Chapter 2.
ACIS Deformable Modeling

Topic: *Deformable Surfaces

The basic use of deformable modeling is to construct a curve or surface, apply constraints and loads, and then update the shape's control point set to satisfy the constraints, respond to the loads, and remain fair. Advanced features of the library support local and global deformations as well as deformations away from a default shape. The intended use of the library is to support interactive mouse based manipulations of the free form shape. Features are included in the library to support mouse based picking and dragging. An example mouse interface is provided, written for the Scheme AIDE environment, that supports editing shapes by adding and manipulating constraints and loads.

ACIS Deformable Modeling Component (ADM) is an interactive sculpting tool for defining fair, free form curves and surfaces. The advantages of ACIS Deformable Modeling include user leverage and automated constraint enforcement. Typically, free form shapes are described by a large set of control point locations. With ACIS Deformable Modeling, users can edit free form shapes by manipulating a small set of parameters which, in turn, cause the deformable modeling algorithm to modify the set of control point locations. Additionally, while manipulating the control point locations, the deformable modeling algorithm simultaneously selects control point locations to automatically enforce a wide variety of design constraints. ACIS Deformable Modeling provides an alternative to control point manipulation for curve and surface design.

This chapter provides the ACIS Deformable Modeling Component (ADM)–specific information. The areas of most importance are as follows:

1. Surface Library
2. Data Management

Libraries

Topic: Deformable Surfaces

The ACIS Deformable Modeling Component (ADM) is built on top of the non-ACIS based deformable modeling component Standalone Deformable Modeling Component (SDM), which is also available from *Spatial*. Beginning with ACIS Release 7.0, the ADM directory is renamed to adm, SDM is in its own component-level directory (ds), and a third deformable modeling component, ACIS Deformable Modeling Graphic Interaction Component (in directory admgi), was added for drawing. These changes involve changes and additions to the deformable modeling libraries.

Interface

Topic: Deformable Surfaces

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

The programming interface consists of ATTRIB_DM2ACIS methods and API function calls that start with api_dm_*.

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

Main Libraries

Topic: Deformable Surfaces

The two main deformable modeling libraries are admhusk for ADM, and dshusk for SDM. The main library for ADM users is named admhusk.lib. ADM users will find the main header files in the adm/admhusk directory. The admhusk library depends on the dshusk library.

Optional Drawing Libraries

Topic: Deformable Surfaces

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.

admicon

The admicon library extends the functionality of the dmicon library by using the ACIS INDEXED_MESH class for surface drawing. The code in this library serves as example drawing code for ADM users.

admgi_draweng

The admgi_draweng library can be used with the ADM icons as the first half of a bridge to the Graphic Interaction Component. With the admgi_control library, it integrates ADM drawing into ACIS. The Scheme AIDE application can be used an example application.

The admgi_draweng library implements the ADM_gidraw_engine and ADM_giicon_draw_args objects, which are derived from the DM_draw_engine and DM_icon_draw_args interface classes.

admgi_control

The admgi_control library can be used with the ADM kernel as the second half of a bridge to the Graphic Interaction Component (GI).

The admgi_control library provides registration services to the GI view controller. This includes a callback function that is called when views are refreshed, and Activate and Deactivate, and Make_backup_copy methods for supporting ACIS roll.

The admgi_control library implements an ADM_giregobj object, which is derived from the ADM_regobj interface class.

Exception Handling Across Interfaces

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.

Library Dependencies

Topic: Deformable Surfaces

The deformable modeling library dependencies are summarized in Figure 2-1. This figure includes several of the “core” ACIS libraries, in addition to the deformable modeling libraries.

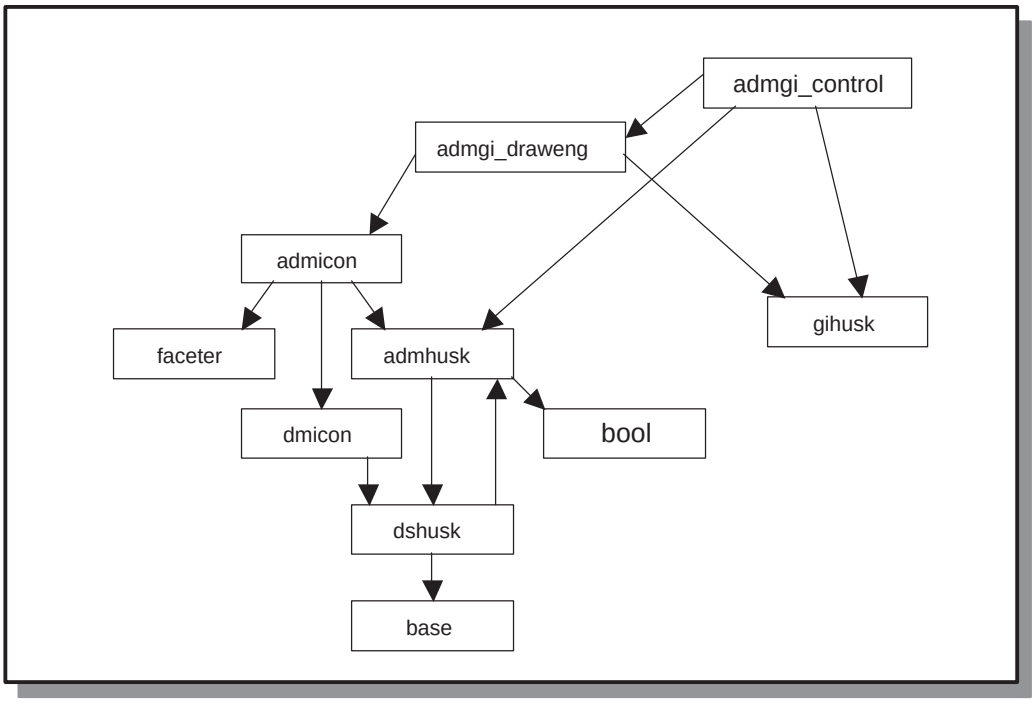


Figure 2-1. Deformable Modeling Library Dependencies

Interface Classes

Topic: Deformable Surfaces

Interface classes are the glue holding libraries together. Interface classes allow libraries to be replaced independently, using derivation. The optional drawing libraries implement 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.

Additionally, the deformable modeling kernel has command objects that allow an application to broadcast commands to all or selected icons in a deformable modeling hierarchy. These are supported by the DM_icon Draw and Set_state methods. There are also command objects for queries, which are supported by the DM_icon Query method.

ADM_draw_engine

The ADM_draw_engine interface class provides abstract draw primitive methods to icons: Draw_point, Draw_polyline, and Draw_mesh. This decouples the icon library from rendering context specifics, such as OpenGL or DirectX commands.

ADM_regobj

The ADM_regobj interface class provides abstract registration methods to the ADM kernel, allowing a deformable modeling hierarchy to register with a view controller. The ADM_regobj contract with the ADM kernel has 3 parts:

- The deformable modeling hierarchy, encapsulated by ATTRIB_DM2ACIS, owns the ADM_regobj object. Registration of ATTRIB_DM2ACIS with the view controller happens at creation, and deregistration happens at destruction. Registration typically involves passing callback and other ATTRIB_DM2ACIS data to the view controller.
- ADM_regobj supports ACIS roll by its Activate and Deactivate methods. When a copy of an ATTRIB_DM2ACIS is placed on a bulletin board, its ADM_regobj's Deactivate method is called so it is no longer drawn; similarly for the ADM_regobj's Activate method, when an ATTRIB_DM2ACIS is restored from a bulletin board.
- ADM_regobj supports ACIS roll by its Make_backup_copy method. When a copy of an ATTRIB_DM2ACIS is placed on a bulletin board, the copy's ADM_regobj pointer is set to the value returned by the original ATTRIB_DM2ACIS's ADM_regobj::Make_backup_copy method. This allows implementations to store data through the ACIS roll.

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.

Draw Command Pipeline

Topic: Deformable Surfaces

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

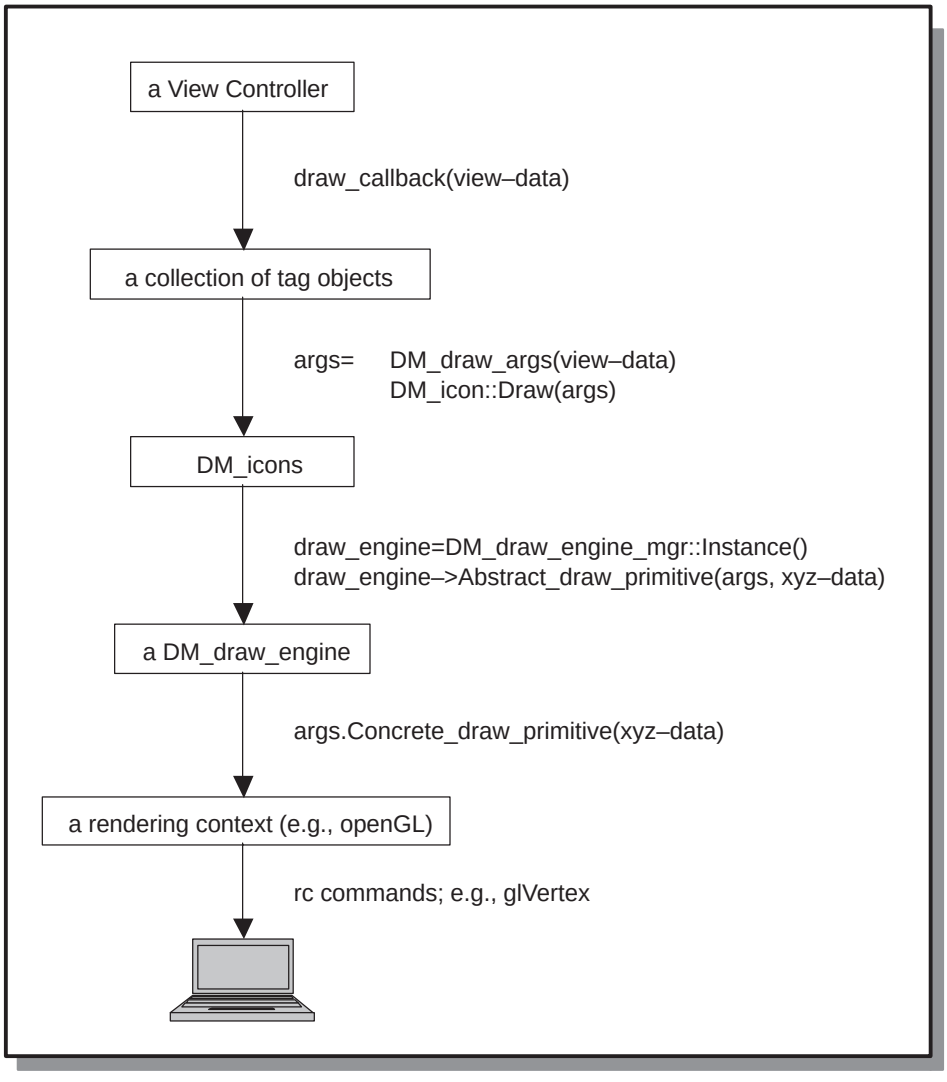


Figure 2-2. Draw Command Pipeline

Data Management

Topic: *Deformable Surfaces

The primary effect of ADM is to modify the control point locations of a spline surface or an interpolated curve associated with the geometry of an ACIS face or edge object. New control point locations are the result of a ADM model calculation. The data needed to represent the ADM model and to make the deformation calculations are stored as attributes associated with the faces or edges being sculpted. The principal attribute classes used to store this information are ATTRIB_DM2ACIS, ATTRIB_DSGROUP, and ATTRIB_DSMODEL.

The ATTRIB_DM2ACIS class is documented and its methods comprise the programmable interface for deformable modeling. The ATTRIB_DM2ACIS class has two classes derived from it: ATTRIB_DS2ACIS and ATTRIB_DC2ACIS for surfaces and curves, respectively. The classes ATTRIB_DSGROUP, ATTRIB_DSMODEL (and the associated base classes ATTRIB_DSLOAD and ATTRIB_DSCSTRN and their derived types) are not intended to be accessed directly by an application programmer. These private classes are used to support the persistent storage of ADM data and are not used to support any deformable modeling functionality. The names of these private attribute classes are listed in this document, because their names appear within the part backup SAT files.

The ATTRIB_DM2ACIS class, derived from ATTRIB, is used during sculpting to hold an expanded version of the ADM data. It includes an independent copy of the spline or interpolated curve geometry data and the ADM graphics, equations, loads and constraints. When sculpting is initiated, an instance of the ATTRIB_DM2ACIS class is made by the function api_get_attrib_dm2acis. Any changes made to the deformable model shape through the ATTRIB_DM2ACIS methods are made to the copy of the spline or interpolated curve geometry and not to the face's original spline geometry. When sculpting is completed, the ATTRIB_DM2ACIS instance is deleted by a call to the function api_remove_attrib_dm2acis. Information in the ATTRIB_DM2ACIS object is not saved nor is it backed up.

Whenever desired, the shape stored in the ATTRIB_DM2ACIS instance can be copied back into the geometry of the edge or face with the function api_commit_attrib_dm2acis. Additionally the commit function creates and associates to the edge or face an instance of the ATTRIB_DSGROUP which marks the face or edge as having a deformable model. The commit function also creates an ATTRIB_DSMODEL attribute and associates it to the ATTRIB_DSGROUP attribute. The ATTRIB_DSMODEL contains a persistent version of the ADM data. It contains the small subset of the data in ATTRIB_DM2ACIS that is sufficient to recreate the ATTRIB_DM2ACIS instance at the time of the commit. The ATTRIB_DSGROUP and ATTRIB_DSMODEL instances are automatically saved, restored, and backed up along with the edge to which they are attached. Changes made by the commit function may be rolled back. Sculpting can also be resumed from the point of the last commit.

When the api_get_attrib_dm2acis function is called, it first attempts to return any ATTRIB_DM2ACIS instance it can find attached to the target entity. Thus, during a sculpting session, the function returns the object which describes the current sculpting state of the face or edge. When no ATTRIB_DM2ACIS is attached to the face or edge, the function creates and returns one. When there is no ATTRIB_DSMODEL associated with the face or edge, the function creates an ATTRIB_DM2ACIS instance with default values, no loads, and no constraints. This happens the very first time sculpting of an entity is attempted. When there is an ATTRIB_DSMODEL associated with the face or edge, the function creates an ATTRIB_DM2ACIS instance built using the saved values. In this case, the sculpting session resumes in the state saved by the last commit. During a sculpting session, the initial state of sculpting can be restored by ending the sculpting session without executing a commit and beginning all over again.

Load and constraint persistent data storage is handled by associating an attribute derived from the base classes ATTRIB_DSLOAD and ATTRIB_DSCSTRN to the ATTRIB_DSMODEL for each load or constraint in the model. Currently, the list of load and constraint attribute class derived types includes the following:

ATTRIB_PT_CSTRN . . . . . For point constraints.

ATTRIB_CRV_CSTRN . . . . . For curve constraints.

ATTRIB_PT_PRESS . . . . . For point pressure loads.

ATTRIB_DIST_PRESS . . . . . For distributed pressure loads.

ATTRIB_DS_SPRING . . . . . For spring loads.

ATTRIB_SPRING_SET . . . . . For a list of spring loads.

ATTRIB_CRV_LOAD . . . . . For curve to curve spring loads.

ATTRIB_VECTOR_LOAD . . . . . For vector loads.

ATTRIB_ATTRACTOR . . . . . For attractor loads.

The following figure illustrates the ADM data layout throughout the various stages of a typical surface sculpting sequence without the use of patches.

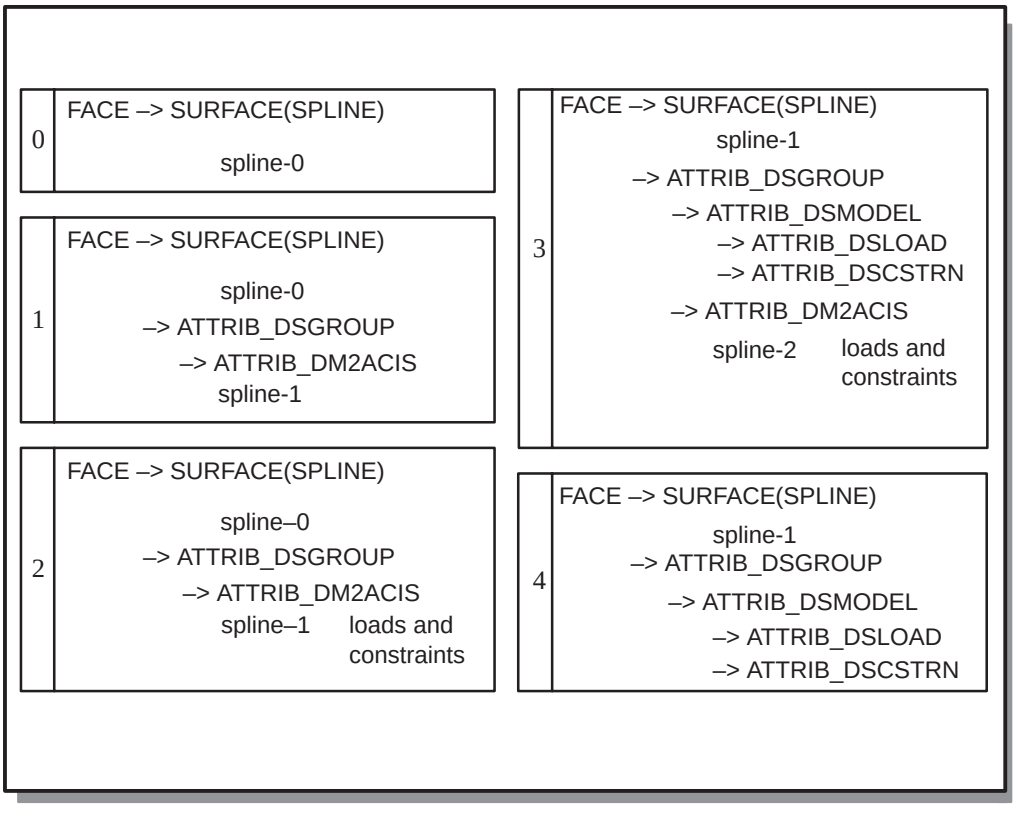


Figure 2-3. Data Organization through a Sculpting Session

Sculpting begins by selecting a face. The face is typically a part of a larger sheet or solid body.

1. Executing the ADM API initialization adds an ATTRIB_DSGROUP attribute to the face which has an ATTRIB_DM2ACIS attribute that holds a copy of the spline geometry to be edited during sculpting.

2. During the sculpting session, loads and constraints may be added and the shape of the copied spline is modified.

3. Executing the ADM API commit function moves the ATTRIB_DM2ACIS copied spline shape back into the face and attaches a ATTRIB_DSMODEL attribute to the face's ATTRIB_DSGROUP attribute. The ATTRIB_DSMODEL attribute contains a persistent set of data and attributes describing deformable modeling parameters, loads and constraints at the time the commit function was executed. This step may be undone with roll back.

4. Ending the sculpting session with the ADM API end command removes the ATTRIB_DM2ACIS attribute from the face. The next time sculpting begins, the ATTRIB_DM2ACIS object will be created from the face's current spline shape and the data stored in ATTRIB_DSMODEL and its attributes.

For local deformations, the topology of the ACIS model is modified every time a patch is added or removed. Each patch imprints a new face or edge onto the existing ACIS entity creating a quilt of faces and boundaries that are equivalent to the original starting face. ATTRIB_DM2ACIS and ATTRIB_DSGROUP attributes are attached to every entity within the quilt to persistently store the parent child relationships of the ADM hierarchical patch model. Selecting any face or edge member of such a quilt for sculpting results in the sculpting of the entire quilt.

The programming interface consists of API functions and a family of sculpting methods supported by the ATTRIB_DM2ACIS C++ object. The API functions include:

- Initialize function that extracts data from ACIS models to build ADM models
- Commit function which places sculpting results back into the source ACIS model
- End sculpting function that deletes the ADM model after it is no longer needed
- Add patch function that creates child patches on their parent patches
- Copy patch function that creates new child patches as copies of existing patches
- Remove patch function that removes and deletes child patches from their parents
- Function that sets the tolerance level used when enforcing curve constraints

The tolerance function is always called by the initialize function to force the ADM system to make its calculations to the same tolerance level being used by the ACIS package. This will probably never be called directly by an application programmer.

The second ADM interface consists of a set of Scheme extensions that expose the API functions and ADM methods to the Scheme interpreter.

The third ADM interface consists of a set of Scheme files which build a ADM rubberband capability. This allows users to experiment with mouse based sculpting and a language of one– and two–letter sculpting commands to simplify exercising the ADM Scheme extensions. Additionally, a small set of canned demonstrations are provided which run simple sequences of the sculpting commands for both deformable curves and surfaces.

Multi–Surface ACIS Interface

Topic: *Deformable Surfaces

The strategy for building a multi–surface mesh in the ACIS integration is based on listing the faces within the mesh. The integration code is smart enough to walk all the topology pointers and to add the link constraints for every edge between two faces in the mesh.

The API function to build a multi–surface mesh is api_dm_add_multi_face. Its two principle arguments include a pointer to an existing ATTRIB_DM2ACIS object (refer to api_dm_get_attrib_dm2acis) and a pointer to the FACE to be added to the mesh. A multi–surface mesh is created by calling the api_dm_add_multi_face function just once for every face in the mesh. The faces may be added in any order as long as each face being connected to the mesh is connected to at least one other face already in the mesh. The api_dm_add_multi_face function ensures that the input face is converted to a spline and then calls the method ATTRIB_DM2ACIS:: Add_multi_face.

The API function api_dm_rm_multi_face can be used to remove faces from the mesh. Once the mesh is created, ADM is executed by calling the methods on the ATTRIB_DM2ACIS object. The root–sibling list may be walked by calling the method ATTRIB_DM2ACIS::Sibling. The deformed shape can be computed by calling ATTRIB_DM2ACIS::Solve_dmod.

The API function api_dm_add_multi_face creates C0 connectivity link loads for every edge connecting the input face to the existing mesh. To modify the behavior of the link to C1 connectivity or to any of the other possible states, use the call ATTRIB_DM2ACIS::Set_cstrn_behavior. The behavior state for an existing link constraint can be queried at run time with the call ATTRIB_DM2ACIS::Get_cstrn_behavior. These commands take and return behavior bit arrays as described in the API interface section.

Scheme Interface

Topic: *Deformable Surfaces

The multi–surface entry point is the Scheme extension ds:link–face. The strategy for its use is the same as for the ACIS programmable interface. To build a multi–surface mesh call ds:link–face once for every face to be added to the mesh. Once again the limitation applies that each face being added to the mesh must be connected to at least one other face already in the mesh. The tag numbers for the root–siblings are returned by the ds:link–face call and can be gotten at run time by parsing the return argument of the Scheme extension ds:get–tag–summary.

The Scheme extension ds:link–face exposes the behavior of the api_dm_add_multi_face call to ACIS. Link constraints are created with C0 connectivity. A link constraint's behavior can be modified with the Scheme extension ds:set–cstrn–behavior. The Scheme extension parses a text string as described in the on–line documentation to build the required behavior bit array.

When using the ds:mouse rubberband driver, a link constraint can be toggled between C0 and C1 behaviors by simply clicking on the constraint with <Shift> + <Right Mouse>.

Deformations at Curve Constraint Corners

Topic: *Deformable Surfaces

Wherever a pair of curve constraints intersect one another to form a corner, the surface normal is fixed. For example, consider a simple planar square sheet constrained along its borders and loaded with a distributed pressure. As the gain of the pressure is increased, the planar face begins to bulge and eventually can be made to inflate like a balloon. The surface deformation is smooth everywhere, except at the corners where two curve constraints meet. At these corners, the surface bends sharply to preserve the original surface normal.

The reason for this has to do with differential geometry and not deformable surfaces. For the points in a curve constraint, the curve constraint is equivalent to both a position constraint and a single tangent constraint in the direction of the curve. For most of the points in a curve constraint, the surface may pivot about the curve constraint much like the pages of a book can pivot about the book's binding. However, those points which lie on the intersection of two curve constraints effectively have two tangent constraints. And any point in any smooth surface with two fixed tangents has a fixed surface normal.

The corner behavior can be avoided by avoiding corners. Use fillets and blends to remove corners in constraint curves before sculpting. Without the corners, the surface is able to deform at every point within the surface.

Scheme Language Extension Concepts

Topic: *Deformable Surfaces

The Scheme language extensions provided have been designed to expose the complete ADM functionality to the Scheme interpreter. Each Scheme extension name is prefixed with the characters "ds:" and the entire list of extensions can be seen interactively by typing (apropos "ds:"), if the apropos procedure has been loaded. For the user's convenience, the ADM Scheme extensions output a description of the required argument types when called with no arguments. The first argument of most of these extensions is the face or edge being sculpted. The extensions call the API initialize function when necessary, freeing the user from having to explicitly initiate sculpting.

The argument types for the ADM Scheme extensions include the standard data types of real, integer, entity, logical, and position. In addition, the extensions expect to see tags and par_pos types.

Scheme Demonstration Files

Topic: *Deformable Surfaces

The files dsdemo.scm, dscurve.scm, and dsmouse.scm have been provided to simplify exercising ADM through Scheme. The file dsdemo.scm defines a set of simple commands which access the rather lengthy Scheme extension names and provides a short set of canned demonstrations for sculpting surfaces which can be launched by typing single commands. The file dscurve.scm provides another short set of canned demonstrations for sculpting curves. The file dsmouse.scm shows an example of mouse based sculpting.

ds–model Variable

Topic: *Deformable Surfaces

All the functions in dsdemo.scm and dscurve.scm operate on the same ACIS entity, identified as ds–model. Any faces made by the dsdemo.scm file and any edges made by the dscurve.scm file are automatically bound to the ds–model global variable. Users may point the dsdemo.scm and dscurve.scm functions to any face or edge by binding the ACIS entity to the ds–model variable using the Scheme extension (set! ds–model face–or–edge–entity), where face-or–edge–entity is the new face or edge. All subsequent use of the dsdemo.scm and dscurve.scm functions act on the specified entity.

ds:mouse

Topic: *Deformable Surfaces

The mouse based sculpting demonstration is initiated by the Scheme extension ds:mouse. The ds:mouse call binds a face to the mouse based sculpting functions and outputs the following function table as a reminder on how to use the mouse.

Bind a load to the slider function with:

```
(ds:set-mouse-slider tag gain_inc)
```

Release select keys after mouse moves begin.

	Left Button	Right Button	Left and Right Buttons
Plain	RotView	ZoomView	PanView
Shift	Track Control Point Z	Add/Toggle Control Point	
Shift Drag		Add Patch	
Control	Track Control Point XY	Add Point Pres-sure	
Shift Control	Slide Gain	Remove Tag	

Figure 2-4. Mouse Function Map

The map in Figure 2-4 summarizes which functions are bound to different combinations of key strokes and mouse buttons. Those functions and their use are described in the following sections.

View Manipulation

Topic: *Deformable Surfaces

Mouse button click–and–drag operations are dedicated to view manipulations. Clicking and dragging with the left mouse button rotates the viewing angle. The right mouse button click–and–drag is used for zooming. Lowering the mouse cursor in the view window increases the apparent size of the object, and raising the mouse decreases its apparent size. Clicking and dragging with both the left and right mouse buttons causes the view image to pan across the screen.

Additional mouse functions are used by clicking mouse buttons and dragging the mouse in combination with depressing the <Shift> and <Ctrl> keys. Due to the event driven nature of mouse I/O, a dramatic improvement in response time can be obtained if you release the <Shift> and <Ctrl> keys after you begin dragging the mouse. Keeping these keys depressed while dragging the mouse causes the system to process more events.

Constraint Point Tracking

Topic: *Deformable Surfaces

<Shift> + <Left Mouse> is used for moving constraint points. Constraint points are rendered in red and are highlighted when the mouse cursor is pointing to one. Hold the <Shift> key down, move the cursor over a constraint point, and depress the left mouse button. The control point begins to track the mouse moving in the *z* direction. Use <Ctrl> + <Left Mouse> to move a constraint point in the *x* and *y* directions.

Add and Toggle Constraints

Topic: *Deformable Surfaces

<Shift> + <Right Mouse> is used four ways: to add constraint points, to toggle constraint points and curves between enabled and disabled states, to add child patches, and to toggle seam constraints attaching a child to a parent patch between C0, C1, and C2 continuity. Hold the <Shift> key down, move the cursor to a position over the deformable surface and depress the right mouse button. If the cursor is highlighting a constraint, that constraint will be toggled between an enabled and a disabled state. Enabled constraints are turned on and disabled constraints are turned off. Some curve constraints cannot be disabled. When the mouse is used to attempt to disable such a curve constraint a warning message is printed to the screen and the curve constraint is not disabled.

When the constraint is a seam for a patch, it toggles between C0, C1, and C2 continuity. When no constraint is highlighted, a line running through the cursor point in the viewing direction is intersected with the deformable model. A constraint point is added at the first intersection.

Add Patch

Topic: *Deformable Surfaces

<Shift> + <Right Mouse> with drag is used to add patches. Hold the <Shift> key down, move the cursor to a position over the deformable model and depress the right mouse button. While holding all keys, drag the mouse cursor to a second point over the deformable model and release the mouse key. The two locations identified by the mouse down and the mouse up events define the corners of a square patch to add to the deformable model. If the mouse is not dragged, it acts to just add and toggle point constraints as described in the section above. Use the Scheme extension ds:set–draw–state to render the various properties of the newly created patch.

Add Point Pressure Loads

Topic: *Deformable Surfaces

<Ctrl> + <Right Mouse> is used for adding point pressure loads. Holding the <Ctrl> key down, move the cursor to the desired location, and depress the right mouse button. A point pressure load is added at the intersection point between a line running through the cursor position and the deformable surface.

Delete Tag Objects

Topic: *Deformable Surfaces

<Shift> + <Ctrl> + <Right Mouse> is used for deleting tag objects. Hold both the <Shift> and <Ctrl> keys down, move the cursor until a constraint or load is highlighted, and depress the right mouse button. The highlighted tag object is deleted from the deformable surface. Some curve constraints cannot be deleted. When the mouse is used to attempt to delete such a curve constraint, a warning message is printed to the screen and the curve constraint is not deleted.

Load Gain Tracking

Topic: *Deformable Surfaces

<Shift> + <Ctrl> + <Left Mouse> is used as a slider bar to adjust the gain of a load. Hold the <Shift> and <Ctrl> keys down, and depress the left mouse button. Release the <Shift> and <Ctrl> keys and drag the mouse. As the cursor moves left and right, the gain of one tag object is decreased and increased. The tag object is selected by the tag number associated with the global variable data–ds:mouse–slider–tag and the amount the gain is increased for each move of the mouse is stored in data–ds:mouse–slider–gain–inc. The two global variables may be set with the Scheme extension ds:set–mouse–slider, as

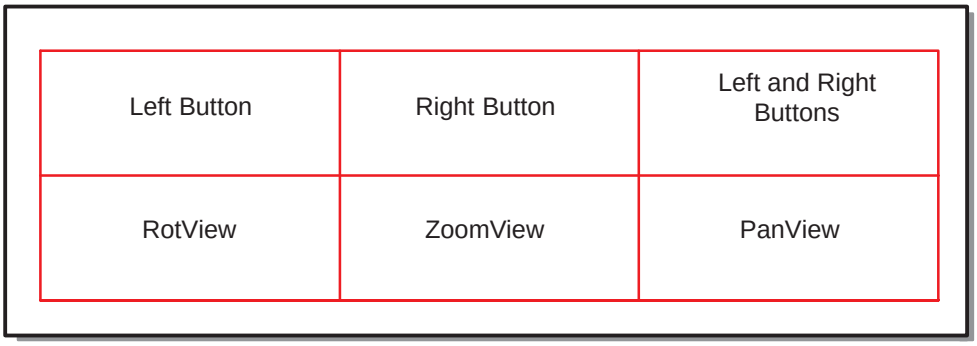
```
(ds:set–mouse–slider tag gain_inc)
```

where tag is the tag number and gain_inc is the amount the gain is increased.

View Manipulations Only

Topic: *Deformable Surfaces

Calling the ds:mouse extension with a null argument as, (ds:mouse '( )), binds only the view manipulation functions to the mouse and prints out the following help message:


Left Button
Right Button
Left and Right Buttons
RotView
ZoomView
PanView