



The HOOPS 3D Graphics System: A Technical Overview

1	What is the HOOPS 3D Graphics System?	2
1.1	SUPPORTED PLATFORMS	2
1.2	DEVICE SUPPORT	3
2	The HOOPS/3dGS Architecture	4
2.1	RETAINED-MODE GRAPHICS SYSTEM	5
2.2	THE OBJECT DATABASE	6
2.2.1	Coordinates and Coordinate Systems	6
2.2.2	Inserting Geometry By Reference	7
2.3	DATABASE TRAVERSAL	7
2.4	RENDERING PIPELINE: THE HOOPS/3dGS STRUCTURED DEVICE INTERFACE® (HDI)	8
2.5	FLOW OF CONTROL	9
3	HOOPS as an Object-Oriented System	10
3.1	ENCAPSULATION AND DATA HIDING WITH SEGMENTS	10
3.1.1	Segment Pathnames and Handles	11
3.1.2	Instancing and Reuse with Include Segments	12
3.1.3	Driver Segments	12
3.2	MESSAGING AND METHODS	13
3.3	ATTRIBUTE INHERITANCE	13
3.3.1	Default Attribute Values	13
3.3.2	Nonstandard Attribute Inheritance	14
3.3.3	Attribute Lock	14
3.3.4	Style Segments	15
3.3.5	Attribute Inheritance Summary	15
4	Geometry and Attributes	16
4.1	GEOMETRY PROVIDED IN HOOPS/3dGS	16
4.2	ATTRIBUTES	20
4.2.1	Display Attributes	20
4.2.2	Modeling & Viewing	21
4.2.3	Rendering Attributes	23
4.2.4	Selection	24
4.2.5	Driver Options	24
4.2.6	System Options	24
4.2.7	Heuristics	25
5	Input and Hit-testing	26
6	Integrating HOOPS/3dGS with a GUI Toolkit	28
6.1	WINDOW SYSTEM INTEGRATION	28
6.2	DRAWING INTO NATIVE APPLICATION WINDOWS	28
6.3	COLORMAP SHARING	28
7	HOOPS/3dGS Intermediate Mode	29
7.1	WHY USE HOOPS/3dGS INTERMEDIATE MODE?	31
8	Appendix A: Immediate Versus Retained Mode Graphics Systems	32



1 What is the HOOPS 3D Graphics System?

HOOPS 3D Graphics System (HOOPS/3dGS) is a high performance 3D rendering toolkit for developers building applications for the Windows and UNIX operating systems and the Internet. HOOPS/3dGS' highly optimized data structures and algorithms dramatically simplify the development of 2D and 3D, interactive, vector and raster graphics-based CAD/CAM/CAE, Scientific Visualization, and Geographical Information System (GIS) applications.

HOOPS/3dGS contains:

- A subroutine library that provides for the creation, management, querying and editing of an application's graphical information and is linked with an application's object code. The libraries can be dynamically or statically linked.
- A large suite of supporting demonstration and integration code to assist developers in learning about HOOPS/3dGS and incorporating it into their application.

Application developers combine their core application logic with HOOPS/3dGS and the user interface to develop interactive graphics applications.

HOOPS/3dGS is just one of the modules contained within Tech Soft America's HOOPS 3D Application Framework (HOOPS/3dAF). HOOPS/3dAF consists of a highly optimized, integrated suite of industry leading software components. This framework provides an extensible, modular base architecture that enables the rapid development of world-class 3D CAD/CAM/CAE applications for the Windows and UNIX operating systems.

For more information on how HOOPS/3dGS and other modules of the HOOPS 3D Application Framework integrate into a customer's interactive graphics application, refer to the HOOPS/3dAF technical overview at <http://www.techsoftamerica.com>.

1.1 Supported Platforms

Windows Operating System

Windows 95/98

Windows NT 4.0 Intel

Unix Operating System

IBM RS6000: AIX

DEC Alpha OSF/1

SGI: IRIX

HP: HP-UX

Sun: Solaris

Intel: LINUX



Available through obfuscated source (already ported):

Dec Ultrix
Dec Alpha OpenVMS
HP-UX for HP300/400
Intergraph Clipper Unix
IRIX 4.0x for SGI
OpenVMS VAX
XDOS MetaWare / Phar Lap
Windows NT for Intel using WATCOM
Windows NT for MIPS
Windows NT DEC Alpha
SunOS

1.2 Device Support

Display Drivers

GL (AIX, SGI)
OpenGL (AIX, Dec Alpha OSF/1, HP/UX, SGI, Solaris, Windows NT)
HP Starbase
Sun XGL
X11 (for all major Unix platforms)
MS Windows GDI (Screen)
MS Windows GDI (Clipboard)

Hardcopy Drivers

PostScript
CGM
HP-GL
PICT
MS Windows GDI printing
TIFF (raster file)
HOOPS/3dGS Image (raster file)

Graphics File Formats (input only)

HMF – HOOPS/3dGS Metafile
GIF – Graphics Interchange Format
STL – Stereo Lithography



2 The HOOPS/3dGS Architecture

HOOPS/3dGS is a retained mode graphics system with a database architecture. HOOPS/3dGS provides the algorithms for creating, editing, manipulating, and querying the graphics information stored in the database.

This encapsulates the graphical data in HOOPS, coupling it with an interface layer. This coupling of data structures with algorithms is the fundamental principle of object-oriented design.

Data-encapsulation, messaging, instancing and attribute inheritance are also fundamental design elements of HOOPS/3dGS. While HOOPS/3dGS is not implemented as a class library, its architecture employs these fundamental aspects of object-oriented design.

The HOOPS/3dGS libraries are incorporated directly into the application's build process, i.e., linked with the application's other object code to produce the executable image. The developer uses the HOOPS/3dGS API, (a.k.a., subroutines or object methods), to create and manage geometrical objects from within the other components of their application.

The HOOPS/3dGS Graphics System, illustrated in Figure 1, consists of 2 major subcomponents: a graphical object database called the HOOPS/3dGS Segment Tree, and a rendering pipeline called the HOOPS/3dGS Structured Device Interface.

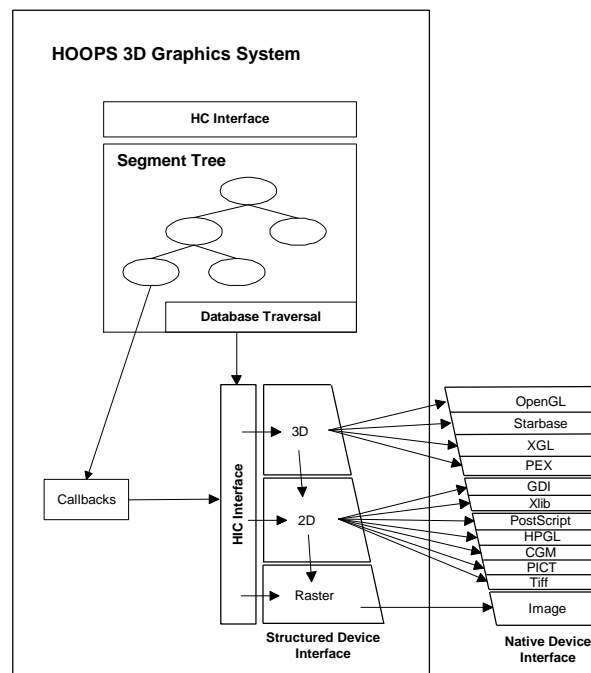


Figure 1: Architecture of the HOOPS 3D Graphics System



2.1 Retained-Mode Graphics System

Graphics systems that render primitives immediately, without storing them in a display list, are called *immediate mode* systems. Systems that store graphics information in data structures specifically designed for the display of graphics are called *retained mode* graphics libraries. (For more information on immediate-mode versus retained-mode graphics systems, see Appendix A.)

HOOPS/3dGS is a retained-mode graphics library. Retaining the graphics primitives in a retained mode system provides significant benefits:

- **Performance**^¾ If the graphics system (rather than the application) retains a copy of all the primitives, it can perform faster modification and traversal of the graphics information as well as optimizations to make rendering the scene faster. For example, the graphics system can calculate bounding volumes that store the location and a rough measure of the extent of a set of primitives. During rendering, many primitives may not be visible on the screen. Bounding volumes allow the system to quickly determine whether a group of primitives is on-screen so that only those graphics primitives that are actually visible are sent to the display hardware to be rendered.
- **Selection**^¾ An important function of a graphics system is performing selections (also called “picking”), where the graphics system determines which graphic primitive the user is pointing at with the cursor. Without a display list, the only way for the graphics system to perform a selection is to have the application resend all the primitives since they were not retained by the graphics system. A display list allows the graphics system to perform selection much faster, typically an order of magnitude faster.
- **Window system support**— If a window containing a scene is partially obscured by another window and then it is brought to the front, the newly-exposed areas of the window need to be redrawn. If the graphics system stores the primitives in a display list, then the window can be redrawn without going back to the application. (The window’s image can also be stored as a bitmap, allowing instant redraws of damaged areas. HOOPS/3dGS supports this method as well.)
- **Global rendering algorithms**— Many kinds of renderers require a copy of all the graphics primitives. For example, for ray tracing and radiosity, the color of an object is affected by other objects, so the renderer requires data on all the primitives before it can start drawing any of them. Such advanced rendering algorithms require the use of a display list to store the primitives.



2.2 The Object Database

The HOOPS/3dGS database stores graphical data in objects called “segments”. Think of a segment as a container for geometry and attributes that describe how the geometry is to be drawn. Segment-to-segment relationships are hierarchical and are described as “parent-child” pairings or, as one segment “owning” others (its children). The mapping is one to many; that is, one parent segment may have many child segments but every child segment has one unique parent segment.

Segments may be instantiated several times and inserted into the tree in multiple places. This process is called *inclusion*, as in “one segment includes another”. Often, only the attribute set of segment need be instantiated and used by other segments; this process is called *styling*. Inclusion and styling are discussed in more detail below in chapter 4.

These segment to segment relationships result in a hierarchical tree structure, or more specifically, a directed acyclic graph. This structure enables attribute inheritance. Child segments have the same attribute values as their parent segment, unless they specifically have their own local settings for certain attributes.

The HOOPS/3dGS database structure ensures optimal speed by partitioning the geometric data into objects with homogeneous attributes. This minimizes the need to change hardware display context during drawing of the graphical information and optimizes display-list throughput.

2.2.1 Coordinates and Coordinate Systems

In HOOPS/3dGS, as in most graphics systems, points in space are specified using Cartesian coordinates. HOOPS/3dGS routines always take three x, y, and z coordinates; this means all points in HOOPS/3dGS are three-dimensional. However, a z-value of zero indicates a two-dimensional object, so HOOPS/3dGS uses more optimized two-dimensional rendering routines as appropriate.

Coordinates in HOOPS/3dGS are specified using single-precision floating point numbers. By default, however, C and C++ perform floating point arithmetic using double-precision (64 bit) numbers. These double-precision numbers are converted to single-precision when passed to a HOOPS/3dGS call.

Version 5.0 of HOOPS/3dGS includes a double-precision module that enables double-precision floating point values to be used when creating and querying geometry. It does this by providing variants of all the HOOPS routines for geometry creation, editing, and querying, as well as for several of the coordinate system conversion routines.



2.2.2 Inserting Geometry By Reference

When geometry is inserted into the HOOPS/3dGS database, HOOPS/3dGS makes a copy of the supplied data; this is normal for a retained-mode system and necessary for data encapsulation. For some kinds of geometry, this data can be quite large. Making a copy wastes space and makes the application bigger and slower than it needs to be.

To avoid this problem, HOOPS/3dGS allows for the insertion of meshes, shells, and images by reference. Inserting geometry by reference tells HOOPS/3dGS that it is acceptable not to make a copy of the data. For a mesh, this data is the point list. For a shell, it is the point list and the face list. For an image, it is the image data.

2.3 Database Traversal

The information in the database must be examined and formatted for the specific device interface available on the computer hardware. This operation is referred to as *database traversal*. The internal traversal routine “walks” from segment to segment in the tree and sends the information it finds to the rendering pipeline.

A single subroutine call, **Update_Display**, instructs HOOPS/3dGS to walk the database tree, visiting each node that contains something to be drawn. To draw a segment, HOOPS/3dGS sends any geometry it contains to the rendering pipeline along with the segment's attributes, and recursively draws any sub-segments. Consequently, drawing a segment draws the entire branch of the tree with that segment as the root.

Because HOOPS/3dGS is a declarative graphics system, the order in which HOOPS/3dGS traverses the tree (or even whether it traverses the entire tree) is not important. This allows HOOPS/3dGS to choose how it walks the database. There are several types of decisions HOOPS/3dGS can make to minimize the amount of geometry sent to the rendering pipeline, thereby enhancing performance. They include:

- **Selective Traversal**— Because HOOPS/3dGS keeps track of portions of the display that are not changing between updates, unnecessary redraws are eliminated and damaged portions of the screen can be either selectively redrawn or instantly re-blitted from the software z-buffer or software frame buffer.
- **Incremental Updates**— HOOPS/3dGS contains update logic that allows geometry to be incrementally added to a scene without a full redraw.



2.4 Rendering Pipeline: The HOOPS/3dGS Structured Device Interface® (HDI)

Key to the optimized performance of HOOPS/3dGS is its rendering pipeline, called the HOOPS/3dGS Structured Device Interface (HDI). HDI accepts information found in the segments, reformats it for the hardware output device interface, and sends it to the device for drawing. HDI successively decomposes the information, passing the database information through software mapping layers until it is in the format the output device can handle.

If the hardware device interface understands 3D information (as with OpenGL), HOOPS/3dGS can pass along information without much change. But if the device interface only understands 2D pixels (as with Xlib or GDI), several layers of decomposition must be used. This ensures both optimal rendering throughput on a given device and consistent functionality of the HOOPS/3dGS API across multiple platforms and devices. This process is presented in Figure 2, below.

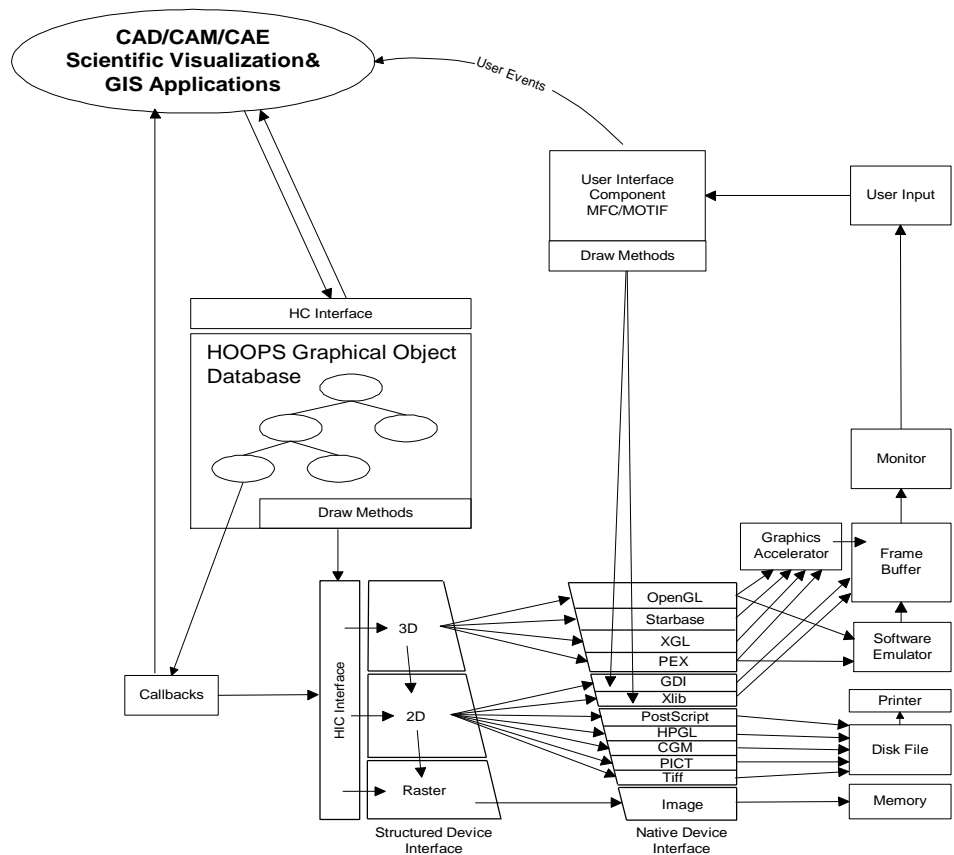


Figure 2: HOOPS/3dGS Structured Device Interface (HDI) Architecture



HDI ensures that application performance scales to system capability. Even on a system with graphics hardware, in some cases HOOPS/3dGS software can be faster than the hardware due to highly optimized rendering algorithms.

2.5 Flow of Control

The flow of control in a typical HOOPS-based application proceeds as follows:

1. The end-user of the application generates events via the user interface in effort to create new or manipulate existing application data (graphical or non-graphical).
2. The application code makes some changes to the information stored in HOOPS/3dGS.
3. HOOPS/3dGS then traverses the segment tree, determines what should be drawn, and sends this information to the HOOPS/3dGS Structured Device Interface which reformats it for the output device's capabilities and then sends it along to the output device.
4. The output device receives the information from the graphics system and draws the graphical information on the monitor or printer.
5. The user sees the new picture, decides what they want to do next and generates new input events with the user interface. (This takes us back to step 1.)



3 HOOPS as an Object-Oriented System

Object-oriented Design (OOD) is just that: design. Once complete, designs must be realized or expressed in a medium. In computer science terminology, the medium is a computer language, like C, C++, FORTRAN, Cobol, Java, etc. While the syntax of some languages encourages the developer to design in an OOD fashion, languages with less rigorous syntactical structure do not hinder the expression of object-oriented designs. Object-oriented architecture is independent of language and can be implemented in any language.

The main themes of object-oriented design are:

- Encapsulation of object interfaces (methods)
- Data Hiding
- Attribute Inheritance
- Reuse of objects via instancing

HOOPS/3dGS was designed prior to the presence of stable, mature implementations of C++. While HOOPS/3dGS is not implemented as a class library, its design incorporates each of these OOD elements. The ability for the HOOPS' development team to continually add functionality to the library over time and for HOOPS/3dGS to flexibly adapt to emerging APIs is strong testimony to its object-oriented design.

3.1 Encapsulation and Data Hiding With Segments

The HOOPS/3dGS graphics database stores graphics scenes as a hierarchy of *segments*, which lends itself naturally to organizing graphics information. In terms of object-oriented programming, think of a segment as an object.

- A segment encapsulates a public interface. Most HOOPS/3dGS commands modify the state of a segment, or return information about a segment or its contents.
- A segment uses data hiding to protect the internal details of its representation from the programmer. This makes life simpler for the programmer by hiding unnecessary details. It allows HOOPS/3dGS to modify its internal representation on different platforms for portability, to take advantage of faster algorithms, or to incorporate improved rendering techniques.
- A segment has attributes, which act similarly to the member variables of an object.
- A segment can have one or more sub-segments. The hierarchy formed by the segments in the database is like the class hierarchy in an object-oriented language (but more dynamic).
- Just as member variables are inherited by a class' subclasses, attributes are inherited by a segment's sub-segments. This form of inheritance is called attribute inheritance.



- A segment can be reused through the use of include segments and style segments.
- A segment contains a single list of the geometry that belongs to the segment.

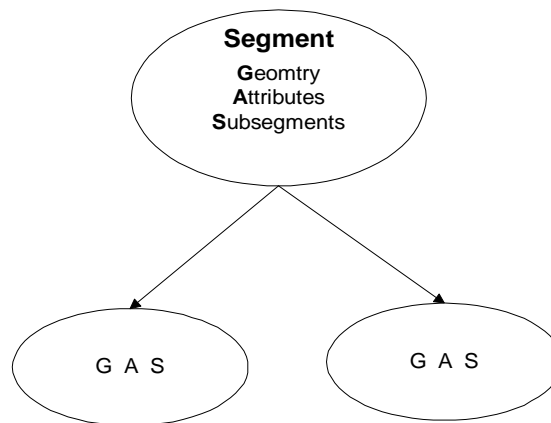


Figure 3: A segment contains geometry, attributes, and sub-segments.

3.1.1 Segment Pathnames and Handles

The entire segment tree is rooted under one segment, the *root-segment*, denoted with a forward-slash "/", and each segment occupies a unique spot in the segment tree, called its *pathname*.

There are several different ways to identify and subsequently reference a segment:

- **Implicitly**— Calls to HOOPS/3dGS attribute and geometry commands reference the currently open (active) segment, or more specifically, the most recently opened segment.
- **Explicitly, by name**— You can refer to a segment by its full pathname by using an ASCII string with the '/' denoting sub-segments. For example "/driver/msw/window0" would refer to the "window0" segment which is a sub-segment of "msw", etc.
- **By Key**— Accessing a segment by key is often more convenient and significantly faster than by name because keys don't require a name lookup. A key is a long integer and is returned by HOOPS/3dGS when a segment is initially created. Keys are commonly used to associate HOOPS/3dGS segments and geometry with application data (such as a data structure pointer).



3.1.2 Instancing and Reuse with Include Segments

HOOPS/3dGS has two kinds of sub-segments: regular segments and include segments. Include segments support a form of reuse. Include segments are similar to symbolic (soft) links in the UNIX file system because they allow multiple links to the same object. An included segment (also called an instance of a segment) behaves like a regular sub-segment, except that the parent of an included segment is always its real parent, not the segment that includes it.

A segment to be included actually lives as a normal segment (with its own geometry, attributes, and even sub-segments) in another part of the graphics database; for convenience, HOOPS/3dGS provides a place called "?Include Library".

3.1.3 Driver Segments

HOOPS/3dGS automatically defines a segment named "/driver". Underneath this segment are a dozen or more pre-created sub-segments—one segment for each kind of device for which HOOPS/3dGS has a device driver. These sub-segments of "/driver" are called driver segments. These are abstract classes and are not to be used directly, but rather instanced and built upon.

Each sub-segment of a particular driver segment is an instance of that driver, or more specifically, a unique connection to an output device. For a display driver on a window system like X11 or Windows, each instance corresponds to a connection to a window on the display. For a hardcopy driver like PostScript, each instance corresponds to a file to be printed. For example, "/driver/msw/window0" refers to a window on the MSW (Microsoft Windows) display device.

Figure 4 provides a graphical representation of segment hierarchy in HOOPS/3dGS.

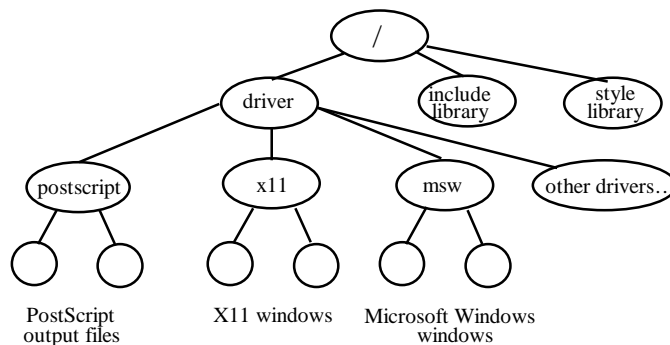


Figure 4: Default Segment Hierarchy



Multiple connections can exist simultaneously, and the use of include segments facilitates the rendering of a scene on disparate devices.

3.2 Messaging and Methods

Interface layers for communicating with graphics information in a database typically are either declarative or procedural. With declarative interfaces, the programmer declares *what* is to be displayed. With procedural interfaces, the programmer declares details of *how* to display the information. HOOPS/3dGS provides a declarative interface.

Procedural graphics systems have been found to be more difficult to use than declarative graphics systems. Because a segment can contain more than one value for an attribute in a procedural graphics system, it is much harder to find the value of any particular attribute, and harder still to change the value of an attribute and know to what geometry the attribute will apply. In addition, such systems cannot support attribute inheritance (in the object-oriented sense) like HOOPS/3dGS does.

In object-oriented programming, data is encapsulated in the object and hidden from external users of the object. An interface layer of methods is presented for creating and manipulating the internal state of the objects' data. HOOPS/3dGS declarative interface provides just such a mechanism for external manipulation of the internal information.

3.3 Attribute Inheritance

Attribute inheritance in HOOPS/3dGS works just like inheritance in an object-oriented language. If an attribute value is not set locally on a segment, it is inherited from a parent segment. Attribute values that are set explicitly on a segment are called local attributes. These are attributes set with any HOOPS/3dGS Set command, such as **Set_Color** or **Set_Line_Weight**.

When HOOPS/3dGS renders the database, it needs to know the attribute values for all the geometry in the database. To determine the values of attributes, HOOPS/3dGS first looks at local attributes set on the current segment. If any required attributes have not been set on the local segment, HOOPS/3dGS looks in the parent of this segment to see if a local attribute was set on it, and then its parent, and so on, until it finds a value for the attribute.

3.3.1 Default Attribute Values

When you start HOOPS/3dGS, it sets a local value for every attribute on the root segment of the database tree. When looking for the value of an attribute, HOOPS/3dGS eventually finds its way up to the root of the tree, and then uses the value found there. So the local attribute value set on the root segment of the database acts as a default value for the attribute value. The default value assigned by HOOPS/3dGS can be changed by changing the local attribute value on the root segment of the tree.



Another way to look at this is that any time a local attribute is set on a segment, a default value is established for all segments below that one in the tree. Thus, in a scene with multiple complex graphical objects, each one represented by its own sub-tree, you can have different default values for each object by setting local attributes on the root segments of their sub-trees.

3.3.2 Nonstandard Attribute Inheritance

Most attributes inherit their values from their parent segment. However, a few attributes inherit their values in a nonstandard manner.

Consider a modeling transform: a translation, rotation, or scale. Modeling transformations are always performed with respect to the parent segment. When an object is rotated, for example, it is useful for all of its children to rotate as well (even if they have their own local modeling transformations).

A segment's local modeling transformation is concatenated (using matrix multiplication) onto the modeling transformation of its parent—it doesn't simply replace it. If a segment has no local modeling transformations, the net transformation for the segment is the same as its parent's net modeling transformation. But if a segment does have a local modeling transformation, it is concatenated onto the net modeling transformation of the parent segment.

There are also a few attributes that inherit up the tree; for example, bounding volumes. The bounding volume of a parent segment is the union of the bounding volumes of its children segments, plus the bounding volume of the parent segment's geometry.

A few attributes, such as normal vectors, do not inherit at all. The HOOPS/3dGS reference manual entry for each type of attribute includes a discussion of any non-standard inheritance characteristics for that attribute.

3.3.3 Attribute Lock

It is possible to override local attributes temporarily, using attribute lock. When a specific attribute is locked in a segment, that attribute's value applies to all sub-segments, regardless of whether those sub-segments have local attribute values set for that same attribute. For example, attribute lock can be used to highlight a part of the database by changing the color of everything to red, temporarily ignoring any local color attributes.



3.3.4 Style Segments

HOOPS/3dGS provides a convenient way to take a set of attributes and make them apply to a large number of segments. You can create a segment called a style segment. Then, you can make other segments reference the style segment, thereby causing the other segments to take on the attributes of the style segment. Attributes that are “styled” act like local attributes.

3.3.5 Attribute Inheritance Summary

Rendering the database consists of walking the database (segment tree), constructing primitives, and rendering these primitives. From the viewpoint of the renderer, a segment consists of geometry and net attributes. “Net attribute” refers to the fact that an attribute’s value is determined based on (in order of priority, from 1 to 3) its local value, its style segment value, or its inherited value.

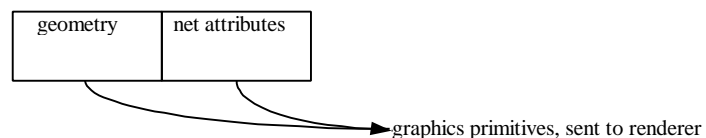


Figure 5: A segment From the Viewpoint of the Renderer

The geometry along with the values of the net attributes are drawn by the renderer.

From the viewpoint of the database tree walker, the net attributes of a segment come from three places: locally set attributes, style segments, and parent segments. (Note: This discussion applies only to attributes that inherit normally.)

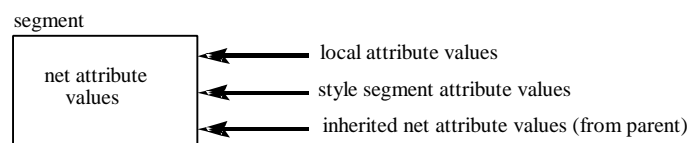


Figure 6: A Segment From the Viewpoint of the Database Tree Walker

In this diagram, local attribute values are attributes that are explicitly set on a segment using one of the HOOPS/3dGS Set routines. Style segment attribute values come from the local attributes of a segment that is used to “style” the current segment using the **Style_Segment** routine. Inherited net attribute values (typically) come from the net attributes of the parent segment of this segment.



4 Geometry and Attributes

In HOOPS/3dGS, geometry is the raw geometric information (such as points or lines) that defines the visual components of the picture. Segments are containers for geometry and attributes. Attributes are used to specify how the geometry stored in the database should be rendered and queried.

4.1 Geometry Provided in HOOPS/3dGS

The HOOPS/3dGS interface contains entry points for each geometric primitive in the form: Insert_XXX, where XXX is the name of the primitive (for example, Insert_Line, Insert_Mesh, or Insert_Distant_Light.) The geometric entities supported by HOOPS/3dGS have been specifically tailored for the needs of the MCAD/CAM/CAE software industry.

All geometric entities in HOOPS/3dGS are represented as flat, infinitely thin surfaces— i.e., there are no solid objects in HOOPS. In order to create an object that appears to be solid, you define the collection of its infinitely thin surfaces. Markers, text, and images differ slightly in that markers and text are defined by one 3D point and have no real surfaces, while images have a 3D anchor point and a 2D pixel array defined in screen space, which is mapped to the pixels of an output device.

HOOPS/3dGS provides the following geometry:

Circles	Markers
Circular Arcs	Meshes
Circular Wedge	Polygons
Circular Chord	Polylines
Ellipses	Shells
Elliptical Arc	
Grids	
Images	
Lines	

Once geometry has been inserted into HOOPS/3dGS segments, it can be copied, deleted, modified, or manipulated. The position of geometry in the world coordinate system is governed by the information given on insertion and the net modeling matrix in effect on the geometry's containing segment.

The following entities, while not traditional geometric entities, behave similarly in that their initial positions are transformed by their segment's net modeling transformation.

Cutting Planes
Lights



Everything that is inserted into the HOOPS/3dGS database (for example, the segment tree, all geometric primitives, all attribute settings) may be queried. Segment level attributes are available for controlling a geometric entities visual rendition and its selectability. Each polygonal entity (circles, ellipses, grids, meshes, polygons, and shells) is composed of a face and an edge. Finer granularity of visual and selection attributes is available for all the components of these entities. Thus, the color of a meshes' faces can be different from its edges.

Circles

A circle is defined by any three points on its circumference. Circles are grouped with ellipses, polygons, shells, and meshes for rendering purposes. Their rendition may be adjusted with the attributes for faces and edges.

Circular Arcs

A circular arc is defined by three points on the circumference of a circle. The order of the points is important. The arc begins at the first point, draws through the second, and continues to the third. Circular arcs are grouped with lines, polylines, and elliptical arcs for rendering purposes.

Circular Chords

A circular chord is defined by three points on the circumference of a circle. The order of the points is important. The wedge begins at the first point, draws through the second point, and continues through the third point. The center of a circle defined by the three points is computed and lines are drawn from the first point to the center and from the third point to the center.

Circular Wedges

A circular wedge is defined by three points on the circumference of a circle. The order of the points is important. The chord begins at the first point, draws through the second point, and continues through the third point. A line connects the first and third points.

Cutting Planes

A cutting plane "cuts away" part of the scene. In particular, if you consider the $[a,b,c]$ portion of a plane equation as the normal vector for the plane, then that half-space by convention is left open, i.e., it is not cut away. Multiple cuttings planes can be inserted in the segment tree.

Ellipses

An ellipse is defined by three points: its center point, and intersection points with its major and minor axes. It is grouped with the set of polygonal entities for rendering purposes.

Elliptical Arcs

An elliptical arc is defined by the same information for an ellipse, plus two floating point numbers between 0 and 1 which indicate the normalized



parametric angle along the perimeter of the ellipse where the arc begins and ends.

Grids

A grid is a flat array of faces, edges, and markers. It can be quadrilateral or radial, finite or infinite.

Images

An image is intended for the display of raster arrays of pixels. An image is unique in that it is laid out in terms of screen space; once the center has been located in 3D space, all other parameters (format, height and width) are defined in terms of pixels on the screen of the current display device. HOOPS/3dGS supports the following image formats:

- RGB - 24 bit truecolor
- Mapped 16 - 16 bit indices into a HOOPS/3dGS colormap
- Mapped 8 – 8 bit indices into a HOOPS/3dGS colormap

Lights

HOOPS/3dGS supports distant, spot, and point sources of light. Multiple lights can be inserted, and each light can have a unique color.

HOOPS/3dGS performs lighting calculations for all the objects in the scene that contain a light. The positions of each type of light can be manipulated with modeling transformations; spot lights can be configured to follow a camera as it is manipulated.

Lines

A line is a segment with two endpoints.

Markers

A marker is a single location in space. The location is drawn with a symbol such as an X, a dot, or a circle. Markers are commonly used to mark data points in a graph, or locations on a map. HOOPS/3dGS directly supports over 30 marker symbols commonly used in the MCAD and AEC industries, and has mechanisms for adding user-defined symbols via HOOPS/3dGS Intermediate Mode.

Meshes

A mesh is a two-dimensional array of 3D points with fixed topology. A mesh is like a rectangular wire screen— it can be bent into an arbitrary curved surface but the points in the mesh are connected into quadrilaterals. Meshes and shells are the only primitives in HOOPS/3dGS with the connectivity information for adjoining faces enabling the phong and gouraud lighting interpolation methods to be used. For more information on these properties, see page 20.



Polygons

A polygon is a flat, infinitely thin surface in space and is defined by the vertices along its perimeter. A polygon consists of two separate parts: the edge and the face. Attributes such as face pattern, edge weight, edge pattern, visibility and color can be set on these parts independently.

Polylines

A polyline is a sequence of connected line segments and is defined by its set of vertices.

Shells

A shell is an arbitrary collection of polygons that forms a three-dimensional object. A shell can represent 1) points (like a scatter plot), 2) edges (a series of lines), or 3) polygons. A shell that represents a polygon would be an array of points and a connectivity list that describes the polygonal faces. A useful way to visualize these two components is as a point cloud and a collection of "connect-the-dots" sequences in the point cloud.

A shell consists of one or more polygonal faces and is typically used to represent a wide range of geometric objects such as cubes, spheres and parametric surfaces. The advantage of using a shell, rather than multiple independent polygons, is that the shell takes up less memory in the database, is faster to render, and can be smoothly shaded.

Shell and mesh faces and have the same attributes as polygons. They also have several features that go beyond the capabilities of polygons:

- **Vertex markers**— The vertices of a shell/mesh are represented as markers which are subject to normal marker attributes. You can change the marker symbol and size for these markers, and you can turn the markers off with `HC_Set_Visibility("markers=off")`. For most uses of shells, it is common to turn the visibility of markers off.
- **Attributes on subparts**— Unlike other kinds geometry, attributes can be set on individual faces, edges, and vertices within a shell or mesh. This allows individual faces (and edges and vertices) to have completely different attributes from each other. This is useful when visualizing data sets with more than four dimensions.
- **Smooth shading**— Because vertices can be shared between shell and mesh faces, these primitives can be smooth shaded using Gouraud and Phong lighting interpolation algorithms. Lighting interpolation requires HOOPS/3dGS to have a normal vector for each vertex in the shell or mesh. HOOPS/3dGS can automatically compute normal vectors for each vertex, or they can be defined by the user.
- **Data mapping**— In addition to performing lighting interpolation (smooth shading), colors can be interpolated across a shell or mesh. This is used to visualize data associated with the surface. For example, color may be used to indicate temperature distribution across a surface. The method for achieving this consists of associating either floating point or integer scalar values with the vertices of a mesh or shell then using these values to determine color values for the



surface at the vertices by using them as indices into a colormap. Color values can be interpolated so that the color changes smoothly across a face, or color indices can be interpolated with the result being contour bands.

Text

Text is inserted as a string with a 3D coordinate as a reference point. Text can be rendered in screen space or as fully transformable strokes. Font, size, slant, rotation and alignment can be individually controlled.

HOOPS/3dGS provides direct support for the Japanese Kanji and ISO-Latin character sets.

HOOPS/3dGS includes an embedded font engine and routines that enable standard fonts to be converted to 3D surfaces. The font engine enables HOOPS/3dGS-based applications to incorporate Adobe Type 1 (PostScript), TrueType and BitStream's Speedo fonts in a scene by simply setting the text font name attribute. This conversion also enables HOOPS/3dGS rendering techniques such as texture mapping and lighting to be applied to the geometry generated from the fonts.

4.2 Attributes

Attributes in HOOPS/3dGS are used to specify how the geometry stored in the database should be rendered and queried. There are some non-graphical attributes as well, most of which help the system attach to and configure output and input devices. The main types of attributes in HOOPS/3dGS are:

- Display
- Modeling and Viewing
- Rendering
- Selection
- Driver Options
- System Options
- Heuristics

Note: This is by no means a complete listing. For a complete list of all attributes, see the HOOPS/3dGS Reference Manual or the book, 3D with HOOPS, from Addison-Wesley.

4.2.1 Display Attributes

Display attributes specify how different types of geometry should be drawn; what color to use for polygons, what edge pattern to use on lines, face patterns for meshes, symbols for markers, and so on. They are segment level attributes. Each geometric type in the same segment can, and often does, have its own setting in the same segment. For example, the lines in a same segment may have a different color attribute from the text.



4.2.1.1 Color and Colormaps

The color of geometry in HOOPS/3dGS may be specified with one of four color space models: RGB, HLS, HIC, and HSV. With regards to the HIC color space model, HOOPS/3dGS also defines the color names of the Crayola® 64 color set, so textual names such as 'orange' and 'bright blue' may be used.

A colormap attribute may be defined and color values given as indices into the colormap. Colormaps may be defined in any of the 4 color space formats, or as Crayola names. The user may elect to define his own set of color names defined in any of the available color space models.

HOOPS/3dGS supports monochrome, mapped (i.e. 8-bit) and true color (i.e. 24-bit) devices by using a 24-bit color model internally and automatically dithering colors for mapped devices via a dither cube. The size and shape of the dither cube HOOPS/3dGS uses are configurable.

4.2.1.2 Visibility

Often, it is useful to control whether geometry and/or segments are drawn without affecting the overall structure of the segment hierarchy. The visibility attribute provides this level of control. Segments and geometry whose visibility attribute is set to "off" are not drawn during an update cycle. You can control many elements independently, such as interior silhouettes (edges), perimeters, and mesh quads.

4.2.1.3 Edge Pattern and Weight

There are separate attributes for controlling the width and pattern of the edges of polygons, circles, ellipses, shells, and meshes.

4.2.1.4 Face Pattern

Face pattern enables patterns to be applied to the faces of polygons, circles, ellipses, meshes and shells.

4.2.2 Modeling & Viewing

The Modeling and Viewing attributes in HOOPS/3dGS define what geometry is being viewed in the segment tree and how it is mapped to a windowing system window. HOOPS/3dGS employs the paradigm of cameras and windows for the viewing and modeling transformations.

Cameras exist in the same world space as the geometry and can 'see' the geometry in the segment tree. The contents of their field of view is then mapped into a HOOPS/3dGS window. HOOPS/3dGS windows are subregions of a root window corresponding to an instance of a driver segment which is attached to an output device. Thus, all windows below the driver segment are subregions of the output device window or printer page, and their contents are mapped to the output device.



4.2.2.1 Modeling

Modeling attributes provide for the scaling, rotating and translating of the geometry contained in a segment and, due to attribute inheritance, its sub-segments. If the order in which successive modeling transformations are applied to geometry is changed, the resulting net transformations will be different. (Modeling transformations are non-commutative operands.) Thus, these attributes are the only ones in HOOPS/3dGS that are order-dependent.

Each segment in HOOPS/3dGS has a local 3D space and geometry inserted into a segment is said to be in that segment's object space coordinate system. Any transformation attributes set on the segment will modify the position of the geometry in the segment. Since modeling transformations are attributes, the final position of the geometry depends on the modeling transforms set locally in its containing segment *and of all the modeling attributes set in the line of segments owning it.*

HOOPS/3dGS provides for either right or left-handed coordinate systems to be used on a per-segment basis. The same segment hierarchy may have multiple instances with different handedness settings.

4.2.2.2 Viewing with Cameras

HOOPS/3dGS uses the paradigm of cameras to define views into a scene. Cameras have a position, target, up vector, field of view and a projection. Supported camera projections include perspective, orthographic, and oblique. These values are defined with 3D coordinates in world space.

HOOPS/3dGS cameras may be manipulated in the same way as their real-world counterparts with routines that dolly, roll, zoom, pan, rotate and orbit. As cameras are segment-level attributes, it is quite possible to have multiple cameras defined under the same driver level segment. The geometry visible in each camera's view will then be projected into the containing (net) window. This may be an attribute of the same segment as the camera or a parent segment. This is a useful technique for generating stereo effects for added depth perception.

The needs of most applications are met by relying on the default camera associated with a driver instance segment and its associated HOOPS/3dGS window.

4.2.2.3 Windows

Window attributes are segment-level attributes and therefore inherit. The visual effect is that of nested sub-regions. Window attributes of child segments are defined as sub-regions of the first window found above them in the hierarchy. The top level window is the one at the driver instance segment and it maps to the extents of the output device. While infinite levels of window nesting are possible, the needs of most applications are met with two levels of window nesting.



4.2.3 Rendering Attributes

Rendering attributes are used to select algorithms for calculating hidden surfaces, lighting effects, the mapping of associated data to geometry (e.g., stress-strain analysis results mapped to a mechanical parts geometry) as well as the level of detail (LOD) to be used when displaying shells and meshes

HOOPS/3dGS' rendering functionality supports transparency, texture mapping, atmospheric attenuation (depth-cueing), color contouring, and dynamic LOD switching.

4.2.3.1 Hidden Surface Removal Algorithms

HOOPS/3dGS supports the following: hardware z-buffer, software z-buffer, extended painter's algorithm, quick painter's algorithm, object-space hidden line removal and wireframe.

4.2.3.2 Lighting Models

HOOPS/3dGS provides flat (faceted), gouraud (smooth), phong lighting techniques; and both diffuse and specular reflections. Gouraud and phong shading are only applicable to shells and meshes.

4.2.3.3 Data Mapping

Often, it is desirable to map data associated with a surface directly on to that surface, particularly in the CAE areas of Finite Element Analysis (FEA) and Computational Fluid Dynamics (CFD). HOOPS/3dGS provides the capability of mapping scalar information (integer or floating values) directly onto surface topology and then using this information to calculate color contours across the surface.

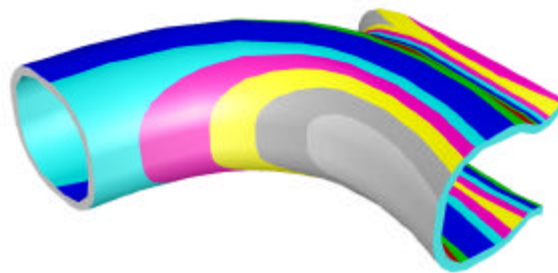


Figure 8: Mapping Analysis Data onto the Surface of a Model

4.2.3.4 Level of Detail (LOD)

The rendering performance of large models can be improved by generating multiple versions of the model's geometry, each with smaller amounts of data or *levels of detail* (LOD), and then dynamically choosing which version to render based on different criteria. For example, multiple LODs for a shell or mesh can be generated each with a smaller number of triangles using a vertex decimation technique and then those LODs can be chosen for rendering as the user manipulates the camera viewing the



model; LODs with more detail are rendered when closer to the model and LODs with less detail used when further away.

The HOOPS/3dGS LOD module provides for the calculation of LODs for shells and meshes as well as enabling the user to supply their own LODs. It also provides for dynamic selection of these LODs based on several different types of threshold calculations.

4.2.4 Selection

The selection attribute enables control over how the objects in the HOOPS/3dGS database respond to a selection query or hit-test. The application will often need to request that HOOPS/3dGS find the drawing primitive or segment currently being pointed at by an input device, usually a mouse or pen.

Selection settings may be given for the entire contents of a segment or set specifically for certain geometric types within the same segment. For example, to implement “snap-to-grid” behavior, one could insert a grid into a segment and then set the selection attribute for everything except markers to be ‘off’.

4.2.5 Driver Options

These attributes are used to provide information to the HOOPS/3dGS graphics system of special device-specific display options such as

- Double-buffering
- Input event queue control
- Number of colors on the device
- Configure the dither cube for lighting calculations
- Landscape or Portrait orientation
- Pen Speed (printers)
- Physical Size
- Hardware Colormap to use, if desired
- Window Handle to use, if desired

4.2.6 System Options

System Options are used to configure aspects of the entire HOOPS 3D Graphics System. For example, you can set the length of a C string, or configure the error reporting in HOOPS/3dGS.



4.2.7 Heuristics

Heuristics are hints to give to HOOPS/3dGS to help it make traversal time decisions about how to optimally render the segment tree. By telling HOOPS/3dGS whether or not to use the following information or techniques, the performance of the system may be tuned to a given applications graphics data.

- **Backplane culling**
- **Clipping**
- **Concave Polygons**
- **Hidden Surfaces**
- **Incremental Updates**
- **Intersecting Polygons**
- **Memory Purge**
- **Polygon Handedness**
- **Quick Moves**



5 Input and Hit-testing

The HOOPS/3dGS architecture includes a mechanism for monitoring input devices for user-generated events. This can be used to build a user interface with HOOPS/3dGS and can be quite useful in the area of rapid-prototyping. More often than not, in commercial applications, this feature is disabled and the application directly controls the event loop, processing all the messages and dispatching them to the appropriate component sub-systems. In this case, events associated with geometry selection (hit-testing) need to be passed to HOOPS/3dGS.

When the application needs to know what geometry the end-user is selecting, HOOPS/3dGS performs a “hit-test” of the input events coordinates against the geometry in the HOOPS/3dGS database.

HOOPS/3dGS provides a routine called `Compute_Selection` which, given a window location, performs a 3D object space intersection test and returns any selected objects. Figure 9 shows the path of a selection event as it starts out as a location event from the window system. The application receives the location event and calls `Compute_Selection` with the location, which does a hit test against the 3D objects in the database and creates a selection event.

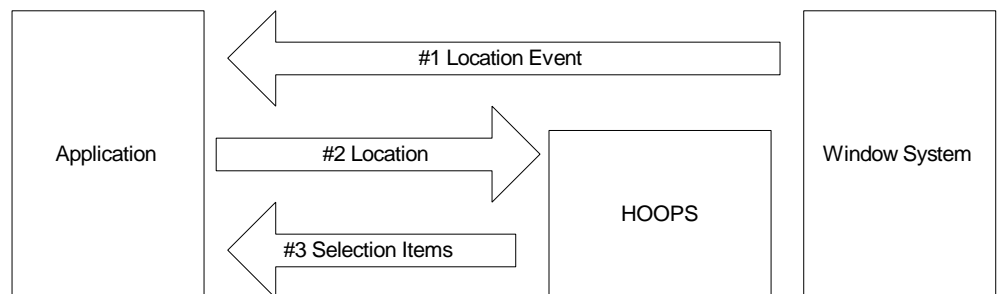


Figure 9: Flow of selection event from window system to HOOPS

White Paper



HOOPS/3dGS supports the following selection methods:

- **Aperture**^¾ A point location. Aperture is defined by a radial distance from the selection point.
- **Area**^¾ A rectangular region
- **Polygon** (lasso) ^¾ A polygonal region
- **Polyline** (fence) ^¾ A series of points connected by line segments
- **Volume**^¾ A 3D object space volume
HOOPS/3dGS also supports a range of picking granularity *within* selected objects:
- **Vertex**^¾ closest definition point
- **Edge**^¾ closest polygonal border
- **Face**^¾ closest polygonal facet
- **Object**^¾ segment containing the selected geometry
HOOPS/3dGS will also compute the analytical point of intersection with the selected geometry in object, world and camera spaces.



6 Integrating HOOPS/3dGS with a GUI Toolkit

Just as HOOPS/3dGS is a component for graphics, there are several components that provide graphical user interface (GUI) technology. Typically, developers want to use HOOPS/3dGS with such GUI components as MFC on Windows, MOTIF on UNIX, or Qt and JAVA for cross-platform GUI.

6.1 Window System Integration

When writing a 3D HOOPS-based graphics application, the developer has a choice of writing the user interface in HOOPS/3dGS or in the target platform's native window system. In most cases, it is desirable to write the user interface using a window system. This allows the application to have a native 'look and feel' and allows the graphics system to remain independent of the event queue and user interface. HOOPS/3dGS has been successfully and seamlessly integrated with MFC, MOTIF, QT, Java, and ActiveX GUI tools. This work is encapsulated into the HOOPS/GUI modules and included as part of the HOOPS 3D Application Framework.

6.2 Drawing into Native Application Windows

Normally, HOOPS/3dGS creates its own output window for each driver instance. When the user interface is written using a window system, the window system is in charge of creating the output window. To address this need, HOOPS/3dGS is able to accept a pre-created window to draw into. The application simply passes the window handle to an instance of a HOOPS/3dGS driver level segment.

6.3 Colormap Sharing

HOOPS/3dGS provides mechanisms that allow you to share colormaps among multiple driver instances and applications.



7 HOOPS/3dGS Intermediate Mode

HOOPS/3dGS Intermediate Mode is a secondary interface layer in the library and provides access to the immediate mode routines used in the HOOPS/3dGS Structured Device Interface layer. HOOPS/3dGS Intermediate Mode complements the classic HOOPS/3dGS library and provides for traversal-time modifications of the object hierarchy (segment tree). The diagram below illustrates the relationship between HOOPS/3dGS' database API, the HC interface, and the immediate mode API, the HIC Interface.

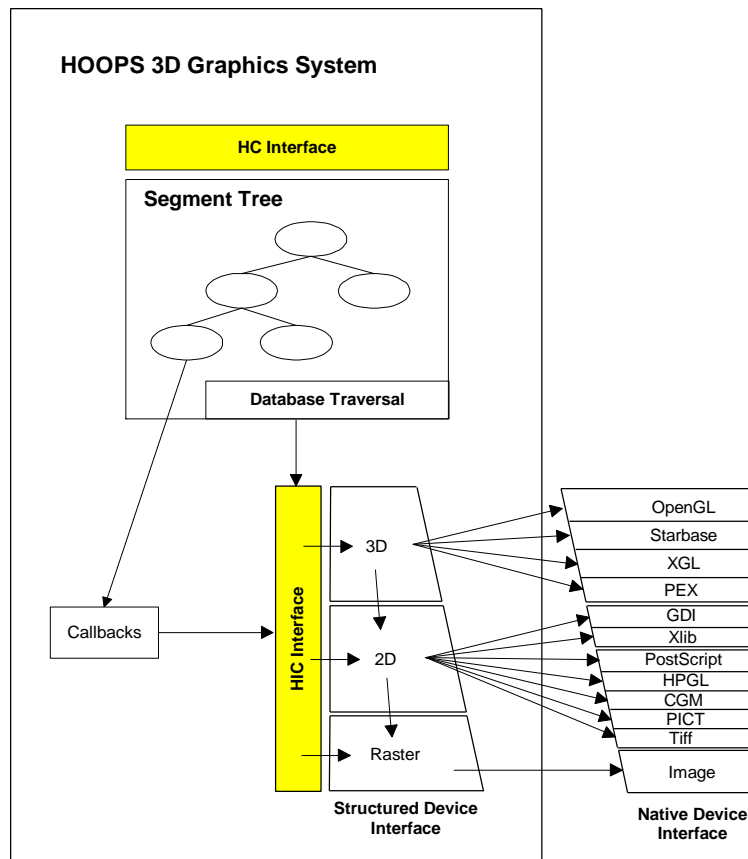


Figure 10: The Relationship between HOOPS/3dGS' HC and HIC APIs.

HOOPS/3dGS is a retained graphics database system. You create a scene by inserting geometry and setting attributes in a segment tree maintained by HOOPS/3dGS. When you call `Update_Display` (or you request input), the system traverses the segment tree and draws the picture on the display device.

White Paper



With the classic HOOPS/3dGS library, additions and modifications to the segment tree cannot be made while HOOPS/3dGS is traversing the tree. The Intermediate Mode library provides a means for the application to trap the HOOPS/3dGS update cycle at certain points in the rendering pipeline by means of a callback mechanism. When the traversal is trapped at a callback point, decisions can be made about what and how something is drawn, or even the traversal process itself can be aborted. In addition to the callback mechanism, the HOOPS/3dGS Intermediate Mode library provides a set of functions that can be called from the callback functions to draw to the display in an “immediate mode” style and to query the graphics database and the device characteristics.

The HOOPS/3dGS selection feature also involves a traversal of the graphics database contained in the segment tree. However, the selection traversal does not draw on the display. Rather, it computes the screen positions of objects in the database to determine which objects have been hit by selection events. HOOPS/3dGS Intermediate Mode provides callback points at which the selection traversal as well as the update traversal can be trapped.



7.1 Why use HOOPS/3dGS Intermediate Mode?

HOOPS/3dGS Intermediate Mode is useful when it is necessary for the application to be able to make traversal-time decisions about the rendering or selection process, or to accomplish special processing that is not provided by the built-in HOOPS/3dGS traversal process.

Some examples of situations in which to use HOOPS/3dGS Intermediate Mode are:

- You may want the graphical representation of your data, i.e. the actual primitives and attribute values used, to depend on the viewing parameters or the screen transformation. In particular, if your model has a hierarchy of scale and the view is zoomed out sufficiently far, then you may want to skip the rendering of entire subtrees that would appear very small on the display.
- You may want to define your own version of a HOOPS primitive. For example, you could implement a spline drawing algorithm through the HOOPS polyline primitive, using the polyline vertices as spline control points. In this case, you would intercept at one of the callback points in the polyline drawing pipeline, and substitute your own spline drawing routine for the HOOPS polyline drawing routine.
- Custom marker symbols or line styles are required, and need to be stroked out at traversal-time drawing. In addition, the rendering style may need to be dependent on actual screen size. For example, in a cartographic application railway tracks could be stored as polylines in the database, but be drawn, for certain map scales, as parallel lines with cross ties.
- If the graphics database is voluminous, it may be useful for the application to avoid spending the memory needed for HOOPS/3dGS to duplicate in its database some of the same information already contained in the application's private data structures. Using HOOPS/3dGS Intermediate Mode in an "immediate mode" style allows primitives to be passed to HOOPS/3dGS one at a time in Intermediate Mode callbacks, rather than storing them in HOOPS/3dGS segments. However, since HOOPS/3dGS no longer "knows about" such primitives, it cannot be used to manage and select them.
- In a real-time application, it may be necessary to be able to modify the picture being displayed according to input received during traversal.



8 Appendix A: Immediate Versus Retained Mode Graphics Systems

Software applications are run on computer hardware. Hardware provides input and output mechanisms for the user of the software application. It is through these input and output mechanisms that the user submits information and requests the application to perform operations on this information.

Input mechanisms can include the keyboard, mouse, data gloves, body suits, or head-mounted displays. The application supplies information on the state of data via various output devices. Output devices may include monitors, printers, haptic display systems, or CAVE immersive projection systems. Interactive computer applications are concerned with the relationship between the mouse as input and monitor as output.

In order for the application to display information on the monitor, there must be a way for the application to communicate with the monitor. For instance, if the application wants to draw a line on the screen, there must be some mechanism that enables the application to ask that a line be drawn. This mechanism is the graphics system. Figure A-1 shows the relationship between an application and a graphics system.

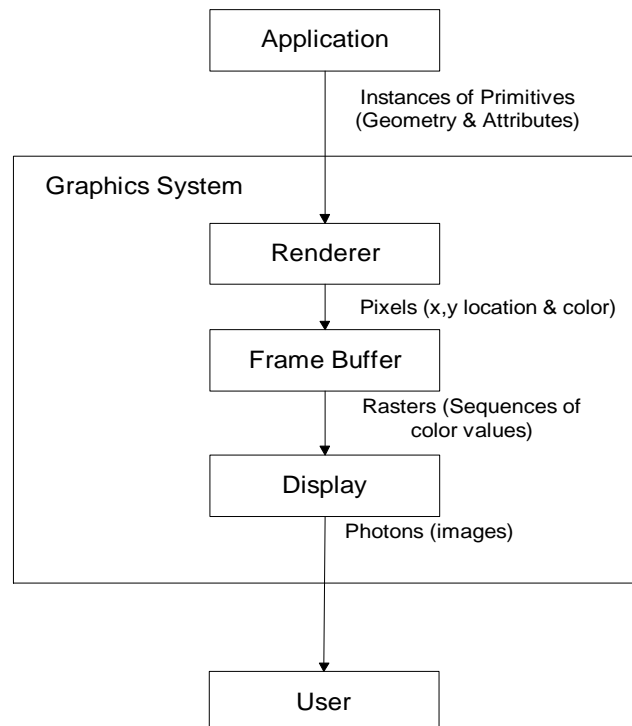


Figure A-1: Relationship Between an Application and a Graphics System



Graphics systems that render primitives immediately, without storing them in a display list, are called **immediate mode** systems. Graphics systems that retain the primitives in a display list or graphics database are called **retained mode** systems.

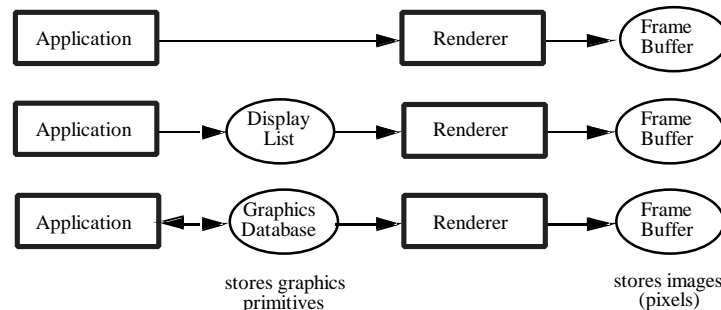


Figure A-2: Immediate Mode versus Retained Mode Graphics Systems

Immediate Mode Systems

Each hardware platform supplies rendering or device interface libraries for displaying graphics on the monitor. These libraries retain no information about what they have drawn and are often called *immediate mode* libraries because they attempt to display requested information immediately upon receipt of a drawing request. Examples are Xlib on Unix machines and GDI on Intel PC's running MS Windows. Specific hardware vendors often provide their own proprietary interfaces tuned specifically for their platform. Examples include OpenGL on Silicon Graphics workstations, Starbase on Hewlett-Packard machines, and XGL on Sun Microsystem computers.

Because the immediate mode libraries retain no information about what they have drawn, the application programmer must create data structures for storing graphical information and develop algorithms for deciding what to draw and how to draw the desired scene. Ideally, the data structures used by the rest of the application would be used to drive the immediate mode graphics library. However, the application's primary focus is something other than the display of graphics, and the data structures employed for the core application logic often are not sufficient for the optimal display of graphics information.

Retained Mode Systems

Systems that store graphics information in data structures specifically designed for the display of graphics are called *retained mode* graphics libraries. There are two types of retained mode systems: display lists and databases. The difference between a display list and a graphics database is that a graphics database is a display list that can be modified in place. For example, for a scene containing several dozen primitives, a graphics database allows you to change the primitives individually (either their geometry or attributes), while a simple display list would require you to resend the entire display list (that is, redraw the entire scene) in order to change one primitive.



Systems that group the algorithms necessary to create and manipulate the graphical information along with the data structures for storing this information are called graphics systems or graphical object stores. HOOPS/3dGS is a retained-mode graphics library. Retaining the graphics primitives in a retained mode system provides significant benefits:

- **Performance**¼ If the graphics system (rather than the application) retains a copy of all the primitives, it can perform faster modification and traversal of the graphics information as well as optimizations to make rendering the scene faster. For example, the graphics system can calculate bounding volumes that store the location and a rough measure of the extent of a set of primitives. During rendering, many primitives may not be visible on the screen. Bounding volumes allow the system to quickly determine whether a group of primitives is on-screen so that only those graphics primitives that are actually visible are sent to the display hardware to be rendered.
- **Selection**¼ An important function of a graphics system is performing selections (also called “picking”), where the graphics system determines which graphic primitive the user is pointing at with the cursor. Without a display list, the only way for the graphics system to perform a selection is to have the application resend all the primitives since they were not retained by the graphics system. A display list allows the graphics system to perform selection much faster, typically an order of magnitude faster.
- **Window system support**— If a window containing a scene is partially obscured by another window and then it is brought to the front, the newly-exposed areas of the window need to be redrawn. If the graphics system stores the primitives in a display list, then the window can be redrawn without going back to the application. (The window’s image can also be stored as a bitmap, allowing instant redraws of damaged areas. HOOPS/3dGS supports this method as well.)
- **Global rendering algorithms**— Many kinds of renderers require a copy of all the graphics primitives. For example, for ray tracing and radiosity, the color of an object is affected by other objects, so the renderer requires data on all the primitives before it can start drawing any of them. Such advanced rendering algorithms require the use of a display list to store the primitives.

Which Mode to Use?

Whether or not a graphics system has a graphics database (*retained mode*) greatly affects how the application interacts with the graphics system. Recall that OpenGL, Starbase and GDI are examples of “immediate mode” rendering pipelines; as such, they provide routines for an application to communicate what should be drawn on the screen, but they do not remember what has been drawn. The application needs to store this information itself, compute the effects of user interaction (e.g. mouse



movement) on the data and then procedurally communicate the new set of information that represents the current state of the 3D information. Use of such an immediate mode system requires the application developer to create data structures for storage and implement algorithms for traversal, rendering and selection of the graphics objects.

Which technique is preferable depends on how much of the data model changes at a time. If the entire data model changes often, then it might be just as easy to resend the entire view each time the data model changes. If the data model is normally changed incrementally, then a retained graphics system can be much faster. Using a graphics database such as HOOPS/3dGS to store the primitives can save time and effort for the application developer. Like any component, however, using a graphics database will only save time if it matches the needs of the application and is easy to use.