

**DNIF77**

**A Program for Shading Molecular Models**

by

**Catherine Helen Johnson**

An essay

presented to the University of Waterloo

in partial fulfillment of the

requirements for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, 1982

• C. H. Johnson

## **ACKNOWLEDGEMENTS**

I am grateful to my supervisor, Professor Kellogg S. Booth, for providing both the topic of this essay and assistance in its development.

I wish also to thank the members of the Computer Graphics Laboratory, especially Paul Breslin and Darlene Plebon, for their advice concerning the Ikonas 3000 frame buffer and its associated software.

# DNIF77

## A Program for Shading Molecular Models

### 1. Introduction

DNIF77 is a FORTRAN 77 extension of the original FORTRAN DNI program written by Nelson Max of Lawrence Livermore Laboratory as a post-processor to add colour shading and highlighting capability to the ATOMS program [Max79a, Max79b]. ATOMS produces monochrome colour perspective depictions of space-filling and ball-and-stick crystal and molecular structures [Know77].

DNI generates a binary input tape for a minicomputer-controlled Dicomed D48 graphic film recorder, which plots directly on black and white or colour film at a maximum resolution of 4096 by 4096 pixels. Colour images are generated by multiply exposing the appropriate areas of the film at the desired brightness levels through a series of colour filters, one at a time; black and white pictures use only a neutral filter.

DNIF77 is capable of displaying its molecular input on or offline using an Ikonas 3000 frame buffer attached to a DEC PDP 11/45 computer running under the UNIX operating system. This configuration is a raster scan display with a two-dimensional array of 512 by 512 32-bit words of refresh memory representing each position, or pixel, on its screen. Each pixel contains a black and white greyscale or colour intensity level. More generally, an index into a colour lookup table specifies the desired colour of the corresponding location on the display screen. An important characteristic of frame buffer memory is its random accessibility.

This essay explains the shading algorithm of the original DNI program and describes the frame buffer implementation. It includes a discussion of goals, justification of the choice of source language, and an explanation of program input and all subroutines, outlines suggestions for performance optimization, and indicates areas of future development for which DNIF77 will provide the basis.

## 2. Shading and Highlighting in DNI (Dicomed Version)

This computation is described in detail as background for the algorithm chosen for the frame buffer implementation.

ATOMS and ATOMLLL divide the visible segments of each atom (sphere) and bond (cylinder) into a set of trapezoids with straight vertical edges and curved or straight horizontal edges. DNI may thus conveniently shade these trapezoids in vertical segments.

The shading equations arise in the following way. For the sphere centred about the point  $C = (XC, YC, ZC)$ , having radius  $R$ , with  $P = (X, Y, Z)$  some arbitrary point on its surface, one can derive the following equation:

$$\cos^2\vartheta = \frac{R^2 - (X - XC)^2 - (Y - YC)^2}{R^2},$$

where  $\vartheta$  = the angle of incidence of the light source with the sphere (assumed to originate from a point infinitely far behind the observer) [Max79a, Max79b].

Because the sphere's constituent trapezoids have vertical sides for a particular vertical scan segment,  $R, X, XC$ , and  $YC$  are fixed constants in the above equation, which then reduces to a quadratic equation in  $Y$ . This quadratic is evaluated efficiently using the technique of finite differences.

If we let

$$f(y) = \cos^2\vartheta$$

then

$$\Delta f = f(y + h) - f(y)$$

$$\Delta^2 f = f(y + 2h) - 2f(y + h) + f(y)$$

where  $\Delta f$  and  $\Delta^2 f$  are the first and second finite differences, respectively, and  $\Delta^2 f$  is a constant equal to  $f''(y)$  when  $f(y)$  is a quadratic equation. The following tableau illustrates the evaluation of  $f(y)$  for  $y$  incrementing by some fixed

quantity  $h$  at each step:

$f$	$\Delta f$	$\Delta^2 f$
$f(y) = k_1$		
	$k_2$	
$f(y + h) = k_1 + k_2$		$k_3$
	$k_2 + k_3$	
$f(y + 2h) = k_1 + 2k_2 + k_3$		$k_3$
	$k_2 + 2k_3$	
$f(y + 3h) = k_1 + 3k_2 + 3k_3$		
<i>etc.</i>	<i>etc.</i>	<i>etc.</i>

NOTE :  $k_1$ ,  $k_2$ , and  $k_3$  are the variable names in DNI and DNIF77 representing the current values of  $f(y)$ ,  $\Delta f$ , and  $\Delta^2 f$  respectively.

Shading of cylinders is accomplished in a similar fashion. However, it is not always possible to use the same quadratic equation for each value of  $y$  along a vertical segment. Cylinders become cones after a perspective transformation. When the projected edges of the cone lie in different quadrants, the half-cone bounded by the edge with negative slope will be shaded with a quadratic polynomial on horizontal segments, and the other half with different polynomials on vertical segments [Max79a, Max79b].

Although the values of  $\cos^2\theta$  obtained in the above manner represent intensity levels, they are unsuitable for direct use as exposure levels on film. Rather, they serve as indices into colour lookup tables whose entries compensate for differing light transmission characteristics of the film for different colour filters. These colour translation tables are computed once at the beginning of the program by subroutine *setabl*. When the Dicomed receives an exposure level, it automatically indexes into the translation table for the currently active colour filter to obtain the corresponding exposure level.

The computation of these translation tables will now be described, since an understanding of their derivation is crucial to the development of analogous tables for the frame buffer implementation.

The Dicomed operates by positioning, one at a time, colour filters through which light is directed onto the photographic film. The file 'fort.4' (FORTRAN logical unit 4) contains a table of measured film densities for each colour filter at selected output intensity levels. By definition, the density of film is

$$D = -\log_{10}(\text{measured intensity}),$$

or that fraction of light transmitted by the film†. Thus film density decreases as output intensity level increases.

For each filter colour, as well as for black and white, a unique table of 256 exposure codes is calculated. First, the equation for density  $D$  is inverted, yielding a parallel table of measured intensities

$$MI = 10^{-(D/100)}$$

Then desired output intensities (i.e., exposure levels) for the full range [0, 255] of values for  $\cos^2\vartheta$  are interpolated from these measured intensities, and scaled up to the range of measured intensity values for the particular colour filter, using the equations

$$\begin{aligned} F(\vartheta) &= .1 + .9(\cos^2\vartheta)^{.61} \\ &= .1 + .9\cos^{1.22\vartheta} \end{aligned} \quad \text{for colour}$$

and

$$\begin{aligned} G(\vartheta) &= .08F(\vartheta) + .39(.8(\cos^2\vartheta)^{14} + .2(\cos^2\vartheta)^{14/4}) \\ &= .08F(\vartheta) + .39(.8\cos^{28\vartheta} + .2\cos^7\vartheta) \end{aligned} \quad \text{for highlights}$$

When  $F$  and  $G$  are thus simplified, it becomes apparent that they are perturbations of Lambert's Cosine Law for diffuse reflection :

$$f(\vartheta) = A + D \cos\vartheta$$

where

---

†From the equations in the program, it appears that these tables actually contain scaled densities of the form  $-100\log_{10}(\text{measured intensity})$ .

$A$  is ambient light from all sources, and

$D$  represents diffuse reflectivity of light off surfaces

and Phong's empirical model for specular reflection (or highlights), which concentrates intensity for small values of  $\vartheta$ :

$$g(\vartheta) = C \cos^n \vartheta$$

where

$C$  is specular reflection of light from the surface and

$n$  is some 'large' number

[Max79a, Fole82].

The extra terms in  $G(\vartheta)$  causing the marked departure from  $g(\vartheta)$  are to eliminate the optical illusion of Mach bands around the central highlight [Max79a].

Key observations in the subroutine *setabl* of Max's DNI program are:

- the variable

$$fk = \text{float}(k) / 255, \quad 0 \leq k \leq 255$$

actually represents  $\cos^2 \vartheta$ ,  $0 \leq \cos^2 \vartheta \leq 1$

- the variables  $d1$  and  $d2$ , described above as  $F(\vartheta)$  and  $G(\vartheta)$ , are roughly the equations  $f$  and  $g$  of Lambert's Law and Phong's empirical model.

### 3. Ikonas Frame Buffer Implementation

The initial phase of this implementation involved familiarization with the FORTRAN and PASCAL versions of the program. It was then necessary to decide on the language of implementation, subject to the constraints that:

- the chosen language be available on the PDP 11/45
- there exist a convenient means of communication with the Ikonas frame buffer input/output software, written in the language C.

The PDP 11/45 currently supports FORTRAN 77 and C language compilers and a PASCAL interpreter, the latter having no separate module interpretation or linking and loading capability.

Porting the PASCAL version of DNI from the Honeywell 66/60 computer to the PDP 11/45 was rejected as an alternative. Not only would subsequent

program development using the interpreter be tediously slow, but there is no way of calling C language routines directly from this implementation of PASCAL. A separate program would have to be written to transmit an output file from PASCAL DNI to the Ikonas.

The UNIX FORTRAN 77 compiler accepts FORTRAN as a subset, and permits separate module compilation and calls to subroutines written in C. In addition, it produces the same intermediate language as the C compiler; C code generation, optimization, and execution time performance profiling capabilities are thus accessible from FORTRAN 77. A further advantage of FORTRAN 77 is that it can directly produce a Dicomed input file in the required 16-bit binary format; the PASCAL version of the program used an intermediate step to translate ASCII output into 16-bit binary Dicomed input.

For all these reasons, it was decided to use FORTRAN 77 for the initial development of the frame buffer implementation. The work required to get the original FORTRAN version of DNI running on the PDP 11/45 was minimal, as outlined in APPENDIX I. Certainly, translating the program to C is a better long-term solution. Performance would be improved, as high usage modules could be translated into microcode destined for residence in the Ikonas microprocessor memory.

Having obtained a FORTRAN 77 version of the program, extensive recoding was undertaken to modularize lengthy subroutines (*main*, *trapez*), eliminate 'do-nothing' code, define COMMON blocks in UNIX 'include' files, and take advantage of the if-then-else construct. At each stage of this revision, the Dicomed output files resulting from fixed test data were compared against output for the same data from the original program. A by-product of this activity was the detection of two bugs in the original version of DNI†.

The next phase of the project was to enhance DNI to produce both Dicomed and Ikonas output.

The organization of computation in DNI is determined strictly by the operating characteristics of the Dicomed film recorder. Highlighting must be performed as a separate pass over identical data, because the correct filter and colour translation table must be pre-selected prior to exposure of the film. The plotting resolution on the Dicomed is either 1024 by 1024, 2048 by 2048, or 4096

†See APPENDIX I.



by 4096 pixels. At the lower resolutions, 'macro-pixels' greater than 1 pixel in dimension are automatically shaded.

The following issues were addressed when introducing the frame buffer output capability to DNI:

- desire to achieve output device independence until the lowest possible level of processing
- selection and specification of an appropriate 'visible window' for display, given the maximum resolution of 512 by 512 pixels on the Ikonas.
- choice of an appropriate colour representation, given the facts that the Ikonas uses an RGB colour representation, and its colour display properties bear no resemblance to those of photographic film.
- desire to produce a modular program useful as a tool for future investigation in such topics as anti-aliasing and colour shading and correction.

The methods chosen to handle these issues are described in the appropriate subroutine summaries of the following program documentation.

#### 4. Program Documentation

##### 4.1. Input

The molecular description file 'fort.9' (FORTRAN logical unit 9) is produced by ATOMLLL. It consists of 220 rows, each containing 10 integers in 10i7 format.

These 220 rows correspond to the 11 arrays, each of length 200, belonging to COMMON block /PARAM/, contained in the file 'param.cm':

n, kt, kb, xl, xr, xct, yct, rt, xcb, ycb, rb.

n(i) represents the record type, which is either a sphere (atom), cylinder (bond), trapezoid, end-of-frame, or end-of-job. The  $i^{\text{th}}$  elements of all 11 vectors in /PARAM/ comprise the complete set of given information about a specific object in the molecular description, and will subsequently be referred to as a record.

It should be noted that the interpretation of each field in a given record depends on that record's type indicator, n(i). For instance, if n(i) indicates a trapezoid record, kt(i) describes its top arc, whereas kt(i) represents the colour of a spherical record.

In general, each cylinder or sphere record is followed by a list of its constituent trapezoid records. Next comes an end-of-frame record, and finally, an end-of-job record.

This knowledge was used to construct the admittedly artificial, but manageable input test files

```
/u/chjohnson/data/cyl1  
    /cyl2  
    /sph2
```

These files represent, respectively, the horizontal and vertical cylinder and the large red sphere of Fig. 1 in APPENDIX VI.

## **4.2. Main Program**

The main program controls the processing of the ATOMLLL-generated input data described in the previous section. Colour shading and highlighting are performed as two separate passes over the input data when Dicomed output is desired, or simultaneously for Ikonas frame buffer output.

Program activity falls very naturally into the categories of initialization, record processing, and output; it is within this functional framework that the subroutines comprising DNIF77 will be discussed.

## **4.3. Initialization**

### **4.3.1. block data**

This subroutine initializes all static constants in COMMON blocks at compile time.

### **4.3.2. init**

This routine queries the user for program execution control parameters; recommended dialogues for both output devices are described in APPENDIX II. A feature of these dialogues is that they loop until valid responses are given. The control parameters are stored in the COMMON blocks /DEBUG/, /FBPARAM/, and /PGMCTL/. Spacing constants and Dicomed control parameters dependent on the output device resolution are also calculated, and, for frame buffer output,

the subset of input data to be plotted is specified.

The latter point requires clarification. The Dicomed film recorder has a maximum resolution of 4096 by 4096 pixels, and a minimum resolution of 1024 by 1024 pixels, whereas the resolution of the Ikonas frame buffer is 512 by 512 pixels. Frame buffer output assumes a device resolution of 1024 (which may be altered by changing the values of the integers `minxd`, `miny`, `maxxd`, and `maxy` in the DATA statement at the beginning of the routine).

By default, the program will display on the Ikonas the 512 by 512 array of intensities calculated for that portion of the input data centred about the pixel location (512, 512). Optionally, the user may select any pixel location whose  $x$  and  $y$  coordinates are within the range [1, 1024] as the centre of his 'visible window'. When the chosen centre is too close to a boundary, the displayed picture will of course be smaller than 512 by 512 pixels.

#### 4.3.3. `setabl`

This routine computes colour lookup tables indexable by values of  $\cos^2\vartheta$  obtained as the solution to the quadratic shading equations. The purpose of these tables is to compensate for nonlinearities in colour perception on the output device, as well as the film, in the case of the Dicomed. There are significant differences in the tables used by the Dicomed and the Ikonas. The underlying derivation of tables for both devices is similar, although the task is simpler in the case of the Ikonas.

In the Ikonas implementation, Max's file of measured film densities is irrelevant, although some equivalent measure is desirable for gamma correction. Two tables are used — one for all colours and one for highlighting — representing the functions  $F(\vartheta)$  and  $G(\vartheta)$  described in Section 2. (These two tables could be combined into a single one by adding corresponding elements for colour and highlights and storing only the resulting sum). The table entries are scaled, not to the range of measured film densities, but rather, to the range [0, 255], on the assumption that the full 8-bit range of intensity for all colours was desired†.

To simplify table lookup at the time exposure levels are emitted to the Ikonas, the function  $G$  is identically equal to 0 when the user requests no

†This assumption could be altered by defining different functions  $F$  for each colour.

highlighting of the displayed molecule.

#### 4.3.4. *reed*

This routine reads the descriptions of visible molecules created by ATOMLLL into the 11 consecutive arrays of /PARAM/. The program depends on the fact that these arrays are consecutive, and declared in a specific fixed order. The conventions of FORTRAN 77 file I/O require that the input file 'fort.9' (FORTRAN logical unit 9) be rewound from its initial position at end-of-file prior to the issue of the read command.

### 4.4. Record Processing

No changes to this logic were necessitated by the addition of frame buffer output. Rather, the goal with this part of the program was to decompose it into functional modules.

#### 4.4.1. *begcyl*

Originally part of the main program, this routine is called to initiate processing of cylinder (bond) records. A bug causing execution failure on input data containing only descriptions of bonds was corrected†. The cylinder top, bottom, and bisecting boundary line type, slope, and intercept are initialized, as are flags indicating the type of quadratic differencing and shading schemes to apply.

#### 4.4.2. *begsph*

This routine performs an analogous function to *begcyl* for sphere (atom) records. Intermediate values required by the quadratic differencing equations are calculated.

#### 4.4.3. *newcol*

This routine is called by *begcyl* and *begsph* to update information related to the colour of the object currently being shaded.

---

†See APPENDIX I.

#### 4.4.4. *trapez*

This routine controls the shading of all spheres and cylinders, which have been subdivided into trapezoids by the ATOMLLL pre-processor. It has been considerably shortened by extracting into separate modules sections representing specific shading strategies. The general tasks it performs include calculation of the radius and slope of the trapezoid top and bottom arcs. Then *trapez* sweeps horizontally across the trapezoid, calculating the number of scanlines to be shaded, and initially positioning the cursor at the first point to be plotted before calling the appropriate routine to shade the type of object from which the trapezoid arises.

#### 4.4.5. *shdsph*

Originally part of *trapez*, this routine computes starting values for the quadratic differencing algorithm before passing control to *quad* to generate the required number of output intensities.

#### 4.4.6. *shdcyl*

The complexity of shading cylinders is greater than that of shading spheres, as explained in Section 2.

First, the top, bottom, and middle coordinates of the bisecting (or highlighting) line are obtained. Different shading equations are required above and below this line, unless it is strictly vertical, in which case the entire trapezoid can be shaded using the same quadratic equation.

#### 4.4.7. *shdah*, *shdbh*

Originally part of *trapez*, these routines shade a vertical scanline of a cylinder's trapezoid above or below its bisecting line, respectively, using a different quadratic equation on each successive horizontal line.

#### 4.4.8. *shvlin*

This routine is a generalization of two nearly identical sections of code in the original version of *trapez*. It is called to shade a vertical scanline above or below the bisecting line of a trapezoid using the same quadratic equation for the entire region; *quad* computes the desired intensities once the first and second differences are initialized. Note that only one of *shdah* and *shvlin*, and one of

*shdbh* and *shulin* is called to shade above and below the bisecting line, depending on the nature of the projected cylindrical image.

#### 4.4.9. quad

This routine is an efficient implementation of the quadratic differencing algorithm described in Section 2. It packs two output intensities per word, requiring one division (or shift) and two additions to calculate each intensity.

### 4.5. Device Output

In extending the capability of DNI to produce frame buffer output, the underlying goal was to achieve some measure of device independence for output†. To this end, all calls in the original program to the routine *dicowd* were replaced by calls to *output*, with an additional parameter describing the nature of the data destined for *output*. For each type of data known to the routine, the appropriate action is taken, depending on the destination device. It is thus easy to add new devices to the program's repertoire.

#### 4.5.1. output

When the Dicoméd is the chosen output device, *output* passes the data directly to *dicowd*, or calls *dicemp* to flush the buffer of all accumulated exposure information. It is in the case of Ikonas output that the processing depends on the nature of the input to the routine.

The data type 'noop' is reserved for Dicoméd control information; such data can be ignored when Ikonas output is selected.

Data flagged as the starting x or y plotting coordinate must be saved for future use once it has been scaled down from the range [1, 4096] used by the program for all pixel address computation‡.

Exposure intensities destined for the Dicoméd may be emitted either one or two per output data word. The frame buffer implementation must remember the format of the emitted data in order to unpack it, if generated by the routine *quad*, before translating it to the appropriate frame buffer representation. The number of output exposure codes must also be saved so that the final exposure

†This is contrary to the suggestions in [Yuen80], which seem to imply eliminating any capability of producing Dicoméd output in a frame buffer implementation.

‡Because this range is so deeply embedded in the Dicoméd address calculation of the original DNI program, it is easier to perform this single rescaling for Ikonas output.

in packed format is ignored if this number is odd (a function performed automatically by the Dicomed hardware).

When *output* is called to emit an exposure level, it must first ensure that the destination pixel is within the frame buffer visible window. If so, the colour and highlight intensities corresponding to the input  $\cos^2\vartheta$  exposure level are obtained by table lookup in  $F(\vartheta)$  and  $G(\vartheta)$ , and converted to the colour representation required by the Ikonas display (three 8-bit bytes, right-justified, representing red, green, and blue components respectively, from the right side of the word). Shading and highlighting are thus performed in a single pass. The two intensities are simply added together one byte at a time to ensure that no byte exceeds the maximum permissible value of 255. Then the desired intensity is stored in a scanline buffer for transmission to the Ikonas.

#### 4.5.2. **exit**

This routine closes all input and output files before terminating execution of the program.

#### 4.5.3. **Dicomed Output Support Routines**

##### 4.5.3.1. **frmhdr**

This routine calls *output* to emit the sequence of Dicomed control commands required to set up a film frame header.

##### 4.5.3.2. **dicowd**

This routine accumulates information destined for the Dicomed in a fixed-length buffer, emptying it to output file 'fort.8' (FORTRAN logical unit 8) as a binary stream of 16-bit integers when the buffer becomes full.

##### 4.5.3.3. **dicemp**

This routine flushes the current contents of the Dicomed output buffer to the file 'fort.8'.

#### 4.5.4. Ikonas Output Support Routines

##### 4.5.4.1. itorgb

The following table describes the mapping between an 8-bit intensity level for each of the colour filters available on the Dicomed and its corresponding RGB representation on the Ikonas:

Colour	Red	Green	Blue
red	X	0	0
green	0	X	0
yellow	X	X	0
blue	0	0	X
magenta	X	0	X
cyan	0	X	X
white	X	X	X

In other words, when the desired 8-bit illumination intensity is X for a known colour, this routine returns a 24-bit value containing X in the fields corresponding to the RGB representation for that colour.

#### 4.5.5. FORTRAN 77 / C Interface Routines

These are C language routines called from FORTRAN 77 to establish communication with the Ikonas I/O software. Conventions for such inter-language communication are outlined in APPENDIX III.

##### 4.5.5.1. Fclofb\_

This routine closes the frame buffer output pathname opened during the previous call to *Finifb\_*.

##### 4.5.5.2. Finifb\_

This routine sets global variables needed by the Ikonas simulator I/O routines in the file '/u/phbreslin/sim/fb\_io.c'. It then calls *init\_fb* (also in .../fb\_io.c) to open either the Ikonas itself, or a simulated frame buffer output file.



#### 4.5.5.3. *Fputpx*

This routine converts arguments from their FORTRAN 77 type(s) to their expected C type(s) before calling a simulator routine to store a value in a specified pixel location.

#### 4.5.5.4. *get\_storage*

Also part of the simulator I/O package, this routine allocates a buffer one scanline long for frame buffer I/O.

### 5. DNIF77 Performance Optimization

The current version of the program is by no means optimal in terms of storage and CPU usage. Following are suggested approaches toward improving runtime performance.

- (1) As described in Section 4.1, input data from ATOMLLL requires a central memory allocation of 2200 32-bit words, a very high proportion of which usually contain zeroes. Since the records spanning these 11 arrays are internally type-coded, including a special end-of-job record, there is no need for ATOMLLL to write out fixed blocks of input to DNIF77, aside from convenience as a magnetic tape format.
- (2) Analysis of the results of running DNIF77 in frame buffer mode in conjunction with the UNIX performance profiler *prof* indicate a full 25% of execution time is spent performing integer division and multiplication. These operations simulate shifting and masking, both in the rescaling of data and packing more than one unit of information in a single word, as done by *quad* and *itorgb*. Any division or multiplication by a power of two could be replaced, either by a call to a C language function using actual shift instructions, or ideally, a microprocessor-resident micro-coded version of such a C function. In fact, the same argument applies to all high-usage routines in the program (eg. *output*, *itorgb*, *quad*).
- (3) As noted in Section 4.3.3, the double lookup of colour and highlight intensities in routine *output* could be eliminated by storing the sum of these intensities in a single table. This would also significantly reduce the amount of shifting and masking performed, since the byte by byte addition of two RGB intensities from *itorgb* could be replaced by a single invocation of *itorgb*.

- (4) A bottleneck in the current program is the sequence of subroutine calls

$$quad \rightarrow output \rightarrow \begin{cases} itorgb \\ Fputpx \end{cases}$$

This sequence preserves the hierarchical nature of the processing, at the same time introducing redundant processing. *quad* packs two 8-bit intensity codes per word, which must immediately be unpacked by *output*. To eliminate this extra work and the overhead of several subroutine invocations, *quad* could be modified so that, for each 8-bit intensity code within the boundaries of the visible frame buffer window, the work of *itorgb* and *Fputpx* are performed by inline code. If such an approach were adopted, one would have to ensure that the inline and subroutine versions of this code always remained in step with each other.

- (5) A problem arises due to the discrepancy in resolution of the Ikonas and Dicomed devices. It is often possible for an intensity to be calculated, only to be rejected for output to the frame buffer when the destination pixel is found to be outside the visible window. One could circumvent this situation by scaling all input data from ATOMLLL downward by a factor of two; however, this immediately creates a dual problem of multiple intensities calculated for the same pixel. Alternatively, program computations could be adjusted to permit 512 by 512 pixel resolution when the Ikonas frame buffer is the selected output device.

## 6. Areas of Future Development

### 6.1. Colour Shading and Correction

Comparison of output generated directly on the Ikonas monitor with the Polaroid photographs of APPENDIX VI reveals several 'features'.

Monitor output tends to exhibit a Mach band around the central highlights, which is removed by subsequent photographing of the generated image. This suggests that by modifying the terms added to the equation for  $G(\vartheta)$  described in Section 2 to eliminate Mach bands in the Dicomed output, one could obtain a better quality image.

Photography of monitor output tends to blur the boundaries of atoms too quickly to black; this could possibly be improved by modifying the ambient light contribution in the equation for  $F(\vartheta)$ .

Another characteristic of frame buffer output is the high concentration of white light in the central highlight. This could be adjusted, for instance, by scaling the values of  $G(\vartheta)$  to some subrange of the full range of 256 possible intensities.

## 6.2. Antialiasing

In general, the outer edges of all atoms (spheres) and bonds (cylinders), as well as all boundaries of their constituent trapezoids, should be antialiased. However, examination of program output suggests several shortcuts can be taken. Natural boundaries of atoms and bonds appear smooth, as the calculated intensities fall off gradually to the background ambient lighting conditions. As well, neither shape of object exhibits internal faceting — i.e., we do not see the boundaries of the trapezoids from which an object is constructed. Based on these empirical observations, it appears that jagged lines in need of antialiasing correction occur where the natural boundary of an object is superimposed on another object. In other words, when an atom is occluded by either another atom or a bond, the trapezoid boundaries of the occluded portion of the atom require antialiasing.

Specific instances of candidates for antialiasing appear in the sample program output of APPENDIX VI. For example:

Fig. 1 : the top and bottom edges of trapezoids in the yellow atom which intersect with the bond

Fig. 2 : the intersections of the red and blue atoms

Fig. 3 : the intersections of the red and yellow atoms

Fig. 4 : the non-vertical intersections of the green atoms with each other

To incorporate antialiasing into DNIF77, one must first devise a method of determining occluding trapezoid boundaries, perhaps by considering absolute changes in boundary slope between adjacent trapezoids of different objects. For the  $x$  and  $y$  coordinates thus selected, one could find the differences between their corresponding floating point and integer values. These differences could then be used as  $x$  and  $y$  coordinate input to *shdsph* or *shdcyl* to calculate an intensity by which the appropriate adjacent pixel would be fractionally

illuminated. Appropriate in this sense should be interpreted as  $x - 1$  ( $y - 1$ ) or  $x + 1$  ( $y + 1$ ), depending whether the difference between floating point and integer values of  $x$  ( $y$ ) is less or greater than 0.5.

A limited form of antialiasing could be achieved by using the double resolution (1024 by 1024) information available from the colour shading computation.  $\frac{1}{4}$  of the calculated intensity for each member of a group of four adjacent pixels could simply be added to produce the output intensity for the destination Ikonas pixel. More generally, any weighting function could be used to derive the individual contribution of each adjacent pixel to the final output intensity [Crow77]. No matter what the selected antialiasing algorithm is, it should be microprocessor-resident to minimize its impact on total program execution time.

## 7. Conclusion

DNIF77 extends the capability of Nelson Max's DNI program to provide output for an Ikonas frame buffer. The shading algorithm of DNI was studied, and Ikonas colour lookup tables computed using a method similar to the derivation of those for the Dicomed. The approach works, but still requires refinement to obtain 'perfect' synthetically-generated molecules.

This essay documents the DNIF77 implementation, evaluates its performance, and suggests topics for further investigation, which should be facilitated by the modular organization of the program.

## REFERENCES

- Catm79 Catmull, E. "A tutorial on compensation tables," *Computer Graphics* **13**, 2 (Aug. 1979), 1-7.
- Crow77 Crow, F. C. "The aliasing problem in computer-generated shaded images," *Communications of the ACM* **20**, 11 (Nov. 1977), 799-805.
- Dico79 DICOMED Corporation. Graphic film recorder model D48 operation and programming manual. February 1979 edition, revision A.
- Feld78 Feldman, S. I., and Weinberger, P. J. "A portable FORTRAN 77 compiler," Bell Laboratories, Murray Hill, New Jersey, Aug. 1978.
- Fole82 Foley, J. D., and Van Dam, A. *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Mass., 1982.
- Know77 Knowlton, K., and Cherry, L. "ATOMS — a three-d opaque molecule system — for colour pictures of space-filling or ball-and-stick models," *Computers and Chemistry* **1**, 3 (1977), 161-166.
- Max79a Max, N. "ATOMLLL — a three-d opaque molecule system, Lawrence Livermore Laboratory version," UCRL-52645, Lawrence Livermore Laboratory, University of California/Livermore, Jan. 1979.
- Max79b Max, N. "ATOMLLL:- ATOMS with shading and highlights," *Computer Graphics* **13**, 2 (Aug. 1979), 165-173.
- Yuen80 Yuen, H. "PASCAL DNI, a program for shading molecular models," Master's Essay, Department of Computer Science, Faculty of Mathematics, University of Waterloo, 1980.

**APPENDIX I**  
**Adaptation of DNI to FORTRAN 77**

**1. Modifications**

- (1) `#include 'filename'` was changed to include `'filename'`
- (2) Initialization of hexadecimal constants:  
eg. `zd000` was changed to `z'd000'`
- (3) The non-default logical input units 4 and 9 must be opened and rewound; they are otherwise positioned automatically at end-of-file.
- (4) All calls to subroutine *llshift* were replaced by division by  $2.0^{22.0}$ .

**2. Bugs Detected and Corrected**

- (1) the comma after column 72 in the declaration of COMMON block `/isp/` in the main program must be moved to the following line to be recognized; otherwise two variable names are concatenated, since intervening blanks and all information after column 72 are ignored by FORTRAN.
- (2) the line `ad = a / two 15` must be added following the call to *newcol* when beginning a new cylinder; otherwise the program fails if its only input is cylinder(s).

**APPENDIX II**  
**DNIF77 User's Guide**

**1. File Requirements in Current Directory**

**1.1. Dicomed or Ikonas Output**

fort.9† : input data generated by ATOMLLL

**1.2. Dicomed Output**

fort.4 : table of measured film densities

fort.8 : will be appended to unless removed from the current directory prior to execution of DNIF77

**Notes:**

- (1) Both of the following dialogues may be carried out either interactively, or using a file of pre-determined responses, as in  
dnif77 <dialogue\_responses
- (2) An invalid response to any query will regenerate the same question until an acceptable response is obtained.

**2. Dialogue to Obtain Dicomed Output**

User-supplied information appears in italics in the following dialogue.

*dnif77*

Target output device:

(1:Dicomed, 2:Ikonas) free format:

*1*

Wait between frames, double film advance, debug trace:

(0:no, 1:yes) 3i1 format:

*NNN*

---

†All references to files named fort.*n* represent FORTRAN logical unit *n*.

Resolution : 1024, 2048, 4096 : free format:

*NNNN*

**Notes:**

- (1) Select 'wait between frames' if special operator handling is required before signalling restart to the film recorder.
- (2) Select 'double film advance' if the exposed film is to be mounted as 35 mm. slides.
- (3) Selection of 'debug trace' option generates output on the file 'fort.7' in the current directory.

**3. Dialogue to Obtain Frame Buffer Output**

User-supplied information appears in italics in the following dialogue.

*fb\_init*

*dnif??*

Target output device:

(1:Dicomed, 2:Ikonas) free format:

*2*

Compute highlights?

(0:no, 1:yes) free format:

*N*

Type of frame buffer:

(0:actual, 1:simulated) free format:

*N*

Frame buffer debug output desired?

(0:no, 1:yes) free format:

*N*

By default, centre portion of ATOMLLL data will be displayed on frame buffer.

Do you wish to change area displayed?

(0:no, 1:yes) free format:

*N*

*x, y* coordinates (in Dicomed raster units) of centre of area to be displayed:

free format:

(Must be in range 1 to 1024)

*NNNN NNNN*



**Notes:**

- (1) Atoms with no highlights will be shaded solid colours; highlighting adds white light radiating out from the angle of incidence of the user viewpoint with the surface of the display screen.
- (2) It is useful to select simulated frame buffer output when a picture is desired quickly, or the frame buffer is in use. The resulting file can be displayed later, as explained below.
- (3) Frame buffer debug output will appear on the file 'fort.7' in the current directory.
- (4) The last question is asked only when the user indicates a desire to alter the default 'visible window'. Depending how close the chosen centre is to the boundaries of the ATOMLLL data, the dimension of the displayed region will be  $\leq 512$  pixels in either direction.

**3.1. Saving an Image Generated Directly on Ikonas Display**

Issue the command

```
/u/daplebon/bin/save_raster destination_save_file_name
```

**3.2. Manipulation of Simulated Frame Buffer Output**

A file called 'frame\_buffer' will have been created in the current directory during execution of DNIF77. It can either be displayed on the Ikonas or examined interactively using the Ikonas simulator.

**3.2.1. Displaying 'frame\_buffer' on the Ikonas**

Issue the command

```
/u/ikonas/bin/draw_raster frame_buffer
```

**3.2.2. Using the Ikonas Simulator**

Issue the command

```
/u/ikonas/bin/iksim +o
```

The +o option tells the simulator to use the existing file 'frame\_buffer' as input.

The commands

```
$L scan y
```

```
$L x,y
```

permit examination of scanline  $y$  and pixel location  $(x, y)$  respectively.

`$q`

exits the simulator. At this point, one can elect to save or release the file 'frame\_buffer'. Since this file is very large, it should only be saved if absolutely necessary for later display or examination.

### APPENDIX III

#### Interlanguage Communication Between FORTRAN 77 and C

Following are the major factors to be considered when defining C procedures called from FORTRAN 77:

- (1) The FORTRAN 77 statement  
    call f(x)  
    requires a corresponding C definition of a procedure  
    f(x)
- (2) All FORTRAN 77 arguments to subroutines and functions are passed by address. Hence they must be declared as pointers to variables of the appropriate types within the called C procedures.
- (3) The returned type of a C function called from FORTRAN 77 must be declared in the calling routine. For such declarations, the reader is referred to the table of corresponding FORTRAN 77 and C declarations in the FORTRAN 77 User's Guide [Feld78].

## APPENDIX IV

### DNIF77 Maintenance Manual

The following files in the directory /u/chjohnson/dni/f are required to compile and load DNIF77:

- (1) all FORTRAN 77 source routines (.f suffix)
- (2) all COMMON block declarations (.cm suffix)
- (3) FC\_interface.c — a file containing 'intermediary' C routines called by DNIF77 to access Ikonas simulator I/O software, also written in C.
- (4) fbsim.lib — an archive library (in UNIX ar format) containing /u/phbreslin/sim/fb\_io.o. The C source corresponding to fbsim.lib is saved under

/u/chjohnson/dni/f/iksim\_io/fb\_io.c.

Ideally, the Ikonas simulator library should be used for this purpose, to ensure that the most up-to-date version of the required routines is accessed. However, loader problems involving duplicate copies of the C library routine *calloc* necessitated this short-term solution.

The UNIX utility *make* is used to maintain an up-to-date executable load module for the program. The file '/u/chjohnson/dni/f/makefile' (attached at the end of this APPENDIX) contains a set of dependencies for each object routine belonging to the final load module; these dependencies include the corresponding source routine and all files included in it ( eg. COMMON blocks).

**N.B.** It is imperative that any changes in included files in the source be reflected in the appropriate section of the makefile. Otherwise, the automatically-recompiled object module will be out of synchronization with its corresponding source language.

#### **Procedure for Modifying DNIF77**

- (1) edit source files

- (2) update makefile file dependencies, if necessary
- (3) execute the UNIX command 'make' in the directory containing 'makefile'
- (4) run the program, requesting Dicomed output, using as input (fort.9) the files

/u/ chjohnson/ dni/ data/ cyl1

/ cyl2

/ sph2

- (5) After each run in (4) above, execute the command
 

```
cmp -l fort.8 /u/chjohnson/dni/data/output/X
```

 where X is cyl1, cyl2, or sph2 respectively. This will ensure that the new version of the program still produces the same Dicomed output as the original†.
- (6) test the new version of the program on the frame buffer. Discrepancies of 1 between corresponding bytes in the two files compared are deemed acceptable (eg. sph2), as they represent floating point to integer roundoff.

#### Listing of makefile

FC = f77 -c -C -O -U

FLGO = f77 -i \$(OBJECTS) -o dnif77 fbsim.lib

OBJECTS = block.o dni.o begcyl.o begsph.o dicemp.o dicowd.o exit.o frmhdr.o\  
 init.o itorgb.o newcol.o output.o quad.o reed.o setabl.o shdah.o\  
 shdbh.o shdcyl.o shdsph.o shvlin.o trap.o FC\_interface.o

dnif77:       \$(OBJECTS)  
               \$(FLGO)

begcyl.o:     begcyl.f bond.cm coltbl.cm manif.cm param.cm qdiff.cm  
               \$(FC) begcyl.f

---

†This is a more realistic method of verification than comparing Dicomed disassembler output from old and new versions. Disassembler output contains unnecessary repetitive text, making the files too long for examination by most UNIX utility functions.

begsph.o: begsph.f coltbl.cm ispace.cm manif.cm param.cm qdiff.cm\  
radius.cm sphere.cm  
\$(FC) begsph.f

block.o: block.f coltbl.cm dicntl.cm dico.cm fbparm.cm ispace.cm\  
manif.cm param.cm  
\$(FC) block.f

dicemp.o: dicemp.f dico.cm  
\$(FC) dicemp.f

dicowd.o: dicowd.f dico.cm  
\$(FC) dicowd.f

dni.o: dni.f bond.cm coltbl.cm debug.cm dicntl.cm dico.cm fbparm.cm\  
ispace.cm manif.cm param.cm pgmctl.cm  
\$(FC) dni.f

exit.o: exit.f manif.cm pgmctl.cm  
\$(FC) exit.f

frmhdr.o: frmhdr.f dicntl.cm manif.cm  
\$(FC) frmhdr.f

init.o: init.f debug.cm dicntl.cm fbparm.cm ispace.cm manif.cm\  
pgmctl.cm  
\$(FC) init.f

itorgb.o: itorgb.f coltbl.cm debug.cm fbparm.cm manif.cm  
\$(FC) itorgb.f

newcol.o: newcol.f coltbl.cm manif.cm  
\$(FC) newcol.f

output.o: output.f coltbl.cm debug.cm fbparm.cm ispace.cm manif.cm\  
pgmctl.cm

```

$(FC) output.f

quad.o:    quad.f manif.cm
           $(FC) quad.f

reed.o:    reed.f manif.cm
           $(FC) reed.f

setabl.o:  setabl.f coltbl.cm debug.cm fbparm.cm manif.cm pgmctl.cm
           $(FC) setabl.f

shdah.o:   shdah.f bond.cm debug.cm dicntl.cm ispace.cm manif.cm qdiff.cm
           $(FC) shdah.f

shdbh.o:   shdbh.f bond.cm debug.cm dicntl.cm ispace.cm manif.cm qdiff.cm
           $(FC) shdbh.f

shdcyl.o:  shdcyl.f bond.cm debug.cm dicntl.cm ispace.cm manif.cm qdiff.cm
           $(FC) shdcyl.f

shdsph.o:  shdsph.f debug.cm dicntl.cm ispace.cm manif.cm qdiff.cm\
           sphere.cm
           $(FC) shdsph.f

shvlin.o:  shvlin.f dicntl.cm ispace.cm manif.cm qdiff.cm
           $(FC) shvlin.f

tabout.o:  tabout.f
           $(FC) tabout.f

trap.o:    trap.f bond.cm debug.cm dicntl.cm ispace.cm manif.cm\
           param.cm qdiff.cm radius.cm sphere.cm
           $(FC) trap.f

FC_interface.o:  FC_interface.c /u/phbreslin/sim/manif.h
                cc -c FC_interface.c

```

**APPENDIX V**  
**Guide to Existing Computer Files**

**CGL UNIX Master Catalogue**

/u/chjohnson/dni

**Subdirectories**

/data	: ATOMLLL-generated input files
/data/output	: output from original DNI program for files of same name under /data
/f	: makefile, source, compiled object code, and executable load module for DNIF77
/f/fb_images	: frame buffer images for Fig. 1 through 4 in APPENDIX VI
/f/iksim_io	: C language source for Ikonas simulator I/O software
/f/original	: DNI converted to FORTRAN 77, as described in APPENDIX I
/f/save	: DNI converted to structured FORTRAN 77 and modularized, prior to adding frame buffer capability

**Honeywell Master Catalogue (H. Yuen)**

gr/./atoms

**Subdirectories**

/atom.jcl  
/tapes



/dni.jcl  
/c.progs  
/essay  
/atom.in  
/source  
/dni.in

### **Files**

/new.1, /new.2: his Fig. 3†  
/new.3, /new.4: his Fig. 5  
/new.5, /new.6: his Fig. 4  
/fort09: his Fig. 2  
/ft09: his Fig. 1

### **VAX Master Catalogue**

/u/cgl

### **Subdirectory**

/chjohnson: Versatec plotter vtroff format input text for this document

---

†[Yuen80].

**APPENDIX VI**  
**Sample Program Output**

The following figures are Polaroid prints taken by a Dunn 631 Colour Camera connected to the Ikonas frame buffer system.

<b>Figure</b>	<b>Input File</b>
1	/u/chjohnson/dni/data/fig1
2	/u/chjohnson/dni/data/fig2
3	/u/chjohnson/dni/data/fig3
4	/u/chjohnson/dni/data/fig4

**Figure 1**

Three atoms with two connecting bonds [Max79a].

**Figure 2**

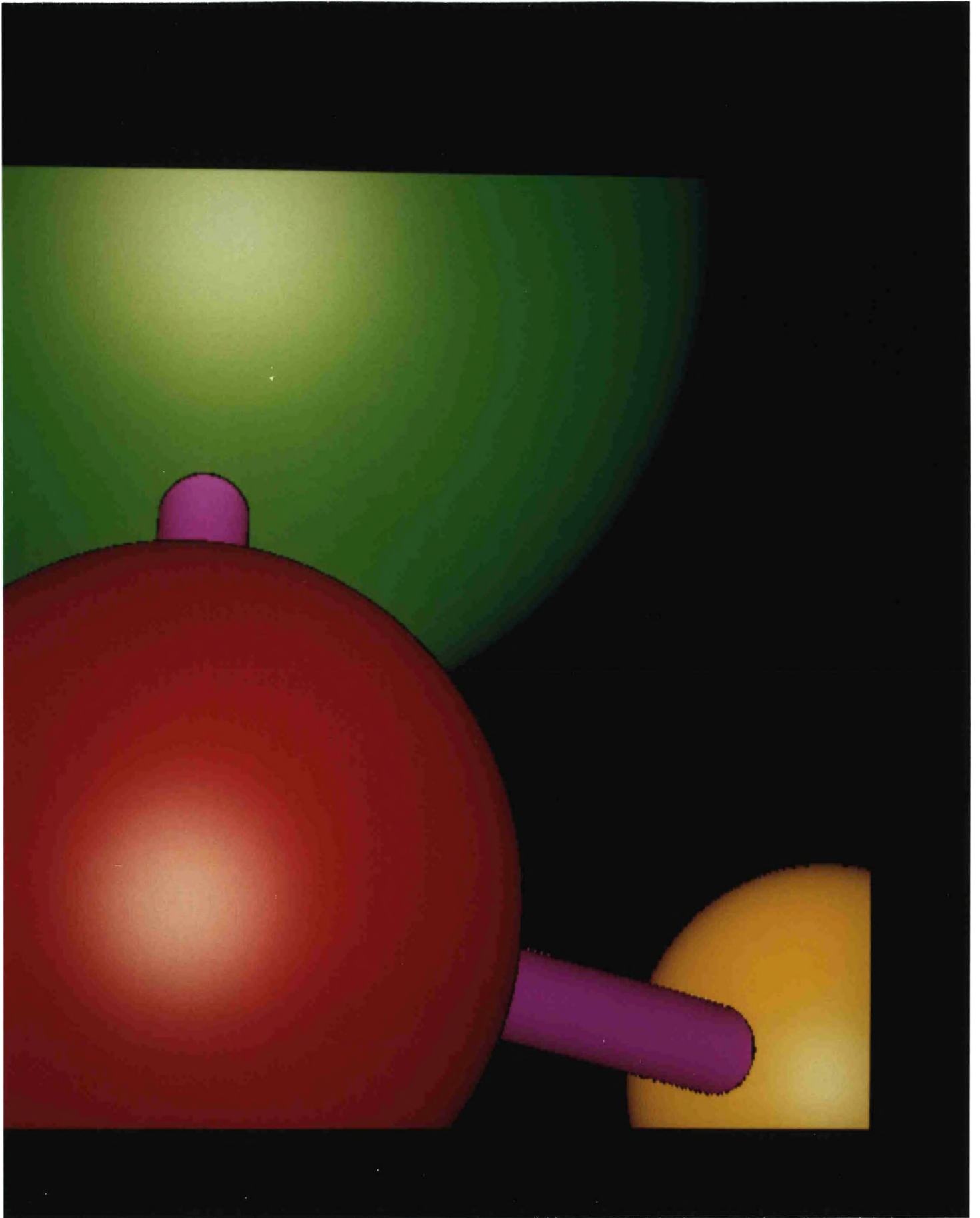
A central atom occluded symmetrically on the outside by four others [Yuen80].

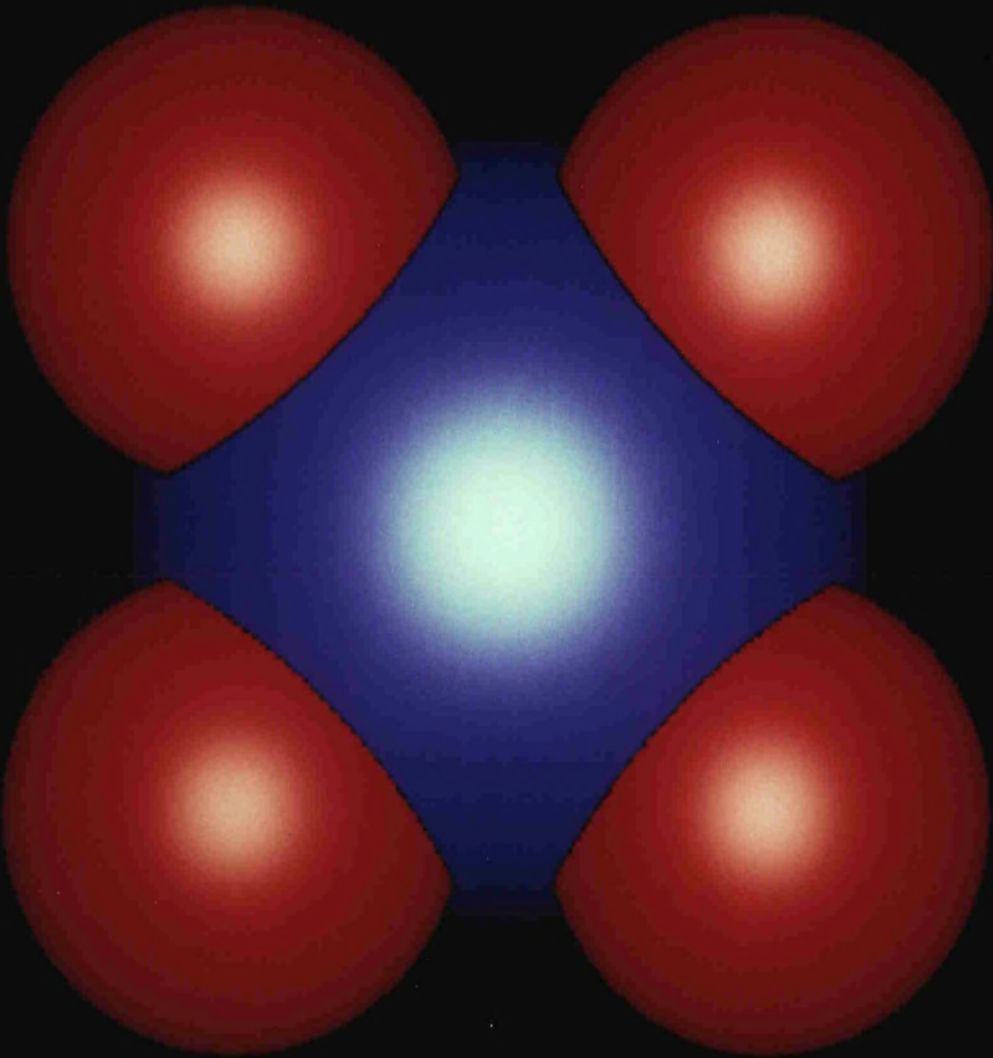
**Figure 3**

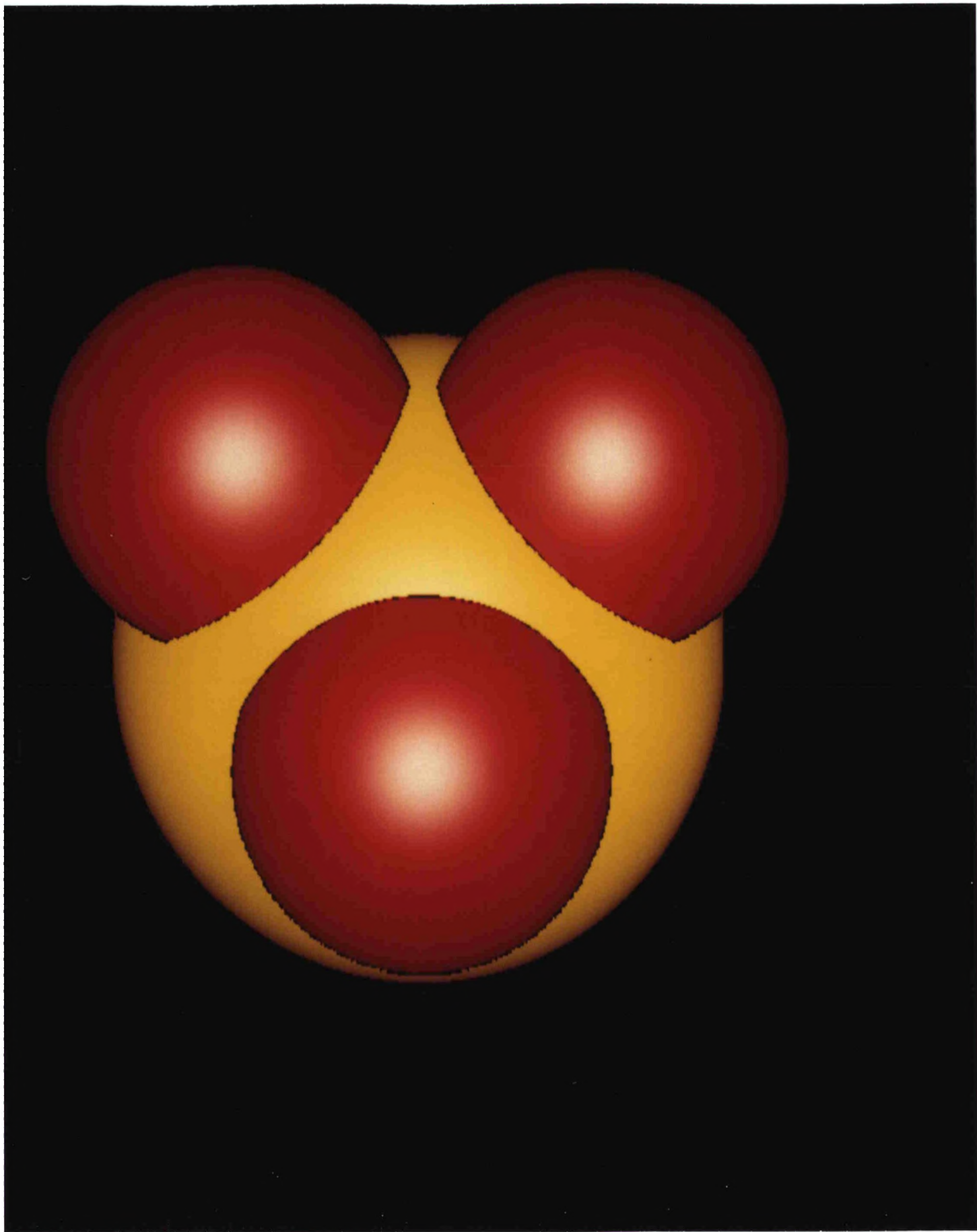
One of the occluding atoms of Figure 2 is hidden by the central atom [Yuen80].

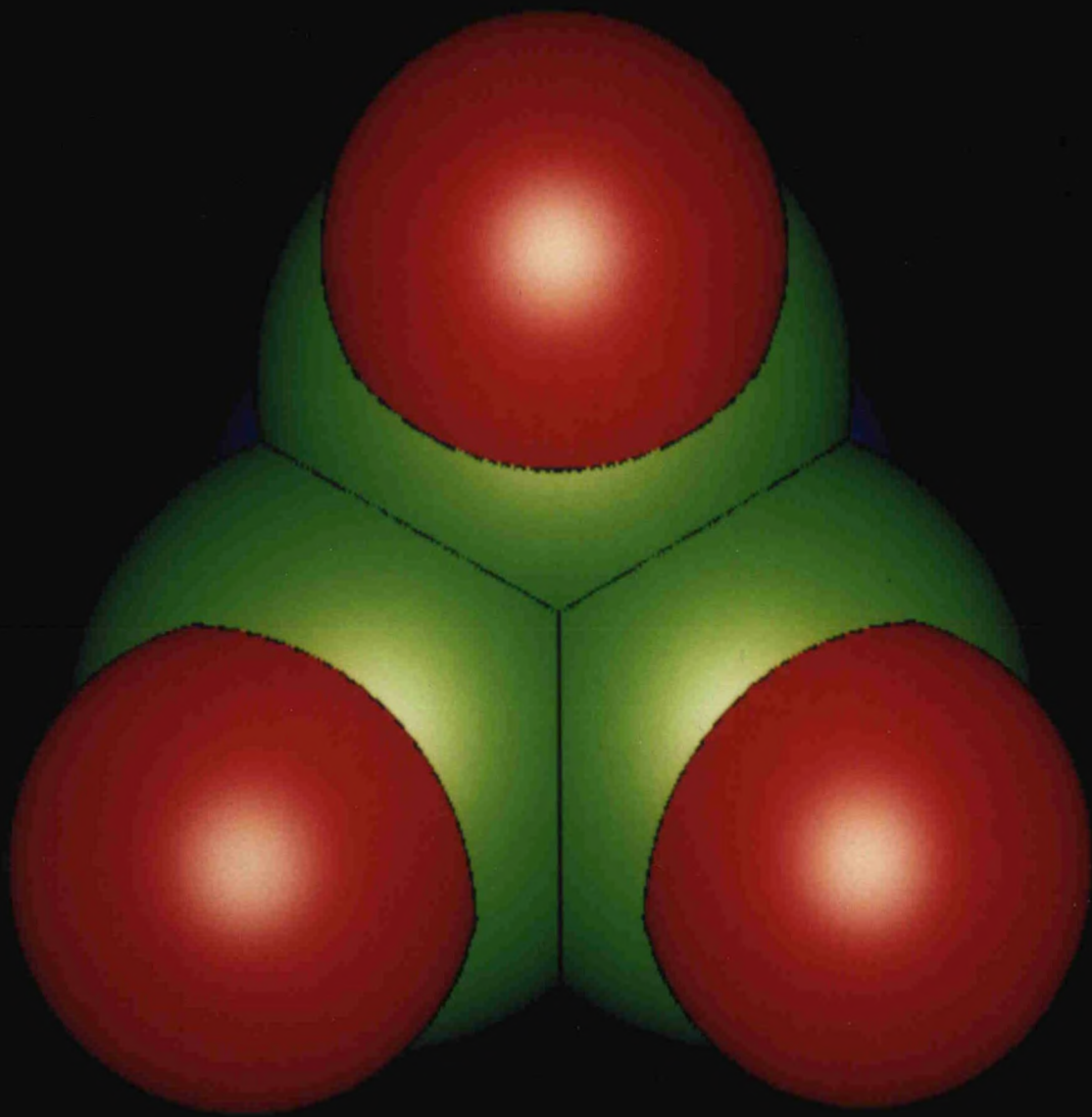
**Figure 4**

In this rendition of seven atoms, only two small pieces of the blue one are visible [Yuen80].









**APPENDIX VII**  
**Program Listings**

- (1) COMMON blocks—alphabetical order
- (2) Main program—file dni.f
- (3) FORTRAN 77 subroutines—alphabetical order
- (4) C procedures—files FC\_interface.c, iksim\_io/fb\_io.c

**bond.cm**

**c====> Molecular Bond Information <====**

**c====> These variables are all initialized in subroutine begcyl.**

```

common / BOND / jt, st, rit,
&          jb, sb, rib,
&          jm, sm, rim,
&          fm, fs,
&          markt, markb

c          jt, st, rit : top line type, slope, intercept
c          jb, sb, rib : bottom " " "
c          jm, sm, rim : middle " " "

c          where line type is :
c          1 : convex down
c          2 : convex up
c          3 : straight
c          4 : special (for normalization)
c          [ abs(slope) > 1 ]

c          fm, fs      : used for intermediate value calculation
c                    in differencing scheme when processing cylinders

c          markt, markb : flags telling which quadratic differencing
c                    and shading schemes to use on trapezoid

c                    i.e., shade above/below highlight line with
c                    quadratics on
c                    0 : vertical lines
c                    1 : horizontal lines

```



**coltbl.cm**

**c====> Colour Intensity Tables <====**

```

      common / COLTBL /  app,
&          bpp,
&          nowcol,
&          ihilit

      dimension  app(7, 2),
&              bpp(7, 2)

c  app :   minimum intensity  (initialized in block data routine)
c  bpp :   range of intensity  "  "  "

c  where
c      row index:  colours
c                  1 : red
c                  2 : green
c                  3 : yellow
c                  4 : blue
c                  5 : magenta
c                  6 : cyan
c                  7 : neutral
c      column index :  shading
c                      1 : colour
c                      2 : black & white

c  nowcol :   colour of object currently being plotted
c              (set in subroutine newcol)
c  ihilit ;   colour (1) or highlight (2) computation
c              (set in main program)

```

**debug.cm**

**c===>    Debug control flags <====**

**&        common / DEBUG /    ldbg,  
                              ifbdbg**

**c    ldbg :    general trace & dump of calculated values**

**c    ifbdbg    :    frame buffer debug information**

**c        0 : off**

**c        1 : on**

**dicntl.cm**

**c====> Command codes for Dicomed D48 graphic film recorder <====**

```

common / DICNTL / lcomp,
&          ifa,
&          ifep1,
&          ifep2,
&          ifes1,
&          ifes2,
&          ifp,
&          ipes,
&          ipes1,
&          ipes2,
&          ipes3,
&          ipos,
&          lres

```

**c====> All page references are to D48 Operation & Programming Manual  
c====> (Publication #12M069, February 1979 Edition, Revision A)**

**c====> The variables ipes1, ipes2, & ipes3 are set in subroutine init.**

**c====> All others are initialized in the block data subroutine.**

**c lcomp ICS (Initial condition select); pp. 3.17 - 3.19**

**c Bits set ( right to left ) :**

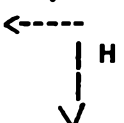
```

c      0: LIN: infinite film exposure code
c      3: AZE: allow 0 exposure (fog level)
c      5: HVINT: interchange horizontal & vertical axes
c      7: HREV: reverse direction of scan along
c          horizontal axis

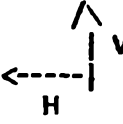
```

**c N.B. Bits 5 & 7 change the axis orientation**

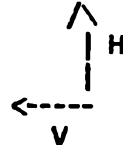
**c from**



**c to**



**c which is actually plotted as**



**c because y coordinates are sent to the Dicomed  
c when x's are expected and v.v.**

**c 11: FBD: disable CRT beam flyback**

- c lfa EFS (exposure & filter select); p. 3.21
- c no change of filter, no film advance, exposure level 0
  
- c lfep1 FES (function element select); pp. 3.8 - 3.14
- c (Always followed by lfep2)
  
- c lfep2 FES function code 3 & ES set; p. 3.12
- c plot the single intensity in the following word
  
- c lfes1 FES
- c (Always followed by lfes2)
- c no deferred status, filter code, film advance,
- c exposure code
  
- c lfes2 FES function code 1 & ES set
- c horizontal raster line will be plotted from the
- c following input stream, containing exposure codes
- c packed 2 / word.
  
- c lfp function 3 output format
- c contains parameters to plot 1 element in direction 0
- c (positive horizontal) at the exposure level in bits 0-7
- c with no change of filter.
  
- c lpes PES (point element select); p. 3.20
  
- c lpes1
- c horizontal element spacing 1- 4 raster units
- c horizontal point spacing 1 raster unit
  
- c lpes2
- c vertical element & point spacing (same as horizontal)
  
- c lpes3
- c background exposure code 0;
- c no. of horizontal points/element = no. of vertical
- c points/element = 1, 2, or 4 (depending on resolution)
  
- c lpos VPA (vector or position absolute); p. 3.15
- c moves CRT beam to position specified by absolute
- c horizontal & vertical coordinates;
- c uses previously-defined exposure code + 2 words:
- c 1st - abs. horizontal 2nd - abs. vertical coordinate
  
- c lres ICS (Initial condition select); pp. 3.17 - 3.19
- c reset D48 to basic operating state described on p. 3-33

**dico.cm**

**c====> Dicomed 16-bit output code buffers <====**

```

      common / DICO / buff,
&          bptr,
&          bufmax

```

```

      integer*2 buff(200)

```

```

&      integer      bptr,
&                  bufmax

```

```

c      buff      :   output buffer; dumped to output
c                  file when full
c                  (set by subroutine dicowd)

```

```

c      bptr      :   pointer to next available slot
c                  in buff
c                  (initialized in block data subroutine;
c                  updated by subroutine dicowd)

```

```

c      bufmax    :   dimension of buff
c                  (defined in block data subroutine)

```

**fbparm.cm**

**c====> Frame buffer output control parameters <====**

```

      common / FBPARAM / minx,
&          miny,
&          maxx,
&          maxy,
&          acminx,
&          acminy,
&          acmaxx,
&          acmaxy,
&          minfbx,
&          minfby,
&          maxfbx,
&          maxfby,
&          minrgb,
&          maxrgb,
&          nextx,
&          nexty,
&          lstart,
&          lend,
&          lcount,
&          shdctl,
&          fthet2,
&          gthet2

```

```

      dimension minrgb (7),
&          maxrgb (7),
&          fthet2 (256),
&          gthet2 (256)

```

```

      integer      acminx,
&          acmaxx,
&          acminy,
&          acmaxy,
&          shdctl,
&          fthet2,
&          gthet2

```

```

c   minx, miny      :   smallest x & y coordinates visible on
c                   frame buffer ( in Dicommed raster units )
c                   (set by subroutine Init, as are maxx & maxy)

c   maxx, maxy     :   largest x & y coordinates visible on frame buffer

c   acminx, acminy  :   smallest calculated coordinates to be sent to
c                   frame buffer ( in Dicommed units )
c                   (set by Init; updated by subroutine output, as
c                   are acmaxx, acmaxy, minfbx, minfby, maxfbx, maxfby)

```

**c acmaxx, acmaxy : largest calculated coordinaes to be sent to**  
**c frame buffer ( In Dicomed units )**

**c minfbx, minfby : least x & y coordinates of an illuminated pixel**  
**c on frame buffer**

**c maxfbx, maxfby : largest x & y coordinates of an illuminated pixel**  
**c on frame buffer**

**c minrgb, maxrgb : minimum and maximum calculated Intensity for**  
**c each colour**  
**c Purpose : to see whether they are too close to**  
**c 255 before adding in highlight component**  
**c (Initialized by block data routine; updated**  
**c by function ltorgb)**

**c nextx, nexty : next point for which Intensity to be emitted**  
**c (set by block data routine; updated by subroutine**  
**c output)**

**c istsart, lend : controls loop extracting exposure codes, which**  
**c occur either 1 or 2 per word.**  
**c (set by subroutine output, as is lcount)**

**c lcount : number of exposure codes remaining to emit for**  
**c current raster line**

**c shdctl : Plot shaded colour only, or include highlights**  
**c (set by routine init)**

**c fthet2 : frame buffer colour lookup table**  
**c (set by routine setabl, as is gthet2)**

**c gthet2 : frame buffer highlight lookup table**  
**c ( contains 0's if no highlighting desired )**

**lspace.cm**

**c====> Spacing parameters <====**

**c====> All variables in this common block are set in subroutine init.**

```

      common / ISPACE /  lsp,
&          lsp2,
&          jsp,
&          jsp2,
&          fjsp,
&          fjsp2,
&          tjsp,
&          tjsq

c      lsp      :      pixel size; single element plotting
c                      resolution
c      lsp2     :      lsp / 2
c      jsp      :      vector size; vector plotting resolution
c      jsp2     :      jsp / 2
c      fjsp     :      float(jsp)
c      fjsp2    :      float(jsp2)
c      tjsp     :      2 * fjsp
c      tjsq     :      2 * fjsp * fjsp

```



**manif.cm**

**c====> Manifest constants <====**

**c====> All variables in this common block are initialized by the block  
c====> data subroutine at compile time.**

```

common / MANIF /idico,
&          lfrbuf,
&          lhoriz,
&          loerr,
&          loff,
&          lon,
&          i2to14,
&          jnoop,
&          jsavex,
&          jsavey,
&          jpacki,
&          junpki,
&          jemiti,
&          jflush,
&          ksph,
&          kcyl,
&          rmaxi,
&          r2to14,
&          smin,
&          two15

```

**c idico : 1 ( Output sent to Dicomed film recorder )**

**c lfrbuf : 2 ( Output sent to Ikonas frame buffer display )**

**c lhoriz : 1**

**c loerr : 2 when error occurs during input**

**c loff : 0 false**

**c lon : 1 true**

**c i2to14 : ( int ) 2 \*\* 14**

**c The following 7 variables are used by the routine output to decide  
c what to do with the accompanying word of integer information;  
c the action chosen depends on the particular output device selected.**

**c jnoop : Used for Dicomed control functions :  
c eg. frame header, setting colour filters, etc.**

**c jsavex : Dicomed starting x raster position**

**c jsavey : Dicomed starting y raster position**

**c jpacki : 2 Dicommed output intensities / word, right-justified**

**c junpk1 : 1 intensity / word**

**c jemiti : intensity to be displayed**

**c jflush : empty Dicommed buffer**

**c ksph: sphere**

**c kcyl : cylinder**

**c rmax1 : ( real ) maximum integer : 32767.0**

**c r2to14 : ( real ) 2 \*\* 14**

**c smin : any arc whose slope is less than smin is**  
**c considered a straight line**

**c two15 : 2. \*\* 22**  
**c ( The logic for this name must be historical! )**

## param.cm

c====> Molecular input data from ATOMSLLL <====

c====> The variable prmlen is set by the block data subroutine;  
c====> all others are set by subroutine reed.

```

      common / PARAM / prmlen,
&          n,
&          kt,
&          kb,
&          xl,
&          xr,
&          xct,
&          yct,
&          rt,
&          xcb,
&          ycb,
&          rb

      integer prmlen,
&          xl, xr,
&          xct, yct,
&          rt, rb,
&          xcb, ycb

      dimension n(200),
&          kt(200),
&          kb(200),
&          xl(200),
&          xr(200),
&          xct(200),
&          yct(200),
&          rt(200),
&          xcb(200),
&          ycb(200),
&          rb(200)

c          prmlen : size of each of following 11 arrays

c          n : record type
c                > 0 : trapezoid
c                -2 : cylinder
c                -3 : sphere
c                -4 : end of job
c                -5 : end of frame

c          kt, kb : top & bottom arc types for trapezoid record

c                1 : convex down
c                2 : convex up
c                3 : straight line

```

**c** **4** : special

**c** top line type & slope for cylinder

**c** colour & x coordinate of centre for sphere.

**c** **xl, xr** : x coordinates of left & right sides of trapezoid

**c** top line intercept & bottom line type for cylinder.

**c** y coordinate of centre & radius for sphere.

**c** **xct, yct, rt** : centre & radius of top arc for trapezoid.

**c** bottom line slope & intercept, & middle line type

**c** for cylinder.

**c** **xcb, ycb, rb** : centre & radius of bottom arc for trapezoid.

**c** middle line slope & intercept, & intermediate

**c** quadratic differencing value for cylinder.

**pgmctl.cm**

**c====> Program control variables <====**

**c====> All variables in this common block are set by subroutine Init.**

```
      common / PGMCTL / ifda,  
&          lwait,  
&          idvice,  
&          ifbsim
```

**c All of the following are user specified at run time:**

**c ifda : double film advance**

**c lwait: generates extra space between completely exposed Dicomed  
c frames so film can be mounted as 35 mm. slides**

**c idvice : output device type  
c (Dicomed film recorder / Ikonas frame buffer)**

**c ifbsim : selects actual or simulated frame buffer output  
c When the frame buffer is simulated, output is placed  
c on the file frame\_buffer in the current directory;  
c this file can be examined using the Ikonas simulator.**

**qdiff.cm**

**c====> Intermediate values used in quadratic differencing calculations <====**

**c====>**

**c====> The equations from which these values originate are detailed in  
c====> section 4 of**

**c====>**

**c====> ATOMLLL - A Three-D Opaque Molecule System**

**c====> (Lawrence Livermore Laboratory Version),**

**c====> N. L. Max, UCRL-52645, Jan. 1979.**

**c====>**

```

      common / QDIFF / a,
&          ad,
&          bp,
&          c,
&          d,
&          e,
&          f,
&          k3

```

**c a : set by routine newcol'**

**c ad : set by routines begcyl & begsph**

**c bp : maximum range for colour exposure  
c (set by routine newcol)**

**c c, d, e, f, k3 : set by routine begsph**

**radius.cm**

**c====> Radius parameters <====**

```
&      common / RADIUS / srt,  
&                srb,  
&                srr
```

```
c  srt      :  top  
c                (set by routine trapez, as is srb)
```

```
c  srb      :  bottom
```

```
c  srr      :  square  
c                (set by routine begsph)
```

**sphere.cm**

**c====> Molecule parameters <====**

**c====> All variables in this common block are set by routine begsph.**

```
& common / SPHERE / r,  
&          xc,  
&          yc
```

```
& integer      r,  
&          xc,  
&          yc
```

**c r : radius**

**c xc : x coordinate of centre**

**c yc : y coordinate of centre**



dni.f

C:~: ---> MAINLINE <---

C:~:

C:~: This program produces shaded highlighted molecular representations  
 C:~: suitable for output on either a Dicomed film recorder or an Ikonas  
 C:~: frame buffer connected to a DEC PDP/11 computer. It is a structured  
 C:~: FORTRAN 77 extension of the original Dicomed-dependent FORTRAN program  
 C:~: obtained from Nelson Max of Lawrence Livermore Laboratory.

C:~:

C:~: References : 1. 'ATOMLLL :- ATOMS with Shading & Highlights',  
 C:~: N. L. Max, Proceedings of SIGGRAPH '79.

C:~:

C:~: 2. 'ATOMLLL - A Three-D Opaque Molecule System  
 C:~: (Lawrence Livermore Laboratory Version)',  
 C:~: N. L. Max, UCRL-52645, Jan. 1979.

C:~:

C:~: 3. Dicomed Graphic Film Recorder Model D48 -  
 C:~: Operation and Programming Manual.

C:~:

C:~:

C:~: Input files :

C:~: fort.4 : measured film densities to compute colour  
 C:~: lookup tables for the Dicomed

C:~:

C:~: fort.9 : molecular description file created by the  
 C:~: ATOMLLL program, which is a front end for  
 C:~: this program.

C:~:

C:~: Output files :

C:~: fort.8 : Dicomed control codes to produce exposed  
 C:~: photographic film on a Dicomed film recorder.

C:~:

C:~: N.B. Due to bugs in the FORTRAN 77 compiler and/or  
 C:~: quirks of the Unix operating system, this file is  
 C:~: always opened in append mode if it already exists.

C:~:

C:~: fort.7 : optional debug output.

C:~:

C:~: user terminal : Execution trace

C:~:

+

C:~: frame buffer statistics

C:~: (if Ikonas is output device)

C:~:

C:~: frame\_buffer : screen image file created for simulated frame  
 C:~: buffer output.

C:~: May be examined with Ikonas simulator or displayed  
 C:~: directly on the Ikonas.

C:~:

C:~: Program execution is controlled by a self-explanatory input dialogue.

C:~:

C:~: An efficient quadratic differencing algorithm computes values for  
 C:~:  $(\cos(\theta) ** 2)$  along vertical lines; these are used as indices  
 C:~: into computed colour lookup tables for shading & object highlighting.  
 C:~: (Details concerning the differencing equations may be found in

c:::: section 4 of Reference 2.)

c::::

c:::: The program requires 2 passes over the input data for the Dicomed,  
c:::: which manipulates colour filters one at a time.

c:::: Shading & highlighting can be performed simultaneously for the Ikonas,  
c:::: which requires only a single intensity in RGB format for each pixel to  
c:::: be illuminated.

c:::: Processing is complete once either an end-of-job indicator or the  
c:::: physical end of input data has been encountered.

c::::

c::::

c:::: N.B. "Do nothing" code from N. Max's original version

c:::: has been commented out.

c::::

```

include 'bond.cm'
include 'coltbl.cm'
include 'debug.cm'
include 'dicntl.cm'
include 'dlco.cm'
include 'fbparm.cm'
include 'ispace.cm'
include 'manif.cm'
include 'param.cm'
include 'pgmctl.cm'

data icolor / 22 /,
&         lbandw / 21 /,
&         ifr / 0 /,
&         newcyl / -2 /,
&         newsph / -3 /,
&         newjob / -4 /,
&         newfrm / -5 /

call init
call output( jnoop, ires )
c====> call setabl( 1 )
call setabl
call frmhdr

30 continue
idum = 0
c====> call tabout( 3 )
50 continue
call reed( n, prmlen * 11, ier )
if ( ier .gt. 0 ) go to 30

c====> Colour shading

kstart = 0
80 ihilit = 1
c====> call tabout( 1 )
call output( jnoop, icolor )

```

```

nowcol = 0
go to 120

c====> Set shading flag to black & white for highlight calculation

100 ihilit = 2
c====> call tabout( 2 )
      call output( jnoop, ibandw )
      nowcol = 0

c====> The following case statement is performed for each record
c====> in every block of input data until either the end-of-job code or
c====> physical end-of-file is encountered.

120 do 500 i = kstart + 1, prmlen
      write(6, 960) i

c====> Continue shading trapezoids in the current atom

      if ( n(i) .gt. 0 ) then
          write(6, 965)
          call trapez( i, kind )

c====> Begin new frame

      else if ( n(i) .eq. newfrm ) then
          write(6, 980)
          if ( ihilit .eq. 1 ) then
              if ( ldvice .eq. ldico ) go to 100
          else
              kstart = i
              call output( jnoop, ifa )
              lfr = lfr + 1
              write(6, 900) lfr
              if ( lwait .eq. lon ) then
c====> call tabout( 3 )
              call output( jflush, jflush )
              read(6, 800) idum
              end if
              if ( lfda .eq. lon ) call output( jnoop, ifa )
              call frmhdr
              if ( kstart - prmlen) 80, 50, 50
          end if

c====> End of Job

      else if ( n(i) .eq. newjob ) then
c====> call tabout( 3 )
          call output( jflush, jflush )
          if ( ldvice .eq. lfrbuf ) then
              write(6, 910) (minrgb(j), maxrgb(j), j = 1, 7)
              write(6, 930)
              write(6, 950) acminx, acmaxx, acminy, acmaxy
              write(6, 940)

```

```

        write(6, 950) minfbx, maxfbx, minfby, maxfby
    end if
    call exit

c====>    Begin New Sphere

        else if ( n(i) .eq. newsph ) then
            write(6, 970)
            call begsph( i, kind )

c====>    Begin New Cylinder

        else if ( n(i) .eq. newcyl ) then
            write(6, 975)
            call begcyl( i, kind )
        end if
500      continue

        if ( ( ihllt .lt. 1 ) .or. ( ihllt .gt. 2 ) ) then
            write(6, 920) ihllt
            call exit
        else
            go to ( 100, 50 ), ihllt
        endif

c====>    Input formats

800      format( I1 )

c====>    Output formats

900      format( ' Finished frame', I4)
910      format( t20, 'Min Intensity', 5x, 'Max Intensity',
&        //'red', t23, I5, t41, I5,
&        //'green', t23, I5, t41, I5,
&        //'yellow', t23, I5, t41, I5,
&        //'blue', t23, I5, t41, I5,
&        //'magenta', t23, I5, t41, I5,
&        //'cyan', t23, I5, t41, I5,
&        //'white', t23, I5, t41, I5 )
920      format( //'Main program : ihllt = ', I5,
&        ' out of range for computed go to' )
930      format( //'Range of calculated pixel addresses : ' )
940      format( //'Range of illuminated pixel addresses : ' )
950      format( t8, 'x : ', I6, ' to ', I6, ' y : ', I6, ' to ', I6 )
960      format( 'Record ...', I5 )
965      format( t20, 'Trapezoid' )
970      format( t20, 'Sphere' )
975      format( t20, 'Cylinder' )
980      format( t20, 'End of frame' )

    end

```

```

subroutine begcyl( I, kind )

c::: This routine is called to initiate processing of a bond (cylinder)
c::: input record.
c:::
c::: Input :
c:::   I :   Index of arrays of / PARAM / containing information
c:::         relevant to this bond.
c:::
c::: Output :
c:::   kind :   indicates cylindrical trapezoid data requires further
c:::             processing.
c:::
c::: N.B.
c:::   1.   Bonds will always be shaded magenta by this program.
c:::
c:::   2.   This routine was originally part of N. Max's dni.f
c:::         mainline module.
c:::

include 'bond.cm'
include 'coltbl.cm'
include 'manif.cm'
include 'param.cm'
include 'qdiff.cm'

data magen   / 5 /

if ( nowcol .ne. magen ) then
  call newcol( 5, a, bp )
c====>   The following line was missing in N. Max's code
c====>   (so the program failed for input data containing only bonds)
  ad = a / two15
end if

c====> Top values

  jt = kt(i)
  st = kb(i) / 32767.
  markt = 0
  if ( ( jt .ne. 1 ) .and. ( abs(st) .le. smln ) ) then
    markt = 1
    jt = 3
  end if
  rit = xl(i)

c====> Bottom values

  jb = xr(i)
  sb = xct(i) / 32767.
  markb = 0
  if ( ( jb .ne. 1 ) .and. ( abs(sb) .le. smln ) ) then

```

```
        markb = 1
        jb = 3
    end if
    rib = yct(i)
```

c====> Middle values

```
    jm = rt(i)
    sm = xcb(i) / 32767.
    if ( ( jm .ne. 1 ) .and. ( abs(sm) .le. smin ) ) then
        markt = 1
        markb = 1
        jm = 3
    end if
    rim = ycb(i)

    if (jb .gt. 1 .and. jm .gt. 1 .and. sb * sm .lt. 0) markb = 1
    if (jt .gt. 1 .and. jm .gt. 1 .and. st * sm .lt. 0) markt = 1

    fm = rb(i) / 32767.
    fs = 255. * fm * bp
    c = fs * two15
    kind = kcy1

    return
end
```

```
subroutine begsph( i, kind )
```

```
c::: This routine is called to initiate processing of an atom (spherical)
c::: Input record.
c:::
c::: Input :
c:::     i :   Index into arrays of / PARAM / to information relevant
c:::           to this atom.
c:::
c::: Output :
c:::     kind :   Indicates spherical trapezoid data for further processing
c:::
c::: Some terms used in the quadratic differencing equations are computed.
c:::
c::: N.B.
c:::     1.   This routine was originally part of N. Max's dni.f
c:::           mainline module.
```

```
include 'coltbl.cm'
include 'lspace.cm'
include 'manif.cm'
include 'param.cm'
include 'qdiff.cm'
include 'radius.cm'
include 'sphere.cm'
```

```
if ( .kt(i) .ne. nowcol) then
    call newcol( kt(i), a, bp )
    ad = a / two15
end if
```

```
xc = kb(i)
yc = xl(i)
r = xr(i)
srr = float(r) ** 2
c = bp * 255. * two15 / srr
d = a + c * srr
e = c * jsp * 2
f = c * jsp * jsp
k3 = -2 * f
kind = ksph
```

```
return
end
```

**block data**

c::: This routine provides initialization of all static constants

c::: In common blocks.

c:::

```
include 'coltbl.cm'
include 'dicntl.cm'
include 'dico.cm'
include 'fbparm.cm'
include 'ispace.cm'
include 'manif.cm'
include 'param.cm'
```

c====> / COLTBL /

```
data app / 14*.01 /,
&      bpp / 6 * 0.99, 0.45, 7 * 0.99 /
```

c====> / DICNTL /

```
data lcomp / z'88a9' /,
&      lfa / z'c100' /,
&      ifep1 / z'2000' /,
&      ifep2 / z'7000' /,
&      ifes1 / z'2000' /,
&      ifes2 / z'3000' /,
&      lfp / z'1000' /,
&      lpes / z'a000' /,
&      lpos / z'4000' /,
&      ires / z'9000' /
```

c====> / DICO /

```
data bptr / 1 /,
&      bufmax / 200 /
```

c====> / FBPARM /

```
data nextx / 0 /,
&      nexty / 0 /,
&      minrgb / 7 * 9999 /,
&      maxrgb / 7 * -9999 /,
&      shdctl / 1 /
```

c====> / MANIF /

```
data ldico / 1 /,
&      lfrbuf / 2 /,
&      lhoriz / 1 /,
&      loerr / 2 /,
&      loff / 0 /,
```



```
&      lon / 1 /,  
&      l2to14 / 16384 /,  
&      jnoop / 1 /,  
&      jsavex / 2 /,  
&      jsavey / 3 /,  
&      jpacki / 4 /,  
&      junpki / 5 /,  
&      jemiti / 6 /,  
&      jflush / 7 /,  
&      kcyl / 2 /,  
&      ksph / 1 /,  
&      rmaxi / 32767.0 /,  
&      r2to14 / 16384.0 /,  
&      smin / 0.1 /,  
&      two15 / 4194304.0 /
```

```
c====> / PARAM /
```

```
      data prmlen / 200 /  
      end
```

**subroutine dicomp**

**c:::** This routine flushes the current contents of the Dicomed output  
**c:::** buffer to the file 'fort.8' in the current directory.

**c:::**

**include 'dico.cm'**

**if ( bptr .ne. 1 ) then**

**limit = bptr - 1**

**write( 8 ) ( buff(i), i = 1, limit)**

**bptr = 1**

**endif**

**return**

**end**

**subroutine dicowd( iword )**

**c:::** This routine adds another 16-bit word to the Dicomed output  
**c:::** buffer, first emptying the buffer if it is already full.

**c:::**

**c:::** Input :

**c:::**     **iword :**    32-bit integer to be added to buffer

**c:::**

**include 'dico.cm'**

**if ( bptr .gt. bufmax )    then**

**.write(8) buff**

**bptr = 1**

**end if**

**buff(bptr) = iword**

**bptr = bptr + 1**

**return**

**end**

**subroutine exit**

c::: This routine closes all files associated with the program before  
c::: terminating execution.  
c:::

include 'manif.cm'  
include 'pgmctl.cm'

c====> Close input files

close ( 4 )  
close ( 5 )  
close ( 9 )

c====> Close output files

write( 6, 900 )  
900 format ( ' End of job... ' )  
close ( 6 )  
if ( idvice .eq. ifrbuf ) then  
call Fclofb  
else  
close ( 8 )  
end if  
  
stop  
end

**subroutine frmhdr**

c::: This routine emits the Dicom control sequence for a frame header.

c:::

c::: N.B. This code was originally part of N. Max's mainline module dni.f

c:::

c::: Reference :

c::: Dicomed Graphic Film Recorder Model D48 -

c::: Operation & Programming Manual.

c:::

include 'dicntl.cm'

include 'manif.cm'

```
c          Reset
call output( jnoop, ires )
c          ECS - select raster mode
call output( jnoop, 16 )
c          ECS - colour exposure
call output( jnoop, 18 )
c          ICS with options
call output( jnoop, lcomp )
c          ECS - colour translation
call output( jnoop, 22 )
c          PES - set horizontal spacing
call output( jnoop, lpes1 )
c          PES - set vertical spacing
call output( jnoop, lpes2 )
c          PES - set background exposure code &
c          no. of points/element in both directions
call output( jnoop, ipes3 )
return
end
```

### subroutine init

```

c::: This routine :
c::: 1. queries the user for program control parameters
c::: 2. computes, if necessary, pixel coordinate boundaries of the
c::: frame buffer visible window.
c::: 3. computes spacing constants dependent on the output device
c::: resolution.
c:::
c::: N.B. This code was originally part of N. Max's mainline module dni.f.
c:::

```

```

include 'debug.cm'
include 'dicntl.cm'
include 'fbparm.cm'
include 'ispace.cm'
include 'manif.cm'
include 'pgmctl.cm'

```

#### Integer resol

```

c          Dicomed boundaries
c      data minxd, minyd / 1, 1 /,
&      maxx, maxy / 1024, 1024 /,
c          Ikonas boundaries
&      minxfb, minyfb / 0, 0 /,
&      maxxfb, maxyfb / 511, 511 /

c====> Obtain run parameters from user

125      write(6, 905)
         read(5, *) idvice
         if ( ( idvice .ne. ldico ) .and. ( idvice .ne. lfrbuf ) ) go to 125

         if ( idvice .eq. lfrbuf ) then

135          write(6, 950)
             read(5, *) shdctl
             if ( ( shdctl .ne. loff ) .and. ( shdctl .ne. lon ) ) go to 135

150          write(6, 915)
             read(5, *) ifbsim
             if ( ( ifbsim .ne. loff ) .and. ( ifbsim .ne. lon ) ) go to 150

175          write(6, 920)
             read(5, *) ifbdbg
             if ( ( ifbdbg .ne. loff ) .and. ( ifbdbg .ne. lon ) ) go to 175

         call Finlfb( loff, ifbsim )

c=====> Compute a visible frame buffer window centred about the
c=====> specified raster position of the Dicomed screen

```

```

200      write(6, 925)
      read(5, *) indic
      if ( indic .eq. loff ) then
          midxd = ( maxxd - minxd ) / 2
          midyd = ( maxyd - minyd ) / 2
      else
210          write(6, 930) minxd, maxxd
          read(5, *) midxd, midyd
          if ( ( midxd .lt. minxd ) .or. ( midxd .gt. maxxd ) .or.
&              ( midyd .lt. minyd ) .or. ( midyd .gt. maxyd ) )
&              go to 210
      endif

      midxfb = ( maxxfb - minxfb ) / 2
      minx = max0( minxd, midxd - midxfb )
      maxx = min0( maxxd, midxd + midxfb )
      if ( mod( maxxfb, 2 ) .ne. 0 ) maxx = maxx + 1

      midyfb = ( maxyfb - minyfb ) / 2
      miny = max0( minyd, midyd - midyfb )
      maxy = min0( maxyd, midyd + midyfb )
      if ( mod( maxyfb, 2 ) .ne. 0 ) maxy = maxy + 1

c====>      Initialize x, y coordinates of range of frame buffer pixels
c====>      calculated by program.
      acminx = maxyd
      acmaxx = minxd
      acminy = maxyd
      acmaxy = minyd

c====>      Initialize x, y coordinates of range of frame buffer pixels
c====>      actually plotted.
      minfbx = maxxfb
      maxfbx = minxfb
      minfby = maxyfb
      maxfby = minyfb

      if ( lfbdbg .eq. ion )
&          write(7, 940) midxd, midyd, minx, maxx, miny, maxy
      resol = 1024
      else
c====>      Idvice = Dicomed
100          write(6, 900) `
      read(5, 800) iwait, lfda, idbg
      if ( ( ( iwait .ne. loff ) .and. ( iwait .ne. lon ) ) .or.
&          ( ( lfda .ne. loff ) .and. ( lfda .ne. lon ) ) .or.
&          ( ( idbg .ne. loff ) .and. ( idbg .ne. lon ) ) ) go to 100
      open(8, file = 'fort.8', form = 'unformatted' )

300          write(6, 910)
      read(5, *) resol
      if ( ( resol .ne. 1024 ) .and. ( resol .ne. 2048 ) .and.
&          ( resol .ne. 4096 ) ) go to 300
&
      endif

```

```

c====> Calculate constants derived from resolution

c====> / ISPACE /
    lsp = 4096 / resol
    lsp2 = lsp / 2
    jsp = 8 * lsp
    jsp2 = 8 * lsp2
    fjsp = float( jsp )
    fjsp2 = float( jsp2 )
    tjsp = 2. * fjsp
    tjsq = 2. * fjsp * fjsp

c====> / DICNTL /
    lpes1 = lpes + 512 * lsp + 8
    lpes2 = 512 * lsp + 8
    lpes3 = lsp * 17

    return

c====> Input formats

    800    format( 3i1 )

c====> Output formats

    900    format( /' Wait between frames, double film advance, debug trace:',
&         /' (0:no, 1:yes) 3i1 format: ' )
    905    format( /' Target output device:',
&         /' (1:Dicomed, 2:Ikona) free format: ' )
    910    format( /' Resolution : 1024, 2048, 4096 : free format: ' )
    915    format( /' Type of frame buffer:',
&         /' (0:actual, 1:simulated) free format: ' )
    920    format( /' Frame buffer debug output desired?',
&         /' (0:no, 1:yes) free format: ' )
    925    format( /' By default, centre portion of ATOMLLL data will be displayed ',
&         ' on frame buffer.', /' Do you wish to change area displayed?',
&         /' (0:no, 1:yes) free format: ' )
    930    format( /' x, y coordinates (in Dicomed raster units) of centre of area ',
&         'to be displayed : free format:',
&         /' (Must be in range ', 17, ' to ', 17 ' )' )
    940    format( /' Init : midxd, midyd = ', 2i8, ' minx, maxx = ', 2i8,
&         ' miny, maxy = ', 2i8 )
    950    format( /' Compute highlights?', /' (0:no, 1:yes) free format: ' )

    end

```



**integer function Itorgb( inten, icolor )**

c::: This function converts an 8-bit integer intensity for 1 of 7 colours  
 c::: (R, G, Y, B, M, C, W) to its equivalent 24-bit RGB code in a format  
 c::: compatible with Ikonas simulator I/O routines.  
 c:::  
 c::: Input :  
 c:::       inten :     integer in the range [0, 255]  
 c:::               (out of range values will be changed to the appropriate  
 c:::               boundary values)  
 c:::  
 c::: Output :  
 c:::       32-bit integer word containing the 4 bytes  
 c:::           O B G R  
 c:::  
 c::: Statistics are gathered on the maximum & minimum intensity requested for  
 c::: each colour.  
 c:::

```

include 'coltbl.cm'
include 'debug.cm'
include 'fbparm.cm'
include 'manif.cm'

data lsh8 / 256 /,
&      lsh16 / 65536 /

inten = max0( inten, 0 )
inten = min0( inten, 255 )

if ( ( icolor .lt. 1 ) .or. ( icolor .gt. 7 ) ) then
  write(6, 900) icolor
  call exit
else
  go to( 20, 50, 80, 110, 140, 170, 200 ), icolor
endif

c      red:
20     Itorgb = Inten
       go to 300

c      green:
50     Itorgb = Inten * lsh8
       go to 300

c      yellow = red + green:
80     Itorgb = inten + inten * lsh8
       go to 300

c      blue:
110    Itorgb = inten * lsh16
       go to 300

```

```
c      magenta = red + blue:
140    itorgb = Inten + Inten * lsh16
      go to 300

c      cyan = green + blue:
170    itorgb = Inten * lsh8 + Inten * lsh16
      go to 300

c      white = red + green + blue:
200    itorgb = Inten + Inten * lsh8 + Inten * lsh16
```

```
300 minrgb(icolor) = min0( Inten, minrgb(icolor) )
      maxrgb(icolor) = max0( Inten, maxrgb(icolor) )
```

```
c====> Output formats
```

```
900    format( 'itorgb : icolor = ', i5, ' out of range for computed go to' )
```

```
      return
      end
```

```

subroutine newcol( ncol, a, bp )

c::: This routine is called to change the current colour & update the
c::: associated intensity minimum & range values.
c::: For Dicomed output, the appropriate colour filter is selected.
c:::
c::: Input :
c:::   ncol :   new colour
c:::
c::: Output :
c:::   a :   scaled-up minimum intensity for this colour
c:::
c:::   bp :   scaled-up range of intensity for this colour
c:::

```

```

      include 'coltbl.cm'
      include 'manif.cm'

      nowcol = ncol
      bp = bpp( ncol, ihilit )
      a = app( ncol, ihilit ) * 255 * two15
      if ( ihilit .eq. 2 ) then
c          select neutral filter
          iadd = 15
      else
c          ncol => which colour filter to select
c          Reference : Table 3-1, p. 3-2, Dicomed Manual
          iadd = 8 + ncol
      end if
      call output( jnoop, iadd )
      return
      end

```

**subroutine output( icntl, iword )**

**c:::** This routine channels output according to the user-selected display  
**c:::** device. It replaces all calls to dicowd & dicemp in the original version;  
**c:::** hence it is assumed that such calls are made in the correct order  
**c:::** for controlling the Dicomed.

**c:::**

**c:::** Input :

**c:::**     **icntl :**     Describes the interpretation of the parameter iword.

**c:::**             It can be 1 of the following, defined in / MANIF / :

**c:::**             jnoop

**c:::**             jsavex

**c:::**             jsavey

**c:::**             jpacki

**c:::**             junpki

**c:::**             jemiti

**c:::**             jflush

**c:::**

**c:::**     **iword :**     32-bit integer destined for output device

**c:::**

**c:::** When the Ikonas frame buffer has been selected as the output device :

**c:::**

**c:::**     1. x, y coordinates must be rescaled down from the range (1, 4096)  
**c:::**         to (1,1024), as the Dicomed permits illumination of  
**c:::**         n by m rectangular 'meta-pixels' when used at less than  
**c:::**         its maximum resolution of 4096.

**c:::**

**c:::**     2. Only pixels within the frame buffer visible window will be  
**c:::**         displayed.

**c:::**

**c:::**     3. Pixels are processed along horizontal scanlines.

**c:::**

**c:::**     4. When highlighting, white is added individually to  
**c:::**         each of the 3 colour components (R, G, B) previously stored  
**c:::**         in that pixel, to avoid overflow of any colour component into  
**c:::**         adjacent one(s).

**c:::**

include 'coltbl.cm'

include 'debug.cm'

include 'fbparm.cm'

include 'lspace.cm'

include 'manif.cm'

include 'pgmctl.cm'

logical vislxb,  
 & visiby

integer and,  
 & ecmask,  
 & itorgb

```

dimension mask (3),
&      ishift (3)

data ecmask / z'00000fff' /,
&      mask / z'00ff0000', z'0000ff00', z'000000ff' /,
&      ishift / 65536, 256, 1 /,
&      iwhite / 7 /

visibx( x ) = ( x .ge. minx ) .and. ( x .le. maxx )
visiby( y ) = ( y .ge. miny ) .and. ( y .le. maxy )

if ( ( icntl .lt. 1 ) .or. ( icntl .gt. 7 ) ) then
    write(6, 910) icntl
    call exit
else
    go to( 20, 40, 60, 80, 100, 120, 240 ), icntl
endif

c      jnoop:
20      go to( 25, 30 ), idvice

25      call dicowd( iword )

30      return

c      jsavex: starting x raster coordinate
40      go to( 25, 50 ), idvice

50      nextx = iword / jsp
        return

c      jsavey: y raster coordinate
60      go to( 25, 70 ), idvice

70      nexty = iword / jsp
        return

c      jpacki: output intensities packed 2/word in rightmost 16 bits
80      go to( 25, 90 ), idvice

90      istart = 2
        go to 112

c      junpki: 1 output intensity/word
100     go to( 25, 110 ), idvice

110     istart = 3
112     lend = 3
        lcount = and( iword, ecmask )
        return

c      jemlti: compute & emit frame buffer output intensity

```

```

c          Input : 8-bit value of (cos(theta) ** 2) used
c                to index colour lookup tables.
c          Output: 24-bit RGB intensity (shaded/unshaded)
c                corresponding to input index.

120      acminy = min0( acminy, nexty )
          acminx = min0( acminx, nextx )
          acmaxy = max0( acmaxy, nexty )
          acmaxx = max0( acmaxx, nextx )
          go to( 25, 130 ), ldvice

130      if ( visiby( nexty ) ) then
          ly = nexty - miny
          do 200 i = istart, lend
c          Plot only visible raster positions,
c          omitting possible extra intensity
c          emitted by quad.

          if ( visibx( nextx ) .and.
&          ( icount .ge. 0 ) ) then

&          icos2 = and( lword, mask(i) ) /
&                ishift(i)
          ix = nextx - minx
          minfby = min0( minfby, ly )
          maxfby = max0( maxfby, ly )
          minfbx = min0( minfbx, ix )
          maxfbx = max0( maxfbx, ix )

          intenc = ltorgb( fthet2(icos2+1),
&                nowcol )
          intenh = ltorgb( gthet2(icos2+1),
&                lwhite )
          intout = 0
          do 175 j = 1, 3
&          lbyte = and(intenc,mask(j))
&                / ishift(j) +
&                and(intenh,mask(j))
&                / ishift(j)
          lbyte = min0( lbyte, 255 )
&          intout = intout +
&                lbyte * ishift(j)
175      continue
          call Fputpx( ix, ly, intout )
          endif
          nextx = nextx + 1
          lcount = lcount - 1

200      continue
          endif
          return

c          jflush: flush Dicomed output buffer
c          240 go to( 245, 250 ), ldvice

```

```
245      call dicemp
```

```
250      return
```

```
c====> Output formats
```

```
910     format( 'output : lcntl = ', l5, ' out of range for computed go to' )
```

```
      end
```

```

subroutine quad( k1, k2, k3, nw )

c:::: This routine represents an efficient calculation of  $\cos^2$  (theta)
c:::: along a vertical scanline, as described in
c:::: "ATOMLLL - A Three-D Opaque Molecule System"
c:::: (Lawrence Livermore Laboratory Version),
c:::: N. L. Max, Jan. 1979 (UCRL-52645)
c::::
c:::: Input :
c:::: k1 : Initial intensity for shading
c:::: k2 : First difference in intensity
c:::: k3 : Second difference in intensity
c:::: (constant since stepsize = 1)
c:::: nw : Number of pixels to plot
c::::
c:::: The routine always produces an even number of points. The extra
c:::: byte will be ignored by the Dicomed because the preceding count
c:::: word was correct.
c::::
c:::: 2 8-bit exposure codes are packed, right-justified, into a 32-bit word.
c::::

include 'manlf.cm'

data idenom / 4194304 /
c====> 2 ** 22 ( ==> shift 22 )

limit = (nw + 1) / 2
do 10 i = 1, limit
  iexp1 = k1 / idenom
  k1 = k1 + k2
  k2 = k2 + k3
  iexp2 = k1 / idenom
  k1 = k1 + k2
  k2 = k2 + k3
  call output( jemiti, 256 * iexp1 + iexp2 )
10 continue
return
end

```



**subroutine read ( n, len, ier )**

**c:::** This routine reads a block of data produced by the ATOMLLL program into  
**c:::** the 11 consecutive arrays of / PARAM /.

**c:::**

**c:::** Input :

**c:::**     **n :** Starting address of the 1st of the 11 arrays in / PARAM /

**c:::**

**c:::**     **len :** no. of words of input data to read

**c:::**

**c:::**     **ier :** error indicator :

**c:::**             < 0 :end-of-file

**c:::**             = 0 :all is well

**c:::**             > 0 :error occurred during read

**c:::**

**include 'manif.cm'**

**integer n(len)**

**open( 9, file = 'fort.9' )**

**rewind( 9 )**

**read( 9, 100, iostat = ier ) n**

**100   format( 10I7 )**

**300   return**

**end**

**subroutine setabl**

c::: When the output device is the Dicomed film recorder :  
c::: This routine computes & emits Dicomed colour translation tables  
c::: which compensate for nonlinearities on the output device & film.

c::: It uses the file 'fort.4', containing a table of measured film  
c::: densities stored as -log base 10(intensity)  
c::: i.e., the fraction of light transmitted by the film.  
c::: These densities decrease as output intensity level increases.  
c::: The format of this file is as follows:  
c::: 32 rows representing intensity levels, each containing:  
c::: film densities for all 7 colour filters at this level  
c::: a single value to trigger debug output (if > 8000)

c::: A table of 256 exposure codes per colour is computed as follows:  
c::: 1. measured intensity I is obtained from corresponding measured  
c::: density :  
c:::  $I = 10^{**} (-den / 100)$   
c::: 2. Desired output intensities are interpolated from measured  
c::: intensities, using the equations  
c:::  $F(\theta) = .1 + .9^{**} ((\cos(\theta))^{**} 2)^{**}.61$   
c::: for colour  
c:::  $G(\theta) = .8^{**} F + .39^{**} (.8^{**} ((\cos(\theta))^{**} 2)^{**} 14 +$   
c:::  $.2^{**} ((\cos(\theta))^{**} 2)^{**} (14/4) )$   
c::: for highlights,  
c::: suitably scaled to produce values within the range of measured  
c::: intensities obtained from the input film densities.  
c::: These equations are approximately equivalent to Lambert's Law  
c::: for the representation of diffuse and specular reflection:  
c::: i.e.,  $f(\theta) = A + D \cos(\theta)$   
c:::  $g(\theta) = C (\cos(\theta))^{**} n$ , n large.  
c::: except:  
c::: 1. they are in terms of  $\cos^{**}2$  rather than  $\cos$   
c::: 2. G has extra terms to eliminate Mach band effects

c::: The 8 sub-tables are calculated & emitted in the order  
c::: black & white, red, green, yellow, blue, magenta, cyan, neutral.  
c::: [Black & white calculations use film densities for neutral (white).]  
c:::

c::: The Dicomed saves these lookup tables; each emitted illumination  
c::: level from the quadratic differencing computation is actually a  
c:::  $\cos^{**}2$  treated as an index into the appropriate colour lookup table.  
c::: Then, according to options previously selected, the film is exposed  
c::: proportional to -log(base 10) of the indexed table value.  
c:::

c::: When the output device is the Ikonas:  
c::: The task is simplified, as the file of film densities can be  
c::: ignored. It is necessary only to compute & save the tables for F & G,  
c::: scaled to the range 0-255 (assuming we wish to allow the full range  
c::: of intensities for each colour).  
c::: G = 0 if no shading is desired.  
c:::

c::: N.B. "Do nothing" code from N. Max's original program has been

**C::::** commented out, and the input parameter removed.  
**C::::**

```

      include 'coltbl.cm'
      include 'debug.cm'
      include 'fbparm.cm'
      include 'manlf.cm'
      include 'pgmctl.cm'

      dimension iden(32, 7),
&          zi(32),
&          jcode(32)

      c      dimension ltbl(2048)
&          equivalence ( zimax, zi(32) ),
&          ( zimin, zi(1) )

      data b1 / .10 /,
&          c1 / .90 /,
&          e1 / .61 /,
&          b2 / .08 /,
&          c2 / .39 /,
&          e2 / 14. /,
&          fbmin / 0.0 /,
&          fbmax / 255.0 /

      data ifs1 / z'2000' /,
&          ifs2 / z'd000' /

      c      ifs1 FES (Deferred status; p. 3.8 in Dicomed manual)
      c      ifs2 F6 - load translation table & ES - read from input
      c      until element count satisfied. (p. 3-14, 3-34)

      c      Options selected:
      c          8-bit exposure codes
      c          exposure on film increases as input exposure code
      c          increases.
      c          " " " proportional to inverse log of exposure
      c          code.

      c====> data lsub / 1 /

      c====> ** Modification **
      c====>          Film densities read for all data types - atoms, bonds, or
      c====>          a mixture of both; previously not done for bonds.
      c====> if ( kind .eq. ksph ) then
&          if ( ldvice .eq. ldco ) then
&              open (4, file='fort.4')
&              rewind (4)
&              do 10 i = 1, 32

```

```

        read (4, 800) jcode(i), ( iden(i, j), j = 1, 7 )
10      continue
        read (4, 800) idum
c====>   else
c====>     if ( kind .ne. kcyl ) then
c====>       write (6, 900)
c====>       return
c====>     endif
c====>   endif

        call output( jnoop, ifs1 )

c====>           2048 (= 8 * 256) is element count
c====>           i.e., the number of entries in the table.
        call output( jnoop, ifs2 + 2048 )

        do 500 m = 1, 8
          if ( m .eq. 1 ) then
c====>           Use neutral filter values
                j = 7
          else
                j = m - 1
          endif

c====>           Obtain measured intensities from measured film densities
          do 30 i = 1, 32
            zi(i) = 10.0 ** ( float( - iden(i, j) ) / 100. )
            30      continue
            zs = zimax - zimin
            do 400 k = 0, 255
              cos2th = float( k ) / 255.

c====>           Compute des, the desired intensity

c====>           if ( kind .eq. kcyl ) then
c====>             des = zimin + cos2th * zs
c====>           else
c====>             kind .eq. ksph
              d1 = b1 + c1 * cos2th ** e1
              if ( m .eq. 1 ) then

c====>                 Compute black & white table values
                  d2 = b2 * d1 + c2 *
&                   (.8 * cos2th ** e2 +
&                   .2 * cos2th ** ( e2 / 4. ))
                  des = zimin + zs * d2
              else
                  des = zimin + zs * d1
c====>                 Compute colour values
              endif
c            endif
c          endif

c====>           Now find an intensity .gt. des
          do 80 i = 2, 32

```

```

        wl = zi(l)
        if ( wl - des ) 80, 70, 90
70          mnt = jcode(l) * 16
            go to 110
80          continue

c====>          Interpolate exposure index from nearest measured intensity
90          fjc = float( jcode(l) )
            fjc1 = float( jcode(l - 1) )
            wl1 = zi(l - 1)
            if ( k .eq. 0 ) then
                mnt = 0
            else
c====>          if ( ( m .eq. 1 ) .and. ( kind .eq. kcy1 ) ) then
c====>                mnt = 4 * k
c====>            else
                mnt = ifix( ( ( des - wl1 ) * ( fjc - fjc1 ) /
&                ( wl - wl1 ) + fjc1 ) * 16. + .5 )
c====>            endif
            endif
c====>          mnt is exposure code
110         call output( jnoop, mnt )
c====>         itbl(isub) = mnt
c====>         isub = isub + 1
            if ( idbg .eq. 6 )
&             write( 7, * ) mnt
            if ( idum .ge. 8000 + m )
&             write( 7, 910 ) m, k, l, mnt, des
400         continue
500         continue
            call output( jflush, jflush )
c====>         write(1, 999) itbl
            else
c====>          Output device is frame buffer
            do 600 k = 1, 256
                cos2th = float(k - 1) / 255.
                fbscal = fbmax - fbmin
                d1 = b1 + c1 * cos2th ** e1
                fthet2(k) = ifix( d1 * fbscal + fbmin + 0.5 )
                if ( shdctl .eq. lon ) then
                    gthet2(k) = ifix( ( b2 * d1 + c2 *
&                    (0.8 * cos2th ** e2 + 0.2 * cos2th **
&                    (e2/4.)) ) * fbscal + fbmin + 0.5 )
                else
                    gthet2(k) = 0
                end if
            continue
600         if ( ifbdbg .eq. lon ) write(7, 999) fthet2, gthet2
            end if
            return

c====>          Input formats

800         format ( 8i5 )

```

c====> Output formats

```
900 format ( ' Error in setabl : input parameter is not sphere',  
&         ' or cylinder ' )
```

```
910 format ( 4i6, f11.4 )
```

```
999 format( 25( 10i8/ ), 6i8, /// )  
end
```

**subroutine shdah( nw, x, y )**

**c:::** This routine shades above entire highlight line with quadratics on  
**c:::** horizontal lines.  
**c:::**  
**c:::** Input :  
**c:::**     nw :    number of pixels to shade in y direction  
**c:::**     x, y :    starting plot position  
**c:::**  
**c:::** The calculated intensity at each y is a function of the top & middle  
**c:::** x coordinates of the region to be shaded.  
**c:::**  
**c:::** N.B. This routine was originally part of the subroutine trapez.  
**c:::**

include 'bond.cm'  
include 'debug.cm'  
include 'dicntl.cm'  
include 'lspace.cm'  
include 'manif.cm'  
include 'qdiff.cm'

```

c      FES - single element plotting of nw elements
c      computed in following loop
      call output( jnoop, ifep1 )
      call output( junpki, ifep2 + nw )
      if ( idbg .eq. lon ) write(7, 1980)

      do 380 j = 1, nw

          if ( ( jt .lt. 1 ) .or. ( jt .gt. 3 ) ) then
              write(6, 900) jt
              call exit
          else
c              Case on top line type:
              go to (320, 320, 330), jt
          endif

c              Convex down / up:
320          xt = (y - rit) / st
              go to 340

c              Straight line:
330          xt = rit + y * st

340          if ( ( jm .lt. 1 ) .or. ( jm .gt. 3 ) ) then
              write(6, 910) jm
              call exit
          else
c              Case on middle line type:
              go to (350, 360, 360), jm
          endif

```

```

c          Convex down:
350      xm = (y - rim) / sm
          go to 370

c          Convex up / straight line:
360      xm = rim + y * sm

370      id = fs * (1. - ((x - xm)/(xt - xm)) ** 2) + ad
          id = min0( 255, max0( 0, id ) )

c====>          Plot 1 element in horizontal direction at exposure
c====>          level id
          call output( jemiti, ifp + id )
          if ( ldbg .eq. 2 ) write(7, 1990 )
          y = y + fjsp
380      continue
          return

c====>  Output formats

900      format( 'shdah: jt = ', i5, ' out of range for computed go to' )
910      format( 'shdah: jm = ', i5, ' out of range for computed go to' )
1980     format( 'trapez #9' )
1990     format( 'trapez #10' )

          end

```



```
subroutine shdbh( nw, x, y )
```

```
c::: This routine shades below entire highlight line with quadratics on
c::: horizontal lines.
c:::
c::: Input :
c:::     nw : no. of pixels to shade in y direction
c:::     x, y : starting plot position
c:::
c::: The calculated intensity for each y is a function of the corresponding
c::: middle & bottom x coordinates of the region to be shaded.
c:::
c::: N.B. This routine was originally part of the subroutine trapez.
c:::
```

```
include 'bond.cm'
include 'debug.cm'
include 'dicntl.cm'
include 'ispace.cm'
include 'manif.cm'
include 'qdiff.cm'
```

```
c      FES - single element plotting of nw elements computed
c      in following loop
      call output( jnoop, lfep1 )
      call output( junpki, lfep2 + nw )
      if ( ldbg .eq. 2 ) write(7, 1950)

      do 780 j = 1, nw
        if ( ( jb .lt. 1 ) .or. ( jb .gt. 3 ) ) then
          write(6, 900) jb
          call exit
        else
c      Case on bottom line type:
          go to (720, 730, 730), jb
        endif

c      Convex down:
720      xb = (y - rib) / sb
          go to 740

c      Convex up / straight line:
730      xb = rib + y * sb

740      if ( ( jm .lt. 1 ) .or. ( jm .gt. 1 ) ) then
          write(6, 910) jb
          call exit
        else
c      Case on middle line type:
          go to (750, 760, 760), jm
        endif

c      Convex down:
750      xm = (y - rim) / sm
```

```

                go to 770

c              Convex up / straight line:
760              xm = rim + y * sm

770              Id = fs * (1. - ((x - xm) / (xb - xm)) ** 2) + ad
                id = min0( 255, max0( 0, id ) )

c              Plot 1 element in horizontal direction at exposure
c              level id
                call output( jemiti, lfp + id )
                if ( ldbg .eq. 2 ) write(7, 1960 )
                y = y + fjsp
780              continue

                return

c====> Output formats

900              format( 'shdbh: jb = ', i5, ' out of range for computed go to' )
910              format( 'shdbh: jm = ', i5, ' out of range for computed go to' )
1950             format( 'trapez #6' )
1960             format( 'trapez #7' )

                end

```

```
subroutine shdcyl( iyb, iyt, x )
```

```

c:::: This routine shades trapezoids belonging to cylinders (bonds).
c:::: It is more complex than shading spheres, since cylinders become
c:::: cones after a perspective projection is applied, and the
c:::: directions of the projected edges may lie in different quadrants.
c::::
c:::: Input :
c::::   iyb, iyt : scaled-up bottom & top y coordinates of cylinder
c::::   x       : current x coordinate
c::::
c:::: The top, middle, & bottom y coordinates of the bisecting (highlighting)
c:::: line are computed. Then the trapezoid is shaded in 2 phases :
c:::: above & below its bisecting line (unless the bisecting line is strictly
c:::: vertical, in which case the entire trapezoid is shaded with a single
c:::: quadratic.
c::::
c:::: N.B. This routine was originally part of the subroutine trapez.
c::::
      include 'bond.cm'
      include 'debug.cm'
      include 'dicntl.cm'
      include 'ispace.cm'
      include 'manif.cm'
      include 'qdiff.cm'

      iyb = iyb - i2to14
      iyt = iyt - i2to14
      y = iyb + jsp2
      if( idbg .eq. 2 ) write(7, 595) jt, jm, jb

      if ( ( jt .lt. 1 ) .or. ( jt .gt. 3 ) ) then
          write(6, 900) jt
          call exit
      else
c          Case on top line type:
          go to (510, 520, 530), jt
      endif

c          Convex down:
      510      yt = st * x + rit
          go to 530

c          Convex up:
      520      yt = (x - rit) / st

      530      if ( ( jb .lt. 1 ) .or. ( jb .gt. 3 ) ) then
          write(6, 910) jb
          call exit
      else
c          Case on bottom line type:
          go to (540, 550, 560), jb
      endif

```

```

c      Convex down:
540      yb = sb * x + rib
      go to 560

c      Convex up:
550      yb = (x - rib) / sb

560      If ( ( jm .lt. 1 ) .or. ( jm .gt. 3 ) ) then
      write(6, 920) jm
      call exit
      else
c      Case on middle line type:
      go to (570, 580, 620), jm
      endif

c      Convex down:
570      ym = sm * x + rim
      go to 590

c      Convex up:
580      ym = (x - rim) / sm
590      if (ym .gt. rmaxi) ym = rmaxi
      If (ym .lt. -rmaxi) ym = -rmaxi
      If( idbg .eq. 2 ) write(7, 595) iyb, iyt, yb, ym, yt
      lym = int( (ym - fjsp2) / fjsp ) * jsp
      if (iyb .gt. iym) then
          nw = ( iyt - iyb ) / jsp + 1
          go to 800

      else if (iyt .lt. iym) then
          nw = ( iyt - iyb ) / jsp + 1
      else
          nw = (iym - iyb) / jsp + 1
      end if
      If (nw .le. 0) return
      if (markb .eq. ihoriz) go to 710

c====> Shade below highlight line with quadratics on vertical lines

      call shvlin( y, ym, yb, nw )
      y = y + nw * jsp
      go to 790

c====> Shade whole vertical segment with quadratics on horizontal lines
c====> This loop is executed when the bisecting line of the trapezoid is
c====> strictly vertical; a single quadratic can then be used to shade
c====> the entire region.

c      Straight line:
620      nw = (iyt - iyb) / jsp + 1
      If (nw .le. 0) return

c      FES & single element plotting of nw elements
c      computed in following loop

```

```

call output( jnoop, ifep1 )
call output( junpki, ifep2 + nw )
if ( idbg .eq. 2 ) write(7, 1930) ifep1, ifep2 + nw

do 700 j = 1, nw
  if ( idbg .eq. 2 )
&      write( 7, 1932 ) y, rit, st, rim, sm, rib, sb
      if ( ( jm .lt. 1 ) .or. ( jm .gt. 3 ) ) then
          write(6, 920) jm
          call exit
      else
c          Case on middle line type:
          go to (630, 640, 640), jm
      endif

c          Convex down:
630      xm = (y - rim) / sm
          go to 650

c          Convex up / straight line:
640      xm = rim + y * sm

650      if ( ( jt .lt. 1 ) .or. ( jt .gt. 3 ) ) then
          write(6, 900) jt
          call exit
      else
c          Case on top line type:
          go to (655, 660, 660), jt
      endif

c          Convex down:
655      xt = (y - rit) / st
          go to 665

c          Convex up / straight line:
660      xt = rit + y * st

665      if ( ( jb .lt. 1 ) .or. ( jb .gt. 3 ) ) then
          write(6, 910) jb
          call exit
      else
c          Case on bottom line type:
          go to (670, 675, 675), jb
      endif

c          Convex down:
670      xb = (y - rib) / sb
          go to 680

c          Convex up / straight line:
675      xb = rib + y * sb

680      if ((x - xm) * (xb - xm)) 685, 690, 690

```

```

c          < 0:
685          id = fs * (1. - ((x - xm) / (xt - xm)) ** 2) + ad
              go to 695

c          >= 0:
690          id = fs * (1. - ((x - xm) / (xb - xm)) ** 2) + ad

c          Plot 1 element in horizontal direction at exposure
c          level ad
695          call output( jemiti, ifp + id )
              if ( idbg .eq. 2 ) write(7, 1940) ifp, id, ifp + id
              y = y + fjsp
              if ( idbg .eq. 2 )
&          write( 7, 1942 ) x, xt, xm, xb, y, fs, id
700          continue
              if( idbg .eq. 2 ) write(7, 696) id, xb, xm, xt, x, fs, ad
              return

```

c====> Shade below highlight line with quadratics on horizontal lines

```

710    call shdbh( nw, x, y )
790    if ( lyt .le. iym ) return
      nw = (lyt - iym) / jsp

```

```

800    if (nw .le. 0) return
      if (markt .eq. ihoriz) then

```

c====> Shade above highlight line with quadratics on horizontal lines

```

call shdah( nw, x, y )
else

```

c====> Shade above highlight line with quadratics on vertical lines

```

call shvlin( y, ym, yt, nw )
end if
return

```

c====> Output formats

```

595    format(3i7, 3f10.2)
1930    format( 'trapez #4', 2i10 )
1932    format ( ' y rit st rlm sm rib sb: ', 7(f10.2, 1x) )
1940    format( 'trapez #5', 3i10 )
1942    format( ' x xt xm xb y fs ld: ', 6(f10.2, 1x), i10 )
696    format(2h d, i7, 6f10.2)
900    format( ' shdcyl: jt = ', i5, ' out of range for computed go to' )
910    format( ' shdcyl: jb = ', i5, ' out of range for computed go to' )
920    format( ' shdcyl: jm = ', i5, ' out of range for computed go to' )

```

end

```
subroutine shdsph( nw, iyb, x )
```

```
c::: This routine shades a trapezoid within a sphere,
```

```
c::: using a quadratic differencing algorithm.
```

```
c:::
```

```
c::: Input :
```

```
c:::   nw  :   no. of pixels to shade in y direction
```

```
c:::   iyb :   y coordinate of bottom of trapezoid
```

```
c:::   x   :   x   "   "   "   "
```

```
c:::
```

```
c::: N.B. This routine was originally part of the subroutine trapez.
```

```
c:::
```

```
include 'debug.cm'
```

```
include 'dicntl.cm'
```

```
include 'ispace.cm'
```

```
include 'manif.cm'
```

```
include 'qdiff.cm'
```

```
include 'sphere.cm'
```

```
c====>          FES - initiate plot of nw elements
```

```
c====>          (elements are computed & emitted by quad)
```

```
call output( jnoop, ifes1 )
```

```
call output( jpacki, ifes2 + nw )
```

```
if ( idbg .eq. 3 ) write(7, 1910)
```

```
y = yc - (iyb + jsp2 - i2to14)
```

```
k1 = d - c * ((x - xc) ** 2 + y * y)
```

```
k2 = e * y - f
```

```
if ( idbg .eq. 2 ) write(7, 9876) nw, k1, k2, k3
```

```
call quad( k1, k2, k3, nw )
```

```
return
```

```
c==== Output formats
```

```
1910 format( ' trapez #2' )
```

```
9876 format( 6hparms , 4i10 )
```

```
end
```

**subroutine shvlin( y, ym, ybnd, nw )**

**c::::** This routine shades entire region above/below highlight line with  
**c::::** quadratics on vertical lines using the quadratic differencing algorithm.

**c::::**

**c::::** Input :

**c::::**     **y**    :    bottom y coordinate

**c::::**     **ym**   :    middle y

**c::::**     **ybnd**:   y boundary value (either top or bottom)

**c::::**     **nw**   :    no. of pixels to shade

**c::::**

**c::::** **N.B.** This routine generalizes 2 segments of nearly identical code

**c::::**     for the above & below cases in the original version of trapez.

**c::::**

**include 'dicntl.cm'**

**include 'ispace.cm'**

**include 'manif.cm'**

**include 'qdiff.cm'**

**yd = y - ym**

**ctrbnd = c / (ym - ybnd) \*\* 2**

**k1 = c - ctrbnd \* yd \* yd + a**

**k2 = -ctrbnd \* tjsp \* (yd + fjsp)**

**k3 = -ctrbnd \* tjsq**

**c====>**        Initiate plot of nw elements computed & emitted by quad

**call output( jnoop, ifes1 )**

**call output( jpacki, ifes2 + nw )**

**if ( ldbg .eq. 2 ) write(7, 1920)**

**call quad( k1, k2, k3, nw )**

**return**

**c====>**    Output formats

**1920   format( 'trapez #3' )**

**end**



**subroutine trapez(i,kind)**

c::: This routine controls the processing of all trapezoids,  
 c::: drawing them on a 4096 \* 4096 grid with element spacing isp.  
 c:::  
 c::: Input :  
 c:::     i :     Index in arrays of / PARAM / containing information  
 c:::             pertinent to the current trapezoid.  
 c:::  
 c:::     kind :     tells whether the trapezoid belongs to a sphere (atom)  
 c:::                or a cylinder (bond).  
 c:::  
 c::: 1. Radius & slope information for the top & bottom arcs of the trapezoid  
 c::: are computed.  
 c::: 2. The number of scan lines is obtained from the top & bottom y  
 c::: coordinates.  
 c::: 3. The cursor is moved to the starting position on the 1st scanline.  
 c::: 4. The appropriate routine is called to shade either a sphere or a  
 c::: cylinder.  
 c:::

```

include 'bond.cm'
include 'debug.cm'
include 'dicnt1.cm'
include 'ispace.cm'
include 'manif.cm'
include 'param.cm'
include 'qdiff.cm'
include 'radius.cm'
include 'sphere.cm'

```

```

ix1 = xl(i) + jsp2
ix3 = mod(ix1, jsp)
if (ix3 .lt. 0) ix3 = ix3 + jsp
ixl = ix1 - ix3
ix2 = xr(i) - jsp2
ix4 = mod(ix2, jsp)
if (ix4 .lt. 0) ix4 = ix4 + jsp
ixr = ix2 - ix4

```

c====>Debugging statements

```

if ( ldbg .eq. ion ) then
  write(7,9870) n(i), kt(i), kb(i), xl(i), xr(i), xct(i),
&      yct(i), rt(i), xcb(i), ycb(i), rb(i)
  write(7,9875) xl(i), xr(i), ix1, ix2, ix3, ix4, ixl, ixr
  write(7, 941) i, kind, it, lb
end if

if (ixr .lt. ixl) return

```

```

It = kt(i)
lb = kb(i)

If ( ( It .lt. 1 ) .or. ( It .gt. 4 ) ) then
    write(6, 9900) It
    call exit
else
c    Top arc:
    go to (900, 900, 910, 910), It
endif

c    Convex down / up:
900    srt = float( rt(i) ) ** 2
    go to 920

c    Straight line / special case:
910    slopt = float( xct(i) ) / rmaxi

920    If ( ( lb .lt. 1 ) .or. ( lb .gt. 4 ) ) then
        write(6, 9910) lb
        call exit
    else
c    Bottom arc:
        go to (930, 930, 940, 940), lb
    endif

c    Convex down / up:
930    srb = float( rb(i) ) ** 2
    go to 950

c    Straight line / special case:
940    slopb = float( xcb(i) ) / rmaxi

c====> This loop sweeps across the trapezoid horizontally,
c====> generating vertical scan lines.
950    do 2 ix = lxl, lxr, jsp
        If( ldbg .eq. lon .and. kind .eq. kcyl ) write(7,391) lxl, lxr, lx
        x = ix + jsp2

        If ( ( It .lt. 1 ) .or. ( It .gt. 4 ) ) then
            write(6, 9900) It
            call exit
        else
c            Find top of quadratically shaded segment
            go to (800, 810, 820, 830), It
        endif

c            Convex down:
800            ysq = srt - (x - xct(i)) ** 2
                yt = yct(i) + sqrt( abs(ysq) )
                go to 850

c            Convex up:

```

```

810      ysq = srt - ( x - xct(l) ) ** 2
      yt = yct(l) - sqrt( abs(ysq) )
      go to 850

c      Straight line:
820      yt = slopt * x + rt(i)
      go to 850

c      Special case:
830      yt = ( x - rt(l) ) / slopt

850      If ( ( lb .lt. 1 ) .or. ( lb .gt. 4 ) ) then
      write(6, 9910) lb
      call exit
      else
c      Find bottom of quadratically shaded segment
      go to (860, 870, 880, 890), lb
      endif

c      Convex down:
860      ysq = srb - ( x - xcb(i) ) ** 2
      yb = ycb(i) + sqrt( abs(ysq) )
      go to 430

c      Convex up:
870      ysq = srb - ( x - xcb(i) ) ** 2
      yb = ycb(i) - sqrt( abs(ysq) )
      go to 430

c      Straight line:
880      yb = slopb * x + rb(i)
      go to 430

c      Special case:
890      yb = ( x - rb(i) ) / slopb

430      yt = yt + r2to14
      yb = yb + r2to14
      lyt = int( ( yt + jsp2 ) / jsp ) * jsp - jsp
      lyb = int( ( yb + jsp2 ) / jsp ) * jsp
      nw = ( lyt - lyb ) / jsp + 1
      If ( ldbg .eq. lon ) write(7, 9874) yb, yt, lyb, lyt, ysq, nw
      If ( nw .le. 0 ) go to 2

c====>      Move CRT beam to following absolute position
      call output( jnoop, ipos )
      call output( jsavex, lyb )
      call output( jsavey, lx + l2to14 )

      If ( ldbg .eq. lon ) write(7, 1900) ipos, lyb, lx + l2to14

      If ( ( kind .lt. 1 ) .or. ( kind .gt. 2 ) ) then
      write(6, 9920) kind

```

```

        call exit
    else
        go to (440, 460), kind
    endif
c      Shade Sphere:
440      call shdsph( nw, lyb, x )
        go to 2

c      Shade Cylinder:
460      call shdcyl( lyb, lyt, x )
        2 continue

    return

c====>  Output formats

391      format(2h x, 3i6)
941      format(2h t, 5i6)
1900     format( ' trapez #1', 3i10 )
9870     format( ' n = ', i3 /
&        ' kt = ', i3 /
&        ' kb = ', i3 /
&        ' xl = ', i10      /
&        ' xr = ', i10      /
&        ' xct = ', i10     /
&        ' yct = ', i10     /
&        ' rt = ', i10     /
&        ' xcb = ', i10     /
&        ' ycb = ', i10     /
&        ' rb = ', i10     )
9874     format ( 'yb = ', 1pe10.4 /
&        'yt = ', 1pe10.4 /
&        'iyb = ', i10      /
&        'iyt = ', i10      /
&        'ysq = ', 1pe10.4 /
&        'nw = ', i4 )
9875     format( 2(1pe14.4, 1x), 6i10 )
9900     format( 'trapez: lt = ', i5, ' out of range for computed go to' )
9910     format( 'trapez: lb = ', i5, ' out of range for computed go to' )
9920     format( 'trapez: kind = ', i5, ' out of range for computed go to' )
    end

```

**FC\_interface.c**

```
#include "/u/phbreslin/sim/manif.h"
```

```
/*      Global Declarations for Ikonas Simulator Routines
```

```
        These force storage allocation, and, where applicable,  
        initialization.
```

```
*/
```

```
int  frame_buffer;  
char disc_fb;  
char *fb_path;  
int  current_scanline;  
char scanline_modified;  
long *scan_line;  
long WriteMask    =  0xFFFFFFFFL;
```

```
/*      Interface Routines Between FORTRAN 77 & C Language
      Ikonas Simulator I/O Software.
```

```
*/
      Calls : Init_fb In /u/phbreslin/sim/fb_io.c
```

```
Finfb_ ( create_new_file, dev_type )
```

```
/*      This routine initializes global variables required to open
      the appropriate frame buffer output pathname.
```

```
Input :
```

```
      create_new_file   :   0
      dev_type          :   0 ==> output on Ikonas
                        1 ==> output on file 'frame_buffer'
```

```
*/
```

```
long *create_new_file,
      *dev_type; {
```

```
      extern long *get_storage ();
      int kind;
```

```
      scan_line = get_storage( RESOLUTION );
      kind = (int) *dev_type;
      switch ( kind ) {
```

```
          case 0 : /* Actual Ikonas */
```

```
              fb_path = "/dev/ike";
              disc_fb = FALSE;
              break;
```

```
          case 1 : /* Simulated Ikonas*/
```

```
              fb_path = "frame_buffer";
              disc_fb = TRUE;
              break;
```

```
          default :
```

```
              printf( "Invalid argument = %d to Finfb\n", kind );
              break;
```

```
      };
      Init_fb ( (int) *create_new_file );
```

```
}
```

**Fputpx\_ ( x, y, pixel\_value )**

**/\* This routine stores information in the specified pixel of the current frame buffer pathname.**

**Input :**

**x, y :** destination pixel address; in range [0, 511]

**pixel\_value :** 24-bit contents to be stored.

**Format :**

**0 B G R            where B = blue**

**G = green**

**R = red**

**Calls :** put\_pixel in /u/phbresln/sim/fb\_lo.c

**The Ikonas routine put\_pixel is a function.**

**However, its returned value is ignored, since it is the rightmost 24 bits of pixel\_value.**

**\*/**

```

long *x,
    *y,
    *pixel_value; {

    long rtn_val;
    extern long put_pixel ();

    rtn_val = put_pixel( (int) *x, (int) *y, *pixel_value );
}

```

```
Fclofb_ () {
```

```
/* This routine closes the current Ikonas output pathname.  
*/
```

```
    close( frame_buffer );  
}
```



```
static long *  
get_storage( nwords )
```

```
/* This routine is used to allocate 1 scanline of storage,  
    permitting output buffering to the Ikonas.  
    Nothing is actually transmitted to the device (or file)  
    until the scanline currently being accessed changes.  
*/
```

```
int nwords;  
{  
    long *memory;  
  
    if( !(memory = (long *)calloc( nwords, sizeof(long) )) )  
        {  
            put_string( "\n** Out of memory **\n" );  
            exit();  
        }  
    return( memory );  
}
```

iksim\_lo/fb\_io.c

```
#include "externals.h"  
#include "manif.h"
```

```
long  
get_pixel( x, y )
```

```
int  x;  
int  y;  
{  
    long    high, low;  
  
    get_scanline( y );  
    high = scan_line[x] << 16;  
    low  = (scan_line[x] >> 16) & 0xFFFFL;  
    return( (high | low) & BITS_PER_PIXEL );  
}
```

```
long  
put_pixel( x, y, value )
```

```
int  x;  
int  y;  
long value;  
{  
    long    data, old_data;  
  
    get_scanline( y );  
    old_data = get_pixel( x, y );  
    scanline_modified = 1;  
    data = value & BITS_PER_PIXEL;  
    data = (old_data & ~WriteMask) | (data & WriteMask);  
    scan_line[x] = (data << 16) | ((data >> 16) & 0xFFFFL);  
    return( data );  
}
```

```

init_fb( old_file )

int  old_file;
{
    char c;
    register int  i;

    printf( "fb: %s\n", fb_path );
    if( disc_fb && !old_file )
    {
        if( (frame_buffer = creat( fb_path, 0x187 )) == -1 )
        {
            put_string( "Could not create frame buffer file.\n" );
            exit(1);
        }
/*
        creat opens for write only. We want read/write.
*/
        close( frame_buffer );
        frame_buffer = open( fb_path, 2 );
        lseek( frame_buffer, ONE_MEG_BYTES, 0 );
        c = 0;
        write( frame_buffer, &c, 1 );

        for( i=0; i < RESOLUTION; ++i )
            scan_line[i] = 0L;
        current_scanline = 0;
    }
    else
        if( (frame_buffer = open( fb_path, 2 )) == -1 )
        {
            put_string( "Could not open frame buffer!\n" );
            if( disc_fb )
            {
                put_string( "Attempting to create...\n" );
                init_fb(0);
            }
            else
                exit(1);
        }
        else
        {
            current_scanline = -1;
            scanline_modified = 0;
            get_scanline(0);
        }
}

```

**get\_scanline( y )**

```
register y;
{
    register nbytes;

    if( current_scanline == y ) return;
    if( scanline_modified )
        put_scanline();
    lseek( frame_buffer, y * (long)(RESOLUTION * sizeof(long)), 0 );
    nbytes = read( frame_buffer, scan_line, (RESOLUTION * sizeof(long)) );
    if( nbytes != (RESOLUTION * sizeof(long)) )
        printf( "frame_buffer read failed(%d)\n", nbytes );
    current_scanline = y;
}
```

**put\_scanline()**

```
{
    lseek( frame_buffer, current_scanline * (long)(RESOLUTION * sizeof(long)), 0 );
    write( frame_buffer, scan_line, RESOLUTION * sizeof(long) );
    scanline_modified = 0;
}
```