

63
104

PHIGS Based Phong Rendering Emulation Software

by

Krishnan V. Kolady

thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

APPROVED:



Dr. Arvid Myklebust, Chairman



Dr. S. Jayaram



Dr. M. P. Deisenroth

May 29, 1990

Blacksburg, Virginia

C.2

LD
5655
V855
1990
K453
C.2

PHIGS Based Phong Rendering Emulation Software

by

Krishnan V. Kolady

Dr. Arvid Myklebust, Chairman

Mechanical Engineering

(ABSTRACT)

Discussed is the design, implementation and use of a graPHIGS (IBM PHIGS) based sub-system that provides for shading of graphical models using the Phong shading technique. The ISO standard for 3D graphics, PHIGS, provides for wireframe display and manipulation of graphics data. PHIGS+ implementations, while providing this capability, will not be widely available for some time. This capability will provide a generally useful extension to PHIGS for use by PHIGS based applications. The software provides the applications programmer with a graPHIGS based instruction set which acts as a superset to the current graPHIGS calls. Using the provided functions the user can quickly do hidden surface elimination and Phong rendering of 3-D models in 3-D views. The program contains approximately 15,000 lines of C code and uses graPHIGS inquiries and calls for information retrieval and datastructure maintenance.

Acknowledgements

First of all, I would like to thank Dr. Jayaram for his support and guidance he provided throughout the project. I would like to thank my advisor Dr. Myklebust for his constant optimism and encouragement. Thanks are also extended to Dr. Deisenroth.

I wish to thank J. R. Gloudemans and the rest of the crew for making my workload as little as possible during the crunch time.

Thanks to IBM Graphics Products Division in Kingston, N.Y., for funding this study.

Finally, I thank my mother and father for their love and support.

Table of Contents

- 1.0 Introduction 1**
- 1.1 Problem definition and Objectives 2**
- 1.2 Background 3**
- 1.3 Thesis Organization 4**

- 2.0 Literature survey 6**

- 3.0 Design Considerations 10**
- 3.1 GraPHIGS+ considerations 11**
 - 3.1.1 GraPHIGS+ geometric primitives 11**
 - 3.1.2 Color model 12**
 - 3.1.3 Geometric normal 12**
 - 3.1.4 Back-face processing 13**
 - 3.1.5 Light sources 13**
- 3.2 Visible surface algorithm 14**

- 4.0 User requirements 16**

5.0	Functional Requirements	19
6.0	Software Functional Description	22
6.1.1.1	Modified graPHIGS + update	23
6.1.1.2	Polygon 3 with data	23
6.1.1.3	Compute polygon 3 geometric normal	24
6.1.1.4	Set depth cue mode	25
6.1.1.5	Set area properties	25
6.1.1.6	Set back area properties	26
6.1.1.7	Set light source state	27
6.1.1.8	Set face distinguishing mode	27
6.1.1.9	Set face culling mode	28
6.1.1.10	Set back interior color	29
6.1.1.11	Set light source representation	29
6.1.1.12	Set hlhsr mode	30
7.0	Detailed design	31
7.1	graPHIGS extension	32
7.1.1.1	PHIGS + routines and data structures	34
7.1.1.2	Retrieving view and structure information	35
7.2	Visibility detection	38
7.3	Phong lighting and shading methods	41
7.3.1.1	Lighting models	41
7.3.1.2	Shading calculations	43
8.0	Implementation	50
8.1	graPHIGS extension	51
8.1.1.1	PHIGS + routines and data structures	51

8.1.1.2	Retrieving view and structure information	52
8.2	Visibility detection	63
8.3	Phong lighting and shading methods	67
9.0	Results and Future Work	69
10.0	References	71
Appendix A.	Flowcharts	74
Vita	240

List of Illustrations

Figure 1. Schematic of graPHIGS internal data structure	36
Figure 2. Tree structure for structure hierarchy manipulation	37
Figure 3. Spot light sources.	44
Figure 4. Vertex normal interpolation on scanline	45
Figure 5. Phong's reflectance model	47
Figure 6. Linked list data structure to maintain Phong shading structures	56
Figure 7. Binary tree data structure for structure hierarchy	57
Figure 8. Attribute tree data structure	58
Figure 9. Cross-referencing nodes of the two data structures	59
Figure 10. Face tree data structure	61
Figure 11. Data structures and cross-referencing	62
Figure 12. Edge preprocessing to maintain parity at vertices	65
Figure 13. Type definition for the edge and face list.	66

1.0 Introduction

The current version of graPHIGS (the IBM version of PHIGS graphics standard) allows the applications programmer to create and manipulate 3-D wireframe models with ease. But there is no easy method for the applications programmer to view 3-D models with hidden surface and shading applied to the faces. This thesis describes the design and implementation of a Phong rendering emulation software based on PHIGS for IBM 5080 graphics devices.

The initial releases of graPHIGS (prior to version 2.0) did not support shading and advanced geometric modeling primitives (example: NURBS, parametric surfaces, etc.). graPHIGS version 2.0 includes these advanced features, but they are supported only on the new IBM 6090 platforms (in hardware). Thus, for the benefit of the users of graPHIGS on the IBM 5080 devices, these features could be implemented as support software which emulates the performance of the 6090 features.

1.1 Problem definition and Objectives

The objective of this research and development effort is to create a device independent rendering system which relies on `grAPHIGS` (IBM `PHIGS`) for support. This system is to be added on as a superset to the current `grAPHIGS` instruction set and allow `grAPHIGS` users to render scenes using Phong shading methods. The software should be able to function on the currently available IBM 5080 terminals (with 128 colors and no hardware z-buffer).

The current `grAPHIGS` instruction set does not have any functions to support the rendering pipeline, for example, advanced output primitives to include normals and colors at the vertices of the 3-D polygons, parametric surfaces, lighting and hidden surface processing, etc.

Thus, the proposed system must incorporate all these new “`grAPHIGS +`” requirements and must supply the user with the appropriate functions. (These are discussed in detail in the appendix on Software Functional Description.) The system must also maintain a software frame buffer (virtual frame buffer) for processing the image.

Finally, the system must include a fast and precise rendering scheme using Phong’s model for the lighting and shading computations. The model created after the computations must be displayed using currently available `grAPHIGS` primitives.

Different aspects of the design considerations are discussed in the forthcoming chapters.

1.2 Background

The CAD/CAM laboratory at VPI & SU has been actively using graPHIGS since the first release of graPHIGS in 1985. graPHIGS forms a major part of a graduate course in the Mechanical Engineering department. graPHIGS is used extensively for various research projects in the CAD/CAM laboratory. In 1987, a research effort was started at VPI & SU (through a grant from NASA - Ames Research Center) to create an interactive, system-independent CAD system for conceptual design of aircraft (ACSYNT).

Recently ACSYNT Institute was created at VPI by NASA-Ames and Amtek. Presently eight aerospace companies and several government agencies have joined the Institute and will be using this software.

PHIGS was chosen as the graphics software to be used for this research effort and the IBM VM/CMS system with graPHIGS was chosen to be the code development platform. Since PHIGS+ (with advanced rendering features) implementations were not available at that time, a system for generating constant-shaded images was created based on graPHIGS. This shading software, although limited in its capabilities, demonstrated the feasibility and importance of having advanced shading functions for the IBM 5080, based graPHIGS. The Phong shading method developed in this thesis is intended to improve the rendering process in ACSYNT.

The work described in this thesis was funded by an IBM research grant and the prototype developed is part of the deliverables of the research project.

1.3 Thesis Organization

In this thesis a detailed description of the design for the **grAPHIGS** based Phong rendering software is presented. The organization of this thesis breaks the project down into the following sections,

- 1) Literature survey,
- 2) Overall design considerations,
- 3) User requirements,
- 4) Functional requirements,
- 5) Detailed design
- 6) Implementation,
- 7) Results and future work, and
- 8) Appendices including the flowcharts.

It is hoped that this structure will allow various types of readers to quickly find the material of interest. Those readers who are interested in learning to use the application programmer interface program should refer to the section on Software Functional Requirements. Finally, readers with an overall interest in this project should refer to the section on detailed design.

For convenience, in this report the terms “PHIGS” and “PHIGS + ” are used to refer to the ISO standard and the term “**grAPHIGS +** ” is used to refer to **grAPHIGS** including the planned superset for Phong shading emulation. PHIGS nomenclature and bindings have been used through out to maintain consistency with the standard.

The software requirements in the report include some of the advanced geometric modeling features (curves, surfaces, etc.) which have already been implemented in **grAPHIGS** version 2.0. These features are included in this report because the rendering system has been designed such that if these

modeling features are supported by IBM 5080 devices in the future, the new primitives can also be shaded using this rendering system without any change to the application software.

2.0 Literature survey

Gouraud [Gour71] was one of the first to work on the idea of smooth shaded images of curved surfaces. The surface was tessellated into small polygons to take care of hidden surface problems. The technique used to shade the polygons were to find the intensities at the polygon vertices and interpolate the calculated intensity over the whole patch. Even though Gouraud worked on this more than two decades ago, most high-end workstations in the market today employ his technique (with built-in hardware) to do the shading of tessellated surfaces. Problems faced by the Gouraud shading method were the Mach band effect and the lack of specular reflections in the interior of the polygons.

Catmull [Catm74, Catm75] worked on a totally different approach of shading parametric surfaces. He found computationally inexpensive methods to subdivide a surface down to the pixel level and then render them. Even though this technique was very compute intensive and did not gain much popularity, Catmull was the first to introduce the concept of z-buffer which is today implemented in hardware in most graphics workstations.

BuiTuong Phong [Phon75] later extended Gouraud's work to get better specular reflections on the surface. Instead of interpolating the color over the polygon, Phong chose to interpolate the normal to the surface at each point of the polygon and then calculate the intensity at that point. This technique reduced the optical illusion of the Mach band effect considerably. For his lighting model, Phong used the laws of optics to produce better rendered scenes. Phong's initial lighting model has been improved by others to include better light source definitions. Even today, though Phong's shading model is too expensive to be implemented in real time, "Gouraud shading with Phong lighting" (Phong's improved lighting model), is a commonly used shading model.

Tom Duff [Duff79] analyzed the works of Gouraud and Phong and presented a faster mathematical model to calculate the Phong shading intensities. Both Gouraud and Phong shading suffer from the defect that if the scene (together with the light sources) is rotated along an axis perpendicular to the viewing window, the shading changes since both these shading techniques employ scan line interpolation. Duff presented a rotation independent shading method.

Several people worked on improving the illumination model to improve the shadow, reflection and refraction effects simulated by the shading model. Blinn [Blin77] introduced shading techniques where the specular highlights were dependent on the light direction, thereby improving the reflection model. The surface to be shaded was simulated by a collection of mirror-like microfacets oriented in random directions. The reflections from these were approximated to find the specular reflection. Cook [Cook82] presents work on generating realistic color models depending on the material of the surface and the light sources. He accounts for the color shifts with change of the reflectance vector. He presents results with different renderings of metallic surfaces. Most shading calculations were done on a local scale, since the visible surface algorithm could not present the global information to the shader. Whitted [Whit80] presented a technique to store the global lighting information in a tree structure for each pixel which is then used by the shader. This method accurately simulates the effects of shadows, reflections and refractions. Hall and Greenberg [Hall83] further improved on the illumination model presented by Whitted by including the Fresnel

relationships for wavelength and angle of incidence dependence of the transmitted and reflected light.

Visible surface or hidden surface algorithms form an initial and inherent part of any shading process. Bouknight [Bouk70] was among the first to present scan line visible surface algorithms. He presents a detailed discussion on the y-sorted and x-sorted list. He also uses coherence to improve the algorithm. Sutherland et. al. [Suth74] presented one of the first papers on comparison of different hidden surface techniques. The paper presents a detailed study of ten of the earliest visible surface algorithms. The paper attempts to understand the similarities and differences of the various visible surface algorithms and their operation. It discusses the hidden surface problem from the point of view of "sorting". It shows that the order of sorting and the types of sorting used, formed the major differences among the different algorithms. Finally it also gives a comparative study of the various algorithms.

Franklin Crow [Crow77] classifies the shadow algorithms for scan line visible surface algorithms into three: shadow computation during scanout, two-pass approach and projected shadow polygons. He gives a comparative study of the three techniques.

Lane and Carpenter [Lane79] extend the scan line algorithms for shading parametrically defined surfaces. The authors subdivide the surface to a certain degree of flatness before rendering it. James Clark [Clar79] presents an improved subdivision technique to subdivide the surface faster and improve the checks on the flatness of the edge before subdivision. This is used to avoid the "crack" problem encountered when neighboring patches are not subdivided equally. Lane et. al. [Lane80] presented a survey of three different techniques to render parametric surfaces, based on edge insertion, edge tracking and subdivision of patches. Schweitzer and Cobb [Schw82] present a technique to render bicubic patches without producing polygonal approximation of the surfaces. Shades are computed by calculating a cubic approximation to the normal surface, and interpolating along the scan line.

Whitted [Whit82] presents a software test-bed system in which the model data is initially converted and routed through a structure called a "span buffer" which retains some of the high-resolution, three-dimensional data of the object description. This structure is then post-processed to produce many different imaging effects. Fiume [Fium83] presents an architecture for a general-purpose ultracomputer whose "serial semantics/parallel execution" feature can be exploited in the formulation of a scan conversion algorithm. Crocker [Croc84] improves on the efficiency of the scan-line hidden surface algorithm by using invisibility coherence. Crocker suggests that often a large portion of any 3-D scene is invisible. Crocker maintains a minimum visible z-depth which categorizes the objects as visible and invisible, and a few final checks confirm this. The invisible objects are not updated, saving considerable compute time. Nelson Max [Max86] uses an adaptation of the shadow volume technique by Crow to apply to atmospheric illumination and shadows.

Although much work has been done on dithering and display on bilevel displays, the literature is sparse on color quantization and color dithering. Heckbert [Heck82] introduces concepts of color quantization based on the popularity algorithm and median cut algorithm which is the basis to choose the best colors for the look-up table. Heckbert uses a three pass process for the color quantization; namely determining the color distribution, selecting the colors for the look-up table, and redrawing the image.

3.0 Design Considerations

There are two major considerations in the design of this rendering system. The first is the `grAPHIGS+` add-on to the current `grAPHIGS` instruction set and the structure hierarchy manipulation of the current `grAPHIGS` data structure. The second is the actual computation and display of the shaded image.

In the first part of the design, care should be taken to smoothly integrate the current `grAPHIGS` instruction set with the new `grAPHIGS+` calls. Moreover, the structure operations performed by this rendering software should not affect the normal processing of the `grAPHIGS` structure hierarchy and structure manipulation.

In the second part, the emphasis is on creating a fast and precise algorithm for the shading computations using Phong's lighting and shading models. Since the visible surface or hidden surface algorithm is usually the most "compute intensive" process in any rendering pipeline, a fast and suitable visibility detection algorithm is also required.

3.1 *GraPHIGS+ considerations*

Since the objective is to design a set of functions which will add-on to the original set of graPHIGS calls, it is of utmost importance, to not change the effect of the original graPHIGS calls in any way.

Some of the other important aspects of the design which need to be considered are:

- GraPHIGS + geometric primitives
- Color model
- Geometric normal
- Back-face processing
- Light sources
- Visible surface algorithms

3.1.1 GraPHIGS + geometric primitives

GraPHIGS + should augment the set of graPHIGS output primitives by introducing a set of higher level output primitives. These primitives will allow for combinations of vertex normals, vertex colors, facet normals, facet colors and edge flags to be specified. Some of the primitives that PHIGS + allows for are:

- Polyline set 3 with data
- Fill area 3 with data (Polygon 3 with data)
- Fill area set 3 with data (Polygon set 3 with data)
- Extended cell array 3 (Extended 3-D pixel primitive)

- Triangular strip 3 with data
- Quadrilateral mesh 3 with data
- Polyhedron 3 with data
- Non-uniform B-spline curve
- Parametric polynomial curve
- Non-uniform B-spline surface
- Parametric polynomial surface

All these geometric primitives should eventually be supported by the `graPHIGS+` instruction set. Since the emphasis in this project is on the Phong rendering scheme, only one `PHIGS+` output primitive (Polygon 3 with data) has been considered in this design. For all the other primitives the same design can be extended without major changes.

3.1.2 Color model

`GraPHIGS` supports four color models: RGB, CIE, HSV and HLS. In the development of this design the color model has been restricted to RGB. Utility routines for conversion between RGB and CIE will be provided.

3.1.3 Geometric normal

The geometric normal is used for determining if a facet of an area defining primitive is back-facing or front-facing. Facet normals, if supplied, are used as the geometric normal. If facet normals are

not supplied with the primitive, a geometric normal is computed for each facet. For a Polygon 3 with data, the cross product of the vectors between the first three points is to be used as the geometric normal.

3.1.4 Back-face processing

PHIGS+ provides control over back-face area defining primitives via the "SET FACE DISTINGUISHING MODE" structure element. For the purpose of front/back determination test, area defining primitives are divided into facets. The geometric normal of the facet is then used as the deciding factor in the test.

For the purpose of this design, the distinguishing attribute between the front and the back-face polygons is restricted to the polygon interior color index attribute. Thus the user can specify a different color for the back-faces to distinguish them on the screen.

3.1.5 Light sources

PHIGS+ allows the light sources to be one of the following types:

- **AMBIENT:** Ambient light sources have a light source color and affect an area defining primitive independently of the orientation and position of the primitive.

- **DIRECTIONAL:** Directional light sources have a light source color and light source direction. Conceptually they are located at infinity. They affect an area defining primitive based on the primitive's orientation, but independently of its position.
- **POSITIONAL:** Positional light sources have a light source color, light source position and attenuation coefficients. They affect an area defining primitive based on the primitive's position and orientation.
- **SPOT:** Spot light sources have a light source color, light source position, light source direction, attenuation coefficients, concentration exponent, and spread angle. When lighting a particular point, it is intended that light source intensity is scaled by the cosine, raised to the concentration exponent, of the angle between the light source direction and the vector from light source position to the point being lit. If the point lies outside of the cone of influence, its color is not affected by the reflectance calculation for this light source.

3.2 Visible surface algorithm

Visible surface algorithms or hidden surface algorithms can be classified in several ways. They can be classified broadly into (a) continuous algorithms and (b) point-sampled algorithms. Of these the point-sampling algorithms can be further classified into:

- Z-buffer algorithms
- ray-tracing algorithms
- painter's algorithms

- scan-line algorithms

Since Phong's rendering model uses a scan-line approach to render the scene, it is natural to use a scan-line algorithm to implement the software frame buffer (virtual frame buffer).

4.0 User requirements

This section specifies the assumed needs of those using the graPHIGS based Phong rendering scheme being developed. This document does not address how the proposed system or the prototype will meet the user's needs, but only lists the needs to aid the design of the software.

It is assumed that the proposed scheme is to be used as a device-independent instruction set, which will add-on as a superset of the available graPHIGS functions.

The proposed system should:

- Allow the user to create shaded frames of 3-D models in 3-D views, specified using area defining primitives.
- Provide the user with a fast and precise rendering system, using Phong's equations for lighting and color computations.
- Allow the user to specify the 3-D model using any of the following area defining primitives (in addition to those currently supported by graPHIGS):

- Polygon 3 with data
 - Triangle strip
 - Quadrilateral mesh
 - Polyhedron
 - Non-uniform B-spline surface and
 - Parametric polynomial surface
- Allow the user to specify the following surface characteristics:
 - Ambient reflection coefficient
 - Diffuse reflection coefficient
 - Specular reflection coefficient
 - Specular color
 - Specular exponent
 - Transparency coefficient
 - Allow the user to specify light sources acting on the 3-D model using the following types of light sources:
 - Ambient light sources
 - Directional light sources
 - Positional light sources
 - Spot light sources
 - Allow the user to activate any combination of the specified light sources at any time.
 - Allow the user to specify the color model used.
 - Allow the user to reduce the update time required for the display of the shaded frames depending on the “coarseness” of the display.

- Allow the user to set the “coarseness” of the display (or the display time, as they are inversely proportional) to different settings. The settings may be real settings between 0.0 and 1.0, where 0.0 corresponds to least shading time and 1.0 corresponds to the best possible picture.
- Allow the user to terminate the updating of the shaded frame at any time.
- Allow the user to check the progress of the shading computation when the process is not in real time. This could be achieved by providing the user with an inquiry routine. This would allow the user to terminate the process at his/her discretion.
- Update the screen partially during the shading process, so that the user may terminate the process at his or her discretion.
- Allow the user to specify whether depth cueing should be activated.

5.0 Functional Requirements

The Software Functional Requirements contain the technical requirements of the software relative to I/O device handling, processing functions, definitions of technical constraints, and stipulation of control functions. The following are the functional requirements of the proposed system:

- **Control**
 - Modified graPHIGS + update
- **Output primitive structure elements**
 - Polygon 3 with data
 - Polygon set 3 with data
 - Triangle strip 3 with data
 - Quadrilateral mesh 3 with data
 - Polyhedron with data
 - Non-Uniform B-spline surface
 - Parametric polynomial surface

- **Utility functions to support output primitives**
 - Compute fill area set geometric normal
- **Indirect attribute selection**
 - Set depth cue representation
- **Individual attribute selection**
 - Set rendering color model
 - Set area properties
 - Set back area properties
 - Set interior shading method
 - Set light source state
 - Set face distinguishing mode
 - Set face culling mode
 - Set back interior color
- **Workstation attribute table definition**
 - Set light source representation
- **Hidden surface attributes**
 - Set HLHSR mode
- **Inquiry functions for workstation state list**
 - Inquire depth cue representation
 - Inquire list of light source indices

- Inquire light source representation
- **Inquiry functions for workstation description table**
 - Inquire light source facilities
 - Inquire predefined light source representation
 - Inquire curve and surface facilities

6.0 Software Functional Description

To use the designed rendering software the user must define the scene to be rendered using area defining primitives. At present the allowed area defining primitive which the software has been designed to support is Polygon 3 with data. The user must also set the “Interior lighting calculation mode” flag within the structure (or any other root structure within which the Phong structure is executed) in which he/she has defined the area defining primitives.

If the programmer now uses the modified update to update the workstation the defined view(s) within the workstation will be rendered and the image will be displayed on the screen.

In addition, the user has the option of setting the light sources, activating and deactivating them, setting back-face processing mode, setting face distinguishing mode, and setting the hidden surface processing mode within the structure to be rendered.

The following is a detailed description of the routines which are presently available to the user:

6.1.1.1 *Modified graPHIGS+ update*

phigs_update(wsid, nviews, views)

State required: (PHOP, WSOP, *, *)

Input:

- wsid - integer, workstation to be updated
- nviews - integer, number of views to be processed
- views(*) - integer, array of views

Description: Modified update for Phong rendering processing. The user can specify the views to be processed within the workstation, or specify nviews = 0 in which case all the views on the workstation will be processed.

6.1.1.2 *Polygon 3 with data*

gppld3(faflag, vflag, gnorm, nv, coords, colors, norms)

State required: (PHOP,*,STOP,*)

Input:

- faflag - integer, code indicating what information applying to the entire fill area is specified.
Valid values are:

- 1 - None
- 2 - Geometric Normal
- vflag - integer, data specified at each vertex. Valid values are:
 - 1 - Coordinate data only
 - 2 - Coordinate and color values
 - 3 - Coordinate and normal values
 - 4 - Coordinate, color and normal value
- gnorm(*) - real, normal in modelling coordinates
- nv - integer, number of vertices
- coords(*) - real, coordinate data (x,y,z)
- colors(*) - real, color data (r,g,b)
- norms(*) - real, normal vector

Description: Polygon 3 with data is used to specify the area defining primitive within an open structure.

6.1.1.3 Compute polygon 3 geometric normal

gpcmgn(nv, coords, err, gnorm)

State required: (PHOP,*,*,*)

Input:

- nv - integer, number of vertices specified
- coords(*) - real, coordinates of the vertices

Output:

- `err` - error code
- `gnorm(*)` - real, geometric normal

Description: Utility routine to compute the geometric normal

6.1.1.4 Set depth cue mode

gpsdcu(mode)

State required: (PHOP,*,STOP,*)

Input:

- `mode` - integer, depth cue mode. Valid values are:
 - 1 - deactivated
 - 2 - activated

Description: Set depth cue mode as a structure element

6.1.1.5 Set area properties

gpsap(ambc, difc, spec, speccol, specexp, trans)

State required: (PHOP,*,STOP,*)

Input:

- ambc - real, ambient reflection coefficient
- difc - real, diffuse reflection coefficient
- spec - real, specular reflection coefficient
- speccol(*) - real, specular color
- specexp - real, specular exponent
- trans - real, transparency coefficient

Description: Sets the surface properties for an area defining primitive

6.1.1.6 Set back area properties

gpbasp(ambc, difc, spec, speccol, specexp, trans)

State required: (PHOP,*,STOP,*)

Input:

- ambc - real, ambient reflection coefficient
- difc - real, diffuse reflection coefficient
- spec - real, specular reflection coefficient
- speccol(*) - real, specular color
- specexp - real, specular exponent
- trans - real, transparency coefficient

Description: Sets the surface properties for back-facing areas

6.1.1.7 *Set light source state*

gpslss(nact, act, ndeact, deact)

State required: (PHOP,*,STOP,*)

Input:

- nact - integer, number of activated light sources
- act(*) - integer, index numbers of light sources to be activated
- ndeact - integer, number of deactivated light sources
- deact(*) - integer, index numbers of the light sources to be deactivated

Description: Used to activate and deactivate light sources to be included in the lighting calculations for the area primitives below this element in the structure.

6.1.1.8 *Set face distinguishing mode*

gpsfdm(mode)

State required: (PHOP,*,STOP,*)

Input:

- mode - integer, mode. Valid values are:
 - 1 - NO
 - 2 - YES

Description: Used to decide whether the front and back-faces are to be displayed.

6.1.1.9 Set face culling mode

gpsfcm(mode)

State required: (PHOP,*,STOP,*)

Input:

- mode - integer, face culling mode. Valid values are:
 - 1 - both front and back-faces are displayed
 - 2 - only front-faces are displayed
 - 3 - only back-faces are displayed

Description: Used to decide which faces are to be displayed.

6.1.1.10 *Set back interior color*

gpsbic(color)

State required: (PHOP,*,STOP,*)

Input:

- color(*) - real, color values (r,g,b)

Description: Used to set the interior color for the back-faces

6.1.1.11 *Set light source representation*

gpslsr(wsid, index, itype, col, pos, dir, exp, att, angle)

State required: (PHOP,WSOP,*,*)

Input:

- wsid - integer, workstation identifier
- index - integer, light source representation index
- itype - integer, light source type. Valid values are:
 - 1 - ambient
 - 2 - directional
 - 3 - positional

- 4 - spot
- col(*) - real, color value (r,g,b)
- pos(*) - real, position (x,y,z)
- dir(*) - real, direction vector
- exp - real, concentration exponent
- att(*) - real, attenuation coefficients
- angle - real, cone angle

Description: Used to set the light source representation

6.1.1.12 Set hlhsr mode

gpshrm(wsid, hrm)

State required: (PHOP,WSOP,*,*)

Input:

- wsid - integer, workstation identifier
- hrm - integer, HLHSR enable flag. Valid values are:
 - 1 - disabled
 - 2 - enabled

Description: Used to enable hidden surface mode

7.0 Detailed design

The next three sections give details on the design of the three major parts of the system, namely, 1) graPHIGS extension 2) visibility detection and 3) Phong lighting and shading models.

The chapter on graPHIGS extension gives details on one of the most important aspects of the project. This part concerns the retrieval of model data and view data from the graPHIGS structures and workstation tables, and setting up the data structure to store the information to render the views. This is done solely with the available inquiry routines that graPHIGS provides.

Once the scene data and model data have been recovered from the graPHIGS structures, the next phase is the preprocessing of this data for the visible surface algorithm. This involves forming the data structure for the edge and face data, maintaining an active edge list for each scanline, and finding the visible surface data in world coordinates for the rendering process.

The rendering process uses the visible surface data in world coordinates, (i.e. point coordinates, normals, surface properties, active light source data) and uses the Phong lighting and shading models to evaluate the intensity of the polygon at that point. This intensity is then loaded into the

virtual frame buffer and displayed as a pixel primitive (in a structure having 0.5 priority) in the same view from which all the data was retrieved. This structure is then deleted, and will disappear during the next workstation update.

Due to the limited color display on the IBM 5080 workstations (8 bits/pixel), the algorithm requires a dithering algorithm which renders the scene using the best possible colors. This problem has not been dealt with in this thesis, though it will be dealt with in the near future.

7.1 *graPHIGS extension*

In creating the Phong rendering environment on top of graPHIGS, the primary consideration was to design the new system to be completely integrated with the current graPHIGS instruction set. Thus, in addition to accessing the new data structure for the additional primitives, attributes and other accessories (e.g. light sources), the system must retrieve the necessary information for rendering the scene from the graPHIGS data structure using available graPHIGS inquiry routines.

Thus, in the rendering process, in addition to the visible surface algorithm there is one other process which may prove to be very compute intensive - i.e. building the scene to be rendered from the graPHIGS data structure. This involves building the transformations from the root structures of the active views to the "Phong shading structure", i.e. the structure in which the flag for the shading method has been set to Phong shading method, and retrieving the view information - view mapping, view orientation and view clipping.

For retrieving the above data, graPHIGS provides the user with the following inquiry routines:

- WSL Inquiries:

- GPQAR - Inquire Set of Associated Roots
 - GPQRV - Inquire Set of Roots in View
 - GPQRVX - Inquire Requested Viewing Transformation
 - GPQVR - Inquire Set of Views Which Contain Root
- WDT Inquiries:
 - GPQHD - Inquire Maximum Hierarchy Depth
 - GPQNV - Inquire Number of Definable View Table Entries
- General Inquiries:
 - GPQEMO - Inquire Error Handling Mode
 - GPQEMS - Inquire Error Message
 - GPQEP - Inquire Element Pointer
 - GPQETS - Inquire Element Type and Size
 - GPQEXS - Inquire Executed Structures

Before discussing the details of retrieving the scene information and building the scene to be rendered, the method of displaying the scene using graPHIGS primitives are presented below along with the associated limitations of such a display.

It was proposed in the design that the rendered scene be displayed on the terminal using the 2-D pixel primitive (or several 2-D pixel primitives, depending on the limitations of the primitive on a particular display device). This 2-D pixel primitive would extend to the height and width of the view window limits. This pixel primitive would be drawn in a structure with a structure priority of 0.5 within the view. Since, the 2-D pixel primitive would cover the entire window for the view, other graPHIGS primitives in the scene (e.g. text, line, etc.) which are in structures with a priority lower than 0.5 would not appear on the screen. Thus, it is up to the discretion of the user to set his/her other structures either in front of or behind the 2-D pixel primitive structure.

All the changes made to the graPHIGS data structure by the rendering process (to display the shaded image on the screen) will be removed after updating the workstation.

Each view is processed from the root with the lowest priority to the root with the highest priority within that particular view. If in any root there exists a “Phong shading structure” (a structure in which the lighting calculation mode is set to PHONG - 4) within its hierarchy, the polygons in that structure are processed depending on whether the hidden surface flag is set ON or OFF, and the shaded colors are written into the virtual frame buffer. After all the roots in a view have been processed, the virtual frame buffer is displayed using the 2-D pixel primitive.

7.1.1.1 PHIGS+ routines and data structures

In addition to the Phong rendering scheme the system should provide the user with PHIGS+ functions for the lighting data structure, the rendering scheme, additional rendering primitives and attributes. The graPHIGS implementation of all these will be effected with the use of application data structure elements.

To implement the proper functioning of the additional data structures to be maintained, it was designed that the entire subset of the Phong rendering calls would act as a finite state machine. The user is provided with a Phong open (vpoppg) and Phong close (vpclpg) call which would dynamically allocate space for the additional data structures used and close them. The Phong open call (vpoppg) requires a state of “workstation open”.

For the implementation of the data structures for the new PHIGS+ primitives and lighting, the use of simple arrays and linked lists should be sufficient. The data structure should incorporate least search time at the expense of memory. It is assumed that the simplest representation is best.

7.1.1.2 Retrieving view and structure information

In order to render the scene, information about the views must be retrieved from the graPHIGS data structure using the inquiry routines within graPHIGS: the views that are associated with a workstation, the details of each view, i.e. the view orientation, the view reference point, viewport limits, window limits, etc..

Collecting the polygon data and its associated attributes in World Coordinates (WC) is the compute intensive process. Because of the inherent hierarchical structure of graPHIGS, all possible paths from the root structure to the Phong shading structure must be determined. After that, within every path, each structure in the path should be opened and the transformation matrix and the surface attributes collected till just before the Phong shading structure. Then, the Phong shading structure should be queried to get the polygon data and transform it into world coordinates.

To find the paths from the root structure to the Phong shading structure and build the transformations (and collect surface attributes) along the path, the graPHIGS data structure (Figure 1 on page 36) is queried and a tree structure (Figure 2 on page 37) of all the structures being executed within the root structure is formed. As the tree is built, if a leaf node (tip structure) which is a Phong shading structure is found, a process is started to traverse up the tree and label all the nodes in the path as useful. Once the tree is built, it is traversed down through all the useful nodes and the transformation matrices are queried at each structure. These transformation matrices are to be stored in a separate data structure, depending on the node structure, the executed structure and the occurrence number of the executed structure. This data structure, to store the transformations, can be implemented as a simple array, or a well balanced binary tree. Along with these transformations the surface characteristics set within that particular node structure are also stored. After building the transformations, the tree is traversed again, collecting the path to the current root. On reaching the leaf node (the Phong shading structure), the transformations for that path are queried from the data structure and the polygons of the Phong shading structure are

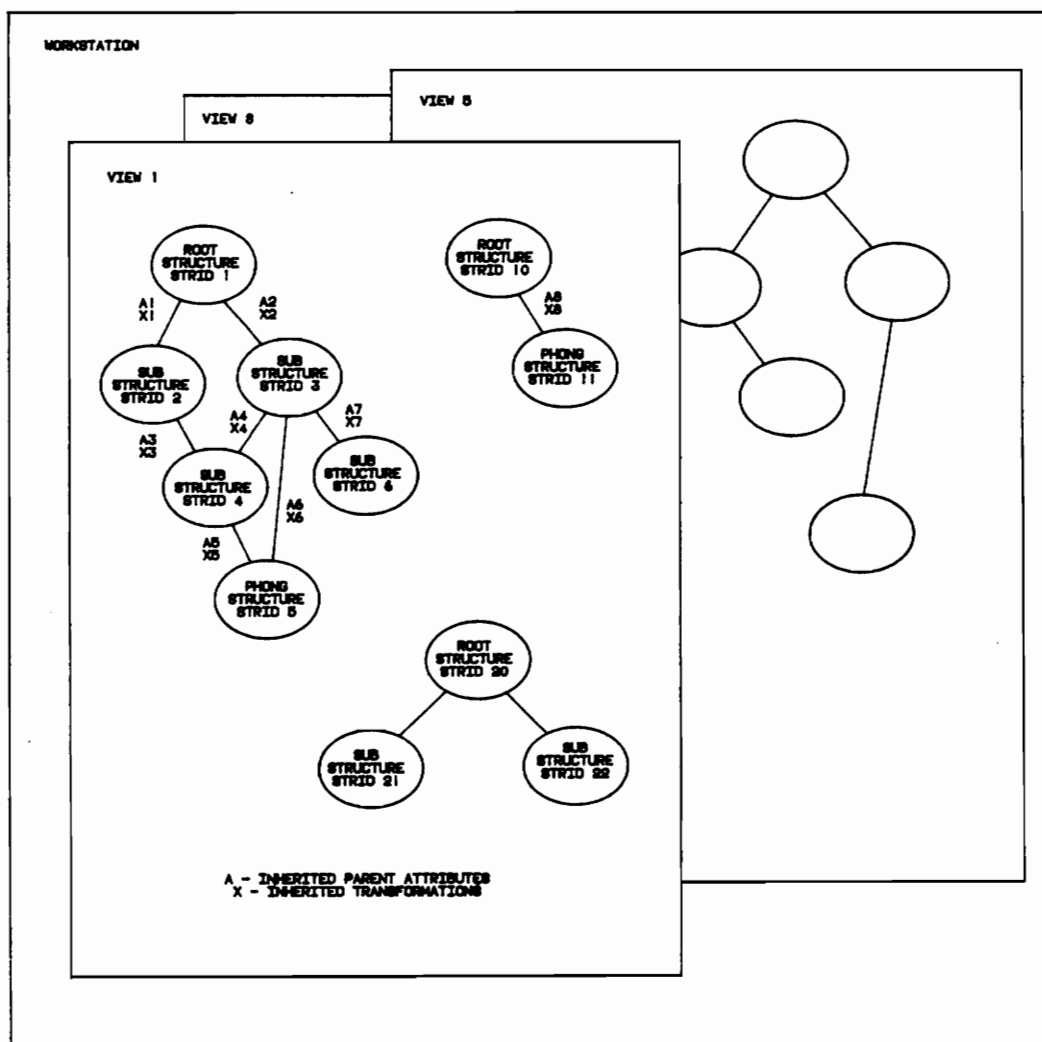


Figure 1. Schematic of graPHIGS internal data structure

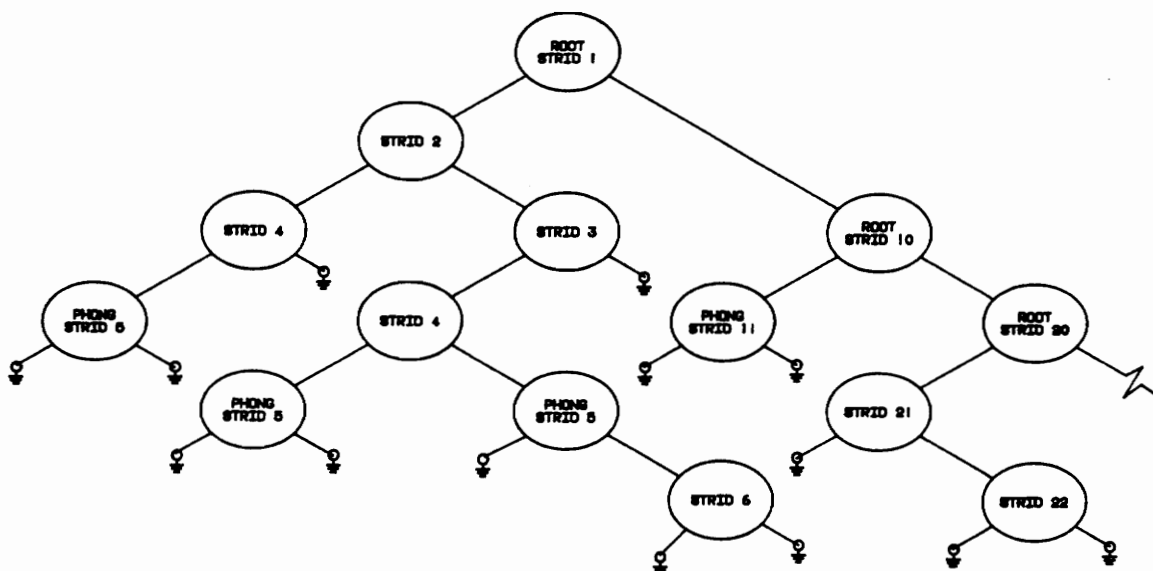


Figure 2. Tree structure for structure hierarchy manipulation

queried and placed in a bin. After processing all the Phong structures executed within a particular root, the whole bin is processed to fill up the virtual frame buffer with the appropriate color values. Then the root with the next higher priority is processed.

Collection of polygon data within a Phong shading structure is done along with the transformation computation traversal. In that traversal of the tree, when a Phong shading structure is reached, the list of processed Phong structures is checked to see if the current structure has been processed. If it has not been processed before, the structure is queried to build objects with the correct attributes. Once the transformations for the different paths are built, the polygons are transformed and added to the list of objects in the scene.

The back-face processing and face distinguishing are performed when the polygon data from within a Phong shading structure are collected. If back face processing mode is “ON” the polygons are sorted, with the front facing polygons forming one object and the back facing polygons forming another object.

Once the view information and the object information within the scene to be rendered are available, there is enough information to start the actual rendering process. The rendering process can again be broken up into two sub-processes, the visibility detection (virtual frame buffer) algorithm and the shading computation. These are discussed in detail in the following sections.

7.2 Visibility detection

The visibility detection algorithm constitutes one of the most compute intensive phases of a rendering pipeline. There are several different types of visible surface algorithms which are commonly used:

Z-buffer algorithm

Painters algorithm

Scan line algorithm

Ray tracing technique

The PHONG shading method uses the scan line information to compute the intensity at a visible point. Hence a scan line algorithm is probably the best for solving the visible surface problem. The scan line algorithm returns the intersection of the scan plane with a visible facet. The shading algorithm computes the normal vectors at the edges of the scan line, by interpolating the normals at the vertices. These normals are then interpolated over the visible portion of the polygon on the scan line.

In most scan line techniques the object data (in object space) is maintained as a linked list of edges and faces. The maximum y-value of the face is also stored. This is done so that the intersection check for each scan line is reduced to the list of active polygons.

A spanning scan line method maintains one scan line of z-buffer into which the intersections of the scanplane are processed. In order to start the process of visibility detection, the objects are sorted by y value. An active list of edges and intersecting polygons is maintained as each scan line is processed. Thus, at any time, there are three lists of faces; the active list, the list of polygons which are above the scan line and the list of polygons which are below the scan line. When the maximum y value of the polygon is above or equal to the scan line (constant y value) and the minimum is below the scan line, the polygon is added to the active list. The intersections of the scan plane with the polygons in the viewing volume create segments of lines from different faces. The segments are subdivided at the edge overlap and the intersection areas, to create span sections. The scan line is then processed from left to right (increasing values of x), processing a z-sort to get the visible scan line. The shading module then calculates the intensity at each pixel and writes the color values into the virtual frame buffer.

This technique can also be extended to parametric surfaces. A first pass is made at all the surfaces to find the maximum y value associated with each surface. The surfaces are subdivided until a surface can be approximated by a planar facet. The limit is usually the size of a pixel. These are then processed as explained above.

To implement this procedure in a graPHIGS environment, the polygon data in modeling coordinates (MC) must be transformed into device coordinates (DC). The process of collecting the model data is done in several steps. In the first pass of building the tree structure, the Phong shading structure is queried to build objects (polygons with the same attributes), at the same time as the transformations for the nodes higher up in the tree structure are determined. These objects are stored in two groups. Any object which lies after the first global transformation within the Phong shading structure has already been completely transformed into world coordinates. The others have been partially transformed, since the global transformation of the tree above the Phong shading structure must be applied to them. Once the tree structure and all the transformations have been built, the tree is traversed for every path to the Phong shading structure, building the global transformations and collecting all shading attributes set in the tree structure. This global transformation is then applied to the non-processed objects of the Phong shading structure to get the objects in world coordinates. All the Phong structures are processed to get a collection of objects in the scene in WC. The viewing information and the workstation transformation are then used to convert this data into device coordinates (DC).

The viewing volume is a rectangular block in device coordinates. The viewing volume can then be processed by stepping down the device coordinates in raster steps to get scan planes ($y = \text{constant}$). The scan planes intersect with the model to get scan line segments. After the spanning segments of the visible faces are determined, each visible point on the object is passed back to the shading computation module in world coordinates (WC) along with the edge points. The color computations are done in WC, and are directly fed into the frame buffer for display.

Thus a virtual frame buffer, (the size of the display in device coordinates) should be maintained. The shading module stores the RGB primary color values into the virtual frame buffer. Since the 2-D pixel primitive does not have direct color indexing and also the IBM 5080 workstations have a color table limitation of 128 colors (with graPHIGS), the virtual frame buffer must be post processed to select the most commonly used colors to define the color table. The rest of the colors are indexed to the nearest color available in the color table.

The frame buffer is then converted into a color index array which is displayed using a 2D pixel primitive in a structure which is associated with the view being processed with a priority of 0.5. Since the pixels of the 2D pixel primitives are specified in modelling coordinates, a local transformation which reverses the effect of the viewing transformation is specified before the 2D pixel array primitive. After the updating the workstation, this structure and all the deleted structure references are reset before passing the control back to the applications program.

7.3 Phong lighting and shading methods

7.3.1.1 Lighting models

In this design of Phong shading, only local lighting is considered. Therefore mirror and shadow effects cannot be created using this shading method. Once the basic rendering method is created, transparency, mirror effects, shadows and other features can be built in without major design changes.

Four types of light sources have been identified for inclusion in this design:

- Ambient

- Positional
- Directional
- Spot

The characteristics for any light source can be set using the “Set Light Source Representation” function. Light sources can be activated and deactivated using the structure element “Set Light Source State”. This element acts as an attribute for the primitives being shaded by maintaining a set of active light sources for every primitive which can be shaded (in this design the only primitives which fall into this category are the Polygon 3 and the Polygon 3 with data). The lights sources which are activated for each object are maintained in the data structure for the object.

Lighting effects can be calculated by first calculating the intensity of the light reaching the geometric primitive and then calculating the amount of light that is reflected in the direction of the viewer’s eye. If “S” represents the intensity of the light source being considered and “I” represents the intensity of the incident light at a point on an object, for **ambient** and **directional** light sources:

$$I = S$$

For **Positional** light sources, the intensity of incident light is attenuated by two attenuation coefficients, a_1 and a_2 which can be specified in the “Set Light Source Representation” call.

$$I = \frac{S}{a_1 + a_2|\overline{P} - \overline{O}|}$$

where,

\overline{P} is the location of the light source (in WC) and

\overline{O} is the location of the illuminated point on the object (in WC)

For Spot light sources,

$$I = \frac{S \cos^c \phi}{a_1 + a_2 |(\bar{P} - \bar{O})|}$$

where,

c is the light source concentration exponent and

ϕ is the angle between the incident ray vector and the direction of the light source (as shown in Figure 3 on page 44).

If the illuminated point falls outside the spread angle (θ) of the light source (i.e. $\phi > \theta$),

$$I = 0$$

7.3.1.2 Shading calculations

In the Phong Shading method, the normal to the surface of an object (at the illuminated point) is calculated by interpolating the normals at the vertices of the polygon. Referring to Figure 4 on page 45, if P is the illuminated point on the object (which is represented by polygons), then

$$\hat{n}_Q = u\hat{n}_A + (1-u)\hat{n}_B \quad 0 \leq u \leq 1$$

$$\hat{n}_R = w\hat{n}_B + (1-w)\hat{n}_C \quad 0 \leq w \leq 1$$

$$\hat{n}_P = t\hat{n}_Q + (1-t)\hat{n}_R \quad 0 \leq t \leq 1$$

where,

$$u = \frac{AQ}{AB}$$

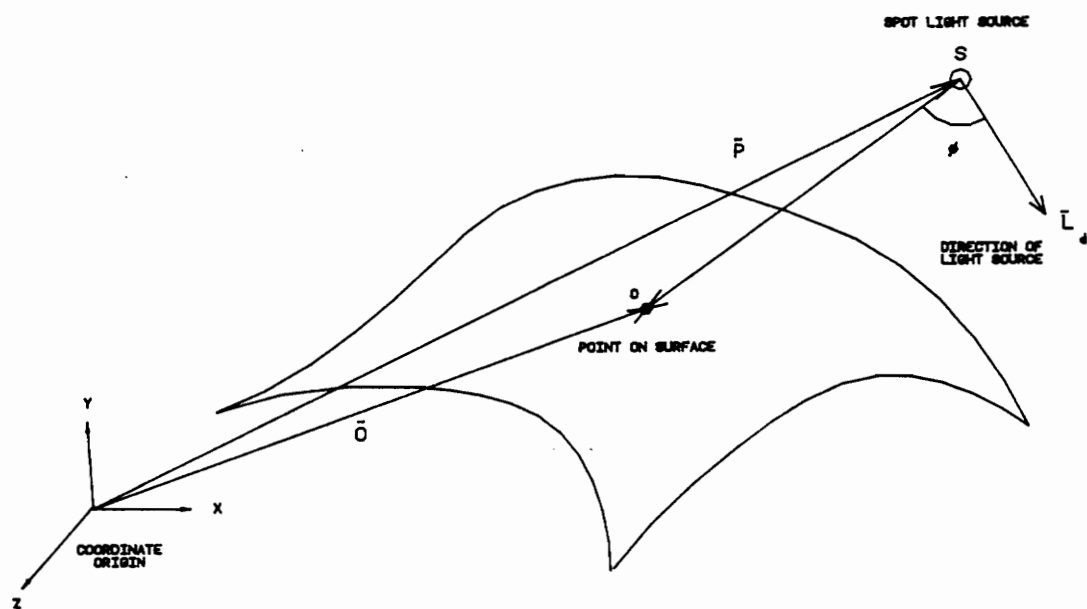


Figure 3. Spot light sources.

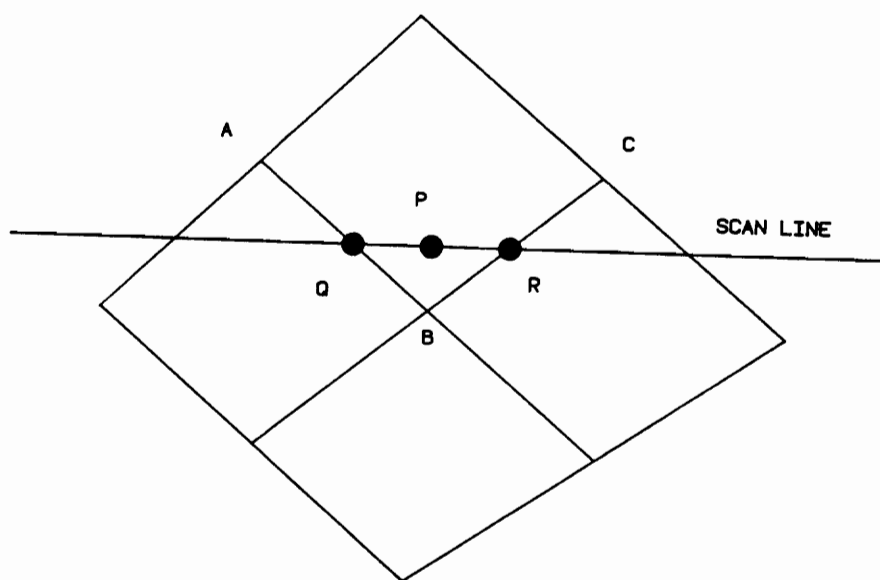


Figure 4. Vertex normal interpolation on scanline

$$w = \frac{BR}{BC}$$

$$t = \frac{QP}{QR}$$

and \hat{n}_p represents the unit normal at the point P.

After calculating the normal at the illuminated point and the intensity of the incident light from each light source, the intensity of light reflected towards the viewers eye can be calculated using the following equations (based on Phong's specular reflection model):

$$I = k_a \sum_{i=1}^{n_a} I_{ai} + \sum_{j=1}^{n_o} \frac{I_{lj}}{d + K} [k_d(\hat{n} \cdot \hat{L}_j) + k_s(\hat{R}_j \cdot \hat{S})^n]$$

where,

I_R = Intensity of reflected light

I_a = Intensity of incident ambient light

I_l = Intensity of incident non-ambient light

k_a = ambient reflection coefficient

k_d = diffuse reflection coefficient

k_s = specular reflection coefficient

n = specular exponent

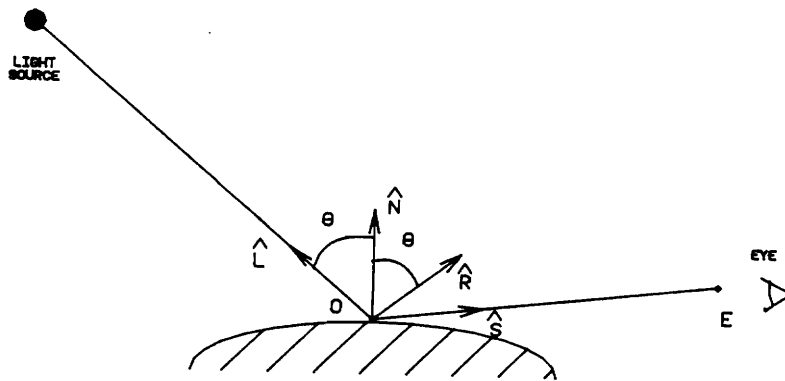


Figure 5. Phong's reflectance model

K = arbitrary constant (usually $K = 1$)

d = distance from perspective viewpoint to illuminated point (OE in Figure 5 on page 47)

\hat{n} = unit normal at the illuminated point

\hat{L} = unit vector from the illuminated point to the light source

\hat{S} = unit vector from illuminated point to the perspective viewpoint (or projection vector for parallel projection)

$\hat{R} = 2(\hat{n} \cdot \hat{L})\hat{n} - \hat{L}$ = unit reflectance vector

n_a = number of ambient light sources

n_o = number of non-ambient light sources

The term $(d + K)$ in the previous equation accounts for the depth cueing of the shaded image. For use with the graPHIGS, the distance “ d ” is the distance from the illuminated point to the projection reference point. If depth cueing is not desired, the value of $(d + K)$ should be set to 1.

For colored light source falling on colored objects, the equation needs to be modified to:

$$(I_R)_c = k_a D_c \sum_{i=1}^{n_a} (I_{ai})_c + \sum_{j=1}^{n_o} \frac{(I_{lj})_c}{d + K} [k_d D_c (\hat{n} \cdot \hat{L}_j) + k_s S_c (\hat{R}_j \cdot \hat{S})^n]$$

where,

$c = R, G, B$ for the three color primaries

and,

D_c = Diffuse color component of the surface

S_c = Specular color component of the surface

8.0 Implementation

The next three sections give details of the implementation of the three major parts of the project for which the design was created, namely, 1) graPHIGS extension, 2) visibility detection and 3) Phong lighting and shading models.

The implementation was done on the IBM-RT network so that the speed of the process was unaffected by any other processes. The AIX operating system also provided an ideal environment to develop C code. Since there were no language limitations, C was chosen as the programming language. C provides for recursive programming techniques and efficient data structures with pointer referencing. All these advanced programming techniques were used in the implementation to allow efficient data structures. The “binary tree” data structure was used to implement most of the data structures.

The design and implementation of the code was done with a “top-down” design methodology. The coding was done with easy code readability and extensive commenting in mind. There is almost 50% of comment statements within the code itself. Each routine was developed to form a compact logical block. Each routine was coded in a separate file and has an introductory header

block with the module name, routine description, input, output and local variables used, and references to external global variables used. All type and macro definitions were done in a separate file called “declare_ph.h” which was then included in all the source code files. In the development of the code the use of global variables was limited to the minimum possible, with each of the major data structures having just one pointer to the data structure as a global variable. All the global variables were declared in the file “global_ph.h” and was included once in the “phong_update.c” file.

With this background on the implementation methodology, we shall now discuss in detail the data structures used for the implementing the different sections of the rendering pipeline.

8.1 graPHIGS extension

In the implementation of the rendering process the graPHIGS extension forms the initial phase. In this section we consider the implementation of the additional PHIGS+ routines which the applications programmer can access to provide all the additional information required for rendering (additional primitive information, lighting information, etc.), and the data structures required to store them in the graPHIGS environment. We shall also consider the implementation of the data structures for information retrieval from graPHIGS to render the scene.

8.1.1.1 PHIGS+ routines and data structures

For the implementation of additional structure elements, no new data structures are required since all of them are stored within the graPHIGS data structure as application data and can be retrieved

during update. But there are some elements which are not structure elements, like the additional workstation state list elements of light source and depth cue representation. These are to be implemented as an array of appropriate data types. These arrays are dynamically allocated during the “Phong open” call. This implementation of dynamic arrays of data types allows for least search time at the cost of memory.

Additional PHIGS+ primitives, for example Polygon 3 with data, are implemented as two structure elements. The first of which is an application data element which stores the vertex coordinates, normal and color information and the second is a standard Polygon 3 structure element. The advantage of this implementation is that when the user does not use the modified Phong update call, the regular graPHIGS update processes the Polygon 3 primitive. This implementation suffers from the disadvantage that the user is not aware of the two structure elements, when he/she is inserting only one. Thus, either the user should be warned of this or some graPHIGS calls will need to be modified (e.g. offset element pointer should be modified to consider the two elements as one offset, etc).

8.1.1.2 Retrieving view and structure information

In the rendering process, the hidden surface algorithm requires the scene content information (polygon information in world coordinates) and the view information (view orientation, mapping, etc.). Thus at the time the user calls the modified Phong update call, the first part of the process is the information retrieval from the scene.

In the information retrieval the view information is fairly straight forward. GraPHIGS provides the routine “GPQRVE” (Inquire requested view table entries) which returns the number of view table entries and the associated view indices in decreasing priority for the specified workstation. This routine is used to check whether the views specified by the user are actually part of the view table

entries. Also when the user has not provided a list of views and requires all the active views on the workstation to be rendered, this routine is used to get the list of view entries. When the user requires all views to be rendered, all views other than view with index "0" will be rendered. It was implemented not to render view "0" as the pixel primitive on the 5080's do not have the display capability of covering the entire screen, and also it is assumed that the user will use the shading routines to render 3-D scenes, and view "0" will most probably be used to display 2-D objects (menu items, etc.).

The query routine "GPQRVX" (Inquire Requested Viewing Transformation) returns the view orientation matrix, the view mapping matrix, the view reference point, clipping information, background shield information, border information, and a flag specifying whether the view is active or not. After the initial check to see whether the view is active further calculations on that view are performed.

The "GPQADS" (Inquire Actual Maximum Display Surface Size) provides the device display size in raster units and "GPQWSX" (Inquire Workstation Transformation) returns the transformations from the NPC (Normalized Projection Coordinates) system to DC (Device Coordinates) system. These two transformations are concatenated to get the transformation from the NPC to DC system.

The above transformations are used to calculate all the primitive description in device coordinates (DC) to pass it through the scan line visible surface detection algorithm. In addition, the above transformations are also used to calculate the 3-D point that the pixel primitive will start from in world coordinates (WC). The whole scene is rendered one scan line at a time, hence each scan line is an independent pixel primitive. To draw the pixel primitive we require the starting point and the number of pixels to be drawn. The number of rows and columns of raster units required to cover the viewing area is calculated from the mapping of NPC (normalized projection coordinates) to DC (device coordinates). This information is used to allocate the frame buffer memory and the starting points of each individual scan line.

Once the view information is retrieved, and it is confirmed that the view is active, and the pixel data has been calculated, the next step is to retrieve the 3-D primitive data with associated attributes and convert everything into device coordinates (DC) to preprocess the data for the scan line visible surface algorithm.

The polygon data collection from the `grPHIGS` structures is a very compute intensive process since each structure has to be opened and each structure element has to be queried for its element content to check whether it is an element which the rendering process requires. This is true for the whole structure hierarchy, since all the attributes and transformations are passed down to the children.

To reduce the time taken to collect all this data the structures are categorized into three groups for the data retrieval process. The first category is classified as the “Phong shading structures” which are the structures which need to be queried for the 3-D primitive data, associated attributes and transformations. The second category is the “parent structures” which are the structures which need to be queried only for the attributes and transformations which are passed down to the Phong shading structures. The third category is all the other structures which need not be opened at all, since we are not affected by any of those structures. The data collected from the first and second categories of structures are then rendered and drawn as pixel primitives within the view. This pixel primitive shows up as the rendered scene in the view during the regular `grPHIGS` update.

Four separate data structures are maintained to retrieve and store information from the structures. The first is a simple linked list of integer data fields which contain structure identifiers. This data structure exists throughout the Phong process, i.e. from Phong update to Phong close. The “`vplcmo`” (set lighting calculation mode) call provided to the user is a structure element which classifies the structure into the first category, i.e. “Phong shading structure”. At the time the user uses this call to set a structure element, an application data element (with lighting calculation code) is placed within the structure and an element is added to the linked list. The elements of this linked

list are updated every time the modified Phong update is called. Thus this linked list maintains the list of Phong shading structures which must be opened for 3-D primitive and attribute retrieval.

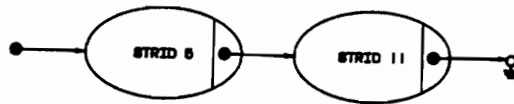
Figure 6 on page 56 shows the data type and the schematic of the data structure for the example structure hierarchy considered in Figure 1 on page 36

The second data structure is a binary tree structure which converts the graph structure of the graPHIGS structure hierarchy (shown in Figure 1 on page 36) into a binary tree structure from the root structure associated with the view to the “Phong shading structure”. This data structure now has only one path from the root to every leaf node. During the modified Phong update, after the view information is retrieved, the “GPQRV” (Inquire Set of Roots in View) call is used to retrieve the roots associated with each view. This routine also returns the corresponding priorities of each root structure. Although it has not been implemented, this information can be used to process the root structure with the lowest priority first and the root structure with the highest priority last if the hidden surface algorithm is not being applied.

Next, the “GPQEXS” (Inquire Executed Structures) call is used recursively to form the binary tree structure. At each node a check is made to see whether the structure is a “Phong shading structure”. If so, then the inquiry process is terminated, the Phong shading structure is made a leaf node and all the nodes from the Phong shading structure to the root node are marked as useful. All the useful nodes in the binary tree form the second category of structures, namely the parent structures, whose attributes and transformations are inherited by the Phong shading structures. Figure 7 on page 57 shows the type declaration and the corresponding structure of the Structure hierarchy data structure for the example case considered.

After the Structure hierarchy data structure has been created, each parent structure (nodes which have been marked useful) is opened and the elements are queried to collect the transformations and structure attributes. The “GPQETS” (Inquire Element Type and Size) and “GPQE” (Inquire Element Content) inquiry routines are used to inquire the element content. The parent structure is opened and the transformations and attributes are collected until an “executed structure” element

PHONG LIST SCHEMATIC FOR EXAMPLE STRUCTURE HIEARARCHY



```
/* type definition for a linked list with integer items */
typedef struct int_list_node {
    int strid;
    struct int_list_node *next;
} int_list;
```

Figure 6. Linked list data structure to maintain Phong shading structures

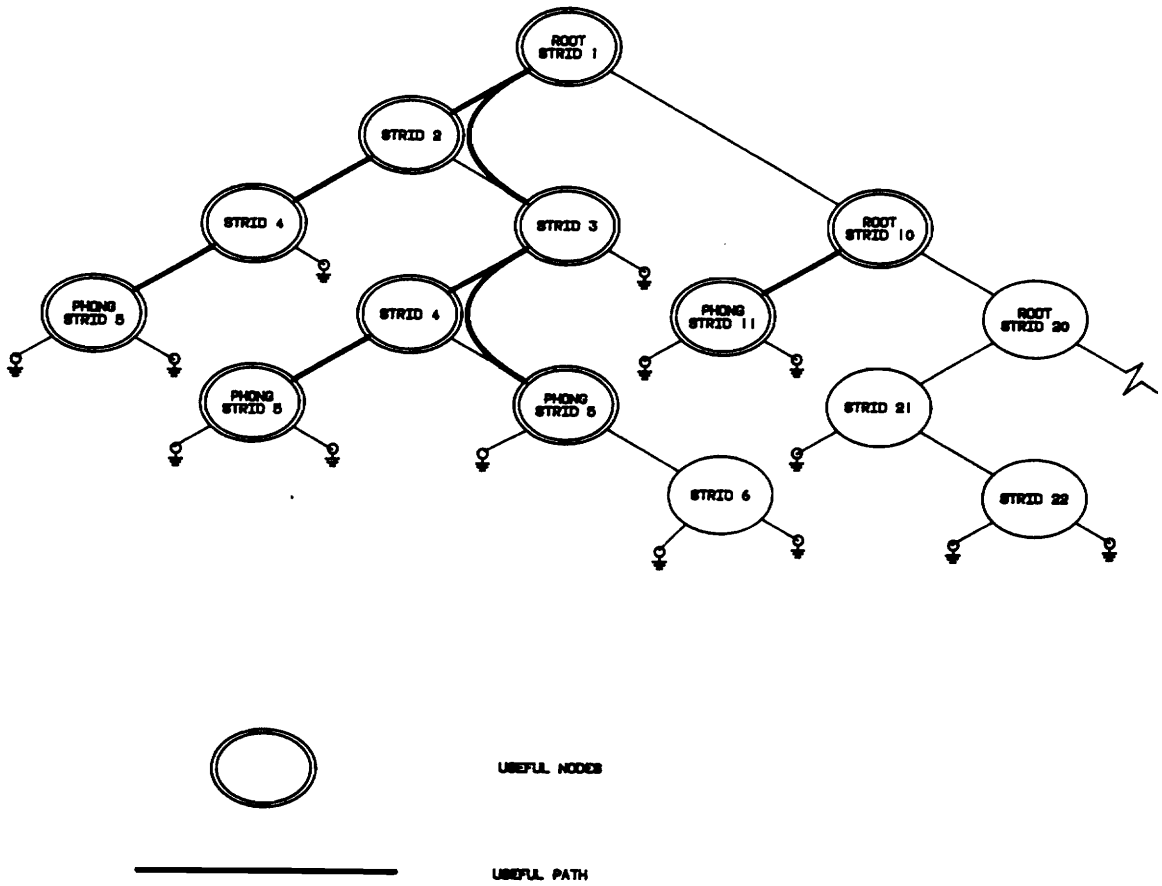


Figure 7. Binary tree data structure for structure hierarchy

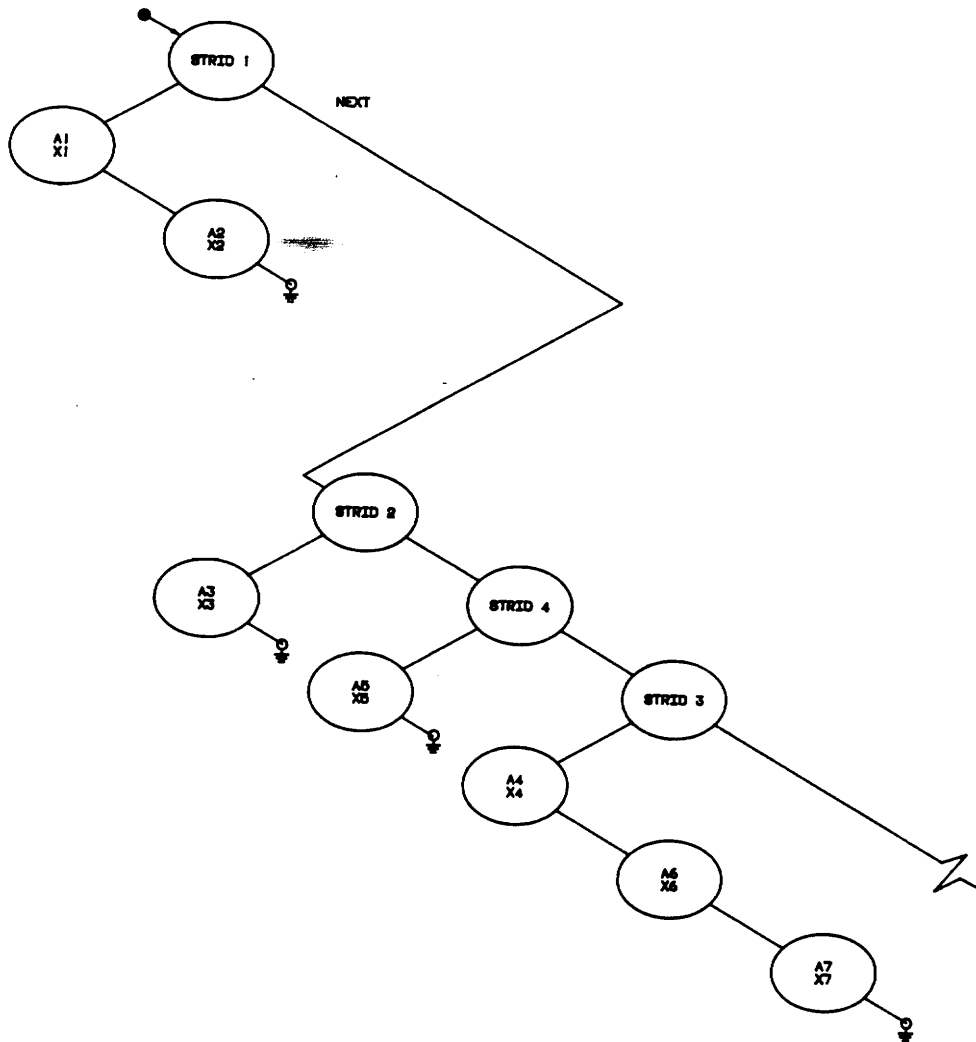


Figure 8. Attribute tree data structure

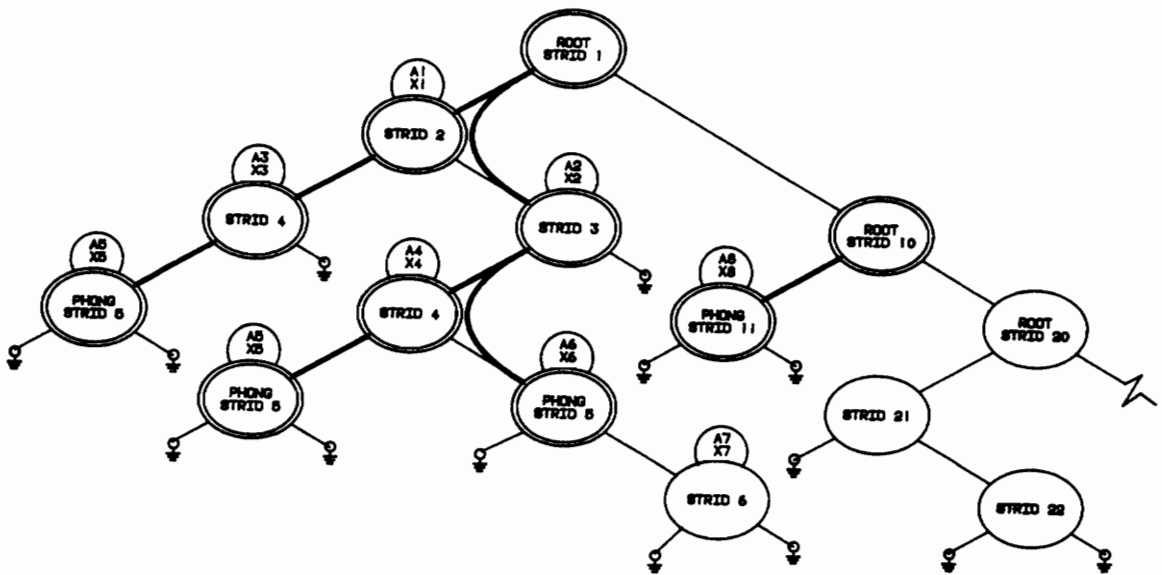


Figure 9. Cross-referencing nodes of the two data structures

is reached. At this point the current transformations and attributes are placed in an attribute node. These are the transformations and attributes associated with the first executed structure in the root structure. This process is continued till the last element has been processed. Once the attribute tree has been formed, the nodes in the Structure hierarchy data structure are cross-referenced to the attributes nodes in the attribute tree as shown in Figure 9 on page 59. To avoid repeating the search for attributes, the same attribute nodes are referenced to all the occurrences of the same root structure.

After the attribute tree data structure has been created, a face tree data structure is build for all the Phong shading structures using the “GPQETS” and “GPQE” inquiry routines. The face tree contains a pointer to the next structure node and a pointer to the face list. The face list is a list of nodes containing a pointer to the vertex list, and the associated attributes of the face. The vertex list contains coordinate data, normal data and color data at each vertex. In addition each vertex node has a vertex break flag which is set to TRUE whenever the vertices are from two different subareas. Each Phong structure leaf node in the original Structure hierarchy data structure is cross-referenced to the corresponding face list in the face tree data structure.

Figure 10 on page 61 shows the the schematic of the face tree data structure for the example considered. This completes the data retrieval from the **grPHIGS** data structure. Figure 11 on page 62 shows all the data structures and their cross referencing for the example considered. This data is now passed into the scan line algorithm.

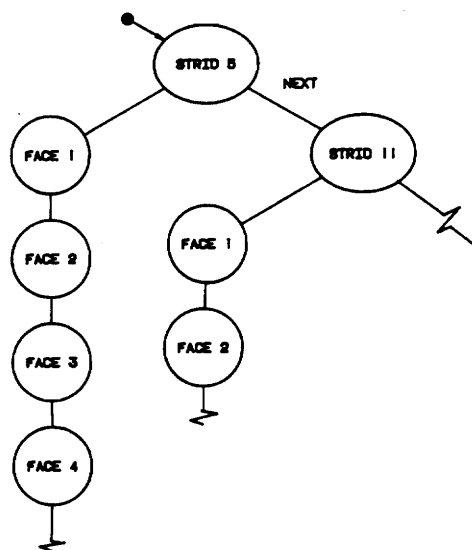


Figure 10. Face tree data structure

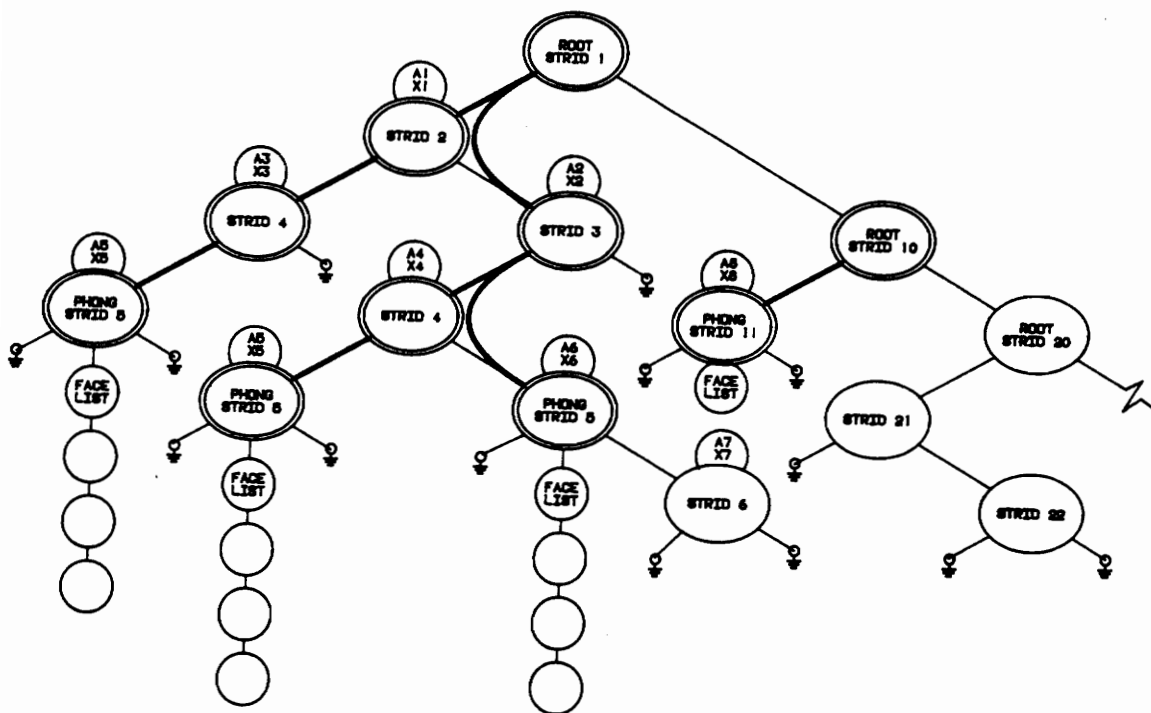


Figure 11. Data structures and cross-referencing

8.2 *Visibility detection*

The previous section showed the data is retrieved and stored in the various data structures and how the data structures have been cross-referenced for easy access. The next step in the rendering process is the transformation of this data into device coordinates (DC).

This is done by traversing the structure hierarchy data structure and concatenating the attributes and transformations as we traverse the tree structure. At the leaf node (Phong shading structure) these attributes and transformations are concatenated with the attributes and transformations of each and every face of the structure and the face is then added to a face list in device coordinates. As the data in the structure hierarchy data structure is in world coordinates (WC), a transformation matrix from world coordinates to device coordinates has to be applied to transform the data into device coordinates. Traversing all the useful paths of the structure hierarchy data structure will give us all the polygons required in device coordinates.

In the collection of the polygon data in device coordinates, a check for front facing and back facing polygons could be made, and depending on the face culling mode, polygons could be placed in the face list with the appropriate attributes. This was not incorporated in the prototype developed.

The next logical step in the rendering pipeline is the clipping of these polygon data to the window limits and near and far clipping planes. In the implementation of this prototype it was assumed that all the polygons rendered were within the clipping volume, and hence the clipping algorithm was not implemented. A simple version of the Sutherland-Hodgman clipping algorithm could be implemented, if the polygon data needs to be clipped.

Having got the data in device coordinates, the actual scan line algorithm is done to process the hidden surface and to display the scene. A simplified outline of the YXZ sorting process of the scan

line algorithm was described in the detailed design. We shall now go into the details of the implementation of the process.

A simplified version of the scan line algorithm could be stated as follows; for each scan line, the intersections of the scan line with all edges of the polygon are found, and sorted by x-values, and then the pixels between consecutive pairs of intersections are filled in with the appropriate intensities. At each intersection of the edge with the scan line the parity of the polygon changes, from inside to outside or vice versa. In the above brief description of the process (which will be expanded later to give a detailed explanation of the data structures used), some special cases have to be considered.

Horizontal edges need not be included in the process, as they are automatically filled.

Any vertex produces two intersections with the scan line, and depending on whether the vertex is a local extremum or not, the upper y value of the lower edge is reduced by one scan line so that the parity does not change. Figure 12 on page 65 shows how the y value is changed for the vertices which are not at local extremum.

To reduce the burden of finding the intersections of each polygon with the scan line we preprocess the face list to an edge list. This edge list is sorted into bins by the starting y-value of each edge. Figure 13 on page 66 shows the type definitions for the edge data and face list and a schematic of the data structure. To minimize the memory space used a pointer is maintained from the face list to the original face in the structure hierarchy data structure so that all the attributes need not be copied into the new face list. In the edge data we maintain the x intersection of the edge with the lowest scan line, the reciprocal of the slope of the edge in the xy-plane. This value is added to the x intersection when we step across scan lines to get the new x intersection of the edge with the next scan line. The depth, coordinate and normals with the corresponding increments in the x direction are also stored. The depth is used to find out which polygon can be seen when several polygons overlap and the coordinate and normal values are used in the intensity calculations.

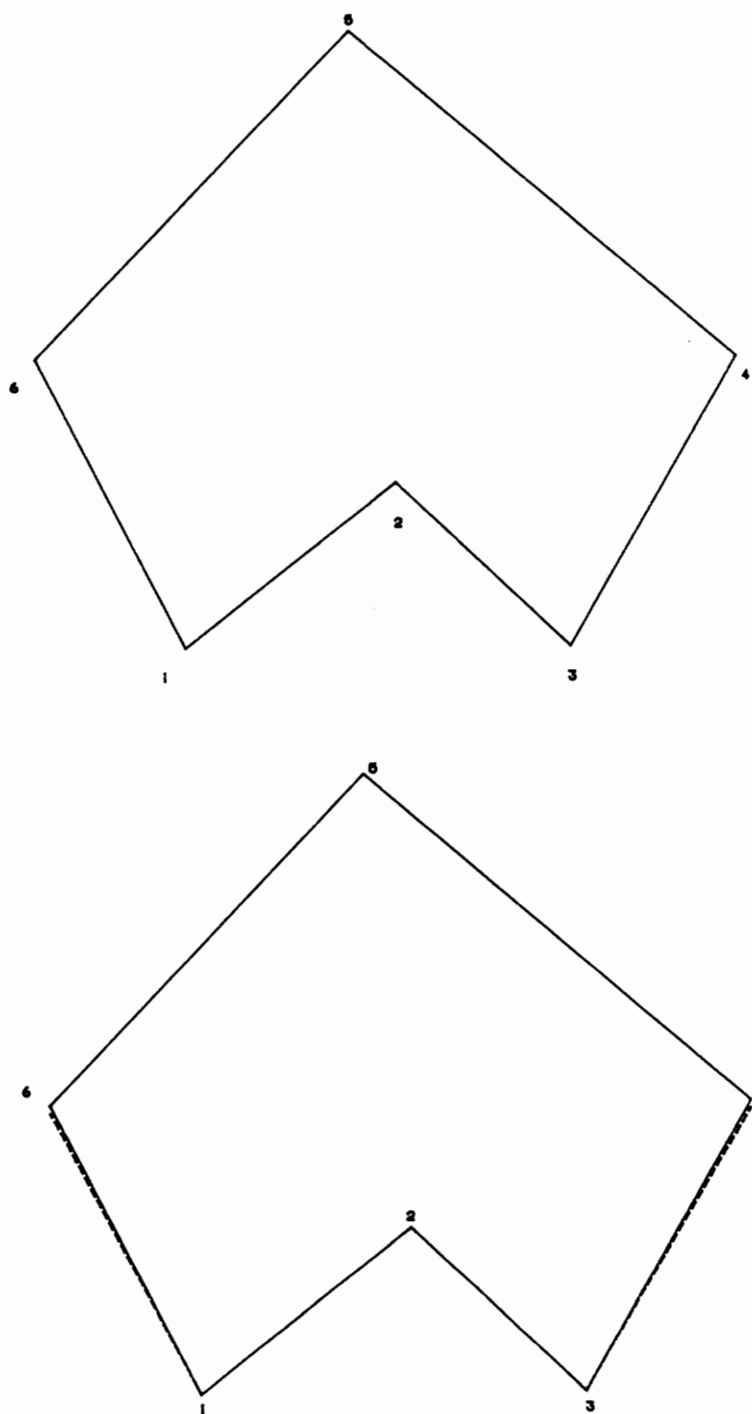


Figure 12. Edge preprocessing to maintain parity at vertices

```

/* type definition for face data */
typedef struct face_data_node {
    boolean inside;
    float intcol[3],
          speccol[3];
    surfprop surf;
    face_list *face;
    struct face_data_node *next;
} face_data;

/* type definition for edge */
typedef struct edge_node {
    face_data *face;
    short y_upper,
          y_lower;
    float x_int,
          recip_slope,
          depth,
          depth_y,
          depth_inc,
          coord[3],
          coord_inc[3],
          norm[3],
          norm_inc[3];
    struct edge_node *next;
} edge;

/* type definition for covers */
typedef struct cover_list_node {
    edge *edata;
    float depth,
          coord[3],
          coord_inc[3],
          norm[3],
          norm_inc[3];
    struct cover_list_node *next;
} cover_list;

```

Figure 13. Type definition for the edge and face list.

In device coordinates the y value increases from bottom to top, the x value increases from left to right and the z value increases from front to back. The edge table is maintained as an array of pointers to the edges starting on that particular scan line. During the visibility detection algorithm, a simplified list of the active edges at each scan line is maintained. The list is started by adding all the edges starting on the scan line to the active edge list (from the edge table). This list is then sorted by the x intersection value. Using the active edge list information we fill in the intensities of all the pixels. Then all the edges with a y-upper value less than or equal to the current scan line are deleted, and the x intersection value is updated for the next scan line. This process is repeated till the whole screen is processed.

Filling in the pixels using the active edge list information becomes complicated when we have several overlapping polygons. To implement this, a data type called “covers” is used, which maintains a list of pointers to polygons which are currently covered. Figure 13 on page 66 gives the data type for “covers”. The data type covers has a pointer to the edge rather than the face. This is done so that the covers can use the data within the edge and the face data structures. While processing the active edge list data (sorted by x intersections), at each edge, a corresponding cover is added or removed, from the cover list depending on how the parity changes at that edge. The cover list is now checked for the cover with minimum depth to give the visible polygon. The coordinate and normal data for the visible polygon are incremented and this data is passed on to the shading routine which calculates the intensity of the polygon.

8.3 Phong lighting and shading methods

The lighting and shading models implementation are very direct and straight forward. All the equations shown in the design were implemented to give the intensity of the pixel depending on the number and type of lights acting on the polygon.

One important point worth mentioning with regard to the shading model is that, **grAPHIGS** or **PHIGS+** does not provide for input of mesh information. That is, if the vertex normals of neighboring polygons do not match up or if the vertex normals of the polygons are not given then we will see a faceted rendered model instead of a smooth shaded image. The applications programmer must make sure that the vertex normals of all neighboring polygons match up. To make this process easier for the applications programmer, a utility routine to compute the average geometric normal for a polygon is provided in the list of interface routines. This can be used to find the normal for each polygon and geometric normals of neighboring polygons can be averaged to get the normal at the vertex. This normal data can be passed to the Polygon 3 with data call.

This completes the description of the implementation of a prototype of the Phong rendering process.

9.0 Results and Future Work

The Phong rendering software was successfully implemented on the IBM-RT (with 5080 head) workstation. Even though the implementation was done on the IBM-RT, the code developed is as such “device independent” and can be ported and successfully used in any UNIX environment supporting “graPHIGS”. The time required to render a scene is approximately linearly proportional to the model size and the number of pixels covered on the screen. The time is dependent on the model size because the larger the model, the greater the time required to retrieve the data from graPHIGS data structure.

Most of the original design was implemented. The graPHIGS data retrieval and data storage data structures were optimized for maximum efficiency in speed. The algorithm used for visibility detection is one of the elementary scan line techniques. The speed of the visibility detection may be further improved by testing out other methods. The lighting and shading models were implemented without any problems.

In the implementation of the prototype the development of the clipping algorithm and the back face processing were skipped. These can be incorporated into the rendering scheme without too much effort.

Although the intensity calculation is done using the Phong shading models, the 128 color look-up table (with graPHIGS) and the 16 shades per gun limitation on the 5080 gives severe banding effects on the rendered model. A good quantization technique for finding the best possible colors to fill the color table and map the intensities to the closest possible colors in the look-up table is required. This in conjunction with dithering will enhance the results to a considerable degree. Without these, the Phong rendering scheme is an overkill for the 5080 heads.

Currently, work is being done at the CAD/CAM laboratory of VPI & SU to develop a quantization scheme for limited color display of scenes. Little work has been done on the quantization of multi-dimensional data. Heckbert [Heck82] discusses methods to quantize images. He discusses ways to break up the color cube into segments based on the color statistics of the image and find the best possible colors for the look-up table and quantize the image. This forms the basis for the current research being done.

Once a good quantization and dithering technique can be coded and the image produced by the Phong rendering software can be encoded using this technique, the images produced by the software can be successfully used on the IBM 5080s by application programmers.

The work described in this thesis was funded by an IBM research grant, and the prototype developed is part of the deliverables of the research project.

10.0 References

- [Blin77] Blinn, J. F., "Models of Light Reflection for Computer Synthesized Pictures," *Proceedings of SIGGRAPH '77, Computer Graphics*, 11, No. 2, 1977, pp.192-198
- [Bouk70] Bouknight, W. J., "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations," *Communications of the ACM*, 13, No. 9, Sept. 1970, pp.527-536
- [Catm74] Catmull, E. E., "A Subdivision Algorithm for Computer Display of Curved Surfaces," Ph.D. Dissertation, University of Utah, Salt Lake City, Utah, Dec. 1974
- [Catm75] Catmull, E. E., "Computer Display of Curved Surfaces," *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures*, May 1975, 11
- [Clar79] Clark, J. H., "A Fast Scan-Line Algorithm for Rendering Parametric Surfaces" *Proceedings of SIGGRAPH '79, Computer Graphics*, 13, No. 2, Aug. 1979, pp.174

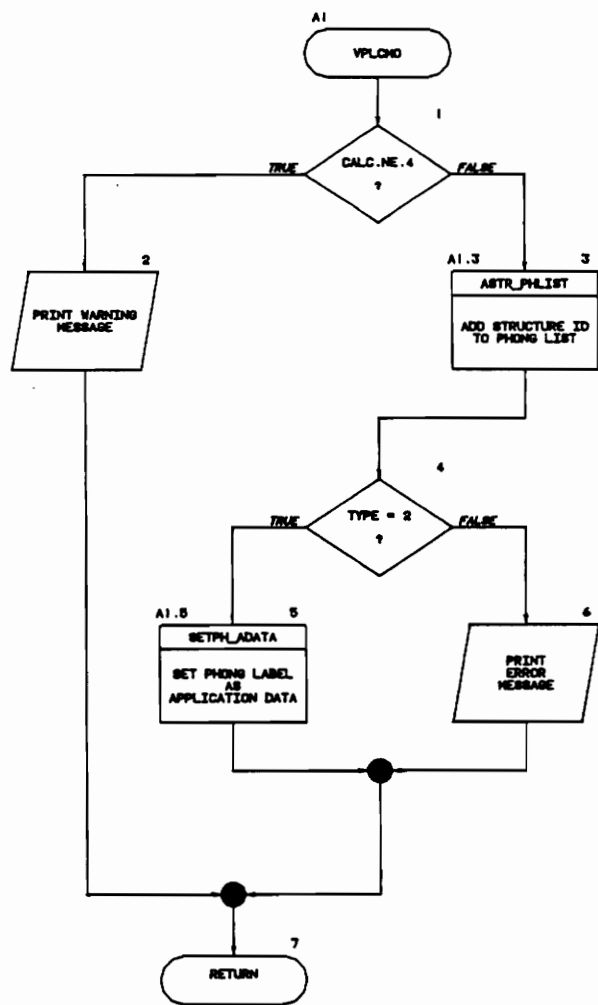
- [Cook82] Cook, R. L. and K. E. Torrance, "A Reflectance Model for Computer Graphics," *ACM Transactions on Graphics*, 1, No. 1, Jan. 1982, pp.7-24
- [Croc84] Croc, G. A., "Invisibility Coherence for Faster Scan-Line Hidden Surface Algorithms," *Proceedings of SIGGRAPH '84, Computer Graphics*, 18, No. 3, July 1984, pp.96-102
- [Crow77] Crow, F. C., "Shadow Algorithms for Computer Graphics," *Proceedings of SIGGRAPH '79, Computer Graphics*, 11, No. 2, 1977, pp.242-248
- [Duff79] Duff, T., "Smoothly Shaded Rendering of Polyhedral Objects on Raster Displays," *Proceedings of SIGGRAPH '79, Computer Graphics*, 13, No. 2, Aug. 1979, pp.270-275
- [Fium83] Fiume, E., A. Fournier, and L. Rudolph, "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer," *Proceedings of SIGGRAPH '83, Computer Graphics*, 17, No. 3, July 1983, pp.141-150
- [Gour71] Gouraud, H., "Computer Display of Curved Surfaces," *IEEE Transactions on Computers*, C-20, 6, June 1971, pp.623-629
- [Hall83] Hall, R. A. and D. P. Greenberg, "Computer Display of Curved Surfaces," *A Testbed for Realistic Image Synthesis*, 3, No. 8, Nov. 1983, pp.10-20
- [Heck82] Heckbert, P. S., "Color Image Quantization for Frame Buffer Display" *Proceedings of SIGGRAPH '82, Computer Graphics*, 16, No. 3, July 1982, pp.297-307
- [Lane79] Lane, J. M. and L. Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing*, 11, 1979, pp.270-297

- [Lane80] Lane, J. M., L. Carpenter, T. Whitted and J.F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Communications of the ACM*, 23, No. 1, Jan. 1980, pp.23-34
- [Max86] Max, N. L., "Atmospheric Illumination and Shadows," *Proceedings of SIGGRAPH '86, Computer Graphics*, 20, No. 4, Aug. 1986, pp.117-124
- [Phon75] Bui-Tuong, Phong, "Illumination for Computer Generated Pictures," *Communications of the ACM*, 20, No. 2, Feb. 1977, pp.100-106
- [Schw82] Schwietzer, D. and E. S. Cobb, "Scanline Rendering of Parametric Surfaces," *Proceedings of SIGGRAPH '82, Computer Graphics*, 15, No. 3, Aug. 1981, pp.17-27
- [Suth74] Sutherland, I.E., R.F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, 6, No. 1, 1974, pp.1-55
- [Whit80] Whitted, T., "An Improved Illumination Model for Shaded Displays," *Communications of the ACM*, 23, No. 6, June 1980, pp.343-349
- [Whit82] Whitted, T., and D.M. Weimer, "A Software Testbed for the Development of 3D Raster Graphics Systems" *ACM Transactions on Graphics*, 1, No. 1, Jan. 1982, pp.43-58

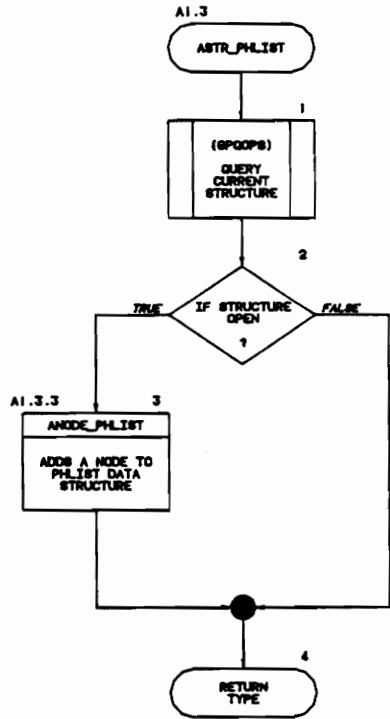
Appendix A. Flowcharts

This section contains the program specification flowcharts for the designed software. The flowcharts are arranged according to module numbers. The application programmer access functions are placed in the beginning with module numbering starting with “A”. Then the rendering scheme flowcharts are module numbered starting with “P”, and finally the utility routines starting with “U” are placed at the very end.

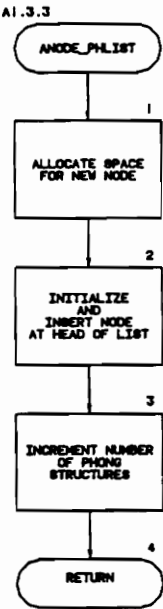
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPLCMD	MODULE: A1
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE ALLOWS THE USER TO SET THE LIGHTING CALCULATION MODE
DATE:	3/9/90	



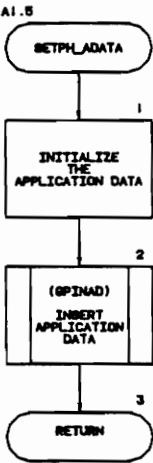
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: ASTR_PHLIST	MODULE: A1.3
DESIGNED BY: KRISHNAN KOLADY	NOTE: ADDS THE CURRENT STRUCTURE ID TO THE PHONG LIST
DATE: 3/9/90	



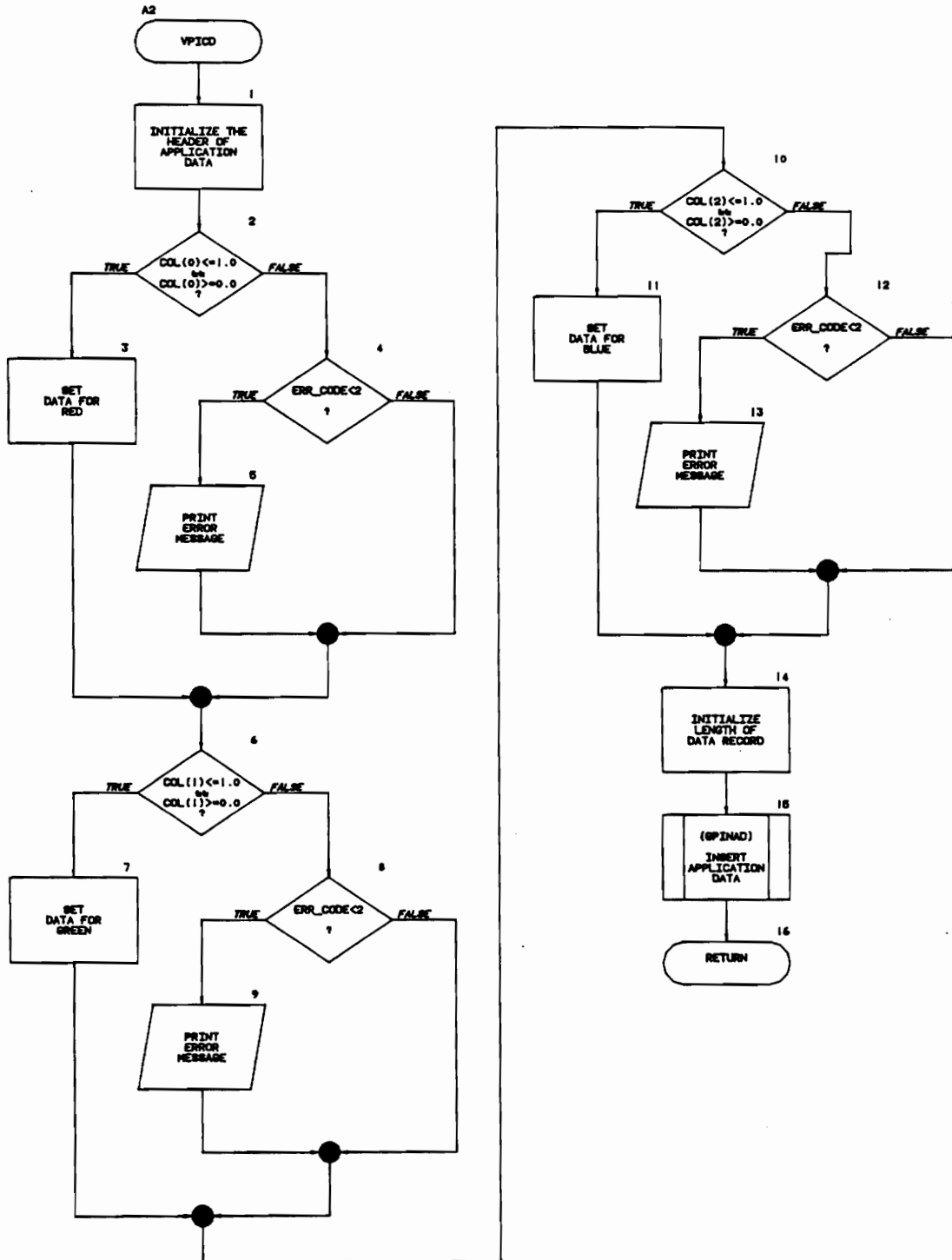
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	ANODE_PHLIST	MODULE: A1.3.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: ADDS THE CURRENT STRUCTURE IDENTIFIER AS A NODE IN THE PHONG LIST
DATE:	3/9/90	



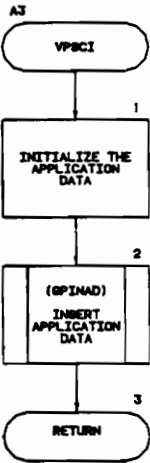
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	SETPH_ADATA	MODULE: A1.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: SETS A PHONG SHADING STRUCTURE LABEL WITHIN THE CURRENT STRUCTURE
DATE:	3/9/90	



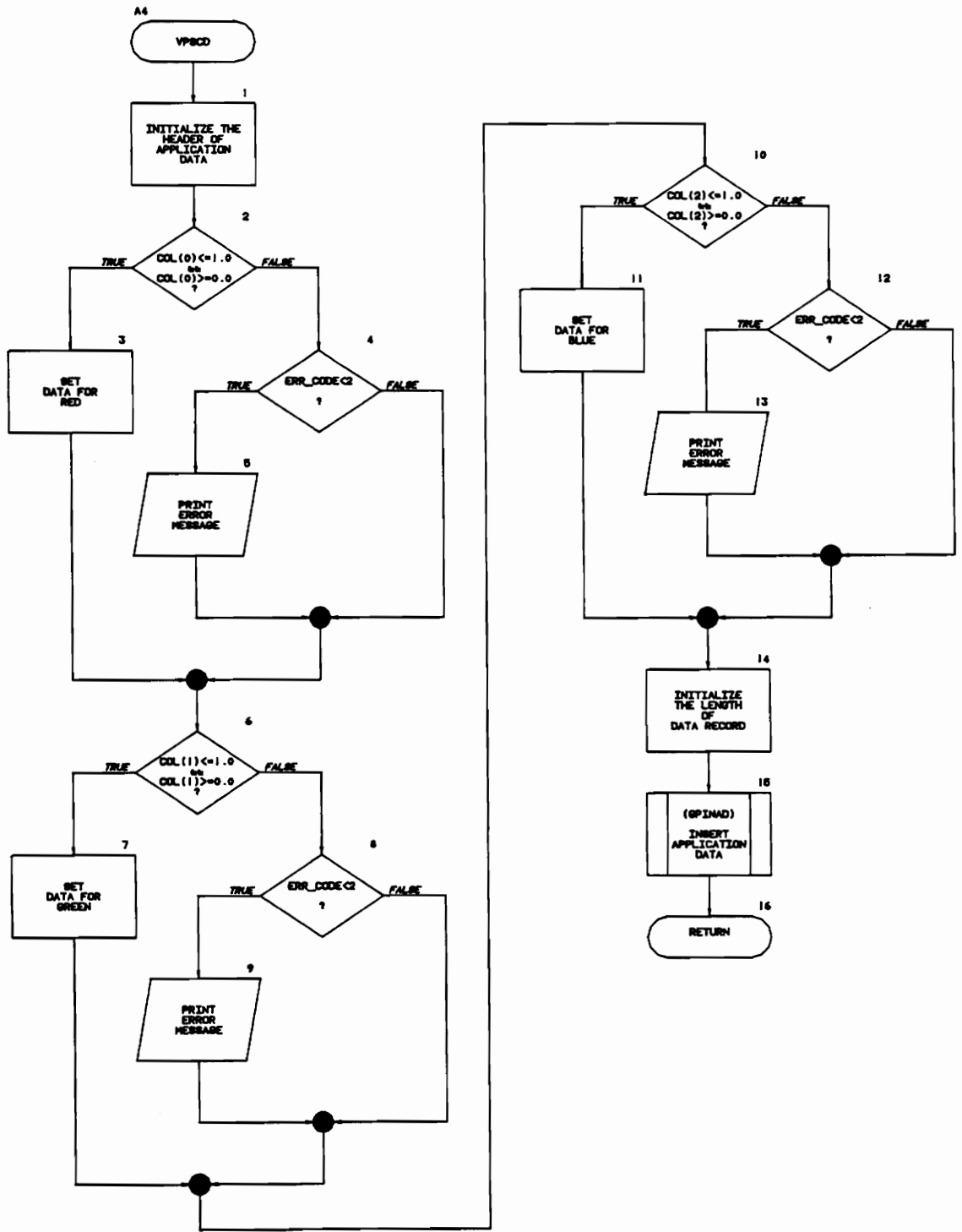
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: VPICD	MODULE: A2
DESIGNED BY: KRISHNAN KOLADY	NOTE: API ROUTINE SET INTERIOR COLOR DIRECT STATE REQUIRED-(PHOP,*,STOP,*)
DATE: 3/23/90	



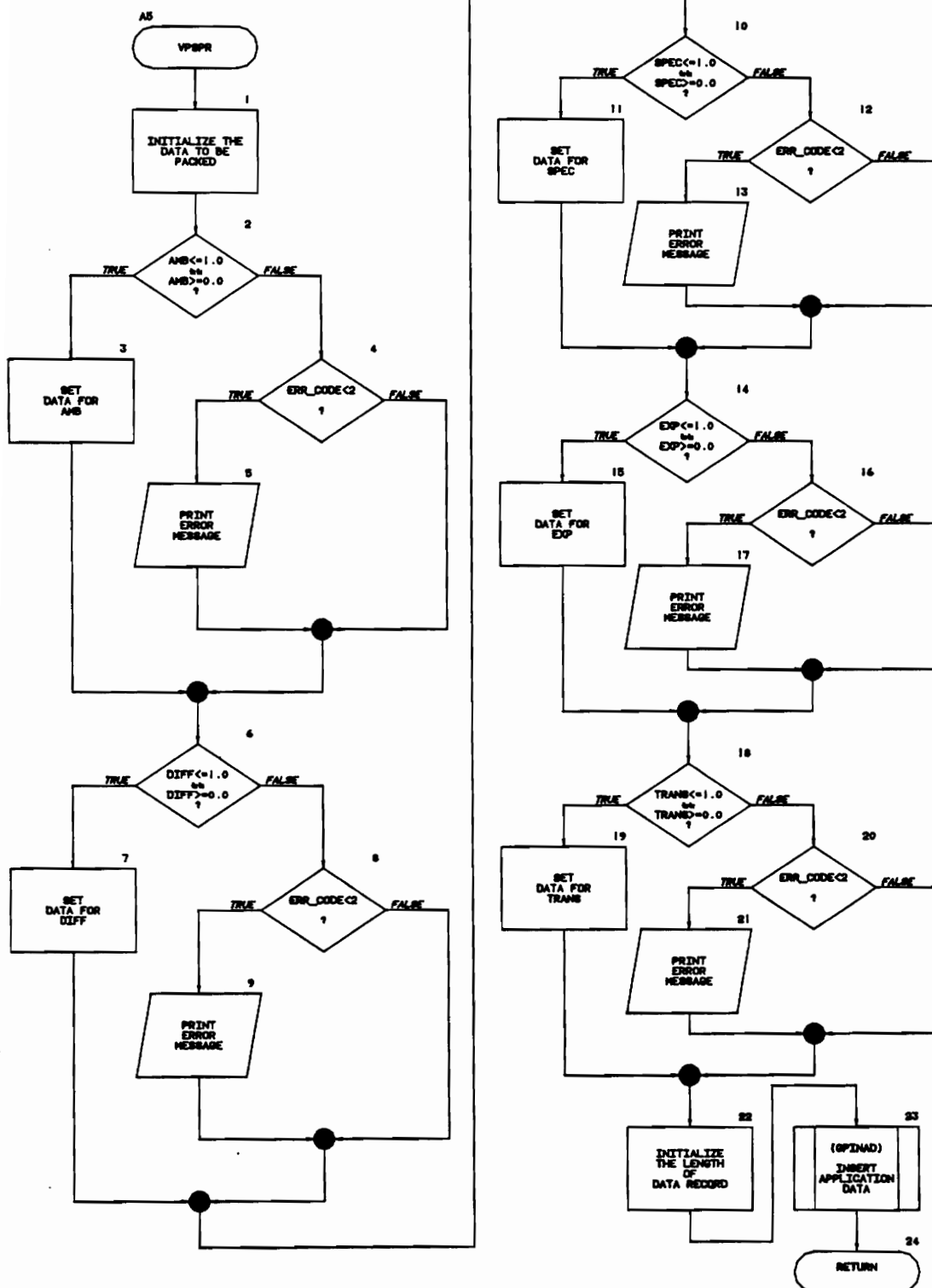
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME :	VPSCI	MODULE : A3
DESIGNED BY :	KRISHNAN KOLADY	NOTE: API ROUTINE SET SPECULAR COLOR INDEX STATE REQUIRED- (PHOP, •, STOP, •)
DATE :	3/23/90	



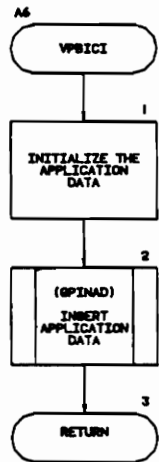
PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPSCD	MODULE: A4
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET SPECULAR COLOR DIRECT STATE REQUIRED-(PHOP,*,STOP,*)
DATE:	3/23/90	



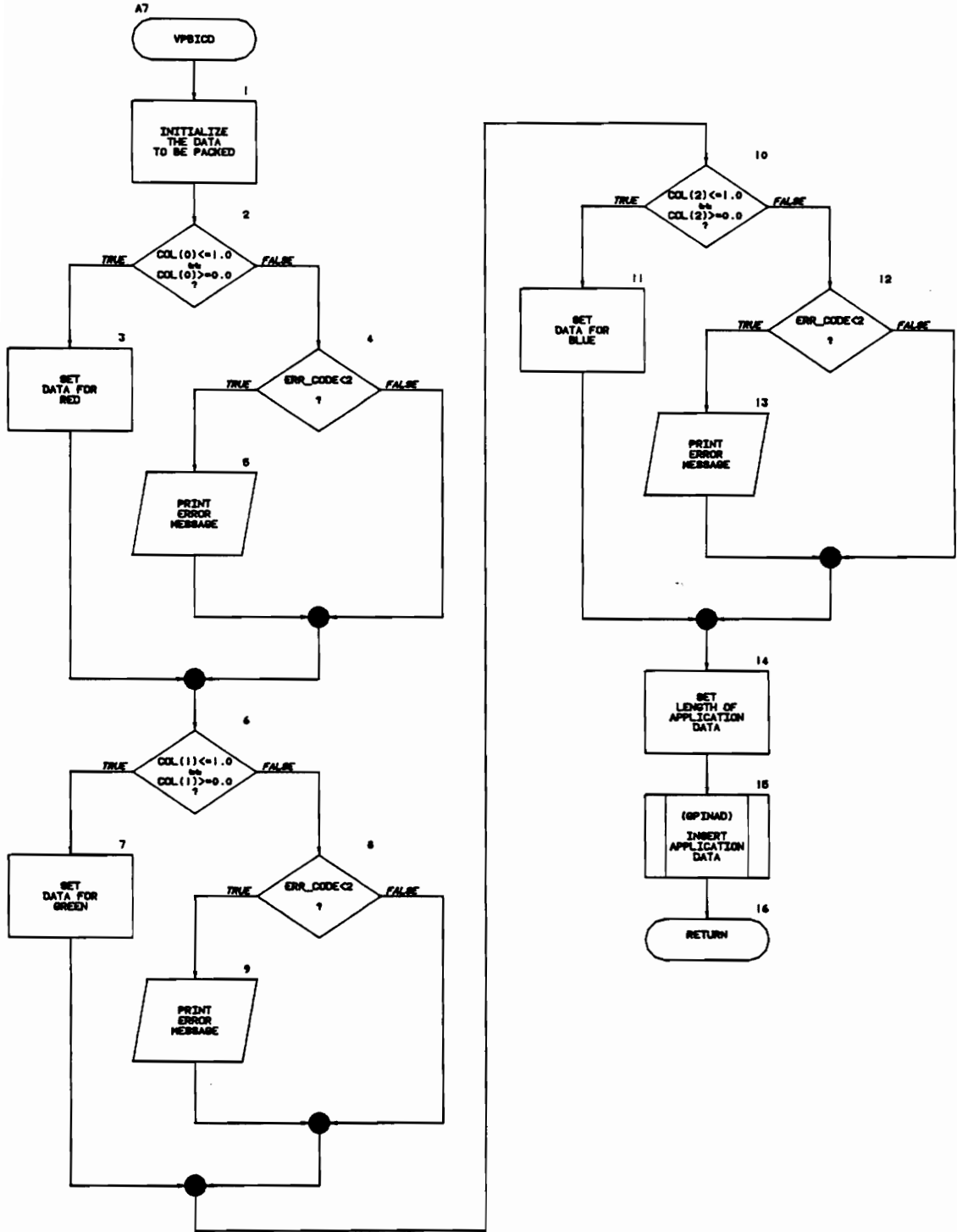
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: VPSPR	MODULE: A5
DESIGNED BY: KRISHNAN KOLADY	NOTE: API ROUTINE SET SURFACE PROPERTIES STATE REQUIRED- (PHOP,*,STOP,*)
DATE: 3/23/90	



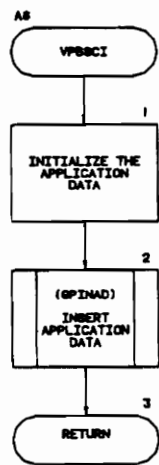
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPBICI	MODULE: A6
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE : SET BACK INTERIOR COLOR INDEX STATE REQUIRED- (PHOP, ●, STOP, ●)
DATE:	3/23/90	



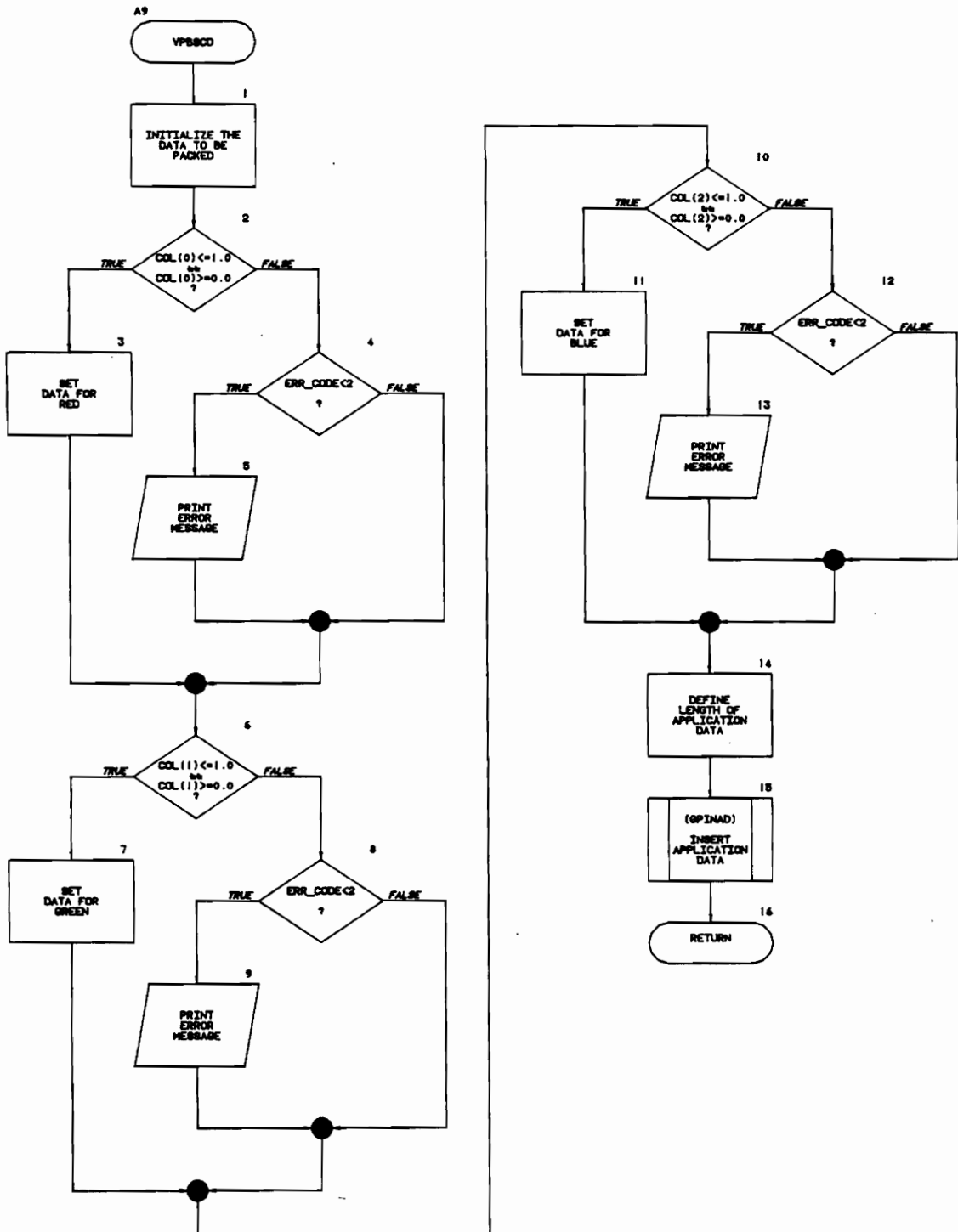
VF PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPBICD	MODULE: A7
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET BACK INTERIOR COLOR DIRECT STATE REQUIRED-(PHOP, *, STOP, *)
DATE:	3/23/90	



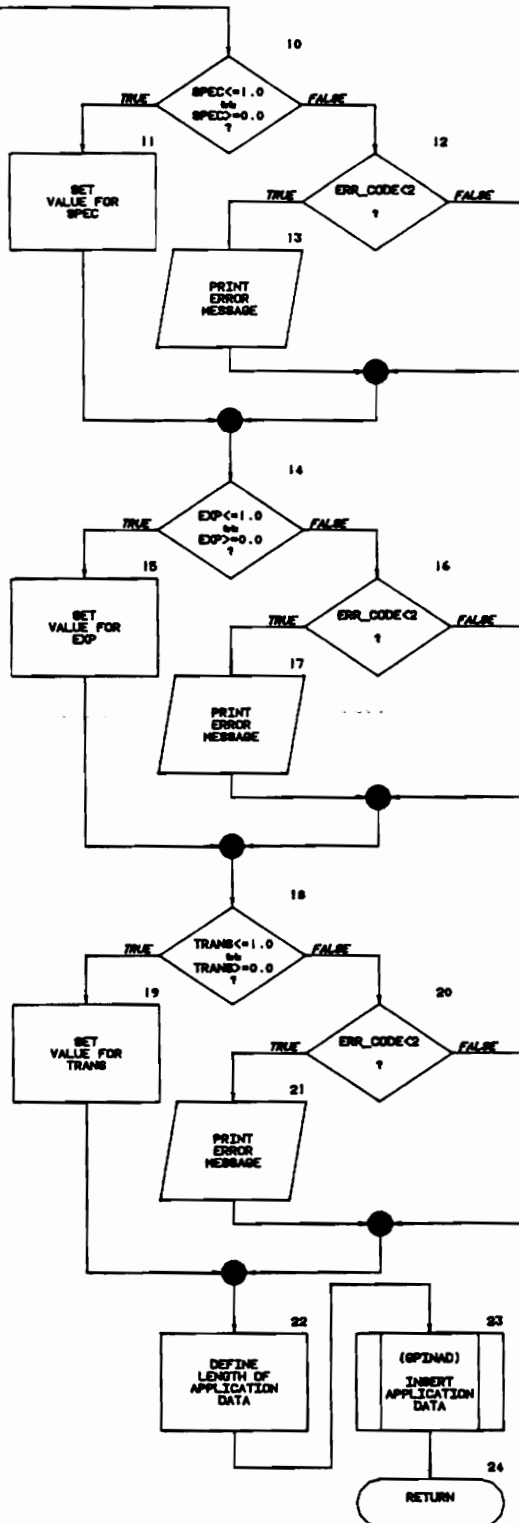
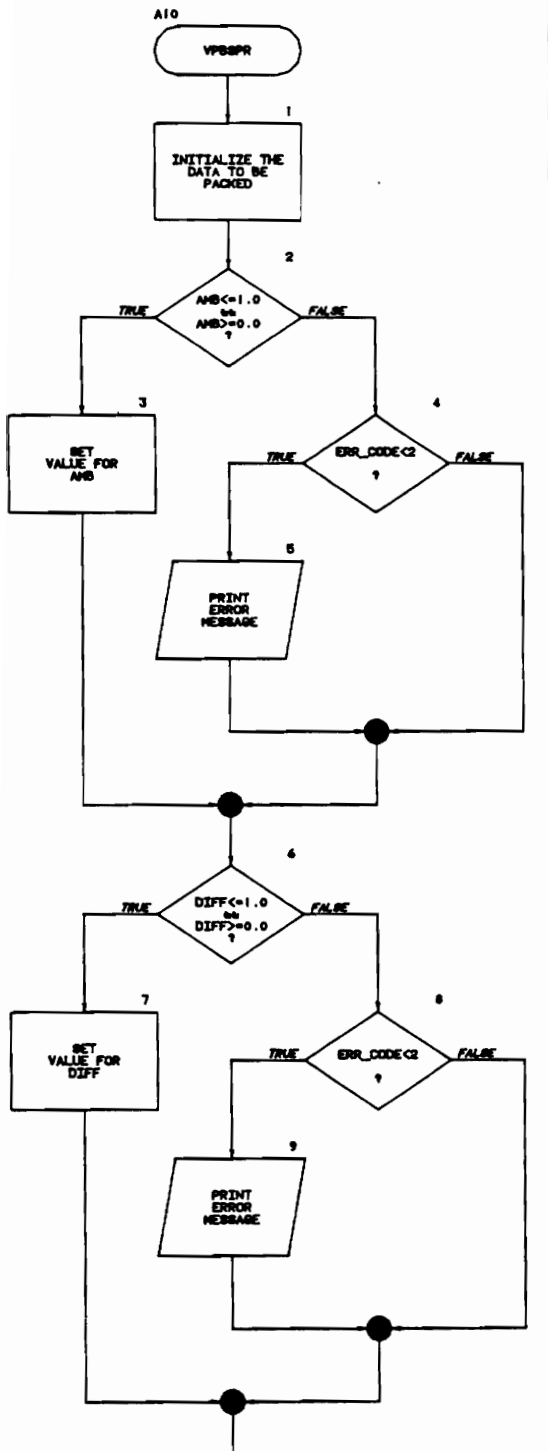
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	VPBSCI	MODULE: A8
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE . SET BACK SPECULAR COLOR INDEX STATE REQUIRED-(PHOP,*,STOP,*)
DATE:	3/23/90	



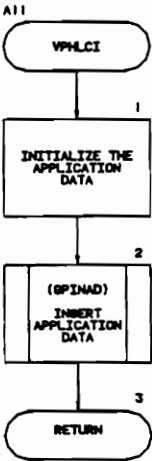
PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPBSCD	MODULE: A9
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET BACK SPECULAR COLOR DIRECT STATE REQUIRED- (PHOP, *, STOP, *)
DATE:	3/23/90	



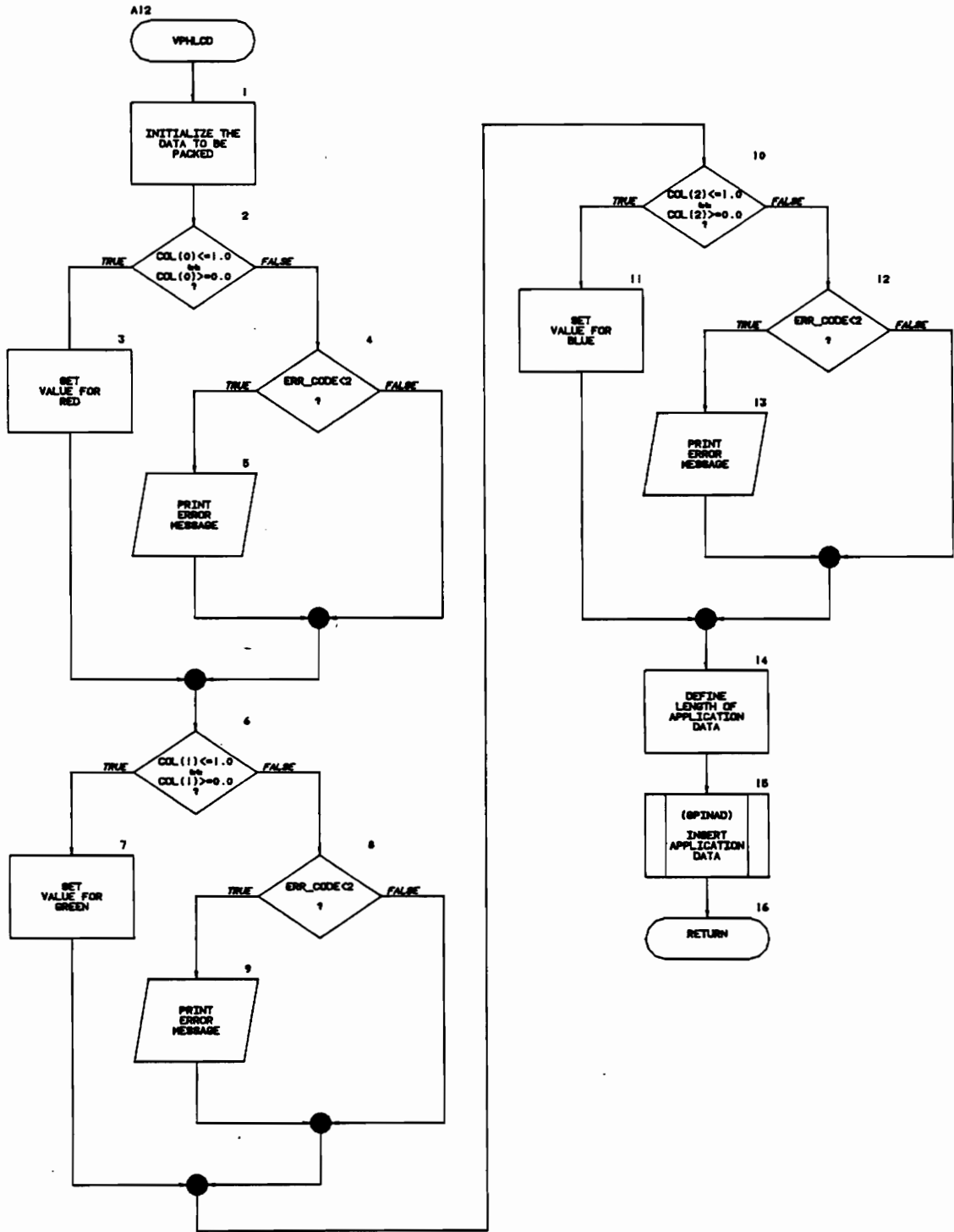
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: VPBSPR	MODULE: A10
DESIGNED BY: KRISHNAN KOLADY	NOTE: API ROUTINE SET BACK SURFACE PROPERTIES STATE REQUIRED- (PHOP, *, STOP, *)
DATE: 3/23/90	



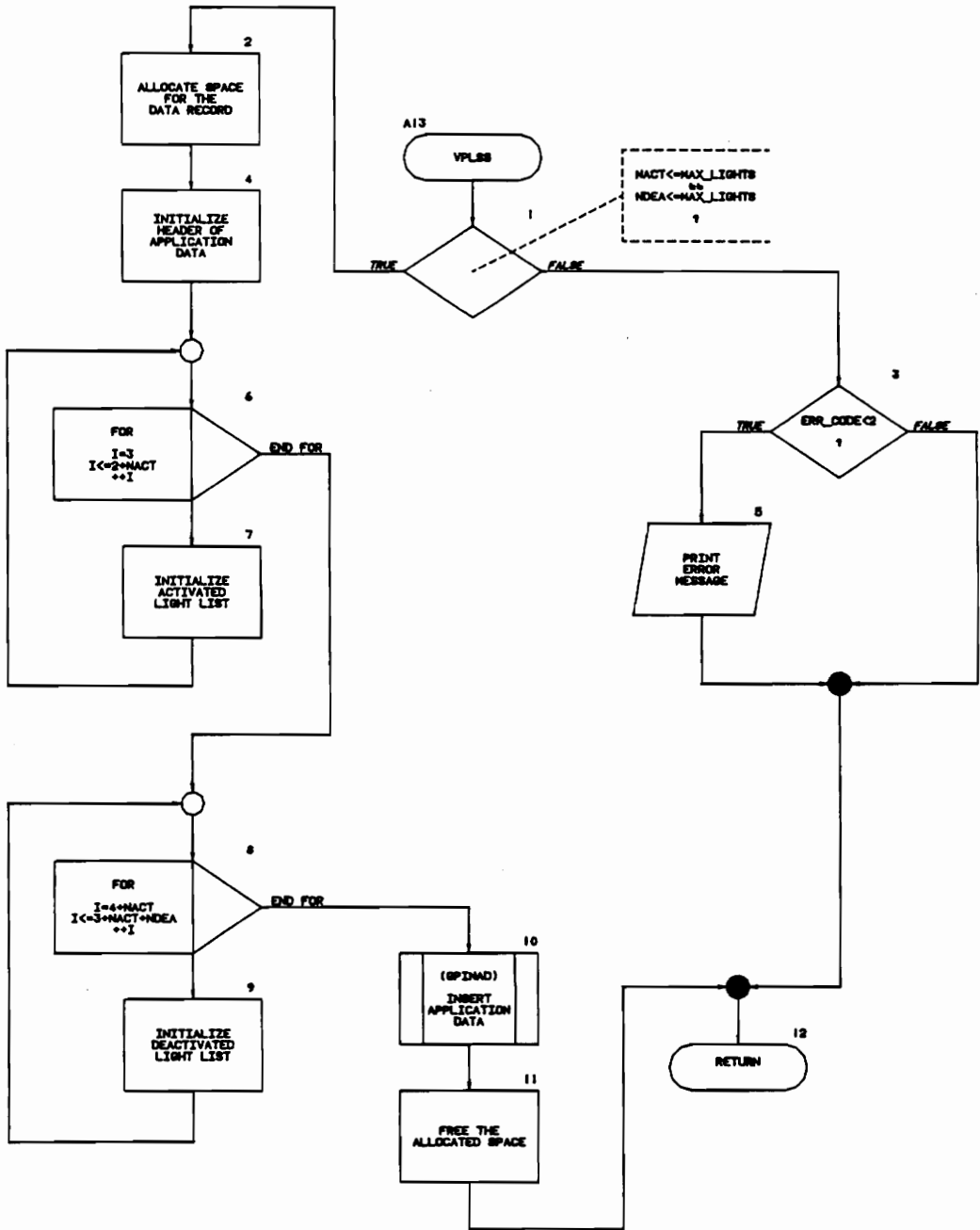
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	VPHLCI	MODULE: A11
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET HIGHLIGHTING COLOR INDEX STATE REQUIRED- (PHOP,*,STOP,*)
DATE:	3/23/90	



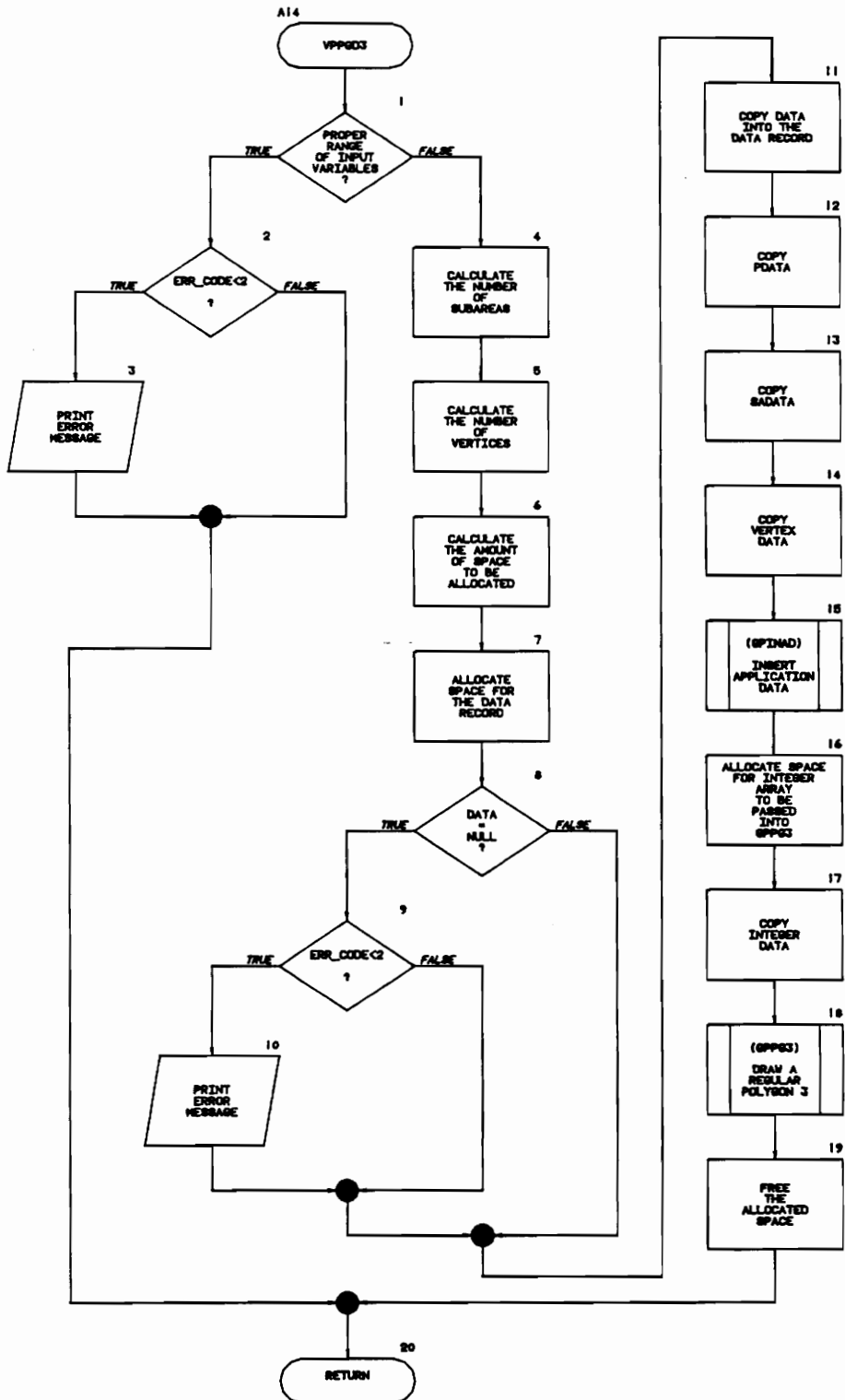
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPHLCD	MODULE: A12
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET HIGHLIGHTING COLOR DIRECT STATE REQUIRED- (PHOP,*,STOP,*)
DATE:	3/23/90	



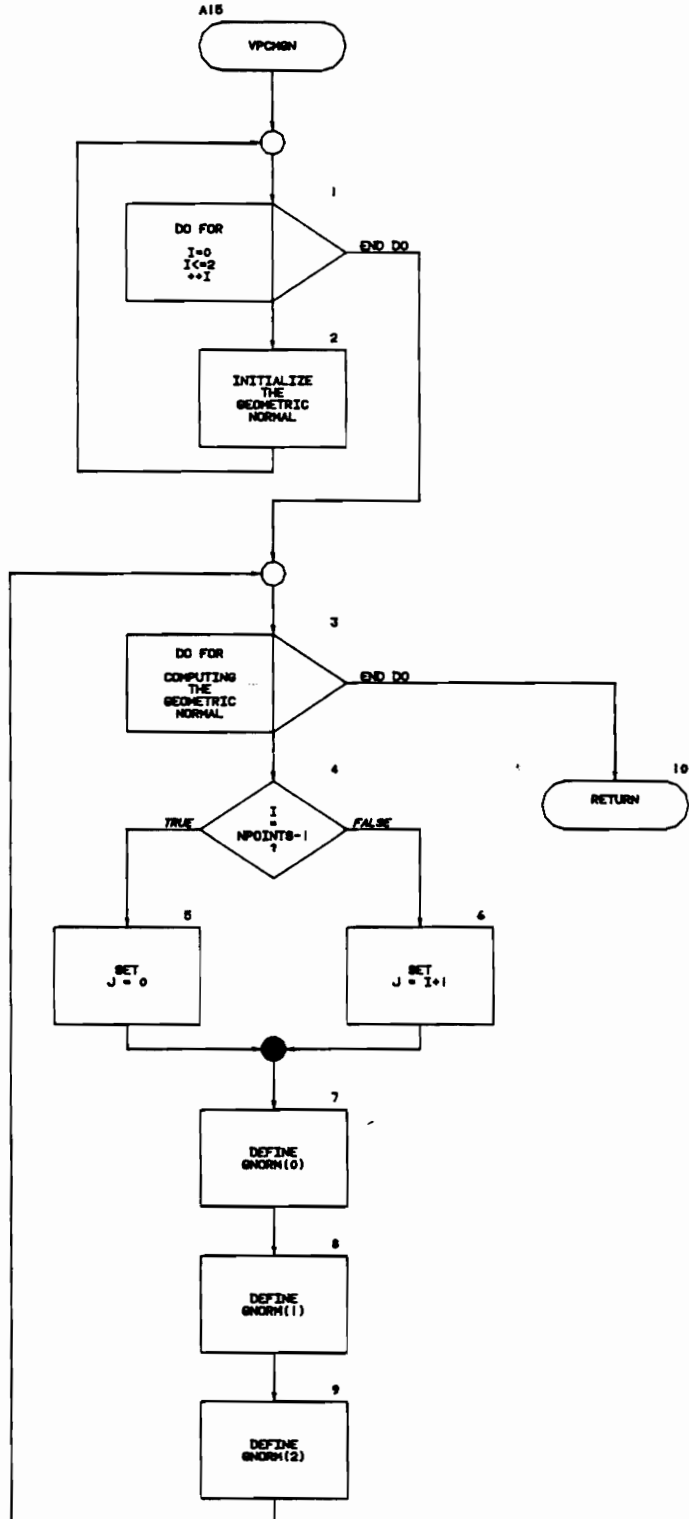
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPLSS	MODULE: A13
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET LIGHT SOURCE STATE STATE REQUIRED-(PHOP,*,STOP,*)
DATE:	3/23/90	



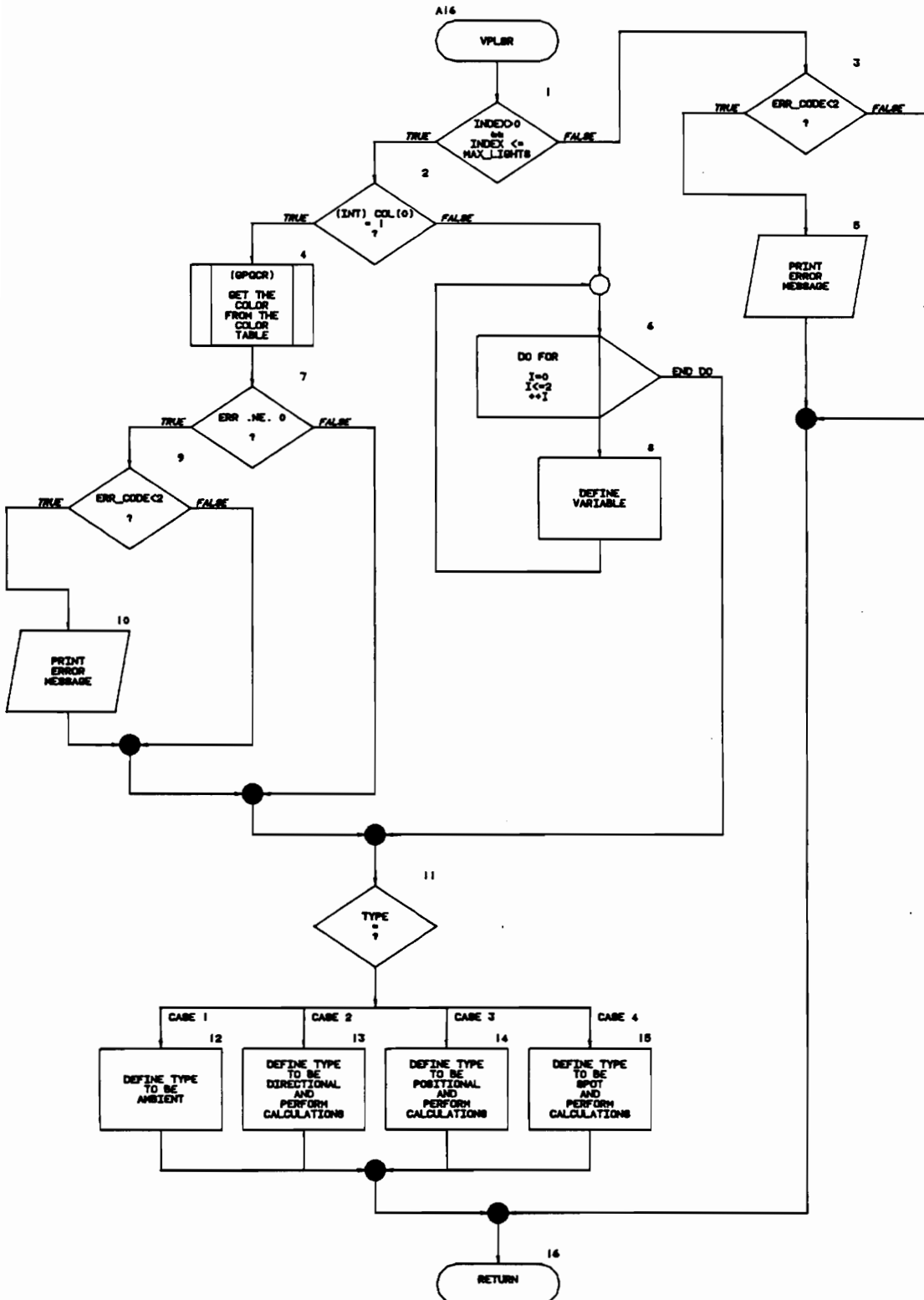
VF PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPPGD3	MODULE: A14
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE - VPPGD3 POLYGON 3 WITH DATA STATE REQUIRED - (PHOP,*,STOP,*)
DATE:	5/24/90	



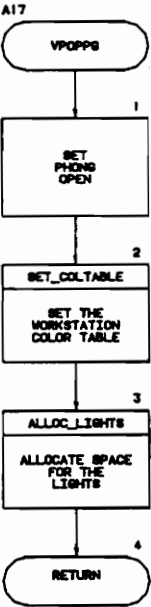
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPCMGN	MODULE: A15
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE - VPCMGN COMPUTE GEOMETRIC NORMAL STATE REQUIRED - (PHOP, ●, ●, ●)
DATE:	5/24/90	



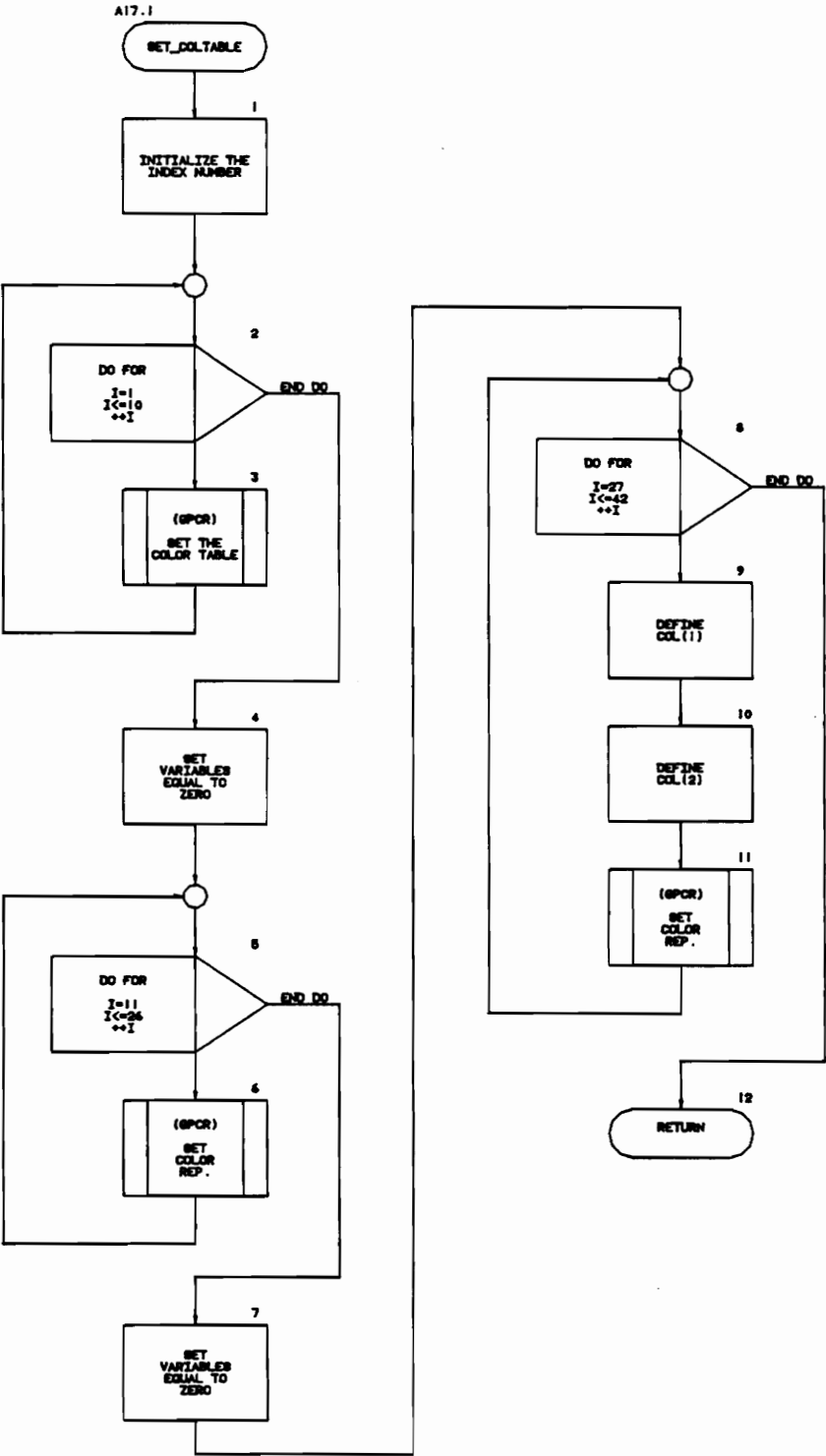
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: VPLSR	MODULE: A16
DESIGNED BY: KRISHNAN KOLADY	NOTE: API ROUTINE - VPLSR SET LIGHT SOURCE REPRESENTATION STATE REQUIRED - (PHOP, •, •, •)
DATE: 6/1/90	



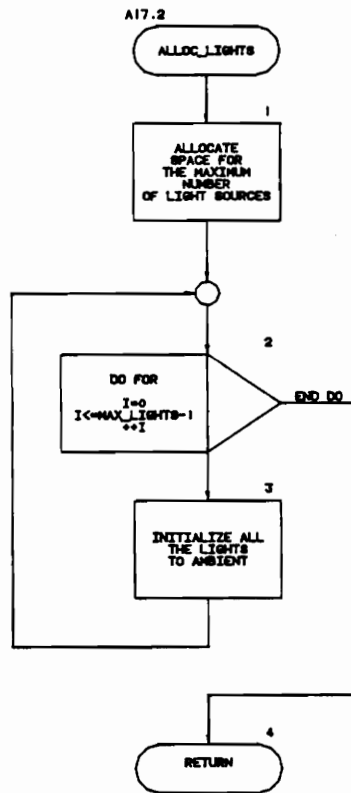
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPOPPG	MODULE: A17
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE - VPOPPG OPEN PHONG SHADING API STATE REQUIRED - (PHOP, WSOP •, •)
DATE:	5/30/90	



<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>	
MODULE NAME: SET_COLTABLE	MODULE: A17.1
DESIGNED BY: KRISHNAN KOLADY	NOTE: INITIALIZES (1 - 125) COLORS OF THE WORKSTATION COLOR TABLE
DATE: 5/30/90	



VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: ALLOC_LIGHTS	MODULE: A17.2
DESIGNED BY: KRISHNAN KOLADY	NOTE: ALLOCATES THE SPACE FOR THE LIGHTS DATA STRUCTURE
DATE: 6/1/90	



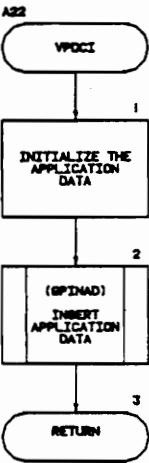
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	VPCLPG	MODULE: A18
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE - VPCLPG OPEN PHONG SHADING API STATE REQUIRED - (PHOP, WSOP •, •)
DATE:	6/1/90	



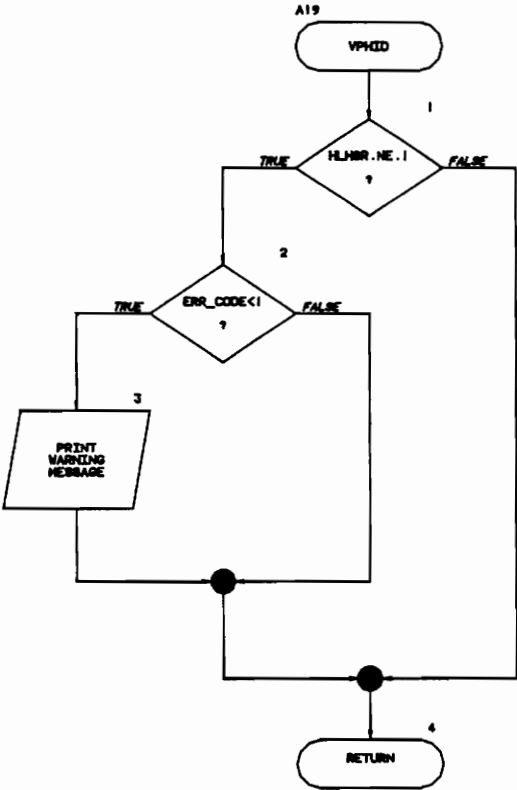
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	FREE_LIGHTS	MODULE: A18.1
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FREES THE LIGHT SPACE ALLOCATED
DATE:	6/1/90	



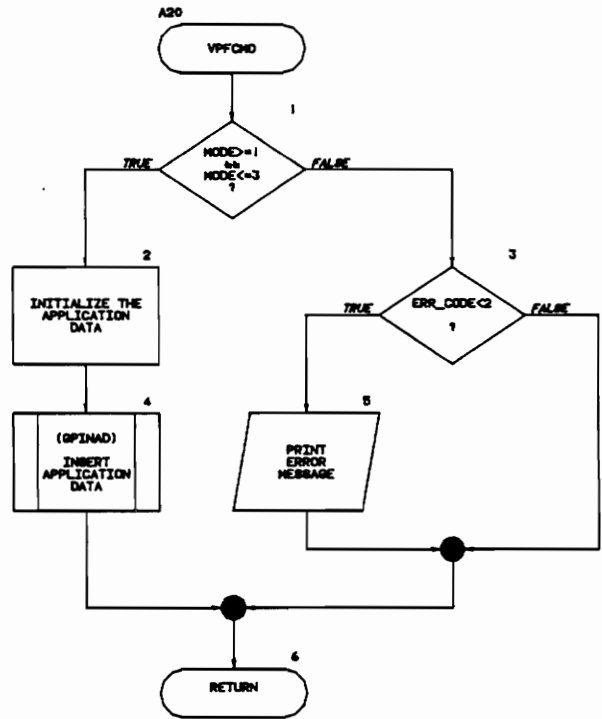
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>		
MODULE NAME:	VPDCI	MODULE: A22
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET DEPTH CUE INDEX STATE REQUIRED- (PHOP, ●, STOP, ●)
DATE:	3/23/90	



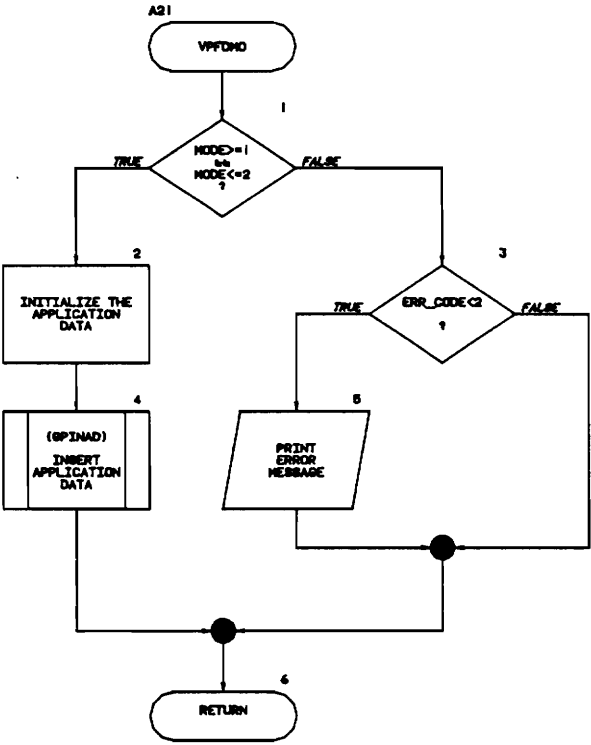
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: VPHID	MODULE: A19
DESIGNED BY: KRISHNAN KOLADY	NOTE: API ROUTINE SET HLHSR IDENTIFIER STATE REQUIRED-(PHOP,•,STOP,•)
DATE: 3/23/90	



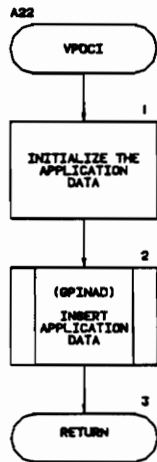
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPFCMO	MODULE: A20
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET FACE CULLING MODE STATE REQUIRED-(PHOP,*,STOP,*)
DATE:	3/23/90	



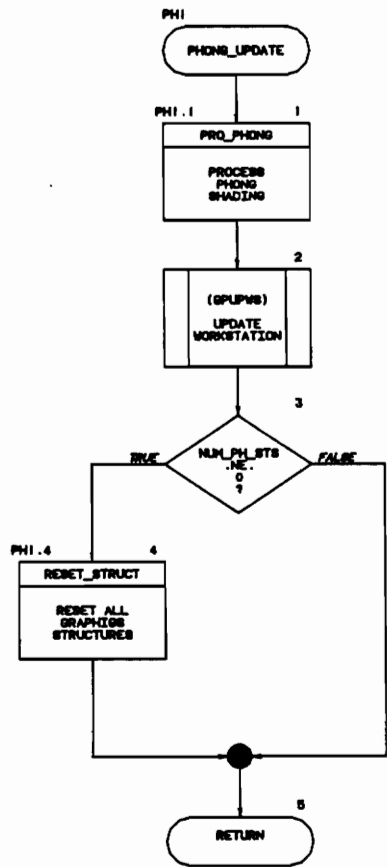
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: VPFDMO	MODULE: A21
DESIGNED BY: KRISHNAN KOLADY	NOTE: API ROUTINE SET FACE DISTINGUISHING MODE STATE REQUIRED- (PHOP,*,STOP,*)
DATE: 3/23/90	



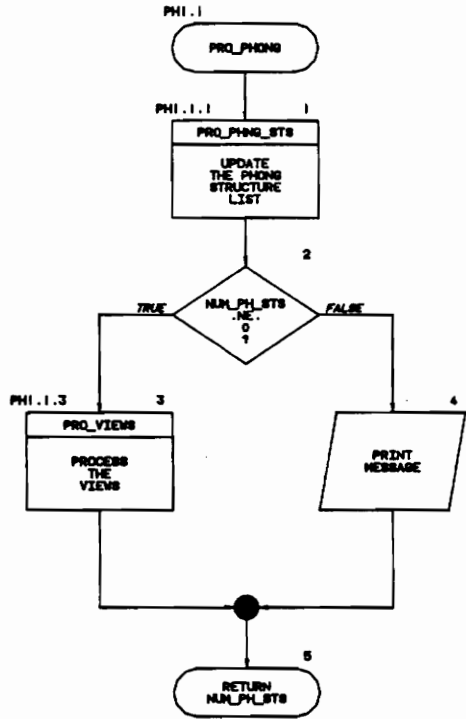
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VPOCI	MODULE: A22
DESIGNED BY:	KRISHNAN KOLADY	NOTE: API ROUTINE SET DEPTH CUE INDEX STATE REQUIRED- (PHOP, ●, STOP, ●)
DATE:	3/23/90	



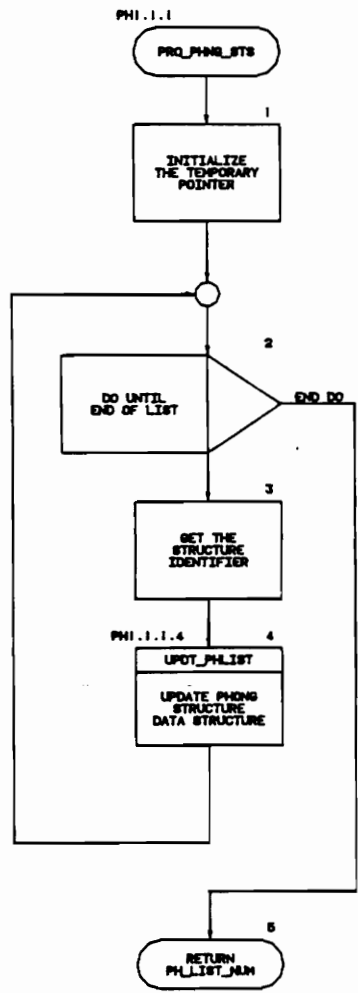
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: PHONG_UPDATE	MODULE: PHI
DESIGNED BY: KRISHNAN KOLADY	NOTE: AN UPDATE CALL TO INITIATE THE PHONG RENDERING ALGORITHM
DATE: 3/9/90	



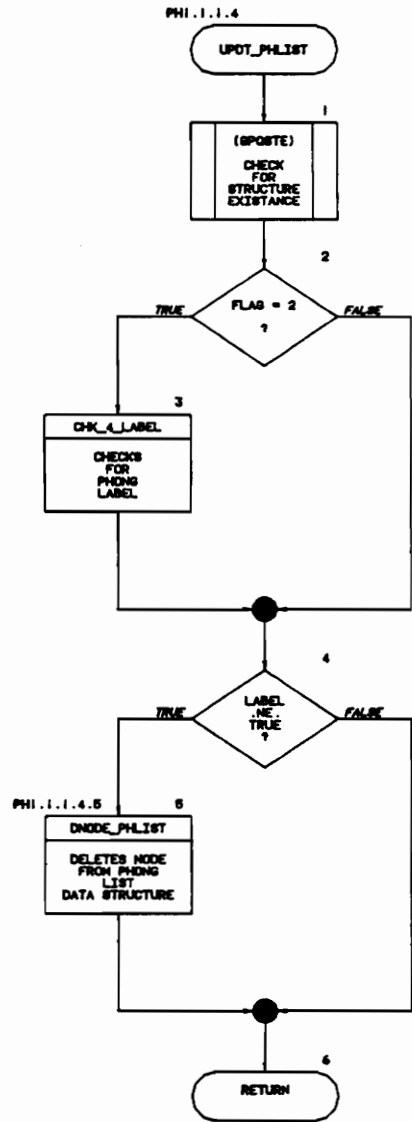
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	PRO_PHONG	MODULE: PHI.1
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES PHONG STRUCTURE TO GET STRUCTURE INFO, CONTENT, AND DATA AND RENDERS THE SCENE
DATE:	3/9/90	



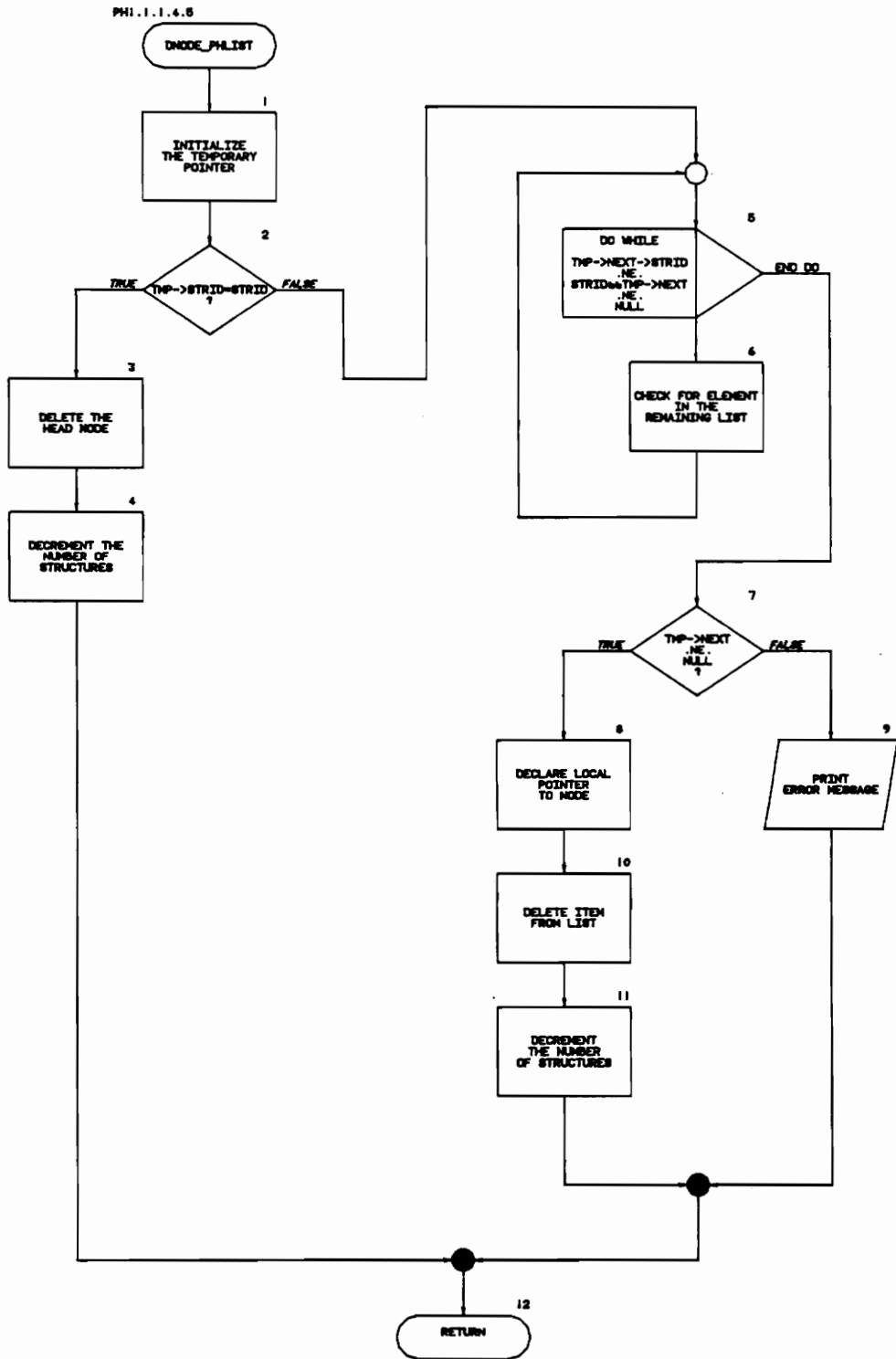
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: PRO_PHNG_STS		MODULE: PHI.1.1	
DESIGNED BY: KRISHNAN KOLADY		NOTE: UPDATES THE PHONG STRUCTURE LIST IN THE DATABASE	
DATE: 3/9/90			



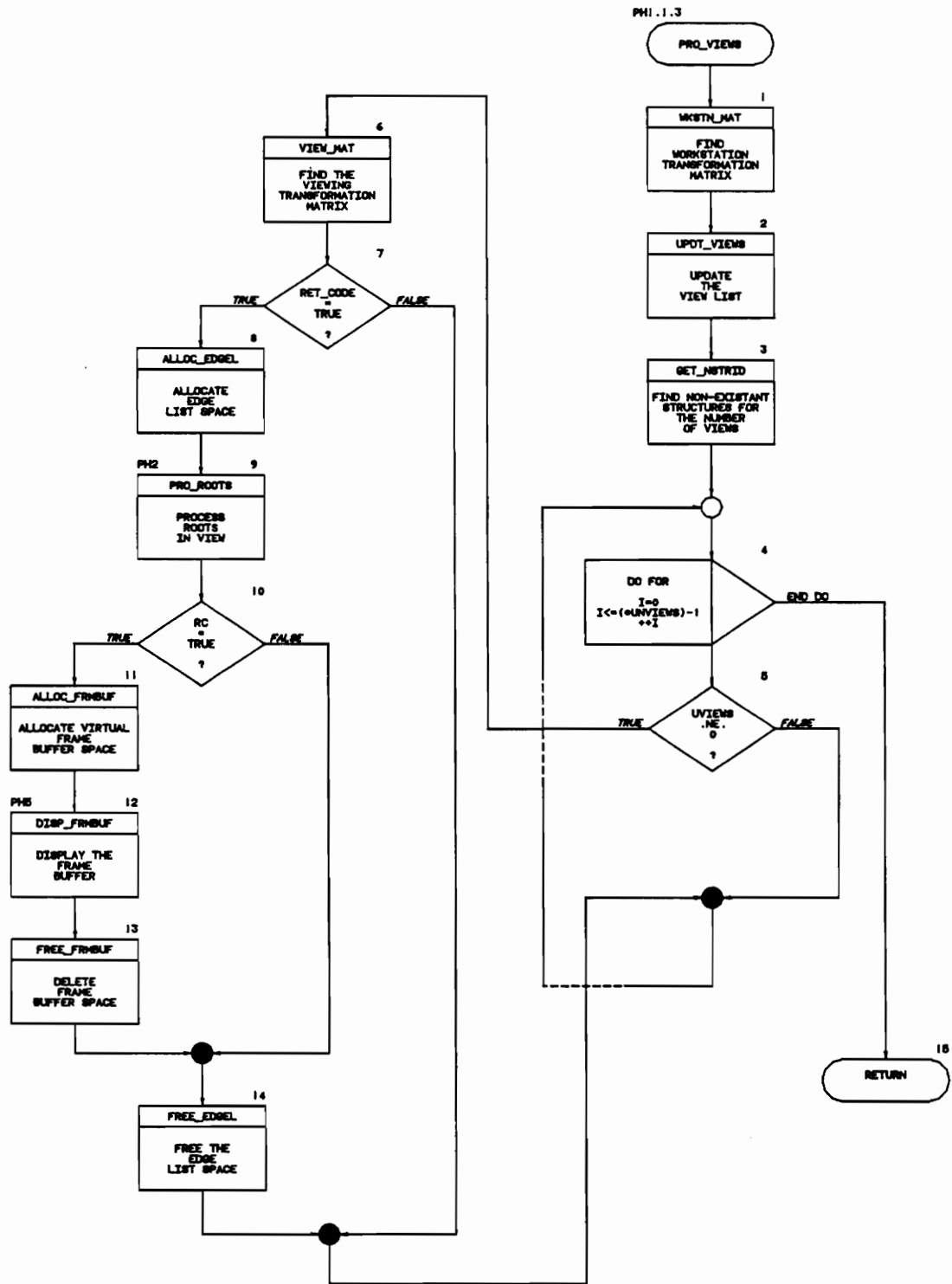
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: UPDT_PHLIST	MODULE: PHI.1.1.4
DESIGNED BY: KRISHNAN KOLADY	NOTE: CHECKS THE CURRENT STRUCTURE FOR THE PHONG LABEL AND UPDATES THE DATA STRUCTURE
DATE: 3/9/90	



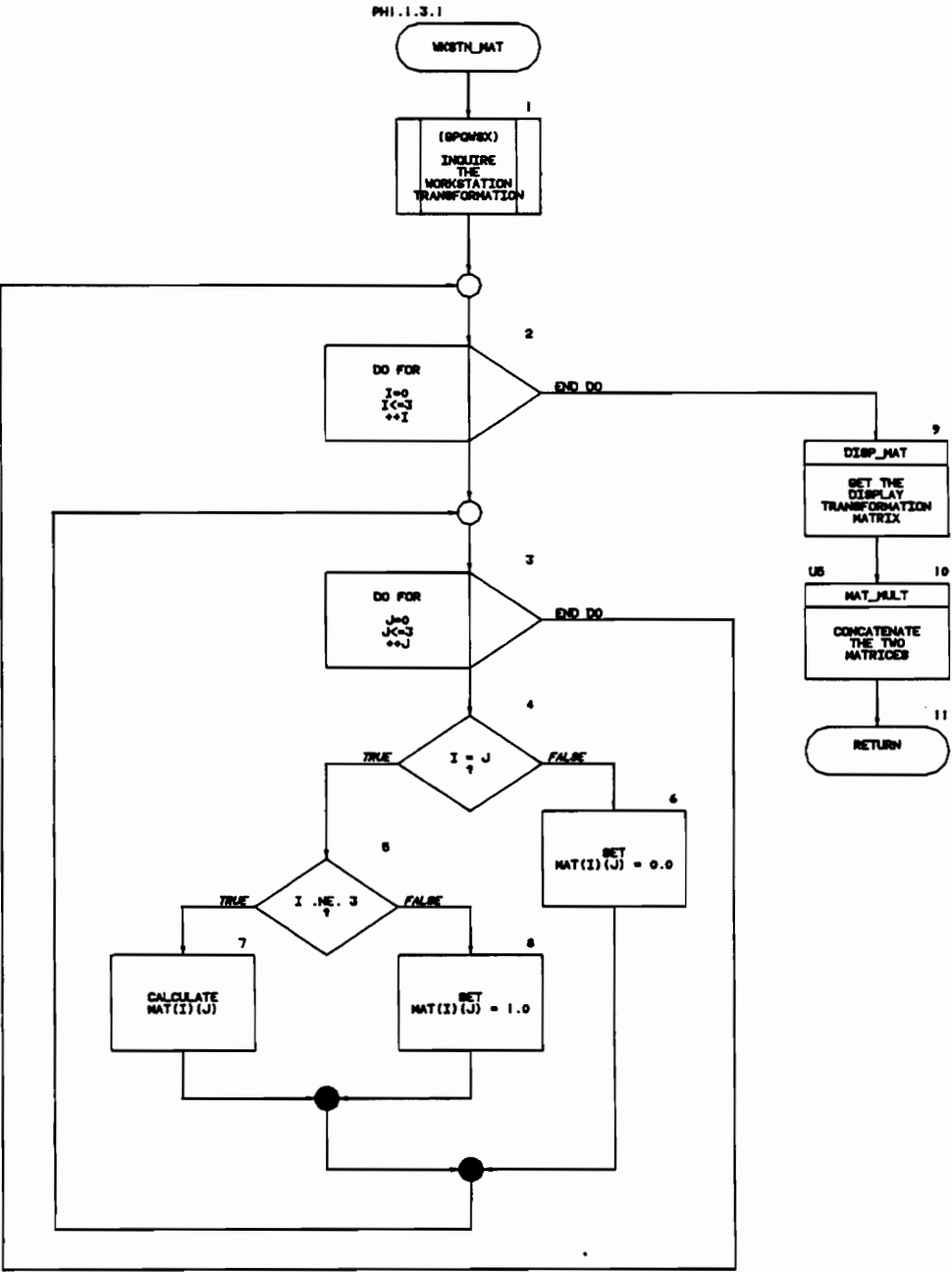
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: DNODE_PHLIST	MODULE: PHI.1.1.4.5
DESIGNED BY: KRISHNAN KOLADY	NOTE: DELETES THE STRUCTURE IDENTIFIER PASSED IN FROM THE PHONG LIST
DATE: 3/9/90	



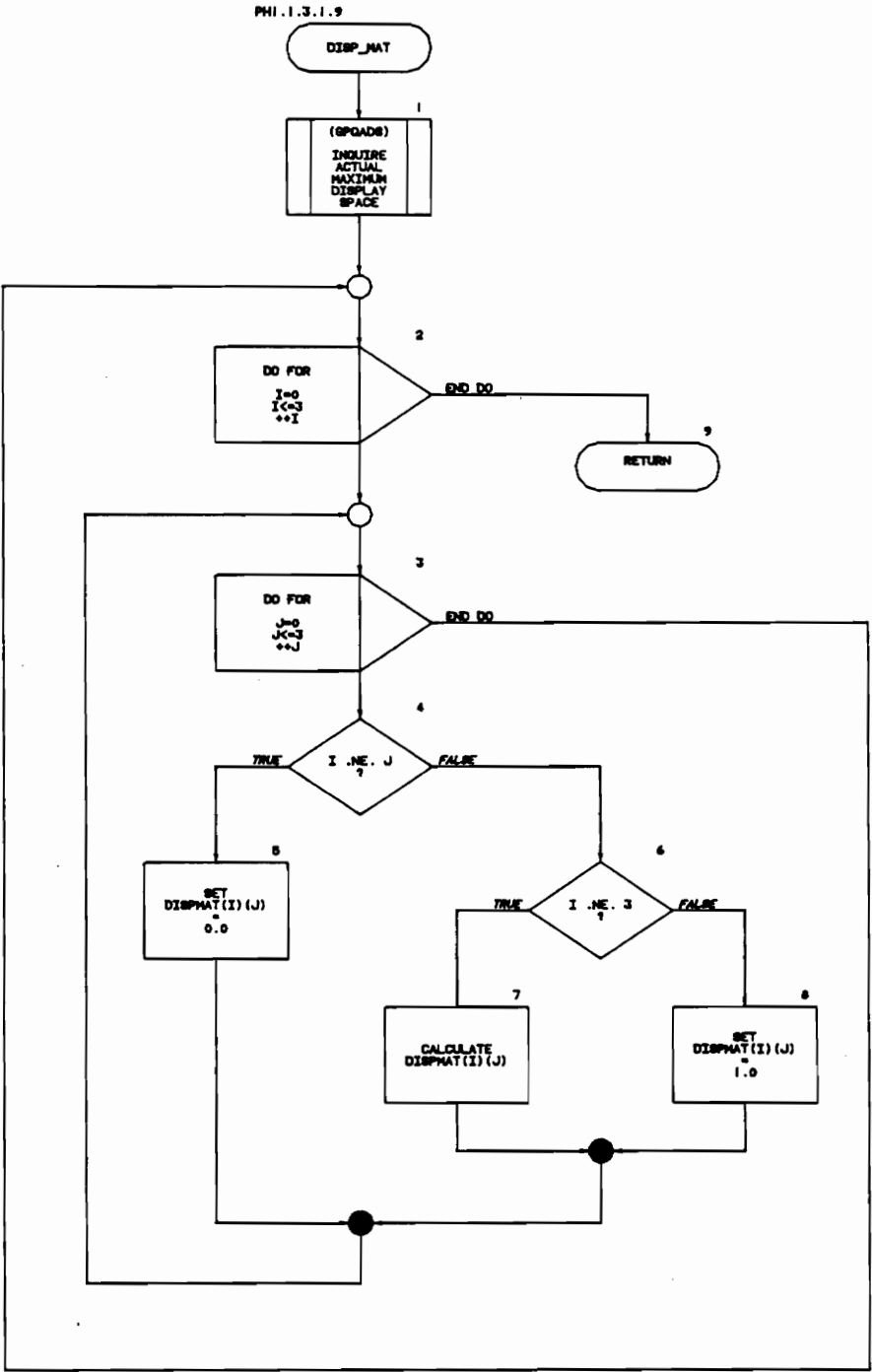
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRO_VIEWS	MODULE: PHI.1.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES OBJECT AND VIEW INFORMATION AND RENDERS SCENE USING PHONG MODEL
DATE:	3/20/90	



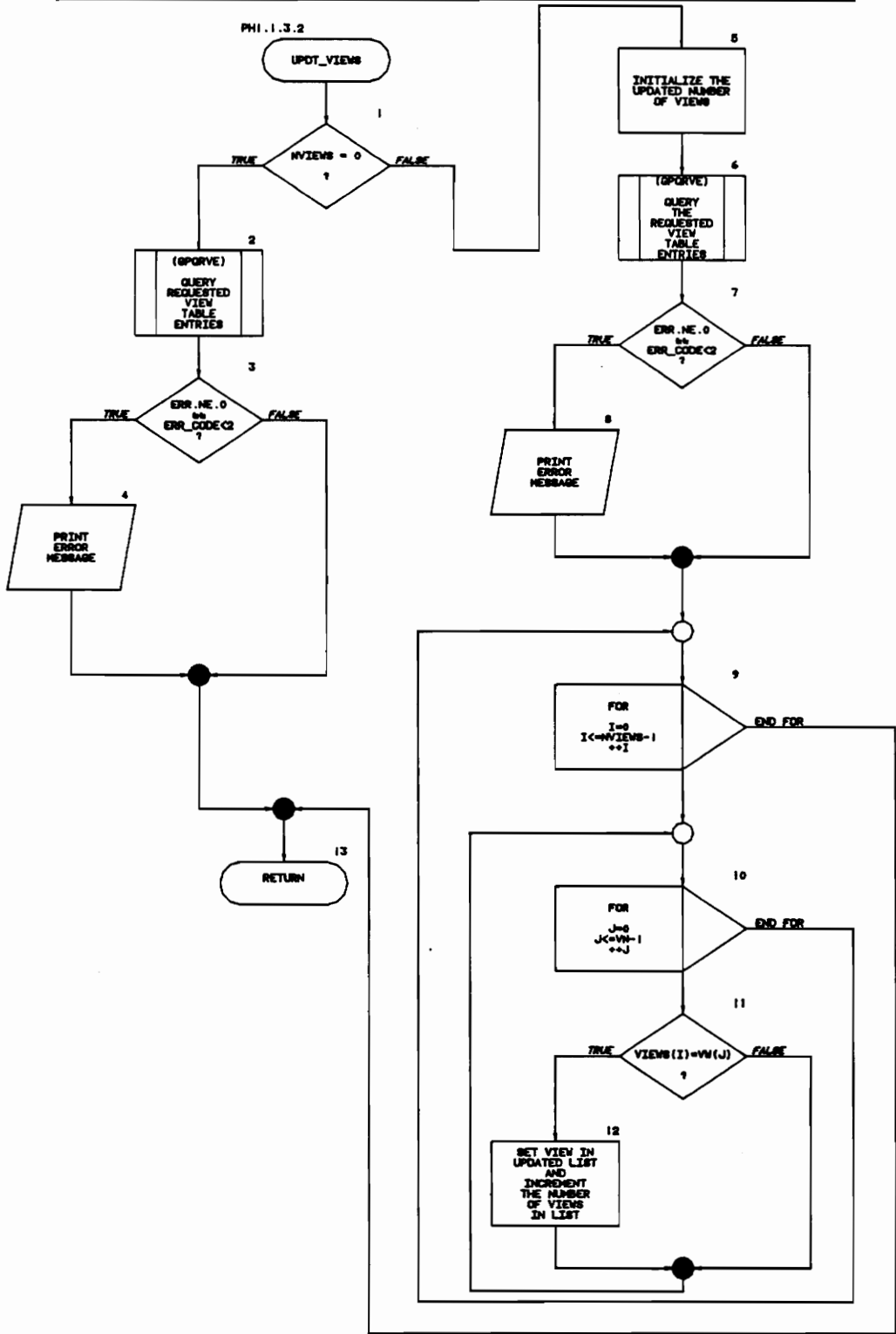
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	WKSTN_MAT	MODULE:PHI.1.3.1
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES WORKSTATION TRANSFORMATION AND DEVICE COORDINATES DATA AND PUTS INTO A TRANSFORMATION MATRIX
DATE:	5/24/90	



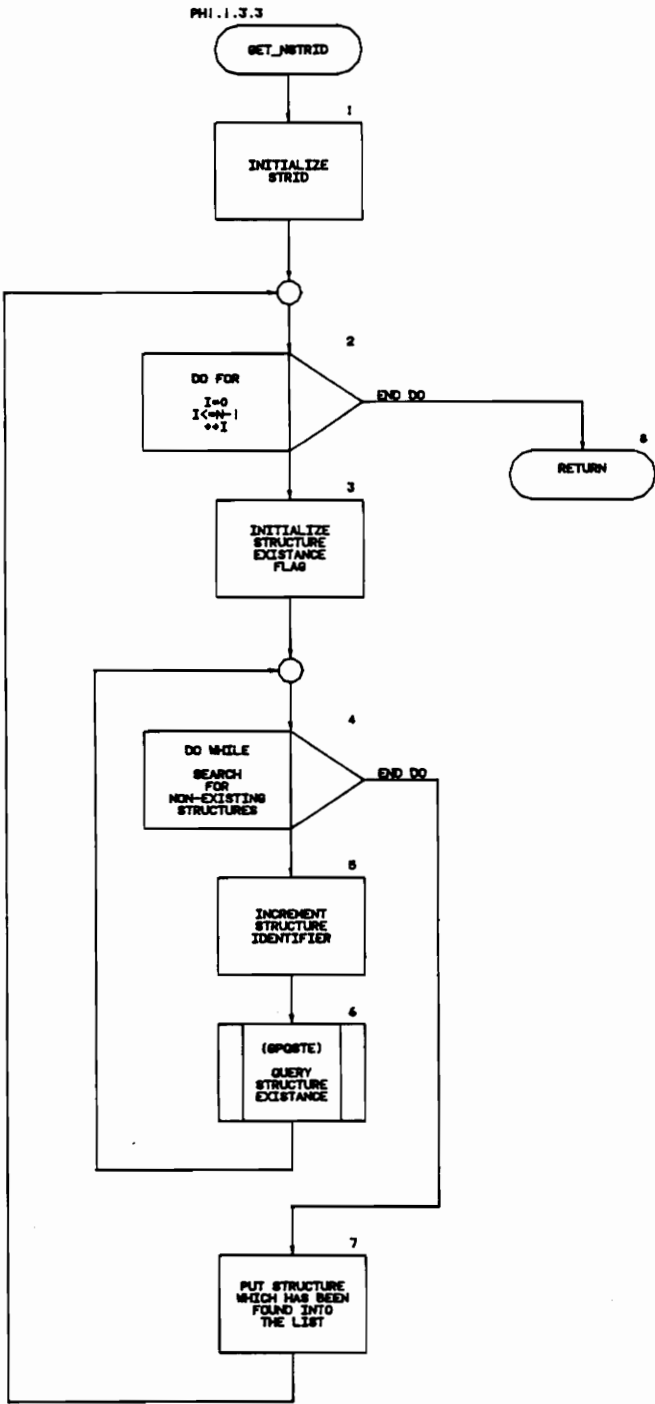
PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	DISP_MAT	MODULE: PHI.1.3.1.9
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES THE ACTUAL MAXIMUM DISPLAY SURFACE AND COMPUTES THE TRANSFORMATION MATRIX
DATE:	5/24/90	



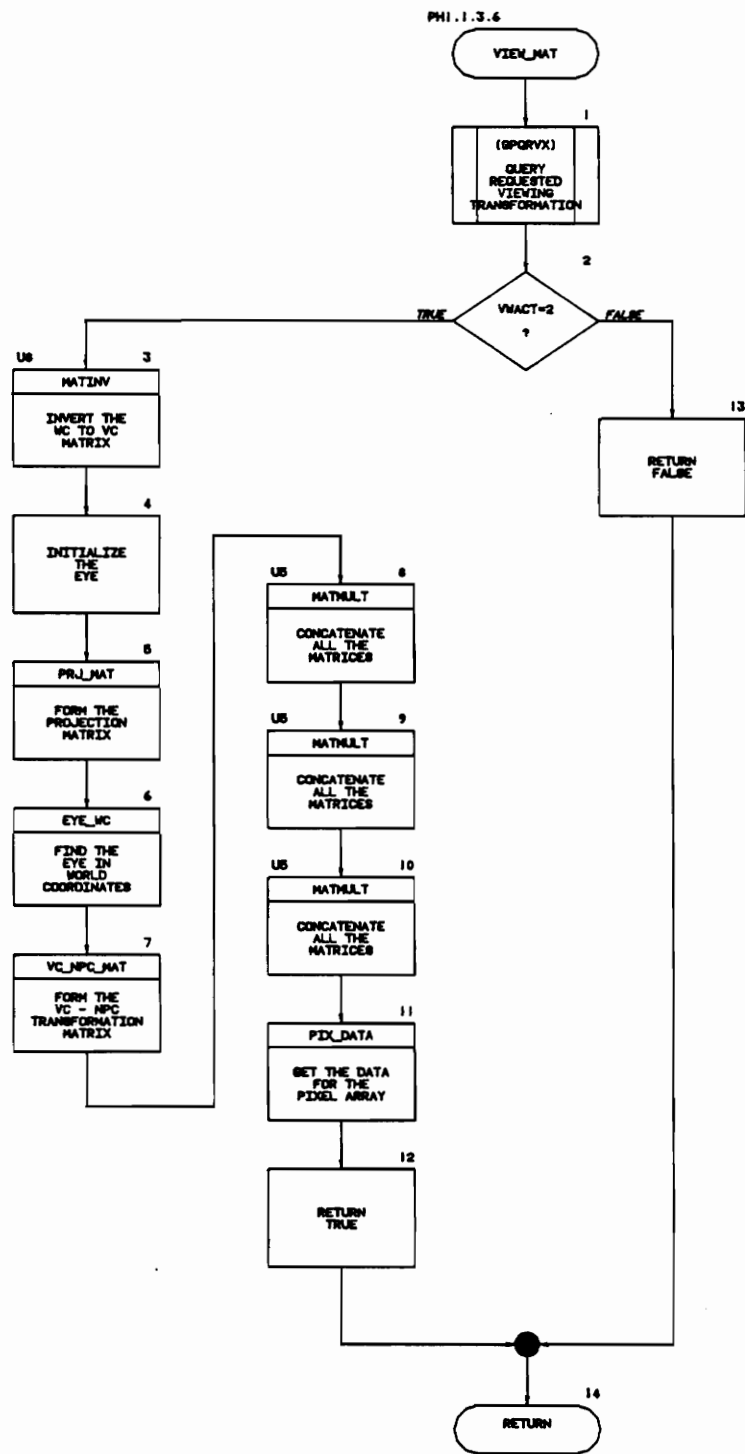
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: UPDT_VIEWS	MODULE: PHI.1.3.2
DESIGNED BY: KRISHNAN KOLADY	NOTE: UPDATES THE VIEW LIST PASSED IN BY THE USER
DATE: 3/9/90	



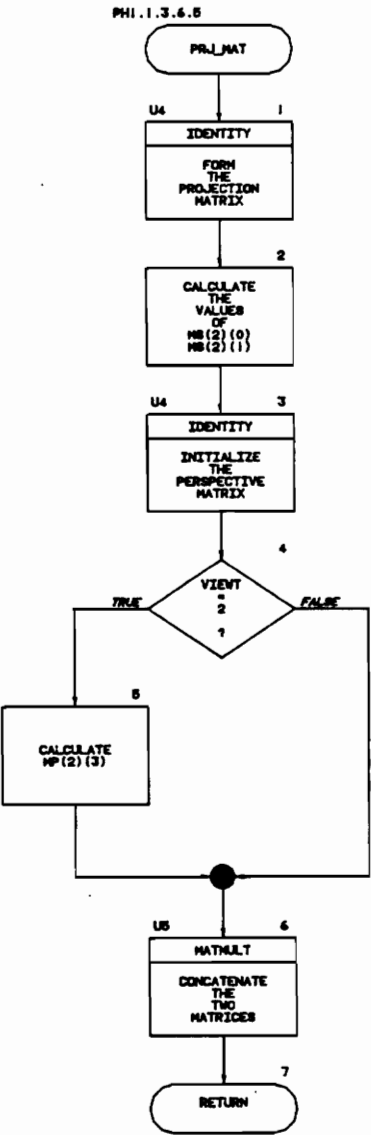
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	GET_NSTRID	MODULE: PHI.1.3.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: GETS N STRUCTURE IDENTIFIERS WHICH NEVER EXISTED BEFORE. THE SEARCH IS STARTED FROM THE STARTING VALUE
DATE:	5/24/90	



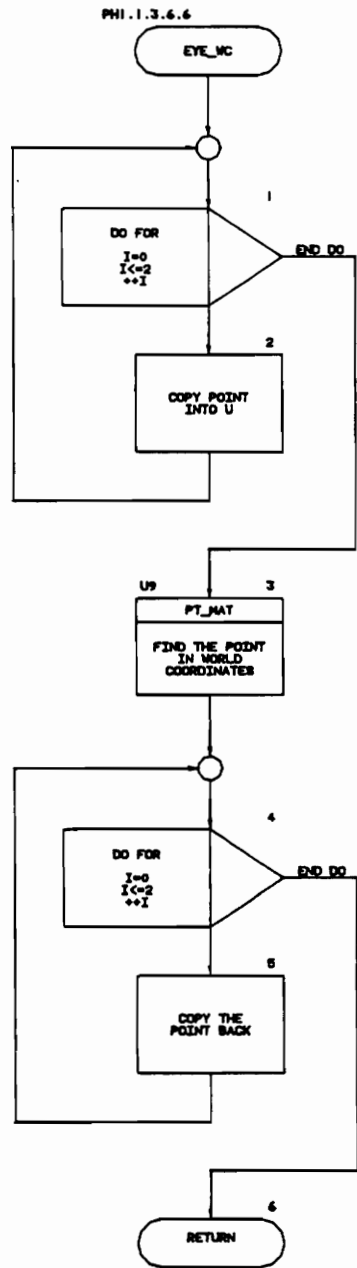
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	VIEW_MAT	MODULE: PHI . 1 . 3 . 6
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES THE VIEWING TRANSFORMATION
DATE:	5/24/90	



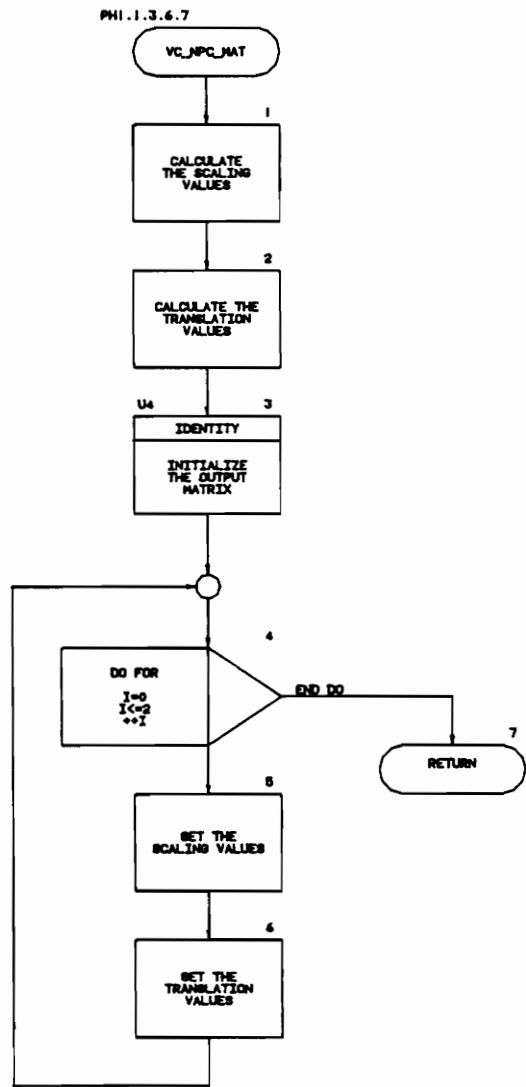
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRJ_MAT	MODULE: PHI.1.3.6.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FORMS THE PROJECTION MATRIX AND THE PERSPECTIVE MATRIX
DATE:	5/24/90	



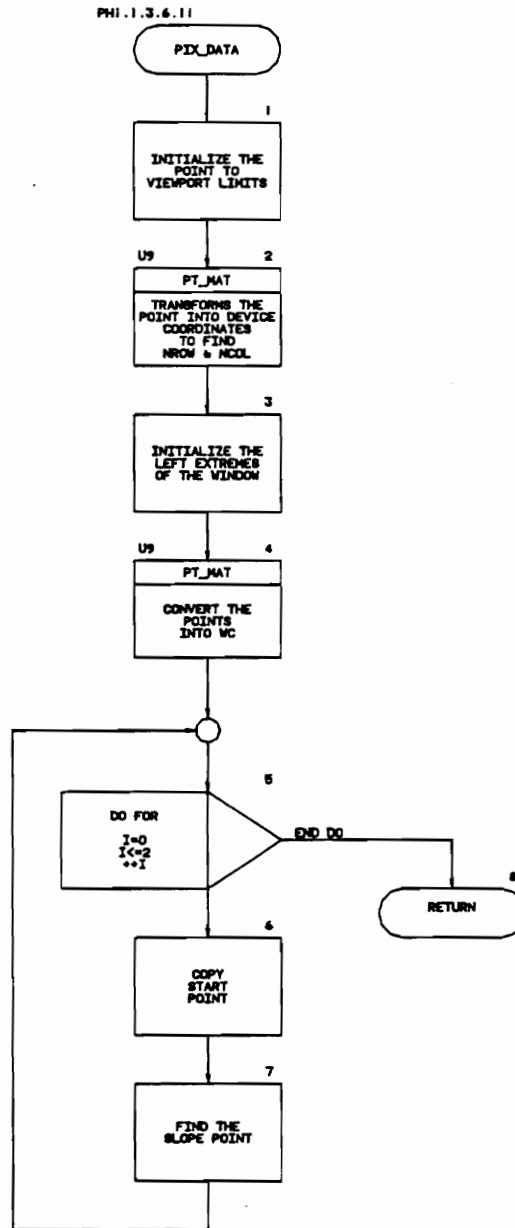
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: EYE_WC		MODULE: PHI.1.3.6.6	
DESIGNED BY: KRISHNAN KOLADY		NOTE: FINDS THE EYE POSITION IN WORLD COORDINATES	
DATE: 6/1/90			



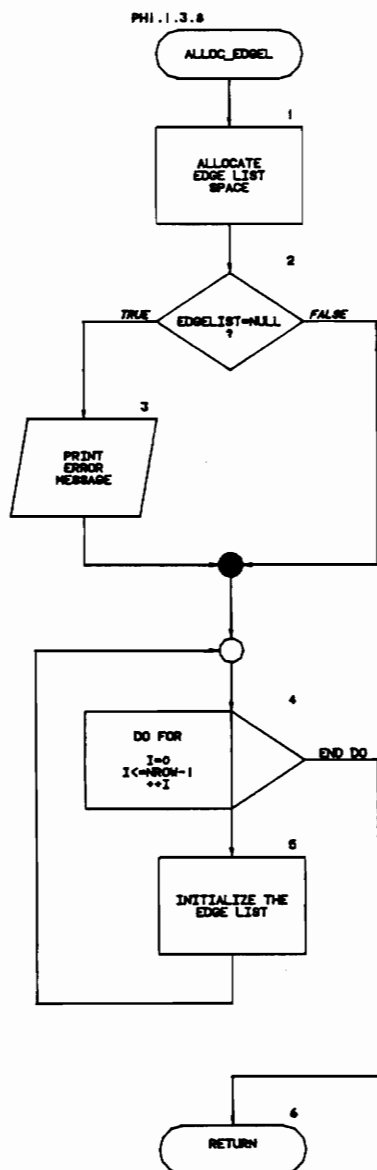
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: VC_NPC_MAT	MODULE: PHI.1.3.6.7
DESIGNED BY: KRISHNAN KOLADY	NOTE: FORMS THE MATRIX TO NORMALIZE THE VIEWING COORDINATES
DATE: 5/30/90	



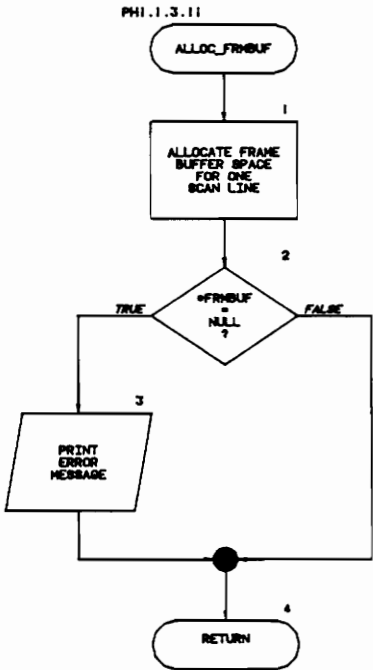
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PIX_DATA	MODULE: PHI.1.3.6.11
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES PIXEL INFORMATION
DATE:	5/30/90	



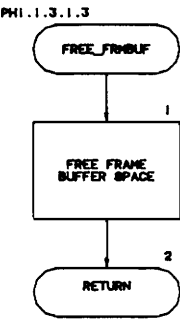
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>		
MODULE NAME:	ALLOC_EDGEL	MODULE:PH1.1.3.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE: ALLOCATES THE EDGE LIST SPACE FOR THE CURRENT VIEW
DATE:	5/30/90	



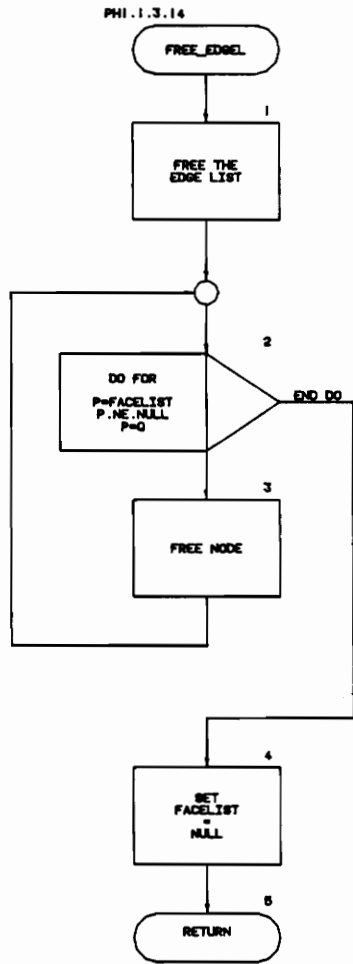
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: ALLOC_FRMBUF		MODULE: PHI . 1 . 3 . 11	
DESIGNED BY: KRISHNAN KOLADY		NOTE: ALLOCATES THE VIRTUAL FRAME BUFFER SPACE FOR THE SPECIFIC VIEW	
DATE: 5/24/90			



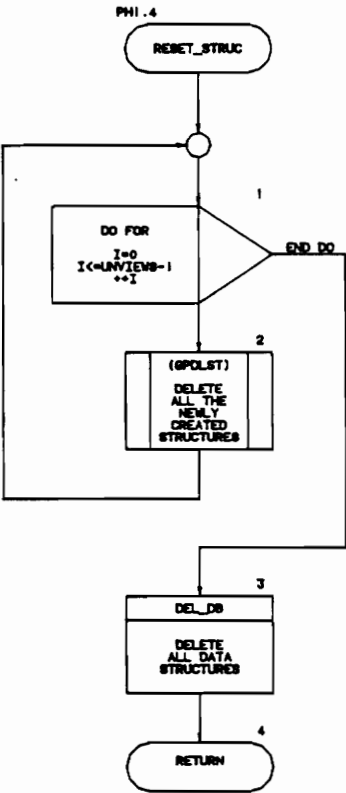
VT		<i>PROGRAM SPECIFICATION - PHONG SHADING</i>
MODULE NAME:	FREE_FRMBUF	MODULE: PHI . 1 . 3 . 1 . 3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FREES THE VIRTUAL FRAME BUFFER SPACE FOR THE SPECIFIC VIEW
DATE:	5/24/90	



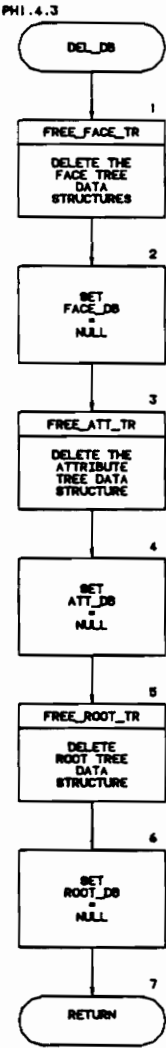
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: FREE_EDGEL	MODULE: PHI.1.3.14
DESIGNED BY: KRISHNAN KOLADY	NOTE: FREES THE EDGE LIST AND FACE LIST SPACE FOR THE CURRENT VIEW
DATE: 5/30/90	



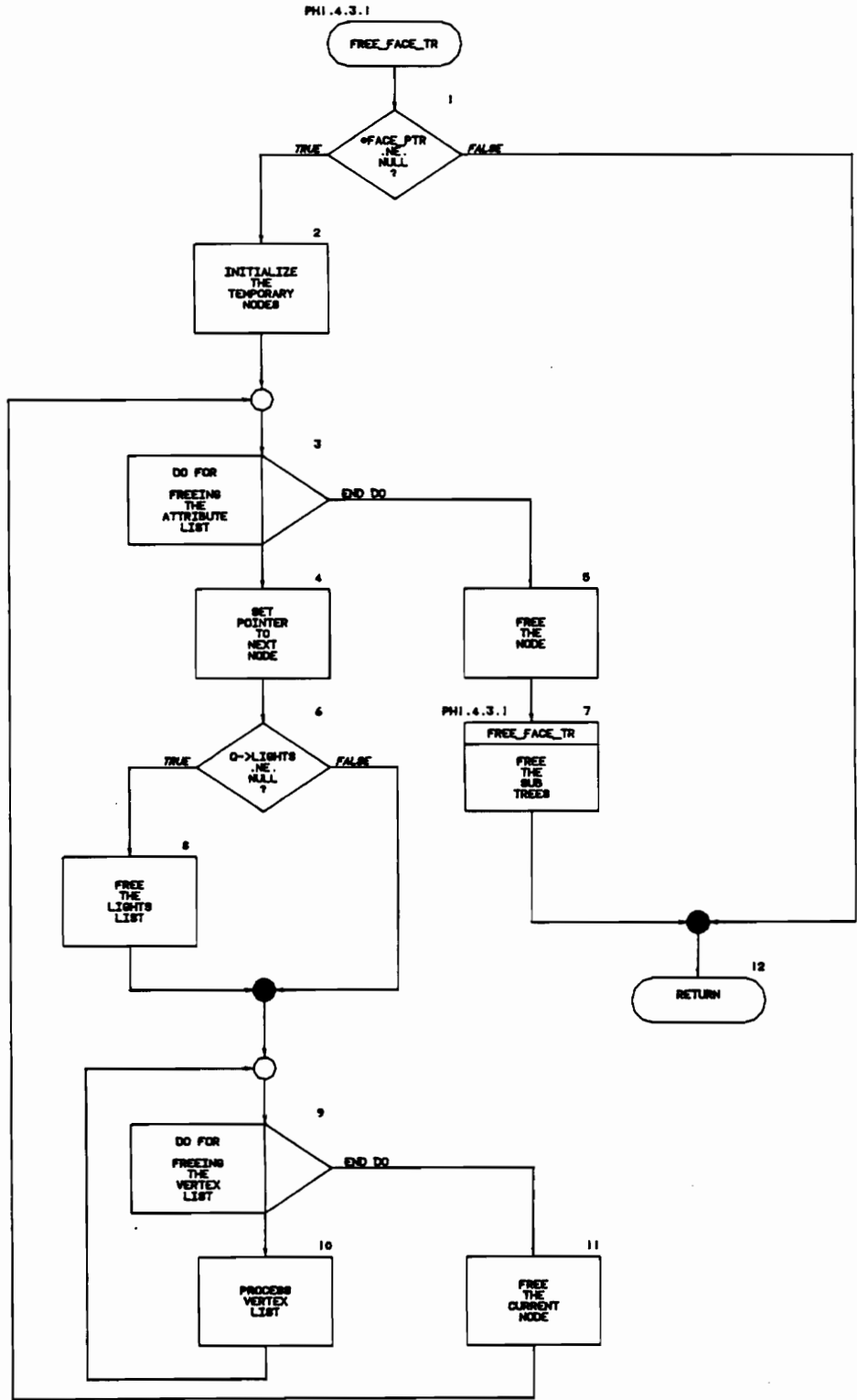
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	RESET_STRUC	MODULE: PHI.4
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RESETS PHIGS STRUCTURE HIERARCHY AND DELETES ALL DATA STRUCTURES THE PHONG RENDERING SCHEME HAS USED
DATE:	3/20/90	



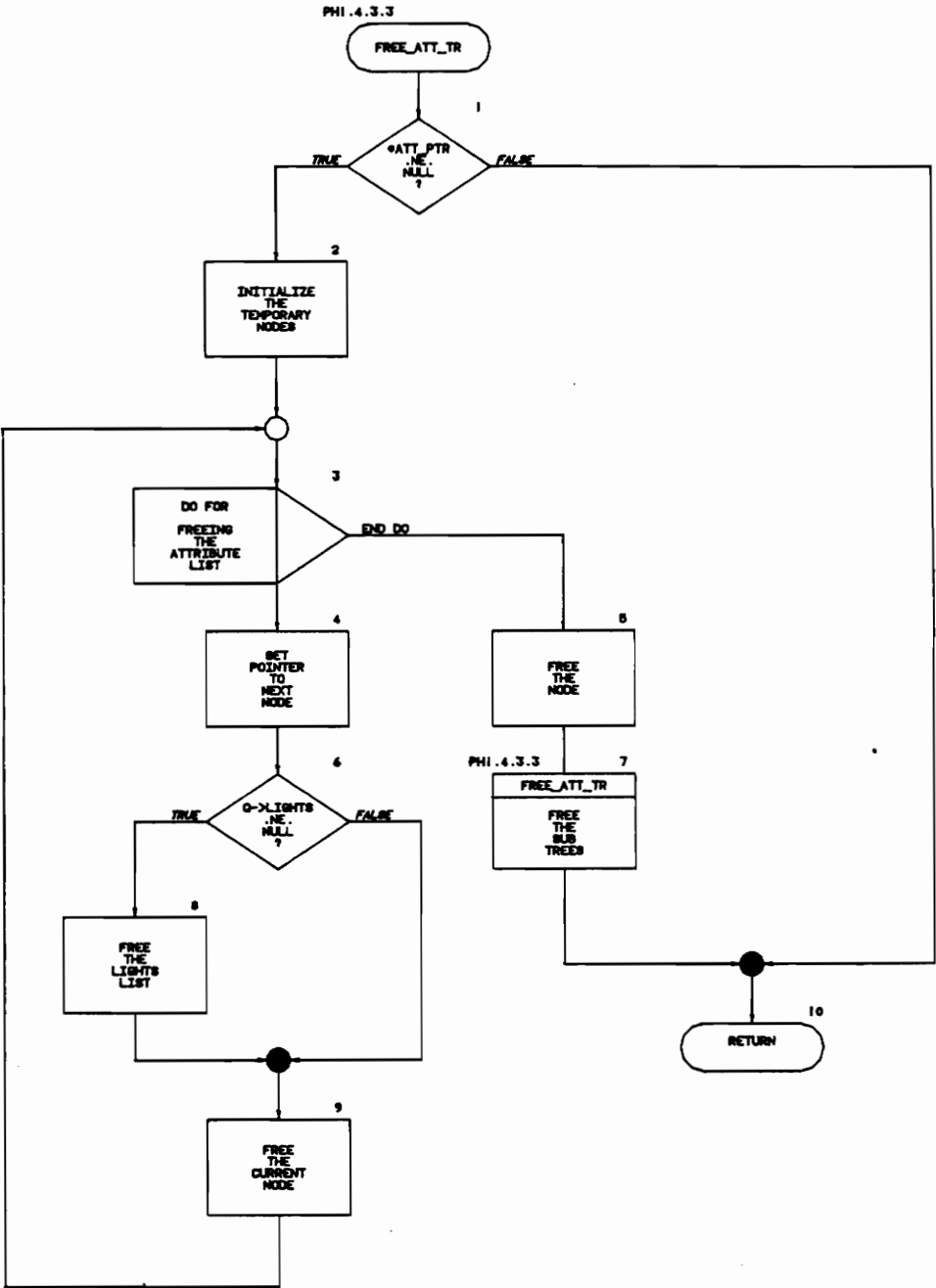
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	DEL_DB	MODULE: PHI . 4 . 3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: DELETES THE ATTRIBUTE TREE AND ROOT TREE DATA STRUCTURES
DATE:	3/19/90	



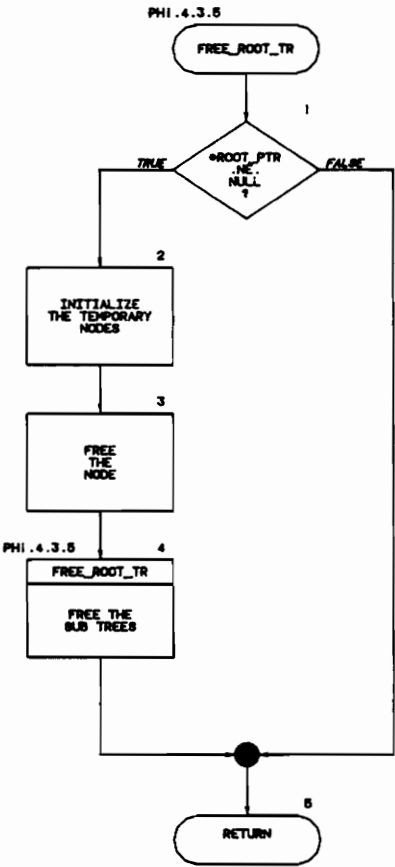
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: FREE_FACE_TR	MODULE: PHI.4.3.1
DESIGNED BY: KRISHNAN KOLADY	NOTE: FREES THE FACE TREE DATA STRUCTURE
DATE: 5/24/90	



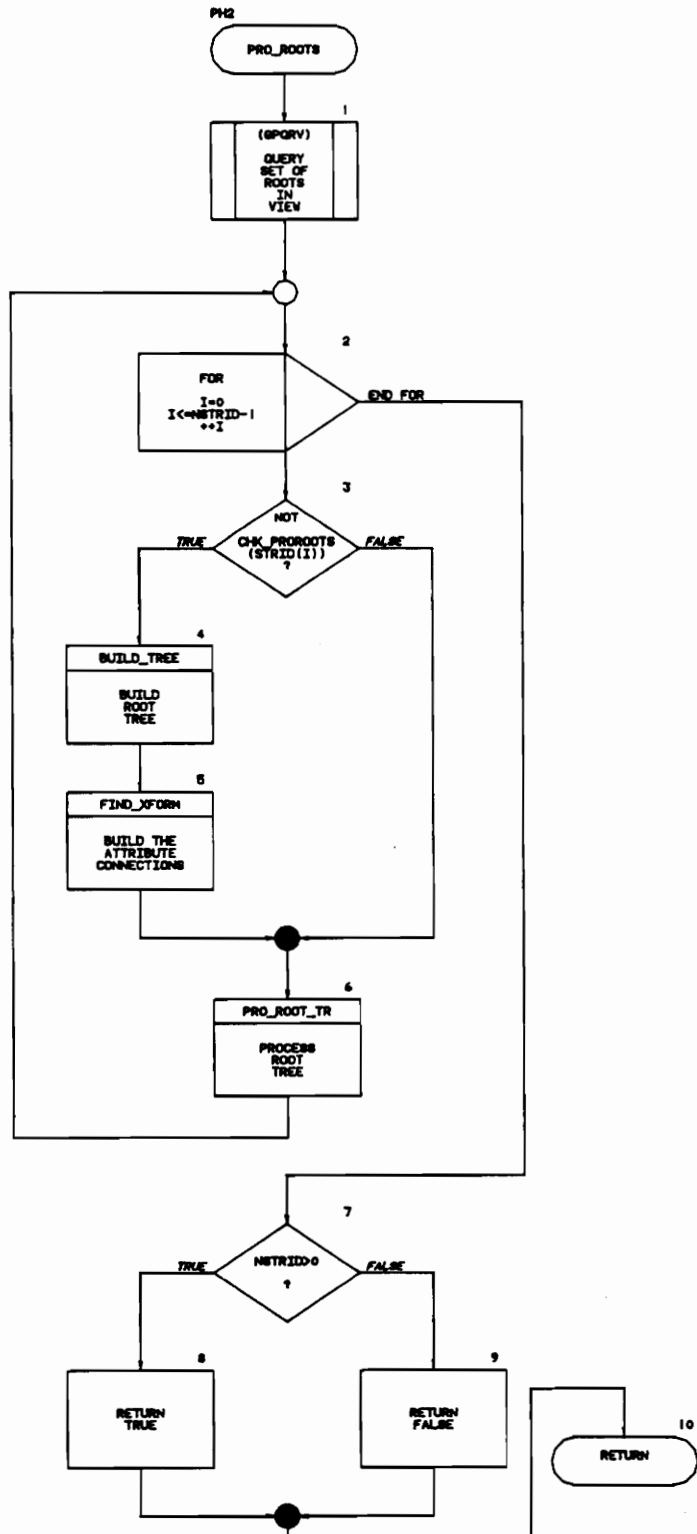
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>	
MODULE NAME: FREE_ATT_TR	MODULE: PHI.4.3.3
DESIGNED BY: KRISHNAN KOLADY	NOTE: FREES THE ATTRIBUTE TREE DATA STRUCTURE
DATE: 5/24/90	




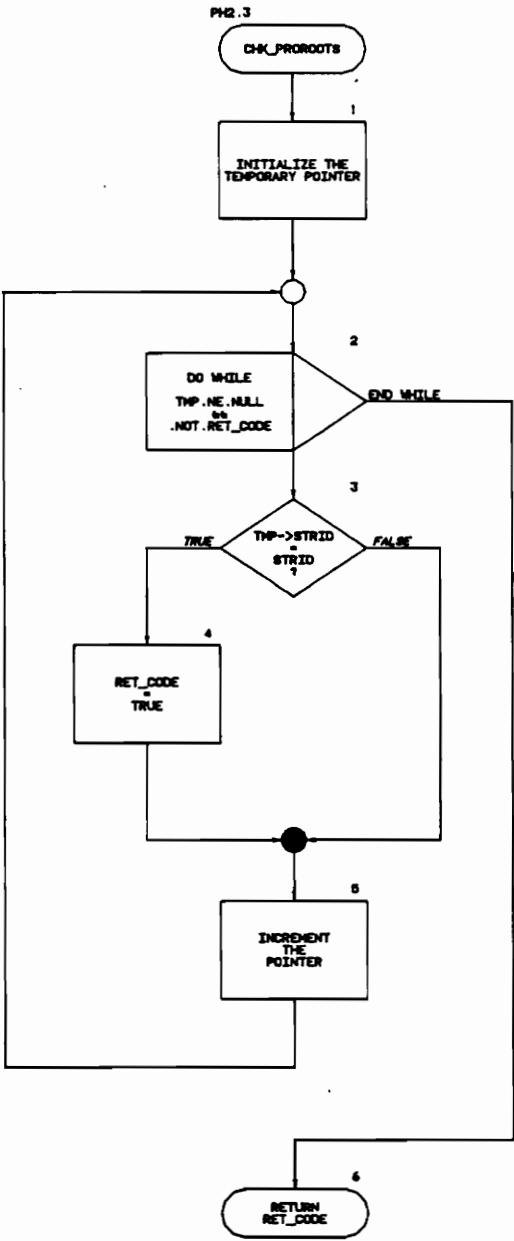
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME : FREE_ROOT_TR		MODULE : PHI . 4 . 3 . 5	
DESIGNED BY : KRISHNAN KOLADY		NOTE : FREES THE ROOT TREE DATA STRUCTURE	
DATE : 5/30/90			



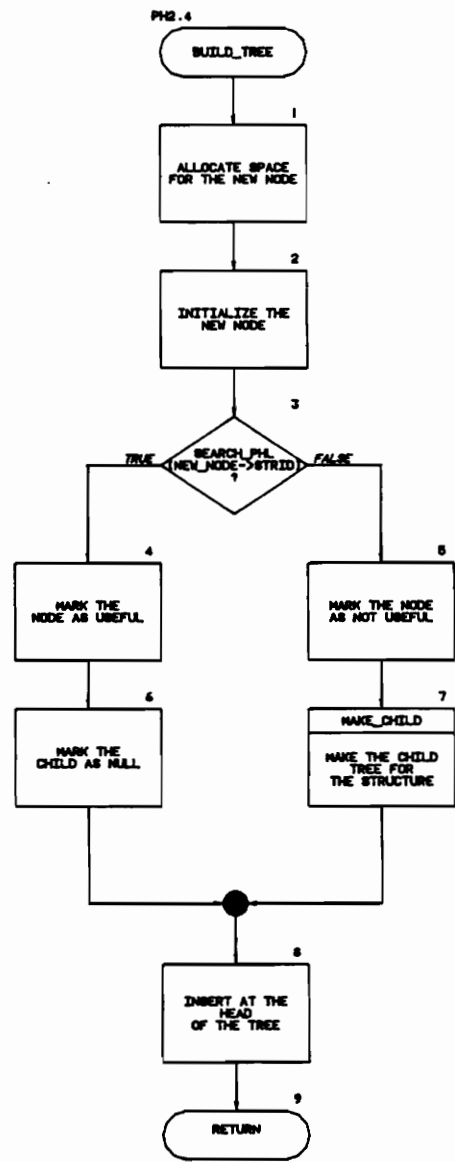
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRO_ROOTS	MODULE: PH2
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FINDS ROOTS ASSOCIATED WITH VIEW AND BUILDS PHIGS STRUCTURE HIERARCHY AS BINARY TREE
DATE:	3/20/90	



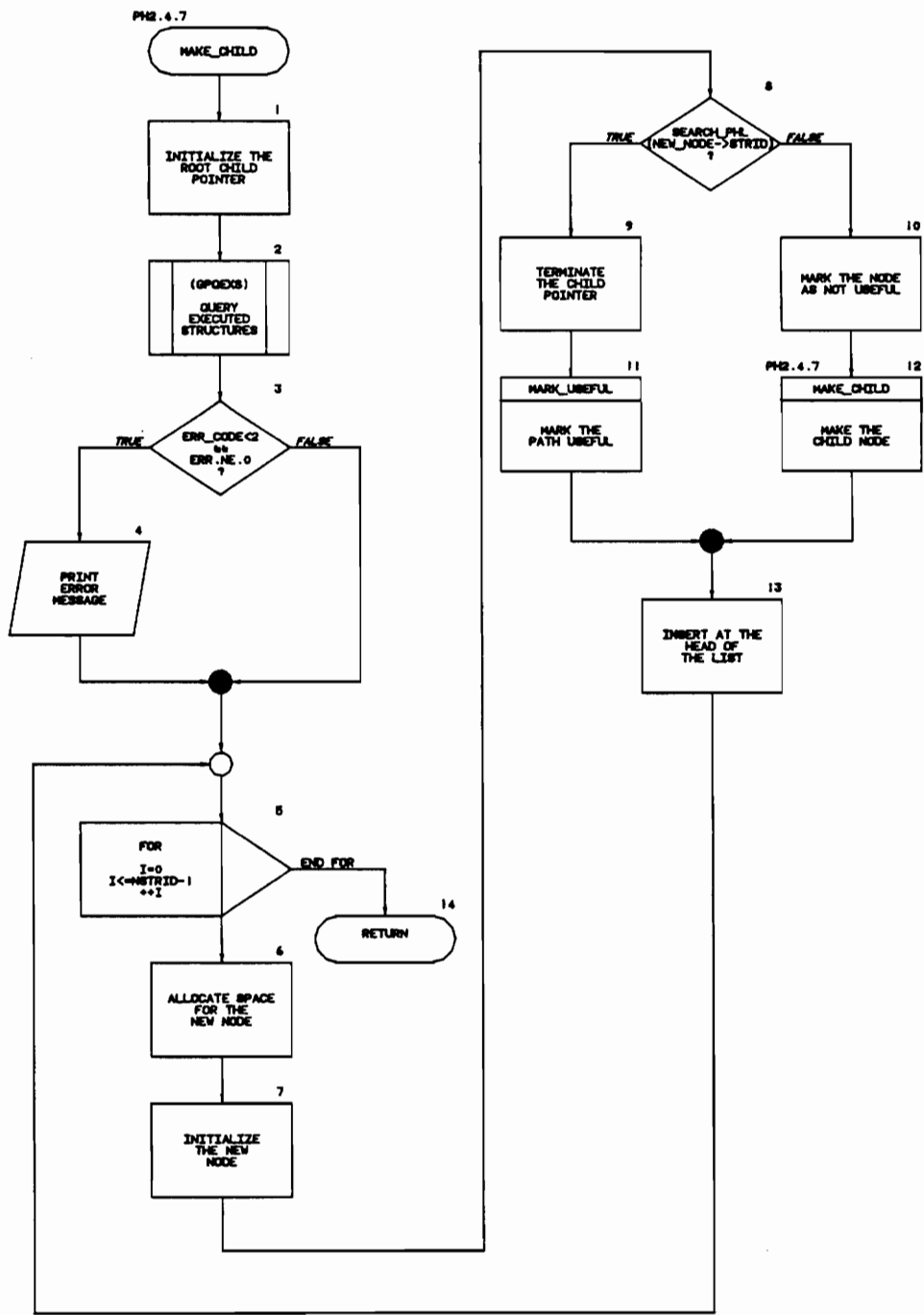
<div>  PROGRAM SPECIFICATION - PHONG SHADING </div>	
MODULE NAME: CHK_PROROOTs	MODULE: PH2.3
DESIGNED BY: KRISHNAN KOLADY	NOTE: CHECKS IF THE STRUCTURE IDENTIFIER PASSED IN HAS BEEN PROCESSED AS A ROOT AND RETURNS A BOOLEAN VALUE
DATE: 3/19/90	



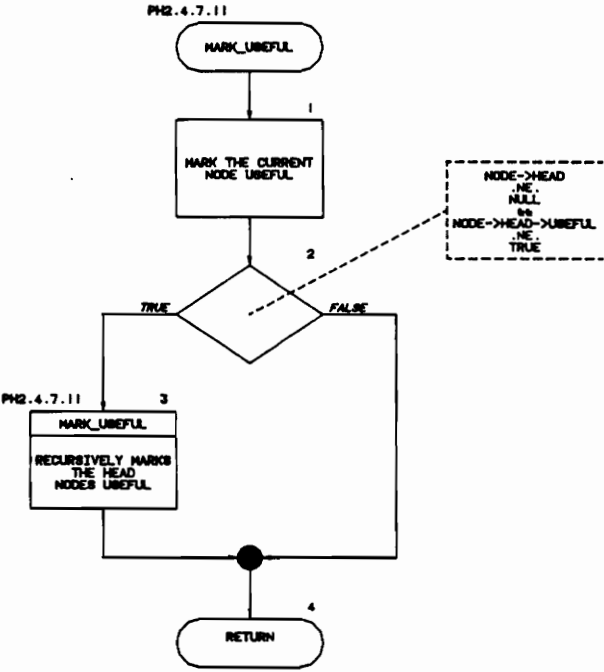
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: BUILD_TREE	MODULE: PH2.4
DESIGNED BY: KRISHNAN KOLADY	NOTE: BUILDS THE TREE STRUCTURE FOR THE STRUCTURE IDENTIFIER PASSED IN
DATE: 3/19/90	



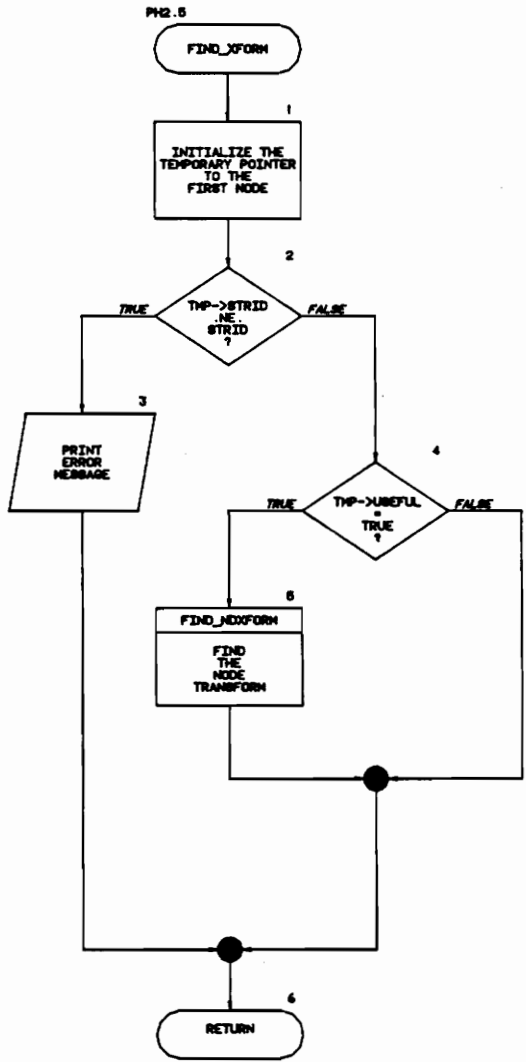
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: MAKE_CHILD	MODULE: PH2.4.7
DESIGNED BY: KRISHNAN KOLADY	NOTE: CREATES CHILD TREE STRUCTURE FOR THE GIVEN STRUCTURE IDENTIFIER
DATE: 3/20/90	



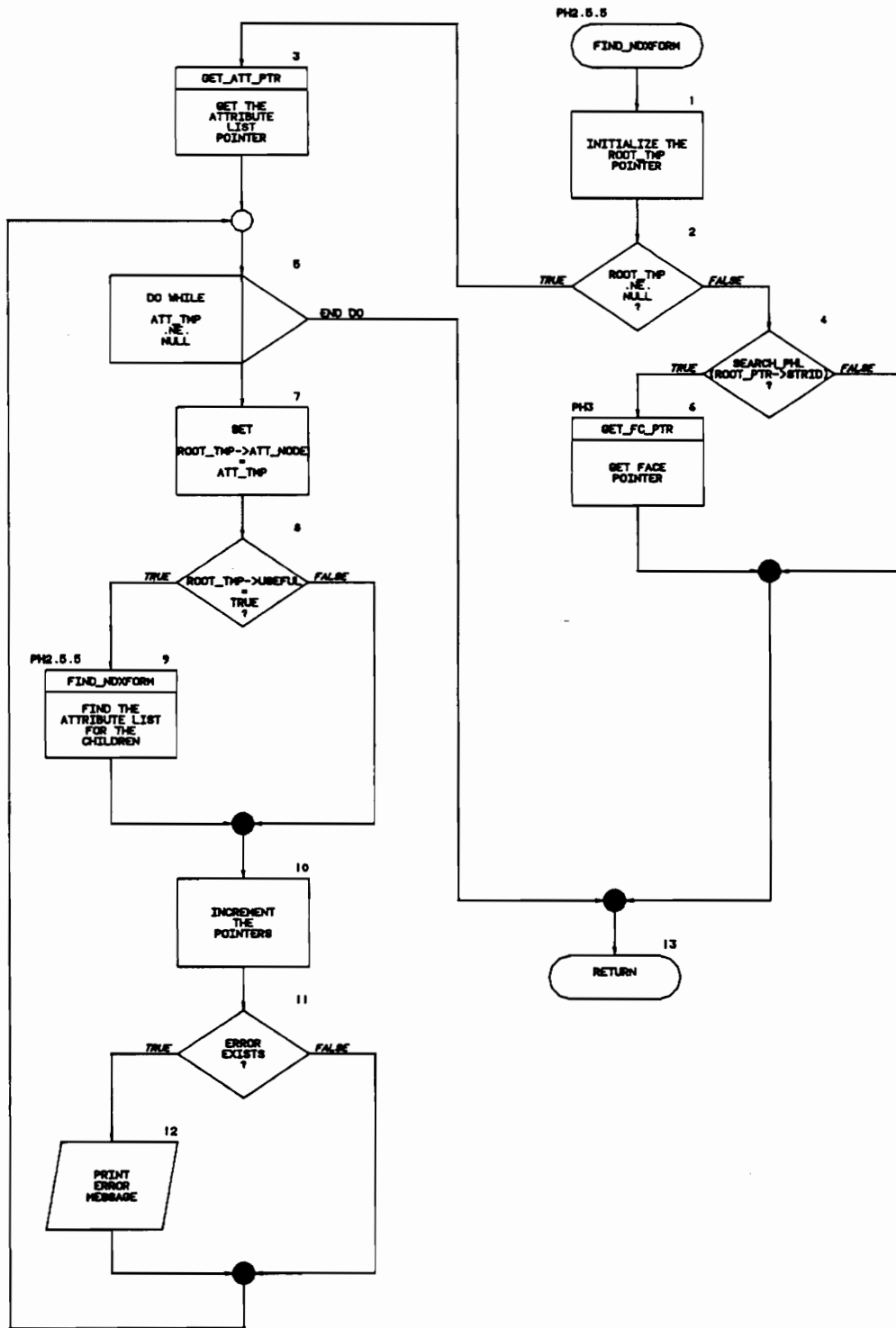
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: MARK_USEFUL	MODULE: PH2.4.7.11
DESIGNED BY: KRISHNAN KOLADY	NOTE: RECURSIVELY MARKS THE PATH TILL THE CURRENT NODE AS USEFUL IN ROOT TREE STRUCTURE
DATE: 3/20/90	



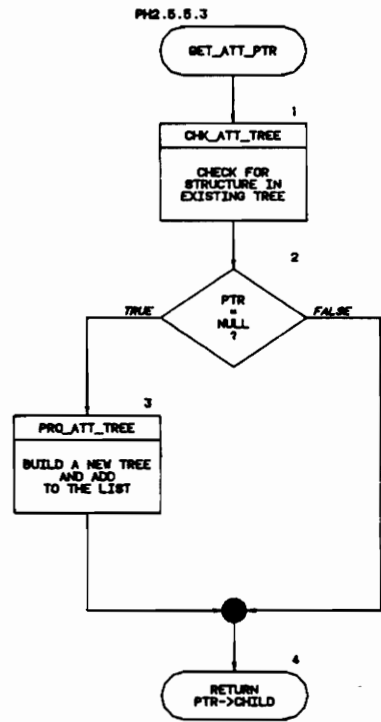
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	FIND_XFORM	MODULE: PH2.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FINDS TRANSFORMATIONS AND ATTRIBUTES WITHIN SPECIFIED CHILDREN TO CARRY TO CHILDREN AND PUT IN DATA STRUCTURE
DATE:	3/19/90	



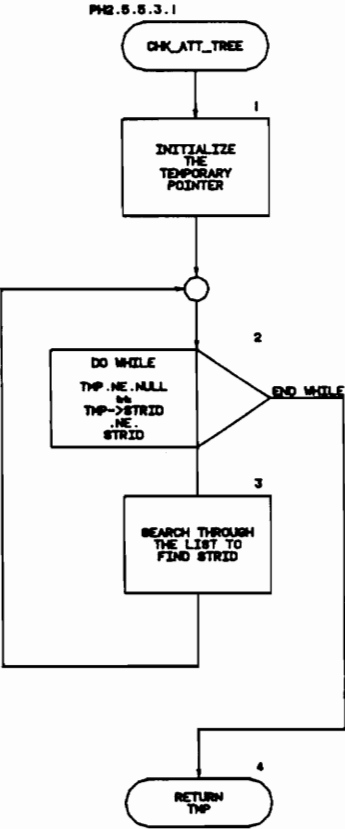
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FIND_NOXFORM	MODULE: PH2.5.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: GETS ATTRIBUTE LIST POINTER AND SETS REFERENCES OF THE NODES TO THEIR RESPECTIVE ATTRIBUTES
DATE:	3/19/90	



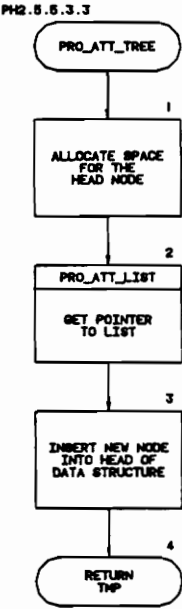
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	GET_ATT_PTR	MODULE: PH2.5.5.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: GETS POINTER TO THE ATTRIBUTE LIST FROM THE ATTRIBUTE DATA STRUCTURE
DATE:	3/20/90	



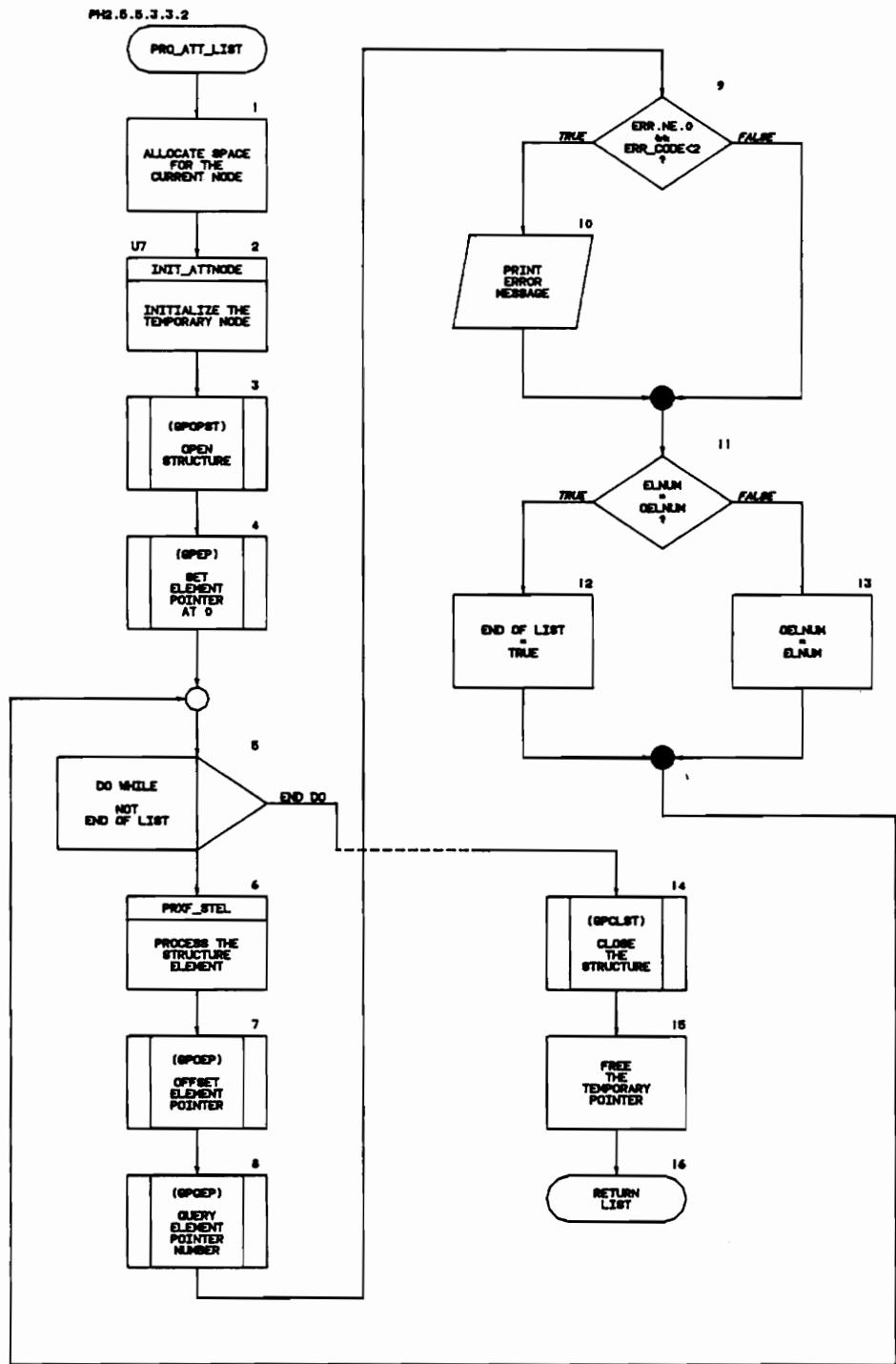
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	CHK_ATT_TREE	MODULE: PH2.5.5.3.1
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CHECKS CURRENT ATTRIBUTE DATA STRUCTURE FOR PROCESSED STRUCTURE IDENTIFIER
DATE:	3/19/90	



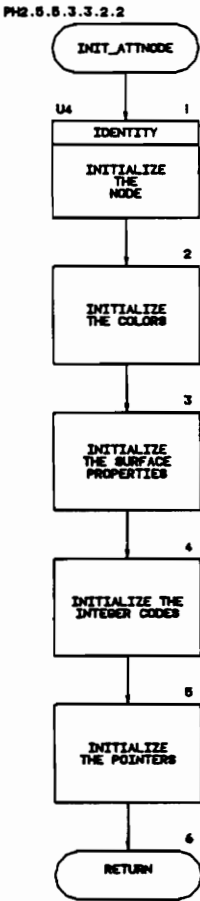
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	PRO_ATT_TREE	MODULE: PH2.5.5.3.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES A NEW TREE FOR THE CURRENT STRUCTURE IDENTIFIER
DATE:	3/20/90	



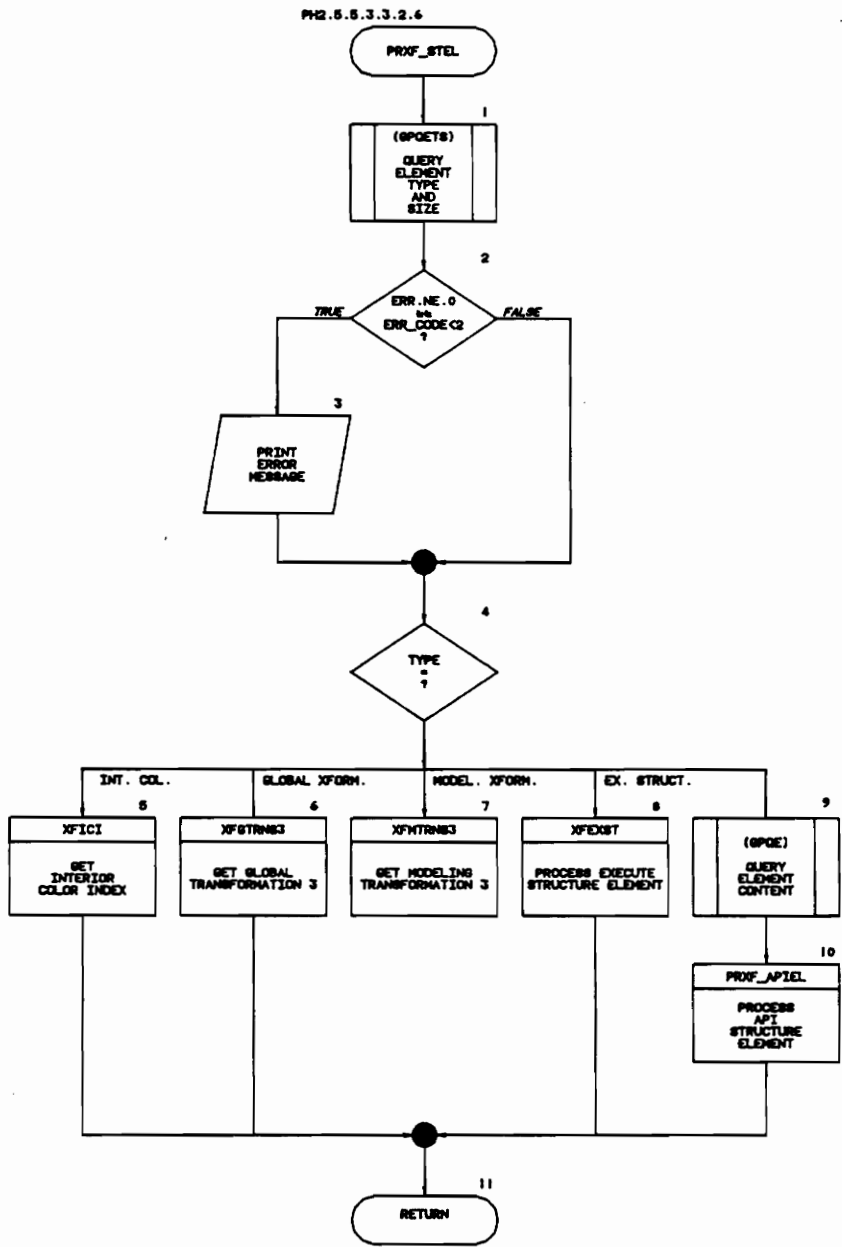
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRO_ATT_LIST	MODULE:PH2.5.5.3.3.2
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CREATES LIST OF TRANSFORMATIONS AND ATTRIBUTES TO PASS DOWN TO THE STRUCTURE HIERARCHY
DATE:	3/20/90	



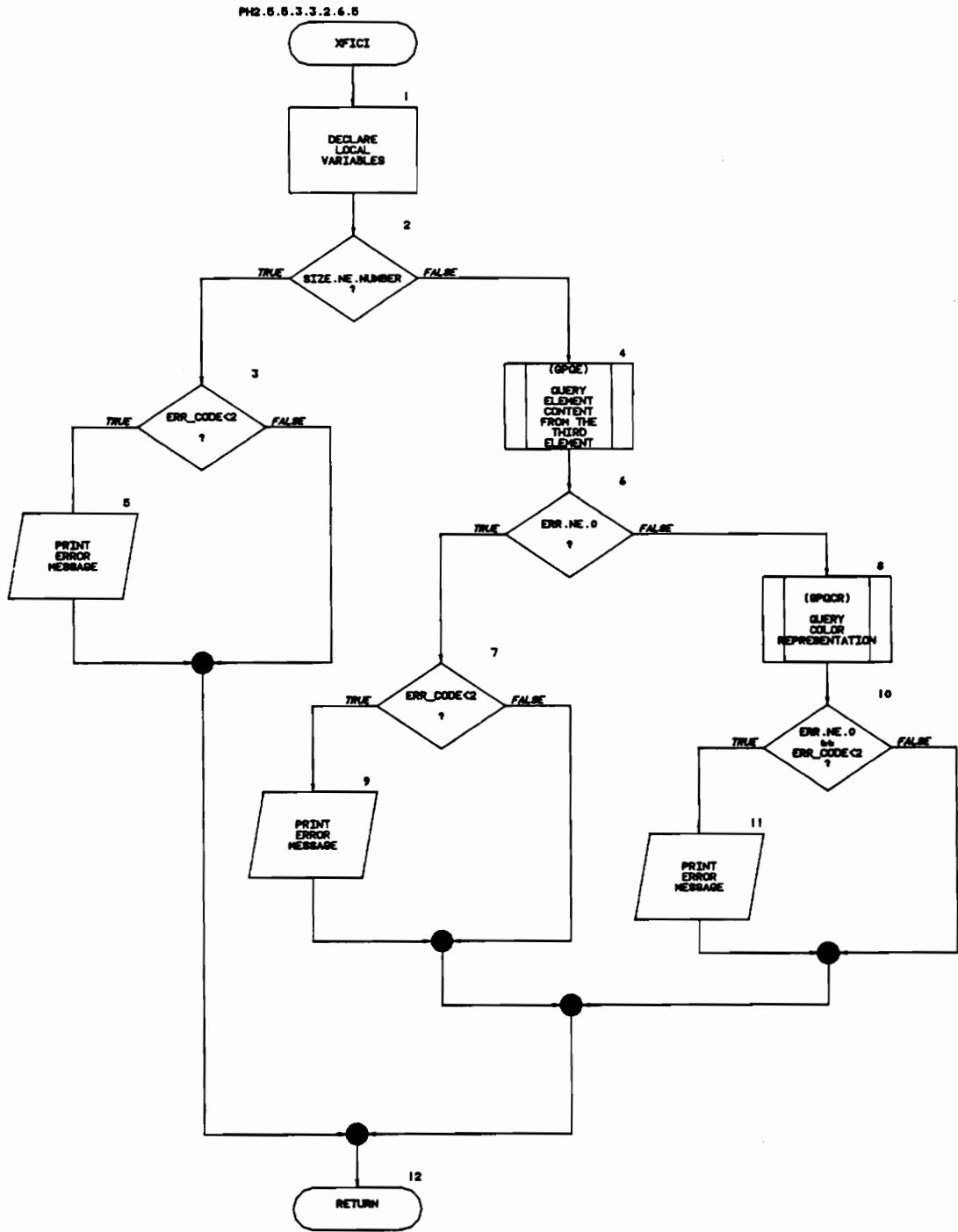
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: INIT_ATTNODE	MODULE: PH2.5.5.3.3.2.2
DESIGNED BY: KRISHNAN KOLADY	NOTE: INITIALIZES THE ATTRIBUTE LIST NODE
DATE: 3/20/90	




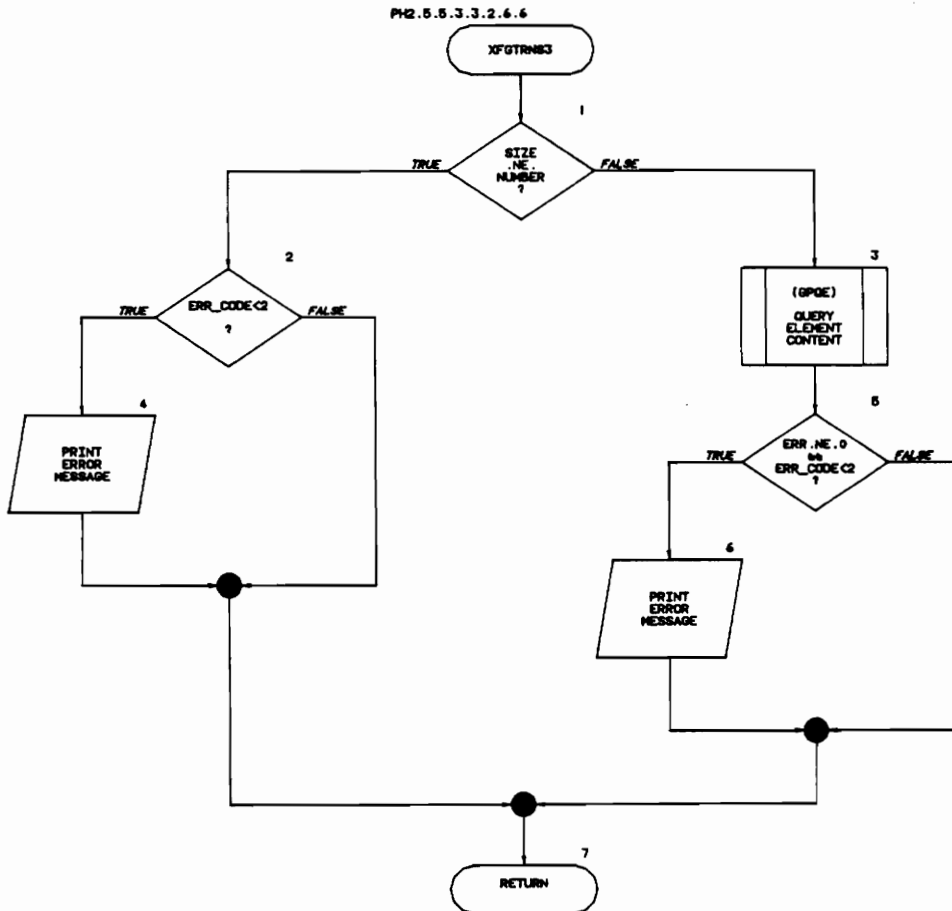
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME:	PRXF_STEL	MODULE:	PH2.5.5.3.3.2.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE:	PROCESSES STRUCTURE ELEMENT OF CURRENTLY OPEN STRUCTURE AND RETRIEVES TRANSFORMATION AND ATTRIBUTE DATA
DATE:	3/20/90		



VF PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	XFICI	MODULE:PH2.5.5.3.3.2.6.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES INTERIOR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	

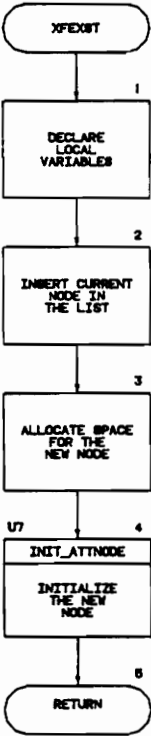


<div>  PROGRAM SPECIFICATION - PHONG SHADING </div>		
MODULE NAME:	XFGTRNS3	MODULE: PH2.5.5.3.3.2.6.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES GLOBAL TRANSFORMATION MATRIX DATA FROM CURRENTLY OPEN STRUCTURE
DATE:	3/20/90	

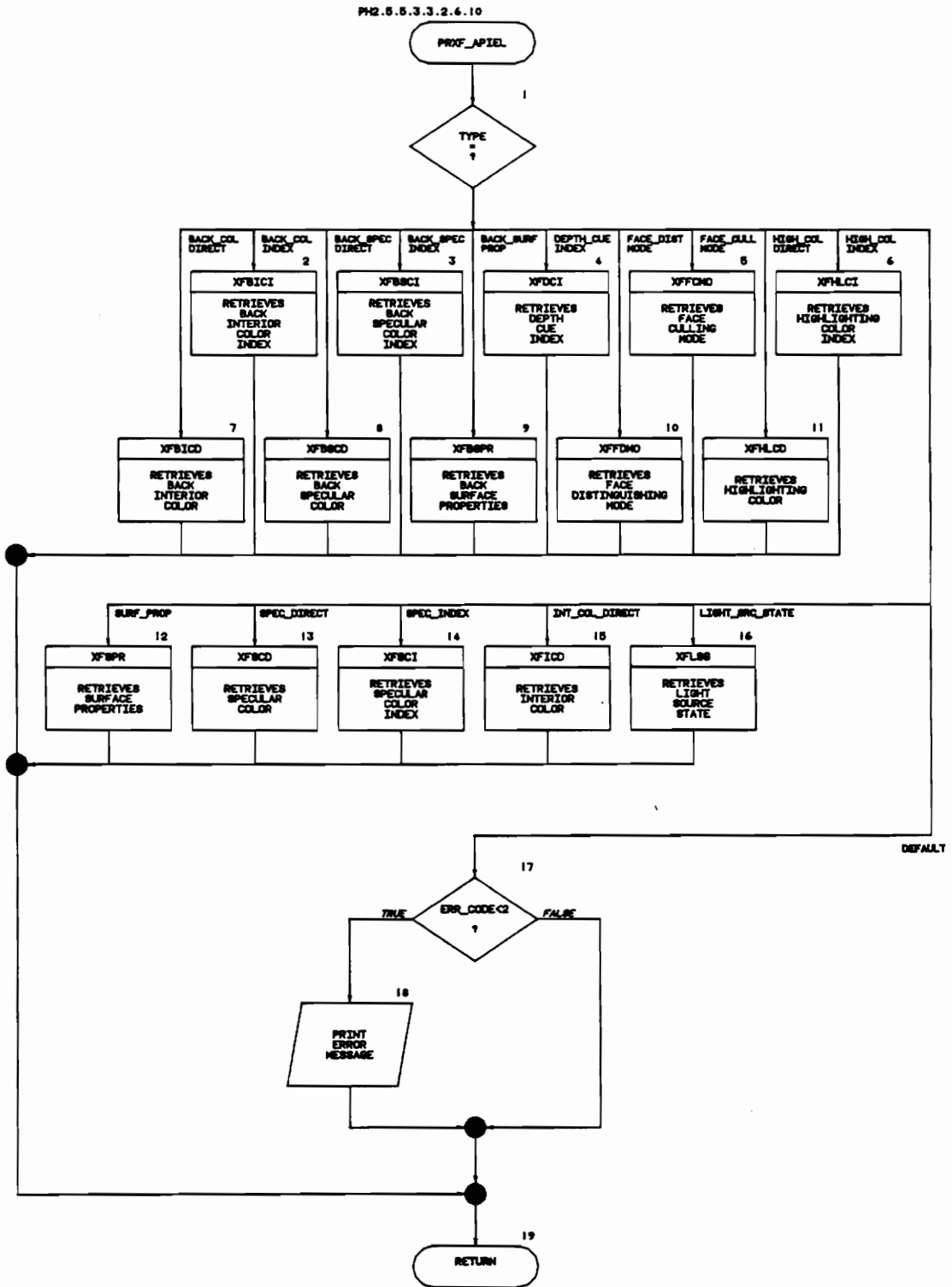


VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: XFEYST	MODULE:PH2.5.5.3.3.2.6.8
DESIGNED BY: KRISHNAN KOLADY	NOTE:PROCESSES EXECUTE STRUCTURE ELEMENT IN CURRENTLY OPEN STRUCTURE
DATE: 3/20/90	

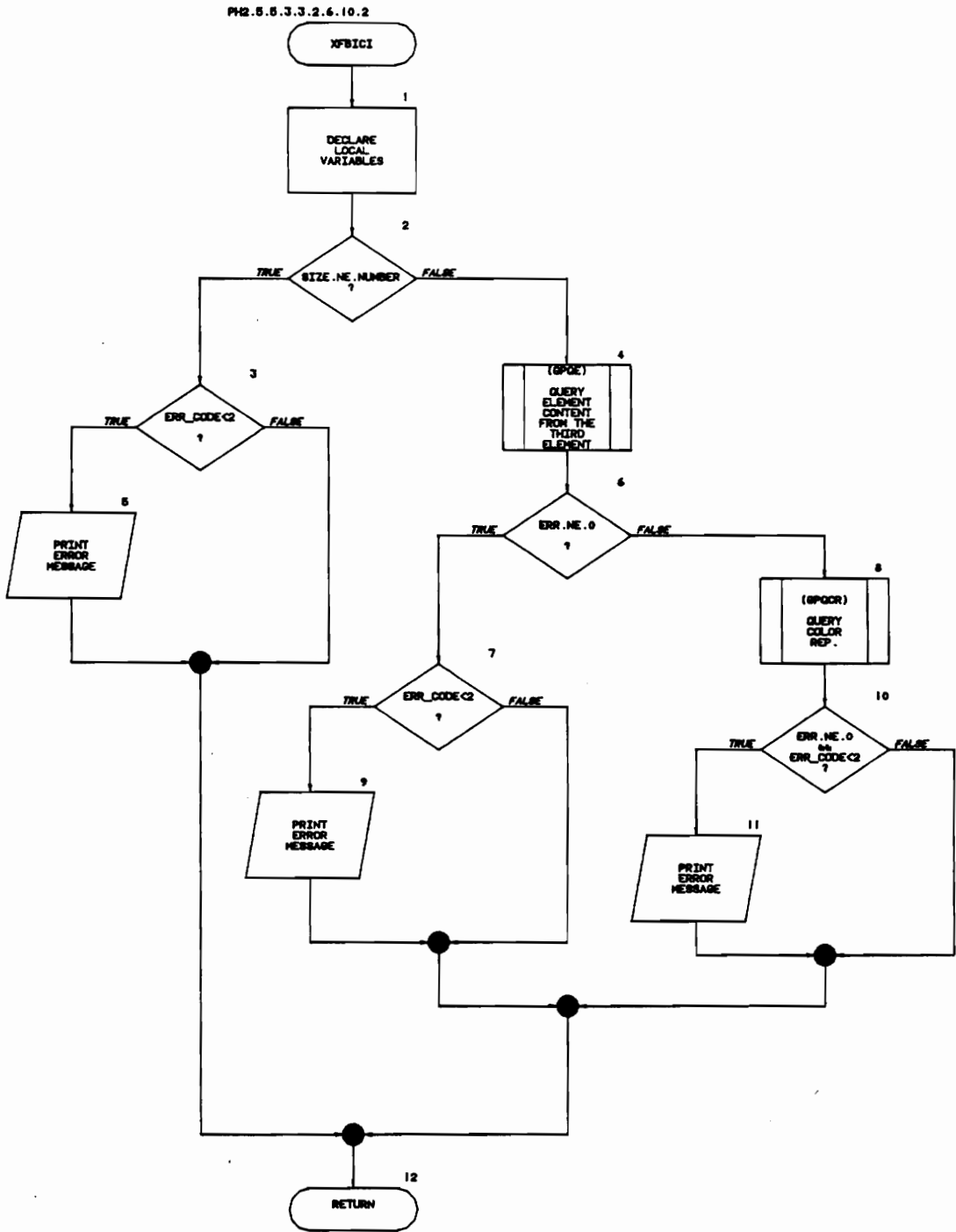
PH2.5.5.3.3.2.6.8



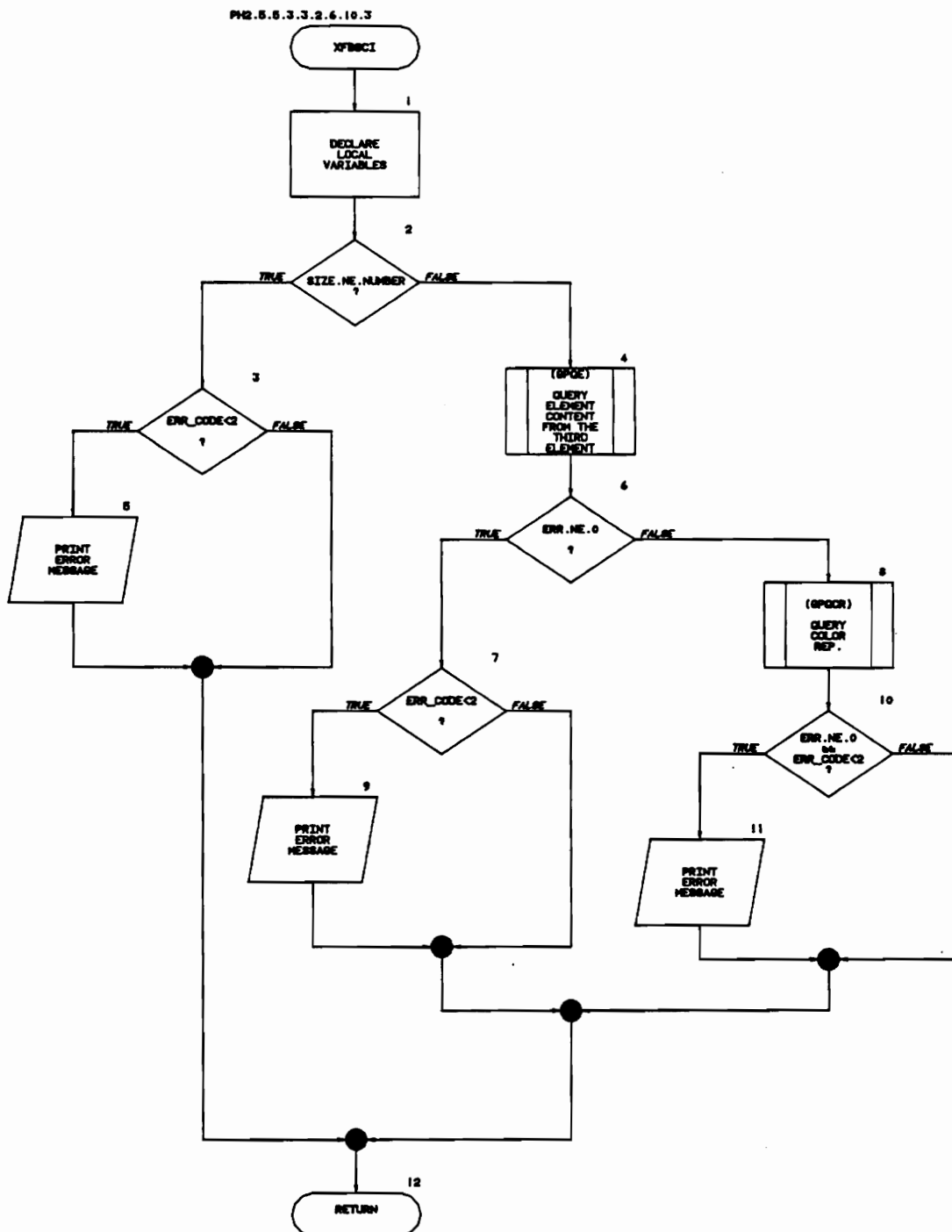
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	PRXF_APIEL	MODULE:PH2.5.5.3.3.2.6.10
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES THE STRUCTURE ELEMENTS CREATED BY THE API AND RETRIEVES THE INFORMATION
DATE:	5/24/90	



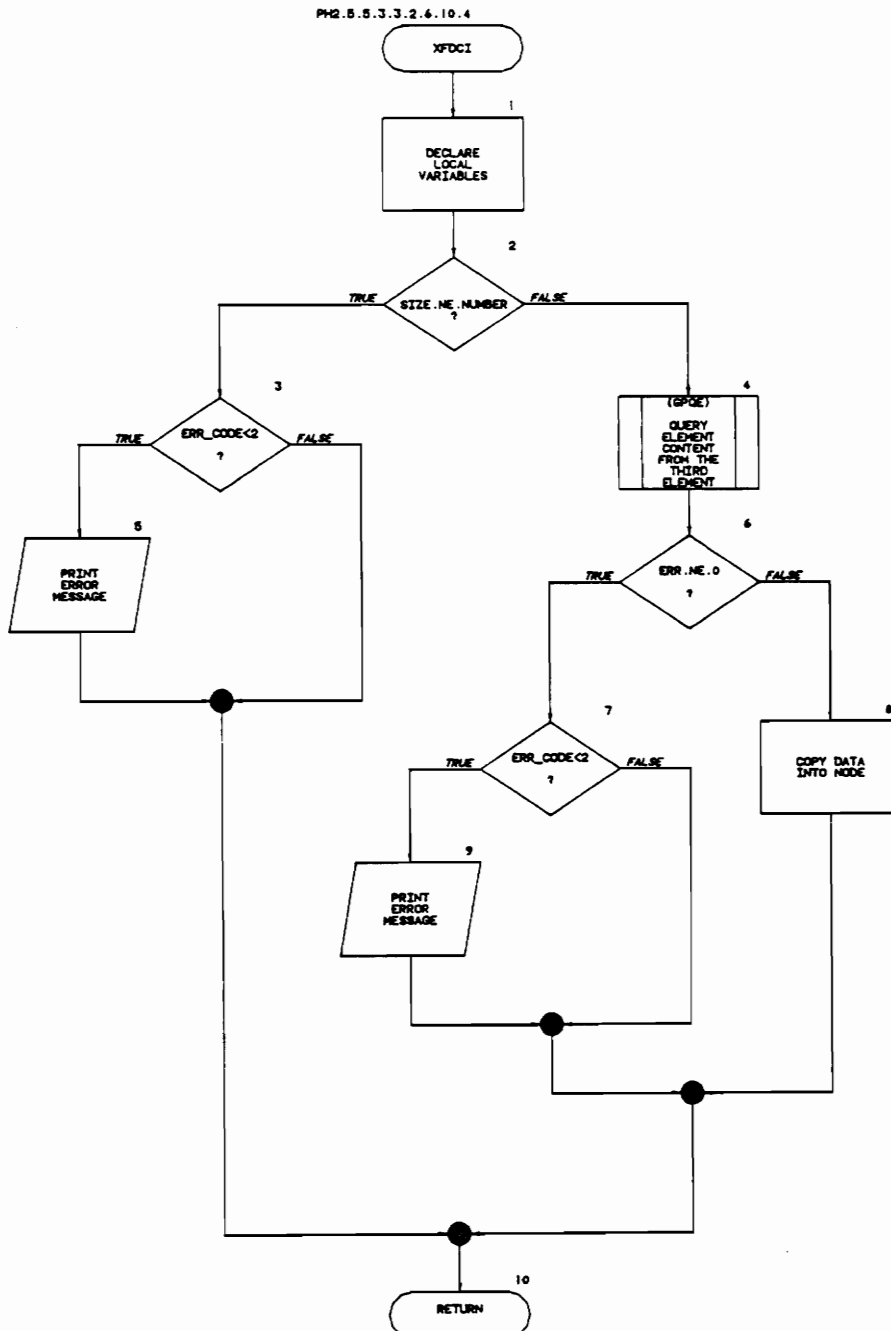
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	XFBICI	MODULE:PH2.5.5.3.3.2.6.10.2
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK INTERIOR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



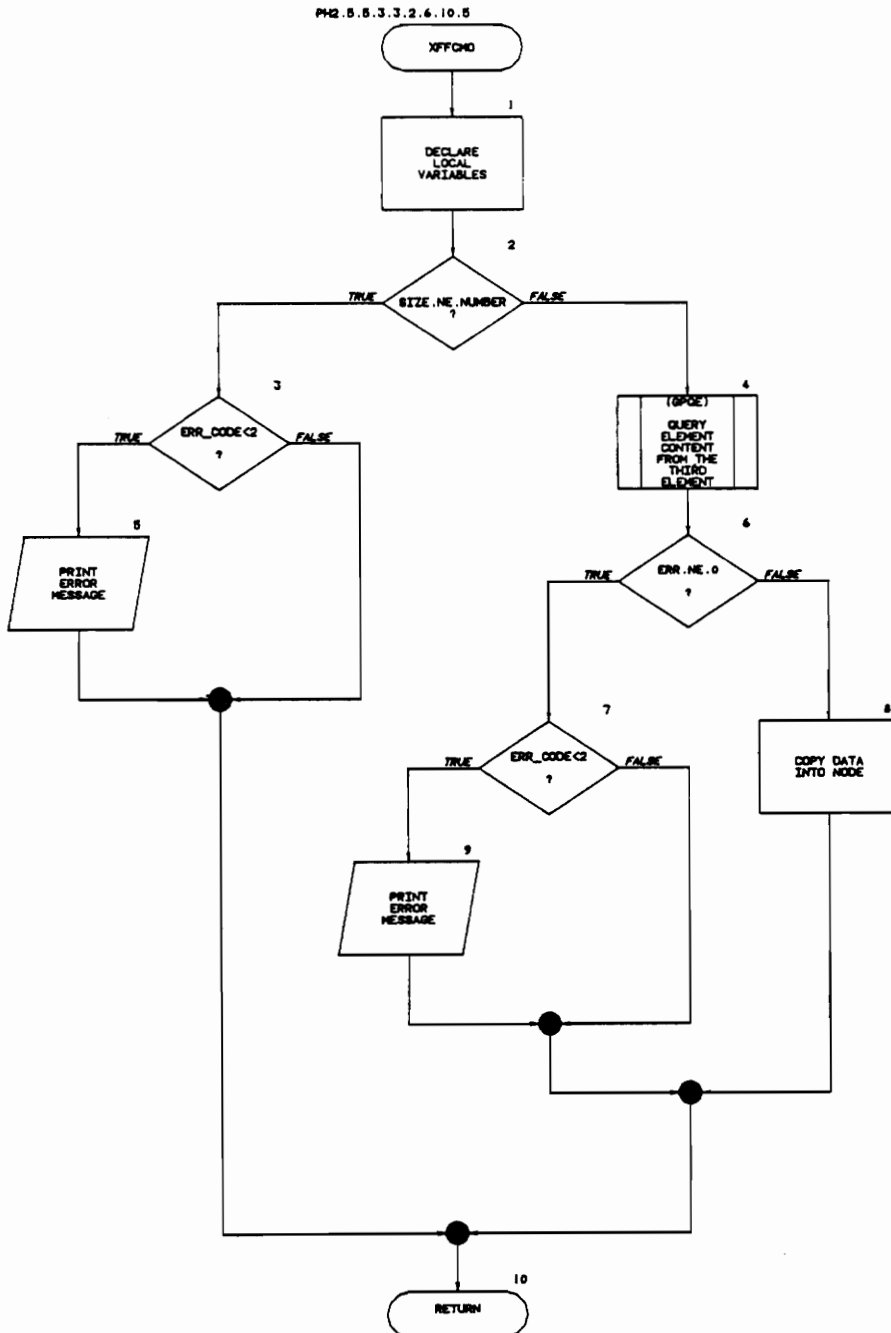
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	XFBSOI	MODULE:PH2.5.5.3.3.2.6.10.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK SPECULAR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



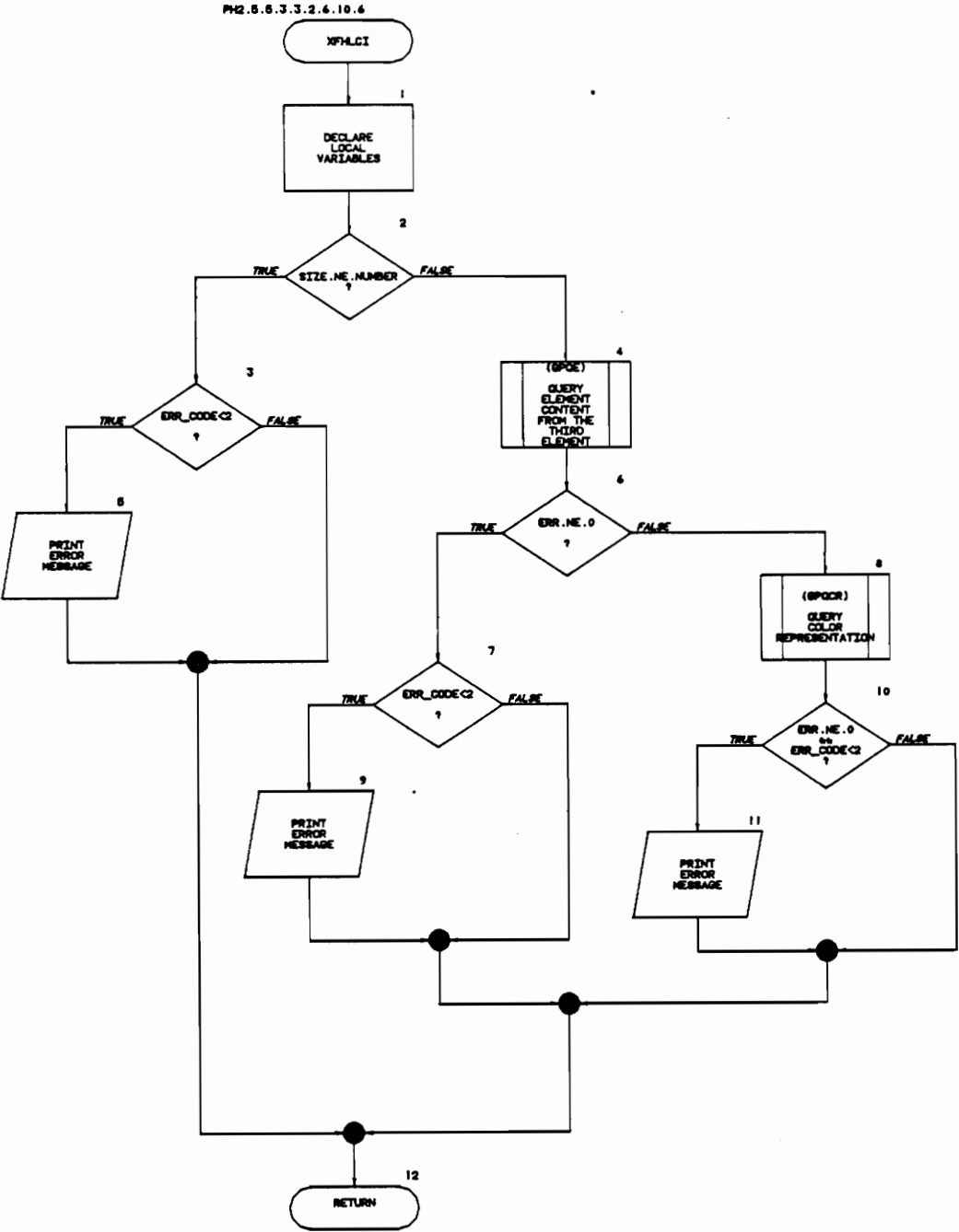
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: XFDCI	MODULE: PH2.5.5.3.3.2.6.10.4
DESIGNED BY: KRISHNAN KOLADY	NOTE: RETRIEVES DEPTH CUE INDEX FROM CURRENTLY OPEN STRUCTURE
DATE: 3/23/90	



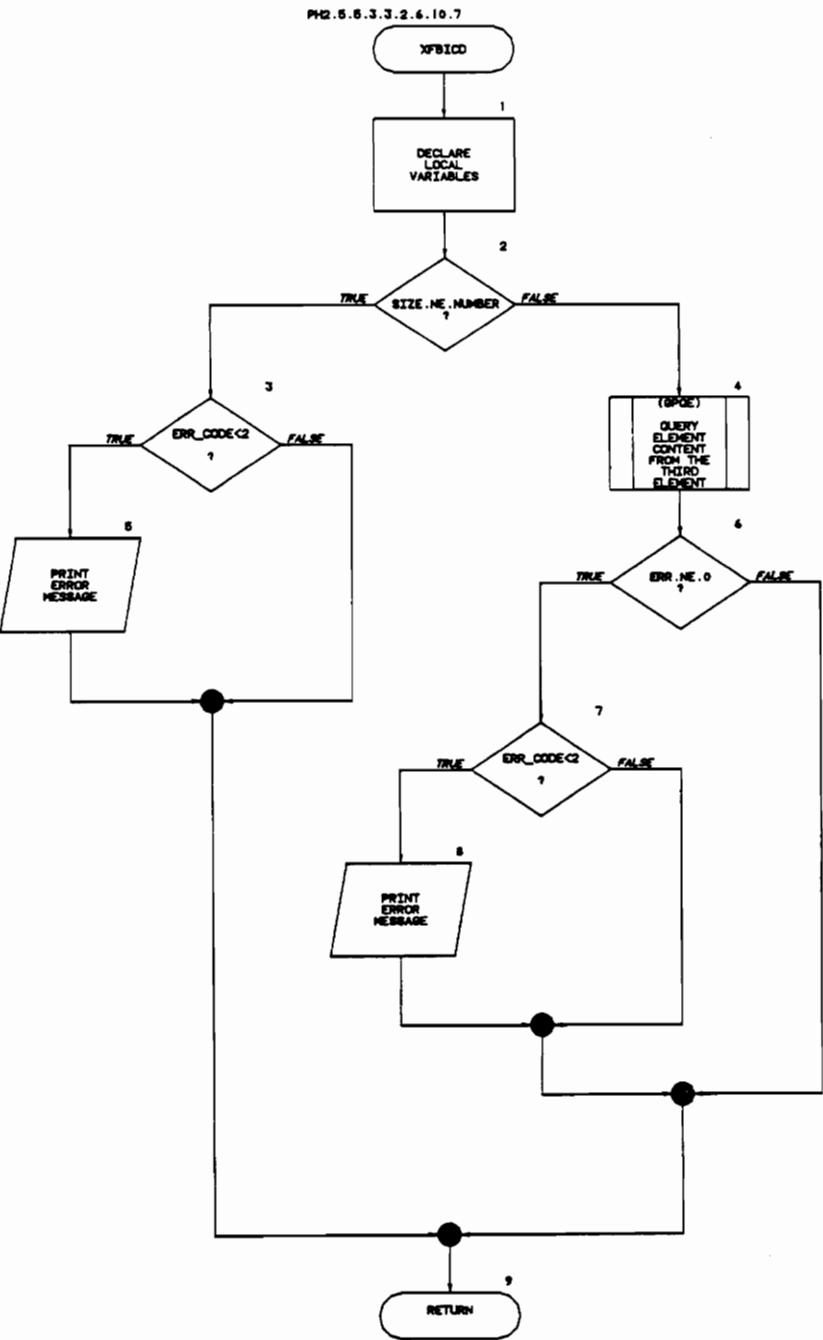
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: XFFCMO	MODULE: PH2.5.5.3.3.2.6.10.5
DESIGNED BY: KRISHNAN KOLADY	NOTE: RETRIEVES FACE CULLING MODE FROM CURRENTLY OPEN STRUCTURE
DATE: 3/23/90	



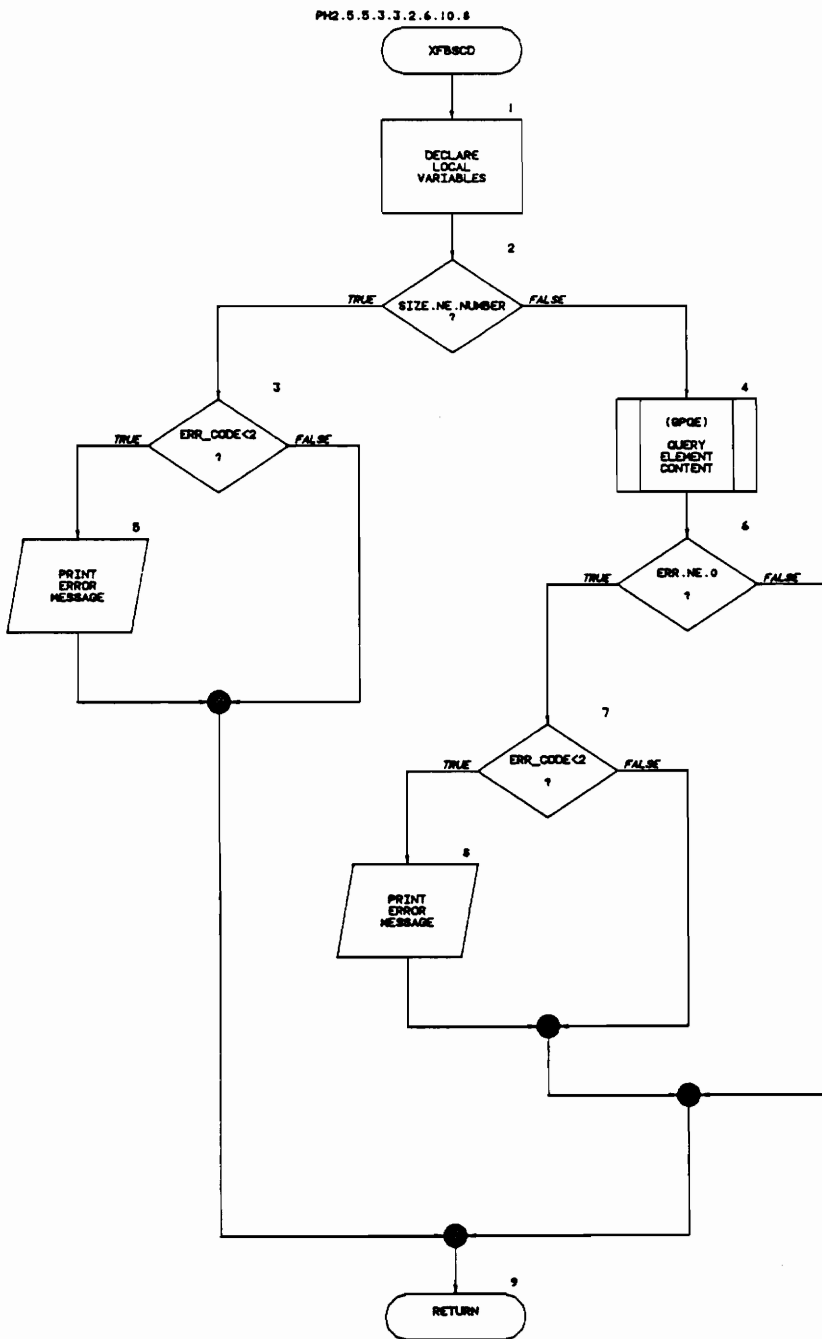
VT			PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:			MODULE: PH2.5.5.3.3.2.6.10.6		
DESIGNED BY: KRISHNAN KOLADY			NOTE: RETRIEVES HIGHLIGHTING COLOR INDEX FROM CURRENTLY OPEN STRUCTURE		
DATE: 3/23/90					




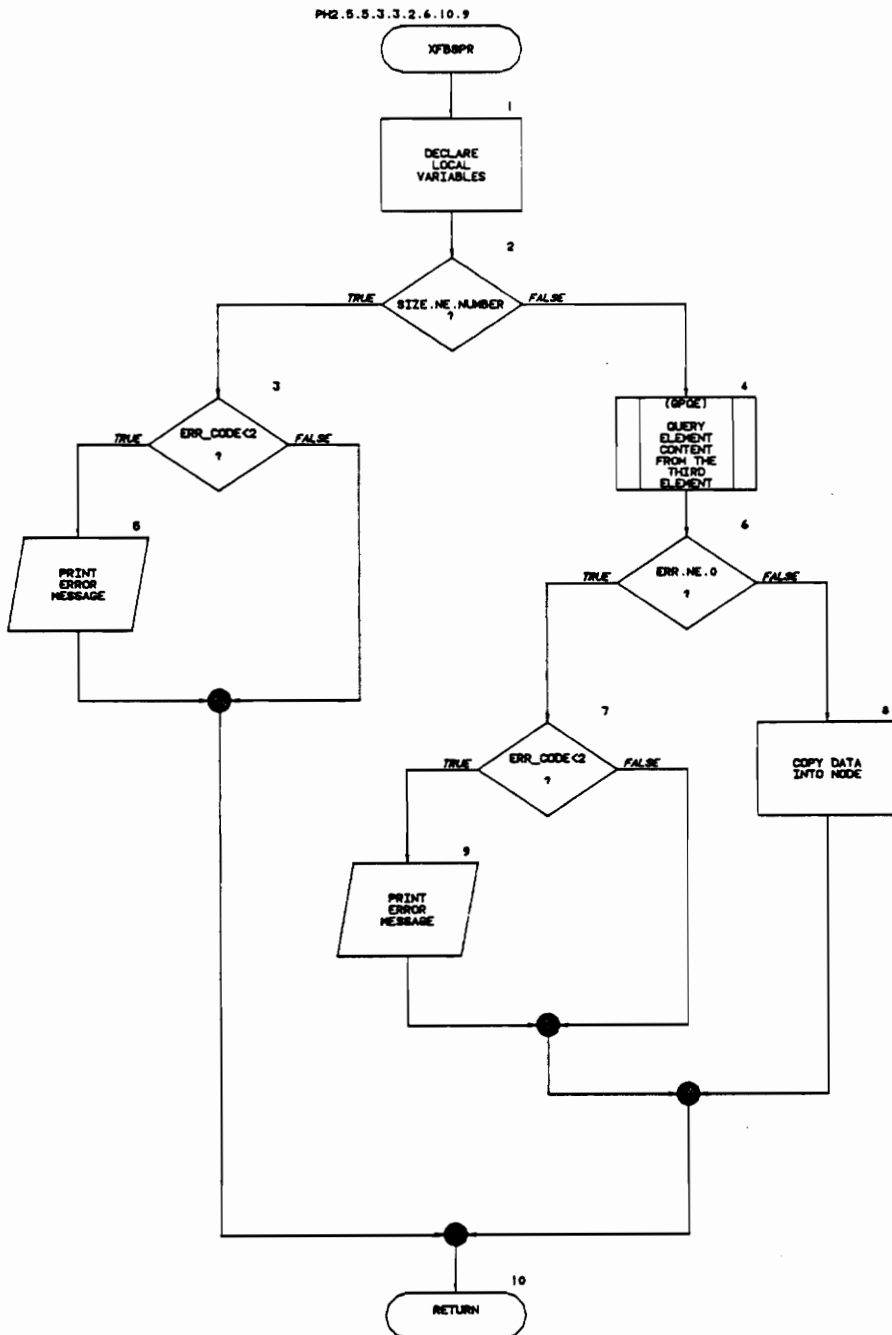
PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	XFBICD	MODULE:PH2.5.5.3.3.2.6.10.7
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK INTERIOR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



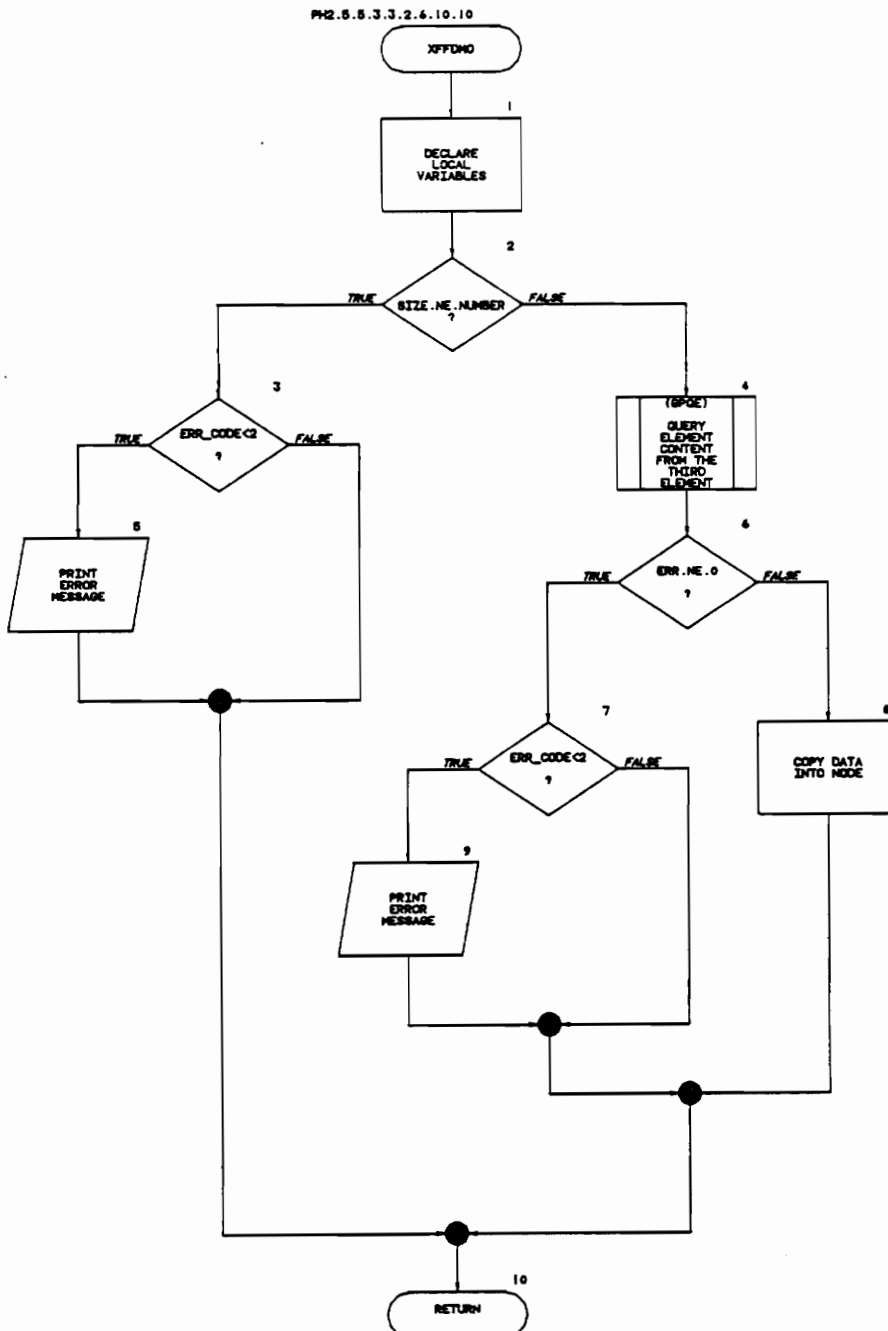
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	XFBSCD	MODULE:PH2.5.5.3.3.2.6.10.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK SPECULAR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



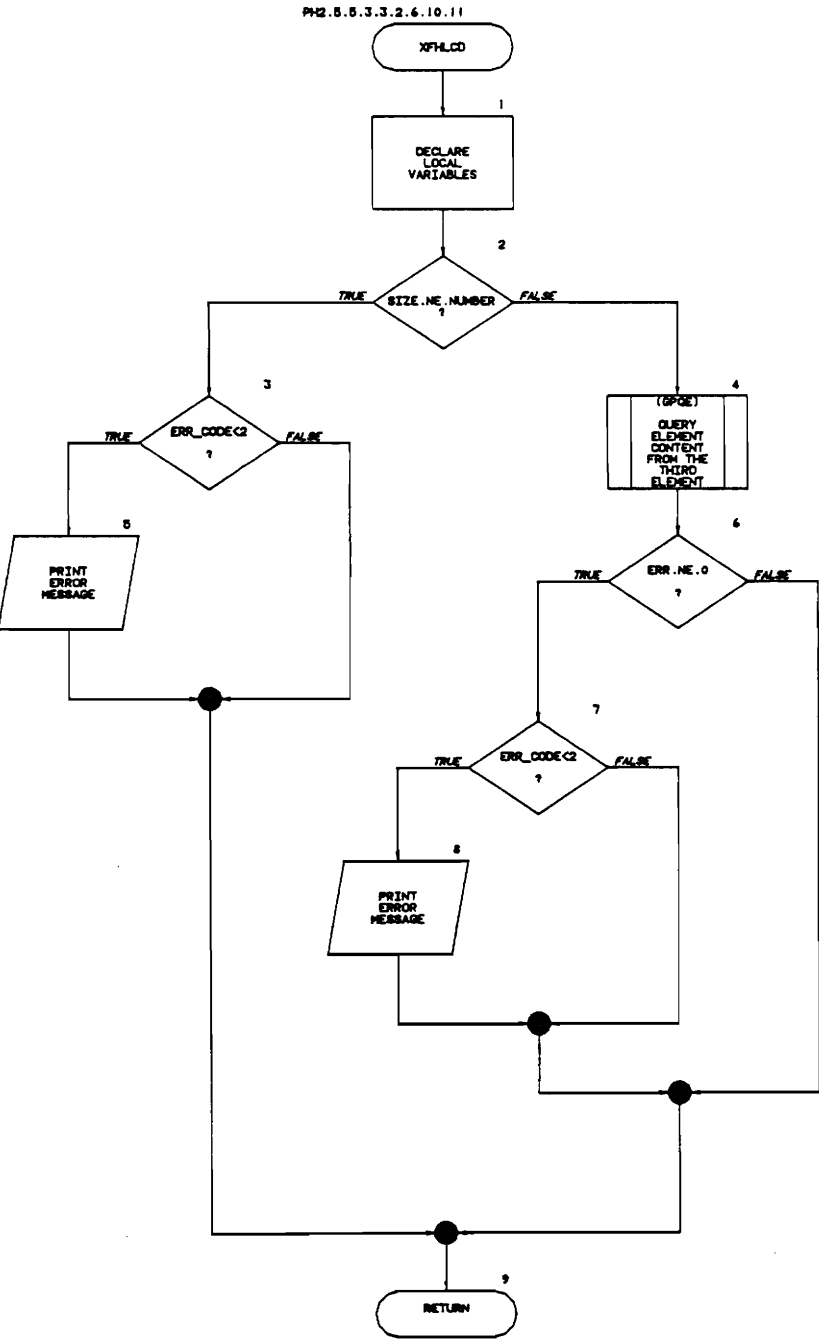
<div>  PROGRAM SPECIFICATION - PHONG SHADING </div>		
MODULE NAME:	XFBSPR	MODULE:PH2.5.5.3.3.2.6.10.9
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK SURFACE PROPERTIES FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



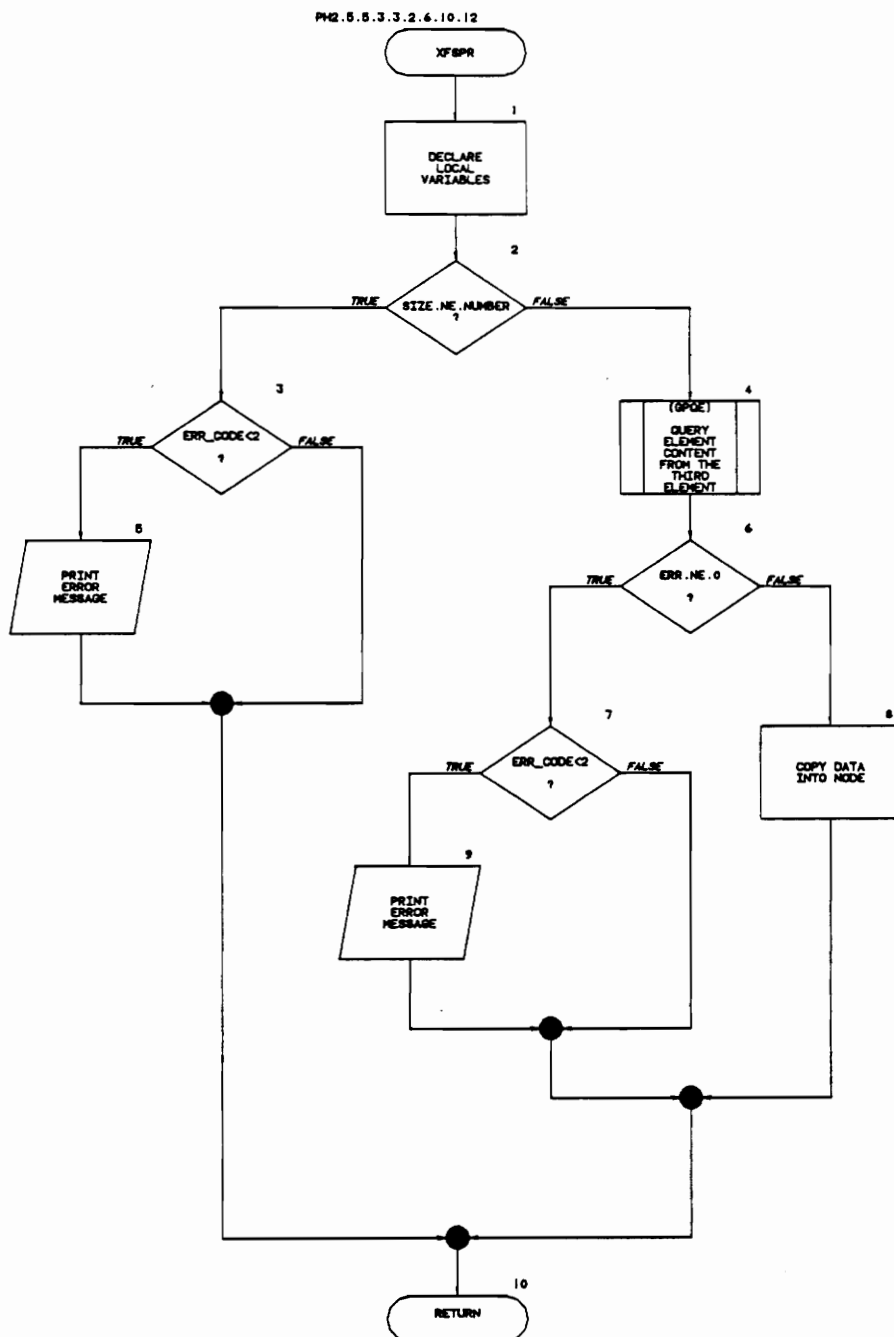
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: XFFDMO	MODULE: PH2.5.5.3.3.2.6.10.10
DESIGNED BY: KRISHNAN KOLADY	NOTE: RETRIEVES FACE DISTINGUISHING MODE FROM CURRENTLY OPEN STRUCTURE
DATE: 3/23/90	



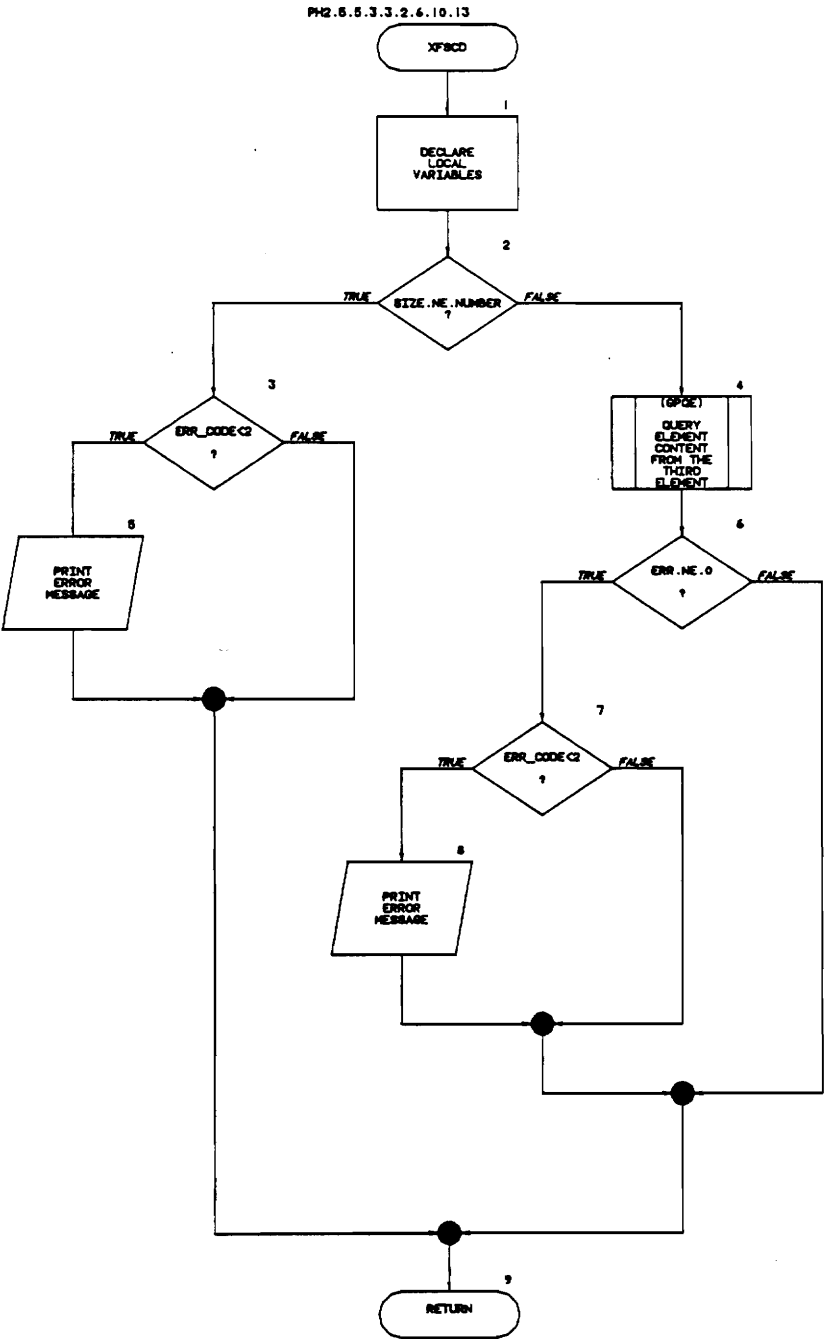
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: XFHLCD	MODULE:PH2.5.5.3.3.2.6.10.11
DESIGNED BY: KRISHNAN KOLADY	NOTE:RETRIEVES HIGHLIGHTING COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE: 3/23/90	



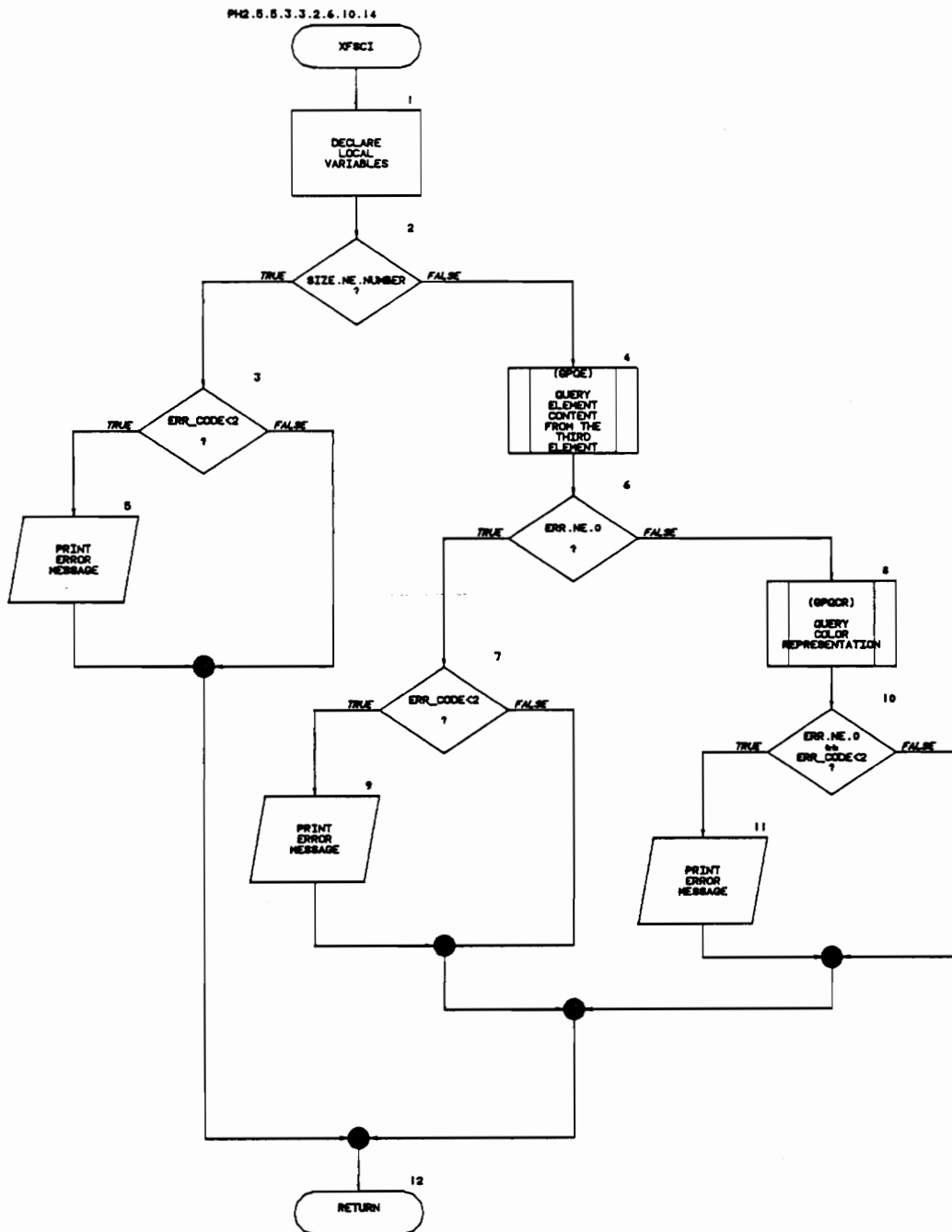
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	XFSPR	MODULE:PH2.5.5.3.3.2.6.10.12
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES SURFACE PROPERTIES FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



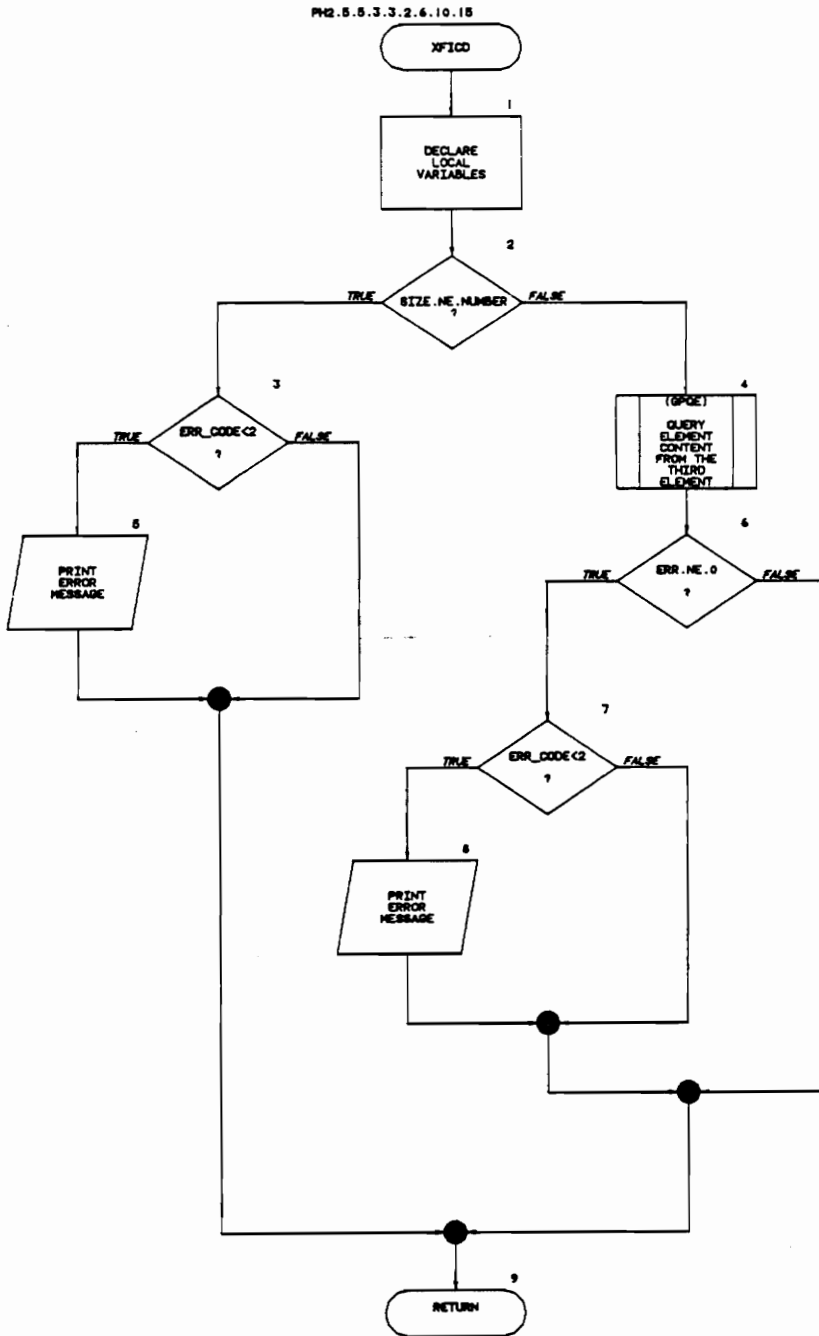
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	XFSCD	MODULE:PH2.5.5.3.3.2.6.10.13
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES SPECULAR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



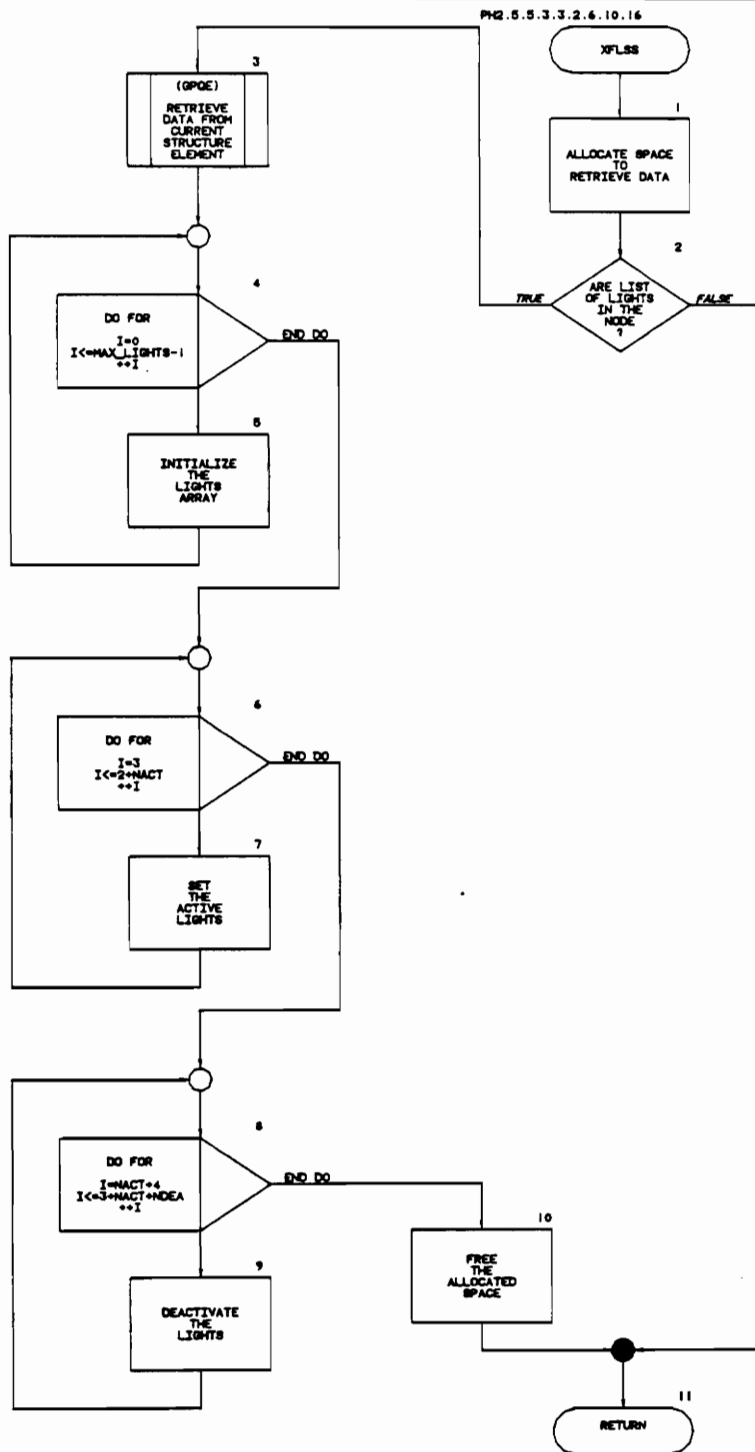
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: XFSCI		MODULE:PH2.5.5.3.3.2.6.10.14	
DESIGNED BY: KRISHNAN KOLADY		NOTE:RETRIEVES SPECULAR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE	
DATE: 3/23/90			



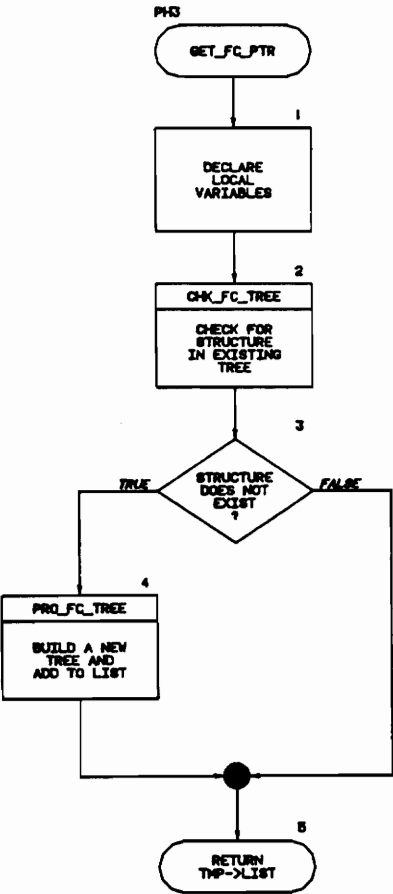
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	XFICD	MODULE:PH2.5.5.3.3.2.6.10.15
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES INTERIOR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	3/23/90	



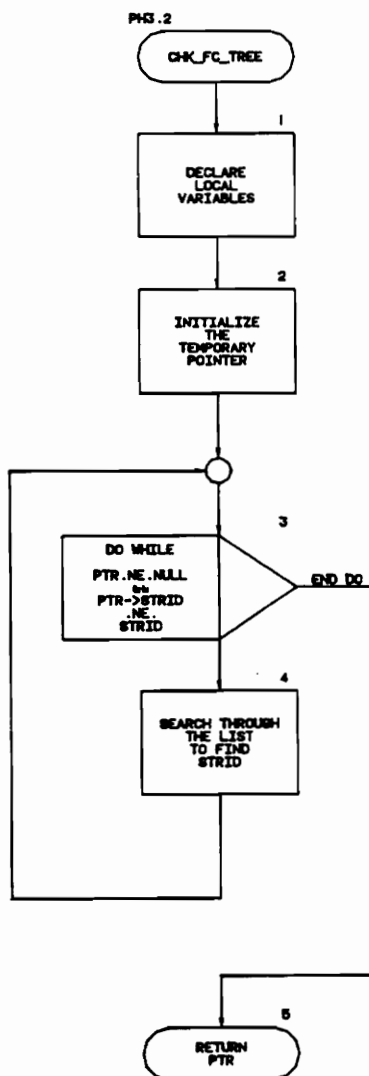
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: XFLSS		MODULE: PH2.5.5.3.3.2.6.10.16	
DESIGNED BY: KRISHNAN KOLADY		NOTE: RETRIEVES LIGHT SOURCE STATE FROM CURRENTLY OPEN STRUCTURE	
DATE: 3/23/90			



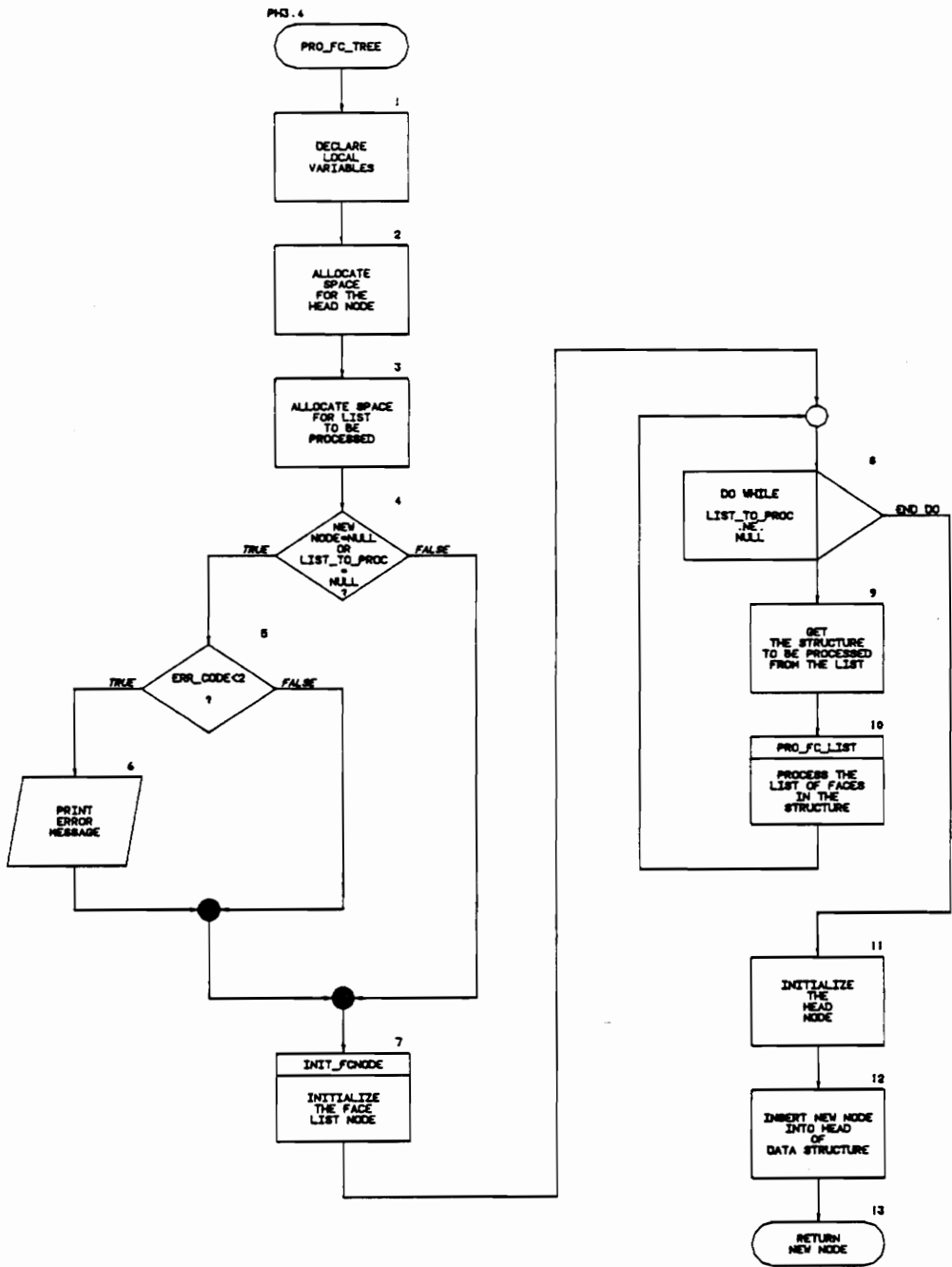
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	GET_FC_PTR	MODULE: PH3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: GETS THE POINTER TO THE FACE LIST FROM THE FACE LIST DATA STRUCTURE
DATE:	4/2/90	



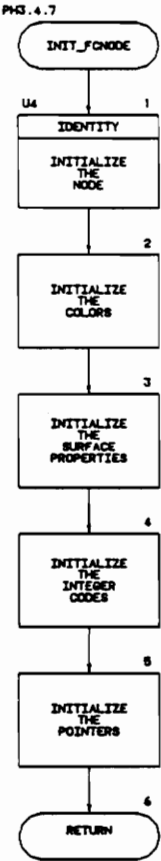
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>		
MODULE NAME:	CHK_FC_TREE	MODULE:PH3.2
DESIGNED BY:	KRISHNAN KOLADY	NOTE:CHECKS FACE TREE STRUCTURE FOR THE PROCESSED STRUCTURE IDENTIFIER
DATE:	3/29/90	



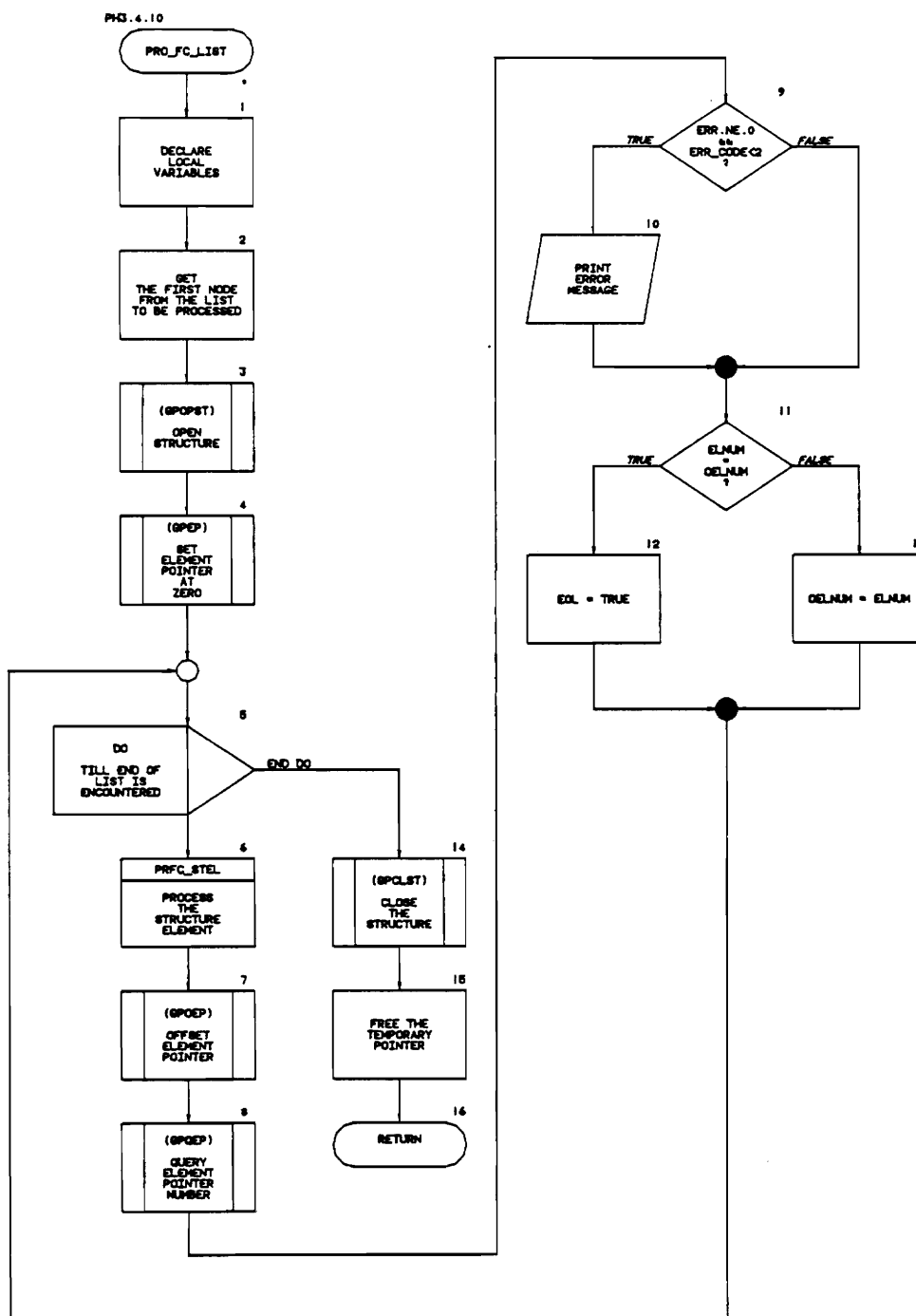
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRO_FC_TREE	MODULE: PH3.4
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES A NEW TREE FOR THE CURRENT STRUCTURE ID
DATE:	4/2/90	



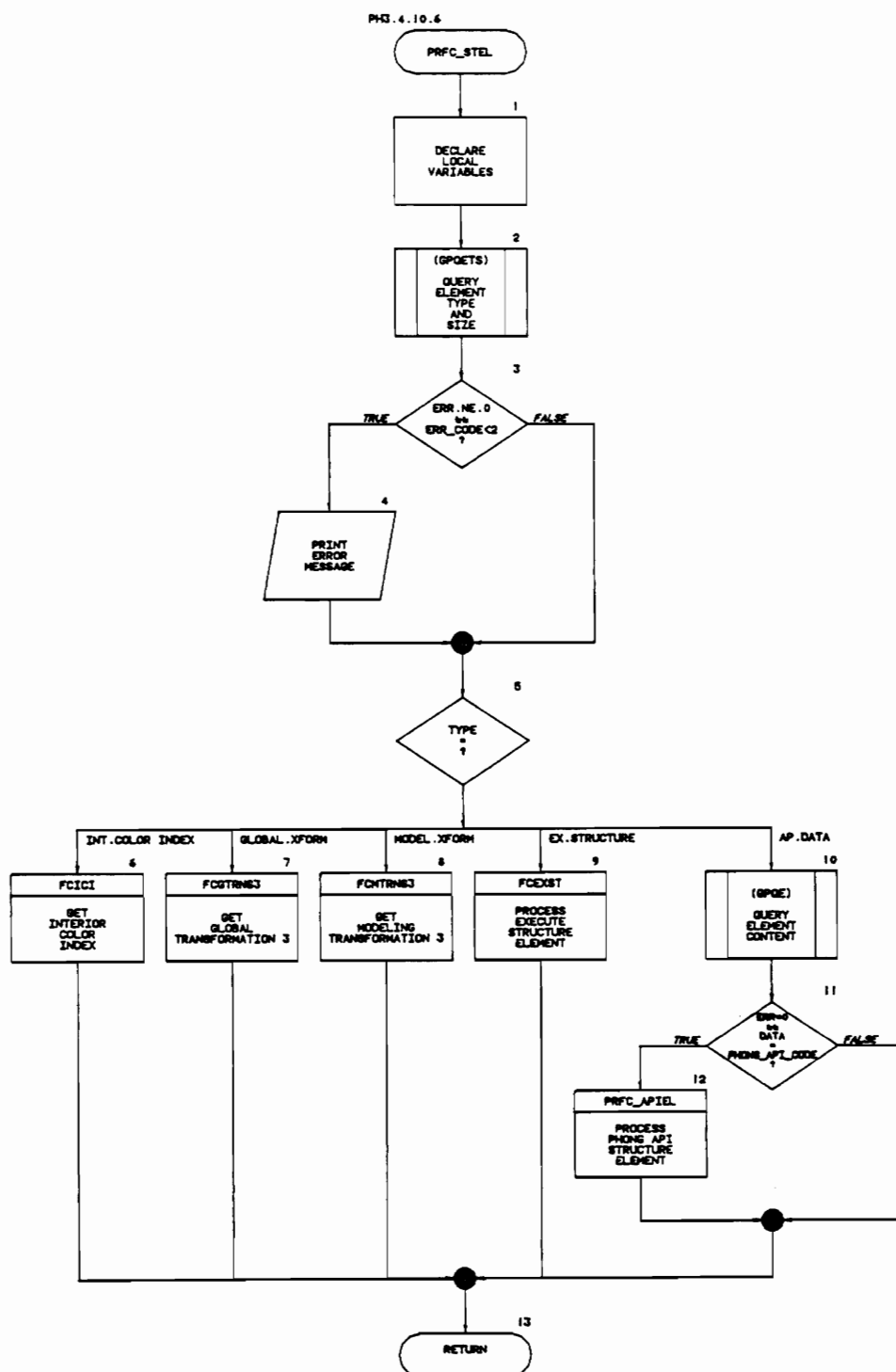
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	INIT_FCNODE	MODULE:PH3.4.7
DESIGNED BY:	KRISHNAN KOLADY	NOTE:INITIALIZES THE FACE LIST NODE
DATE:	4/2/90	



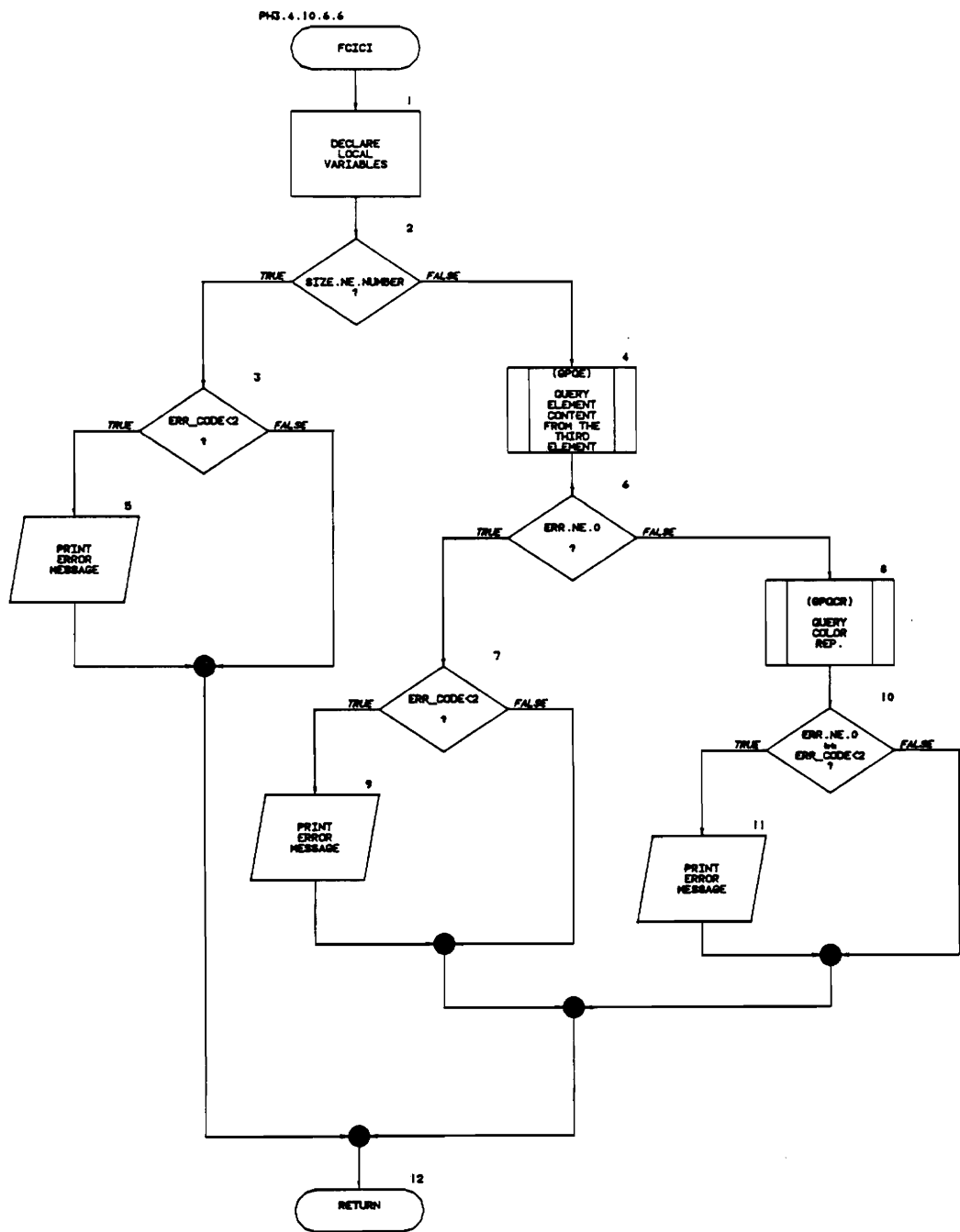
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRO_FC_LIST	MODULE: PH3.4.10
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CREATES THE LIST OF FACES WITH VERTICE AND ATTRIBUTE DATA IN THE CURRENT STRUCTURE
DATE:	4/2/90	



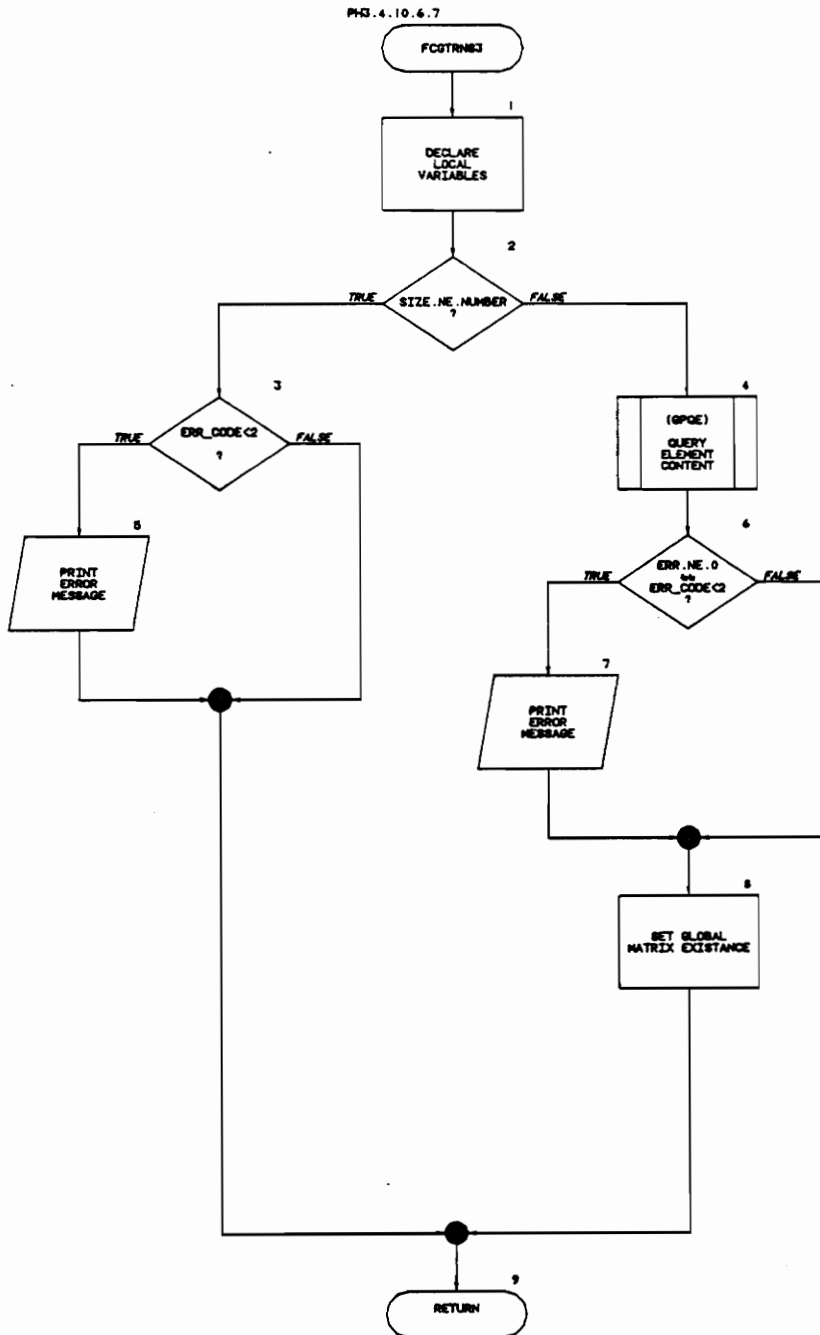
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME :	PRFC_STEL	MODULE:PH3.4.10.6
DESIGNED BY :	KRISHNAN KOLADY	NOTE: PROCESSES THE STRUCTURE ELEMENT OF CURRENTLY OPEN STRUCTURE AND RETRIEVES FACE, VERTEX, AND ATTRIBUTE DATA
DATE :	4/2/90	



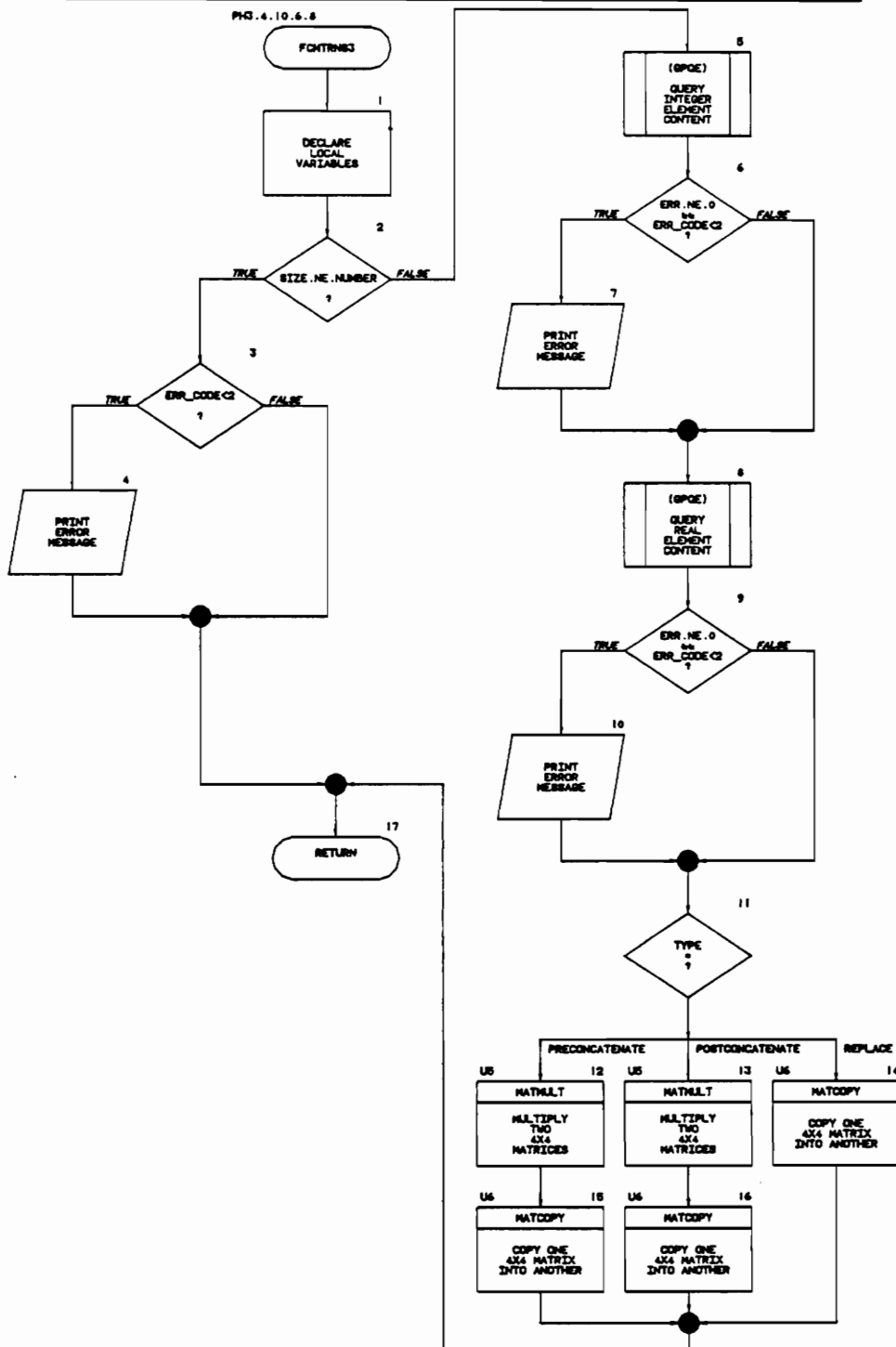
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCICI	MODULE:PH3.4.10.6.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES INTERIOR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



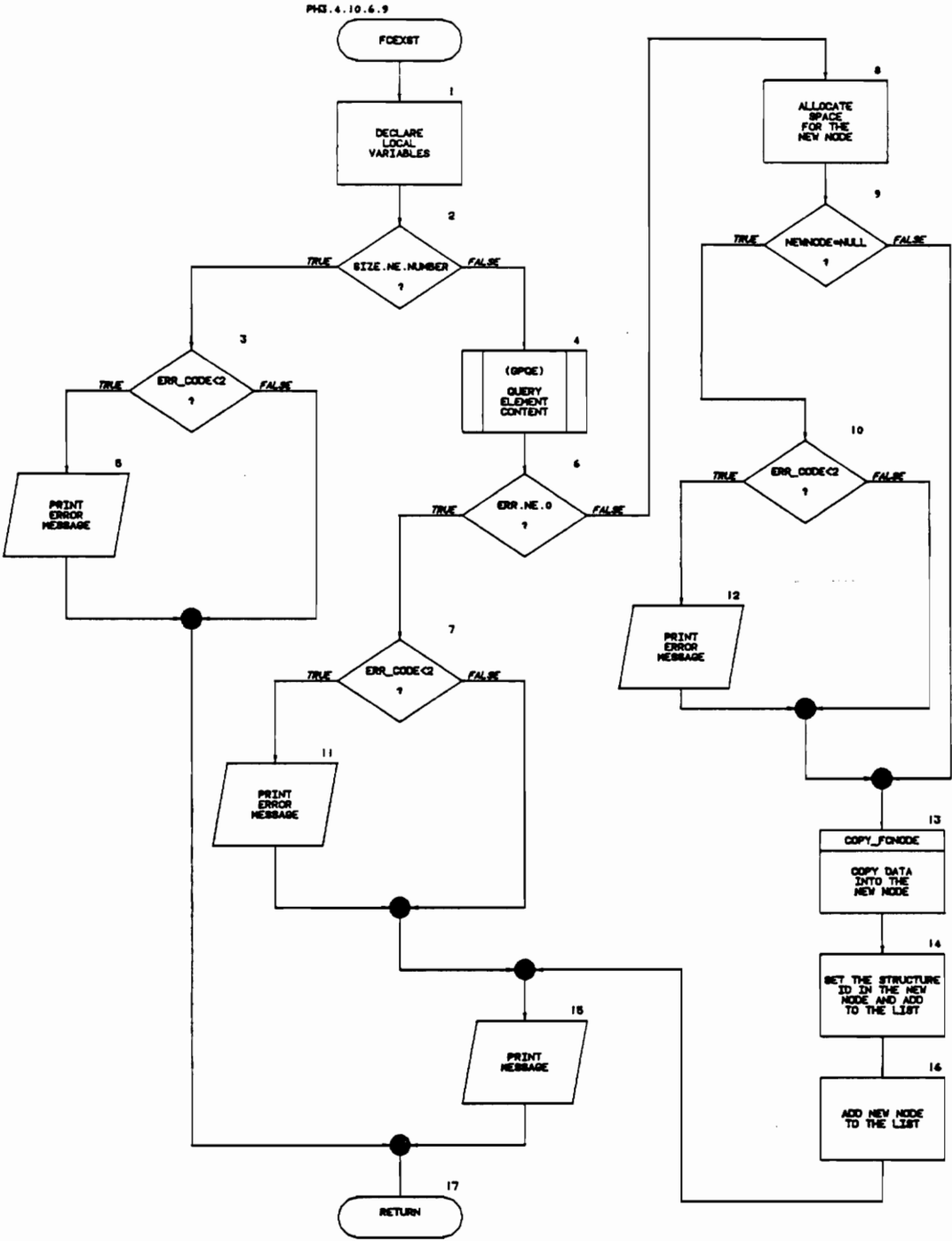
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCGTRNS3	MODULE: PH3.4.10.6.7
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES GLOBAL TRANSFORMATION MATRIX DATA FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



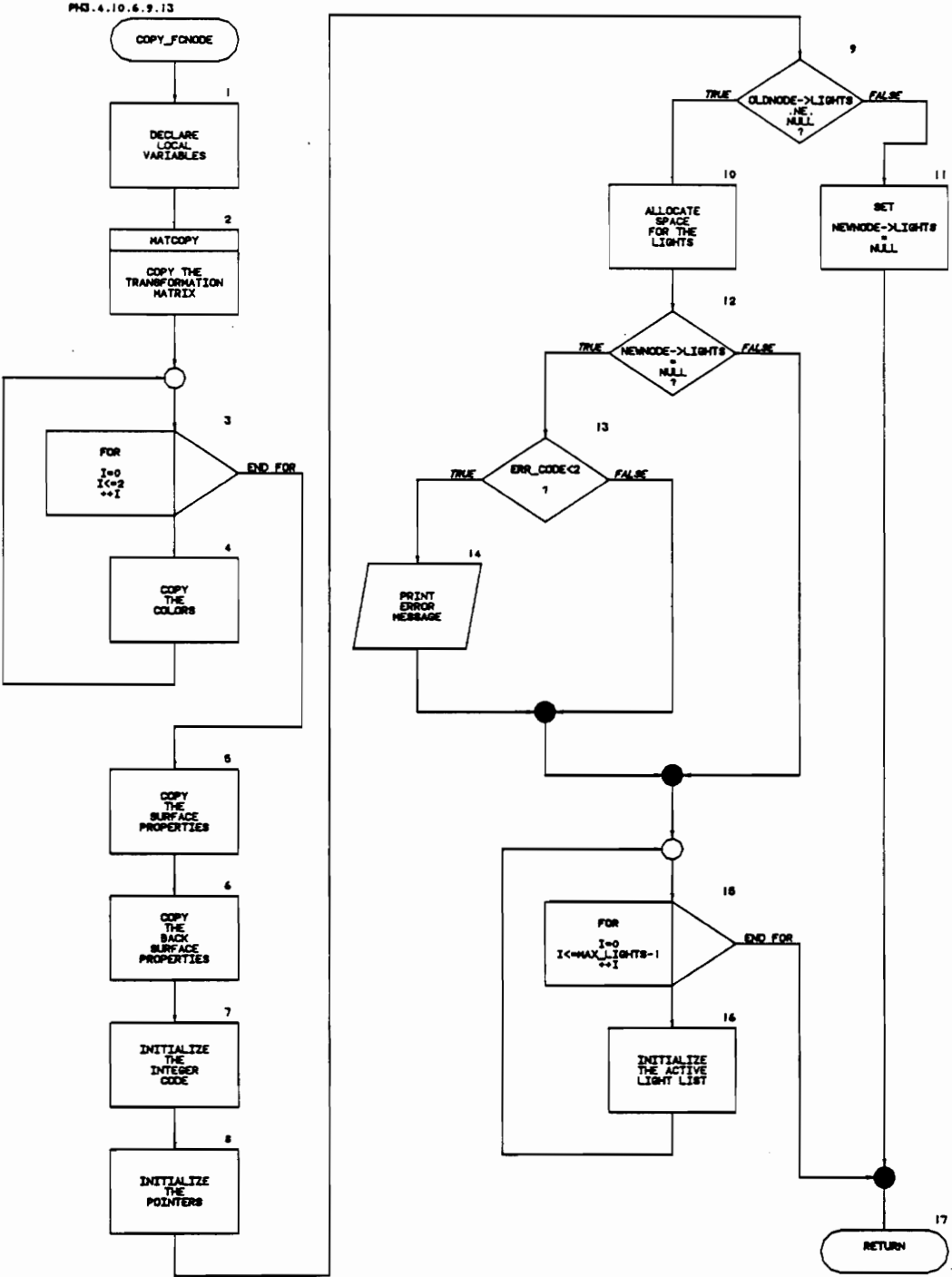
PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCMTRNS3	MODULE: PH3.4.10.6.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES MODELING TRANSFORMATION MATRIX DATA FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



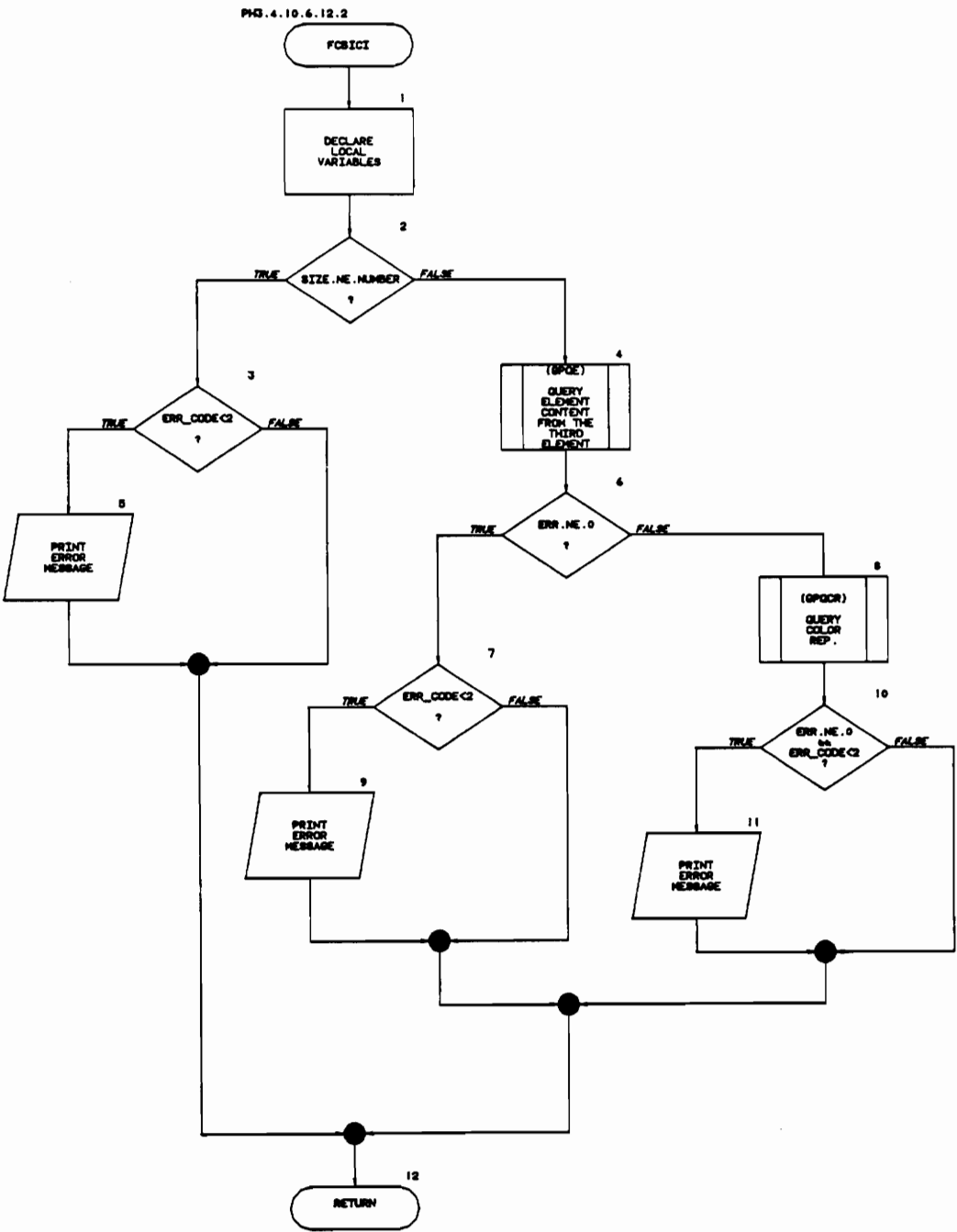
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCEXST	MODULE:PH3.4.10.6.9
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES EXECUTE STRUCTURE ELEMENT IN CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



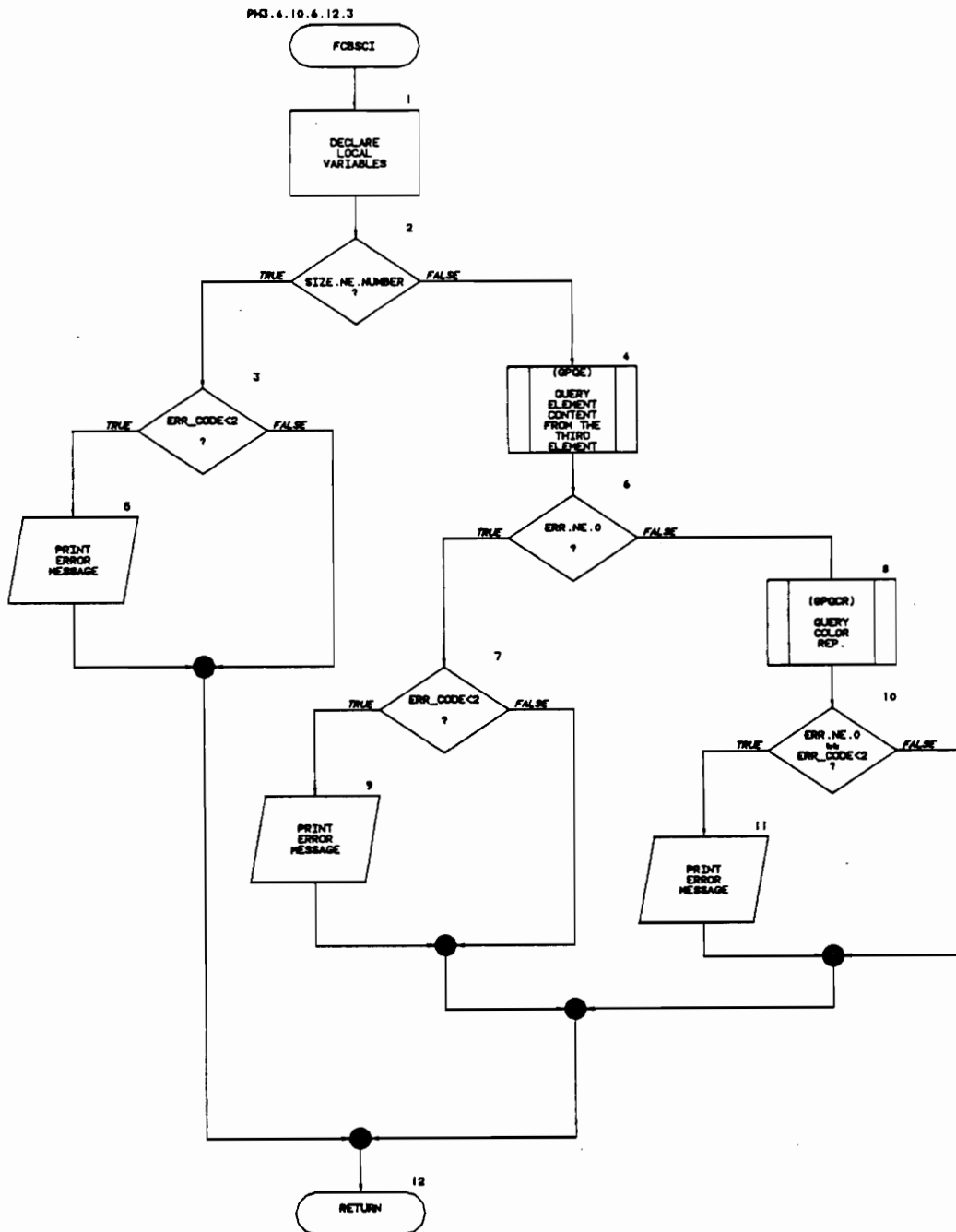
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: COPY_FCNODE	MODULE: PH3.4.10.6.9.13
DESIGNED BY: KRISHNAN KOLADY	NOTE: COPIES THE OLD NODE DATA INTO A NEW FACE LIST NODE
DATE: 3/29/90	



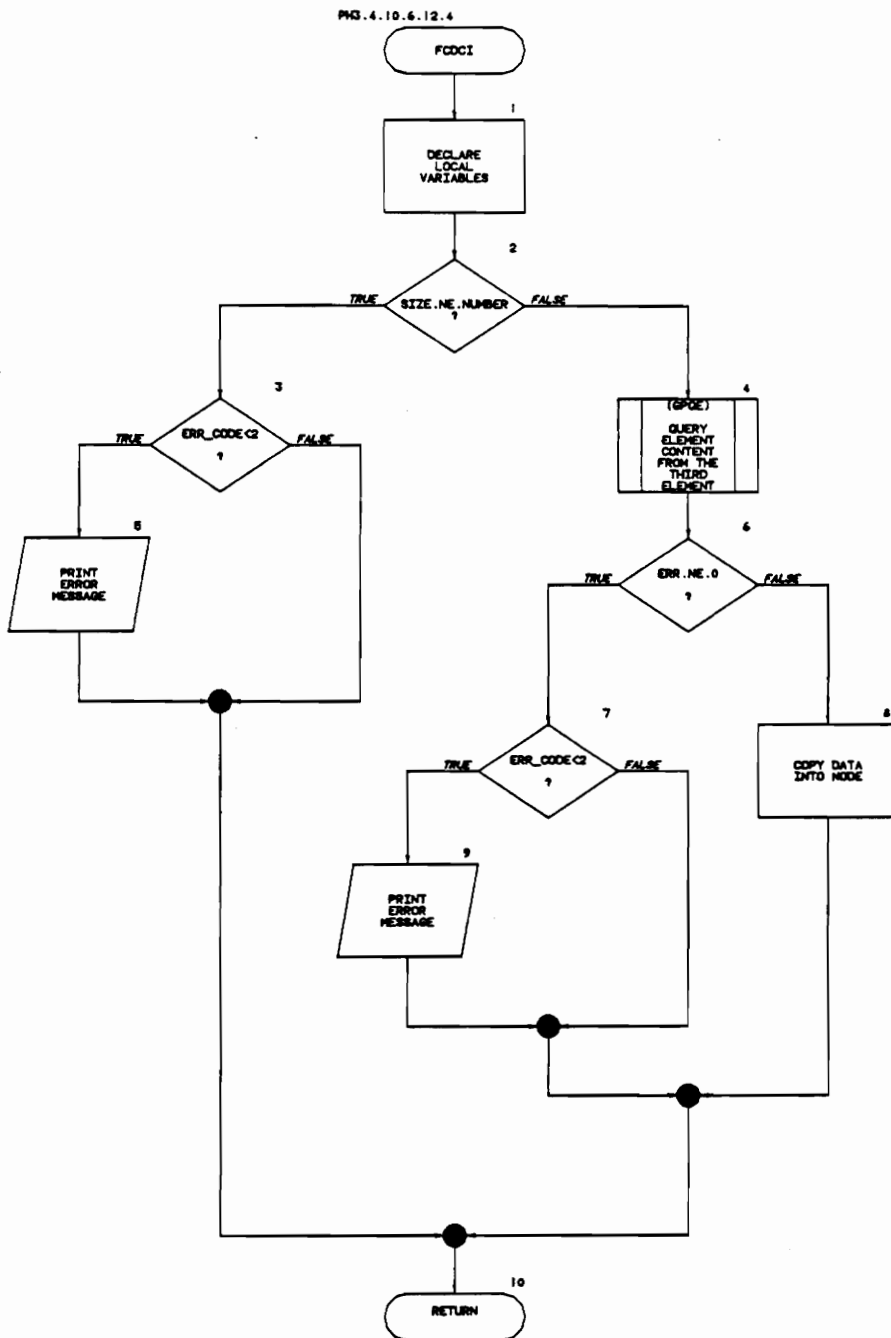
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCBICI	MODULE:PH3.4.10.6.12.2
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK INTERIOR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	3/29/90	



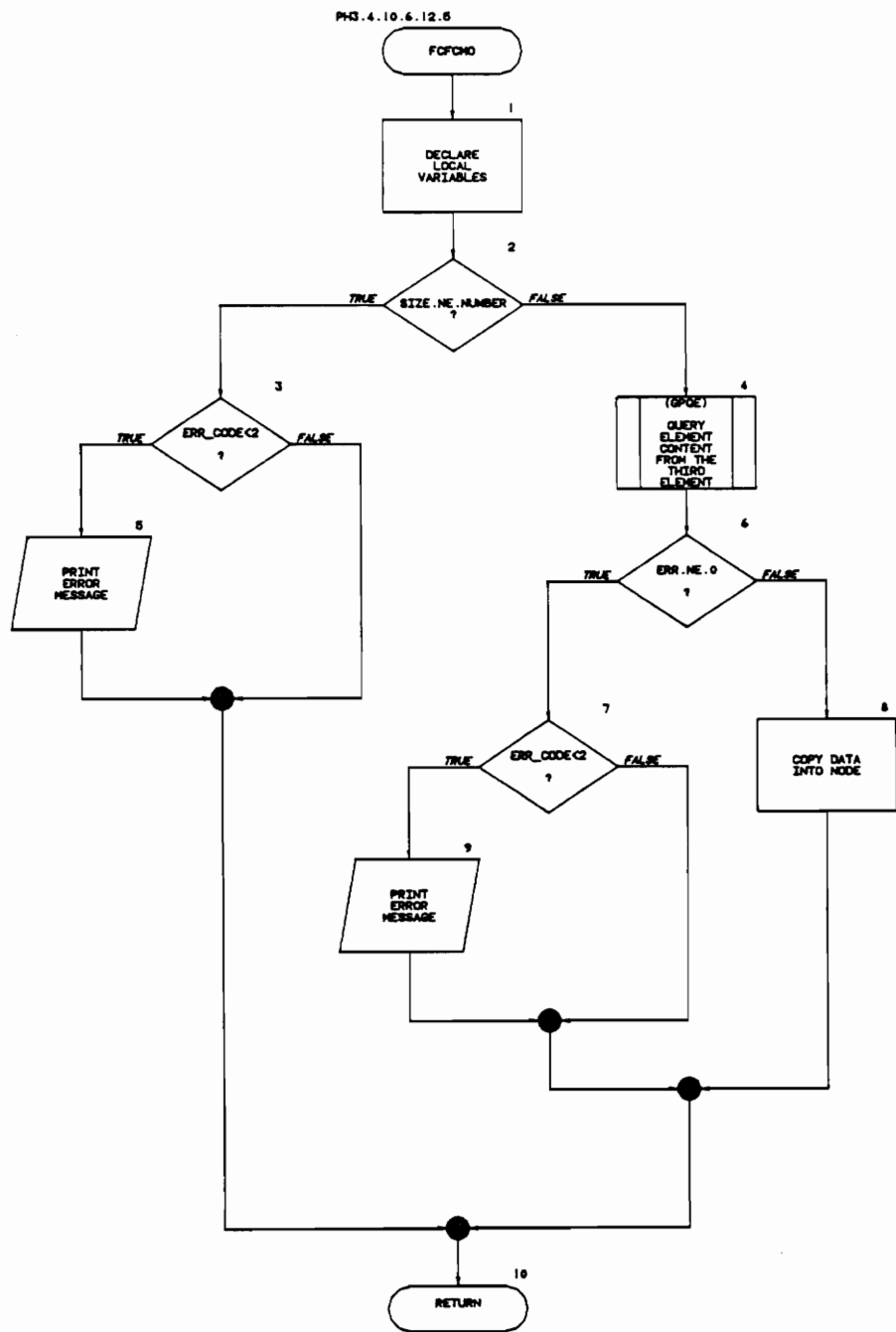
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCBSCI	MODULE:PH3.4.10.6.12.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK SPECULAR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



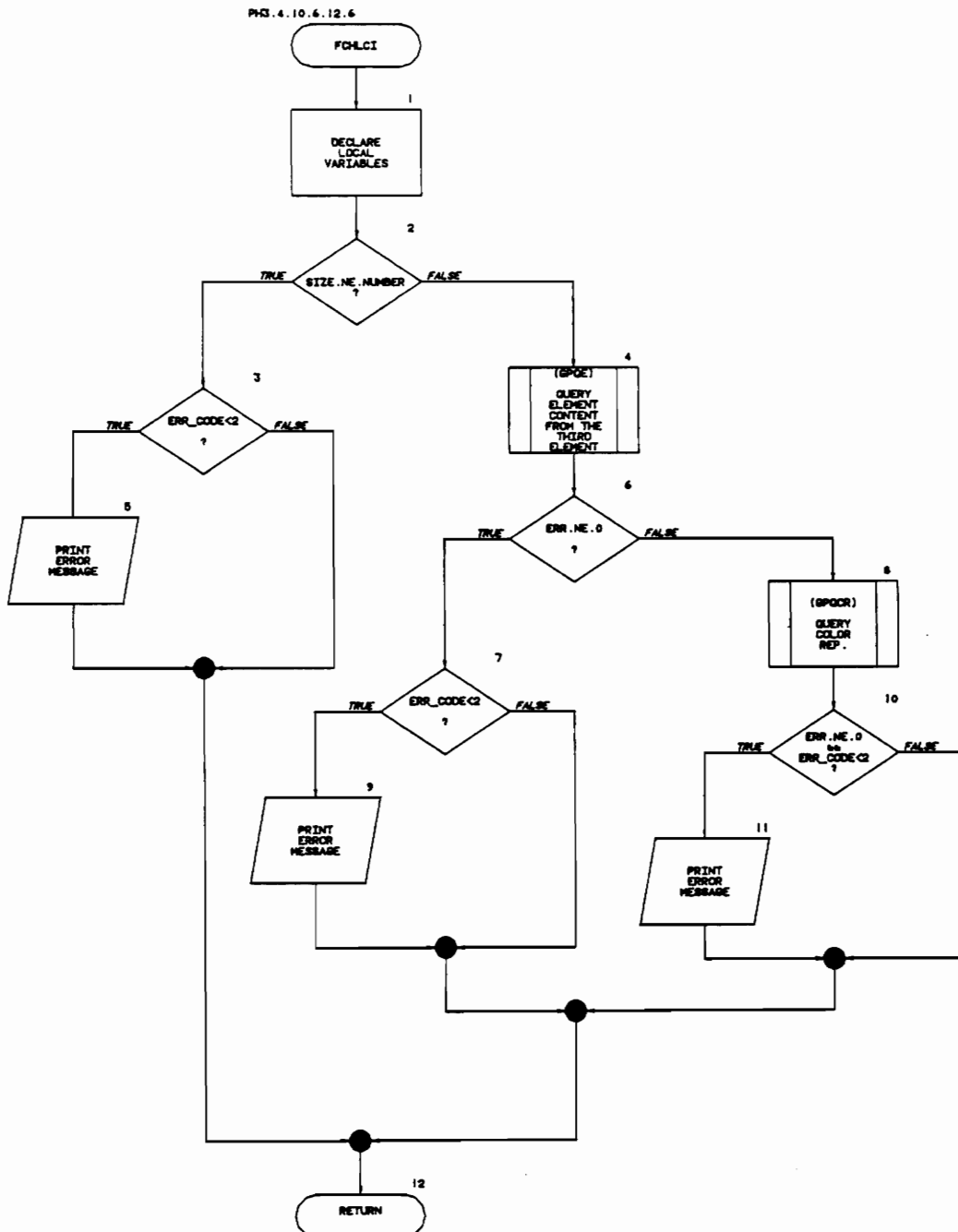
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCDCI	MODULE:PH3.4.10.6.12.4
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES DEPTH CUE INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



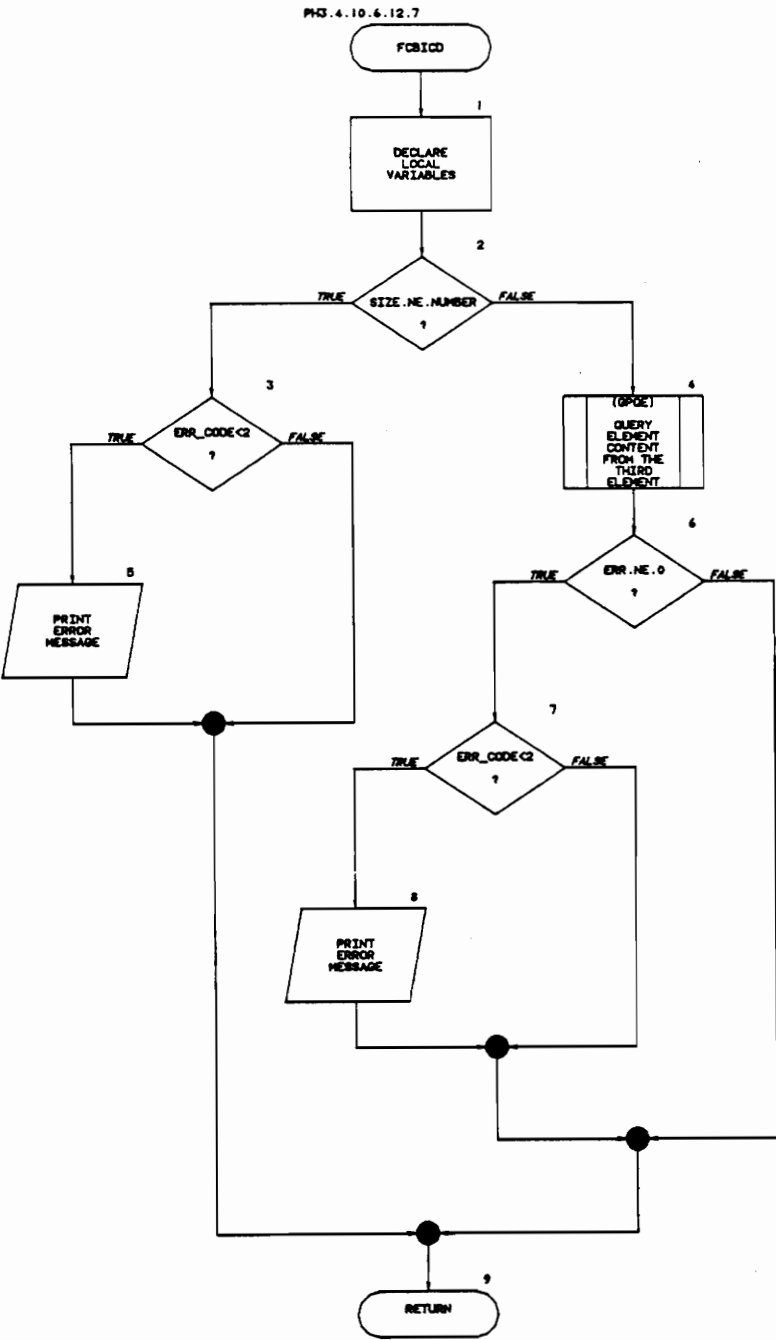
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCFCMO	MODULE:PH3.4.10.6.12.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES FACE CULLING MODE FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



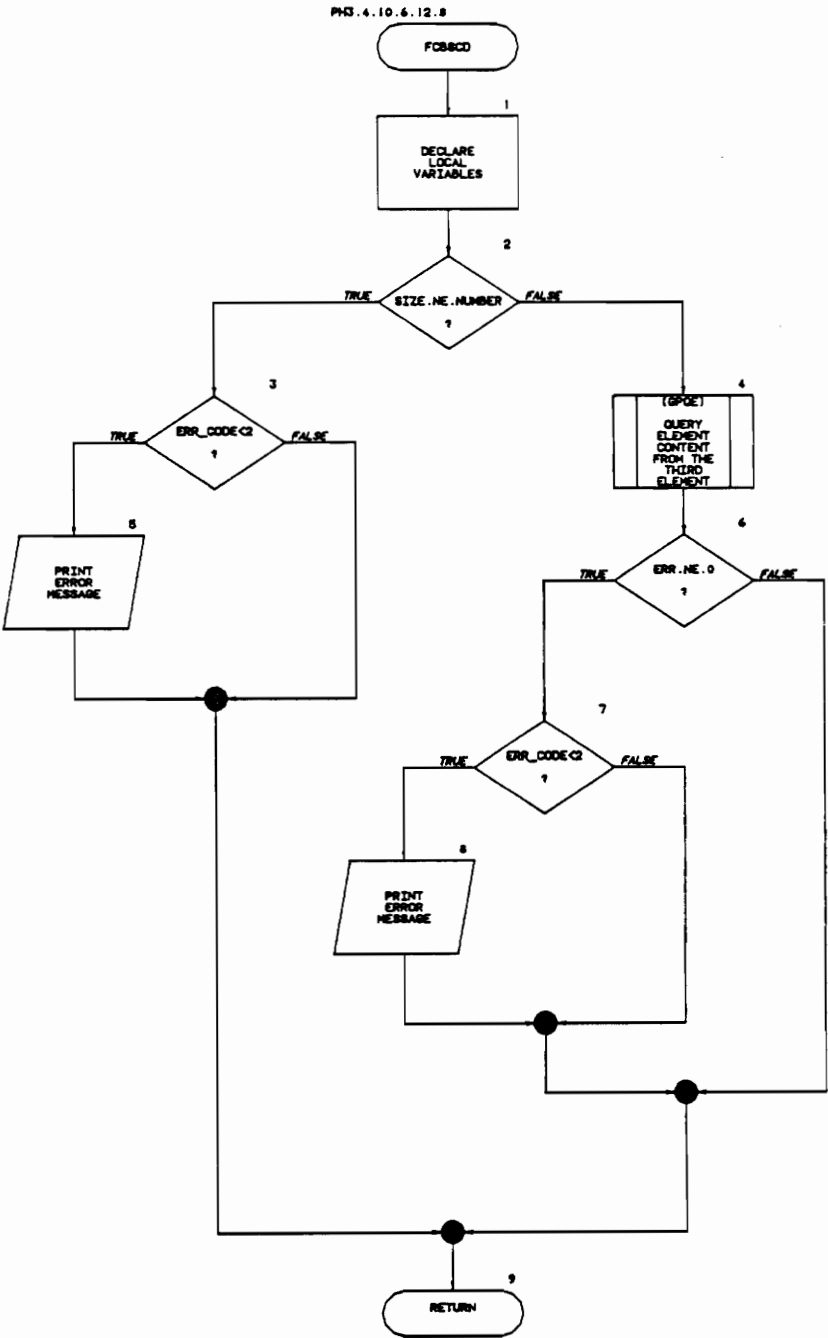
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCHLCI	MODULE:PH3.4.10.6.12.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES HIGHLIGHTING COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



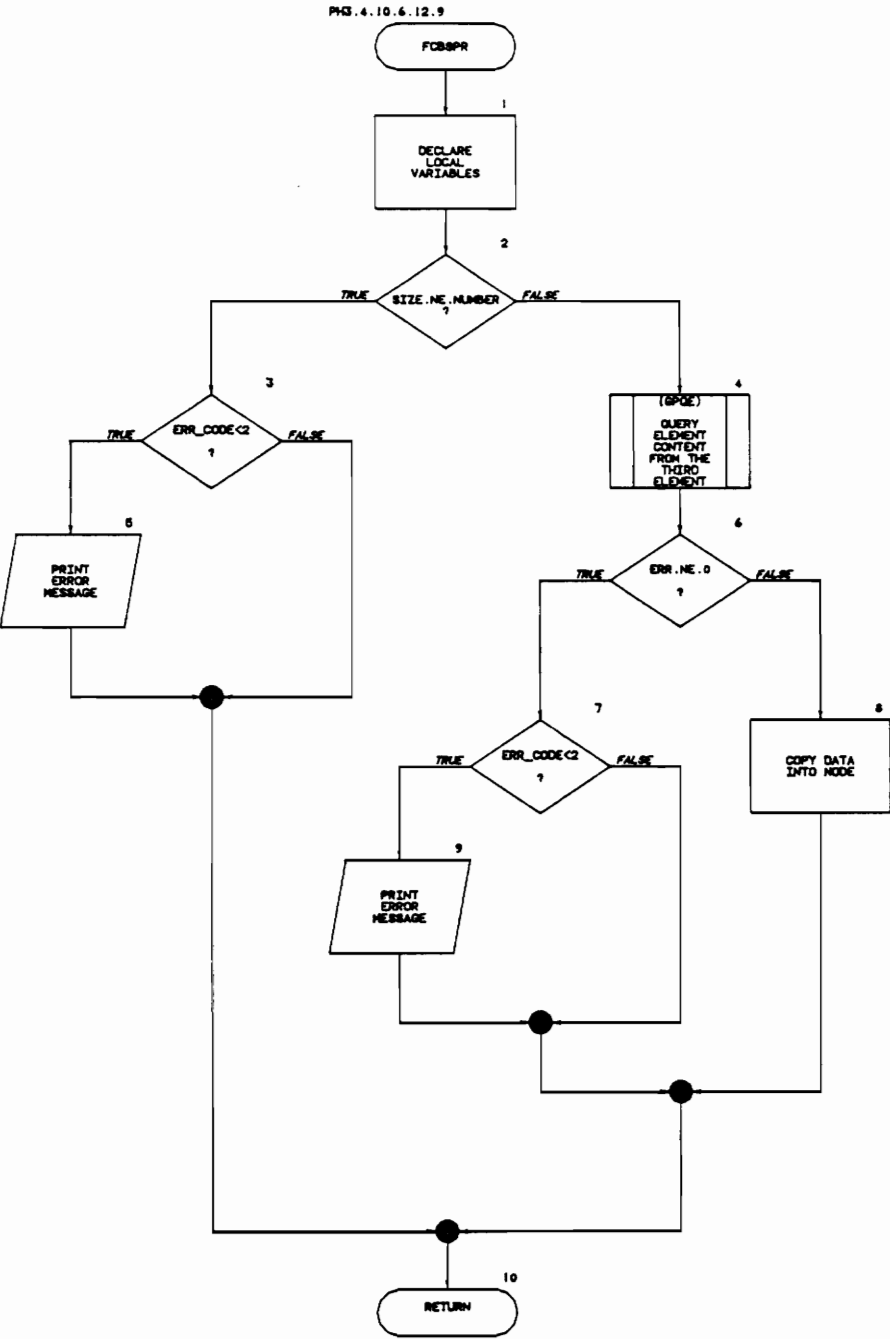
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCBICD	MODULE:PH3.4.10.6.12.7
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK INTERIOR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	3/29/90	



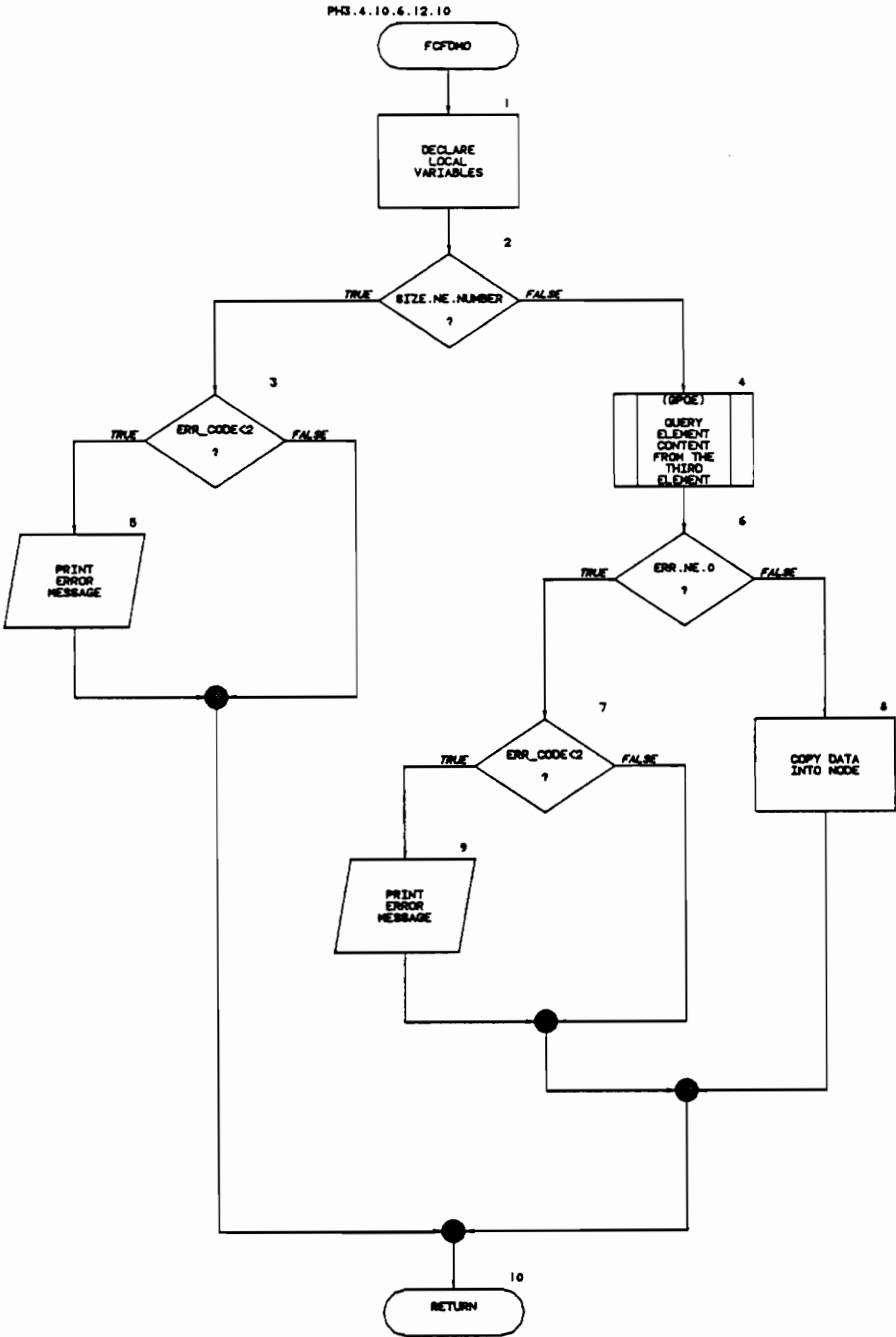
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCBSCD	MODULE:PH3.4.10.6.12.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES BACK SPECULAR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	3/29/90	



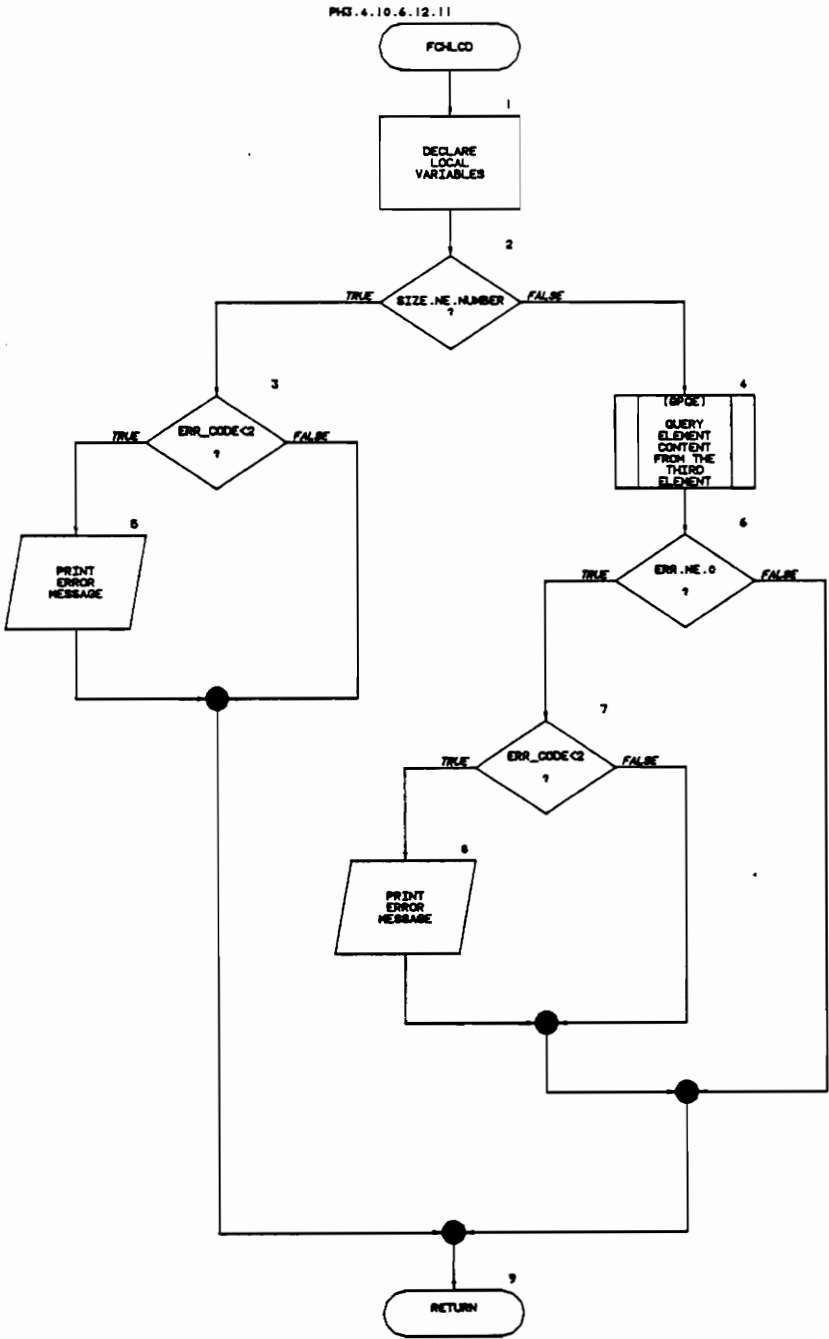
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: FCBSPR	MODULE: PH3.4.10.6.12.9
DESIGNED BY: KRISHNAN KOLADY	NOTE: RETRIEVES BACK SURFACE PROPERTIES FROM CURRENTLY OPEN STRUCTURE
DATE: 4/2/90	



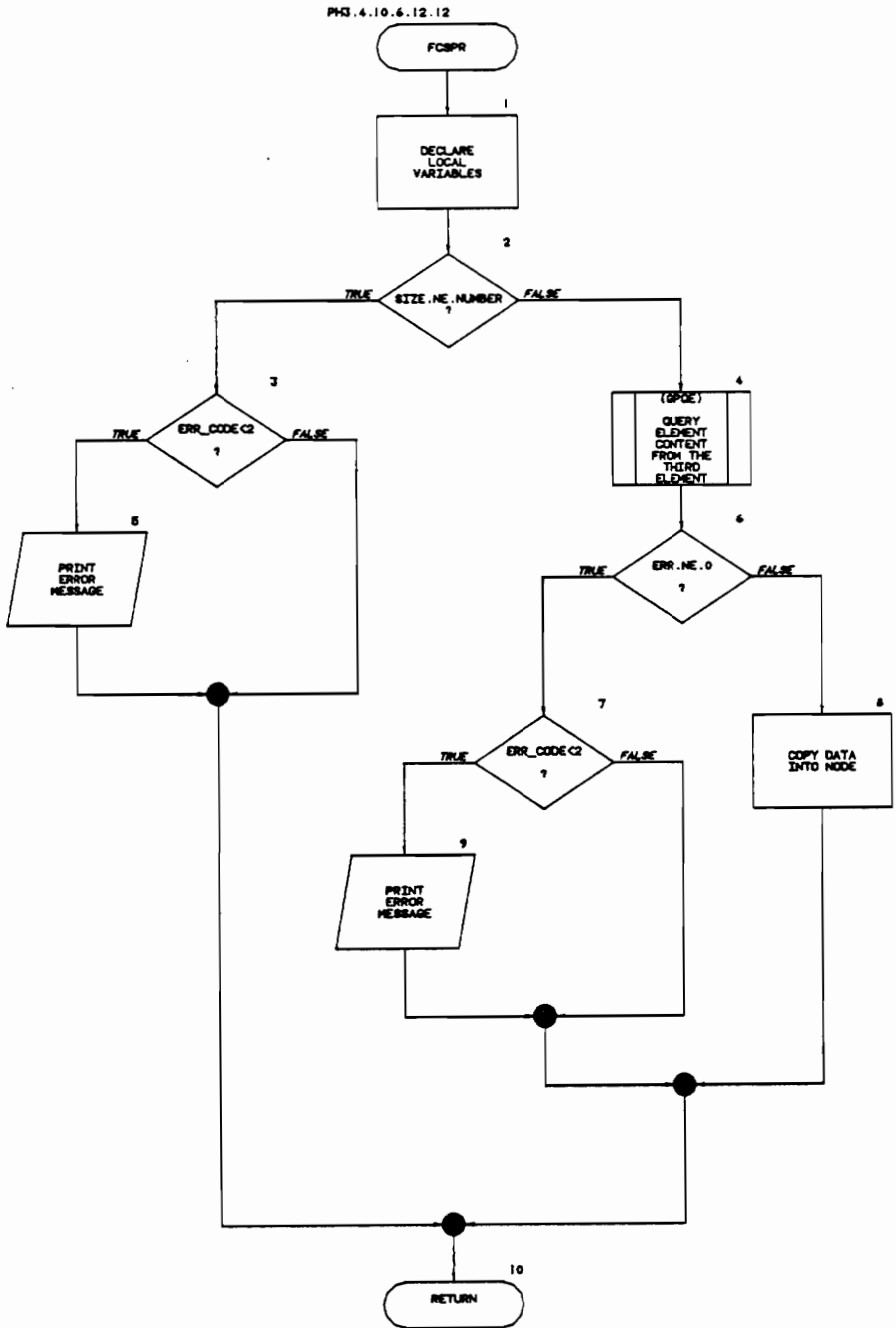
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: FCFDMD	MODULE: PH3.4.10.6.12.10
DESIGNED BY: KRISHNAN KOLADY	NOTE: RETRIEVES FACE DISTINGUISHING MODE FROM CURRENTLY OPEN STRUCTURE
DATE: 4/2/90	



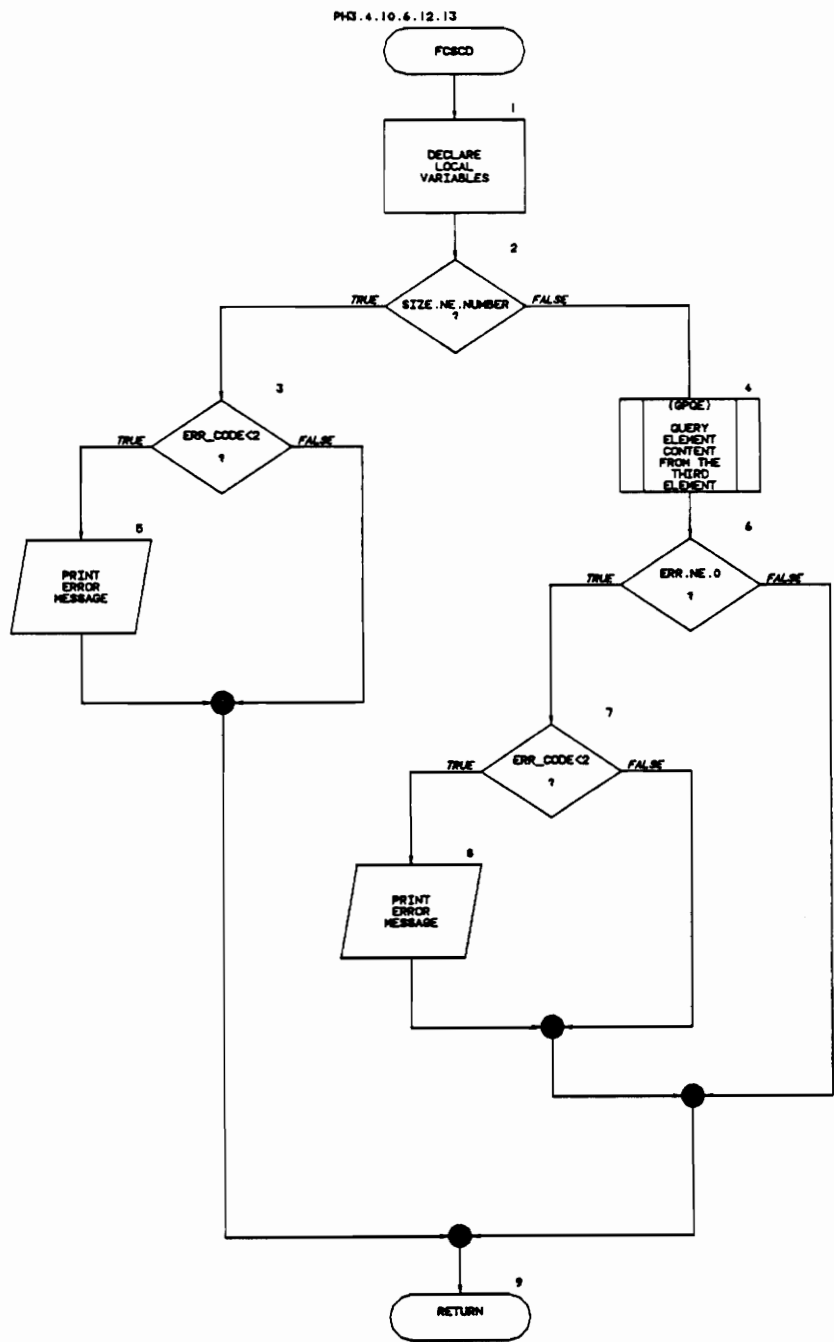
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCHLCD	MODULE:PH3.4.10.6.12.11
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES HIGHLIGHTING COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



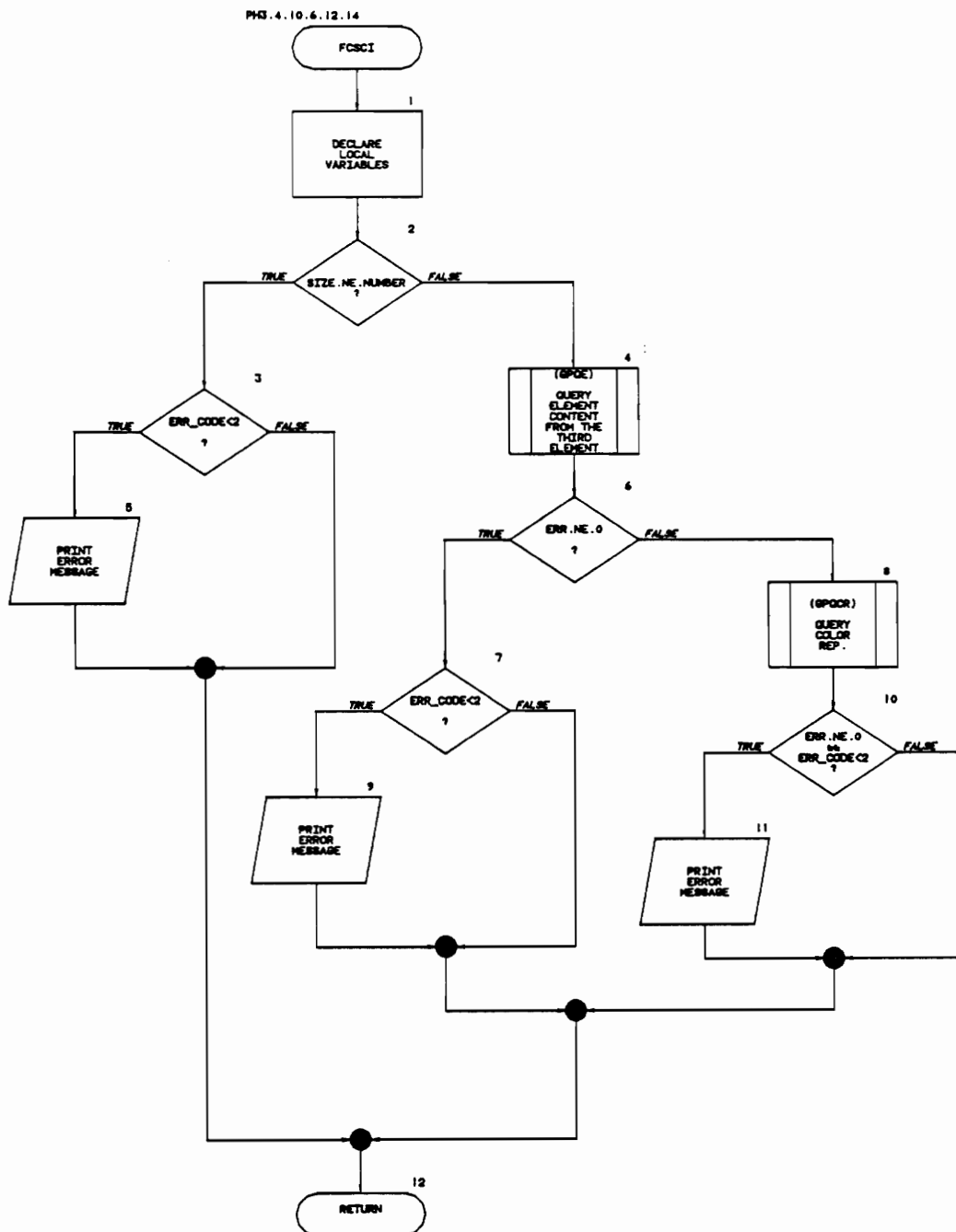
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: FCSPR	MODULE: PH3.4.10.6.12.12
DESIGNED BY: KRISHNAN KOLADY	NOTE: RETRIEVES SURFACE PROPERTIES FROM CURRENTLY OPEN STRUCTURE
DATE: 4/2/90	



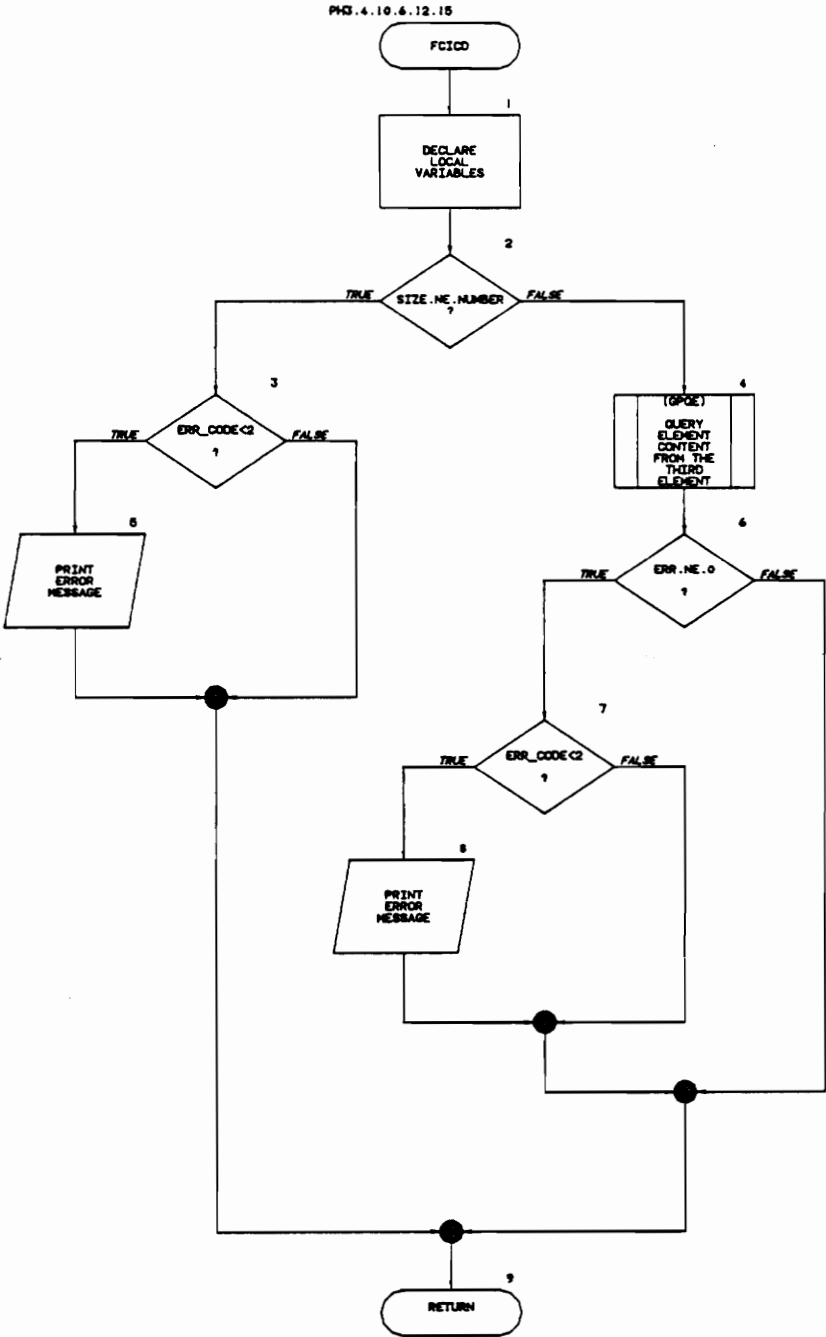
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCSCD	MODULE:PH3.4.10.6.12.13
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES SPECULAR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



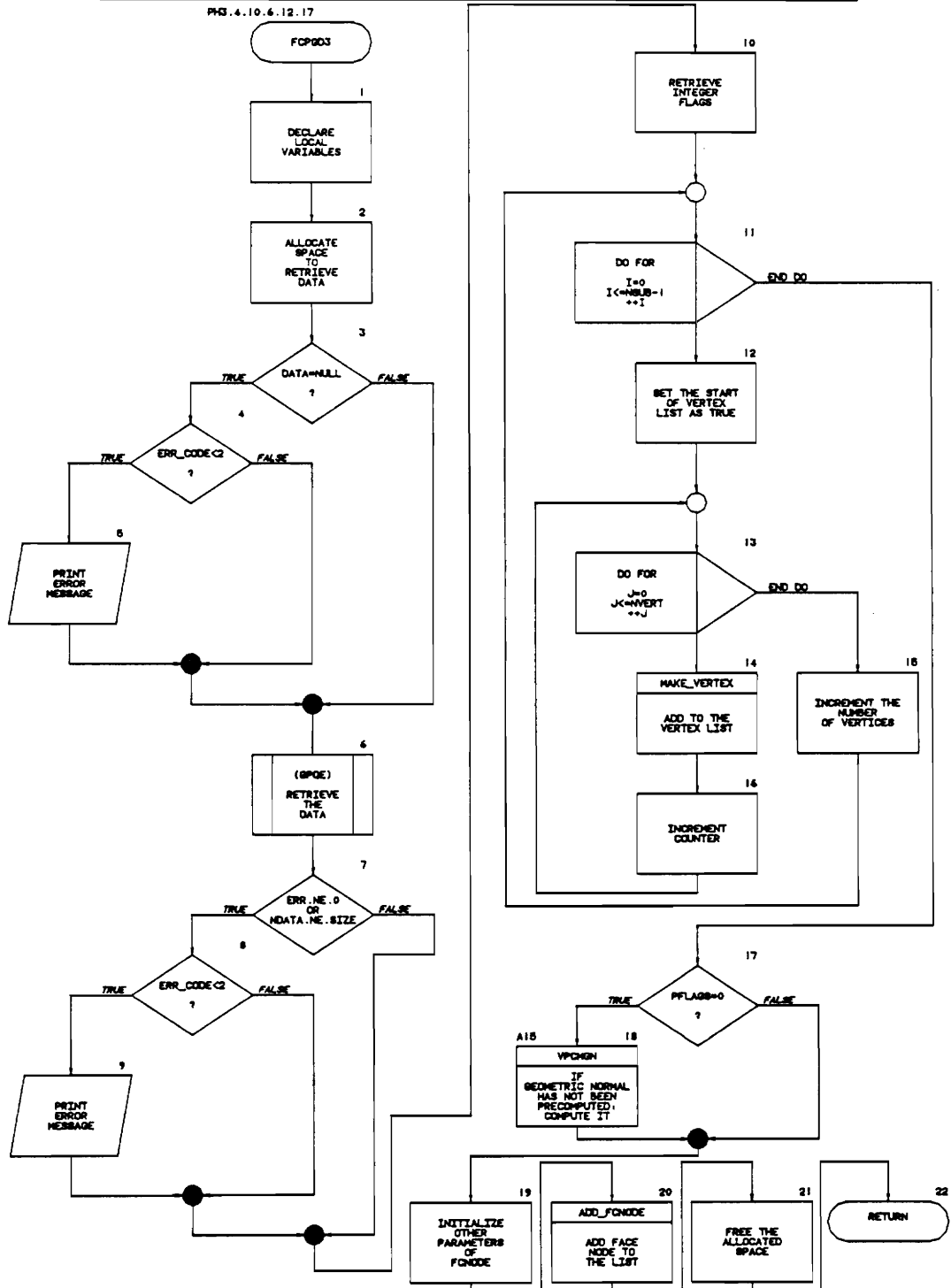
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCSCI	MODULE:PH3.4.10.6.12.14
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETRIEVES SPECULAR COLOR INDEX FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	



VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCICD	MODULE:PH3.4.10.6.12.15
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES INTERIOR COLOR DIRECT FROM CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	

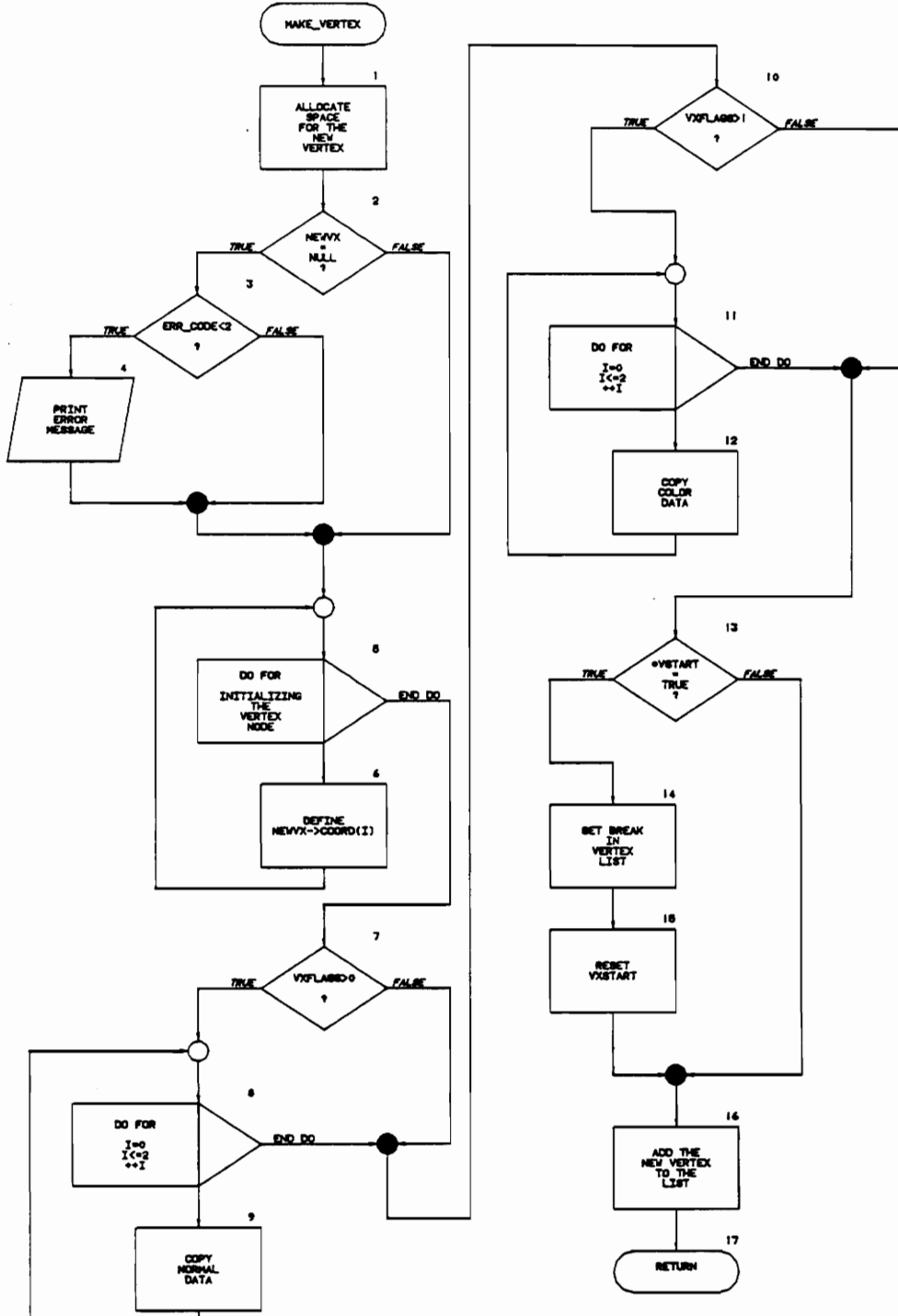


VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FCPGD3	MODULE:PH3.4.10.6.12.17
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETRIEVES POLYGON 3 WITH DATA INFORMATION FROM THE CURRENTLY OPEN STRUCTURE
DATE:	4/2/90	

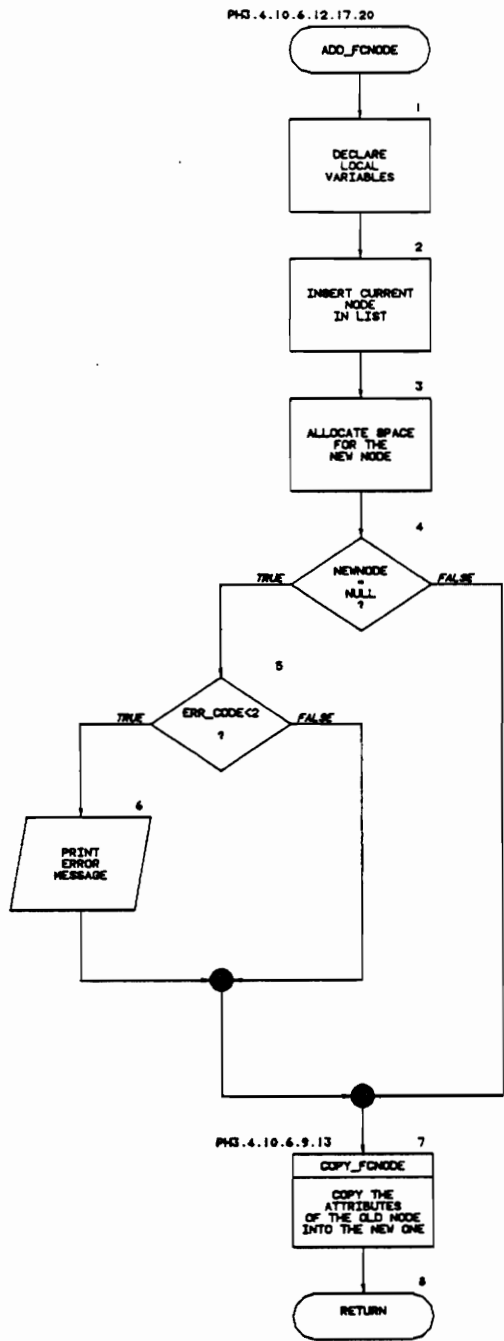


VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: MAKE_VERTEX	MODULE: PH3.4.10.6.12.17.14
DESIGNED BY: KRISHNAN KOLADY	NOTE: ADDS THE INPUT DATA AS A VERTEX TO THE VERTEX LIST
DATE: 5/24/90	

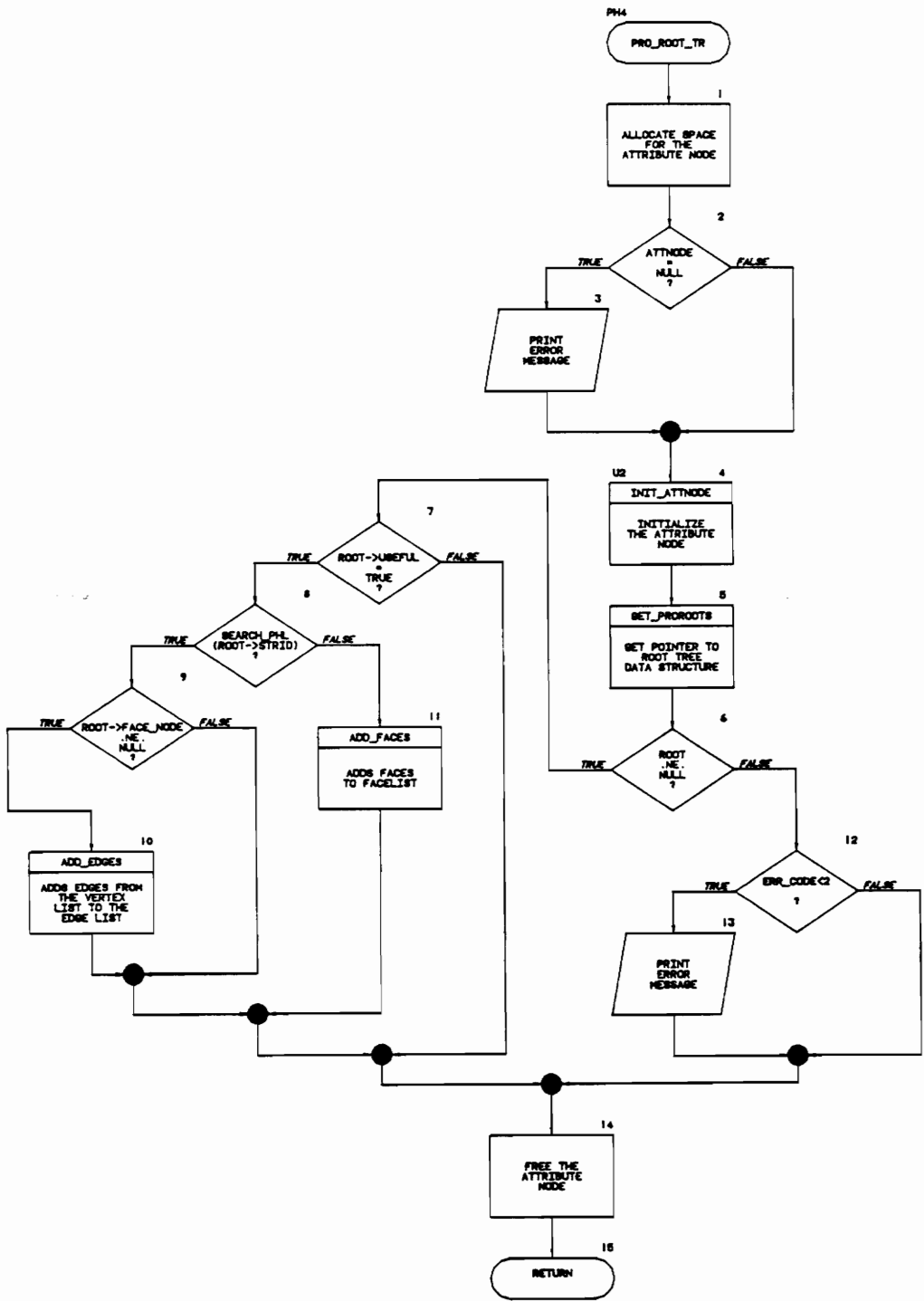
PH3.4.10.6.12.17.14




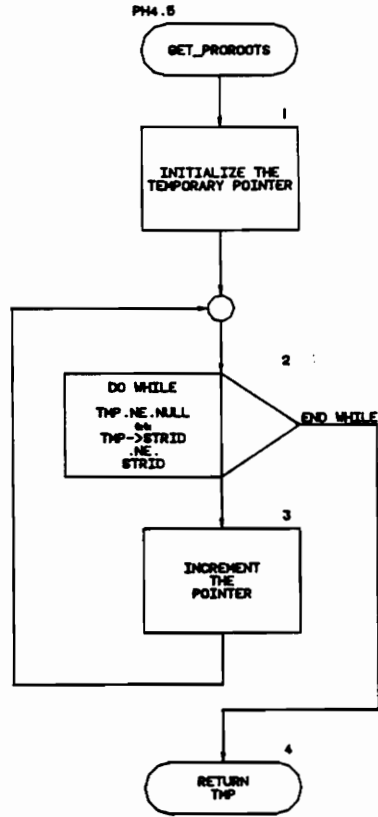
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	ADD_FCNODE	MODULE:PH3.4.10.6.12.17.20
DESIGNED BY:	KRISHNAN KOLADY	NOTE: ADDS THE CURRENT FACE NODE TO THE LIST
DATE:	3/29/90	



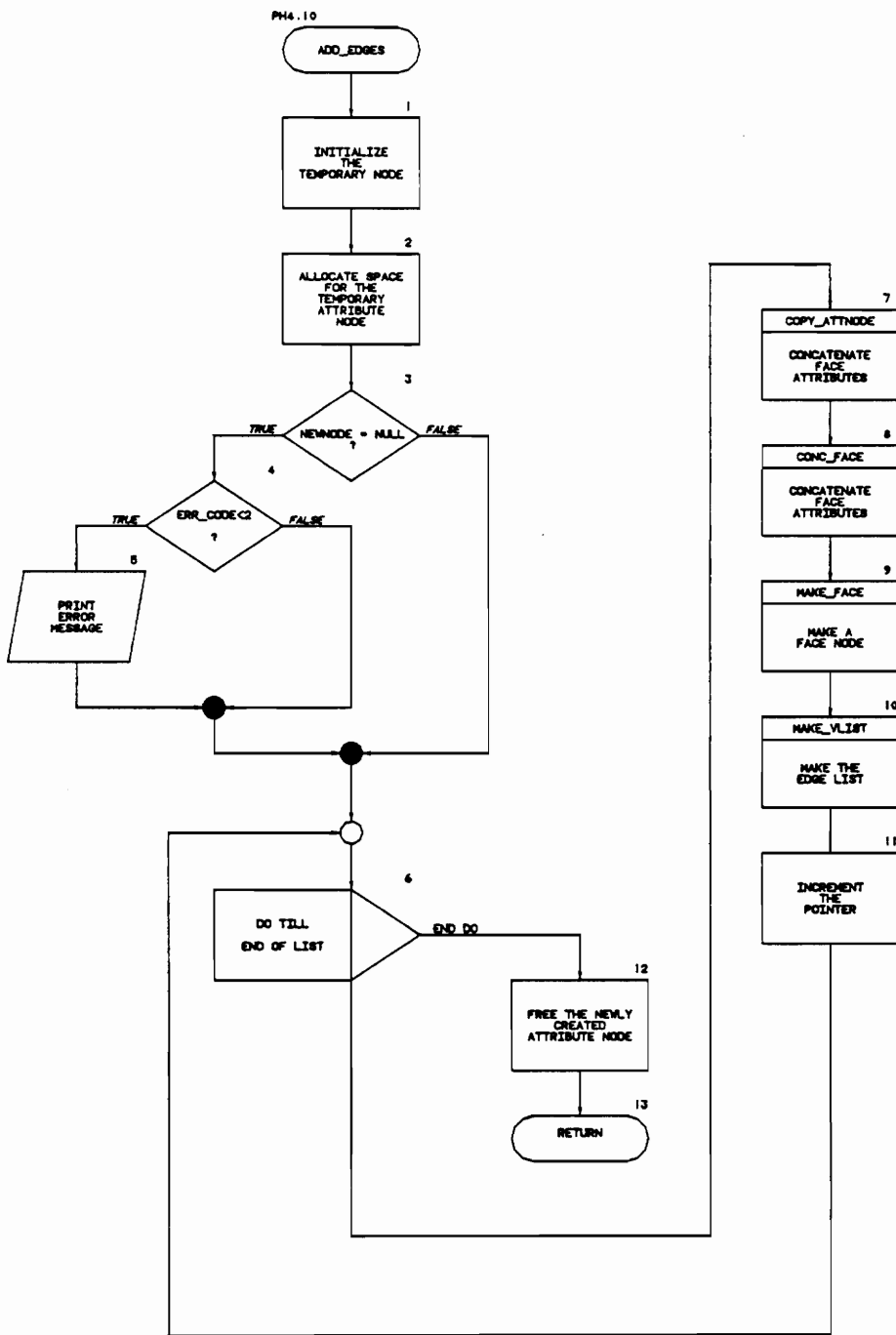
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PRO_ROOT_TR	MODULE: PH4
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES THE ROOT DATA STRUCTURE AND ADDS ALL THE EDGES TO THE EDGE LIST
DATE:	5/30/90	



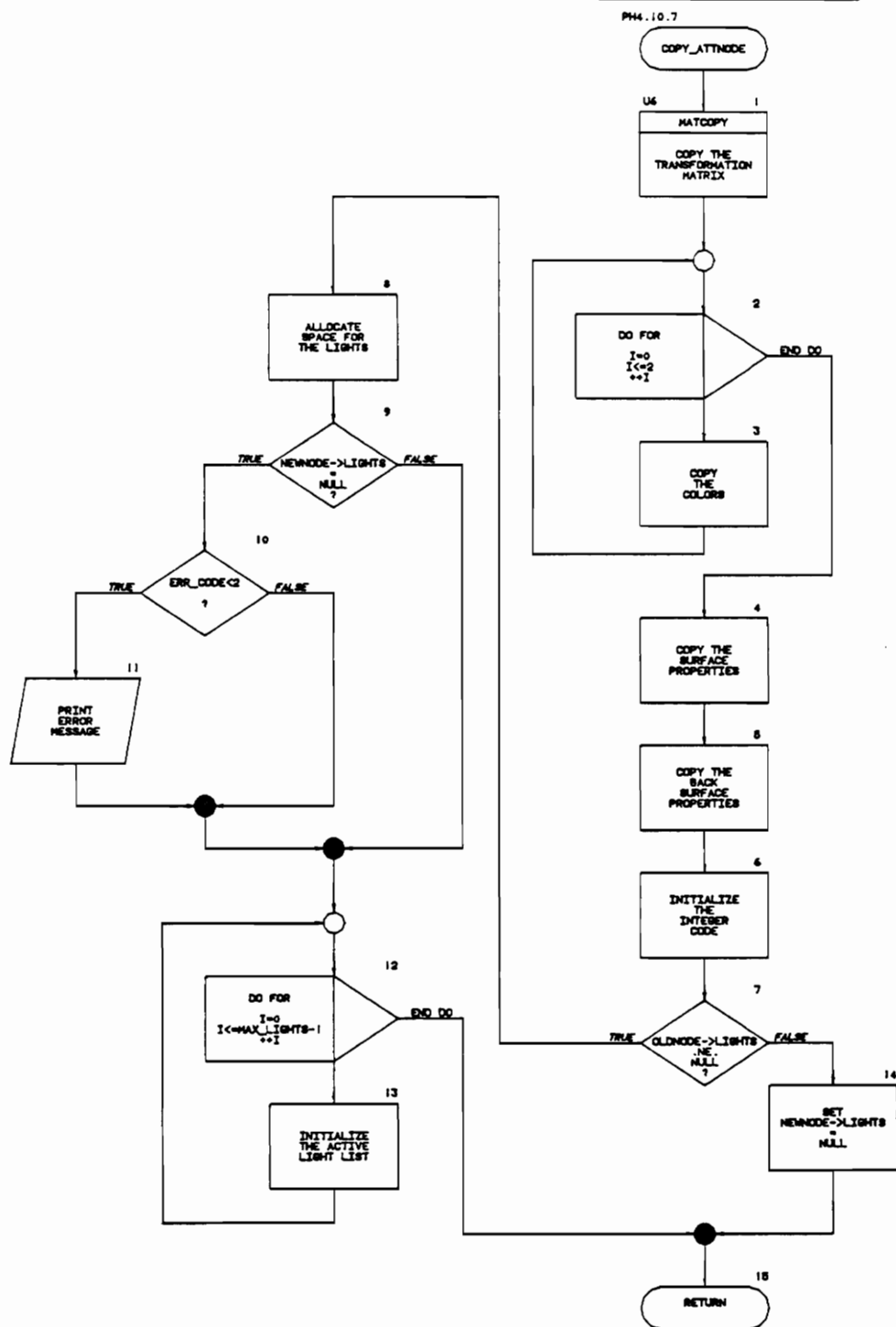
<div>  PROGRAM SPECIFICATION - PHONG SHADING </div>		
MODULE NAME:	GET_PROROOTS	MODULE: PH4.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETURNS THE POINTER TO THE ROOT TREE STRUCTURE FOR THE STRUCTURE ID PASSED IN
DATE:	5/30/90	



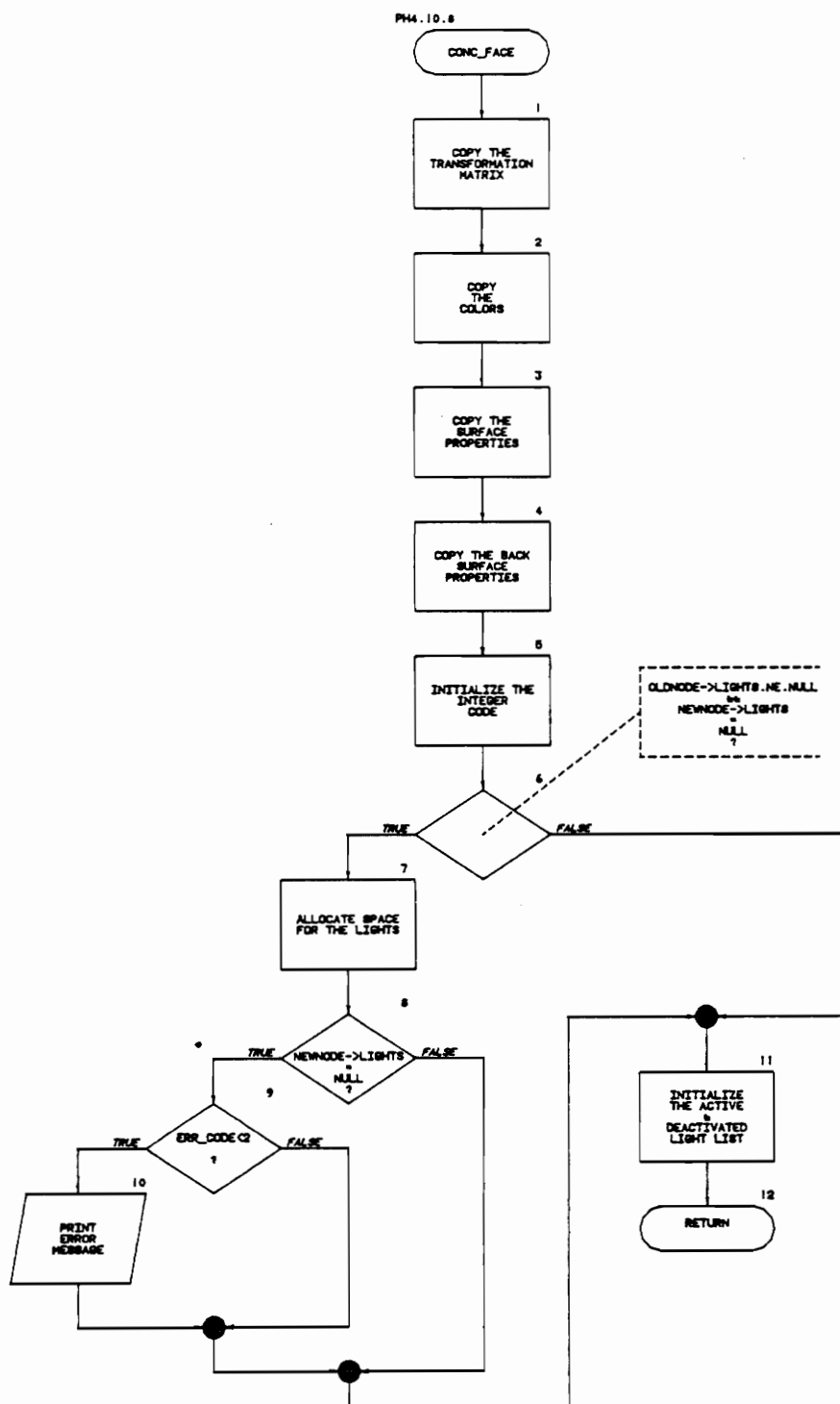
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	ADD_EDGES	MODULE:PH4.10
DESIGNED BY:	KRISHNAN KOLADY	NOTE:ADDS EDGES FROM THE VERTEX LIST TO THE EDGE LIST
DATE:	5/30/90	



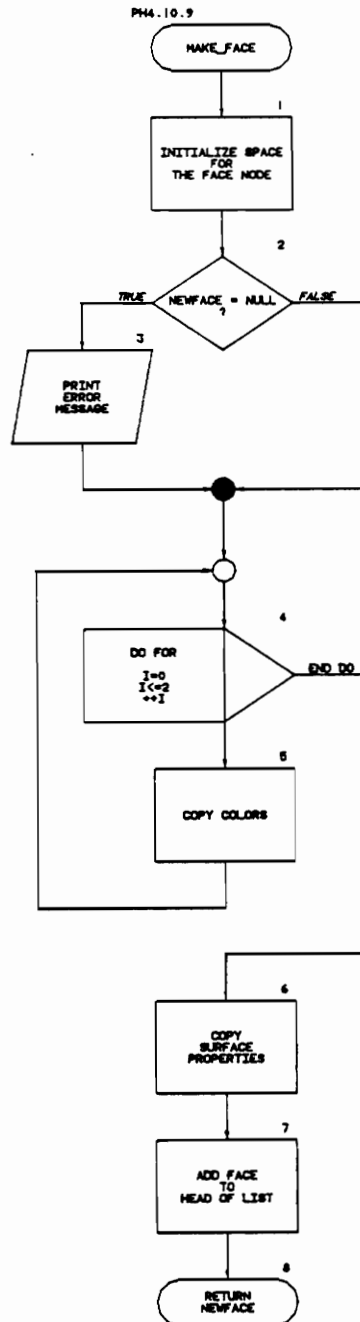
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: COPY_ATTNODE	MODULE: PH4.10.7
DESIGNED BY: KRISHNAN KOLADY	NOTE: COPIES THE OLD NODE DATA INTO A NEW ATTRIBUTE NODE
DATE: 5/30/90	



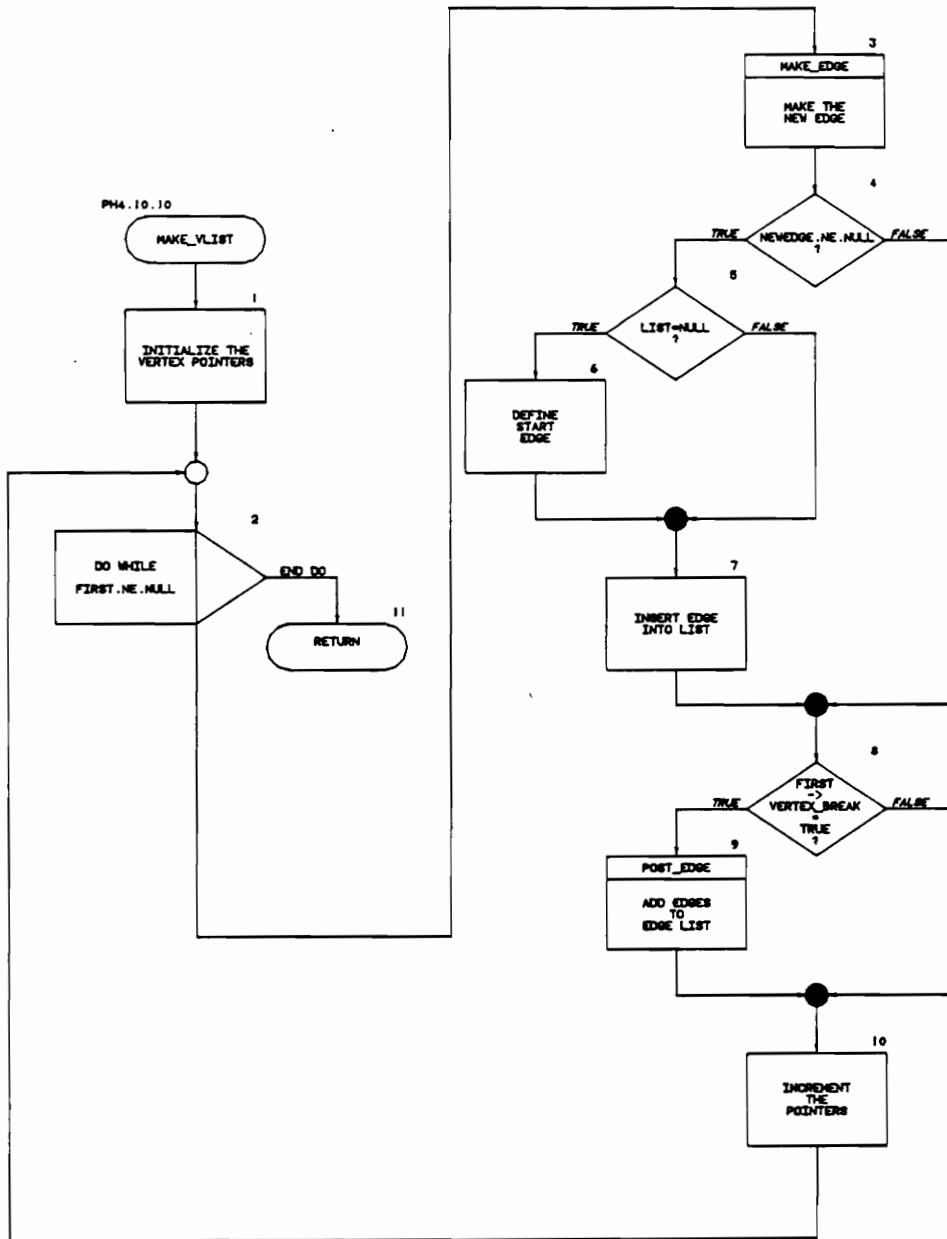
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	CONC_FACE	MODULE:PH4.10.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CONCATENATES THE ATTRIBUTES OF THE FACE NODES
DATE:	5/30/90	



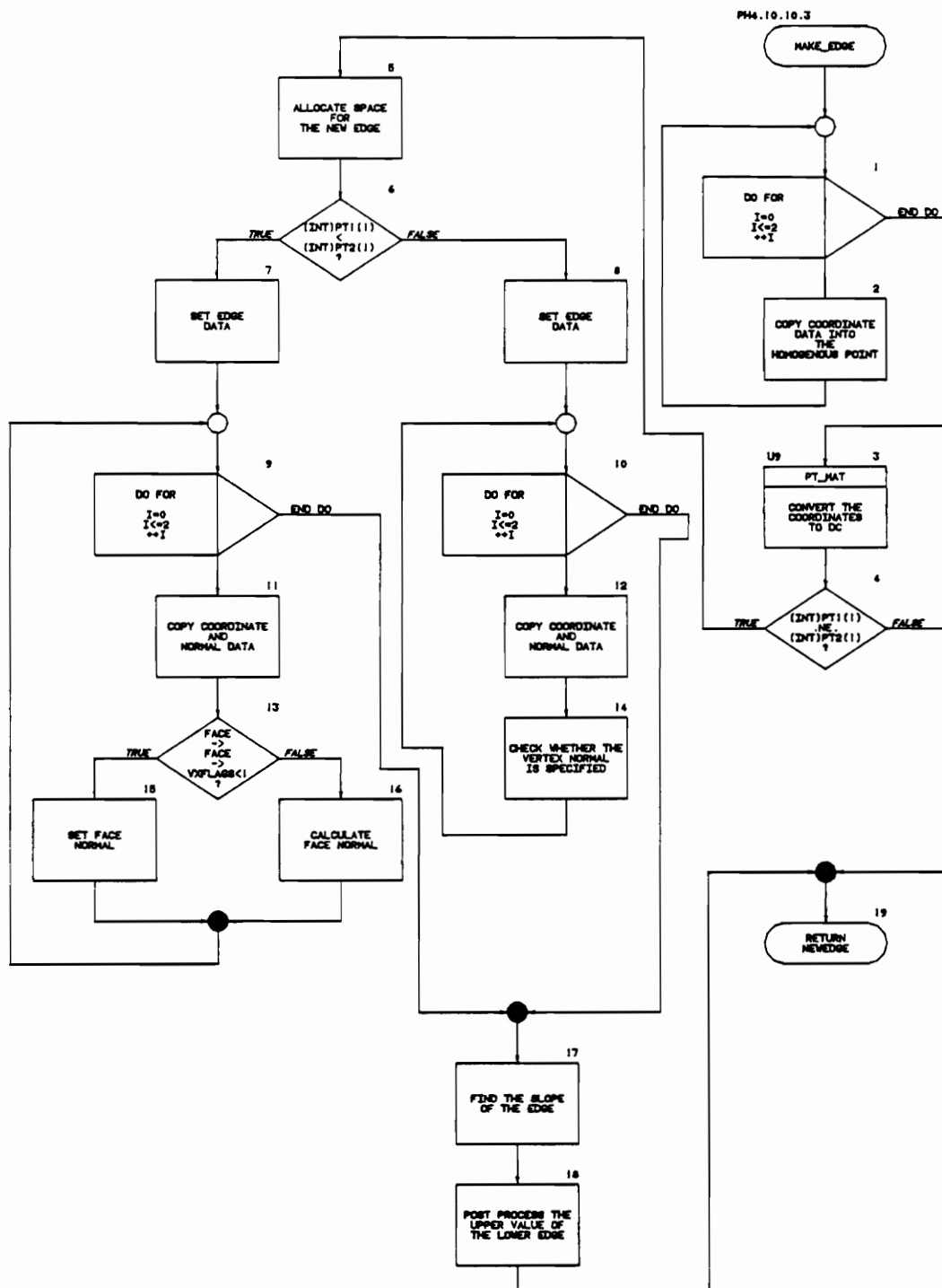
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	MAKE_FACE	MODULE:PH4.10.9
DESIGNED BY:	KRISHNAN KOLADY	NOTE: MAKES A NEW FACE AND ADDS TO THE FACE LIST TO BE PROCESSED
DATE:	5/30/90	



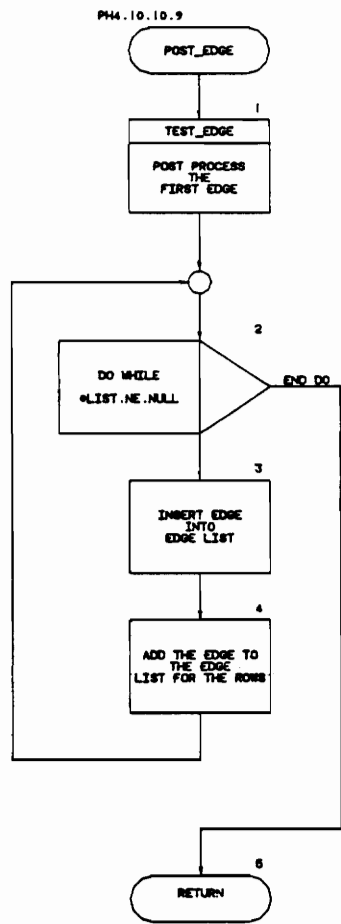
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	MAKE_VLIST	MODULE: PH4.10.10
DESIGNED BY:	KRISHNAN KOLADY	NOTE: MAKES A EDGE LIST FROM THE VERTEX LIST
DATE:	5/30/90	



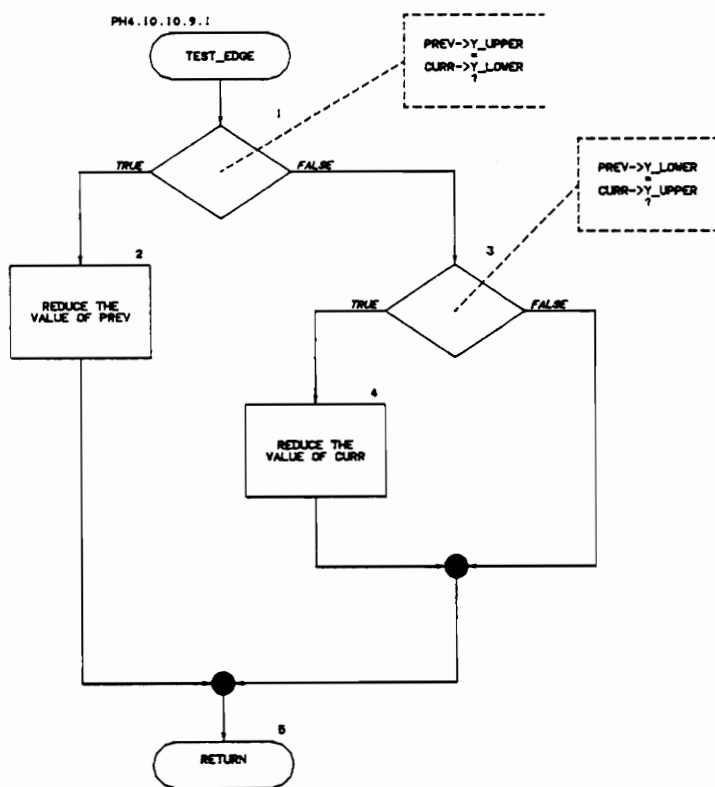
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	MAKE_EDGE	MODULE: PH4.10.10.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: MAKES AN EDGE FROM 2 VERTEX POINTERS
DATE:	5/30/90	



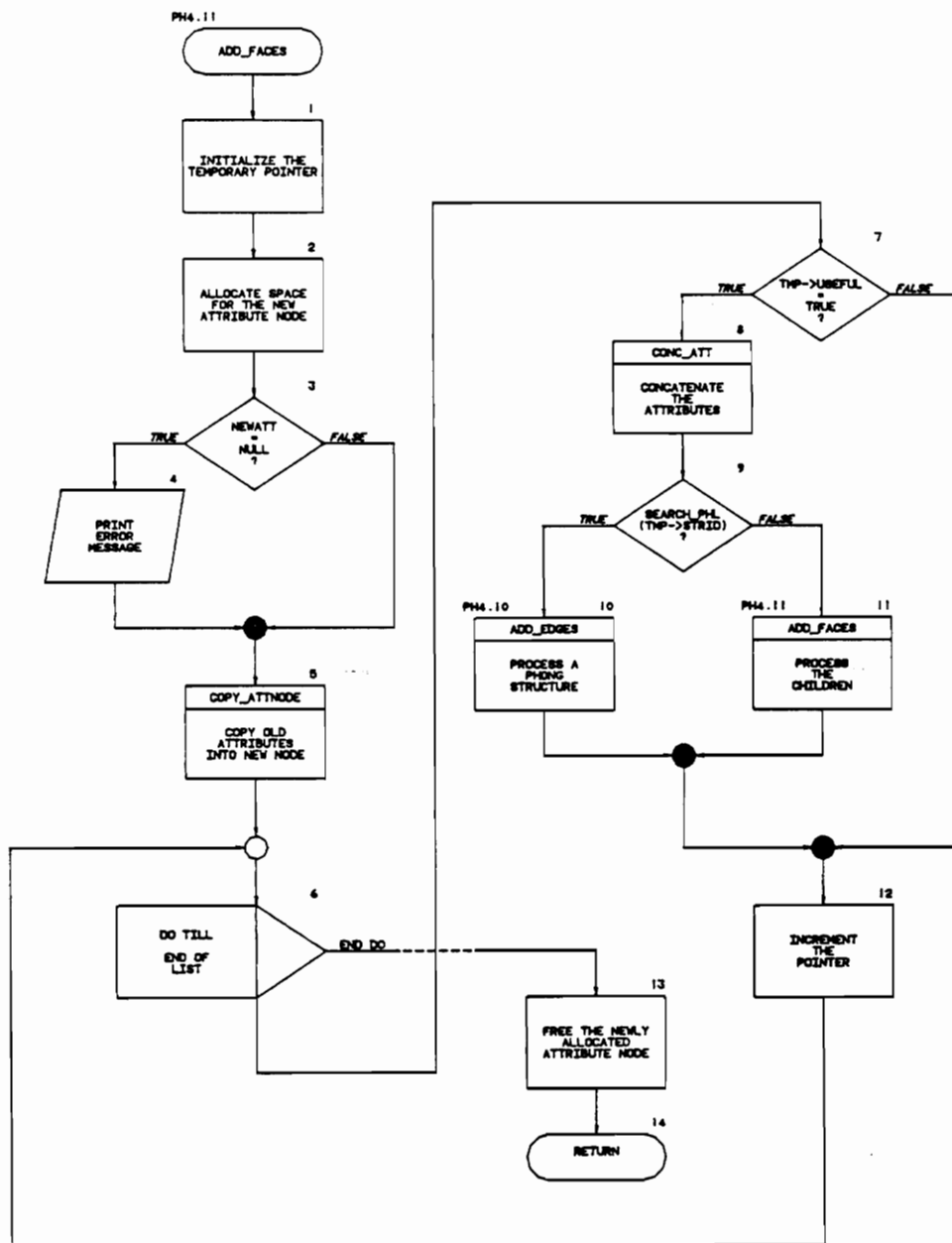
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	POST_EDGE	MODULE: PH4.10.10.9
DESIGNED BY:	KRISHNAN KOLADY	NOTE: POST PROCESS THE FIRST EDGE AND ADD ALL THE EDGES TO THE EDGE LIST
DATE:	5/30/90	



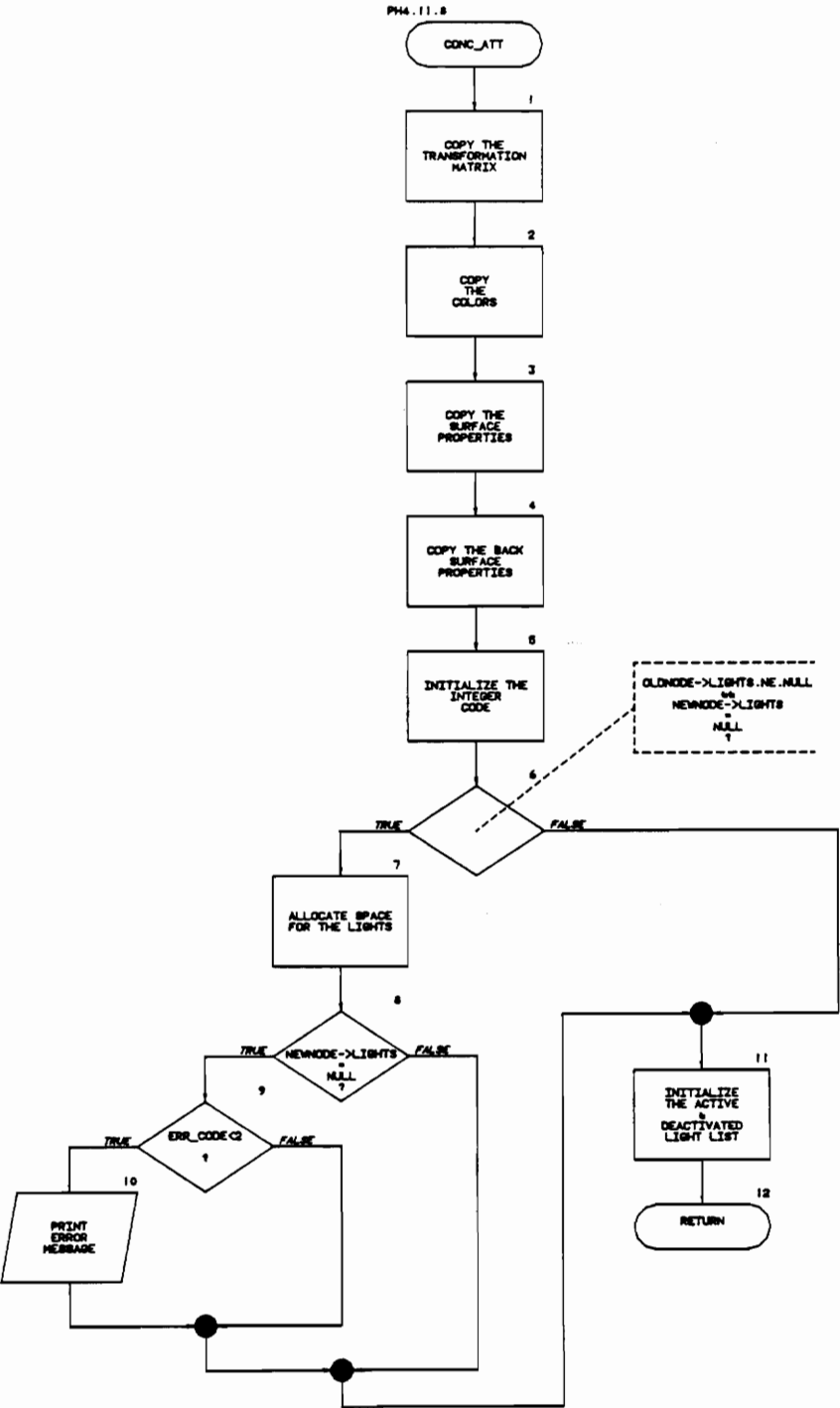
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: TEST_EDGE	MODULE: PH4.10.10.9.1
DESIGNED BY: KRISHNAN KOLADY	NOTE: TESTS THE PREVIOUS AND CURRENT EDGES FOR REDUCING THE VALUE OF THE Y_UPPER OF THE LOWER EDGE
DATE: 5/30/90	



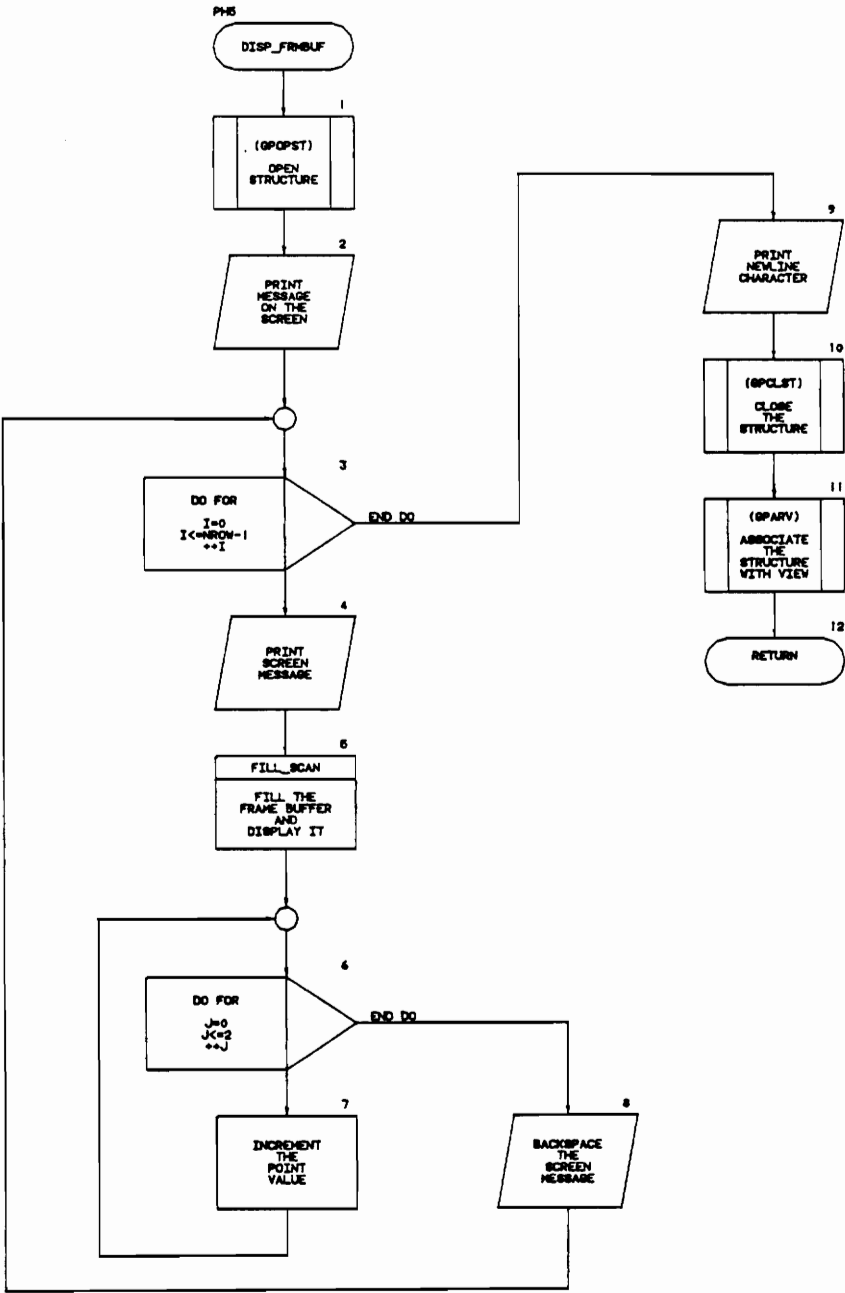
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	ADD_FACES	MODULE: PH4.11
DESIGNED BY:	KRISHNAN KOLADY	NOTE: PROCESSES THE ROOT DATA STRUCTURE TO CONCATENATE ATTRIBUTES AND COLLECT FACES AND EDGES
DATE:	5/30/90	



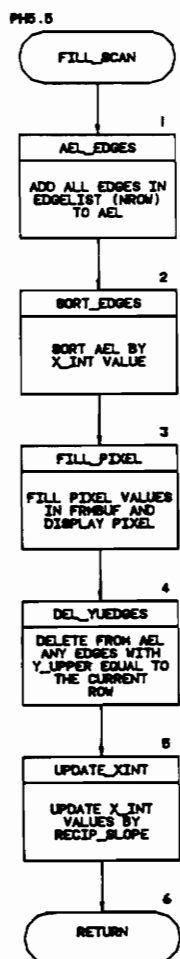
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	CONC_ATT	MODULE:PH4.11.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE:CONCATENATES THE ATTRIBUTES OF THE NODES
DATE:	5/30/90	



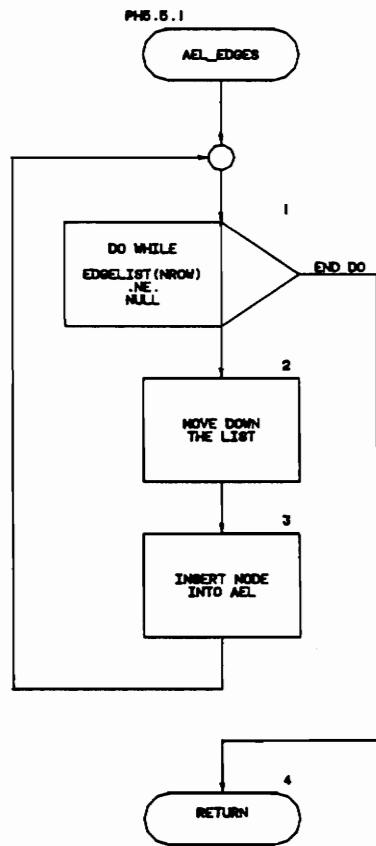
VT		PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: DISP_FRMBUF		MODULE: PH5	
DESIGNED BY: KRISHNAN KOLADY		NOTE: DISPLAYS THE VIRTUAL FRAME BUFFER SPACE ON THE VIEWPORT	
DATE: 5/24/90			



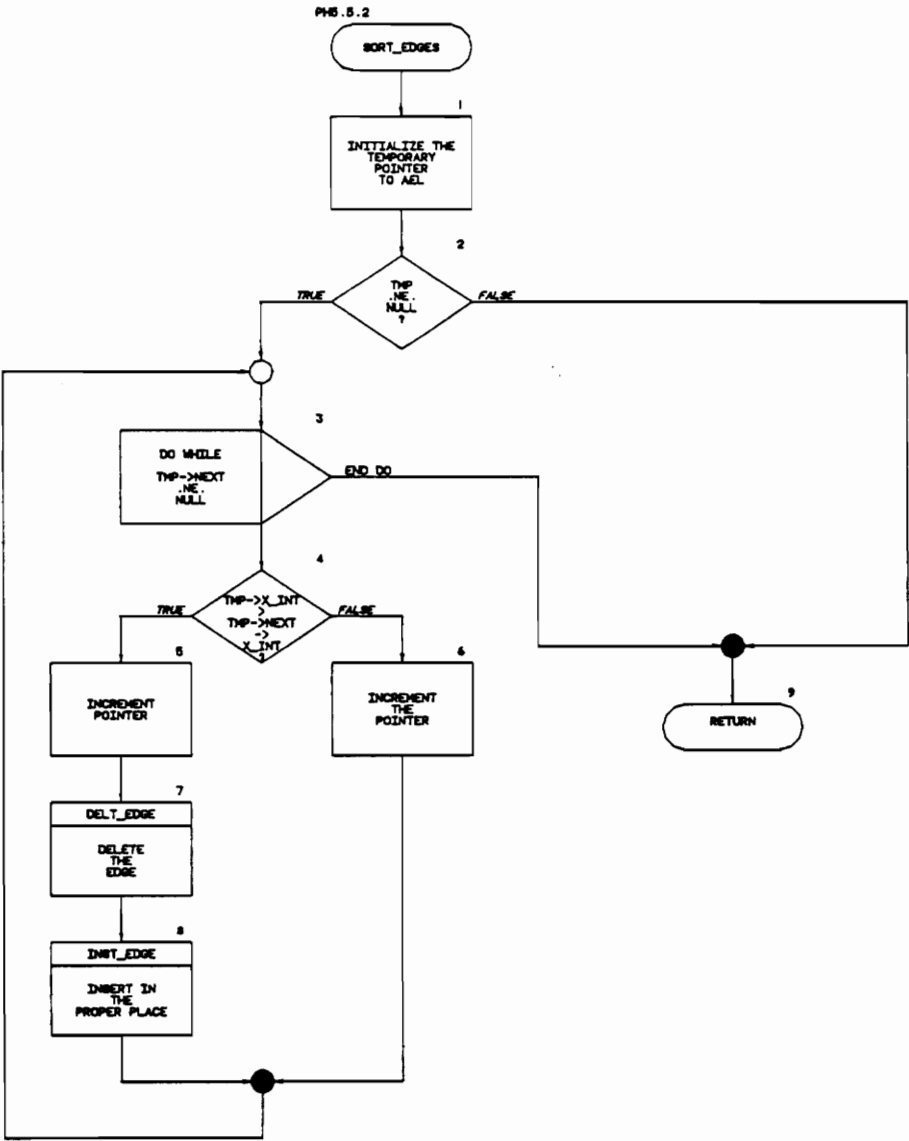
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FILL_SCAN	MODULE:PH5.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CALCULATES THE INTERSECTIONS OF A SCAN LINE WITH THE EDGES AND FILLS THE SCANLINE WITH THE COLOR INDICES
DATE:	5/30/90	



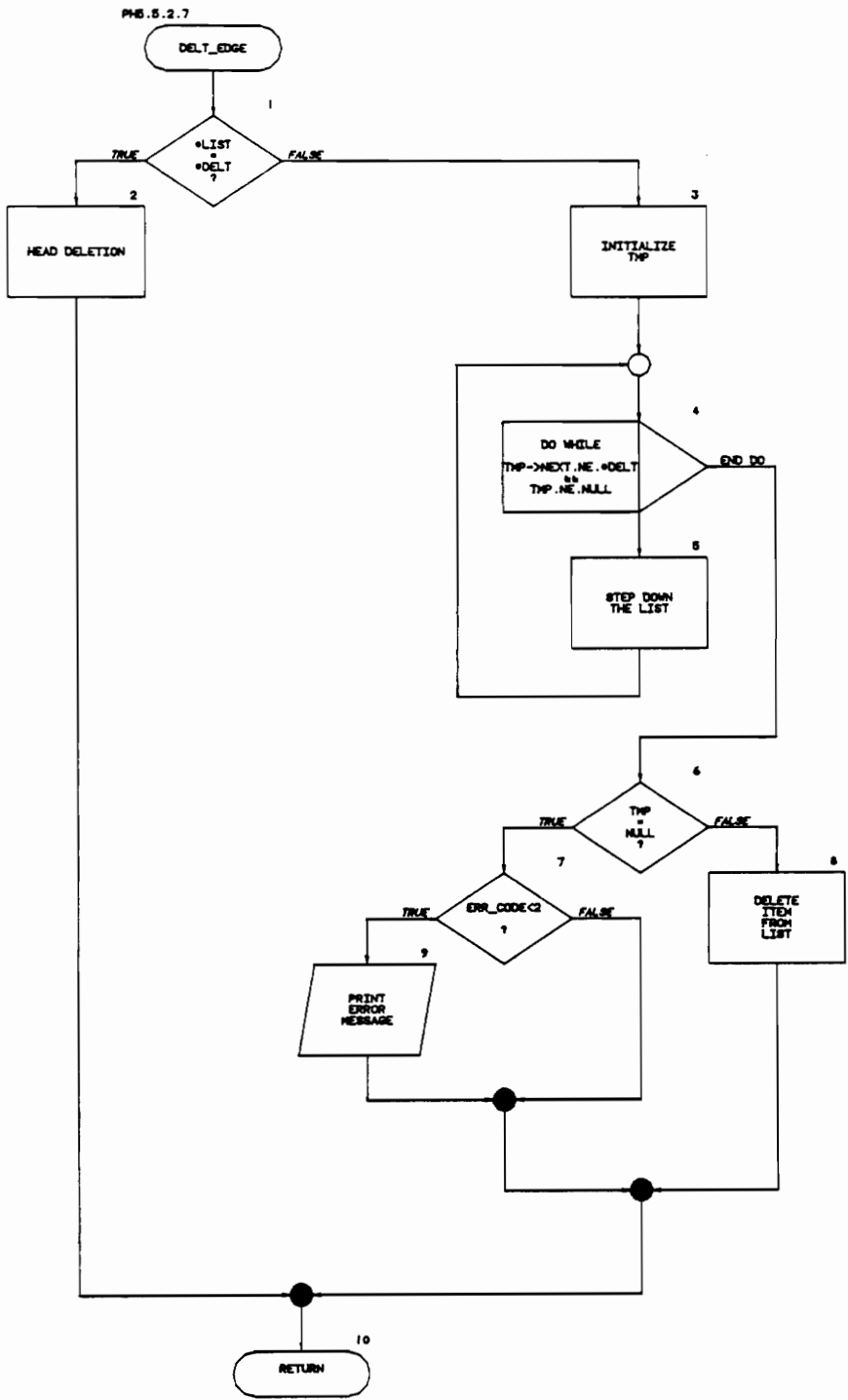
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	AEL_EDGES	MODULE: PH5.5.1
DESIGNED BY:	KRISHNAN KOLADY	NOTE: ADDS EDGES FROM THE EDGE LIST FOR THE CURRENT ROW TO THE ACTIVE EDGE LIST
DATE:	5/30/90	



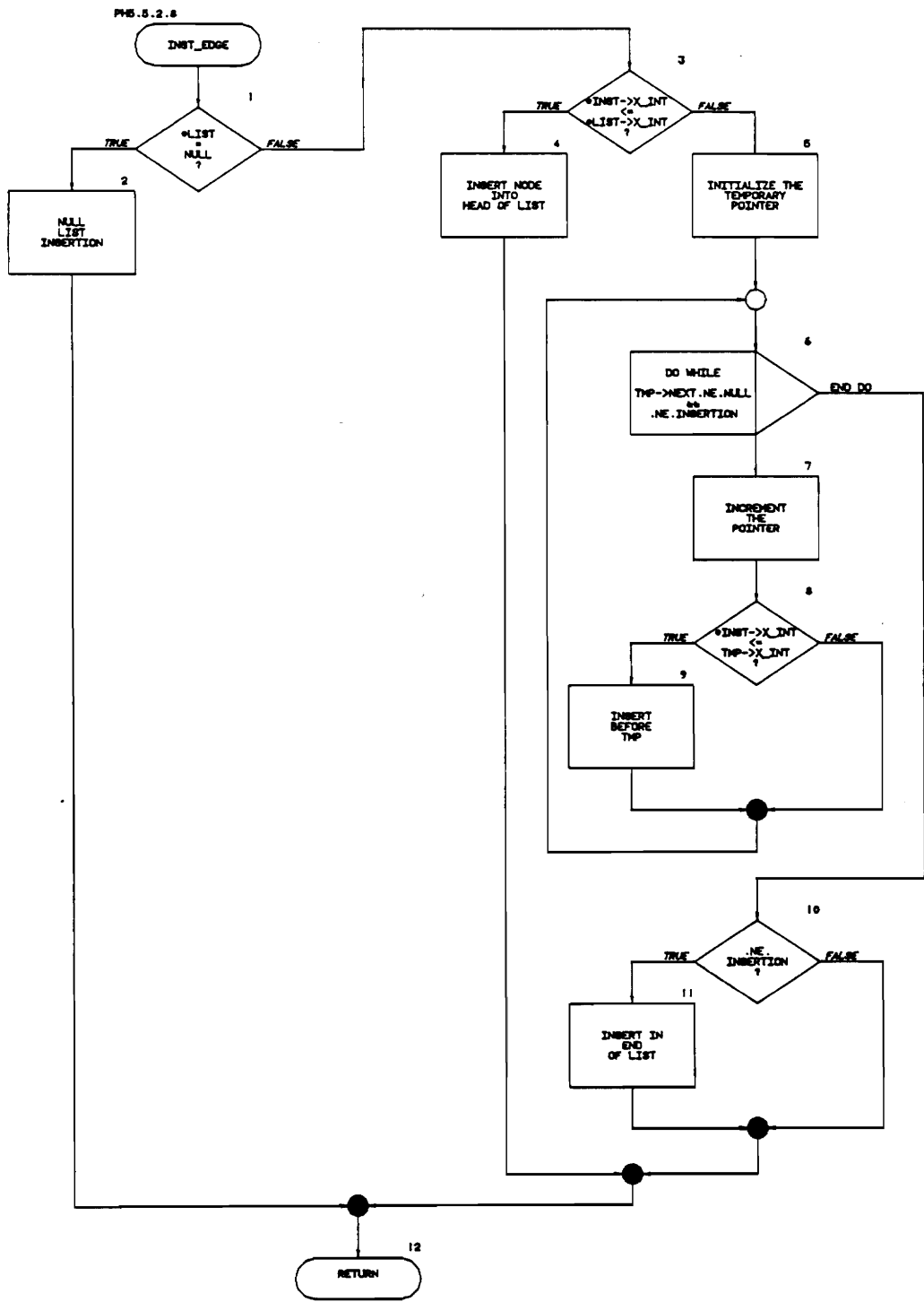
VT			PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	SORT_EDGES	MODULE:	PH5.5.2
DESIGNED BY:	KRISHNAN KOLADY	NOTE:	SORTS THE EDGES IN THE ACTIVE EDGE LIST
DATE:	5/30/90		



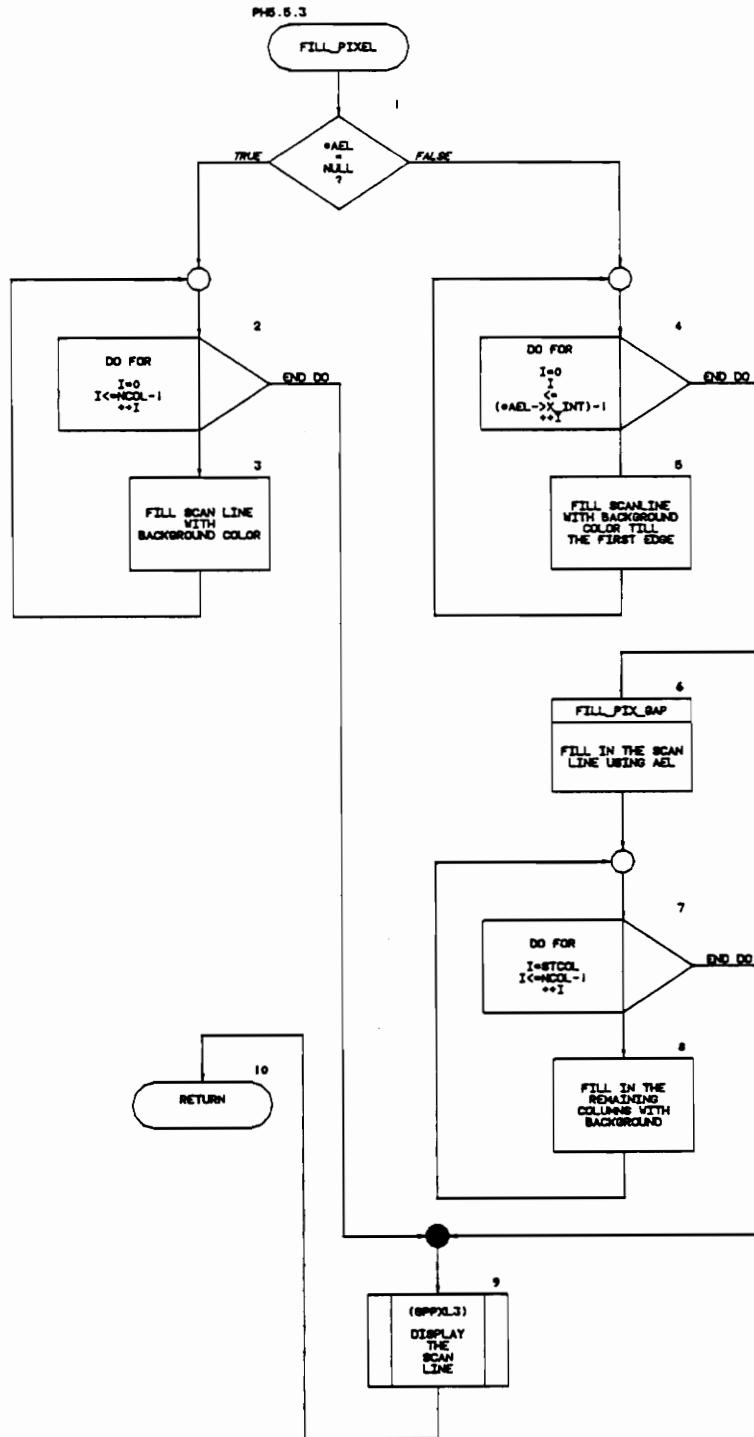
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	DELT_EDGE	MODULE:PH5.5.2.7
DESIGNED BY:	KRISHNAN KOLADY	NOTE:DELETES THE EDGE FROM THE LIST
DATE:	5/30/90	



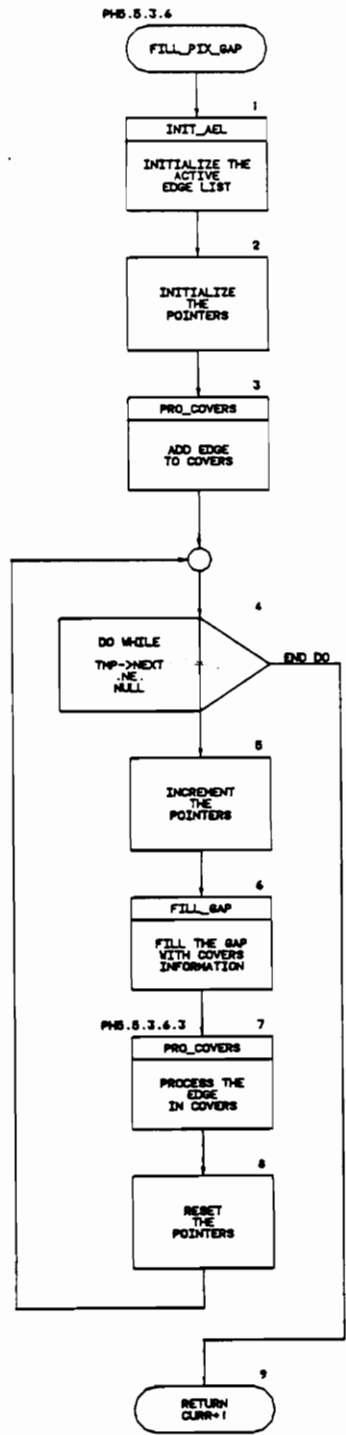
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	INST_EDGE	MODULE: PH5.5.2.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE: INSERTS AN EDGE INTO THE LIST IN THE CORRECT POSITION
DATE:	5/30/90	



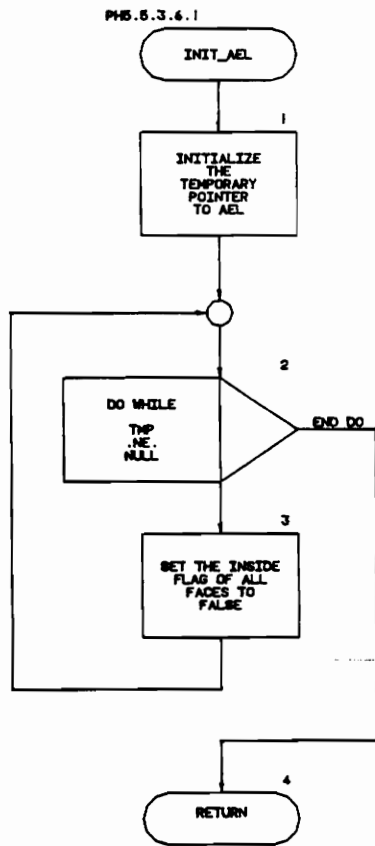
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: FILL_PIXEL	MODULE: PH5.5.3
DESIGNED BY: KRISHNAN KOLADY	NOTE: CALCULATES THE INTENSITY PER PIXEL OF THE SCAN LINE AND DRAWS THE SCAN LINE
DATE: 6/1/90	



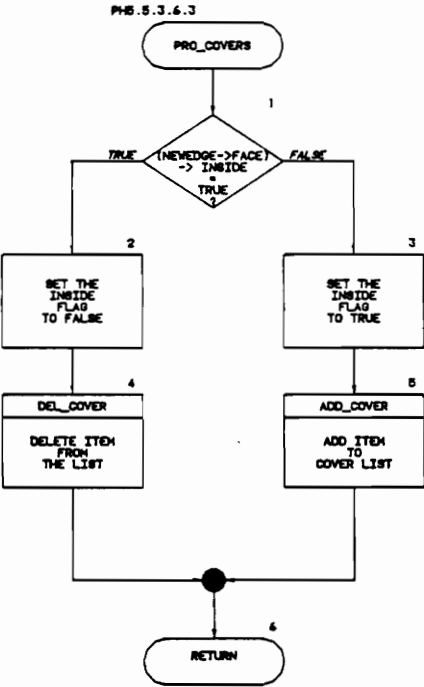
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FILL_PIX_GAP	MODULE:PH5.5.3.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CALCULATES THE INTENSITY PER PIXEL OF THE SCAN LINE BETWEEN EDGES
DATE:	6/1/90	



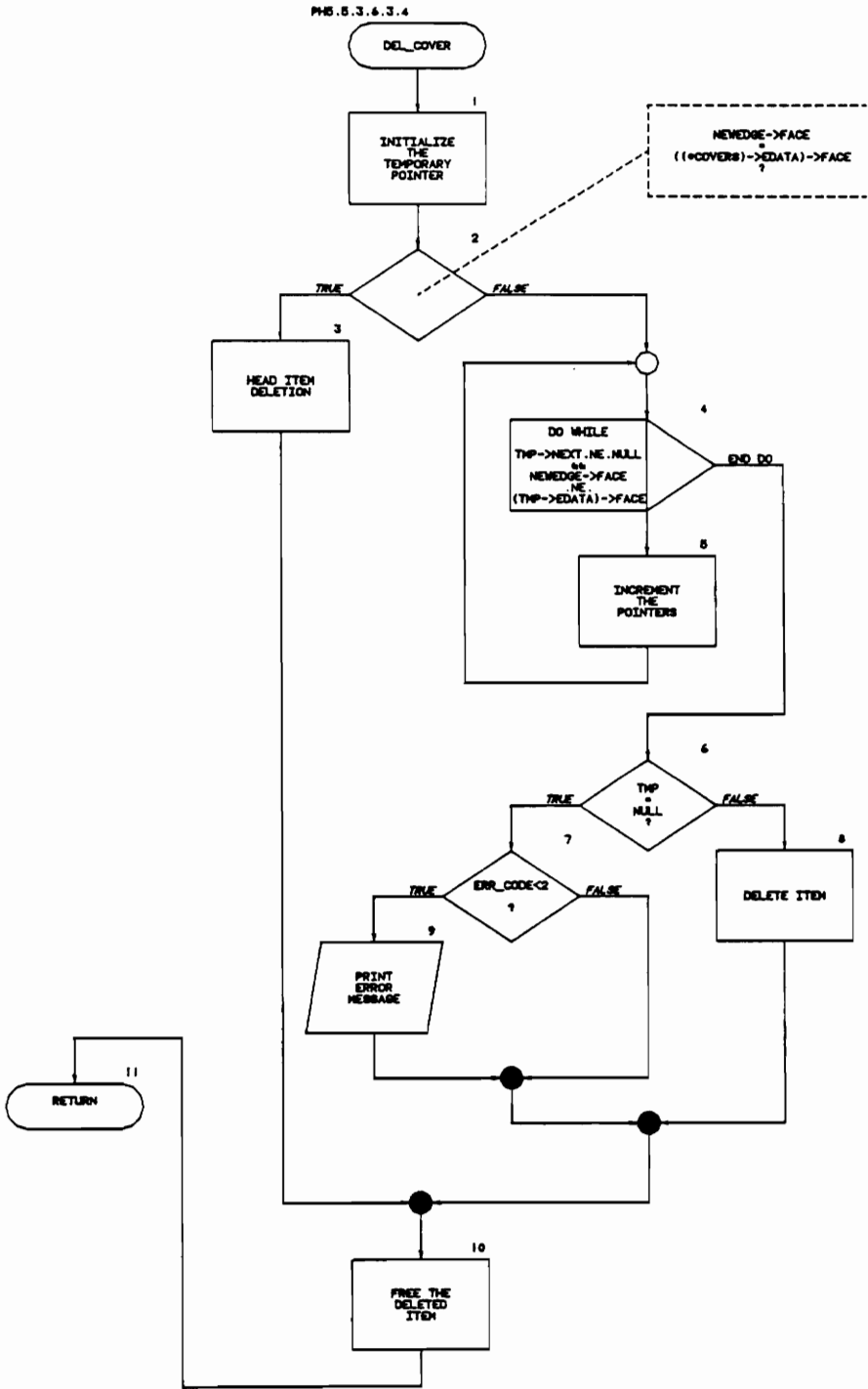
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: INIT_AEL	MODULE: PH5.5.3.6.1
DESIGNED BY: KRISHNAN KOLADY	NOTE: INITIALIZES THE ACTIVE EDGE LIST BEFORE EACH SCAN LINE PROCESSING
DATE: 6/1/90	



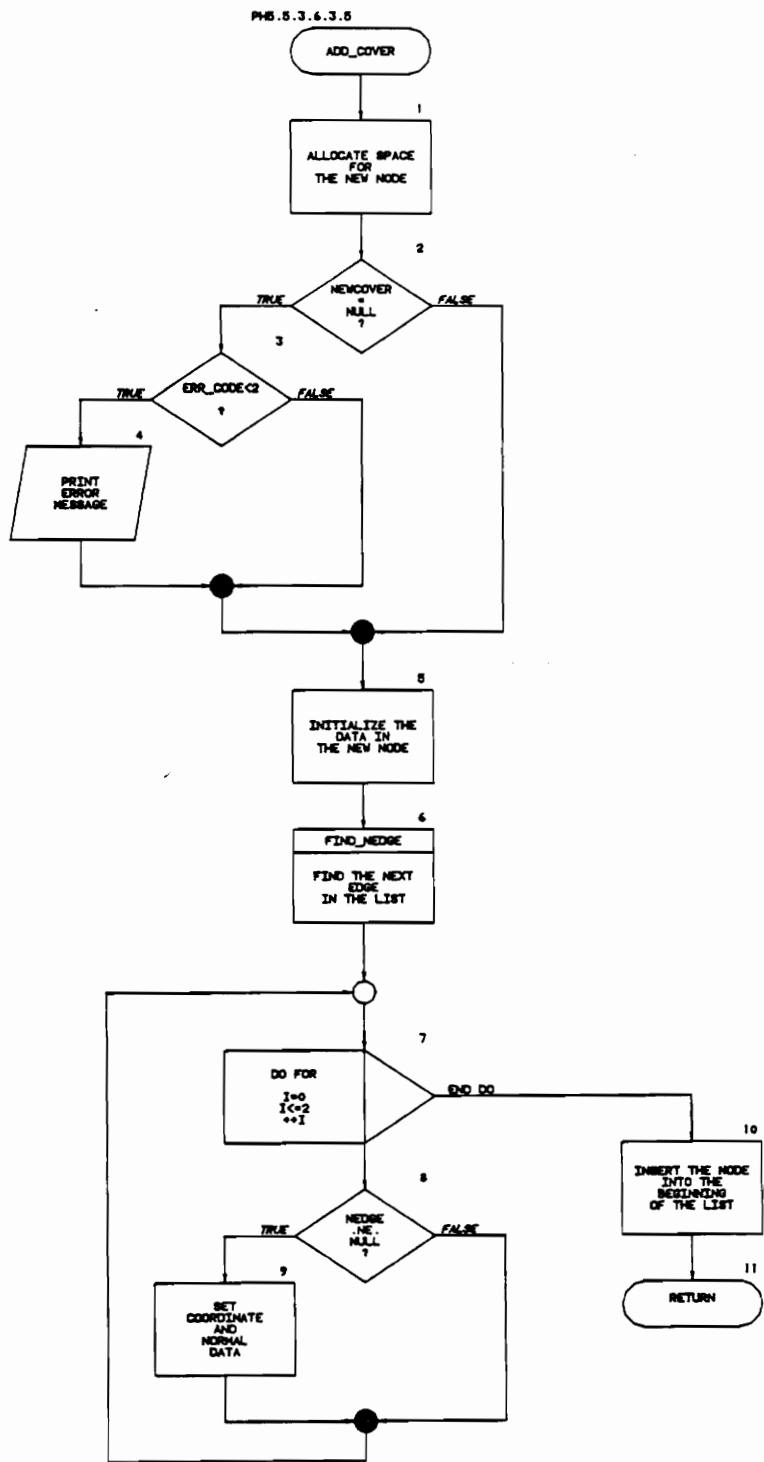
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: PRO_COVERS	MODULE: PH5.5.3.6.3
DESIGNED BY: KRISHNAN KOLADY	NOTE: PROCESSES THE FACES COVERING THE PIXELS
DATE: 6/1/90	



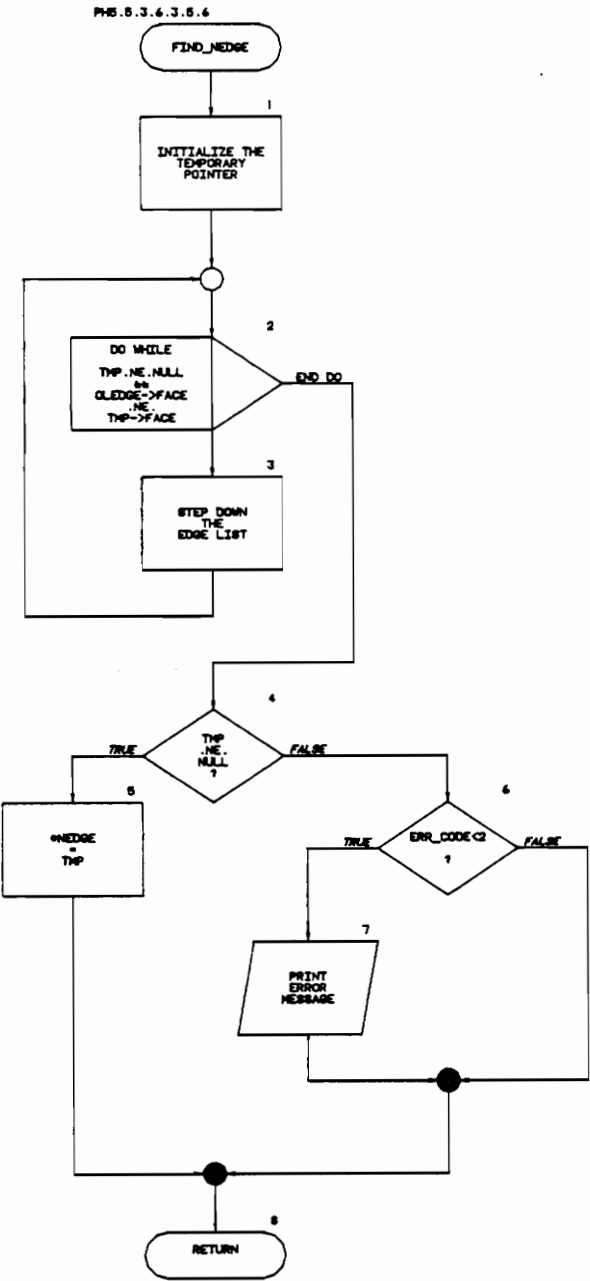
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	DEL_COVER	MODULE:PH5.5.3.6:3.4
DESIGNED BY:	KRISHNAN KOLADY	NOTE:DELETES THE ITEM FROM THE COVER LIST
DATE:	6/1/90	



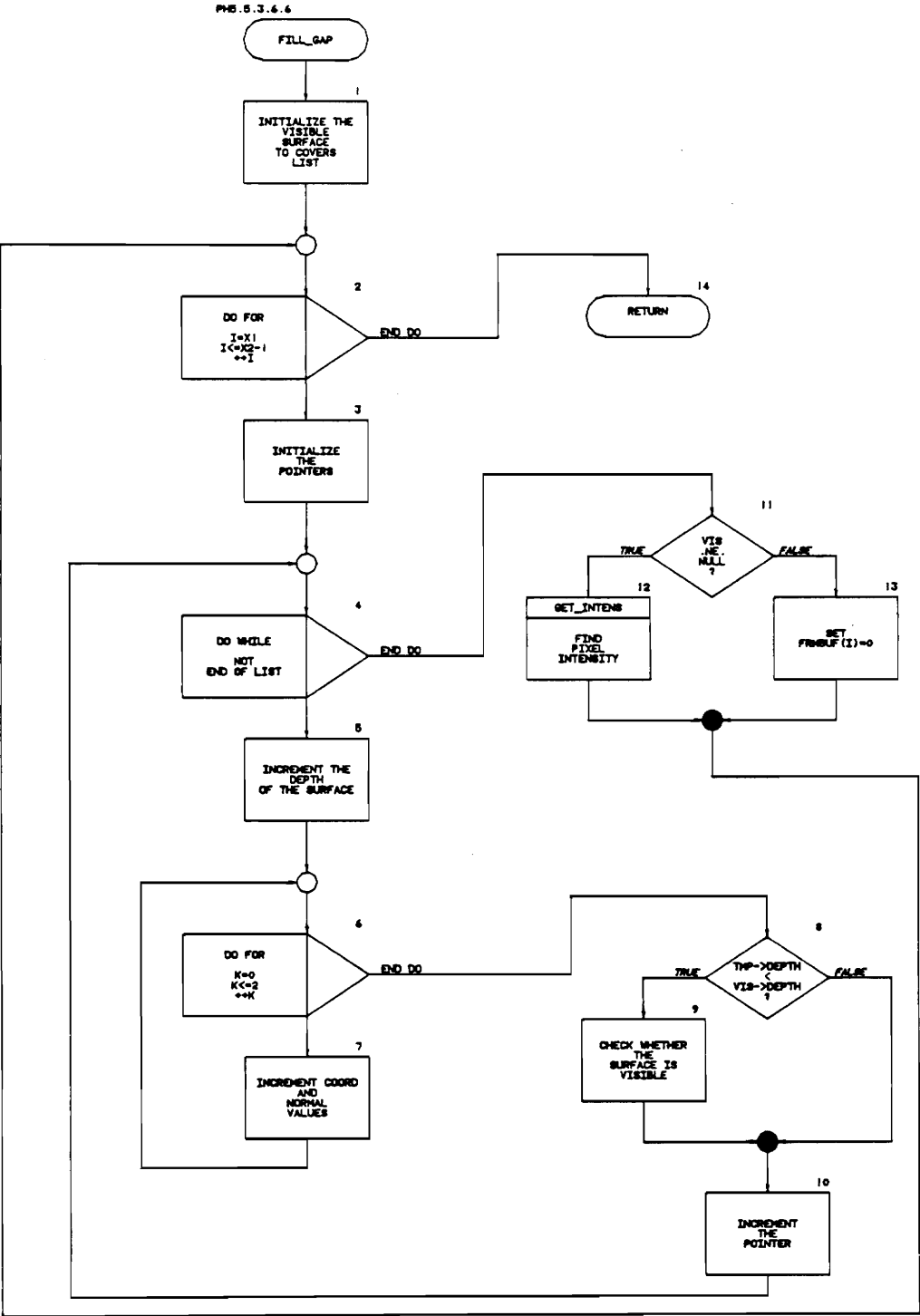
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>		
MODULE NAME:	ADD_COVER	MODULE: PH5.5.3.6.3.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: ADDS THE EDGE TO THE COVER LIST AS A FACE THAT IS COVERED
DATE:	6/1/90	



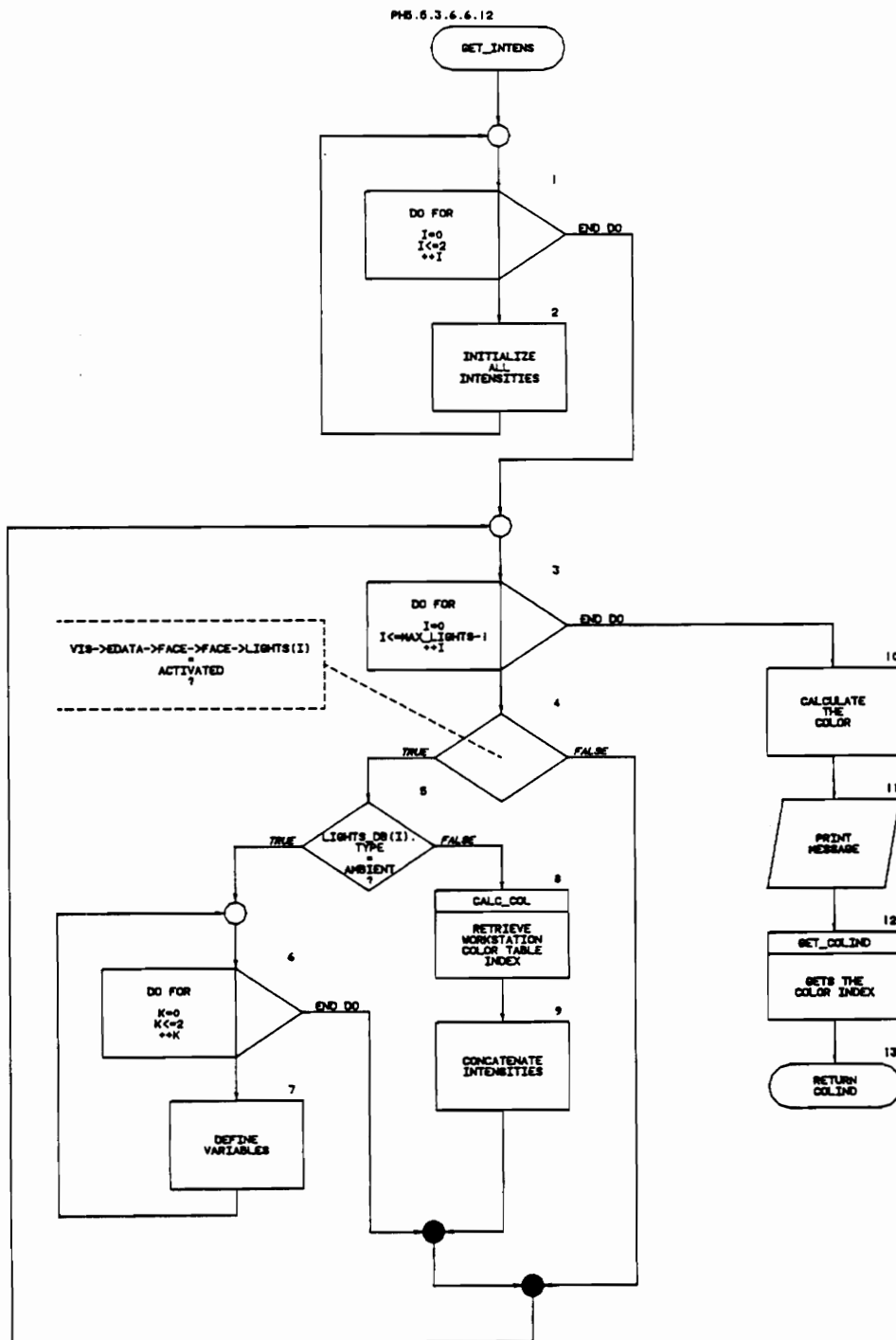
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	FIND_NEDGE	MODULE:PH5.5.3.6.3.5.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FINDS THE NEXT EDGE POINTING TO THE SAME FACE IN THE LIST
DATE:	6/1/90	



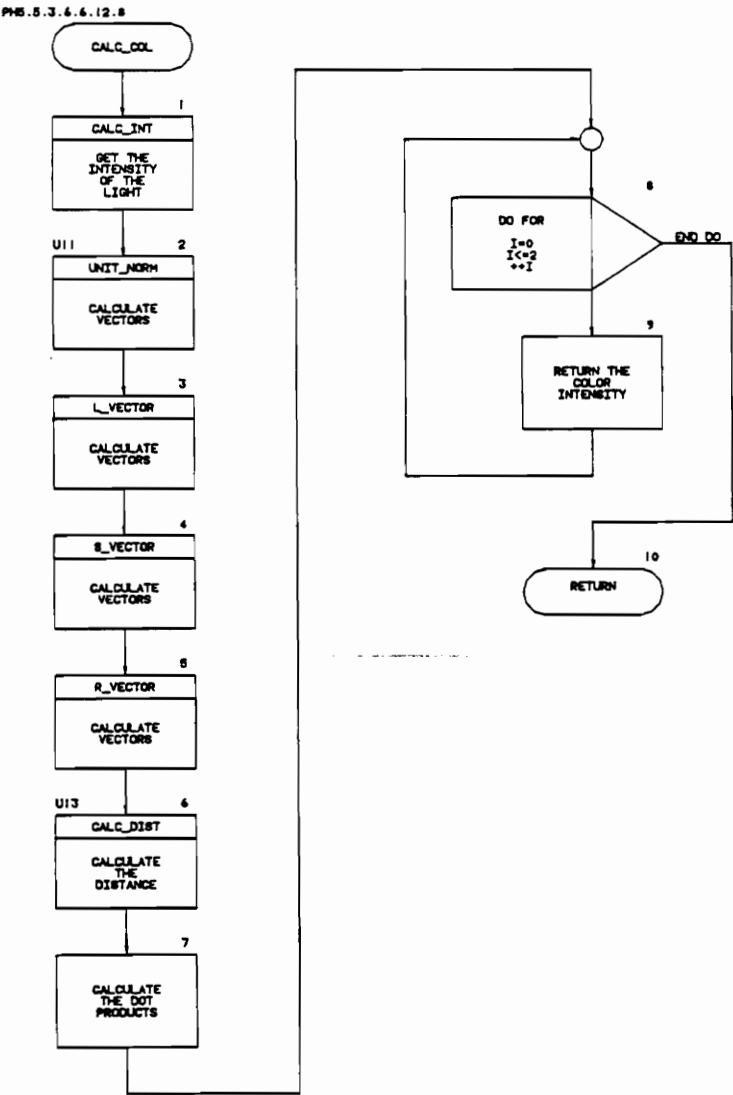
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	FILL_GAP	MODULE:PH5.5.3.6.6
DESIGNED BY:	KRISHNAN KOLADY	NOTE:FILLS THE FRAME BUFFER WITHIN THE SPECIFIED LIMITS
DATE:	6/1/90	



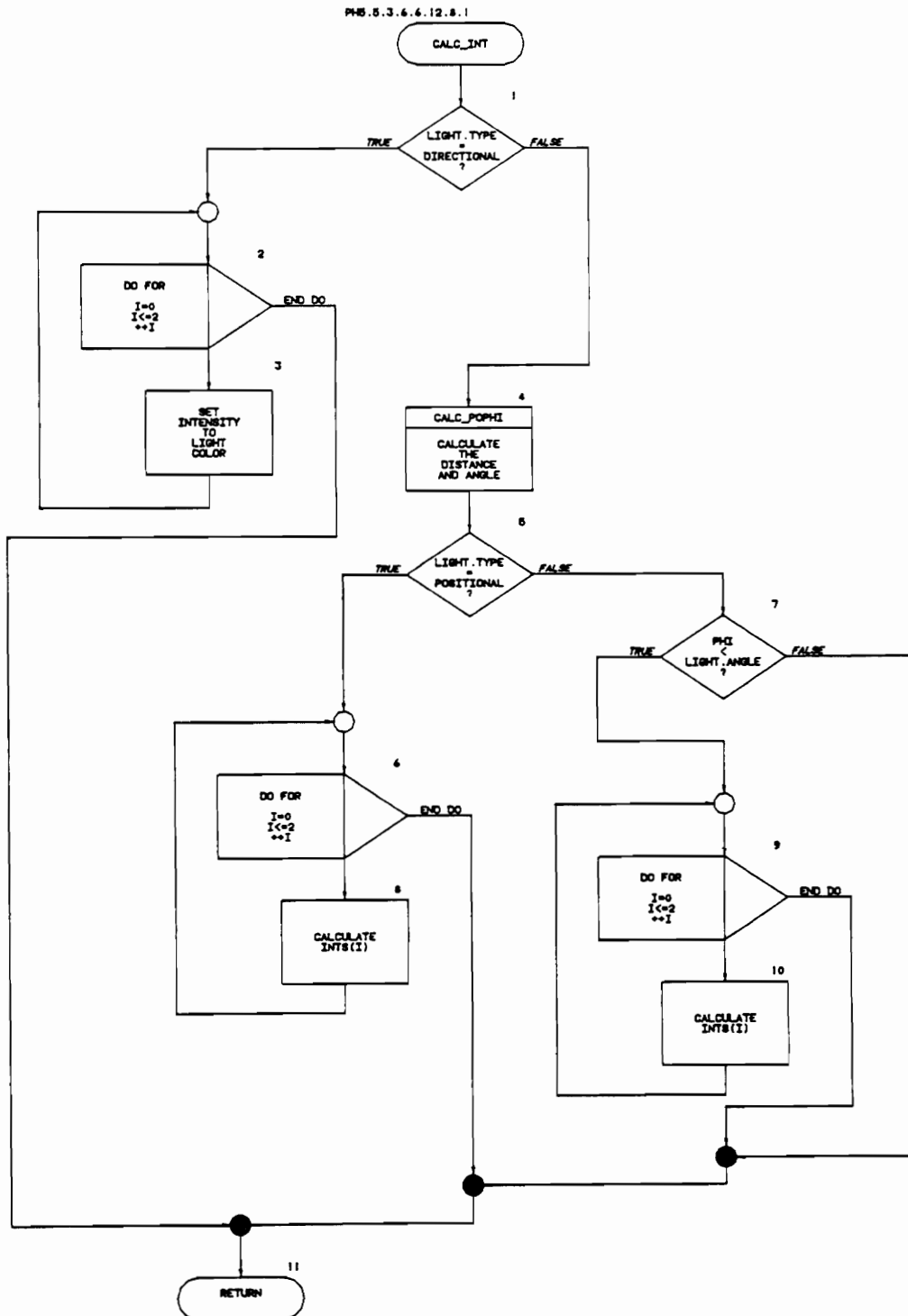
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: GET_INTENS	MODULE:PH5.5.3.6.6.12
DESIGNED BY: KRISHNAN KOLADY	NOTE:FINDS THE INTENSITY OF THE PIXEL FROM THE ACTIVE LIGHT SOURCES
DATE: 6/1/90	



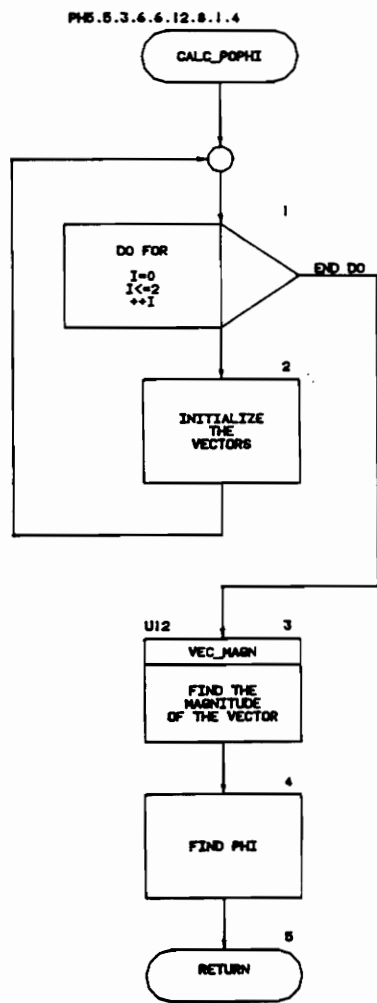
<div> <div>VT</div> <div>PROGRAM SPECIFICATION - PHONG SHADING</div> </div>		
MODULE NAME:	CALC_COL	MODULE:PH5.5.3.6.6.12.8
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CALCULATES INTENSITY FOR GIVEN LIGHT SOURCE
DATE:	6/1/90	



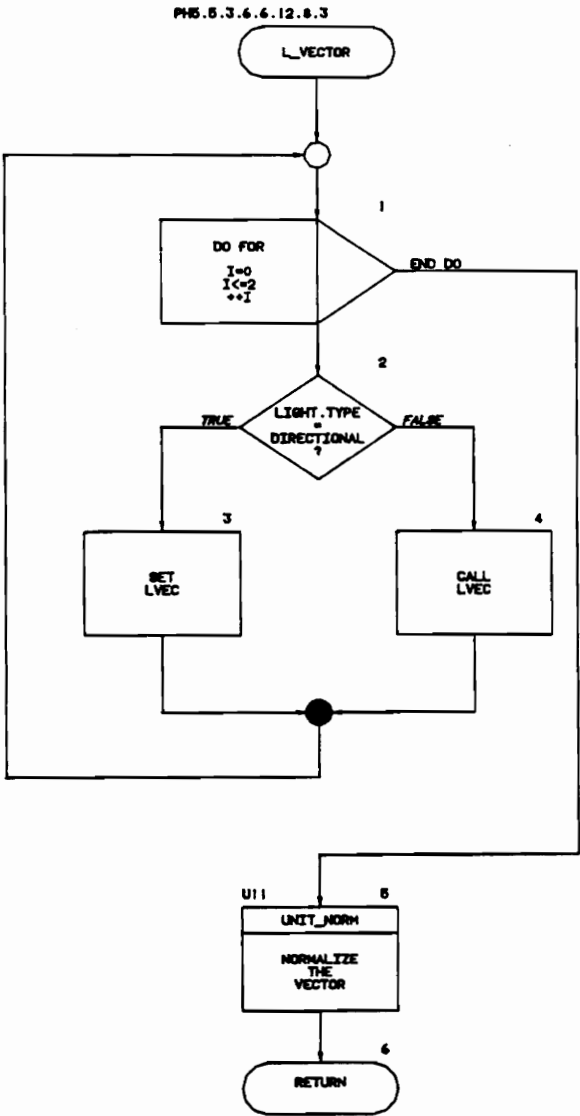
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	CALC_INT	MODULE:PH5.5.3.6.6.12.8.1
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CALCULATES INTENSITY DUE TO LIGHT COLOR AND ANGLE
DATE:	6/1/90	



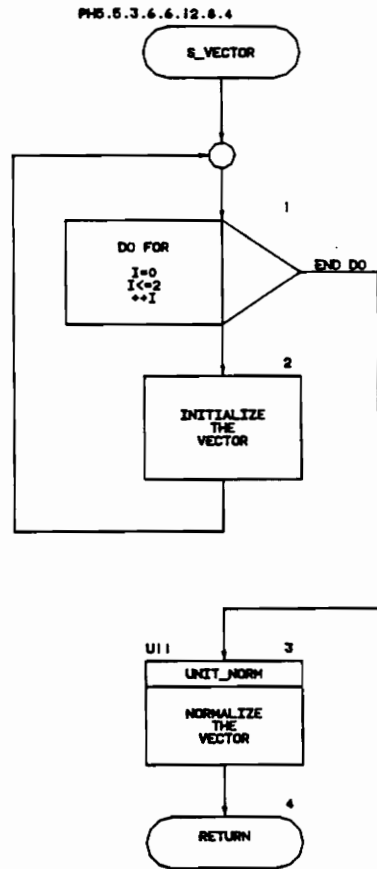
VT			PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	CALC_POPHI	MODULE:	PH5.5.3.6.6.12.8.1.4
DESIGNED BY:	KRISHNAN KOLADY	NOTE:	FINDS ANGLE BETWEEN LIGHT DIRECTION AND NORMAL
DATE:	6/1/90		



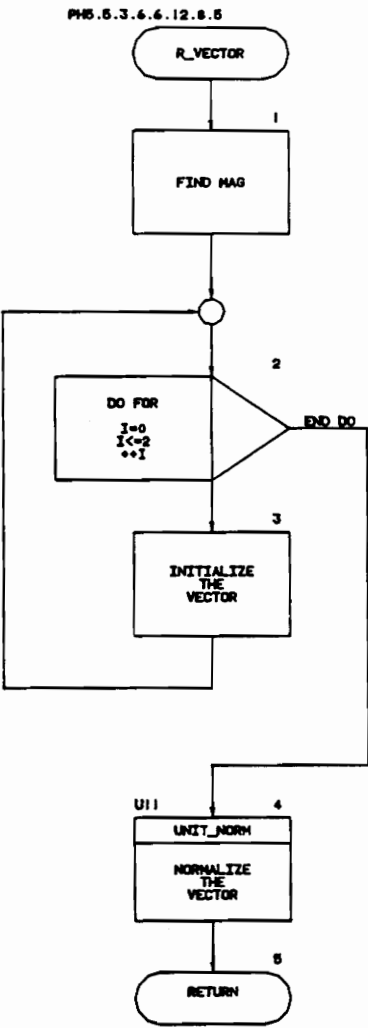
VT		PROGRAM SPECIFICATION - PHONG SHADING
MODULE NAME:	L_VECTOR	MODULE:PH5.5.3.6.6.12.8.3
DESIGNED BY:	KRISHNAN KOLADY	NOTE:FIND L VECTOR
DATE:	6/1/90	



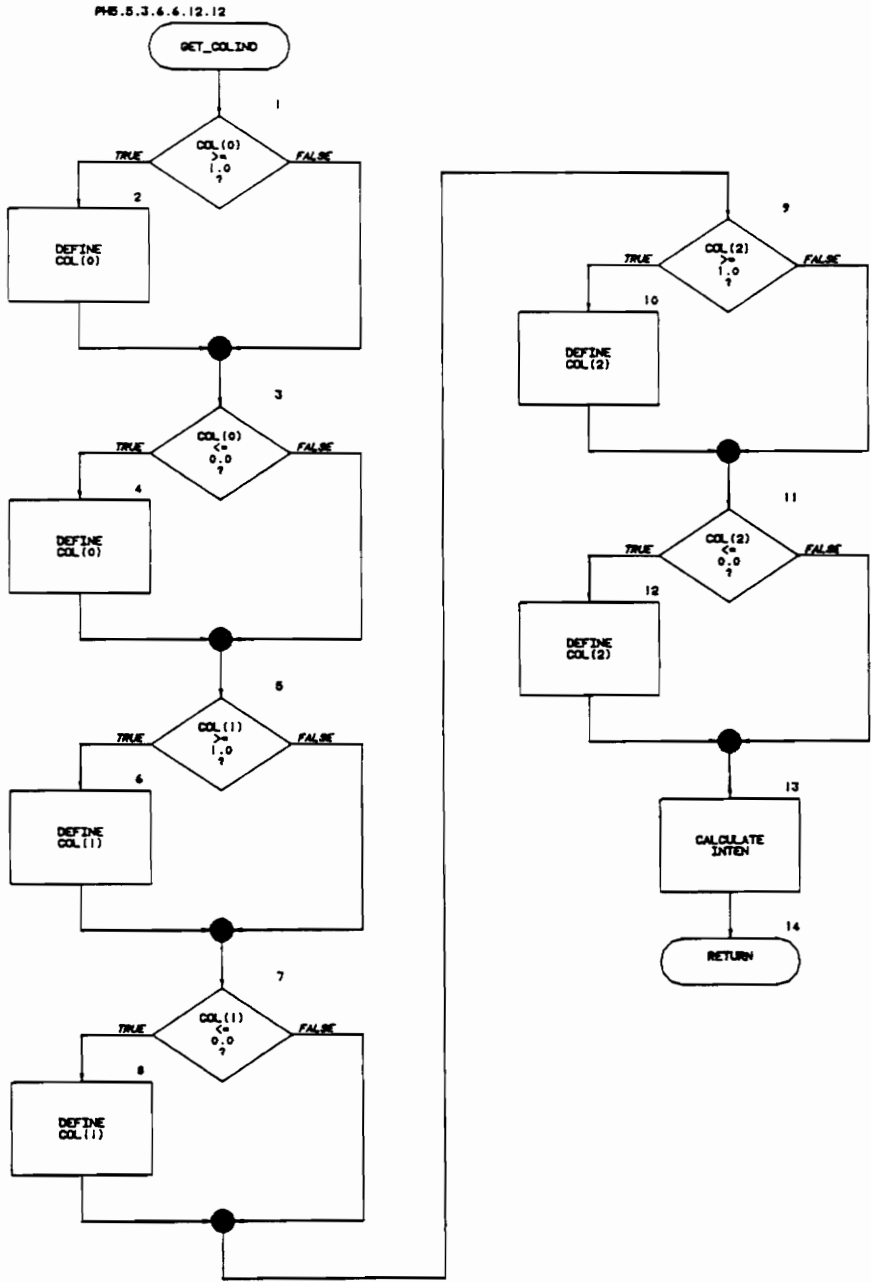
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: S_VECTOR	MODULE: PH5.5.3.6.6.12.8.4
DESIGNED BY: KRISHNAN KOLADY	NOTE: FIND S VECTOR
DATE: 6/1/90	



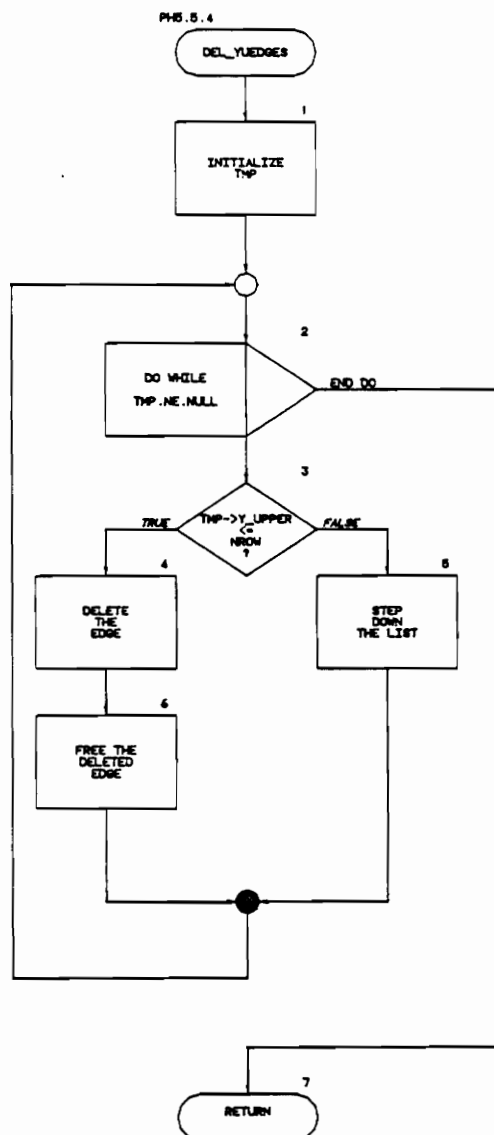
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	R_VECTOR	MODULE:PH5.5.3.6.6.12.8.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE:FIND R VECTOR
DATE:	6/1/90	



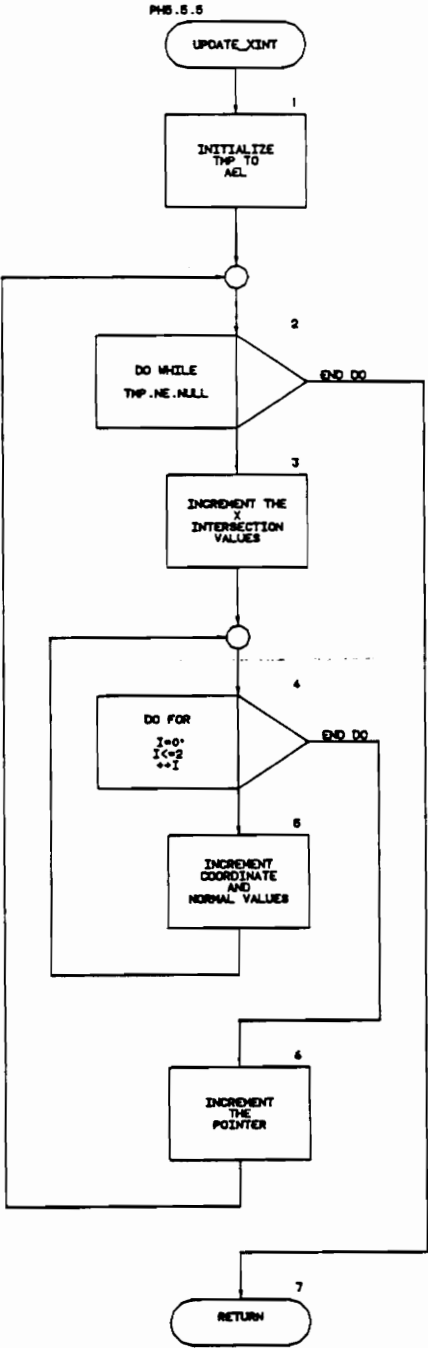
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: GET_COLIND	MODULE: PH5.5.3.6.6.12.12
DESIGNED BY: KRISHNAN KOLADY	NOTE: GETS THE COLOR INDEX FOR GIVEN INTENSITY
DATE: 5/30/90	



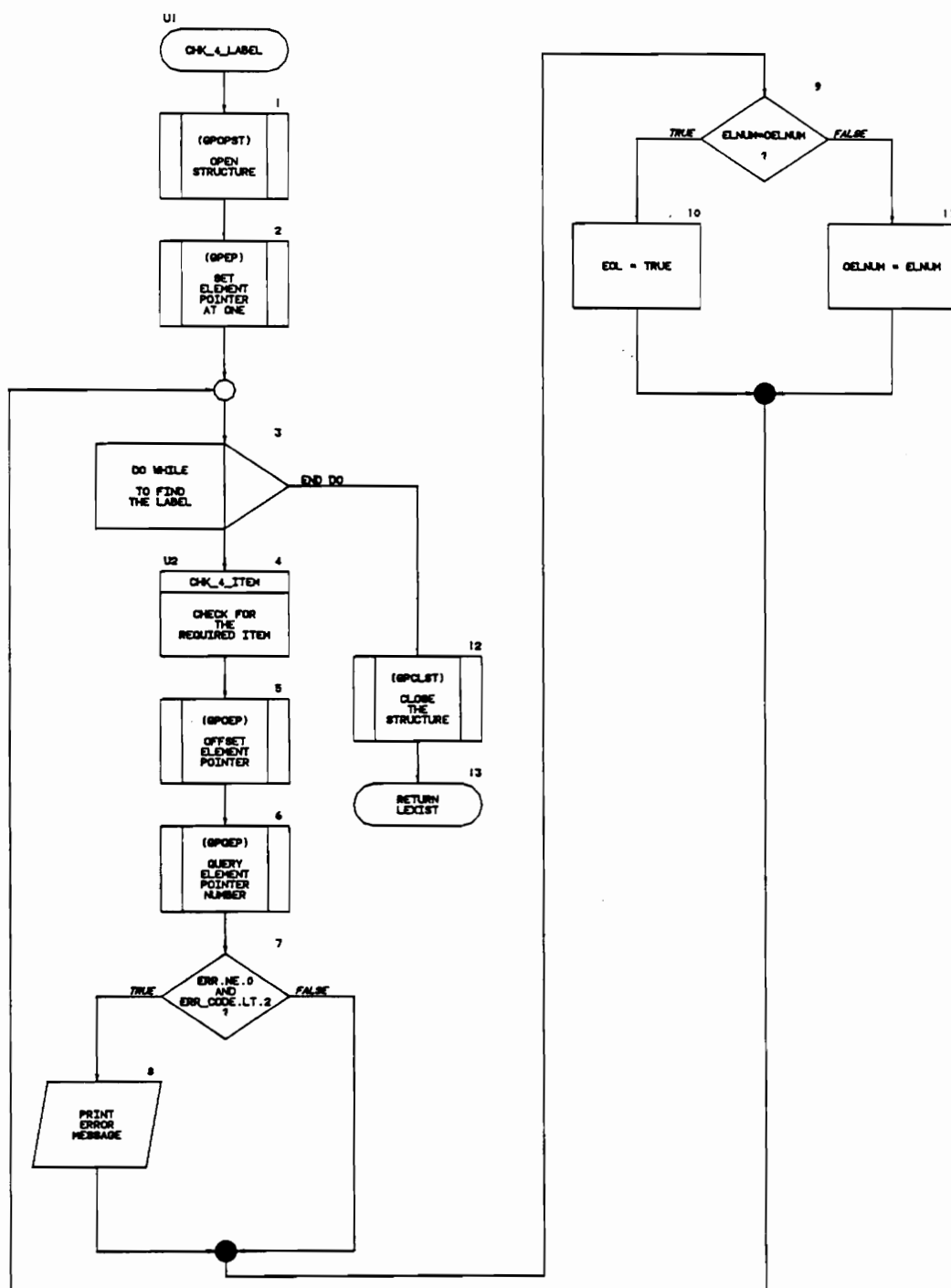
VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: DEL_YUEDGES	MODULE: PH5.5.4
DESIGNED BY: KRISHNAN KOLADY	NOTE: DELETES EDGES WHICH HAVE THE SAME Y UPPER LIMIT AS THE CURRENT ROW FROM THE ACTIVE EDGE LIST
DATE: 5/30/90	



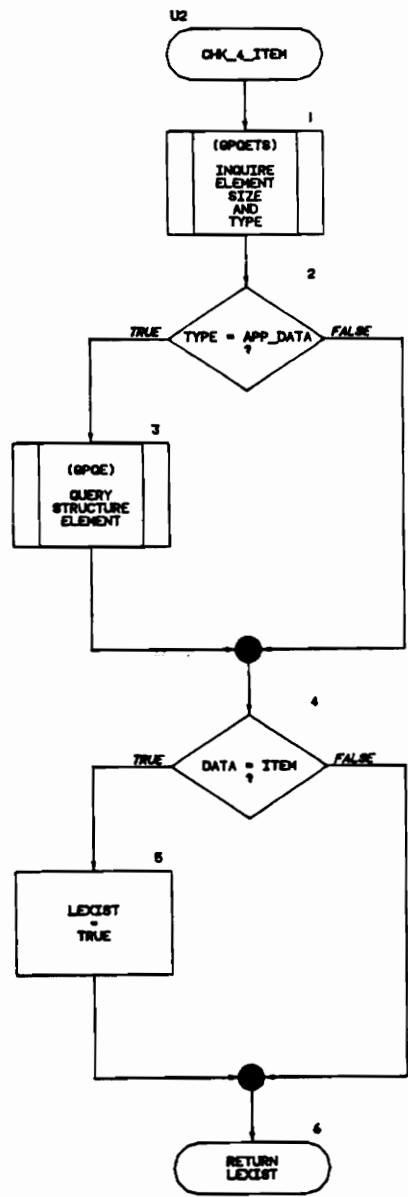
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	UPDATE_XINT	MODULE: PH5.5.5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: UPDATES THE EDGE INFORMATION ACROSS A SCAN LINE
DATE:	6/1/90	



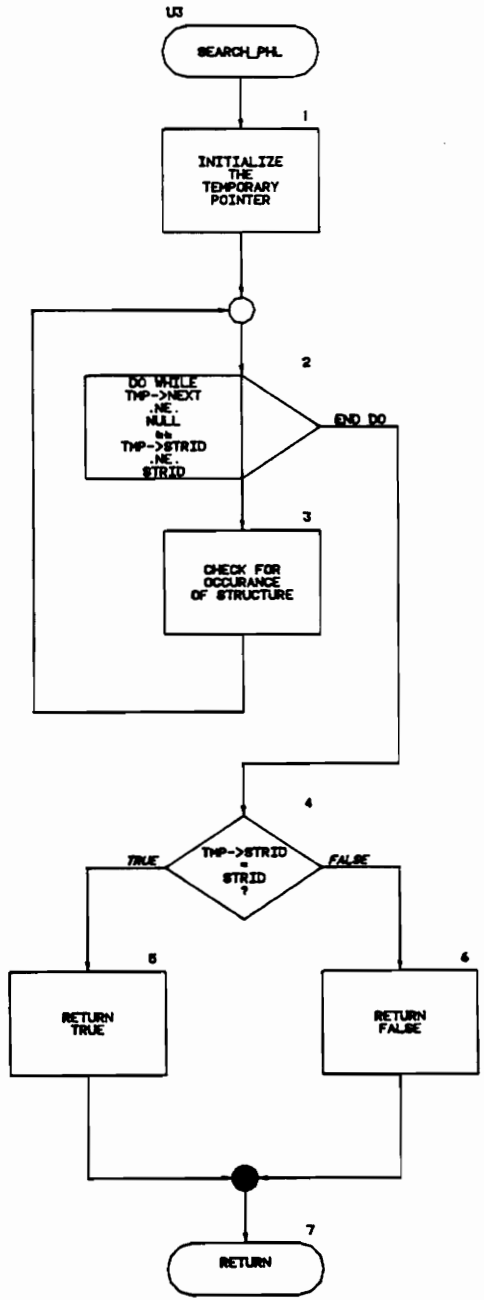
VT PROGRAM SPECIFICATION - PHONG SHADING	
MODULE NAME: CHK_4_LABEL	MODULE: U1
DESIGNED BY: KRISHNAN KOLADY	NOTE: CHECKS WHETHER THE REQUIRED LABEL EXISTS WITHIN THE STRUCTURE
DATE: 3/9/90	



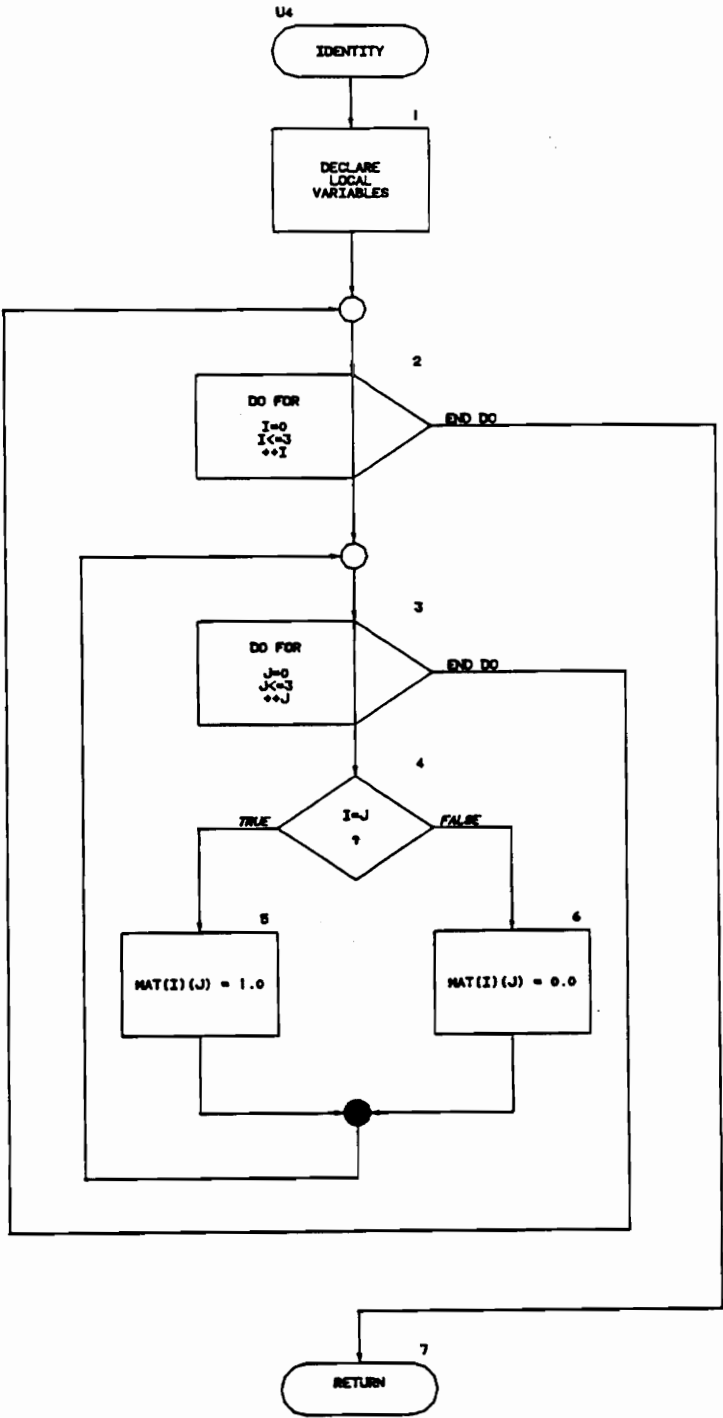
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	CHK_4_ITEM	MODULE: U2
DESIGNED BY:	KRISHNAN KOLADY	NOTE: CHECKS WHETHER THE SPECIFIED ITEM EXISTS AS AN APPLICATION DATA WITHIN THE OPEN STRUCTURE AND POINTER ELEMENT
DATE:	3/9/90	



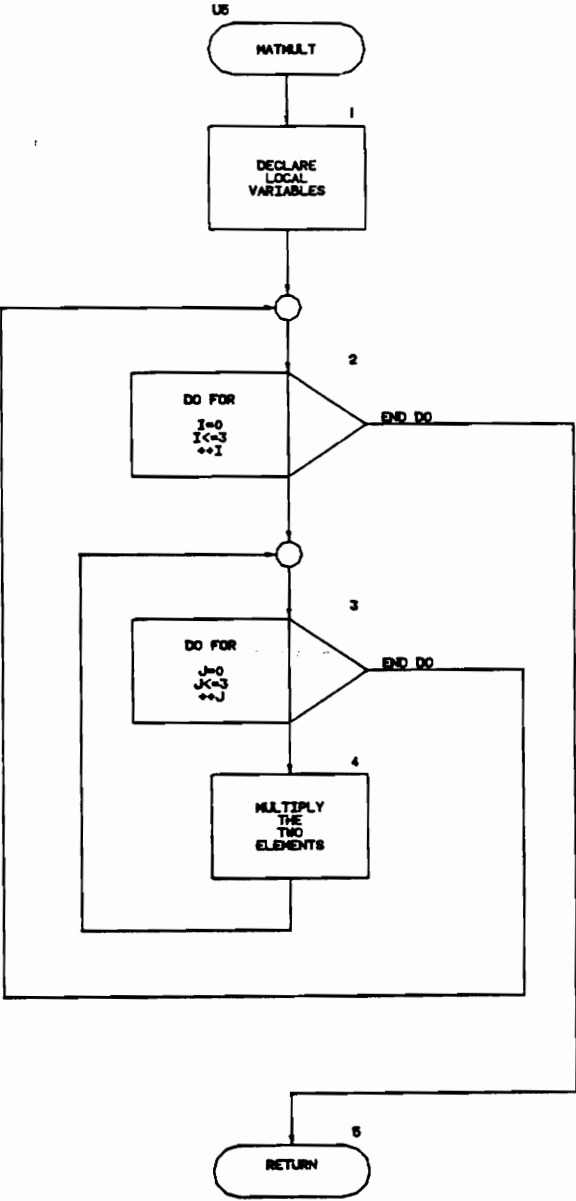
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	SEARCH_PHL	MODULE: U3
DESIGNED BY:	KRISHNAN KOLADY	NOTE: SEARCHES PHONG STRUCTURE LIST FOR THE STRUCTURE IDENTIFIER PASSED IN
DATE:	3/20/90	



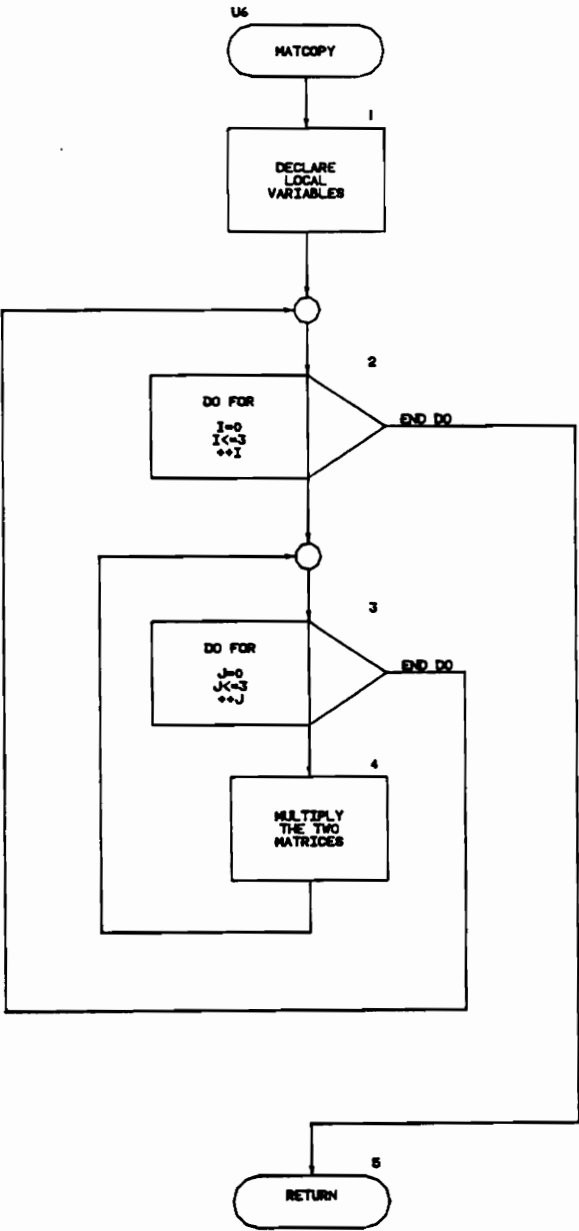
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	IDENTITY	MODULE: U4
DESIGNED BY:	KRISHNAN KOLADY	NOTE: INITIALIZES THE INPUT MATRIX TO AN IDENTITY MATRIX
DATE:	3/20/90	



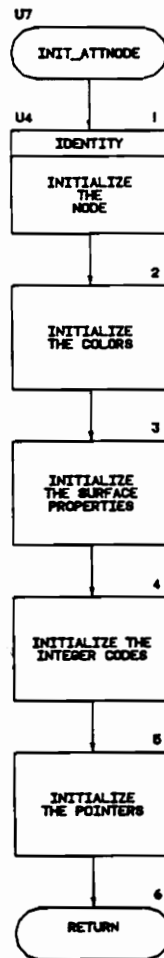
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	MATMULT	MODULE: U5
DESIGNED BY:	KRISHNAN KOLADY	NOTE: MULTIPLIES TWO 4X4 MATRICES
DATE:	3/20/90	



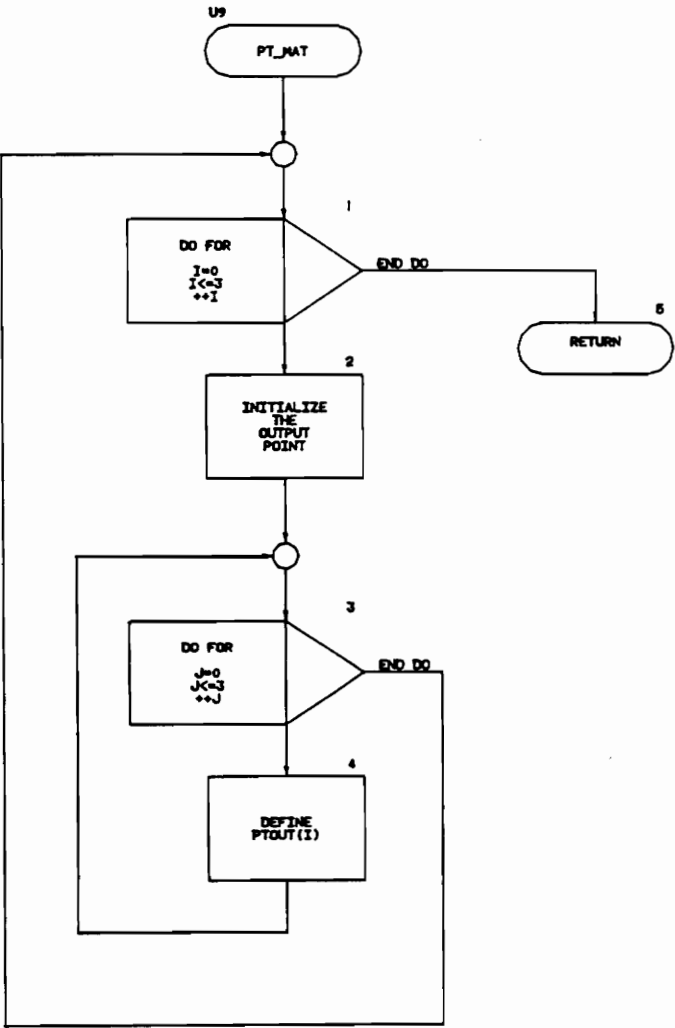
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	MATCOPY	MODULE: U6
DESIGNED BY:	KRISHNAN KOLADY	NOTE: COPIES ONE 4X4 MATRIX INTO ANOTHER
DATE:	3/20/90	



VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: INIT_ATTNODE	MODULE: U7
DESIGNED BY: KRISHNAN KOLADY	NOTE: INITIALIZES THE ATTRIBUTE LIST NODE
DATE: 3/20/90	



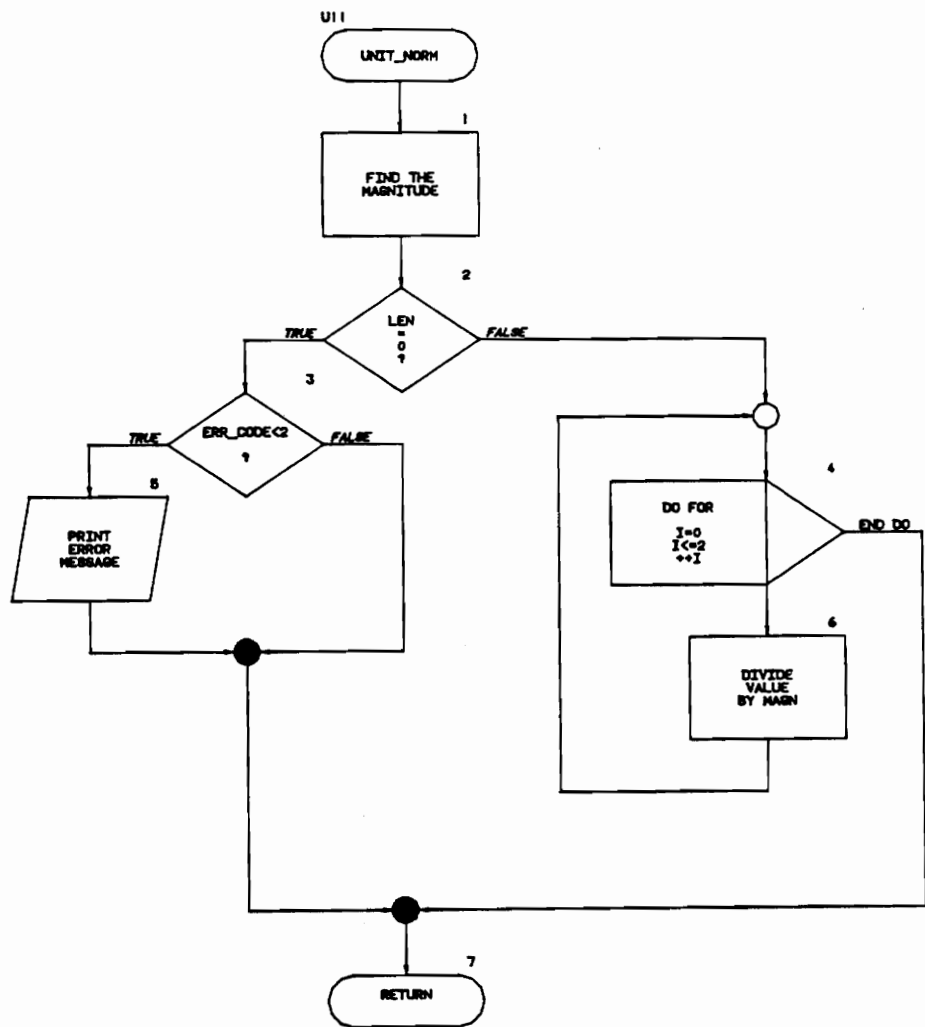
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	PT_MAT	MODULE:U9
DESIGNED BY:	KRISHNAN KOLADY	NOTE: GIVES THE PRODUCT OF A 4X1 POINT IN HOMOGENEOUS COORDINATES AND A 4X4 MATRIX
DATE:	5/24/90	



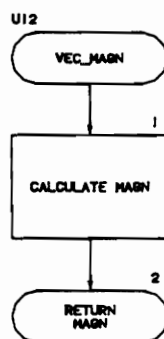
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	DOT	MODULE:U10
DESIGNED BY:	KRISHNAN KOLADY	NOTE: FINDS THE DOT PRODUCT OF TWO VECTORS
DATE:	6/1/90	



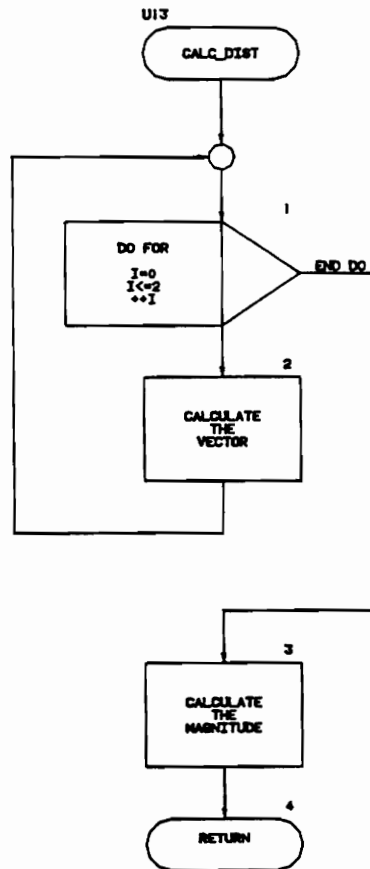
VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	UNIT_NORM	MODULE:U11
DESIGNED BY:	KRISHNAN KOLADY	NOTE:RETURNS A UNIT VECTOR IN THE DIRECTION OF THE GIVEN VECTOR
DATE:	6/1/90	



VT PROGRAM SPECIFICATION - PHONG SHADING		
MODULE NAME:	VEC_MAGN	MODULE: U12
DESIGNED BY:	KRISHNAN KOLADY	NOTE: RETURNS THE MAGNITUDE OF THE GIVEN VECTOR
DATE:	6/1/90	



VT <i>PROGRAM SPECIFICATION - PHONG SHADING</i>	
MODULE NAME: CALC_DIST	MODULE: U13
DESIGNED BY: KRISHNAN KOLADY	NOTE: CALCULATES THE DISTANCE BETWEEN TWO POINTS
DATE: 6/1/90	



Vita

The author was born in 1965 and grew up in Bombay, India. After schooling in Don Bosco High School, Bombay, he received a Bachelor of Technology degree in Mechanical Engineering in the Fall of 1987 from the Government Engineering College, Trichur, Kerala. After completing his master's in Mechanical Engineering at Virginia Tech, the author will pursue a career as a Research Associate at Virginia Tech.

Krishnan V. Kolady