



The HOOPS/Parasolid Bridge

- The HOOPS/Parasolid Bridge 1**
- 1 Why Build a Bridge between Parasolid and HOOPS? 2**
 - 1.1.1 Complementary, Cross Platform Toolkits 2
 - 1.1.2 Parasolid Provides No Graphics Functionality 2
 - 1.1.3 HOOPS 3D Graphics System: Designed for MCAD 2
 - 1.1.4 Some Geometry must be separate from the Parasolid model 2
- 2 Parasolid GO Routines 5**
 - 2.1.1 HOOPS Segment Structure generated 6
 - 2.1.2 Mapping of PK_Entities to HOOPS geometry 6
- 3 A Cross Platform Parasolid Frustrum 7**
- 4 HOOPS/Parasolid Bridge API..... 7**
 - 4.1.1 HP_Init () 7
 - 4.1.2 HP_Close () 7
 - 4.1.3 HP_Read_Xmt_File () 7
 - 4.1.4 HP_Show_Render_Options () 8
 - 4.1.5 HP_Set_Render_Options () 8
 - 4.1.6 HP_Compute_Geometry_Keys () 8
 - 4.1.7 HP_Compute_Geometry_Key_Count () 8
 - 4.1.8 HP_Compute_TagID () 8
- 5 Integrate following into above 9**
 - 5.1.1 Parasolid Frustrum must be implemented..... 9

www.hoops3d.com



1 Why Build a Bridge between Parasolid and HOOPS?

There are several compelling factors that led the Parasolid and HOOPS teams to integrate the two components:

- The components are complementary, cross-platform toolkits
- Parasolid provides not graphics functionality
- The HOOPS 3D Graphics System has been designed to be the graphics sub-system for Mechanical CAD applications.
- Applications need to create and modify geometry separate from the Parasolid model

The integration of the two products provides developers with a base architecture and extensible application development framework for implementing their application while ensuring optimal design and rendering performance of applications built with both components.

1.1.1 Complementary, Cross Platform Toolkits

1.1.2 Parasolid Provides No Graphics Functionality

The Parasolid Kernel Modeler from Unigraphics Solutions provides no graphical services. This is the primary motivation for integrating Parasolid's geometric modeling capabilities with HOOPS' 3D graphics capabilities. Until now there has been no way to display and interact with the information in the Parasolid database short of creating a graphics subsystem and connecting it to Parasolid via its Graphical Output interface.

1.1.3 HOOPS 3D Graphics System: Designed for MCAD

1.1.4 Some Geometry must be separate from the Parasolid model

Most applications have needs for geometry that cannot be implemented with Parasolid.

Almost all the applications in the manufacturing process including:

Egs:

Modeling – Construction Geometry; Grids; Axis triads

Analysis - Contour legends for result information

Drafting – dimensional geometry including lines and fonts

View markup – 3D text and Markers



Virtual Prototyping – LOD generation and dynamic switching; Images for texture mapping
CAM – Cutting Tool Paths

HOOPS customers span each area of the manufacturing process. Development of the HOOPS Graphics System has been directly input from these customers.

HOOPS feature set includes extensive functionality addressing the needs of each of these application domain's needs.

Primitive Set –
Markers,
radial and quadrilateral grids
Images
Fonts

Rendering Capabilities
Texture Mapping

Text
Application requiring functionality involving text of any kind must implement this outside of Parasolid.

Color Contouring of data connected with Parasolid geometry. Analysis data typically (heat gradients, stress/strain, magnetic fields)

1. **Rich Primitive set** – HOOPS has a complete set of geometric primitives for 3D/2D CAD/CAM applications (including point clouds, and raster images)
2. **Geometry separate from the model** : For dimensioning and other notational reasons developers need to have geometry in the scene which is not part of the model. Examples of these include legends, view triads, dimensioning lines.
3. **Fonts** – HOOPS has fully 3D transformable, system (X11 & UNIX), bit-mapped, stroked and outlined spline fonts (true type and Adobe Type 1). Developers can also define their own font types.
4. **Kanji language support** – complete 3D stroked character set.
5. **Highly optimized analytical hidden line removal algorithm:** Enhancements to ensure optimal rendering performances – HOOPS' hidden line removal algorithm is significantly faster than the ACIS algorithm. Developers can also control the visibility, pattern and dim factor of the hidden and visible lines.
6. **User Defined Patterns** for polygon faces, edges and vertex markers



7. **Color Interpolation** for Surfaces based on supplied data values - CFD, FEA analysis, Curvature visualization for CAM
8. **Hardcopy support** - HOOPS directly supports PostScript, HPGL, CGM and tiff image.
9. **Temporary/Construction Graphics** - easily implemented with HOOPS' quick moves attribute.
10. **Setting Drawing Sequence** - HOOPS priority attribute available for user control of drawing sequence for primitives. This is a "user defined hsra"
11. **Procedural graphics** - HOOPS Immediate mode available at the 3D and 2D callback points, with full APIs for drawing. Enables developer to create scale independent geometry, create own line styles, face patterns etc.
12. **Coordinate conversion** from one space to another- e.g., construction planes easily implemented in viewport space. Spaces include object, world, view, window, pixels.
13. **Snap to Grid** - implemented with HOOPS radial or quad grids and selectability attribute.
14. **Large Model Capabilities** - on-going R&D in the areas of polygon decimation/ Levels of Detail (LODs), view frustum culling.
15. **Dynamic View dependent Geometry Loading**– on-going R&D in the dynamic switching of different tessellation levels of geometry. Designed for the Large Model capabilities however can also be used for view dependent tessellation of models. These are implemented in HOOPS as user supplied LODs.
16. **Fully supported application framework (HOOPS/AFC) components** for GUI integration, HOOPS/MVO classes (application level objects -e.g., camera and model manipulators, selection set operators, geometry creation, ...)
17. **JAVA, GUI (MOTIF, MFC) Toolkit integration** - Fully supported integrations with all the major GUI toolkits. Integrations include support for standard system services such as printing, print preview, color palette handling, copying to clipboard etc. A HOOPS/Qt integration is also in development. HOOPS works seamlessly with other GUI toolkits, only requiring the unique window ID of the underlying win32 or x11 window.
18. **ActiveX Control** – We supply developers with an ActiveX control which enables them to easily create lightweight versions of their



applications which can be easily embedded in ActiveX container applications such as the Microsoft Internet Explorer, Netscape Navigator and the Microsoft Office Application Suite.

Developers building applications with the Parasolid kernel must implement a "Frustrum" which consists of supplying routines for memory management, file I/O, and graphical display of geometric information. The graphical information is passed to the developer via the GO_Routines and must be processed and communicated to a computer screen or monitor. Thus the developer must supply a list of routines that accept the output of the Go_Routines. These routines must upon receipt of this information, send it to an output device via its programming interface, such as OpenGL, Direct3D or in the case where hardcopy is the goal, PostScript or HPGL.

TSA's implementation of the Parasolid Frustrum, the HOOPS/Parasolid Bridge implements all the necessary work to connect Parasolid and HOOPS to each other and

in an MFC application. It encapsulates file opening, parsing, reading, writing, starting and stopping of Parasolid & an implementation of the GO_Routines for graphical output via HOOPS.

The tessellation of the Parasolid model is cached in HOOPS. The application then uses HOOPS routines for viewing manipulations, querying, creating and managing temporary graphics for defining the parameters of new Parasolid entities, etc.

2 Parasolid GO Routines

The Parasolid PK_Render_XXX routines cause the modeler to traverse all or part of the topological information currently in memory and tessellate it for output to a computer monitor or printer. Parasolid includes no mechanisms for rendering but rather provides a set of output routines that must be implemented by the developer. These routines receive the tessellated information from Parasolid and must arrange to map this to the desired output device or devices. This set of output routines is called the Parasolid GO_Routines.

The primary value of the integration of Parasolid and HOOPS is in the implementation of the GO_Routines. They have been designed to optimally map the Parasolid tessellated information to the HOOPS graphical database.



2.1.1 HOOPS Segment Structure generated

The GO Routines assume the existence of an Open HOOPS Segment. A modal attribute controls whether or not a set of segments under the currently open segment are created, each with their own HOOPS color attribute whose value is determined by the corresponding PK_Entity's color attribute. The default case is to partition the geometry into HOOPS segments based on their PK_Color attribute.

Depending on what an application wants to provide in the way of operations on the PK_Body, PK_Shell, & PK_Region entities, a more or less complicated segment hierarchy must be created by the application before calling PK_Render_XXX.

2.1.2 Mapping of PK_Entities to HOOPS geometry

GO Routines map all Parasolid tessellated geometry to HOOPS entities, inserting a new HOOPS geometric primitive into the HOOPS database for each PK_Entity.

Each of the components use handles for accessing their entities; Parasolid entities are said to have "Tag IDs" and HOOPS entities are said to have "keys". The integration provides a one to one mapping of Parasolid to HOOPS entities by renumbering the HOOPS keys to be the same as the Parasolid Tag ID.

2.1.2.1 PK_Faces: Edit or View mode

A modal attribute on the HOOPS Frustrum controls how the PK_Faces are mapped to HOOPS Shells.

When in "edit" mode, the modeler operations will be creating, deleting and modifying PK_Faces frequently and the amount of data flowing from Parasolid to HOOPS should be minimized. This is accomplished by mapping all the tessellated geometry associated with one PK_Face to one HOOPS shell. An additional optimization step ensures the minimum amount of vertices are used and that the longest triangle strips possible may be computed internally by HOOPS.

When in "view" mode all the tessellated geometry associated with all the PK_Faces in one PK_Body are mapped to one HOOPS shell. This results in significant reduction in amount of vertices for complex PK_Bodies and results in increased rendering through-put. It is anticipated that this mode will be useful to applications primarily focused on viewing & querying model or assembly data. (View/Markup sector).

2.1.2.2 PK_Edges

Map to one of the following HOOPS primitives depending on the form of their tessellated geometry generated by the Parasolid modeler.



- HOOPS Polyline
- HOOPS Circular Arc
- HOOPS Circle
- HOOPS Line

2.1.2.3 PK_Vertices

These Parasolid entities do not appear in the GO_Routines. If graphical display of these entities is desired, the application must use the appropriate PK_Query routines and manually insert HOOPS geometry, typically markers, to represent them.

3 A Cross Platform Parasolid Frustrum

Developers must connect Parasolid to the OS and provide platform specific services such as memory management..

This is time consuming and as.dfasdfj
Now you don't have to.

TSA supplies as part of the HOOPS/Parasolid Bridge a Parasolid Frustrum implementation for both Windows and UNIX.

4 HOOPS/Parasolid Bridge API

The HOOPS specific integration of the Parasolid Frustrum for the Windows operating system & Microsoft Foundation Classes is supplied as a dynamic link library (DLL). The library's application programming interface (API) consists of the following function calls. Developers will include the frustrum dll in the list of libraries needed for building their application and reference the header file "frustrum.h" in the files using these API calls.

4.1.1 HP_Init ()

void HP_Init (void)

Initializes the Parasolid kernel, registers the HOOPS-specific implementation of the GO_Routines, initializes HOOPS and its connection to the UI (MFC).

4.1.2 HP_Close ()

void HP_Close (void)

Shuts down the Parasolid kernel.

4.1.3 HP_Read_Xmt_File ()

void HP_Read_Xmt_File (const char *fname, char* SchemaFilePath)



Parses the input Parasolid file, populates the kernel with its contents, and creates the corresponding geometry in the HOOPS database using the Parasolid renderer and HOOPS frustrum.

4.1.4 HP_Show_Render_Options ()

```
void HP_Show_Render_Options (PK_TOPOL_render_line_o_t  
*line_Options, PK_TOPOL_render_facet_go_o_t *go_options,  
PK_TOPOL_facet_mesh_o_t *generation_settings);
```

This routine will show the render options that are currently being used to render any geometry to the HOOPS frustrum.

4.1.5 HP_Set_Render_Options ()

```
void HP_Set_Render_Options (PK_TOPOL_render_line_o_t *line_Options,  
PK_TOPOL_render_facet_go_o_t *go_options,  
PK_TOPOL_facet_mesh_o_t *generation_settings);
```

This routine will alter the render options that are currently being used to render any geometry to the HOOPS frustrum.

4.1.6 HP_Compute_Geometry_Keys ()

```
long HP_Compute_Geometry_Keys (PK_ENTITY_t tagID, HC_KEY* keys,  
long geomTypes, long count);
```

Given a Parasolid tag ID for any entity, this routine will return an array of HOOPS keys for the geometry in the HOOPS database generated by the GO_Routines by the PK_Render_XXX function.

4.1.7 HP_Compute_Geometry_Key_Count ()

```
long HP_Compute_Geometry_Key_Count (PK_ENTITY_t tagID, long  
geomTypes);
```

Given a Parasolid tag ID for any entity, this routine will return the number of HOOPS geometric entities in the HOOPS database which graphically represent the Parasolid entity.

4.1.8 HP_Compute_TagID ()

```
PK_ENTITY_t HP_Compute_TagID (HC_KEY key, PK_CLASS_t  
paraClass);
```

Given a HOOPS Key this routine will return the Parasolid Tag ID for the Parasolid entity that contains the specified HOOPS geometric entity.



5 Integrate following into above

5.1.1 Parasolid Frustrum must be implemented

Additional work independent of the graphics subsystem must be performed to connect Parasolid with the operating system for services such as memory management and file I/O. These services are connected via its Frustrum interface.

Building Parasolid Graphical Output and Frustrum implementations is a lengthy undertaking which can take many person years to complete, especially if advanced graphical features are desired and/or support of multiple platforms is desired.

Windows and UNIX specific Parasolid Frustrum

Parasolid requires several connections to the operating system for tasks such as file I/O and memory allocation. These platform-specific facilities must be provided by the application developer. This set of routines is called the Parasolid Frustrum.

The Integration code contains a full implementation of these routines for the Microsoft Windows operating system using the Microsoft Foundation Classes.

5.1.1.1 HOOPS Specific Parasolid GO Routines

Parasolid provides no rendering or printing capability. The modeler instead outputs tessellated geometric information via the Parasolid Graphical Output (GO) Routines. The developer must supply routines, which accept this information and then manage the display, interaction and printing of this information on computer monitors and printers.

The Integration Code contains an implementation of the GO routines, which use the HOOPS 3D Graphics database for rendering and printing the tessellated geometry.