87
 *
 */

readheader(edbfile)
char *edbfile;
{
    int head;
    FILE *fptr;

    fptr = fileopen(edbfile);

  /* read the header line containing windowing information */
    read(fileno(fptr),&head,sizeof(head));
    winsize = (FLOAT) head;

    fclose(fptr);
}
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readbonds:  Read the atom file and the bond connectivity file
 *
 *  In:         afile - drug atomic coordinate file
 *
 *  Out:        number of bonds
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                12-feb-87
 *
 */

readbonds(afile)
char *afile;
{
    int nAtoms,nBonds;
    register int i;

    if(STREQ(afile,"")) {
        postinfo("Warning: no drug file specified\n\n");
        return(0);
    }

    postinfo("Reading drug atomic coordinates ...");

    nAtoms = readatomfile(afile);
    winatoms(nAtoms);
    afile[strlen(afile)-1] = 'b';  /* convert "file.a" to "file.b" */

    postinfo(" done.\n");
    return(readbondfile(afile));
}
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readatomfile:      Read the atomic coordinate file
 *
 *  In:         afile - atomic coordinate file name
 *
 *  Out:        number of atoms
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                8-jan-87
 *
 */

readatomfile(afile)
char *afile;
{
    register int nAtoms;
    register FILE *fptr;
    char linebuf[100];
    double x,y,z;
    char atype[6];
    char msg[80];

    fptr = fileopen(afile,"r");
    nAtoms = 0;
    while((fgets(linebuf,sizeof(linebuf),fptr)) != NULL) {
        sscanf(linebuf,"%f %f %f %*f %*s %s",&x,&y,&z,atype);
        switch(atype[0]) {
          case 'C': case 'c': atoms[nAtoms].cindex = 31; break; /* carbon */
          case 'O': case 'o': atoms[nAtoms].cindex = 63; break; /* oxygen */
          case 'N': case 'n': atoms[nAtoms].cindex = 95; break; /* nitrogen */
          case 'S': case 's': atoms[nAtoms].cindex = 127; break; /* sulfer */
          case 'P': case 'p': atoms[nAtoms].cindex = 159; break; /*phosphorus*/
          case 'U': case 'u': atoms[nAtoms].cindex = 191; break; /* copper */
          case 'Z': case 'z': atoms[nAtoms].cindex = 223; break; /* zinc */
          case 'H': case 'h':
            atoms[nAtoms].cindex = BACKGRDRAMP+20; break; /* hydrogen */
          default:
            atoms[nAtoms].cindex = 31; break; /* (carbon) */
        }

        atoms[nAtoms].x =  x * ATOMSCALE;
        atoms[nAtoms].y =  y * ATOMSCALE;
        atoms[nAtoms].z = -z * ATOMSCALE;       /* Make left-handed here */

        if(++nAtoms == MAXATOMS) {
            sprintf(msg,"Error: Max atoms (%d) exceeded.\n", MAXATOMS);
            shutdown(msg, -1);
            exit(1);
        }
    }

    if(nAtoms == 0) shutdown("Error: zero atoms read from drug file.\n",-1);

    sprintf(msg, " (%d atoms in drug)", nAtoms);
    postinfo(msg);

    return(nAtoms);
}
```

```c
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  readbondfile:      Read the bond connectivity file and load
 *                     the bond data.
 *
 *  In:         bfile - bond connectivity file name
 *
 *  Out:        number of bonds
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                8-jan-87
 *
 */

readbondfile(bfile)
char *bfile;
{
    register int nBonds;
    register FILE *fptr;
    char linebuf[100];
    char msg[80];
    int a1,a2;

    fptr = fileopen(bfile,"r");
    nBonds = 0;
    while((fgets(linebuf,sizeof(linebuf),fptr)) != NULL) {
        sscanf(linebuf,"%d %d",&a1,&a2);
        sticks[nBonds].p1.x = x9bit(atoms[a1].x);
        sticks[nBonds].p1.y = y9bit(atoms[a1].y);
        sticks[nBonds].p1.z = x9bit(atoms[a1].z);       /* use x9bit for z */

        sticks[nBonds].p2.x = x9bit(atoms[a2].x);
        sticks[nBonds].p2.y = y9bit(atoms[a2].y);
        sticks[nBonds].p2.z = x9bit(atoms[a2].z);       /* use x9bit for z */

        sticks[nBonds].cindex1 = atoms[a1].cindex;
        sticks[nBonds].cindex2 = atoms[a2].cindex;

        nBonds++;
        if(nBonds >= MAXBONDS) {
            sprintf(msg,"Error: Max bonds (%d) exceeded.\n", MAXBONDS);
            shutdown(msg, -1);
        }
    }

    if(nBonds == 0) shutdown("Error: zero bonds read.\n",-1);

    sprintf(msg, "%d bonds\n", (nBonds));
    postinfo(msg);

    transbonds(nBonds);
    return(nBonds);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  transbonds: Translate bond coordinates to origin
 *
 *  In:         nBonds - number of bonds
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                18-feb-87
 *
 */

transbonds(nBonds)
int nBonds;
{
    register int i;

    for(i=0; i<nBonds; i++) {
        sticks[i].p1.x += TRANSBONDS;
        sticks[i].p1.y += TRANSBONDS;
        sticks[i].p1.z += TRANSBONDS;
        sticks[i].p2.x += TRANSBONDS;
        sticks[i].p2.y += TRANSBONDS;
        sticks[i].p2.z += TRANSBONDS;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  winatoms:   Apply the windowing specified in the header
 *              of the encoded depth buffer file.
 *
 *  In:         nAtoms - number of atoms
 *
 *  Notes:         Adapted from Mike Pique's "atoss.c"
 *
 *  Revisions:     Initial   TCP                8-jan-87
 *
 */

winatoms(nAtoms)
int nAtoms;
{
    FLOAT scale;
    register int i;

    scale = (((long)IKBUFSIZE) * UNITY)/winsize;

    for(i=0; i<nAtoms; i++) {
        atoms[i].x = (atoms[i].x) * scale;
        atoms[i].y = (atoms[i].y) * scale;
        atoms[i].z = (atoms[i].z) * scale;
    }
}

 /* end readdrug.c */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        ikclip.g:  contains:

    >>>>    main:
    >>>>    doparse:
    >>>>    doclip:
    >>>>    transparent:
    >>>>    translucent:
    >>>>    changeplane:
    >>>>    punchcenter:
    >>>>    punchradius:
    >>>>    setup:
    >>>>    redraw:
    >>>>    setxbar:
    >>>>    switchplanes:
    >>>>    gettranslucent:
    >>>>    erase:
    >>>>    shutdown:
                                        Thomas C. Palmer
                                        28-oct-86
*/

#include "/unc/glassner/include/ik.h"
#include "/grip4/palmer/docktool/loadik/defconstants.h"
#include "/grip4/palmer/docktool/loadik/gik.h"
#include "config.h"
#include "cmdtype.h"
#include "gikclip.h"
#include "gikdrug.h"

#define DEBUG

main() {

#ifdef IDB
    idbinit();
    idbtrap(0);
#endif
    setup();
    doparse();
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  doparse:    Parse incomming commands
 *
 *  Notes:
 *
 *  Revisions:     Initial    TCP               3-nov-86
 *
 */

doparse() {

    while(TRUE) {
        waitforupdate();
#ifdef IDB
        idbtrap(2);
#endif
        /* Parse new command */
        if      (cmdvector[0] == ZUPDATE) {
            doclip(cmdvector[1]); /* Draw new clipped image */
            call switchplanes;  /* switch image planes (double buffering) */
        }
        else if (cmdvector[0] == XFORM) {
            restore();          /* restore the current write buffer */
            xformbonds();       /* transform bond coordinates and redraw  */
            call switchplanes;  /* switch image planes (double buffering) */
        }
#ifdef TRANSP
        else if (cmdvector[0] == TRANSPARENT) {
            transparent(cmdvector[1]);
        }
#endif
#ifdef TRANSL
        else if (cmdvector[0] == TRANSLUCENT) {
            translucent(cmdvector[1]);
        }
#endif
#ifdef CLIPPLANES
        else if (cmdvector[0] == PROCPLANE) {
            changeplane(cmdvector[1]);
        }
        else if (cmdvector[0] == PUNCHCENTER) {
            punchcenter(cmdvector[1],cmdvector[2]);
        }
        else if (cmdvector[0] == PUNCHRAD) {
            punchradius(cmdvector[1]);
        }
#endif
#ifdef BUMPCHECK
        else if (cmdvector[0] == BUMPC) {
            bumpenable(cmdvector[1]);
        }
#endif
        else if (cmdvector[0] == DEPTHCUE) {
            drugdepthcue(cmdvector[1]);
        }
        else if (cmdvector[0] == QUIT) {
            shutdown();
            return;
        }
    }
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  doclip:     Generate the visible image given a new Z clip value
 *
 *  In:         fz - new Z clip value
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 29-oct-86
 *
 */

doclip(fz)
int fz;
{
    register int Xptrbuf, Yptrbuf, Xstop, Ystop, fast_z;
    register int nodeptr, node_z, node_clr;
    register int iknull, empty, pbxstart, rbgc;
#ifdef CLIPPLANES
    static int clipjmp;
    label do_leftin,do_rightin,do_upin,do_downin;
    label do_punch,do_vin,do_vout,gotz;
#endif
#ifdef TRANSP
    int tagcheck;
#endif

  /* Load constants to registers */
    fast_z = fz;
    iknull = IKNULL;
    empty = EMPTY;
    pbxstart = PBXSTART;
    if(curwriteplane == BLUEPLANE) {
        rbgc = (redtogreen(BGC)) | (redtoblue(BGC));
    }
    else { rbgc = (BGC) | (redtogreen(BGC)); }

#ifdef TRANSP
    tagcheck = curtransp;
#endif

#ifdef CLIPPLANES
  /* Decide which procedural clipping routine to jump to. */
  /* Eliminates the decision in the inner loop.  */
    if      (curclip == LEFTIN)  clipjmp = do_leftin;
    else if (curclip == RIGHTIN) clipjmp = do_rightin;
    else if (curclip == UPIN)    clipjmp = do_upin;
    else if (curclip == DOWNIN)  clipjmp = do_downin;
    else if (curclip == PUNCH)   clipjmp = do_punch;
    else if (curclip == VIN)     clipjmp = do_vin;
    else if (curclip == VOUT)    clipjmp = do_vout;
    else                         clipjmp = gotz;
#endif

  /* Save z value */
    curz = fast_z;

  /* Set loop ending constants */
    Xstop = pbxstart + IKBUFSIZE;
    Ystop = PBYSTART + IKBUFSIZE;

  /* For each X,Y in the image buffer, get the pointer at X,Y  */
  /* of the clipping plane pointer buffer.  Search through the */
  /* depth vector to find the dexel that has the smallest Z of */
  /* all those greater than the Z clip value. */

    for(Yptrbuf=PBYSTART; Yptrbuf<Ystop; Yptrbuf++) {
        for(Xptrbuf=pbxstart; Xptrbuf<Xstop; Xptrbuf++) {
            getpixel(Xptrbuf, Yptrbuf, nodeptr);  /* Get node pointer */

            if(nodeptr == empty) {    /* clear buffer with background color */
                putpixel((Xptrbuf-pbxstart),(Yptrbuf),rbgc);
            }
            else {              /* need to search through depth vector */
                fbread(nodeptr, node_z);
#ifdef CLIPPLANES
                goto *clipjmp;  /* Jump to procedural clipping routine */
do_leftin:      leftin ((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
do_rightin:     rightin((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
do_upin:        upin   ((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
do_downin:      downin ((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
do_punch:       punch  ((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
do_vin:         vin    ((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
do_vout:        vout   ((Xptrbuf-pbxstart),Yptrbuf,curz,fast_z);  goto gotz;
gotz:
#endif
                if(node_z == iknull) {
                  /* Check here for fastz < node_z && one back = IKNULL */
                    searchbackward(nodeptr, node_z);
                }
                else {
                    if(node_z > fast_z) {
                  /* Check here for possible speed up */
                        searchbackward(nodeptr, node_z);
                    }
                        else { searchforward(nodeptr, node_z); }
                }

                if(node_z != iknull) {
#ifdef TRANSL
                    if((node_z & TAGMASK) == curtransl)
                        node_z = gettranslucent(nodeptr,node_z);
#endif
                  /* Write to both the current plane and the green bits */
                  /* clear blue & green bits prior to shifting and ORing */
                    node_z &= REDBITS;
                    if(curwriteplane == BLUEPLANE) {
                        node_z = ((redtoblue(node_z))|(redtogreen(node_z)));
                    }
                    else {
                        node_z = ((node_z)|(redtogreen(node_z)));
                    }
                    putpixel((Xptrbuf-pbxstart),(Yptrbuf),node_z);
                }
                else {
                    putpixel((Xptrbuf-pbxstart),(Yptrbuf),rbgc);
                }

              /* Update clipping plane pointer buffer */
                writemask(ALLBITS);
                putpixel(Xptrbuf,Yptrbuf, nodeptr);
                if(curwriteplane == BLUEPLANE)
                    { writemask(BLUEGREENBITS); }
                else { writemask(REDGREENBITS);  }
            }
        }
    }

  /* Redraw bonds on top of new image */
    if(curwriteplane == BLUEPLANE) {
        writemask(BLUEBITS);
    } else {
        writemask(REDBITS);
    }
    xformbonds();      /* transform bond coordinates and redraw  */
  /* Mark current write buffer as corrupt  */
  /* (as far as drawing bonds is concerned) */
  /* and reset the write mask */
    if(curwriteplane == BLUEPLANE) {
        Bcorrupt = TRUE;  restoreBcnt = 0;  writemask(BLUEGREENBITS);
    }
    else {
        Rcorrupt = TRUE;  restoreRcnt = 0;  writemask(REDGREENBITS);
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  setup:      Initialize the Ikonas for DockTool
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP              29-oct-86
 *
 */

setup() {
    register int i;

  /* Read bond data from Vax */
    loadbonds();

  /* Set FBC to 256x256 scanout mode */
    FBC[2] = WIND;
    FBC[3] = ZOOM;

  /* Initialize all crossbar outputs to null */
    for(i=0; i<NUMXBAR_REGS; i++) {
        XBAR[i] = NULLXBAR;
    }

#ifdef CLIPPLANES
    curclip = STD;
    punchx = 128;
    punchy = 128;
    punchr = 20;
#endif
#ifdef TRANSP
    curtransp = NONE;
#endif
#ifdef TRANSL
    curtransl = NONE;
#endif

#ifdef BUMPCHECK
    bumpcnt = 0;        /* init bumppos index */
    checkbumps = TRUE;  /* enable bump checking */
#endif

  /* Init save/restore counters */
     restoreRcnt = 0;    restoreBcnt = 0;

  /* Init drug depth cueing parameter */
    K = 1000;

  /* Initialize image buffer to background */
    call erase;     /* Must be called before write mask is set */

  /* Set, draw initial clipped image */
    curwriteplane = REDPLANE;
    writemask(REDGREENBITS);
    doclip(0);
    call switchplanes;
}
```

```
/*
 *      cmdtype.h:  Header file for cliptool.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#define NUMCMDS 3     /* Size of the command buffer sent to the Ikonas */

#define QUIT        0      /* Shutdown Ikonas */
#define ZUPDATE     1      /* Process a new Z clipping plane */
#define TRANSPARENT 2      /* Make an object transparent */
#define TRANSLUCENT 3      /* Make an object transparent */
#define PROCPLANE   4      /* Choose a new procedural plane */
#define PUNCHCENTER 5      /* Set new center for "punch" clipping plane */
#define PUNCHRAD    6      /* Set new radius for "punch" clipping plane */
#define XFORM       7      /* Process new transformation matrix */
#define BUMPC       8      /* enable/disable bump checking */
#define BEEP        9      /* enable/disable bump check sound feedback */
#define DEPTHCUE   10      /* reset drug depth cueing */

#define NOTRANSP    0      /* No objects transparent */
#define NOTRANSL    0      /* No objects translucent */

#define STD     0          /* Types of clipping planes */
#define LEFTIN  1
#define RIGHTIN 2
#define UPIN    3
#define DOWNIN  4
#define PUNCH   5
#define VIN     6
#define VOUT    7

#if (defined TRANSP) || (defined TRANSL)
#define TAGMASK  0x300          /* Object tag constants */
#define NONE     0x000
#define SURFTAG  0x100
#define BONDTAG  0x200
#endif

  /* end cmdtype.h */
```

```
/*
 *      config.h:  Header file for DockTool.
 */

/* System configuration flags */

#define FLOAT double

/* #define DBX          /* Enable dbx debugging: don't use keyboard routines */
/* #define IDB          /* Enable idb Ikonas debugging */

#define BUMPCHECK       /* Enable bump checking */
#define CLIPPLANES      /* Enable procedural clipping planes */
#define TRANSP          /* Enable transparency */
#define TRANSL          /* Enable translucency */

#define BONDCMAPBASE 192        /* bond cmap base addr (currently zinc) */


  /* end config.h */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  dmatmat:    double precision 3x3 matrix times matrix
 *
 *  In:         m1, m2 - matrices
 *
 *  Notes:          Adapted from Mike Pique's fastspheres code.
 *
 *  Revisions:     Initial   TCP                9-feb-87
 *
 */

/* dmatmat: double precision 3x3 matrix times matrix */
/* uses matrix multiplication package - M. Pique - UNC  */
#define element double
#define PRODUCT(a,b)     ((a)*(b))


dmatmat(m1, m2, rm)
#define P1ORGRANGE      NR1*NC1
#define P1ORGSTEP       NC1
#define P2ORGSTEP       1
#define P2ORGRANGE              NC2
#define P1RANGE         NC1
#define P1STEP          1
#define P2STEP          NC2
#define NR1     3
#define NC1     3
#define NR2     NC1
#define NC2     3

#include "matcom.h"


 /* end dev.c */
```

```
/*
 *      kb.h:  Header file for kb.c
 */

/* :::  Include Files  :::::::::::::::::::::::::::::::::::::::::::: */

#include <sgtty.h>
#include <ctype.h>


/* :::  Globals  :::::::::::::::::::::::::::::::::::::::::::::::::  */

static struct sgttyb tty;
static struct ltchars lchars;
static struct tchars chars;
static FILE *devtty;


/* :::  Macros  ::::::::::::::::::::::::::::::::::::::::::::::::::  */

#define IOCTL(op, par)                          \
        if (ioctl(fileno(stdin), op, par) < 0)  \
        {                                       \
                perror("tty");                  \
                exit(-1);                       \
        }

  /* end kb.h */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  setxbar:    Gia-style routine.  Switch the crossbar.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                29-oct-86
 *
 */

setxbar {

  /* Crossbar must be set to read bits NOT currently being written. */
    if(curwriteplane == BLUEPLANE) {
        XBAR[0] =  0;   XBAR[1] =  1;
        XBAR[2] =  2;   XBAR[3] =  3;
        XBAR[4] =  4;   XBAR[5] =  5;
        XBAR[6] =  6;   XBAR[7] =  7;
        XBAR[8] =  0;   XBAR[9] =  1;
        XBAR[10] =  2;  XBAR[11] =  3;
        XBAR[12] =  4;  XBAR[13] =  5;
        XBAR[14] =  6;  XBAR[15] =  7;
        XBAR[16] =  0;  XBAR[17] =  1;
        XBAR[18] =  2;  XBAR[19] =  3;
        XBAR[20] =  4;  XBAR[21] =  5;
        XBAR[22] =  6;  XBAR[23] =  7;
    }
    else {
        XBAR[0] =  16;  XBAR[1] =  17;
        XBAR[2] =  18;  XBAR[3] =  19;
        XBAR[4] =  20;  XBAR[5] =  21;
        XBAR[6] =  22;  XBAR[7] =  23;
        XBAR[8] =  16;  XBAR[9] =  17;
        XBAR[10] =  18; XBAR[11] =  19;
        XBAR[12] =  20; XBAR[13] =  21;
        XBAR[14] =  22; XBAR[15] =  23;
        XBAR[16] =  16; XBAR[17] =  17;
        XBAR[18] =  18; XBAR[19] =  19;
        XBAR[20] =  20; XBAR[21] =  21;
        XBAR[22] =  22; XBAR[23] =  23;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  switchplanes:       Gia-style routine.  Switch the image planes
 *                      being written to.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                4-nov-86
 *
 */

switchplanes {

    if (curwriteplane == REDPLANE) {
        curwriteplane = BLUEPLANE;
        writemask(BLUEGREENBITS);
    }
    else {
        curwriteplane = REDPLANE;
        writemask(REDGREENBITS);
    }

    call setxbar;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  erase:      Gia-style routine.  Erase the image buffer.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               29-oct-86
 *
 */

erase {
    register int x,y,ebgc;

#ifdef CCCLEAR    /* Use Ikonas video generator to clear screen */
        while( ! ccblnk );              /* wait for frame end */
        FBC[5] = FBCSTANDARD|FBCCLEAR;
        while( ccblnk );                /* wait for frame to start */
        while( ! ccblnk );              /* wait for frame end */
        FBC[5] = FBCSTANDARD;
#else
    ebgc = (BGC) | (redtogreen(BGC)) | (redtoblue(BGC));
    ebgc = 0xe0e0e0;
    for (y=0; y<IKBUFSIZE; y++) {
        x = -1;
        repeat IKBUFSIZE times {
            putpixel(++x, y, ebgc);
        }
    }
#endif
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  loadbonds:
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               12-feb-87
 *
 */

loadbonds() {

    waitforupdate();    /* wait until Vax process loads bond data */
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  xformbonds:
 *
 *  Notes:          Adapted from Mike Pique's "fastspheres" code.
 *
 *  Revisions:     Initial   TCP               16-feb-87
 *
 */

xformbonds() {
    register int bnum;                 /* number of current bond */
    register int *bptr;                /* ptr to current bond */
    register int *ip1, *ip2;           /* inner product scan pointers */
    register int tally;                /* inner product summation */
    int point1[3],point2[3];           /* transformed endpoints */
    register int *trans;               /* ptr to post-rotation translation */
    register int temp;                 /* tmp for inner product summation */
    register int *s, *d;               /* src,dest ptrs in bonds structure */
    register int dim;
    static int origin[3] = {MIDSCRX,MIDSCRY,MIDSCRZ};
    register int *orgp;                /* ptr to element of origin */

#ifdef BUMPCHECK
    dobeep = FALSE;
#endif

    bptr = bonds;

    for(bnum=0; bnum<nBonds; bnum++) {

      /* Transform first endpoint */
        s = bptr + PT1_OFFSET;  /* set src/dest ptrs */
        d = point1;
        trans = xform + 9;      /* set translation vector ptr */
        orgp = origin;
        ip2 = xform;

        for(dim=0; dim<3; dim++) {
            ip1 = s;    /* prevent fixed overflow (with asr) */
            ikmul(*ip1++, *ip2++, temp);  tally  = asr(temp);
            ikmul(*ip1++, *ip2++, temp);  tally += asr(temp);
            ikmul(*ip1++, *ip2++, temp);  tally += asr(temp);

            if(tally >= 0) tally = tally>>14;
            else { tally = -tally; tally = tally>>14; tally = -tally; }

            if(dim != 2) {
                *d++ = tally + *orgp++ + *trans++;
            }
            else {              /* handle Z differently */
                *d = tally + *orgp;
                *d = (((*d)+128)<<7)-16384;  /* undo TRANSBONDS & x9bit(z) */
                *d += *trans;
            }
        }

      /* Transform second endpoint */
        s = bptr + PT2_OFFSET;  /* set src/dest ptrs */
        d = point2;
        trans = xform + 9;      /* set translation vector ptr */
        orgp = origin;
        ip2 = xform;

        for(dim=0; dim<3; dim++) {
            ip1 = s;    /* prevent fixed overflow (with asr) */
            ikmul(*ip1++, *ip2++, temp);  tally  = asr(temp);
            ikmul(*ip1++, *ip2++, temp);  tally += asr(temp);
            ikmul(*ip1++, *ip2++, temp);  tally += asr(temp);

            if(tally >= 0) tally = tally>>14;
            else { tally = -tally; tally = tally>>14; tally = -tally; }

            if(dim != 2) {
                *d++ = tally + *orgp++ + *trans++;
            }
            else {              /* handle Z differently */
                *d = tally + *orgp;
/* should be:   *d = (((*d)+TRANSBONDS)<<logUNITY)-16384; */
                *d = (((*d)+128)<<7)-16384;  /* undo TRANSBONDS & x9bit(z) */
                *d += *trans;
            }
        }

#ifdef BUMPCHECK
        if(checkbumps) {      /* perform bump checking */
            p1bump = bumpcheck(point1);
            p2bump = bumpcheck(point2);
            if((p1bump) || (p2bump)) dobeep = TRUE;
        }
#endif

      /* draw bond */
        drawbond(point1, point2,*(bptr+CINDEX1_OFFSET),*(bptr+CINDEX2_OFFSET));
        bptr += BSTRUCTSIZE;    /* Move to next bond */
    }

#ifdef BUMPCHECK      /* show bumps on screen */
    if(checkbumps) { showbumps(); }
#endif
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  drawbond:   Draw bond using modified Bresenham algorithm
 *
 *  In:         p1,p2 - bond endpoints
 *              c - colormap index
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP               16-feb-87
 *
 */

#ifdef IDB
/*int db_z1, db_z2, db_dz, db_s3, db_tmp; */
#endif

drawbond(p1,p2,c1,c2)
int *p1, *p2;
int  c1, c2;
{
    register int x,y,z;          /* x,y,z to plot in screen coords */
    int dx,dy,dz;                /* delta x,y,z */
    register int s1,s2,s3;       /* increments */
    register int interchanged;   /* dx/dy swapped? */
    register int yerr,zerr;      /* Bresenham error term */
    register int halfway;        /* halfway through? switch colors */
    register int putc;           /* current color to write */
    register int tmp,addr;
    int i,bclipped,bclipc;
    int depthcue;
#ifdef BUMPCHECK
    int lastx,lasty;             /* save point for p2 bump position */
#endif

#ifdef BUMPCHECK
    lastx = -1;
#endif

  /* Set depth cueing */
    tmp = (*(p1+2)) + K;
    if (tmp > 32767) tmp = 32767;
    ikmul(tmp,32,depthcue);
    if(depthcue < 0) {
        depthcue = 0;
    }
    else {
        for(i=0; depthcue>=0; i++) { depthcue -= 18000; }
        depthcue = i;
        if(depthcue > 31) depthcue = 31;
    }

  /* Adjust colormap indices for depthcueing and current write buffer */
    bclipc = BONDCMAPBASE+31;
    c1 -= depthcue;    c2 -= depthcue;
    if(curwriteplane == BLUEPLANE) {
        c1 = redtoblue(c1); c2 = redtoblue(c2);
        bclipc = redtoblue(bclipc);
    }
    putc = c1;

  /* Initialize Bresenham variables */
    x = *p1;  y = *(p1+1);  z = *(p1+2);
    dx = abs(*p2 - x);  dy = abs(*(p2+1) - y);  dz = abs(*(p2+2) - z);
    s1 = sign(*p2 - x);  s2 = sign(*(p2+1) - y);  s3 = sign(*(p2+2) - z);

#ifdef IDB
/*    db_z1 = z;  db_z2 = *(p2+2);  db_dz = dz;  db_s3 = s3; */
/*    idbtrap(3); */
#endif
    if((dx == 0) && (dy == 0)) { return; }

  /* Put the largest slope in dx */
    if(dy > dx) {
        swap(dx,dy);
        interchanged = TRUE;
    }
    else interchanged = FALSE;

/* Initialize dz ~= (p1.z - p2.z)/dx
    dz = *(p1+2) - *(p2+2);
    if(dz<0) {dz = -dz; tmp = 1; } else { tmp = 0; }
    if(dz > dx) {
        for(i=0; dz>=0; i++) { dz -= dx; }
        if(tmp == 1) { dz = i; } else { dz = -i; }
    }
    else { dz = 0; }
*/

  /* Initialize error terms and halfway counter */
    yerr = (dy<<1) - dx;
    zerr = (dz<<1) - dx;
    halfway = dx>>1;

  /* Bresenham loop */
    for(i=0; i<=dx; i++) {

      /* check x,y screen clipping */
        if((x < MINSCRX)||(x > MAXSCRX)||(y < MINSCRY)||(y > MAXSCRY)) {
            goto zclipped;
        }

      /* check z macromolecule surface clipping */
        getpixel(x+PBXSTART,y+PBYSTART,addr);
        bclipped = FALSE;
        if(addr != EMPTY) {
            fbread(addr,tmp);
            if(tmp != IKNULL) {
                tmp = tmp>>9;
                if(z > tmp) bclipped = TRUE;
            }
        }

#ifdef BUMPCHECK   /* save possible p1 bump position */
        if(p1bump && checkbumps && !bclipped) {
            bumppos[bumpcnt++] = x;  bumppos[bumpcnt++] = y;
            p1bump = FALSE;
        }
      /* save x,y for possible p2 bump position */
        if(!bclipped) { lastx = x;  lasty = y; }
#endif

        /* If not clipped, save background and write */
        if(!bclipped) {
            savebg_putclr(x,y,putc);
        }
        else {
            savebg_putclr(x,y,bclipc);
        }

zclipped:
        while(yerr >= 0) {
            if(interchanged)  x += s1;  else y += s2;
            yerr = yerr - (dx<<1);
        }
        while(zerr >= 0) {
            z += s3;
            zerr = zerr - (dx<<1);
        }
        if(interchanged)  y += s2;  else x += s1;
        if (i == halfway) putc = c2;

        yerr = yerr + (dy<<1);
        zerr = zerr + (dz<<1);
    }

#ifdef BUMPCHECK   /* save possible p2 bump position */
    if((lastx != -1) && p2bump && checkbumps) {
        bumppos[bumpcnt++] = lastx;  bumppos[bumpcnt++] = lasty;
        p2bump = FALSE;
    }
#endif
    return;
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  restore:    Restore previously overwritten pixels to
 *              the current write buffer.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP              24-feb-87
 *
 */

restore () {
    register int x,y,cwp,tmp,clr;

    cwp = curwriteplane;        /* use register cwp for speed */

  /* set writemask for drug transform.  switchplanes will reset */
    if(cwp == BLUEPLANE) { writemask(BLUEBITS);} else { writemask(REDBITS);}

  /* If this is the first time restoration after clipping we */
  /* must restore the current write buffer from the green bits */
    if((Rcorrupt)||(Bcorrupt)) {
        for(y=0; y<IKBUFSIZE; y++) {
            for(x=0; x<IKBUFSIZE; x++) {
                tmp = fbaddr(x,y);
                fbget(tmp,clr);
                if(cwp == BLUEPLANE) {
                    clr = greentoblue(clr);
                }
                else { clr = greentored(clr); }
                fbput(tmp,clr);
            }
        }
        if(cwp == BLUEPLANE) { Bcorrupt = FALSE; } else { Rcorrupt = FALSE; }
        return;
    }

  /* Otherwise, restore from saved restore data */
    if(cwp == BLUEPLANE) {
        for(x=0; x<restoreBcnt; x+=2) {
            tmp = restoreB[x+1];
            fbput(restoreB[x],tmp);
        }
        restoreBcnt = 0;
    }
    else {
        for(x=0; x<restoreRcnt; x+=2) {
            tmp = restoreR[x+1];
            fbput(restoreR[x],tmp);
        }
        restoreRcnt = 0;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  bumpcheck:
 *
 *  In:
 *
 *  Out:
 *
 *  Errors:
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 10-mar-87
 *
 */

#ifdef BUMPCHECK
#ifdef IDB
int db_atom_z, db_surf_z;
#endif
bumpcheck(pt)
int *pt;
{
    register int nptr,bump,surf_z,atom_z,cpr_z;
    register int x,y;

    x = *pt;  y = *(pt+1);  atom_z = *(pt+2);
#ifdef IDB
    db_atom_z = atom_z;
    idbtrap(4);
#endif

  /* clip to image buffer */
    if((x < MINSCRX)||(x > MAXSCRX)||(y < MINSCRY)||(y > MAXSCRY)) {
        return(FALSE);
    }

  /* set nptr to the head of the depth vector */
    getpixel(x+PBXSTART,y+PBYSTART,nptr);
    if(nptr == EMPTY) { return(FALSE); }

    do {
        nptr -= 2;
        fbread(nptr,surf_z);
    } while(surf_z != IKNULL);

    /* Assume we are initially outside of molecular surface */
#ifdef IDB
    db_surf_z = surf_z;
    idbtrap(5);
#endif
    bump = 1;
    do {
        nptr += 2;
        fbread(nptr,surf_z);
        if(surf_z == IKNULL) break;
        if((surf_z&TAGMASK) == SURFTAG) { bump++; }
        cpr_z = surf_z>>9;     /* undo packdexel (sprep) shift for z compare */
    } while(((surf_z&TAGMASK) == BONDTAG)||(cpr_z < atom_z));

    if(surf_z == IKNULL) { return(FALSE); }
    else { return(bump&0x1); }  /* convert count to TRUE/FALSE */
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  showbumps:
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 10-mar-87
 *
 */

#ifdef BUMPCHECK
showbumps() {
    register int i;

    for(i=0; i<bumpcnt; i+=2) {
#ifdef BUMPCROSS
        drawcross(bumppos[i],bumppos[i+1]);
#else
        drawstar(bumppos[i],bumppos[i+1]);
#endif
    }
    bumpcnt = 0;
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  drawstar:   Saves (to the restore arrays) and draws a star
 *              at x,y.  The star looks like:
 *
 *                      x
 *                    xxxxx
 *                    x   x
 *                   xx   xx
 *                    x   x
 *                    xxxxx
 *                      x
 *
 *  In:         x,y - screen coordinates of star center
 *
 *  Notes:
 *
 *  Revisions:    Initial   TCP                 10-mar-87
 *
 */

#ifdef BUMPCHECK
#ifndef BUMPCROSS
drawstar(x,y)
register int x,y;
{
    register int i,clr;

  /* clip to image buffer */
    if((x < 3)||(x > 252)||(y < 3)||(y > 252)) { return; }

    if(curwriteplane == BLUEPLANE) {
        clr = redtoblue(STARCINDEX);
    }
    else clr = STARCINDEX;

  /* four "arms" of star */
    savebg_putclr(x,y-3,clr);
    savebg_putclr(x,y+3,clr);
    savebg_putclr(x-3,y,clr);
    savebg_putclr(x+3,y,clr);

    for(i = -2; i<2; i++) {
        savebg_putclr(x+i,y-2,clr);     /* top */
        savebg_putclr(x+2,y+i,clr);     /* right */
    }

    for(i = -1; i<3; i++) {
        savebg_putclr(x-2,y+i,clr);     /* left */
        savebg_putclr(x+i,y+2,clr);     /* bottom */
    }
}
#endif
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  drawcross:  Saves (to the restore arrays) and draws a cross
 *              at x,y.  The cross looks like:
 *
 *                      x
 *                      x
 *                    xx xx
 *                      x
 *                      x
 *
 *  In:         x,y - screen coordinates of cross center
 *
 *  Notes:
 *
 *  Revisions:    Initial   TCP                 10-mar-87
 *
 */

#ifdef BUMPCHECK
#ifdef BUMPCROSS
drawcross(x,y)
register int x,y;
{
    register int i,clr;

  /* clip to image buffer */
    if((x < 3)||(x > 252)||(y < 3)||(y > 252)) { return; }

    if(curwriteplane == BLUEPLANE) {
        clr = redtoblue(STARCINDEX);
    }
    else clr = STARCINDEX;

  /* four "arms" of cross */
    savebg_putclr(x,y-2,clr);
    savebg_putclr(x,y-1,clr);
    savebg_putclr(x,y+1,clr);
    savebg_putclr(x,y+2,clr);
    savebg_putclr(x-2,y,clr);
    savebg_putclr(x-1,y,clr);
    savebg_putclr(x+1,y,clr);
    savebg_putclr(x+2,y,clr);
}
#endif
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  savebg_putclr:
 *
 *  In:
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 10-mar-87
 *
 */

savebg_putclr(x,y,clr)
register int x,y,clr;
{
    register int addr,tmp;

    addr = fbaddr(x,y);
    fbget(addr,tmp);

    if(curwriteplane == BLUEPLANE) {
        restoreB[restoreBcnt++] = addr;
        restoreB[restoreBcnt++] = greentoblue(tmp);
#ifdef DEBUG
        if(restoreBcnt >= MAXSAVE) { shutdown(); }
#endif
    }
    else {      /* curwriteplane == REDPLANE */
        restoreR[restoreRcnt++] = addr;
        restoreR[restoreRcnt++] = greentored(tmp);
#ifdef DEBUG
        if(restoreRcnt >= MAXSAVE) { shutdown(); }
#endif
    }

    fbput(addr,clr);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  gettranslucent:     Compute the translucent color as a weighted
 *                      average of the translucent voxel and the one
 *                      behind it.
 *
 *  In:         nptr - pointer to translucent voxel
 *              c1   - color of translucent voxel
 *
 *  Out:        translucent color
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 10-nov-86
 *         Adapted to new colormap scheme  TCP   7-feb-87
 *
 */

#ifdef TRANSL
gettranslucent(nptr,c1)
register int nptr,c1;
{
    register int c2;

#ifdef TRANSP
    do {
        nptr += 2;
        fbread(nptr,c2);
        if (c2 == IKNULL) { return(c1); }
    } while((c2&TAGMASK)==curtransp);
#else
    nptr += 2;
    fbread(nptr,c2);
    if(c2 == IKNULL) { return(c1); }
#endif

  /* If dexel is not a bond return front color;  */
  /* else return bond shade    */
    if((c2&TAGMASK) != BONDTAG) {
        return(c1);
    }
    else {
        return(c2);
    }
}
#endif
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  shutdown:   Reset various Ikonas attributes to their standard
 *              settings.
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                13-nov-86
 *
 */

shutdown() {
    register int i;

  /* Enable writes to all bits in frame buffer */
    writemask(ALLBITS);

  /* Reset frame buffer controller registers */
    FBC[2] = STDWIND;
    FBC[3] = STDZOOM;

  /* Reset cross bar registers */
    for(i=0; i<NUMXBAR_REGS; i++)
        XBAR[i] = i;
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  asr:        Arithmetic shift right (1 bit) for 2's complement
 *
 *  In:         x - value to shift
 *
 *  Out:        shifted value
 *
 *  Notes:      Adapted from Mike Pique's "fastspheres" code.
 *
 *  Revisions:     Initial   TCP                16-feb-87
 *
 */

asr(x)
register int x;
{
    x = SMTC x;         /* convert from 2's to sign-magnitude */
    x = SRA x;          /* right shift with sign bit not participating */
    x = SMTC x;         /* convert from sign-magnitude to 2's */
    return(x);
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  abs:        return the absolute value of the argument
 *
 *  In:         arg - argument
 *
 *  Out:        absolute value
 *
 *  Revisions:     Initial   TCP                 16-feb-87
 *
 */

abs(arg)
register int arg;
{
    if(arg<0)
        return(-arg);
    else return(arg);
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  sign:       Compute sign of argument
 *
 *  In:         arg - argument
 *
 *  Out:        {-1, 0, 1} as arg is {< = >} zero
 *
 *  Revisions:     Initial   TCP                 16-feb-87
 *
 */

sign(arg)
register int arg;
{
    if(arg<0)      return(-1);
    else if(arg>0) return(1);
    else           return(0);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  transparent: Reset the transparent object and redraw the image
 *
 *  In:          objno - new transparent object number
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 14-nov-86
 *
 */

#ifdef TRANSP
transparent(objno)
register int objno;
{
    curtransp = objno;
    redraw();
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  translucent: Reset the translucent object and redraw the image
 *
 *  In:          objno - new translucent object number
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 17-nov-86
 *
 */

#ifdef TRANSL
translucent(objno)
register int objno;
{
    curtransl = objno;
    redraw();
}
#endif
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  changeplane: Reset the clipping plane and redraw the image
 *
 *  In:          planeno - new clipping plane number
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                17-nov-86
 *
 */

#ifdef CLIPPLANES
changeplane(planeno)
register int planeno;
{
    curclip = planeno;
    redraw();
}
#endif
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  punchcenter: Reset the punch center and redraw the image
 *
 *  In:          pcx,pcy - new punch center
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                17-nov-86
 *
 */

#ifdef CLIPPLANES
punchcenter(pcx,pcy)
register int pcx,pcy;
{
    punchx = pcx;
    punchy = pcy;
    redraw();
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  punchradius: Reset the punch radius and redraw the image
 *
 *  In:          prad - new punch radius
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 17-nov-86
 *
 */

#ifdef CLIPPLANES
punchradius(prad)
register int prad;
{
    punchr = prad;
    redraw();
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  bumpenable: Enable/disable bump checking
 *
 *  In:          state - on/off
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 11-mar-87
 *
 */

#ifdef BUMPCHECK
bumpenable(state)
register int state;
{
    checkbumps = state;
    redraw();
}
#endif
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  drugdepthcue: Reset the drug depth cueing
 *
 *  In:          k - new depth cue parameter
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                  21-mar-86
 *
 */

drugdepthcue(k)
register int k;
{
    K = k;
    redraw();
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  redraw:     Redraw the image
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 17-nov-86
 *
 */

redraw() {

    Bcorrupt = TRUE;  restoreBcnt = 0;
    Rcorrupt = TRUE;  restoreRcnt = 0;
    doclip(curz);
    call switchplanes;
}

  /* end ikclip.g */
```

```
/*
 *      dev.h:  Header file for dev.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

#define IKINTSCALE 32767        /* "One" scaled up to Ikonas integer */

#define TOOTHPICK "/dev/adtp2"  /* x,y toothpick and z knob on grey box */
#define KNOBS     "/dev/adrp1"  /* knobs 1 through 8 on grey box */

#define THRESHOLD      (4)      /* Delta value for device thresholding */
#define KNOBMAX     (2047)      /* Knob range is -2047 to +2047 */

#define PI     (3.14159265)
#define TWO_PI (6.2831853)

#define ZCLIP_SCALE  (4)        /* MUST be (int) IKZMAX/(KNOBMAX * 2) */

#define ROTX_SCALE  (0.0016666667)
#define ROTY_SCALE  (0.0016666667)
#define ROTZ_SCALE (-0.0016666667)

#define TRANSX_SCALE (0.125061064973)   /* 256/2047 */
#define TRANSY_SCALE (-0.125061064973)
#define TRANSZ_SCALE (8)


/* :::  Globals  ::::::::::::::::::::::::::::::::::::::::::::::::  */

int rot_fd;             /* x,y,z rotation device file descriptor */
int cliptrans_fd;       /* z clip/x,y,z translate device file descriptor */


  /* end dey.h */
 /* matrix multiplication package generator common code - M. Pique. Feb 79 */
element m1[NR1][NC1], m2[NR2][NC2], rm[NR1][NC2];
{
register element *p1, *p2;
element *p1org, *p2org ; /* Origins for inner product scan */
element *p1orglim, *p2orglim; /* Limits for outer row-col loops */
element *resultp; /* pointer into workarea */
element work[NR1*NC2];  /* workarea for building result  */
register element *p1lim;  /* Limit for inner product scan */
resultp = &work[0];

for(p1org= (element *) m1, p1orglim = p1org + P1ORGRANGE;
    p1org<p1orglim;
    p1org += P1ORGSTEP){

        for(p2org= (element *) m2, p2orglim= p2org + P2ORGRANGE;
            p2org<p2orglim;
            p2org += P2ORGSTEP){
                for( *resultp=0, p1=p1org, p1lim= p1+P1RANGE, p2=p2org;
                   p1<p1lim;
                   p1 += P1STEP, p2 += P2STEP)
                        *resultp += PRODUCT( *p1, *p2);
                resultp++;
                }
        }
/* copy result from workarea into result array */
for( p1 = &rm[0][0], p2 = &work[0], p1lim = p1 + NR1*NC2;
    p1<p1lim;
    *p1++ = *p2++)
   ;

}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

       dev.c:  contains:

    >>>>    initdevices:
    >>>>    openknobs:
    >>>>    opentoothpick:
    >>>>    polldevices:
    >>>>    getangles:
    >>>>    getcliptrans:
    >>>>    raacalc:
    >>>>    dmatmat:

                                        Thomas C. Palmer
                                        10-nov-86
*/

/* $Header: dev.c,v 1.4 87/03/26 21:51:02 palmer Exp $ */
static char rcsid[] = "$Header: dev.c,v 1.4 87/03/26 21:51:02 palmer Exp $";

#include <math.h>
#include "/grip4/palmer/docktool/spheres/standard.h"
/* #include "/unc/palmer/src/lib/stdlib.h" */
#include "config.h"
#include "docktool.h"
#include "dev.h"

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  initdevices:  Initialize devices
 *
 *  Notes:
 *
 *  Revisions:    Initial   TCP               10-nov-86
 *
 */

void initdevices() {

    openknobs();
    opentoothpick();
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  openknobs:  Open the knobs device.
 *
 *  Errors:          Can't open knobs
 *
 *  Revisions:    Initial   TCP               10-nov-86
 *
 */

openknobs() {

    cliptrans_fd = open(KNOBS, 0);
    if(cliptrans_fd < 0) {
        shutdown("Can't open clipping/translation device. Bye.\n", -1);
    }
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  opentoothpick:      Open the toothpick device.
 *
 *  Errors:         Can't open toothpick
 *
 *  Revisions:     Initial   TCP                 10-nov-86
 *
 */

opentoothpick() {

    rot_fd = open(TOOTHPICK, 0);
    if(rot_fd < 0) {
        shutdown("Can't open rotation device. Bye.\n", -1);
    }
}
```

```
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  polldevices:        Poll the toothpick and the knobs
 *                      for user updates
 *
 *  In:         zclip - ptr to zclip value
 *
 *  Out:        code for type of update
 *
 *  Notes:
 *
 *  Revisions:     Initial   TCP                 26-feb-87
 *
 */

polldevices(zclip)
int *zclip;
{
    static FLOAT pitch,yaw,roll;
    static int trans[3];
    int nochange_cliptrans=FALSE;
    FLOAT mat[3][3][3], /* 3 matrices (each 3x3): pitch, yaw, roll */
          newmat[3][3], /* new rotation matrix */
          tmpmat[3][3]; /* tmp matrix: roll in pitch */
    register int i, j;

    static FLOAT axdata[3][9] = {  /* x,y,z axis with outer product */
        { 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 },  /* x */
        { 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 },  /* y */
        { 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 },  /* z */
    };

    switch(getcliptrans(zclip,&trans[0],&trans[1],&trans[2])) {
      case NEW_ZCLIP:
        return(NEW_ZCLIP);
        break;
      case NOCHANGE:
        nochange_cliptrans = TRUE;
        break;
      case NEW_TRANS:
        break;
    }

    switch(getangles(&pitch,&yaw,&roll)) {
      case NOCHANGE:
        if(nochange_cliptrans) return(NOCHANGE);
        break;
      case NEW_ROT:
        break;
    }

  /* Clamp values at +/- 180 degrees */
    while(pitch >  PI) pitch -= TWO_PI;
    while(pitch < -PI) pitch += TWO_PI;
    while(yaw >  PI)   yaw -= TWO_PI;
    while(yaw < -PI)   yaw += TWO_PI;
    while(roll >  PI)  roll -= TWO_PI;
    while(roll < -PI)  roll += TWO_PI;

    raacalc(axdata[0],pitch,mat[0]);
    raacalc(axdata[1],yaw,mat[1]);
    raacalc(axdata[2],roll,mat[2]);

  /* Nest z rotation in x rotation in y rotation */
    dmatmat(mat[0],mat[2],tmpmat);
    dmatmat(mat[1],tmpmat,newmat);

  /* Build integer 4x3 matrix suitable for Ikonas */
```

```
    for(j=0; j<3; j++) {
        for(i=0; i<3; i++) {
            xform[j][i] = IKINTSCALE * newmat[j][i];
            if(xform[j][i] > IKINTSCALE)
                postinfo("Error: bad matrix to Ikonas\n");
        }
        xform[3][j] = trans[j];
    }

    return(NEW_MATRIX);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  getangles:  Check new values from toothpick
 *
 *  In:         pitch,yaw,roll - ptrs to angles
 *
 *  Out:        code for type of update
 *
 *  Notes:          Adapted from Mike Pique's fastspheres code.
 *
 *  Revisions:     Initial   TCP               9-feb-87
 *
 */

getangles(pitch,yaw,roll)
FLOAT *pitch, *yaw, *roll;
{
    static short values[3];
    static short oldvalues[3];
    register int i,changed=FALSE;

    read(rot_fd, values, sizeof(values));

    if(abs(values[0] - oldvalues[0]) < THRESHOLD &&
       abs(values[1] - oldvalues[1]) < THRESHOLD &&
       abs(values[2] - oldvalues[2]) < THRESHOLD) {
        for(i=0; i<3; i++)
            values[i] = oldvalues[i];
    }
    else {
        changed = TRUE;
        for(i=0; i<3; i++)
            oldvalues[i] = values[i];
    }

    *pitch = values[0] * ROTX_SCALE;
    *yaw   = values[1] * ROTY_SCALE;
    *roll  = values[2] * ROTZ_SCALE;
    if(changed)
        return(NEW_ROT);
    else return(NOCHANGE);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  getcliptrans:   Check new values from knobs
 *
 *  In:         zclip,tx,ty,tz - ptrs to values
 *
 *  Out:        code for type of update
 *
 *  Notes:          Adapted from Mike Pique's fastspheres code.
 *
 *  Revisions:     Initial   TCP               9-feb-87
 *
 */

getcliptrans(zclip,tx,ty,tz)
int *zclip,*tx, *ty, *tz;
{
    static short values[8];
    static short oldvalues[8];
    register int i,changed=FALSE;

    read(cliptrans_fd, values, sizeof(values));

  /* Check for new Z clipping update */
    if(abs(values[4] - oldvalues[4]) > THRESHOLD) {
        oldvalues[4] = values[4];
        *zclip = (int) ((values[4]+KNOBMAX)*ZCLIP_SCALE); /* Z clip - knob 5 */
        return(NEW_ZCLIP);
    }

  /* Check for new translation update */
    if(abs(values[5] - oldvalues[5]) < THRESHOLD &&
       abs(values[6] - oldvalues[6]) < THRESHOLD &&
       abs(values[7] - oldvalues[7]) < THRESHOLD) {
        for(i=5; i<8; i++)
            values[i] = oldvalues[i];
    }
    else {
        changed = TRUE;
        for(i=5; i<8; i++)
            oldvalues[i] = values[i];
    }

    *tx = (int) (values[6] * TRANSX_SCALE);     /* X - knob 7 */
    *ty = (int) (values[7] * TRANSY_SCALE);     /* Y - knob 8 */
    *tz = (int) (values[5] * TRANSZ_SCALE);     /* Z - knob 6 */

    if (changed)
        return(NEW_TRANS);
    else return(NOCHANGE);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  raacalc:
 *
 *  In:
 *
 *  Out:
 *
 *  Notes:          Adapted from Mike Pique's fastspheres code.
 *
 *  Revisions:     Initial   TCP               9-feb-87
 *
 */

#define frmul(a,b)  (((double)(a))*((double)(b)))
#define frdiv(a,b)  (((double)(a))/((double)(b)))
#define fradd(a,b)  (((double)(a))+((double)(b)))
#define frsub(a,b)  (((double)(a))-((double)(b)))

raacalc(axdata, rotangle, matrix)  /* set matrix from axdata & rotangle */
FLOAT axdata[9];
FLOAT rotangle;
register FLOAT *matrix; /* 3 by 3 matrix, set */
{
    FLOAT  ca1a2, ca2a3, ca1a3, sa1, sa2, sa3;
    struct axdat{ FLOAT a1, a2, a3, a1a1, a1a2, a1a3, a2a2, a2a3, a3a3;};
    register struct axdat *a;

register FLOAT c,s; /* cosine and sine of rotangle */

a = (struct axdat *) &axdata[0];
s = sin(rotangle);
c = cos(rotangle);

/* calculate commonly used terms */
ca1a2 = frmul(c, a->a1a2);
ca2a3 = frmul(c, a->a2a3);
ca1a3 = frmul(c, a->a1a3);
sa1 = frmul(s, a->a1);
sa2 = frmul(s, a->a2);
sa3 = frmul(s, a->a3);

#define diag(x)      (fradd((a->x), (frmul(c, frsub(1.0, (a->x))))))
#define ndiag(x,y,z) (fradd((a->x), fradd((y), (z))))


*matrix++ = diag(a1a1);
*matrix++ = ndiag(a1a2, -ca1a2, -sa3);
*matrix++ = ndiag(a1a3, -ca1a3,  sa2);

*matrix++ = ndiag(a1a2, -ca1a2,  sa3);
*matrix++ = diag(a2a2);
*matrix++ = ndiag(a2a3, -ca2a3, -sa1);

*matrix++ = ndiag(a1a3, -ca1a3, -sa2);
*matrix++ = ndiag(a2a3, -ca2a3,  sa1);
*matrix   = diag(a3a3);
}
```

```c
/* :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        kb.c:  contains:

    >>>>    pollkeyboard:
    >>>>    initkb:
    >>>>    kbunset:
                                          Thomas C. Palmer
                                          12-nov-86
*/

/* $Header: kb.c,v 1.3 87/03/26 21:51:22 palmer Exp $ */
static char rcsid[] = "$Header: kb.c,v 1.3 87/03/26 21:51:22 palmer Exp $";

#include "/grip4/palmer/docktool/spheres/standard.h"
/* #include "/unc/palmer/src/lib/stdlib.h" */
#include "config.h"

#ifndef DBX
#include "kb.h"
#endif

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  pollkeyboard:       Poll the keyboard for user input
 *
 *  Notes:
 *
 *  Revisions:     Initial  (from Grinch keyboard.c)    13-nov-86
 *
 */

#ifndef DBX
void pollkeyboard() {
    char c;

    while (read(fileno(stdin), &c, 1) > 0)
        do_char(c);
}
#endif

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  initkb:     Initialize the keyboard
 *
 *  Notes:
 *
 *  Revisions:     Initial  (from Grinch keyboard.c)    13-nov-86
 *
 */

#ifndef DBX
initkb() {
    int on = 1;

    /*
     **  Open /dev/tty (use stderr if not possible)
     */
    if ( (devtty = fopen("/dev/tty", "w")) )
        setbuf(devtty, NULL);
    else devtty = stderr;

    /*
     ** Turn on non-blocking I/O
     */
    IOCTL(FIONBIO, &on);

    IOCTL(TIOCGETP, &tty);
    IOCTL(TIOCGETC, &chars);
    IOCTL(TIOCGLTC, &lchars);

    /*
     **  Turn on cbreak; turn off echo.
     */
    tty.sg_flags |= CBREAK;
    tty.sg_flags &= ~ECHO;
    IOCTL(TIOCSETP, &tty);
}
#endif

/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  kbunset:    Reset the keyboard characteristics
 *
 *  Notes:
 *
 *  Revisions:     Initial  (from Grinch keyboard.c)    13-nov-86
 *
 */

#ifndef DBX
kbunset() {
    int off = 0;

    /*
     ** Turn off non-blocking I/O
     */
    IOCTL(FIONBIO, &off);

    tty.sg_flags |= ECHO;
    tty.sg_flags &= ~CBREAK;
    IOCTL(TIOCSETP, &tty);
}
#endif

 /* end kb.c */
```

```
/*
 *      screen.h:  Header file for screen.
 */

/* :::  Include files  :::::::::::::::::::::::::::::::::::::::::: */

#undef TRUE
#undef FALSE
#include <curses.h>


/* :::  Defined constants  :::::::::::::::::::::::::::::::::::::: */

                                /* Window sizes (should add to 24)*/
#define INFOSIZE   18           /* we could be smarter and use LINES */
#define DASHSIZE    1           /* and adjust to variable screen sizes */
#define CMDSIZE     5

#define CMDTERM    '\n'     /* newline: command terminator */
#define BACKSPACE  '\010'   /* ascii backspace */
#define CNTL_X     '\030'   /* ascii control-X */
#define CNTL_W     '\027'   /* ascii control-W */

#define MAXCMD      160
#define BIGBUF       2000
#define LONGSTRING  400
#define PROMPTSTR  "\n> "


/* :::  Globals  :::::::::::::::::::::::::::::::::::::::::::::::  */

WINDOW *infowin,
       *dashwin,
       *cmdwin;

char cmdbuf[MAXCMD];            /* Buffer holding command string */


/* end screen.h */
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        screen.c:  Contains:

    >>>>    initscreen:     Initialize curses
    >>>>    getnextcmd:     Get the next command
    >>>>    query_user:     Get a yes/no reply from user
    >>>>    postinfo:       Post a message in the info window
    >>>>    clearinfo:      Clear the info window
    >>>>    posthelp:       Post the help file in the info window
    >>>>    prompt:         Display the prompt
    >>>>    closescreen:    Shutdown curses

    Screen.c contains the screen management functions that
    interface with the library package "curses".  The user is
    presented with three windows:

        - info window:  Cliptool messages
        - cmd window:   user typed commands

                                            Tom Palmer
                                            November 1986
*/

/* $Header: screen.c,v 1.4 87/03/26 21:51:44 palmer Exp $ */
static char rcsid[] = "$Header: screen.c,v 1.4 87/03/26 21:51:44 palmer Exp $";

#include "/grip4/palmer/docktool/spheres/standard.h"
/*#include "/unc/palmer/src/lib/stdlib.h" */
#include "config.h"
#include "docktool.h"
#include "screen.h"
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  initscreen:     Initialize curses and set up cliptool screen windows.
 *
 */

initscreen() {
register int i;

 /* Initialize curses environment.  */
    initscr();
    crmode();
    noecho();

 /* Initialize windows */
    infowin = newwin(INFOSIZE-1, COLS, 2, 0);
    dashwin = newwin(DASHSIZE, COLS, INFOSIZE+1, 0);
    cmdwin = newwin(CMDSIZE, COLS, INFOSIZE+2, 0);

    scrollok(cmdwin, TRUE);
    scrollok(infowin, TRUE);

  /* Draw window boundaries */
    for (i=0; i<COLS; i++) {
        mvwaddch(dashwin, 0, i, '_');
    }

    wmove(infowin, 0, 0);  wrefresh(infowin);
    wmove(dashwin, 0, 0);  wrefresh(dashwin);
    wmove(cmdwin, 0, 0);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  do_char:    Parse an single input character
 *
 *  In:         c - input character
 *
 *  Notes:
 *
 *  Revisions:     Initial      TCP            13-nov-86
 *
 */

do_char(c)
char c;
{
    static int i=0;
    static int column=2;

    switch(c) {
      case CMDTERM:
        cmdbuf[i] = NULL;
        i = 0; column = 2;
        queue_cmd(cmdbuf);
        break;
      case BACKSPACE:
        if (i > 1) i--;
        if (column > 2) column--;
        wmove(cmdwin, cmdwin->_cury, column);
        wclrtoeol(cmdwin);
        wrefresh(cmdwin);
        break;
      default:
        column++;
        waddch(cmdwin,c);
        wrefresh(cmdwin);
        cmdbuf[i++] = c;
    }
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  postinfo:   Post a message in the info window.
 *
 *
 *  In:         buf - message to post.
 *
 */

postinfo(buf)
char *buf;
{
#ifndef DBX
    waddstr(infowin, buf);
    wrefresh(infowin);
    wrefresh(cmdwin);
#else
    fprintf(stdout,"%s",buf);
    fflush(stdout);
#endif
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  clearinfo:  Clear info window.
 *
 */

clearinfo() {
int i;

    wmove(infowin, 0, 0);
    for(i=0; i<INFOSIZE-1; i++) {
        wclrtoeol(infowin);
        wmove(infowin, i, 0);
    }
    wmove(infowin, 0, 0);
    wrefresh(infowin);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  posthelp:   Post a help file in the info window.
 *
 */

posthelp() {
    FILE *help;
    char getbuf[LONGSTRING];
    char displaybuf[BIGBUF];

    clearinfo();
    bzero(displaybuf, BIGBUF);
    help = fopen(HELPFILE,"r");
    if (help == NULL) {
        postinfo("Can't open help file.\n");
        return(ERR);
    }
    while(fgets(getbuf, sizeof(getbuf), help) != NULL)
        strcat(displaybuf, getbuf);

    postinfo(displaybuf);
    fclose(help);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  postplaneshelp:     Post a help file in the info window.
 *
 */

postplaneshelp() {
    FILE *help;
    char getbuf[LONGSTRING];
    char displaybuf[BIGBUF];

    clearinfo();
    bzero(displaybuf, BIGBUF);
    help = fopen(PLANESHELPFILE,"r");
    if (help == NULL) {
        postinfo("Can't open help file.\n");
        return(ERR);
    }
    while(fgets(getbuf, sizeof(getbuf), help) != NULL)
        strcat(displaybuf, getbuf);

    postinfo(displaybuf);
    fclose(help);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  prompt:     Display the prompt in the command window
 *
 */

prompt() {

    waddstr(cmdwin, PROMPTSTR);
    wrefresh(cmdwin);
}
```

```
/* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 *
 *  closescreen:    Shutdown curses.
 *
 */

closescreen()
{
    clear();
    refresh();
    nocrmode();
    endwin();
}

 /* end screen.c */
```

```
/*
 *      docktool.h:  Header file for docktool.
 */

/* :::  Defined constants  ::::::::::::::::::::::::::::::::::::::: */

/* Types of docktool starts */
#define LOADIKSTART 1
#define APUTSTART   2
#define QUICKSTART  3

/* Ikonas data structure loaders: */
#define LOADIK "/grip4/palmer/docktool/bin/loadik"      /* for .edb files */
#define APUT   "/usr/ikonas/aput"                       /* for .ag files */

/* Ikonas BMP code file */
#define IKCLIP_KRFILE  "/grip4/palmer/docktool/dock/ikclip.kr"

/* Help files */
#define HELPFILE       "/grip4/palmer/docktool/dock/lib/help"
#define PLANESHELPFILE "/grip4/palmer/docktool/dock/lib/planeshelp"

#define NOCHANGE    -1  /* No change (under threshold) of device */
#define NEW_ZCLIP    0  /* Device return codes */
#define NEW_TRANS    1
#define NEW_ROT      2
#define NEW_MATRIX   3

#define MAXSTR      80


/* :::  Globals  ::::::::::::::::::::::::::::::::::::::::::::::  */

int xform[4][3];        /* Drug transformation matrix: rot and trans */

boolean debug;          /* If true enable debugging */


/* :::  Macros  :::::::::::::::::::::::::::::::::::::::::::::::  */


  /* end docktool.h */
```

Bibliography

Atherton, Peter R. August, 1981. "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display," *Computer Graphics*, **15**(3), ACM.

Barry, C. D. 1980. , MMS-X Newsletter, 2pp.

Bishop, Gary 1982. *Gary's Ikonas Assembler, Version 2; Differences Between Gia2 and C*, TR82-010 UNC Department of Computer Science, Chapel Hill, NC.

Braunstein, Myron 1976. *Depth Perception Through Motion*, Academic Press, New York, NY.

Bresenham, J. E. February, 1977. "An Incremental Algorithm for Digital Display of Circular Arcs," *Communications of the ACM*, **20**(2), 100-106, ACM.

Connolly, M. L. 1981. , University of California at Berkeley. Master's Thesis.

Connolly, M. L. March 1985. "Depth Buffer Algorithms for Molcular Modelling," *Journal of Molecular Graphics*, **3**(1), Butterworth and Co. Ltd..

Connolly, M. L. 1983. "Analytical Molecular Surface Calculation," *Journal of Applied Crystallography*, **16**, 548-558.

Cory, M. March 17, 1987., Personal Communication.

Fuchs, H., S. M. Pizer, E. R. Heinz, L. C. Tsai, and S. H. Bloomberg. September 1982. "Adding a True 3-D Display to a Raster Graphic System," *IEEE Computer Graphics and Applications*, **2**(7), 73-78.

Fuchs, H., J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, F. P. Brooks, Jr., J. G. Eyles, and J. Poulton. August, 1985. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *Computer Graphics*, **19**(3), 111-120, ACM.

Hook, Tim Van August, 1986. "Real Time Shaded NC Milling Display," *Computer Graphics*, **20**(3), 15-20, ACM.

Kaufman, A. and E. Shimony. October, 1986. "3D Scan-Conversion Algorithms for Voxel-Based Graphics," *1986 Workshop on Interactive 3D Graphics*, UNC Department Computer Science , Chapel Hill, NC.

Kuyper, L. March 31, 1987., Personal Communication.

Langridge, R., T.E. Ferrin, I.D. Kuntz, and M.L. Connolly. 1981. "Real-Time Color Graphics in Studies of Molecular Interactions," *Science*, **211**(4483).

Lee, B. and F. M. Richards. 1971. "The Interpretation of Protein Structures: Estimation of Static Accessibility," *Journal of Molecular Biology*, **55**, 379-400.

Lipscomb, J. S. 1981. *Three-Dimensional Cues for a Molecular Computer Graphics System*, UNC Department of Computer Science, Chapel Hill, NC. Ph.D. Dissertation.

McBurney, D.H. and V.B. Collings. 1977. *Introduction to Sensation/Perception*, Prentice-Hall, Englewood Cliffs, NJ, 178-212pp.

Pique, M. E. 1983. *Fast 3D Display of Space-Filling Molecular Models*, TR83-004 UNC Department of Computer Science, Chapel Hill, NC.

Porter, Thomas K. August, 1978. "Spherical Shading," *Computer Graphics*, **12**(3), ACM.

Rawson, E. G. September 1969. "Vibrating Varifocal Mirrors for 3-D Imaging," *IEEE Spectrum*, **6**(9).

Rogers, D. F. 1985. *Procedural Elements for Computer Graphics*, McGraw-Hill, Inc, New York, NY.

Traub, A. T. 1967. "Stereoscopic Display Using Rapid Varifocal Mirror Oscillations," *Applied Optics*, **6**, 1085-1087.

Wallach, H. and D. N. O'Connell. 1953. "The Kinetic Depth Effect," *Journal of Experimental Psycology*, **45**(4), 205-217.

West, L.J. 1986. "Human Perception," *Encyclopædia Britannica*, **25**, Chicago, Ill.

Wright, William V. 1972. *An Interactive Computer Graphic System for Molecular Studies*, UNC Department of Computer Science, Chapel Hill, NC. Ph.D. Dissertation.