Chapter 5.
Functions DM Ja thru Zz

Topic: Ignore

DM_journal_off

Function: Deformable Modeling, DML Create and Query

Action: Closes a journal file and stops further journaling of DM API calls.

Prototype:
```
void DM_journal_off (
    int& rtn_err,            // out: 0=success or
                             // neg err code
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function modifies the library global values DM_journal_file, DM_journal, and DM_cascade.

This function ends the recording of a DM API journal. DM API call journaling is a debugging tool for customers who run the DM API without ACIS.

This function closes any currently open DM API journal file and sets the global variables DM_journal_file, DM_journal and DM_cascade to stop journaling all future DM API calls.

When journaling is currently disabled, this function takes no action.

Errors: None

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_journal_on

Function: Deformable Modeling, DML Create and Query

Action: Enables journaling and opens a file for write.

Prototype:

```
void DM_journal_on (
    int& rtn_err,           // out: 0=success or
                            // neg err code
    char* file_name,        // file to write
                            // max sized:[256]
    int cascade             // 0=only journal DM API
        = 0,                // entry calls
                            // 1=journal entry and
                            // cascading DM calls
                            // 2=journal entry and
                            // xsect callbacks
                            // 3=journal entry,
                            // cascading DM calls
                            // and xsect callbacks
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function modifies the library global values DM_journal_file, DM_journal, and DM_cascade.

A call to this function begins a DM API journal session. The call opens for write the file indicated by the input filename and sets the global variables DM_journal and DM_cascade to begin journaling all future DM API calls. DM API call journaling is a debugging tool for DM customers who do not run ACIS. When DM_journal is set to 1, every call to a DM API routine generates an input report listing all the input variable values when the routine was entered. The DM API routines generate an output report listing all the output variable values just before the routine exits.

The cascade argument controls how much information gets written to the journal file as shown in the following table.

The cascade argument controls how much information gets written to the journal file. When the value of cascade is:

0 = all entry level DM calls are journaled.
1 = all entry level and cascaded DM calls are journaled.
2 = all entry level and intersection callback functions are journaled.
3 = all entry level DM calls, cascaded DM calls, and intersection callback functions are journaled.

When this function is called with journaling enabled, it closes the current journal file and opens a new one. Opening a journal file with the same name as the current journal file will cause an overwrite of the old journal file data.

Errors: DM_NULL_INPUT_PTR
The filename cannot be NULL on input

DM_BAD_CASCADE_VALUE
The cascade value must be within the range 0 to 3

DM_FAILED_FILE_OPEN_WRITE
The designated file must be opened for write

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_journal_play

Function: Deformable Modeling, DML Create and Query

Action: Replays a sequence of DM API calls from a DM API journal file for debugging purposes only.

Prototype:

```
void DM_journal_play (
    int& rtn_err,           // out: 0=success or
                            // neg err code
    char* file_name,        // name of file to read
                            // max sized:[256]
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function modifies the library global values DM_journal_file, DM_journal, and DM_cascade.

This function controls the opening and replaying of all the DM API calls held within a DM API call journal file. DM API call journaling is a debugging tool for DM customers who do not run run ACIS.

This function opens the indicated file and checks to see if it is a journal file. If so it replays all the recorded entries. If not, it returns an error code. When journaling is currently enabled, this function journal file before opening the new file. During replay, if all the returned arguments for every replayed DM API calls are the same as the journal entries then a 0 is returned; otherwise, a negative error code is returned. This replay feature can be used to compare the behavior of the DM API as integrated within two different applications and/or interfaced to two different geometric modelers.

Errors: DM_NULL_INPUT_PTR
The filename cannot be NULL on input.

DM_BAD_CASCADE_VALUE
The cascade value must be within the range 0 to 3.

DM_FAILED_FILE_OPEN_READ
The designated file must be opened for read.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_bspline_curve

Function: Deformable Modeling, DML Create and Query

Action: Creates a B–spline curve from data and returns a pointer to the DS_pfunc B–spline curve or NULL.

Prototype:

```
DS_pfunc* DM_make_bspline_curve (
    int& rtn_err,           // out: 0=success or a
                            // negative err code
    int image_dim,          // in: image space size
                            // (2=2d,3=3d)
    int degree,             // in: polynomial degree
    int dof_count,          // in: control point
                            // count
    int knot_count,         // in: number of distinct
                            // knot values
    int* knot_index,        // in: multiple knot
                            // count array.
                            // sized:[knot_count]
                            // specify the multiple
                            // knots by setting
                            // count[i] = maximum
                            // knot index value for
                            // location knot[i]
                            // Examples:
                            // no multiples count
                            // = [0,1,2,3,4]
                            // some multiples
                            // = [1,2,4]
    double* knot,           // in: knot value array
                            // sized:[knot_count]
                            // ordered [u0 < u1 <
                            // u2 ... ]
    double* dof_vec,        // in: init dof_vec vals
                            // or NULL for 0
                            // sized:[image_dim*
                            // dof_count]
                            // ordered:[xyz0, xyz1,
                            // xyz2, ...]
    double* dof_def,        // in: init default shape
                            // or NULL for 0
                            // sized:[image_dim*
                            // dof_count]
                            // ordered:[xyz0, xyz1,
                            // xyz2, ...]
    int end_cond            // in: enforce continuity
        = 0,                // between end points
                            // with constraints. If
                            // used must have
                            // multiple knots on
                            // the ends of the
```

```
                                    // B–spline. 0=open
                                    // (no constraints)
                                    // 1=closed (C0
                                    // continuity)
                                    // 2=periodic (C1
                                    // continuity)
                                    // default value = 0
    SDM_options* sdmo               // in:SDM_options pointer
        = NULL                      // note: total_knot_count
                                    // = dof_count+degree–1
                                    // the total and distinct
                                    // knot counts vary when
                                    // there are multiple
                                    // knots
                                    // note: the input array
                                    // values are only
                                    // copied. The calling
                                    // program still needs to
                                    // manage that memory
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies return_err.

Returns a pointer to the allocated and initialized memory used to represent a B–spline curve as a DS_pfunc description and sets the rtn_err as described below. Call DM_delete_pfunc() on the returned pointer to free its memory. All inputs are copied, not nested, within the constructed DS_pfunc.

Creates and returns a pointer to the DS_pfunc B–spline curve representation. image_dim is the size of the model space, using 2 for 2D and 3 for 3D. degree is the degree of B–spline polynomials. dof_count is the number of B–spline control points. knot_count is the number of unique knot values. The total_knot_count equals dof_count + degree – 1.

The total_knot_count will not equal the distinct knot_count when the B–spline has any multiple knots. The multiplicity of each knot is communicated through the knot_index array. Each knot, starting with the lowest *u* value knot, is assigned a knot_index starting at 0. The knot_index array records the highest index value for all knots at each distinct knot location. For example, a common B–spline is one with multiple knots on the end points and single knots in the interior. A map of the knot index values might look like:

```
+------+------+-------+------+
0       3       4       5       6
1                                7
2                                8
```

This knot array corresponds to a degree = 3 B–spline with seven control points and four spans. The B–spline interpolates the first and last control point positions due to the multiple end knots.

There are nine total knot values and only five distinct knot values. The knot index array for this curve would be [2,3,4,5,8]. The knot array contains the domain space values (u values) for each distinct knot. For the above example, the knot array might be something like [0.00, 0.25, 0.50, 0.75, 1.00] for an even distribution of knot values from 0 to 1.

Other common B–spline representation schemes store an extra unused knot at each end of the knot array. These extra knot values seem to appear in all the B–spline evaluation functions but in reality are never used and are not input to the deformable modeling package. Developers moving B–spline data between this and other systems must take care to watch out for this mismatch possibility on the end knots.

Deformable curves have both a current shape and a default shape. The default shape is stored as a second array of control points called dof_def. dof_def is sized and stored in the same manner as dof_vec. end_cond is a flag which can be used to have the deformable modeling package force the two ends of a B–spline curve to be continuous with one another. when end_cond equals 0 the curve remains open, when end_cond = 1 it will be C0 across the end points, and when end_cond = 2 it will be C1 across the end points.

Errors: DM_BAD_IMAGE_DIM_VALUE
The input value of the image dimension must be positive.

DM_BAD_DEGREE_VALUE
The requested spline degrees must be 0 or greater.

DM_BAD_KNOT_COUNT_VALUE
The number of knots must be positive.

DM_NULL_INPUT_PTR
The knot_index and the knot_array pointers cannot be NULL on input.

DM_BAD_KNOT_TO_CPT_COUNT
The last knot index number must equal dof_count + degree – 2.

DM_BAD_NTGRL_DEGREE_VALUE
The required gauss integration degree is not supported. Currently limited to the range 1 to 79. The ntgrl_degree must be equal to or larger than twice the spline degree for valid deformations.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_bspline_surface

Function: Deformable Modeling, DML Create and Query

Action: Creates a B–spline surface from data and returns a pointer to DS_pfunc B–spline surf or NULL.

Prototype:

```
DS_pfunc* DM_make_bspline_surface (
    int& rtn_err,           // out: 0=success or
                            // negative err code
    int image_dim,          // in: image space size
                            // (2=2d,3=3d)
    int degree_u,           // in: u dir
                            // polynomial degree
    int dof_count_u,        // in: u dir control
                            // point count
```

```
                        int knot_count_u,       // in: u dir distinct
                                                // knot value count
                        int* knot_index_u,      // in: u dir multiple
                                                // knot count array.
                                                // sized:[knot_count]
                                                // specify the multiple
                                                // knots by setting
                                                // count[i] = maximum
                                                // knot index value for
                                                // location knot[i]
                                                // Examples:
                                                // no multiples count =
                                                // [0,1,2,3,4]
                                                // some multiples =
                                                // [1,2,4]
                        double* knot_u,         // in: u dir knot values
                                                // sized:[knot_count]
                                                // ordered [u0<u1<u2...]
                        int degree_v,           // in: v dir polynomial
                                                // degree
                        int dof_count_v,        // in: v dir control
                                                // point count
                        int knot_count_v,       // in: v dir distinct
                                                // knot value count
                        int* knot_index_v,      // in: v dir multiple
                                                // knot count array.
                                                // sized:[knot_count]
                                                // specify the multiple
                                                // knots by setting
                                                // count[i] = maximum
                                                // knot index value for
                                                // location knot[i]
                                                // Examples:
                                                // no multiples count =
                                                // [0,1,2,3,4]
                                                // some multiples =
                                                // [1,2,4]
                        double* knot_v,         // in: v dir knot value
                                                // array
                                                // sized:[knot_count]
                                                // ordered [u0<u1<u2...]
                        double* dof_vec,        // in: init dof_vec vals
                                                // or NULL for 0
                                                // sized:[image_dim*
                                                // dof_count_u*
```

```
                                    // dof_count_v]
                                    // ordered:[C00, C01, ...
                                    // C0m, C10, ... Cnm]
    double* dof_def,                // in: init default shape
                                    // or NULL for 0
                                    // ordered:[C00, C01, ...
                                    // C0m, C10, ... Cnm]
    int end_cond_u                  // in: one of 0=open or
        = 0,                        // 1=closed or 2=periodic
    int singular_u                  // in: one of 0=none or
        = 0,                        // 1=low or 2=high or
                                    // 3=both
    int end_cond_v                  // in: one of 0=open or
        = 0,                        // 1=closed or 2=periodic
    int singular_v                  // in: one of 0=none or
        = 0,                        // 1=low or 2=high or
                                    // 3=both
    SDM_options* sdmo               // in:SDM_options pointer
        = NULL                      // note: total_knot_count
                                    // = dof_count+degree–1
                                    // the total and distinct
                                    // knot counts vary when
                                    // there are multiple
                                    // knots
                                    // note: the input array
                                    // values are only
                                    // copied. The calling
                                    // program still needs to
                                    // manage that memory
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”dshusk/dskernel/dmapi.hxx”
#include ”dshusk/dskernel/dspfunc.hxx”
#include ”dshusk/dskernel/sdm_options.hxx”
```

Description: Modifies return_err.

Creates and returns a pointer to DS_pfunc B–spline surface representation. A B–spline surface is the tensor product of the underlying B–spline u–curve and v–curve basis functions. The total number of dofs in the surface is just the product of the dof counts for *u* and *v* curve basis functions. The global dof numbers for each dof are the cross product of the u–curve and v–curve dof numbers with the v–curve dof numbers varying the fastest. This function's input arguments contain enough information to make both the u–curve and the v–curve basis functions as well as the dof values for the surface.

image_dim is the size of the model space, using 2 for 2D and 3 for 3D. degree_u and degree_v are the B–spline polynomial degrees in the surface's *u* and *v* directions. dof_count_u and dof_count_v are the number of B–spline control points in the surface's *u* and *v* directions. dof_count_u and dof_count_v are the number of B–spline control points in the surface's *u* and *v* directions. The knot_count_u and knot_count_v are the number of unique knot values in the surface's *u* and *v* directions.

The total_knot_count equals the dof_count + degree – 1 for each *u* and *v* surface direction. The total_knot_count will not equal the distinct knot_count when the NURB has any multiple knots. The multiplicity of each knot is communicated through the knot_index array. Each knot, starting with the lowest *u* value knot, is assigned a knot_index starting at 0. The knot_index array records the highest index value for all knots at each distinct knot location. For example, a common B–spline is one with multiple knots on the end points and single knots in the interior. A map of the knot index values might look like:

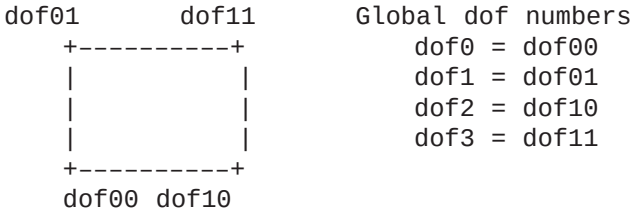
```
+------+------+-------+------+
0       3       4       5       6
1                                7
2                                8
```

This knot array corresponds to a degree = 3 B–spline with seven control points and four spans. The B–spline interpolates the first and last control point positions due to the multiple end knots.

There are nine total knot values and only five distinct knot values. The knot index array for this curve would be [2,3,4,5,8]. The knot array contains the domain space values (u values) for each distinct knot. For the above example, the knot array might be something like [0.00, 0.25, 0.50, 0.75, 1.00] for an even distribution of knot values from 0 to 1.

Other common B–spline representation schemes store an extra unused knot at each end of the knot array. These extra knot values seem to appear in all the B–spline evaluation functions but in reality are never used and are not input to the deformable modeling package. Developers moving B–spline data between this and other systems must take care to watch out for this mismatch possibility on the end knots.

The dof_vec contains the surface's control point locations and is sized [image_dim*dof_count_u*dof_count_v]. It is ordered by control point, so a two dimensional surface with a 2x2 grid of control points will be organized as:



and will have a dof_vec stored as [x0, y0, x1, y1, x2, y2, x3, y3], and the same surface in 3d would be stored as, [x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3]. Deformable surfaces have both a current shape and a default shape. The default shape is stored as a second array of control points called dof_def. dof_def is sized and stored in the same manner as dof_vec.

Surfaces can have three different end conditions applied to them through the use of constraints. These include an end_cond which can force opposite edges of the surface square to connect to each other with either a C0 or a C1 continuity, and a singularity case which can collapse an entire boundary edge into a single point. These end conditions are communicated through the use of four flags, end_cond_u, end_cond_v, singular_u, and singular_v. The connectivity of the *u* and *v* boundary curves are set independently by the end_cond_u and end_cond_v flag values. When end condition equals 0 the surface remains open in that direction. When end condition equals 1 it will be C0 across the end curves. And when end condition equals 2 it will be C1 across the end curves. The singularity of the *u* and *v* boundaries are set independently by the singular_u and singular_v flag values. When singular equals 0 all edges are normal. When singular equals 1 the low end boundary curve is collapsed to a point. When singular equals 2 the high end boundary curve is collapsed to a point. When singular equals 3 both low and high end boundary curves are collapsed to a single point.

Returns a pointer to DS_pfunc when successful or NULL when a problem is encountered in the input.

mallocs: The calling program needs to call DM_delete_pfunc on the returned pointer. The new data structure does not reference the input arrays after this call.

Errors:

DM_BAD_IMAGE_DIM_VALUE
The input value of the image dimension must be positive.

DM_BAD_DEGREE_VALUE
The requested spline degrees must be 0 or greater.

DM_BAD_KNOT_COUNT_VALUE
The number of knots must be positive.

DM_NULL_INPUT_PTR
The knot_index and the knot_array pointers cannot be NULL on input.

DM_BAD_KNOT_TO_CPT_COUNT
The last knot index number must equal dof_count + degree – 2.

DM_BAD_END_COND_VALUE
The end condition values must be 0, 1, or 2.

DM_BAD_SINGULAR_VALUE
The singular values must be 0, 1, 2, or 3.

DM_BAD_NTGRL_DEGREE_VALUE
The required gauss integration degree is not supported. Currently limited to the range 1 to 79. The ntgrl_degree must be equal to or larger than twice the spline degree for valid deformations.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_circ_curve

Function: Deformable Modeling, DML Create and Query

Action: Creates a circular curve and returns a pointer to DS_pfunc circular curve or NULL.

Prototype:

```
DS_pfunc* DM_make_circ_curve (
    int& rtn_err,              // rtn: ptr to DS_pfunc
                               // circ curve or NULL
                               // eff: make crv
                               // W=xc+xa*cos(s)
                               // +xb*sin(s) with
                               // 0 <= s <= 2 pi
                               // out: 0=success or
                               // negative err code
    int image_dim,             // in: image space size
                               // (2=2d,3=3d)
    double* dof_vec,           // in: the 3 dofs
                               // [xc,xa,xb].
                               // [not-nested]
                               // xc = center_point
                               // xa = vec from
                               // center_pt to axis_1
                               // xb = vec from
                               // center_pt to axis_2
                               // Sets dof_def = dof_vec
                               // sized:[3*image_dim]
    double u_min               // in: domain min
        = 0.0,                 // 0 <= u_min <= 2*DS_PI
    double u_max               // in: domain max
    = 2* 3.1415926535898,      // 0 <= u_max <= 2*DS_PI
                               // note: the input array
                               // values are only
                               // copied. The calling
                               // program still needs to
                               // manage that memory.
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
                               // note: the input array
                               // values are only
                               // copied. The calling
                               // program still needs to
                               // manage that memory.
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function returns a pointer to the allocated and initialized memory used to represent a circular curve as a DS_pfunc description and sets the rtn_err as described below. Call DM_delete_pfunc() on the returned pointer to free its memory.

The circular curve represents complete or arc fragments of circles and ellipses in the form used by the deformable modeling package.

```
W = W(s) = xc + xa*cos(s) + xb*sin(s)
    0 <= u_min <= s <= u_max <= 2 pi
```

The curve has three dofs including

xc = the center–point of the curve,
xa = the vector from the center–point to the end–point of axis 1,
xb = the vector from the center–point to the end–point of axis 2,

Note that *xa* and *xb* are 2D vectors and not 2D points, i.e., the end points of the ellipse vectors are computed by:

end_point_axis_1 = xc + xa, and
end_point_axis_2 = xc + xb.

When
xa = {r, 0} and xb = {0, r} we get a circle
xa = {r1,0} and xb = {0,r2} we get an ellipse.

When the vectors *xa* and *xb* equal each other the circular curve degenerates into a double line segment. This will not bother the deformable modeling package but may confuse the end user and is therefore prohibited. image_dim is the size of the model space. Use 2 for 2D and 3 for 3D.

When u_min equals 0 and u_max equals 2*DS_PI, the circle or ellipse is a complete closed curve. Any other values for u_min and u_max make a circular or elliptical arc fragment.

dof_vec is an array of [*xc,xa,xb*]. It is sized 3*image_dim, where a 2D circ dof_vec array is organized as [*Xxc, Yxc, Xxa, Yxa, Xxb, Yxb*].

mallocs is the calling program needs to call DM_delete_pfunc on the returned pointer. The new data structure does not reference the input arrays after this call.

Returns pointer to DS_pfunc when successful or NULL when a problem is encountered in the input.

Errors: DM_BAD_IMAGE_DIM_VALUE
The input value of the image dimension must be positive.

DM_NULL_INPUT_PTR
The dof_vec pointer cannot be NULL on input.

DM_BAD_ELEM_COUNT_VALUE
The elem_count must be greater than DS_DEFAULT_ELEM_COUNT.

DM_BAD_NTGRL_DEGREE_VALUE
The required gauss integration degree is not supported. Currently limited to the range 1 to 79. The ntgrl_degree must be equal to or larger than the DS_DEFAULT_NTGRL_DEGREE for valid deformations.

DM_BAD_CIRC_SHAPE
The *xa* and *xb* vectors cannot be parallel or of zero length which would create a circular curve with no internal area.

DM_BAD_CIRC_PARAM_VALUE
Input u_min and u_max values must be in the range of 0 to 2*DS_PI.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_dcurve_image

Function: Deformable Modeling, DML Create and Query, DML Graphics

Action: Creates a projected domain curve and returns the projected domain curve or NULL.

Prototype:

```
DS_pfunc* DM_make_dcurve_image (
    int& rtn_err,              // out: 0=success or
                               // negative err code
    int domain_flag,           // in: dcurve
                               // domain_space
                               // 0=original_dmod_space
                               // 2 =
                               // internal_pfunc_space
    double domain_scale,       // in: used to map
                               // internal_pfunc_space
                               // = domain_scale *
                               // original_dmod_space
    DS_pfunc* dcurve,          // in: domain curve to be
                               // projected
    DS_pfunc* surface,         // in: surface defining
                               // curve projection
    double tolerance,          // in: accuracy of the
                               // output curve to actual
                               // projection as tol >=
                               // dist_between(W
                               // (C(s)),W(s)) the max
                               // dist between the
                               // projection curve and
                               // any projected points.
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
                               // note:
                               // dcurve->Image_dim =
                               // surface->Domain_dim
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies return_err.

Builds and returns the pointer to a DS_pfunc curve shape which approximates the projection of the input dcurve into image space through the shape of the surface.

When domain_flag == 0, the dcurve is assumed to exist in the orig_dmod_space. When domain_flag == 2, the dcurve is assumed to exist in the internal_pfunc_space. When domain_flag is 0, the dcurve is projected from the orig_dmod_space to the internal_pfunc_space, prior to being projected into the image_space through the surface pfunc using the simple scaling relationship,

internal_pfunc_space = domain_scale * orig_dmod_space.

When domain_flag == 0, the required value for domain_scale may be retrieved by us call, DM_get_dmod_domain_scale(rtn_err, dmod_containing_surface). Note, the input dcurve may not be given in unit_space for this function.

Returns the newly constructed projected DS_pfunc curve or NULL for an error.

Errors: DM_NULL_INPUT_PTR
The dcurve and surface cannot be NULL on entry.

DM_MIXED_DCRV_DIM
The dcurve's image_dim must equal the surface's domain_dim.

DM_BAD_TOLERANCE_VALUE
The tolerance cannot be 0.0.

DM_BAD_DOMAIN_PT_RANGE
dcurve must be completely contained within the deformable model.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_dmod_curve

Function: Deformable Modeling, DML Create and Query

Action: Creates a deformable curve model and returns a pointer to the deformable model object.

Prototype:

```
DS_dmod* DM_make_dmod_curve (
    int& rtn_err,           // out: 0=success or
                            // negative err code
    DS_pfunc* pfunc,        // in: defines the shape
                            // model
    void* dmod_entity       // in: application ptr
        = NULL,             // stored with dmod
    int draw_state          // in: draw bits for
        = (1 << 1)|         // application graphics
        (1 << 2)|           // (draw loads,
        (1 << 3),           // constraints, and
                            // seams)
    int tag                 // in: unique id for
        = 2,                // this object
    double alpha            // in: resistance to
        = 1.0,              // stretch [1.0]
    double beta             // in: resistance to
        = 5.0,              // bending [5.0]
    double gamma            // in: resist bending
        = 0.0,              // change rate
    double delta            // in: resistance to
        = 0.0,              // moving from default
                            // shape
    double dt               // in: implicit
        = 1.0,              // integration time
                            // step [1.0]
    double mass             // in: physical param
        = 0.0,              // (causes rippling)[1.0]
    double damp             // in: physical param
        = 0.0,              // (stable integration)
                            // [5.0]
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”dshusk/dskernel/dmapi.hxx”
#include ”dshusk/dskernel/dsdmod.hxx”
#include ”dshusk/dskernel/dspfunc.hxx”
#include ”dshusk/dskernel/sdm_options.hxx”
```

Description: Modifies return_err.

Builds and returns a deformable model of the pfunc shape. The deformable model entity pointer is made available to applications that may want to store data with each deformable model. This pointer is never accessed (and so may be set to NULL) but can be retrieved by an application with the DS_dmod method call DS_dmod–>Entity() and it may be changed with the method DS_dmod–>Set_entity(entity). draw_state is a bit–array integer which may be used by the application to control which pieces of deformable modeling information are rendered for each deformable model. See the DM_DRAW_... environment variables defined in the file “dmapi.hxx” for a complete list. The draw_state defaults to drawing loads, constraints, and seams. tag is unique identifier for this deformable model. Since this deformable model will act as the root of its own patch hierarchy a good tag value is 2 which is its default value. alpha (default=1) is this deformable model’s resistance to stretch parameter value. beta (default=5) is this deformable model’s resistance to bending parameter value. delta (default=0) is this deformable model’s resistance to displacement from the default shape. DS_solve() (default =1) is the deformable modeling time step size taken with each call to DM_solve(). mass (default 1.0) and damp (default 5) are the dynamic parameters for this deformable model. With mass, a deformable model can overshoot and oscillate on its way to an equilibrium position. With damping the oscillations can diminish over time and the number of time steps until equilibrium is achieved may be increased.

mass (default 1.0) and damp (default 5) are the dynamic parameters for this deformable model. With mass a deformable model can overshoot and oscillate on its way to an equilibrium position. With damping the oscillations can diminish over time and the number of time steps until equilibrium is achieved may be increased.

mallocs is the calling program must free the memory of the returned deformable model pointer with a call to DM_delete_dmod. The pfunc pointer is stored within the deformable model and will be deleted when the deformable model is deleted. Do not call DM_delete_pfunc on that pointer after this call. Do not make a second deformable model with this pfunc pointer. Use DM_copy_pfunc if you want to make another deformable model with this shape.

Errors: DM_NULL_INPUT_PTR
The pfunc cannot be NULL on input.

DM_BAD_TAG_VALUE
The tag value must be –1, to have the system assign one, or 2 or greater.

DM_BAD_ALPHA_VALUE
The value of alpha must be 0 or positive.

DM_BAD_BETA_VALUE
The value of beta must be 0 or positive.

DM_ZERO_ALPHA_AND_BETA
Both alpha and beta values cannot be 0.

DM_BAD_DELTA_VALUE
The value of delta must be positive.

DM_BAD_DT_VALUE
The value of dt must be positive.

DM_BAD_MASS_VALUE
The value of mass must be 0 or positive.

DM_BAD_DAMP_VALUE
The value of damp must be 0 or positive.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_dmod_surface

Function: Deformable Modeling, DML Create and Query

Action: Creates a deformable surface model and returns a pointer to the deformable model object or NULL.

Prototype:

```
DS_dmod* DM_make_dmod_surface (
    int& rtn_err,              // out: 0=success or
                               // negative err code
    DS_pfunc* pfunc,           // in: defines the shape
                               // model
    void* dmod_entity          // in: application ptr
        = NULL,                // stored with dmod
    int draw_state             // in: draw bits for
        = (1 << 1)|            // application graphics
        (1 << 2)|              // (draw loads,
        (1 << 3),              // constraints, and
                               // seams)
    int tag                    // in: unique id for this
        = 2,                   // object
    double au                  // in: resist stretch u
        = 1.0,                 // direction
    double av                  // in: resist stretch v
        = 1.0,                 // direction
    double atheta              // in: resist stretch
        = 0.0,                 // rotation angle
    double bu                  // in: resist bending u
        = 5.0,                 // direction
    double bv                  // in: resist bending v
        = 5.0,                 // direction
    double btheta              // in: resist bending
        = 0.0,                 // rotation angle
    double gamma               // in: resist bending
        = 0.0,                 // change rate
    double delta               // in: resistance to
        = 0.0,                 // moving from default
                               // shape
    double dt                  // in: implicit
        = 1.0,                 // integration time
                               // step [1.0]
    double mass                // in: physical param
        = 0.0,                 // (causes rippling)[1.0]
    double damp                // in: physical param
        = 0.0,                 // (stable integration)
                               // [5.0]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies return_err.

Builds and returns a deformable model of the pfunc shape. The dmod_entity pointer is made available to applications that may want to store data with each deformable model. This pointer is never accessed (and so may be set to NULL) but can be retrieved by an application with the deformable model method call DS_dmod–>Entity() and it may be changed with the method DS_deformable_model–>Set_entity (entity). draw_state is a bit–array integer which may be used by the application to control which pieces of deformable modeling information are rendered for each deformable model. See the DM_DRAW_... environment variables defined in the file "dshusk/dskernel/dmapi.hxx" for a complete list. The draw_state defaults to drawing loads, constraints, and seams. tag is unique identifier for this deformable model. Since this deformable model will act as the root of its own patch hierarchy a good tag value is 2 which is its default value. *au* (default=1) is this deformable model's resistance to stretch parameter value in the *u* direction, *av* (default=1) is this deformable model's resistance to stretch parameter value in the *v* direction, and *atheta* is an amount in radians by which those stretch resistance may be rotated within the surface. *bu* (default=5) is this deformable model's resistance to bending parameter value in the *u* direction, *bv* (default=5) is this deformable model's resistance to bending parameter value in the *v* direction, and *btheta* is an amount in radians by which those bending resistances may be rotated within the surface. *delta* (default=0) is this deformable model's resistance to displacement from the default shape. dt (default =1) is the deformable modeling time step size taken with each call to DM_solve(). mass (default 0.0) and damp (default 0.0) are the dynamic parameters for this deformable model. With mass, a deformable model can overshoot and oscillate on its way to an equilibrium position. With damping the oscillations can diminish over time and the number of time steps until equilibrium is achieved may be increased. When both mass and damping are 0 (default behavior), a DM_solve() solves directly for the equilibrium position, ignoring the current value of dt.

mallocs is the calling program must free the memory of the returned deformable model pointer with a call to DM_delete_dmod. The pfunc pointer is stored within the deformable model and will be deleted when the deformable model is deleted. Do not call DM_delete_pfunc on that pointer after this call. Do not make a second deformable model with this pfunc pointer. Use DM_copy_pfunc if you want to make another deformable model with this shape.

Errors: DM_NULL_INPUT_PTR
The pfunc cannot be NULL on input.

DM_BAD_TAG_VALUE
The tag value must be –1, to have the system assign one, or 2 or greater.

DM_BAD_ALPHA_VALUE
The value of alpha must be 0 or positive.

DM_BAD_BETA_VALUE
The value of beta must be 0 or positive.

DM_BAD_DELTA_VALUE
The value of delta must be positive.

DM_BAD_DT_VALUE
The value of dt must be positive.

DM_BAD_MASS_VALUE
The value of mass must be 0 or positive.

DM_BAD_DAMP_VALUE
The value of damp must be 0 or positive.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_nurb_curve

Function: Deformable Modeling, DML Create and Query

Action: Creates a NURB curve from data and returns a pointer to DS_pfunc, NURB curve or NULL.

Prototype:

```
DS_pfunc* DM_make_nurb_curve (
    int& rtn_err,           // out: 0=success or
                            // negative err code
    int image_dim,          // in: image space size
                            // (2=2d,3=3d)
    int degree,             // in: polynomial degree
    int dof_count,          // in: control point
                            // count
    int knot_count,         // in: number of distinct
                            // knot values
    int* knot_index,        // in: multiple knot
                            // count array.
                            // [not-nested]
                            // sized:[knot_count]
                            // specify the multiple
                            // knots by setting
                            // count[i] = maximum
                            // knot index value for
                            // location knot[i]
                            // Examples:
                            // no multiples count
                            // = [0,1,2,3,4]
                            // some multiples
                            // = [1,2,4]
    double* knot,           // in: knot value array
                            // [not-nested]
                            // sized:[knot_count]
                            // ordered [u0 < u1 <
                            // u2 ... ]
    double* dof_vec,        // in: init dof_vec vals
                            // or NULL for 0
                            // [not-nested]
                            // sized:[image_dim*
                            // dof_count]
                            // ordered:[xyz0, xyz1,
                            // xyz2, ...]
    double* dof_def,        // in: init default shape
                            // or NULL for 0
                            // [not-nested]
                            // sized:[image_dim*
                            // dof_count]
                            // ordered:[xyz0, xyz1,
                            // xyz2, ...]
    double* weight,         // in: weight vals for
                            // each control pt
```

```
                                    // [not-nested]
                                    // sized:[dof_count]
                                    // ordered:[w0, w1,
                                    // ... wn]
    int end_cond                    // in: enforce continuity
        = 0,                        // between end points
                                    // with constraints. If
                                    // used must have
                                    // multiple knots on
                                    // the ends of the nurb.
                                    // 0=open (no
                                    // constraints)
                                    // 1=closed (C0
                                    // continuity)
                                    // 2=periodic (C1
                                    // continuity)
                                    // default value = 0.
    SDM_options* sdmo               // in:SDM_options pointer
        = NULL                      // note: total_knot_count
                                    // = dof_count+degree-1
                                    // the total and distinct
                                    // knot counts vary when
                                    // there are multiple
                                    // knots
                                    // note: the input array
                                    // values are only
                                    // copied. The calling
                                    // program still needs to
                                    // manage that memory
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Returns a pointer to the allocated and initialized memory used to represent a circular curve as a DS_pfunc description and sets the rtn_err as described below. Call DM_delete_pfunc() on the returned pointer to free its memory.

Creates and returns a pointer to the DS_pfunc NURB curve representation. image_dim is the size of the model space, use 2 for 2D and 3 for 3D. degree is the degree of NURB polynomials. dof_count is the number of NURB control points. knot_count is the number of unique values. The total_count equals dof_count + degree – 1. The total_knot_count will not equal the distinct knot_count when the NURB has any multiple s. The multiplicity of each is communicated through the knot_index array. Each starting with the lowest *u* value, is assigned a knot_index starting at 0. The knot_index array records the highest index value for all s at each distinct location. For example, a common NURB is one with multiple s on the end points and single s in the interior. A map of the index values might look like,

```
+------+------+-------+------+
0       3       4       5       6
1                                7
2                                8
```

This knot array corresponds to a degree = 3 B–spline with seven control points and four spans. The NURB interpolates the first and last control point positions due to the multiple end knots.

There are nine total knot values and only five distinct knot values. The knot index array for this curve would be [2,3,4,5,8]. The knot array contains the domain space values (u values) for each distinct knot. For the above example, the knot array might be something like [0.00, 0.25, 0.50, 0.75, 1.00] for an even distribution of knot values from 0 to 1.

Other common NURB representation schemes store an extra unused knot at each end of the knot array. These extra knot values seem to appear in all the NURB evaluation functions but in reality are never used and are not input to the deformable modeling package. Developers moving NURB data between this and other systems must take care to watch out for this mismatch possibility on the end knots.

The dof_vec contains the curve's control point locations and is sized [image_dim*dof_count_u*dof_count_v]. It is ordered by control point, so a two dimensional curve with four control points would have a dof_vec stored as [x0, y0, x1, y1, x2, y2, x3, y3], and a three dimensional curve would be stored as, [x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3].

Deformable curves have both a current shape and a default shape. The default shape is stored as a second array of control points called dof_def. dof_def is sized and stored in the same manner as dof_vec. end_cond is a flag which can be used to have the deformable modeling package force the two ends of a NURB curve to be continuous with one another. when end_cond equals 0 the curve remains open, when end_cond = 1 it will be C0 across the end points, and when end_cond = 2 it will be C1 across the end points. Every control point is associated with a weight value. These weight values remain constant during deformable model sculpting and are input through the weight array which is sized:[dof_count] and ordered [w0, w1, ... wn]. Where *wn* is the weight associated to the *nth* control point.

Errors:

DM_BAD_IMAGE_DIM_VALUE
The input value of the image dimension must be positive.

DM_BAD_DEGREE_VALUE
The requested spline degrees must be 0 or greater.

DM_BAD_KNOT_COUNT_VALUE
The number of knots must be positive.

DM_NULL_INPUT_PTR
The knot_index, knot, and weight array pointers cannot be NULL on input.

DM_BAD_KNOT_TO_CPT_COUNT
The last knot index number must equal dof_count + degree – 2.

DM_BAD_NTGRL_DEGREE_VALUE
The required gauss integration degree is not supported. Currently limited to the range 1 to 79. The ntgrl_degree must be equal to or larger than twice the spline degree for valid deformations.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_nurb_surface

Function: Deformable Modeling, DML Create and Query

Action: Constructs a NURB surface from data and returns pointer to DS_pfunc NURB surf or NULL.

Prototype:

```
DS_pfunc* DM_make_nurb_surface (
    int& rtn_err,           // out: 0=success or
                            // negative err code
    int image_dim,          // in: image space size
                            // (2=2d,3=3d)
    int degree_u,           // in: u dir
                            // polynomial degree
    int dof_count_u,        // in: u dir control
                            // point count
    int knot_count_u,       // in: u dir distinct
                            // knot value count
    int* knot_index_u,      // in: u dir multiple
                            // knot count array.
                            // sized:[knot_count]
                            // specify the multiple
                            // knots by setting
                            // count[i] = maximum
                            // knot index value for
                            // location knot[i]
                            // Examples:
                            // no multiples count =
                            // [0,1,2,3,4]
                            // some multiples =
                            // [1,2,4]
    double* knot_u,         // in: u dir knot value
                            // array
                            // sized:[knot_count]
                            // ordered [u0<u1<u2...]
    int degree_v,           // in: v dir polynomial
                            // degree
    int dof_count_v,        // in: v dir control
                            // point count
    int knot_count_v,       // in: v dir distinct
                            // knot value count
    int* knot_index_v,      // in: v dir multiple
                            // knot count array.
                            // sized:[knot_count]
                            // specify the multiple
                            // knots by setting
                            // count[i] = maximum
```

```
                                // knot index value for
                                // location knot[i]
                                // Examples:
                                // no multiples count =
                                // [0,1,2,3,4]
                                // some multiples =
                                // [1,2,4]
double* knot_v,                 // in: v dir knot value
                                // array
                                // sized:[knot_count]
                                // ordered [u0<u1<u2...]
double* dof_vec,                // in: init dof_vec vals
                                // or NULL for 0
                                // sized:[image_dim*
                                // dof_count_u*
                                // dof_count_v]
                                // ordered:[C00, C01, ...
                                // C0m, C10, ... Cnm]
double* dof_def,                // in: init default shape
                                // or NULL for 0
                                // ordered:[C00, C01, ...
                                // C0m, C10, ... Cnm]
double* weight,                 // in: init weight for
                                // each control pt
                                // sized:[dof_count_u*
                                // dof_count_v]
                                // ordered:[w00, w01, ...
                                // w0m, ... wnm]
int end_cond_u                  // in: one of 0=open or
    = 0,                        // 1=closed or 2=periodic
int singular_u                  // in: one of 0=none or
    = 0,                        // 1=low or 2=high or
                                // 3=both
int end_cond_v                  // in: one of 0=open or
    = 0,                        // 1=closed or 2=periodic
int singular_v                  // in: one of 0=none or
    = 0,                        // 1=low or 2=high or
                                // 3=both
SDM_options* sdmo               // in:SDM_options pointer
    = NULL                      // note: total_knot_count
                                // = dof_count+degree–1
                                // the total and distinct
                                // knot counts vary when
                                // there are multiple
                                // knots
```

```
                                        // note: the input array
                                        // values are only
                                        // copied. The calling
                                        // program still needs to
                                        // manage that memory
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies return_err.

Creates and returns a pointer to DS_pfunc NURB surface representation. A NURB surface is the tensor product of the underlying u–curve and v–curve basis functions. The total number of dofs in the surface is just the product of the dof counts for *u* and *v* curve basis functions. The global dof numbers for each dof are the cross product of the u–curve and v–curve dof numbers with the v–curve dof numbers varying the fastest. This function's input arguments contain enough information to make both the u–curve and the v–curve basis functions as well as the dof values for the surface.

image_dim is the size of the model space, using 2 for 2D and 3 for 3D. degree_u and degree_v are the NURB polynomial degrees in the surface's *u* and *v* directions. dof_count_u and dof_count_v are the number of NURB control points in the surface's *u* and *v* directions. knot_count_u and knot_count_v are the number of unique knot values in the surface's *u* and *v* directions.

The total_knot_count equals the dof_count + degree – 1 for each *u* and *v* surface direction. The total_knot_count will not equal the distinct knot_count when the NURB has any multiple knots. The multiplicity of each knot is communicated through the knot_index array. Each knot, starting with the lowest *u* value knot, is assigned a knot_index starting at 0. The knot_index array records the highest index value for all knots at each distinct knot location. For example, a common NURB is one with multiple knots on the end points and single knots in the interior. A map of the knot index values might look like:

```
+------+------+-------+------+
0       3       4       5       6
1                               7
2                               8
```

This knot array corresponds to a degree = 3 B–spline with seven control points and four spans. The B–spline interpolates the first and last control point positions due to the multiple end knots.

There are nine total knot values and only five distinct knot values. The knot index array for this curve would be [2,3,4,5,8]. The knot array contains the domain space values (u values) for each distinct knot. For the above example, the knot array might be something like [0.00, 0.25, 0.50, 0.75, 1.00] for an even distribution of knot values from 0 to 1.

Other common B–spline representation schemes store an extra unused knot at each end of the knot array. These extra knot values seem to appear in all the B–spline evaluation functions but in reality are never used and are not input to the deformable modeling package. Developers moving B–spline data between this and other systems must take care to watch out for this mismatch possibility on the end knots.

The dof_vec contains the surface's control point locations and is sized [image_dim*dof_count_u*dof_count_v]. It is ordered by control point, so a two dimensional surface with a 2x2 grid of control points will be organized as:

```
dof01       dof11       Global dof numbers
    +-----------+           dof0 = dof00
    |           |           dof1 = dof01
    |           |           dof2 = dof10
    |           |           dof3 = dof11
    +-----------+
    dof00 dof10
```

and will have a dof_vec stored as [x0, y0, x1, y1, x2, y2, x3, y3], and the same surface in 3d would be stored as, [x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3]. Deformable surfaces have both a current shape and a default shape. The default shape is stored as a second array of control points called dof_def. dof_def is sized and stored in the same manner as dof_vec.

Surfaces can have three different end conditions applied to them through the use of constraints. These include an end_cond which can force opposite edges of the surface square to connect to each other with either a C0 or a C1 continuity, and a singularity case which can collapse an entire boundary edge into a single point. These end conditions are communicated through the use of four flags, end_cond_u, end_cond_v, singular_u, and singular_v. The connectivity of the *u* and *v* boundary curves are set independently by the end_cond_u and end_cond_v flag values. When end condition equals 0 the surface remains open in that direction. When end condition equals 1 it will be C0 across the end curves. And when end condition equals 2 it will be C1 across the end curves. The singularity of the *u* and *v* boundaries are set independently by the singular_u and singular_v flag values. When singular equals 0 all edges are normal. When singular equals 1 the low end boundary curve is collapsed to a point. When singular equals 2 the high end boundary curve is collapsed to a point. When singular equals 3 both low and high end boundary curves are collapsed to a single point.

Errors: DM_BAD_IMAGE_DIM_VALUE
The input value of the image dimension must be positive.

DM_BAD_DEGREE_VALUE
The requested spline degrees must be 0 or greater.

DM_BAD_KNOT_COUNT_VALUE
The number of knots must be positive.

DM_NULL_INPUT_PTR
The knot_index, knot, and weight array pointers cannot be NULL on input.

DM_BAD_KNOT_TO_CPT_COUNT
The last knot index number must equal dof_count + degree – 2.

DM_BAD_NTGRL_DEGREE_VALUE
The required gauss integration degree is not supported. Currently limited to the range 1 to 79. The ntgrl_degree must be equal to or larger than twice the spline degree for valid deformations.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_make_orig_dmod_space_pfunc

Function: Deformable Modeling, DML Create and Query

Action: Creates a pfunc copy with the pfunc's domain set to equal the orig_dmod_space domain range.

Prototype:
```
DS_pfunc* DM_make_orig_dmod_space_pfunc (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_dmod* dmod,          // in: dmod to query
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Copies the DS_pfunc ptr contained within the input dmod object and scales its domain so that the returned pfunc's domain is equal to the dmod'sorig_dmod_space.

This function mallocs the returned DS_pfunc object, and it is the responsibility of the calling program to free this memory when done with it.

```
internal_pfunc_space = domain_scale *
orig_dmod_space.
```

The domain_scale factor may be queried by the function, DM_get_dmod_domain_scale().

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_param_max

Function: Deformable Modeling

Action: Gets the tag object maximum parameterization value.

Prototype:
```
void DM_param_max (
    int& rtn_err,           // error code
    DS_dmod* dmod,          // model to check
    int tag,                // tag object ID
    DM_dbl_array& s_arr,    // tag object ID
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dm_dbl_array.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Many tag objects are parameterized by either a continuous (e.g., curve load) or discrete parameter (e.g., point spring set). This function returns the upper corner of the parameterization bounding box. For example, for a curve load, this is a single value, and for an area constraint this a pair (*uv*) of values. rtn_err returns 0 for success or a negative error code.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
Could not find tag object with the given tag.

DM_TAGOBJ_HAS_NO_PARAM
Tag object is not a parameterized type.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Read–only

DM_param_min

Function: Deformable Modeling

Action: Gets the tag object minimum parameterization value.

Prototype:
```
void DM_param_min (
    int& rtn_err,           // error code
    DS_dmod* dmod,          // model to check
    int tag,                // tag object ID
    DM_dbl_array& s_arr,    // tag object ID
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dm_dbl_array.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Many tag objects are parameterized by either a continuous (e.g., curve load) or discrete parameter (e.g., point spring set). This function returns the lower corner of the parameterization bounding box. For example, for a curve load, this is a single value, and for an area constraint this a pair (*uv*) of values. rtn_err returns 0 for success or a negative error code.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
Could not find tag object with the given tag.

DM_TAGOBJ_HAS_NO_PARAM
Tag object is not a parameterized type.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Read–only

DM_parse_tag_flag

Function: Deformable Modeling, DML Tags

Action: Converts a tag_flag identifier into a deformable model pointer and a walk_flag value.

Prototype:

```
DS_dmod* DM_parse_tag_flag (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: member of
                            // dmod hierarchy to
                            // search
    int tag_flag,           // in: specify target
                            // dmod and how deep to
                            // go, 1=active, 2=root,
                            // else=member(tag=
                            // tag_flag)
                            // pos=tgt only, neg=tgt
                            // + offspring
    int& walk_flag,         // out: 0=tgt_only, 1=tgt
                            // and offspring, 2=tgt,
                            // siblings, and
                            // offspring.
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: makes found
                            // tgt_dmod active
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies walk_flag, returns pointer to identified deformable model.

tag_flag identifies the target deformable model as follows. When tag_flag is equal to +1 or –1, the target is the active deformable model. When tag_flag is equal to +2 or –2, the target is the root deformable model. When tag_flag is equal to some other value, then the target is the deformable model that has the same tag identifier as the absolute value of the tag_flag value. When there is no deformable model whose tag identifier matches the absolute value of the tag_flag value, then there is no target and NULL is returned.

In summary:

+1 or –1	= active deformable model
+2 or –2	= root deformable model
+ or –num	= deformable model with tag_id == |num|

The sign of the tag_flag value specifies the value of the walk_flag. walk_flag is a typical input required by all functions which are capable of walking the deformable modeling hierarchy recursively.

Valid walk_flag values include:

0 = operate only on the target deformable model.
1 = operate only on the target deformable model and its offspring.
2 = operate on the target deformable model, its younger siblings, and all their offspring.

For multisurface models: When input tag_flag value is –2 (root dmod and offspring) or when tag_flag is –1 (the active deformable model and offspring) and the active deformable model is one of the root siblings, then the walk_flag value is set to 2 (target, siblings, and offspring), and the returned deformable model pointer is set to the oldest root sibling.

Errors: Returns NULL for any errors, and sets rtn_err to one of the following possible error values:

DM_DMOD_NOT_A_ROOT_DMOD
The deformable model must be the root of a hierarchy tree.

DM_BAD_TAG_FLAG_VALUE
The tag_flag cannot equal 0.

DM_NO_ACTIVE_DMOD
The input deformable model's hierarchy has no currently active deformable model.

DM_TAG_NOT_A_DMOD_MEMBER
The tag_flag does not identify a deformable model in the patch hierarchy.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Read–only

DM_print_dmod

Function: Deformable Modeling

Action: Generates a deformable model report and returns 0 for success or an error.

Prototype:

```
void DM_print_dmod (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: target dmod
                            // to query
    FILE* file,             // in: stream to
                            // be written
    int walk_flag           // in: specify how deep
        = 0,                // to go 0=dmod only,
                            // 1=dmod & offspring
                            // 2=dmod, siblings,
                            // and offspring
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies file.

Using fprintf statements, writes a report about the deformable model and its tag objects. When walk_flag is equal to 0 then the report covers only the input deformable model object. When walk_flag is equal to 1 then the report is written for deformable model and all deformable model's offspring. When walk_flag is 2 then the report is written for the deformable model, its siblings, and all their offspring.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
when walk_flag is not 0, 1, or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_print_dmod_cstrns

Function: Deformable Modeling, DML Constraints

Action: Generates a deformable model reports and returns 0 for success or an error.

Prototype:
```
void DM_print_dmod_cstrns (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to query
    FILE* file,             // in: stream to be
                            // written
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies the file. Using fprintf statements this function writes a report about all the objects operating on the input deformable model.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_query_icon

Function: Deformable Modeling

Action: Passes the command encapsulated by the query_args to the icon. Used to query the icon state.

Prototype:
```
void DM_query_icon (
    int& rtn_err,           // error code
    DM_icon_query_args& args,   // args passed
    DS_dmod* dmod,          // model to query
    int tag,                // tag object owning icon
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dm_icon_args.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function calls the icon's Query method. The data encapsulated by the query_args is passed through this method. rtn_err returns 0 for success or a negative error code.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
Could not find tag object with the given tag

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_refine_dmod_fit

Function: Deformable Modeling

Action: Modifies the shape of the deformable model to fit to its constraints. This function should be used after sculpting.

Prototype:

```
void DM_refine_dmod_fit (
    int& rtn_err,                // out: 0=success
                                 // or negative err
                                 // code
    DS_dmod* dmod,               // in: member of
                                 // hierarchy to
                                 // modify
    double err_size,             // in: max ok crv_err
                                 // without
                                 // postprocessing
    double& residual,            // out: abs max elem
                                 // value in r = Cx–d
    double& beg_crv_dist_err,    // out: max
                                 // edge/crv_cstrn
                                 // dist before refine
    double& end_crv_dist_err,    // out: max
                                 // edge/crv_cstrn
                                 // dist after refine
    double& max_dist_moved,      // out: max distance
                                 // one dof moved
    int walk_flag                // in: specify how
        = 0,                     // deep to go
                                 // 0=dmod only,
                                 // 1=dmod and
                                 // offspring
                                 // 3=dmod,siblings
                                 // and offspring
    SDM_options* sdmo            // in:SDM_options
        = NULL                   // pointer
                                 // eff: Moves the
                                 // dofs by a minimal
                                 // amount to minimize
                                 // the residual of
                                 // the constraint
                                 // equations,
                                 // r = Cx – d
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”dshusk/dskernel/dmapi.hxx”
#include ”dshusk/dskernel/dsdmod.hxx”
#include ”dshusk/dskernel/sdm_options.hxx”
```

Description: Modifies the target deformable model’s control point locations (dofs).

This function is used to minimize the numerical tolerances seen when enforcing the deformable modeling constraints on a deformable model.

It is intended to be used as a single post–processing step after sculpting is completed but before the deformed shape is used in a geometric modeling application.

This function tweaks the deformable model's shape slightly to improve the tolerances held by the curve constraints. Typically this function moves dof values slightly (1.E–05) to improve the constraint tolerances from about 1.E–05 to 3.E–07. This function is expensive to run and should only be run to restore the curve constraint tolerances after a sculpting session. This function should only be run once per sculpting session.

Before refining, this function stores in beg_crv_dist_err the maximum distance between an edge and the surface's curve points as found by DS_crv_–>Compare_src_to_out_W_pts(). When beg_crv_dist_err is less than err_size processing aborts and a value of 0 is returned. Otherwise, the refinement processing is executed by calling the DS_symeq–>Refine_Cx_equal_d() function to improve compliance with the constraints and computes and stores in end_crv_dist_err the refined distance error. The output residual is set with the maximum residual error in the constraint equations, res = Cx – d .

Increased compliance to the curve constraint equations is achieved by moving the dofs the minimum amount of distance to put the dof state back into the range of the constraint matrix. In other words, this function moves the control points to improve compliance with the constraints.

When the input dmod deformable model is one root sibling of a multi–surface sibling list, its link constraints are treated like curve constraints and the refinement is run only on the input dmod. To refine an entire sibling list, call this function once on each root sibling.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_rm_patch

Function: Deformable Modeling, DML Patches

Action: Removes and deletes a patch from a target deformable modeling hierarchy.

Prototype:
```
void DM_rm_patch (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: member of target
                            // dmod hierarchy
    int tag_flag,           // in: specify the
                            // tgt_dmod to remove
                            // 1 = active dmod
                            // 2 = member dmod with
                            // tag=tag_flag
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
                            // note: also always
                            // removes offspring
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Removes the deformable model identified by the tag_flag and all its children from the deformable modeling patch hierarchy. The deformable model identified by tag_flag must be a patch in the deformable modeling hierarchy. It can not be a root level deformable model.

After executing, this function sets the deformable modeling hierarchy's active patch to be the parent of the removed patch.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_rm_tag_object

Function: Deformable Modeling, DML Tags

Action: Removes a deformable modeling tag object from a deformable modeling hierarchy.

Prototype:

```
DS_TAGS DM_rm_tag_object (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: member of target
                               // dmod hierarchy to
                               // search
    int tag,                   // in: unique tag
                               // identifier to match
    DS_dmod*&                  // out: ptr to detached
        detached_dmod,         // sibling or NULL
    int deletable_flag         // 1=only rm DM_DELETABLE
        = 1,                   // cstrns,
                               // 0=remove any cstrn.
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dmapinum.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description:

Finds, removes, and deletes the tag object, identified by the input tag value, from the input dmod's deformable modeling hierarchy. When a tag object is found, the function removes it from the deformable modeling hierarchy, deletes it, sets rtn_err value to 0, and returns the type of the tag object that was removed. Otherwise, it sets rtn_err to a negative error code and returns a value of DS_tag_none.

This function removes constraints, loads, and descendant patches from a deformable modeling hierarchy. It can not remove a root sibling deformable model. An entire list of root siblings may be deleted by a call to DM_delete_dmod().

A single root_sibling can be isolated and deleted by calling this function on every link_constraint that connects the root_sibling in question to its deformable modeling neighbors and then calling the DM_delete_dmod() function on the isolated root_sibling.

When deleting link constraints, it is possible to eliminate the last link between two sets of siblings. When this happens, all the internal sibling pointers are modified to separate the decoupled sets of siblings into two distinct deformable modeling hierarchies. The output argument detached_sibling is set to point to the root of the deformable modeling hierarchy which does not contain the input deformable model. The calling application must take care to manage the detached_sibling list since those objects are no longer a part of this hierarchy.

When deleting the specified tag object does not detach a sibling dmod, the output argument detached_sibling is set to NULL.

When deletable_flag == 1, only those constraints which are deletable are deleted as determined by the call DM_get_cstrn_rights. When deletable_flag == 0, any constraint will be deleted.

When any error is encountered, the rtn_err value is set to a negative error code and a value of DS_tag_none is returned.

Errors:

DM_NO_ROOT_DMOD
The root deformable model cannot be NULL on input.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a deformable model, a load, a spring, a spring set, an attractor object, or a constraint in the model hierarchy.

DM_CANT_RM_ROOT_DMOD
This function cannot be used to attempt to remove the root deformable model of a deformable model hierarchy.

DM_UNDELETABLE_CSTRN
The identified constraint cannot be set to undeletable.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_pfunc_image

Function: Deformable Modeling, Deformable Model Management

Action: Scales the image of the input pfunc.

Prototype:
```
void DM_scale_pfunc_image (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_pfunc* pfunc,        // in: pfunc being
                            // modified.
    double scale,           // in: amount to scale
                            // pfunc->image
                            // (for error checking)
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies pfunc. Scales the image of the input pfunc argument. For B–splines and NURBs, this is equivalent to scaling the image space location of every control point.

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when pfunc is NULL on entry.
DM_BAD_SCALE_PARAM_VALUE when input scale parameter is 0.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_array_from_dmod

Function: Deformable Modeling, Deformable Model Management

Action: Scales an array of dpts from a deformable model's parameter range to the unit_square parameter range.

Prototype:

```
void DM_scale_unit_dpt_array_from_dmod (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_dmod* dmod,          // in: dmod being
                            // queried.
    int domain_dim,         // in: dmod domain_dim
                            // (for error checking)
    int pt_count,           // in: number of uv
                            // points in dpt
    double* dpt,            // i/o: change uv coord
                            // values from dmod space
                            // to unit square.
                            // 1d=[u0,u1,...,un]
                            // 2d=[uv0,uv1,...,uvn]
                            // in unit square, sized
                            // [Domain_dim*pt_count]
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies dpt. Scales the dpt coordinate values to map an array of domain points from the dmod's actual domain space into the unit–square space. When routine returns error dpt is left in an unknown state. For example, when domain_dim = 2 and pt_count = 2, an input dpt array of [7.0,7.0,4.0,6.0] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output dpt array value of [.5,.5,.2,.4].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when dmod is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to dmod–>Domain_dim.
DM_BAD_DOMAIN_DIM when domain_dim not equal 1 or 2.
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in dmod's range.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_array_from_pfunc

Function: Deformable Modeling, Deformable Model Management

Action: Scales an array of dpts from a pfunc's param range to the unit_square parameter range.

Prototype:

```
void DM_scale_unit_dpt_array_from_pfunc (
    int& rtn_err,              // out: 0=success
                               // or neg err code
    DS_pfunc* pfunc,           // in: pfunc being
                               // queried
    int domain_dim,            // in: pfunc domain_dim
                               // (for error checking)
    int pt_count,              // in: number of uv
                               // points in dpt
    double* dpt,               // i/o: change uv coord
                               // values from pfunc
                               // space to unit square.
                               // 1d=[u0,u1,...,un]
                               // 2d=[uv0,uv1,...,uvn]
                               // in unit square, sized
                               // [Domain_dim*pt_count]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies dpt. Scales the dpt coordinate values to map an array of domain points from the pfunc's actual domain space into the unit–square space. When routine returns error dpt is left in an unknown state. For example, when domain_dim = 2 and pt_count = 2, an input dpt array of [7.0,7.0,4.0,6.0] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output dpt array value of [.5,.5,.2,.4].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when pfunc is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to pfunc–>Domain_dim.
DM_BAD_DOMAIN_DIM when domain_dim not equal 1 or 2.
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in pfunc's range.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_array_to_dmod

Function: Deformable Modeling, Deformable Model Management

Action: Scales an array of domain point values that lie from 0 to 1 into the dmod's actual domain_space range.

Prototype:

```
void DM_scale_unit_dpt_array_to_dmod (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_dmod* dmod,          // in: dmod to query
    int domain_dim,         // in: dmod domain_dim
                            // (for error checking)
    int pt_count,           // in: number of uv
                            // points in dpt
    double* dpt,            // i/o: change uv coord
                            // values from unit
                            // square to dmod
                            // domain space.
                            // 1d=[u0,u1,...,un]
                            // 2d=[uv0,uv1,...,uvn]
                            // in unit square,
                            // sized:
                            // [Domain_dim*pt_count]
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *uv*.

Scales the dpt coordinate values to map the unit square into the actual parameter range of the curve or surface. For example, with main_dim = 2 and pt_count = 2, an input dpt array of [.5,.5,.2,.4] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output *uv* value of [7.0,7.0,4.0,6.0].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors:

rtn_err set to
0 for success
DM_NULL_INPUT_PTR when dmod is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to
dmod–>Domain_dim()
DM_BAD_DOMAIN_DIM when domain_dim not equal to 1 or 2
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in 0 to 1 range

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_array_to_pfunc

Function: Deformable Modeling, Deformable Model Management

Action: Scales an array of domain point values that lie from 0 to 1 into the pfunc's actual domain_space range.

Prototype:

```
void DM_scale_unit_dpt_array_to_pfunc (
    int& rtn_err,              // out: 0=success
                               // or neg err code
    DS_pfunc* pfunc,           // in: pfunc to query
    int domain_dim,            // in: pfunc domain_dim
                               // (for error checking)
    int pt_count,              // in: number of uv
                               // points in dpt
    double* dpt,               // i/o: change uv coord
                               // values from unit
                               // square
                               // to pfunc domain space.
                               // 1d=[u0,u1,...,un]
                               // 2d=[uv0,uv1,...,uvn]
                               // in unit square, sized
                               // [Domain_dim*pt_count]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *uv*.

Scales the dpt coordinate values to map the unit square into the actual parameter range of the curve or surface. For example, with domain_dim = 2 and pt_count = 2, an input dpt array of [.5,.5,.2,.4] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output *uv* value of [7.0,7.0,4.0,6.0].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when pfunc is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to
pfunc–>Domain_dim()
DM_BAD_DOMAIN_DIM when domain_dim not equal to 1 or 2
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in 0 to 1 range

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_from_dmod

Function: Deformable Modeling, Deformable Model Management

Action: Scales domain point values specified in the deformable model's domain space into the unit_square domain space.

Prototype:
```
void DM_scale_unit_dpt_from_dmod (
    int& rtn_err,            // out: 0=success
                             // or neg err code
    DS_dmod* dmod,           // in: dmod to query
    int domain_dim,          // in: dmod domain_dim
                             // (for error checking)
    double* uv,              // i/o: change uv coord
                             // value to fit in unit
                             // square
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *uv*.

Scales *uv* coordinate values to map surface parameter points from the dmod's actual domain range into the unit square range. For example, an input *uv* value of [7.0,7.0] on a surface whose knot vector runs from 2.0 to 12.0 for both the u and v directions will have an output uv value of [.5,.5].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when dmod is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to dmod–>Domain_dim()
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in dmod's domain range or domain_dim ! = 1 or 2

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_from_pfunc

Function: Deformable Modeling, Deformable Model Management

Action: Scales domain point values specified in the pfunc's domain_space into the unit_square domain_space.

Prototype:

```
void DM_scale_unit_dpt_from_pfunc (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_pfunc* pfunc,        // in: pfunc to query
    int domain_dim,         // in: pfunc domain_dim
                            // (for error checking)
    double* uv,             // i/o: change uv coord
                            // value to fit in unit
                            // square
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *uv*.

Scales *uv* coordinate values to map surface parameter points from the pfunc's actual domain range into the unit square range. For example, an input *uv* value of [7.0,7.0] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output *uv* value of [.5,.5].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when pfunc is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to
pfunc–>Domain_dim()
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in pfunc's domain range or domain_dim != 1 or 2

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_to_dmod

Function: Deformable Modeling, Deformable Model Management

Action: Scales domain points from the unit domain range into the dmod's actual domain range.

Prototype:

```
void DM_scale_unit_dpt_to_dmod (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_dmod* dmod,          // in: dmod to query
    int domain_dim,         // in: dmod domain_dim
                            // (for error checking)
    double* uv,             // i/o: scale uv coords
                            // to dmod domain range.
                            // sized:[domain_dim]
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *uv*.

Scales *uv* coordinate values to map the unit square into the actual parameter range of the input dmod. For example, an input *uv* value of [.5,.5] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output *uv* value of [7.0,7.0].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_dmod *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when dmod is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to
dmod–>Domain_dim() or domain_dim != 1 or 2
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in 0 to 1 range

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dpt_to_pfunc

Function: Deformable Modeling, Deformable Model Management

Action: Scales domain points from the unit domain range into the pfunc's actual domain range.

Prototype:
```
void DM_scale_unit_dpt_to_pfunc (
    int& rtn_err,           // out: 0=success
                            // or neg err code
    DS_pfunc* pfunc,        // in: pfunc to query
    int domain_dim,         // in: pfunc domain_dim
                            // (for error checking)
    double* uv,             // i/o: scale uv coords
                            // to pfunc domain range.
                            // sized:[domain_dim]
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *uv*.

Scales *uv* coordinate values to map the unit square into the actual parameter range of the input pfunc. For example, an input *uv* value of [.5,.5] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and v directions will have an output uv value of [7.0,7.0].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when pfunc is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to pfunc–>Domain_dim() or domain_dim != 1 or 2
DM_BAD_DOMAIN_PT_RANGE = input par_pos not in 0 to 1 range

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dvec_to_dmod

Function: Deformable Modeling, Deformable Model Management

Action: Scales domain vector values that are specified in a unit domain space that ranges from 0 to 1 into the deformable model's domain range.

Prototype:
```
void DM_scale_unit_dvec_to_dmod (
    int& rtn_err,              // out: 0=success
                               // or neg err code
    DS_dmod* dmod,             // in: dmod to query
    int domain_dim,            // in: dmod domain_dim
                               // (for error checking)
    double* duv,               // i/o: scale duv vector
                               // from unit square to
                               // dmod domain range
                               // sized: [domain_dim]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *duv.*

Scales *duv* coordinate values to map vectors expressed in the unit square into vectors in the actual parameter range of the curve or surface. For example, an input vector value of [.2,–.5] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output *uv* value of [2.0,–5.0].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when dmod is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to
dmod–>Domain_dim() or domain_dim != 1 or 2

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_scale_unit_dvec_to_pfunc

Function: Deformable Modeling, Deformable Model Management

Action: Scales domain vector values that are specified in a unit domain space that ranges from 0 to 1 into the pfunc's domain range.

Prototype:

```
void DM_scale_unit_dvec_to_pfunc (
    int& rtn_err,            // out: 0=success
                             // or neg err code
    DS_pfunc* pfunc,         // in: pfunc to query
    int domain_dim,          // in: pfunc domain_dim
                             // (for error checking)
    double* duv,             // i/o: scale duv vector
                             // from unit square to
                             // pfunc domain range
                             // sized:[domain_dim]
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies *duv*.

Scales *duv* coordinate values to map vectors expressed in the unit square into vectors in the actual parameter range of the curve or surface. For example, an input vector value of [.2,–.5] on a surface whose knot vector runs from 2.0 to 12.0 for both the *u* and *v* directions will have an output *uv* value of [2.0,–5.0].

Note *The* DS_dmod*'s domain range may be different from its contained* DS_pfunc*'s domain range. The* DS_dmod *domain range remains constant during a deformable modeling session while the* DS_pfunc *domain range may change. Initially they start out the same.*

Errors: rtn_err set to
0 for success
DM_NULL_INPUT_PTR when pfunc is NULL on entry.
DM_BAD_DOMAIN_DIM = input domain_dim != to
pfunc–>Domain_dim() or domain_dim != 1 or 2

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_setstate_icon

Function: Deformable Modeling, DML Graphics

Action: Broadcasts the command encapsulated by the cmd_args to the icons.

Prototype:

```
void DM_setstate_icon (
    int& rtn_err,            // error code
    const DM_icon_cmd_args& args,// args passed
    DS_dmod* dmod,           // model to check
    const int* tags,         // tag object owning icon
    int ntags,               // number of tag objects
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               //
    );

void DM_setstate_icon (
    int& rtn_err,            // error code
    const DM_icon_cmd_args& args,// args passed
    DS_dmod* dmod,           // model to query
    int tag,                 // tag object owning icon
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dm_icon_args.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function calls the icon's Setstate method. The data encapsulate by the cmd_args is passed through this method.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
Could not find tag object with the given tag.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_set_active_patch

Function: Deformable Modeling, DML Patches

Action: Sets the input deformable model to the hierarchy's active deformable model.

Prototype:
```
void DM_set_active_patch (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to make
                            // active
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the deformable model's root deformable model dmo_active pointer to be this deformable model's value. Making a deformable model active allows many of the input and output commands to work on this deformable model by default.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_set_alpha

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's alpha and returns 0 for success or an error.

Prototype:

```
void DM_set_alpha (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: dmod to modify
    double* alpha,             // in: desired value to
                               // store curves:
                               // [alpha=1.0], sized:[1]
                               // surfs: [au=1.0,
                               // av=1.0, atheta=0.0],
                               // sized:[3]
    int walk_flag              // in: specify how deep
        = 0,                   // to go
                               // 0=dmod only,
                               // 1=dmod and offspring
                               // 2=dmod, siblings,
                               // and offspring
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the deformable model's resistance to stretch alpha value. Curves have just one alpha value while surfaces have three, a resistance to stretch in the u_dir and the v_dir, and a rotation angle. For surfaces, alpha is sized:[3] and stored as [*au, av, atheta*]. See DM_get_alpha for a further description of alpha and a note about its usage. A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified. A walk_flag value of 2 specifies that the deformable model, its siblings, and all their offspring be modified.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

DM_BAD_ALPHA_VALUE
The value of alpha must be 0 or positive.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_area_cstrn_flag

Function: Deformable Modeling, DML Constraints

Action: Modifies the fixed inside/outside flag for an area constraint.

Prototype:
```
void DM_set_area_cstrn_flag (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: root of
                            // hierarchy tree
    int tag,                // in: unique tag
                            // identifier to match
    int zone_flag,          // in: 0=zone area moves
                            // zone compliment fixed
                            // 1=zone area fixed
                            // zone compliment moves
                            // –1=toggle current
                            // zone_flag value
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Searches the deformable model hierarchy for the area constraint with the given input tag value and modifies that area constraint's fixed inside/outside flag value. Also sets the active deformable model.

– When zone_flag == 0, the zone area moves and the zone compliment area is fixed.
– When zone_flag == 1, the zone area is fixed and the zone compliment area is free to move.
– When zone_flag == –1, the current zone_flag value is toggled.

Passes the change request down to the deformable model so that all affected arrays can be updated.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag does not identify an area constraint in the model hierarchy.

DM_BAD_ZONE_FLAG_VALUE
The zone_flag must be equal –1, 0, or 1.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_array_size

Function: Deformable Modeling

Action: Sets the size of a DM_dbl_array and initializes the memory.

Prototype:
```
void DM_set_array_size (
    int& rtn_err,           // error code
    DM_dbl_array& arr,      // array to resize
    int new_size,           // size to set
    double init_val         // value stored in
        = 0.0,              // returned memory
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dm_dbl_array.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: The DM_dbl_array is a container class provided by deformable modeling. The DM_dbl_array can be used as growable array via the DM_set_array_size method. There are no user heap allocations or deletes associated with the DM_dbl_array.

Errors: None

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_set_attractor_power

Function: Deformable Modeling

Action: Sets an attractor's power and returns 0 for success or an error.

Prototype:

```
void DM_set_attractor_power (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: member of target
                            // dmod hierarchy to
                            // search
    int tag,                // in: unique load tag
                            // identifier to match
    int power               // in: measure of load's
        = 2,                // locality
                            // 0,1=global:
                            // 2,3.. =more local
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the tag load's power value when the input tag value identifies a tag object of type DS_attractor. Otherwise, returns DM_TAG_OBJECT_NOT_FOUND.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a deformable model, a load, a spring, a spring set, an attractor object, or a constraint in the model hierarchy.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_beta

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's beta and returns 0 for success or an error.

Prototype:
```
void DM_set_beta (
    int& rtn_err,            // out: 0=success
                             // or negative err code
    DS_dmod* dmod,           // in: dmod to modify
    double* beta,            // out: resistance to
                             // bending values
                             // curves: [beta=5.0],
                             // sized:[1]
                             // surfs: [bu=5.0,
                             // bv=5.0, btheta=0.0],
                             // sized:[3]
    int walk_flag            // in: specify how deep
        = 0,                 // to go
                             // 0=dmod only,
                             // 1=dmod and offspring
                             // 2=dmod, siblings
                             // and offspring
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the deformable model's resistance to bending beta values. Curves have just one beta value while surfaces have three, a resistance to bending in the u_dir and the v_dir, and a rotation angle. For surfaces, beta is sized:[3] and stored as [*bu, bv, btheta*]. See DM_get_beta for a further description of beta and a note about its usage. A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified. A walk_flag value of 2 specifies that the deformable model, its siblings, and their offspring are modified.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_comb_graphics

Function: Deformable Modeling, DML Graphics

Action: Sets the deformable model's comb_graphics and returns 0 for success or an error.

Prototype:

```
int DM_set_comb_graphics (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to modify
    int comb_pt_count,      // in: desired number of
                            // tines-per-elem
    double comb_gain,       // in: desired gain on
                            // comb vector magnitudes
    int walk_flag           // in: specify how deep
        = 0,                // to go
                            // 0=dmod only,
                            // 1=dmod and offspring
                            // 2=dmod, siblings, and
                            // offspring
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the parameters that control the rendering of curvature combs, and curvature values for curvature point constraints. Curvature values are plotted for points on curves as a vector. The plotted vector is attached to the curve at the point being measured and points in the direction of the normal to the curve at that point. The curvature vector's magnitude is equal to the curvature * the input comb_gain value. So, for a given curve W=W(u)=[x(u0,y(u0,z(u)]t, the curvature vector plotted for some point, u0, on that curve is:

```
plotted curvature base_pt = W(u0)
plotted curvature vector = comb_gain * k * m
```

where

- k = curvature = size(cross_product(Wu,Wuu)) / size(Wu)**3
- m = normal direction = cross_product(t,b)
- Wu = dW(u0)/du = 1st parametric derivative taken at location u0.
- Wuu = d2W(u0)/du2 = 2nd parametric derivative taken at location u0.
- t = tangent = normalized(Wu)
- b = bi–normal = normalized(cross_product(Wu,Wuu))

A curvature comb report is generated by plotting several curvature vectors along the length of the curve. The input value comb_pt_count specifies how many evenly spaced curvature vectors to plot within each element of the target curve. The comb_pt_count value does not affect the rendering of curvature values for point constraints.

A unique set of curvature plotting parameters are stored for each deformable model within a deformable modeling hierarchy. For convenience, this command can apply the input curvature comb parameters to a set of deformable models with one call as determined by the value of walk_flag. When walk_flag is 0, the parameters are applied only to the target input dmod. When walk_flag is 1, the parameters are applied to the target dmod and all of its offspring, and a walk_flag value of 2 applies the parameters to the target dmod, its younger siblings, and all their offspring.

The comb_gain value is used when using the DM_set_pt_xyz() function to place the curvature vector end–point to determine the new target value for the curvature at that point.

Errors: DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_cstrn_behavior

Function: Deformable Modeling, DML Constraints

Action: Modifies the behavior of a constraint and returns 0 for success or an error.

Prototype:

```
void DM_set_cstrn_behavior (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: member of dmod
                               // hierarchy tree
    int tag,                   // in: unique tag
                               // identifier to match
    int behavior,              // in: or of
                               // DM_POS_FREE
                               // DM_POS_FIXED
                               // DM_POS_LINKED
                               // DM_POS_2_FREE
                               // DM_POS_2_FIXED
                               // DM_POS_2_LINKED
                               // DM_TAN_FREE
                               // DM_TAN_FIXED
                               // DM_TAN_LINKED
                               // DM_TAN_2_FREE
                               // DM_TAN_2_FIXED
                               // DM_TAN_2_LINKED
                               // DM_CURV_FREE
                               // DM_CURV_FIXED
                               // DM_CURV_LINKED
                               // DM_CURV_2_FREE
                               // DM_CURV_2_FIXED
                               // DM_CURV_2_LINKED
                               // DM_NORM_FIXED
                               // DM_BINORM_FIXED
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies the tag's behavior bit array and sets the active deformable model. This function passes the change request down to the deformable model so that all affected arrays can be updated. The behavior argument is a bit array composed as an OR of the following bit constants:

```
DM_POS_FREE     DM_POS_2_FREE       DM_CURV_FREE
DM_POS_FIXED    DM_POS_2_FIXED      DM_CURV_FIXED
DM_POS_LINKED   DM_POS_2_LINKED     DM_CURV_LINKED
DM_TAN_FREE     DM_TAN_2_FREE       DM_CURV_2_FREE
DM_TAN_FIXED    DM_TAN_2_FIXED      DM_CURV_2_FIXED
DM_TAN_LINKED   DM_TAN_2_LINKED     DM_CURV_2_LINKED
DM_NORM_FIXED   DM_BINORM_FIXED
```

The basic flavor for using a constraint is:

1. Create and apply the constraint to a deformable model,
   (DM_add_pt_cstrn()),
   (DM_add_crv_cstrn()),
   (DM_add_link_cstrn()),
   (DM_add_area_cstrn()).
2. Select which geometry properties are being constrained,
   (DM_set_cstrn_behavior()).
3. For pt_cstrns, modify the geometry property values by moving the point constraint display points as described above,
   (DM_set_pt_xyz()).
4. Call solve to see how the constraint modifications changed the deformable model shape,
   (DM_solve()).
5. Optionally, for pt_cstrns, the point constraint's domain position and domain directions (for surfaces) may be modified with
   (DM_set_pt_uv()),
   (DM_set_cstrn_pttan_uv_dir()).
6. Optionally, modify the constraint rendering and image_pt locations to optimize viewing and/or interactions,
   DM_set_tan_display_gain(),
   DM_set_comb_graphics().

Not all the behavior bits are used by all the constraints. Behavior bits not used by an object are ignored by that object. Point constraints on curves do not use the ..._LINKED or the ..._2_... behaviors. Point constraints on surfaces do not use the ..._LINKED or the DM_BINORM_FIXED bits. Curve constraints do not use the DM_NORM_FIXED or the DM_BINORM_FIXED bits.

Point constraints couple some of the constraint behaviors together. Turning on the following behavior causes other behaviors to be turned on as follows:

DM_CURV_FIXED turns on DM_TAN_FIXED and DM_NORM_FIXED.

DM_CURV_2_FIXED turns on DM_TAN_2_FIXED and DM_NORM_FIXED.

For curves DM_NORM_FIXED turns on DM_TAN_FIXED.

Turning off a cascaded behavior without turning off the behavior which causes it to be turned on won't work. The behavior state will remain unmodified.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a constraint in the model hierarchy.

DM_BAD_CSTRN_CHANGE
The identified constraint's state prohibits toggling of DM_POSITION behavior. This would be for unstoppable constraints.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_cstrn_pttan_uv_dir

Function: Deformable Modeling, DML Constraints

Action: Sets the surface domain space direction for a point constraint's tangent and curvature constraint behaviors.

Prototype:

```
void DM_set_cstrn_pttan_uv_dir (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: member of dmod
                            // hierarchy tree
    int tag,                // in: point constraint
                            // identifier
    double* domain_dir,     // in: uv direction in
                            // which to evaluate
                            // tangent vector
    int which_dir,          // in: which uv
                            // direction is being
                            // specified
                            // one of:
                            // DM_TANG1_LEG
                            // DM_TANG2_LEG
                            // DM_CURV1_LEG
                            // DM_CURV2_LEG
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the domain space direction on a surface in which to evaluate a point tangent or curvature constraint. The direction is a unit vector in the "domain_dir" direction. For deformable surfaces, two domain space directions can be specified at a point; the which_dir argument selects between them. This function along with DM_set_pt_xyz() and DM_set_cstrn_behavior() gives complete run time control of point tangent and curvature constraints.

Errors: DM_NULL_INPUT_PTR
Neither the deformable model nor the domain_dir can be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a constraint in the model hierarchy.

DM_BAD_WHICH_DIR_VALUE
The input which_dir value was not one of DM_TANG1_LEG, DM_TANG2_LEG, DM_CURV1_LEG, or DM_CURV2_LEG.

DM_ZERO_LENGTH_DOMAIN_DIR
When input domain direction vector has a zero length.

DM_PARALLEL_DOMAIN_DIRS
When the input domain directions is parallel to the other domain direction.

DM_NOT_A_PT_CSTRN
The constraint identified by the input tag value was not a point constraint.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_cstrn_src_data

Function: Deformable Modeling, DML Constraints

Action: Modifies the value of the pass through src_data pointer stored with each constraint.

Prototype:

```
void DM_set_cstrn_src_data (
    int& rtn_err,           // out: 0=success or
                            // neg err code
    DS_dmod* dmod,          // in: member of target
                            // dmod hierarchy
                            // to search
    int tag,                // cstrn identifier
    int tgt                 // for link_cstrns,
        = 1,                // tgt = 1 or 2
    void* src_data          // new src_data ptr value
        = NULL,             //
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the tag constraint's src_data pointer value. The src_data pointer value is stored with each constraint as a convenience for the application program which might want to store application specific data. The DM Modeler package never accesses or uses the src_data pointer. Calling applications that use this pointer must guard against memory leaks on their own as the constraints are deleted.

Errors:

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on input.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a constraint in the deformable model hierarchy.

DM_BAD_SRC_DATA_TGT_VALUE
The input target value must be 1 or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_cstrn_src_pfuncs

Function: Deformable Modeling, DML Constraints

Action: Sets a curve or link constraint's source DS_pfunc objects, which are used to specify the shape of the constraint.

Prototype:

```
void DM_set_cstrn_src_pfuncs (
    int& rtn_err,            // out: 0=success or
                             // neg err code
    DS_dmod* dmod,           // in: member of target
                             // dmod hierarchy
                             // to search
                             // cstrn identifier
                             // for link_cstrns
    int tag,                 // cstrn identifier
    int tgt,                 // for link_cstrns,
                             // tgt = 1 or 2
    DS_pfunc* src_W_pfunc,   // in: cstrn pos shape,
                             // [nested]
    DS_pfunc* src_Wn_pfunc,  // in: cross-tang shape,
                             // [nested]
    DS_pfunc* src_Wnn_pfunc,// in: curvature shape,
                             // [nested]
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the target curve or link constraint's defining source DS_pfunc objects. When supplied, these shapes are used to specify the target location for the constrained surface. Whenever any of the input values are NULL, the internal DS_pfunc object is removed from the constraint data structure and the shape of the constraint is defined as the current shape of the constrained surface. Whenever any of the input values equal an existing stored value, no actions are taken on that object.

When a previously stored DS_pfunc object is replaced or removed from its constraint with this call, its reference count is decremented. If that reference count reaches zero then the DS_pfunc will be automatically deleted. Calling delete on any of the src_pfunc objects will cause a memory error.

Refer to functions DM_add_crv_cstrn(), DM_add_link_cstrn(), and DM_get_cstrn_src_pfuncs().

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on input.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a constraint in the deformable model hierarchy.

DM_NOT_A_CRV_LINK_CSTRN
The input tag did not identify a curve or link constraint.

DM_BAD_SRC_DATA_TGT_VALUE
The input target value must be 1 or 2.

DM_BAD_SRC_PFUNC_MAPPING
Whenever a src_pfunc does not have a proper domain_dim or image_dim value. The domain_dim value for all src_pfuncs must be 1 (they are all curves), and the image_dim for the src_W and src_Wn DS_pfuncs must equal the image_dim of the object being constrained, and the image_dim of the src_Wnn DS_pfunc must be 1.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_cstrn_state

Function: Deformable Modeling, DML Constraints

Action: Modifies the enabled/disabled state for a deformable modeling constraint.

Prototype:

```
void DM_set_cstrn_state (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: root of
                               // hierarchy tree
    int tag,                   // in: unique tag
                               // identifier to match
    int state_flag,            // in: –1= toggle current
                               // cstrn state,
                               // 0 = turn cstrn off
                               // 1 = turn cstrn on
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Searches the deformable model hierarchy for the constraint with a given input tag value and modifies that constraint's enabled/disabled behavior bit and sets the active deformable model.

Passes the change request down to the deformable model so that all affected arrays can be updated.

Errors:

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a deformable model, a load, a spring, a spring set, an attractor object, or a constraint in the model hierarchy.

DM_BAD_CSTRN_CHANGE
The crv_cstrn prohibits varying its curvature gain.

DM_BAD_STATE_FLAG_VALUE
The state_flag must be equal to –1, 0, or 1.

DM_BAD_DOMAIN_PT_RANGE
The domain point must be completely contained by the deformable model's pfunc.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_cstrn_value

Function: Deformable Modeling, DML Constraints, DML Loads

Action: Sets the value for a point constraint's constraint behavior.

Prototype:

```
void DM_set_cstrn_value (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: member of target
                               // dmod hierarchy to
                               // search
    int tag,                   // in: unique tag
                               // identifier to match
    int pt_index,              // in: index of tag's
                               // image_pt to update
                               // one of DM_BASE_PT
                               //      DM_TANG_LEG
                               //      DM_TANG1_LEG
                               //      DM_TANG2_LEG
                               //      DM_NORM_LEG
                               //      DM_CURV1_LEG
                               //      DM_CURV2_LEG
                               //      DM_BINORM_LEG
    int cstrn_val_count,       // in: size of cstrn_val
                               // array
    double* cstrn_val,         // in: value to assign to
                               // constraint behavior.
                               // sized:
                               // [cstrn_val_count]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function sets the value for a point constraint's behavior and modifies its related display point location. Constraint values are all associated with a display point which can often be used for updating the constraint values in a more intuitive fashion. Refer to DM_get_pt_xyz() and DM_set_pt_xyz().

The input arguments dmod and tag identify which point constraint to modify. The argument pt_index selects which of the point constraint's behavior values to update. Accepted values for pt_index include:

DM_BASE_PT	= set base point position. cstrn_val_count = Image_dim
DM_TANG_LEG or DM_TANG1_LEG	= set tangent vector in the domain1_dir direction. cstrn_val_count = Image_dim
DM_TANG2_LEG	= set tangent vector in the domain2_dir direction. cstrn_val_count = Image_dim
DM_CURV_LEG or DM_CURV1_LEG	= set curvature value in the domain1_dir direction. cstrn_val_count = 1
DM_CURV2_LEG	= set curvature value in the domain2_dir direction. cstrn_val_count = 1
DM_NORM_LEG	= set normal vector direction for a curve or surface. cstrn_val_count = Image_dim
DM_BINORM_LEG	= Set binormal vector direction for a curve. cstrn_val_count = Image_dim

The desired value for the selected constraint behavior is passed in through the pointer, cstrn_val, which is treated as a cstrn_val_count length array. The constraint behavior value will be an image space position, vector, or scalar value depending on which constraint behavior is being updated as indicated above.

Modifying a constraint behavior value will cause the associated constraint's display point to be repositioned.

This function is provided to simplify accessing point constraint internal values directly by an application. The deformable modeling library requires that certain relationships be maintained by the these internal values. It is the responsibility of the calling application to ensure that these relationships are satisfied prior to the next call to DM_solve(). Otherwise, conflicting constraints may be found. These requirements can be automatically met by using the function DM_set_pt_xyz() which can automatically change some constraint values to preserve these relationships.

The pt_cstrn constraint relationships include the following. Only those rules that use properties which are being constrained by the pt_cstrn's behavior (see DM_set_cstrn_behavior) need to be enforced. Unconstrained properties will satisfy these rules after the next call to DM_solve().

Rules enforced on the constraint values for DS_pt_cstrns as follows:

For curves in 2d:

- tang1_val perp to norm_val
- curv1_val not negative
- norm_val is an unit vector

For curves in 3d:

- tang1_val perp to norm_val
- norm_val perp to binorm_val
- binorm_val perp to tang1_val
- curv1_val not negative
- norm_val is an unit vector
- binorm_val is an unit vector

For surfaces:

- domain1_dir is an unit vector
- domain2_dir is an unit vector
- domain1_dir not parallel to domain2_dir
- tang1_val perp to norm_val
- tang2_val perp to norm_val
- tang1_val, tang2_val, and norm_val make a right hand coordinate system (note depends domain1_dir and domain2_dir)
- norm_val is unit vector
- no restrictions on curv1_val or curv2_val

Errors: DM_NO_ROOT_DMOD
The root deformable model cannot be NULL on input.

DM_NULL_INPUT_PTR
When cstrn_val is NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag does not identify any tag object in the model hierarchy.

DM_NOT_A_PT_CSTRN
The input tag does not identify a point constraint in the model hierarchy.

DM_BAD_PT_INDEX_VALUE
When pt_index is not an appropriate value.

DM_BAD_CURVATURE_VALUE
When input curvature value for a curve is not zero or positive.

DM_BAD_CSTRN_VAL_SIZE
When input cstrn_val_count is not appropriate for the selected constraint behavior.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_default_shape

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's default shape and returns 0 for success or an error.

Prototype:

```
void DM_set_default_shape (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to modify
    int shape_flag,         // in: 0=disable
                            // default shapes
                            // 1=set current shape as
                            // default shape
    int walk_flag           // in: specify how deep
        = 0,                // to go
                            // 0=dmod only,
                            // 1=dmod and offspring
                            // 2=dmod, siblings,
                            // and offspring.
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_default_shape

Sets the deformable model's resistance to displacement default shape values. A walk_flag value of 0 specifies that just the deformable model is modified, a walk_flag value of 1 specifies that the deformable model and all its offspring are modified, and a value of 2 specifies that the deformable model, its siblings, and all of their offspring are modified.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_SHAPE_FLAG_VALUE
The default shape must be 0 or 1.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_delta

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's delta and returns 0 for success or an error.

Prototype:

```
void DM_set_delta (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: dmod to modify
    double delta               // in: desired value
        = 0.0,                 // to store
    int walk_flag              // in: specify how deep
        = 0,                   // to go
                               // 0=dmod only,
                               // 1=dmod and offspring
                               // 2=dmod, siblings
                               // and offspring
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_delta.

Sets the deformable model's resistance to displacement delta values. A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified. A walk_flag value of 2 specifies that the deformable model, its siblings, and all their offspring are modified.

The delta term was added as an experiment to approximate the behavior of the default shape mechanism before it was built. Since default shapes have been added to the system, we do not recommend that you use a non–zero value for delta when sculpting. The default shapes with delta set to 0 are much more intuitive to use.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_dmod_tag

Function: Deformable Modeling, DML Create and Query, DML Tags

Action: Checks and returns the tag type for DS_pfunc and returns 0 for success or an error.

Prototype:
```
void DM_set_dmod_tag (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to modify
    int tag,                // in: unique tag
                            // identifier to match
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Sets the tag number for the input deformable model.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_draw_state

Function: Deformable Modeling, DML Graphics

Action: Sets the target deformable model's draw_state.

Prototype:

```
int DM_set_draw_state (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: target dmod
    int draw_state,         // desired value to store
    int walk_flag           // in: specify how deep
        = 0                 // to go
                            // 0=dmod only,
                            // 1=dmod and offspring
                            // 2=dmod, siblings
                            // and offspring
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_draw_state

Sets the deformable model's draw_state value and returns 0 when no bits in draw_state are changed, and returns 1 when bits are altered.

A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified. A walk_flag value of 2 specifies that the deformable model, its siblings, and all their offspring are modified.

These draw bits are used to cull tag objects from consideration by the function DM_find_tag_by_image_line() which finds and returns the closest tag object to an image space line.

The set of values for draw_state may be made by the OR of the DM_DRAW_... variables defined in the "dmapi.hxx".

Errors: DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_dynamics

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's dynamics and returns 0 for success or an error.

Prototype:
```
void DM_set_dynamics (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to modify
    double dt               // in: dynamic sculpting
        = 1.0,              // time step size
    double mass             // in: dynamic deformable
        = 1.0,              // mass value
    double damp             // in: dynamic deformable
        = 5.0,              // damping value
    int walk_flag           // in: specify how deep
        = 0,                // to go
                            // 0=dmod only,
                            // 1=dmod and offspring
                            // 2=dmod,siblings, and
                            // offspring
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dynamics terms

Sets the deformable model's dt, mass, and damping dynamics values. See DM_get_dynamic.

To turn off dynamic behavior, set both mass and damp to 0.0.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

DM_BAD_DT_VALUE
The value of dt must be positive.

DM_BAD_MASS_VALUE
The value of mass must be 0 or positive.

DM_BAD_DAMP_VALUE
The value of damp must be 0 or positive.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_end_conds

Function: Deformable Modeling, DML Constraints

Action: Sets the end states for a deformable model and returns 0 for success or returns an error.

Prototype:

```
void DM_set_end_conds (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: root of
                            // hierarchy tree
    int end_cond_u,         // in: 0=open 1=closed
                            // 2=periodic
    int singular_u,         // in: 0=none 1=low
                            // 2=high 3=both
    int end_cond_v,         // in: 0=open 1=closed
                            // 2=periodic
    int singular_v,         // in: 0=none 1=low
                            // 2=high 3=both
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: only end_cond_u
                            // used for 1d curves
                            // note: only dmod may be
                            // closed or periodic
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>end_conds and deformable model–>singulars.

Assigns values to the end condition descriptions contained in the deformable model–>pfunc(). Only the root deformable model may have non–open end conditions at this time.

end_cond:
0 = open, no effect on surface
1 = closed, end control points are constrained to be the same
2 = periodic, end tangents are constrained to be the same

singular:
0 = none, no effect
1 = low, all control points on low boundary are constrained to be the same
2 = high, all control points on high boundary are constrained to be the same
3 = both, all control points on low boundary are constrained to be the same and all control points on high boundary are constrained to be the same.

Errors: DM_DMOD_NOT_A_ROOT_DMOD
The deformable model must be the root of a hierarchy tree.

DM_BAD_END_COND_VALUE
The end condition values must be 0, 1, or 2.

DM_BAD_SINGULAR_VALUE
The singular values must be 0, 1, 2, or 3.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_entity

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's entity pointer value and returns 0 for success or an error.

Prototype:
```
void DM_set_entity (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to modify
    void* entity,           // in: desired value
                            // to store
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_entity

Sets the deformable model's application entity pointer value.

This value is never used by the deformable modeling library it is just stored and retrieved with the deformable model for an application's convenience.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_epsilon

Function: Deformable Modeling

Action: Sets the deformable model's epsilon value and returns 0 or an error.

Prototype:
```
void DM_set_epsilon (
    int& rtn_err,           // return code
                            // 0=success, neg=error
    DS_dmod* dmod,          // tag object identifier
    int tag,                // tag object identifier
    double eps,             // epsilon
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "kernel/acis.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Epsilon regulates a shape fairing (energy minimization) term that is used to dampen control point oscillations in high degree splines (degree>8).

Like the primary fairing terms, alpha and beta, epsilon should be 0 or positive. Epsilon is considered a supplement to the alpha and beta shape fairing terms, and should be relatively small compared to beta, the chief shape fairing term.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_NOT_A_DMOD_MEMBER
The tag_flag does not identify a deformable model in the patch hierarchy.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_gamma

Function: Deformable Modeling

Action: Sets the deformable model's gamma and returns 0 for success or an error.

Prototype:

```
void DM_set_gamma (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: dmod to modify
    double gamma               // desired value to
        = 0.0,                 // store
    int walk_flag              // in: specify how deep
        = 0,                   // to go
                               // 0=dmod only,
                               // 1=dmod and offspring
                               // 2=dmod, siblings
                               // and offspring
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_gamma

Sets the deformable model's resistance to bending rate of change gamma values.

A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified. A walk_flag value of 2 specifies that the deformable model, its siblings, and all their offspring are modified. are modified. See DM_get_gamma.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_icon

Function: Deformable Modeling, DML Graphics

Action: Sets the icon for a tag object.

Prototype:
```
void DM_set_icon (
    int& rtn_err,           // error code
    DS_dmod* dmod,          // tag object ID
    int tag,                // tag object ID
    DM_icon* dmicon,        // new icon
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dmicon.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This call should be followed with a call to DM_set_icon_owner to allow the icon to initialize itself. rtn_err returns 0 for success or a negative error code.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
Could not find tag object with the given tag.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_set_icon_owner

Function: Deformable Modeling, DML Graphics

Action: Sets the icon's owner.

Prototype:

```
void DM_set_icon_owner (
    int& rtn_err,           // error code
    DS_dmod* dmod,          // tag object ID
    const int* tags,        // tag array
    int ntags,              // number of tags
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );

void DM_set_icon_owner (
    int& rtn_err,           // error code
    DS_dmod* dmod,          // model
    int tag,                // tag object ID
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This API serves as notification to the icon that the owning tag object is complete, and the icon can be initialized. rtn_err returns 0 for success or a negative error code.

Errors:

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
Could not find tag object with the given tag.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_set_interior_state

Function: Deformable Modeling, Deformable Model Management

Action: Sets the deformable model's interior state value and returns 0 for success or a negative error code.

Prototype:
```
void DM_set_interior_state (
    int& rtn_err,           // out: 0=success
    DS_dmod* dmod,          // or negative err code
    int interior_state,     // in: dmod to modify
    int walk_flag,          // in: 0=allow C0 bends
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model's interior state value.

When interior_state == 0, the Deformable model is allowed to bend with C0 discontinuity between elements. For B–splines and NURBs, C0 discontinuity within a surface can be achieved by increasing the knot count at an internal knot boundary.

When interior_state == 1, the Deformable model prohibits C0 bending between elements by adding C1 internal tangent constraints between any elements whose internal representation will allow a C0 bend. For B–splines and NURBs, this means that the deformation will be at least C1 everywhere even if the underlying representation has multiple knots that would normally allow a C0 internal bend.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_INTER_STATE_VALUE
The interior state must be 0 or 1.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_load_gain

Function: Deformable Modeling, DML Loads

Action: Sets the deformable model's load_gain and returns 0 for success or an error.

Prototype:
```
void DM_set_load_gain (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: member of target
                            // dmod hierarchy to
                            // search
    DS_TAGS& type,          // i/o: in: type of
                            // loads to modify
                            // (used when group_flag
                            // = 1)
                            // out: modified load or
                            // DS_tag_none
    int tag,                // in: unique tag
                            // identifier to match
    double gain,            // in: desired value to
                            // store
    int group_flag          // in: 0=set tag object
        = 0,                // only
                            // 1=set all active dmod
                            // loads of type
    int inc_flag            // in: 0:tag(gain) = gain
        = 0,                // 1:tag(gain)+= gain
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dmapinum.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies tag–>DS_load–>lds_gain and sets the active deformable model.

Sets the tag load's gain value, sets the containing deformable model as the active deformable model and returns 0 when the tag identifies a load object. Otherwise, returns DM_TAG_OBJECT_NOT_FOUND and leaves output arguments unmodified.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_NO_ACTIVE_DMOD
The input deformable model's hierarchy has no currently active deformable model .DM_BAD_PT_COUNT_VALUE

DM_TAG_OBJECT_NOT_FOUND when the input tag value fails to identify a load or constraint

DM_BAD_GROUP_FLAG_VALUE
The group_flag must be 0 or 1.

DM_BAD_INC_FLAG_VALUE
The inc_flag must be 0 or 1.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_mesh_count

Function: Deformable Modeling, DML Graphics

Action: Sets the deformable model's mesh_count and returns 0 for success or an error.

Prototype:

```
int DM_set_mesh_count (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: dmod to modify
    int mesh_u,                // desired u_dir polygon
                               // count (crvs & srfs)
    int mesh_v,                // desired v_dir polygon
                               // count (srfs only)
    int walk_flag              // in: specify how deep
        = 0,                   // to go
                               // 0=dmod only,
                               // 1=dmod and offspring
                               // 2=dmod, siblings, and
                               // offspring
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_mesh_grid

Sets the parameters of the deformable model's mesh_grid. mesh_u is the number of polygons to draw in the *u* direction and mesh_v is the number of polygons to draw in the *v* direction. See DM_get_mesh_count() for a description of rendering meshes for deformable models. Returns 0 when the mesh parameters already equal the input values and no changes have to be made, otherwise returns 1 when changes to these values are made.

A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified. A walk_flag value of 2 specifies that the deformable model, its siblings, and all their offspring are modified.

The comb_graphic parameter values are not used in this library but may be passed to a rendering module.

Errors: DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

DM_BAD_DRAW_MESH_VALUE
The values of mesh_u and mesh_v cannot be 0.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_patch_continuity

Function: Deformable Modeling, DML Patches

Action: Sets patch–to–parent continuity.

Prototype:
```
void DM_set_patch_continuity (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: target dmod
    int continuity,         // in: –1=Toggle between
                            // C0, C1, and C2
                            // 0=C0 (position
                            // continuity)
                            // 1=C1 (pos and tang
                            // continuity)
                            // 2=C2 (pos,tang,curv
                            // continuity)
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>dmo_seam_state.

A convenience routine that sets the behaviors of all the seam or link constraints in one deformable model to a common value. This command is equivalent to running the command DM_set_cstrn_behavior() once on all the seams or links within the target deformable model. There is no requirement that all the seams and links of one deformable model share a common continuity behavior.

For child patches, the continuity behavior of all the patch's seams are modified, for root sibling patches, the continuity behavior of all the patch's link constraints are modified.

A patch may be connected to another patch with either C0 (position), C1 (position and tangent) or C2 (position, tangent, and curvature) continuity. The value of the input argument, continuity, specifies the desired continuity as follows:

continuity = 0	for position.
continuity = 1	for position and tangent.
continuity = 2	for position, tangent, and curvature.
continuity = –1	to increment continuity from C0 to C1 to C2 and back to C0 again.

Errors: DM_BAD_CONTINUITY_VALUE
when the continuity input value is not 0 for C0 continuity, 1 for C1 continuity, 2 for C2 continuity, or –1 for toggling.

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: Currently curvature constraints are only available for curve patches 2 for C2 continuity, or –1 for toggling.

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_pfunc_default_state

Function: Deformable Modeling, DML Create and Query

Action: Sets the pfunc's default state.

Prototype:

```
void DM_set_pfunc_default_state (
    int& rtn_err,           // out: 0=success or
                            // neg err code
    DS_pfunc* pfunc,        // pfunc to query
    int default_state,      // 0=no default shape
                            // being used
                            // 1=default shape
                            // being used
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dspfunc.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function modifies the pfunc's default state value and sets the pfunc's internal dof_def array of default shape dof values appropriately. When the default_state equals 0, the pfunc will not be using a default shape and the dof_def values are all set to 0.0. When the default_state equals 1, the pfunc will be using a default shape and the dof_def values are all copied from the internal dof_vec array.

Note *If this* pfunc *is nested within a* DS_dmod *object do not use this function to change its default state. Instead, use the* DM_set_default_state *function which works directly on the* DM_dmod *object.*

Errors: DM_NULL_INPUT_PTR
The pfunc cannot be NULL on entry.

DM_NULL_OUTPUT_PTR
The domain_max cannot be NULL on entry,

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_pt_uv

Function: Deformable Modeling, DML Constraints, DML Loads

Action: Sets domain points in a load or and returns 0 for success or an error.

Prototype:

```
DS_TAGS DM_set_pt_uv (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: member of target
                               // dmod hierarchy to
                               // search
    int tag,                   // in: unique tag
                               // identifier to match
    int domain_flag,           // in: 0=dpt in
                               // orig_dmod_space,
                               // 1=dpt in unit_space
                               // 2=dpt in
                               // internal_pfunc_space.
    double* dpt,               // in: pt's domain loc
                               // in unit square,
                               // sized:[Domain_dim]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dmapinum.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies tag–>Domain_pt(), active deformable model

Searches for a constraint or load with a matching tag. When found sets its domain point to the given input through the deformable model so that the deformable model is aware of the change. Works for tags of type DS_pressure_point, DS_spring, DS_point_constraint. For all other tag types returns DM_NO_UV_PT_FOR_TAG_OBJ.

Errors: DM_NO_ROOT_DMOD
The root deformable model cannot be NULL on input.

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a deformable model, a load, a spring, a spring set, an attractor object, or a constraint in the model hierarchy.

DM_NO_UV_PT_FOR_TAG_OBJ
The identified tag object has no distinct *uv* point to update dyn_loads, dist_press, crv_loads and crv_cstrns, or the deformable model, or when the input dpt is not contained by the target deformable model.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_pt_xyz

Function: Deformable Modeling, DML Constraints, DML Loads

Action: Sets the image space point for a constraint or control point.

Prototype:
```
DS_TAGS DM_set_pt_xyz (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: member of target
                            // dmod hierarchy to
                            // search
    int tag,                // in: unique tag
                            // identifier to match
    int pt_index,           // in: index of tag's
                            // image_pt to update
                            // one of DM_BASE_PT
                            //      DM_END_LEG
                            //      DM_TANG_LEG
                            //      DM_TANG1_LEG
                            //      DM_TANG2_LEG
                            //      DM_NORM_LEG
```

```
                                //      DM_CURV1_LEG
                                //      DM_CURV2_LEG
                                //      DM_BINORM_LEG
                                // or a control point tag
                                // or an id number in a
                                // spring set
double* p0,                     // new ipt or p0 of
                                // iline=p0+u*(p1-p0)
                                // [not-nested]
                                // sized:[image_dim]
double* p1,                     // in: NULL or p1 of
                                // iline=p0+u*(p1-p0)
                                // [not-nested]
                                // sized:[image_dim]
int dir_flag,                   // in: 0=let image_pt
                                // equal P0,
                                // for pt_index
                                // == DM_BASE_PT
                                //  1=move parallel to
                                //  xy plane,
                                //  2=move parallel to
                                //  z axis.
                                // else
                                //  1=rotate current
                                // vector,
                                //  2=change vector
                                // magnitude.
int cascade_flag                // in: 2=move tangent
    = 1,                        // vecs independently,
                                // but update
                                // normal vecs.
                                //  1=update related
                                // pt_cstrn
                                // values and image_pts.
                                //  0=don't.
int curvature_sign              // in: used when
    = 0,                        //  dir_flag ==0 &&
                                //  domain_dim==2 &&
                                //  PT_index==PST_CURV1
                                //  or PST_CURV2.
                                // +1: use curv_pt =
                                //      k * n,
                                //      (k positive),
                                // -1: use curv_pt =
                                //      -k * -n,
```

```
                                    //  (k negative),
                                    // to solve for k and n
                                    // vals.
    SDM_options* sdmo               // in:SDM_options pointer
        = NULL                      //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/dmapinum.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description:

This function sets the location for an image_point or a control point of the tag object identified by the input tag value.

When p1 is a reference to a NULL pointer, the tag's image point is set to the value p0. To help support mouse pick–and–drag, this subroutine can be used to compute the tag's new image point location from an image space line. The image space line or iline, is specified as the line which runs through p0 and p1. Usually this line will be generated from the window system's pick–ray and pick–point for a user–generated pick event. Four different new point calculations are supported based on the dir_flag, pt_index and tag_type values.

When pt_index equals DM_BASE_PT and dir_flag equals 1, the new image point is the intersection between the image line and the plane parallel to the *xy* axis that contains the current image point location. This is good for dragging a point in its *xy* plane.

When pt_index equals DM_BASE_PT and dir_flag equals 2, the new image point is the point on a line parallel to the *z* axis containing the current image point location that is nearest to the input image line. This is good for dragging a point along its *z* axis.

For objects with vectors (such as tangent point constraints and vector loads) the input pt_index value specifies whether the vector's base point or one of the end points is moved. The object's vector runs from the vector's base point to its end point.

When pt_index equals DM_BASE_PT, the vector's base point and end point are both modified so that the vector's magnitude and direction remains constant while its position is modified. When pt_index equals any option that specifies an end point, e.g., DM_TANG1_LEG, the vector's end point is modified while the base point remains fixed. The vector's position remains constant while its magnitude and direction are modified.

When the pt_index corresponds to a vector end point and dir_flag equals 1, the new image position of the end point is the intersection of the input image line and a hemisphere centered on the point constraint's base point with a radius equal to the current tangent vector's magnitude. This is good for rotating a vector by dragging.

When pt_index corresponds to one of the vector end points and dir_flag equals 2, the new image position of the end point is the point on a line running through the vector's base point and end point closest to the input image line. This is good for changing the magnitude of a vector by dragging.

Tag numbers less than –500 are used to change control point locations. When moving a control point location, the control point on the active deformable model is moved.

When pt_index equals DM_CURV1_LEG or DM_CURV2_LEG the vector is constrained to remain perpendicular to the point constraint's tangent vector. In this case the dir_flag equals 1 option will only rotate the vector in the plane perpendicular to the tangent vector.

When pt_index equals DM_TANG1_LEG or DM_TANG2_LEG and the object being manipulated is a pt_cstrn then the point constraint's DM_CURV1_LEG and DM_CURV2_LEG values are updated to keep the curvature vector perpendicular to the tangent and to keep the curvature value a constant. The osculating plane will be reoriented in an unspecified, but minimal, fashion. Modify the DM_CURV1_LEG or DM_CURV2_LEG plane as desired.

When dir_flag equals 0, no constraints are applied to the input parameter and it is the responsibility of the calling program to make supply a valid position, e.g., the normal vector is perpendicular to the tangent and so on.

To summarize, for tag objects with vectors and a pt_index equal to 1:

dir_flag = 1	Sets the point closest to the pick ray on a base point centered hemisphere (for rotation).
dir_flag = 2	Sets the point closest to the pick ray on the base point, end point line (base_pt, end_pt) line (for scaling).

For cases where pt_index is equal DM_BASE_PT, or the index for a spring_set or a control point tag:

dir_fla = 1	Sets the point at the intersection of the pick ray and the image point's *xy* plane (move in xy plane).
dir_flag = 2	Sets the point closest to the pick ray on the image point's z_vec line (move in z direction).

These four cases are intended to allow an application interface to move an *xyz* point so that it tracks a 2D mouse location and moves in three dimensional space in a reasonable fashion.

Note

- Use DM_END_LEG for the vector loads,
- Use DM_TANG_LEG for the pt_cstrn on a curve tangent end_pt,
- Use DM_TANG1_LEG for the pt_cstrn on a surface's first tangent end_pt,
- Use DM_TANG2_LEG for the pt_cstrn on a surface's second tangent end_pt,
- Use DM_NORM_LEG for the pt_cstrn on a surface's normal vector end_pt,
- Use DM_CURV_LEG for the pt_cstrn on a curve's curvature vector end_pt,
- Use DM_CURV1_LEG for the pt_cstrn on a surface's first curvature end_pt,
- Use DM_CURV2_LEG for the pt_cstrn on a surface's second curvature end_pt,
- Use DM_BINORM_LEG for the pt_cstrn on a curve's binormal vector end_pt.

Setting the curvature constraint_value from the image_pt when constraining a surface (domain_dim = 2) is ambiguous. There are two solutions for k and n which satisfy the relation,

curv_pt = k * n and curv_pt = –k *–n

The first solution, k positive, will have a surface concave in the direction of the surface normal n. The second solution, k negative, will have a surface convex in the direction of the surface normal n. For curves, k is always positive. When setting the curv_pt location directly (dir_flag = 0) one must tell the library which case to use. Otherwise, it chooses to preserve the sign of the curvature when rotating the curvature vec (dir_flag = 1), and chooses to preserve the direction of the surface norm_val when lengthening the vector (dir_flag = 2).

Many of a pt_cstrn's constraint values are related. This function can update the pt_cstrn's constraint values one at a time, or as a convenience cascade a change to other cstrn values to make sure they are all in a valid state. When cascade_flag == 0, only the image_pt and its related cstrn_value indicated by the pt_index value will be updated. In this case the application will have to ensure that all the pt_cstrn value relationships are satisfied prior to calling DM_solve() otherwise a conflicting constraint may occur.

When cascade_flag = 1, all related cstrn_vals will be modified by a minimum amount to preserve the cstrn_val relationships. The angle between tangent vectors for point constraints on surfaces will be preserved. When cascade_flag == 2, all related cstrn_vals will be modified by a minimum amount, but the angle between tangent vectors for constraint points on surfaces is allowed to vary.

The pt_cstrn rules include the following. Only those rules that use properties which are being constrained by the pt_cstrn's behavior (see DM_set_cstrn_behavior()) need to be enforced. Unconstrained properties will satisfy these rules after the next call to DM_solve().

Rules enforced on the constraint values for DS_pt_cstrns.

1. for curves in 2d
1.a tang1_val perp to norm_val
1.b curv1_val not negative
1.c norm_val is an unit vector
2. for curves in 3d
2.a tang1_val perp to norm_val
2.b norm_val perp to binorm_val
2.c binorm_val perp to tang1_val
2.d curv1_val not negative
2.e norm_val is an unit vector
2.f binorm_val is an unit vector
3. for surfaces
3.a domain1_dir is an unit vector
3.b domain2_dir is an unit vector
3.c domain1_dir not parallel to domain2_dir
3.d tang1_val perp to norm_val
3.e tang2_val perp to norm_val
3.f tang1_val, tang2_val, and norm_val make a right hand coordinate system (note depends domain1_dir and domain2_dir)
3.g norm_val is an unit vector
3.h no restrictions on curv1_val or curv2_val

Errors: DM_NO_ROOT_DMOD
The root deformable model cannot be NULL on input.

DM_BAD_DIR_FLAG_VALUE
The value of dir must be 0, 1, or 2.

DM_NULL_INPUT_PTR
When the dir is 1 or 2, P1 cannot be NULL on entry.

DM_NULL_INPUT_PTR
The value of P0 or p1 cannot be NULL on entry

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a deformable model, a load, a spring, a spring set, an attractor object, or a constraint in the model hierarchy.

DM_NO_XYZ_PT_FOR_TAG_OBJ
The identified tag object has no distinct *xyz* point to report dyn_loads, dist_press, crv_loads and crv_cstrns, or deformable model.

DM_BAD_PT_INDEX_VALUE
The spring_set and pt_index must be within the point count range.

DM_PARALLEL_PROJECT_DIR
The the projection direction (i.e., the *z* axis, the *xy* plane, or the direction of a tangent vector) cannot be parallel to the image line. If executed, the projection would move the image_pt out to infinity.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_tag_count

Function: Deformable Modeling, DML Tags

Action: Sets the deformable model's dmo_tag_count value and returns 0 for success or an error.

Prototype:
```
void DM_set_tag_count (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: target dmod to
                            // modify
    int tag_count,          // largest tag value used
                            // by a tag object in the
                            // dmod hierarchy
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              //
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies deformable model–>root–>dmo_tag_count

Sets the value of the largest currently assigned tag value for the deformable model hierarchy. Be sure that this number will generate unique tag values for new tag objects being added to the current hierarchy. The system assumes each tag identifier for every tag object is unique.

Errors: DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_tan_display_gain

Function: Deformable Modeling, DML Graphics

Action: Sets display gain for the deformable model's tangent vectors and returns 0 for success or an error.

Prototype:

```
void DM_set_tan_display_gain (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    DS_dmod* dmod,          // in: dmod to modify
    double tan_display_gain // display scaling for
        = 1.0,              // tang_vec constraints
    int walk_flag           // in: specify how deep
        = 0,                // to go
                            // 0=dmod only,
                            // 1=dmod and offspring
                            // 2=dmod, siblings, and
                            // offspring
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies the deformable model dmo_tan_display_gain and sets the active deformable model.

Sets the displacement gain for the target deformable model. tan_display_gain may not be equal to 0.0.

Errors:

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_BAD_WALK_FLAG_VALUE
The walk_flag must be 0, 1 or 2.

DM_BAD_DISP_GAIN_VALUE
tan_display_gain cannot be 0.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_tight_state

Function: Deformable Modeling

Action: Modifies the tight enabled/disabled state.

Prototype:
```
void DM_set_tight_state (
    int& rtn_err,           // success or error code
    DS_dmod* dmod,          // root of deformable
                            // model hierarchy tree
    int tag,                // unique tag identifier
                            // to match
    int state_flag,         // how to set tight state
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: This function searches the deformable model hierarchy (using dmod) for the constraint with the given input tag value, and modifies that constraint's enabled/disabled behavior bit based on state_flag. The possible values for state_flag are:

–1	toggle current state
0	set to off (turn tight off)
1	set to on

This function passes the change request down to the deformable model so that all affected arrays can be updated. It also sets the active deformable model. The rtn_err indicates either success (if equal to 0) or error (a negative error code).

Errors:

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

DM_TAG_OBJECT_NOT_FOUND
The input tag cannot identify a deformable model, a load, a spring, a spring set, an attractor object, or a constraint in the model hierarchy.

DM_BAD_STATE_FLAG_VALUE
The state_flag must be equal to –1, 0, or 1.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_set_tolerance

Function: Deformable Modeling, DML Create and Query

Action: Sets tolerance limits and returns 0 for success or an error.

Prototype:
```
void DM_set_tolerance (
    int& rtn_err,           // out: 0=success
                            // or negative err code
    double dist_tol,        // in: min dist between
                            // two points
    double ang_tol,         // in: min angle between
                            // two lines
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: making these
                            // values too small will
                            // prevent the package
                            // from running
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies Library global values DS_tolerance and DS_angle_tol.

Sets the DS_tolerance and DS_angle_tol values.

The DS_tolerance value is used to determine when the curve constraints are being held. Due to machine round–off this number should be no more than 10 orders of magnitude less than the range of the image space. So if you build your models in a 10,000 unit cube, dist_tol should be no smaller than 1.0E–06.

The angle tolerance is not currently used.

Errors: DM_BAD_TOLERANCE_VALUE
The dist_tol or ang_tol must be positive numbers.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_solve

Function: Deformable Modeling, Deformable Model Management

Action: Computes a deformable model's optimal control point positions or its current set of constraints and loads.

Prototype:

```
void DM_solve (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: root of
                               // hierarchy tree
    int iter_count             // in: number of
        = 1,                   // time-steps to iterate
                               // or -1 to iter to
                               // convergence
    double max_move            // in: Used when
        = 0.0,                 // iter=-1, converges
                               // when dof max_motion
                               // is less than max_move
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description: Modifies the deformable model

Computes the optimal control–point positions for a deformable model given the deformable model's current set of loads, constraints, and deformation parameters using an optimization procedure. When the deformable model has been linked into a multi–deformable model mesh through the use of link_cstrns (see DM_add_link_cstrn) solve will simultaneously operate on all the deformable models within the deformable mesh.

After generating a solution for this deformable model or its deformable mesh, this function recursively generates a solve solution for all of this deformable model's siblings and offspring or all of this deformable model's mesh offspring. So a single call to this function on the root of a hierarchy tree will update all of the hierarchy's shapes. Alternatively, performance can be increased by calling this function on one of the descendents of a deformable modeling hierarchy. In this case solve will only be run on the passed in descendent and all of its younger siblings, and all of their offspring. Older siblings and ancestors will not be modified by such a call.

The state and parameter values of all of the deformable model's constraints, loads, and deformation parameters affect the computation. A change in any of these values will cause the shape of the deformable model to change on the next subsequent call to DM_solve().

The intended use of this function is to act as part of the deformable modeling interaction loop. The steps in this loop are as follows;

1. Construct a shape model of a curve or surface
2. Construct a deformable model for the shape model
3. Add constraints and loads to the deformable model
4. Modify the parameters of the constraints, loads, or deformation parameters.
5. Call DM_solve() to change the control point positions of the deformable model
6. Render the deformable model to see the change. You can loop back to steps 3 or 4 for interactive sculpting.
7. When satisfied, commit the deformable model control positions back to the original shape model.

The iter_count can be used to run a solve on a deformable model more than one time before returning. It is slightly more efficient to call DM_solve once with an iter_count rather than to call DM_solve iter_count number of times in a row. The results will be the same.

For most of the deformable modeling package a steady state solution is found with one call to solve. This means that a second call to solve will not change the shape of the deformable model. Some features within the package may require more than one call to solve before reaching a steady state solution. These features include, dynamics, pressure loads, and attractor loads, all of which are forces whose effects on the deformable model depend on the shape of the deformable model.

When iter_count is positive, solve will be run on the input deformable model iter_count number of times before returning.

When iter_count is negative, solve will run repeatedly until a steady state solution is achieved or until a maximum number of iterations are computed.

Steady state is achieved when the maximum motion of any one control point is less than the input max_move value. A default value of max_move equal to the square root of DS_tolerance (equal to SPAresabs in ACIS) is used when the input value of max_move is set to 0.0 or is negative. max_move is only used when iter_count is equal to –1.

The maximum iteration count is set to DS_SOLVE_MAX_ITER_COUNT. If DM_solve() ever reaches DS_SOLVE_MAX_ITER_COUNT number of iterations, the package assumes that a steady state solution is not possible and restores the deformable model to the shape it had when entering this call and returns an error code.

Calling DM_solve with iter_count equal to –1 when not using pressures, attractors, or dynamics will slow the system down. The package will execute two solves before it discovers that it achieved steady state in one iteration.

Errors: DM_CONFLICTING_CONSTRAINT
The solution process has stopped because some of the constraints were found to conflict with one another. If this happens the deformable model is returned in the same state as it was input to this routine.

DM_SOLVE_NEVER_CONVERGED
Solve iterated DS_SOLVE_MAX_ITER_COUNT number of times before reaching steady state

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Read–only

DM_split_dmod

Function: Deformable Modeling, Deformable Model Management

Action: Splits the deformable model into finer elements and returns 0 for success or an error.

Prototype:

```
void DM_split_dmod (
    int& rtn_err,              // out: 0=success
                               // or negative err code
    DS_dmod* dmod,             // in: dmod to modify
    int domain_flag,           // in: 0=split_pts in
                               // orig_dmod_space,
                               // 1=split_pts in
                               // unit_space,
                               // 2=split_pts in
                               // internal_pfunc_space.
    int split_pt_count,        // in: number of split
                               // locations
    double* split_pts,         // i/o: locs at which to
                               // split domain
                               // sized:[ split_pt_count
                               // *dmod->Pfunc()->
                               // Domain_dim()]
                               // pts given in the
                               // unit_square ordered
                               // curves=[u0 u1 u2 ...]
                               // surfs =
                               // [u0 v0 u1 v1 ...]
    SDM_options* sdmo          // in:SDM_options pointer
        = NULL                 //
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description:

This function calls the DS_dmod–>Split_dmod() method to add element boundaries into existing elements. This is a recursive function and all child patches of deformable model that span the split points will be split as well.

No action is taken for each par–pos *u* and *v* value which is already a boundary in the deformable model's domain. In B–spline and NURB terms this means that this algorithm cannot be used to increase the multiplicity of an existing knot.

For B–spline and NURB shapes this is just the knot insertion algorithm. The split_pt_count tells how many split points will be added to the model. The split_pts array lists the locations for each split. For curves, split_pts is just a list of *u* coordinate values. For surfaces, split pts is a list of *uv* pairs. When the domain_flag is set to 1, each *u* or *uv* coordinate is given in the unit square and this function will map each point to the domain space of the deformable model. When the domain_flag is set to 0, each *u* and *uv* coordinate is given in the dmod's orig_dmod_space domain range. When the domain_flag is set to 1, each *u* and *v* coordinate is given in the dmod's pfunc's internal_pfunc_space.

This function may be used to make a finer deformable model out of a coarser one. Experience has shown that deformable models with regularly spaced finite elements works the best. A good four element curve would have element boundaries boundaries as [0.0 0.25 0.50 .75 1.00]. When inserting boundaries into surfaces the split point is used to add both a u_direction and v_direction into the B–spline basis functions. This acts to add an element boundary that runs throughout the entire length of the model. The rule of creating regular elements still applies so always add split points along the diagonal axis of the unit square. So to turn a one element model into a 3x3 element model split at the following points split_pts= [.333 .333 .667 .667].

Errors: DM_BAD_DOMAIN_PT_RANGE
The split points must be completely contained within the unit square.

DM_NULL_INPUT_PTR
The deformable model cannot be NULL on entry.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model

DM_update_cstrn_src_pts

Function: Deformable Modeling, DML Constraints

Action: Forces a curve or link constraint to update its internally stored data after the constraint's source points have been modified.

Prototype:

```
void DM_update_cstrn_src_pts (
    int& rtn_err,           // out: 0=success
    DS_dmod* dmod,          // or negative err code
    int tag,                // in: member of target
    SDM_options* sdmo       // in:SDM_options pointer
        = NULL              // note: sets active dmod
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "dshusk/dskernel/dmapi.hxx"
#include "dshusk/dskernel/dsdmod.hxx"
#include "dshusk/dskernel/sdm_options.hxx"
```

Description:

This function helps to enable curve tracking. It causes a curve constraint or link constraint to rebuild its constraint equations from the constraint's current source point definitions.

This function must be used for curve tracking. Create a curve or link constraint using a function to describe the target shape. This can be done by either building the constraint with a callback function, src_CW_func(), or with a combination of the DS_pfunc objects, src_W_pfunc, src_Wn_pfunc, and src_Wnn_pfunc. Modify the source shape. For example, the src_pfunc DS_pfunc objects can themselves be shaped as deformable curves. After the source shape is modified call this function to have the constraint update its internal equations so that the next call to DM_solve() will cause the constrained shape to deform to the new source shape.

This routine will set the constraint tight state on.

Refer to DM_add_crv_cstrn() and DM_add_link_cstrn().

The input arguments dmod and tag identify which constraint to update.

The basic flavor for using a constraint is:

– Create and apply the constraint to a deformable model (DM_add_pt_cstrn()), (DM_add_crv_cstrn()), (DM_add_link_cstrn()), (DM_add_area_cstrn()).
– Select which geometry properties are being constrained,(DM_set_cstrn_behavior()).
– For tracking point constraints, modify the geometry property values by moving the point constraint display points as described above,(DM_set_pt_xyz()).
– For tracking curve constraints and link constraints, use the src_W_pts, src_Wn_pts, and/or src_Wnn_pts, update their shapes, and ask this deformable model to track the change with (DM_update_cstrn_src_pts()).
– Call solve to see how the constraint modifications changed the deformable model shape, (DM_solve()).
– Optionally, for point constraints, the point constraint's domain position and domain directions (for surfaces) may be modified with (DM_set_pt_uv()) or (DM_set_cstrn_pttan_uv_dir()).
– Optionally, modify the constraint rendering and image_pt locations to optimize viewing and/or interactions, DM_set_tan_display_gain() DM_set_comb_graphics().

Errors: DM_NO_ROOT_DMOD
The root deformable model cannot be NULL on input.

DM_TAG_OBJECT_NOT_FOUND
The input tag does not identify any object in the model hierarchy.

DM_NOT_A_CRV_LINK_CSTRN
The input tag does not identify a crv_cstrn or link_cstrn constraint in the model hierarchy.

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: System routine

DM_xsect_dmod_by_image_line

Function: Deformable Modeling, DML Graphics

Action: Finds uv_params for a ray deformable model intersection and returns 0 for success or an error.

Prototype:

```
void DM_xsect_dmod_by_image_line (
    int& rtn_err,            // out: 0=success
                             // or negative err code
    DS_dmod* dmod,           // in: target dmod
    int walk_flag,           // in: specify how deep
                             // to go
                             // 0=dmod only,
                             // 1=dmod and offspring
                             // 2=dmod,siblings,and
                             // offspring
    double* p0,              // in: p0 of iline =
                             // p0 + u*(p1 – p0)
                             // sized:[image_dim]
    double* p1,              // in: p1 of iline =
                             // p0 + u*(p1 – p0)
                             // sized:[image_dim]
    double max_dist,         // in: width of ray in
                             // which to search for
                             // crvs
    double& image_line_u,    // out: image_line u
                             // param value of xsect
    int domain_flag,         // in: 0=rtn domain_pt in
                             // orig_dmod_space,
                             // 1=rtn domain_pt in
                             // unit_space,
                             // 2=rtn domain_pt in
                             // internal_pfunc_space.
    double* domain_pt,       // out: shape uv param
                             // value of xsect
                             // sized:[domain_dim]
    double* image_pt,        // out: image pt of xsect
                             // sized:[image_dim]
    SDM_options* sdmo        // in:SDM_options pointer
        = NULL               // note: sets active dmod
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”dshusk/dskernel/dmapi.hxx”
#include ”dshusk/dskernel/dsdmod.hxx”
#include ”dshusk/dskernel/sdm_options.hxx”
```

Description: Modifies domain_pt, image_pt, the active deformable model.

When the intersection is successful, this extension loads domain_pt and image_pt with nearest intersection point data, and sets the active deformable model to be the deformable model owner of the intersection, and return 0.

max_dist defines the distance between two curves which is considered a near–miss.

domain_pt and image_pt are loaded with the position and the domain_pt of the intersection. The return is –1 when no intersections are found.

When domain_flag is set to 1, domain_pt is returned in the unit range. When domain_flag is set to 0, domain_pt is returned in the dmod's original domain range. When domain_flag is set to 2, domain_pt is returned in the dmod's pfunc's internal domain range.

The input argument walk_flag controls the depth and breadth of the search for deformable models that intersect the input image line. A walk_flag value of 0 specifies that just the deformable model is modified, and a walk_flag value of 1 specifies that the deformable model and all its offspring are modified and the greatest descendant is returned. When walk_flag value is 2 the target deformable model, its siblings, and all their descendants are searched for an intersection.

Errors: None

Limitations: None

Library: dshusk

Filename: ds/dshusk/dskernel/dmapi.hxx

Effect: Changes model