



## The Added Value of HOOPS®

Tech Soft 3D  
931 Ashby Ave.  
Berkeley, CA 94710

Ph: 510.883.2180  
[www.techsoft3d.com](http://www.techsoft3d.com)  
[info@techsoft3d.com](mailto:info@techsoft3d.com)

1.0 Introduction .....	3
2.0 Immediate-Mode 3D APIs .....	3
3.0 The HOOPS 3D Application Framework.....	3
4.0 The Added Value of HOOPS/3dAF over Immediate Mode APIs .....	6
4.1 Primitive Set .....	6
4.2 Retained-Mode Functionality.....	7
4.3 The Scalable Device Interface.....	8
4.4 Hidden Line Removal and Sorting Algorithms.....	9
4.5 Extensive Text Support .....	10
4.6 Hardcopy Output .....	11
4.7 Selection.....	12
4.8 Flexible Rendering.....	13
4.9 Level-of-Detail .....	13
4.10 Optimized Performance.....	15
4.11 Immediate Mode with Context: HOOPS Intermediate Mode .....	15
4.12 HOOPS/MVO .....	16
4.13 HOOPS/Stream.....	17
4.14 HOOPS/GM.....	18
4.15 HOOPS/Net Server .....	18
5.0 Technical Support and Consulting Services .....	19
6.0 Summary.....	19

## 1.0 Introduction

This paper begins with a general description of the [HOOPS 3D Application Framework](#) (HOOPS/3dAF) and its relationship to lower-level graphics devices interfaces. It then reviews some of the specific areas of functionality found in HOOPS that a developer would otherwise need to implement if they choose to write a proprietary graphics subsystem on top of an immediate mode interface like OpenGL or Direct3D. Implementation details and development effort estimates of key features are included in order to give some idea of the work involved with internally developing such functionality.

## 2.0 Immediate-Mode 3D APIs

[OpenGL](#) is a specification for an immediate mode 3D graphics library based on streaming, and is currently the most popular 3D device interface. Microsoft has a competing specification called [Direct3D](#). While OpenGL will continue to exist on Windows for at least the next few years and will maintain its leadership position on non-Windows platforms for some time, it is likely that Direct3D will gradually become a sound, and perhaps superior alternative for engineering applications running on the Windows platform.

As immediate mode libraries, OpenGL and Direct3D do not provide any of the higher level functionality that HOOPS/3dAF encapsulates. While developers can create professional grade applications by writing directly to OpenGL and Direct3D, it requires significantly more time, effort and resources. Later in this paper we will discuss just some of the capabilities of HOOPS technology which a developer would need to reinvent on their own in order to develop a full-featured graphics subsystem.

## 3.0 The HOOPS 3D Application Framework

HOOPS/3dAF is a high-level framework specifically geared toward the rapid development of engineering software applications. It is developed by TS3D and distributed and supported by TS3D and Spatial worldwide. It is provided as a set of libraries and is available on [Windows, all major UNIX platforms, Linux and Mac OS X](#). Source code to many of the sub-components is provided.

The HOOPS 3D Graphics System (HOOPS/3dGS) sits at the core of the framework, and provides for the creation, storage, manipulation, querying and rendering of 2D & 3D graphical information. In addition to core HOOPS/3dGS, the framework includes an extensive library of common engineering operator objects and application-level functionality (HOOPS/MVO), integrations with numerous UI toolkits (HOOPS/GUI) and bridges to tightly link HOOPS/3dAF with popular modeling kernels (HOOPS/GM). Licensees also receive numerous source-code readable sample applications which further jumpstart the development of their HOOPS/3dAF-based applications.

Fundamentally HOOPS/3dGS consists of two main parts: the object database (an acyclic, directed scene-graph) and the structured device interface, as illustrated below.

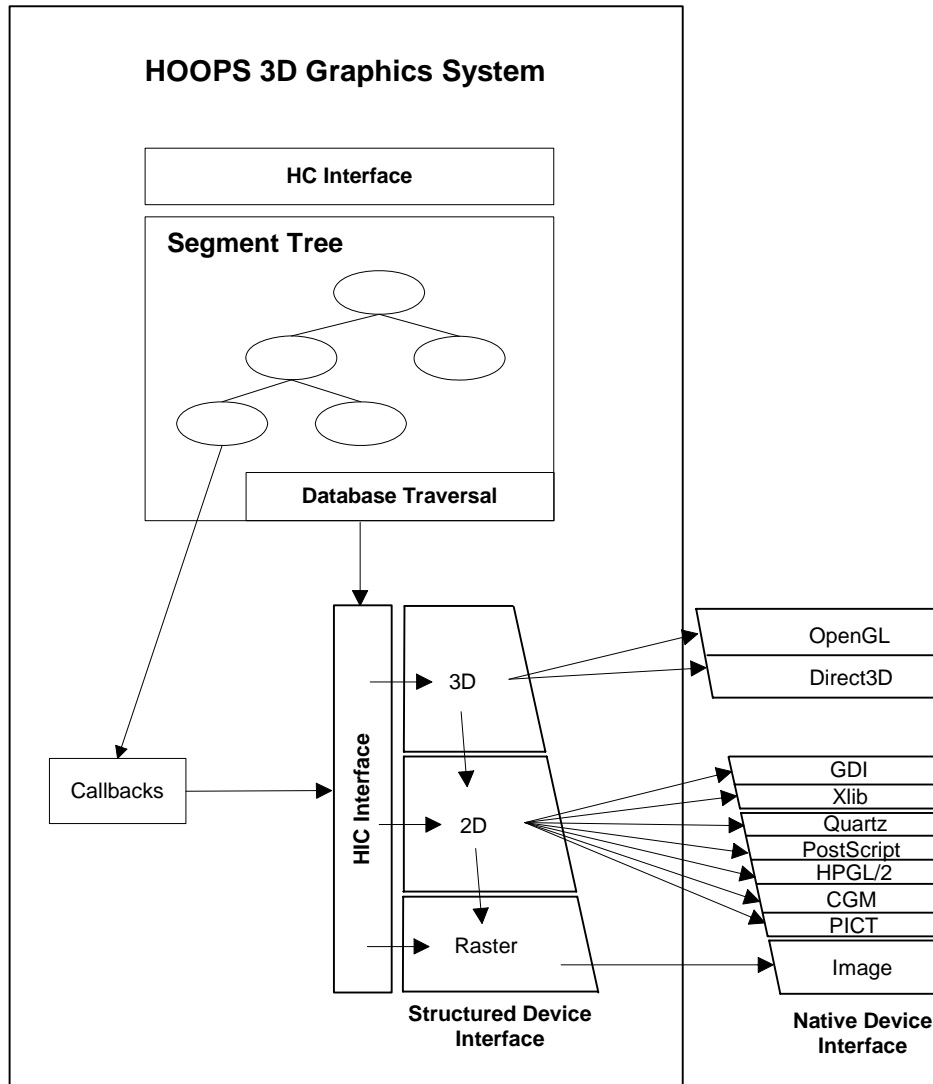


Figure 1. The HOOPS 3D Graphics System: Object Database & Structured Device Interface

The database stores graphical objects and provides methods for traversing and sending the objects to the structured device interface. This interface accepts information found in the object database, reformats it for the specific hardware output device interface and then sends it to the device for drawing. A successive decomposition technique for reformatting is used where the information in the database passes 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 or Direct3D, HOOPS/3dGS can pass along information without much change, but if the device interface only understands 2D vectors or rasters, like PostScript, GDI or HPGL, then the appropriate level of decomposition will be used.

HOOPS/3dAF contains several other modules which complement HOOPS/3dGS and provide additional functionality not found in lower-level device interfaces. These add-on modules include HOOPS/MVO, HOOPS/GM, HOOPS/Stream, and HOOPS/Net, and will be discussed in more detail later in the paper” The following diagram helps to illustrate:

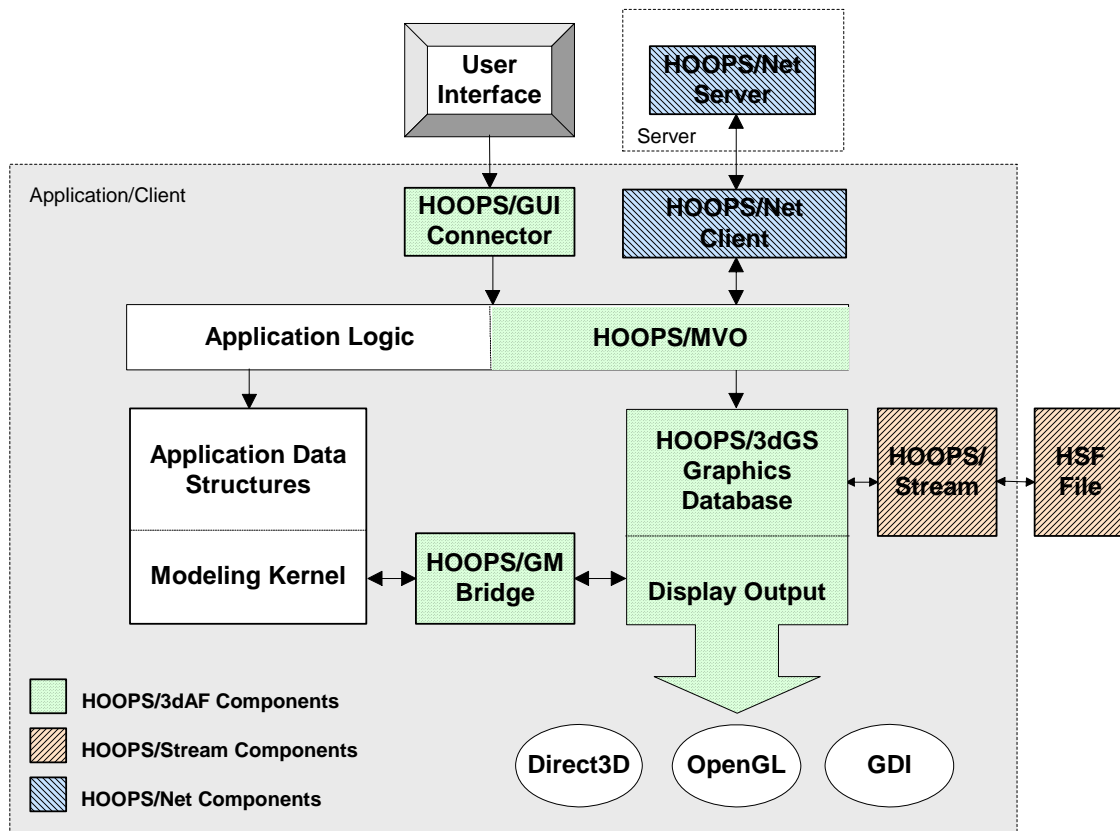


Figure 2. Typical Architecture of HOOPS/3dAF-based Application

## 4.0 The Added Value of HOOPS/3dAF over Immediate Mode APIs

Immediate mode graphics libraries such as OpenGL and Direct3D are unable to remember what has been drawn. While they provide display list or vertex buffer modes which allow the graphics information to be cached on the card, these structures cannot be queried, edited in place, or selectively traversed. Therefore, use of only an immediate-mode API requires the application developer to create data-structures for storage and implement algorithms for traversing, rendering & querying of all graphics objects. A wide variety of commonly required graphics-related logic not contained in the lower-level API must be implemented by the developer. It also requires the developer to do all the work in areas such as geometric modeler integration, UI toolkit integration, separate hardcopy implementation, and text handling, to name just a few.

The next section lists some of the specific areas of HOOPS/3dAF functionality that a developer might need to re-create if they were writing directly to an immediate-mode 3D API.

### 4.1 Primitive Set

OpenGL has a limited set of low-level primitives consisting primarily of polylines, polygons and triangle strips. Consequently, developers must provide the logic to map onto these primitives. This requires a significant amount of effort when a developer is working with non-trivial data sets, particularly if performance is an issue. Here are two examples:

If your application needs to represent N-faceted, irregular polyhedra you will need to tessellate them into a set of optimal length triangle strips to ensure maximum performance.

HOOPS/3dGS provides a rich full set of geometric primitives for the needs of CAD/CAM/CAE, Scientific Visualization and GIS applications, many of which are not directly supported by lower-level APIs. HOOPS/3dGS' decomposition logic ensures that the lower-level device capabilities will be used optimally. HOOPS/3dGS supports the following primitives:

- Circle
- Circular Arc
- Circular Wedge
- Circular Chord
- Cylinder
- Ellipse
- Elliptical Arc
- Grid

- Image
- Line
- Marker
- Mesh
- NURBS Curve
- NURBS Surface
- Polycylinder
- Polygon
- Polyline
- Shell
- Text

HOOPS also provides support for distant and spot lights, cutting planes and capping geometry (capping geometry refers to the polylines and/or polygons of intersection between the cutting planes and geometry).

## 4.2 Retained-Mode Functionality

Because immediate mode APIs do not store the graphical information in a retained manner, they cannot provide support for functionality which needs to operate on the stored data and make traversal-time decisions. A custom graphics database such as HOOPS/3dGS allows for highly optimized and scalable data traversal and rendering. The following details several examples of functionality which is built into HOOPS/3dGS:

- **Database traversal optimizations require** that the graphics system maintain bounding hierarchies and change flags, and use this information when deciding what information actually needs to be sent to the low level immediate mode API for drawing.
- **View frustum culling** involves logic to traverse and render only objects that are within the viewing area of the user (on the screen).
- **Occlusion culling** determines which objects may be hidden, or occluded, by other objects in the scene. While lower-level 3D APIs provide functions which perform an 'occlusion query' for a given object, it is up to the application-logic to choose good occluders. Efficient and effective implementation of this entire process requires that the graphics objects be placed into some sort of spatial organization, which must then be maintained and traversed.
- **Accurate rendering of transparent objects** requires sorting of transparent objects to produce correct back-to-front ordering. HOOPS/3dGS accurately renders transparent objects, and also allows the developer to control the sorting algorithm used to render the objects. The developer can also specify a fast

'screen-door' mode which simulates transparency by drawing a fraction of the transparent object's pixels to optimize performance.

- **Incremental updates** involve redrawing newly created objects into the scene, or changing the color of an object, without redrawing the entire model.
- **'Quick Moves'** is a HOOPS/3dGS term which refers to drawing/undrawing objects without redrawing the scene. HOOPS/3dGS provides several methods which can be selected by the user, including XOR drawing, overlay plane drawing and 3D Spriting.
  - Overlay plane drawing, unlike XOR drawing, allows for overlaid (rubber-banded) objects to be drawn with accurate coloring. It requires querying of widely disparate hardware capabilities
  - 3D Spriting, is a very powerful selective redrawing feature that allows objects to be accurately moved/edited in 3D. It also requires query of special hardware extensions.

These intelligent traversal-time decisions provide significant benefits, particularly when interacting with today's rapidly expanding model sizes. Creation of a dedicated database along with the types of logic described above would typically take more than a year of development from a highly skilled graphics programmer.

### **4.3 The Scalable Device Interface**

The HOOPS/3dGS Scalable Device Interface has at its foundation a full 3D software rendering pipeline, with the ability to hand off vector/raster information at the highest level that the active drawing device understands. This approach ensures both optimal rendering throughput on a given device and consistent functionality of the HOOPS/3dGS API across multiple platforms and devices. All of the functionality provided by HOOPS/3dGS and other components of the framework is portable across all the supported platforms.

#### **4.3.1 Device Drivers**

Interactive graphics applications require a separate mapping layer, or software device driver, for each low-level 2D/3D API. Consider rendering to a Postscript file a scene that contains 3D shaded objects, 3D polylines and 2D text. While the 3D shaded information could perhaps be captured by performing an OGL screen-grab, the 3D polylines must be converted to 2D and mapped to Postscript commands, along with the text. Various attributes such as line color/pattern/weight, and text color/font/size would need to be mapped to Postscript commands. Even 3D->2D transform logic and a Postscript interface would not be enough to provide a correct result, since the 2D vectors and text would have to undergo a hidden line calculation so that they were properly clipped

against the 3D information represented by the background image. (HOOPS/3dGS automatically addresses this, and is discussed in Section 4.6 below.)

Creation of a single device driver can require a month of development, with the required time scaling up depending on the level of sophistication and performance required by the application. There exist non-trivial ongoing maintenance requirements as device interface APIs are enhanced (e.g. HPGL -> HPGL/2 ; OpenGL 1.0 -> 2.0 ; GDI -> GDI+), new hardware extensions become available, and interface-specific problems or limitations are encountered and need to be worked around. Internally developed graphics subsystems typically require at least 3 device drivers. HOOPS/3dGS includes stable, optimized device drivers for all common hardcopy, 2D, 3D and raster interfaces, totaling 10 in all, which represent man-years of development.

The HOOPS/3dGS OpenGL and Direct3D drivers are tested on a wide variety of graphics cards, listed [here](#). Few companies would be able to invest resources into this degree of graphics card qualification.

#### **4.3.2 Software Rendering Pipeline**

HOOPS includes support for both concave and convex clip regions, which enable multi-view layout and circular/detail views. These are just one example of why a software rendering pipeline is necessary to support engineering application features. While an internally developed graphics subsystem could leverage an available hardware stencil buffer to enable rendering of convex clip regions (similar to the HOOPS/3dGS OpenGL driver), a full software pipeline complete with software clipping is necessary to provide robust support on 2D devices interfaces such as GDI or Postscript. For example, when rendering a convex clipped scene to Windows Print/Clipboard DC, usage of the OpenGL 'image-grab' approach is not an option unless the end-user is satisfied with a fully rasterized, blocky image. To provide a correct result the application must interface with the 2D GDI driver at its native level and a software convex clip path is required to produce an accurate and high-quality result.

Creation of a full featured, optimized software pipeline requires months of development effort by a team of programmers.

### **4.4 Hidden Line Removal and Sorting Algorithms**

Since immediate-mode 3D APIs like OpenGL and Direct3D are based on rasterization and z-buffering of primitives, they cannot supply true analytical hidden line algorithms such as those found in HOOPS. Similar to Section 4.2 above, this type of functionality must be performed via custom logic that operates on the retained graphics information. In addition to providing access to hardware z-buffering when graphics accelerators are present, HOOPS also supplies several software hidden line/hidden surface removal techniques including true analytical hidden line, fake hidden-line, z-sorting, and painter's

algorithm. A painter's algorithm is necessary to provide an accurate sorting of transparent objects, also discussed in the previous section.

End-user requirements in the manufacturing industry often call for a true analytical hidden line drawing on both the computer screen and paper hardcopy. HOOPS' true analytical hidden line algorithm provides for dynamic calculation of lines obscured from the viewer at any given vantage point. These "hidden lines" can either be not drawn or drawn with user selected attributes, e.g. a slightly dimmer or different color, a different line pattern, a different line thickness, etc.

Development of the aforementioned hidden-surface removal algorithms could take a year of development effort. The highly optimized HOOPS algorithms already encapsulate several man-years of development.

## 4.5 Extensive Text Support

Immediate-mode 3D APIs have very limited text support. HOOPS provides extensive 2D and 3D text capabilities, which allow HOOPS-based applications to easily provide professional, optimized support for text and annotations within a 2D, 3D or 2D/3D hybrid scene.

- **Robust, portable TrueType font (TTF) support** - While a platform such as Windows provides a low-level API to access TTF fonts, this does nothing to address higher-level text attributes, performance optimizations or platform/driver portability. Via an embedded font engine, HOOPS/3dGS provides portable support for True Type fonts/collections as well as OpenType and Type 1 fonts. These fonts can be drawn as bitmaps in 'screen-space' or be treated like any other piece of 3D geometry in the scene. Thus, HOOPS text is fully transformable in three dimensions and comes with all appropriate database methods such as selection, editing and attribute settings.
- **Support for Unicode character strings** - This allows developers to easily distribute international versions of their software with full support for Asian, Hebrew and other languages.
- **Access to native Windows (GDI) and Unix (X11) system bitmapped fonts** - This allows for system fonts to be integrated into the OpenGL scene, and enables developers to write portable font search/usage logic that will provide consistent results across the OpenGL, Direct3D, MSW-Display, MSW-Printer, and MSW-Clipboard drivers.
- **Highly optimized text rendering** - The bitmaps for fonts are cached and used when drawing text in 2D screen-space, and the triangles/outlines for 3D fonts are also cached and drawn when rendering text in full 3D. This support alone

requires creation and traversal of custom text displays lists, within the context of HOOPS/3dGS primary scene-graph traversal.

- **Extensive text attribute support** - The following lists just some HOOPS/3dGS' granular text capabilities, which encapsulate a significant amount of coding effort:
  - Multiline (block) text with controllable line spacing.
  - Character spacing, slant and follow-path capabilities.
  - Text greeking, which involves drawing a simple box at the text location if the text is going to be rendered below a certain size.
  - Underline, overline and strikethrough text.
  - Automatic text scaling and fallback (controlled by the user), with a variety of supported size units (pixel, pt, window relative, object relative, etc...)
  - Custom Positioning – text can be inserted to fit a certain region, and the system can be queried to return the 2D screen-space extents of a given piece of text.

Such extensive text support would require several years of development effort, making it impractical for most in-house implementations. Development of even the most basic elements of text support require several man-months of effort.

## 4.6 Hardcopy Output

Immediate-mode 3D APIs do not support hardcopy output other than the shortcut of copying an image of the screen which produces poor quality results and is frequently unacceptable by today's standards. In order to provide native hardcopy support, a developer would have to write a completely separate module to output to a hardcopy-rendering pipeline and this work would have to be replicated for each additional hardcopy format.

HOOPS/3dGS supports output of over 8 different hardcopy types including native Windows GDI Printer/Clipboard, Postscript, CGM and HPGL/2. It utilizes the native vector primitives supplied by hardcopy device interfaces and provides built-in support for hybrid vector/raster hardcopy generation. This means that rasterized parts of the scene such as textured objects will be accurately rendered using the device's raster interface, while vector parts of the scene such as lines, edges, and text are drawn using the device's native vector interface. The vector primitives are drawn using hidden-line removal, allowing them to be effectively 'interleaved' with the 3D rasterized objects (rather than just drawn on top). This allows for efficient, high-quality hardcopy regardless of the scene-contents or hardcopy interface.

The DPI resolution of the rasterized geometry can be controlled by the developer, allowing them to determine the trade off between accuracy and speed/memory-usage during the hardcopy generation.

## 4.7 Selection

Immediate-mode 3D APIs generally do not perform picking operations other than a square box pick. Further, since low-level 3D APIs do not 'remember' what has been drawn, they perform picking by re-drawing the world and watching which objects have hit the pick rectangle.

HOOPS supports analytical selection which computes the intersection of the geometry and the selection region and returns precise information, as well as visual selection which utilizes an image buffer to only consider objects whose pixels are currently visible. Additionally, HOOPS can leverage higher level object information such as visibility, selectability and bounding box attributes to optimally traverse the object database in searching for primitives to hit-test against the pointing device position. As a result, the performance of picking is often several orders of magnitude faster in HOOPS.

HOOPS supports a wide range of selection methods:

- **Aperture** - A point location. Aperture is defined by a radial distance from the selection point.
- **Area** - A rectangular region
- **Polygon** (lasso) - A polygonal region
- **Polyline** (fence) - A series of points connected by line segments
- **Volume** - A cuboid 3D space
- **Object** - A HOOPS/3dGS shell primitive (enables clash-detection/interference-checking between objects, discussed further in the HOOPS/MVO section below)

HOOPS also supports a range of picking granularity *within* selected objects:

- **Vertex** - closest definition point
- **Edge** - closest polygonal border
- **Face** - closest polygonal facet
- **Object** - segment containing the selected geometry

HOOPS/3dGS will also compute the analytical point of intersection with the selected geometry in object, world and camera coordinate systems.

The various selection methods supplied by HOOPS enable the fine grain interaction techniques required for applications interested in creating and editing 3D geometry. These selection capabilities encapsulated several man-months of development effort.

## **4.8 Flexible Rendering**

HOOPS/3dGS provides high-level support for a wide variety of advanced rendering techniques, almost all of which are portable across platforms and device drivers. This is again made possible by HOOPS' scalable device interface, which includes software implementations for each technique. They include:

- Texture Mapping - A high-level interface allows the developer to easily apply texture or environment maps to objects. Multi-texturing, decaling and transparent textures are supported. Textures are cached and the image used for the texture is processed to ensure maximum hardware throughput.
- Color Contouring - When drawing faceted objects, HOOPS/3dGS can interpolate between vertex colors (color interpolation), or colormap colors (color index interpolation), which is critical for applications in the CAE market.
- Shadows
- Anti-aliasing
- Fog
- Stereo Viewing

These capabilities allow developers to easily create applications that have a very high level of realism, while maintaining interactive performance. These portable yet optimized rendering capabilities in HOOPS/3dGS encapsulate many man-months of effort.

## **4.9 Large Model Visualization**

HOOPS/3dAF includes a variety of techniques which enable applications to effectively interact with high polygon-count models. As model sizes scale up beyond the throughput capabilities of the graphics card, it is impractical to send all the triangles to the card and still expect reasonable framerates.

#### 4.9.1 Level-of-Detail

HOOPS/3dGS contains a module for the creation and management of Levels-of-Detail (LODs). Use of LODs can have a dramatic effect on the ability to render large scenes in real time. LOD support includes the following features:

- control over the number of LODs to generate and the percentage fall-off between these levels
- control over the logic used to switch between these various levels; both view-dependent and view-independent switching is supported
- access to 2 different LOD generation algorithms; one is a faster, less accurate algorithm, while the other produces a better visual result at the expense of initial computation time
- automatic interpolation of any vertex data at different levels of detail. Specifically, vertex colors/normals/texture-coordinates will be interpolated and applied to the different levels of complexity. This allows better interaction with scenes containing large amounts of costly texture maps and color-interpolated geometry such as CAE data.

A complete and stable implementation of just the most basic LOD calculation algorithms contained in the HOOPS/3dGS LOD module would require at least 6 months of development by a skilled graphics programmer. The entire module represents more than a man-year of development.

#### 4.9.2 Occlusion culling

Mentioned in Section 4.2, this feature determines which objects may be hidden, or occluded, by other objects in the scene, and can provide significant performance improvements when visualizing large models.

#### 4.9.3 Constant Framerate Logic

This includes the ability to set a desired framerate and control how the scene degrades. Such logic requires monitoring current rendering performance and adapting the rendering quality in order to meet the target framerate.

## 4.10 Optimized Performance

HOOPS/3dGS provides excellent performance regardless of platform, 2D/3D API or graphics cards. This is achieved via a combination of well-designed algorithms, tuned device drivers, data caching and heuristics capabilities. The following lists some specific examples:

- The graphics database is structured around attribute coherence in order to minimize graphics context switching and improve throughput.
- Tristrips are automatically calculated/cached and sent to the lower-level 3D device interface.
- The HOOPS/3dGS OpenGL driver leverages display lists and vertex arrays, while the Direct3D driver leverages vertex buffers. Performance tests are conducted to determine optimal lengths of these structures on various graphics cards, and the cutoff values are dynamically set in the 3D device drivers.
- Developers are given an interface to supply hints which allow HOOPS to make further optimizations; examples including specify polygon handedness, backplane culling, the existence of concave polygons, etc...
- When Intel SSE instructions are available, HOOPS/3dGS can perform backplane culling faster than the graphics card. (HOOPS/3dGS uses a dot-product instead of the graphics card's winding rule and the dot-product calculation can be accelerated using SSE.) If they are not available, HOOPS lets the graphics card do the culling.
- HOOPS/3dGS creates a variety of custom, temporary application-level 'display-lists' (arrays, quad-trees, or octrees) separate from its main scene-graph in order to improve rendering performance. These are used in features such as overlapped text, 2D/3D fonts, hidden-line removal, clash detection and occlusion culling.

Basic optimizations of internally developed scene-graph logic like the above requires months of effort. Man-years of effort are reflected in the extent of HOOPS/3dGS' optimizations. Just the work of determining optimal tristrip length on a wide variety of graphics cards would be a significant task for a development organization.

## 4.11 Immediate Mode with Context: HOOPS Intermediate Mode

Often it is useful for application developers to program with a procedural, non-retained mode interface language similar to OpenGL. With HOOPS I.M. it is possible to interrupt

the normal HOOPS update cycle at various points during traversal and rendering and have process control returned to the application via the use of callback functions. The callback routines are registered with HOOPS as attributes associated with segments and/or geometry and when invoked, the callback routines are passed all the information for the associated geometry and attributes. When traversal is trapped at a callback point, decisions can then be made about what and how something should be drawn, or the traversal process itself can be aborted.

From within the callbacks, the HOOPS HIC immediate mode interface can be used to query the graphics database and device characteristics, set attributes and perform immediate-mode drawing of 2D and 3D primitives. The traversal process may be trapped at either the 3D or 2D layers in the rendering pipeline and drawing functions are available at both of these levels.

This interface not only gives the developer the same degree of control that they would have when direct to OpenGL or Direct3D, but adds additional value because it is a fully portable, platform/device independent, immediate mode 2D/3D API. The developer can program to a single interface, and obtain consistent results regardless of the underlying driver or platform.

## 4.12 HOOPS/MVO

HOOPS/3dAF contains a set of customizable application-level C++ objects in source-code form. These objects make it easy for a developer to support many of the common higher-level operations required by an interactive 3D application. All of this code must be created internally if only a low-level immediate mode 3D API is being used. The following lists a subset of the HOOPS/MVO functionality areas:

- **Camera manipulation** – orbit/pan/zoom/walk; smooth transition between views
- **Render/view mode** - wireframe/shaded/HLR; toggle of hidden line display; toggling of vertex/edge/face/text visibility
- **Model annotation/query/measure**
- **Application of texture and environment maps**
- **Object creation/manipulation/deletion**
- **Insertion/manipulation of cutting planes** – toggling of capping edge/face visibility
- **Selection** – aperture/window/polygon/polyline

- **Animation authoring/playback** – includes built-in support for interpretation and playback of [HSF XML-based animation information](#); a sample animation authoring interface is included
- **Framerate functionality** - includes ability to set desired framerate and control how the scene degrades.
- **Level-of-detail generation** – this can be used standalone or in conjunction with the framerate support
- **Hiding of overlapped text** – a simple flag instructs MVO to hide pieces of annotation text that are entirely/partially obscured by others, aiding in the display of a non-cluttered scene with easily discernible annotations. While this feature is simple in concept, it must be implemented using a view-dependent hidden surface removal calculation similar to an HLR algorithm. This involves placing the text annotations in a quad tree which is traversed on each redraw to clip the annotations against each other.
- **Clash detection** – object to object intersections are calculated and displayed. This functionality leverages HOOPS/3dGS 'shell' selection functionality. The facets of the involved shells are placed in an octree in order to perform selection-frustum culling and improve performance for numerous, large objects.

HOOPS/MVO encapsulates a significant amount of value. While existing applications include support for some of these items, several of them are too costly an endeavor for application developers to undertake themselves. For example, constant framerate, animation authoring/playback and clash detection would take several man-months of effort to replicate.

## 4.13 HOOPS/Stream

HOOPS/Stream allows developers to create and stream customized HOOPS Stream Files (HSF). It is a mature, commercial-grade toolkit and encapsulates a variety of logic to create lightweight stream-capable files, including:

- lossless file-wide compression
- lossless compression of the connectivity list for indexed face sets (shells)
- lossy, configurable compression of vertices, normals, and texture parameters
- priority ordering of objects in the file, based on a size:complexity heuristic
- instance matching, to help remove duplicated objects and further reduce file size
- a file dictionary, to allow clients to support on-demand or view-dependent streaming (used in conjunction with the HOOPS/Net Server)
- ability to tag objects and thus associate them with other application data

Flagship applications and downstream product lines (viewers, plug-ins) generally work with an existing proprietary file format developed in house. It has become more desirable to embed lightweight 2D/3D vector information inside the proprietary format, in order to rapidly display the scene when loading the file, or easily transmit graphics information to downstream applications. If the graphics system were also developed in house, then there are 2 choices: to embed an existing file format or to define a new 2D/3D vector format for internal use.

Embedding an existing format is challenging because such formats are not intended to be used as components within other formats and usually are not flexible enough to represent the wide variety of graphical data and attributes found in today's engineering applications. Manually embedding graphical data is not that difficult if no compression or file ordering is required, but defining a compressed 3D format and creating logic to import/export the format would take several weeks to perhaps months of effort.

HOOPS/Stream makes it simple for graphics developers to create lightweight archives of 2D/3D vector information, or to publish it to the web for downstream viewing/interaction. Because the HSF specification matches the graphical primitive and attribute support encapsulated in HOOPS/3dGS, it naturally is able to capture and convey that information. The result is an extremely rich, engineering-oriented file format that addresses the needs of today's 3D engineering applications.

#### **4.14 HOOPS/GM**

Any application which contains a solid or surface modeling engine must integrate that engine with their graphics database. If a developer is maintaining an internal graphics system, they need to create a 'bridge' which connects to the modeler's tessellator, and keep the graphical representations in sync with the modeling kernel objects. Because HOOPS/3dAF is so often used alongside a modeling kernel, it includes Geometric Modeler Bridges which encapsulates this functionality. The ACIS, Parasolid and Granite modeling kernels are supported.

#### **4.15 HOOPS/Net Server**

The HOOPS Net Server (HOOPS/Net) is a flexible server-side toolkit for enhancing applications to include collaborative functionality and/or server-based data streaming. Session creation/management, flexible message-passing paradigm, session capture, data spooling and other key features make HOOPS/Net a powerful tool for adding synchronous (real-time) or asynchronous collaboration to your application

The toolkit supports HTTP and TCP/IP protocols for internet or intranet use and can be fully customized to handle any kind of application-specific data as well as binary stream-capable data from the HOOPS/Stream toolkit. Support for WinINet, firewall access,

proxy authentication and fault-tolerance for lost connections make HOOPS/Net a secure and reliable toolkit for client/server applications.

In conjunction with the HOOPS/Stream Toolkit, HOOPS/Net may be used to implement intelligent server-side streaming, where specific sub-elements of a file may be served to one or more clients as needed. Default logic includes support for on-demand or view-dependent streaming, and developers may apply their own logic regarding what data should be streamed and when.

Creation of such a server side module and associated client logic would require several man months of effort, with HOOPS/Net encapsulating over a man-year of development.

## **5.0 Technical Support and Consulting Services**

HOOPS is more than just a set of libraries: Tech Soft 3D and Spatial provide a complete set of technical support and consulting services and have a proven track record in supporting hundreds of software developers from design through delivery.

HOOPS has been in use by industry leading software development organizations for over 15 years and the technical support and consulting group has gained valuable insight and experience with each implementation effort. In many instances, the product direction of HOOPS, from functionality in the library to specialized demonstration or integration code, stems directly from this on-going dialogue with the customer base.

Tech Soft strives to serve as a virtual extension of our customer's development organization and we pride ourselves on the level of responsiveness, customization, and flexibility behind the HOOPS technology. Tech Soft engineers work closely with our customer's engineering development teams to collaborate on feature specification and scheduling, and consistently deliver useful and well-tested features with short turnaround times. This ensures that HOOPS customers do not lose control over the future technical direction in which they would like to take their products.

## **6.0 Summary**

With over 150 man-years of expert development effort, the HOOPS/3dAF components provide extensive functionality needed for commercial-grade CAD/CAM/CAM applications, and are stable, mature, and well documented. Immediate-mode 3D interfaces have now become a small piece in the puzzle that is an engineering software application and the question of whether or not to use a component such as HOOPS is truly a question of whether or not to 'do it yourself'. Developers writing applications to an immediate mode interface are choosing to re-implement the HOOPS capabilities outlined here.

While the amount of functionality required in each of the technical areas described above depends on product requirements, it is becoming more and more common for today's engineering application's to require a significant amount of value in most, if not all, of the areas. Use of HOOPS/3dAF liberates developers from either building or further enhancing internal retained mode graphics functionality, thus saving development time and enabling them to focus on core areas of added-value.

If your organization has determined that it makes sense to investigate and potentially leverage a component such as HOOPS, a logical question is how it compares to similar technologies. Our experience along with those of many HOOPS customers has been that no other scene-graph related component matches the expanse, depth, flexibility and stability of HOOPS' features as they apply to engineering markets. We encourage developers to work with a Tech Soft 3D consulting engineer and evaluate HOOPS relative to other options under consideration.