Chapter 2. ACIS Overview

ACIS is an object-oriented three-dimensional (3D) geometric modeling engine from *Spatial Technology Inc.* (*Spatial*). It is designed for use as the geometry foundation within virtually any end user 3D modeling application.

The ACIS product line consists of the **ACIS 3D Toolkit** geometric modeling engine and a variety of *optional husks*, which are separately-sold products that may be added on to ACIS for advanced or specialized functionality.

ACIS is written in C++ and consists of a set of C++ classes (including data and member functions, or methods) and functions. A developer uses these classes and functions to create an end user 3D application. ACIS complements existing applications by offering a unified environment for the modeling of curves, surfaces, and solids. ACIS also supports the integration of proprietary curve and surface subsystems. ACIS provides a foundation of common modeling functionality and the flexibility to be adapted and extended for particular application requirements.

ACIS integrates wireframe, surface, and solid modeling by allowing these alternative representations to coexist naturally in a unified data structure, which is implemented in a hierarchy of C++ classes. ACIS bodies can have any of these forms or combinations of them. Linear and quadric geometry is represented analytically, and non-uniform rational B-splines (*NURBS*) represent free-form geometry.

In addition to manifold geometry, ACIS can represent nonmanifold geometry. Geometries can be bounded, unbounded, or semi-bounded, allowing for complete and incomplete bodies. A solid can have faces missing and existing faces can have missing edges. Solids can also have internal faces that divide the solid into cells.

ACIS is a boundary-representation (*B-rep*) modeler, which means that it defines the boundary between solid material and empty space. This boundary is made from a closed set of surfaces.

Modeling Overview

A *wireframe model* defines an object only by its edges and vertices. A *surface model* is similar to a wireframe model, but defines an object by its visible surfaces, including faces. A *solid model* defines an object in terms of its size, shape, density, and physical properties (weight, volume, center of gravity, etc.). ACIS is a solid modeler, but wireframe and surface models may also be represented in ACIS.

ACIS *separately* represents the geometry (detailed shape) and the topology (connectivity) of objects. This concept is called *boundary representation*, or *B-rep*, modeling. This provides the ability to determine whether a position is inside, outside, or on the boundary of a volume (which distinguishes a solid modeler from surface or wireframe modelers).

The ACIS model representation consists of various geometric and topologic *entities*, as well as *attributes* that may be attached to the entities. The model is implemented in C++ using a hierarchy of classes.

Geometry

Geometry refers to the physical items represented by the model, such as points, curves, and surfaces, independent of their spatial—or topological—relationships. ACIS implements geometry in two distinct forms:

Construction geometry	Refers to the C++ classes that hold the mathematical definitions of geometric objects. Construction geometry classes have lowercase names.
Model geometry	Refers to the C++ classes that add model management functionality to the construction geometry (they include pointers to construction geometry classes as part of their data structures). Model geometry is permanent and saved with the model. Model geometry classes have uppercase names.

Both construction geometry and model geometry can be conceptually separated into *abstract* geometry and *specific* geometry. Abstract geometry is usually broken down into specific geometry. The specific geometry classes are derived from the abstract ones, and thus inherit properties common to all geometry of that type. For example, the abstract geometry for a curve is broken down into specific geometry for specific types of curves, such as ellipse, straight, interpolated curve, etc.

Model Topology

Topology refers to the spatial relationships between the various entities in a model. Topology describes how geometric entities are connected. On its own, topology defines a “rubber” model, whose position is not fixed in space. For example, a circular edge and an elliptical edge are topologically equivalent (but not geometrically). Likewise, a square face and a rhomboid face are topologically equivalent (but not geometrically).

Entities and Model Objects

An *entity* is the most basic ACIS object. It is implemented in the C++ class ENTITY. All entities have a common set of functionality, such as the ability to save and restore themselves to and from a file, copy themselves, and debug themselves. All other geometric and higher level ACIS model objects are derived from the ENTITY class.

A *model object* is any object that can be saved to and restored from a save file (.sat or .sab). ACIS model objects are implemented in C++ using a hierarchy of classes that are derived from the ENTITY class.

Attributes

Attributes are used to attach data to entities. Every entity may have zero or more attributes. The C++ class ATTRIB, which is derived directly from the ENTITY class, provides the data and functionality that all attributes share, for both user-defined attributes and system attributes. The ATTRIB class performs housekeeping operations to maintain attribute lists attached to model entities.

Attributes can carry simple data, pointers to other entities, or links to application-specific, variable length data. Many attribute classes that perform specific tasks are derived from the parent ATTRIB class. Developers can also derive their own application-specific attribute classes from ATTRIB.

Architecture Overview

A *software component* is a functionally specialized unit of software—a collection of software items (functions, classes, etc.) grouped together to serve some distinct purpose. It serves as a constituent part of a whole software system or product. A *product* is one or more software components that are assembled together and sold as a package.

Components

Table 2-1 lists all the components available for the ACIS product line. This table includes the components for optional husks.

Table 2-1. Software Components

Component Name	Description
ACIS MFC (AMFC)	Support for MFC based applications
Advanced Blending (ABL)	Optional husk for blending
Advanced Rendering (AR)	Optional husk for rendering
AG Spline (AG)	AG spline library and interface
Basic Rendering (BR)	Rendering supplied with ACIS
Blending (BLND)	Standard blending operations
Boolean (BOOL)	Unite, intersect, and subtract operations
Cellular Topology (CT)	Divide lumps into sets of cells
Clearance (CLR)	Determine minimum distance between bodies or faces; determine if bodies or faces are within a certain minimum distance
Constructors (CSTR)	Basic topology construction; wireframe construction and editing; analysis (area, length, mass properties)
Covering (COVR)	Cover wires and sheets (all boundaries specified)
Deformable Surface (DS)	Optional husk for performing local deformations
Euler Operations (EULR)	Expand, flatten, separate, and combine lumps
Faceted Hidden Line (FHL)	Removes hidden lines fro solids and surfaces using facet data
Faceter (FCT)	Generate faceted (polygonal) representation
Generic Attributes (GA)	Attributes that allow applications to exchange data
Graphic Interaction (GI)	Commonly needed graphic display functionality
Intersectors (INTR)	Curve/curve, curve/surface, surface/surface intersectors; ray testing; silhouettes; parameter lines; point classification; body checking; curve and surface extension

Component Name	Description
Kernel (KERN)	Basic entity and attribute support; topology and geometry ENTITY classes; geometry classes (curve, surface, point, intcur, splsur); math classes (position, vector, transforms, etc.); save and restore support; history and roll support
Local Ops (LOP)	Optional husk for performing local operations
Local Op Tools (LOPT)	Tools used by Local Ops; this is part of the optional **Local Operations Husk** product
Meshing (MESH)	Optional husk for defining mesh surfaces
Offsetting (OFST)	Wire and face offsetting
OpenGL (GL)	Rendering for Windows NT and SGI platforms using OpenGL
Operators (OPER)	Spline conversion
Part Management (PART)	Support for grouping entities
Persistent ID (PID)	Attach identifiers that persist across saves
Precise Hidden Line (PHL)	Optional husk for hidden line removal
QuickDraw (QD3D)	Rendering for Macintosh platforms using QuickDraw 3D
Remove Faces (REM)	Remove faces, if necessary, after a local operation; this is part of the optional **Local Operations Husk** product
Rendering Base (RBASE)	Interface common to all renderers
RenderWare (RW)	Optional husk for rendering
Romulus Converter (ROM)	Romulus data format conversion
Scheme AIDE Main Program (TKMAIN)	Main program for the Scheme based demonstration application (Scheme ACIS Interface Driver Extension)
Scheme Support (SCM)	Scheme Interpreter; basic Scheme extensions; Scheme AIDE interface to rendering, part management, etc.
Shelling (SHL)	Optional husk for creating shelled bodies
Skinning (SKIN)	Skinning, lofting, and net surface operations to fit surfaces through curves

Component Name	Description
Sweeping (SWP)	Sweep a profile along a path
Test Harness Commands (TB)	Commands for the ACIS Test Harness application
Test Harness Main Program (AMAIN)	Main program for the ACIS Test Harness application

Component Dependencies

A component may be dependent on others. All components are ultimately dependent upon the Kernel Component, which is codependent on the AG Spline Component. Figure 2-1 shows the dependency graph for the “core” **ACIS 3D Toolkit** (**ACIS 3DT**) components. The core components are those components of the **ACIS 3DT** product that provide ACIS modeling functionality and are not simply for supporting an interface into ACIS (such as the ACIS Test Harness, Scheme, or MFC). This does not include optional husk components.

The dependencies are indicated by lines flowing from one component to the components on which it is dependent. The KERN and AG components are codependent, which is reflected in the dependency graph with a loop. Except for this loop from AG back up and into KERN, all dependency paths flow down.

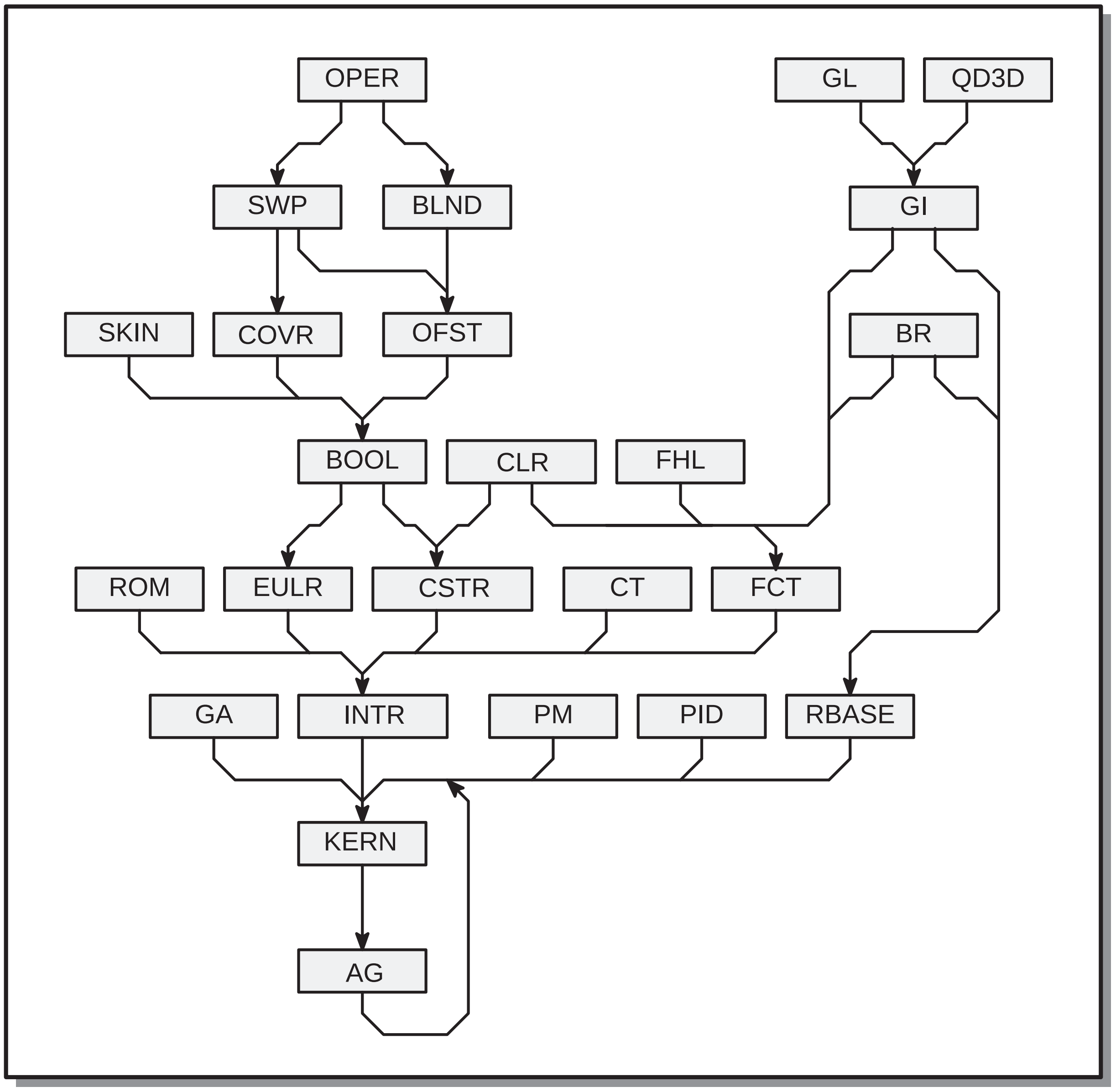

Figure 2-1. ACIS 3DT Component Dependencies

Figure 2-2 shows the dependency graph for the optional husk components. This dependency graph includes only those core **ACIS 3DT** components (shown with dashed lines) on which the optional husk components depend, either directly or indirectly.

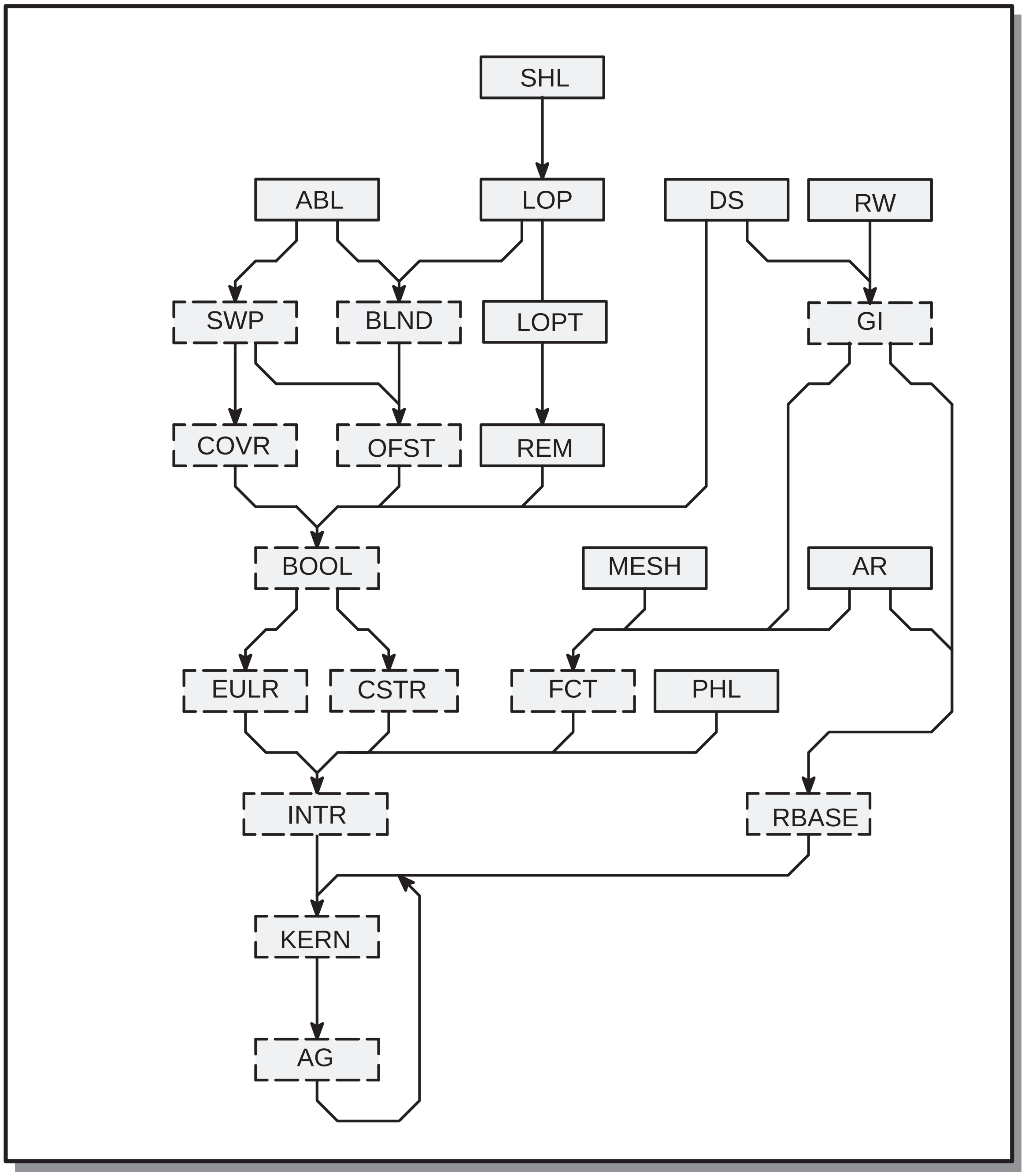

Figure 2-2. Optional Husk Component Dependencies