Carderock Division
Naval Surface Warfare Center
Bethesda, Maryland 20084-5000

CARDEROCKDIV-94/000
May 2000

DT_NURBS Spline Geometry Subprogram Library Users' Manual

Version 3.7

Prepared by
Boeing Shared Services Group

Edited by
United States Navy
Naval Surface Warfare Center/Carderock Division
David Taylor Model Basin
Bethesda, MD 20084

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

CHAPTER 2 - DTRC SPLINE GEOMETRY

CHAPTER 3 - DYNAMIC MEMORY MANAGEMENT FACILITY

CHAPTER 4 - CREATING CURVES

CHAPTER 5 - CREATING SURFACES

CHAPTER 6 - NOTES ON INTERSECTING SURFACES

CHAPTER 7 - GEOMETRY, GRIDS AND ANALYSIS

CHAPTER 8 - IGES FILE MANIPULATION

CHAPTER 9 - USER-DEFINED DATA TYPES

CHAPTER 10 - REINITIALIZING DYNAMIC MEMORY ARRAYS

CHAPTER 11 - SPECIAL INTERSECTION

CHAPTER 12 - POINT ON INTERSECTION

CHAPTER 13 - TRAVERSE TRIMMED BOUNDARY

CHAPTER 14 - TRIMMED & JOINED - DT_NURBS VS IGES

CHAPTER 15 - TRIMMED TO SUBRANGE

CHAPTER 16 - COMPUTING MOMENTS OF INERTIA

APPENDIX A - LICENSE TERMS AND CONDITIONS

INTRODUCTION 1

1.1 THE DT_NURBS LIBRARY

This Users' Manual contains general and introductory information about the DT_NURBS Spline Geometry Subprogram Library developed for the David Taylor Research Center by Boeing Computer Services.[1] The Library uses a Non-Uniform Rational B-Spline (NURBS) representation for general spline functions. It is implemented in the FORTRAN 77 programming language.

The other two manuals for the Library are the Reference Manual, which contains the complete interface information for each user-callable subroutine, and the Theory Document, which contains technical articles about the algorithms, implementation, and application of some of the more complex components of the Library.

The development of complex physical systems (ships, airplanes, turbine engines, etc.) typically involves the application of many independently created computer-based tools. A principal motivation for the creation of the DT_NURBS Spline Geometry Subprogram Library was the need for a tool kit for building interfaces between these many design and analysis tools, and to help establish standard representations for geometric design data and related analysis data which are not dependent on any particular tool's peculiarities.

1.2 SPECIAL FEATURES OF DT_NURBS

The NURBS representation is most often used as a computationally efficient way to express curves in a plane or curves and surfaces in space. It forms the basis for many Computer-Aided Design (CAD) systems. However, almost all the underlying mathematics works equally well for *n*-parameter functions to *m*-dimensional space. One of the unusual features of the DT_NURBS Library is that it was designed to support these more general functions. Some applications of the capability to represent functions with arbitrary numbers of independent and dependent variables are the ability to model a surface together with the pressure and temperature at each surface point as a single NURBS object, the ability to model a time-varying surface as a three-parameter NURBS object, the ability to model parametric solids, the ability to model fluid flows in a volume as a single continuous NURBS function, and the ability to create, manipulate, and analyze all these things with the same set of tools.

Standard FORTRAN 77 lacks the features needed to create complex data structures and to dynamically allocate memory based on run-time needs. Another unusual feature of the DT_NURBS Library is the inclusion of facilities which mitigate these deficiencies as much as possible while remaining fully compliant with the FORTRAN 77 standard and fully portable to any system with a FORTRAN 77 compiler.

[1] The formal names of both institutions have changed since the project began. See the title page for the current names.

1.3 OUTLINE OF THE USERS MANUAL

Chapter 1 contains introductory and version information about the Library. Chapter 2 describes the NURBS representation and why it is used. Chapter 3 describes the Library's Dynamic Memory Management Facility. Chapter 4 describes the tools in the Library for creating curves. Chapter 5 describes the tools in the Library for creating surfaces. Chapter 6 contains information about the various subroutines which compute intersections of surfaces. Chapter 7 explains the intent behind a new data structure which relates the geometry, the grids used in analyzing design performance and the analysis results together in one structure. The remaining chapters consist of case studies illustrating in brief form the application of the subroutines in the Library to a representative set of problems. It is hoped that concrete examples will be more helpful than hundreds of pages of generalities.

1.4 DISTRIBUTION OF THE DT_NURBS LIBRARY

The DT_NURBS Spline Geometry Subroutine Library source code, test code and documentation is available free under license to qualified organizations and individuals. An example of typical license terms and conditions is given in Appendix A.

The Library is distributed through a World-Wide Web site with the URL (at this writing) of http://ocean.dt.navy.mil. The most current information about the Library is likely to be found on this site. In particular, the procedure for registering for a free license will be found there.

The many files comprising a particular release of the DT_NURBS Library are assembled into a small number of large, compressed files in both UNIX "tar" format and "zip" format. The individual files and updates to individual files made between formal releases are also available.

As of this writing, the contact person for licenses and general information is

```
Robert Ames                          AmesRM@nswccd.navy.mil
NSWC/Carderock Division               (301)227-3657
David Taylor Model Basin           fax(301)227-1125
Code 50
Bethesda, MD 20084-5000
```

Library development is an ongoing activity, although the pace varies widely depending on funding and other factors. Please communicate with "ames" regarding any bugs, documentation errors, performance problems, suggested improvements, general needs, etc. User input has a major influence on priorities and general direction.

1.5 NAMING CONVENTIONS FOR LIBRARY GLOBAL NAMES

In order to make it easy to distinguish the Phase 2 routines which use dynamic memory management from routines which don't, and user-callable routines from Library internal subroutines, the naming convention for Library subroutines has been refined. All Phase 1 routines had names beginning with "DT", except for twenty low-level internal subroutines whose names began with "D". These initial characters were followed by three or four alphanumeric characters of "unique" name. The refined naming convention requires all "global" Library names to begin with "D", but varies the second letter to identify important major subsets of names as follows:

D0 Internal low-level routines, not user-callable.

D1 Internal "shadow" routines, not user-callable. The "shadow" part refers to the fact that there is a corresponding user-callable routine with the same "unique" name that does input checking and then calls the "shadow" routine to do the real work. The "shadow" routine is preferred for internal calls by other Library routines.

DT User-callable, non-memory-managing routines. All the user-callable Phase 1 routines remain in this category. The DT routines may be preferred for simple tasks that do not involve complex data structures or very many NURBS objects

DU Internal low-level routines, not user-callable, non-memory-managing.

D2 User-callable, memory-managing routines. All these routines have the dynamic memory arguments "CMEM, IMEM, DMEM" as their first three arguments. Where a DT subroutine and a D2 subroutine perform essentially the same function, the same "unique" name is used, although the argument lists differ significantly.

DC Reserved for Library common blocks. Communication via arguments is preferred. An internal common block is used by the Library error routines.

DI Reserved for Library "include" files. While "include" is a very common and useful feature, it is not standard. Hence it will not be used in Library source code. However, there are occasional instances of useful files which users may wish to "include" in their own code, either by using their own version of the "include" feature, or by directly embedding the text via a text editor.

During a transition period, there may be Phase 1 subroutines in the Library which belong in the DU category, but whose names have not yet changed from the original DT or D. Since none of these are intended to be user-callable, it should have no impact on user code.

1.6 HISTORY AND VERSION NOTES

The DT_NURBS Library has been evolving over time in response to the needs of its users. The first phase developed most of the DTxxxx subroutines. The fundamental data construct manipulated by these subroutines is the single NURBS function. The second phase extended the Library with more DTxxxx subroutines and new D2xxxx subroutines. The new series of subroutines implements a dynamic memory management scheme and supports composite data structures of arbitrary complexity in standard, portable FORTRAN.

1.6.1 VERSION 2.1 CHANGES

Version 2.1 of the Library adds two new capabilities: a full IGES file interface and a dynamic memory file interface. The IGES file interface is implemented by the definition of the structure of data types IGES Entity and IGES Index and by thirty-seven new D2xxxx subroutines. The key subroutines are D2IGRD, which reads an IGES file and creates a corresponding data structure in dynamic memory, and D2IGWR, which writes an IGES file from an IGES data structure in dynamic memory. The IGES File Reader and Writer include options to restrict transfers to the entities listed in the NASA-IGES Protocol. The relevant entities are listed in Chapter 3 of the Reference Manual. An alternate text-file-based method for saving, transferring, and retrieving arbitrary data entity structures built using the Library's dynamic memory management facility is implemented by the subroutines D2READ and D2WRIT. This new D2 File Format is also described in Chapter 3 of the Reference Manual.

1.6.2 VERSION 2.2 CHANGES

With Version 2.2 of the Library, dynamic memory versions of several popular spline evaluation and interpolation subroutines are added. The following table lists the subroutines and their dynamic memory names. Both versions are documented in the Reference Manual.

DT Name	D2 Name	Purpose
DTNPDR	D2NPDR	Evaluation of a partial derivative of a tensor product spline
DTNPVL	D2NPVL	Evaluation of a tensor product spline
DTSPDR	D2SPDR	Evaluation of a spline and its derivatives
DTSPVL	D2SPVL	Evaluation of a spline
DTCNTR	D2CNTR	Contour generation
DTNSI	D2NSI	Construction of a natural spline interpolant

To support this conversion, several new entity types and their support subroutines had to be created. Following is a list of these new subroutines.

Entity type	New Subroutine	Purpose
SA: Sub-Array Access	D2SADF	Define
SM: Sub-Matrix Access	D2SMDF	Define
CA, IA and DA: Character, Integer and Double Precision Array	D2ADF	Define
	D2ASZ	Get dimensions
	D2AETC	Transfer one element of appropriate type
	D2AETI	
	D2AETD	
	D2AAT	Transfer entire array or sub-array
	D2AATD	Transfer double precision array to/from a Sub-Array Access (SA) entity
CM, IM and DM: Character, Integer and Double Precision Matrix	D2MDF	Define
	D2MSZ	Get dimensions
	D2METC	Transfer one element of appropriate type
	D2METI	
	D2METD	
	D2MMT	Transfer entire matrix or sub-matrix
	D2MMTD	Transfer double precision matrix to/from a Sub-Matrix Access (SM) entity
	D2TPSM	Transpose matrix
CQ, IQ and DQ: Character, Integer and Double Precision Sequence	D2QDF	Define

Entity type	New Subroutine	Purpose
	D2QSZ	Get dimensions
	D2QATC	Transfer sequence of appropriate type
	D2QATI	
	D2QATD	
BF: B-spline Function Array	DTGETS	Get size information
	D2BFSZ[2]	
	DTKNOT	Transfer the knot sequence
	D2BFKN	
	DTBRKP	Transfer the breakpoints
	D2BFBP	
	DTCTLP	Transfer the control points
	D2BFCP	

Also included in this Version are a new Composition of Functions (CF) data type and subroutines which support this new structure. Following is a list of these new subroutines.

New Subroutine	Purpose
D2POSE	Compose two existing functions to produce a new Composition of Functions (CF) entity
D2POSX	Extend a composition
D2DPSE	Decompose a Composition of Functions (CF) entity
D2BINE	Combine two functions to produce a new Spline Function (BF) entity
D2CONC	Concatenate two functions to produce a new Spline Function (BF) entity
D2MPBC	Build a spline mapping from the unit square into a planar region
D2MPQD	Build a spline mapping onto a planar region defined by a quadrilateral
D2EVLF	Evaluate a Composition of Function (CF) entity
D2EVL1	Evaluate a Composition of Function (CF) entity which has just one independent variable
D2EVL2	Evaluate a Composition of Function (CF) entity which has just two independent variables
D2EVLS	Evaluate a "small" Composition of Function (CF) entity which no more than two independent variables and three dependent variables
D2EVDF	Evaluate partial derivatives of a Composition of Function (CF) entity
D2EVD1	Evaluate derivatives of a Composition of Function (CF) entity which has just one independent variable
D2EVD2	Evaluate partial derivatives of a Composition of Function (CF) entity which has just two independent variables
D2EVDS	Evaluate partial derivatives of a "small" Composition of Function (CF) entity which no more than two independent variables and three dependent variables

1.6.3 VERSION 2.3 CHANGES

With Version 2.3 of the Library, two new entity types were added to support the Grid, Geometry and Analysis concept. See Chapter 2 of the Reference Manual for details.

[2] In cases where there is both a new DTmmmm and a D2nnnn subroutine, the "D2" subroutine is a dynamic memory form of the "DT" subroutine.

The following new subroutines were added to support these new entities.

New Subroutine	Purpose
D2GACF	Geometry And Analysis Add Function
D2GADF	Geometry And Analysis Define
D2GADN	Evaluate Derivative Of A Geometry And Analysis Function By Name
D2GADV	Evaluate Partial Derivative Of A Geometry And Analysis Function
D2GAEN	Evaluate A Geometry And Analysis Function By Name
D2GAEV	Evaluate A Geometry And Analysis Function
D2GRDF	Grid Define

Five new IGES-style entity types were defined in order to write the new Grid (GR) and Geometry and Analysis (GA) entities to an IGES file. These are described in Chapter 3 of the Reference Manual.

The following new subroutines were added to convert the Grid (GR) and Geometry and Analysis (GA) entities and their dependencies to and from the DT_NURBS-defined IGES types.

New Subroutine	Purpose
D2BFDI	Convert B-Spline Function (BF) To DT_NURBS IGES Type
D2DIBF	Convert DT_NURBS IGES Type To B-Spline Function (BF)
D2CFDI	Convert Composition Of Functions (CF) To DT_NURBS IGES Type
D2DICF	Convert DT_NURBS IGES Type To Composition Of Functions (CF)
D2CQDI	Convert Character Sequence (CQ) To DT_NURBS IGES Type
D2DICQ	Convert DT_NURBS IGES Type To Character Sequence (CQ)
D2DADI	Convert Double Precision Array (DA) To DT_NURBS IGES Type
D2DIDA	Convert DT_NURBS IGES Type To Double Precision Array (DA)
D2DMDI	Convert Double Precision Matrix (DM) To DT_NURBS IGES Type
D2DIDM	Convert DT_NURBS IGES Type To Double Precision Matrix (DM)
D2DQDI	Convert Double Precision Sequence (DQ) To DT_NURBS IGES Type
D2DIDQ	Convert DT_NURBS IGES Type To Double Precision Sequence (DQ)
D2GADI	Convert Geometry And Analysis (GA) To DT_NURBS IGES Type
D2DIGA	Convert DT_NURBS IGES Type To Geometry And Analysis (GA)
D2GRDI	Convert Grid (GR) To DT_NURBS IGES Type
D2DIGR	Convert DT_NURBS IGES Type To Grid (GR)
D2IADI	Convert Integer Array (IA) To DT_NURBS IGES Type
D2DIIA	Convert DT_NURBS IGES Type To Integer Array (IA)

Also, the B-Spline Function (BF) and Composition of Function (CF) entity types were modified to accommodate labeling of independent and dependent variables.

The following new subroutines were added to support variable labeling.

New Subroutine	Purpose
D2BFDP	Dump A B-Spline Function Entity To A File
D2BFER	Erase B-Spline Function
D2BFLV	B-Spline Function Label Variables
D2BFMG	B-Spline Function Merge
D2BFSL	B-Spline Function Select By Labels
D2CFDP	Dump A Composition Of Functions Entity To A File
D2CFLV	Composition Of Functions Label Variables
D2CFSL	Composition Of Functions Select By Labels
D2FEVL	Fetch Variable Label

Two new distance subroutines were added to determine the Hausdorff distance between two functions, and to determine a distance function between two curves.

New Subroutine	Purpose
D2HDIS	Hausdorff Distance Between Functions
D2PDIS	Distance Between Parametric Functions

Significant Changes to Existing Subroutines

The derivative evaluation routines for the Composition of Functions (CF) entity type [**D2EVDF**, **D2EVDS**, **D2EVD1**, **D2EVD2**] were found to be in error, if the composition contained more than one B-Spline Function (hereafter referred to as a "complex composition"). These have been corrected. If a derivative of a complex composition is desired, use **D2EVDF**. A second, or higher, derivative of a complex composition is not available at this time.

Because several conversions of DT_NURBS entities into IGES involve dependent entities, it made sense that the subroutines themselves should keep track of the current directory entry number, rather than the user. Thus, the JDE parameter for the IGES conversion routines [**D2BFIG**, **D2CYIG**, **D2LPIG**, **D2TSIG**] is now an INPUT/OUTPUT parameter. On input to the subroutine, JDE should point to the directory entry in which the parent entity will be stored. On output, JDE will be set to the next free directory location. If the last entry has been used, the IGES Index (GI) entity will be expanded and JDE will point to the newly created space. The calling program should no longer need to increment JDE between subroutine calls. The user should confirm that none of the listed subroutines, as well as the new **D2*xx*DI** subroutines, is called with a constant in the place of the JDE parameter.

D2IGWR has been modified to remove null directory pointers from the end of the IGES Index (GI) entity before writing the IGES file.

1.6.4 VERSION 2.4 CHANGES

With Version 2.4 of the Library, extensions to the IGES translation capability have been added.

A powerful new surface-surface intersector D2SSXT, using the dynamic memory management scheme was added.

Also, the contour subroutine D2CNTR was re-written to accommodate higher order splines. D2CNTR was previously limited to cubic splines.

The following new subroutines were added.

New Subroutine	Purpose
D2GRGA	Construct Geometry and Analysis from Grid of Points and a Parent Surface
D2IGBQ	Convert IGES type 102 (Composite Curve) to a B-Spline Sequence entity (BQ)
D2IGCF	Convert IGES type 142 (Curve on a Surface) to a Composition of Functions (CF)
D2IGDM	Convert IGES type 124 (Transformation Matrix) to a Double Precision Matrix (DM)
D2SSXT	New Surface-Surface Intersection Subroutine

Modified Sub-routine	Purpose
D2IGBF, D2IGCY	Extended to handle IGES types 100 (Circular Arc), 104 (Conic Arc), 106 (Copious Data--Polyline), and 110 (Line Segment)
D2IGTS, D2IGLP, D2IGEG, D2LPIG	Extended to handle IGES type 144 (Trimmed Surface)

1.6.5 VERSION 2.5 CHANGES

Version 2.5 of the Library primarily involves a restructuring of the DT_NURBS documentation. The previous Users' Manual has been extended to include much more usage information and examples. The subroutine documentation has been moved to a new Reference Manual.

In addition to the documentation changes, the following new subroutines were added:

New Subroutine	Purpose
D2ARLN	Dynamic memory version of DTARLN ("Compute Arc Length")
D2CLPM	Compute the Point(s) on a Set of Surfaces and/or Curves Closest to a Given Point
D2CLSP	Dynamic memory version of DTCLSP ("Compute the Point on a Surface or Curve Closest to a Given Point")
D2EQAR	Dynamic memory version of DTEQAR ("Extract Points at Equal Arc Length")
D2MDIS	Minimum Distance between Functions
D2RADC	Dynamic memory version of DTRADC ("Compute Radius of Curvature")

Modified Sub-routine	Purpose
D2HDIS	Add parameter IFSOL= parameters of each function at the solution point
D2PDIS	Add parameter IFSOL= parameters of each function at the solution point

1.6.6 VERSION 3.0 CHANGES

With version 3.0 of the Library, the definition of the pointer has been modified, so that the headers are calculated from the top of the IMEM Integer Memory Array. This enables memory to be expanded by the user, and D2REIN to be called to shift the headers up with the expansion, with no change to the existing pointers. See Section 3.3 for an explanation of dynamic memory pointers.

The following new subroutines were added:

New Subroutine	Purpose
D2ALFR	Compute Arc Length Fraction (Curve)
D2ALFS	Compute Arc Length Fraction (Surface)
D2ALPT	Compute Arc Length Point
D2CAPP	Approximate a Conic Section with a Polynomial Spline
D2CLXT	Curve and Line Intersection
D2CNPR	Extract a Constant Parameter Curve
D2CRBL	Blend Curves
D2CSI	Construction of Complete Spline Interpolant
D2DIGL	Compute Double Integral
D2GCSI	Complete Spline Interpolant on a Multivariate Grid
D2GELE	General Linear Equation Solver
D2GESL	General Linear Equation Solver for Additional Right Hand Sides
D2GNSI	Natural Spline Interpolant on a Multivariate Grid
D2GPAR	Multivariate Grid Parameterization
D2GRBL	Blend a Grid (Network) of Curves
D2GTPA	Weighted Least Square Tensor Product Spline Approximation of Data on a Multivariate Grid
D2HSI	Construction of Hermite Spline Interpolant
D2JOIN	Join Two Splines Together
D2LCMB	Linear Combination of Two Splines
D2LCMS	Form Linear Combination of a Mesh
D2LCST	Form a Linear Combination of Strings of Data
D2LSA	Weighted Least Square Spline Fit to Data
D2MGKT	Combine Knots
D2MIRI	Mirror a Spline Surface
D2MMPS	Matrix-Matrix Product
D2MVPS	Matrix-Vector Product
D2OSLO	Add Knots to a Spline
D2PCS1	Construction of Periodic Cubic Spline Curve
D2PCS2	Construction of Periodic Cubic Spline Surface
D2PCUT	Compute the Intersection of a Surface and a Plane
D2PCXT	Curve and Plane Intersection
D2PLAR	Compute Planar Area
D2PLN3	Generate a Spline Representation of a Plane, Given 3 Points
D2PLNE	Generate a Spline Representation of a Plane
D2QUAD	Compute Single Integral
D2RDCA	Read B-Spline Data from a File
D2REAX	Read D2 File Format with Extra Output
D2REIN	Reinitialize Dynamic Storage Arrays After Expansion
D2RHOC	Build a Rho-Conic
D2RMKT	Remove a Knot from a SPLINE
D2RPRM	Reparameterize a Spline
D2SCHK	Check a B-Spline Function Entity and Data for Validity
D2SCHT	Find Points Along a Curve, Satisfying a Chord-Height Tolerance
D2SCR3	Generate a Circular Arc, Given 3 Points
D2SCRC	Generate a Circular Arc Given the Center and …
D2SEPP	Extract Points from a Curve At Equally Spaced Parameter Values
D2SFAR	Compute Surface Area

New Subroutine	Purpose
D2SLIN	Generate a Spline Representation of a Line
D2SLNE	Generate a Spline Representation of a Line, Given the End Points
D2SLXT	Surface and Line Intersection
D2SMGD	Smooth a Grid of Points
D2SMTP	Smooth a String of Points
D2SOFF	Determine an Offset Curve
D2SPAD	Analytic Derivative of a Spline
D2SPJN	Join Two Splines Together
D2SREV	Surface of Revolution
D2SRFC	Surface Curvatures
D2SRFN	Surface Normal
D2STRM	Trim a Spline
D2SZER	Zeros of a Univariate Function
D2TROT	Create a Transformation Matrix to Rotate an Object
D2TSCL	Create a Transformation Matrix to Scale an Object
D2TTRN	Create a Transformation Matrix to Translate an Object
D2UPDG	Raise the Degree of a Spline
D2WRCA	Write B-Spline Data to a File
DTCSIZ	Compute DTRC Spline Array Size
DTRPRM	Reparameterize a Spline

Many of the dynamic memory (D2xxxx) subroutines and functions were modified to accommodate the new pointer structure. In addition, the following routines were modified:

Modified Subroutine	**Purpose**
DTCNPR	Remove dimension restrictions
DTRDCA	Remove dimension restrictions
DTWRCA	Remove dimension restrictions

1.6.7 VERSION 3.1 CHANGES

The following new subroutines were added:

New Subroutine	Purpose
D2BQIG	Convert B-Spline Sequence to IGES
D2CCTR	Trim a Plane/Space Curve by a Plane/Space Curve
D2CCXT	Curve-Curve Intersection
D2CFER	Erase a Composition of Functions Entity
D2CFSZ	Fetch Composition of Functions Size Information
D2CSTR	Trim a Space Curve by a Space Surface
D2CSXT	Curve-Surface Intersection
D2FTRM	Trim a Spline From a BF or CF
D2GAER	Erase a Geometry and Analysis Entity
D2IGNL	Convert IGES to Name Label Entity
D2IGPA	Convert IGES Associativity Instance to a Pointer Array
D2JSCR	Create a Joined Surface From Two Edge Entities
D2JSER	Erase a Joined Surface Entity
D2JSMP	Moment of Inertia for B-Rep Solids
D2MFDF	Define Matrix Function

New Subroutine	Purpose
D2MOIN	Moment of Inertia for Parametric Solids
D2NEPT	Find Nearest Edge of a Trimmed or Joined Surface to a Given Point
D2NLIG	Convert Name Label Entity to IGES
D2NVLS	Weighted Least Squares Fit to Bivariate Data
D2OFFS	Offset a Spline Surface
D2PAIG	Convert Pointer Array to IGES Associativity Instance
D2PATD	Parameterize and Approximate Three Dimensional Data
D2PCOS	Project a Curve Onto a Surface
D2SFDF	Define Select Function
D2SIMP	Simplify a Spline by Removing Knots
D2TSCR	Create a Trimmed Surface From a Surface and Trimming Curve
D2TSCV	Curvatures of a Trimmed Surface at a Point
D2TSDR	Partial Derivative of a Trimmed Surface at a Point
D2TSET	Tangent Vector to the Nearest Edge Point of a Trimmed Surface at a Point
D2TSNR	Unit Normal Vector of a Trimmed Surface at a Point
DTCHTA	Find Points Along a Spline Satisfying a Chord-Height Tolerance for Each Coordinate
DTPMTZ	Multivariate Grid Parameterization

In addition to minor bug-fixes, the following routines were modified:

Modified Sub-routine	Purpose
D2BINE	No longer restrict IFU1 to being a BF
D2EGDF	Allow curve input to be CF in addition to BF entity.
D2EGER	Erase curve entity based on its type - BF or CF.
D2GRBL	Extend to cases where there are more knots than curves
D2JOIN	Ensured that various pieces of DMEM are unlocked on return.
D2PJDC	IER returns changed due to allowing the Surface function to be a BF, CF or GA and the Curve function to be a BF or CF.
D2PJIC	IER returns changed due to allowing the Surface function to be a BF, CF or GA and the Curve function to be a BF or CF.
D2PJMC	IER returns changed due to allowing the Surface function to be a BF, CF or GA and the Curve function to be a BF or CF.
D2PJMS	IER returns changed due to allowing the Surface function to be a BF, CF or GA and the Curve function to be a BF or CF.
D2POSE	Extend to functions expressed as U0 subtypes
D2POSX	Extend to functions expressed as U0 subtypes
D2RMKT	Ensured that space pointed to by IBF is unlocked on return.
D2RPRM	Revised to unlock IBF before returning.
D2SSXT	Bug fixes.
D2STRM	Improved algorithm. Also, now permits output location to be the same as the input. In this case, the output spline data overwrites the input spline data. The DMEM allocation to the spline is not changed, however.
D2TSER	Erase surface entity based on its type - BF, CF, or GA.
DTGPAR	Add handling of degenerate cases where successive rows are the same
DTGRBL	Extend to cases where there are more knots than curves
DTSTRM	Improved algorithm.

1.6.8 VERSION 3.2 CHANGES

Version 3.2 was created specifically to support the Portable Extendable Viewer (PEV) version 1.0 developed at NASA's Lewis Research Center. It includes only those changes which were needed by the PEV development team, the surface intersector improvements which just missed the previous version's deadline, and minor corrections believed to have no impact on users other than the avoidance of warning messages from some compilers.

The following subroutines were modified:

Modified Sub-routine	Purpose
D2CFDI	Remove doubly declared variables
D2DPSE	Remove doubly declared variables
D2EQAR	Removed limitation to four dependent variables. Parametrized other constants.
D2MPBC	Fix nonuse of IHOSTn for n=1..4. Make distinct error codes in place of -999
D2NSI	Remove doubly declared variables
D2PCS1	Remove doubly declared variables
D2PCS2	Remove doubly declared variables
D2SCHT	Remove doubly declared variables
D2SSXT	Improved ability to handle surface and surface derivative discontinuities - in particular, discontinuous, C0, and C1 surfaces.

1.6.9 VERSION 3.3 CHANGES

The following subroutines were added:

New Subroutine	Purpose
D2CFBF	Combine CF to make simplest BF composition
D2CFFU	Extract function component from a CF entity
D2CFKN	Fetch composition of functions knots
D2CFSX	Fetch composition of functions size information (extended)
D2CHTA	Advanced chord-height-based curve approximation
D2DTTR	General purpose dt_nurbs to iges translator
D2EXTP	Closest(or farthest) point on a geometric object to a given point
D2GASZ	Fetch geometry and analysis function size information
D2IGTR	General purpose iges to dt_nurbs translator
D2INVP	Inverse image of a point
D2INVQ	Inverse image of a point quit
D2INVS	Inverse image of a point setup
D2NORP	Normals to a geometric object through a given point
D2SFVL	Select function variables by label
D2SFVN	Select function variables by number
D2SWCH	Switch parameters in a two-parameter BF entity
D2TRCE	Entity trace

The following routines were modified:

Modified Sub-routine	Purpose
D2ALFR	Revise to avoid integrating across a knot and reporting of unexpected and out-of-memory errors.
D2ALFS	Improve reporting of out-of-memory errors.
D2ARLN	Fix computation on CFS.
D2BFCP	Prevent zeroing of V array.
D2CFBF	Add IER = -11 out-of-bound check.
D2CFSZ	Fix problem with Identity map components.
D2CLSP	Fix calculation of ndep for rational BFs.
D2CNTR	Added IER = 2 check/bug fix.
D2IGTS	Remove IMEM setting after D2EGDF call.
D2PCXT	Change DATA SUBNAM from 8 characters to 6.
D2RDCA	Improve reporting error message.
D2SOFF	Change DATA SUBNAM from 8 characters to 6.
D2SSXT	Added code so that if d0ssxt returns a positive ier, the order of the surfaces is switched and d0ssxt is called again.
DTCTLP	Prevent zeroing of V array.
DTRDCA	Improve reporting error message.
DTRPRM	Minor correction to handle rational splines.
DTSPRM	Fix coefficient reordering problem by using LAT stuff and eliminates need for work-space.

Minor corrections have been made to:

D2GELE
D2GPAR
D2GRGA
D2PCS2
D2RADC
D2RHOC
D2SMGD
D2BFMG
D2CONC
D2BFLV
D2CFLV
D2CYIG
D2EVLF
D2SZER
D2GQNM
D2RADC
D2RPRM
D2SIMP
D2MDF

1.6.10 VERSION 3.4 CHANGES

Significant changes were made to:

Modified Sub-routine	**Purpose**
D2CFSZ	Fix errors in handling CF's with no BF's and in memory release upon error.
D2EVD1	Improve speed. Add input and output options. Support option to request one derivative at a time.
D2EVD2	Improve speed. Simplify by making use of new low-level routines used by D2EVDF. Prevent use on one-variable functions.
D2EVDF	Improve speed and correct handling of certain CF's. Also prepare for extension to second derivatives of CF's. Suppress second and higher derivative cases which used to be computed, but not always correctly. Add simpler LSADRV, LSAPAR, and LSAFDV options.
D2EVDS	Improve speed. Simplify by making use of new low-level routines used by D2EVDF.
D2EVL1	Improve speed. Simplify by making use of new low-level routines used by D2EVLF.
D2EVL2	Improve speed. Simplify by making use of new low-level routines used by D2EVLF. Prevent use on one-variable functions.
D2EVLF	Improve speed. Add simpler LSAPAR and LSAFV options.
D2EVLS	Improve speed. Simplify by making use of new low-level routines used by D2EVLF.
D2INVP	All new version with more general capabilities than previous versions.

The D2EVxx series of evaluators was reworked to allow isolated selection, linear, and affine functions as well as B-spline functions and compositions of functions. The two-variable routines lost the ability to accept one-variable functions. The specification of derivative orders for compositions of functions is now in terms of the actual input variables to the composition and not in terms of the first B-spline component that happens to be within the composition. Second and higher derivatives will no longer be computed for compositions with a single B-spline. They were not always computed correctly by the former version. Correctly computed second derivatives for any composition of functions will be added in a later revision.

1.6.11 VERSION 3.5 CHANGES

The following subroutines were added:

New Subroutine	Purpose
D2DSPF	Construct a spline function which is the sum of a geometry function and a pointwise displacement function
D2FECS	Fetch compete character string content of a Dynamic Memory Entity
D2GAFL	Extract a label for a component CF from a Geometry and Analysis (GA) entity
D2GAFU	Extract a component Composition of Functions (CF) from a Geometry and Analysis (GA) entity
D2MSPX	Extrapolate multivariate spline function in one of its independent variables
D2PLMT	Compute moments of inertia for region in plane bounded by curve.
D2PSXT	Find all intersection components of a plane and a surface
DTSPEX	Extrapolate univariate spline function

The following routines were modified:

Modified Subroutine	Purpose
D2ALFR	Improve robustness
D2BQBF	Deal with case having only one spline function in the spline sequence
D2CFBF	Correct handling of out-of-memory error from lower-level
D2CRBL	Fix case of bad swap when IDOM = 2
D2DTTR	Fix bug involving 3D rational spline surfaces
D2IGED	Bug fixes
D2IGLP	Add orientation recognition and outer/inner boundary determination
D2IGTS	Improve error detection, translation of type 102, and add outer boundary calculation for type 144
D2INVP	Internal simplification
D2JOIN	Increased work space for curves
D2MGKT	Replace -999s with distinct error numbers to improve diagnostic capabilities
D2NVLS	Extended from bivariate to multivariate data
D2PATD	Correct mistaken call of lower-level routine
D2PCOS	Correct mistaken call of lower-level routine
D2SSXT	Fix memory leak and improve robustness
D2UPDG	Remove unnecessary restriction to non-rational splines
DTRPRM	Fix to protect against changes to input A and other minor corrections
DTSZER	Fixed an inconsistency in the use of tolerances, failure to detect empty intervals, and a workspace computation.

Three changes have been made in lower-level components of the equation-solving subroutines to improve performance and robustness. Small changes in the outputs of all the intersection subroutines may be observable. Some solution points not previously found may now be found.

The Reference Manual is no longer structured as a Microsoft Word Master Document with five subdocuments representing each of the five chapters. Instead each chapter is now a separate Microsoft Word document and the front matter is labeled as if it were a separate Chapter 0. Three word macros have been added to the style file DT_NURBS.DOT. The macro MakeTocText creates a table of contents in text file form for each chapter it is executed in. These text file tables are then used by macro MakeToc (executed on Chapter 0) to update the subsections and page numbers in the master Table of Contents, and by macro UpdateFuncIndex (executed on Chapter 1) to update the page numbers and any missing subroutine titles in the functional index. The earlier system for keeping page numbers up to date using the Master Document structure and Word "bookmarks" was unreliable. Moreover, the document was getting impractically large.

1.6.12 VERSION 3.6 CHANGES

The following subroutines were added:

New Subroutine	Purpose
D2FLLT	Construct a constant radius fillet surface between two given surfaces.
D2MEMU	Report CMEM, IMEM, DMEM maximum memory usage.

The following routines were modified:

Modified Sub-routine	Purpose
D2BINE	Minor corrections involving rational splines.
D2DEFE	Minor change supporting D2MEMU.
D2DEFX	Minor change supporting D2MEMU.
D2DPSE	Fix variable labels corruption bug and others.
D2EQAR	Replace with new technology like that in D2ALFR.
D2EVLF	Fix initialization omission.
D2HDIS	Fix initialization omission.
D2INIT	Changes supporting D2MEMU and backslash problem fix.
D2MBAD	Changes to error codes resulting from D2MEMU implementation.
D2MDMP	Fix backslash problem. Handle changes to Master Control Block from D2MEMU implementation.
D2MDIS	Fix initialization omission.
D2MPBC	Add tolerance to corner point checking.
D2MSPX	Fix initialization omissions and null string problem.
D2OFFS	Fixed error handling.
D2PCUT	Fix initialization omission and another error.
D2PDIS	Fix initialization omission.
D2PLMT	Fix null string problem
D2POSE	Fix incorrect label check.
D2PSXT	Correct workspace requests.
D2READ	Fix backslash problem.
D2REAX	Fix backslash problem.
D2REIN	Fix two errors in moving header blocks and updating Master Control Block
D2SSXT	A series of changes were made to improve robustness. Results should no longer depend on the order in which the two surfaces are given.
D2SZER	Rewritten to use D2CFBF to handle CF input.
D2TSIX	Fix initialization omission.
D2UPDG	Fix errors in comments and improve efficiency slightly.
D2WRIT	Fix backslash problem.
DCOPY	Remove unrolled loop, which seems to trigger an optimization error in latest SGI compiler, in favor of letting compilers figure their own optimizations.
DTUPDG	Minor improvements

Many of the corrections mentioned above are actually made in lower-level subroutines called by the user-callable routines listed.

The documentation files (and associated macros) have migrated to a version of Word which the "about" window identifies as "Microsoft Word 97 SR-1". They cannot be saved as Word 95 / 6.0 without gross corruption despite the fact that the format has not changed at all and the content has not changed significantly.

1.6.13 VERSION 3.7 CHANGES

Version 3.7 collects bug fixes and requested extensions made in the year plus period since version 3.6 was released. It provides the opportunity to synchronize all users at a well-defined version with matching documentation.

The following subroutines were added:

New Subroutine	Purpose
D2ACBR	Area of surface defined on a curve-bounded region and many variations on that theme.
D2BINL	Variant of D2BINE which limits the degree of the resulting curve, thereby greatly improving the accuracy of the results.
D2CCJN	Simple-minded Curve-Curve Join. Avoids sometimes troublesome reparameterizatons done by D2SPJN and other behaviors by D2JOIN.
D2CVPR	Convert a polynomial spline to the corresponding rational spline.
D2DIGV	Provide a minor variant of D2DIGL for use by D2ICBR.
D2ICBR	Integrate over curve-bounded planar region.

The following routines were modified:

Modified Sub-routine	Purpose
D2BINE	Fix errors connected with rational splines.
D2CCXT	Several improvements to robustness and reliablility.
D2DTTR	Fix several errors in data management.
D2FEVL	Minor code improvements.
D2FLLT	Minor code improvements.
D2FTRM	Fix initialization omission.
D2GADN	Fix locked memory problem.
D2GADV	Fix locked memory problem.
D2GAEN	Fix locked memory problem.
D2GAEV	Fix locked memory problem.
D2GAFL	Fix locked memory problem.
D2GAFU	Fix locked memory problem.
D2GTPA	Fix working storage problem.
D2IGLP	Revise to handle IGES type 143 subtype 0 (no parameter curves).
D2IGRD	Expand the string parameter capacity and handle multi-line strings.
D2IGTR	Fix several errors in data management.
D2IGTS	Extend to handle IGES type 143 subtype 0.
D2IGWR	Fix multi-line string errors.
D2IGXI	Fix errors in translation index management and improve efficiency.
D2LPIG	Change to call D2BINL instead of D2BINE.
D2MPQD	Simplified code.
D2MSPX	Fix error.
D2NORS	Fix minor error.
D2NVLS	Fix declaration and header comments.
D2LSA	Fix working storage problem.
D2PATD	Fix minor error.
D2PCOS	Fix errors related to IGES types 141 and 143 and relax certain restrictions.
D2SFAR	Extend to handle TS (Trimmed Surface) and JS (Joined Surface) using D2ACBR. Also improved accuracy and robustness.
D2SIMP	Fix minor error.
D2SOFF	Fix minor error.
D2SPJN	Fix memory management errors.
DTGTPA	Fix working storage problem.
DTLSA	Fix working storage problem.
DTLSAA	Fix working storage problem.
DTPCUT	Improve code portability.
DTPMTZ	Fix minor error.
DTSCHT	Improve code portability.
DTSCR3	Improve code portability.
DTSLXT	Improve code portability.
DTSPJN	Fix error in handling of rational splines.
DTSPDR	Improve code portability.
DTSPVL	Improve code portability.
DTSPRM	Fix error in handling of rational splines.
DTSSXT	Improve code portability.

Many of the corrections mentioned above are actually made in lower-level subroutines called by the user-callable routines listed.

DTRC SPLINE GEOMETRY 2

2.1 INTRODUCTION

Consider a circle of radius R, centered at the origin, in a plane. How should this be represented in the computer? The answer depends on what one is going to do with that circle, and on what other geometric objects might also need to be used in the same context. Suppose one needs to be able to represent any "reasonable" continuous curve. Then there are two main strategies: one can use the equation of the curve, in this case: $x^2 + y^2 = R^2$, or one can describe the curve by parametric functions, for example: $x = R \cos t$ and $y = R \sin t$, where $0 \leq t < 2\pi$. At first sight the equation is simpler. It provides an easily computed criterion for when a point (x,y) is, or is not, on the circle. But what it doesn't do is provide an easy way to find a lot of those points, which is what is needed to, for example, draw the circle on a screen. The parametric representation introduces a new variable, t, has two formulas instead of one equation, and involves bounds on t, but is nevertheless easier to use for drawing and many other operations. To find a sequence of well-spaced points along the circle, pick a sequence of well-spaced values of t between 0 and 2π and compute the corresponding x's and y's.

Parametric representations are not unique. The parametric functions: $x = R \cos 2\pi u$ and $y = R \sin 2\pi u$, where $0 \leq u < 1$, describe exactly the same circle. The parametric functions:

$$x = R(1-t^2)/(1+t^2) \text{ and } y = R(2t)/(1+t^2) \text{, where } 0 \leq t < 1,$$

describe the quarter of the same circle in the first quadrant. With some sign changes, the latter parameterization can be used for each quarter circle, thereby giving the whole circle in four pieces using rational functions.

The purpose of the preceding paragraphs is to provide the foundation for an understanding of the strategic decision to represent geometric objects primarily in terms of parametric functions. In the following sections we touch briefly on the concepts on which the DTRC Spline Geometry rests: splines, degree, order, break-points, knots and multiplicities, B-spline representations, useful properties of splines and B-splines, and rational and tensor product splines. Readers needing or desiring a more thorough understanding of the concepts are referred to the DTRC Theory Document and Reference Manual and to the references therein.

The section is organized in seven subsections: polynomial splines, B-splines, geometry, computation, rational splines, tensor product splines, and the DTRC Spline Representation.

2.2 POLYNOMIAL SPLINES

What is a spline? Intuitively, a spline is a finite sequence of polynomial arcs satisfying certain smoothness conditions at their join points. The following are four examples.

Example 2.2.1:

$$s_1(t) = |t| = \begin{cases} -t & \text{if } -1 \le t < 0; \\ t & \text{if } 0 \le t \le 1 \end{cases}.$$

Example 2.2.2:

$$s_2(t) = \begin{cases} t & \text{if } -1 \le t < 0; \\ t^2 & \text{if } 0 \le t \le 1 \end{cases}.$$

Example 2.2.3:

$$s_3(t) = \begin{cases} -t^2 & \text{if } -1 \le t < 0; \\ t^2 & \text{if } 0 \le t \le 1 \end{cases}.$$

Example 2.2.4:

$$s_4(t) = \begin{cases} -t^3 + t^2 & \text{if } -1 \le t < 0; \\ t^2 & \text{if } 0 \le t \le 1 \end{cases}.$$

Definition: Let points ξ_0 = < ξ_1 < … < ξ_q and polynomials p_1, … , p_q be given. The function s defined by

$$s(t) = \begin{cases} 0 & \text{if } t < \xi_0; \\ p_1(t) & \text{if } \xi_0 \le t < \xi_1; \\ \vdots & \vdots \\ p_q(t) & \text{if } \xi_{\text{q-1}} \le t \le \xi_q; \\ 0 & \text{if } t > \xi_q \end{cases}$$

is a spline function.

Even though we are only interested in the spline on the interval [ξ_0, ξ_q], we actually define it for all values by extending to the zero function. Throughout these discussions, all splines are extended in this manner.

- The join points are called break-points. In the examples, the values -1, 0, and 1 are the break-points.
- Degree is the maximum degree of any component. This means that Example 2.2.2 has degree two and Example 2.2.4 has degree three even though, in both cases, there are segments of lower degree.
- Order is one plus the degree.
- A knot is a break-point combined with a multiplicity. Multiplicity is calculated from the order and the smoothness at the break-point. If the order is k and the derivatives of the spline are continuous up to and including the (j - 1)st derivative, the multiplicity of the knot is (k - j). There is no requirement that knots be distributed evenly. In fact, for many geometric operations it is advantageous to distribute the knots in a fashion other than uniform. The DTRC spline library provides for this and this is the source of the NU in the acronym DT_NURBS.

In the examples, the first is continuous but there is a break at zero in its derivative. Since its order is 2 (degree + 1) the knot at zero has multiplicity one. In the second example there is also a discontinuity in the first derivative. But now the order is 3 so the knot has multiplicity two. A break in continuity occurs in the second derivative of example 2.2.3 and in the third derivative of example 2.2.4. In both cases the multiplicity of the knot at zero is one. In all the examples, -1 and 1 are knots of full (i.e. equal to the order) multiplicity.

Here are some properties of splines worth noting.

- Splines of order *k* are also splines of order *k*+1 with the multiplicity of each knot increased by one.
- The derivative of a spline of order *k* is a spline of order *k*-1 with the same knots.
- Polynomials of degree less than or equal to *k*-1 are splines of or *k* for any and all choices of knots and multiplicities.

There are two ways of representing knots and their multiplicities, either listing the knot and its multiplicity explicitly or replicating the knot a number of times correspond to the desired multiplicity. Thus, the knots for Example **2.2**.2 could be listed as -1, 0, 1 with multiplicities 3, 2, 3 or equivalently as -1, -1, -1, 0, 0, 1, 1, 1.

2.3 B-SPLINES

B-splines are specific splines of order k defined over subsets of *k* + 1 knots. Here are three examples.

Example 2.3.1: *T*= {-1, -1, 0, 1, 1} and order = 2. There are three B-splines given by:

$$s_1(t) = \begin{cases} t & \text{if } -1 \le t \le 0; \\ 0 & \text{if } 0 < t. \end{cases}$$

$$s_2(t) = \begin{cases} t+1 & \text{if } -1 \le t < 0; \\ 1-t & \text{if } 0 \le t. \end{cases}$$

$$s_3(t) = \begin{cases} 0 & \text{if } -1 \le t < 0; \\ t & \text{if } 0 \le t \le 1. \end{cases}$$

Example 2.3.2: *T*= {-1, -1, -1, 0, 1, 1, 1} and order = 3. Then the four B-splines are:

$$s_1(t) = \begin{cases} t^2 & \text{if } -1 \le t \le 0; \\ 0 & \text{if } 0 < t. \end{cases}$$

$$s_2(t) = .5\begin{cases} 4(t+1) - 3(t+1)^2 & \text{if } -1 \le t < 0; \\ (1-t)^2 & \text{if } 0 \le t. \end{cases}$$

$$s_3(t) = .5\begin{cases} (t+1)^2 & \text{if } -1 \le t < 0; \\ 4(1-t) - 3(1-t)^2 & \text{if } 0 \le t \le 1. \end{cases}$$

$$s_4(t) = \begin{cases} 0 & \text{if } -1 \le t < 0; \\ t^2 & \text{if } 0 \le t \le 1. \end{cases}$$

Example 2.3.3: The Bernstein polynomials $\phi_{i,k}$ of order k defined by

$$\phi_{i,k}(t) = \frac{k!}{i!(k-i)!}(1-t)^i t^{k-i}$$

are the B-splines for the knot set consisting of 0 and 1 each with multiplicity k. They also form the basis for Bezier curves.

The important fact about B-splines is the following. If order and knots k, $T = \{\tau_0 \le \tau_1 \le \ldots \le \tau_{m+k}\}$ are given, then any spline s of order k with these knots has a unique set of constants a_1, …, a_m for which

$$s(t) = \sum_{j=1}^{m} a_j B_j(t).$$

The importance of this fact is that we now have a clear cut route to travel in developing algorithms for defining splines. One simple consequence is that determining splines which interpolate data reduces to solving relatively simple sets of linear equations.

There are a number of beneficial properties of B-splines. Among these are:

- **Partition of Unity:** $\sum_{j=1}^{m-k} B_j(t) \equiv 1$ for $\tau_k \le t \le t_{m-k+1}$.
- **Local Support:** $B_j(t) \equiv 0$ if $t < \tau_j$ or $t > \tau_{j+k}$.
- **Positivity:** $B_j(t) > 0$ if $\tau_j < t < \tau_{j+k}$.
- **Variation Diminishing:** The number of sign changes of a spline is less than or equal to the number of sign changes in the sequence of coefficients.
- **Derivative Representation:** $s'(t) = (k-1)\sum_{j=2}^{m-k}((a_j - a_{j-1})/(\tau_{j-1+k} - \tau_j))C_j(t)$ where C_j are the B-splines of one less order based on the knots τ_2, , τ_{m-k-1}.

Each of these properties has numerical or geometric significance. The local support property leads to well-posed and sparse linear algebra equations for solving for spline coefficients. It also enables the local modification of geometry. Positivity leads to robust evaluation algorithms for B-splines.

The variation diminishing property is a powerful tool for controlling spline curves. For example, if all the coefficients are non-negative then so is the spline. (Warning, the converse is not true. Splines can be non-negative and still have negative B-spline coefficients.

The derivative representation means that exact B-spline based formulas for derivatives and for integrals are easily derived. It can also be used, together with the variation diminishing property, to control derivative behavior.

2.4 GEOMETRY

The B-spline representation for a curve has geometric significance. Let the curve be given as $C(t) = (x(t), y(t)) = (\sum_{j=1}^{m} x_j B_i(t), \sum_{j=1}^{m} y_j B_i(t))$.

Let $\vec{P}_j = (x_j, y_j)$ and rewrite the curve as

$$C(t) = \sum_{j=1}^{m} \vec{P}_j B_i(t) \qquad (2.4.1).$$

The quantities $\vec{P}_j$ are called the *control points* of the curve C and the polygon formed by the control points is called the control polygon.

The curve matches the control polygon at its first and last points. The tangents to the curve at the first and last points are parallel to the first and last legs of the control polygon.

Since, at each t the B-splines are positive and sum to one, (2.4.1) shows that the curve is a convex combination of its control points. Thus, $C(t)$ is contained in the convex hull of its control polygon. This is known as the *convex hull* property.

Other interesting properties derivable from elementary relationships are:

- An affine transformation (rotation plus translation) of a curve is accomplished by applying the affine transformation to each of the control points
- Convexity of the control polygon is sufficient but not necessary for convexity of the curve.
- Bezier curves of order k are curves for which each knot is effectively a k^{th} order knot.

2.5 COMPUTING B-SPLINES AND SPLINES

One of the most powerful reasons for usings B-splines is in evaluation. Formulas exist for evaluating B-splines and splines which are computationally stable.

The recurrence relation for B-splines relates the values of a k-order B-spline to those of a pair of (k-1)-order B-splines. Let T be the usual set of m knots and k the order. Denote by $B_{j,k}$ the j^{th} B-spline of order k. Then:

$$B_{j,k}(t) = \frac{x - \tau_j}{\tau_{j+k-1} - \tau_j} B_{j,k-1}(t) + \frac{\tau_{j+k} - x}{\tau_{j+k} - \tau_{j+1}} B_{j+1,k-1}(t) . \qquad (2.5.1)$$

This formula means that B-splines are evaluated by forming positive combinations of positive quantities. Thus, reducing the danger of errors through cancellation effects.

Formula 2.5.1 may be applied to obtain a formula for splines of order k in terms of lower order B-splines:

$$s(t) = \sum_{j=1}^{m} a_j B_{j,k}(t) = \sum_{j=2}^{m} a_j^{[2]} B_{j,k-1}(t) \qquad (2.5.2)$$

where the coefficients $a_j^{[2]}$ are given by

$$a_j^{[2]} = \frac{(t-\tau_j)a_j + (\tau_{j+k-1}-t)a_{j-1}}{\tau_{j+k-1}-\tau_j}.$$

These formulas do not require the knots to be simple. Any multiplicity ($\leq k$) are allowed. Thus, splines with multiple knots are as easily evaluated as those with simple knots.

The DTRC Spline Geometry Library evaluators implement these formulas. Equation 2.5.2 is used to evaluate a univariate spline function. Combinations of 2.5.1 and 2.5.2 are used to evaluate planar and higher order curves and surfaces.

2.6 RATIONAL SPLINES

Conic sections can be parametrized as ratios of quadratic polynomials. In particular, the conic section passing through fixed points Q_0, Q_2 and having tangents at those points which intersect at a third point Q_1 has the representation

$$r(u) = \frac{w_0 Q_0 (1-u)^2 + 2w_1 Q_1 u(1-u) + w_2 Q_2 u^2}{w_0 (1-u)^2 + 2w_1 u(1-u) + w_2 u^2}.$$

- The numbers w_i are called weights,
- If the weights are all equal, the conic $r(u)$ is a parabola.
- If all the weights are positive, the conic r is contained in the convex hull of the points Q_0, Q_1, Q_2.

Rational splines $R(t)$ are defined as

$$R(t) = \frac{x(t)}{y(t)} = \frac{\sum_{j=1}^{m} x_i B_i(t)}{\sum_{j=1}^{m} y_i B_i(t)}.$$

Planar rational spline curves are given by

$$C(t) = \left(\frac{x(t)}{w(t)}, \frac{y(t)}{w(t)}\right). \tag{2.6.1}$$

The B-spline coefficients $\{w_j\}_{j=1}^{m}$ of the denominator are called *weights*. The control points are the planar points $\vec{P}_i = (x_i/w_i, y_i/w_i)$. They enjoy the same properties as the control points for planar spline curves.

Example 2.6.1: A rational spline parameterization of a circular arc:

$$C(t) = \left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right).$$

The DTRC Spline Geometry Library represents conic sections as rational splines of the form (2.6.1).

2.7 TENSOR PRODUCT SPLINES

Tensor product splines and spline surfaces are direct generalizations of univariate splines and spline curves and enjoy many of the same properties. As in the univariate case, a tensor product spline is a collection of individual polynomial surfaces connected along lines called knot lines with certain continuity. They are not, however, made up of polynomial surfaces over arbitrary domains. Each domain is required to be rectangular and the collection of domains fits together as a rectangular grid.

Let orders k_u, k_v and knots $T=\{\tau_p\}_{p=1}^{P+k_u}$, $X=\{\xi_q\}_{q=1}^{Q+k_v}$ be given. The tensor product splines of order (k_u,k_v) with knots T, X are the functions f of the form

$$f(u,v)=\sum_{p=1}^{P}\sum_{q=1}^{Q} f_{i,j}B_p(u)C_q(v)$$

where the functions B_p, C_q are the B-splines of order k_u, k_v respectively with the knots T and X. Thus, f is a function defined on the rectangle $\left[\tau_1,\tau_{P+k_u}\right]\times\left[\xi_1,\xi_{Q+k_v}\right]$. As in the univariate case, f is taken to be zero outside the rectangle.

2.8 DTRC SPLINE REPRESENTATIONS

The DTRC Spline Geometry Library common spline representation is based on the B-spline representation. Accordingly, the information required to define a spline consists of:

- the order k of the spline
- the list of knots $\tau_1, \dots, \tau_{m+k}$ and
- the list of B-spline coefficients $a_1, \dots, a_m$.

In addition, the DTRC format stores the dimension of the parameter space (for curves this is one), the dimension of the model space (for functions one, for planar curves two, for space curves three and m for higher-dimensional curves in R^m). It also stores the number m-k of B-spline coefficients. Finally, one remaining parameter (called `jspan') is stored. This parameter has no geometric or mathematical significance but storing it improves evaluation performance.

The DTRC Spline Geometry Library common rational spline representation stores a rational spline in homogeneous coordinates with the parameter corresponding to the dimension of the model space (homogeneous coordinates) set to the negative of the number of dependent variables. Thus, the rational spline curve

$$C(t)=(\frac{x(t)}{w(t)},\frac{y(t)}{w(t)})$$

is stored as the space curve ($x(t)$, $y(t)$, $w(t)$) with the model space parameter set to -3. The spline evaluators DTSPVL and D2SPVL return the two numbers $x(t)/w(t)$ and $y(t)/w(t)$.

The complete specification of the DTRC representation for general spline functions with arbitrary numbers of independent and dependent variables may be found in Chapter 2 of the Reference Manual.

DYNAMIC MEMORY MANAGEMENT FACILITY 3

3.1 INTRODUCTION

The first phase of the DT_NURBS Spline Geometry Subprogram Library development provided routines to create and manipulate Non-uniform Rational B-spline (NURBS) objects. Single NURBS objects represent single curve segments, single surface patches, or higher-dimensional analogs thereof. A major goal of the second phase of development was to extend the Library to support representations of composite entities such as trimmed surfaces (surface patch with boundary curves), joined surfaces (multiple trimmed surfaces joined at designated edges), and solids (interiors of closed, oriented, joined surfaces).

In order to implement composite entities like trimmed and joined surfaces, one needs pointers to connect the components. The only portable way to implement pointers in standard FORTRAN 77 is to use indices to large arrays containing all the related or relatable components. This leads naturally to putting all components in one set of arrays managed at run time by a new set of Library subroutines.

3.2 DYNAMIC MEMORY MANAGEMENT SCHEME

The three main kinds of data of interest to the Library are character, integer, and double precision. Character data (or integer data) can serve for logical data, and double precision can serve for complex or real (i.e., single precision) data. Equivalence statements could be used to place all data in one array, but would tend to be non-portable and complex. One could also encode everything as double precision data, but not simply or efficiently. Thus, the simplest portable solution seems to be to dynamically manage three arrays, one for each main type. In order to take advantage of possible hardware string handling functions, the master "character array" is implemented as a single character string.

Thus, the routines which use dynamic memory (D2xxxx routines) all have as their first three arguments: CMEM, IMEM, and DMEM, which are the dynamically managed character, integer, and double precision memory areas, respectively. In return for these three constant arguments, there is no need for the usual work area and work area length arguments because the new Library routines can allocate their own temporary work areas as needed. Moreover, composite entities of arbitrary complexity can be communicated by a single pointer (integer) argument. For most subroutines, both Library and user, the total number of arguments is reduced, and the complexity of argument interrelationships is greatly reduced.

In the Library dynamic memory management scheme, a dynamically allocated data entity consists of memory for some number of characters, some number of integers, some number of double precision values, and some header information that ties the entity together, defines its type, and supports memory management of it. "Some number" includes zero as a common case. A pointer to a data entity is a positive integer encoding the index (subscript) to the header information and a sequence number used in error detection.

3.3 MANAGED MEMORY LAYOUT

The three managed memory areas will be referred to as CMEM, IMEM and DMEM throughout this document. These arrays must be initialized with a call to the subroutine D2INIT before any of the other memory management routines may be called.

The beginnings and ends of these arrays are reserved for Library use. Each data entity allocated using the Library dynamic memory management routines consists of a header block and up to three data areas, one in CMEM for character data, one in IMEM for integer data, and one in DMEM for double precision data. The data areas assigned to consecutively created data entities are consecutive blocks of locations proceeding from low to high subscript values for CMEM, IMEM, and DMEM. Except when prevented by locks, the data areas are kept compacted as data entities are erased.

A data area for an entity is in one of four memory management states: unlocked, explicitly locked, implicitly locked, or implicitly erased. Unlocked is the "normal" state in which all operations are permitted. A data area is explicitly locked whenever its absolute subscripts are being made available outside the Library routines. The explicit lock prevents the data area from being moved by garbage collection (data compaction) activities. A data area is implicitly locked whenever some data area at higher subscripts in the same array is explicitly locked. A data area is implicitly erased whenever it has been erased but not yet compacted away because some data area at higher subscripts in the same array is explicitly locked. The state of a given data area is determinable from information in the master control block and the entity header block.

The general layouts of the CMEM, IMEM, and DMEM managed memory areas are as follows:

IMEM

MEMAXI	U S E D	check value	<= IMEM(2)
. . . .		header blocks	<= IMEM(10, 11, 12, 17 and 18) <= IMEM(16)
. . . .	F R E E	free space	<= IMEM(5)
. . .	U S E D	unlocked	<= IMEM(8)
. . .		mixed	<= IMEM(14)
. . . 20		implicit	
19 . 1		master control block	

CMEM

MEMAXC	F R E E	~	<= IMEM(1)
. . . .		free space	<= IMEM(4)
. . .	U S E D	unlocked	<= IMEM(7)
. . .		mixed	<= IMEM(13)
. . 2		implicit	
1		H, M or L	

DMEM

MEMAXD	F R E E	9.87654321E9	<= IMEM(3)
. . . .		free space	<= IMEM(6)
. . .	U S E D	unlocked	<= IMEM(9)
. . .		mixed	<= IMEM(15)
. . 2		implicit	
1		9.87654321E9	

The segment labeled "unlocked" contains the most recently allocated data areas and is the only one in which data compaction or resizing in place can occur. The segment labeled "mixed" contains all explicitly locked areas and implicitly erased areas, plus implicitly locked areas between (and possibly below) them. If non-empty, its high end must consist of an explicitly locked area. If empty, the implicit area is also empty. The segment labeled "implicit" contains only implicitly locked data areas, and is empty when no explicit locks are present. The master control block at the beginning of IMEM contains the system data needed to manage memory. In particular, it contains the indices defining the segments in all the managed areas.

3.3.1 CONTENTS OF RESERVED LOCATIONS AND MASTER CONTROL BLOCK

The first character in CMEM holds the requested error check level, which should be 'L', 'M', or 'H' for low, medium, or high, respectively. The last character in CMEM is used for a clobber check value, which is chosen to be '~'. The first twenty-five locations in IMEM are reserved for memory management use as follows:

1	Length of CMEM, in characters
2	" IMEM, in integers
3	" DMEM, in double precision values
4	Next (lowest) free location in CMEM
5	" IMEM
6	" DMEM
7	Lowest unlocked location in CMEM
8	" IMEM
9	" DMEM
10	Location of header belonging to highest lock in CMEM
11	" IMEM
12	" DMEM
13	Lowest location to check when compacting memory in CMEM
14	" IMEM
15	" DMEM

16	Lowest used header location in IMEM
17	Beginning of active header block linked list in IMEM
18	Beginning of free header block linked list in IMEM
19	Last sequence number used in a new header block
20	Number of locations reserved for system use in CMEM
21	Number of locations reserved for system use in IMEM (length of Master Control Block + 1)
22	Number of locations reserved for system use in DMEM
23	Maximum locations used in CMEM since last reset by D2MEMU (or since the call to D2INIT)
24	Maximum locations used in IMEM since last reset by D2MEMU
25	Maximum locations used in DMEM since last reset by D2MEMU

The last location in IMEM is the sum of the first three, and serves as a primitive checksum for damage to IMEM. The first and last locations in DMEM are set to 0.987654321D10 to serve as clobber checks.

3.3.2 CONTENTS OF HEADER BLOCKS

Header blocks are eight integers long and allocated in the high end of IMEM. Once allocated, header blocks do not move. If an entity is erased, its header block may be reused for another entity, but only after incrementing the sequence number. The only time a header block is erased is if it is the lowest one in IMEM. Otherwise it is placed on the free header block linked list. Header blocks for new entities are taken from the free header block linked list if the list is non-empty, or allocated from the high end of the free area (low end of header block area) in IMEM.

The header block contents of an active header are as follows:

header_index	+ 0	pointer validity check (including seq. num.)
	+ 1	beginning of character data in CMEM
	+ 2	beginning of integer data in IMEM
	+ 3	beginning of double precision data in DMEM
	+ 4	length of character data
	+ 5	length of integer data
	+ 6	length of double precision data
	+ 7	link value (and type id number)

The pointer validity check value is identical to the current pointer value for the data entity in whose header it is found. Pointer values are defined as:

$$\text{Pointer} = 256\ \frac{\text{MEMAXI} - \text{header\_index}}{8} + \text{sequence\_number}$$

MEMAXI (=IMEM(2)) is the length of the IMEM array in integers. Sequence numbers are incremented modulo 256 every time a header block is reused for a new entity. This ensures that pointers to the old erased entity are recognized as invalid, even though the header index is the same. Of course, after a header block is reused 256 times, it will repeat an old pointer value, but the probability of an invalid pointer being accepted is quite small. The integer equality test between pointer and pointer validity check also provides some protection against data corruption in the header block. Hence, this test is performed even at the low error check level.

The link value's primary function is to locate the next header block in either the active header block linked list or the free header block linked list, whichever the header block currently belongs to. In the active header list, it also encodes a data type identification number (hereafter referred to as the "type id"). In the active header list, the link value is defined as:

$$\text{Link\_value} = 256\ \frac{\text{MEMAXI} - \text{next\_header\_index}}{8} + \text{type\_id}$$

In the free header list, the link value is defined as:

Link_value = next_header_index.

A next_header_index of zero marks the end of a list. The active header list is maintained in the order of decreasing data area subscripts as it is traced from beginning to end. Beginning locations for data areas are defined even when the length of that data area is zero. In this case they are identical to the beginning location of the next allocated entity (previous header block in the active header list).

Erasure is indicated by negation of the pointer check value. Note that negation of the pointer check value makes all references to the entity invalid by the usual test, even though the data may still be present.

Explicit locks are indicated by negating the length value of the corresponding data area in the header block. Hence, the absolute value function must be employed to get a proper length value independent of the locked state. Note also that a zero length data area cannot be explicitly locked.

The header block of a free header is the same as that of an active header except that the three lengths are necessarily zero, the pointer check value is negated, and the location holding the link to the next free header does not include any type id number.

The header block of an erased, but (implicitly) locked, entity has its pointer check value negated, and may have had some of its data areas compacted away (length reduced to zero), but otherwise looks like an active header, and remains in the active header linked list.

3.4 MEMORY ERROR DETECTION

The extensive use of pointers is prone to errors of a kind which are very difficult to trace, namely the inadvertent use of invalid pointer values to modify the wrong area of memory. This kind of error is not usually detected until something completely bizarre happens in an unrelated part of the program. The Library routines include some automatic pointer validity checking and data type checking which should catch most such errors immediately. In addition, a number of memory consistency checking (D2MBAD) and memory dump (D2MDMP, D2EDMP) routines have been included to aid in debugging.

Before the managed memory areas can be used, they must be initialized by a call to D2INIT. One of the arguments to D2INIT allows the caller to set the automatic error check level to low, medium or high. A high check level improves the likelihood of detecting damage to the managed memory areas promptly after it happens, but is also likely to significantly degrade program speed.

3.5 GARBAGE COLLECTION AND LOCKING

Garbage collection is accomplished by shifting active data spaces downward in the arrays and correcting the header blocks. In the absence of locking, used memory is always packed. This scheme is conveniently simple and is reasonably efficient if erasure happens predominantly to recently allocated data entities. If erasure or resizing happens frequently to data entities allocated near the beginnings of the arrays, a lot of time can be wasted, but this does not correspond to expected usage.

Maximal efficiency is obtained by computing directly with the data in the CMEM, IMEM and DMEM arrays, but the correct subscripts to identify data of a particular data entity at one time can become incorrect at a later time as a result of garbage collections. Hence the need for a locking scheme to prevent such data movements during periods of direct user access. Whenever the library routines which fetch the absolute subscripts (D2FEBx) of a data entity are called, the relevant data space is explicitly locked. It is up to the user to unlock the data space (D2UNLx) when he has finished his direct access to that data. It is better never to unlock than to unlock prematurely. The only penalty for never unlocking is that one may run out of memory much sooner, but this is always detectable. Unlocking and then using absolute indexes into CMEM, IMEM or DMEM after the data space has moved can corrupt memory and lead to errors which are difficult to trace. When the quantity of data to be accessed is small or speed is not too critical, the library routines which fetch and store the data values using subscripts relative to the data entity (D2FEAx, D2FEEx, D2STAx, D2STEx) are preferable to obtaining the absolute subscripts and accessing directly. These routines locate the data wherever it may be at the moment of the call, and are therefore much simpler and safer to use.

When a data area of an entity is explicitly locked, all data located at lower subscripts in the same managed memory array are implicitly locked. That is, no data movement is permitted. Erasure of an implicitly locked data entity is marked in the header block and access is subsequently denied, but no changes occur in the data and the space cannot be reused until it is subsequently unlocked. An explicit lock can only be removed by an explicit unlock (D2UNLx) or by erasing the entire entity (D2ERAS).

3.6 EXAMPLE APPLICATION

```
      PROGRAM EXMPL1
C  -- VERSION -- EXMPL1 :  1995-05-19
C
C     Load an IGES file and save it in the DT_NURBS D2 File Format.
C     Get the input and output file names from the user.
C
C     HISTORY
C        5/19/95   P. Kraushar  Adapted from T2IGRD
C

C     Define parameters for the sizes of the Dynamic Memory areas
      INTEGER     MEMAXC, MEMAXI, MEMAXD
      PARAMETER  (MEMAXC = 30000,
     +            MEMAXI = 150000,
     +            MEMAXD = 100000)

C     Allocate the Dynamic Memory areas
      CHARACTER         CMEM*(MEMAXC)
C     For those systems which impose a small limit on the length of
C     individual character strings, it may be necessary to define
C     CMEM as an array of individual characters instead of a string:
C     CHARACTER*1       CMEM(MEMAXC)
C     Reports to date are that the Library continues to work fine in
C     this case
      INTEGER           IMEM(MEMAXI)
      DOUBLE PRECISION  DMEM(MEMAXD)

C     Allocate other local variables
      CHARACTER*(80) FILEIN, FILEOU
      INTEGER        LUI, LUO, LVLI, IGI, LVLE, IER

C     ****

      LUI      = 8
      LUO      = 9

C     Get input and output file names interactively from user
      PRINT *, 'Name of input IGES file'
      READ (*,'(A)') FILEIN
      OPEN (LUI, FILE=FILEIN, STATUS='OLD', MODE='READ', ERR=9999)
      CLOSE (LUI)

      PRINT *, 'Name of output D2-format file'
      READ (*,'(A)') FILEOU

C     Initialize the Dynamic Memory.  The ' ' argument lists no user-
C     defined types, and the 'L' sets the error check level to low.
      CALL D2INIT (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, ' ', 'L',
     +      IER)
      IF (IER .NE. 0) GOTO 9999
C       Stop on any error or warning.

C     Read the IGES file into the Dynamic Memory.  Set input level to 3
C     to accept any IGES entity type.
      LVLI     = 3
      CALL D2IGRD (CMEM, IMEM, DMEM, FILEIN, -LUI, LVLI, IGI, LVLE, IER)
C       Negating the logical unit number causes more extensive reporting
C       on the standard output of the progress made in reading the file.

C     Report results from IGES read operation
      WRITE (*,'(/,A)') 'D2IGRD results:'
      WRITE (*,'(A,I1)') '   Highest Level Encountered (LVLE) = ', LVLE
      WRITE (*,'(A,I5)') '   Returned error status (IER) = ', IER
C       The output variable IGI contains the Dynamic Memory pointer to
C       the IGES Index entity.
      IF (IER .LT. 0) GOTO 9999
C       Stop on any error (but continue on any warning).

C     Write the DT_NURBS internal IGES data structure out in D2 File
C     Format.  Request the writing of a list of 1 pointer, namely IGI,
C     and use the default "user-supplied subroutine" (D2LPUT) to locate
C     pointers in user-defined types (since there are no user-defined
C     types).  All entities accessible from the IGES Index entity are
C     written to the output file automatically.
      CALL D2WRIT (CMEM, IMEM, DMEM, 1, IGI, D2LPUT, FILEOU, LUO, IER)
      WRITE (*,'(/,A,I5)') 'D2WRIT returned error status (IER) = ', IER

 9999 CONTINUE
      STOP
      END
```

CREATING CURVES 4

4.1 CURVES

By definition, a curve is a function from an interval in R^1 into R^n. Most commonly, n is two for a plane curve, or n is three for a space curve. When n is one, it is customary to include the parameter as the x coordinate and picture the function graph as the "curve", although, in fact, the parametric curve lies in a one-dimensional space. Cases where n is greater than three are also important, although harder to visualize. One case where a four-dimensional curve arises naturally is in intersecting two parametric surfaces in space. The intersection curve one first imagines is the three-dimensional curve lying where the surfaces intersect in space. However, for many purposes, it is the parameter values in each of the two surfaces' parameter spaces that map to this intersection which are most useful. A curve function providing the two parameters for the first surface and the two parameters for the second surface is a four-dimensional curve. It has the visualization of being two planar curves (in separate planes) with a common parameterization.

4.2 CURVES FROM POINTS

This section describes the main facilities in the Library for creating curves from information based at points. There are two major classes of problems - interpolating and approximating. An interpolating curve goes through each point. An approximating curve comes as close to each point as possible while satisfying other restrictions. When conditions are chosen just right, a tool designed to produce an approximating curve will produce an interpolating curve, since that is certainly a curve which comes as close as possible to each point, other restrictions permitting.

In the simplest case, a sequence of points in some (n-dimensional) space has been given and an interpolating curve is desired. That problem has infinitely many solutions, so one must be more particular. For computational simplicity, restrict attention to spline curves. There are still infinitely many possibilities, so a series of additional constraints must be imposed to narrow the possible solutions to one curve that can easily be computed. Ideally, the constraints should be derived from a deeper understanding of the application. In practice, the choices are much restricted by the availability of tools and techniques to apply them. Traditionally, constraints broadly involving smoothness and constraints on the parameterization are supported.

Smoothness means "number of continuous derivatives". The individual pieces of a spline are polynomial or rational, and are therefore infinitely smooth (except at zeroes of the denominator in the rational case, but most systems or applications require the denominators to be uniformly positive.) So smoothness constraints mainly involve what happens at the breakpoints between pieces. Choosing the overall degree (equivalently, order) of the spline immediately limits the possible smoothness at the breakpoints to at most one less than the degree. To see this, consider a breakpoint between two cubic polynomial pieces that had three continuous derivatives at that point. That would mean that the function value and its first three derivatives match on each side of the point. But those four values completely determine a cubic polynomial. Hence, both pieces are exactly the same polynomial, and the breakpoint is purely nominal - there is no freedom to "change course" in any way.

Consequently, choosing the overall smoothness takes the form of choosing the degree of the spline. A spline of degree one produces a piecewise linear curve, that is, a series of straight line segments, joined end to end. This is a simple, continuous curve, but its first derivative is discontinuous. A spline of degree two produces a piecewise conic curve, that is, a series of conic arcs, joined end to end, and normally with a common tangent at the joints. Now the curve is continuous and its first derivative is continuous, but the second derivative is not. A spline of degree three can produce a curve with continuous second derivative, and so on. As smoothness increases, a price may be paid in another quarter - the interpolating curve may become wavy, eventually oscillating wildly beyond the initial data points. Space and time costs also increase steadily. Hence, the optimum smoothness for most purposes is somewhere near degree three.

In some applications, there may be particular points where smoothness is deliberately reduced below the natural limit to achieve other effects - for example, a sharp corner, or a specified tangent, or monotonicity.

Another set of constraints concerns the parameterization. What parameter values should be assigned to each of the data points? These selections have subtle effects on the resulting curve, and the matter is still a subject of research. The Library provides tools for computing the three most common parameterizations - chord length (D2GPAR, DTGPAR), uniform (TBD), and centripetal (TBD). If one has no better idea, try chord length first, examine the resulting curve under a microscope, and if it isn't satisfactory, try the others.

The final set of constraints concerns placement of the knots, that is, the places where the curve is permitted to switch from one polynomial to another. If too many knots are provided, this decision acquires a second part, which involves specifying additional conditions to be satisfied by the curve. There are many complex constraints on these choices and most users prefer to avoid them. Avoidance takes the form of choosing a tool which applies some fixed strategy for placing the knots and selecting additional conditions in a way which causes the curve interpolation problem to have a single solution.

A popular solution is called "natural" spline interpolation. It places single knots at each data point for odd degree curves and midway between data points for even degree curves. This is "too many" knots for a unique solution.[3] A single solution is obtained by requiring an appropriate number of the second and higher derivatives to be zero at the end points. The Library implements this strategy in D2NSI (and DTNSI).

A variation on natural spline interpolation is complete spline interpolation. In this variation, the user may specify values for some of the derivatives at the endpoints of the curve. This strategy is implemented in D2CSI (and DTCSI).

When the point set is appropriate and the interpolating curve is intended to be a closed curve, that is, returning "smoothly" to its starting point, a "periodic" cubic spline may be created using D2PCS1.

[3] To understand why this is "too many", see Chapter 2 for an introduction to spline theory, and then see the spline theory article in the DT_NURBS Theory Document.

When both points and tangent slopes at each point are provided, Hermite spline interpolation should be used. The strategy used here places a double knot at each point. Hermite spline interpolation is implemented by D2HSI (and DTHSI). (Note how the addition of tangent constraints reduces the smoothness.)

The most general-purpose curve generator is the Least-Squares Approximant. In this case the user takes full control and supplies the knots as an input. With appropriate choices of knots, the resulting curve is actually interpolating. Otherwise, the algorithm finds the curve which minimizes the sum of the squared distances from the curve to the given data points. The Library routine is named D2LSA (and DTLSA).

4.3 CURVES FROM CURVES

This section describes the main facilities in the Library for creating curves from other curves. The situations in which a new curve is created from an old one are quite varied.

A segment of a spline curve can be turned into spline curve in its own right by calling the "spline trim" subroutine D2STRM (or DTSTRM).

Two spline curves that share an end point can be converted into a single spline curve by the "spline join" subroutine D2SPJN (or DTSPJN).

The curve representing the derivative of a spline curve can be created by D2SPAD (or DTSPAD).

Curves created by offset from a given curve can be created by TBD.

4.4 CURVES FROM SURFACES

This section describes the main facilities in the Library for creating curves from surfaces.

"Constant parameter" curves are obtained by fixing the value of one of the two parameters of the surface and letting the other be the parameter of the curve. Library routines D2CNPR (and DTCNPR) perform this operation. It can also be achieved by trimming one parameter to a single point using D2STRM (or DTSTRM).

Intersection of a surface with a plane to obtain intersection curves in the parameter space of the surface can be accomplished using D2PCUT (or DTPCUT).

Intersection of two spline surfaces, obtaining the four-dimensional "intersection" curve consisting of two planar curves in the two surfaces' parameter domains, can be accomplished by D2SSXT (or DTSSXT).

4.5 EXAMPLE USING D2LSA

Suppose one is given a sequence of points from the cross-section of a wing and the task is to construct a curve interpolating these points. Suppose the points start at the trailing edge, go along the upper surface to a known point on the leading edge, then return along the lower surface to end at the trailing edge point from which they began. Suppose further that this wing is to have a "sharp" leading edge as well as a sharp trailing edge, but otherwise be fairly smooth. The most reasonable translation of this last requirement is that a degree three (cubic) spline is required across the top and bottom (second derivatives continuous), but the first derivative should be discontinuous at the leading and trailing edges. The natural spline interpolant could produce a sharp trailing edge merely by starting the curve there and ending there, but it is deliberately designed to prevent such discontinuities in the middle of the curve. One solution would be to use the natural spline interpolant (D2NSI) to fit the upper curve, then the lower curve, then join them (D2SPJN) at the leading edge point. A more elegant and efficient solution is to use the least-squares approximant (D2LSA) in a way which produces interpolation and the desired derivative discontinuity all at once.

To permit a discontinuity in the first derivative at a point in a cubic spline, one must place a triple knot there. (See chapter two for further explanation.) The end points of a cubic spline are automatically quadruple knots. To maintain smoothness elsewhere, the remainder of the knots should be single knots. To guarantee interpolation, the number of interior knots for a cubic spline must be the number of interior data points minus two. Furthermore, these knots must be distributed in a way which satisfies the "interlacing" conditions. Qualitatively, the interlacing conditions require that the knots be distributed roughly evenly among the data points. If a knot is placed at each data point and a triple knot is placed at the leading edge point, there will be four knots too many. If the knots at the points next to the end points and at the points on either side of the leading edge point are omitted, the correct number is attained and the interlacing conditions are satisfied.

The following subroutine implements this plan. In the interest of brevity, all the IER checking and error handling that should occur after each call to a Library routine has been omitted. Where error handling is required, a user-supplied subroutine named "FAILED" is assumed to take care of it.

```
      SUBROUTINE EXMPL7 (CMEM, IMEM, DMEM, IDMPTS, JLEAD, IBFSEC, IER)
C  Create airfoil section curve with a sharp leading edge from given points
C  and given leading edge point.
C
C  Inputs:
C     IDMPTS  Pointer to DM entity whose rows are the points, starting from
C             the trailing edge, over the top surface to the leading edge,
C             and back along the bottom surface to the same trailing edge
C             point.  (3D pts)
C     JLEAD   Row index of leading edge point in IDMPTS
C  Outputs:
C     IBFSEC  Pointer to BF entity expressing the airfoil section curve
C     IER     Error code (Zero indicates no error.)
C
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IMPTS, JLEAD, IBFSEC, IER
      DOUBLE PRECISION DMEM(*)
C
      INTEGER NPTS, NDEP, NDEG, NKNOTS, IFAIL, JDMPTS, IDAPAR, JDAPAR
      INTEGER JDUM, IDAKNT, JDAKNT
C
C     Obtain size parameters from size of points matrix
      CALL D2MSZ (CMEM, IMEM, DMEM, IDMPTS, 'D', NPTS, NDEP, IER)
C     Verify JLEAD has a reasonable value
      IF (JLEAD .LT. 4 .OR. JLEAD .GT. NPTS-3) THEN
         CALL FAILED ("EXMPL7 - JLEAD or NPTS out of range",-2,IER)
         RETURN
      ENDIF
C
C     Allocate space for the parameterization of the points
      CALL D2ADF (CMEM, IMEM, DMEM, 'D', NPTS, IDAPAR, IER)
C     Find the absolute location in DMEM of the parameter array
      CALL D2FEBD (CMEM, IMEM, DMEM, IDAPAR, JDAPAR, JDUM, IER)
C     Find the absolute location in DMEM of the points array
      CALL D2FEBD (CMEM, IMEM, DMEM, IDMPTS, JDMPTS, JDUM, IER)
C     Construct the normalized chord length parameterization
      CALL D2GPAR (CMEM, IMEM, DMEM, NPTS, DMEM(JDMPTS), 1, NDEP, NPTS,
     + DMEM(JDAPAR), IER)
C
C     Next allocate space for the knot vector
      NKNOTS = NPTS - 4
      CALL D2ADF (CMEM, IMEM, DMEM, 'D', NKNOTS, IDAKNT, IER)
C     Get its absolute location in DMEM
      CALL D2FEBD (CMEM, IMEM, DMEM, IDAKNT, JDAKNT, JDUM, IER)
C     Place the interior knots at the interior data points other than the
C     first and last, i.e. at the 3rd to NPTS-2nd data points, then make
C     the knot at the JLEAD point a triple knot while omitting the adjacent
C     knots.
      DO 100 I=0,NKNOTS-1
         DMEM(JDAKNT+I) = DMEM(JDAPAR+I+2)
  100 CONTINUE
      DMEM(JDAKNT+JLEAD-2) = DMEM(JDAKNT+JLEAD-1)
      DMEM(JDAKNT+JLEAD) = DMEM(JDAKNT+JLEAD-1)
C
C     Construct the interpolating cubic curve, using the default of equal
C     weights (NDEG = 3, IWT = 0, WHT = 1.0D0)
      CALL D2LSA (CMEM, IMEM, DMEM, NPTS, DMEM(JDAPAR), DMEM(JDMPTS), NPTS,
     + NDEP, 3, 0, 1.0D0, DMEM(JDAKNT), NKNOTS, IBFSEC, IFAIL, IER)
C
C.....Delete the workspace
      CALL D2ERAS (CMEM, IMEM, DMEM, IDAPAR, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IDAKNT, IER)
C     Unlock the points array
......CALL D2UNLD (CMEM, IMEM, DMEM, IDMPTS, IER)
      RETURN
      END
```

CREATING SURFACES 5

5.1 SURFACES

By definition, a surface is a function from a domain in R^2 into R^n. For the "tensor product spline surfaces" which are the basic building block of all the surfaces the Library works with, the domain is always a rectangle whose sides are parallel to the axes. Most commonly, the image space dimension, *n*, is three, producing a surface situated in three-dimensional space. However, an *n* of two, indicating a mapping from a rectangle into another patch of a plane, arises naturally in the so-called subrange surface, usually used to identify and parameterize a curve-bounded subset of the parameter domain of an ordinary surface. The composition of the subrange surface and the ordinary surface then serves to select a "trimmed" portion of the ordinary surface image in space. When *n* is one, it is customary to include the surface parameters as the *x* and *y* coordinates and picture the function graph (i.e. the points (x,y,z) such that $z = f(x,y)$) as a "surface" in three-dimensional space, although, in fact, the image of the function lies in a one-dimensional space. Cases where *n* is greater than three arise by including additional variables (e.g. pressure, temperature, velocity vector components) along with geometric position as outputs of the function.

5.2 SURFACES FROM POINTS

This section describes the main facilities in the Library for creating surfaces from points. There are two major classes of problems - interpolating and approximating. An interpolating surface goes through each point. An approximating surface comes as close to each point as possible while satisfying other restrictions. When conditions are chosen just right, a tool designed to produce an approximating surface will produce an interpolating surface, since that is certainly a surface which comes as close as possible to each point, other restrictions permitting.

The creation of a tensor product spline surface is much simpler and more efficient if the data points are arranged in a a topologically rectangular grid, where the rows are assumed to occur at places where one parameter is constant and the columns occur at places where the other parameter is constant. Fortunately, this is very often the case.

The remainder of the considerations in building a spline surface are quite similar to those described for curves in section 4.2, except that there are now two parameters to consider. In particular, a degree, data parameter values, and a knot vector must be chosen for each parameter. Again, the easiest course is to use a tool which chooses satisfactory knot vectors automatically.

One popular solution is a generalization of "natural" spline interpolation for curves. The Library subroutine D2GNSI (or DTGNSI) implements this, both for surfaces and for three-, four-, or higher-parameter functions.

Similarly, complete spline interpolation for multi-parameter functions, most often surfaces, is implemented in D2GCSI (or DTGCSI).

When the point set is appropriate and the interpolating surface is intended to be closed in one direction, that is, returning "smoothly" to its starting edge in that direction, a "periodic" cubic spline surface may be created using D2PCS2.

The most general-purpose surface generator is the General Tensor Product Approximant, D2GTPA (or DTGTPA), which is the higher-dimensional analogue to the Least-Squares Approximant for curves. In this case the user takes full control and supplies the knots as an input. With appropriate choices of knots, the resulting surface is actually interpolating. Otherwise, the algorithm finds the surface which minimizes the sum of the squared distances from the surface to the given data points.

5.3 SURFACES FROM CURVES

This section describes the main facilities in the Library for creating surfaces from curves.

The simplest case is the generation of a surface of revolution from a curve and an axis. The Library routine which does this is D2SREV (or DTSREV).

A subrange surface can be constructed from four boundary curves using D2MPBC, or from four points using D2MPQD.

A more interesting challenge is to "blend" a family of roughly parallel curves into a surface. The actual surface construction is performed by D2CRBL (or DTCRBL), but it only works on curves with the same degree and knot vector. Additional routines to modify the structure of the input curves, without changing their spatial positions or shapes, to achieve identical degrees and knot vectors are D2UPDG (or DTUPDG), which increases the apparent degree of a curve, and D2MGKT (or DTMGKT), which adds knots to the members of a family of curves until all members have the same knot vector.

A variation on curve blending is grid blending, where two families of curves intersecting in a consistent grid pattern are interpolated by a surface. The Library routine which implements this is D2GRBL (or DTGRBL).

5.4 SURFACES FROM SURFACES

This section describes the main facilities in the Library for creating surfaces from other surfaces. The situations in which a new surface may be created from an old one are quite varied.

A portion of a spline surface defined by further restricting the two parameter intervals can be turned into a spline surface in its own right by calling the "spline trim" subroutine D2STRM (or DTSTRM).

A mirror image of a surface in a plane in space can be produced by D2MIRI (or DTMIRI).

A surface can be rotated, scaled, and/or translated by combining it with the appropriate linear or affine mapping using D2BINE. The matrix representing the mapping can be created using D2TROT, D2TSCL, or D2TTRN (or DTTROT, DTTSCL or DTTTRN).

Surfaces created by offset from a given surface can be created by TBD.

5.5 EXAMPLE USING D2CRBL

Suppose a family of airfoil section curves has been provided, and the task is to construct the wing surface interpolating between them. The following code fragment illustrates using D2UPDG, D2MGKT and D2CRBL to accomplish this.

```
      SUBROUTINE EXMPL8 (CMEM, IMEM, DMEM, IPASEC, IDAPAR, NDEG, IBFSUR,
     +                   IER)
C     Blend a list of airfoil sections at given parameter locations into a
C     wing surface.
C
C  Inputs:
C     IPASEC  Pointer array whose pointers point to the BF curve entities
C             representing the wing sections, in order spanwise.
C     IDAPAR  Pointer to array of parameter locations for the curves in
C             IPASEC.
C     NDEG    Degree to use in spanwise direction of wing
C  Outputs:
C     IBFSUR  Pointer to BF entity representing the wing surface
C     IER     Error code  (Zero means no error)
C
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IPASEC, IDAPAR, NDEG, IBFSUR, IER
      DOUBLE PRECISION DMEM(*)
C
      INTEGER NCUR, IPA, KORD, JDUM, JDAPAR, IBFSEC, IBFCHG, JBFCHG, IPACHG
      INTEGER I, J, K, N, M, NC
      DOUBLE PRECISION CLO, CHI, PLO, PHI, ORDER
      LOGICAL SAMDEG, R
C
C     Get number of curves
      CALL D2ASZ (CMEM, IMEM, DMEM, IPASEC, 'P', NCUR, IER)
C
C     Find maximum curve order and obtain first curve's parameter range
      CALL D2AETI (CMEM, IMEM, DMEM, IPASEC, 1, '>' IBFSEC, IER)
      CALL D2BFSZ (CMEM, IMEM, DMEM, IBFSEC, 1, N, M, R, KORD, NC, CLO,
     +             CHI, IER)
      DO 100 I=2,NCUR
         CALL D2AETI (CMEM, IMEM, DMEM, IPASEC, I, '>', IBFSEC, IER)
C        Take advantage of the fact that the order of a curve is always the
C        third element of its spline array
         CALL D2FEED (CMEM, IMEM, DMEM, IBFSEC, 3, ORDER, IER)
         IF (INT (ORDER) .GT. KORD)  KORD = INT (ORDER)
  100 CONTINUE
C
C     Make a copy of the input curve list so that its elements may be
C     processed without corrupting the original input data.
C     Allocate a new list
      CALL D2ADF (CMEM, IMEM, DMEM, 'P', NCUR, IPA, IER)
C     Copy the old list, adjusting the parameter domains and orders as
C     necessary
      DO 200 I=1,NCUR
         CALL D2AETI (CMEM, IMEM, DMEM, IPASEC, I, '>', IBFSEC, IER)
         CALL D2BFSZ (CMEM, IMEM, DMEM, IBFSEC, 1, N, M, R, K, NC, PLO,
     +                PHI, IER)
         IF (K .LT. KORD) THEN
C           Increase the order until it matches the maximum among all the
C           curves.  This process also creates a new copy of the curve
```

```
            DO 210 J=K+1,KORD
               CALL D2UPDG (CMEM, IMEM, DMEM, IBFSEC, 1, IBFCHG, IER)
C              Clean up by erasing the previous copy, if not the original
               IF (J .GT. K+1)  CALL D2ERAS (CMEM, IMEM, DMEM, IBFSEC, IER)
               IF (J .LT. KORD) IBFSEC = IBFCHG
  210       CONTINUE
         ELSE
C         Copy the old curve, ignoring any variable labels that may be
C         present
            CALL D2FELD (CMEM, IMEM, DMEM, IBFSEC, M, IER)
            CALL D2BFDF (CMEM, IMEM, DMEM, M, '', IBFCHG, IER)
            CALL D2FEBD (CMEM, IMEM, DMEM, IBFCHG, JBFCHG, JDUM, IER)
            CALL D2FEAD (CMEM, IMEM, DMEM, IBFSEC, 1, M, DMEM(JBFCHG), IER)
            CALL D2UNLD (CMEM, IMEM, DMEM, IBFCHG, IER)
         ENDIF
C        Fix the parameter bounds, if necessary
         IF (PLO .NE. CLO .OR. PHI .NE. CHI)
     +    CALL D2RPRM (CMEM, IMEM, DMEM, IBFCHG, 1, CLO, CHI, IER)
C        Store the copy in the new list
         CALL D2AETI (CMEM, IMEM, DMEM, IPA, I, '<', IBFCHG, IER)
  200 CONTINUE
C
C     Make the knot vectors match
      CALL D2MGKT (CMEM, IMEM, DMEM, IPA, 0.0D0, 0, 0, IPACHG, IER)
C
C     Erase the intermediate local copy of the input curves
      DO 300 I=1,NCUR
         CALL D2AETI (CMEM, IMEM, DMEM, IPA, I, '>', IBFCHG, IER)
         CALL D2ERAS (CMEM, IMEM, DMEM, IBFCHG, IER)
  300 CONTINUE
      CALL D2ERAS (CMEM, IMEM, DMEM, IPA, IER)
C
C     Create the wing surface by blending the processed local copy of the
C     airfoil section curves
      CALL D2FEBD (CMEM, IMEM, DMEM, IDAPAR, JDAPAR, JDUM, IER)
      CALL D2CRBL (CMEM, IMEM, DMEM, IPACHG, DMEM(JDAPAR), 1, NDEG, IBFSUR,
IER)
      CALL D2UNLD (CMEM, IMEM, DMEM, IDAPAR, IER)
C
C     Erase the local copy of the input curves
      DO 400 I=1,NCUR
         CALL D2AETI (CMEM, IMEM, DMEM, IPACHG, I, '>', IBFCHG, IER)
         CALL D2ERAS (CMEM, IMEM, DMEM, IBFCHG, IER)
  400 CONTINUE
      CALL D2ERAS (CMEM, IMEM, DMEM, IPACHG, IER)
C
......RETURN
      END
```

NOTES ON INTERSECTING SURFACES 6

6.1 SURFACE - SURFACE INTERSECTION

The Phase 1 subroutine for surface-surface intersection, DTSSI, required the user to supply a starting point and constructed three lists of points (one in model space and one in each surface's parameter space) for the one intersection curve nearest that starting point. (An intersection curve, since it lies in both surfaces, is most usefully described in the form of two planar curves, one in each surface's parameter space.) The Phase 2 subroutine for surface-surface intersection, DTSSXT, is a far more capable routine. It attempts to find all components of the intersection of two surface patches without user guidance. It has no difficulty with loop detection, which is often considered a special problem in the literature. DTSSXT has the unique feature that it describes the intersection curves in the four-dimensional Cartesian product of the two surface parameter spaces. This amounts to providing a simultaneous parameterization of the corresponding pair of planar curves. Although cases are known in which DTSSXT fails (e.g., crossing intersection curves), it appears to be state-of-the-art at this time. DTSSXT does require that both surfaces be continuously differentiable.

The "contour refining" component of DTSSXT is sufficiently general that it has been made available to users directly as DTCNTR, which is also described below.

In version 2.4 of the Library, D2SSXT replaced DTSSXT as the recommended surface intersection routine. In addition to using the dynamic memory facility to allocate all required workspaces and the solution space, D2SSXT incorporates a better algorithm for solving the various subsidiary systems of nonlinear equations that arise in the DTSSXT intersection algorithm. This improvement eliminates the problematic PROB parameter of DTSSXT as well as greatly increasing the likelihood that all solutions will be found.

6.2 USAGE OF DTCNTR FOR SURFACE - SURFACE INTERSECTION

If DTSSXT fails to find an intersection curve, but you know where the curve starts and ends, then DTCNTR can be used to find the curve. To do so, your two surfaces must be defined by spline arrays C and D as in DTCNTR. Set up the AUX array as follows: AUX(1) should be MC, the length of the array C. C should be copied into AUX starting at AUX(2), and D should be copied into AUX starting at AUX(MC+2). You don't need to create FCN; you can use DTSS09, which is a lower level subprogram for DTSSXT. To get the input form of CT, use DTLSA as follows (note that WHT and XKNOTS are dummy arguments):

```
      PARAMETER ( N = 4, NROWY = 4, NCURV = 4, NDEG = 3,
     +       IWT = 0, NKNOTS = 0, MCT = 29, NHOLD = 100 )
      DOUBLE PRECISION T(N), Y(NROWY,NCURV), WHT, XKNOTS, CT(MCT),
     +       HOLD(NHOLD)

      T(1) = 0.D0
      T(2) = 1.D0/3.D0
      T(3) = 2.D0/3.D0
      T(4) = 1.D0
```

Y(1,1) = first parameter value of starting point on first surface
Y(1,2) = second parameter value of starting point on first surface
Y(1,3) = first parameter value of starting point on second surface
Y(1,4) = second parameter value of starting point on second surface

(the fourth row of Y is set up similarly for the ending point)

```
      DO 10 J = 1, 4
            Y(2,J) = ( 2.D0*Y(1,J) +    Y(4,J) ) / 3.D0
            Y(3,J) = (    Y(1,J) + 2.D0*Y(4,J) ) / 3.D0
   10 CONTINUE

      CALL DTLSA (N, T, Y, NROWY, NCURV, NDEG, IWT, WHT,
     +      XKNOTS, NKNOTS,HOLD, NHOLD, MCT, CT, MC, IFAIL,
     +      IER)
```

When AUX and CT are ready, the call to DTCNTR will be

```
      CALL DTCNTR (DTSS09, AUX, TOL, MCT, WORK, NWORK, CT, IER)
```

If you have more information about the curve than its endpoints, you can modify T(2), T(3), and the middle rows of Y accordingly, or you can add additional points.

6.3 USAGE OF D2CNTR FOR SURFACE - SURFACE INTERSECTION

As above, if D2SSXT fails to find an intersection curve, but you know where the curve starts and ends, then D2CNTR can be used to find the curve. The two surfaces must be defined by B-spline function (BF) entities, IBF1 and IBF2. IAUX is a Pointer Array (PA) entity, containing only pointers to IBF1 and IBF2. You don't need to create CNTRFU; you can use D0SS00, which is a lower level subprogram for D2SSXT.

To get the input form of the initial guess (B-spline function IGUESS), use DTLSA as above, to create a spline array CT. Then create an initial guess B-Spline Function (BF) entity IGUESS and copy the spline array into it as follows:

```
      CALL D2BFDF (CMEM, IMEM, DMEM, MCT, LABEL, IGUESS, IER)
      CALL D2STAD (CMEM, IMEM, DMEM, CT, 1, MCT, IGUESS, IER)
```

Create IAUX as follows:

```
      CALL D2ADF (CMEM, IMEM, DMEM, 'P', 2, IAUX, IER)
      CALL D2STEI (CMEM, IMEM, DMEM, IBF1, 1, IAUX, IER)
      CALL D2STEI (CMEM, IMEM, DMEM, IBF2, 2, IAUX, IER)
```

When IAUX and IGUESS are ready, the call to D2CNTR will be

```
      CALL D2CNTR (CMEM, IMEM, DMEM, D0SS00, IAUX, TOL, IGUESS, IER)
```

GEOMETRY, GRIDS AND ANALYSIS 7

7.1 INTRODUCTION

The purpose of the Geometry, Grid & Analysis (GGA) object is to provide a common mathematical framework and a convenient structure for managing geometry, grid, and analysis data. It is based on an hierarchical object architecture. In general, it will:

1. Store geometry, grid, and analysis data in a compact yet flexible form.
2. Store entities so that the child changes as a parent object changes. This would allow, for instance, grids to easily reflect changes in geometry.
3. Provide a method of storing analysis results with geometry.
4. Enable standard methods for extracting information such as Cartesian coordinates, surface area and solid volume, curvatures and normals, and analysis solutions.
5. Allow transparent handling of many independent and dependent variables. This will allow, for example, geometry, grid, and analysis data to vary with time without changing the number of objects in the data base.

The GGA object is a mathematical framework that provides a consistent format for storing geometry data, grid data, and analysis data. The basic idea is to relate all three forms of data to a common parameterization and to allow the parameterization to have as many independent variables as is needed for description. Before discussing each item in detail consider some of the problems that occur using traditional formats.

First, consider geometry. In most CAD systems the basic building block is a single "patch" described as a polynomial or piecewise polynomial surface. But, in many, if not most, industrial applications the geometry of interest is actually only a piece of the defined geometry. For example, for many applications the only part of a ship's hull needed for analysis is the wetted portion, i.e., the portion below the waterline. Also, the portion of interest may change over time. For example, the waterline may change as the ship executes various maneuvers or the sea conditions change. Dealing with changing geometries is difficult. The GGA object is designed to permit the "parent" geometry to be stored along with the relevant, time varying "child" geometry.

Grids present similar problems. If grids and grid lines are described directly on the geometry then, as the geometry changes, grids will have to be rederived. In the wetted surface example, as the sea conditions change grid points will no longer be on the wetted surface while new grid points will be required. This type of change needs to be handled but the typical geometry system does not have facilities for doing so. The GGA object is designed to store a basic underlying grid on the parametric domain of the "child" geometry which then allows the grid to change automatically as the geometry changes.

Finally, the results of computational analysis are usually given at various discrete points, the grid points, on the geometry. Interpolating results to different grids as required in multi-disciplinary applications is difficult. The GGA object is designed to permit the analysis results to be identified with points in the parametric domain of the geometry. Then, the multi-dimensional capability of the DT_NURBS spline type can be used to construct interpolatory or approximating surfaces and store them directly with the grid and geometry.

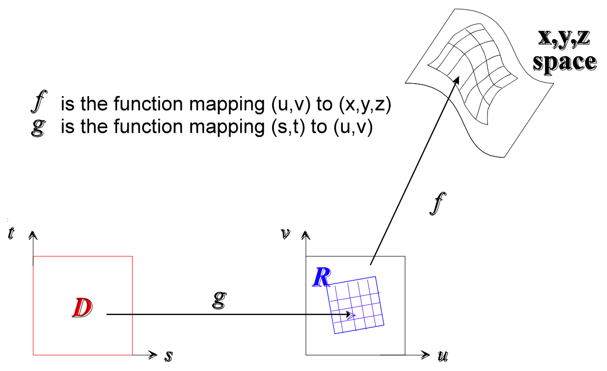
The building blocks of a GGA object are tensor product splines of any dimensionality, compositions of functions, and grids. Parent geometry may be any composition of functions. Child geometries are obtained by composing subrange functions with parent geometries. Subrange functions consist of some of the dependent variables of a tensor product spline. The remaining dependent variables are analysis results.

7.2 GEOMETRY

Geometry is described by two functions. The first, parent geometry, can be any entity of type Composition of Functions (CF). The second, subrange function, is part of a tensor product spline entity (BF).

The domain of the parent geometry is an n-dimensional cube *R*. It will generally be referred to as (*u*,*v*) parameter space. The domain of the subrange function is another *n*-dimensional cube *D and* will be referred to as (*s*,*t*) parameter space. An example of a third dimension in *D* would be the use of a time parameter used to describe the boundary of relevant geometry on the parent surface as it changes with time.

The model of geometry is depicted in the following diagram where the mapping $f\colon R \rightarrow E^3$ space is the parent geometry of the particular joined surface component and $g\colon D \rightarrow R$ is a subrange mapping defining the relevant piece of the parent geometry.


x,y,z space
f is the function mapping (u,v) to (x,y,z)
g is the function mapping (s,t) to (u,v)
f
g
D
R
t
s
v
u

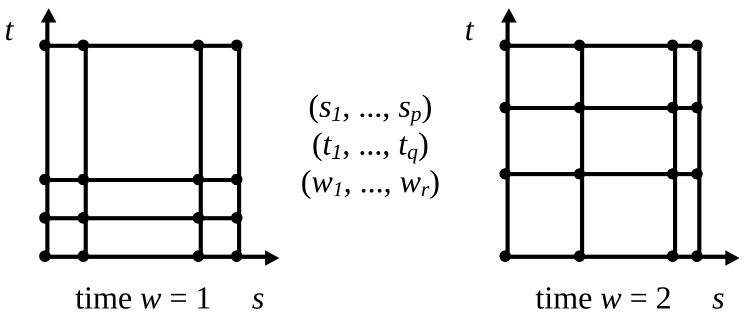
7.3 GRIDS

Grids are logical descriptions of particular geometric points together with connectivity relationships such as which grid points form the vertices of a cell and so forth. Grids are data structures that describe a cellular decomposition of the domain *D* of the subrange function *g*. The reason for this is to allow geometry to change while holding the description of the geometric points constant. This allows those points to change with changes in the geometry without requiring continual regridding of the geometry. We use the notation $\{...\} \xrightarrow[grid]{} D$ to mean the mapping from the grid data to specific points in the domain *D* of the function *g*.

Initially, grids will be provided in one of the following three formats.

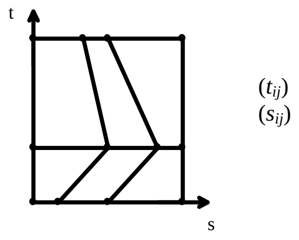
1. Rectangular Grid (RG) described by two or more vectors. For example, the vectors $(s_1, ..., s_p)$ and $(t_1, ..., t_q)$ describe a grid in the two-dimensional domain *D*. For this grid, the points in *D* are $P_{ij} = (s_i, t_j)$ and the elements are the rectangles with vertices P_{ij}, $P_{(i+1)j}$, $P_{(i+1)(j+1)}$, $P_{i(j+1)}$.

In the example, if a third dimension *w* such as time and a vector $(w_1, .., w_r)$ are added then the points in *D* are $P_{ijk} = (s_i, t_j, w_k)$.

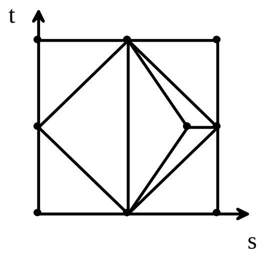


1. Topologically Rectangular Grid (TRG) described by two (or more) matrices each of dimension *p* x *q*. For this grid, the points in *D* are $P_{ij} = (s_{ij}, t_{ij})$ and the elements are the rectangles with vertices P_{ij}, $P_{(i+1)j}$, $P_{(i+1)(j+1)}$, $P_{i(j+1)}$. See previous comments about other parameters such as time. See previous comments about other parameters such as time.

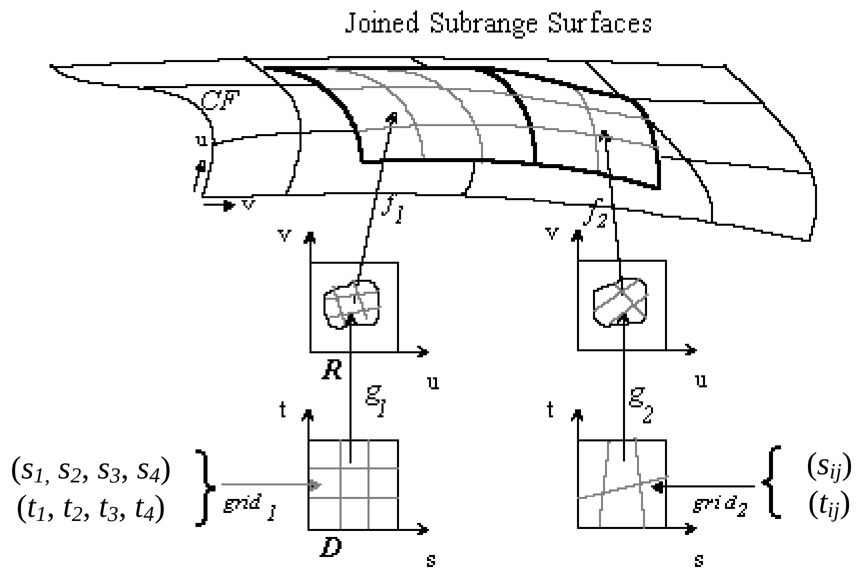
For example, the matrices (t_{ij}), (s_{ij}) describe a grid in a two-dimensional domain *D*.



1. Triangular (or unstructured) Grid (TG) described by a collection P_i of vertices of triangles in *D* together with a list of which vertices are connected in a triangle.



The following diagram shows how the grids, together with the joined surfaces are used to generate geometric points for use in an analysis code.



7.4 ANALYSIS

The last component is the analysis component. Analysis results are stored as an extension of the range of the subrange map *g*. Initially, the subrange maps D → R. After completing the analysis, the range of *g* will be extended as $g: D \rightarrow (u, v, ..., a_1, a_2, ...)$ where the first several dependent components *u*, *v*, ... coordinates in *R* and the last several components a_1, ... are the results of various analyses.

For example, the analysis results could be coefficients of pressure C_P. The subrange function *g* is then extended so that $g:(s,t) \rightarrow (u,v,C_P)$ where C_P is the coefficient of pressure at the point $(x,y,z) = f(u,v)$.

The spline coefficients for the analysis results are normally obtained as follows. After an analysis is completed results are available at each of the geometric points and each geometric point is associated with a particular (u, v) point in the domain *D*. Thus, there is a mapping from points in *D* to results. DT_NURBS functionality then can be used to generate a spline fit, either an interpolation or a smoothing, to the data and that spline fit becomes the variable a_i.

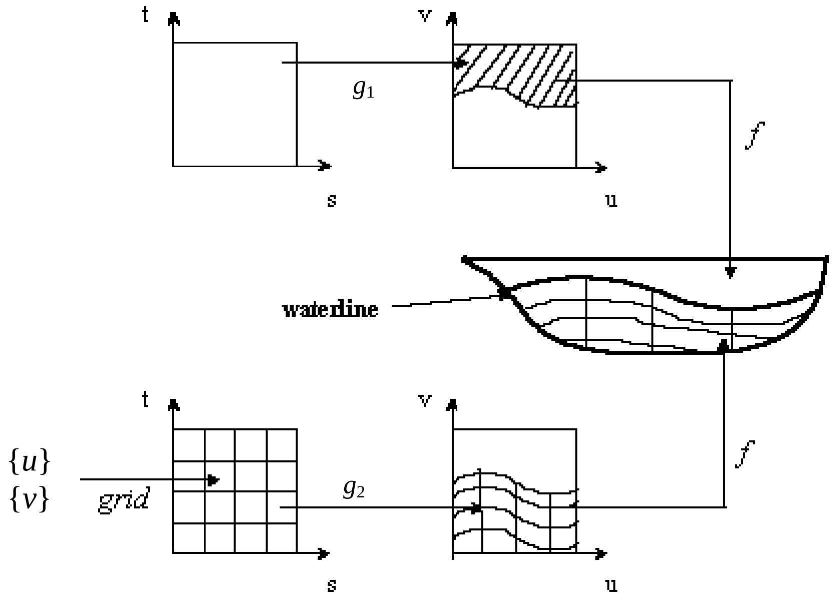
7.5 SUMMARY

The structure for each piece of the joined surface is in the form of a triple (*f*, *g*, *grid*) where *f* describes the parent geometry, *g* describes both the relevant subset of geometry (the subrange) and the results of analysis, and *grid* describes the various grids involved in generating the analyses.

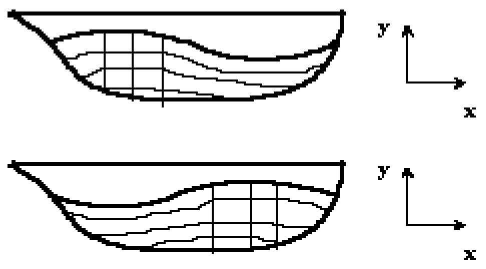
7.6 EXAMPLE

The following example is a good one to keep in mind. Imagine a ship where we want to calculate water speed at points along the ship's wetted surface (i.e., points below the waterline). The geometry consists of the parent geometry *f* describing the entire hull, and two subsurfaces, the wetted surface and the free surface. The boundary between the two subsurfaces is the waterline.

For the free subsurface only the mapping g_1 which defines it is provided and the grid is null. For the wetted surface a mapping g_2 and a grid are provided and the velocity vector (v_x, v_y, v_z) is stored.



If things were static this is all that would be needed. However, we can assume the waterline itself varies with time and, as it varies, different grids are needed. Imagine the situation in the next diagram where the "vertical" grid lines change from being located near the bow to being located near the stern of the hull.



The following data are needed to fully describe the dynamics involved.

I. The parent geometry $f: (u, v) \rightarrow$ hull

II. The free surface

A. A null grid

B. $g_1: (s, t, w) \rightarrow (u, v)$ describing the free surface at time w.

In this case $f \circ g_1 (s, t, w)$[4] is a point which, at time w, is above the waterline.

III. The wetted surface

[4]The notation $f \circ g\ (x)$ means "function f composed with function g," and could be written as $f(g(x))$

A. A 3-dimensional grid in (s, t, w) space (of type RG, TRG or TG)

B. g_2: $(s, t, w) \rightarrow (u, v, v_x, v_y, v_z)$ describing the wetted surface and the analysis results at time w.

Denote by $S(i,j)$ the selection map that chooses the i,j coordinates of a vector. Similarly, $S(i,j,k)$ chooses the i,j,k coordinates. Also, denote by $f \circ g$ the composition of f with g. Then, in the case described, $f \circ S(1,2) \circ g_2\,(s,t,w)$ is a point which, at time w, is below the waterline and $S(3,4,5) \circ g_2\,(s,t,w)$ is the velocity vector at that point and time.

7.7 EXAMPLE PROGRAM

This example demonstrates a simple creation of a Geometry and Analysis (GA) entity. A few evaluations are performed to show how this is done.

In a more realistic program, the splines would have been formed by a data fitting program, which may take a Grid (GR) entity as one of its arguments.

Also, after some evaluations are made, some components of the "*g*" spline may be refit, and merged back into the spline with the D2BFMG subroutine.

```
      PROGRAM GGA
C
C     Example Program to demonstrate the use of the GGA entities
C     and subroutines.
C
C     HISTORY:
C        October 24, 1994      D. Parsons  Created.
C
C     ---------------
C
      EXTERNAL DTJCON
      INTEGER  DTJCON

      INTEGER  LUNIT

      INTEGER MEMAXC, MEMAXI, MEMAXD
      PARAMETER (
     +      MEMAXC = 500,
     +      MEMAXI = 500,
     +      MEMAXD = 500)

      CHARACTER          CMEM*(MEMAXC)
      INTEGER            IMEM(MEMAXI)
      DOUBLE PRECISION   DMEM(MEMAXD)

      INTEGER     IER
      INTEGER     MAXCF, IXCF(3), IGA
      INTEGER     IBFF, IBFG, ICF1, ICF2, ICF3
      INTEGER     JCF
      INTEGER     S1(4), S2(3), S3(5), IS1, IS2, IS3
      INTEGER     LSAPAR, LSAFV, LSADRV, LSAFDV
      INTEGER     JDRV(2)
      DOUBLE PRECISION FV(3), FDV(3), PAR(2)
      CHARACTER   CHKLVL, TYPNAM*(5), CFLBLS*(30)
      CHARACTER   CFNAME*(15)
      DOUBLE PRECISION BFF(36), BFG(74)
      LOGICAL     INITIZ

      DATA CFLBLS /'~Geometry~Temperature~Velocity'/

C     Splines Functions:

C     The following functions would probably be generated by creating
C     a grid entity, and some data arrays and passing these to some
C     user data-fitting program to fit the spline data.  For simplicity
C     this example just uses DATA statements.

C     B-spline function f(u,v) -> (x,y,z)
C                        such that x(u,v) = u
C                                  y(u,v) = v
C                                  z(u,v) = u*v*v

      DATA BFF  /2.D0, 3.D0, 2.D0, 3.D0, 2.D0, 3.D0, 0.D0, 0.D0,
     +           0.D0, 0.D0, 1.D0, 1.D0,
     +           0.D0, 0.D0, 0.D0, 1.D0, 1.D0, 1.D0,
     +           0.D0, 1.D0, 0.D0, 1.D0, 0.D0, 1.D0,
     +           0.D0, 0.D0, .5D0, .5D0, 1.D0, 1.D0,
     +           0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 1.D0/

C     B-spline function g(s,t) -> (u,v,T,Vx,Vy,Vz)
C                        such that u(s,t)  = s
C                                  v(s,t)  = t
C                                  T(s,t)  = s*t
C                                  Vx(s,t) = s*s
C                                  Vy(s,t) = t*t
C                                  Vz(s,t) = 1

      DATA BFG  /2.D0, 6.D0, 3.D0, 3.D0, 3.D0, 3.D0, 0.D0, 0.D0,
     +           0.D0, 0.D0, 0.D0, 1.D0, 1.D0, 1.D0,
     +           0.D0, 0.D0, 0.D0, 1.D0, 1.D0, 1.D0,
     +           0.D0, .5D0, 1.D0, 0.D0, .5D0, 1.D0, 0.D0, .5D0, 1.D0,
     +           0.D0, 0.D0, 0.D0, .5D0, .5D0, .5D0, 1.D0, 1.D0, 1.D0,
     +           0.D0, 0.D0, 0.D0, 0.D0, .25D0,.5D0, 0.D0, .5D0, 1.D0,
     +           0.D0, 0.D0, 1.D0, 0.D0, 0.D0, 1.D0, 0.D0, 0.D0, 1.D0,
     +           0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 1.D0, 1.D0, 1.D0,
     +           1.D0, 1.D0, 1.D0, 1.D0, 1.D0, 1.D0, 1.D0, 1.D0, 1.D0/

C     Select functions:

C     Select (u,v) from g

      DATA S1   /6, 2, 1, 2/

C     Select (T) from g

      DATA S2   /6, 1, 3/

C     Select (Vx,Vy,Vz) from g

      DATA S3   /6, 3, 4, 5, 6/

C     ---------------

      LUNIT  = DTJCON(6)
      TYPNAM = 'DUMMY'
      CHKLVL = 'H'

      INITIZ = .TRUE.

C     Initialize dynamic memory and create entities and parameters

      CALL D2INIT (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, TYPNAM,
     +      'H', IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9910)
         GOTO 8000
      ENDIF

C     Function f(u,v) -> (x,y,z)

      CALL D2BFDF (CMEM, IMEM, DMEM, 36, 'Function f', IBFF, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9920)
         GOTO 8000
      ENDIF

      CALL D2STAD (CMEM, IMEM, DMEM, BFF, 1, 36, IBFF, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9980)
         GOTO 8000
      ENDIF

C     Function g(s,t) -> (u,v,T,Vx,Vy,Vz)

      CALL D2BFDF (CMEM, IMEM, DMEM, 74, 'Function g', IBFG, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9920)
         GOTO 8000
      ENDIF

      CALL D2STAD (CMEM, IMEM, DMEM, BFG, 1, 74, IBFG, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9980)
         GOTO 8000
      ENDIF

C     Select functions

      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 4, IS1, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9950)
         GOTO 8000
      ENDIF

      CALL D2STAI (CMEM, IMEM, DMEM, S1, 1, 4, IS1, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9960)
         GOTO 8000
      ENDIF

      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 3, IS2, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9950)
         GOTO 8000
      ENDIF

      CALL D2STAI (CMEM, IMEM, DMEM, S2, 1, 3, IS2, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9960)
         GOTO 8000
      ENDIF

      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 5, IS3, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9950)
         GOTO 8000
      ENDIF

      CALL D2STAI (CMEM, IMEM, DMEM, S3, 1, 5, IS3, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9960)
         GOTO 8000
      ENDIF


C     Create function compositions

C     f(s1(g(s,t))) -> (x,y,z)  = 'Geometry'

      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBFG, 'S', IS1, ICF1, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9930)
         GOTO 8000
      ENDIF

      CALL D2POSX (CMEM, IMEM, DMEM, ICF1, 'B', IBFF, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9940)
         GOTO 8000
      ENDIF

C     s2(g(s,t)) -> (T)  = 'Temperature'

      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBFG, 'S', IS2, ICF2, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9930)
         GOTO 8000
      ENDIF

C     s3(g(s,t)) -> (Vx,Vy,Vz) = 'Velocity'

      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBFG, 'S', IS3, ICF3, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9930)
         GOTO 8000
      ENDIF

C     Create IGA

      MAXCF   = 3
      IXCF(1) = ICF1
      IXCF(2) = ICF2
      IXCF(3) = ICF3
      CALL D2GADF (CMEM, IMEM, DMEM, MAXCF, CFLBLS, IXCF, IGA, IER)
      IF (IER .NE. 0) THEN
```

```
         WRITE (LUNIT, 9990)
         GOTO 8000
      ENDIF

C     ------------------------------------------------------------------

C     Perform sample evaluations at (.25,.75)

C     Since no fancy use of subarrays for input are needed for this
C     example, just use the dummy setting of zero for the subarray
C     access arrays.

      LSAPAR = 0
      LSADRV = 0
      LSAFV  = 0

      PAR(1) = .25D0
      PAR(2) = .75D0

C     Evaluate "Geometry" (by name).

      CFNAME = 'GEOMETRY'
      CALL D2GAEN (CMEM, IMEM, DMEM, IGA, CFNAME, LSAPAR, PAR,
     +                   LSAFV, FV, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9010)
         GOTO 8000
      ELSE
         WRITE (LUNIT, 9110) PAR(1), PAR(2), FV(1), FV(2), FV(3)
      ENDIF

C     Evaluate the partial derivative of "Temperature" (by name)
C     with respect to the second independent variable (t).

      CFNAME = 'TEMPERATURE'
      JDRV(1) = 0
      JDRV(2) = 1
      CALL D2GADN (CMEM, IMEM, DMEM, IGA, CFNAME, LSADRV, JDRV,
     +                   LSAPAR, PAR, LSAFDV, FDV, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9020)
         GOTO 8000
      ELSE
         WRITE (LUNIT, 9120) PAR(1), PAR(2), FDV(1)
      ENDIF


C     Evaluate "Velocity" using the knowledge that it is the 3rd
C     Composite Function in the structure.

      JCF    = 3
      CALL D2GAEV (CMEM, IMEM, DMEM, IGA, JCF, LSAPAR, PAR,
     +                   LSAFV, FV, IER)
      IF (IER .NE. 0) THEN
         WRITE (LUNIT, 9030)
         GOTO 8000
      ELSE
         WRITE (LUNIT, 9130) PAR(1), PAR(2), FV(1), FV(2), FV(3)
      ENDIF

 8000 CONTINUE
```

```
      WRITE (LUNIT, '(//)')

C     ---  End of Program ---


      STOP

C     ==============================================================

 9010 FORMAT (/,' Unexpected failure of D2GAEN',/)
 9020 FORMAT (/,' Unexpected failure of D2GADN',/)
 9030 FORMAT (/,' Unexpected failure of D2GAEV',/)

 9110 FORMAT (/,' Geometry coordinates (x,y,z) at (',
     +            F4.2,',',F4.2,'):',//,10X,'(',
     +            F5.2,',',F5.2,',',F5.2,')'/)
 9120 FORMAT (/,' Partial of Temperature (T), with respect to t, at (',
     +            F4.2,',',F4.2,'):',//,10X,'(',F5.2,')'/)
 9130 FORMAT (/,' Velocity vectors (Vx,Vy,Vz) at (',
     +            F4.2,',',F4.2,'):',//,10X,'(',
     +            F5.2,',',F5.2,',',F5.2,')'/)

 9910 FORMAT (/,' D2INIT Unable to initalize Dynamic Memory for test',/)
 9920 FORMAT (/,' D2BFDF Unable to create spline',/)
 9930 FORMAT (/,' D2POSE Unable to compose function',/)
 9940 FORMAT (/,' D2POSX Unable to expand CF function',/)
 9950 FORMAT (/,' D2ADF  Unable to create IIA for test',/)
 9960 FORMAT (/,' D2STAI Unable to copy integers to IIA',/)
 9970 FORMAT (/,' D2MDF  Unable to create IDM',/)
 9980 FORMAT (/,' D2STAD Unable to copy dbl. prec. to IDM',/)
 9990 FORMAT (/,' D2GADF Unable to create IGA',/)

      END
```

Program Output

```
Geometry coordinates (x,y,z) at ( .25, .75):

         (  .25,  .75,  .14)


Partial of Temperature (T), with respect to t, at ( .25, .75):

         (  .25)


Velocity vectors (Vx,Vy,Vz) at ( .25, .75):

         (  .06,  .56, 1.00)

Stop - Program terminated.
```

7.8 SECOND EXAMPLE

In this example, a subrange of a surface will be constructed, a grid placed on it, analysis data for the pressure and temperature added to it, and, finally, a constant pressure contour extracted. The central data object in this series of manipulations will be a Geometry and Analysis (GA) entity.

First we give an example of a subroutine which constructs a GA entity with the subrange structure indicated above. Assume IBFSUR is a MEM pointer to the B-spline Function entity representing the base surface. Assume IBFCUT is a MEM pointer to a P-spline Function entity representing a curve cutting the parameter domain of the base surface from the right edge to the left edge, and that the bottom half of this domain is to be gridded and analyzed. Assume FAILED is a user subroutine which handles error message reporting. Assume EQCHK is a user subroutine which decides whether two double precision numbers are close enough to be considered equal.

```
      SUBROUTINE EXMP02 (CMEM, IMEM, DMEM, IBFSUR, IBFCUT, IGA, IER)
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IBFSUR, IBFCUT, IGA, IER
      DOUBLE PRECISION DMEM(*)
C
      EXTERNAL EQCHK
      INTEGER NSURI, NSURO, KSUR(2), NCSUR(2), NCUTI, NCUTO, KCUT
      INTEGER NCCUT, IBFBOT, IBFLFT, IBFRGT, IBFSUB, ICF(3)
      DOUBLE PRECISION SURPLO(2), SURPHI(2), CUTPLO, CUTPHI, VR, VL
      DOUBLE PRECISION DUM, U, PA(2), PB(2)
      LOGICAL RATNL, EQCHK
C
C     Extract parameter range information from surface and cut curve
      CALL D2BFSZ (CMEM, IMEM, DMEM, IBFSUR, 2, NSURI, NSURO, RATNL,
     + KSUR, NCSUR, SURPLO, SURPHI, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 1 to D2BFSZ failed', -1, IER)
         RETURN
      ENDIF
      CALL D2BFSZ (CMEM, IMEM, DMEM, IBFCUT, 1, NCUTI, NCUTO, RATNL,
     + KCUT, NCCUT, CUTPLO, CUTPHI, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 2 to D2BFSZ failed', -2, IER)
         RETURN
      ENDIF
C     Check cutting curve endpoints
      CALL D2EVLS (CMEM, IMEM, DMEM, IBFCUT, CUTPLO, DUM, U, VR, DUM,
     + IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 1 to D2EVLS failed', -3, IER)
         RETURN
      ENDIF
      IF (.NOT. EQCHK(U, SURPHI(1))) THEN
         CALL FAILED ('EXMP02: Cutting curve does not start at right'
     +     // ' edge', -4, IER)
         RETURN
      ENDIF
      CALL D2EVLS (CMEM, IMEM, DMEM, IBFCUT, CUTPHI, DUM, U, VL, DUM,
     + IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 2 to D2EVLS failed', -5, IER)
         RETURN
      ENDIF
      IF (.NOT. EQCHK(U, SURPLO(1))) THEN
         CALL FAILED ('EXMP02: Cutting curve does not end at left edge',
     +     -6, IER)
         RETURN
      ENDIF
C     Construct left, bottom and right edge portions of boundary
```

```
      PA(1) = SURPLO(1)
      PA(2) = VL
      PB(1) = SURPLO(1)
      PB(2) = SURPLO(2)
      CALL D2SLNE (CMEM, IMEM, DMEM, 2, PA, PB, IBFLFT, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 1 to D2SLNE failed', -7, IER)
         RETURN
      ENDIF
      PA(1) = SURPHI(1)
      PA(2) = SURPLO(2)
      CALL D2SLNE (CMEM, IMEM, DMEM, 2, PB, PA, IBFBOT, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 2 to D2SLNE failed', -8, IER)
         RETURN
      ENDIF
      PB(1) = SURPHI(1)
      PB(2) = VR
      CALL D2SLNE (CMEM, IMEM, DMEM, 2, PA, PB, IBFRGT, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call 3 to D2SLNE failed', -9, IER)
         RETURN
      ENDIF
C     Construct subrange mapping
      CALL D2MPBC (CMEM, IMEM, DMEM, IBFBOT, IBFRGT, IBFCUT, IBFLFT,
     + IBFSUB, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call to D2MPBC failed', -10, IER)
         RETURN
      ENDIF
C     Construct the composition of functions expressing the subrange
C     surface
      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBFSUB, 'B', IBFSUR, ICF(1),
     + IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call to D2POSE failed', -11, IER)
         RETURN
      ENDIF
C     Use null pointers for the Pressure and Temperature analysis
C     functions which will be added later
      ICF(2) = 0
      ICF(3) = 0
C     Construct the Geometry and Analysis entity
      CALL D2GADF (CMEM, IMEM, DMEM, 3,'|GEOMETRY|PRESSURE|TEMPERATURE',
     + ICF, IGA, IER)
      IF (IER .LT. 0) THEN
         CALL FAILED ('EXMP02: Call to D2GADF failed', -12, IER)
         RETURN
      ENDIF
      RETURN
      END
```

The next example subroutine shows the generation of the model space grid (on the surface) that corresponds to the input grid on the parameter domain of the subrange surface.

All the usual IER checking and error reporting code will be omitted in the remainder of this example in the interest of brevity and in order to emphasize the main sequence of Library calls.

```
      SUBROUTINE EXMP03 (CMEM, IMEM, DMEM, IGA, IGR, NROWS, G, IER)
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IGA, IGR, NROWS, IER
      DOUBLE PRECISION DMEM(*), G(3,NROWS,*)
C
      INTEGER ENTGR
      PARAMETER (ENTGR=229)
C
      INTEGER I, J, LGR(4), MTYP, ILO, IHI, JLO, JHI, ICF, IERX
      DOUBLE PRECISION U, V
C
C     Check the Grid input and extract the useful information
      CALL D2FEET (CMEM, IMEM, DMEM, IGR, MTYP, IER)
      IF (MTYP .NE. ENTGR) THEN
         CALL FAILED ('EXMP03: IGR is not a grid entity', -1, IER)
         RETURN
      ENDIF
      CALL D2FEAI (CMEM, IMEM, DMEM, IGR, 1, 4, LGR, IER)
      IF (LGR(1) .NE. 1 .OR. LGR(2) .NE. 2) THEN
         CALL FAILED ('EXMP03: Grid is unacceptable type or dimension',
     +    -2, IER)
         RETURN
      ENDIF
C     Locate the DMEM bounds for the grid parameters
      CALL D2FEBD (CMEM, IMEM, DMEM, LGR(3), ILO, IHI, IER)
      IF (IHI - ILO .GE. NROWS) THEN
         CALL FAILED ('EXMP03: Output grid has too few rows', -3, IER)
         CALL D2UNLD (CMEM, IMEM, DMEM, LGR(3), IERX)
         RETURN
      ENDIF
      CALL D2FEBD (CMEM, IMEM, DMEM, LGR(4), JLO, JHI, IER)
C     Extract the geometry function from the GA entity
      CALL D2FEEI (CMEM, IMEM, DMEM, IGA, 3, ICF, IER)
C     Evaluate the geometry function at each grid point, putting the
C     output points in the G array
      DO 200 I=ILO,IHI
         DO 100 J=JLO,JHI
            CALL D2EVL2 (CMEM, IMEM, DMEM, ICF, DMEM(I), DMEM(J), 0,
     +       G(1,I-ILO+1,J-JLO+1), IER)
  100    CONTINUE
  200 CONTINUE
C     Unlock the grid parameter arrays, now that we are done with direct
C     access to their DMEM data
      CALL D2UNLD (CMEM, IMEM, DMEM, LGR(3), IER)
      CALL D2UNLD (CMEM, IMEM, DMEM, LGR(4), IER)
      RETURN
      END
```

We assume some analysis code(s) have been called using the G array as input and producing corresponding P and T arrays containing the computed pressure and temperature at each surface point in G. The next part interpolates the P and T data to produce pressure and temperature functions defined on the subrange domain, and adds these to the Geometry and Analysis entity.

```
      SUBROUTINE EXMP04 (CMEM, IMEM, DMEM, IGA, IGR, NROWS, P, T, IER)
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IGA, IGR, NROWS, IER
      DOUBLE PRECISION DMEM(*), P(NROWS,*), T(NROWS,*)
C
```

```
      INTEGER NMAX, JDM, IDM, IBFP, IBFT, ICFP, ICFT, LGR(4), N(2), J
C
C     Extract the data from the grid entity into the LGR integer array
      CALL D2FEAI (CMEM, IMEM, DMEM, IGR, 1, 4, LGR, IER)
C     Get the lengths of the two parameter lists comprising the grid
      CALL D2ASZ (CMEM, IMEM, DMEM, LGR(3), 'D', N(1), IER)
      CALL D2ASZ (CMEM, IMEM, DMEM, LGR(4), 'D', N(2), IER)
C     Allocate and load a temporary matrix entity to serve as the
C     parameter array input for the surface interpolation routine
      NMAX = MAX (N(1), N(2))
      CALL D2MDF (CMEM, IMEM, DMEM, 'D', NMAX, 2, IDM, IER)
      CALL D2FEBD (CMEM, IMEM, DMEM, IDM, JDM, J, IER)
      CALL D2FEAD (CMEM, IMEM, DMEM, LGR(3), 1, N(1), DMEM(JDM), IER)
      CALL D2FEAD (CMEM, IMEM, DMEM, LGR(4), 1, N(2), DMEM(JDM+NMAX),
     + IER)
C     Construct the Pressure and Temperature analysis functions by
C     interpolating the analysis point data on the grid
      CALL D2GNSI (CMEM, IMEM, DMEM, DMEM(JDM), NMAX, 2, N, P, N, 1, 3,
     + IBFP, IER)
      CALL D2GNSI (CMEM, IMEM, DMEM, DMEM(JDM), NMAX, 2, N, T, N, 1, 3,
     + IBFT, IER)
C     Erase the temporary matrix
      CALL D2ERAS (CMEM, IMEM, DMEM, IDM, IER)
C     Turn the analysis function BF entities into CF entities by
C     composing them with the identity function
      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBFP, 'I', 0, ICFP, IER)
      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBFT, 'I', 0, ICFT, IER)
C     Add the Pressure and Analysis functions to the GA entity
      CALL D2STEI (CMEM, IMEM, DMEM, ICFP, 4, IGA, IER)
      CALL D2STEI (CMEM, IMEM, DMEM, ICFT, 5, IGA, IER)
      RETURN
      END
```

Finally, we find all contours of constant pressure Q in the subrange parameter domain.

```
      SUBROUTINE EXMP05 (CMEM, IMEM, DMEM, IGA, Q, TOL, MXPTS, IPACON,
     + IER)
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IGA, IPACON, IER
      DOUBLE PRECISION DMEM(*), Q
C
      INTEGER IBF, ICF, NCUTS
      DOUBLE PRECISION PLANE(2)
C
C     Extract the pointer to the actual spline function (BF entity)
C     expressing the Pressure
      CALL D2FEEI (CMEM, IMEM, DMEM, IGA, 4, ICF, IER)
      CALL D2FEEI (CMEM, IMEM, DMEM, ICF, 4, IBF, IER)
C     Construct the "plane" in the one-dimensional range space of the
C     Pressure function whose equation is 1*p = Q
      PLANE(1) = 1.0D0
      PLANE(2) = Q
C     Call the planar cut subroutine to find all the curves in the
C     pressure surface where the pressure is the constant Q.  IPACON
C     is an array of pointers to the BFs expressing these curves.
      CALL D2PCUT (CMEM, IMEM, DMEM, IBF, PLANE, TOL, MXPTS, 1, NCUTS,
     + IPACON, IER)
      RETURN
      END
```

IGES FILE MANIPULATION 8

8.1 PROBLEM STATEMENT

Suppose you have been given a large IGES file containing Spline Surfaces (type 128), Spline Curves (type 126), and some other entities, and you need to create a smaller IGES file containing only the DT_NURBS IGES 5001 B-Spline Function entity.

8.2 PROBLEM ANALYSIS

IGES (Initial Graphics Exchange Specification) format is a standard file format for transferring geometry data from one CAD system to another.

The NASA-IGES-NURBS entities are a subset of the IGES entity set, dealing primarily with conics and splines.

The DT_NURBS extensions to IGES and the NASA-IGES-NURBS entity types are described in detail in the DT_NURBS Reference Manual.

Because you are only interested in the Spline data from the IGES file, which happens to be one of the NASA-IGES-NURBS entity types, you should set the LVLI parameter of the D2IGRD (IGes ReaD) subroutine to its lowest value (1), to avoid overfilling your memory arrays with uninteresting entities.

Once the desired spline entity has been located using the D2IGNT (IGes Next of Type) subroutine, the entity needs to be converted to the DT_NURBS B-Spline Function (BF) entity type, using D2IGBF (IGes to B-spline Function). That B-Spline Function entity is now of the form used by the spline evaluators and manipulators that make up most of the DT_NURBS library.

For this simple example, however, the B-Spline Function is simply converted to the DT_NURBS IGES extension type 5001, using the subroutine D2BFDI (B-spline Function to Dt Iges). D2IGDI (IGes Define Index) is called first, to define a new IGES index, representing the new IGES file. After the conversion, this internal IGES file is written out to a new physical IGES file by D2IGWR (IGes WRite).

8.3 EXAMPLE PROGRAM

<u>Input IGES File</u>

```
                                                                        S      1
1H,, 1H;, 1H , 1H , 1H ,12HDT_NURBS 1.0,32,38,7,307,16, 1H ,            G0000001
 .10000000E+01,1, 2HIN,1,  .33330000E-02, 1H ,  .10000000E-05,          G0000002
 .10000000E+06, 1H , 1H ,9,0, 1H ;                                      G0000003
     143       1       0       0       0       0       0       000000000D0000001
     143       0       0       1       0               TS              0D0000002
     128       2       0       0       0       0       0       000010000D0000003
     128       0       0       8       0               S               0D0000004
     141      10       0       0       0       0       0       000010000D0000005
     141       0       0       1       0                LP             0D0000006
     126      11       0       0       0       0       0       000010000D0000007
     126       0       0       5       0                               0D0000008
     126      16       0       0       0       0       0       000010500D0000009
     126       0       0       5       0                E              1D0000010
     126      21       0       0       0       0       0       000010000D0000011
     126       0       0       5       0                               0D0000012
     126      26       0       0       0       0       0       000010500D0000013
     126       0       0       5       0                E              2D0000014
     126      31       0       0       0       0       0       000010000D0000015
     126       0       0       5       0                               0D0000016
     126      36       0       0       0       0       0       000010500D0000017
     126       0       0       5       0                E              3D0000018
     126      41       0       0       0       0       0       000010000D0000019
     126       0       0       5       0                               0D0000020
     126      46       0       0       0       0       0       000010500D0000021
     126       0       0       5       0                E              4D0000022
143,1,3,1,5;                                                    0000001P0000001
128,1,1,1,1,0,0,1,0,0,  .00000000E+00,  .00000000E+00,          0000003P0000002
  .10000000E+01,  .10000000E+01,  .00000000E+00,  .00000000E+00, 0000003P0000003
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .10000000E+01, 0000003P0000004
  .10000000E+01,  .10000000E+01,  .00000000E+00,  .00000000E+00, 0000003P0000005
  .00000000E+00,  .10000000E+01,  .00000000E+00,  .10000000E+01, 0000003P0000006
  .00000000E+00,  .10000000E+01,  .10000000E+01,  .10000000E+01, 0000003P0000007
  .10000000E+01,  .20000000E+01,  .00000000E+00,  .10000000E+01, 0000003P0000008
  .00000000E+00,  .10000000E+01;                                 0000003P0000009
141,1,2,3,4,7,1,1,9,11,1,1,13,15,1,1,17,19,1,1,21;              0000005P0000010
126,1,1,0,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000007P0000011
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .50000000E+00, 0000007P0000012
 -.24825342E-16,  .50000000E+00,  .10000000E+01,  .50000000E+00, 0000007P0000013
  .15000000E+01,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000007P0000014
  .00000000E+00,  .00000000E+00;                                 0000007P0000015
126,1,1,1,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000009P0000016
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .50000000E+00, 0000009P0000017
  .00000000E+00,  .00000000E+00,  .10000000E+01,  .50000000E+00, 0000009P0000018
  .00000000E+00,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000009P0000019
  .00000000E+00,  .10000000E+01;                                 0000009P0000020
126,1,1,0,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000011P0000021
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .10000000E+01, 0000011P0000022
  .50000000E+00,  .15000000E+01,  .50000000E+00,  .10000000E+01, 0000011P0000023
  .15000000E+01,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000011P0000024
  .00000000E+00,  .00000000E+00;                                 0000011P0000025
126,1,1,1,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000013P0000026
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .10000000E+01, 0000013P0000027
  .50000000E+00,  .00000000E+00,  .50000000E+00,  .10000000E+01, 0000013P0000028
  .00000000E+00,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000013P0000029
  .00000000E+00,  .10000000E+01;                                 0000013P0000030
126,1,1,0,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000015P0000031
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .50000000E+00, 0000015P0000032
  .10000000E+01,  .15000000E+01,  .00000000E+00,  .50000000E+00, 0000015P0000033
  .50000000E+00,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000015P0000034
  .00000000E+00,  .00000000E+00;                                 0000015P0000035
126,1,1,1,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000017P0000036
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .50000000E+00, 0000017P0000037
  .10000000E+01,  .00000000E+00,  .00000000E+00,  .50000000E+00, 0000017P0000038
  .00000000E+00,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000017P0000039
  .00000000E+00,  .10000000E+01;                                 0000017P0000040
126,1,1,0,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000019P0000041
  .10000000E+01,  .10000000E+01,  .10000000E+01, -.24825342E-16, 0000019P0000042
  .50000000E+00,  .50000000E+00,  .50000000E+00,  .00000000E+00, 0000019P0000043
  .50000000E+00,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000019P0000044
  .00000000E+00,  .00000000E+00;                                 0000019P0000045
126,1,1,1,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000021P0000046
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .00000000E+00, 0000021P0000047
  .50000000E+00,  .00000000E+00,  .50000000E+00,  .00000000E+00, 0000021P0000048
  .00000000E+00,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000021P0000049
  .00000000E+00,  .10000000E+01;                                        0000021P0000050
S0000001G0000003D0000022P0000050                                              T0000001
```

Example Program

```
      PROGRAM SAMPLE

C     Example program demonstrating the manipulation of IGES files
C     and data

      INTEGER           MEMAXC, MEMAXI, MEMAXD
      PARAMETER        (MEMAXC=10000, MEMAXI=10000, MEMAXD=10000)
      CHARACTER         CMEM*(MEMAXC)
      INTEGER           IMEM(MEMAXI)
      DOUBLE PRECISION DMEM(MEMAXD)

      INTEGER           IER
      INTEGER           IGI1, IGI2, LVLI, LVLO, LVLE, ITYPE, IGE, IBF
      INTEGER           NSL, NDE, JDE1, JDE2
      LOGICAL           INIT

      INTEGER           IGESLU
      CHARACTER*10      FILIN, FILOUT

      DATA              IGESLU  /7/
      DATA              FILIN   /'SAMPLE.IGS'/
      DATA              FILOUT  /'OUTPUT.IGS'/


C     Begin program execution

      WRITE (6, 1000)

C     Initialize dynamic memory arrays

      CALL D2INIT (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, '*',
     +             'H', IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2INIT'
         GOTO 9900
      ENDIF

C     Read only the NASA-IGES-NURBS entities from the IGES file
C     Specify a negative unit for verbose status messages

      OPEN (UNIT=IGESLU, FILE=FILIN, ERR=9900)

      LVLI = 1
      CALL D2IGRD (CMEM, IMEM, DMEM, FILIN, -IGESLU, LVLI, IGI1,
     +             LVLE, IER)

      CLOSE (UNIT=IGESLU)

      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGRD'
         GOTO 9900
      ENDIF

      IF (LVLE .GT. 1)
     +   WRITE (6, 1020) LVLE
```

```
C     Locate the first B-Spline Curve entity (Type 126)

      ITYPE = 126
      JDE1  = 1
      CALL D2IGNT (CMEM, IMEM, DMEM, IGI1, ITYPE, JDE1, IGE, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGNT'
         GOTO 9900
      ENDIF

C     Convert the entity to a DT_NURBS B-Spline Function (BF) entity

      CALL D2IGBF (CMEM, IMEM, DMEM, IGE, IBF, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGBF'
         GOTO 9900
      ENDIF

C     Make a new IGES file

      NSL  = 1
      NDE  = 2
      INIT = .TRUE.
      CALL D2IGDI (CMEM, IMEM, DMEM, NSL, NDE, INIT, IGI2, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGDI'
         GOTO 9900
      ENDIF

C     Convert the B-Spline Function entity to DT_NURBS IGES type 5001

      JDE2 = 1
      CALL D2BFDI (CMEM, IMEM, DMEM, IBF, IGI2, JDE2, IGE, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2BFDI'
         GOTO 9900
      ENDIF

C     Write out just the new IGES file containing the 5001 entity

      OPEN (UNIT=IGESLU, FILE=FILOUT, ERR=9900)

      LVLO = 3
      CALL D2IGWR (CMEM, IMEM, DMEM, IGI2, LVLO, FILOUT, -IGESLU, LVLE,
     +             IER)

      CLOSE (UNIT=IGESLU)

      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGWR'
         GOTO 9900
      ENDIF

 9900 RETURN

 1000 FORMAT (1X,' Begin IGES read/write demonstration',//)
 1010 FORMAT (1X,A,': Unexpected Failure -- ABORTING',//)
 1020 FORMAT (//,1X,'D2IGRD encountered entities at level=',I1,
     +           '; greater than the level requested -- ignored',//)
```

```
      END
```

Program Output

```
 Begin IGES read/write demonstration


D2IGRD: Processing Entity (IDE) Number     1; Entity Type =   143 Form =     0
D2IGRD: Processing Entity (IDE) Number     3; Entity Type =   128 Form =     0
D2IGRD: Processing Entity (IDE) Number     5; Entity Type =   141 Form =     0
D2IGRD: Processing Entity (IDE) Number     7; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number     9; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number    11; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number    13; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number    15; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number    17; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number    19; Entity Type =   126 Form =     0
D2IGRD: Processing Entity (IDE) Number    21; Entity Type =   126 Form =     0
D2IGWR: Processing Entity (IDE) Number     1; Entity Type =  5001 Form =     0
```

Output IGES File

```
                                                                        S      1
 1H,, 1H;, 1H , 1H , 1H ,12HDT_NURBS 1.0,32,38,7,307,16, 1H ,           G0000001
  .10000000E+01,1, 2HIN,1,  .33330000E-02, 1H ,  .10000000E-05,         G0000002
  .10000000E+06, 1H , 1H ,9,0, 1H ;                                     G0000003
    5001       1       0       0       0       0       0       000000000D0000001
    5001       0       0       3       0                               0D0000002
5001,1,3,1,2,2,  .00000000E+00,  .00000000E+00,  .10000000E+01,  0000001P0000001
  .10000000E+01,  .50000000E+00,  .10000000E+01, -.24825342E-16, 0000001P0000002
  .50000000E+00,  .50000000E+00,  .15000000E+01,;                0000001P0000003
S0000001G0000003D0000002P0000003                                        T0000001
```

NAMED GROUPS OF ENTITIES AND IGES 9

9.1 PROBLEM STATEMENT

Suppose you have an IGES file containing a group of entities associated by an IGES entity type 402 (Associativity Instance) and 406-15 (Name Label) and you need to translate this group of entities into DT_NURBS entities, do some analysis, and output a new group of entities into an IGES type 406-15 with a backpointer to an IGES type 402.

9.2 PROBLEM ANALYSIS

The DT_NURBS routine D2IGNL allows an IGES type 406-15 to be translated to a U1 entity (Universal with one pointer). The U1 has a subtype identifier of "_NLE" for "Name Label Entity." If the 406-15 IGES entity contains a backpointer to an IGES type 402 entity, the 402 is translated to a DT_NURBS Pointer Array (PA) and the pointer in the U1 points to this PA.

The DT_NURBS routine D2NLIG allows a DT_NURBS Name Label Entity (U1 with subtype identifier "_NLE") to be translated to an IGES type 406-15 with a backpointer to an IGES type 402.

The pair of DT_NURBS routines D2IGTR and D2DTTR, which are the general purpose IGES-DT Translation routines, also perform the above translation processes.

9.3 EXAMPLE PROGRAM

Input IGES File

```
                                                                        S      1
 1H,, 1H;, 1H , 1H , 1H ,12HDT_NURBS 1.0,32,38,7,307,16, 1H ,           G0000001
 0.10000000E+01,1, 2HIN,1, 0.33330000E-02, 1H , 0.10000000E-05,         G0000002
 0.10000000E+06, 1H , 1H ,9,0, 1H ;                                     G0000003
     402       1       0       0       0       0       0       000000000D0000001
     402       0       0       1       1                               0D0000002
     128       2       0       0       0       0       0       000010000D0000003
     128       0       0       8       0                S              0D0000004
     124      10       0       0       0       0       0       000010201D0000005
     124       0       0       1       0                TRANSFOR       0D0000006
     126      11       0       0       0       0       0       000010000D0000007
     126       0       0       5       0                               0D0000008
     406      16       0       0       0       0       0       000000000D0000009
     406       0       0       1      15                               0D0000010
402,4,3,5,7,9;                                                   0000001P0000001
128,1,1,1,1,0,0,1,0,0,  .00000000E+00,  .00000000E+00,           0000003P0000002
  .10000000E+01,  .10000000E+01,  .00000000E+00,  .00000000E+00, 0000003P0000003
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .10000000E+01, 0000003P0000004
  .10000000E+01,  .10000000E+01,  .00000000E+00,  .00000000E+00, 0000003P0000005
  .00000000E+00,  .10000000E+01,  .00000000E+00,  .10000000E+01, 0000003P0000006
  .00000000E+00,  .10000000E+01,  .10000000E+01,  .10000000E+01, 0000003P0000007
  .10000000E+01,  .20000000E+01,  .00000000E+00,  .10000000E+01, 0000003P0000008
  .00000000E+00,  .10000000E+01;                                 0000003P0000009
124,0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0,0;         0000005P0000010
126,1,1,0,0,1,0,  .00000000E+00,  .00000000E+00,  .10000000E+01, 0000007P0000011
```

```
  .10000000E+01,  .10000000E+01,  .10000000E+01,  .50000000E+00, 0000007P0000012
  .00000000E+00,  .50000000E+00,  .10000000E+01,  .50000000E+00, 0000007P0000013
  .15000000E+01,  .00000000E+00,  .10000000E+01,  .00000000E+00, 0000007P0000014
  .00000000E+00,  .00000000E+00;                                 0000007P0000015
406,1,1,10HTEST GROUP,1,1;                                       0000009P0000016
S0000001G0000003D0000010P0000016                                        T0000001
```

Example Program

```
      PROGRAM SAMPLE

C     Example program demonstrating the use of D2IGNL and D2NLIG to
C     translate an IGES type 406-15 to a DT_NURBS U1 entity and back.

      INTEGER     MEMAXC, MEMAXI, MEMAXD
      PARAMETER  (MEMAXC = 32760,
     +            MEMAXI = 32760,
     +            MEMAXD = 32760)

      CHARACTER         CMEM*(MEMAXC)
      INTEGER           IMEM(MEMAXI)
      DOUBLE PRECISION  DMEM(MEMAXD)

      INTEGER IER, LVLI, LVLE, LVLO, IGI1, IGI2, ITYPE, ITYP
      INTEGER JDE, JDE1, IGE, IPA, IPA1, LENPA, INL, INL1, IFU
      INTEGER IBF, IDM, ICF, NSL, NDE, I, J
      LOGICAL INIT
      DOUBLE PRECISION NDOM, NDEP

      INTEGER  IGESLU
      CHARACTER*10 FILIN, FILOUT, GRPNAM

      DATA IGESLU / 7 /
      DATA FILIN  / 'SAMPLE.IGS' /
      DATA FILOUT / 'OUTPUT.IGS' /
      DATA GRPNAM / 'NEW GROUP ' /

      WRITE (6, 1000)

C     Initialize Dynamic Memory Arrays
      CALL D2INIT (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, '*',
     *             'H', IER)
      IF (IER .NE. 0) THEN
         WRITE (6,1010) 'D2INIT'
         GOTO 9900
      ENDIF

C     Read the IGES input file
      OPEN (UNIT = IGESLU, FILE = FILIN, ERR = 9900)

      LVLI = 2
      CALL D2IGRD (CMEM, IMEM, DMEM, FILIN, -IGESLU, LVLI, IGI1,
     *             LVLE, IER)
      CLOSE (UNIT = IGESLU)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGRD'
         GOTO 9900
      ENDIF

C     Get the number of Entities in the IGES Index (NDE/2)
      CALL D2IGLI (CMEM, IMEM, DMEM, IGI1, NSL, NDE, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGLI'
         GOTO 9900
      ENDIF
```

```
C     Locate the IGES type 406-15
      ITYPE = 406
      JDE1 = 1
      CALL D2IGNT (CMEM, IMEM, DMEM, IGI1, ITYPE, JDE1, IGE, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGNT'
         GOTO 9900
      ENDIF

C     Convert the entity to a DT_NURBS Name Label Entity (U1)
      CALL D2IGNL (CMEM, IMEM, DMEM, IGE, INL, IPA, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGNL'
         GOTO 9900
      ENDIF

      CALL D2FELI(CMEM, IMEM, DMEM, IPA, LENPA, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2FELI'
         GOTO 9900
      ENDIF

C     Create a new PA Entity and NL Entity
      CALL D2ADF (CMEM, IMEM, DMEM, 'P', LENPA-1, IPA1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2ADF'
         GOTO 9900
      ENDIF

      INIT = .TRUE.
      CALL D2DEFE (CMEM, IMEM, DMEM, 226, 14, 1, 0, INIT,
     *             INL1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2DEFE'
         GOTO 9900
      ENDIF
      CALL D2STAC (CMEM, IMEM, DMEM, '_NLE', 1, 4, INL1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2STAC'
         GOTO 9900
      ENDIF

C     Store the Name Label Pointer in the PA
      CALL D2STEI (CMEM, IMEM, DMEM, INL1, 1, IPA1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2STEI'
         GOTO 9900
      ENDIF

C     Store the PA Pointer and Name in the NL
      CALL D2STEI (CMEM, IMEM, DMEM, IPA1, 1, INL1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2STEI'
         GOTO 9900
      ENDIF
      CALL D2STAC (CMEM, IMEM, DMEM, GRPNAM, 5, 14, INL1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2STAC'
         GOTO 9900
      ENDIF

C     Compose the DM and Surface from the IGES file
      J = 2
      DO 100 I = 1, LENPA
         CALL D2FEEI(CMEM, IMEM, DMEM, IPA, I, IFU, IER)
         IF (IER .NE. 0) THEN
            WRITE (6, 1010) 'D2FEEI'
            GOTO 9900
         ENDIF

         CALL D2FEET(CMEM, IMEM, DMEM, IFU, ITYP, IER)
         IF (IER .NE. 0) THEN
            WRITE (6, 1010) 'D2FEET'
            GOTO 9900
         ENDIF

         IF (ITYP .EQ. 246) THEN
C           Check if this BF is the surface
            CALL D2FEED (CMEM, IMEM, DMEM, IFU, 1, NDOM, IER)
            IF (IER .NE. 0) THEN
               WRITE (6, 1010) 'D2FEED'
               GOTO 9900
            ENDIF
            CALL D2FEED (CMEM, IMEM, DMEM, IFU, 2, NDEP, IER)
            IF (IER .NE. 0) THEN
               WRITE (6, 1010) 'D2FEED'
               GOTO 9900
            ENDIF
            IF (NDOM .EQ. 2 .AND. NDEP .EQ. 3) THEN
               IBF = IFU
               GOTO 100
            ENDIF
         ELSEIF (ITYP .EQ. 234) THEN
            IDM = IFU
            GOTO 100
         ELSEIF (ITYP .EQ. 226) THEN
            GOTO 100
         ENDIF
         CALL D2STEI (CMEM, IMEM, DMEM, IFU, J, IPA1, IER)
         IF (IER .NE. 0) THEN
            WRITE (6, 1010) 'D2STEI'
            GOTO 9900
         ENDIF
         J = J + 1
 100  CONTINUE

C     Compose the DM and Surface
      CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBF, 'A', IDM, ICF, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2POSE'
         GOTO 9900
      ENDIF

C     Store the Composition in the PA
      CALL D2STEI (CMEM, IMEM, DMEM, ICF, J, IPA1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2STEI'
         GOTO 9900
      ENDIF

C     Create a new IGES Index Entity and store the new NL entity
      NSL = 1
      INIT = .TRUE.
      CALL D2IGDI (CMEM, IMEM, DMEM, NSL, NDE+2, INIT, IGI2, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGDI'
         GOTO 9900
      ENDIF

C     Translate the new NL Entity to IGES 406-15
      JDE = 1
      CALL D2NLIG (CMEM, IMEM, DMEM, INL1, IGI2, JDE, IGE, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2NLIG'
         GOTO 9900
      ENDIF

C     Write a new IGES file containing the new 406-15 Entity
      OPEN (UNIT = IGESLU, FILE = FILOUT, ERR = 9900)

      WRITE (6, *)
      LVLO = 3
      CALL D2IGWR (CMEM, IMEM, DMEM, IGI2, LVLO, FILOUT, -IGESLU,
     *             LVLE, IER)
      CLOSE (UNIT = IGESLU)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2IGWR'
         GOTO 9900
      ENDIF

      WRITE (6, 1020)

 9900 CONTINUE

 1000 FORMAT (1X,/, ' Begin IGES 406-15 Conversion Demonstration'/)
 1010 FORMAT (1X, A, ': Unexpected Failure -- ABORTING',//)
 1020 FORMAT (1X,/, ' IGES 406-15 Conversion Demonstration Complete',/)

      END
```

Program Output

```
 Begin IGES 406-15 Conversion Demonstration

 D2IGRD: Processing Entity (IDE) Number     1; Entity Type =   402 Form =     1
 D2IGRD: Processing Entity (IDE) Number     3; Entity Type =   128 Form =     0
 D2IGRD: Processing Entity (IDE) Number     5; Entity Type =   124 Form =     0
 D2IGRD: Processing Entity (IDE) Number     7; Entity Type =   126 Form =     0
 D2IGRD: Processing Entity (IDE) Number     9; Entity Type =   406 Form =    15

 D2IGWR: Processing Entity (IDE) Number     1; Entity Type =   402 Form =     1
 D2IGWR: Processing Entity (IDE) Number     3; Entity Type =   406 Form =    15
 D2IGWR: Processing Entity (IDE) Number     5; Entity Type =  5001 Form =     0
 D2IGWR: Processing Entity (IDE) Number     7; Entity Type =  5002 Form =     0
 D2IGWR: Processing Entity (IDE) Number     9; Entity Type =  5001 Form =     0
 D2IGWR: Processing Entity (IDE) Number    11; Entity Type =  5005 Form =     2

 IGES 406-15 Conversion Demonstration Complete



Output IGES File
                                                                        S      1
 1H,, 1H;, 1H , 1H , 1H ,12HDT_NURBS 1.0,32,38,7,307,16, 1H ,           G0000001
 0.10000000E+01,1, 2HIN,1, 0.33330000E-02, 1H , 0.10000000E-05,         G0000002
 0.10000000E+06, 1H , 1H ,9,0, 1H ;                                     G0000003
     402       1       0       0       0       0       0       000000000D0000001
     402       0       0       1       1                               0D0000002
     406       2       0       0       0       0       0       000000000D0000003
     406       0       0       1      15                               0D0000004
    5001       3       0       0       0       0       0       000000000D0000005
    5001       0       0       3       0                               0D0000006
    5002       6       0       0       0       0       0       000000000D0000007
    5002       0       0       1       0                               0D0000008
    5001       7       0       0       0       0       0       000000000D0000009
    5001       0       0       6       0                S              0D0000010
    5005      13       0       0       0       0       0       000000000D0000011
    5005       0       0       4       2                               0D0000012
402,3,3,5,7;                                                     0000001P0000001
406,1,10HNEW GROUP ,1,1;                                         0000003P0000002
```

```
5001,1,3,1,2,2, 0.00000000E+00, 0.00000000E+00, 0.10000000E+01,  0000005P0000003
 0.10000000E+01, 0.50000000E+00, 0.10000000E+01, 0.00000000E+00, 0000005P0000004
 0.50000000E+00, 0.50000000E+00, 0.15000000E+01,;                0000005P0000005
5002,2,3,2,5,9,4,11,;                                            0000007P0000006
5001,2,3,1,2,2,2,2, 0.00000000E+00, 0.00000000E+00,              0000009P0000007
 0.10000000E+01, 0.10000000E+01, 0.00000000E+00, 0.00000000E+00, 0000009P0000008
 0.10000000E+01, 0.10000000E+01, 0.00000000E+00, 0.10000000E+01, 0000009P0000009
 0.00000000E+00, 0.10000000E+01, 0.00000000E+00, 0.00000000E+00, 0000009P0000010
 0.10000000E+01, 0.10000000E+01, 0.00000000E+00, 0.10000000E+01, 0000009P0000011
 0.10000000E+01, 0.20000000E+01,;                                0000009P0000012
5005,2,3,4, 0.00000000E+00, 0.10000000E+01, 0.00000000E+00,      0000011P0000013
 0.10000000E+01, 0.00000000E+00, 0.00000000E+00, 0.00000000E+00, 0000011P0000014
 0.00000000E+00, 0.10000000E+01, 0.00000000E+00, 0.00000000E+00, 0000011P0000015
 0.00000000E+00;                                                 0000011P0000016
S0000001G0000003D0000012P0000016                                        T0000001
```

USER-DEFINED DATA TYPES 10

10.1 PROBLEM STATEMENT

Suppose that the user's application frequently uses 3 × 3 matrices which have a string name, and which are linked into sequences. Suppose further that these entities are to be readily checkable for correct type and that they must be communicated correctly in D2 Format files.

10.2 PROBLEM ANALYSIS

.This structure does not match any predefined Library data type. Hence, it is a natural candidate for a user-defined data type

Once the user has decided the structure of his new entity type, it is very simple to add it to the set of defined types. The names of user-defined data types are stored when the dynamic memory is initialized by the subroutine D2INIT (INITialize), simply by including a delimited string for the parameter TYPNAM. These user-defined names can be fetched from the dynamic memory, using the subroutine D2FETN (FETch Name). The user can create allocator, accessor, modifier, and deallocator subroutines for entities of a user-defined type by applying the type-neutral general-purpose subroutines in the Library. Also, if D2 Format files containing this type are to be created, the user will need to create a special "locate_pointers" subroutine and pass its name as an argument in the call to D2WRIT (WRITe). The "locate_pointers" subroutine is responsible for telling D2WRIT where the pointers are in any given entity of user-defined type. It is very special in that it may not call on any Library subroutines to help it perform this task. (See the description of the SUBROU argument of D2WRIT.)

10.3 EXAMPLE PROGRAM

<u>Example Program</u>

```
      PROGRAM SAMPLE

C     Example program demonstrating the manipulation of user-
C     defined data types

      EXTERNAL UWRITE

      INTEGER          MEMAXC, MEMAXI, MEMAXD
      PARAMETER       (MEMAXC=10000, MEMAXI=10000, MEMAXD=10000)
      CHARACTER        CMEM*(MEMAXC)
      INTEGER          IMEM(MEMAXI)
      DOUBLE PRECISION DMEM(MEMAXD)

      INTEGER          IER
      INTEGER          IDE1, IDE2, MEMPA
      DOUBLE PRECISION DA(9)

C     Begin program execution
```

```
      WRITE (6, 1000)

C     Initialize dynamic memory arrays
C        Define User type 1 to be called "Matrix"

      CALL D2INIT (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, '*Matrix',
     +             'H', IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2INIT', IER
         GOTO 9900
      ENDIF

C     Define and initialize two "Matrix" entities

      DA(1) = 1.0D0
      DA(2) = 2.0D0
      DA(3) = 3.0D0
      DA(4) = 4.0D0
      DA(5) = 5.0D0
      DA(6) = 6.0D0
      DA(7) = 7.0D0
      DA(8) = 8.0D0
      DA(9) = 9.0D0

      CALL CREATE (CMEM, IMEM, DMEM, 'Matrix One', 0, DA, IDE1, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'CREATE', IER
         GOTO 9900
      ENDIF

      DA(1) = 9.0D0
      DA(2) = 8.0D0
      DA(3) = 7.0D0
      DA(4) = 6.0D0
      DA(5) = 5.0D0
      DA(6) = 4.0D0
      DA(7) = 3.0D0
      DA(8) = 2.0D0
      DA(9) = 1.0D0

      CALL CREATE (CMEM, IMEM, DMEM, 'Matrix Two', IDE1, DA, IDE2, IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'CREATE', IER
         GOTO 9900
      ENDIF

C     Write out all of the dynamic memory entities

      MEMPA=0
      CALL D2WRIT (CMEM, IMEM, DMEM, 0, MEMPA, UWRITE, 'OUTPUT.D2', 7,
     +             IER)
      IF (IER .NE. 0) THEN
         WRITE (6, 1010) 'D2WRIT', IER
         GOTO 9900
      ENDIF

 9900 RETURN

 1000 FORMAT (1X,' Begin User-defined entity demonstration',//)
 1010 FORMAT (1X,A,': Unexpected Failure (',I3,') -- ABORTING',//)
```

```
      END

      SUBROUTINE CREATE (CMEM, IMEM, DMEM, CA, INEXT, DA, IUE, IER)

C     Subroutine to create the user-defined entity type Matrix

      CHARACTER        CMEM*(*)
      INTEGER          IMEM(*)
      DOUBLE PRECISION DMEM(*)

      INTEGER          ENTUSR
      PARAMETER       (ENTUSR=1)

      INTEGER          INEXT, IUE, IER
      DOUBLE PRECISION DA(*)
      CHARACTER        CA*(*)

      INTEGER          LENC, LENI, LEND
      LOGICAL          INITIZ

      LENC = LEN(CA)
      LENI = 1
      LEND = 9

      INITIZ = .TRUE.

      CALL D2DEFE (CMEM, IMEM, DMEM, ENTUSR, LENC, LENI, LEND, INITIZ,
     +             IUE, IER)
      IF (IER .NE. 0) GOTO 9900

      CALL D2STAC (CMEM, IMEM, DMEM, CA, 1, LENC, IUE, IER)
      IF (IER .NE. 0) GOTO 9900

      CALL D2STEI (CMEM, IMEM, DMEM, INEXT, 1, IUE, IER)
      IF (IER .NE. 0) GOTO 9900

      CALL D2STAD (CMEM, IMEM, DMEM, DA, 1, 9, IUE, IER)
      IF (IER .NE. 0) GOTO 9900

 9900 CONTINUE
      RETURN
      END


      SUBROUTINE UWRITE (CMEM, IMEM, DMEM, LP)

C     User-modified version of the D0LPUT routine
C     Handles the user-defined "Matrix" entity
C
C     LP      Array of context data.
C             LP(1) = Header index for entity
C             LP(2) = MEM data type number of entity.
C             LP(3) = 0  Initial call for this entity.
C                   > 0  Beginning absolute index of previous pointer block.
C             LP(4) = Ending absolute index of previous pointer block.
C             LP(5) = Absolute index of end of entity's integer data space.
C             Upon return, LP(3) and LP(4) have been updated to locate the
C             beginning and end of the next pointer block in the entity's
C             integer data space, or to LP(5)+1 if no more pointers.
```

```
      CHARACTER        CMEM*(*)
      INTEGER          IMEM(*)
      DOUBLE PRECISION DMEM(*)
      INTEGER          LP(14)

C****
      IF (LP(3) .EQ. 0) THEN
         LP(3) = LP(4) + 1
         LP(4) = LP(5)
      ELSE
         LP(3) = LP(4) + 1
         LP(4) = LP(4) + 1
      ENDIF

      RETURN
      END
```

Output "D2 File" from D2WRIT

```
D2 File Format 1.0  80  11  23  16   3
          0           2           1          20          18          18
    2555906*    2553859*
  1  6Matrix

    2555906*         10           1           9           1
Matrix One
          0*
 1.000000000000000      2.000000000000000      3.000000000000000
 4.000000000000000      5.000000000000000      6.000000000000000
 7.000000000000000      8.000000000000000      9.000000000000000

    2553859*         10           1           9           1
Matrix Two
    2555906*
 9.000000000000000      8.000000000000000      7.000000000000000
 6.000000000000000      5.000000000000000      4.000000000000000
 3.000000000000000      2.000000000000000      1.000000000000000
```

REINITIALIZING DYNAMIC MEMORY ARRAYS 11

11.1 PROBLEM STATEMENT

Sometimes it is difficult to predict the size of the dynamic memory arrays. Also, problems may be so complex that it is not practical to abort and restart if a failure occurs.

11.2 PROBLEM ANALYSIS

Many users are using Fortran compilers that allow allocatable arrays, or C to drive their DT_NURBS subroutines. These users needed a simple way to reinitialize the dynamic memory after an expansion (REALLOC), so that the program can continue. This subroutine D2REIN (RE-INitialize dynamic memory) is designed to be called after more storage has been tacked onto otherwise valid dynamic memory arrays.

It is best to save the data before calling the subroutine which is most likely to fail, and restore that saved data. If, instead, the user were to simply reallocate and reinitialize the arrays after the failure, then repeat the call, there is likely to be too many working storage entities and locks left in place by the failed routine.

11.3 EXAMPLE CODE

```
      PROGRAM S2REIN
C
C     Example program to demonstrate subroutine D2REIN.
C
      INTEGER     MEMAXC, MEMAXI, MEMAXD, IMAX, DMAX
      PARAMETER  (MEMAXC = 1000,
     +            IMAX   = 1000,
     +            DMAX   = 5000)

      CHARACTER         CMEM*(MEMAXC)
      INTEGER           IMEM[ALLOCATABLE] (:), IMEMSV(IMAX)
      DOUBLE PRECISION  DMEM[ALLOCATABLE] (:), DMEMSV(DMAX)

      INTEGER           IER, IERX

      INTEGER           LEND1, LEND2, IBF1, IBF2, NCUTS, IPCUTS
      PARAMETER        (LEND1 = 72, LEND2 = 28)
      DOUBLE PRECISION  BF1(LEND1), BF2(LEND2), TOL

      DATA BF1 /
     +  2.D0, 3.D0, 4.D0, 4.D0, 4.D0, 4.D0, 0.D0, 0.D0,
     +  0.D0, 0.D0, 0.D0, 0.D0, 1.D0, 1.D0, 1.D0, 1.D0,
     +  0.D0, 0.D0, 0.D0, 0.D0, 1.D0, 1.D0, 1.D0, 1.D0,
     +  0.D0, .33333333D0, .66666666D0, 1.D0, 0.D0,
     +   .33333333D0, .66666666D0, 1.D0,
     +  0.D0, .33333333D0, .66666666D0, 1.D0, 0.D0,
     +   .33333333D0, .66666666D0, 1.D0,
     +  0.D0, 0.D0, 0.D0, 0.D0, .33333333D0, .33333333D0,
     +   .33333333D0, .33333333D0,
     +  .66666666D0, .66666666D0, .66666666D0, .66666666D0,
     +   1.D0, 1.D0, 1.D0, 1.D0,
     +  1.D0, 1.D0, 1.D0, 1.D0, -.9D0, 1.D0, -2.4D0, 1.D0,
     +  -1.D0, 1.D0, -2.D0, 1.D0, 1.D0, 1.D0, 1.D0, 1.D0 /

      DATA BF2 /
     +  2.D0, 3.D0, 2.D0, 2.D0, 2.D0, 2.D0, 0.D0, 0.D0,
     +  0.D0, 0.D0, 1.D0, 1.D0, 0.D0, 0.D0, 1.D0, 1.D0,
     +  0.D0, 1.D0, 0.D0, 1.D0, 0.D0, 0.D0, 1.D0, 1.D0,
     +  0.D0, 0.D0, 0.D0, 0.D0 /

      TOL = 1.0D-6
C     ---------------

      CALL DTERRS (3,1000)

C     Initialize dynamic memory to 100-word arrays

      MEMAXI = IMAX
      MEMAXD = DMAX

      ALLOCATE (IMEM(MEMAXI),
     +          DMEM(MEMAXD),
     +          STAT=IER)

      IF (IER .NE. 0)
     +   STOP ' * NOT ENOUGH MEMORY FOR INITIAL ALLOCATION *'

      CALL D2INIT (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, ' ',
     +      'H', IER)
      IF (IER .NE. 0)
     +   STOP ' * UNEXPECTED ERROR FROM D2INIT *'

C     Create two B-spline entities

      CALL D2BFDF (CMEM, IMEM, DMEM, LEND1, 'BF 1', IBF1, IER)
      IF (IER .NE. 0)
     +   STOP ' * UNEXPECTED FAILURE ON SPLINE ALLOCATION 1 *'

      CALL D2STAD (CMEM, IMEM, DMEM, BF1, 1, LEND1, IBF1, IER)
      IF (IER .NE. 0)
     +   STOP ' * UNEXPECTED FAILURE ON SPLINE INITIALIZATION 1 *'

      CALL D2BFDF (CMEM, IMEM, DMEM, LEND2, 'BF 2', IBF2, IER)
      IF (IER .NE. 0)
     +   STOP ' * UNEXPECTED FAILURE ON SPLINE ALLOCATION 2 *'

      CALL D2STAD (CMEM, IMEM, DMEM, BF2, 1, LEND2, IBF2, IER)
      IF (IER .NE. 0)
     +   STOP ' * UNEXPECTED FAILURE ON SPLINE INITIALIZATION 2 *'

C     Save memory state in case a re-start is necessary
C     This might be replaced by a call to D2WRIT

      CALL D0ICPY (IMAX, IMEM, 1, IMEMSV, 1)
      CALL DCOPY (DMAX, DMEM, 1, DMEMSV, 1)

 1000 CONTINUE

C        Find the intersection of the splines
```

```
         CALL D2SSXT (CMEM, IMEM, DMEM, IBF1, IBF2, TOL, NCUTS, IPCUTS,
     +                IER)
         IF (IER .NE. 0) THEN
            IF (IER .EQ. -102) THEN

C              Insufficient IMEM
C              Allocate more integer memory and start over

               MEMAXI = MEMAXI+1000

               WRITE (6,*) ' > Increasing IMEM to ',MEMAXI

               DEALLOCATE (IMEM)
               ALLOCATE (IMEM(MEMAXI), STAT=IERX)
               IF (IERX .NE. 0)
     +           STOP ' * NOT ENOUGH MEMORY FOR INTEGER RE-ALLOCATION *'

            ELSE IF (IER .EQ. -103) THEN

C              Insufficient DMEM
C              Allocate more real memory and start over

               MEMAXD = MEMAXD+5000

               WRITE (6,*) ' > Increasing DMEM to ',MEMAXD

               DEALLOCATE (DMEM)
               ALLOCATE (DMEM(MEMAXD), STAT=IERX)
               IF (IERX .NE. 0)
     +           STOP ' * NOT ENOUGH MEMORY FOR REAL RE-ALLOCATION *'

            ELSE
               STOP ' * UNEXPECTED ERROR FROM INTERSECTOR *'
            ENDIF

            CALL D0ICPY (IMAX, IMEMSV, 1, IMEM, 1)
            CALL DCOPY (DMAX, DMEMSV, 1, DMEM, 1)

            CALL D2REIN (CMEM, IMEM, DMEM, MEMAXC, MEMAXI, MEMAXD, IER)
            IF (IER .NE. 0)
     +         STOP ' * UNEXPECTED ERROR FROM D2REIN *'

            GOTO 1000
         ENDIF

      WRITE (6,*)
      WRITE (6,*) ' Intersection completed successfully'
C     Intersection is done.  Now process the output
C           ...

      STOP
      END
```

Output

```
***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
        28 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =    -103)
```

```
 > Increasing DMEM to       10000

***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
       800 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =   -103)

 > Increasing DMEM to       15000

***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
        64 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =   -102)

 > Increasing IMEM to        2000

***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2CNTR
      3958 WORDS OF WORKING STORAGE NEEDED
      SEE  D2CNTR ABSTRACT (IER =    -23)


***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
         0 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =   -103)

 > Increasing DMEM to       20000

***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SPDR
        66 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SPDR   ABSTRACT (IER =     -9)


***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2CNTR
         0 WORDS OF WORKING STORAGE NEEDED
      SEE  D2CNTR ABSTRACT (IER =    -23)


***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
         0 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =   -103)

 > Increasing DMEM to       25000

***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2CNTR
     12850 WORDS OF WORKING STORAGE NEEDED
      SEE  D2CNTR ABSTRACT (IER =    -23)


***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
         0 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =   -103)

 > Increasing DMEM to       30000

***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2CNTR
       754 WORDS OF WORKING STORAGE NEEDED
      SEE  D2CNTR ABSTRACT (IER =    -23)


***** INPUT ARGUMENT ERROR REPORTED BY SUBROUTINE D2SSXT
         0 WORDS OF WORKING STORAGE NEEDED
      SEE  D2SSXT ABSTRACT (IER =   -103)
 > Increasing DMEM to       35000

 Intersection completed successfully
Stop - Program terminated.
```

SPECIAL INTERSECTION 12

12.1 PROBLEM STATEMENT

Suppose there are two analysis functions defined on overlapping subranges of some base geometry surface. Suppose the first one has two independent variables and four dependent variables - u, v, T, and E_c, where u and v are the parameters in the domain of the base geometry, T is the temperature, and E_c is the rate of radiative cooling. Suppose the second one has three independent variables, the third being time, and five dependent variables - u, v, T, P and E_h, where u and v are the parameters in the domain of the base geometry, T is the temperature, P is the pressure, and E_h is the rate of frictional heating. The problem is to distinguish, at some sample of discrete times, the areas of the surface where the temperature is rising from the areas where it is falling.

12.2 PROBLEM ANALYSIS

Let us ignore the dubious physics and simply focus on the mathematics. First, reformulate the problem as one of finding the boundaries between the areas where the temperature is rising and where it is falling, i.e. find the places where the radiative cooling equals the frictional heating. Knowing the boundaries, one can test any one point within a region to determine whether the heating rate exceeds the cooling rate or vice versa. Second, observe that the input data consists of three functions - the base geometry surface function:

$$f(u,v) = (x,y,z),$$

the first analysis function:

$$g(p,q) = (u,v,T,E_c),$$

and the second analysis function:

$$h(r,s,t) = (u,v,T,P,E_h).$$

Third, note that although we may ultimately want to view curves on the surface in xyz-space, it is better to find the curves in the parameter domain of the surface (uv-space) and then map them onto the surface with the surface function f, than it is to find them in the xyz-space directly. So the problem can now be stated as finding those points where (u,v,E_c) from the g analysis function match (u,v,E_h) from the h analysis function. This means that the other dependent variables in the analysis functions are extraneous to the current problem. In Library terminology, they can be eliminated by composing the original functions with "select and permute" functions:

$$s_{1,2,4} \circ g(p,q) = (u,v,E) = s_{1,2,5} \circ h(r,s,t).$$

The problem is now very close to being a standard surface intersection problem. The Library surface intersection subroutine takes two "surface functions", that is, functions with two independent variables and three dependent variables each, and finds the places (normally curves) where their three dependent variables match up. The one aspect that doesn't fit is that the second analysis function, *h*, has three independent variables. The solution is to specialize to one time slice at a time. Selecting time t_i creates a two-parameter function

$$h_i(r,s) \equiv h(r,s,t_i).$$

Mathematically, the problem has now been reformulated as a surface intersection problem. In practice, there remains one more problem that must be solved before applying the Library intersection subroutine (D2SSXT). The Library routine is restricted to pure spline functions of two independent and three dependent variables and this restriction is inherent in the algorithm, which makes essential use of the convex hull property of B-spline functions. Therefore, the extraneous independent and dependent variables must be physically removed from the *g* and *h* functions to produce pure B-spline representations of the functions to be intersected. Assuming that the *g* and *h* functions were initially expressed as pure B-spline functions, this requirement is easily met. The removal of the extra independent variable in *h* by setting it to a constant t_i should be done by applying D2CNPR (CoNstant PaRameter) after a future enhancement makes D2CNPR accept functions with more than two independent variables. In the meantime, the same result can be obtained by applying D2STRM (Spline TRiM) and trimming the third parameter interval to the point t_i. To remove the dependent variables, use D2BINE (comBINE functions) instead of D2POSE (comPOSE functions).

12.3 EXAMPLE CODE

```
      SUBROUTINE EXMPL9 (CMEM, IMEM, DMEM, IBFG, IBFH, TI, TOL, IPA, IER)
C     Find the intersection curves belonging to the intersection of the
C     1st, 2nd and 4th dependent variables of G and the 1st, 2nd and
C     5th dependent variables of H at the slice determined by setting
C     the 3rd independent variable of H to the constant TI.
C
C  Inputs:
C     IBFG    Pointer to B-spline Function entity expressing
C             analysis function G, where G(p,q) = (u,v,T,E)
C     IBFH    Pointer to B-spline Function entity expressing
C             analysis function H, where H(r,s,t) = (u,v,T,P,E)
C     TI      Time splice of H at which to do intersection
C     TOL     Tolerance to which intersection curves are to be determined
C  Outputs:
C     IPA     Pointer to pointer array containing pointers to the B-spline
C             Function entities representing the intersection curves.
C     IER     Error code  (Zero means no error)
C
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), IBFG, IBFH, IPA, IER
      DOUBLE PRECISION DMEM(*), TI, TOL
C
C     Local variables
      INTEGER IBFG1, IBFH1, IBFH2, NCUTS, ISELG, LSELG(5), ISELH, LSELH(5)
      INTEGER N, M, K(3), NC(3)
      DOUBLE PRECISION TRIM(3,2)
      LOGICAL R
C
C     Data for dependent variable selection functions
      DATA LSELG / 4, 3, 1, 2, 4 /
      DATA LSELH / 5, 3, 1, 2, 5 /
C
C     Select 1st, 2nd and 4th dependent variables of G, making IBFG1
      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 5, ISELG, IER)
      CALL D2STAI (CMEM, IMEM, DMEM, LSELG, 1, 5, ISELG, IER)
      CALL D2BINE (CMEM, IMEM, DMEM, IBFG, 'S', ISELG, IBFG1, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, ISELG, IER)
C
C     Select 1st, 2nd and 5th dependent variables of H, making IBFH1
      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 5, ISELH, IER)
      CALL D2STAI (CMEM, IMEM, DMEM, LSELH, 1, 5, ISELH, IER)
      CALL D2BINE (CMEM, IMEM, DMEM, IBFH, 'S', ISELH, IBFH1, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, ISELH, IER)
C
C     Take the constant parameter surface slice corresponding to t=TI
C     from IBFH1, making IBFH2.
      CALL D2BFSZ (CMEM, IMEM, DMEM, IBFH1, 3, N, M, R, K, NC, TRIM(1,1),
     + TRIM(1,2), IER)
      IF (TI .LT. TRIM(3,1) .OR. TI .GT. TRIM(3,2)) THEN
         CALL FAILED ('EXMPL9: Time slice out of range', -11, IER)
         RETURN
      ENDIF
      TRIM(3,1) = TI
      TRIM(3,2) = TI
      CALL D2STRM (CMEM, IMEM, DMEM, IBFH1, TRIM, 3, IBFH2, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IBFH1, IER)
C
C     Perform the intersection on IBFG1 and IBFH1
      CALL D2SSXT (CMEM, IMEM, DMEM, IBFG1, IBFH2, TOL, NCUTS, IPA, IER)
C
C     Clean up
      CALL D2ERAS (CMEM, IMEM, DMEM, IBFG1, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IBFH2, IER)
      RETURN
      END
```

POINT ON INTERSECTION 13

13.1 PROBLEM STATEMENT

Determine whether a given point in model space is on the intersection of two surfaces.

13.2 PROBLEM ANALYSIS

While this problem sounds simple, it doesn't have a simple answer. It will rarely be the case that one can find pairs of parameter values for each surface function that compute to the given point exactly to the last binary digit. So the return question becomes "How close is close enough?" The answer to that will depend on the particular application.

Instead let us consider the problem of computing the distance of a given point in model space from the intersection of two surfaces.

The D2CLSP (CLoSe Point) subroutine can be used to find the distance of the given point to each surface individually. For some purposes, having both such distances sufficiently small is good enough. For other purposes, it may not be. If the surfaces intersect at a very small angle, one can find points close to both surfaces which are nevertheless a long way from the "true" intersection.

Suppose the surface intersection subroutine D2SSXT has been used to compute the one or more intersection curves comprising the "intersection of two surfaces". The D2CLSP subroutine can be used to find the distance of the given point to each component of the intersection curves, or D2CLPM (CLose Point Multiple) might be used to scan all of them at once. When using this interpretation of "intersection", two additional considerations come up. First, these intersection curves are only accurate to the tolerance that was specified in the call to D2SSXT, so it would make no sense to use a criterion of closeness that was any less than that. Second, for each theoretical intersection component, there are two implied intersection curves in model space derivable from the output of D2SSXT, one for each surface, and no reason to prefer one over the other.

13.3 EXAMPLE CODE

```
      SUBROUTINE EXPL11 (CMEM, IMEM, DMEM, ISU1, ISU2, PT, TOL, DIS, IER)
C     Find the distance from PT to the intersection of surfaces ISU1 and
C     ISU2 with a tolerance of TOL.  This example leaves out the necessary
C     IER checking after each call for the sake of expository simplicity.
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), ISU1, ISU2, IER
      DOUBLE PRECISION DMEM(*), PT(3), TOL, DIS
C
C     Local Variables
      INTEGER NCUTS, IPCUTS, IPA, I, LSEL1, LSEL2, IA1, IA2, ICF, IBF
      INTEGER IPIN, MXITER, MXHALF, MXMARQ, DEFP(4)
      INTEGER NCP, INDEX, IP, ID, INITER, IINFO
      DOUBLE PRECISION EPS(4), GAMMA, FNU, TOL2, TMP
C
C     Data for selection functions and D2CLPM inputs
```

```
      DATA LSEL1 / 4, 2, 1, 2 /
      DATA LSEL2 / 4, 2, 3, 4 /
      DATA DEFP / 0.5D0, 0.0D0, 0.5D0, 0.0D0 /
      DATA MXITER, MXHALF, MXMARQ / 30, 3, 10 /
      DATA GAMMA, FNU / 0.25D0, 2.0D0 /
C
C     Find the 4D intersection curves
      TOL2 = 0.5 * TOL
      CALL D2SSXT (CMEM, IMEM, DMEM, ISU1, ISU2, TOL2, NCUTS, IPCUTS, IER)
      IF (NCUTS .EQ. 0) THEN
         CALL FAILED ('EXPL11: Surfaces do not intersect!?', -31, IER)
         RETURN
      ENDIF
C
C     Construct selection functions
      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 4, IA1, IER)
      CALL D2ADF (CMEM, IMEM, DMEM, 'I', 4, IA2, IER)
      CALL D2STAI (CMEM, IMEM, DMEM, LSEL1, 1, 4, IA1, IER)
      CALL D2STAI (CMEM, IMEM, DMEM, LSEL2, 1, 4, IA2, IER)
C
C     Construct the pairs of model space curves corresponding to each
C     intersection curve
      CALL D2ADF (CMEM, IMEM, DMEM, 'P', 2*NCUTS, IPA, IER)
      DO 100 I=1,NCUTS
         CALL D2FEEI (CMEM, IMEM, DMEM, IPCUTS, I, IBF, IER)
         CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBF, 'S', IA1, ICF, IER)
         CALL D2POSX (CMEM, IMEM, DMEM, ICF, 'B', ISU1, IER)
         CALL D2STEI (CMEM, IMEM, DMEM, ICF, 2*I-1, IPA, IER)
         CALL D2POSE (CMEM, IMEM, DMEM, 'B', IBF, 'S', IA2, ICF, IER)
         CALL D2POSX (CMEM, IMEM, DMEM, ICF, 'B', ISU2, IER)
         CALL D2STEI (CMEM, IMEM, DMEM, ICF, 2*I, IPA, IER)
  100 CONTINUE
C
C     Find distances to these curves
      CALL D2ADF (CMEM, IMEM, DMEM, 'D', 4*NCUTS, IPIN, IER)
      DO 200 I=1,NCUTS
         CALL D2STAD (CMEM, IMEM, DMEM, DEFP, 4*I-3, 4*I, IPIN, IER)
  200 CONTINUE
C     Try mainly to achieve absolute distance (EPS(1))
      EPS(1) = TOL2
      EPS(2) = 16.0D0 * DTMCON(6)
      EPS(3) = 0.0625 * TOL2
      EPS(4) = 16.0D0 * DTMCON(6)
      CALL D2CLPM (CMEM, IMEM, DMEM, IPA, PT, IPIN, EPS, GAMMA, FNU,
     + MXITER, MXHALF, MXMARQ, NCP, INDEX, IP, ID, INITER, IINFO, IER)
C
C     Return the average of these distances
      DIS = 0.0D0
      DO 300 I=1,NCP
         CALL D2FEED (CMEM, IMEM, DMEM, ID, I, TMP, IER)
         DIS = DIS + TMP
  300 CONTINUE
      DIS = DIS / DBLE(NCP)
C
C     Clean up, preferably in reverse order of creation
      CALL D2ERAS (CMEM, IMEM, DMEM, IINFO, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, INITER, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, ID, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IP, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, INDEX, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IPIN, IER)
      DO 400 I=1,NCUTS
         CALL D2FEEI (CMEM, IMEM, DMEM, IPA, 2*I, ICF, IER)
         CALL D2ERAS (CMEM, IMEM, DMEM, ICF, IER)
         CALL D2FEEI (CMEM, IMEM, DMEM, IPA, 2*I-1, ICF, IER)
         CALL D2ERAS (CMEM, IMEM, DMEM, ICF, IER)
  400 CONTINUE
      CALL D2ERAS (CMEM, IMEM, DMEM, IPA, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IA2, IER)
      CALL D2ERAS (CMEM, IMEM, DMEM, IA1, IER)
      DO 410 I=1,NCUTS
         CALL D2FEEI (CMEM, IMEM, DMEM, IPCUTS, I, IBF, IER)
         CALL D2ERAS (CMEM, IMEM, DMEM, IBF, IER)
  410 CONTINUE
      CALL D2ERAS (CMEM, IMEM, DMEM, IPCUTS, IER)
      IER = 0
      RETURN
      END
```

TRAVERSE TRIMMED BOUNDARY 14

14.1 PROBLEM STATEMENT

Given a Trimmed Surface entity and a chord-height tolerance, find a series of points on the boundary of the trimmed surface which includes all edge endpoints of all loops and enough additional points from each edge to ensure that the polyline drawn through the list of points does not deviate from the "true" boundary by more than the given tolerance.

14.2 PROBLEM ANALYSIS

This is basically an application of the D2SCHT (Spline Chord Height Tolerance) subroutine. All the complexity stems from tracing through the data structure to get all the pieces and from possibly dealing with compositions of functions rather than pure splines.

One instance where an edge is normally expressed as a composition of functions occurs when a trimming edge arises from an intersection of surfaces computed by D2SSXT. The intersection curve produced directly by D2SSXT has four dependent variables, two parameter values for each surface. When it is to be used as a trimming edge in the first surface, it should be composed with a "select and permute" function which selects the first two dependent variables. When it is to be used as a trimming edge in the second surface, it should be composed with a "select and permute" function which selects the second two dependent variables. When building a Joined Surface entity, these two edges are naturally linked as "joined" edges.

14.3 EXAMPLE CODE

```
      SUBROUTINE EXPL12 (CMEM, IMEM, DMEM, ITS, TOL, NLP, IPTS, IER)
C     Locate a string of points on the boundary of trimmed surface ITS
C     satisfying a chord-height tolerance TOL.  Output NLP is the number
C     of disjoint boundaries, and IPTS is a pointer array containing
C     pointers to the lists of model space points for each boundary, each
C     list expressed as a DM entity whose columns are the points.
C
C     This example leaves out the necessary IER checking after each Library
C     call for the sake of expository simplicity.
C
      CHARACTER CMEM*(*)
      INTEGER IMEM(*), ITS, NLP, IPTS, IER
      DOUBLE PRECISION DMEM(*), TOL
C
C     Type identifier constants
      INTEGER ENTBF, ENTCF, KEYSA
      PARAMETER (ENTBF=246, ENTCF=244, KEYSA=-12321)
C
C     Local Variables
      CHARACTER SUFIC*1, EGFIC*1
      INTEGER ILP, IXLP, ISURF, ITYP, NOV, NEG, IPA, I, J, K, IEG, IEFU, IECF
      INTEGER IDA, NEPT, NLPT, IDM, JDM, LSA(8), LASDM
      DOUBLE PRECISION TMP, DUM
C
```

```
C     Extract essential data about the trimmed surface
      CALL D2FEEI (CMEM, IMEM, DMEM, ITS, 5, NLP, IER)
      CALL D2ADF (CMEM, IMEM, DMEM, 'P', NLP, IPTS, IER)
      CALL D2FEEI (CMEM, IMEM, DMEM, ITS, 1, ISURF, IER)
      CALL D2FEET (CMEM, IMEM, DMEM, ISURF, ITYP, IER)
      IF (ITYP .EQ. ENTBF) THEN
         CALL D2FEED (CMEM, IMEM, DMEM, ISURF, 2, TMP, IER)
         NOV = TMP
         IF (NOV .LT. 0)  NOV = -NOV - 1
         SUFIC = 'B'
      ELSE IF (ITYP .EQ. ENTCF) THEN
         CALL D2FEEI (CMEM, IMEM, DMEM, ISURF, 2, NOV, IER)
         SUFIC = 'C'
      ELSE
         CALL FAILED ('EXPL11: Unknown surface type in ITS', -32, IER)
      ENDIF
C
C     Process the loops individually
      DO 200 I=1,NLP
         CALL D2STEI (CMEM, IMEM, DMEM, 0, I, IPTS, IER)
         CALL D2TSLP (CMEM, IMEM, DMEM, ITS, I, ILP, IER)
         IF (ILP .EQ. 0) GOTO 200
         CALL D2FEEI (CMEM, IMEM, DMEM, ILP, 3, NEG, IER)
         CALL D2ADF (CMEM, IMEM, DMEM, 'P', 2*NEG, IPA, IER)
C
C        Process the edges in the loop
         NLPT = 0
         DO 100 J=1,NEG
            CALL D2STEI (CMEM, IMEM, DMEM, 0, J, IPA, IER)
            CALL D2LPEG (CMEM, IMEM, DMEM, ILP, J, IEG, IER)
            IF (IEG .EQ. 0) GOTO 100
C
C           Extract the geometric edge and build the function expressing
C           it in model space
            CALL D2FEEI (CMEM, IMEM, DMEM, IEG, 1, IEFU, IER)
            CALL D2FEET (CMEM, IMEM, DMEM, IEFU, ITYP, IER)
            IF (ITYP .EQ. ENTBF) THEN
               EGFIC = 'B'
            ELSE IF (ITYP .EQ. ENTCF) THEN
               EGFIC = 'C'
            ELSE
               CALL FAILED ('EXPL11: Unknown curve type in edge',-33,IER)
C              Clean up should be done here.
               RETURN
            ENDIF
            CALL D2POSE (CMEM, IMEM, DMEM, EGFIC, IEFU, SUFIC, ISURF, ICF,
     +                   IER)
C
C           Find parameter locations of points on this edge
            CALL D2SCHT (CMEM, IMEM, DMEM, ICF, TOL, 1, IDA, NEPT, IER)
C
C           Save the parameter locations and model space edge curve for later
            CALL D2STEI (CMEM, IMEM, DMEM, IDA, J, IPA, IER)
            CALL D2STEI (CMEM, IMEM, DMEM, ICF, NEG+J, IPA, IER)
C
C           Accumulate the total number of points on this loop
            NLPT = NLPT + NEPT
  100    CONTINUE
C
C        Allocate space for the list of model space points for this loop
C        Ensure closure by repeating the initial point at the end
```

```
         NLPT = NLPT + 1
         CALL D2MDF (CMEM, IMEM, DMEM, 'D', NOV, NLPT, IDM, IER)
         CALL D2STEI (CMEM, IMEM, DMEM, IDM, I, IPTS, IER)
C
C        Treat IDM as a one-dimensional array, and use the fact that elements
C        in a column are stored consecutively, to set up a sub-array access
C        into the matrix IDM.  The model space points will be stored directly
C        in IDM.  The "sub-array access" list is described in Reference Manual
C        2.2.14.
         LSA(1) = KEYSA
         LSA(2) = NOV
         LSA(3) = 1
         LSA(4) = NOV*NLPT
         LSA(5) = 1
         LSA(6) = 0
         LSA(7) = IDM
         LSA(8) = 3
C
C        Cycle through the edges again, evaluating to obtain model space
C        points
         DO 150 J=1,NEG
            CALL D2FEEI (CMEM, IMEM, DMEM, IPA, J, IDA, IER)
            CALL D2FEEI (CMEM, IMEM, DMEM, IPA, NEG+J, ICF, IER)
            CALL D2FELD (CMEM, IMEM, DMEM, IDA, NEPT, IER)
C
C           Evaluate the points on this edge, accumulating them in IDM at
C           the location indicated by LSA
            DO 140 K=1,NEPT
               CALL D2FEED (CMEM, IMEM, DMEM, IDA, K, TMP, IER)
               CALL D2EVL1 (CMEM, IMEM, DMEM, ICF, TMP, LSA, DUM, IER)
               LSA(3) = LSA(3) + NOV
  140       CONTINUE
C
C           Clean up as we go, erasing the parameter values for each edge
            CALL D2ERAS (CMEM, IMEM, DMEM, IDA, IER)
C           and the composition function (but NOT the edge and surface
C           components of the CF, which are part of the Trimmed Surface!)
            CALL D2ERAS (CMEM, IMEM, DMEM, ICF, IER)
  150    CONTINUE
C
C        Copy the first point exactly, to make sure the loop is closed.
         CALL D2FEBD (CMEM, IMEM, DMEM, IDM, JDM, LASDM, IER)
         LASDM = LASDM - NOV + 1
         DO 170 J=0,NOV-1
            DMEM(LASDM+J) = DMEM(JDM+J)
  170    CONTINUE
         CALL D2UNLD (CMEM, IMEM, DMEM, IDM, IER)
C
C        Clean up for current loop
         CALL D2ERAS (CMEM, IMEM, DMEM, IPA, IER)
  200 CONTINUE
C
      IER = 0
      RETURN
      END
```

TRIMMED & JOINED - DT_NURBS VS IGES 15

15.1 PROBLEM STATEMENT

Place a DT_NURBS Library Joined Surface entity in an IGES file.

15.2 PROBLEM ANALYSIS

The short answer is that it can't be done at this time. The nearest thing to a Library Joined Surface (JS) entity in IGES is a "Shell Entity" (Type 514). The Shell Entity and its component entities are still in "proposed, but not yet formally adopted" status with respect to IGES. There are no Library routines yet for creating IGES Shell, Face, Loop, Edge or Vertex entities. Moreover, IGES Shell entities are required to be closed surfaces, while Library Joined Surfaces are not required to be closed.

A Library Joined Surface is essentially just a list of Trimmed Surfaces plus some edge connecting information. For Trimmed Surfaces there is another IGES entity, the "Bounded Surface Entity" (Type 143), which the Library does support to some degree. The basic ideas behind a Library Trimmed Surface and an IGES Bounded Surface Entity are the same, but the technical details vary enough that there are examples of each which do not translate into the other.

The cases which the Library routines D2TSIG (Trimmed Surface to IGes) and D2IGTS (IGes to Trimmed Surface) can successfully translate are those in which the "untrimmed surface" is represented by a B-spline Function and all components of the boundary curves are represented by B-spline Functions. IGES also permits non-parametric surfaces and non-B-spline surfaces and boundary curves. On the other side, the Library is expanding its original definition of allowed surface and curve components to include Composition of Functions (CF) entities, and, in the case of the surface, a Geometry and Analysis (GA) entity. These latter entity types can only be translated into IGES using "vendor-defined entities" which extend the standard. The expansion to allow CF and GA entities in TS and JS entities is underway but not yet available in version 2.5.

15.3 EXAMPLE CODE

Given the state of flux of both IGES and the Library on this issue, creating an example is not appropriate at this time.

TRIMMED TO SUBRANGE 16

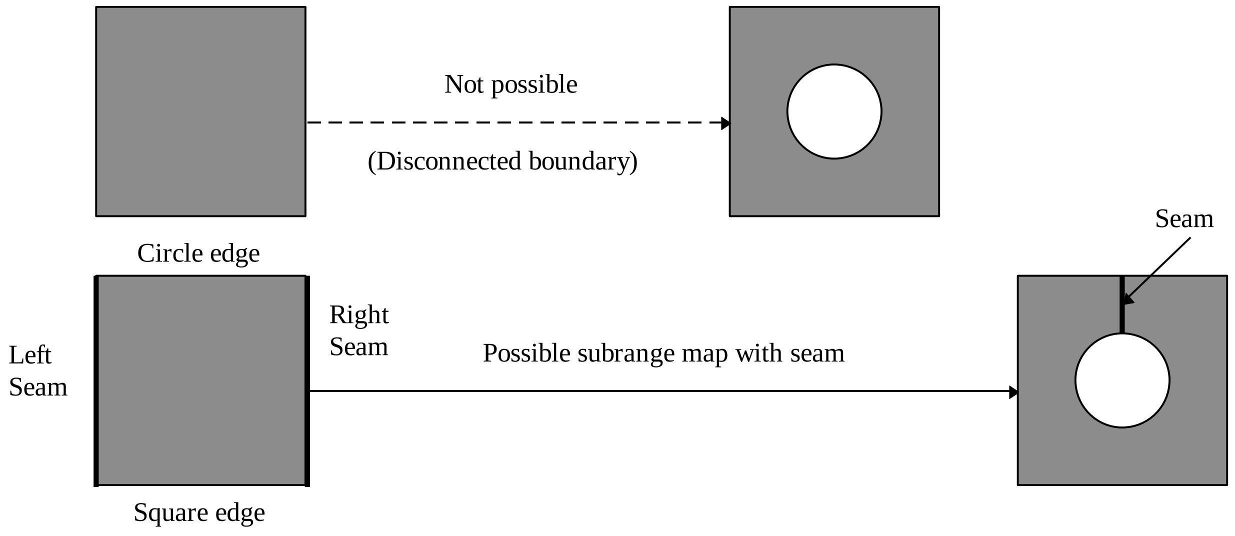
16.1 PROBLEM STATEMENT

Construct a subrange surface matching a given trimmed surface.

16.2 PROBLEM ANALYSIS

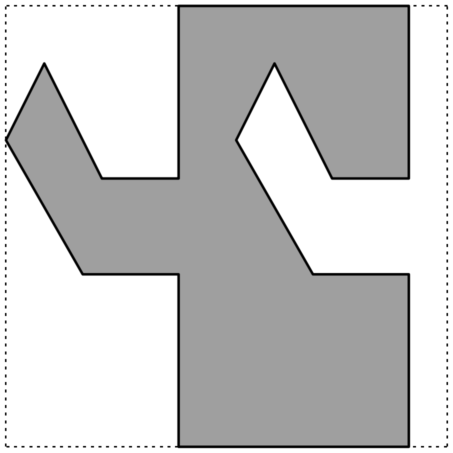
Stated more precisely, the problem is to construct a spline function which maps a coordinate rectangle in the plane (usually the unit square) onto the active region in the parameter domain of the given trimmed surface.

If the active region of the trimmed surface has holes, there will be a problem. If it is permissible for the subrange image to have seams (i.e. places where parts of the edges of the image overlap), the task is still possible. See the diagram below.



Making a subrange map to an active region with a hole

If the boundary of the active region of the trimmed surface is very convoluted, the prospects for a satisfactory subrange mapping become dim. Consider the active region indicated below. The natural subrange construction results in a mapping which doubles back on itself and spreads outside the active region, as shown below.



Convoluted active region of Trimmed Surface

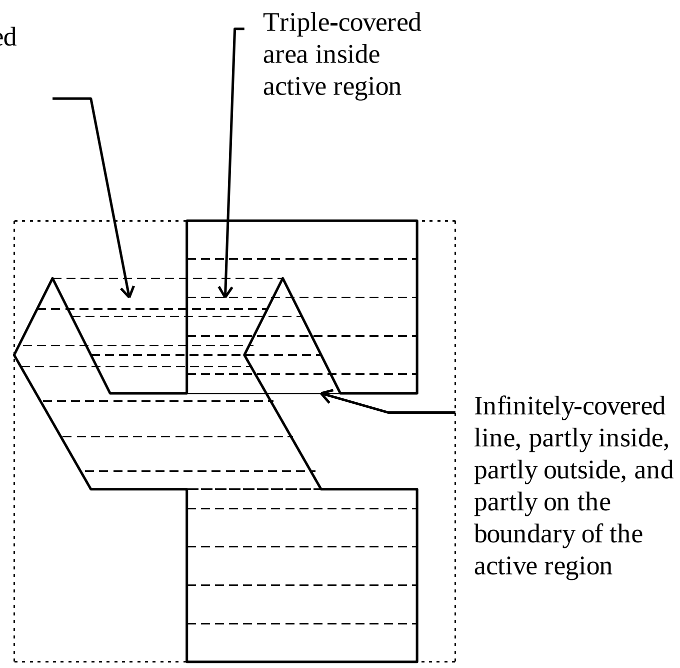

Double-covered area outside active region
Triple-covered area inside active region
Infinitely-covered line, partly inside, partly outside, and partly on the boundary of the active region

Image of subrange mapping resulting from natural construction

No
w suppose there are no holes and the boundary is not very convoluted. The basic Library subroutine for building a subrange mapping is D2MPBC (MaP to Boundary Curves). It requires a boundary consisting of exactly four curves. These four curves will become the images of the four edges of the unit square. If the boundary of the active region happens to contain exactly four curves, these become natural choices as inputs to D2MPBC. If the result is not satisfactory, one can rearrange the boundary as described below and try again.

If the boundary does not consist of four curves, or if the original set of four curves does not give good results, the user must choose four "corner" points and reformulate the boundary between successive corner points as single B-spline curves. If any of the original boundary edges are expressed as Compositions of Functions (CF), they can be converted into pure B-spline Functions (BF) by D2BINE. Where an original boundary edge needs to be divided because it contains a corner point, D2STRM (Spline TRiM) can be applied. Where two consecutive B-spline curves need to be combined into one, D2SPJN (SPline JoiN) can be used. If some of the original edges are rational splines, additional problems may arise. In this case, D2MGKT (MerGe KnoT) may be helpful.

Unless the specific application provides some a priori knowledge about locating good choices for corner points, it is very difficult to write an automatic procedure for building subrange surfaces.

16.3 EXAMPLE CODE

TBD

COMPUTING MOMENTS OF INERTIA 17

17.1 PROBLEM STATEMENT

Compute mass, centroid, and moments of inertia for uniform solids given a tool for computing

$$\int_\Omega x_1^\alpha x_2^\beta x_3^\gamma \, dV$$

over a solid Ω.

17.2 PROBLEM ANALYSIS

Given a B-rep solid represented by a joined surface (JS) entity, use DT_NURBS subroutine D2JSMP to compute

$$\int_\Omega x^{m_x} y^{m_y} z^{m_z} \, dV$$

where *x*, *y*, and *z* are the components of points in the solid and Ω is the solid. These integrals make it possible to compute all the mass properties of the solid.

Given a spline mapping from $R^k \to R^n$ for k=3 and n arbitrary, use DT_NURBS subroutine D2MOIN to compute

$$\int_\Omega x_1^{m_1} x_2^{m_2} \cdots x_n^{m_n} \, dV$$

where the x_i are the components of the mapping and Ω is the domain of the mapping. The mass properties of the solid may be computed from this routine.

17.2.1 MASS AND VOLUME

Computing the volume is easy. Just set (α, β, γ) = (0,0,0) and compute

$$v = \int_\Omega dV \, .$$

The mass *m* is just the product of the volume with the density, i.e.

$$m = \rho v.$$

17.2.2 CENTER OF MASS

Computing the centroid is almost as easy. To get $\bar{x}_1$, compute

$$\int_\Omega x_1 dV$$

and divide the result by *v*. Similarly, $\bar{x}_2$ can be computed by calculating

$$\int_\Omega x_2 dV$$

and dividing the result by *v*. The computation for $\bar{x}_3$ is analogous. Compute

$$\int_\Omega x_3 dV$$

and divide by *v*. These three quantities are the three components of the centroid

$$\bar{x} = \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \end{pmatrix}.$$

17.2.3 MOMENT OF INERTIA

Now consider the moment of inertia about an axis *w* through the origin. If *r* is the distance of any point $x = (x_1, x_2, x_3)^T$ from this axis, then the inertia can be computed as:

$$\begin{aligned}\int_\Omega r^2 dV &= \int_\Omega [(x \cdot x) - (x \cdot w)^2] dV \\ &= w \cdot Jw\end{aligned},$$

where *J* is the 3 × 3 matrix whose entries are given by

$$J_{ij} = \int_\Omega [\delta_{ij}(x_1^2 + x_2^2 + x_3^2) - x_i x_j] dV.$$

In this expression, $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. The matrix *J* is called the **inertia tensor**, and its entries can be computed by evaluating (and combining) the integrals

$$M_{ij} = \int_\Omega x_i x_j dV.$$

Specifically, the J_{ij} are given by

$$J_{ij} = \delta_{ij}(M_{11} + M_{22} + M_{33}) - M_{ij}.$$

Given this matrix, it is possible to compute the moment of inertia $I(w, p)$ about any point *p* and any rotational axis *w* using the Parallel Axis Theorem. It is given by

$$I(w, p) = w \cdot Jw + v(p - 2\bar{x}) \cdot (p - (p \cdot w)w).$$

17.2.4 RADIUS OF GYRATION

The radius of gyration $k(w, p)$ can be computed from the equation

$$I(w, p) = vk(w, p)^2.$$

Thus, it is given by

$$k(w,p) = \sqrt{\frac{I(w,p)}{v}}\,.$$

LICENSE TERMS AND CONDITIONS
APPENDIX A

DT_NURBS SPLINE GEOMETRY SUBROUTINE LIBRARY LICENSE

Terms and Conditions

The DT_NURBS Spline Geometry Subroutine Library, referred henceforth as DT_NURBS, is distributed under the terms and conditions set forward in this document.

1. Licensee shall use reasonable efforts not to disclose the DT_NURBS source files, not including documentation files, sample code, or test codes distributed with DTNURBS, other than to its employees, staff, faculty, and students, and shall not authorize the removal of DT_NURBS from Licensee's premises. The Naval Surface Warfare Center, Carderock Division, at David Taylor Model Basin, or an agency, institution, or corporation appointed by NSWC to distribute this code, are the only parties authorized to distribute library source files. Exceptions are allowed for purposes of distributing, within the United States, applications in source format written using the DT_NURBS library. In this circumstance only those DT_NURBS routines, header, or include files, or other related source files used by the application may be distribute in source format.
2. The software is provided "as is", without any warranty by NSWC, or its contractors. In no event shall NSWC, or its contractors, be liable for any loss or for any indirect, special, punitive, exemplary, incidental, or consequential, damages arising from use, possession, or performance of the software.
3. When bugs or problems are found, the licensee will make a reasonable effort to report them to NSWC at the address stated on this document.
4. The licensee has full rights to any software developed with this library, whether commercial, academic, or internal use only.
5. This software library, is not available to foreign nationals, or foreign owned companies at this time. Foreign nationals working or studying in a U.S. company or U.S. academic institution are authorized to use the library under the recognizance of the Licensee. Licensee agrees to instruct its employees, staff, faculty, and students who will have access to the software as to their obligations under Paragraph 1 of this agreement.
6. Applications developed with this software can be distributed outside the United States freely provided the DT_NURBS routines used in the application are in an executable binary format.
7. To obtain a licensed copy of DT_NURBS you must sign this form and submit it along with a written request to the address below. Please make a copy of this agreement for your records and reference.

____________________________/__________

Printed Name of Licensee / Date

____

Postal Address

____

E-mail Address

_________________/__________________

Phone / Fax

____________________________/___________

Signature of Licensee / Date

All requests should be sent to:

Bob Ames   ames@oasys.dt.navy.mil
NSWC/Carderock Division
David Taylor Model Basin
Code 50
Bethesda, Md. 20084-5000   (301)227-1339(ph) (301)227-1125(fax)