6 8 6

Pascal DNI
A Program for Shading Molecular Models

By

Harry Yuen

An essay
presented to the University of Waterloo
in partial fulfillment of the
requirements for the degree of
Masters of Mathematics
in
Computer Science

Waterloo, Ontario, 1980

ACKNOWLEDGEMENTS

I wish to express my gratitude to my supervisor, Professor Kellogg S. Booth, for suggesting the topic of this essay and for providing guidance and constructive criticism throughout its preparation.

I would also like to thank Alex White for his patience and assistance afforded me with work done using the Unix operating system.

Finally, I thank Martin Tuori of the Defense and Civilian Institute for Environmental Medicine at Downsview, Ontario for the use of the Dicomed D-48 color film recorder and photographic processing facilities.

PASCAL DNI
A PROGRAM FOR SHADING MOLECULAR MODELS

Abstract - DNI is the post-processor for a system which renders the visible parts of synthetic molecular models in realistic color. The ATOMLLL program produces input to DNI in the form of parts of atoms and bonds ready for color shading and highlights. This essay describes the function of DNI, the conversion of DNI from Fortran to Pascal, some of the problems encountered during the conversion and some suggestions for possible improvements to the DNI code.

1.0 INTRODUCTION

The ATOMS program, written by Ken Knowlton and Lorinda Cherry of Bell Telephone Laboratory, produces a list of the visible pieces of atoms (with or without joining bonds) from a description of the X, Y and Z coordinates of the spheres and the atom pairs joined by bonds. The Lawrence Livermore Laboratory version of this program (ATOMLLL) was modified by Steve Levine and Nelson Max. Max's version added the color shading and highlights to the pictures, implemented in the current DNI program. This essay describes DNI in detail and outlines the conversion effort from Fortran into Pascal. A brief discussion of the Lawrence Livermore system contrasted with the new system then follows.

Input to ATOMLLL consists of center coordinates for atoms, their radii, colors, and bond connections. The output is a binary description of the atoms and bonds after they have been decomposed into "trapezoids" which describe only the visible parts of the picture.

At Lawrence Livermore, DNI reads the binary tape on a Varian V-75 minicomputer and drives an attached Dicomed D-48 color film recorder to produce the molecular scenes. The new version of DNI at Waterloo currently produces an ASCII file of Dicomed codes which are then processed by a Unix filter (written in C) to produce 16-bit binary codes. These are fed offline to a Dicomed film recorder. This procedure is somewhat awkward. However, the primary consideration was portability instead of efficiency. DNI runs on Waterloo's Honeywell 66/60 and the only accessible Dicomed is connected to a PDP 11/34 running Unix; portability is a necessity.

It is expected that DNI will soon run on a PDP-11 to produce the 16-bit binary codes directly.

2.0 THE REVISION OF DNI

2.1 Objectives of the Revision

The main reason for restructuring DNI was to make it more understandable and thus easier to modify and maintain. Changes that were made to the code include

1) removal of GOTO statements and replacement with more disciplined constructs,

2) declaration of common blocks as record structures to show the specific lexical scope of different variables,

3) renaming of most variables to give them mnemonic significance,

4) splitting of large subprograms into smaller, more manageable procedures,

5) replacing "magic numbers" with Pascal defined constants to provide flexibility and increase understandability,

6) the addition of a large number of comments to explain what the code is doing, and

7) the introduction of two debugging levels which produce program trace information on an internally defined file.

2.2 Approach to the Revision

The conversion of DNI from Fortran to Pascal was done in an orderly fashion. Initially, a data dictionary was developed from the variables in the Fortran version of the program. These were renamed where necessary to add meaning to the values used. The next step involved reorganizing the Fortran common blocks into Pascal record structures. At the same time, variables which logically did not belong in their group were reclassified. This lead to the analysis of the Fortran GOTO structures and their dissolution, followed by construction of more fluid Pascal code. From this point, the Fortran code no longer contributed any significant information and the general techniques of program debugging were applied to produce a working version of DNI.

3.0 DOCUMENTATION FOR DNI

3.1 The ATOMLLL Interface

There are several restrictions which must be respected in DNI because it is the post-processor for the ATOMLLL program. The main problem is that ATOMLLL is written in Fortran and thus the output it produces is tailored for Fortran. Primarily, this means that instead of a "trapezoid" description appearing as one contiguous 11 element record (which will be described more fully later), the first elements for each of the first 200 records appear, then, the second elements for the first 200 records appear, and so on for each of the eleven sets of elements. The number 200 is a buffer size chosen for the Lawrence Livermore Laboratory environment. This presents a minor problem in the Pascal program. Because a "trapezoid" is described by 11 parameters, 11 is a logical record size to select. Due to this decision, the routine READ_BUFFER must cross record boundaries to obtain the required input. This is clearly inefficient, but can only be resolved if the output format of ATOMLLL is changed; the buffer size of 200 has been retained in the current Pascal program, although it has been set up as a CONST value and is very easy to change. The above concessions have been made for the data passed by ATOMLLL, though this is not an elegant solution and should eventually be changed, preferably by rewriting ATOMLLL in Pascal.

Another important consequence of the input restriction pertains to the order in which the data is processed. At present, the first pass of DNI renders in color all of the 200 records (or a complete frame, whichever comes first) then recomputes the exposure codes for the highlights. In the current implementation the computation time is not a bottleneck because it is done offline, but if DNI is to be run on a minicomputer coupled directly to the film recorder, this could become critical. A proposed solution for this problem would be to maintain a vector buffer which holds all the exposure codes for the vector being processed. Thus, the algorithm would be modified to perform the color shading and store copies of the computed exposure codes after sending them to the film recorder. Next, the neutral filter would be selected and then the stored group of exposure codes would be sent to the film recorder. Clearly, the most important point to consider before introducing this change is the question of how large is the delay time changing the color filter as opposed to the recomputation of exposure codes for a vector.

A secondary consideration is the delay in recomputing the exposure codes, which can cause phosphor burns unless some form of buffering is done to maintain a smooth flow of data to the film recorder. Alternatively, if the cost of changing the filter is too high for a vector, it could be done for a "trapezoid". This may create a problem with respect to the number of exposure codes which must be stored. In any case, since the timing information is not available, implementation and comparison of these various methods are required to make a valid decision which may result in a hybrid of the two solutions suggested above.

ATOMLLL decomposes the atoms and bonds into visible regions called "trapezoids". In general, a trapezoid consists of two vertical sides and two circular arcs, as in Figure 1. The two arcs may be either convex up (e.g. top arc of Figure 1), convex down, or the degenerate case of a straight line. Each sphere representing an atom is initially decomposed into two trapezoids, as in Figure 2. This is done to satisfy the restriction placed on arcs that they must be either non-increasing or non-decreasing in the Y direction. Note that initially one of the vertical sides is always degenerate for a trapezoid of a full sphere. The addition of other spheres or bonds will cause more trapezoids to be produced. For more details on trapezoids and the hidden surface algorithm which computes them, see references (2) and (3).

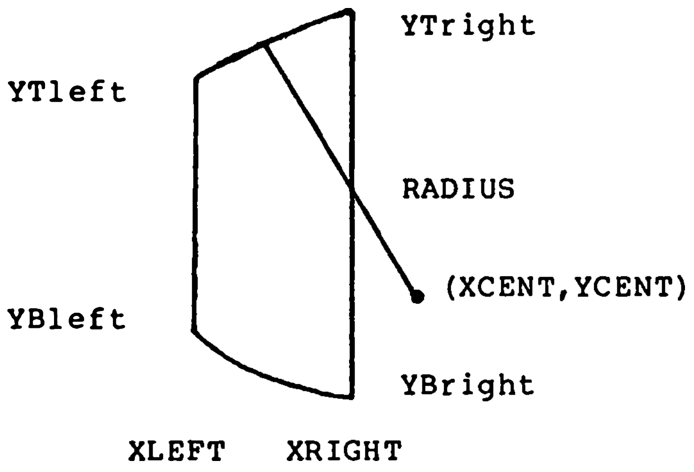

Figure 1

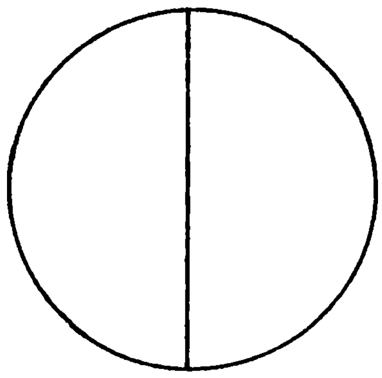

Figure 2

There are five different types of input records. These are the sphere, cylinder, trapezoid, end of frame, and end of job records. Each of these is introduced by a negative value in the ATOM_BOND_NO field. A complete description of individual records follows below.

The sphere record contains four pieces of information which describe a sphere and are used for shading purposes. In the PROCESS_SPHERE procedure, NEW_COLOR receives the color of the sphere from the input record and SPHERE_XCENT, SPHERE_YCENT, and SPHERE_RADIUS are loaded with their respective values.

----

* Figures 1 and 2 appear originally in reference (3).

The next record type is a bond. When a bond goes through the perspective transformation, it becomes a truncated cone. Information about this "cone" can be derived by describing the top and bottom edges and the center line which bisects them. The data given in the record consists of three groups of 3 numbers which come after the ATOM_BOND_NO. For each group, if the first number is a 1, the next number is (2 to the power 15) - 1 times the slope, and the third number is the normalized Y intercept. If the first number is a 2, the reciprocal slope has been supplied, and the X intercept follows. The last entry in the 11 element record contains the highlight intensity of the bond, which is dependent on the angle between the axis of the cylinder and the light source.

There are eleven numbers which are required to describe a trapezoid. The names of these elements are given as they appear in the DNI program. The first parameter is the ATOM_BOND_NO. This integer describes the sphere or bond to which the trapezoid belongs. The next two parameters are TYPE_TOP_ARC and TYPE_BOT_ARC. Both of these integers describe the type of the top and bottom arcs. The arcs can be one of the following:

1) convex downward,

2) convex upward,

3) straight, or

4) a special value to keep some of the other numbers in range after real values have been normalized into integers.

The values for XLEFT and XRIGHT are shown in Figure 1. They are the extreme values in the X direction. The next three values are TOP_ARC_XCENT, TOP_ARC_YCENT, and TOP_ARC_RADIUS. These numbers represent a center and radius which sweep out the top arc between the values of YTleft and YTright (as in Figure 1). Similarly, the last three numbers BOT_ARC_XCENT, BOT_ARC_YCENT, and BOT_ARC_RADIUS perform identically for the bottom arc.

The last two types of records are the end of frame and end of job indicators. The only relevant data on these records is the ATOM_BOND_NO which describes the record type.

The normalization process mentioned above applies to the last eight elements of each record which are actually real values in ATOMLLL, but have been converted to integers for compatibility with 16-bit minicomputers such as the PDP-11 and Varian V-75. If the real values are forced into the range -2 to 2, we can multiple by a factor of 16384 (2 to the power 14) and make them directly compatible as Dicomed vector screen coordinates. The original reason for this conversion was that at Lawrence Livermore, the Varian V-75 minicomputer does not have floating point hardware and converting to a floating point representation would be awkward.

3.2 Internal Operation of DNI

All of the procedures of which DNI consists will be discussed in the following section. The first of these is the main program.

The main program provides the control structure for processing the input file received from ATOMLLL. It begins by performing some initialization functions both for internally defined variables and user defined input parameters. Following this, it consists of three major nested while loops. The outer loop simply controls reading of input buffer information. The next inner loop allows for the two passes through the input data. These are the coloring phase and the highlighting phase. Finally, the innermost loop determines the type of each input record and calls an appropriate procedure to do the required processing. A trace of what types of input records are being processed will be printed on the user's default output device while the trapezoids are being processed. Unfortunately, because of Pascal's buffered output, long periods may occur between interactive responses on a timeshared system.

Some of the initialization routines which are called are OPEN_FILES, GET_OPTIONS, INITIALIZE, READ_BUFFER, and SET_TABLE.

The routine OPEN_FILES provides the interface to the file system by opening the files for I/O processing.

The procedure GET_OPTIONS requests the necessary input parameters from the user. These options include

1) waiting between finished frames for the operator to input a signal to start the next frame when special handling is required between frames,

2) a double film advance between frames to allow for spacing so film can be cut to mount as 35mm slides,

3) a debugging option which produces a procedure execution trace or a complete procedure, and variable dump trace, and

4) a request for the resolution: 4096 X 4096, 2048 X 2048, or 1024 X 1024.

INITIALIZE loads various constants such as frame counters, pixel sizes for point and vector calculations, screen coordinates, color tables, and several others. It also sends out Dicomed codes to ready the film recorder for the first frame.

The READ_BUFFER routine reads in a block of 200 (in general bufsize) input records when required.

The SET_TABLE procedure loads values into the color look-up tables for the Dicomed film recorder. Initially, it reads a set of measured film densities and computes 2048 exposure codes, one set of 256 for each of the eight filters: black and white, red, green, yellow, blue, magenta, cyan, and neutral.

As their names suggest, the next group of four procedures cover a major part of the processing. These are PROCESS_FRAME, PROCESS_SPHERE, PROCESS_CYLINDER, and PROCESS_TRAPEZOID. Each will be described in the following paragraphs.

The PROCESS_FRAME procedure is invoked for post-processing after a frame is finished. It sends a frame advance command to the film recorder. Then, if the wait mode has been selected by the user, it waits until a signal to continue is entered by the operator. The film recorder is then initialized for the next frame.

The PROCESS_SPHERE routine changes the color filter, if necessary, to begin the trapezoid for a new sphere. Next, it obtains some information about the sphere which will be pertinent to processing the trapezoids which make up the sphere. Several constants used in the differencing scheme for computing exposure codes are calculated, completing the procedure.

PROCESS_CYLINDER performs similar computations for a bond record. For each of the three groups of numbers appearing after ATOM_BOND_NO (i.e. one group for each of top line, bisecting line, and bottom line), the type of the line is determined and then the slope for the line (the second entry), and the intercept value (the third entry) are computed. The eleventh entry is used as a highlight intensity for the bond, to compute initial values for the differencing scheme.

The procedure PROCESS_TRAPEZOID is the major processor; it handles the majority of the records. This routine has been broken up into several smaller pieces. Thus, PROCESS_TRAPEZOID computes some radius and slope information (depending on the types of the lines), the number of scan lines in the Y direction, and then calls the routine SHADING_CONTROL.

SHADING_CONTROL is one large loop which moves across the trapezoid in horizontal (X) increments generating vertical (Y) scan lines. It computes a maximum and a minimum Y value for the vertical scan line and the number of pixels on the line. Then, if this trapezoid belongs to a sphere, the QUAD routine can be called directly to shade the entire region. However, if this trapezoid belongs to a bond, the values already computed may be of no use at all and the procedure SHADE_CYLINDER is called to make further decisions on the required shading.

If SHADE_CYLINDER is called, the trapezoid is further decomposed into two regions for shading. These are the parts of the trapezoid above and below the bisecting line. The reason for this split is that two different quadratic equations have to be used to compute the shading in the two different areas. (In the case that the bisecting line is directly vertical, the entire trapezoid can be shaded by one quadratic and the routine SHADE_VERT_SEG may be used.) The maximum Y value, the minimum Y value, and the median Y value on the bisecting line are computed and the procedure CONTINUE_SHADING must be called for the general case.

The routine CONTINUE_SHADING uses the three values of Y computed by SHADE_CYLINDER to decide which routines to use to shade each of the two regions.

There are five different shading routines which may be used to render a bond. These are SHADE_VERT_SEG, SHADE_LOW_VERT, SHADE_LOW_HORZ, SHADE_HI_VERT, and SHADE_HI_HORZ.

The procedure SHADE_VERT_SEG treats the entire trapezoid the same way, using only one quadratic to produce the shading for the entire region. A loop in the procedure steps through the Y values sequentially as the X values for the top, middle, and bottom lines are computed. These allow the required exposure code for each pixel to be computed and sent to the Dicomed film recorder.

The two routines SHADE_LOW_HORZ and SHADE_LOW_VERT form a pair. Only one of these routines is executed for each trapezoid. The former does the same job as SHADE_VERT_SEG for the region below the highlight (bisecting) line, the latter routine computes the first value of the quadratic and the first two differences and calls QUAD to generate exposure codes. The major difference between these two routines is that Shade_Low_Horz generates scan lines running in the X direction and thus must calculate individual exposure codes itself, whereas Shade_Low_Vert can take advantage of the differencing scheme because codes run along Y scan lines.

The last two shading routines for bonds are SHADE_HI_HORZ and SHADE_HI_VERT. They perform similar functions to their 'LOW' counterparts described in the preceding paragraph. Again, only one of these routines need be executed for each trapezoid since they both shade above the highlight line.

The remainder of the procedures are composed of various support routines which are called from other parts of the program when necessary.

DICOWD handles all Dicomed commands and stores them in a buffer of size buff_len. When this buffer is full, the routine flushes it and starts filling the buffer again, from the start. The Lawrence Livermore Laboratory version of DNI used double buffering, but this is unnecessary for the new version because the program generates all of the Dicomed codes before they are sent to the film recorder. Also, it was noticed that periodic pauses in programs generating codes directly left unpleasant burns in the phosphur which resulted in poor film quality. For this reason, an Unix filter was used to smooth out the output of Dicomed codes from programs coupled directly to the film recorder.

QUAD evaluates a quadratic equation by using a differencing scheme. Any quadratic can be evaluated at succeeding points with two additions if the initial value and the first and second differences are supplied. This routine is passed the above three pieces of information plus the number of pixels at which the quadratic is to be evaluated. A loop in QUAD produces two Dicomed exposure codes at a time using the differencing method, concatenates them into a 16-bit word and sends them to the DICOWD routine. This process continues until the pixel count has been exhausted. For a more detailed explanation of the differencing scheme, see reference (3).

CHANGE_COLOR flips the color filter on the Dicomed film recorder.

TABOUT selects either the color translate mode or the black and white translate mode for color or highlight shading.

EMPTY_BUFFER flushes the remainder of the Dicomed commands at the end of a frame when DICOWD's buffer is partially full.

POWER is a function which raises a number to the given exponent. The reason for the existence of this function is that Pascal does not support such an operator. The recommended method of doing this is to take the log of a number, multiply it by the power and take the exponential of the result.

AMAX and AMIN are real functions whose two arguments are real. They compute the maximum and minimum respectively.

MAX_INT and MIN_INT are the integer equivalents of AMAX and AMIN.

A simplified version of how the major procedures from above tie together follows:

```
Main Program, read a buffer.
     Begin processing, first in color then
          repeat in black and white for highlights.

Case Sphere :
     Read a sphere record.
          Store data about the sphere.
     Read a trapezoid record for this sphere.
          Invoke Process_Trapezoid to determine the
               number of scan lines in Y.
          Invoke Shading_Control to compute the
               maximum and minimum y values.
          Invoke Quad to shade the trapezoid
               using a differencing scheme.
     Repeat until no more trapezoids.

Case Bond :
     Read a bond record.
          Store data about the bond.
     Read a trapezoid for this bond.
          Invoke Process_Trapezoid to determine
               the number of scan lines in Y.
          Invoke Shading_Control to compute the
               maximum and minimum Y values.
          Invoke Shade_Cylinder to compute maximum,
               median, and minimum Y values.
          If median line is vertical,
               Invoke Shade_Vert_Seg to shade
               the entire trapezoid.
          Else
               Split the trapezoid into two regions.
               Shade below the highlight line using either
                    Shade_Low_Horz or Shade_Low_Vert.
               Shade above the highlight line using either
                    Shade_Hi_Horz or Shade_Hi_Vert.
     Repeat until no more trapezoids.

Repeat for the next record.
```

3.3 Output Description

Currently, the DNI program produces Dicomed commands encoded into ASCII as decimal digits. The reason for this is that the Dicomed film recorder expects 16-bit words containing commands, while the Honeywell system produces 36-bit integers. By not producing binary directly, the program will behave the same way on any system with a Pascal compiler, independent of the machine word size (as long as it s at least 16-bits). The film recorder commands are thus written out with a single blank between commands and without any end of line characters.

A Unix filter program decodes the character representations of the command codes and produces the appropriate binary format for the film recorder. A very simple C filter to do this job has been included in the appendices. The file it produces can be transferred to a tape or used directly to drive a Dicomed.

The above is a description of what happens when the program runs smoothly. DNI may generate program trace and variable dump information, or error messages when values are not in their expected range. The trace facility and variable dump data are used for checking intermediate results and for following program flow. They print out the internal names of procedures and their variables. The error messages (for the most part related to variables indexing into case statements) identify the procedure in which they occur and the variable in error. The program attempts to continue execution even when errors occur.

3.4 Testing of the Pascal Program

Testing the validity of the output from DNI turned out to be a difficult problem. Some of the reasons for this difficulty were the size of the output files generated and the inconvenience of travelling to the site of the Dicomed film recorder.

The first phase of testing the new version of DNI involved comparing the disassembled codes generated from both the Fortran and Pascal versions of the program. This was not a completely satisfactory test because of the amount of space used up by the files the disassembler created. Thus, only the first twenty thousand codes were compared. After the disassembler verified that this much of the output was acceptable, an Unix C program was used to perform a byte by byte comparison. This test proved the validity of the output of the Pascal version of DNI by showing that all exposure codes were either identical or different by at most one unit (attributable to roundoff error) when compared to the Fortran output.

The second phase of testing involved producing pictures of atoms. Three of these pictures appear originally in reference (4). They show simple scenes of three atoms with two joining bonds. The second group of three pictures show several atoms intersecting in a "CPK space-filling model".

This latter group of pictures were aided in creation by the preview capability in the Waterloo version of the ATOMLLL code which drew outlines of the models on a Tektronix storage tube screen. Thus, a significant amount of testing was performed before the pictures were commmitted to film. A group of five pictures appear in the appendices.

4.0 DNI PORTABILITY

The new version of DNI has been written to adhere to the version of Pascal described by Jensen and Wirth (see reference (6)). Hopefully, there are only a small number of portability problems. To determine some of these problems after the program was debugged on the Honeywell system, DNI was moved to the IBM 370/158 at Waterloo. The Waterloo Pascal compiler, developed by the Computer Systems Group, also implements the language developed by Jensen and Wirth. By running DNI on these two different systems, many unnecessary system dependencies were eleminated. Most of these involved file I/O or features such as the curly braces used in DNI to enclose comments.

Open statements for the files are the standard Pascal defined procedures reset and rewrite. Each procedure takes one argument which is the internal name of a file which appears in the program statement and is declared in the main program's Var section. The Honeywell version adheres to this standard, but the IBM version requests an additional character string containing the triple (filename, filetype, filemode) to associate an external file with an internal one. The problem of the comment delimiters will not arise if the curly brackets exist in the character set one is using; otherwise, some substitution must be made for them. Because (* and *) are accepted by all Pascal systems, these characters were used to enclose comments.

One final note which refers to the output discussed in the previous section, is that the output format will be compatible with almost all other systems. However, because DNI does use large integers, the computer system it runs on must define integers to be at least 32-bits long. This may be a problem on some 16-bit machines, because of the accuracy needed for differencing, in which case multiple precision arithmetic will be needed. The Pascal compiler on Waterloo's Unix system does not have 32-bit integers. The program needs to take this into account to run on that system.

5.0 CONVERSION OF DNI FOR A FRAME BUFFER DEVICE

5.1 Discussion of Operation

A frame buffer is an array which contains the intensities for each pixel of a complete picture. One reasonable method of employing a frame buffer is to store an index for a color lookup table at each pixel. Thus, a table of colors and/or intensities can be indexed, where each entry contains three values to describe the red, blue, and green components for this color.

To convert DNI to produce output for a frame buffer, several things must be done. First, and most obviously, the generation of Dicomed commands would no longer be necessary. If the frame buffer contains 24-bits of intensity, the problem becomes one of producing 8-bits representing each of the colors red, green and blue. DNI currently computes cos(theta) squared for a table lookup value. To maintain a degree of compatibility with this scheme, the value generated by DNI would still be used for a table lookup. The table would contain the corresponding values of intensity for A + D cos(theta) and C $\cos(\theta)^{28}$. A color table would also be used. This would contain the components of red, green and blue for each of the colors to be used plus one additional entry with equal amounts of these colors for white. The first lookup value of A + D cos(theta) would be used to modify the values of the red, green and blue components for the selected color. This would provide for the color, diffuse shading of the object. The value of C $\cos(\theta)^{28}$ would be modified by the color components of white to produce the effects of specular reflection. Finally the two components for each of the three colors would be combined and stored for each point in the frame buffer.

The above discussion pertains to a 24-bit frame buffer because the University of Waterloo will soon be acquiring such a device. However, this amount of resolution is not necessary to achieve the same results as those obtained on the Dicomed. If a color lookup table is used, eleven bits per pixel is sufficient. These would represent the 8-bit exposure code DNI already computes and the 3-bits of color selection information. The best method to use for the color calculations would depend on the operating characteristics of the device and how flexible it is. In the final analysis, the fastest color decoding algorithm would be the most desirable.

The use of a table lookup for the intensity values is a useful idea because it makes changing the expressions for shading very easy. Also, it would make the second pass through the data for highlighting unnecessary. The expressions for shading which were suggested originally appear in reference (4).

5.2 The Aliasing Problem

The problem of aliasing has not been addressed in DNI for output on the Dicomed film recorder. If a different device like a frame buffer is used, an anti-aliasing procedure could be added to remove jagged edges. (This could also be done for the Dicomed and is suggested as an improvement.)

The current version of ATOMLLL already keeps track of the outlines of all the atoms and bonds for the preview feature. These are exactly the areas which will be affected by an anti-aliasing routine. On the regions between two differently colored atoms or an atom and a bond, there exist two separate lines colored in black. These lines prevent the bleeding together of the two colored regions. This means that all the edges of the atoms or bonds border completely black regions. Therefore, the exposure codes along the edges can be averaged given that one knows the outside is black and the calculated exposure code for these points (Note, the outline regions would no longer be drawn). This gives rise to intensities at the edges which will reduce the effects of aliasing.

5.3 Specific Changes to DNI

The code in DNI would be affected by the implementation of the frame buffer algorithm in the following manner. The SET_TABLE procedure would no longer be necessary in it's present form. However, the color lookup table would still have to be loaded for the frame buffer device. The PROCESS_FRAME procedure would have to be modified and may not be necessary any more because it's prime function was sending out Dicomed codes to end a frame. SHADING_CONTROL sends out a position CRT command to move the beam to the correct position to draw the next vector in Dicomed vector coordinates (in the range 1 to 32768). The two coordinates in Y then X it generates would have to be divided by (8 * pixel_size) to bring them back to the correct range for the selected resolution. The routines SHADE_VERT_SEG, SHADE_LOW_HORZ and SHADE_HI_HORZ need only call a new routine instead of DICOWD. The new routine would produce the exposure code by performing the table lookups and computing the color information in the manner previously described. QUAD would also call this new routine to produce the necessary information for the 8-bit frame buffer intensities. The CHANGE_COLOR routine could be modified to keep track of the current color for the new implementation. Finally, the procedures TABOUT and EMPTY_BUFFER would no longer be necessary and the second pass in the main program could be eliminated.

6.0 SUMMARY

The major points of interest in converting DNI from Fortran to Pascal will now be reviewed. The first step that was taken was to write down a dictionary of all the variables in the Fortran program and to describe the purpose of each. Next, the common blocks were reorganized to structure them into consistent units of usage. These were then used to create the Pascal record structures which contain most of the variables used in DNI. Through the use of the with statement, the scope of these record variables was well defined. This gave rise to a one-to-one mapping between most of the old variables and the new ones. Following this, the convoluted Fortran source was analyzed to determine what the control structures were and how they should be organized in Pascal. At the same time, because of this restructuring a great deal more insight into how DNI worked was obtained. Finally, the new Pascal version of DNI was written and debugged to the point where it produced output.

This output was converted using a C filter into the same 16-bit binary words which the original Fortran program produced. After this, the binary from the old and new versions of DNI were run through a Dicomed code disassembler written by Alex White. The resulting files from that program were then processed by the DIFF utility program on the Unix operating system to determine any differences. When none were found, another C program was written to perform a direct comparison on a byte by byte basis on the two binary files.* Once the two output files matched within reasonable limits, a test file was run with the new Pascal DNI to produce an output file for processing on the Dicomed film recorder at the Defence and Civilian Institute for Environmental Medicine at Downsview, Ontario. When this test run had been verified to operate correctly, the DNI code was transported to the IBM system running Waterloo Pascal. By running DNI on another Pascal compiler with

----

* The two C programs and a reference to the disassembler appear in the appendices.

slightly different specifications, most unnecessary system dependent features of the Pascal code were located and removed. Following this, DNI was retested on the Honeywell system to ensure that it still worked properly. Finally, several more test files were run through DNI to obtain output for this essay.

Some points which are of interest with respect to conversions in general follow.

The block data subprogram in the Fortran source code was the cause of several problems in running Nelson Max's version of DNI. It would appear that many loaders will only satisfy external references that have been used in the program. Thus, since there is no way to reference a block data subprogram, it does not get loaded and the variables initialized there are undefined. Two possible solutions exist to this problem. Either one does not use block data subprograms or some form of "kludge" is necessary to force the loader to load the block data area. Neither is very elegant.

The use of Pascal record structures to represent Fortran common blocks was very successful. Not only did it make the conversion effort easier, but it also allowed one to explicitly restrict the scope of the record variables through the use of the with statement. The only immediate drawback of this scheme was that potential ambiguities could arise with respect to other variables. For example, if a local variable in a procedure has the same name as one of the record variables, within the scope of a with statement, it would not be possible to access the local variable. Clearly, this situation is syntactically correct, but there is no method available to distinguish between the two different variables.

Through general experience gained in this project, it is not recommended to attempt to transport files from one system to a second before some form of conversion is done. The program or data which is essential on the second system should be made as compatible as possible on the transport medium before the information is moved. An example of this is the transport of DNI to the IBM system. DNI was converted from ASCII to upper case EBCDIC before the IBM system received the file. This saved a considerable amount of effort. Another example was the transport of the ASCII file of Dicomed codes from the Honeywell system to the Unix operating system. This phase of the project turned out to be a major headache. The main reason for this was that the files being manipulated were very large. Because Unix had very restricted amounts of available disk storage and fairly heavy CPU usage, conversion from ASCII to 16-bit binary code was a considerable bottleneck. A suggestion to alleviate this problem on the Honeywell would be to add an assembler subprogram to take 9 Dicomed codes of 16-bits and concatenate them into four 36-bit words. These could be written out to tape directly to bypass the C filter and the Unix operating system. Such a subprogram would have to be written from scratch at each different installation at which DNI was implemented because it depends on machine word size.

One last point which should be stressed: every effort should be made to avoid writing unnecessary code. In general, this refers to becoming acquainted with the operating system and finding out what utility programs are available. Examples of this included the use of ACL for disk to tape conversion on the Honeywell 66/60, using DIFF and CMP on the Unix system for file comparison, various routines like DD for input/output conversion on Unix, and the Dicomed code disassembler. Thus, considerable effort was saved by not writing programs already in existence.

References

1. Crow, F. C. "The Aliasing Problem in Computer-generated Shaded Images", CACM, 20(11):799, November 1977.

2. Knowlton, Ken and Cherry, Lorinda. "ATOMS, a three-d opaque molecule system", Computers and Chemistry Vol. 1, no. 3 (1977) pp. 161-166.

3. Max, Nelson. "ATOMLLL - ATOMS with Shading and Highlights", Computer Graphics, 13(1):165, Spring 1979.

4. Max, Nelson. "ATOMLLL - a three-d opaque molecule system, Lawrence Livermore Laboratory version", UCRL-52645, Lawrence Livermore Laboratory 1979.

5. Newman, W. M. and R. F. Sproull, Principles of Interactive Computer Graphics, 2nd edition, McGraw-Hill, New York, 1979.

6. Wirth, N. and K. Jensen, PASCAL : User Manual and Report, 2nd edition, Springer-Verlag, New York, 1975.

APPENDIX I Honeywell Batch JCL

The following Honeywell JCL is used to run the DNI program in batch mode. The files which are referenced by the program supply the following:

1) gr/./hyuen/dni contains the source code for DNI,

2) gr/./hyuen/new.2 contains the output description supplied by the ATOMLLL program,

3) gr/./hyuen/f02 will be used to dump debug information onto,

4) gr/./hyuen/new.two will receive the Dicomed codes,

5) hyuen/fort04 contains measured film densitites for the color lookup tables in the film recorder, and

6) hyuen/dni.in contains the input parameters requested by DNI.

The JCL is presented below.

```
ident    hyuen,dni
lowload
option   nofcb
program  pascal,range,singlecase,stack=10000
select   pascal/compile
limits   5
prmfl    s*,r,s,gr/./hyuen/dni
select   pascal/execute
limits   20,50k
prmfl    at,r,s,gr/./hyuen/new.2
prmfl    de,w,s,gr/./hyuen/f02
prmfl    di,w,s,gr/./hyuen/new.two
prmfl    ta,r,s,hyuen/fort04
prmfl    i*,r,s,hyuen/dni.in
endjob
```

APPENDIX II ACL Tape Transfers

The two Honeywell JCL programs in this appendix perform tape transfer functions. The first one converts the ASCII version of DNI into EBCDIC and writes it onto a tape which is used to transport it to the IBM 370/158. The second program is used to write out a Dicomed code file onto tape for use on the Unix system.

Program 1

```
          ident      hyuen,dni.tape
          msg2       l,need tape 12012 with ring
          program    acl
          prmfl      **,r,r,./batchacl
          tape       t0,xld,,12012,,dni.p,,den8
     copy gr/./hyuen/dni to at:t0(ebcdic,upper,fb=80,blksize=800)
     close
     end
          endjob
```

Program 2

```
          ident      hyuen,unix.tape
          msg2       l,need tape 12351 with ring - scratch tape
          program    acl
          prmfl      **,r,r,./batchacl
          tape       t0,xld,,12351,,dni.2,,den8
     copy gr/./hyuen/new.two to at:t0(blksize=6000)
     copy gr/./hyuen/new.for
     copy gr/./hyuen/new.six
     end
          endjob
```

APPENDIX III C Filters

The first C program performs the byte by byte comparison of two files to determine the differences. The second C program decodes ASCII Dicomed codes and packs them into 16-bit words.

Program 1

```
#include <stdio.h>
main (argc,argv)
int argc;
char *argv[];
{   static int ftn[256], pasc[256];
    int nl, n2, i;
    long pointer, count = 0;
    FILE *fopen(), *fptr, *pptr;

    fptr = fopen(*++argv,"r");
    pptr = fopen(*++argv,"r");
    while ( (nl = fread(ftn,sizeof(*ftn),256,fptr)) > 0 )
    {
        n2 = fread(pasc,sizeof(*pasc),256,pptr);
        if (nl != n2)
           {printf("Length of files not equall\n");
            if (count > 0)
               printf("Files equal up to %ld blocks\n",count);
            printf("Count = %ld\n",count);
            break;
           }
        else
               for (i=0; i < 256; i++)
               {pointer = (count * 256) + i;
                if (ftn[i] != pasc[i])
                 {
                  printf("Word number %ld different ftn = %4x, ",
                         "pasc = %4x\n",pointer,ftn[i],pasc[i]);
                 }
               }
        count++;
    }
    fclose(fptr);
    fclose(pptr);
}
```

Program 2

```
#include <stdio.h>
main()
{
    int inword;

    while(scanf("%d", &inword) == 1)
        if(fwrite(&inword, sizeof(int), 1, stdout) != 1)
            error("Urk -- probably no more space\n");
}
```

The Dicomed code disassembler is available on Unix under the catalog /u/ksbooth/dicomed. The compiled version is named DICODE and the source is available as DASM.C.

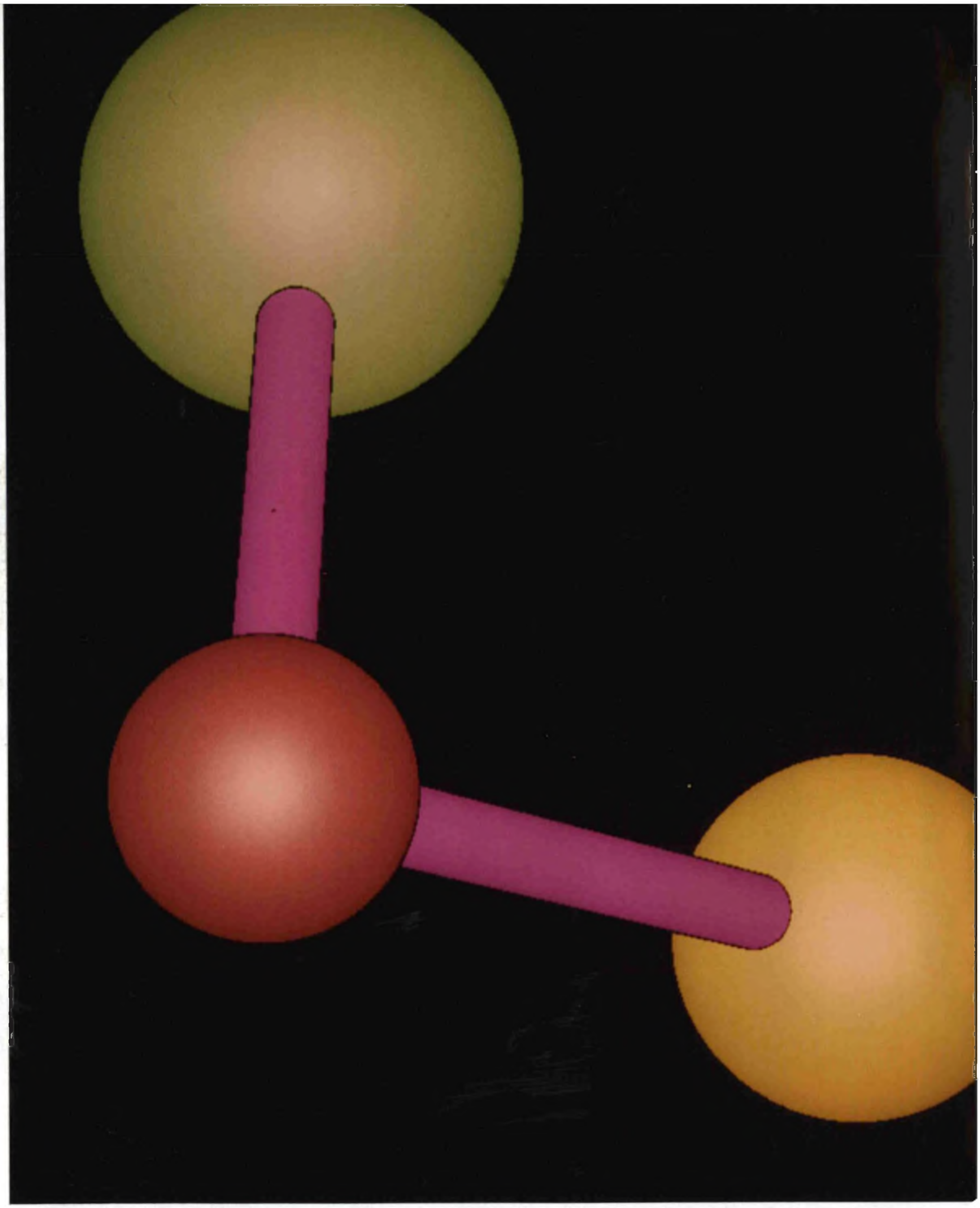

Figure 1 Three distinct atoms with two joining bonds. This frame appears in reference (4).

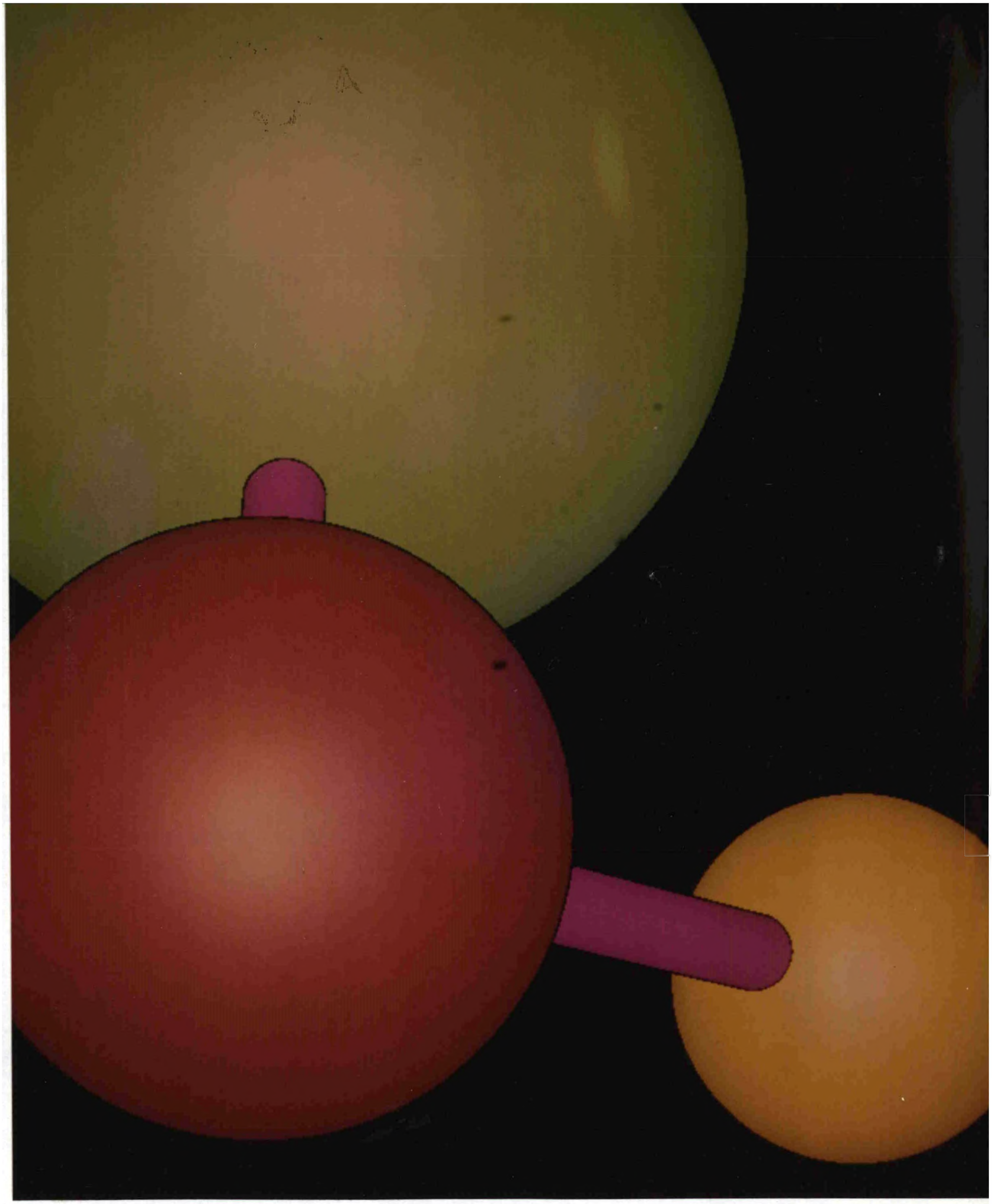

Figure 2 This is the same as Figure 1 with two of the atoms increased in size. This frame also appears in reference (4).

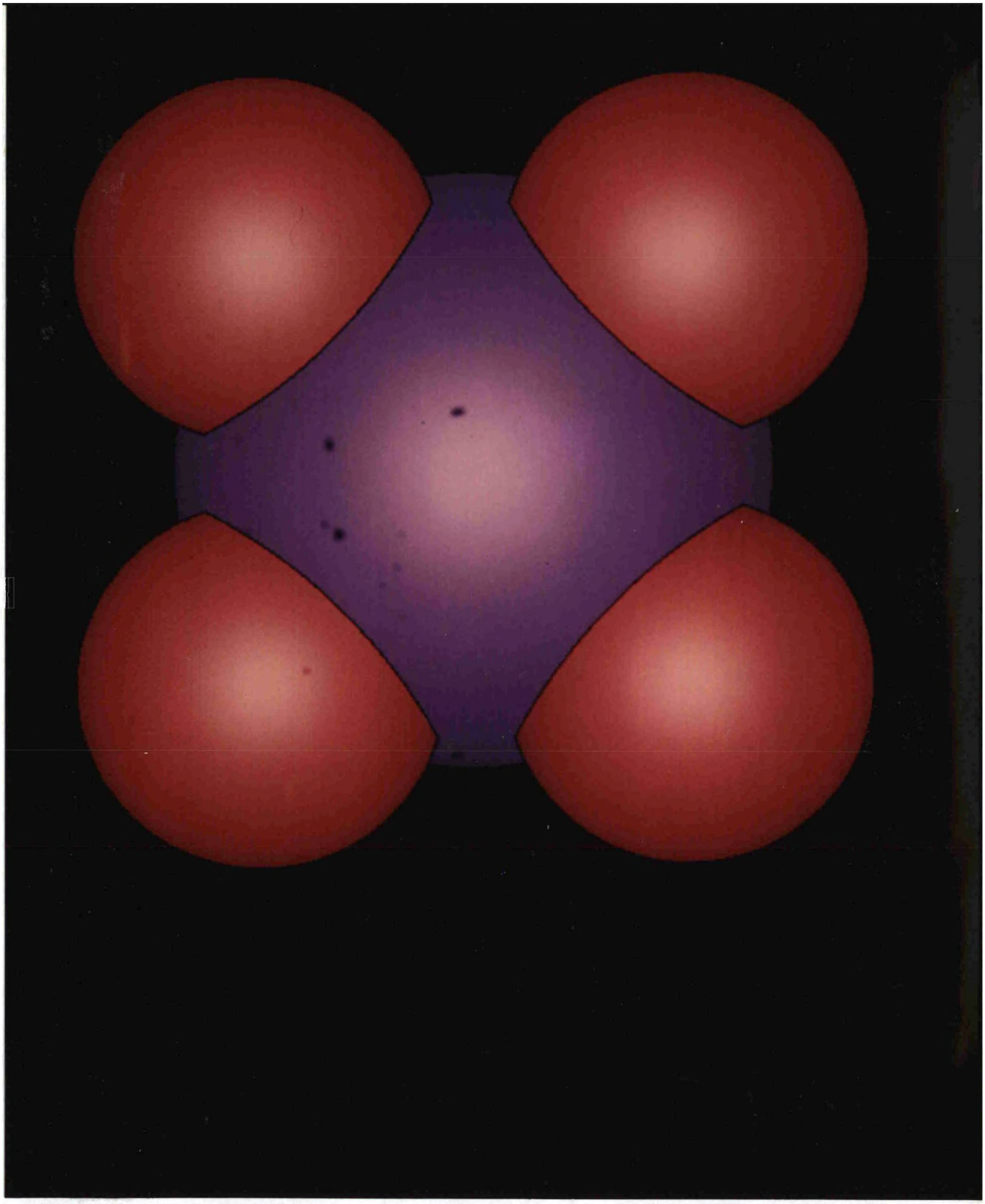

Figure 3 One central atom is occulated by four other atoms on the outside.

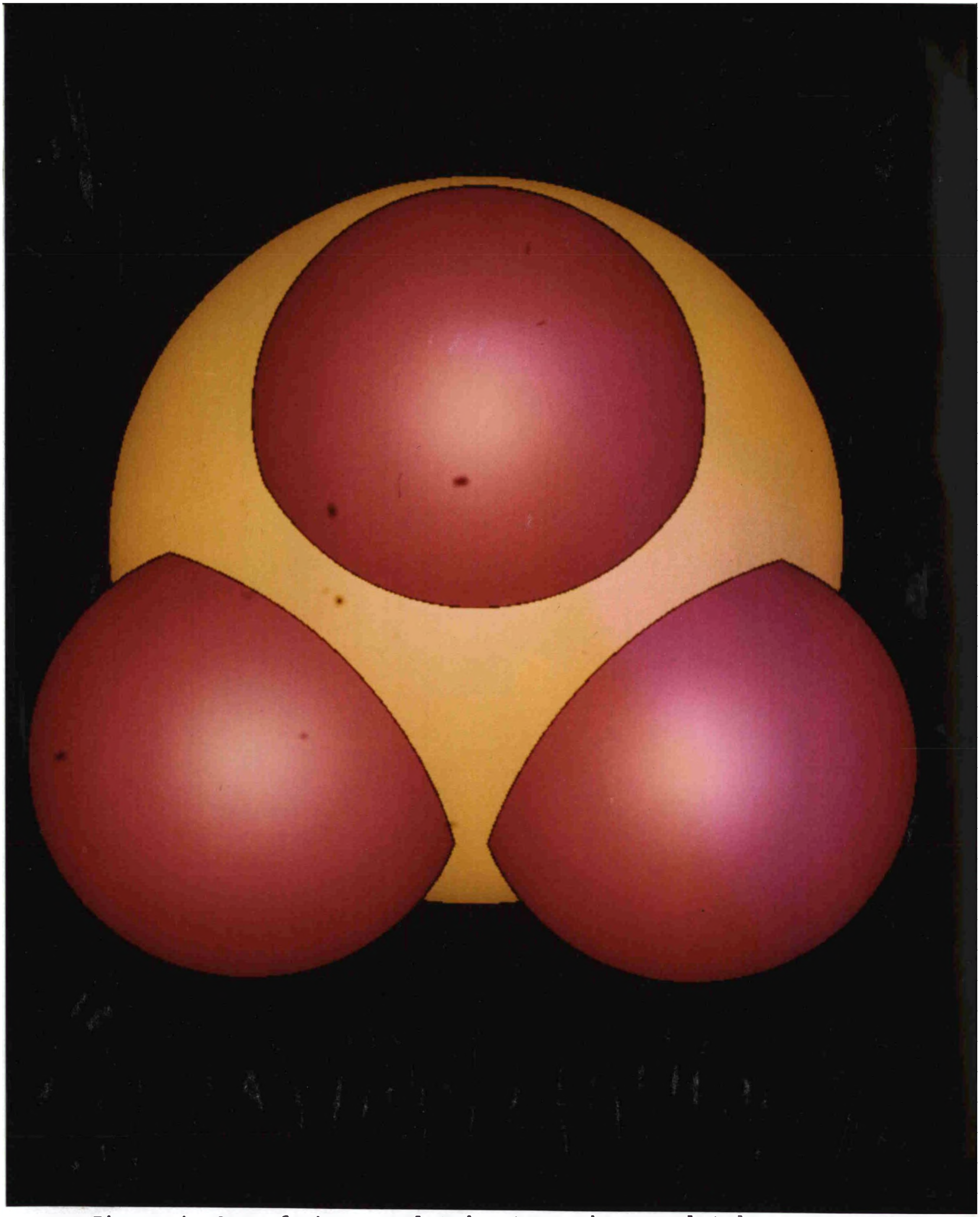

Figure 4 One of the occulated atoms is completely surrounded by the central atom in this frame.

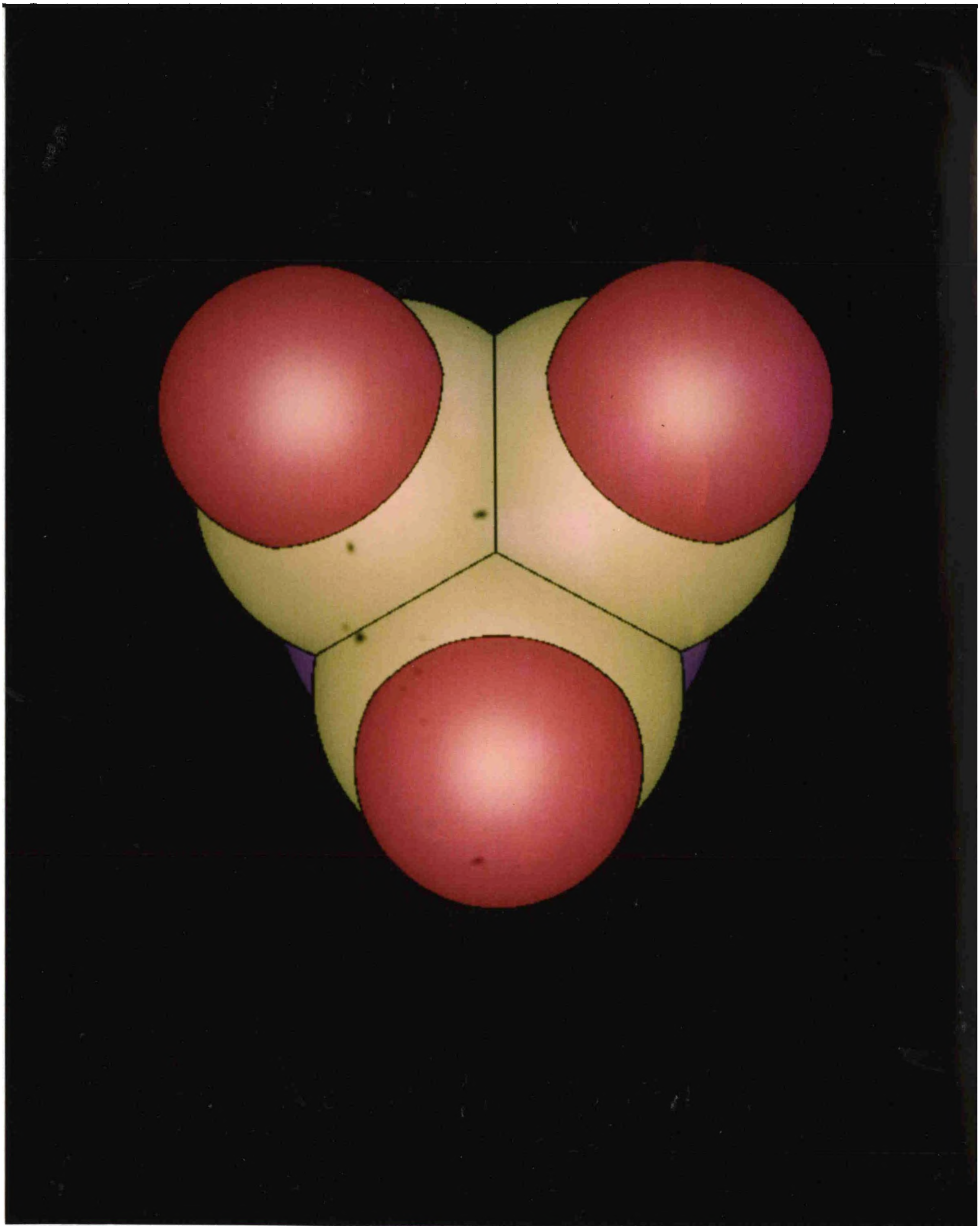

Figure 5 There are seven atoms in this picture. Only two very small parts of one of the atoms is visible; it is shaded in blue. The perspective ratio has been increased to produce a smaller sized model.

APPENDIX IV Data Files and Pictures of Atoms

The first two figures appear in reference (4). The ATOMLLL data to generate these pictures can be found there. The file used as input to ATOMLLL for figures 4, 5 and 6 follows below.

ATOMLLL Input File

```
      5  0.49
  0.0    0.0  14.8   1.0   4
  -.7    -.7  14.5   0.6   1
   .7     .7  14.5   0.6   1
  -.7     .7  14.5   0.6   1
   .7    -.7  14.5   0.6   1
      0  0.0
      4  0.49
  0.0    0.0  14.5   1.2   3
 -0.7    0.7  14.0   0.7   1
  0.7    0.7  14.0   0.7   1
  0.0   -0.4  13.7   0.7   1
      0  0.0
      7  0.30
  0.0    0.0  15.0   1.0   4
 -0.5   -0.4  14.5   0.8   2
  0.5   -0.4  14.5   0.8   2
  0.0    0.5  14.5   0.8   2
 -0.8   -0.7  14.0   0.6   1
  0.8   -0.7  14.0   0.6   1
  0.0    0.8  14.0   0.6   1
      0  0.0
```

APPENDIX V DNI Source Code

```
program dni (input,atomfi,debugfo,dicofo,tablefi,output);
const
    b_w = 2;                      (* Black and white shading *)
    b_w_translate = 21;           (* Black and white translation; P. 3-2 *)
    buff_len = 20;                (* Size of output buffer *)
    bufsize = 200;                (* Size of input buffer *)
    color = 1;                    (* Color shading *)
    color_mode = 18;              (* Select color mode; P. 3-2 *)
    color_translate = 22;         (* Select color translation; P. 3-2 *)
    cylinder = 4;                 (* Type of record being processed *)
    end_of_job = 2;               (* Type of record being processed *)
    frame = 1;                    (* Type of record being processed *)
    frame_advance = 49408;        (* Exposure and filter select; P. 3-21 *)
    func_select = 8192;           (* Function element select; P. 3-8 *)
    init_cond = 34985;            (* Initial condition select; P. 3-17 *)
    load_trans_table = 53248;     (* Second word of func_select; P. 3-14 *)
    neutral_filt = 15;            (* Select neutral filter; P. 3-2 *)
    plot_elements = 28672;        (* Second word of func_select; P. 3-12 *)
    plot_horz = 4096;             (* Offset from bottom of screen *)
    plot_line = 12288;            (* Second word of func_select; P. 3-9 *)
    plot_vector = 20480;          (* Vector or position absolute; P. 3-15 *)
    point_select = 40960;         (* Point element select; P. 3-20 *)
    position_crt = 16384;         (* Vector or position absolute; P. 3-15 *)
    raster_mode = 16;             (* Select raster output mode; P. 3-2 *)
    resett = 36864;               (* Initial condition select; P. 3-17 *)
    shade_hi_v_call = 5;          (* Identifies calling routine in Quad *)
    sphere = 3;                   (* Type of record being processed *)
    trapezoid = 5;                (* Type of record being processed *)

    (* NOTE : All page numbers refer to DICOMED film recorder
              manual (Publication No. 12M069, February, 1979
              Edition - Revision A.
    *)
```

```
type
    input_rec =
        record
            atom_bond_no : integer;    (*  Describes what the current
                                          record is : (-3) a sphere,
                                          (-2) a bond, (-1) a trapezoid,
                                          (-5) the end of frame, or
                                          (-4) the end of job.
                                        *)
            type_top_arc : integer;    (* Arc is :                    *)
            type_bot_arc : integer;    (*   1 convex downward
                                            2 convex upward
                                            3 a straight line
                                            4 special case to reflect the
                                              results of normalization

                                              These flags are used to
                                          determine the type of processing
                                          required for the rest of the
                                          record.
                                        *)
            xleft : integer;           (* X coordinates of the left and *)
            xright : integer;          (* right sides of the trapezoid.
                                        *)
            top_arc_xcent : integer;   (* Center and                 *)
            top_arc_ycent : integer;   (* radius of                  *)
            top_arc_radius : integer;  (* the top arc
                                        *)
            bot_arc_xcent : integer;   (* Center and                 *)
            bot_arc_ycent : integer;   (* radius of                  *)
            bot_arc_radius : integer;  (* the bottom arc             *)
        end;
```

```
bond_rec =
    record
         (* top_line_typ, bot_line_typ, mid_line_typ :
           1 : line convex downward
           2 : line convex upward
           3 : line straight
           4 : special case to deal with normalization
         *)
        top_line_typ : integer;
        top_slope : real;
        top_intercept : real;
        bot_line_typ : integer;
        bot_slope : real;
        bot_intercept : real;
        mid_line_typ : integer;
        mid_slope : real;
        mid_intercept : real;

        fm : real;                  (* Used to compute intermediate
                                      values *)
        fs : real;                  (* for the differencing scheme -
                                      done in process cylinder. *)
        top_indicator : integer; (* Flags used to indicate what
                                      quadratic scheme to use for *)
        bot_indicator : integer; (* shading the particular trapezoid
                                      and  thus,  which  shading
                                      procedures to use.       *)
    end;
```

```
isp_rec =
    record
        pixel_size : integer;        (* Single element plotting
                                       resolution *)
        half_pixel : integer;
        vector_size : integer;       (* Vector plotting resolution *)
        half_vector_size : integer;
        real_vector_size : real;     (* Equal to vector size - real *)
        real_half_vector : real;
        twice_vector : real;
        two_sqr_vect : real;
    end;
```

```
screen_rec =
    record
        screen_top : integer;
        screen_bot : integer;
    end;

sphere_rec =
    record
        sphere_radius : integer;
        sphere_xcent : integer;
        sphere_ycent : integer;
    end;

diff_rec =
    record
        c : real;     (* Various computed values    *)
        d : real;     (* from either process sphere *)
        e : real;     (* or process cylinder, used  *)
        f : real;     (* in the differencing scheme *)
        k3 : integer; (* Second difference of quadratic equation *)
    end;

radius_rec =
    record
        sqr_top_rad : real;
        sqr_bot_rad : real;
        radius_sqr : real;
    end;

color_tbl_rec =
    record
        color_min : array[1..7,1..2] of real;
        color_range : array[1..7,1..2] of real;
    end;
```

```
main_rec =
    record
        a : real;                     (* Used to compute first quadratic
                                        value - initialized in
                                        Change_color. *)
        ad : real;                    (* Used to compute x_exposure,
                                        initialized in Process_Sphere *)
        back_exposure : integer;      (* Computed DICOMED code
                                      *)
        buffer_start : integer;       (* Pointer to first record of
                                        current buffer or frame *)
        curr_color : integer;         (* Currently selected DICOMED color
                                        filter
                                      *)
        eoj : boolean;                (* Used in main program control
                                        structure
                                      *)
        frame_no : integer;           (* Current frame count
                                      *)
        horz_space : integer;         (* Computed DICOMED code
                                      *)
        i : integer;                  (* Pointer to current input record
                                      *)
        idum : integer;               (* Used to read into to cause a
                                        wait for the operator between
                                        frames
                                      *)
        ier : integer;                (* Not currently used, was an error
                                        flag in the Read_Buffer routine
                                      *)
        kind : integer;               (* Either a sphere or a bond
                                      *)
        min_slope : real;             (* If the slope of an arc is less
                                        than this value, it is straight
                                      *)
        new_color : integer;          (* Contains new color from the
                                        current input record *)
        not_done : boolean;           (* Used in main program control
                                        structure
                                      *)
        range_col : real;             (* Maximum range for color
                                        exposure
                                      *)
        shading : integer;            (* Either color or black and white
                                      *)
        shift_op : integer;           (* Normalization factor, used to
                                        convert reals to 16 bit
                                        integers *)
        this_buffer : boolean;        (* Used in main program control
                                        structure
                                      *)
        vert_space : integer;         (* Computed DICOMED code
                                      *)
    end;
```

```
trap_rec =
    record
        debug_flag : boolean;
        max_y_scan : real;     (* Used to compute  *)
        mid_y_scan : real;     (* scan line length *)
        min_y_scan : real;     (* - real values    *)
        no_pixels : integer;
        top_y_bond : integer; (* Used to compute  *)
        mid_y_bond : integer; (* scan line length *)
        bot_y_bond : integer; (* - integer values *)
        x : integer;
        y : integer;
        xtop : real;          (* Used to compute scan line *)
        xmid : real;          (* length for horizontal     *)
        xbot : real;          (* shading lines.            *)
        x_exposure : integer; (* Computed exposure code according
                                 to x value on a horizontal scan
                                 line. *)
    end;

dico_rec =
    record
        buffer1 : array[1..buff_len] of integer;
        buffer2 : array[1..buff_len] of integer;
        buff_ptr : integer;
    end;
```

```
var
    buffer : array[1..bufsize] of input_rec;
    bond_c : bond_rec;
    isp_c : isp_rec;
    screen_c : screen_rec;
    sphere_c : sphere_rec;
    diff_c : diff_rec;
    radius_c : radius_rec;
    color_c : color_tbl_rec;
    main_c : main_rec;
    trap_c : trap_rec;
    dico_c : dico_rec;

    processing : integer;  (* Contains current record type *)

    advance_flag, wait_flag : 0..1;
    debug_option            : 0..2;

    advance_mode : boolean; (* If advance_flag = 1, allow blank film
                               between 35mm slides. *)
    debug_proc   : boolean; (* If debug_option = 1 or 2,
                                  print a trace of procedure calls. *)
    debug_var    : boolean; (* If debug_option = 2,
                                  print values of variables *)
    wait_mode    : boolean; (* If wait_flag = 1, pause between frames
                                  by reading a dummy variable. *)

    atomfi, debugfo, dicofo, tablefi : text;
```

```
function amax (first, second : real) : real; forward;

function amin (first, second : real) : real; forward;

procedure continue_shading (i:integer); forward;

procedure change_color (var curr_color : integer;
                        new_color : integer; var a,bp : real;
                        shading : integer); forward;

procedure dicowd (iword : integer); forward;

procedure empty_buffer; forward;

function max_int (first, second : integer) : integer; forward;

function min_int (first, second : integer) : integer; forward;

function power (no, exponent : real) : real; forward;

procedure quad (k1, k2, k3, nw, icall : integer); forward;

procedure set_table (kind : integer); forward;

procedure shade_cylinder (i : integer); forward;

procedure shade_hi_horz; forward;

procedure shade_hi_vert; forward;

procedure shade_low_horz; forward;

procedure shade_low_vert; forward;

procedure shade_vert_seg; forward;

procedure shading_control (tmax, ix, trap_xleft, trap_xright,
                           arc_bot, arc_top : integer;
                           trap_slope_top, trap_slope_bot : real);
                           forward;

procedure tabout (translate_mode : integer); forward;
procedure open_files;
begin
    reset(atomfi);     (* ATOMSLLL input to DNI *)
    rewrite(debugfo); (* Debug information file *)
    rewrite(dicofo);  (* Output of DICOMED codes *)
    reset(tablefi);   (* File of measured film
                         densities
                       *)
end;
```

```
procedure get_options;
var resolution : integer;
begin
    with main_c, isp_c do
        begin
        writeln('Enter options for waiting, double film advance ');
        writeln('      - 0 to disable and 1 to enable. ');
        read(wait_flag, advance_flag);
        writeln('Enter debug options - output on file debugfo');
        writeln('       0 - no debug information');
        writeln('       1 - Procedure trace');
        writeln('       2 - Procedure trace and variable dump');
        writeln('         - WARNING : This produces huge amounts of ');
        writeln('                     output.');
        read(debug_option);
        wait_mode := wait_flag = 1;
        advance_mode := advance_flag = 1;
        if debug_option in [0..2] then
            case debug_option of
              0 : begin
                   debug_proc := false;
                   debug_var  := false;
                  end;
              1 : begin
                   debug_proc := true;
                   debug_var  := false;
                  end;
              2 : begin
                   debug_proc := true;
                   debug_var  := true;
                  end;
            end (* of case debug_option *)
        else begin
                debug_proc := false;
                debug_var  := false;
             end;
        writeln('Enter resolution : 1024, 2048 or 4096 ');
        read(resolution);
        pixel_size := 4096 div resolution;
        end;
end (*of procedure get options *);
```

```
procedure initialize;
    (*
      Initialize various constants and reset the DICOMED
      plus set up horizontal and vertical spacing.
    *)
var i,j : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Init');
    with color_c do
    begin
        for i := 1 to 7 do
            for j := 1 to 2 do
                begin
                color_min[i,j] := 0.01;
                color_range[i,j] := 0.99;
                end;
        color_range[7,1] := 0.45;
    end (*of with statement*) ;
    dico_c.buff_ptr := 1;
    with main_c, isp_c do
    begin
        frame_no := 0;
        dicowd(resett);
        set_table(sphere);
        shift_op := round(power(2.0,22.0));
        half_pixel := pixel_size div 2;
        vector_size := 8 * pixel_size;
        half_vector_size := 8 * half_pixel;
        real_vector_size := vector_size;
        real_half_vector := real_vector_size / 2.0;
        twice_vector := 2 * real_vector_size;
        two_sqr_vect := 2 * sqr(real_vector_size);
        min_slope := 0.1;
        screen_c.screen_top := 7 * 4096 - vector_size;
        screen_c.screen_bot := 4096;
        horz_space := point_select + (512 * pixel_size) + 8;
        vert_space := 512 * pixel_size + 8;
        back_exposure := pixel_size * 17;

        dicowd(resett);
        dicowd(raster_mode);
        dicowd(color_mode);
        dicowd(init_cond);
        dicowd(color_translate);
        dicowd(horz_space);
        dicowd(vert_space);
        dicowd(back_exposure);

        idum := 0;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Init');
end (* of procedure initialize *);
```

```
procedure read_buffer( buf_len : integer; var error : integer);
    (*
      Read a buffer of size bufsize (in const section) of input
      records.  These records are read according to the format
      created by ATOMSLLL (i.e. Fortran columnwise order) which
      runs across record boundaries for the Pascal records.
    *)
var i : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Read Buffer');
    for i := 1 to buf_len do
        read(atomfi, buffer[i].atom_bond_no);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].type_top_arc);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].type_bot_arc);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].xleft);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].xright);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].top_arc_xcent);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].top_arc_ycent);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].top_arc_radius);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].bot_arc_xcent);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].bot_arc_ycent);
    for i := 1 to buf_len do
        read(atomfi, buffer[i].bot_arc_radius);
    error := 0;
    if debug_proc then writeln(debugfo,'    Exit Read Buffer');
end (* of procedure read buffer *) ;
```

```
procedure set_table;
    (*
      The following procedure reads a set of measured film densities
      from a file.  These are then used to compute color look-up
      tables for color shading, highlights, and standard linear
      compensations.

      Parameters Passed :
         kind - Either a sphere or a cylinder
    *)
var density : array [1..32,1..7] of integer;
    film_density : array [1..32] of real;
    out_intensity : array [1..32] of integer;
    b1, c1, e1, b2, c2, e2 : real;
    max_fintensity, min_fintensity, range_fintensity : real;
    norm_k, desired_intensity, d1, d2, w1 : real;
    i, j, idum, m, k, l, mnt, md : integer;
    not_done : boolean;

    procedure compute_mnt;
    begin
        mnt := round (16.0 * ((desired_intensity - film_density[l-1]) *
                      (out_intensity[l] - out_intensity[l-1]) /
                      (w1 - film_density[l-1]) +
                       out_intensity[l-1] )
                     );
        if (k = 0) then mnt := 0;
        if (m = 1)  and  (kind = cylinder) then mnt := 4 * k;
    end;

begin
    if debug_proc then writeln(debugfo,'    Enter Set Table');
    b1 := 0.10;  c1 := 0.90;  e1 := 0.61;
    b2 := 0.08;  c2 := 0.39;  e2 := 14.0;
    if not (kind = cylinder) then
        begin
        for i := 1 to 32 do
            begin
            read(tablefi, out_intensity[i]);
            for j := 1 to 7 do read(tablefi, density[i,j]);
            end;
        read(tablefi, idum);
        end;
    if (idum >= 8000)
       then writeln(debugfo, '    Exposure Translation Table');
    dicowd(func_select);
    dicowd(load_trans_table + 2048);
    for m := 1 to 8 do
        begin
        j := m - 1;
        if ( j = 0 ) then j := 7;
        for i := 1 to 32 do
            film_density[i] := exp(ln(10.0) * (-density[i,j]/100.0));
        max_fintensity := film_density[32];
        min_fintensity := film_density[1];
        range_fintensity := max_fintensity - min_fintensity;
        (* Compute the desired intensity *)

        for k := 0 to 255 do
            begin
            norm_k := k / 255.0;
            if not (kind = sphere) then
                desired_intensity := min_fintensity +
                                     norm_k * range_fintensity
            else
                begin
                dl := bl + cl * power(norm_k,el);
                desired_intensity := min_fintensity +
                                     range_fintensity * dl;
                if (m = 1) then
                    begin
                    d2 := b2 * dl + c2 * ( 0.8 * power(norm_k,e2)
                          + 0.2 * power(norm_k,e2/4.0) );
                    desired_intensity := min_fintensity +
                                         range_fintensity * d2;
                    end;
                end;

            (* Now look for an intensity greater than the
                  desired one found above. *)

            l := 2;
            not_done := true;
            while ( l <= 32)  and  (not_done) do
                begin
                wl := film_density[l];
                if (wl > desired_intensity) then
                    begin
                    not_done := false;
                    compute_mnt;
                    end
                else
                    if (wl = desired_intensity) then
                        begin
                        mnt := out_intensity[l] * 16;
                        not_done := false;
                        end
                    else l := l + l;
                end (* of while loop *);

            if not_done then compute_mnt;
            dicowd(mnt);
            md := mnt div 16;
            if (idum >= 8000 + m) then
                writeln(debugfo, '     ', m:6, k:6, l:6, md:6, mnt:6,
                                         desired_intensity:11:4);
        end (* of for k := ... *) ;
     end (* of for m := ... *) ;
    empty_buffer;
    if debug_proc then writeln(debugfo,'    Exit Set Table');
 end (* of procedure set_table *);
```

```
procedure process_frame(i : integer);
    (*
      This routine is called when a frame has been finished, to
      perform some post-processing and resett the DICOMED film
      recorder.

      Parameters Passed :
         i - pointer to current input record.
    *)
begin
    if debug_proc then writeln(debugfo,'    Enter Frame');
    with main_c do
    begin
        buffer_start := i;
        dicowd(frame_advance);
        frame_no := frame_no + 1;
        writeln(' Finished frame ', frame_no:1);
        if wait_mode then
           begin
           empty_buffer;
           read(idum);    (* Used to cause a wait between frames *)
           end;
        if advance_mode then dicowd(frame_advance);
        dicowd(resett);
        dicowd(raster_mode);
        dicowd(color_mode);
        dicowd(init_cond);
        dicowd(color_translate);
        dicowd(horz_space);
        dicowd(vert_space);
        dicowd(back_exposure);
    end;
    if debug_proc then writeln(debugfo,'    Exit Frame');
end (* of procedure process_frame *);
```

```
procedure process_sphere;
    (*
      This procedure is called to initialize variables associated with
      processing trapezoids which are part of this sphere.
    *)
begin
    if debug_proc then writeln(debugfo,'    Enter Sphere');
    with main_c do
    begin
    with buffer[i], diff_c, isp_c, radius_c, sphere_c do
    begin

        (* Change color filter and compute 'a' for differencing
              and 'ad' for the number of points to plot. *)

        new_color := type_top_arc;
        if not (new_color = curr_color) then
            begin
            change_color(curr_color,new_color,a,range_col,shading);
            ad := a / shift_op;
            end;

        (* Load data into sphere record *)

        sphere_xcent := type_bot_arc;
        sphere_ycent := xleft;
        sphere_radius := xright;
        radius_sqr := sqr(sphere_radius);

        (* Compute constants for differencing scheme *)

        c := range_col * 255.0 * shift_op / radius_sqr;
        d := a + c * radius_sqr;
        e := c * vector_size * 2;
        f := c * sqr(vector_size);
        k3 := trunc(-2 * c * sqr(vector_size));
        kind := sphere;
    end;
    end;
    if debug_proc then writeln(debugfo,'    Exit Sphere');
end (* of procedure process_sphere *);
```

```
procedure process_cylinder;
    (* This procedure sets the values of the items in the bond record.
      These parameters will be used to compute scan line length for
      the following trapezoids.  A cylinder is defined by it's
      left and right edges and a middle line which bisects the
      two edges.  These are the lines refered to below.
    *)
var five : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Cylinder');
    five := 5;
    with main_c do
    begin
    with buffer[i], bond_c do
    begin

        (* Change color filter and compute 'a' for the
              differencing scheme. *)

        if not (new_color = 5) then
            change_color(five,five,a,range_col,shading);

        (* Compute values to process the upper line. *)

        top_line_typ := type_top_arc;
        top_slope := type_bot_arc / 32767.0;
        top_indicator := 0;
        if (not (top_line_typ = 1)) and (abs(top_slope) <= min_slope) then
            begin
            top_indicator := 1;
            top_line_typ := 3;
            end;
        top_intercept := xleft;
        if debug_var then
           writeln(debugfo, '    Top line type = ',top_line_typ:1,
                            ' Top slope = ',top_slope:10:4,
                            ' Top intercept = ',top_intercept:10:4);

        (* Compute values to process the bottom line. *)

        bot_line_typ := xright;
        bot_slope := top_arc_xcent / 32767.0;
        bot_indicator := 0;
        if (not (bot_line_typ = 1)) and (abs(bot_slope) <= min_slope) then
            begin
            bot_indicator := 1;
            bot_line_typ := 3;
            end;
        bot_intercept := top_arc_ycent;
        if debug_var then
           writeln(debugfo, '    Bot line type = ',bot_line_typ:1,
                            ' Bot slope = ',bot_slope:10:4,
                            ' Bot intercept = ',bot_intercept:10:4);
        (* Compute values to process the middle line. *)

        mid_line_typ := top_arc_radius;
        mid_slope := bot_arc_xcent / 32767.0;
        if (not (mid_line_typ = 1)) and (abs(mid_slope) <= min_slope) then
            begin
            top_indicator := 1;
            bot_indicator := 1;
            mid_line_typ := 3;
            end;
        mid_intercept := bot_arc_ycent;
        if debug_var then
           writeln(debugfo, '    Mid line type = ',mid_line_typ:1,
                            ' Mid slope = ',mid_slope:10:4,
                            ' Mid intercept = ',mid_intercept:10:4);
        if (bot_line_typ > 1) and (mid_line_typ > 1 )
            and ((bot_slope * mid_slope) < 0)
           then bot_indicator := 1;
        if (top_line_typ > 1) and (mid_line_typ > 1)
            and ( (top_slope * mid_slope) < 0 )
           then top_indicator := 1;
        if debug_var then
           writeln(debugfo, '    Top indicator = ',top_indicator:1,
                            ' Bot indicator = ',bot_indicator:1);

        (* Compute the constant 'c' for use in the
              differencing scheme. *)

        fm := bot_arc_radius / 32767.0;
        fs := 255.0 * fm * range_col;
        diff_c.c := fs * shift_op;
        if debug_var then
           writeln(debugfo, '    Diff_c.c = ',diff_c.c:10:4);
        kind := cylinder;
    end;
    end;
    if debug_proc then writeln(debugfo,'    Exit Cylinder');
end (* of procedure process_cylinder *);
```

```
procedure process_trapezoid;
    (*
      This procedure computes the control variables (i.e. xleft, xright
      values, etc.) to process the current trapezoids.
    *)
var ix1, ix2, ix3, ix4, ix, tmax :  integer;
    trap_xleft, trap_xright :       integer;
    arc_top, arc_bot :              integer;
    trap_slope_top, trap_slope_bot: real;
begin
    if debug_proc then writeln(debugfo,'    Enter Trapezoid');
    with main_c do
    begin
    with buffer[i], isp_c, radius_c, sphere_c, trap_c do
    begin

        (* Dump current input record *)

        if debug_var then
           begin
           writeln(debugfo, '    No. = ', atom_bond_no:1);
           writeln(debugfo, '    Top Arc = ', type_top_arc:1);
           writeln(debugfo, '    Bot Arc = ', type_bot_arc:1);
           writeln(debugfo, '    Xleft   = ', xleft:1);
           writeln(debugfo, '    Xright  = ', xright:1);
           writeln(debugfo, '    Top Xc  = ', top_arc_xcent:1);
           writeln(debugfo, '    Top Yc  = ', top_arc_ycent:1);
           writeln(debugfo, '    Top Rad = ', top_arc_radius:1);
           writeln(debugfo, '    Bot Xc  = ', bot_arc_xcent:1);
           writeln(debugfo, '    Bot Yc  = ', bot_arc_ycent:1);
           writeln(debugfo, '    Bot Rad = ', bot_arc_radius:1);
           end;

        (* Compute left and right edges of this trapezoid
              for x coordinates. *)

        ix1 := xleft + half_vector_size;
        ix3 := ix1 mod vector_size;
        if (ix3 < 0) then ix3 := ix3 + vector_size;
        trap_xleft := ix1 - ix3;
        ix2 := xright - half_vector_size;
        ix4 := ix2 mod vector_size;
        if (ix4 < 0) then ix4 := ix4 + vector_size;
        trap_xright := ix2 - ix4;
        if debug_var then
           begin
           writeln(debugfo, '     Ix1 = ',ix1:1, ' Ix3 = ',ix3:1,
                            ' Ix2 = ',ix2:1,' Ix4 = ',ix4:1);
           writeln(debugfo, '     Trap Xleft = ',trap_xleft:1,
                           'Trap Xright = ',trap_xright:1);
           end;
        arc_top := type_top_arc;
        arc_bot := type_bot_arc;
        if debug_var then
           writeln(debugfo, '     i = ',i:1,' kind = ',kind:1,
                            ' Arc top = ',arc_top:1,
                            ' Arc bot = ',arc_bot:1);
        if (trap_xright >= trap_xleft) then
            begin

            (* Compute values necessary to determine
                 the scan line length in the y direction. *)

            if (arc_top in [1,2]) then
                sqr_top_rad := sqr(top_arc_radius)
            else if (arc_top in [3,4]) then
                    trap_slope_top := top_arc_xcent / 32767.0
            else writeln('Process Trapezoid - Error in type ',
                         'of top arc = ', arc_top:5);

            if (arc_bot in [1,2]) then
                sqr_bot_rad := sqr(bot_arc_radius)
            else if (arc_bot in [3,4]) then
                    trap_slope_bot := bot_arc_xcent / 32767.0
            else writeln('Process Trapezoid - Error in type ',
                         'of bottom arc = ', arc_bot:5);


            (* Ix is the starting value of x for this trapezoid.
              Tmax is the number of pixels to be processed in x. *)

            ix := trap_xleft - vector_size;
            tmax := trunc( (trap_xright-trap_xleft) / vector_size) + 1;
            shading_control(tmax,ix,trap_xleft,trap_xright,arc_bot,
                            arc_top,trap_slope_top,trap_slope_bot);
            if debug_var then
               writeln(debugfo, '    Sphere Rad = ',sphere_radius:1,
                                ' Sphere Xc = ',sphere_xcent:1,
                                ' Sphere Yc = ',sphere_ycent:1,
                                ' C = ',diff_c.c:10:4);
        end (* of if xright >= xleft *);
    end;
    end;
    if debug_proc then writeln(debugfo,'    Exit Trapezoid');
end (* of procedure process_trapezoid *);
```

```
procedure shading_control;
    (*
      This routine determines how many vertical scan lines will be
      produced for the current trapezoid.  Also, the length of the
      vertical scan line is computed.  If a sphere is being processed,
      all necessary data is already available and shading may proceed.
      However, if a bond is being processed several further cases
      must be handled.

      Parameters Passed :
         tmax - number of points in x direction for shading.
         ix   - starting x value
         trap_xleft     - left x boundary value
         trap_xright    - right x boundary value
         arc_bot        - type of this trapezoids bottom arc
         arc_top        - type of this trapezoids top arc
         trap_slope_top - slope of top arc
         trap_slope_bot - slope of bottom arc
    *)
var temp, k1, k2 : integer;
    ysq  : real;
begin
    if debug_proc then writeln(debugfo,'    Enter Shading Control');
    with buffer[main_c.i], diff_c, isp_c, radius_c, sphere_c, trap_c do
    begin
        for temp := 1 to tmax do
            begin
            ix := ix + vector_size;
            if debug_var and (main_c.kind = cylinder) then
               writeln(debugfo, '    Tmax = ',tmax:1,' Ix = ',ix:1,
                                 ' Trap Tslope = ',trap_slope_top:10:4,
                                 ' Trap Bslope = ',trap_slope_bot:10:4
                         );
            x := ix + half_vector_size;

            (*Find top and bottom of quadratically shaded segment
                 according to the values of arc_top and arc_bot.
                 Explanation of these values can be found in the
                 front of the declaration of Bond_Rec. *)


            (* Compute the maximum y value for the current scan line *)

            if arc_top in [1..4] then
               case arc_top of
                1 : begin
                     ysq := sqr_top_rad - sqr(x - top_arc_xcent);
                     max_y_scan := top_arc_ycent + sqrt(abs(ysq));
                    end;
                2 : begin
                     ysq := sqr_top_rad - sqr(x - top_arc_xcent);
                     max_y_scan := top_arc_ycent - sqrt(abs(ysq));
                    end;
                3 : max_y_scan := trap_slope_top * x + top_arc_radius;
                4 : max_y_scan := (x - top_arc_radius) / trap_slope_top;
               end (* of case arc_top *)
            else writeln('Shading Control - Error in ',
                         'arc top = ', arc_top:1);
```

```
(* Compute the minimum y value for the current scan line *)

if arc_bot in [1..4] then
   case arc_bot of
    1 : begin
         ysq := sqr_bot_rad - sqr(x - bot_arc_xcent);
         min_y_scan := bot_arc_ycent + sqrt(abs(ysq));
        end;
    2 : begin
         ysq := sqr_bot_rad - sqr(x - bot_arc_xcent);
         min_y_scan := bot_arc_ycent - sqrt(abs(ysq));
        end;
    3 : min_y_scan := trap_slope_bot * x + bot_arc_radius;
    4 : min_y_scan := (x - bot_arc_radius) / trap_slope_bot;
   end (* of case arc_bot *)
else writeln('Shading Control - Error in ',
             'arc bot = ', arc_bot:1);

max_y_scan := max_y_scan + 16384;
min_y_scan := min_y_scan + 16384;

(* Top_y_bond and Bot_y_bond are rounded integer values
       of max_y_scan and min_y_scan. *)

top_y_bond := trunc((max_y_scan+half_vector_size)
              / vector_size) * vector_size - vector_size;
bot_y_bond := trunc((min_y_scan+half_vector_size)
              / vector_size) * vector_size;
no_pixels := trunc((top_y_bond - bot_y_bond) / vector_size)
             + 1;
if debug_var then
   begin
   writeln(debugfo,'    Min_y_scan = ',min_y_scan:10:4,
                   ' Max_y_scan = ',max_y_scan:10:4);
   writeln(debugfo,'    Bot_y_bond = ',bot_y_bond:1,
                   ' Top_y_bond = ',top_y_bond:1);
   writeln(debugfo,'    Ysq = ',ysq:10:4,
                   ' No pixels = ',no_pixels:1);
   end;
```

```
            if (no_pixels > 0) then
                begin

                (* Ready the DICOMED to begin drawing the
                      current trapezoid. *)

                dicowd(position_crt);
                dicowd(bot_y_bond);
                dicowd(ix + 16384);

                case main_c.kind of
                 sphere : begin

                            (* Plot the next scan line *)

                            dicowd(func_select);
                            dicowd(plot_line + no_pixels);
                            y := sphere_ycent - (bot_y_bond
                                 + half_vector_size - 16384);
                            k1 := trunc(d - c *
                                  (sqr(x - sphere_xcent) + sqr(y)));
                            k2 := trunc(e * y - f);
                            if debug_var then
                               writeln(debugfo,'    Sphere :  Kl = ',
                                                k1:1,' K2 = ',k2:1,
                                                ' K3 = ',k3:1);
                            quad(k1,k2,k3,no_pixels,sphere);
                          end;
                 cylinder : shade_cylinder(main_c.i);
                end (* of case kind *);

            end (* of no_pixels > 0 *);
        end (* of for loop *);
    end;
    if debug_proc then writeln(debugfo,'    Exit Shading Control');
end (* of procedure shading_control *);
```

```
procedure shade_cylinder;
    (*
      This procedure processes a trapezoid which is part of a bond.

      Parameters Passed :
         i - pointer to the current input record.
    *)
begin
    if debug_proc then writeln(debugfo,'    Enter Shade Cylinder');
    with trap_c, bond_c do
    begin
        bot_y_bond := bot_y_bond - 16384;
        top_y_bond := top_y_bond - 16384;
        y := bot_y_bond + isp_c.half_vector_size;
        if debug_var then
           writeln(debugfo,'    Top line type = ',top_line_typ:1,
                           ' Mid line type = ',mid_line_typ:1,
                           ' Bot line type = ',bot_line_typ:1);

        (* Compute parameters using data from last cylinder processed. *)


        (* Compute the y value on the top edge of the cylinder *)

        if top_line_typ in [1..3] then
            case top_line_typ of
             1 : max_y_scan := top_slope * x + top_intercept;
             2 : max_y_scan := (x - top_intercept) / top_slope;
             3 : (*Do nothing*) ;
            end (* of case *)
        else writeln('Shade Cylinder - Error in top_',
                     'line_typ = ', top_line_typ:1);


        (* Compute the y value on the bottom edge of the cylinder *)

        if bot_line_typ in [1..3] then
            case bot_line_typ of
             1 : min_y_scan := bot_slope * x + bot_intercept;
             2 : min_y_scan := (x - bot_intercept) / bot_slope;
             3 : (*Do nothing*) ;
            end (* of case *)
        else writeln('Shade Cylinder - Error in bot_',
                     'line_typ = ', bot_line_typ:3);
        (* Compute the y value on the line bisecting the cylinder *)

        if mid_line_typ in [1..3] then
            case mid_line_typ of
             1 : begin
                     mid_y_scan := mid_slope * x + mid_intercept;
                     continue_shading(i);
                 end;
             2 : begin
                     mid_y_scan := (x - mid_intercept) / mid_slope;
                     continue_shading(i);
                 end;
             3 : begin
                     no_pixels := trunc((top_y_bond - bot_y_bond)
                                  / isp_c.vector_size) + 1;
                     if (no_pixels > 0) then shade_vert_seg;
                 end;
            end (* of case *)
        else writeln('Shade Cylinder - Error in mid_',
                     'line_typ = ', mid_line_typ:3);

        if debug_proc then writeln(debugfo,'    Exit Shade Cylinder');
    end;
end (* of procedure shade_cylinder *) ;
```

```
procedure continue_shading;
    (*
      This procedure uses the data computed about the bond to determine
      which quadratics must be computed to produce the appropriate
      shading for this trapezoid.  Different quadratics may have to be
      used for different trapezoids on the same bond.  However, their
      shading must agree where they join.

      Parmeters Passed :
         i - pointer to the current input record.
    *)
var endflag : boolean;
begin
    if debug_proc then writeln(debugfo,'    Enter Continue Shading');
    with trap_c, isp_c, bond_c do
    begin
        endflag := false;
        mid_y_scan := amax( amin(mid_y_scan,32767.0), -32767.0);
        if debug_var then
           begin
           writeln(debugfo,'    I = ',i:1);
           writeln(debugfo,'    Bot y bond = ',bot_y_bond:1,
                           ' Top y bond = ',top_y_bond:1);
           writeln(debugfo,'    Min y scan = ',min_y_scan:10:4,
                           ' Mid y scan = ',mid_y_scan:10:4,
                           ' Max y scan = ',max_y_scan:10:4);
           end;
        mid_y_bond := trunc((mid_y_scan - real_half_vector)
                   / real_vector_size) * vector_size;

        (* Determine the orientation of the cylinder by the y values
              and compute the number of pixels to shade accordingly. *)

        if (bot_y_bond <= mid_y_bond) then
            begin
            if (top_y_bond < mid_y_bond) then
                no_pixels := trunc((top_y_bond - bot_y_bond)
                           / vector_size) + 1
            else
                no_pixels := trunc((mid_y_bond - bot_y_bond)
                           / vector_size) + 1;
            if (no_pixels <= 0) then
                endflag := true
            else
                begin
                if (bot_indicator = 1) then shade_low_horz
                else shade_low_vert;
                if (top_y_bond > mid_y_bond) then
                    no_pixels := trunc( (top_y_bond - mid_y_bond)
                                      / vector_size )
                else
                    endflag := true;
                end;
            end
        else
            no_pixels := trunc( (top_y_bond - bot_y_bond)
                       / vector_size) + 1;
        if not endflag then
            begin
            if (no_pixels > 0) then
                if (top_indicator = 1) then shade_hi_horz
                else shade_hi_vert;
            end;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Continue Shading');
end (* of procedure continue_shading *);
```

```
procedure shade_vert_seg;
    (*
      This routine computes the length of a horizontal scan line each
      time through the loop (i.e. x values) and shades a vertical
      segment using horizontal lines.
    *)
var j : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Shade Vert Seg');
    with trap_c, bond_c do
    begin
        dicowd(func_select);
        dicowd(plot_elements + no_pixels);
        for j := 1 to no_pixels do
            begin

            (* Y is constant and shading is along the horizontal line *)


            (* Compute x for left edge of cylinder *)

            if top_line_typ in [1..3] then
                case top_line_typ of
                   1 : xtop := (y - top_intercept) / top_slope;
                  2,3 : xtop := top_intercept + y * top_slope;
                end (* of case *)
            else  writeln('Shade Vert Seg - Error in type ',
                        'of top line = ', top_line_typ:1);


            (* Compute x for bisecting line of cylinder *)

            if mid_line_typ in [1..3] then
                 case mid_line_typ of
                   1 : xmid := (y - mid_intercept) / mid_slope;
                 2,3 : xmid := mid_intercept + y * mid_slope;
                end (* of case *)
            else  writeln('Shade Vert Seg - Error in type ',
                        'of middle line = ',mid_line_typ:1);


            (* Compute x for right edge of cylinder *)

            if bot_line_typ in [1..3] then
                case bot_line_typ of
                   1 : xbot := (y - bot_intercept) / bot_slope;
                  2,3 : xbot := bot_intercept + y * bot_slope;
                end (* of case *)
            else writeln('Shade Vert Seg - Error in type ',
                       'of bottom line = ', bot_line_typ:1);

            if ((x - xmid) * (xbot - xmid) < 0) then
                x_exposure := trunc(fs * (1.0 - sqr((x-xmid)
                                     /(xtop-xmid))) + main_c.ad)
            else
                x_exposure := trunc(fs * (1.0 - sqr((x-xmid)
                                     /(xbot-xmid))) + main_c.ad);
            dicowd(plot_horz + x_exposure);
            y := trunc(y + isp_c.real_vector_size);
        end (* of for loop *);
```

```
         if debug_var then
            begin
            writeln(debugfo,'    X exposure = ',x_exposure:1);
            writeln(debugfo,'    Xbot = ',xbot:10:2,
                            ' Xmid = ',xmid:10:2,
                            ' Xtop = ',xtop:10:2, ' X = ',x:1);
            writeln(debugfo,'    Fs = ',fs:10:2,
                            ' Ad = ', main_c.ad:10:2);
            end;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Shade Vert Seg');
end (* of procedure shade_vert_seg *);
```

```
procedure shade_low_vert;
    (*
      This procedure computes the first value, the first difference,
      and the second difference for the quadratic equation and
      initializes the shading for vertical scan lines below the
      highlight line.
    *)
var k1, k2 : integer;
    yd, cb : real;
begin
    if debug_proc then writeln(debugfo,'    Enter Shade Low Vert');
    with trap_c, isp_c, diff_c do
    begin
        yd := y - mid_y_scan;
        cb := c / sqr(mid_y_scan - min_y_scan);
        k1 := trunc(c - cb * sqr(yd) + main_c.a);
        k2 := trunc(-cb * twice_vector * (yd + real_vector_size));
        k3 := trunc(-cb * two_sqr_vect);
        if debug_var then
           writeln(debugfo, '    No pixels = ',no_pixels:1,
                            ' K1 = ',k1:1, ' K2 = ',k2:1,
                            ' K3 = ',k3:1);
        dicowd(func_select);
        dicowd(plot_line + no_pixels);
        quad(k1,k2,k3,no_pixels,cylinder);
        y := y + no_pixels * vector_size;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Shade Low Vert');
end (* of procedure shade_low_vert *);
```

```
procedure shade_low_horz;
    (*
      This procedure computes the length of a horizontal scan line
      each time through the loop and shades below the highlight line
      using horizontal lines.  Shading is done from the bisecting
      line of the cylinder to it's right edge.
    *)
var j : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Shade Low Horz');
    with trap_c, bond_c do
    begin
        dicowd(func_select);
        dicowd(plot_elements + no_pixels);
        for j := 1 to no_pixels do
            begin


            (* Compute x for the right edge of the cylinder *)

            if bot_line_typ in [1..3] then
                 case bot_line_typ of
                   1 : xbot := (y - bot_intercept) / bot_slope;
                 2,3 : xbot := bot_intercept + y * bot_slope;
                end (* of case *)
            else writeln('Shade Low Horz - Error in type ',
                         'of bottom line = ', bot_line_typ:1);


            (* Compute x for the bisecting line of the cylinder *)

            if mid_line_typ in [1..3] then
                case mid_line_typ of
                   1 : xmid := (y - mid_intercept) / mid_slope;
                 2,3 : xmid := mid_intercept + y * mid_slope;
                end (* of case *)
            else writeln('Shade Low Horz - Error in type ',
                         'of middle line = ', mid_line_typ:1);

            x_exposure := trunc(fs * (1.0 - sqr((x-xmid)/(xbot-xmid)))
                                   + main_c.ad);
            x_exposure := max_int( min_int(x_exposure,255), 0 );
            dicowd(plot_horz + x_exposure);
            y := y + trunc(isp_c.real_vector_size);
        end (* of for loop *);
        if debug_var then
           begin
           writeln(debugfo,'    X exposure = ',x_exposure:1);
           writeln(debugfo,'    Xbot = ',xbot:10:2,
                           ' Xmid = ',xmid:10:2,
                           ' Xtop = ',xtop:10:2, ' X = ',x:1);
           writeln(debugfo,'    Fs = ',fs:10:2,
                           ' Ad = ', main_c.ad:10:2);
           end;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Shade Low Horz');
end (* of procedure shade_low_horz *);
```

```
procedure shade_hi_vert;
    (*
      This routine computes the first value, the first difference, and
      the second difference of the quadratic equation and initializes
      the shading for vertical scan lines above the highlight line.
    *)
var yd, ct : real;
    kl, k2 : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Shade Hi Vert');
    with trap_c, diff_c, isp_c do
    begin
        yd := y - mid_y_scan;
        ct := c / sqr(mid_y_scan - max_y_scan);
        kl := trunc(c - ct * sqr(yd) + main_c.a);
        k2 := trunc(-ct * twice_vector * (yd + real_vector_size));
        k3 := trunc(-ct * two_sqr_vect);
        if debug_var then
           writeln(debugfo, '    No pixels = ',no_pixels:1,
                            ' Kl = ',kl:1, ' K2 = ',k2:1,
                            ' K3 = ',k3:1);
        dicowd(func_select);
        dicowd(plot_line + no_pixels);
        quad(kl,k2,k3,no_pixels,shade_hi_v_call);
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Shade Hi Vert');
end (* of procedure shade_hi_vert *);
```

```
procedure shade_hi_horz;
    (*
      This procedure computes the length of a horizontal scan line each
      time through the loop and shades above the highlight line using
      horizontal lines.  Shading is done from the left edge to
      the bisecting line of the cylinder.
    *)
var j : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Shade Hi Horz');
    with trap_c, bond_c do
    begin
        dicowd( func_select );
        dicowd( plot_elements + no_pixels );
        for j := 1 to no_pixels do
            begin


            (* Compute x for the left edge of the cylinder *)

            if top_line_typ in [1..3] then
                case top_line_typ of
                 1,2 : xtop := (y - top_intercept) / top_slope;
                   3 : xtop := top_intercept + y * top_slope;
                end (* of case *)
            else writeln('Shade Hi Horz - Error in type ',
                         'of top line = ', top_line_typ:1);


            (* Compute x for the bisecting line of the cylinder *)

            if mid_line_typ in [1..3] then
                case mid_line_typ of
                   1 : xmid := (y - mid_intercept) / mid_slope;
                 2,3 : xmid := mid_intercept + y * mid_slope;
                end (* of case *)
            else  writeln('Shade Hi Horz - Error in type ',
                          'of middle line = ',mid_line_typ:1);

            x_exposure := trunc(fs * (1.0 - sqr((x-xmid)/(xtop-xmid)))
                                + main_c.ad);
            x_exposure := max_int( min_int(x_exposure,255), 0 );
            dicowd(plot_horz + x_exposure);
            y := y + trunc(isp_c.real_vector_size);
        end (* of for loop *);
        if debug_var then
           begin
           writeln(debugfo,'    X exposure = ',x_exposure:1);
           writeln(debugfo,'    Xbot = ',xbot:10:2,
                           ' Xmid = ',xmid:10:2,
                           ' Xtop = ',xtop:10:2, ' X = ',x:1);
           writeln(debugfo,'    Fs = ',fs:10:2,
                           ' Ad = ', main_c.ad:10:2);
           end;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Shade Hi Horz');
end (* of procedure shade_hi_horz *);
```

```
procedure dicowd;
    (*
      This routine buffers the output to the DICOMED from the program
      and handles the buffer manipulation.

      Parmeters Passed :
         iword - contains DICOMED code
    *)
var word, i : integer;
begin
    with dico_c do
    begin
        word := iword;
        if (buff_ptr > buff_len) then
            begin
            for i := 1 to buff_len do write(dicofo,' ',bufferl[i]:1);
            buff_ptr := 1;
            end;
        bufferl[buff_ptr] := word;
        buff_ptr := buff_ptr + 1;
    end (* of with statement *);
end (* of procedure dicowd *);
```

```
procedure quad;
    (*
      Quad computes 8 bit exposure codes and concatenates pairs of
      these to produce 16 bit words for output to the DICOMED.

      Parameters Passed :
         kl - initial value of the quadratic equation
         k2 - first difference of the quadratic equation
         k3 - second difference of the quadratic equation
         nw - number of pixels to generate exposure codes for
         icall - debugging parameter, identifies calling routine
                 3 : called from shading control
                 4 : called from shade low vert
                 5 : called from shade hi vert
    *)
var limit, upper, lower, i : integer;
begin
    limit := (nw + 1) div 2;
    for i := 1 to limit do
    begin
        upper := kl div main_c.shift_op;
        kl := kl + k2;
        k2 := k2 + k3;
        lower := kl div main_c.shift_op;
        kl := kl + k2;
        k2 := k2 + k3;
        dicowd(256*upper + lower);
    end (* of for loop *);
end (* of procedure quad *);
```

```
procedure change_color;
     (*
       This routine changes the color filter for the DICOMED and
       computes some constants related to color shading.

       Parameters Passed :
          curr_color - currently selected color filter
          new_color  - new selection for color filter
          a          - used to compute initial value of quadratic eqn.
          bp         - contains limiting range for color exposure code
          shading    - either color or black and white
     *)
var iadd : integer;
    ap : real;
begin
    if debug_proc then writeln(debugfo,'    Enter Change Color');
    with color_c do
    begin
        curr_color := new_color;
        ap := color_min[new_color,shading];
        bp := color_range[new_color,shading];
        a := ap * 255 * main_c.shift_op;
        iadd := 8 + new_color;
        if (shading = b_w) then dicowd(neutral_filt)
        else dicowd(iadd);
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Change Color');
end (* of procedure change_color *);
```

```
procedure tabout;
    (*
      This procedure sends a code to the DICOMED for either color or
      black and white shading.

      Parameters Passed :
         translate_mode - either color or black and white
    *)
begin
    if debug_proc then writeln(debugfo,'    Enter Tabout');
    if translate_mode in [b_w,color] then
        case translate_mode of
         color : dicowd(color_translate);
         b_w   : dicowd(b_w_translate);
        end (* of case *)
    else writeln('Possible error in Tabout ');
    if debug_proc then writeln(debugfo,'    Exit Tabout');
end (* of procedure tabout *);
```

```
procedure empty_buffer;
    (*
      This routine empties the remainder of the DICOMED buffer.
    *)
var i, limit : integer;
begin
    if debug_proc then writeln(debugfo,'    Enter Empty Buffer');
    with dico_c do
    begin
        if not (buff_ptr = 1) then
            begin
            limit := buff_ptr - 1;
            for i := 1 to limit do write(dicofo,' ',buffer1[i]:1);
            buff_ptr := 1;
            end;
    end (* of with statement *);
    if debug_proc then writeln(debugfo,'    Exit Empty Buffer');
end (* of procedure empty_buffer *);
```

```
function power;
    (*
      Compute the value of 'no' raised to the power 'exponent'
    *)
var abs_power : real;
begin
    if no = 0 then abs_power := 0
    else abs_power := exp( ln(abs(no)) * exponent );
    if (no < 0) and (trunc(exponent) = exponent)
    then if odd(trunc(exponent))
         then abs_power := -abs_power;
    power := abs_power;
end;
```

```
function amax;
     (*
       A type real maximum function.
       Both first and second are real values.
     *)
begin
   if first > second
       then amax := first
       else amax := second;
end;
```

```
function amin;
     (*
       A type real minimum function.
       Both first and second are real values.
     *)
begin
   if first < second
      then amin := first
      else amin := second;
end;
```

```
function max_int;
    (*
      A type integer maximum function.
      First and second are integer values.
    *)
begin
    if first > second
       then max_int := first
       else max_int := second;
end;
```

```
function min_int;
     (*
       A type integer minimum function.
       First and second are integer values.
     *)
begin
   if first < second
      then min_int := first
      else min_int := second;
end;
```

```
procedure close_files;
begin
    reset(atomfi);
    reset(debugfo);
    reset(dicofo);
    reset(tablefi);
end;
```

```
(* Main Program
    Contains the control structure which processes the input on a
    record by record basis for a buffer of size, bufsize.  Each buffer
    is initially processed for color shading and then, it is
    reprocessed for the highlights.
*)


begin
    with main_c do
    begin
        writeln('   Program Start');
        open_files;
        get_options;
        Initialize;
        eoj := false;
        shading := color;
        while not eoj do
            begin
            read_buffer(bufsize, ier);
            buffer_start := 0;
            this_buffer := true;

            (* Process the current buffer *)

            while this_buffer do
                begin
                tabout(shading);
                curr_color := 0;
                i := buffer_start + 1;
                not_done := true;
```

```
(* Process the current record *)

while (i <= bufsize) and (not_done) do
    begin
    if (buffer[i].atom_bond_no > 0) then
        processing := trapezoid
    else
        processing := buffer[i].atom_bond_no + 6;

    (* Call appropriate routines to process this record *)

    if processing in [sphere,cylinder,trapezoid,frame,
                      end_of_job] then
       case processing of
        sphere : begin
                   writeln(debugfo,'*** New Sphere');
                   writeln('*** New Sphere');
                   process_sphere;
                   writeln('*** End Sphere');
                  end;
        cylinder : begin
                     writeln(debugfo,'*** New Cylinder');
                     writeln('*** New Cylinder');
                     process_cylinder;
                     writeln('*** End Cylinder');
                    end;
        trapezoid : begin
                      writeln(debugfo,'*** New Trap');
                      writeln('*** New Trapezoid');
                      process_trapezoid;
                      writeln('*** End Trapezoid');
                     end;
        frame : begin
                  writeln(debugfo,'*** End of Frame');
                  writeln('*** End of Frame');
                  if (shading = b_w) then
                      begin
                      process_frame(i);
                      if (buffer_start >= bufsize) then
                          this_buffer := false;
                      end;
                  not_done := false;
                  writeln('*** End Frame');
                 end;
        end_of_job : begin
                       writeln(debugfo,'*** End of Job');
                       writeln('*** End of Job');
                       not_done := false;
                       this_buffer := false;
                       eoj := true;
                       writeln('*** End of Job');
                      end;
       end (*of case Processing*)
    else writeln('Main Program - Processing error ');
    i := i + 1;
    if ( i > bufsize) and (shading = b_w)
       then this_buffer := false;
end (* of while i <= bufsize ... *) ;
```

```
                  (* Change the processing switch to do highlights after
                       color or color after highlights. *)

                  case shading of
                      color : shading := b_w;
                      b_w   : shading := color;
                  end (*of case shading*) ;

              end (* of while this_buffer *);
          end (* of while not eoj *);
          empty_buffer;
      end;
      close_files;
  end. (* of program *)
```