Parasolid V34.1

Foreign Geometry User Manual

January 2022

Important Note

This Software and Related Documentation are proprietary to Siemens Industry Software Inc. © 2022 Siemens Industry Software Inc. All rights reserved

Francis House
112 Hills Road
Cambridge CB2 1PH
UK
Tel: +44 (0)1223 371555
email: parasolid.support.plm@siemens.com
Web: www.parasolid.com

SIEMENS

Trademarks

Siemens and the Siemens logo are registered trademarks of Siemens AG.

Parasolid is a registered trademark of Siemens Industry Software Inc.

Convergent Modeling is a trademark of Siemens Industry Software Inc.

All other trademarks are the property of their respective owners. See “Third Party Trademarks” in the HTML documentation.

Table of Contents

A

Introduction and Summary

1

1.1 Document overview

This document provides a complete guide to enable the user to implement in-house curves and surfaces for modeling with within Parasolid. Parasolid terms such externally defined geometry as **Foreign Geometry** (FG) or simply **foreign**. The contents of the chapters and appendices are as follows:

- Chapter 1 provides an overview of this document and a summary of how the user can implement Foreign Geometry.
- Chapter 2 gives complete details for implementing a Foreign Geometry system and linking it into Parasolid.
- Chapter 3 provides details of modeling using Foreign Geometry, of interface and KID functions, and of modeling operations not yet supported for Foreign Geometry.

The appendices are as follows:

- Appendix A provides details of geometric restrictions required by Parasolid of FG.
- Appendix B gives details of the FG module interface functions.

Note: You are advised to consult with Parasolid Support before you use Foreign Geometry as there may be a better solution for your modelling needs.

1.2 Summary: implementing and modeling with foreign geometry

1.2.1 Foreign geometry

Parasolid terms externally defined curves and surfaces as Foreign Geometry (FG) or foreign. FG functionality allows Parasolid to access user defined surfaces and curves via the FG module interface. Consequently, the users are able to use in-house surfaces and curves for Parasolid modeling alongside the standard Parasolid curves and surface types.

1.2.2 FG curve and surface definitions

FG objects (i.e. curves and surfaces) are identified by a key. In addition, the user is able to provide numerical data (reals and integers) when an FG object is initialized, enabling a single evaluator to define a family of geometries. Parasolid requires that the foreign curve or surface is defined parametrically and is able to supply (via the FG module interface) position and derivatives at specific parameter values.

Specifically, a curve evaluator must be parametrised over a finite interval $t_0 \leq t \leq t_1$ and be able to supply position, first and second derivatives over this interval.

A surface evaluator must be parametrised over a finite region $u_0 \leq u \leq u_1$ and $v_0 \leq v \leq v_1$ and must be able to supply position, first and all second derivatives over this region.

Both curves and surfaces are subject to geometric restrictions as detailed in Appendix A.

1.2.3 The FG module

The user must write their own FG module, based upon the supplied example and on the details given in Chapter 2. This module is where requests from Parasolid for details of externally evaluated curves and surfaces are resolved. Once this module has been linked into Parasolid the user may model with their own in-house curves and surfaces using the standard Parasolid interface functions.

FG module functions

There are six functions exported by the FG module. The user must provide definitions for each of these functions, and use PK_SESSION_register_frustrum to register them with Parasolid, along with the standard frustrum functions. Parasolid obtains all details of externally evaluated geometry via calls to these functions. The functions are:

- FGCRCU, FGCRSUfor curve/surface evaluator initialization;
- FGPRCU, FGPRSUfor defining curve/surface evaluator parameter ranges;
- FGEVCU, FGEVSUfor evaluating curves/surfaces.

The creation functions, FGCRCU and FGCRSU receive a **key** (character string) identifying the evaluator function along with numerical data required for evaluator initialization. In addition a block of space, **fg_data**, is supplied, in which sufficient details for future identification of the evaluator are stored. These received arguments are passed directly through to the FG module by the functions, PK_FCURVE_create and PK_FSURF_create. The functions FGCRCU and FGCRSU may be called as a result of the user creating new foreign curves or surfaces using the functions PK_FCURVE_create and PK_FSURF_create, or indirectly to re-initialize any foreign geometry present on an archived part that is loaded into Parasolid using PK_PART_receive.

The parameter properties functions, FGPRCU, FGPRSU are called once the evaluator has been successfully initialized as above. They are passed the fg_data block along with the numerical data PK_FCURVE_create, PK_FSURF_create and supply the parameter range of the evaluator concerned. A simple status return from the parameter properties functions ensures Parasolid uses default parameter ranges of the unit interval (for curves) or unit square (for surfaces).

The evaluation functions, FGEVCU, FGEVSU are called by Parasolid to request position and derivatives of Foreign Geometry objects. These functions are passed the fg_data block to identify the evaluator function along with the further numerical data supplied through the interface by PK_FCURVE_create, PK_FSURF_create. They are also passed details of the specific position and derivatives required and an array in which these evaluations are to be placed.

Linking the FG module

The FG module must be linked into Parasolid to allow the user access to their curves and surfaces defined therein. The details of how to do this are machine specific but the principle is straightforward.

Command scripts are provided with the Parasolid release enabling the user to link the supplied example FG module together with Parasolid to form a customized version of KID.

1.2.4 Modeling with foreign geometry

Once the FG module has been successfully implemented and connected in to Parasolid, the user is able to proceed with solid modeling using FG curves and surfaces. Modeling requests are submitted, as usual, via the interface. The user sees Foreign Geometry objects alongside the standard Parasolid geometry types, and is able to use the same modeling commands upon them.

There are in addition, four functions specific to FG. These are:

- PK_FCURVE_create, PK_FSURF_create for creation of FG curves/surfaces;
- PK_FCURVE_ask, PK_FSURF_ask for output of FG curves/surfaces.

The exact specification of these functions is given in the PK Interface Programming Reference Manual and full details of their use is given in Chapter 3 of this document.

The creation functions, PK_FCURVE_create and PK_FSURF_create, are called when an evaluator is to be initialized for modeling with. Supplied arguments are the evaluator key and the length of this key, together with extra numerical data required by the evaluator, and the length of the fg_data block which is used to store information to identify the evaluator in future. These arguments are passed on to the FG module in calls to FGCRCU, FGCRSU (see previous section, “FG module functions”).

The FG output functions, PK_FCURVE_ask, PK_FSURF_create receive the curve, surface respectively and an array into which the returned key string is written. Details of the curve, surface are returned: i.e. the key, key length, lists of numerical data used to initialize the curve/surface, along with the curve/surface sense and any transformation applied.

Implementing an FG System

2

2.1 Introduction

This chapter explains the process whereby the user creates FG evaluators and links them into Parasolid. An overall description of this procedure is given in Chapter 1, “Introduction and Summary”.

2.2 Writing the FG module

2.2.1 An example evaluator

Let’s start by taking an example of a surface for which we want to write an evaluator:

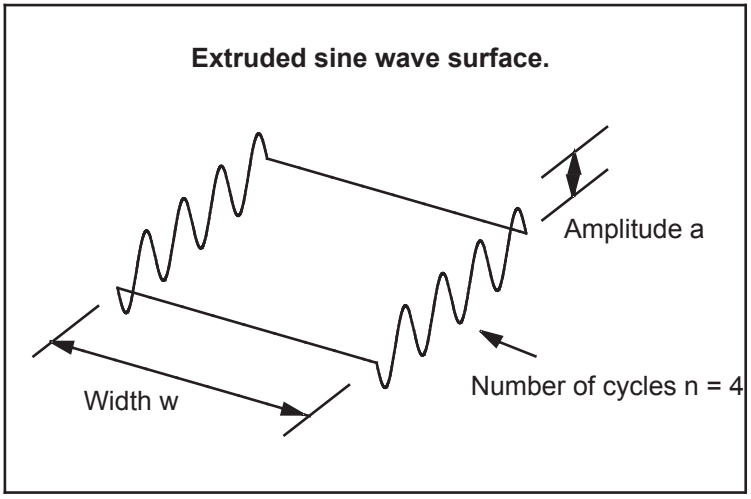
The surface function:

```
P( u, v ) = ( u, v, a sin(2 pi u) )
```

generates an extruded sine wave surface. If we further allow the parameter ranges to be:

```
U in the range [ 0, n ]
V in the range [ 0, w ]
```

then we generate the surface shown in the diagram below:



To implement an evaluator function for this surface we have to make some basic design decisions :

- We supply the parameters a, n and w as real numbers submitted through the interface
- We identify the generic surface by the key `EDS/corrugated` (the key should give an indication of the source of the evaluator – in this case Electronic Data Systems)
- The code for the evaluator is linked to Parasolid (so that we can access the evaluator function by its address in memory)

The tasks we have to complete in order to implement this surface are :

- Enable the initialization of a corrugated surface – (through function FGCRSU)
- Supply its parameter limits to Parasolid – (through function FGPRSU)
- Enable evaluations of the surface – (through function FGEVSU)

2.2.2 Evaluator initialization

Our surface is requested by a call to function PK_FCURVE_create. The arguments to this function should be:

- Keylen is 14 (i.e. the number of characters in the evaluator key string),
- Key is `EDS/corrugated`
- nspace is 1 (the minimum value)
- nints is 0
- ivals is a null pointer
- nreals is 3
- rvals is a pointer to a real array containing (a, n, w)

This call generates a call to the function FGCRSU with essentially the same arguments, except that a double array of length 1 has been allocated by Parasolid for initialization by FGCRSU.

In FGCRSU a section of code is added to recognize our corrugated surface and enable evaluations of it. The code which achieves this is :

```
if( keycmpu( "SDL/corrugated", key, keylen ))
{
if (( *kir == 3 ) &&  /*corrugated evaluator requires 3 reals*/
  ( ki_reals[1] > 0.0 ) &&   /*of which last two are positive*/
  ( ki_reals[2] > 0.0 ))
             *(surface_evaluator *) fg_data = corrugated;
else *ifail = FGRERR;        /*...otherwise real data error*/
}
```

Note: 1. The function FGCRCU performs the initialization of curve evaluators, receiving the same arguments and performing the same tasks.

Note: 2. The function `keycmpu` does a string comparison between the supplied key and the valid key for our surface. This function is part of the sample code. A straightforward C string comparison does not work here since, for FORTRAN compatibility reasons, the key has no standard C null terminator.

Note: 3. It is at this point in the code that any validation of data needs to be implemented – a successful call to FGCRSU should indicate that valid evaluations are available. In this case three real data items must be available of which the final two (the number of cycles, n, and the width of the surface , w) need to be positive. If the real data supplied through the interface does not fit this form, then the ifail return code is set to FGRERR to indicate unsatisfactory real data.

Note: 4. The type definition **surface_evaluator** is: **typedef void (* surface_evaluator)()**

Once the data has been validated, the address of the evaluator function is stored in the fg_data space that was requested. The evaluator is called by a call to this address. All future references to this evaluator (by functions FGPRSU, FGEVSU) are made by passing the fg_data array, as initialized here – the key is no longer supplied.

Note: 5. If an evaluator is written which uses secure data stored in the fg_data array, FGCRSU needs to implement the loading of this data. Evaluators of this type should have a key which itself contains the filename/database entry where the data can be found. For example, a key `EDS/Gordon/Example1` could indicate that the Gordon surface evaluator should be used with data loaded from file `Example1`.

Note: 6. FGCRSU is only called once per FG surface. The structure of our example could get unwieldy if many tens or hundreds of surfaces were implemented. In this case some means of avoiding a lot of string comparisons would be useful, but is not critical.

Note: 7. FGCRSU is called whenever a foreign surface is to be initialized for use in Parasolid. This can happen directly when a user calls up a new foreign surface using PK_FSURF_create, or indirectly when a user loads an archived part into Parasolid using PK_PART_receive, when FGCRSU is called once for every foreign surface present in the part.

2.2.3 Parameter properties function

Once a surface is successfully initialized by a call to FGCRSU, a call is made to function FGPRSU so that Parasolid can be informed of the parametric properties of the surface. In our example code the relevant code is :

```
{
  surface_evaluator eval;
  eval = *(surface_evaluator *)fg_data;
  *ifail = FGPROP;              /* initialize properties to
defaults*/
  if ( eval == corrugated )
  {                             /* establish 'corrugated'
properties*/
    range[0] = 0.0;             /* low u value    */
    range[1] = ki_reals[1];     /* high u value   */
    range[2] = 0.0;             /* low v value    */
    range[3] = ki_reals[2];     /* high v value   */
    period[0] = FGPRBD;         /* bounded u parametrisation */
    period[1] = FGPRBD;         /* bounded v parametrisation */
    *ifail = FGOPOK;
  }......
```

Note: 1. The parameter ranges [0,n] and [0,w] are returned to Parasolid.

Note: 2. The surface key is no longer available, so our surface is recognized by the address stored in fg_data.

Note: 3. The `period` arguments must be set as shown, to FGPRBD – 'not periodic'. At the current Parasolid version, periodic surfaces are not allowed so this argument is for future extensions.

Note: 4. Default parameter properties may be assumed by setting the return ifail code to FGPROP. These properties are u and v ranges both [0, 1] and periodicity flags both FGPRBD.

Note: 5. The corresponding function for curves, FGPRCU, behaves in a very similar way. The only differences are that the range array only has two entries, range[0] and range[1], for low and high values of the curve's parametrisation, and that only period[0] needs to be initialized to FGPRBD to indicate that the t parametrisation is non-periodic. Again, default properties may be used by setting the return code to FGPROP.

2.2.4 The evaluation function

Enabling evaluations of the surface

Evaluations of surfaces are routed through function FGEVSU. In our example this is implemented as :

```
eval = *(surface_evaluator *) fg_data;
                            /* initialize evaluator function*/
*ifail = FGOPOK;                        /* initialize ifail */
(*eval)( ki_ints, ki_reals, fg_data, u, v,
                                        /* & call evaluator */
     nu, nv, triang, results, ifail ); }
```

Note: 1. We set ifail to 'success' so that evaluators only have to set other outcomes.

Note: 2. FGEVSU then simply routes the request directly to our evaluator function. Clearly this mechanism depends upon evaluators being written with the same arguments as FGEVSU. In existing systems this may well not be possible. In these cases the call to the evaluator needs to be made and the results processed, by this function, into the form which Parasolid is expecting.

Note: 3. The performance of this linkage to the external evaluators is quite critical to the performance of FG within Parasolid – many thousands of calls are made to the evaluator in a modeling session. Therefore every effort should be made to optimize this function. In FORTRAN, for example, storage of an integer in fg_data followed by a computed GOTO here works.

The corrugated evaluator function.

The evaluator call supplies arguments as follows :

- Pointers to the three data arrays which the evaluator can use (ki_ints, ki_reals, fg_data)
- Parameter position (u, v). Parasolid ensures that only parameter values within the parameter range of the surface are requested, so no tests need to be done for this.
- Derivative specification (nu, nv, triang) which refers to the number of u derivatives requested, the number of v derivatives requested, and the triangular flag.
- The results array is supplied for the evaluator to return its results. Each position or derivative is stored in three elements of this array.

Let's look at a section of the sample `corrugated` evaluator code:

```
if ( su_dP_du( &n, *nu, *nv, *triang ))
{
  /*    compute : Pu  */
  results[n++] = 1.0;
  results[n++] = 0.0;
  results[n]    = two_pi * A * cos(*u * two_pi);
}
```

and later on after all evaluations are complete...

```
*ifail = su_check_params( *ifail, *nu, *nv, *triang );
```

Note: 1. The decision as to whether a particular derivative is required, and where it should be returned in the results array, is not entirely straightforward. Therefore we have provided a set of utility functions for this purpose. In the example above, su_dP_du returns TRUE if the dP/du derivative is requested and n points to the correct position in the results array. More details on these issues are given below.

Note: 2. The function su_check_params sets ifail to FGEVIN if the evaluation requested could not be completely fulfilled. It assumes that evaluators are written to supply up to and including all second derivatives (the minimum requirement). Cases where Parasolid asks for more than the evaluator can supply may occur, for example, if a surface is created which is the offset of an FG surface. In these cases FGEVIN alerts Parasolid to the need to approximate the missing derivatives. Parasolid recognizes missing derivatives by checking that no assignment has been made to the relevant entries in the results array, so it is important NOT to set elements of the results array that have not been successfully calculated.

Note: 3. The corresponding function for curves, FGEVCU, has been similarly coded but is more straightforward in that the derivative request is a single argument, nderiv, for the number of t derivatives requested as opposed to the surface derivative request (nu, nv, triang) for the numbers of u and v derivatives requested along with the triangular flag.

2.2.5 Derivative specifications and output

The results of an evaluation are returned in the `results` array passed down to the evaluator. The specification of the numbers of derivatives required is supplied by means of the `nu`, `nv` and `triang` arguments. The meanings of these is as follows:

- The `results` array should be considered as an array of vectors each of which is stored in 3 successive locations in the array. For example, if 6 results are required then the `results` array is 18 doubles long.
- The `nu`, `nv` and `triang` specify the derivatives required, and their positions in the `results` array as follows:
  - If `triang` is FGEVSQ. In this case a rectangular array of derivatives is required. `nu` specifies the highest derivative of form dnP/dun that is required. `nv` specifies the highest derivative of form dnP/dvn that is required. Other entries are according to the pattern '*each successive column is a higher derivative in u and each successive row is a higher derivative in v*'. Thus, for example, if nu=2 , nv=1 and triang=0 then the results array looks as follows:

```
P dP/du d2P/du2
dP/dv d2P/dudv d3P/du2dv
```

  - These positions/derivatives are then stored in the `results` array in row order. For example, counting from position 0 (in Fortran add 1 to these positions):

    dP/du is stored in elements 3-5 of the `results` array.

    d2p/dudv is stored in elements 12-14 of the `results` array.

  - Finally, if a derivative is NOT available then the corresponding array elements should be returned unmodified. For example, if your evaluator cannot return d3p/

du2dv in the above specification you should not modify elements 15-17 of results and return `ifail' as FGEVIN (Evaluation incomplete)

- The `triang` argument may only take the value FGEVTR when nu = nv. In this case the array pattern is triangular and returned in row order. For example, if nu=nv=2 and triang=1 the pattern of the results array is as follows:

```
P dP/du 2P/du2
dP/dv d2P/dudv
d2P/dv2
```

- Thus, for example:

  d2P/du2 is stored in elements 6-8 of the `results` array.

  d2P/dv2 is stored in elements 15-17 of the `results` array.

2.3 FG module design issues

2.3.1 General design

In Parasolid the style of implementation used for FG is as shown in the supplied example source code and detailed in this chapter. We have found it to be a fast and straightforward interface to use. We have an advantage here in that we are generally writing FG evaluators from scratch. We have designed the FG interface to be as compatible as possible with other FG systems that we have encountered as well as being already specified in a form that allows us to increase the range of curve and surface types that are implementable as FG.

2.3.2 Performance issues

Parasolid does not replace users' evaluators with internal approximations: it must always request spatial information regarding external curves and surfaces by calling FGEVCU and FGEVSU respectively in order to model exactly. Therefore it is essential that the user codes these functions as efficiently as possible since any delay in returning information to Parasolid is directly reflected in the speed of modeling using FG.

2.3.3 Data usage issues

Data available to the FG developer

A simple evaluator may be designed to generate a single specific curve or surface. In this case no information other than parameter positions is required by the evaluator in order for it to be able to supply positions and derivatives.

However, we imagine that developers will more typically choose to write general purpose evaluators capable of generating whole families of curves or surfaces. In these cases the evaluators will require additional data to specify the particular example required.

Parasolid provides two distinct means to supply this sort of data to evaluators as arguments to the function which specifies the required curve or surface or from data supplied by the FG module when the curve or surface is initialized.

Data passed to the interface

The interface allows the user to supply an array of real numbers and/or an array of integers to the evaluator. In most cases this is the preferred method of supplying data:

- This data is stored in transmit files of Parasolid bodies, so that such files are sufficient to exchange models between sites with the required evaluators linked to Parasolid.
- This data must be in the form of reals or integers – as attempts to cast character strings into elements of the real array creates difficulties in generating transmit files.

Data passed to the FG module

The interface additionally allows the user to specify an area of memory which is filled by the FG module call which requests that the FG item specified by the interface call be initialized. This data :

- Is not stored on any file generated by Parasolid. It therefore provides a means by which users can implement evaluators where security considerations require that no associated data be accessible from transmit files or snapshots.
- Is simply stored by Parasolid and made available to the evaluator. Therefore any form of data, such as pointers or character strings, can be cast into this area.
- Requires that the source of this data, such as files or databases, be available on all sites where this form of evaluator is to be used.

There is one specific use of such data which is inherent in the design of the FG interface. All evaluation requests leave Parasolid as calls to functions FGEVCU or FGEVSU. These functions therefore need to be able to route these requests to the appropriate evaluator function.

It would be possible to design these functions to switch to evaluators based upon the key supplied to the interface. However, performance considerations indicate that this is a very inefficient mechanism, particularly if many evaluators are linked into Parasolid. The interface therefore requires that at least space for one double is requested. This first element of the Frustrum array is intended to be used to implement this switch between evaluators.

In the example discussed below the address of the evaluator function is stored. This is an efficient method of implementing this switch in C but other methods such as a stored integer used in a Fortran computed GOTO statement are equally valid.

2.3.4 Key format for evaluator selection

There is no format which is enforced for keys by Parasolid. However, we believe that the opportunities that the FG facilities create for extending the geometrical repertoire of Parasolid may find widespread applications. We need to provide support facilities for users of such geometry that meet Parasolid's high customer support standards. In order to do this we need to be able to identify the originators of FG evaluators. We propose a standard for keys as follows:

key = <company_identifier> / <evaluator_name> / <data_source>

So for example :

- Only evaluators which use the `fg_data` route for input need to use the <data_source> component
- our corrugated surface has key `EDS/corrugated`
- A Coons surface evaluator written by company ABC which uses secure data from a file `Example_data` would have key `ABC/Coons/Example_data`

2.3.5 Deletion of evaluators

The Foreign Geometry interface comprises six functions:

- FGCRSU, FGCRCU to initialize curves and surfaces
- FGPRCU, FGPRSU to return their parametric properties
- FGEVCU, FGEVSU to perform evaluations

Missing from this list are functions to 'release' curves and surfaces that Parasolid has no further need of. The reason for this omission is that the decision as to when to release an evaluator is a complicated one since it depends on such things as copying geometry, the existence of dependent geometry and the Parasolid rollback mechanism. To attempt this would require much of the rollback mechanism to be implemented inside the Frustrum and would force all customers to rewrite their Frustrums, whether using foreign geometry or not. UGS reserves the right, however, to implement this in future releases, as this change in functionality is more generally useful.

2.4 Linking the FG module into Parasolid

The FG module must be linked into Parasolid to allow the user access to their curves and surfaces defined therein. The details of how to do this are machine specific but the principle is straightforward.

Command scripts are provided with the Parasolid release enabling the user to link the supplied example FG module together with Parasolid to form a customized version of KID.

Modeling using Foreign Geometry

3

3.1 Introduction

This chapter explains how to model using Foreign Geometry. This chapter forms a supplement to the Parasolid Programming Reference Manuals since all modeling operations are performed using Parasolid functions. Sections of this chapter are:

- Parasolid functions specific to Foreign Geometry,
- Modeling,

3.2 Functions specific to foreign geometry

Four functions exist at the interface specifically for dealing with foreign geometry. The headers for these functions are contained in the PK Interface Programming Reference Manual.

3.2.1 PK_FSURF_create – Create foreign geometry surface

Receives:
- length of foreign geometry key
- foreign geometry key
- user defined double array length available to evaluator
- number of integer values supplied
- integer values
- number of real values supplied
- real values

Returns:
- tag of foreign geometry surface
- failure indicator

Example using "EDS/corrugated" surface. The evaluator requires three real values to uniquely define the surface. The first is the amplitude of the waves on the surface, the second is the number of wave cycles in the surface and the third is the width of the surface.

Receives:
- 4
- "EDS/corrugated"
- 1
- 0
- null
- 3
- (1.0 2.0 3.0)

Returns:
- tag of "EDS/corrugated" surface
- 0 (if no errors)

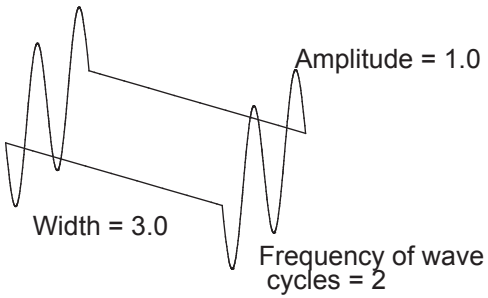


Figure 3–1 "EDS/corrugated" foreign geometry surface

3.2.2 PK_FSURF_ask – Output foreign geometry surface data

Receives:
- tag of foreign geometry surface
- space available for key string

Returns:
- key
- keylength
- list of integer values (if present)
- list of real values (if present)
- surface sense
- surface transformation (if present)
- failure indicator

Example using "EDS/corrugated" surface.

Receives:
- tag of "EDS/corrugated" surface
- 14

Returns:
- "EDS/corrugated"
- 14
- null
- (1.0 2.0 3.0)
- true
- null
- 0 (if no errors)

3.2.3 PK_FCURVE_create – Create foreign geometry curve

Similar arguments and procedure exist for creating a foreign geometry curve.

3.2.4 PK_FCURVE_ask – Output foreign geometry curve data

Similar arguments and procedure exist for outputting foreign geometry curve data.

3.3 Modeling

Once successfully created Foreign Geometry entities may be employed in modeling operations in the same way as any standard Parasolid geometry.

Following are details of three modeling sessions employing Foreign Geometry. In these example sessions the evaluators used are defined in the supplied example FG module.

3.3.1 Creation of EDS/corrugated surface followed by sheet boolean

The following example illustrates how a surface may be created from an external evaluator with key name “EDS/corrugated”. The evaluator requires three items of real data to uniquely define the surface. A sheet body is made from the surface which is copied and translated such that the two sheets may be united as one using boolean unite operation. The function PK_FSURF_ask is also used to output details about the surface.

Creation of a sheet body using an external evaluator “EDS/corrugated”.

Function call:

- PK_FSURF_create – *creates a surface using the external evaluator “EDS/corrugated”*
- PK_SURF_make_sheet_body – *creates a sheet body from the geometry*
- PK_FSURF_ask – *output details of foreign geometry surface*

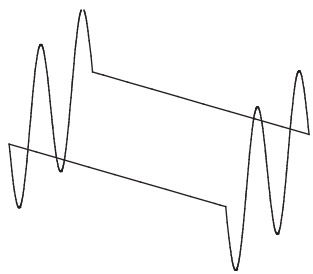


Figure 3–2 Sheet body with underlying foreign geometry

Copying, transforming and uniting sheet bodies

Function calls

- PK_TRANSF_create_translation – *creates a translation transformation*
- PK_ENTITY_copy_2 – *creates a copy of the sheet body*
- PK_BODY_transform_2 – *applies the transformation to one sheet body*
- PK_BODY_boolean_2 – *boolean unite operation*

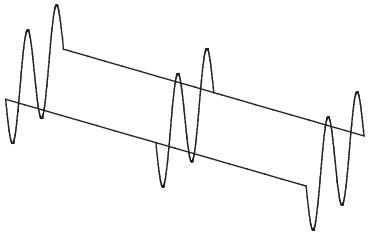


Figure 3–3 Result of uniting two "EDS/corrugated" sheet bodies

3.3.2 Creation of an “EDS/incline” foreign surface followed by boolean operations

The second example uses a foreign geometry surface “EDS/incline”, requiring no real or integer data in its construction, in sheet body unite operations. The resulting body has a portion removed by boolean subtraction with a cylindrical solid.

Creation of sheet body using an external evaluator EDS/incline

Function calls

- PK_FSURF_create – *creates a surface using the external evaluator "EDS/incline"*
- PK_SURF_make_sheet_body – *creates a sheet body from the “EDS/incline” surface*

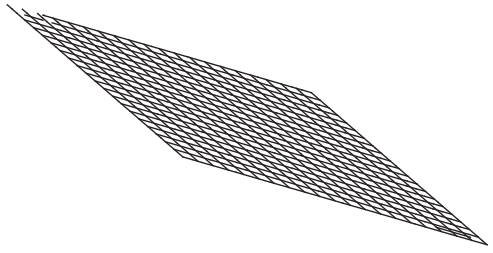


Figure 3–4 View of foreign geometry surface “EDS/incline” (with parametric hatching)

Uniting sheet bodies

Function calls

- PK_PLANE_create – *creates a planar surface*
- PK_SURF_make_sheet_body – *create sheet body from planar geometry*

  Repeat the above for an additional adjacent planar surface with parameter ranges u[1.4 2], v[0 1]

- PK_BODY_boolean_2 – *unite foreign geometry sheet and planar sheet*

  Repeat the above PK_BODY_boolean_2 to attach the second planar sheet body to the current body

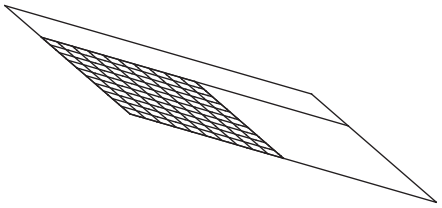


Figure 3–5 View of resultant of uniting sheet bodies (foreign geometry parametrically hatched)

Subtract a cylindrical portion from the current body

Function calls
- PK_CYL_make_solid_body – *creates the cylindrical solid*
- PK_BODY_boolean_2 – *subtract a cylindrical portion from the current body*

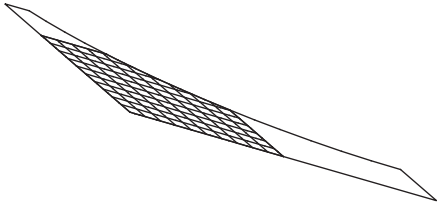


Figure 3–6 Fig. 3-6 Result of cylindrical solid subtraction (foreign geometry parametrically hatched)

3.3.3 Creation of "EDS/franke" sheet body used in Boolean Operations

The final example uses the foreign geometry with key "EDS/franke" to cut a solid block. One half of the block is then used in a boolean subtract operation with a cylindrical solid.

Creation of Sheet Body

Function calls
- PK_FSURF_create – *create foreign geometry surface using "EDS/franke" evaluator*
- PK_SURF_make_sheet_body – *create sheet body from "EDS/franke" surface*

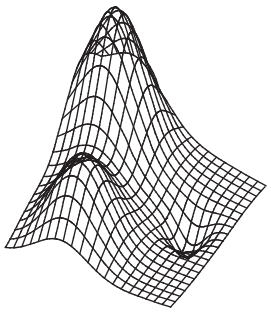


Figure 3–7 Sheet body of “EDS/franke” surface (with parametric hatching)

Use “EDS/franke” sheet body to “cut” solid block

Function calls
- PK_BODY_create_solid_block – *create a solid block*
- PK_BODY_boolean_2 – *cut block with foreign geometry*

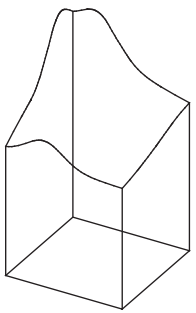


Figure 3–8 One of the bodies resulting from the block/”EDS/franke” boolean

Subtract a cylindrical solid from current body

Function calls
- PK_CYL_make_solid_body – *create cylindrical solid*
- PK_BODY_boolean_2 – *subtract cylinder from block*

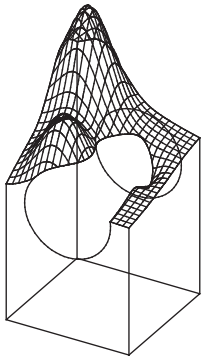


Figure 3–9 Current body with cylindrical subtraction (foreign geometry parametrically hatched)

Geometric Restrictions A

The following assumptions are imposed on Foreign Geometry at the current release of Parasolid.

A.1 Parameter range and derivatives

- FG curves are assumed to be parametrized over a finite interval $[t_0, t_1]$
  (the default interval is [0, 1]).
- FG surfaces are assumed to be parametrized over a finite range $[u_0, u_1]$ $[v_0, v_1]$
  (the default range is [0, 1] [0, 1]).
- FG evaluators must return evaluations up to and including second derivatives
  Thus for a curve evaluator C(t) where t in $[t_0, t_1]$ we have:
  $C, dC/dt, d^2C/dt^2 : [t_0, t_1] \; ! \; R^3$
  and for a surface evaluator S(u, v) where (u, v) in $[u_0, u_1]$ $[v_0, v_1]$ we have:
  $S, dS/du, dS/dv, d^2S/duduv, d^2S/du^2, d^2S/dv^2 : [u_0, u_1] [v_0, v_1] \; ! \; R^3$

A.2 Continuity

A curve is said to be G1 continuous if it has a continuous unit tangent vector, and to be C1 continuous if position and first derivative are both continuous.

A surface is said to be G1 continuous if it has a continuous unit normal vector, and to be C1 continuous if position and first derivatives are all continuous. C1 continuity implies G1 continuity and is the stronger condition.

- FG curves are required to be G1 continuous.
- FG surfaces are required to be G1 continuous. In addition, Parasolid requires that constant parameter lines on the surface are G1 continuous.

A.3 Geometric and parametric properties

- FG curves and surfaces are required to not have periodic parametrization.
- FG curves and surfaces may not be closed.
- No degeneracies or singularities on FG curves,
- The only allowed singularity on an FG surface is the case when, at a corner (i.e. parameter positions (u_0, v_0), (u_0, v_1), (u_1, v_0), (u_1, v_1)) first partial derivatives are

parallel. Therefore, local to that corner the surface degenerates to a line and the surface normal is not well defined.

- Edges of an FG surface are allowed to degenerate subject to:
  - the whole edge must degenerate to a point,
  - adjacent edges on an FG surface may not be degenerate (so that there are at most two degenerate edges).

FG Module Interface Functions

B

B.1 Introduction

This appendix provides details of the FG module interface functions designed for the management and evaluation of FG curves and surfaces. These functions are:

- FGCRCU (page 29), FGCRSU (page 30) – For initialization of FG curve and surface evaluators
- FGEVCU (page 30), FGEVSU (page 31) – For evaluation of FG curves and surfaces
- FGPRCU (page 32), FGPRSU (page 33) – For parametric properties of FG curves and surfaces

B.2 FGCRCU – Initialize a foreign curve

```
void        FGCRCU
(
                          /* received arguments        */
const char  *key,         /* Curve key                 */
int         *keylen,      /* Curve key length          */
int         *n_kii,       /* Number of integers passed */
int         ki_ints[],    /* KI integer array          */
int         *n_kir,       /* Number of reals passed    */
double      ki_reals[],   /* KI real array             */
int         *n_data,      /* FG data length            */
                          /* returned arguments        */
double      fg_data[],    /* FG data array             */
int         *ifail        /* Failure indicator         */
)
```

Specific errors

```
FGGEOM  Curve evaluator not available
FGDATA  Curve evaluator data not found
FGFILE  Curve evaluator data allocation fault
FGIERR  Curve evaluator integer data error
FGRERR  Curve evaluator real data error
(FGOPOK Operation successful)
```

This function is called when an FG curve is requested by the KI function CRFGCU. The purpose of this function is to initialize any FG data that is required by the evaluator and to check that the real and integer data items meet the requirements of the requested evaluator.

A successful call to this function indicates to Parasolid that evaluations of this curve may now be requested.

B.3 FGCRSU – Initialize a foreign surface

```
void        FGCRSU
(
                          /* received arguments        */
const char *key,          /* Surface key               */
int        *keylen,       /* Surface key length        */
int        *n_kii,        /* Number of integers passed */
int         ki_ints[],    /* KI integer array          */
int        *n_kir,        /* Number of reals passed    */
double      ki_reals[],   /* KI real array             */
int        *n_data,       /* FG data length            */
                          /* returned arguments        */
double      fg_data[],    /* FG data array             */
int        *ifail         /* Failure indicator         */
)
```

Specific errors

```
FGGEOM  Surface evaluator not available
FGDATA  Surface evaluator data not found
FGFILE  Surface evaluator data allocation fault
FGIERR  Surface evaluator integer data error
FGRERR  Surface evaluator real data error
(FGOPOK Operation successful)
```

This function is called when an FG surface is requested by the KI function CRFGSU. The purpose of this function is to initialize any FG data that is required by the evaluator. A successful call to this function indicates to Parasolid that evaluations of this surface may now be requested.

B.4 FGEVCU – Evaluate a foreign curve

```
void   FGEVCU
(
                     /* received arguments           */
int    ki_ints[],    /* KI integer array             */
double ki_reals[],   /* KI real array                */
double fg_data[],    /* FG data array                */
double *t,           /* Parameter value              */
int    *nderiv,      /* No. of derivatives requested */
```

```
                    /* returned arguments          */
double results[],   /* Evaluation results          */
int    *ifail       /* Failure indicator           */
)
```

Specific errors

```
FGEVIN  Evaluation incomplete (not all derivatives requested
        may be returned)
FGOPFA  Evaluation failed
(FGOPOK Operation successful)
```

This function is called when an evaluation of a foreign curve is required by Parasolid. The function should be capable of supplying position and up to second derivatives for a foreign curve which has previously been initialized by a call to FGCRCU.

The argument fg_data identifies the curve evaluator. The arguments ki_ints and ki_reals give the numerical data passed in through the KI when the curve was initialized.

B.5 FGEVSU – Evaluate a foreign surface

```
void   FGEVSU
(
                    /* received arguments          */
int    ki_ints[],   /* KI integer array            */
double ki_reals[],  /* KI real array               */
double fg_data[],   /* FG data array               */
double *u,          /* Parameter value             */
double *v,          /* Parameter value             */
int    *nu,         /* No. of u derivatives requested */
int    *nv,         /* No. of v derivatives requested */
int    *triang,     /* Triangular array flag       */
                    /* returned arguments          */
double results[],   /* Evaluation results          */
int    *ifail       /* Failure indicator           */
)
```

Specific errors

```
FGEVIN  Evaluation incomplete (not all derivatives requested
        may be returned)
FGOPFA  Evaluation failed
(FGOPOK Operation successful)
```

This function is called when an evaluation of a foreign surface is required by Parasolid. The function should be capable of supplying position and up to second derivatives for a foreign surface which has previously been initialized by a call to FGCRSU.

The argument fg_data identifies the surface evaluator. The arguments ki_ints and ki_reals give the numerical data passed in through the KI at surface initialization.

The triangular flag, triang, may take the value FGEVSQ (indicating a request for a rectangular array of derivatives) or FGEVTR (indicating a triangular array of derivatives request). The arguments nu, nv specify the number of u an v derivatives requested.

B.6 FGPRCU – Return foreign curve parametrization properties

```
void   FGPRCU
(
                    /* received arguments                          */
int    ki_ints[],   /* KI integer array                            */
double ki_reals[],  /* KI real array                               */
double fg_data[],   /* FG data array                               */
                    /* returned arguments                          */
double range[2],    /* Parameter range                             */
int    *period,     /* Periodicity flag:                           */
                       FGPRBD => bounded (non- periodic)
                       FGPRPE => periodic (not currently allowed)
int    *ifail       /* Failure indicator                           */
)
```

Specific errors

```
FGOPFA  Operation failed
FGPROP  Use default properties
(FGOPOK Operation successful)
```

This function is called, after FGCRCU has initialized the curve, to indicate to Parasolid some of the parametric properties of the curve. The function can specify the parameter range of the curve and indicate whether it is to be treated as periodic. If the ifail value is set to FGPROP then Parasolid uses default properties for the curve evaluator: there is no need to set the range and periodicity flag.

Default properties are parameter range [0, 1] and periodicity flag FGPRBD (i.e. non-periodic parametrization).

If the ifail value is set to FGOPOK then this function must set the parameter range and periodicity flag:

- range[0] = lowest parameter value
- range[1] = highest parameter value
- *period = FGPRBD (must have non-periodic parametrization)

Note: For the current Parasolid version closed and periodic curves are NOT permitted. This function includes the periodic argument for upwards compatibility with future planned enhancements to FG capability.

B.7 FGPRSU – Return foreign surface parametrization properties

```
void    FGPRSU
(
                    /* received arguments                          */
int     ki_ints[],  /* KI integer array                            */
double  ki_reals[], /* KI real array                               */
double  fg_data[],  /* FG data array                               */
                    /* returned arguments                          */
double  range[4],   /* Parameter ranges                            */
int     period[2],  /* Periodicity flags - may take FGPRBD,        */
                       FGPRPE for bounded and periodic
                       parametrizations
int     *ifail      /* Failure indicator                           */
)
```

Specific errors

```
FGOPFA   Operation failed
FGPROP   Use default parameter properties
(FGOPOK  Operation successful)
```

This function is called, after FGCRSU has initialized the surface, to indicate to Parasolid some of the parametric properties of the surface. The function can specify the parameter ranges of the surface and indicate whether it is to be treated as periodic in u and/or v.

If the ifail value is set to FGPROP then Parasolid uses default properties for the surface evaluator: there is no need to set the range and periodicity flag. Default properties are parameter ranges [0, 1] for both u and v parameters and periodicity flags FGPRBD for both parameters (i.e. non-periodic parametrization).

If the ifail value is set to FGOPOK then this function must set the range array and the periodicity flags array.

- range[0], range[1] are lower and upper u parameter values
- range[2], range[3] are lower and upper v parameter values
- period[0] is the u parameter periodicity flag (must be FGPRBD)
- period[1] is the v parameter periodicity flag (must be FGPRBD)

Note: For the current Parasolid version closed and periodic surfaces are NOT permitted. This function includes the periodic argument for upwards compatibility with future planned enhancements to FG capability.

C

D

E

F

G

L

M

O

P

HTML Index

P