Chapter 2.
Deformable Modeling Library

Topic: *Deformable Modeling

The deformable modeling library is a self-contained C++ component designed to enhance an existing geometry kernel with deformable modeling capabilities.

This chapter discusses topics specific to the Standalone Deformable Modeling Component library. The deformable modeling library is not intended to be used as a standalone geometry kernel. It depends on an external geometry kernel for application functions such as persistence and rollback and for some simple intersection and evaluation functions.

A deformable model (sometimes abbreviated 'dmod' in the code) refers to the geometry to be deformed. Deformable models can be splines, surfaces, or curves. Deformable models are modified by applying behaviors, such as loads, to domains in the model. Each time a behavior changes, the deformable model is solved for a new shape.

Library Overview

Topic: *Deformable Modeling

The kernel library for Standalone Deformable Modeling is dshusk.lib. The set of functions defined in the file dmapi.cxx comprise the interface to the deformable modeling library. All the major types, constructors, and concepts of the deformable modeling library are exported through the APIs.

The optional drawing libraries provide low-overhead drawing functionality with a flexible, incremental development path. Deformable modeling application developers can modify or replace some or all of the optional drawing libraries deriving from the interface classes.

dmicon

The dmicon library provides icon objects for all deformable modeling tag objects (surfaces, curves, constraints, and loads). A common command/query interface is provided by the DM_default_icon base class, which is derived from the DM_icon interface class.

Application Examples

The admicon library, admgi_draweng library, and admgi_control library are provided as ACIS–based examples. The admicon library shows how to extend the dmicon library. admgi_draweng shows a sample draw engine, and admgi_control shows a sample view controller (see Figure 2-1).

Include Paths

Topic: Deformable Modeling

The include paths for the Standalone Deformable Modeling Component library are as follows:

```
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dmicon.hxx"
#include "dshusk/dskernel/dm_dbl_array.hxx"
#include "dshusk/dskernel/dm_tag_array.hxx"
#include "dshusk/dskernel/dm_icon_args.hxx"
```

```
ds/dshusk/dskernel/dmicon_factory.hxx
ds/dshusk/dskernel/dm_icon_args.hxx
ds/dshusk/dskernel/spatial_abs_hurler.hxx
ds/dmicon/dm_draw_engine.hxx
ds/dshusk/dsgeomint/dssrcdat.hxx
```

Interface

Topic: Deformable Surfaces

The programming interface consists of classes and API functions. SDM calls typically start with DM_*.

The deformable model library may be compiled with any level of optimization and debugging.

Interface Classes

Topic: Deformable Surfaces

Interface classes are the glue holding libraries together. Interface classes allow libraries to be replaced independently, using derivation. The examples in the optional drawing libraries demonstrate derivations of the following interface classes.

DM_icon

A DM_icon knows how to draw itself, and has a public Draw method. The DM_icon interface class also provides abstract methods to notify the deformable modeling kernel for drawing services.

The contract with the DM_icon class gives the deformable modeling kernel three responsibilities:

- DM_icon objects are owned by tag objects; they are created by the tag object constructor and destroyed by the tag object destructor.

- The deformable modeling kernel calls the icon Set_owner method when the tag object is fully constructed. This allows the icon to initialize itself, and retain knowledge of its owner.
- The deformable modeling kernel calls the icon Tagobject_changed method to notify the icon when the tag object state has changed (e.g., geometry or behavior). The icon can then redraw, or set a flag for lazy update, etc.

DM_icon_draw_args

The DM_icon_draw_args interface class provides a command object to forward client requests through the DM_icon::Draw methods to the DM_draw_engine. A typical example is the particular view or views to draw into: the DM_icon can tell the DM_draw_engine what geometry to draw, and the DM_icon_draw_args can be set up to tell the DM_draw_engine onto what canvas to draw.

The command object can forward through the deformable modeling interface, providing deformable modeling hierarchy broadcast capabilities.

DM_icon_cmd_args

The DM_icon_cmd_args interface class provides a command object that encapsulates icon commands, such as set line width in DM_default_icon. The command object can forward through the deformable modeling interface, providing deformable modeling hierarchy broadcast capabilities.

Exception Handling Across Interfaces

Topic: Deformable Surfaces

Exception handling across interfaces is problematic, because different libraries may have different exception handling methods, such as C++ try/catch or C setjmp/longjmp. The Spatial_abs_hurler interface class provides a protocol for handling exceptions across interfaces. Interface routines taking a Spatial_abs_hurler agree to trap all exceptions, translate them to an integer code, and then call the Spatial_abs_hurler::rethrow_error method with the integer code.

The interface class methods for the optional deformable modeling drawing libraries take a Spatial_abs_hurler. Users developing their own drawing libraries are thus required to follow this exception handling protocol.

Draw Command Pipeline

Topic: Deformable Surfaces

The schematic in Figure 2-1 depicts the draw command pipeline in an event driven application. This is the typical situation for a GUI application supporting multiple views.

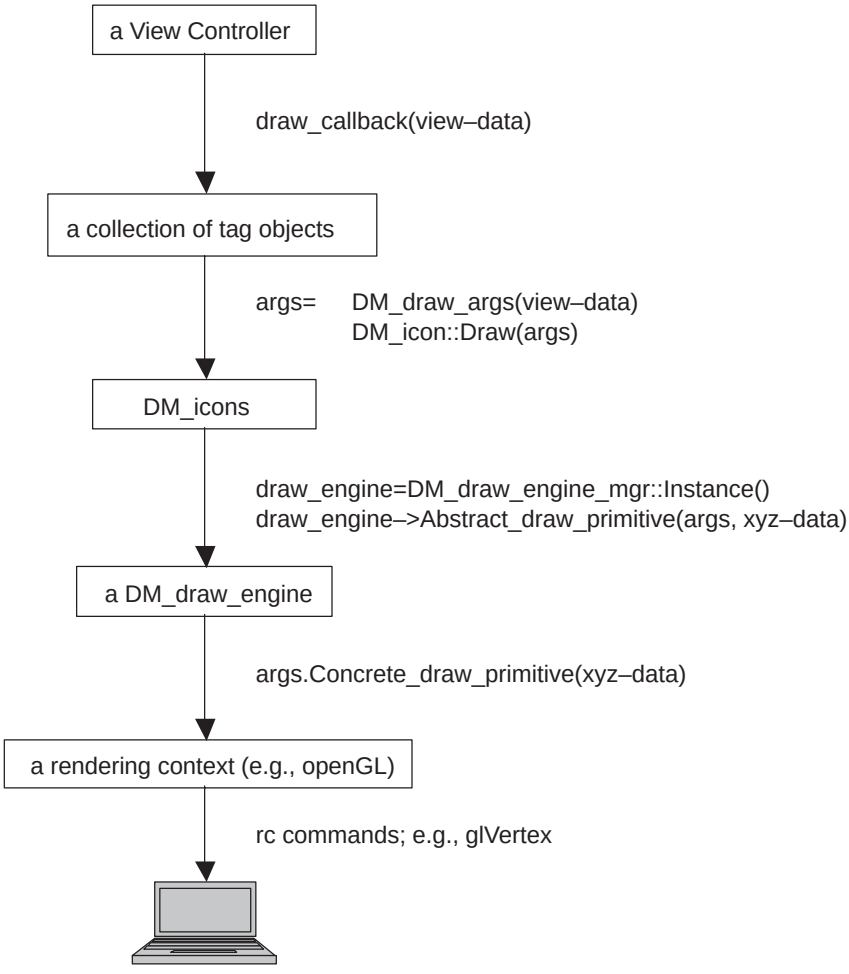

Figure 2-1. Draw Command Pipeline

Using Journal Files

Topic: Deformable Modeling

Journaling is a debugging tool for deformable modeling customers who do not run ACIS. The DM_journal_on function opens a file and sets global variables to begin journaling all future deformable modeling API calls. The cascade argument controls how much information gets written to the journal file. The resulting *.jrl file can be used to report problems. Journaling is turned off with a call to DM_journal_off.