Contents

Chapter 1

Introduction

This manual describes the different libraries of the IRIT solid modeling environment. Quite a few libraries can be found to manipulate geometry in general, freeform curves and surfaces, symbolic computation, trimmed surfaces, freeform trivariate functions, Boolean operations, input output data file parsing, and miscelleneous.

All interface to the libraries should be made via the appropriate header files that can be found in the include subdirectory. Most libraries have a single header file that is named the same as the library. Functions and constants that are visible to the users of the libraries are prefixed with a unique prefix, usually derived from the library name itself. External definitions that start with an underscore should not be used, even if found in header files.

The header file **include/irit_sm.h** must be sourced by every source file in the solid modeller. In most cases, this file is sourced indirectly via local header files.

The following libraries are avaliable in IRIT:

Name of Library	Tasks
bool	Boolean operations on polygonal models.
cagd	Low level freeform curves and surfaces.
geom	General geometry functions.
misc	memory allocation, configuration files, attributes, etc.
prsr	input and output for file/sockets of objects of IRIT.
symb	Symbolic manipulation of curves and surfaces.
trim	Trimmed surfaces support.
triv	Freeform trivariate functions.
xtra	Public domain code that is not part of IRIT.

The following chapters reference the different function of different libraries. Chapter 11 provides several examples of writing C code using the IRIT libraries.

Chapter 2

Boolean Library, bool_lib

2.1 General Information

On of the crucial operation in any solid modeling environment is the ability to perform Boolean operations among different geometric objects. The interface of the library is defined in *include/bool_lib.h*. This library supports only Boolean operations of polygonal objects. The Boolean operations of OR, AND, SUBtract, NEGate, CUT, and MERGE are supported via the **BoolOperType** typedef:

BoolOperType	Meaning
BOOL_OPER_OR	Union of two geometrical objects
BOOL_OPER_AND	Intersection of two geometrical objects
BOOL_OPER_SUB	The difference of two geometrical objects
BOOL_OPER_NEG	Unary Inside-out of one geometrical object
BOOL_OPER_CUT	Boundary of one object outside the other
BOOL_OPER_MERGE	Simple merge without any computation

The **BoolOperType** typedef is used in two dimensional Boolean operations invoked via **Boolean2D**. Three dimensional Boolean operations are invoked via **BooleanXXX** where XXX is one of OR, AND, SUB, NEG, CUT or MERGE, denoting the same as in the table above.

In addition several state functions are available to control the way the Boolean operations are conducted. Functions to enable the dump of (only) the intersection curves, to handle coplanar polygons, and to set the axis along which the sweep is performed are provided.

All globals in this library have a profix of **Bool**.

2.2 Library Functions

2.2.1 BoolExtractPolygons (bool_lib/bool2low.c:887)

Booleans

```
IPObjectStruct *BoolExtractPolygons(IPObjectStruct *PObj, int AinB)
```

PObj: Object that need to be rebuilt according to the intersection curves that were found, both closed and open loops.

AinB: Type of inclusion/exclusion requested.

Returns: The newly created clipped object.

Description: This routine coordinates all the extraction of the polygons from the intersecting lists. Does it in the following steps:

1. Mark all polygons with no intersection at all as complete polygons. (this is because this polygon will be totally in or out, according to inter-polygon adjacencies propagation...) Also mark them as undefined (if in output or undefined) yet. Uses IPPolygonStruct Tags to save these bits.
2. do
    - 2.1. Convert the unordered segment list of each polygon to closed loops (if create a hole in polygon) or open (if crosses its boundary).
    - 2.2. Order the open loops along the perimeter of the polygon (note these loops cannt intersect. For example (5 vertices polygon):

```
          -----------------------------L3
         |  ---------------L1  -----L2 |          --------L4   --L5
         | |               |   |      | |          |        |    |  |
P0 ------ P1 ------- P2 ----- P3 -------- P4 ------ P5 -------- P0
```

Note L1, L2 are enclosed in L3 loop, and that the order is circular.

2.3. "Break" the polygon at each open loop that has no enclosed loops in it. For example we can start at L1, L2, L4, L5 and then L3. "Break" means - replace the vertex chain between the two loop end points, with the loops itself. Depends upon the relation required we may need to output a new polygon form from the deleted chain and that loop. In addition we may form a new polygon from last loop and was was left from the original polygon For each formed polygon, for each complete edge of it (i.e. edge which was originally in the polygon) test the adjacent polygon if it is complete (as marked in 1.) and if in or undefined (marked undefined in 1.) is still undefined:

2.3.1. set it to be in.

2.3.2. push it on adjacency stack.

2.4. For each closed loop - find in which polygon (from polygons created in 2.3.) it is enclosed, and decompose it.

3. While adjacencies stack not empty do:

3.1. pop top polygon from stack and output it.

3.2. For each of its edges (which obviousely must be complete edges) if adjacent polygon is complete and undefined:

3.3.1. set it to be in.

3.3.2. push it on adjacency stack.

3.3 go back to 3.

The above algorithm defines in as in output, but dont be confused with the required inter-object AinB (or AoutB if FALSE), which used to determine which side of the trimming loop should be output. Note this routine may return non-convex polygons (but marked as so) even though the input for the booleans must be convex polygons only! In order to keep the given object unchanged, a whole new copy off the polygon list is made. The polygons of the list that are not in the output are freed: a global list of all polygons (pointers) is used to scan them in the end and free the unused ones (list PolysPtr).

2.2.2 BoolGenAdjacencies (bool_lib/adjacncy.c:89)

adjacency

topology

```
int BoolGenAdjacencies(IPObjectStruct *PObj)
```

PObj: The polygonal object to compute the adjacency information for.

Returns: TRUE if all adjacencies were resolved, or the object is completely closed.

Description: Routine to generate adjacencies to the given object. Note an edge might be only partially adjacent to another edge, and a second attempt is made to find (again only part of - see below) them. Any case, FALSE will be returned as there is no way we can say the object is perfectly closed! This is the only routine to generate the adjacencies of a geometric object. These adjacencies are needed for the boolean operations on them. Algorithm: for each edge, for each polygon in the object, the edges are sorted according to the key defined by EdgeKey routine (sort in hash tbl). A second path on the table is made to match common keys edges and set the pointers from one to another. Note that each edge is common to exactly 2 faces if it is internal, or exactly 1 face if it is on the border (if the object is open).

2.2.3 BoolLoopsFromInterList (bool_lib/bool1low.c:728)

```
void BoolLoopsFromInterList(IPPolygonStruct *Pl,
                            InterSegListStruct **PClosed,
                            InterSegListStruct **POpen)
```

Pl: Polygon with intersection information in its PAux slot.

PClosed: To be updated with the closed loops found in Pl.

POpen: To be updated with the open loops found in Pl.

Description: Given a polygon with the intersection list, creates the polylines loop(s) out of it, which can be one of the two:

1. Closed loop - all the intersections create a loop in one polygon.

2. Open polyline - if the intersections cross the polygon boundary. In this case the two end point of the polyline, must lay on polygon boundary.

In both cases, the polyline will be as follows: First point at first list element at PtSeg[0] (see InterSegmentStruct). Second point at first list element at PtSeg[1] (see InterSegmentStruct). Point i at list element (i-1) at PtSeg[0] (PtSeg[1] is not used!). In the closed loop case the last point is equal to first. Both cases returns NULL terminated list.

2.2.4 BoolSetHandleCoplanarPoly (bool_lib/bool-hi.c:742)

Booleans

```
void BoolSetHandleCoplanarPoly(int HandleCoplanarPoly)
```

HandleCoplanarPoly: If TRUE, coplanar polygons are handled.

Description: Controls if coplanar polygons should be handled or not.

2.2.5 BoolSetOutputInterCurve (bool_lib/bool-hi.c:724)

Booleans

```
void BoolSetOutputInterCurve(int OutputInterCurve)
```

OutputInterCurve: If TRUE only intersection curves are computed, If false, full blown Boolean is applied.

Description: Controls if intersection curves or full Boolean operation is to be performed.

2.2.6 BoolSetPolySortAxis (bool_lib/bool1low.c:367)

Booleans

```
void BoolSetPolySortAxis(int PolySortAxis)
```

PolySortAxis: Sorting axis. Either 0(x), 1 (y), or 2 (z).

Description: Routine to set polygonal sorting axis.

2.2.7 BoolSortOpenInterList (bool_lib/bool1low.c:944)

```
void BoolSortOpenInterList(IPPolygonStruct *Pl, InterSegListStruct **POpen)
```

Pl: To sort the loops for.

POpen: The set of open loops. Updated in place.

Description: Sorts the open loops of given polygon to an order that can be used in subdividing into sub polygons later (see comment of BoolExtractPolygons). This order is such that each loops will have no other loop between its end points, if we walk along the polygon in the (linked list direction) perimeter from one end to the other, before it. For example:

```
             -----------------------------L3
            |  ---------------L1  -----L2 |           --------L4   --L5
            | |               |  |      | |          |        |    |  |
     P0 ------ P1 ------- P2 ----- P3 -------- P4 ------ P5 -------- P0
```

In this case, any order such that L1, L2 are before L3 will do. Obviously this is not a total order, and they are few correct ways to sort it. Algorithm: For each open loop, for each of its two end, evaluate a RealType key for the end point P between segment P(i) .. P(i+1) to be i + t, where: t is the ratio (P - P(i)) / (P(i+1) - P(i)) . This maps the all perimeter of the polygon onto 0..N-1, where N is number of vertices of that polygon. Sort the keys, and while they are keys in data sturcture, search and remove a consecutive pair of keys assosiated with same loop, and output it. Note that each open loop point sequence is tested to be such that it starts on the first point (first and second along vertex list) on polygon perimeter, and the sequence end is on the second point, and the sequence is reversed if not so. This order will make the replacement of the perimeter from first to second points by the open loop much easier. This may be real problem if there are two intersection points almost identical - floating point errors may cause it to loop forever. We use some reordering heuristics in this case, and return fatal error if fail!

2.2.8 Boolean2D (bool_lib/bool-2d.c:70)

Booleans

```
IPPolygonStruct *Boolean2D(IPPolygonStruct *Pl1,
                           IPPolygonStruct *Pl2,
                           BoolOperType BoolOper)
```

Pl1: First polygon to compute 2D Boolean for.

Pl2: Second polygon to compute 2D Boolean for.

BoolOper: Boolean operation requested (and, or, etc.)

Returns: The resulting Boolean operation.

Description: Given two polygons assumed to be in the same plane, compute their 2D Boolean operation BoolOper and return it as a new polygon. NULL is returned if an error occur (No intersection or invalid BoolOper).

2.2.9 BooleanAND (bool_lib/bool-hi.c:189)

Booleans

```
IPObjectStruct *BooleanAND(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object to perform the Boolean operation on.

PObj2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean AND between two objects.

2.2.10 BooleanCUT (bool_lib/bool-hi.c:275)

Booleans

```
IPObjectStruct *BooleanCUT(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object to perform the Boolean operation on.

PObj2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean CUT between two objects.

2.2.11 BooleanICUT (bool_lib/bool-hi.c:314)

Booleans

```
IPObjectStruct *BooleanICUT(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object to perform the Boolean operation on.

PObj2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean Inside CUT between two objects.

2.2.12 BooleanLow1In2 (bool_lib/bool1low.c:114)

Booleans

```
IPObjectStruct *BooleanLow1In2(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object of Boolean operation.

PObj2: Second object of Boolean operation.

Returns: Result of one in two.

Description: Finds the part of PObj1 which is in of PObj2:

2.2.13 BooleanLow1Out2 (bool_lib/bool1low.c:80)

Booleans

```
IPObjectStruct *BooleanLow1Out2(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object of Boolean operation.

PObj2: Second object of Boolean operation.

Returns: Result of one out two.

Description: Finds the part of PObj1 which is out of PObj2:

2.2.14 BooleanMERGE (bool_lib/bool-hi.c:353)

Booleans

```
IPObjectStruct *BooleanMERGE(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object to perform the Boolean operation on.

PObj2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean MERGE between two objects.

2.2.15 BooleanNEG (bool_lib/bool-hi.c:389)

Booleans

```
IPObjectStruct *BooleanNEG(IPObjectStruct *PObj)
```

PObj: Object to negate.

Returns: The result of the Boolean operation.

Description: Performs a Boolean NEG between two objects. Negation is simply reversing the direction of the plane equation of each polygon - the simplest Boolean operation...

2.2.16 BooleanOR (bool_lib/bool-hi.c:132)

Booleans

```
IPObjectStruct *BooleanOR(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object to perform the Boolean operation on.

PObj2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean OR between two objects.

2.2.17 BooleanSUB (bool_lib/bool-hi.c:229)

Booleans

```
IPObjectStruct *BooleanSUB(IPObjectStruct *PObj1, IPObjectStruct *PObj2)
```

PObj1: First object to perform the Boolean operation on.

PObj2: Second object to perform the Boolean operation on.

Returns: The result of the Boolean operation.

Description: Performs a Boolean SUBtracion between two objects.

Chapter 3

CAGD Library, cagd_lib

3.1 General Information

This library provides a rich set of function to create, convert, display and process freeform Bezier and NURBs curves and surfaces. The interface of the library is defined in *include/cagd_lib.h*. This library mainly supports low level freeform curve and surface operations. Supported are curves and surfaces from scalars to five dimensions as E1/P1 to E5/P5 using the **CagdPointType**. Pi is a rational (projective) version of Ei, with an additional W coefficient. Polynomial in the power basis have some very limited support as well. Different data structures to hold UV parameter values, control points, vectors, planes, bounding boxes, polylines and polygons are defined as well as the data strcutures to hold the curves and surfaces themselves,

```
typedef struct CagdCrvStruct {
    struct CagdCrvStruct *Pnext;
    struct IPAttributeStruct *Attr;
    CagdGeomType GType;
    CagdPointType PType;
    int Length;            /* Number of control points (== order in Bezier). */
    int Order;     /* Order of curve (only for Bspline, ignored in Bezier). */
    CagdBType Periodic;    /* Valid only for Bspline curves. */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType *KnotVector;
} CagdCrvStruct;

typedef struct CagdSrfStruct {
    struct CagdSrfStruct *Pnext;
    struct IPAttributeStruct *Attr;
    CagdGeomType GType;
    CagdPointType PType;
    int ULength, VLength;  /* Mesh size in the tensor product surface. */
    int UOrder, VOrder;   /* Order in tensor product surface (Bspline only). */
    CagdBType UPeriodic, VPeriodic;      /* Valid only for Bspline surfaces. */
    CagdRType *Points[CAGD_MAX_PT_SIZE];     /* Pointer on each axis vector. */
    CagdRType *UKnotVector, *VKnotVector;
} CagdSrfStruct;
```

Curves and surfaces have a geometric type **GType** to prescribe the type of entity (such as CAGD_SBEZIER_TYPE for Bezier surface) and a point type **PType** to prescribe the point type of the entity (such as CAGD_PT_E3_TYPE for three dimensional Euclidean control points). **Length** and **Order** slots are used to hold the number of control points in the mesh and or control polygon and the order(s) of the basis functions. **Periodic** flag(s) are used to denote periodic end conditions. In addition, **KnotVector** slot(s) are used if the entity exploits Bspline basis functions, or NULL otherwise.

The control polygon and/or mesh itself is organized in the **Points** slot as a vector of size **CAGD_MAX_PT_SIZE** of vectors of **CagdRTypes**. For surfaces, the mesh is ordered U first and the macros of **CAGD_NEXT_U** **CAGD_NEXT_V**, and **CAGD_MESH_UV** can be used to determine indices in the mesh.

All structures in the cagd library can be allocated using New constrcutures (i.e. **CagdUVNew** or **CagdCrfNew**, freed using Free destructores (i.e. **CagdSrfFree** or CagdBBoxFree, linked list free using FreeList destructores (i.e. **CagdPolylineFreeList**), and copied using copy constructores i.e. **CagdPtCopy** or **CagdCtlPtCopyList**).

This library has its own error handler, which by default prints an error message and exit the program called **CagdFatalError**.

Most globals in this library have a prefix of **Cagd** for general cagd routines. Prefix of **Bzr** is used for Bezier routines, prefix of **Bsp** for Bspline specific routines, prefix of **Cnvrt** for conversion routines, and **Afd** for adaptive forward differencing routines.

3.2 Library Functions

3.2.1 AfdApplyEStep (cagd_lib/afd_cube.c:166)

forward differencing

```
void AfdApplyEStep(CagdRType Coef[4])
```

Coef: Four coefficients of the AFD basis functions.

Description: Given four coefficents of a cubic afd polynomial, apply the E (step 1) in place.

3.2.2 AfdApplyLn (cagd_lib/afd_cube.c:71)

forward differencing

```
void AfdApplyLn(CagdRType Coef[4], int n)
```

Coef: Four coefficients of the AFD basis functions.

n: How many times to compute the L transform.

Description: Given four coefficents of a cubic afd polynomial, apply the L (half the step size) n times to them, in place. We basically precomputed L^n and apply it here once. Every instance of L half the domain and so L^n divides the domain by 2^n.

3.2.3 AfdBzrCrvEvalToPolyline (cagd_lib/afd_cube.c:240)

forward differencing

```
void AfdBzrCrvEvalToPolyline(CagdCrvStruct *Crv,
                             int FineNess,
                             CagdRType *Points[])
```

Crv: A cubic Bezier curve to piecewuise linear sample using AFD.

FineNess: Of samples.

Points: Whre to put the piecewise linear approximation.

Description: Samples the curves at FineNess location equally spaced in the Bezier parametric domain [0..1]. If Cache is enabled, and FineNess is power of two, upto or equal to CacheFineNess, the cache is used, otherwise the points are evaluated manually for each of the samples. Data is saved at the Points array of vectors (according to Curve PType), each vector is assumed to be allocated for FineNess CagdRType points. Bezier curve must be cubic.

3.2.4 AfdCnvrtCubicBzrToAfd (cagd_lib/afd_cube.c:43)

forward differencing

```
void AfdCnvrtCubicBzrToAfd(CagdRType Coef[4])
```

Coef: Converts, in place, cubic Bezier Coef to AFD Coef.

Description: Given four coefficents of a cubic Bezier curve, computes the four coefficients of the cubic afd basis functions, in place.

3.2.5 AfdComputePolyline (cagd_lib/afd_cube.c:197)

forward differencing

```
void AfdComputePolyline(CagdRType Coef[4],
                        CagdRType *Poly,
                        int Log2Step,
                        CagdBType NonAdaptive)
```

Coef: Four coefficients of a cubic Bezier curve.

Poly: Where to put the polyline computed.

Log2Step: How many steps to take (2 to the power of this).

NonAdaptive: if TRUE, ignore the adaptive option.

Description: Given four coefficents of a cubic Bezier curve, computes the four coefficients of the cubic afd basis functions and step along them to create a piecewise polynomial approximating the curve. If NonAdaptive is TRUE then 2^Log2Step constant steps are taken, creating 2^Log2Step + 1 points along the curve. Otherwise the full blown adaptive algorithm is used.

3.2.6 BspCrv2Polyline (cagd_lib/bsp2poly.c:569)

piecewise linear approximation

polyline

```
CagdPolylineStruct *BspCrv2Polyline(CagdCrvStruct *Crv,
                                    int SamplesPerCurve,
                                    BspKnotAlphaCoeffType *A,
                                    CagdBType OptiLin)
```

Crv: To approximate as a polyline.

SamplesPerCurve: Number of samples to approximate with.

A: Alpha matrix (Oslo algorithm) if precumputed.

OptiLin: If TRUE, optimize linear curves.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single Bspline curve as a polyline with SamplesPerCurve samples. Polyline is always E3 CagdPolylineStruct type. Curve is refined equally spaced in parametric space, unless the curve is linear in which the control polygon is simply being copied. If A is specified, it is used to refine the curve. NULL is returned in case of an error, otherwise CagdPolylineStruct.

3.2.7 BspCrvBiNormal (cagd_lib/cbsp_aux.c:510)

binormal

```
CagdVecStruct *BspCrvBiNormal(CagdCrvStruct *Crv, CagdRType t)
```

Crv: Crv for which to compute a unit binormal.

t: The parameter at which to compute the unit binormal.

Returns: A pointer to a static vector holding the binormal information.

Description: Returns a unit vector, equal to the binormal to Crv at parameter value t. Algorithm: insert (order - 1) knots and using 3 consecutive control points at the refined location (p1, p2, p3), compute to binormal to be the cross product of the two vectors (p1 - p2) and (p2 - p3). Since a curve may have not BiNormal at inflection points or if the 3 points are colinear, NULL will be returned at such cases.

3.2.8 BspCrvCoxDeBoorBasis (cagd_lib/bspcoxdb.c:110)

evaluation

Bsplines

```
CagdRType *BspCrvCoxDeBoorBasis(CagdRType *KnotVector,
                                int Order,
                                int Len,
                                CagdRType t,
                                int *IndexFirst)
```

KnotVector: To evaluate the Bspline Basis functions for this space.

Order: Of the geometry.

Len: Number of control points in the geometry. The length of KnotVector is equal to Len + Order.

t: At which the Bspline basis functions are to be evaluated.

IndexFirst: Index of the first Bspline basis function that might be non zero.

Returns: A vector of length Order thats holds the values of the Bspline basis functions for the given t. A Bspline of order Order might have at most Order non zero basis functions that will hence start at IndexFirst and upto (*IndexFirst + Order - 1).

Description: Returns a pointer to a vector of size Order, holding values of the non zero basis functions of a given curve at given parametric location t. This vector SHOULD NOT BE FREED. Although it is dynamically allocated, the returned pointer does not point to the beginning of this memory and it it be maintained by this routine (i.e. it might be freed next time this routine is being called). IndexFirst returns the index of first non zero basis function for the given parameter value t. Uses the recursive Cox de Boor algorithm, to evaluate the Bspline basis functions. Algorithm: Use the following recursion relation with B(i,0) == 1.

```
          t     -    t(i)            t(i+k)    -   t
B(i,k) = --------------- B(i,k-1) + --------------- B(i+1,k-1)
          t(i+k-1) - t(i)            t(i+k) - t(i+1)
```

Starting with constant Bspline (k == 0) only one basis function is non zero and is equal to one. This is the constant Bspline spanning interval t(i)...t(i+1) such that t(i) <= t < t(i+1). We then raise this constant Bspline to the prescribed Order and find in this process all the basis functions that are non zero in t for order Order. Sound simple hah!?

3.2.9 BspCrvCreateCircle (cagd_lib/cagd_arc.c:172)

circle

```
CagdCrvStruct *BspCrvCreateCircle(CagdPtStruct *Center, CagdRType Radius)
```

Center: Of circle to be created.

Radius: Of circle to be created.

Returns: A circle centered at Center and radius Radius that is parallel to the XY plane represented as a rational quadratic Bspline curve.

Description: Creates a circle at the specified position as a rational quadratic Bspline curve. Circle is always paralell to the XY plane.

3.2.10 BspCrvCreatePCircle (cagd_lib/cagd_arc.c:264)

circle

```
CagdCrvStruct *BspCrvCreatePCircle(CagdPtStruct *Center, CagdRType Radius)
```

Center: Of circle to be created.

Radius: Of circle to be created.

Returns: A circle approximation centered at Center and radius Radius that is parallel to the XY plane represented as a polynomial cubic Bspline curve.

Description: Approximates a circle as a cubic polynomial Bspline curve at the specified position and radius. Construct the circle as four 90 degrees arcs of polynomial cubic Bezier segments using predefined constants. See Faux & Pratt "Computational Geometry for Design and Manufacturing" for a polynomial approximation to a circle.

3.2.11 BspCrvCreateUnitCircle (cagd_lib/cagd_arc.c:131)

circle

```
CagdCrvStruct *BspCrvCreateUnitCircle(void)
```

Returns: A rational quadratic bsplinecurve representing a unit circle.

Description: Creates a circle at the specified position as a rational quadratic Bspline curve. Constructs a unit circle as 4 90 degrees arcs of rational quadratic Bezier segments using a predefined constants.

3.2.12 BspCrvCreateUnitPCircle (cagd_lib/cagd_arc.c:204)

circle

```
CagdCrvStruct *BspCrvCreateUnitPCircle(void)
```

Returns: A cubic polynomial Bspline curve approximating a unit circle

Description: Approximates a unit circle as a cubic polynomial Bspline curve. Construct a circle as four 90 degrees arcs of polynomial cubic Bezier segments using predefined constants. See Faux & Pratt "Computational Geometry for Design and Manufacturing" for a polynomial approximation to a circle.

3.2.13 BspCrvDegreeRaise (cagd_lib/cbsp_aux.c:352)

degree raising

```
CagdCrvStruct *BspCrvDegreeRaise(CagdCrvStruct *Crv)
```

Crv: To raise it degree by one.

Returns: A curve with one degree higher representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with one degree higher.

3.2.14 BspCrvDegreeRaiseN (cagd_lib/cbsp_aux.c:295)

degree raising

```
CagdCrvStruct *BspCrvDegreeRaiseN(CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree to a NewOrder.

NewOrder: NewOrder for Crv.

Returns: A curve of order NewOrder representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with order N. Degree raise is computed by multiplying by a constant 1 curve of order

3.2.15 BspCrvDerive (cagd_lib/cbsp_aux.c:624)

derivatives

```
CagdCrvStruct *BspCrvDerive(CagdCrvStruct *Crv)
```

Crv: To differentiate.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

3.2.16 BspCrvDomain (cagd_lib/bsp_gen.c:192)

domain

parametric domain

```
void BspCrvDomain(CagdCrvStruct *Crv, CagdRType *TMin, CagdRType *TMax)
```

Crv: To get its parametric domain.

TMin: Where to put the minimal domain's boundary.

TMax: Where to put the maximal domain's boundary.

Description: Returns the parametric domain of a Bspline curve.

3.2.17 BspCrvEvalAtParam (cagd_lib/cbspeval.c:92)

evaluation

```
CagdRType *BspCrvEvalAtParam(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Returns a pointer to a static data, holding the value of the curve at given parametric location t. The curve is assumed to be Bspline. Uses the Cox de Boor recursive algorithm.

3.2.18 BspCrvEvalCoxDeBoor (cagd_lib/bspcoxdb.c:36)

evaluation

Bsplines

```
CagdRType *BspCrvEvalCoxDeBoor(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Returns a pointer to a static data, holding the value of the curve at the prescribed parametric location t. Uses the recursive Cox de-Boor algorithm, to evaluate the spline, which is not very efficient if many evaluations of the same curve are necessary Use knot insertion when multiple evaluations are to be performed.

3.2.19 BspCrvEvalVecAtParam (cagd_lib/cbspeval.c:40)

evaluation

```
CagdRType BspCrvEvalVecAtParam(CagdRType *Vec,
                               int VecInc,
                               CagdRType *KnotVector,
                               int Order,
                               int Len,
                               CagdBType Periodic,
                               CagdRType t)
```

Vec: Coefficents of a scalar Bspline univariate function.

VecInc: Step to move along Vec.

KnotVector: Knot vector of associated geoemtry.

Order: Order of associated geometry.

Len: Length of control vector.

Periodic: If this geometry is Periodic.

t: Parameter value where to evaluate the curve.

Returns: Geometry's value at parameter value t.

Description: Assumes Vec holds control points for scalar Bspline curve of order Order length Len and knot vector KnotVector. Evaluates and returns that curve value at parameter value t. Vec is incremented by VecInc (usually by 1) after each iteration.

3.2.20 BspCrvHasBezierKV (cagd_lib/bsp_knot.c:27)

conversion

```
CagdBType BspCrvHasBezierKV(CagdCrvStruct *Crv)
```

Crv: To check for KV that mimics Bezier polynomial curve.

Returns: TRUE if same as Bezier curve, FALSE otherwise.

Description: Returns TRUE iff the given curve has no interior knot open end KV.

3.2.21 BspCrvHasOpenEC (cagd_lib/bsp_knot.c:87)

open end conditions

```
CagdBType BspCrvHasOpenEC(CagdCrvStruct *Crv)
```

Crv: To check for open end conditions.

Returns: TRUE, if curve has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given Bspline curve has open end coditions.

3.2.22 BspCrvIntegrate (cagd_lib/cbsp_aux.c:708)

integrals

```
CagdCrvStruct *BspCrvIntegrate(CagdCrvStruct *Crv)
```

Crv: Curve to integrate.

Returns: Integrated curve.

Description: Returns a new Bspline curve, equal to the integral of the given Bspline crv. The given Bspline curve should be nonrational.

```
            l            l               l       l+1
  /        /-            -      /        -   P    -
 |        | \     n      \     |  n      \    i   \             n+1
 | C(t) = | / P  B (t) = / P   | B (t) = / -----  / ( t   - t ) B   (t) =
/        /  -  i  i      -  i /   i      - n + 1  -    j+n   j   j
           i=0          i=0             i=0     j=i+1
```

```
       l+1 j-1
        -   -
    1   \   \              n+1
 = ----- /   / P  ( t   - t ) B   (t)
   n + 1 -   -  i    j+n   j   j
       j=1 i=0
```

3.2.23 BspCrvInterpPts (cagd_lib/cbsp_int.c:40)

interpolation

least square approximation

```
CagdCrvStruct *BspCrvInterpPts(CagdPtStruct *PtList,
                               int Order,
                               int CrvSize,
                               CagdParametrizationType ParamType)
```

PtList: List of points to interpolate/least square approximate.

Order: Of interpolating/approximating curve.

CrvSize: Number of degrees of freedom (control points) of the interpolating/approximating curve.

ParamType: Type of parametrization.

Returns: Constructed interpolating/approximating curve.

Description: Given a set of points, PtList, computes a Bspline curve of order Order that interpolates or least square approximates the set of points. The size of the control polygon of the resulting Bspline curve defaults to the number of points in PtList (if CrvSize = 0). However, this number is can smaller to yield a least square approximation. The created curve can be parametrized as specified by ParamType.

3.2.24 BspCrvInterpPtsError (cagd_lib/cbsp_int.c:221)

error estimation

interpolation

least square approximation

```
CagdRType BspCrvInterpPtsError(CagdCrvStruct *Crv,
                               CagdPtStruct *PtList,
                               CagdParametrizationType ParamType)
```

Crv: Curve that was fitted to the data set.

PtList: The data set.

ParamType: Parameter values at with curve should interpolate PtList.

Returns: Error measured in the L1 norm.

Description: Given a set of points, and a curve least square fitting them using the BspCrvInterpPts function, computes an error measure as a the maximal distance between the curve and points (L1 norm).

3.2.25 BspCrvInterpolate (cagd_lib/cbsp_int.c:149)

interpolation

least square approximation

```
CagdCrvStruct *BspCrvInterpolate(CagdCtlPtStruct *PtList,
                                 int NumPts,
                                 CagdRType *Params,
                                 CagdRType *KV,
                                 int Length,
                                 int Order,
                                 CagdBType Periodic)
```

PtList: List of points to interpolate/least square approximate.

NumPts: Size of PtList.

Params: At which to interpolate the points in PtList.

KV: Computed knot vector for the constructed curve.

Length: Number of degrees of freedom (control points) of the interpolating/approximating curve.

Order: Of interpolating/approximating curve.

Periodic: If dat a and hence constructed curve should be Periodic.

Returns: Constructed interpolating/approximating curve.

Description: Given set of points, PtList, parameter values the curve should interpolate or approximate these points, Params, and the expected knot vector, KV, length Length and order Order of the Bspline curve, computes the Bspline curve's coefficients. All points in PtList are assumed of the same type. If Periodic, Order - 1 more constraints (and DOF's) are added so that the first Order - 1 points are the same as the last Order - 1 points.

3.2.26 BspCrvKnotInsert (cagd_lib/bspboehm.c:54)

refinement

knot insertion

```
CagdCrvStruct *BspCrvKnotInsert(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To refine by adding a new knot with value equal to t. If Crv is a periodic curve, it is first unwrapped to a float end condition curve.

t: New knot to insert into Crv.

Returns: The refined curve.

Description: Returns a new curve refined at t (t is inserted as a new knot in Crv). If however the multiplicity of t in the current knot vector is equal (or greater!?) to the degree or t is not in the curve's parametric domain, no new knot is insert and NULL is returned instead. Control mesh is updated as follows (P is old ctl polygon, Q is new): Let Index be the last knot in old knot vector less than t and let j be j = Index - order + 1. Also let k be the curve order. Then,

```
Case 1: Q(i) = P(i), i <= j



               t     -    t(i)        t(i+k-1)  -   t
case 2: Q(i) = --------------- P(i) + --------------- P(i-1), j<i<=Index
               t(i+k-1) - t(i)        t(i+k-1) - t(i)



case 3: Q(i) = P(i-1), Index < i
```

Note: Altough works, this is not the optimal way to insert many knot! See also the BspKnotEvalAlpha set of routines.

For more see: "Recursive proof of Boehm's knot insertion technique", by Phillip J Barry Ronald N Goldman, CAD, Volume 20 number 4 1988, pp 181-182. Which also references the original 1980 paper by Boehm.

3.2.27 BspCrvKnotInsertNDiff (cagd_lib/cbsp_aux.c:198)

refinement

subdivision

```
CagdCrvStruct *BspCrvKnotInsertNDiff(CagdCrvStruct *Crv,
                                     CagdBType Replace,
                                     CagdRType *t, int n)
```

Crv: To refine by insertion (upto) n knot of value t.

Replace: if TRUE, the n knots in t should replace the knot vector of size n of Crv. Sizes must match. If False, n new knots as defined by t will be introduced into Crv.

t: New knots to introduce/replace knot vector of Crv.

n: Size of t.

Returns: Refined Crv with n new knots.

Description: Inserts n knot with different values as defined by the vector t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector.

3.2.28 BspCrvKnotInsertNSame (cagd_lib/cbsp_aux.c:150)

refinement

subdivision

```
CagdCrvStruct *BspCrvKnotInsertNSame(CagdCrvStruct *Crv, CagdRType t, int n)
```

Crv: To refine by insertion (upto) n knot of value t.

t: Parameter value of new knot to insert.

n: Maximum number of times t should be inserted.

Returns: Refined Crv with n knots of value t.

Description: Inserts n knot, all with the value t. In no case will the multiplicity of a knot be greater or equal to the curve order.

3.2.29 BspCrvMaxCoefParam (cagd_lib/bsp_knot.c:982)

extremum

```
CagdRType BspCrvMaxCoefParam(CagdCrvStruct *Crv, int Axis, CagdRType *MaxVal)
```

Crv: To compute the parameter node value of the largest coefficient.

Axis: Which axis should we search for maximal coefficient? 1 for X, 2 for Y, etc.

MaxVal: The coefficient itself will be place herein.

Returns: The node parameter value of the detected maximal coefficient.

Description: Finds the parameter value with the largest coefficient of the curve using nodes values to estimate the coefficients' parameters.

3.2.30 BspCrvNew (cagd_lib/bsp_gen.c:158)

allocation

```
CagdCrvStruct *BspPeriodicCrvNew(int Length,
                                 int Order,
                                 CagdBType Periodic,
                                 CagdPointType PType)
```

Length: Number of control points

Order: The order of the curve

Periodic: Is this curve periodic?

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline curve. If Periodic is FALSE, this function is identical to BspCrvNew.

Description: Allocates the memory required for a new, possibly periodic, Bspline curve.

3.2.31 BspCrvNew (cagd_lib/bsp_gen.c:118)

allocation

```
CagdCrvStruct *BspCrvNew(int Length, int Order, CagdPointType PType)
```

Length: Number of control points

Order: The order of the curve

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline curve.

Description: Allocates the memory required for a new Bspline curve.

3.2.32 BspCrvNoraml (cagd_lib/cbsp_aux.c:592)

normal

```
CagdVecStruct *BspCrvNormal(CagdCrvStruct *Crv, CagdRType t)
```

Crv: Crv for which to compute a unit normal.

t: The parameter at which to compute the unit normal.

Returns: A pointer to a static vector holding the normal information.

Description: Returns a unit vector, equal to the normal of Crv at parameter value t. Algorithm: returns the cross product of the curve tangent and binormal.

3.2.33 BspCrvOpenEnd (cagd_lib/bsp_gen.c:251)

open end conditions

```
CagdCrvStruct *BspCrvOpenEnd(CagdCrvStruct *Crv)
```

Crv: To convert to a new curve with open end conditions.

Returns: Same curve as Crv but with open end conditions.

Description: Returns a curve with open end conditions, similar to given curve. Open end curve is computed by extracting a subregion from Crv that is the entire curve's parametric domain, by inserting multiple knots at the domain's boundary.

3.2.34 BspCrvSubdivAtParam (cagd_lib/cbsp_aux.c:31)

subdivision

refinement

```
CagdCrvStruct *BspCrvSubdivAtParam(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To subdivide at parametr value t.

t: Parameter value to subdivide Crv at.

Returns: A list of the two subdivided curves.

Description: Given a Bspline curve - subdivides it into two sub-curves at the given parametric value. Returns pointer to first curve in a list of two subdivided curves. The subdivision is achieved by inserting (order-1) knot at the given parameter value t and spliting the control polygon and knot vector at that location.

3.2.35 BspCrvTangent (cagd_lib/cbsp_aux.c:422)

tangent

```
CagdVecStruct *BspCrvTangent(CagdCrvStruct *Crv, CagdRType t)
```

Crv: Crv for which to compute a unit tangent.

t: The parameter at which to compute the unit tangent.

Returns: A pointer to a static vector holding the tangent information.

Description: Returns a unit vector, equal to the tangent to Crv at parameter value t. Algorithm: insert (order - 1) knots and return control polygon tangent.

3.2.36 BspKnotAffineTrans (cagd_lib/bsp_knot.c:447)

knot vectors

affine transformation

```
void BspKnotAffineTrans(CagdRType *KnotVector,
                        int Len,
                        CagdRType Translate,
                        CagdRType Scale)
```

KnotVector: To affinely transform.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

Translate: Amount to translate the knot vector.

Scale: Amount to scale the knot vector.

Description: Applies an affine transformation to the given knot vector. Note affine transformation on the knot vector does not change the Bspline curve. Knot vector is translated by Translate amount and scaled by Scale as

```
KV[i] = (KV[i] - KV[0]) * Scale + (KV0 + Translate).
```

All transformation as taken place in place.

3.2.37 BspKnotAffineTrans2 (cagd_lib/bsp_knot.c:482)

knot vectors

affine transformation

```
void BspKnotAffineTrans2(CagdRType *KnotVector,
                         int Len,
                         CagdRType MinVal,
                         CagdRType MaxVal)
```

KnotVector: To affinely transform.

Len: Of knot vector. This is not the length of the curve or surface using this knot vector.

MinVal, MaxVal: New parametric domain of knot vector.

Description: Applies an affine transformation to the given knot vector. Note affine transformation on the knot vector does not change the Bspline curve. Knot vector is translated and scaled so as to span the domain from MinVal top MaxVal. This works for open end condition curves only.

```
KV[i] = (KV[i] - KV[0]) * Scale + MinVal,
```

where Scale = (MaxVal - MinVal) / (KV[Len - 1] - KV[0]). All transformation as taken place in place.

3.2.38 BspKnotAllC1Discont (cagd_lib/bsp_knot.c:1266)

knot vectors

continuity

discontinuity

```
CagdRType *BspKnotAllC1Discont(CagdRType *KnotVector,
                               int Order,
                               int Length,
                               int *n)
```

KnotVector: To test for potential C1 dicontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

n: Length of returned vector - number of potential C1 discontinuities found.

Returns: Vector holding all parametr values with potential C1 discontinuities.

Description: Scans the given knot vector for all potential C1 discontinuity. Returns a vector holding the parameter values of the potential C1 discontinuities, NULL of none found. Sets n to length of returned vector. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order - 1) can be C1 discontinuous at that knot. However, this is only a necessary condition for C1 discontinuity in the geometry.

3.2.39 BspKnotAverage (cagd_lib/bsp_knot.c:804)

knot vectors

node values

```
CagdRType *BspKnotAverage(CagdRType *KnotVector, int Len, int Ave)
```

KnotVector: To average out.

Len: Length of KnotVector. This is not the length of the curve or surface using this knot vector.

Ave: How many knots to average each time.

Returns: The averaged knot vector of length (Len - Ave + 1).

Description: Creates a new knot vector from the given KnotVector that averages Ave consequetive knots. Resulting vector will have (Len - Ave + 1) elements. See also BspKnotNodes.

3.2.40 BspKnotC1Discont (cagd_lib/bsp_knot.c:1214)

knot vectors

continuity

discontinuity

```
CagdBType BspKnotC1Discont(CagdRType *KnotVector,
                           int Order,
                           int Length,
                           CagdRType *t)
```

KnotVector: To test for potential C1 dicontinuities.

Order: Of geometry that exploits KnotVector.

Length: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: Where to put the parameter value (knot) that can be C2 discontinuous.

Returns: TRUE if found a potential C1 discontinuity, FALSE otherwise.

Description: Scans the given knot vector to a potential C1 discontinuity. Returns TRUE if found one and set t to its parameter value. Assumes knot vector has open end condition. A knot vector with multiplicity of a knot of (Order - 1) can be C1 discontinuous at that knot. However, this is only a necessary condition for C1 discontinuity in the geometry.

3.2.41 BspKnotContinuityMergeTwo (cagd_lib/bsp_knot.c:719)

knot vectors | compatibility | refinement

```
CagdRType *BspKnotContinuityMergeTwo(CagdRType *KnotVector1,
                                     int Len1,
                                     int Order1,
                                     CagdRType *KnotVector2,
                                     int Len2,
                                     int Order2,
                                     int ResOrder,
                                     int *NewLen)
```

KnotVector1: First knot vector.

Len1: Length of KnotVector1. This is not the length of the curve or surface using this knot vector.

Order1: Order of first knot vector's geometry.

KnotVector2: Second knot vector.

Len2: Length of KnotVector2. This is not the length of the curve or surface using this knot vector.

Order2: Order of secon knot vector's geometry.

ResOrder: Expected order of geometry that will use the merged knot vector.

NewLen: To save the size of the knot vector that contains the merged knot vectors.

Returns: The merged knot verctor (KnotVector1 U KnotVector2).

Description: Merges two knot vector KnotVector1 and KnotVector2 of length Len1 and Len2 respectively into one, from geometries of orders Order1 and Order2. Merged knot vector is for order ResOrder so that the resulting curve can represent the discontinuities in both geometries. Assumes both knot vectors are open end spanning the same domain.

3.2.42 BspKnotCopy (cagd_lib/bsp_knot.c:509)

allocation | knot vectors

```
CagdRType *BspKnotCopy(CagdRType *KnotVector, int Len)
```

KnotVector: Knot vector to duplicate

Len: Length of knot vector. This is not the length of the curve or surface using this knot vector.

Returns: The duplicated knot vector.

Description: Creates an identical copy of a given knot vector KnotVector of length Len.

3.2.43 BspKnotEvalAlphaCoef (cagd_lib/cagdoslo.c:71)

alpha matrix | refinement

```
BspKnotAlphaCoeffType *BspKnotEvalAlphaCoef(int k,
                                            CagdRType *KVT,
                                            int LengthKVT,
                                            CagdRType *KVt,
                                            int LengthKVt)
```

k: Order of geometry.

KVT: Original knot vector.

LengthKVT: Length of original knot vector.

KVt: Refined knot vector. Must contain all knots of KVT.

LengthKVt: Length of refined knot vector.

Returns: A matrix to multiply the coefficients of the geometry using KVT, in order to get the coefficients under the space defined using KVt that represent the same geometry.

Description: Computes the values of the alpha coefficients, Ai,k(j) of order k:

```
        n              n    m                   m    n
        _              _    _                   _    _
C(t) = \ Pi Bi,k(t) = \ Pi \ Ai,k(j) Nj,k(t) = \  ( \ Pi Ai,k(j) ) Nj,k(t)
        /              /    /                   /    /
        -              -    -                   -    -
       i=0            i=0  j=0                 j=0  i=0
```

Let T be the original knot vector and let t be the refined one, i.e. T is a subset of t. The Ai,k(j) are computed from the following recursive definition:

```
            1, T(i) <= t(i) < T(i+1)
          /
Ai,1(j) =
          \
            0, otherwise.
```

```
          T(j+k-1) - T(i)             T(i+k) - T(j+k-1)
Ai,k(j) = --------------- Ai,k-1(j) + --------------- Ai+1,k-1(j)
          T(i+k-1) - T(i)             T(i+k) - T(i+1)
```

LengthKVT + k is the length of KVT and similarly LengthKVt + k is the length of KVt. In other words, LengthKVT and LengthKVt are the control points len...
The output matrix has LengthKVT rows and LengthKVt columns (#cols > #rows) ColIndex/Length hold LengthKVt pairs of first non zero scalar and length of non zero values in that column, so not all LengthKVT scalars are blended.

3.2.44 BspKnotEvalAlphaCoefMerge (cagd_lib/cagdoslo.c:193)

alpha matrix

refinement

```
BspKnotAlphaCoeffType *BspKnotEvalAlphaCoefMerge(int k,
                                                 CagdRType *KVT,
                                                 int LengthKVT,
                                                 CagdRType *NewKV,
                                                 int LengthNewKV)
```

k: Order of geometry.

KVT: Original knot vector.

LengthKVT: Length of original knot vector.

NewKV: A sequence of new knots to introduce into KVT.

LengthNewKV: Length of new knot sequence.

Returns: A matrix to multiply the coefficients of the geometry using KVT, in order to get the coefficients under the space defined using KVt that represent the same geometry.

Description: Same as EvalAlphaCoef but the new knot set NewKV is merged with KVT to form the new knot vector KVt.

3.2.45 BspKnotFindMult (cagd_lib/bsp_knot.c:1171)

knot vectors

```
int BspKnotFindMult(CagdRType *KnotVector, int Order, int Len, CagdRType t)
```

KnotVector: To test multiplicity of knot value t at.

Order: Of geometry that exploits KnotVector.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: The knot to verify the multiplicity of.

Returns: Multiplicity of t in KnotVector.

Description: Returns the multiplicity of knot t in knot vector KnotVector, zero if none.

3.2.46 BspKnotFirstIndexG (cagd_lib/bsp_knot.c:292)

knot vectors

```
int BspKnotFirstIndexG(CagdRType *KnotVector, int Len, CagdRType t)
```

KnotVector: To search for a knot with the G relation to t.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

t: The parameter value to search for the G relation.

Returns: Index of first knot in KnotVector that is greater than t or Len if t is greater than last knot in KnotVector.

Description: Returns the index of the first knot which is greater than t in the given knot vector KnotVector of length Len. APX_EQ_EPS is used for equality. Parameter t is assumed to be in the parametric domain for the knot vector.

3.2.47 BspKnotFreeAlphaCoef (cagd_lib/cagdoslo.c:159)

alpha matrix

refinement

```
void BspKnotFreeAlphaCoef(BspKnotAlphaCoeffType *A)
```

A: Alpha matrix to free.

Description: Frees the BspKnotAlphaCoeffType data structrure.

3.2.48 BspKnotHasBezierKV (cagd_lib/bsp_knot.c:69)

knot vectors

conversion

```
CagdBType BspKnotHasBezierKV(CagdRType *KnotVector, int Len, int Order)
```

KnotVector: To check for open end and no interior knots conditions.

Len: Of knot vector.

Order: Of curve/surface the exploits this knot vector.

Returns: TRUE if has open end conditions and no interior knots, FALSE otherwise.

Description: Returns TRUE iff the given knot vector has no interior knots and it has an open end cond.

3.2.49 BspKnotHasOpenEC (cagd_lib/bsp_knot.c:157)

knot vectors

open end conditions

```
CagdBType BspKnotHasOpenEC(CagdRType *KnotVector, int Len, int Order)
```

KnotVector: To check for open end condition.

Len: Of knot vector.

Order: Of curve/surface the exploits this knot vector.

Returns: TRUE if KV has open end conditions.

Description: Returns TRUE iff the given knot vector has open end conditions.

3.2.50 BspKnotInsertMult (cagd_lib/bsp_knot.c:1123)

knot vectors

knot insertion

refinement

```
CagdRType *BspKnotInsertMult(CagdRType *KnotVector,
                             int Order,
                             int *Len,
                             CagdRType t,
                             int Mult)
```

KnotVector: To insert new knot t in.

Order: Of geometry that exploits KnotVector.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: The new knot t to insert.

Mult: The multiplicity that this knot should have in resulting knot vector.

Returns: A new knot vector derived from KnotVector that has a mltiplicity of exacly Mult at the knot t.

Description: Inserts Mult knots with value t into the knot vector KnotVector. Attempt is made to make sure t in knot vector domain. If a knot equal to t (up to APX_EQ) already exists with multiplicity i only (Mult - i) knot are being inserted into the new knot vector. Len is updated to the resulting knot vector. It is possible to DELETE a knot using this routine by specifying multiplicity less then current multiplicity! This function only constructs a refined knot vector and does not compute the actual refined coefficients.

3.2.51 BspKnotInsertOne (cagd_lib/bsp_knot.c:1084)

knot vectors

knot insertion

refinement

```
CagdRType *BspKnotInsertOne(CagdRType *KnotVector,
                            int Order,
                            int Len,
                            CagdRType t)
```

KnotVector: To insert new knot t in.

Order: Of geometry that exploits KnotVector.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

t: The new knot t to insert.

Returns: A new knot vector larger by one than KnotVector that contains t.

Description: Creates a new vector with t inserted as a new knot. Attempt is made to make sure t is in the knot vector domain. No test is made for the current multiplicity of knot t in KnotVector. This function only constructs a refined knot vector and does not compute the actual refined coefficients.

3.2.52 BspKnotLastIndexL (cagd_lib/bsp_knot.c:259)

knot vectors

```
int BspKnotLastIndexL(CagdRType *KnotVector, int Len, CagdRType t)
```

KnotVector: To search for a knot with the L relation to t.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

t: The parameter value to search for the L relation.

Returns: Index of last knot in KnotVector that is less than t or -1 if t is below the first knot.

Description: Returns the index of the last knot which is less t in the given knot vector KnotVector of length Len. APX_EQ_EPS is used for equality. Parameter t is assumed to be in the parametric domain for the knot vector.

3.2.53 BspKnotLastIndexLE (cagd_lib/bsp_knot.c:227)

knot vectors

```
int BspKnotLastIndexLE(CagdRType *KnotVector, int Len, CagdRType t)
```

KnotVector: To search for a knot with the LE relation to t.

Len: Of knot vector. This is not the length of the curve/surface using this KnotVector.

t: The parameter value to search for the LE relation.

Returns: Index of last knot in KnotVector that is LE t, or -1 if t is below the first knot.

Description: Returns the index of the last knot which is less than or equal to t in the given knot vector KnotVector of length Len. APX_EQ_EPS is used in equality. Parameter t is assumed to be in the parametric domain for the knot vector.

3.2.54 BspKnotMakeRobustKVm knot vectors (cagd_lib/bsp_knot.c:1399)

```
void BspKnotMakeRobustKV(CagdRType *KV, int Len)
```

KV: Knot vector to make robust, in place.

Len: Length of knot vector KV.

Description: Given a knot vector, make sure adjacent knots that are close "enough" are actually identical. Important for robustness of subdiv/refinement algs.

3.2.55 BspKnotMergeTwo (cagd_lib/bsp_knot.c:636)

knot vectors

compatibility

refinement

```
CagdRType *BspKnotMergeTwo(CagdRType *KnotVector1,
                           int Len1,
                           CagdRType *KnotVector2,
                           int Len2,
                           int Mult,
                           int *NewLen)
```

KnotVector1: First knot vector.

Len1: Length of KnotVector1. This is not the length of the curve or surface using this knot vector.

KnotVector2: Second knot vector.

Len2: Length of KnotVector2. This is not the length of the curve or surface using this knot vector.

Mult: Maximum multiplicity to allow in merged knot vector.

NewLen: To save the size of the knot vector that contains the merged knot vectors.

Returns: The merged knot verctor (KnotVector1 U KnotVector2).

Description: Merges two knot vector KnotVector1 and KnotVector2 of length Len1 and Len2 respectively into one. If Mult is not zero then knot multiplicity is tested not to be larger than Mult value. NewLen is set to new KnotVector length.

3.2.56 BspKnotNodes (cagd_lib/bsp_knot.c:859)

knot vectors

node values

```
CagdRType *BspKnotNodes(CagdRType *KnotVector, int Len, int Order)
```

KnotVector: To average out as nodes.

Len: Length of KnotVector. This is not the length of the curve or surface using this knot vector.

Order: Of curve or surface that exploits this knot vector.

Returns: The nodes computed for the given knot vector.

Description: Creates a new vector with the given KnotVector Node values. The nodes are the approximated parametric value associated with the each control point. Therefore for a knot vector of length Len and order Order there are Len - Order control points and therefore nodes. Nodes are defined as (k = Order - 1 or degree):

```
         i+k
          -                          First Node N(i = 0)
          \
          / KnotVector(j)            Last Node  N(i = Len - k - 2)
          -
        j=i+1
 N(i) = -----------------
              k
```

See also BspKnotAverage.

3.2.57 BspKnotParamInDomain (cagd_lib/bsp_knot.c:197)

parametric domain

knot vectors

```
CagdBType BspKnotParamInDomain(CagdRType *KnotVector,
                               int Len,
                               int Order,
                               CagdBType Periodic,
                               CagdRType t)
```

KnotVector: To verify t is indeed in.

Len: Length of curve/surface using KnotVector. This is NOT the length of KnotVector which is equal to (Length + Order).

Order: Order of curve/surface using KnotVector.

Periodic: TRUE if this KnotVector is periodic.

t: Parametric value to verify.

Returns: TRUE, if t is contained in the parametric domain, FALSE otherwise.

Description: Returns TRUE iff t is in the parametric domain as define by the knot vector KnotVector, its length Len, and the order Order.

3.2.58 BspKnotParamValues (cagd_lib/bsp_knot.c:1323)

piecewise linear approximation

knot vectors

```
CagdRType *BspKnotParamValues(CagdRType PMin,
                              CagdRType PMax,
                              int NumSamples,
                              CagdRType *C1Disconts,
                              int NumC1Disconts)
```

PMin: Minimum of parametric domain.

PMax: Maximum of parametric domain.

NumSamples: To allocate for the vector of samples.

C1Disconts: A vector of potential C1 discontinuities in the (PMin, PMax) domain. This vector is freed by this routine, if it is not NULL.

NumC1Disconts: Length of C1Discont. if zero then C1Discont == NULL.

Returns: A vector of the suggested set of sampling locations.

Description: Routine to determine where to sample along the provided paramtric domain, given the C1 discontinuities along it. Returns a vector of length NumSamples. If C1Disconts != NULL (NumC1Disconts > 0), C1Discont is being freed.

3.2.59 BspKnotPrepEquallySpaced (cagd_lib/cagdoslo.c:230)

refinement

```
CagdRType *BspKnotPrepEquallySpaced(int n, CagdRType Tmin, CagdRType Tmax)
```

n: Number of knots to introduce.

Tmin: Minimum domain to introduce knots.

Tmax: Maximum domain to introduce knots.

Returns: A vector of n knots uniformly spaced between TMin and TMax.

Description: Prepares a refinement vector for the given knot vector domain with n inserted knots equally spaced.

3.2.60 BspKnotReverse (cagd_lib/bsp_knot.c:540)

knot vectors

reverse

```
CagdRType *BspKnotReverse(CagdRType *KnotVector, int Len)
```

KnotVector: Knot vector to be reversed.

Len: Length of knot vector. This is not the length of the curve or surface using this knot vector.

Returns: The reversed knot vector.

Description: Reverse a knot vector of length Len. Reversing of knot vector keeps the knots monotonically non-decreasing as well as the parametric domain. Only the spaces between the knots are being flipped. For example the knot vector:

```
[0 0 0 0 1 2 2 6 6 6 6]
```

is reversed to be:

```
[0 0 0 0 4 4 5 6 6 6 6]
```

3.2.61 BspKnotSubtrTwo (cagd_lib/bsp_knot.c:580)

knot vectors

compatibility

refinement

```
CagdRType *BspKnotSubtrTwo(CagdRType *KnotVector1,
                           int Len1,
                           CagdRType *KnotVector2,
                           int Len2,
                           int *NewLen)
```

KnotVector1: First knot vector.

Len1: Length of KnotVector1. This is not the length of the curve or surface using this knot vector.

KnotVector2: Second knot vector.

Len1: Length of KnotVector2. This is not the length of the curve or surface using this knot vector.

NewLen: To save the size of the knot vector that contains the computed subset of KnotVector1 / KnotVector2.

Returns: The subset of knot in KnotVector1 that are not in KnotVector2 (KnotVector1 / KnotVector2).

Description: Returns a knot vector that contains all the knots in KnotVector1 that are not in KnotVector2. NewLen is set to new KnotVector length.

3.2.62 BspKnotUniformFloat (cagd_lib/bsp_knot.c:364)

knot vectors

floating end conditions

end conditions

```
CagdRType *BspKnotUniformFloat(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that are to use this knot vector.

Order: Of the curve/surface that are to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform floating knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a uniform floating knot vector for Len Control points and order Order. The actual length of the KV is Len + Order. If KnotVector is NULL it is being allocated dynamically.

3.2.63 BspKnotUniformOpen (cagd_lib/bsp_knot.c:402)

knot vectors

open end conditions

end conditions

```
CagdRType *BspKnotUniformOpen(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that are to use this knot vector.

Order: Of the curve/surface that are to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform open knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a uniform open knot vector for Len Control points and order Order. The actual length of the KV is Len + Order. If KnotVector is NULL it is being allocated dynamically.

3.2.64 BspKnotUniformPeriodic (cagd_lib/bsp_knot.c:326)

knot vectors

periodic end conditions

end conditions

```
CagdRType *BspKnotUniformPeriodic(int Len, int Order, CagdRType *KnotVector)
```

Len: Of control polygon/mesh of curve/surface that are to use this knot vector.

Order: Of the curve/surface that are to use this knot vector.

KnotVector: If new knot vector is to be saved here, otherwise a new space is allocated.

Returns: The created uniform periodic knot vector, either newly allocated or given in Knotvector and just initialized.

Description: Returns a uniform periodic knot vector for Len Control points and order Order. The actual length of the KV is Len + Order + Order - 1. If KnotVector is NULL it is being allocated dynamically.

3.2.65 BspPeriodicSrfNew (cagd_lib/bsp_gen.c:80)

allocation

```
CagdSrfStruct *BspPeriodicSrfNew(int ULength,
                                 int VLength,
                                 int UOrder,
                                 int VOrder,
                                 CagdBType UPeriodic,
                                 CagdBType VPeriodic,
                                 CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

UOrder: The order of the surface in the U direction.

VOrder: The order of the surface in the V direction.

UPeriodic: Is this surface periodic in the U direction?

VPeriodic: Is this surface periodic in the V direction?

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline surface. If both UPeriodic and VPeriodic are FALSE, this function is identical to BspSrfNew.

Description: Allocates the memory required for a new, possibly periodic, Bspline surface.

3.2.66 BspSrf2Curves (cagd_lib/bsp2poly.c:483)

curves

isoparametric curves

```
CagdCrvStruct *BspSrf2Curves(CagdSrfStruct *Srf, int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a Bspline surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.

3.2.67 BspSrf2Polygons (cagd_lib/bsp2poly.c:44)

polygonization

surface approximation

```
CagdPolygonStruct *BspSrf2Polygons(CagdSrfStruct *Srf,
                                   int FineNess,
                                   CagdBType ComputeNormals,
                                   CagdBType FourPerFlat,
                                   CagdBType ComputeUV)
```

Srf: To approximate into triangles.

FineNess: Control on accuracy, the higher the finer.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single Bspline surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of CagdPolygonStruct. This routine looks for C1 discontinuities in the surface and splits it into C1 continuous patches to invoke BspC1Srf2Polygons to gen. polygons.

3.2.68 BspSrf2Polylines (cagd_lib/bsp2poly.c:335)

polylines

isoparametric curves

```
CagdPolylineStruct *BspSrf2Polylines(CagdSrfStruct *Srf,
                                     int NumOfIsocurves[2],
                                     int SamplesPerCurve)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Returns: List of polygons representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single Bspline surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.

3.2.69 BspSrfCrvFromMesh (cagd_lib/sbspeval.c:245)

isoparametric curves

curve from mesh

```
CagdCrvStruct *BspSrfCrvFromMesh(CagdSrfStruct *Srf,
                                 int Index,
                                 CagdSrfDirType Dir)
```

Srf: To extract a curve from.

Index: Index along the mesh of Srf to extract the curve from.

Dir: Direction of extracted curve. Either U or V.

Returns: A curve from Srf. This curve inherit the order and continuity of surface Srf in direction Dir. However, thiscurve is not on surface Srf, in general.

Description: Extracts a curve from the mesh of a tensor product Bspline surface Srf in direction Dir at index Index.

3.2.70 BspSrfCrvFromSrf (cagd_lib/sbspeval.c:165)

isoparametric curves

curve from surface

```
CagdCrvStruct *BspSrfCrvFromSrf(CagdSrfStruct *Srf,
                                CagdRType t,
                                CagdSrfDirType dir)
```

Srf: To extract an isoparametric ocurve from.

t: Parameter value of extracted isoparametric curve.

dir: Direction of the isocurve on the surface. Either U or V.

Returns: An isoparametric curve of Srf. This curve inherits the order and continuity of surface Srf in direction Dir.

Description: Extracts an isoparametric curve out of the given tensor product Bspline surface in direction Dir at the parameter value of t. Operations should prefer the CONST_U_DIR, in which the extraction is somewhat faster if that is possible.

3.2.71 BspSrfDegreeRaise (cagd_lib/sbsp_aux.c:555)

degree raising

```
CagdSrfStruct *BspSrfDegreeRaise(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To raise it degree by one.

Dir: Direction of degree raising. Either U or V.

Returns: A surface with one degree higher in direction Dir, representing the same geometry as Srf.

Description: Returns a new Bspline surface, identical to the original but with one degree higher, in the requested direction Dir.

3.2.72 BspSrfDerive (cagd_lib/sbsp_aux.c:719)

derivatives

partial derivatives

```
CagdSrfStruct *BspSrfDerive(CagdSrfStruct *Srf,
                            CagdSrfDirType Dir)
```

Srf: To differentiate.

Dir: Direction of differentiation. Either U or V.

Returns: Differentiated surface.

Description: Returns a new surface equal to the given surface, differentiated once in the direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

This is applied to all rows/cols of the surface.

3.2.73 BspSrfDomain (cagd_lib/bsp_gen.c:222)

domain

parametric domain

```
void BspSrfDomain(CagdSrfStruct *Srf,
                  CagdRType *UMin,
                  CagdRType *UMax,
                  CagdRType *VMin,
                  CagdRType *VMax)
```

Srf: To get its parametric domain.

UMin: Where to put the minimal U domain's boundary.

UMax: Where to put the maximal U domain's boundary.

VMin: Where to put the minimal V domain's boundary.

VMax: Where to put the maximal V domain's boundary.

Description: Returns the parametric domain of a Bspline surface.

3.2.74 BspSrfEvalAtParam (cagd_lib/sbspeval.c:40)

evaluation

Bsplines

```
CagdRType *BspSrfEvalAtParam(CagdSrfStruct *Srf, CagdRType u, CagdRType v)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given tensor product Bspline surface at a given point, by extracting an isoparamteric curve along u from the surface and evaluating the curve at parameter v.

```
          u -->
 +-----------------------+
 |P0                 Pi-1|
V |Pi               P2i-1|  Parametric space orientation - control mesh.
 ||                      |
v|Pn-i               Pn-1|
 +-----------------------+
```

3.2.75 BspSrfHasBezierKVs (cagd_lib/bsp_knot.c:45)

conversion

```
CagdBType BspSrfHasBezierKVs(CagdSrfStruct *Srf)
```

Srf: To check for KVs that mimics Bezier polynomial surface.

Returns: TRUE if same as Bezier curve, FALSE otherwise.

Description: Returns TRUE iff the given surface has no interior knot open end KVs.

3.2.76 BspSrfHasOpenEC (cagd_lib/bsp_knot.c:105)

open end conditions

```
CagdBType BspSrfHasOpenEC(CagdSrfStruct *Srf)
```

Srf: To check for open end conditions.

Returns: TRUE, if surface has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given Bspline surface has open end coditions.

3.2.77 BspSrfHasOpenECDir (cagd_lib/bsp_knot.c:127)

open end conditions

```
CagdBType BspSrfHasOpenECDir(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To check for open end conditions.

Dir: Either the U or the V parametric direction.

Returns: TRUE, if surface has open end conditions, FALSE otherwise.

Description: Returns TRUE iff the given Bspline surface has open end coditions in the specified direction.

3.2.78 BspSrfInterpPts (cagd_lib/sbsp_int.c:50)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpPts(CagdPtStruct **PtList,
                               int UOrder,
                               int VOrder,
                               int SrfUSize,
                               int SrfVSize,
                               CagdParametrizationType ParamType)
```

PtList: A NULL terminating array of linked list of points.

UOrder: Of the to be created surface.

VOrder: Of the to be created surface.

SrfUSize: U size of the to be created surface. Must be at least as large as the array PtList.

SrfVSize: V size of the to be created surface. Must be at least as large as the length of each list in PtList.

ParamType: Type of parametrization.

Returns: Constructed interpolating/approximating surface.

Description: Given a set of points, PtList, computes a Bspline surface of order UOrder by VOrder that interpolates or least square approximates the given set of points. PtList is a NULL terminated array of linked lists of CagdPtStruct structs. All linked lists in PtList must have the same length. U direction of surface is associated with array, V with the linked lists. The size of the control mesh of the resulting Bspline surface defaults to the number of points in PtList (if SrfUSize = SrfVSize = 0). However, either numbers can smaller to yield a least square approximation of the gievn data set. The created curve can be parametrized as specified by ParamType.

3.2.79 BspSrfInterpolate (cagd_lib/sbsp_int.c:212)

interpolation

least square approximation

```
CagdSrfStruct *BspSrfInterpolate(CagdCtlPtStruct *PtList,
                                 int NumUPts,
                                 int NumVPts,
                                 CagdRType *UParams,
                                 CagdRType *VParams,
                                 CagdRType *UKV,
                                 CagdRType *VKV,
                                 int ULength,
                                 int VLength,
                                 int UOrder,
                                 int VOrder)
```

PtList: A long linked list (NumUPts * NumVPts) of points to interpolated or least square approximate.

NumUPts: Number of points in PtList in the U direction.

NumVPts: Number of points in PtList in the V direction.

UParams: Parameter at which surface should interpolate or approxximate PtList in the U direction.

VParams: Parameter at which surface should interpolate or approxximate PtList in the V direction.

UKV: Requested knot vector form the surface in the U direction.

VKV: Requested knot vector form the surface in the V direction.

ULength: Requested length of control mesh of surface in U direction.

VLength: Requested length of control mesh of surface in V direction.

UOrder: Requested order of surface in U direction.

VOrder: Requested order of surface in U direction.

Returns: Constructed interpolating/approximating surface.

Description: Given a set of points on a rectangular grid, PtList, parameter values the surface should interpolate or approximate these grid points, U/VParams, the expected two knot vectors of the surface, U/VKV, the expected lengths U/VLength and orders U/VOrder of the Bspline surface, computes the Bspline surface's coefficients. All points in PtList are assumed of the same type.

3.2.80 BspSrfKnotInsert (cagd_lib/bspboehm.c:132)

refinement

knot insertion

```
CagdSrfStruct *BspSrfKnotInsert(CagdSrfStruct *Srf,
                                CagdSrfDirType Dir,
                                CagdRType t)
```

Srf: To refine by adding a new knot with value equal to t. If Srf is a periodic curve, it is first unwrapped to a float end condition curve.

Dir: Of refinement, either U or V.

t: New knot to insert into Srf.

Returns: The refined surface.

Description: Returns a new surface refined at t (t is inserted as a new knot in Srf) in parametric directipn Dir. See BspCrvKnotInsert for the mathematical background of this knot insertion algorithm.

3.2.81 BspSrfKnotInsertNDiff (cagd_lib/sbsp_aux.c:381)

refinement

subdivision

```
CagdSrfStruct *BspSrfKnotInsertNDiff(CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     int Replace,
                                     CagdRType *t,
                                     int n)
```

Srf: To refine by insertion (upto) n knot of value t.

Dir: Direction of refinement. Either U or V.

Replace: if TRUE, the n knots in t should replace the knot vector of size n of Srf. Sizes must match. If False, n new knots as defined by t will be introduced into Srf.

t: New knots to introduce/replace knot vector of Srf.

n: Size of t.

Returns: Refined Srf with n new knots in direction Dir.

Description: Inserts n knot with different values as defined by the vector t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector.

3.2.82 BspSrfKnotInsertNSame (cagd_lib/sbsp_aux.c:306)

refinement

subdivision

```
CagdSrfStruct *BspSrfKnotInsertNSame(CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     CagdRType t,
                                     int n)
```

Srf: To refine by insertion (upto) n knot of value t.

Dir: Direction of refinement. Either U or V.

t: Parameter value of new knot to insert.

n: Maximum number of times t should be inserted.

Returns: Refined Srf with n knots of value t in direction Dir.

Description: Inserts n knot, all with the value t in direction Dir. In no case will the multiplicity of a knot be greater or equal to the curve order.

3.2.83 BspSrfMaxCoefParam (cagd_lib/bsp_knot.c:1027)

extremum

```
CagdRType *BspSrfMaxCoefParam(CagdSrfStruct *Srf, int Axis, CagdRType *MaxVal)
```

Srf: To compute the parameter node value of the largest coefficient.

Axis: Which axis should we search for maximal coefficient? 1 for X, 2 for Y, etc.

MaxVal: The coefficient itself will be place herein.

Returns: The node UV parameter values of the detected maximal coefficient.

Description: Finds the parameter value with the largest coefficient of the surface using nodes values to estimate the coefficients' parameters. Returns a pointer to a static array of two elements holding U and V.

3.2.84 BspSrfMeshNormals (cagd_lib/sbsp_aux.c:933)

normal

```
CagdVecStruct *BspSrfMeshNormals(CagdSrfStruct *Srf,
                                 int UFineNess,
                                 int VFineNess)
```

Srf: To compute normals on a grid of its parametric domain.

UFineNess: U Fineness of imposed grid on Srf's parametric domain.

VFineNess: V Fineness of imposed grid on Srf's parametric domain.

Returns: An vector of unit normals (u increments first).

Description: Evaluates the unit normals of a surface at a mesh defined by subdividing the parametric space into a grid of size UFineNess by VFineNess. The normals are saved in a linear CagdVecStruct vector which is allocated dynamically. Data is saved u inc. first. This routine is much faster than evaluating normal for each point, individually.

3.2.85 BspSrfMeshNormalsSymb (cagd_lib/sbsp_aux.c:1123)

normal

```
CagdVecStruct *BspSrfMeshNormalsSymb(CagdSrfStruct *Srf,
                                     int UFineNess,
                                     int VFineNess)
```

Srf: To compute normals on a grid of its parametric domain.

UFineNess: U Fineness of imposed grid on Srf's parametric domain.

VFineNess: V Fineness of imposed grid on Srf's parametric domain.

Returns: An vector of unit normals (u increments first).

Description: Evaluates the unit normals of a surface at a mesh defined by subdividing the parametric space into a grid of size UFineNess by VFineNess. The normals are saved in a linear CagdVecStruct vector which is allocated dynamically. Data is saved u inc. first. This routine is much faster than evaluating normal for each point, individually.

3.2.86 BspSrfNew (cagd_lib/bsp_gen.c:30)

allocation

```
CagdSrfStruct *BspSrfNew(int ULength,
                         int VLength,
                         int UOrder,
                         int VOrder,
                         CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

UOrder: The order of the surface in the U direction.

VOrder: The order of the surface in the V direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bspline surface.

Description: Allocates the memory required for a new Bspline surface.

3.2.87 BspSrfNormal (cagd_lib/sbsp_aux.c:877)

normal

```
CagdVecStruct *BspSrfNormal(CagdSrfStruct *Srf, CagdRType u, CagdRType v)
```

Srf: Bspline surface to evaluate normal vector for.

u, v: Parametric location of required unit normal.

Returns: A pointer to a static vector holding the unit normal information.

Description: Evaluate the unit normal of a surface at a given parametric location. If we fail to compute the normal at given location we retry by moving a tad.

3.2.88 BspSrfOpenEnd (cagd_lib/bsp_gen.c:279)

open end conditions

```
CagdSrfStruct *BspSrfOpenEnd(CagdSrfStruct *Srf)
```

Srf: To convert to a new curve with open end conditions.

Returns: Same surface as Srf but with open end conditions.

Description: Returns a surface with open end conditions, similar to given surface. Open end surface is computed by extracting a subregion from Srf that is the entire surface's parametric domain, by inserting multiple knots at the domain's boundary.

3.2.89 BspSrfSubdivAtParam (cagd_lib/sbsp_aux.c:45)

subdivision

refinement

```
CagdSrfStruct *BspSrfSubdivAtParam(CagdSrfStruct *Srf,
                                   CagdRType t,
                                   CagdSrfDirType Dir)
```

Srf: To subdivide at parameter value t.

t: Parameter value to subdivide Srf at.

Dir: Direction of subdivision. Either U or V.

Returns: A list of the two subdivided surfaces.

Description: Given a Bspline surface - subdivides it into two sub-surfaces at the given parametric value. Returns pointer to first surface in a list of two subdivided surfaces.

3.2.90 BspSrfTangent (cagd_lib/sbsp_aux.c:835)

tangent

```
CagdVecStruct *BspSrfTangent(CagdSrfStruct *Srf,
                             CagdRType u,
                             CagdRType v,
                             CagdSrfDirType Dir)
```

Srf: Bspline surface to evaluate tangent vector for.

u, v: Parametric location of required unit tangent.

Dir: Direction of tangent vector. Either U or V.

Returns: A pointer to a static vector holding the unit tangent information.

Description: Evaluates the unit tangent to a surface at a given parametric location u, v) and given direction Dir.

3.2.91 BzrCrv2Polyline (cagd_lib/bzr2poly.c:392)

piecewise linear approximation

polyline

```
CagdPolylineStruct *BzrCrv2Polyline(CagdCrvStruct *Crv, int SamplesPerCurve)
```

Crv: To approximate as a polyline.

SamplesPerCurve: Number of samples to approximate with.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single Bezier curve as a polyline with SamplesPerCurve samples. Polyline is always E3 CagdPolylineStruct type. Curve is sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise CagdPolylineStruct.

3.2.92 BzrCrvBiNormal (cagd_lib/cbzr_aux.c:264)

binormal

```
CagdVecStruct *BzrCrvBiNormal(CagdCrvStruct *Crv, CagdRType t)
```

Crv: Crv for which to compute a unit binormal.

t: The parameter at which to compute the unit binormal.

Returns: A pointer to a static vector holding the binormal information.

Description: Returns a unit vector, equal to the binormal to Crv at parameter value t. Algorithm: insert (order - 1) knots and using 3 consecutive control points at the refined location (p1, p2, p3), compute to binormal to be the cross product of the two vectors (p1 - p2) and (p2 - p3). Since a curve may have not BiNormal at inflection points or if the 3 points are colinear, NULL will be returned at such cases.

3.2.93 BzrCrvCreateArc (cagd_lib/cagd_arc.c:57)

circle

arc

```
CagdCrvStruct *BzrCrvCreateArc(CagdPtStruct *Start,
                               CagdPtStruct *Center,
                               CagdPtStruct *End)
```

Start: Point of beginning of arc.

Center: Point of arc.

End: Point of end of arc.

Returns: A rational quadratic Bezier curve representing the arc.

Description: Creates an arc at the specified position as a rational quadratic Bezier curve. The arc is assumed to be less than 180 degrees from Start to End in the shorter path as arc where Center as arc center.

3.2.94 BzrCrvDegreeRaise (cagd_lib/cbzr_aux.c:128)

degree raising

```
CagdCrvStruct *BzrCrvDegreeRaise(CagdCrvStruct *Crv)
```

Crv: To raise it degree by one.

Returns: A curve of one order higher representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with one degree higher. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
                      i            k-i
Q(0) = P(0), Q(i) = --- P(i-1) + (---) P(i), Q(k) = P(k-1).
                      k             k
```

3.2.95 BzrCrvDegreeRaiseN (cagd_lib/cbzr_aux.c:85)

degree raising

```
CagdCrvStruct *BzrCrvDegreeRaiseN(CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree to a NewOrder.

NewOrder: NewOrder for Crv.

Returns: A curve of order NewOrder representing the same geometry as Crv.

Description: Returns a new curve, identical to the original but with order N. Degree raise is computed by multiplying by a constant 1 curve of order

3.2.96 BzrCrvDerive (cagd_lib/cbzr_aux.c:387)

derivatives

```
CagdCrvStruct *BzrCrvDerive(CagdCrvStruct *Crv)
```

Crv: To differentiate.

Returns: Differentiated curve.

Description: Returns a new curve, equal to the given curve, differentiated once. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then: Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.

3.2.97 BzrCrvEvalAtParam (cagd_lib/cbzreval.c:148)

evaluation

```
CagdRType *BzrCrvEvalAtParam(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Returns a pointer to a static data, holding the value of the curve at given parametric location t. The curve is assumed to be Bezier.

3.2.98 BzrCrvEvalToPolyline (cagd_lib/cbzreval.c:189)

conversion refinement evaluation

```
void BzrCrvEvalToPolyline(CagdCrvStruct *Crv,
                          int FineNess,
                          CagdRType *Points[])
```

Crv: To approximate as a polyline.

FineNess: Control on number of samples.

Points: Where to put the resulting polyline.

A: Optional alpha matrix for refinement.

Returns: The actual number of samples placed in Points. Always less than or eaul to FineNess.

Description: Samples the curve at FineNess location equally spaced in the curve's parametric domain.

3.2.99 BzrCrvEvalVecAtParam (cagd_lib/cbzreval.c:110)

evaluation

```
CagdRType BzrCrvEvalVecAtParam(CagdRType *Vec,
                               int VecInc,
                               int Order,
                               CagdRType t)
```

Vec: Coefficents of a scalar Bspline univariate function.

VecInc: Step to move along Vec.

Order: Order of associated geometry.

t: Parameter value where to evaluate the curve.

Returns: Geometry's value at parameter value t.

Description: Assumes Vec holds control points for scalar bezier curve of order Order, and evaluates and returns that curve value at parameter value t. Vec is incremented by VecInc (usually by 1) after each iteration.

3.2.100 BzrCrvIntegrate (cagd_lib/cbzr_aux.c:446)

integrals

```
CagdCrvStruct *BzrCrvIntegrate(CagdCrvStruct *Crv)
```

Crv: Curve to integrate.

Returns: Integrated curve.

Description: Returns a new Bezier curve, equal to the integral of the given Bezier crv. The given Bezier curve should be nonrational.

```
            n            n               n       n+1
  /        /-            -      /        -   P    -
 |        | \     n      \     | n       \    i   \  n+1
 | C(t) = | / P  B (t) = / P   | B (t) = / -----  / B   (t) =
/        /  -  i  i      -  i /  i       - n + 1  -  j
           i=0          i=0             i=0     j=i+1
```

```
       n+1 j-1
        -   -
    1   \   \     n+1
= ----- /   / P  B   (t)
  n + 1 -   -  i  j
       j=1 i=0
```

3.2.101 BzrCrvNew (cagd_lib/bzr_gen.c:51)

allocation

```
CagdCrvStruct *BzrCrvNew(int Length, CagdPointType PType)
```

Length: Number of control points

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bezier curve.

Description: Allocates the memory required for a new Bezier curve.

3.2.102 BzrCrvNoraml (cagd_lib/cbzr_aux.c:355)

normal

```
CagdVecStruct *BzrCrvNormal(CagdCrvStruct *Crv, CagdRType t)
```

Crv: Crv for which to compute a unit normal.

t: The parameter at which to compute the unit normal.

Returns: A pointer to a static vector holding the normal information.

Description: Returns a unit vector, equal to the normal of Crv at parameter value t. Algorithm: returns the cross product of the curve tangent and binormal.

3.2.103 BzrCrvSetCache (cagd_lib/cbzreval.c:52)

evaluation

caching

```
void BzrCrvSetCache(int FineNess, CagdBType EnableCache)
```

FineNess: Number of samples to support (power of 2).

EnableCache: Are we really planning on using this thing?

Description: Sets the bezier sampling cache - if enabled, a Bezier can be evaluated directly from presampled basis function.

3.2.104 BzrCrvSubdivAtParam (cagd_lib/cbzr_aux.c:28)

subdivision

refinement

```
CagdCrvStruct *BzrCrvSubdivAtParam(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To subdivide at parametr value t.

t: Parameter value to subdivide Crv at.

Returns: A list of the two subdivided curves.

Description: Given a Bezier curve - subdivides it into two sub-curves at the given parametric value. Returns pointer to first curve in a list of two subdivided curves.

3.2.105 BzrCrvTangent (cagd_lib/cbzr_aux.c:170)

tangent

```
CagdVecStruct *BzrCrvTangent(CagdCrvStruct *Crv, CagdRType t)
```

Crv: Crv for which to compute a unit tangent.

t: The parameter at which to compute the unit tangent.

Returns: A pointer to a static vector holding the tangent information.

Description: Returns a unit vector, equal to the tangent to Crv at parameter value t. Algorithm: pseudo subdivide Crv at t and using control point of subdivided curve find the tangent as the difference of the 2 end points.

3.2.106 BzrSrf2Curves (cagd_lib/bzr2poly.c:333)

curves

isoparametric curves

```
CagdCrvStruct *BzrSrf2Curves(CagdSrfStruct *Srf, int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In reach (U or V) direction

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a bezier surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.

3.2.107 BzrSrf2Polygons (cagd_lib/bzr2poly.c:39)

polygonization

surface approximation

```
CagdPolygonStruct *BzrSrf2Polygons(CagdSrfStruct *Srf,
                                   int FineNess,
                                   CagdBType ComputeNormals,
                                   CagdBType FourPerFlat,
                                   CagdBType ComputeUV)
```

Srf: To approximate into triangles.

FineNess: Control on accuracy, the higher the finer.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single Bezier surface to set of triangles approximating it. FineNess is a fineness control on result and the larger is more triangles may result. A value of 10 is a good start value. NULL is returned in case of an error, otherwise list of CagdPolygonStruct.

3.2.108 BzrSrf2Polylines (cagd_lib/bzr2poly.c:266)

polylines

isoparametric curves

```
CagdPolylineStruct *BzrSrf2Polylines(CagdSrfStruct *Srf,
                                     int NumOfIsocurves[2],
                                     int SamplesPerCurve)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Returns: List of polygons representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single Bezier surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct.

3.2.109 BzrSrfCrvFromMesh (cagd_lib/sbzreval.c:139)

isoparametric curves

curve from mesh

```
CagdCrvStruct *BzrSrfCrvFromMesh(CagdSrfStruct *Srf,
                                 int Index,
                                 CagdSrfDirType Dir)
```

Srf: To extract a curve from.

Index: Index along the mesh of Srf to extract the curve from.

Dir: Direction of extracted curve. Either U or V.

Returns: A curve from Srf. This curve inherit the order and continuity of surface Srf in direction Dir. However, thiscurve is not on surface Srf, in general.

Description: Extracts a curve from the mesh of a tensor product Bezier surface Srf in direction Dir at index Index.

3.2.110 BzrSrfCrvFromSrf (cagd_lib/sbzreval.c:73)

isoparametric curves

curve from surface

```
CagdCrvStruct *BzrSrfCrvFromSrf(CagdSrfStruct *Srf,
                                CagdRType t,
                                CagdSrfDirType Dir)
```

Srf: To extract an isoparametric ocurve from.

t: Parameter value of extracted isoparametric curve.

dir: Direction of the isocurve on the surface. Either U or V.

Returns: An isoparametric curve of Srf. This curve inherits the order and continuity of surface Srf in direction Dir.

Description: Extracts an isoparametric curve out of the given tensor product Bezier surface in direction Dir at the parameter value of t. Operations should prefer the CONST_U_DIR, in which the extraction is somewhat faster if that is possible.

3.2.111 BzrSrfDegreeRaise (cagd_lib/sbzr_aux.c:106)

degree raising

```
CagdSrfStruct *BzrSrfDegreeRaise(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To raise it degree by one.

Dir: Direction to degree raise. Either U or V.

Returns: A surface with one degree higher in direction Dir, representing the same geometry as Srf.

Description: Returns a new Bezier surface, identical to the original but with one degree higher, in the requested direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
                     i           k-i
Q(0) = P(0), Q(i) = --- P(i-1) + (---) P(i), Q(k) = P(k-1).
                     k            k
```

This is applied to all rows/cols of the surface.

3.2.112 BzrSrfDerive (cagd_lib/sbzr_aux.c:186)

derivatives

partial derivatives

```
CagdSrfStruct *BzrSrfDerive(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To differentiate.

Dir: Direction of differentiation. Either U or V.

Returns: Differentiated surface.

Description: Returns a new surface equal to the given surface, differentiated once in the direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

This is applied to all rows/cols of the surface.

3.2.113 BzrSrfEvalAtParam (cagd_lib/sbzreval.c:39)

evaluation

Bezier

```
CagdRType *BzrSrfEvalAtParam(CagdSrfStruct *Srf, CagdRType u, CagdRType v)
```

Srf: Surface to evaluate at the given (u, v) location.

u, v: Location where to evaluate the surface.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given tensor product Bezier surface at a given point, by extracting an isoparamteric curve along u from the surface and evaluating the curve at parameter v.

```
            u -->
  +-----------------------+
  |P0                 Pi-1|
V |Pi                P2i-1|  Parametric space orientation - control mesh.
 ||                       |
 v|Pn-i               Pn-1|
  +-----------------------+
```

3.2.114 BzrSrfNew (cagd_lib/bzr_gen.c:24)

allocation

```
CagdSrfStruct *BzrSrfNew(int ULength, int VLength, CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform Bezier surface.

Description: Allocates the memory required for a new Bezier surface.

3.2.115 BzrSrfNormal (cagd_lib/sbzr_aux.c:295)

normal

```
CagdVecStruct *BzrSrfNormal(CagdSrfStruct *Srf, CagdRType u, CagdRType v)
```

Srf: Bezier surface to evaluate normal vector for.

u, v: Parametric location of required unit normal.

Returns: A pointer to a static vector holding the unit normal information.

Description: Evaluate the unit normal of a surface at a given parametric location. If we fail to compute the normal at given location we retry by moving a tad.

3.2.116 BzrSrfSubdivAtParam (cagd_lib/sbzr_aux.c:39)

subdivision

refinement

```
CagdSrfStruct *BzrSrfSubdivAtParam(CagdSrfStruct *Srf,
                                   CagdRType t,
                                   CagdSrfDirType Dir)
```

Srf: To subdivide at parameter value t.

t: Parameter value to subdivide Srf at.

Dir: Direction of subdivision. Either U or V.

Returns: A list of the two subdivided surfaces.

Description: Given a Bezier surface - subdivides it into two sub-surfaces at the given parametric value. Returns pointer to first surface in a list of two subdivided surfaces.

3.2.117 BzrSrfTangent (cagd_lib/sbzr_aux.c:253)

tangent

```
CagdVecStruct *BzrSrfTangent(CagdSrfStruct *Srf,
                             CagdRType u,
                             CagdRType v,
                             CagdSrfDirType Dir)
```

Srf: Bezier surface to evaluate tangent vector for.

u, v: Parametric location of required unit tangent.

Dir: Direction of tangent vector. Either U or V.

Returns: A pointer to a static vector holding the unit tangent information.

Description: Evaluates the unit tangent to a surface at a given parametric location u, v) and given direction Dir.

3.2.118 CagdBBoxArrayFree (cagd_lib/cagd2gen.c:685)

free

```
void CagdBBoxArrayFree(CagdBBoxStruct *BBoxArray, int Size)
```

BBoxArray: To be deallocated.
Size: Of the deallocated array.

Description: Deallocates and frees an array of BBox structure.

3.2.119 CagdBBoxArrayNew (cagd_lib/cagd1gen.c:476)

allocation

```
CagdBBoxStruct *CagdBBoxArrayNew(int Size)
```

Size: Size of BBox array to allocate.

Returns: An array of BBox structures of size Size.

Description: Allocates and resets all slots of an array of BBox structures.

3.2.120 CagdBBoxCopy (cagd_lib/cagd1gen.c:868)

copy

```
CagdBBoxStruct *CagdBBoxCopy(CagdBBoxStruct *BBox)
```

BBox: To be copied.

Returns: A duplicate of BBox.

Description: Allocates and copies all slots of a BBox structure.

3.2.121 CagdBBoxCopyList (cagd_lib/cagd2gen.c:87)

copy

```
CagdBBoxStruct *CagdBBoxCopyList(CagdBBoxStruct *BBoxList)
```

BBoxList: To be copied.

Returns: A duplicated list of bbox's.

Description: Allocates and copies a list of bbox structures.

3.2.122 CagdBBoxFree (cagd_lib/cagd2gen.c:638)

free

```
void CagdBBoxFree(CagdBBoxStruct *BBox)
```

BBox: To be deallocated.

Description: Deallocates and frees all slots of a BBox structure.

3.2.123 CagdBBoxFreeList (cagd_lib/cagd2gen.c:660)

free

```
void CagdBBoxFreeList(CagdBBoxStruct *BBoxList)
```

BBoxList: To be deallocated.

Description: Deallocates and frees a BBox structure list:

3.2.124 CagdBBoxNew (cagd_lib/cagd1gen.c:504)

allocation

```
CagdBBoxStruct *CagdBBoxNew(void)
```

Returns: A BBox structure.

Description: Allocates and resets all slots of a BBox structure.

3.2.125 CagdBilinearSrf (cagd_lib/cagdruld.c:121)

Bilinear surface

surface constructors

```
CagdSrfStruct *CagdBilinearSrf(CagdPtStruct *Pt00,
                               CagdPtStruct *Pt01,
                               CagdPtStruct *Pt10,
                               CagdPtStruct *Pt11)
```

Pt00, Pt01, Pt10, Pt11: The four points to consturct a bilinear between.

Returns: A bilinear surface with four corners at Ptij.

Description: Constructs a bilinear surface between the four provided points.

3.2.126 CagdBoolSumSrf (cagd_lib/cagdbsum.c:31)

Boolean sum

surface constructors

```
CagdSrfStruct *CagdBoolSumSrf(CagdCrvStruct *CrvLeft,
                              CagdCrvStruct *CrvRight,
                              CagdCrvStruct *CrvTop,
                              CagdCrvStruct *CrvBottom)
```

CrvLeft: Left boundary curve of Boolean sum surface to be created.

CrvRight: Right boundary curve of Boolean sum surface to be created.

CrvTop: Top boundary curve of Boolean sum surface to be created.

CrvBottom: Bottom boundary curve of Boolean sum surface to be created.

Returns: A Boolean sum surface constructed using given four curves.

Description: Constructs a Boolean sum surface using the four provided boundary curves. Curve's end points must meet at the four surface corners if surface boundary are to be identical to the four given curves.

3.2.127 CagdBoolSumSrf (cagd_lib/cagdbsum.c:154)

Boolean sum

surface constructors

```
CagdSrfStruct *CagdOneBoolSumSrf(CagdCrvStruct *BndryCrv)
```

BndryCrv: To be subdivided into four curves for a Boolean sum construction.

Returns: A Boolean sum surface constructed using given curve

Description: Constructs a Boolean sum surface using the single boundary curve. The curve is subdivided into four, equally spaced in parameter space, sub-regions which are used as the four curves to the Boolean sum constructor. See CagdBoolSumSrf.

3.2.128 CagdCoerceCrvTo (cagd_lib/cagdcoer.c:325)

coercion

```
CagdCrvStruct *CagdCoerceCrvTo(CagdCrvStruct *Crv, CagdPointType PType)
```

Crv: To be coerced to a new point type PType.

PType: New point type for Crv.

Returns: The new, coerced to PType, curve.

Description: Coerces a curve to a new point type PType. If given curve is E1 or P1 and requested new type is E2 or P2 the Y coefficients are updated to hold the parametric domain of the curve.

3.2.129 CagdCoercePointTo (cagd_lib/cagdcoer.c:219)

coercion

```
void CagdCoercePointTo(CagdRType *NewPoint,
                       CagdPointType NewPType,
                       CagdRType *Points[CAGD_MAX_PT_SIZE],
                       int Index,
                       CagdPointType OldPType)
```

NewPoint: Where the coerced information is to besaved.

NewPType: Point type of the coerced new point.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

OldPType: Point type to be expected from Points.

Description: Coerces Srf/Crv Point from index Index of Points array of Type PType to a new type NewPType. If however Index < 0 Points is considered single point.

3.2.130 CagdCoercePointsTo (cagd_lib/cagdcoer.c:276)

coercion

```
void CagdCoercePointsTo(CagdRType *Points[],
                        int Len,
                        CagdPointType OldPType,
                        CagdPointType NewPType)
```

Points: Where the old and new points are placed.

Len: Length of vectors in the array of vectors, Points.

OldPType: Point type to be expected from Points.

NewPType: Point type of the coerced new point.

Description: Coerces an array of vectors, Points. of point type OldPType to point type NewPType, in place.

3.2.131 CagdCoerceSrfTo (cagd_lib/cagdcoer.c:370)

coercion

```
CagdSrfStruct *CagdCoerceSrfTo(CagdSrfStruct *Srf, CagdPointType PType)
```

Srf: To be coerced to a new point type PType.

PType: New point type for Srf.

Returns: The new, coerced to PType, surface.

Description: Coerces a surface to a new point type PType. If given surface is E1 or P1 and requested new type is E3 or P3 the Y and Z coefficients are updated to hold the parametric domain of the surface.

3.2.132 CagdCoerceToE2 (cagd_lib/cagdcoer.c:30)

coercion

```
void CagdCoerceToE2(CagdRType *E2Point,
                    CagdRType *Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

E2Point: Where the coerced information is to besaved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type E2. If however Index < 0 Points is considered single point.

3.2.133 CagdCoerceToE3 coercion (cagd_lib/cagdcoer.c:76)

```
void CagdCoerceToE3(CagdRType *E3Point,
                    CagdRType *Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

E3Point: Where the coerced information is to besaved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type E3. If however Index < 0 Points is considered single point.

3.2.134 CagdCoerceToP2 (cagd_lib/cagdcoer.c:122)

coercion

```
void CagdCoerceToP2(CagdRType *P2Point,
                    CagdRType *Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

P2Point: Where the coerced information is to besaved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type P2. If however Index < 0 Points is considered single point.

3.2.135 CagdCoerceToP3 (cagd_lib/cagdcoer.c:170)

coercion

```
void CagdCoerceToP3(CagdRType *P3Point,
                    CagdRType *Points[CAGD_MAX_PT_SIZE],
                    int Index,
                    CagdPointType PType)
```

P3Point: Where the coerced information is to besaved.

Points: Array of vectors if Index >= 0, a single point if Index < 0.

Index: Index into the vectors of Points.

PType: Point type to be expected from Points.

Description: Coerce Srf/Crv Point from index Index of Points array of Type PType to a point type P3. If however Index < 0 Points is considered single point.

3.2.136 CagdCrv2CtrlPoly (cagd_lib/cagdmesh.c:22)

control polygon

```
CagdPolylineStruct *CagdCrv2CtrlPoly(CagdCrvStruct *Crv)
```

Crv: To extract a control polygon from.

Returns: The control polygon of Crv.

Description: Extracts the control polygon of a curve as a polyline.

3.2.137 CagdCrvBBox (cagd_lib/cagdbbox.c:23)

bbox

bounding box

```
void CagdCrvBBox(CagdCrvStruct *Crv, CagdBBoxStruct *BBox)
```

Crv: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a freeform curve.

3.2.138 CagdCrvBiNormal (cagd_lib/cagd_aux.c:943)

binormal

```
CagdVecStruct *CagdCrvBiNormal(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To compute unit binormal vector for.

t: Location where to evaluate the binormal of Crv.

Returns: A pointer to a static vector holding the unit binormal information.

Description: Given a curve Crv and a parameter value t, returns the unit binormal direction of Crv at t.

3.2.139 CagdCrvCopy (cagd_lib/cagd1gen.c:638)

copy

```
CagdCrvStruct *CagdCrvCopy(CagdCrvStruct *Crv)
```

Crv: To be copied.

Returns: A duplicate of Crv.

Description: Allocates and copies all slots of a curve structure.

3.2.140 CagdCrvCopyList (cagd_lib/cagd1gen.c:947)

copy

```
CagdCrvStruct *CagdCrvCopyList(CagdCrvStruct *CrvList)
```

CrvList: To be copied.

Returns: A duplicated list of curves.

Description: Allocates and copies a list of curve structures.

3.2.141 CagdCrvDegreeRaise (cagd_lib/cagd_aux.c:520)

degree raising

```
CagdCrvStruct *CagdCrvDegreeRaise(CagdCrvStruct *Crv)
```

Crv: To raise its degree.

Returns: A curve with same geometry as Crv but with one degree higher.

Description: Returns a new curve representing the same curve as Crv but with its degree raised by one.

3.2.142 CagdCrvDegreeRaiseN (cagd_lib/cagd_aux.c:552)

degree raising

```
CagdCrvStruct *CagdCrvDegreeRaiseN(CagdCrvStruct *Crv, int NewOrder)
```

Crv: To raise its degree.

NewOrder: Expected new order of the raised curve.

Returns: A curve with same geometry as Crv but with order that is equal to NewOrder.

Description: Returns a new curve representing the same curve as Crv but with its degree raised to NewOrder

3.2.143 CagdCrvDerive (cagd_lib/cagd_aux.c:219)

derivatives

Hodograph

```
CagdCrvStruct *CagdCrvDerive(CagdCrvStruct *Crv)
```

Crv: To compute its Hodograph curve.

Returns: Resulting hodograph.

Description: Given a curve, computes its derivative curve (Hodograph).

3.2.144 CagdCrvDomain (cagd_lib/cagd_aux.c:28)

domain

parametric domain

```
void CagdCrvDomain(CagdCrvStruct *Crv, CagdRType *TMin, CagdRType *TMax)
```

Crv: To get its parametric domain.

TMin: Where to put the minimal domain's boundary.

TMax: Where to put the maximal domain's boundary.

Description: Returns the parametric domain of a curve.

3.2.145 CagdCrvEval (cagd_lib/cagd_aux.c:65)

evaluation

```
CagdRType *CagdCrvEval(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To evaluate at the given parametric location t.

t: The parameter value at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of curve Crv's point type. If for example the curve's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Given a curve and parameter value t, evaluate the curve at t.

3.2.146 CagdCrvEvalToPolyline (cagd_lib/cbspeval.c:125)

conversion

refinement

evaluation

```
int CagdCrvEvalToPolyline(CagdCrvStruct *Crv,
                          int FineNess,
                          CagdRType *Points[],
                          BspKnotAlphaCoeffType *A,
                          CagdBType OptiLin)
```

Crv: To approximate as a polyline.

FineNess: Control on number of samples.

Points: Where to put the resulting polyline.

A: Optional alpha matrix for refinement.

OptiLin: If TRUE, optimize linear curves.

Returns: The actual number of samples placed in Points. Always less than or eaul to FineNess.

Description: Samples the curve at FineNess location equally spaced in the curve's parametric domain. Computes a refinement alpha matrix (If FineNess > 0), refines the curve and uses refined control polygon as the approximation to the curve. If FineNess $== 0$, Alpha matrix A is used instead. Returns the actual number of points in polyline ($<=$ FineNess). Note this routine may be invoked with Bezier curves as well as Bspline.

3.2.147 CagdCrvFirstMoments (cagd_lib/cbsp_int.c:286)

```
void CagdCrvFirstMoments(CagdCrvStruct *Crv,
                         int n,
                         CagdPType Loc,
                         CagdVType Dir)
```

Crv: To compute zero and first moment.

n: Number of samples the curve should be sampled at.

Loc: Center of curve as zero moment.

Dir: Main direction of curve as first moment.

Description: Computes zero and first moments of a curve.

3.2.148 CagdCrvFree (cagd_lib/cagd2gen.c:174)

free

```
void CagdCrvFree(CagdCrvStruct *Crv)
```

Crv: To be deallocated.

Description: Deallocates and frees all slots of a curve structure.

3.2.149 CagdCrvFreeList (cagd_lib/cagd2gen.c:206)

free

```
void CagdCrvFreeList(CagdCrvStruct *CrvList)
```

CrvList: To be deallocated.

Description: Deallocates and frees a curve structure list:

3.2.150 CagdCrvFromMesh (cagd_lib/cagd_aux.c:655)

isoparametric curves

curve from mesh

```
CagdCrvStruct *CagdCrvFromMesh(CagdSrfStruct *Srf,
                               int Index,
                               CagdSrfDirType Dir)
```

Srf: To extract a curve from.

Index: Index along the mesh of Srf to extract the curve from.

Dir: Direction of extracted curve. Either U or V.

Returns: A curve from Srf. This curve inherit the order and continuity of surface Srf in direction Dir. However, thiscurve is not on surface Srf, in general.

Description: Extracts a curve from the mesh of surface Srf in direction Dir at index Index.

3.2.151 CagdCrvFromSrf (cagd_lib/cagd_aux.c:619)

isoparametric curves

curve from surface

```
CagdCrvStruct *CagdCrvFromSrf(CagdSrfStruct *Srf,
                              CagdRType t,
                              CagdSrfDirType Dir)
```

Srf: To extract an isoparametric curve from.

t: Parameter value of extracted isoparametric curve.

Dir: Direction of extracted isoparametric curve. Either U or V.

Returns: An isoparametric curve of Srf. This curve inherit the order and continuity of surface Srf in direction Dir.

Description: Extracts an isoparametric curve from the surface Srf in direction Dir at the parameter value of t.

3.2.152 CagdCrvIntegrate (cagd_lib/cagd_aux.c:248)

integrals

```
CagdCrvStruct *CagdCrvIntegrate(CagdCrvStruct *Crv)
```

Crv: To compute its integral curve.

Returns: Resulting integral curve.

Description: Given a curve, compute its integral curve.

3.2.153 CagdCrvListBBox (cagd_lib/cagdbbox.c:50)

bbox

bounding box

```
void CagdCrvListBBox(CagdCrvStruct *Crvs, CagdBBoxStruct *BBox)
```

Crvs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a list of freeform curves.

3.2.154 CagdCrvMatTransform (cagd_lib/cagd2gen.c:1013)

scaling

rotation

translation

transformations

```
void CagdCrvMatTransform(CagdCrvStruct *Crv, CagdMType Mat)
```

Crv: To be transformed.

Mat: Defining the transformation.

Description: Applies an homogeneous transformation, in place, to the given curve Crv as specified by homogeneous transformation Mat.

3.2.155 CagdCrvMinMax (cagd_lib/cagdbbox.c:200)

bbox

bounding box

minimum

maximum

```
void CagdCrvMinMax(CagdCrvStruct *Crv,
                   int Axis,
                   CagdRType *Min,
                   CagdRType *Max)
```

Crv: To test for minimum/maximum.

Axis: 0 for W, 1 for X, 2 for Y etc.

Min: Where minimum found value should be place.

Max: Where maximum found value should be place.

Description: Computes a min max bound on a curve in a given axis. The curve is not coerced to anything and the given axis is tested directly where 0 is the W axis and 1, 2, 3 are the X, Y, Z etc.

3.2.156 CagdCrvNew (cagd_lib/cagd1gen.c:27)

allocation

```
CagdCrvStruct *CagdCrvNew(CagdGeomType GType, CagdPointType PType, int Length)
```

GType: Type of geometry the curve should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

Length: Number of control points

Returns: An uninitialized freeform curve.

Description: Allocates the memory required for a new curve.

3.2.157 CagdCrvNodes (cagd_lib/bsp_knot.c:892)

node values

```
CagdRType *CagdCrvNodes(CagdCrvStruct *Crv)
```

Crv: To compute node values for.

Returns: Node values of the given curve.

Description: Returns the nodes of a freeform curve.

3.2.158 CagdCrvNormal (cagd_lib/cagd_aux.c:975)

normal

```
CagdVecStruct *CagdCrvNormal(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To compute unit normal vector for.

t: Location where to evaluate the normal of Crv.

Returns: A pointer to a static vector holding the unit normal information.

Description: Given a curve Crv and a parameter value t, returns the unit normal direction of Crv at t.

3.2.159 CagdCrvRefineAtParams (cagd_lib/cagd_aux.c:426)

refinement

subdivision

```
CagdCrvStruct *CagdCrvRefineAtParams(CagdCrvStruct *Crv,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

Crv: To refine.

Replace: If TRUE, t holds knots in exactly the same length as the length of the knot vector of Crv and t simply replaces the knot vector.

t: Vector of knots with length of n.

n: Length of vector t.

Returns: A refined curve of Crv after insertion of all the knots as specified by vector t of length n.

Description: Given a curve - refines it at the given n knots as defined by vector t. If Replace is TRUE, the values in t replaces current knot vector. Returns pointer to refined surface (Note a Bezier curve will be converted into a Bspline curve).

3.2.160 CagdCrvRegionFromCrv (cagd_lib/cagd_aux.c:345)

regions

subdivision

```
CagdCrvStruct *CagdCrvRegionFromCrv(CagdCrvStruct *Crv,
                                    CagdRType t1,
                                    CagdRType t2)
```

Crv: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Returns: Sub-region extracted from Crv from t1 to t2.

Description: Given a curve - extracts a sub-region within the domain specified by t1 and t2.

3.2.161 CagdCrvReverse (cagd_lib/cagd_aux.c:462)

reverse

```
CagdCrvStruct *CagdCrvReverse(CagdCrvStruct *Crv)
```

Crv: To be reversed.

Returns: Reversed curve of Crv.

Description: Returns a new curve that is the reversed curve of Crv by reversing the control polygon and the knot vector of Crv is a Bspline curve. See also BspKnotReverse.

3.2.162 CagdCrvSubdivAtParam (cagd_lib/cagd_aux.c:312)

subdivision

```
CagdCrvStruct *CagdCrvSubdivAtParam(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To subdivide at the prescibed parameter value t.

t: The parameter to subdivide the curve Crv at.

Returns: A list of the two curves resulting from the process of subdivision.

Description: Given a curve - subdivides it into two curves at the given parameter value t. Returns pointer to first curve in a list of two subdivided curves.

3.2.163 CagdCrvTangent (cagd_lib/cagd_aux.c:911)

tangent

```
CagdVecStruct *CagdCrvTangent(CagdCrvStruct *Crv, CagdRType t)
```

Crv: To compute unit tangent vector for.

t: Location where to evaluate the tangent of Crv.

Returns: A pointer to a static vector holding the unit tanegnt information.

Description: Given a curve Crv and a parameter value t, returns the unit tangent direction of Crv at t.

3.2.164 CagdCrvToMesh (cagd_lib/cagd_aux.c:692)

curve from mesh

```
void CagdCrvToMesh(CagdCrvStruct *Crv,
                   int Index,
                   CagdSrfDirType Dir,
                   CagdSrfStruct *Srf)
```

Crv: To substitute into the surface Srf.

Index: Of mesh where the curve Crv should be substituted in.

Dir: Either U or V.

Srf: That a row or a column of should be replaced by Crv.

Description: Substitutes a row/column of surface Srf from the given curve Crv at surface direction Dir and mesh index Index. Curve must have the same PtType/Length as the surface in the selected direction.

3.2.165 CagdCrvTransform (cagd_lib/cagd2gen.c:890)

scaling

translation

transformations

```
void CagdCrvTransform(CagdCrvStruct *Crv,
                      CagdRType *Translate,
                      CagdRType Scale)
```

Crv: To be affinely transformed.

Translate: Translation amount.

Scale: Scaling amount.

Description: Applies an affine transform, in place, to given curve Crv as specified by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.

3.2.166 CagdCtlPtArrayFree (cagd_lib/cagd2gen.c:475)

free

```
void CagdCtlPtArrayFree(CagdCtlPtStruct *CtlPtArray, int Size)
```

CtlPtArray: To be deallocated.

Size: Of the deallocated array.

Description: Deallocates and frees an array of CtlPt structure.

3.2.167 CagdCtlPtArrayNew (cagd_lib/cagd1gen.c:319)

allocation

```
CagdCtlPtStruct *CagdCtlPtArrayNew(CagdPointType PtType, int Size)
```

Size: Size of CtlPt array to allocate.

Returns: An array of CtlPt structures of size Size.

Description: Allocates and resets all slots of an array of CtlPt structures.

3.2.168 CagdCtlPtCopy (cagd_lib/cagd1gen.c:793)

copy

```
CagdCtlPtStruct *CagdCtlPtCopy(CagdCtlPtStruct *CtlPt)
```

CtlPt: To be copied.

Returns: A duplicate of CtlPt.

Description: Allocates and copies all slots of a CtlPt structure.

3.2.169 CagdCtlPtCopyList (cagd_lib/cagd1gen.c:1063)

copy

```
CagdCtlPtStruct *CagdCtlPtCopyList(CagdCtlPtStruct *CtlPtList)
```

CtlPtList: To be copied.

Returns: A duplicated list of CtlPt's.

Description: Allocates and copies a list of CtlPt structures.

3.2.170 CagdCtlPtFree (cagd_lib/cagd2gen.c:428)

free

```
void CagdCtlPtFree(CagdCtlPtStruct *CtlPt)
```

CtlPt: To be deallocated.

Description: Deallocates and frees all slots of a CtlPt structure.

3.2.171 CagdCtlPtFreeList (cagd_lib/cagd2gen.c:450)

free

```
void CagdCtlPtFreeList(CagdCtlPtStruct *CtlPtList)
```

CtlPtList: To be deallocated.

Description: Deallocates and frees a CtlPt structure list:

3.2.172 CagdCtlPtNew (cagd_lib/cagd1gen.c:348)

allocation

```
CagdCtlPtStruct *CagdCtlPtNew(CagdPointType PtType)
```

Returns: A CtlPt structure.

Description: Allocates and resets all slots of a CtlPt structure.

3.2.173 CagdDbg (cagd_lib/cagd_dbg.c:24)

debugging

```
void CagdDbg(void *Obj)
```

Obj: Either a curve or a surface - to be printed to stderr.

Description: Prints curves and surfaces to stderr. Should be linked to programs for debugging purposes, so curves and surfaces may be inspected from the debugger.

3.2.174 CagdDescribeError (cagd_lib/cagd_err.c:93)

error handling

```
char *CagdDescribeError(CagdFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this cagd library as well as other users. Raised error will cause an invokation of CagdFatalError function which decides how to handle this error. CagdFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

3.2.175 CagdDistPtPlane (cagd_lib/mshplanr.c:175)

point plane distance

```
CagdRType CagdDistPtPlane(CagdPlaneStruct *Plane,
                          CagdRType **Points,
                          int Index,
                          int MaxDim)
```

Plane: To compute the distance to.

Points: To compute the distance from.

Index: Index in Points for the point to consider.

MaxDim: Number of dimensions to consider. Less or equal to three.

Returns: Resulting distance.

Description: Computes and returns distance between point Index and given plane which is assumed to be normalized, so that the A B C plane;s normal has a unit length. Also assumes the Points are non rational with MaxDim dimension.

3.2.176 CagdDistanceTwoCtlPts (cagd_lib/mshplanr.c:29)

distance

```
CagdRType CagdDistanceTwoCtlPts(CagdRType **Points,
                                int Index1,
                                int Index2,
                                CagdPointType PType)
```

Points: To compute the distance between.

Index1: Index of first point to consider.

Index2: Index of second point to consider.

PType: Type of points. Must be E2 or E3

Returns: Euclidean distance between the two points.

Description: Computes the distance between two control points at given indices Index?. Only E2 or E3 point types are supported.

3.2.177 CagdEditSingleCrvPt (cagd_lib/cagdedit.c:31)

curve editing

```
CagdCrvStruct *CagdEditSingleCrvPt(CagdCrvStruct *Crv,
                                   CagdCtlPtStruct *CtlPt,
                                   int Index,
                                   CagdBType Write)
```

Crv: Curve to be modified/query.

CtlPt: New control point to be substituted into Crv. Must carry the same PType as Crv if to be written to Crv.

Index: In curve CRV's control polygon to substitute/query CtlPt.

Write: If TRUE CtlPt is copied into Crv, if FALSE the point is copied from Crv to CtlPt.

Returns: If Write is TRUE, the new modified curve, if WRITE is FALSE, NULL.

Description: Provides the way to modify/get a single control point into/from the curve.

3.2.178 CagdEditSingleSrfPt (cagd_lib/cagdedit.c:86)

surface editing

```
CagdSrfStruct *CagdEditSingleSrfPt(CagdSrfStruct *Srf,
                                   CagdCtlPtStruct *CtlPt,
                                   int UIndex,
                                   int VIndex,
                                   CagdBType Write)
```

Srf: Surface to be modified/query.

CtlPt: New control point to be substituted into Srf. Must carry the same PType as Srf if to be written to Srf.

UIndex, VIndex: In surface Srf's control mesh to substitute/query CtlPt.

Write: If TRUE CtlPt is copied into Srf, if FALSE the point is copied from Srf to CtlPt.

Returns: If Write is TRUE, the new modified curve, if WRITE is FALSE, NULL.

Description: Provides the way to modify/get a single control point into/from a surface.

3.2.179 CagdEstimateCrvColinearity (cagd_lib/mshplanr.c:245)

conversion

colinearity

```
CagdRType CagdEstimateCrvColinearity(CagdCrvStruct *Crv)
```

Crv: To measure its colinearity.

Returns: Colinearity relative measure.

Description: Tests polygonal colinearity by testing the distance of interior control points from the line connecting the two control polygon end points. Returns a relative ratio of deviation from line relative to its length. Zero means all points are colinear. If two end points are same (no line can be fit) INFINITY is returned.

3.2.180 CagdEstimateSrfPlanarity (cagd_lib/mshplanr.c:332)

conversion

coplanarity

```
CagdRType CagdEstimateSrfPlanarity(CagdSrfStruct *Srf)
```

Srf: To measure its coplanarity.

Returns: Coplanarity measure.

Description: ests mesh colinearity by testing the distance of interior points from the plane thru 3 corner points. Returns a relative ratio of deviation from plane relative to its size. Zero means all points are coplanar. If end points are same (no plane can be fit) INFINITY is returned.

3.2.181 CagdEvaluateSurfaceVecField (cagd_lib/cagd_aux.c:176)

normal

vector field

```
void CagdEvaluateSurfaceVecField(CagdVType Vec,
                                 CagdSrfStruct *VecFieldSrf,
                                 CagdRType U,
                                 CagdRType V)
```

Vec: Where resulting unit length vector is to be saved.

VecFieldSrf: A surface representing a vector field.

U, V: Parameter locations.

Description: Evaluates a vector field surface to a unit size vector. If fails, moves a tad until success. Useful for normal field evaluations.

3.2.182 CagdExtrudeSrf (cagd_lib/cagdextr.c:26)

surface constructors

```
CagdSrfStruct *CagdExtrudeSrf(CagdCrvStruct *Crv, CagdVecStruct *Vec)
```

Crv: To exturde in direction specified by Vec.

Vec: Direction as well as magnitude of extursion.

Returns: An extrusion surface with Orders of the original Crv order and 2 in the extrusion direction.

Description: Constructs an extrusion surface in the Vector direction for the given profile curve. Input curve can be either a Bspline or a Bezier curve and the resulting output surface will be of the same type.

3.2.183 CagdFatalError (cagd_lib/cagd_ftl.c:25)

error handling

```
void CagdFatalError(CagdFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Description: Trap Cagd_lib errors right here. Provides a default error handler for the cagd library. Gets an error description using CagdDescribeError, prints it and exit the program using exit.

3.2.184 CagdFitPlaneThruCtlPts (cagd_lib/mshplanr.c:71)

plane fit

```
CagdRType CagdFitPlaneThruCtlPts(CagdPlaneStruct *Plane,
                                 CagdPointType PType,
                                 CagdRType **Points,
                                 int Index1,
                                 int Index2,
                                 int Index3,
                                 int Index4)
```

Plane: To compute and save here.

PType: Point type expected of four points. Must be E2 or E3.

Points: Point array where to look for the four points.

Index?: Four indices of the points.

Returns: Measure on the distance between the data points, 0.0 if fitting failed.

Description: Fits a plane through the four points from Points indices Index?. Points may be either E2 or E3 only. Returns 0.0 if failed to fit a plane, otherwise a measure on the size of the mesh data (distance between points) is returned.

3.2.185 CagdIChooseK (cagd_lib/cbzreval.c:296)

evaluation

combinatorics

```
CagdRType CagdIChooseK(int i, int k)
```

i, k: Coefficients of i choose k.

Returns: Result of i choose k, in floating point, to prevent from overflows.

Description: Evaluates the following (in floating point arithmetic):

```
 k         k!
( ) = -------------
 i    i! * (k - i)!
```

3.2.186 CagdListLast (cagd_lib/cagd2gen.c:833)

```
VoidPtr CagdListLast(VoidPtr List)
```

List: To return its last element,

Returns: Last element.

Description: Returns the last element of given list of cagd library objects.

3.2.187 CagdListLength (cagd_lib/cagd2gen.c:859)

```
int CagdListLength(VoidPtr List)
```

List: List of cagd objects.

Returns: Length of list.

Description: Computes the length of a list.

3.2.188 CagdListReverse (cagd_lib/cagd2gen.c:801)

reverse

```
VoidPtr CagdListReverse(VoidPtr List)
```

List: To be reversed.

Returns: Reversed list.

Description: Reverses a list of cagd library objects, in place.

3.2.189 CagdMakeCrvsCompatible (cagd_lib/cagdcmpt.c:35)

compatibility

```
CagdBType CagdMakeCrvsCompatible(CagdCrvStruct **Crv1,
                                 CagdCrvStruct **Crv2,
                                 CagdBType SameOrder,
                                 CagdBType SameKV)
```

Crv1, Crv2: Two curves to be made compatible, in place.

SameOrder: If TRUE, this routine make sure they share the same order.

SameKV: If TRUE, this routine make sure they share the same knot vector and hence continuity. *

Returns: TRUE if successful, FALSE otherwise.

Description: Given two curves, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same curve type.
3. Raising the degree of the lower one to be the same as the higher.
4. Refining them to a common knot vector (If Bspline and SameOrder).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both curves are modified IN PLACE.

3.2.190 CagdMakeSrfsCompatible (cagd_lib/cagdcmpt.c:183)

compatibility

```
CagdBType CagdMakeSrfsCompatible(CagdSrfStruct **Srf1,
                                 CagdSrfStruct **Srf2,
                                 CagdBType SameUOrder,
                                 CagdBType SameVOrder,
                                 CagdBType SameUKV,
                                 CagdBType SameVKV)
```

Srf1, Srf2: Two surfaces to be made compatible, in place.

SameUOrder: If TRUE, this routine make sure they share the same U order.

SameVOrder: If TRUE, this routine make sure they share the same order.

SameUKV: If TRUE, this routine make sure they share the same U knot vector and hence continuity. *

SameVKV: If TRUE, this routine make sure they share the same knot vector and hence continuity.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two surfaces, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same curve type.
3. Raising the degree of the lower one to be the same as the higher.
4. Refining them to a common knot vector (If Bspline and SameOrder).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both surface are modified IN PLACE.

3.2.191 CagdMatTransform (cagd_lib/cagd2gen.c:1140)

scaling

rotation

translation

transformations

```
void CagdMatTransform(CagdRType **Points,
                      int Len,
                      int MaxCoord,
                      CagdBType IsNotRational,
                      CagdMType Mat)
```

Points: To be affinely transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

IsNotRational: Do we have weights as vector Points[0]?

Mat: Defining the transformation.

Description: Applies an homogeneous transformation, in place, to given set of points Points which as array of vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.).

3.2.192 CagdMergeBBox (cagd_lib/cagdbbox.c:167)

bbox

bounding box

```
void CagdMergeBBox(CagdBBoxStruct *DestBBox, CagdBBoxStruct *SrcBBox)
```

DestBBox: One BBox operand as well as the result.

SrcBBox: Second BBox operand.

Description: Merges (union) two bounding boxes into one, in place.

3.2.193 CagdMergeCrvCrv (cagd_lib/cagdcmrg.c:36)

merge

```
CagdCrvStruct *CagdMergeCrvCrv(CagdCrvStruct *Crv1,
                               CagdCrvStruct *Crv2,
                               int InterpolateDiscont)
```

Crv1: To connect to Crv1's starting location at its end.

Crv2: To connect to Crv2's end location at its start.

InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged curve.

Description: Merges two curves by connecting the end of Crv1 to the beginning of Crv2. If the end of Crv1 is identical to the beginning of Crv2 then the result is as expected. However, if the curves do not meet their end points are linearly interpolated if InterpolateDiscont is TRUE or simply blended out in a freeform shape if If InterpolateDiscont is FALSE.

3.2.194 CagdMergeCrvList (cagd_lib/cagdcmrg.c:168)

merge

```
CagdCrvStruct *CagdMergeCrvList(CagdCrvStruct *CrvList, int InterpDiscont)
```

CrvList: To connect into one curve.

InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged curve.

Description: Merges a list of curves by connecting the end of one curve to the begining of the next. See also CagdMergeCrvCrv.

3.2.195 CagdMergeCrvPt (cagd_lib/cagdcmrg.c:205)

merge

```
CagdCrvStruct *CagdMergeCrvPt(CagdCrvStruct *Crv, CagdPtStruct *Pt)
```

Crv: To connect to Pt its end.

Pt: To connect to Crv's end point.

Returns: The merged curve.

Description: Merges a curve and a point by connecting the end of Crv to Pt, using a linear segment.

3.2.196 CagdMergePointType (cagd_lib/cagdcoer.c:429)

coercion

```
CagdPointType CagdMergePointType(CagdPointType PType1, CagdPointType PType2)
```

PType1, PType2: To point types to find the point type of their union.

Returns: A point type of the union of the spaces of PType1 and PType2.

Description: Returns a point type which spans the spaces of both two given point types.

3.2.197 CagdMergePtCrv (cagd_lib/cagdcmrg.c:288)

merge

```
CagdCrvStruct *CagdMergePtCrv(CagdPtStruct *Pt, CagdCrvStruct *Crv)
```

Pt: To connect to Crv's starting point.

Crv: To connect to Pt its starting point.

Returns: The merged curve.

Description: Merges a point and a curve by connecting Pt to the starting point of Crv, using a linear segment.

3.2.198 CagdMergePtPt (cagd_lib/cagdcmrg.c:370)

merge

```
CagdCrvStruct *CagdMergePtPt(CagdPtStruct *Pt1, CagdPtStruct *Pt2)
```

Pt1, Pt2: Two points to connect using a linear segment.

Returns: The merged curve.

Description: Merges two points by connecting Pt1 to Pt2, using a linear segment.

3.2.199 CagdMergeSrfList (cagd_lib/cagdsmrg.c:298)

merge

```
CagdSrfStruct *CagdMergeSrfList(CagdSrfStruct *SrfList,
                                CagdSrfDirType Dir,
                                CagdBType SameEdge,
                                int InterpolateDiscont)
```

SrfList: To connect into one surface.

Dir: Direction the merge should take place. Either U or V.

SameEdge: If the two surfaces sharea common edge.

InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surface.

Description: Merges a list of surfaces by connecting the end of one surface to the begining of the next. See also CagdMergeSrfSrf.

3.2.200 CagdMergeSrfSrf (cagd_lib/cagdsmrg.c:39)

merge

```
CagdSrfStruct *CagdMergeSrfSrf(CagdSrfStruct *Srf1,
                               CagdSrfStruct *Srf2,
                               CagdSrfDirType Dir,
                               CagdBType SameEdge,
                               int InterpolateDiscont)
```

Srf1: To connect to Srf1's starting boundary at its end.
Srf2: To connect to Srf2's end boundary at its start.
Dir: Direction the merge should take place. Either U or V.
SameEdge: If the two surfaces sharea common edge.
InterpolateDiscont: If TRUE, linearly interpolate discontinuity.

Returns: The merged surface.

Description: Merges two surfaces in the requested direction Dir. If SameEdge, it is assumed last edge of Srf1 is identical to first edge of Srf2 and one row is dropped from new mesh. Otherwise a ruled surface is fit between the two edges.

3.2.201 CagdPeriodicCrvNew (cagd_lib/cagd1gen.c:73)

allocation

```
CagdCrvStruct *CagdPeriodicCrvNew(CagdGeomType GType,
                                  CagdPointType PType,
                                  int Length,
                                  CagdBType Periodic)
```

GType: Type of geometry the curve should be - Bspline, Bezier etc.
PType: Type of control points (E2, P3, etc.).
Length: Number of control points
Periodic: Is this curve periodic?

Returns: An uninitialized freeform curve.

Description: Allocates the memory required for a new, possibly periodic, curve.

3.2.202 CagdPeriodicSrfNew (cagd_lib/cagd1gen.c:173)

allocation

```
CagdSrfStruct *CagdPeriodicSrfNew(CagdGeomType GType,
                                  CagdPointType PType,
                                  int ULength,
                                  int VLength,
                                  CagdBType UPeriodic,
                                  CagdBType VPeriodic)
```

GType: Type of geometry the surface should be - Bspline, Bezier etc.
PType: Type of control points (E2, P3, etc.).
ULength: Number of control points in the U direction.
VLength: Number of control points in the V direction.
UPeriodic: Is this surface periodic in the U direction?
VPeriodic: Is this surface periodic in the V direction?

Returns: An uninitialized freeform surface.

Description: Allocates the memory required for a new, possibly periodic, surface.

3.2.203 CagdPlaneArrayFree (cagd_lib/cagd2gen.c:615)

free

```
void CagdPlaneArrayFree(CagdPlaneStruct *PlaneArray, int Size)
```

PlaneArray: To be deallocated.
Size: Of the deallocated array.

Description: Deallocates and frees an array of plane structure.

3.2.204 CagdPlaneArrayNew (cagd_lib/cagd1gen.c:424)

allocation

```
CagdPlaneStruct *CagdPlaneArrayNew(int Size)
```

Size: Size of Plane array to allocate.

Returns: An array of Plane structures of size Size.

Description: Allocates and resets all slots of an array of Plane structures.

3.2.205 CagdPlaneCopy (cagd_lib/cagd1gen.c:843)

copy

```
CagdPlaneStruct *CagdPlaneCopy(CagdPlaneStruct *Plane)
```

Plane: To be copied.

Returns: A duplicate of Plane.

Description: Allocates and copies all slots of a Plane structure.

3.2.206 CagdPlaneCopyList (cagd_lib/cagd2gen.c:58)

copy

```
CagdPlaneStruct *CagdPlaneCopyList(CagdPlaneStruct *PlaneList)
```

PlaneList: To be copied.

Returns: A duplicated list of planes.

Description: Allocates and copies a list of plane structures.

3.2.207 CagdPlaneFree (cagd_lib/cagd2gen.c:568)

free

```
void CagdPlaneFree(CagdPlaneStruct *Plane)
```

Plane: To be deallocated.

Description: Deallocates and frees all slots of a plane structure.

3.2.208 CagdPlaneFreeList (cagd_lib/cagd2gen.c:590)

free

```
void CagdPlaneFreeList(CagdPlaneStruct *PlaneList)
```

PlaneList: To be deallocated.

Description: Deallocates and frees a plane structure list:

3.2.209 CagdPlaneNew (cagd_lib/cagd1gen.c:452)

allocation

```
CagdPlaneStruct *CagdPlaneNew(void)
```

Returns: A Plane structure.

Description: Allocates and resets all slots of a Plane structure.

3.2.210 CagdPointsBBox (cagd_lib/cagdbbox.c:130)

bbox

bounding box

```
void CagdPointsBBox(CagdRType **Points, int Length, CagdBBoxStruct *BBox)
```

Points: To compute bounding box for.

Length: Length of vectors of Points array.

BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a set of control points.

3.2.211 CagdPolygonArrayNew (cagd_lib/cagd1gen.c:528)

allocation

```
CagdPolygonStruct *CagdPolygonArrayNew(int Size)
```

Size: Size of Polygon array to allocate.

Returns: An array of Polygon structures of size Size.

Description: Allocates and resets all slots of an array of Polygon structures.

3.2.212 CagdPolygonCopy (cagd_lib/cagd1gen.c:893)

copy

```
CagdPolygonStruct *CagdPolygonCopy(CagdPolygonStruct *Poly)
```

Poly: To be copied.

Returns: A duplicate of Polygon.

Description: Allocates and copies all slots of a Polygon structure.

3.2.213 CagdPolygonCopyList (cagd_lib/cagd2gen.c:145)

copy

```
CagdPolygonStruct *CagdPolygonCopyList(CagdPolygonStruct *PolyList)
```

PolyList: To be copied.

Returns: A duplicated list of polygons.

Description: Allocates and copies a list of polygon structures.

3.2.214 CagdPolygonFree (cagd_lib/cagd2gen.c:755)

free

```
void CagdPolygonFree(CagdPolygonStruct *Poly)
```

Poly: To be deallocated.

Description: Deallocates and frees all slots of a polygon structure.

3.2.215 CagdPolygonFreeList (cagd_lib/cagd2gen.c:777)

free

```
void CagdPolygonFreeList(CagdPolygonStruct *PolyList)
```

PolyList: To be deallocated.

Description: Deallocates and frees a polygon structure list:

3.2.216 CagdPolygonNew (cagd_lib/cagd1gen.c:556)

allocation

```
CagdPolygonStruct *CagdPolygonNew(void)
```

Returns: A Polygon structure.

Description: Allocates and resets all slots of a Polygon structure.

3.2.217 CagdPolylineArrayNew (cagd_lib/cagd1gen.c:580)

allocation

```
CagdPolylineStruct *CagdPolylineArrayNew(int Length, int Size)
```

Size: Size of Polyline array to allocate.

Returns: An array of Polyline structures of size Size.

Description: Allocates and resets all slots of an array of Polyline structures.

3.2.218 CagdPolylineCopy (cagd_lib/cagd1gen.c:918)

copy

```
CagdPolylineStruct *CagdPolylineCopy(CagdPolylineStruct *Poly)
```

Poly: To be copied.

Returns: A duplicate of Polyline.

Description: Allocates and copies all slots of a Polyline structure.

3.2.219 CagdPolylineCopyList (cagd_lib/cagd2gen.c:116)

copy

```
CagdPolylineStruct *CagdPolylineCopyList(CagdPolylineStruct *PolyList)
```

PolyList: To be copied.

Returns: A duplicated list of polylines.

Description: Allocates and copies a list of polyline structures.

3.2.220 CagdPolylineFree (cagd_lib/cagd2gen.c:708)

free

```
void CagdPolylineFree(CagdPolylineStruct *Poly)
```

Poly: To be deallocated.

Description: Deallocates and frees all slots of a polyline structure.

3.2.221 CagdPolylineFreeList (cagd_lib/cagd2gen.c:731)

free

```
void CagdPolylineFreeList(CagdPolylineStruct *PolyList)
```

PolyList: To be deallocated.

Description: Deallocates and frees a polyline structure list:

3.2.222 CagdPolylineNew (cagd_lib/cagd1gen.c:610)

allocation

```
CagdPolylineStruct *CagdPolylineNew(int Length)
```

Returns: A Polyline structure.

Description: Allocates and resets all slots of a Polyline structure.

3.2.223 CagdPromoteCrvToSrf (cagd_lib/cagdruld.c:153)

```
CagdSrfStruct *CagdPromoteCrvToSrf(CagdCrvStruct *Crv, CagdSrfDirType Dir)
```

Crv: A Crv to promote into a surface

Dir: Direction of ruling. Either U or V.

Returns: The surface promoted from Crv.

Description: Promotes a curve to a surface by creating a ruled surface between the curve to itself. Dir controls if the curve should be U or V surface direction. The resulting surface is degenerate in that its speed is zero in the ruled direction and hence thesurface is not regular.

3.2.224 CagdPtArrayFree (cagd_lib/cagd2gen.c:405)

free

```
void CagdPtArrayFree(CagdPtStruct *PtArray, int Size)
```

PtArray: To be deallocated.
Size: Of the deallocated array.
Description: Deallocates and frees an array of Pt structure.

3.2.225 CagdPtArrayNew (cagd_lib/cagd1gen.c:268)

allocation

```
CagdPtStruct *CagdPtArrayNew(int Size)
```

Size: Size of Pt array to allocate.
Returns: An array of Pt structures of size Size.
Description: Allocates and resets all slots of an array of Pt structures.

3.2.226 CagdPtCopy (cagd_lib/cagd1gen.c:768)

copy

```
CagdPtStruct *CagdPtCopy(CagdPtStruct *Pt)
```

Pt: To be copied.
Returns: A duplicate of Pt.
Description: Allocates and copies all slots of a Pt structure.

3.2.227 CagdPtCopyList (cagd_lib/cagd1gen.c:1034)

copy

```
CagdPtStruct *CagdPtCopyList(CagdPtStruct *PtList)
```

PtList: To be copied.
Returns: A duplicated list of points.
Description: Allocates and copies a list of point structures.

3.2.228 CagdPtFree (cagd_lib/cagd2gen.c:358)

free

```
void CagdPtFree(CagdPtStruct *Pt)
```

Pt: To be deallocated.
Description: Deallocates and frees all slots of a point structure.

3.2.229 CagdPtFreeList (cagd_lib/cagd2gen.c:380)

free

```
void CagdPtFreeList(CagdPtStruct *PtList)
```

PtList: To be deallocated.
Description: Deallocates and frees a point structure list:

3.2.230 CagdPtNew (cagd_lib/cagd1gen.c:295)

allocation

```
CagdPtStruct *CagdPtNew(void)
```

Returns: A Pt structure.
Description: Allocates and resets all slots of a Pt structure.

3.2.231 CagdRuledSrf (cagd_lib/cagdruld.c:31)

ruled surface

surface constructors

```
CagdSrfStruct *CagdRuledSrf(CagdCrvStruct *Crv1,
                            CagdCrvStruct *Crv2,
                            int OtherOrder,
                            int OtherLen)
```

Crv1, Crv2: The two curves to form a ruled surface in between.

OtherOrder: Usually two, but one can specify higher orders in the ruled direction. OtherOrder must never be larger than OrderLen.

OtherLen: Usually two control points in the ruled direction which necesitates a linear interpolation.

Returns: The rule surface.

Description: Constructs a ruled surface between the two provided curves. OtherOrder and OtherLen (equal for Bezier) specifies the desired order and refineness level (if Bspline) of the other ruled direction.

3.2.232 CagdSetLinear2Poly (cagd_lib/cagd2gen.c:1261)

polygonization

polygonal approximation

```
void CagdSetLinear2Poly(CagdLin2PolyType Lin2Poly)
```

Lin2Poly: Specification.

Description: Sets the way (co)linear surfaces are converted into polygons.

3.2.233 CagdSrf2CtrlMesh (cagd_lib/cagdmesh.c:55)

control mesh

```
CagdPolylineStruct *CagdSrf2CtrlMesh(CagdSrfStruct *Srf)
```

Srf: To extract a control mesh from.

Returns: The control mesh of Srf.

Description: Extracts the control mesh of a surface as a list of polylines.

3.2.234 CagdSrfBBox (cagd_lib/cagdbbox.c:76)

bbox

bounding box

```
void CagdSrfBBox(CagdSrfStruct *Srf, CagdBBoxStruct *BBox)
```

Srf: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a freeform surface.

3.2.235 CagdSrfCopy (cagd_lib/cagd1gen.c:686)

copy

```
CagdSrfStruct *CagdSrfCopy(CagdSrfStruct *Srf)
```

Srf: To be copied.

Returns: A duplicate of Srf.

Description: Allocates and copies all slots of a surface structure.

3.2.236 CagdSrfCopyList (cagd_lib/cagd1gen.c:976)

copy

```
CagdSrfStruct *CagdSrfCopyList(CagdSrfStruct *SrfList)
```

SrfList: To be copied.

Returns: A duplicated list of surfaces.

Description: Allocates and copies a list of surface structures.

3.2.237 CagdSrfDegreeRaise (cagd_lib/cagd_aux.c:584)

degree raising

```
CagdSrfStruct *CagdSrfDegreeRaise(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To raise its degree.

Dir: Direction of degree raising. Either U or V.

Returns: A surface with same geometry as Srf but with one degree higher.

Description: Returns a new surface representing the same surface as Srf but with its degree raised by one.

3.2.238 CagdSrfDerive (cagd_lib/cagd_aux.c:279)

derivatives

partial derivatives

```
CagdSrfStruct *CagdSrfDerive(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To compute its derivative surface in direction Dir.

Dir: Direction of differentiation. Either U or V.

Returns: Resulting partial derivative surface.

Description: Given a surface, computes its partial derivative in the prescibed direction Dir.

3.2.239 CagdSrfDomain (cagd_lib/cagd_aux.c:102)

domain

parametric domain

```
void CagdSrfDomain(CagdSrfStruct *Srf,
                   CagdRType *UMin,
                   CagdRType *UMax,
                   CagdRType *VMin,
                   CagdRType *VMax)
```

Srf: To get its parametric domain.

UMin: Where to put the minimal U domain's boundary.

UMax: Where to put the maximal U domain's boundary.

VMin: Where to put the minimal V domain's boundary.

VMax: Where to put the maximal V domain's boundary.

Description: Returns the parametric domain of a surface.

3.2.240 CagdSrfEval (cagd_lib/cagd_aux.c:141)

evaluation

```
CagdRType *CagdSrfEval(CagdSrfStruct *Srf, CagdRType u, CagdRType v)
```

Srf: To evaluate at the given parametric location (u, v).

u, v: The parameter values at which the curve Crv is to be evaluated.

Returns: A vector holding all the coefficients of all components of surface Srf's point type. If for example the surface's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Given a surface and parameter values u, v, evaluate the surface at (u, v).

3.2.241 CagdSrfFree (cagd_lib/cagd2gen.c:230)

free

```
void CagdSrfFree(CagdSrfStruct *Srf)
```

Srf: To be deallocated.

Description: Deallocates and frees all slots of a surface sstructure.

3.2.242 CagdSrfFreeList (cagd_lib/cagd2gen.c:264)

free

```
void CagdSrfFreeList(CagdSrfStruct *SrfList)
```

SrfList: To be deallocated.

Description: Deallocates and frees a surface structure list:

3.2.243 CagdSrfFromCrvs (cagd_lib/cagdcsrf.c:31)

surface constructors

```
CagdSrfStruct *CagdSrfFromCrvs(CagdCrvStruct *CrvList, int OtherOrder)
```

CrvList: List of curves to consturct a surface with.

OtherOrder: Other order of surface.

Returns: Constructed surface from curves.

Description: Constructs a surface using a set of curves. Curves are made to be compatible and then each is substituted into the new surface's mesh as a row. If the OtherOrder is less than the number of curves, number of curves is used. A knot vector is formed with uniform open end for the other direction, so it interpolates the first and last curves. Note, however, that only the first and the last curves are interpolated if OtherOrder is greater than 2.

3.2.244 CagdSrfListBBox (cagd_lib/cagdbbox.c:103)

bbox

bounding box

```
void CagdSrfListBBox(CagdSrfStruct *Srfs, CagdBBoxStruct *BBox)
```

Srfs: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a list of freeform surfaces.

3.2.245 CagdSrfMatTransform (cagd_lib/cagd2gen.c:1072)

scaling

rotation

translation

transformations

```
void CagdSrfMatTransform(CagdSrfStruct *Srf, CagdMType Mat)
```

Srf: To be transformed.

Mat: Defining the transformation.

Description: Applies an homogeneous transformation, in place, to the given surface Srf as specified by homogeneous transformation Mat.

3.2.246 CagdSrfMinMax (cagd_lib/cagdbbox.c:246)

bbox

bounding box

minimum

maximum

```
void CagdSrfMinMax(CagdSrfStruct *Srf,
                   int Axis,
                   CagdRType *Min,
                   CagdRType *Max)
```

Srf: To test for minimum/maximum.

Axis: 0 for W, 1 for X, 2 for Y etc.

Min: Where minimum found value should be place.

Max: Where maximum found value should be place.

Description: Computes a min max bound on a surface in a given axis. The surface is not coerced to anything and the given axis is tested directly where 0 is the W axis and 1, 2, 3 are the X, Y, Z etc.

3.2.247 CagdSrfNew (cagd_lib/cagd1gen.c:119)

allocation

```
CagdSrfStruct *CagdSrfNew(CagdGeomType GType,
                          CagdPointType PType,
                          int ULength,
                          int VLength)
```

GType: Type of geometry the surface should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

Returns: An uninitialized freeform surface.

Description: Allocates the memory required for a new surface.

3.2.248 CagdSrfNodes (cagd_lib/bsp_knot.c:931)

node values

```
CagdRType *CagdSrfNodes(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: To compute node values for.

Dir: Either the U or the V parametric direction.

Returns: Node values of the given surface and given parametric direction.

Description: Returns the nodes of a freeform surface.

3.2.249 CagdSrfNormal (cagd_lib/cagd_aux.c:1043)

normal

```
CagdVecStruct *CagdSrfNormal(CagdSrfStruct *Srf, CagdRType u, CagdRType v)
```

Srf: To compute unit normal vector for.

u, v: Location where to evaluate the normal of Srf.

Returns: A pointer to a static vector holding the unit normal information.

Description: Given a surface Srf and a parameter values u, v, returns the unit normal vector of Srf.

3.2.250 CagdSrfRefineAtParams (cagd_lib/cagd_aux.c:874)

refinement

subdivision

```
CagdSrfStruct *CagdSrfRefineAtParams(CagdSrfStruct *Srf,
                                     CagdSrfDirType Dir,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

Srf: To refine.

Dir: Direction of refinement. Either U or V.

Replace: If TRUE, t holds knots in exactly the same length as the length of the knot vector of Srf and t simply replaces the knot vector.

t: Vector of knots with length of n.

n: Length of vector t.

Returns: A refined curve of Srf after insertion of all the knots as specified by vector t of length n.

Description: Given a surface - refines it at the given n knots as defined by vector t. If Replace is TRUE, the values in t replaces current knot vector. Returns pointer to refined surface (Note a Bezier surface will be converted into a Bspline surface).

3.2.251 CagdSrfRegionFromSrf (cagd_lib/cagd_aux.c:794)

regions
subdivision

```
CagdSrfStruct *CagdSrfRegionFromSrf(CagdSrfStruct *Srf,
                                    CagdRType t1,
                                    CagdRType t2,
                                    CagdSrfDirType Dir)
```

Srf: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Dir: Direction of region extraction. Either U or V.

Returns: Sub-region extracted from Srf from t1 to t2.

Description: Given a surface - extracts a sub-region within the domain specified by t1 and t2, in the direction Dir.

3.2.252 CagdSrfReverse (cagd_lib/cagd_aux.c:1074)

reverse

```
CagdSrfStruct *CagdSrfReverse(CagdSrfStruct *Srf)
```

Srf: To be reversed.

Returns: Reversed surface of Srf.

Description: Returns a new surface that is the reversed surface of Srf by reversing the control mesh and the knot vector (if Bspline surface) of Srf in the U direction. See also BspKnotReverse.

3.2.253 CagdSrfReverse (cagd_lib/cagd_aux.c:1134)

reverse

```
CagdSrfStruct *CagdSrfReverse2(CagdSrfStruct *Srf)
```

Srf: To be reversed.

Returns: Reversed surface of Srf.

Description: Returns a new surface that is the reversed surface of Srf by flipping the U and the V directions of the surface. See also BspKnotReverse.

3.2.254 CagdSrfSubdivAtParam (cagd_lib/cagd_aux.c:760)

subdivision

```
CagdSrfStruct *CagdSrfSubdivAtParam(CagdSrfStruct *Srf,
                                    CagdRType t,
                                    CagdSrfDirType Dir)
```

Srf: To subdivide at the prescibed parameter value t.

t: The parameter to subdivide the curve Crv at.

Dir: Direction of subdivision. Either U or V.

Returns: A list of the two surfaces resulting from the process of subdivision.

Description: Given a surface - subdivides it into two sub-surfaces at given parametric value t in the given direction Dir. Returns pointer to first surface in a list of two subdivided surfaces.

3.2.255 CagdSrfTangent (cagd_lib/cagd_aux.c:1011)

tangent

```
CagdVecStruct *CagdSrfTangent(CagdSrfStruct *Srf,
                              CagdRType u,
                              CagdRType v,
                              CagdSrfDirType Dir)
```

Srf: To compute unit tangent vector for.

u, v: Location where to evaluate the tangent of Srf.

Dir: Direction of tangent, Either U or V. *

Returns: A pointer to a static vector holding the unit tangent information.

Description: Given a surface Srf and a parameter values u, v, returns the unit tangent vector of Srf in direction Dir.

3.2.256 CagdSrfTransform (cagd_lib/cagd2gen.c:932)

scaling

translation

transformations

```
void CagdSrfTransform(CagdSrfStruct *Srf,
                      CagdRType *Translate,
                      CagdRType Scale)
```

Crv: To be affinely transformed.

Translate: Translation amount.

Scale: Scaling amount.

Description: Applies an affine transform, in place, to given surface Srf as specified by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.

3.2.257 CagdSurfaceRev (cagd_lib/cagdsrev.c:37)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRev(CagdCrvStruct *Crv)
```

Crv: To create surface of revolution around Z with.

Returns: Surface of revolution.

Description: Constructs a surface of revolution around the Z axis of the given profile curve. Resulting surface will be a Bspline surface, while input may be either a Bspline or a Bezier curve.

3.2.258 CagdSurfaceRevPolynomialApprox (cagd_lib/cagdsrev.c:133)

surface of revolution

surface constructors

```
CagdSrfStruct *CagdSurfaceRevPolynomialApprox(CagdCrvStruct *Crv)
```

Crv: To approximate a surface of revolution around Z with.

Returns: Surface of revolution approximation.

Description: Constructs a surface of revolution around the Z axis of the given profile curve. Resulting surface will be a Bspline surface, while input may be either a Bspline or a Bezier curve. Resulting surface will be a polynomial Bspline surface, approximating a surface of revolution using a polynomial circle approx. (See Faux & Pratt "Computational Geometry for Design and Manufacturing").

3.2.259 CagdSweepAxisRefine (cagd_lib/cagdswep.c:464)

sweep

refinement

```
CagdCrvStruct *CagdSweepAxisRefine(CagdCrvStruct *Axis,
                                   CagdCrvStruct *ScalingCrv,
                                   int RefLevel)
```

Axis: Axis to be used in future sweep operation with the associated ScalingCrv.

ScalingCrv: To use as an estimate on refinement to apply to Axis.

RefLevel: Some refinement control. Keep it low like 2 or 3.

Returns: Refined Axis curve.

Description: Routine to refine the axis curve, according to the scaling curve to better approximate the requested sweep operation.

3.2.260 CagdSweepSrf (cagd_lib/cagdswep.c:68)

sweep

surface constructors

```
CagdSrfStruct *CagdSweepSrf(CagdCrvStruct *CrossSection,
                            CagdCrvStruct *Axis,
                            CagdCrvStruct *ScalingCrv,
                            CagdRType Scale,
                            VoidPtr Frame,
                            CagdBType FrameIsCrv)
```

CrossSection: Of the constructed sweep surface.

Axis: Of the constructed sweep surface.

ScalingCrv: Optional scale or profiel curve.

Scale: if no Scaling Crv, Scale is used to apply a fixed scale on the CrossSection curve.

Frame: An optional vector or a curve to specified the binormal orientation. Otherwise Frame must be NULL.

FrameIsCrv: If TRUE Frame is a curve, if FALSE a vector (if Frame is not NULL).

Returns: Constructed sweep surface.

Description: Constructs a sweep surface using the following curves:

1. CrossSection - defines the basic cross section of the sweep. Must be in the XY plane.
2. Axis - a 3D curve the CrossSection will be swept along such that the Axis normal aligns with the Y axis of the cross section. If Axis is linear (i.e. no normal), the normal is picked randomly or to fit the non linear part of the Axis (if any).
3. Scale - a scaling curve for the sweep, If NULL a scale of Scale is used.
4. Frame - a curve or a vector that specifies the orientation of the sweep by specifying the axes curve's binormal. If Frame is a vector, it is a constant binormal. If Frame is a curve (FrameIsCrv = TRUE), it is assumed to be a vector field binormal. If NULL, it is computed from the Axis curve's pseudo Frenet frame, that minimizes rotation.

This operation is only an approximation. See CagdSweepAxisRefine for a tool to refine the Axis curve and improve accuracy.

3.2.261 CagdTransform (cagd_lib/cagd2gen.c:983)

scaling

translation

transformations

```
void CagdTransform(CagdRType **Points,
                   int Len,
                   int MaxCoord,
                   CagdBType IsNotRational,
                   CagdRType *Translate,
                   CagdRType Scale)
```

Points: To be affinely transformed. Array of vectors.

Len: Of vectors of Points.

MaxCoord: Maximum number of coordinates to be found in Points.

IsNotRational: Do we have weights as vector Points[0]?

Translate: Translation amount.

Scale: Scaling amount.

Description: Applies an affine transform, in place, to given set of points Points which as array of vectors, each vector of length Len. Array Points optionally contains (if !IsNotRational) in Points[0] the weights coefficients and in Points[i] the coefficients of axis i, up to and include MaxCoord (X = 1, Y = 2, etc.). Points are translated and scaled as prescribed by Translate and Scale. Each control point is first translated by Translate and then scaled by Scale.

3.2.262 CagdUVArrayFree (cagd_lib/cagd2gen.c:335)

free

```
void CagdUVArrayFree(CagdUVStruct *UVArray, int Size)
```

UVArray: To be deallocated.

Size: Of the deallocated array.

Description: Deallocates and frees an array of UV structure.

3.2.263 CagdUVArrayNew (cagd_lib/cagd1gen.c:217)

allocation

```
CagdUVStruct *CagdUVArrayNew(int Size)
```

Size: Size of UV array to allocate.

Returns: An array of UV structures of size Size.

Description: Allocates and resets all slots of an array of UV structures.

3.2.264 CagdUVCopy (cagd_lib/cagd1gen.c:743)

copy

```
CagdUVStruct *CagdUVCopy(CagdUVStruct *UV)
```

UV: To be copied.

Returns: A duplicate of UV.

Description: Allocates and copies all slots of a UV structure.

3.2.265 CagdUVCopyList (cagd_lib/cagd1gen.c:1005)

copy

```
CagdUVStruct *CagdUVCopyList(CagdUVStruct *UVList)
```

UVList: To be copied.

Returns: A duplicated list of UV's.

Description: Allocates and copies a list of UV structures.

3.2.266 CagdUVFree (cagd_lib/cagd2gen.c:288)

free

```
void CagdUVFree(CagdUVStruct *UV)
```

UV: To be deallocated.

Description: Deallocates and frees all slots of a UV structure.

3.2.267 CagdUVFreeList (cagd_lib/cagd2gen.c:310)

free

```
void CagdUVFreeList(CagdUVStruct *UVList)
```

UVList: To be deallocated.

Description: Deallocates and frees a UV structure list:

3.2.268 CagdUVNew (cagd_lib/cagd1gen.c:244)

allocation

```
CagdUVStruct *CagdUVNew(void)
```

Returns: A UV structure.

Description: Allocates and resets all slots of a UV structure.

3.2.269 CagdVecArrayFree (cagd_lib/cagd2gen.c:545)

free

```
void CagdVecArrayFree(CagdVecStruct *VecArray, int Size)
```

VecArray: To be deallocated.

Size: Of the deallocated array.

Description: Deallocates and frees an array of vector structure.

3.2.270 CagdVecArrayNew (cagd_lib/cagd1gen.c:373)

allocation

```
CagdVecStruct *CagdVecArrayNew(int Size)
```

Size: Size of Vec array to allocate.

Returns: An array of Vec structures of size Size.

Description: Allocates and resets all slots of an array of Vec structures.

3.2.271 CagdVecCopy (cagd_lib/cagd1gen.c:818)

copy

```
CagdVecStruct *CagdVecCopy(CagdVecStruct *Vec)
```

Vec: To be copied.

Returns: A duplicate of Vec.

Description: Allocates and copies all slots of a Vec structure.

3.2.272 CagdVecCopyList (cagd_lib/cagd2gen.c:29)

copy

```
CagdVecStruct *CagdVecCopyList(CagdVecStruct *VecList)
```

VecList: To be copied.

Returns: A duplicated list of vectors.

Description: Allocates and copies a list of vector structures.

3.2.273 CagdVecFree (cagd_lib/cagd2gen.c:498)

free

```
void CagdVecFree(CagdVecStruct *Vec)
```

Vec: To be deallocated.

Description: Deallocates and frees all slots of a vector structure.

3.2.274 CagdVecFreeList (cagd_lib/cagd2gen.c:520)

free

```
void CagdVecFreeList(CagdVecStruct *VecList)
```

VecList: To be deallocated.

Description: Deallocates and frees a vector structure list:

3.2.275 CagdVecNew (cagd_lib/cagd1gen.c:400)

allocation

```
CagdVecStruct *CagdVecNew(void)
```

Returns: A Vec structure.

Description: Allocates and resets all slots of a Vec structure.

3.2.276 CnvrtBezier2BsplineCrv (cagd_lib/cbzr_aux.c:521)

conversion

```
CagdCrvStruct *CnvrtBspline2BezierCrv(CagdCrvStruct *Crv)
```

Crv: A Bspline curve to convert to a Bezier curve.

Returns: A list of Bezier curves representing the Bspline curve Crv.

Description: Converts a Bspline curve into a set of Bezier curves by subdividing the Bspline curve at all its internal knots. Returned is a list of Bezier curves.

3.2.277 CnvrtBezier2BsplineCrv (cagd_lib/cbzr_aux.c:487)

conversion

```
CagdCrvStruct *CnvrtBezier2BsplineCrv(CagdCrvStruct *Crv)
```

Crv: A Bezier curve to convert to a Bspline curve.

Returns: A Bspline curve representing Bezier curve Crv.

Description: Converts a Bezier curve into Bspline curve by adding an open knot vector.

3.2.278 CnvrtBezier2BsplineSrf (cagd_lib/sbzr_aux.c:340)

conversion

```
CagdSrfStruct *CnvrtBezier2BsplineSrf(CagdSrfStruct *Srf)
```

Srf: Bezier surface to convert to a Bspline surface.

Returns: A Bspline surface representing same geometry as Srf.

Description: Converts a bezier surface into a Bspline surface by adding open end knot vector with no interior knots.

3.2.279 CnvrtBezier2BsplineSrf (cagd_lib/sbzr_aux.c:377)

conversion

```
CagdSrfStruct *CnvrtBspline2BezierSrf(CagdSrfStruct *Srf)
```

Srf: Bspline surface to convert to a Bezier surface.

Returns: A list of Bezier surfaces representing same geometry as Srf.

Description: Convert a Bspline surface into a set of Bezier surfaces by subdiving the Bspline surface at all its internal knots. Returned is a list of Bezier surface.

3.2.280 CnvrtBezier2PowerCrv (cagd_lib/cbzr_pwr.c:52)

power basis

conversion

```
CagdCrvStruct *CnvrtBezier2PowerCrv(CagdCrvStruct *Crv)
```

Crv: To convert into Power basis function representation.

Returns: Same geometry, but in the Power basis.

Description: Converts the given curve from Bezier basis functions to a Power basis functions. Using:

```
        n
        --
 n      \     j-i n   j   j
B (t) = /  (-1)  ( ) ( ) t
 i      --        j   i
       j=i
                                             n-i
Which can be derived by expanding the (1-t)    term in bezier basis
function definition as:



         n-i
          --
     n-i  \  n-i      j
(1-t)   = / (   ) (-t)        using binomial expansion.
          --  j
         j=0
```

This routine simply take the weight of each Bezier basis function B(t) and spread it into the different power basis t^j function scaled by:

```
      j-i n   j
  (-1)   ( ) ( )
          j   i
```

3.2.281 CnvrtBezier2PowerSrf (cagd_lib/cbzr_pwr.c:179)

power basis

conversion

```
CagdSrfStruct *CnvrtBezier2PowerSrf(CagdSrfStruct *Srf)
```

Srf: To convert into Power basis function representation.

Returns: Same geometry, but in the Power basis.

Description: Power to Bezier conversion from surfaces.

3.2.282 CnvrtFloat2OpenSrf (cagd_lib/sbsp_aux.c:1253)

conversion

```
CagdSrfStruct *CnvrtFloat2OpenSrf(CagdSrfStruct *Srf)
```

Srf: Bspline surface to convert to open end conditions.

Returns: A Bspline surface with open end conditions, representing the same geometry as Srf.

Description: Converts a Bspline surface to a Bspline surface with open end conditions.

3.2.283 CnvrtPeriodic2FloatCrv (cagd_lib/cbsp_aux.c:806)

conversion

```
CagdCrvStruct *CnvrtPeriodic2FloatCrv(CagdCrvStruct *Crv)
```

Crv: Bspline curve to convert to floating end conditions. Assume Crv is either periodic or has floating end condition.

Returns: A Bspline curve with floating end conditions, representing the same geometry as Crv.

Description: Converts a Bspline curve to a Bspline curve with floating end conditions.

3.2.284 CnvrtPeriodic2FloatCrv (cagd_lib/cbsp_aux.c:853)

conversion

```
CagdCrvStruct *CnvrtFloat2OpenCrv(CagdCrvStruct *Crv)
```

Crv: Bspline curve to convert to open end conditions.

Returns: A Bspline curve with open end conditions, representing the same geometry as Crv.

Description: Converts a Bspline curve to a Bspline curve with open end conditions.

3.2.285 CnvrtPeriodic2FloatSrf (cagd_lib/sbsp_aux.c:1185)

conversion

```
CagdSrfStruct *CnvrtPeriodic2FloatSrf(CagdSrfStruct *Srf)
```

Srf: Bspline surface to convert to floating end conditions. Assume Crv is either periodic or has floating end condition.

Returns: A Bspline surface with floating end conditions, representing the same geometry as Srf.

Description: Converts a Bspline surface into a Bspline surface with floating end conditions.

3.2.286 CnvrtPolyline2LinBsplineCrv (cagd_lib/cbsp_aux.c:769)

linear curves

conversion

```
CagdCrvStruct *CnvrtPolyline2LinBsplineCrv(CagdPolylineStruct *Poly)
```

Poly: To convert to a linear bspline curve.

Returns: A linear Bspline curve representing Poly.

Description: Returns a new linear Bspline curve constructed from the given polyline.

3.2.287 CnvrtPower2BezierCrv (cagd_lib/cbzr_pwr.c:110)

power basis

conversion

```
CagdCrvStruct *CnvrtPower2BezierCrv(CagdCrvStruct *Crv)
```

Crv: To convert to Bezier basis functions.

Returns: Same geometry, in the Bezier basis functions.

Description: Converts the given curve from Power basis functions to Bezier basis functions. Using:

```
    n    j
    __  ( )
 i  \    i    n
t = /  ----- B (t)
    --   n    j
    j=i ( )
         i
```

This routine simply take the weight of each Power basis function t^i and spread it into the different basis basis function B(t) scaled by:

```
   j   / n
  ( ) / ( )
   i /   i
```

3.2.288 CnvrtPower2BezierSrf (cagd_lib/cbzr_pwr.c:198)

power basis

conversion

```
CagdSrfStruct *CnvrtPower2BezierSrf(CagdSrfStruct *Srf)
```

Srf: To convert into Bezier basis function representation.

Returns: Same geometry, but in the Bezier basis.

Description: Bezier to Power conversion from surfaces.

Chapter 4

Geometry Library, geom_lib

4.1 General Information

This library handles general geometric queries such as a distance between two lines, bounding boxes, convexity of polygons etc. Several header files can be found for this library:

Header (include/*.h)	Functionality
bbox.h	Bounding boxes related functions
convex.h	Convexity of polygons and convex polygon decomposition
geomat3d.h	General three dimensional geometry queries
geomvals.h	Area, Volume, and Size queries on polygonal objects
intrnrml.h	Internal normal interpolation for polygonal models
ln_sweep.h	A line sweep implemetation
poly_cln.h	Clean up degenerated polygons/polylines
primitiv.h	Constructors of polygonal BOXes, CYLINDERs, etc.

4.2 Library Functions

4.2.1 AnimDoAnimation (geom_lib/animate.c:386)

animation

```
void AnimDoAnimation(AnimationStruct *Anim, IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Description: Routine to run a sequence of objects through an animation according to animation attributes of matrices and curves that are attached to them.

4.2.2 AnimDoSingleStep (geom_lib/animate.c:456)

animation

```
void AnimDoSingleStep(AnimationStruct *Anim, IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Description: Routine to exectue a single step the animation, at current time.

4.2.3 AnimFindAnimationTime (geom_lib/animate.c:224)

animation

```
void AnimFindAnimationTime(AnimationStruct *Anim, IPObjectStruct *PObjs)
```

Anim: Animation structure to update.

PObjs: Objects to scan for animation attributes.

Description: Computes the time span for which the animation executes.

4.2.4 AnimGenAnimInfoText (geom_lib/animate.c:73)

animation

```
void AnimGetAnimInfoText(AnimationStruct *Anim)
```

Anim: The animation state to update.

Description: Getting input parameters of animation from user using textual user interface.

4.2.5 AnimResetAnimStruct (geom_lib/animate.c:42)

animation

```
void AnimResetAnimStruct(AnimationStruct *Anim)
```

Anim: The animation state to reset.

Description: Resets the slots of an animation structure.

4.2.6 AnimSaveIterationsToFiles (geom_lib/animate.c:484)

animation

```
void AnimSaveIterationsToFiles(AnimationStruct *Anim, IPObjectStruct *PObjs)
```

Anim: Animation structure.

PObjs: Objects to render.

Description: Saves one iteration of the animation sequence as IRIT data (*.dat). The objects that are saved are those that are visibled on the current time frame as set via current animation_mat attribute.

4.2.7 BBComputeBboxObject (geom_lib/bbox.c:39)

bounding box

```
BBBboxStruct *BBComputeBboxObject(IPObjectStruct *PObj)
```

PObj: To compute a bounding box for.

Returns: A pointer to a statically allocated bounding box holding bounding box information on PObj.

Description: Computes a bounding box of a given object of any type.

4.2.8 BBComputeBboxObjectList (geom_lib/bbox.c:100)

bounding box

```
BBBboxStruct *BBComputeBboxObjectList(IPObjectStruct *PObj)
```

PObj: To compute a bounding box for.

Returns: A pointer to a statically allocated bounding box holding bounding box information on objects PObj.

Description: Computes a bounding box of a list of objects of any type.

4.2.9 BBComputeOnePolyBbox (geom_lib/bbox.c:128)

bounding box

```
BBBboxStruct *BBComputeOnePolyBbox(IPPolygonStruct *PPoly)
```

PPoly: To compute a bounding box for.

Returns: A pointer to a statically allocated bounding box holding bounding box information on PPoly.

Description: Computes a bounding box of a polygon/polyline/pointlist object.

4.2.10 BBComputeOnePolyListBbox (geom_lib/bbox.c:160)

bounding box

```
BBBboxStruct *BBComputePolyListBbox(IPPolygonStruct *PPoly)
```

PPoly: To compute a bounding box for.

Returns: A pointer to a statically allocated bounding box holding bounding box information on PPoly list.

Description: Computes a bounding box for a list of polygon/polyline/pointlist objects.

4.2.11 BBComputePointBbox (geom_lib/bbox.c:195)

bounding box

```
BBBboxStruct *BBComputePointBbox(RealType *Pt)
```

Pt: To compute a bounding box for.

Returns: A pointer to a statically allocated bounding box holding bounding box information on Pt.

Description: Computes a bounding box of a point object.

4.2.12 BBMergeBbox (geom_lib/bbox.c:221)

bounding box

```
BBBboxStruct *BBMergeBbox(BBBboxStruct *Bbox1, BBBboxStruct *Bbox2)
```

Bbox1: First bounding box to union up.

Bbox2: Second bounding box to union up.

Returns: A unioned bounding box the contains both BBox1 and BBox2.

Description: Merges (union) given two bounding boxes into one. Either Bbox1 or Bbox2 can be pointing to the static area Bbox used herein.

4.2.13 CG2PointsFromLineLine (geom_lib/geomat3d.c:776)

line line distance

```
int CG2PointsFromLineLine(PointType Pl1,
                          PointType Vl1,
                          PointType Pl2,
                          PointType Vl2,
                          PointType Pt1,
                          RealType *t1,
                          PointType Pt2,
                          RealType *t2)
```

Pl1, Vl1: Position and direction defining the first line.

Pl2, Vl2: Position and direction defining the second line.

Pt1: Point on Pt1 that is closest to line 2.

t1: Parameter value of Pt1 as (Pl1 + Vl1 * t1).

Pt2: Point on Pt2 that is closest to line 1.

t2: Parameter value of Pt2 as (Pl2 + Vl2 * t2).

Returns: TRUE, if successful.

Description: Routine to find the two points Pti on the lines (Pli, Vli) , i = 1, 2 with the minimal Euclidian distance between them. In other words, the distance between Pt1 and Pt2 is defined as distance between the two lines. The two points are calculated using the fact that if V = (Vl1 cross Vl2) then these two points are the intersection point between the following: Point 1 - a plane (defined by V and line1) and the line line2. Point 2 - a plane (defined by V and line2) and the line line1. This function returns TRUE iff the two lines are not parallel!

4.2.14 CGDistLineLine (geom_lib/geomat3d.c:834)

line line distance

```
RealType CGDistLineLine(PointType Pl1,
                        PointType Vl1,
                        PointType Pl2,
                        PointType Vl2)
```

Pl1, Vl1: Position and direction defining the first line.

Pl2, Vl2: Position and direction defining the second line.

Returns: Distance between the two lines.

Description: Routine to find the distance between two lines (Pli, Vli) , i = 1, 2.

4.2.15 CGDistPointLine (geom_lib/geomat3d.c:577)

point line distance

```
RealType CGDistPointLine(PointType Point, PointType Pl, PointType Vl)
```

Point: To find the distance to on the line.

Pl, Vl: Position and direction that defines the line.

Returns: The computed distance.

Description: Routine to compute the disstance between a 3d point and a 3d line. The line is prescribed using a point on it (Pl) and vector (Vl).

4.2.16 CGDistPointPlane (geom_lib/geomat3d.c:612)

point plane distance

```
RealType CGDistPointPlane(PointType Point, PlaneType Plane)
```

Point: To find the distance to on the plane.

Plane: To find the distance to on the point.

Returns: The computed distance.

Description: Routine to compute the distance between a Point and a Plane. The Plane is prescribed using its four coefficients : Ax + By + Cz + D = 0 given as four elements vector.

4.2.17 CGDistPointPoint (geom_lib/geomat3d.c:452)

point point distance

```
RealType CGDistPointPoint(PointType P1, PointType P2)
```

P1, P2: Two points to compute the distance between.

Returns: Computed distance.

Description: Routine to compute the distance between two 3d points.

4.2.18 CGGenTransMatrixZ2Dir (geom_lib/geomat3d.c:1075)

transformations

```
void CGGenTransMatrixZ2Dir(MatrixType Mat,
                           VectorType Trans,
                           VectorType Dir,
                           RealType Scale)
```

Mat: To place the computed transformation.

Trans: Translation factor.

Dir: Direction to take Z axis to.

Scale: Scaling factor.

Description: Routine to prepar a transformation martix to do the following (in this order): scale by Scale, rotate such that the Z axis is in direction Dir and then translate by Trans. Algorithm: given the Trans vector, it forms the 4th line of Mat. Dir is used to form the second line (the first 3 lines set the rotation), and finally Scale is used to scale first 3 lines/columns to the needed scale:

```
                | Tx  Ty  Tz  0 |   A transformation which takes the coord
                | Bx  By  Bz  0 |  system into T, N & B as required and
[X  Y  Z  1] *  | Nx  Ny  Nz  0 |  then translate it to C. T, N, B are
                | Cx  Cy  Cz  1 |  scaled by Scale.
```

N is exactly Dir (unit vec) but we got freedom on T & B which must be on a plane perpendicular to N and perpendicular between them but thats all! T is therefore selected using this (heuristic ?) algorithm: Let P be the axis of which the absolute N coefficient is the smallest. Let B be (N cross P) and T be (B cross N).

4.2.19 CGGenTransMatrixZ2Dir2 (geom_lib/geomat3d.c:1138)

transformations

```
void CGGenTransMatrixZ2Dir2(MatrixType Mat,
                            VectorType Trans,
                            VectorType Dir,
                            VectorType Dir2,
                            RealType Scale)
```

Mat: To place the computed transformation.

Trans: Translation factor.

Dir: Direction to take Z axis to.

Dir2: Direction to take X axis to.

Scale: Scaling factor.

Description: Routine to prepar a transformation martix to do the following (in this order): scale by Scale, rotate such that the Z axis is in direction Dir and X axis is direction T and then translate by Trans. Algorithm: given the Trans vector, it forms the 4th line of Mat. Dir is used to form the second line (the first 3 lines set the rotation), and finally Scale is used to scale first 3 lines/columns to the needed scale:

```
                | Tx  Ty  Tz  0 |   A transformation which takes the coord
                | Bx  By  Bz  0 |  system into T, N & B as required and
[X  Y  Z  1] *  | Nx  Ny  Nz  0 |  then translate it to C. T, N, B are
                | Cx  Cy  Cz  1 |  scaled by Scale.
```

N is exactly Dir (unit vec) and T is exactly Dir2.

4.2.20 CGPlaneFrom3Points (geom_lib/geomat3d.c:487)

plane

```
int CGPlaneFrom3Points(PlaneType Plane,
                       PointType Pt1,
                       PointType Pt2,
                       PointType Pt3)
```

Plane: To compute.

Pt1, Pt2, Pt3: Three points to fit a plane through.

Returns: TRUE if successful, FALSE otherwise.

Description: Routine to construct the plane from given 3 points. If two of the points are the same it returns FALSE, otherwise (successful) returns TRUE.

4.2.21 CGPointFromLinePlane (geom_lib/geomat3d.c:657)

line plane intersection

```
int CGPointFromLinePlane(PointType Pl,
                         PointType Vl,
                         PlaneType Plane,
                         PointType InterPoint,
                         RealType *t)
```

Pl, Vl: Position and direction that defines the line.

Plane: To find the intersection with the line.

InterPoint: Where the intersection occured.

t: Parameter along the line of the intersection location (as Pl + Vl * t).

Returns: TRUE, if successful.

Description: Routine to find the intersection point of a line and a plane (if any). The Plane is prescribed using four coefficients : Ax + By + Cz + D = 0 given as four elements vector. The line is define via a point on it Pl and a direction vector Vl. Returns TRUE only if such point exists.

4.2.22 CGPointFromLinePlane01 (geom_lib/geomat3d.c:711)

line plane intersection

```
int CGPointFromLinePlane01(PointType Pl,
                           PointType Vl,
                           PlaneType Plane,
                           PointType InterPoint,
                           RealType *t)
```

Pl, Vl: Position and direction that defines the line.

Plane: To find the intersection with the line.

InterPoint: Where the intersection occured.

t: Parameter along the line of the intersection location (as Pl + Vl * t).

Returns: TRUE, if successful and t between zero and one.

Description: Routine to find the intersection point of a line and a plane (if any). The Plane is prescribed using four coefficients : Ax + By + Cz + D = 0 given as four elements vector. The line is define via a point on it Pl and a direction vector Vl. Returns TRUE only if such point exists. this routine accepts solutions only for t between zero and one.

4.2.23 CGPointFromPointLine (geom_lib/geomat3d.c:533)

point line distance

```
void CGPointFromPointLine(PointType Point,
                          PointType Pl,
                          PointType Vl,
                          PointType ClosestPoint)
```

Point: To find the closest to on the line.

Pl, Vl: Position and direction that defines the line.

ClosestPoint: Where closest point found on the line is to be saved.

Description: Routine to compute the closest point on a given 3d line to a given 3d point. the line is prescribed using a point on it (Pl) and vector (Vl).

4.2.24 CGPolygonRayInter (geom_lib/geomat3d.c:900)

ray polygon intersection

Jordan theorem

```
int CGPolygonRayInter(IPPolygonStruct *Pl, PointType PtRay, int RayAxes)
```

Pl: To compute "Jordan Theorem" for the given ray.

PtRay: Origin of ray.

RayAxes: Direction of ray. 0 for X, 1 for Y, etc.

Returns: Number of intersections of ray with the polygon.

Description: Routine that implements "Jordan Theorem": Fire a ray from a given point and find the number of intersections of a ray with the polygon, excluding the given point Pt (start of ray) itself, if on polygon boundary. The ray is fired in +X (Axes == 0) or +Y if (Axes == 1). Only the X/Y coordinates of the polygon are taken into account, i.e. the orthogonal projection of the polygon on an X/Y parallel plane (equal to polygon itself if on X/Y parallel plane...). Note that if the point is on polygon boundary, the ray should not be in its edge direction.
Algorithm:

1. 1.1. Set NumOfIntersection = 0;

   1.2. Find vertex V not on Ray level and set AlgState to its level (below or above the ray level). If none goto 3;
   1.3. Mark VStart = V;
2. Do
   2.1. While State(V) == AlgState do
      2.1.1. V = V -> Pnext;
      2.1.2. If V == VStart goto 3;
   2.2. IntersectionMinX = INFINITY;
   2.3. While State(V) == ON_RAY do
      2.3.1. IntersectionMin = MIN(IntersectionMin, V -> Coord[Axes]);
      2.3.2. V = V -> Pnext;
   2.4. If State(V) != AlgState do
      2.4.1. Find the intersection point between polygon edge Vlast, V and the Ray and update IntersectionMin if lower than it.
      2.4.2. If IntersectionMin is greater than Pt[Axes] increase the NumOfIntersection counter by 1.
   2.5. AlgState = State(V);
   2.6. goto 2.2.
3. Return NumOfIntersection;

4.2.25 CGPolygonRayInter3D (geom_lib/geomat3d.c:1009)

ray polygon intersection

Jordan theorem

```
int CGPolygonRayInter3D(IPPolygonStruct *Pl, PointType PtRay, int RayAxes)
```

Pl: To compute "Jordan Theorem" for the given ray.

PtRay: Origin of ray.

RayAxes: Direction of ray. 0 for X, 1 for Y, etc.

Returns: Number of intersections of ray with the polygon.

Description: Same as CGPolygonRayInter but for arbitrary oriented polygon. The polygon is transformed into the XY plane and then CGPolygonRayInter is invoked on it.

4.2.26 CleanUpPolygonList (geom_lib/poly_cln.c:26)

zero length edges

cleaning

```
void CleanUpPolygonList(IPPolygonStruct **PPolygons)
```

PPolygons: List of polygons to clean, in place.

Description: Routine to clean up polygons - delete zero length edges, and polygons with less than 3 vertices.

4.2.27 CleanUpPolylineList (geom_lib/poly_cln.c:86)

zero length polyline

cleaning

```
void CleanUpPolylineList(IPPolygonStruct **PPolylines)
```

PPolylines: List of polylines to clean, in place.

Description: Routine to clean up polylines of zero length.

4.2.28 Colinear3Vertices (geom_lib/intrnrml.c:118)

colinearity

```
int Colinear3Vertices(IPVertexStruct *V1,
                      IPVertexStruct *V2,
                      IPVertexStruct *V3)
```

V1, V2, V3: Vertices to test for colinearity.

Returns: TRUE if colinear, FALSE otherwise.

Description: Verify the colinearity of the given three vertices.

4.2.29 ConvexPolyObject (geom_lib/convex.c:170)

convexity

convex polygon

```
void ConvexPolyObject(IPObjectStruct *PObj)
```

PObj: To test for convexity of its polygons, and split into convex polygons non convex polygons found, in place. Either a polygonal object or a list of polygonal objects.

Description: Routine to test all polygons in a given object for convexity, and split non convex ones, in place. This function will introduce new vertices to the split polygons.

4.2.30 ConvexPolyObjectN (geom_lib/convex.c:143)

convexity

convex polygon

```
IPObjectStruct *ConvexPolyObjectN(IPObjectStruct *PObj)
```

PObj: To test for convexity of its polygons.

Returns: A duplocate of PObj, but with convex polygons only.

Description: Routine to test all polygons in a given object for convexity, and split non convex ones, non destructively - the original object is not modified. This function will introduce new vertices to the split polygons.

4.2.31 ConvexPolygon (geom_lib/convex.c:244)

convexity

convex polygon

```
int ConvexPolygon(IPPolygonStruct *Pl)
```

Pl: To test its convexity condition.

Returns: TRUE if convex, FALSE otherwise.

Description: Routine to test if the given polygon is convex or not. Algorithm: The polygon is convex iff the normals generated from cross products of two consecutive edges points to the same direction. Note a 5 star polygon satisfies this constraint but it is self intersectingand we assume given polygon is not selft intersecting. The computed direction is alos verified against the polygon's plane normal. The routine returns TRUE iff the polygon is convex. In addition the polygon CONVEX tag (see IPPolygonStruct) is also updated. If the polygon is already marked as convex, nothing is tested!

4.2.32 GMColinear3Pts (geom_lib/geomat3d.c:122)

colinearity

```
int GMColinear3Pts(PointType Pt1, PointType Pt2, PointType Pt3)
```

Pt1, Pt2, Pt3: Three points to verify for colinearity.

Returns: TRUE if colinear, FALSE otherwise.

Description: Verifys the colinearity of three points.

4.2.33 GMGenMatObjectRotX (geom_lib/geomat3d.c:168)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectRotX(RealType *Degree)
```

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the X axis in Degree degrees:

4.2.34 GMGenMatObjectRotY (geom_lib/geomat3d.c:190)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectRotY(RealType *Degree)
```

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the Y axis in Degree degrees:

4.2.35 GMGenMatObjectRotZ (geom_lib/geomat3d.c:212)

rotation

transformations

```
IPObjectStruct *GMGenMatObjectRotZ(RealType *Degree)
```

Degree: Amount of rotation, in degrees.

Returns: A matrix object.

Description: Routine to generate rotation object around the Z axis in Degree degrees:

4.2.36 GMGenMatObjectScale (geom_lib/geomat3d.c:257)

scaling

transformations

```
IPObjectStruct *GMGenMatObjectScale(VectorType Vec)
```

Vec: Amount of scaling, in X, Y, and Z.

Returns: A matrix object.

Description: Routine to generate a scaling object.

4.2.37 GMGenMatObjectTrans (geom_lib/geomat3d.c:234)

translation

transformations

```
IPObjectStruct *GMGenMatObjectTrans(VectorType Vec)
```

Vec: Amount of translation, in X, Y, and Z.

Returns: A matrix object.

Description: Routine to generate a translation object.

4.2.38 GMTransformObject (geom_lib/geomat3d.c:281)

transformations

```
IPObjectStruct *GMTransformObject(IPObjectStruct *PObj, MatrixType Mat)
```

PObj: Object to be transformed.

Mat: Transformation matrix.

Returns: Transformed object.

Description: Routine to transform an object according to the transformation matrix.

4.2.39 GMTransformObjectList (geom_lib/geomat3d.c:421)

transformations

```
IPObjectStruct *GMTransformObjectList(IPObjectStruct *PObj, MatrixType Mat)
```

PObj: Object list to transform.

Mat: Transformation matrix.

Returns: Transformed object list.

Description: Routine to transform an list of objects according to a transformation matrix.

4.2.40 GMVecCopy (geom_lib/geomat3d.c:33)

copy

```
void GMVecCopy(VectorType Vdst, VectorType Vsrc)
```

Vdst: Destination vector.

Vsrc: Source vector.

Description: Routine to copy one vector to another:

4.2.41 GMVecCrossProd (geom_lib/geomat3d.c:98)

cross prod

```
void GMVecCrossProd(VectorType Vres, VectorType V1, VectorType V2)
```

Vres: Result of cross product

V1, V2: Two vectors of the cross product.

Description: Routine to compute the cross product of two vectors. Note Vres may be the same as V1 or V2.

4.2.42 GMVecDotProd (geom_lib/geomat3d.c:150)

dot product

```
RealType GMVecDotProd(VectorType V1, VectorType V2)
```

V1, V2: Two vector to compute dot product of.

Returns: Resulting dot product.

Description: Routine to compute the dot product of two vectors.

4.2.43 GMVecLength (geom_lib/geomat3d.c:78)

magnitude

```
RealType GMVecLength(VectorType V)
```

V: To compute its magnitude.

Returns: Magnitude of V.

Description: Routine to compute the magnitude (length) of a given 3D vector:

4.2.44 GMVecNormalize (geom_lib/geomat3d.c:51)

normalize

```
void GMVecNormalize(VectorType V)
```

V: To normalize.

Description: Routine to normalize the vector length to a unit size.

4.2.45 GenRotateMatrix (geom_lib/convex.c:100)

transformations

```
void GenRotateMatrix(MatrixType Mat, VectorType Dir)
```

Mat: To place the constructed homogeneous transformation.

Dir: To derive a transformation such that Dir goes to Z axis.

Description: Routine to prepare a transformation martix to rotate such that Dir is parallel to the Z axes. Used by the convex decomposition to rotate the polygons to be XY plane parallel. Algorithm: form a 4 by 4 matrix from Dir as follows:

```
                | Tx  Ty  Tz  0 |   A transformation which takes the coord
                | Bx  By  Bz  0 |  system into T, N & B as required.
 [X  Y  Z  1] * | Nx  Ny  Nz  0 |
                | 0   0   0   1 |
```

N is exactly Dir, but we got freedom on T & B which must be on a plane perpendicular to N and perpendicular between them but thats all! T is therefore selected using this (heuristic ?) algorithm: Let P be the axis of which the absolute N coefficient is the smallest. Let B be (N cross P) and T be (B cross N).

4.2.46 InterpNrmlBetweenTwo (geom_lib/intrnrml.c:160)

normals

```
void InterpNrmlBetweenTwo(IPVertexStruct *V,
                          IPVertexStruct *V1,
                          IPVertexStruct *V2)
```

V: Vertex that its normal is to be updated.

V1, V2: Edge V is assumed to be on so that the two normals of V1 and V2 can be blended to form the normal of V.

Description: Update Normal of the middle vertex V, assumed to be between V1 and V2.

4.2.47 InterpNrmlBetweenTwo2 (geom_lib/intrnrml.c:198)

normals

```
void InterpNrmlBetweenTwo2(PointType Pt,
                           VectorType Normal,
                           IPVertexStruct *V1,
                           IPVertexStruct *V2)
```

Pt: Middle position at which a normal is to be ucomputed.

Normal: Where resulting vector is to be placed.

V1, V2: Edge V is assumed to be on so that the two normals of V1 and V2 can be blended to form the normal of V.

Description: Update normal of the middle vertex V, assumed to be between V1 and V2.

4.2.48 LineSweep (geom_lib/ln_sweep.c:43)

line sweep

line line intersections

```
void LineSweep(LsLineSegStruct **Lines)
```

Lines: To compute all intersections against each other, in the plane.

Description: Computes all intersections between all given lines, in the plane. The Lines segments are updated so the Inters slot holds the list of intersections with the other segments, NULL if none. Returned is a list with possibly a different order than that is given.

4.2.49 PolyCountPolys (geom_lib/geomvals.c:303)

```
double PolyCountPolys(IPObjectStruct *PObj)
```

PObj: To count number of polygons in.

Returns: Number of polygons, an integer value returned as double.

Description: Routine to count the number of polygons in the given geometric object.

4.2.50 PolyObjectArea (geom_lib/geomvals.c:49)

area

```
double PolyObjectArea(IPObjectStruct *PObj)
```

PObj: A polyhedra object to compute its surface area.

Returns: The area of object PObj.

Description: Routine to evaluate the Area of the given geom. object, in object unit.

```
Algorithm (for each polygon):                           V3
1. Set Polygon Area to be zero.                        /\
   Make a copy of the original polygon              /    \
   and transform it to a XY parallel plane.       /        \V2
   Find the minimum Y value of the polygon     V4/          |
   in the XY plane.                              \          |
2. Let V(0) be the first vertex, V(n) the last one.   \       |
   For i goes from 0 to n-1 add to Area the area        \_______|
   below edge V(i), V(i+1):                             V0      V1
   PolygonArea += (V(i+1)x - V(i)x) * (V(i+1)y' - V(i)y') / 2
   where V(i)y' is V(i)y - MinY, where MinY is polygon minimum Y value.
3. The result of step 2 is the area of the polygon itself.
   However, it might be negative, so take the absolute result of step 2
   and add it to the global ObjectArea.
```

Note step 2 is performed by another auxiliary routine: PolygonXYArea.

4.2.51 PolyObjectVolume (geom_lib/geomvals.c:171)

volume

```
double PolyObjectVolume(IPObjectStruct *PObj)
```

PObj: To compute volume for.

Returns: Computed volume.

Description: Routine to evaluate the Volume of the given geom object, in object unit. This routine has a side effect that all non-convex polygons will be splitted to convex ones. Algorithm (for each polygon, and let ObjMinY be the minimum OBJECT Y):

```
                                                        V3
1. Set Polygon Area to be zero.                        /\
   Let V(0) be the first vertex, V(n) the last.     /    \
   For i goes from 1 to n-1 form triangles        /        \V2
               by V(0), V(i), V(i+1).          V4/          |
   For each such triangle di:                    \          |
   1.1. Find the vertex (out of V(0), V(i), V(i+1))  \        |
        with the minimum Z - TriMinY.                  \_______|
   1.2. The volume below V(0), V(i), V(i+1) triangle,  V0      V1
        relative to ObjMinZ level, is the sum of:
        1.2.1. volume of V'(0), V'(i), V'(i+1) - the
               area of projection of V(0), V(i), V(i+1) on XY parallel
               plane, times (TriMinZ - ObjMinZ).
        1.2.2. Assume V(0) is the one with the PolyMinZ. Let V"(i) and
               V"(i+1) be the projections of V(i) and V(i+1) on the plane
               Z = PolyZMin. The volume above 1.2.1. and below the polygon
               (triangle!) will be: the area of quadraliteral V(i), V(i+1),
               V"(i+1), V"(i), times distance of V(0) for quadraliteral
               plane divided by 3.
   1.3. If Z component of polygon normal is negative add 1.2. result to
        ObjectVolume, else subtract it.
```

4.2.52 PrimGenBOXObject (geom_lib/primitiv.c:79)

box

primitives

```
IPObjectStruct *PrimGenBOXObject(VectorType Pt,
                                 RealType WidthX,
                                 RealType WidthY,
                                 RealType WidthZ)
```

Pt: Low end corner of BOX.

WidthX: Width of BOX (X axis).

WidthY: Depth of BOX(Y axis).

WidthZ: Height of BOX(Z axis).

Returns: A BOX primitive

Description:

```
  Routine to create a BOX geometric object defined by Pt - the minimum
 3d point, and Width - Dx Dy & Dz vector.            4
   Order of vertices is as                      5       7
 follows in the picture:                        |   6   |
                                                |   |   |
 (Note vertex 0 is hidden behind edge 2-6)      |   |   |
                                                1   |   3
                                                    2
```

All dimensions can be negative, denoting the reversed direction.

4.2.53 PrimGenCONE2Object (geom_lib/primitiv.c:297)

cone

primitives

```
IPObjectStruct *PrimGenCONE2Object(VectorType Pt,
                                   VectorType Dir,
                                   RealType R1,
                                   RealType R2)
```

Pt: Center location of Base of CON2.

Dir: Direction and distance from Pt to center of other base of CON2.

R1, R2: Two base radii of the truncated CON2

Returns: A CON2 Primitive.

Description: Routine to create a truncated CONE, CON2, geometric object defined by Pt - the base 3d center point, Dir - the cone direction and height, and two base radii R1 and R2. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.54 PrimGenCONEObject (geom_lib/primitiv.c:192)

cone

primitives

```
IPObjectStruct *PrimGenCONEObject(VectorType Pt, VectorType Dir, RealType R)
```

Pt: Center location of Base of CONE.

Dir: Direction and distance from Pt to apex of CONE.

R: Radius of Base of the cone.

Returns: A CONE Primitive.

Description: Routine to create a CONE geometric object defined by Pt - the base 3d center point, Dir - the cone direction and height, and base radius R. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.55 PrimGenCYLINObject (geom_lib/primitiv.c:423)

cylinder

primitives

```
IPObjectStruct *PrimGenCYLINObject(VectorType Pt, VectorType Dir, RealType R)
```

Pt: Center location of Base of CYLINder.

Dir: Direction and distance from Pt to other base of cylinder.

R: Radius of Base of the cylinder.

Returns: A CYLINDER Primitive.

Description: Routine to create a CYLINder geometric object defined by Pt - the base 3d center point, Dir - the cylinder direction and height, and radius R. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.56 PrimGenEXTRUDEObject (geom_lib/primitiv.c:1192)

extrusion surface

primitives

```
IPObjectStruct *PrimGenEXTRUDEObject(IPObjectStruct *Cross, VectorType Dir)
```

Cross: To extrude in direction Dir.

Dir: Direction and magnitude of extrusion.

Returns: An extrusion surface.

Description: Routine to a create an extrusion surface out of the given cross section and the given direction. Input can either be a polygon/line or a freefrom curve object. If input is a polyline/gon, it must never be coplanar with Dir. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.57 PrimGenGBOXObject (geom_lib/primitiv.c:117)

general box

box

primitives

```
IPObjectStruct *PrimGenGBOXObject(VectorType Pt,
                                  VectorType Dir1,
                                  VectorType Dir2,
                                  VectorType Dir3)
```

Pt: Low end corner of GBOX.

Dir1, Dir2, Dir3: Three independent directional vectors to define GBOX.

Returns: A GBOX primitive.

Description: Routine to create a GBOX geometric object defined by Pt - the minimum 3d point, and 3 direction Vectors Dir1, Dir2, Dir3. If two of the direction vectors are parallel the GBOX degenerates to a zero volume object. A NULL pointer is returned in that case.

```
                                                   4
Order of vertices is as                        5       7
follows in the picture:                        |   6   |
                                               |   |   |
(Note vertex 0 is hidden behind edge 2-6)      |   |   |
                                               1   |   3
                                                   2
```

4.2.58 PrimGenObjectFromPolyList (geom_lib/primitiv.c:973)

primitives

```
IPObjectStruct *PrimGenObjectFromPolyList(IPObjectStruct *PObjList)
```

PObjList: List of polygonal objects.

Returns: A single object containing all polygons in all provided objects, by a simple union.

Description: Routine to create an OBJECT directly from set of sspecified polygons. m No test is made for the validity of the model in any sense.

4.2.59 PrimGenPOLYDISKObject (geom_lib/primitiv.c:807)

disk

primitives

```
IPObjectStruct *PrimGenPOLYDISKObject(VectorType N, VectorType T, RealType R)
```

N: Normal to the plane this disk included in.

T: A translation factor of the center of the disk.

R: Radius of teh disk.

Returns: A single polygon object - a disk.

Description: Routine to create a POLYDISK geometric object defined by the normal N and the translation vector T. The object is a planar disk (a circle of GlblResolution points in it...) and its radius is equal to R. The normal direction is assumed to point to the inside of the object. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.60 PrimGenPOLYGONObject (geom_lib/primitiv.c:880)

polygon

polyline

primitives

```
IPObjectStruct *PrimGenPOLYGONObject(IPObjectStruct *PObjList, int IsPolyline)
```

PObjList: List of vertices/points to construct as a polygon/line.

IsPolyline: If TRUE, make a polyline, otherwise a polygon.

Returns: A polygon/line constructed from PObjList.

Description: Routine to create a POLYGON/LINE directly from its specified vertices. The validity of the elements in the provided list is tested to make sure they are vectors or points. No test is made to make sure all vertices are on one plane, and that no two vertices are similar.

4.2.61 PrimGenSPHEREObject (geom_lib/primitiv.c:549)

sphere

primitives

```
IPObjectStruct *PrimGenSPHEREObject(VectorType Center, RealType R)
```

Center: Center location of SPHERE. R Radius of sphere.

Returns: A SPHERE Primitive.

Description: Routine to create a SPHERE geometric object defined by Center, the center of the sphere and R, its radius. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.62 PrimGenSURFREVObject (geom_lib/primitiv.c:1060)

surface of revolution

primitives

```
IPObjectStruct *PrimGenSURFREVObject(IPObjectStruct *Cross)
```

Cross: To rotate around the Z axis forming a surface of revolution.

Returns: A surface of revolution.

Description: Routine to a create surface of revolution by rotating the given cross section along the Z axis. Input can either be a polygon/line or a freefrom curve object. If input is a polyline/gon, it must never be coplanar with the Z axis. See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.63 PrimGenTORUSObject (geom_lib/primitiv.c:701)

torus

primitives

```
IPObjectStruct *PrimGenTORUSObject(VectorType Center,
                                   VectorType Normal,
                                   RealType Rmajor,
                                   RealType Rminor)
```

Center: Center location of the TORUS primitive.

Normal: Normal to the major plane of the torus.

Rmajor: Major radius of torus.

Rminor: Minor radius of torus.

Returns: A TOURS Primitive.

Description: Routine to create a TORUS geometric object defined by Center - torus 3d center point, the main torus plane normal Normal, major radius Rmajor and minor radius Rminor (Tube radius). Teta runs on the major circle, Fee on the minor one. Then

```
X = (Rmajor + Rminor * cos(Fee)) * cos(Teta)
Y = (Rmajor + Rminor * cos(Fee)) * sin(Teta)
Z = Rminor * sin(Fee)
```

See also PrimSetResolution on fineness control of approximation of the primitive using flat faces.

4.2.64 PrimSetResolution (geom_lib/primitiv.c:1516)

primitives

resolution

```
void PrimSetResolution(int Resolution)
```

Resolution: To set as new resolution for all primitve constructors.

Description: Routine to set the polygonal resolution (fineness). Resolution sroutghly the number of edges a circular primitive will have along the entire circle.

4.2.65 SplitNonConvexPoly (geom_lib/convex.c:322)

convexity

convex polygon

```
IPPolygonStruct *SplitNonConvexPoly(IPPolygonStruct *Pl)
```

Pl: Non convex polygon to split into convex ones.

Returns: A list of convex polygons resulting from splitting up Pl.

Description: Routine to split non convex polygon into a list of convex ones.

1. Remove a polygon from GlblList. If non exists stop.
2. Search for non convex corner. If not found stop - polygon is convex. Otherwise let the non convex polygon found be V(i).
3. Fire a ray from V(i) in the opposite direction to V(i-1). Find the closest intersection of the ray with polygon boundary P.
4. Split the polygon into two at V(i)-P edge and push the two new polygons on the GlblList.
5. Goto 1.

4.2.66 UpdateVerticesNormals (geom_lib/intrnrml.c:46)

normals

```
void UpdateVerticesNormals(IPPolygonStruct *PlList,
                           IPPolygonStruct *OriginalPl)
```

PlList: List of polygons to update normal for.

OriginalPl: Original polygons PlList was derived from, probably using Boolean operations.

Description: For each polygon in PlList update any vertex normal which is zero to an interpolated value using the Original polygon vertex list OriginalPl. All the new vertices are enclosed within the original polygon which must be convex as well.

Chapter 5

Miscelleneous Library, misc_lib

5.1 General Information

This library holds general miscelleneous functions such as reading configuration files, low level homogeneous matrices computation, low level attributes and general machine specific low level routines. Several header files can be found for this library:

Header (include/*.h)	Functionality
config.h	Manipulation of configuration files (*.cfg files)
dist_pts.h	Enegry based distribution of points.
gen_mat.h	Homogeneous matrices manipulation
getarg.h	Command line parsing for application tools
imalloc.h	Low level dynamic memory functions for IRIT
miscattr.h	Low level attribute related functions
priorque.h	An implementation of a priority queue
xgeneral.h	Low level, machine specific, routines

5.2 Library Functions

5.2.1 Attr2String (misc_lib/miscattr.c:318)

attributes

```
char *Attr2String(IPAttributeStruct *Attr)
```

Attr: To convert to a string.

Returns: A pointer to a static string representing/describing the given attribute.

Description: Routine to convert an attribute to a string.

5.2.2 AttrCopyAttributes (misc_lib/miscattr.c:519)

attributes

```
IPAttributeStruct *AttrCopyAttributes(IPAttributeStruct *Src)
```

Src: Attribute list to duplicate.

Returns: Duplicated attribute list.

Description: Routine to copy an attribute list.

5.2.3 AttrFindAttribute (misc_lib/miscattr.c:398)

attributes

```
IPAttributeStruct *AttrFindAttribute(IPAttributeStruct *Attrs, char *Name)
```

Attrs: Attribute list to search.

Name: Attribute to search by this name.

Returns: Attribute if found, otherwise NULL.

Description: Routine to search for an attribute by Name.

5.2.4 AttrFreeAttributes (misc_lib/miscattr.c:486)

attributes

```
void AttrFreeAttributes(IPAttributeStruct **Attrs)
```

Attrs: To remove and delete.

Description: Routine to remove anddelete all attributes of the given Attr list.

5.2.5 AttrFreeOneAttribute (misc_lib/miscattr.c:445)

attributes

```
void AttrFreeOneAttribute(IPAttributeStruct **Attrs, char *Name)
```

Attrs: To search for an attribute named Name and remove and delete it.

Name: Name of attribute to remove and delete.

Description: Routine to remove and delete the attribute named Name from the given Attr list.

5.2.6 AttrGetIntAttrib (misc_lib/miscattr.c:61)

attributes

```
int AttrGetIntAttrib(IPAttributeStruct *Attrs, char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute.

5.2.7 AttrGetPtrAttrib (misc_lib/miscattr.c:124)

attributes

```
VoidPtr AttrGetPtrAttrib(IPAttributeStruct *Attrs, char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute.

5.2.8 AttrGetRealAttrib (misc_lib/miscattr.c:182)

attributes

```
RealType AttrGetRealAttrib(IPAttributeStruct *Attrs, char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a RealType attribute.

5.2.9 AttrGetStrAttrib (misc_lib/miscattr.c:251)

attributes

```
char *AttrGetStrAttrib(IPAttributeStruct *Attrs, char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute.

5.2.10 AttrResetAttributes (misc_lib/miscattr.c:379)

attributes

```
void AttrResetAttributes(IPAttributeStruct **Attrs)
```

Attrs: To initialize.

Description: Routine to initialize the attributes of the given Attr list.

5.2.11 AttrSetIntAttrib (misc_lib/miscattr.c:29)

attributes

```
void AttrSetIntAttrib(IPAttributeStruct **Attrs, char *Name, int Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of thely introducedattribute

Data: Ingeter attribute to save.

Description: Routine to set an integer attribute.

5.2.12 AttrSetPtrAttrib (misc_lib/miscattr.c:93)

attributes

```
void AttrSetPtrAttrib(IPAttributeStruct **Attrs, char *Name, VoidPtr Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of thely introducedattribute

Data: Pointer attribute to save.

Description: Routine to set a pointer attribute.

5.2.13 AttrSetRealAttrib (misc_lib/miscattr.c:150)

attributes

```
void AttrSetRealAttrib(IPAttributeStruct **Attrs, char *Name, RealType Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of thely introducedattribute

Data: RealType attribute to save.

Description: Routine to set a RealType attribute.

5.2.14 AttrSetStrAttrib (misc_lib/miscattr.c:219)

attributes

```
void AttrSetStrAttrib(IPAttributeStruct **Attrs, char *Name, char *Data)
```

Attrs: Attribute list where to place new attribute.

Name: Name of thely introducedattribute

Data: String attribute to save.

Description: Routine to set a string attribute.

5.2.15 AttrTraceAttributes (misc_lib/miscattr.c:284)

attributes

```
IPAttributeStruct *AttrTraceAttributes(IPAttributeStruct *TraceAttrs,
                                       IPAttributeStruct *FirstAttrs)
```

TraceAttrs: If not NULL, contains the previously returned attribute.

FirstAttrs: First attribute in list, usually NULL in all but the first invokation in a psuence.

Returns: Next attribute in list.

Description: Routine to aid in scanning a list of attributes. If TraceAttrs != NULL, a ptr to its attribute list is saved and the next attribute is returned every call until the end of the list is reached, in which NULL is returned. FirstAttrs should be NULL in all but the first call in the sequence. Attributes with names starting with an underscore '_' are assumed to be temporary or internal and are skipped.

5.2.16 Config (misc_lib/config.c:113)

configuration

cfg files

```
void Config(char *PrgmName, ConfigStruct *SetUp, int NumVar)
```

PrgmName: Name of program that uses this data base.

SetUp: Configuration data based.

NumVar: Number of entries on configuration data base.

Description: Main routine of configuration file handling. Gets the program name, PrgmName, and the configuration data base that defines the acceptable variables, Setup, with Numvarentries.

5.2.17 ConfigPrint (misc_lib/config.c:59)

configuration

cfg files

```
void ConfigPrint(ConfigStruct *SetUp, int NumVar)
```

SetUp: Configuration data based.

NumVar: Number of entries on configuration data base.

Description: Routine to print the current configuration data structure contents.

5.2.18 DistPoint1DWithEnergy (misc_lib/dist_pts.c:40)

point distribution

```
RealType *DistPoint1DWithEnergy(int N,
                                RealType XMin,
                                RealType XMax,
                                int Resolution,
                                DistEnergy1DFuncType EnergyFunc)
```

N: Number of points to distribute,

XMin: Minimum of domain to distribute points.

XMax: Minimum of domain to distribute points.

Resolution: Fineness of integral calculation.

EnergyFunc: Energy function to use.

Returns: A vector of N points distributed as requested.

Description: Distributes N points with a given energy in the region in the X line that is bounded by XMin, XMax. Energy is specified via the EnergyFunc that recieves the X location. Resolution * N specifies how many samples to take from EnergyFunc. Returns an array of N distributed points. The solution to the distribution is analythic provided EnergyFunc can be integrated. Herein, this integral is computed nomerically.

5.2.19 GAGetArgs (misc_lib/getarg.c:196)

command line arguments

```
#ifdef USE_VARARGS
int GAGetArgs(int va_alist, ...)
```

va_alist: Do "man stdarg".

...: Rest of optional parameters

Returns: TRUE if command line was valid, FALSE otherwise.

Description: Routine to access command line arguments and interpret them, by getting access to the main routine's argc/argv interface and a control string that prescribes the expected options. Returns ARG_OK (0) is case of succesfull parsing, error code else...
Format of CtrlStr format: The control string passed to GAGetArgs controls the way argv (argc) are parsed. Each entry in this string must have no spaces in it. The First Entry is the name of the program which is usually ignored except when GAPrintHowTo is called. All the other entries (except the last one which will be discussed shortly) must have the following format:

1. One letter which sets the option letter (i.e. 'x' for option '-x').
2. '!' or '%' to determines if this option is really optional ('%') or it must be provided by the user ('!').
3. '-' always.

4. Alpha numeric string, usually ignored, but used by GAPrintHowTo to describe the meaning of this option.
5. Sequences that start with either '!' or '%'. Again if '!' then this sequence must exists (only if its option flag is given), and if '%' it is optional. Each sequence will be followed by one or two characters which defines the kind of the input:
   - 5.1. d, x, o, u - integer is expected (decimal, hex, octal base or unsigned).
   - 5.2. D, X, O, U - long integer is expected (same as above).
   - 5.3. f - float number is expected.
   - 5.4. F - double number is expected.
   - 5.5. s - string is expected.
   - 5.6. *? - any number of '?' kind (d, x, o, u, D, X, O, U, f, F, s) will match this one. If '?' is numeric, it scans until none numeric input is given. If '?' is 's' then it scans up to the next option or end of argv.

If the last parameter given in the CtrlStr, is not an option (i.e. the second char is not in ['!', '%'] and the third one is not '-'), all what remained from argv is hooked to it.
The variables passed to GAGetArgs (starting from 4th parameter) MUST match the order of options in the CtrlStr. For each option, an address of an integer must be passed. This integer must initialized by 0. If that option is given in the command line, it will be set to one. Otherwise, this integer will not be affected. In addition, the sequences that might follow an option require the following parameter(s) to be passed

1. d, x, o, u - pointer to an integer (int *).
2. D, X, O, U - pointer to a long (long *).
3. f - pointer to a float (float *).
4. F - pointer to a double (double *).
5. s - pointer to a char * (char **). NO pre-allocation is required.
6. *? - TWO variables are passed for each such wild character request. The first variable is an address of an integer, and it will return the number of parameters actually hooked to this sequence. The second variable is a pointer to a pointer to type ? (? **). It will return an address of a vector of pointers of type ?, terminated with a NULL pointer. NO pre-allocation is required. These two variables behaves very much like the argv/argc pair and are used the "trap" unused command line options.

Examples:

```
    "Example1  i%-OneInteger!d  s%-Strings!*s  j%-  k!-Float!f  Files!*s"
 Will match: Example1 -i 77 -s String1 String2 String3 -k 88.2 File1 File2
   or match: Example1 -s String1 -k 88.3 -i 999 -j
    but not: Example1 -i 77 78 (i expects one integer, k must be specified).
```

The option k must exists in the above example and if '-i' is prescribed one integer argument must follow it. In the first example, File1 & File2, will match Files in the control string. The order of the options in the command line is irrelevant. A call to GAPrintHowTo with this CtrlStr will print to stderr:

```
 Example1 [-i OneIngeter] [-s Strings...] [-j] -k Float Files...
```

The parameters below are stdarg style and in fact are expecting the following:

```
   GAGetArgs(argc, argv, CtrlStr, ...);
```

1. argc, argv: The usual C interface from the main routine of the program.
2. CtrlStr: Defining the types/options to expect in the command line.
3. ...: list of addreses of variables to initialize according to parsed command line.

5.2.20 GAPrintErrMsg (misc_lib/getarg.c:708)

command line arguments

```
void GAPrintErrMsg(int Error)
```

Error: Error type as returned by GAGetArgs.

Description: Routine to print a description of an error to stderr, for this module:

5.2.21 GAPrintHowTo (misc_lib/getarg.c:751)

command line arguments

```
void GAPrintHowTo(char *CtrlStr)
```

CtrlStr: Defining the types/options to expect in the command line.

Description: Routine to print the correct format of command line allowed, to stderr. For example, for the following control string,

```
"Example1  i%-OneInteger!d  s%-Strings!*s  j%-  k!-Float!f  Files"
```

This routine will print

```
Example1 [-i OneIngeter] [-s Strings...] [-j] -k Float Files...
```

5.2.22 IritCPUTime (misc_lib/xgeneral.c:217)

time

```
RealType IritCPUTime(int Reset)
```

Reset: If TRUE, clock is reset back to zero.

Returns: CPU time since last reset or beginning of execution.

Description: Routine to compute the cpu time of the running process.

5.2.23 IritFatalError (misc_lib/irit_ftl.c:24)

error trap

```
void IritFatalError(char *Msg)
```

Msg: Error message to print.

Description: Default trap for IRIT programs for irit fatal errors. This function just prints the given error message and die.

5.2.24 IritFree (misc_lib/imalloc.c:218)

allocation

```
void IritFree(VoidPtr p)
```

p: Pointer to a block that needs to be freed.

Description: Routine to free dynamic memory for all IRIT program/tool/libraries. All requests to free dynamic memory should invoke this function. If the environment variable "IRIT_MALLOC" is set when an IRIT program is executed, the consistency of the dynamic memory is tested on every invokation of this routine. See IritTestAllDynMemory function for more.

5.2.25 IritMalloc (misc_lib/imalloc.c:130)

allocation

```
VoidPtr IritMalloc(unsigned size)
```

size: Size of block to allocate, in bytes.

Returns: A pointer to the allocated block. A function calling this may assume return value wil never be NULL, since no more memory cases are trapped locally.

Description: Routine to allocate dynamic memory for all IRIT program/tool/libraries. All requests for dynamic memory should invoke this function. If the environment variable "IRIT_MALLOC" is set when an IRIT program is executed, the consistency of the dynamic memory is tested on every invokation of this routine. See IritTestAllDynMemory function for more.

5.2.26 IritRandom (misc_lib/xgeneral.c:183)

random numbers

```
RealType IritRandom(RealType Min, RealType Max)
```

Min: Minimum range of random number requested.

Max: Maximum range of random number requested.

Returns: A random number between Min andMax.

Description: Routine to compute a random number in a specified range. See also IritRandomInit.

5.2.27 IritRandomInit (misc_lib/xgeneral.c:155)

random numbers

```
void IritRandomInit(long Seed)
```

Seed: To initialize the random number generator with.

Description: Routine to initialize the random number generator.

5.2.28 IritRealTimeDate (misc_lib/xgeneral.c:265)

date

time

```
char *IritRealTimeDate(void)
```

Returns: A string describing current date and time.

Description: Routine to create and return a string describing current date and time.

5.2.29 IritSleep (misc_lib/xgeneral.c:91)

sleep

```
void IritSleep(int MilliSeconds)
```

MilliSeconds: Sleeping time required, in miliseconds.

Description: Routine to force a process to sleep.

5.2.30 IritStrdup (misc_lib/xgeneral.c:49)

strdup

```
char *IritStrdup(char *s)
```

s: String to duplicate.

Returns: Duplicated string.

Description: Routine to duplicate a string. Exists in some computer environments.

5.2.31 IritTestAllDynMemory (misc_lib/imalloc.c:81)

allocation

```
void IritTestAllDynMemory(int PrintAlloc)
```

PrintAlloc: If TRUE, prints information all all allocated block, not just block with errors.

Description: Tests for error in dynamically allocated memory, without affecting any such allocation or allocated blocks. This routine may be used only if "IRIT_MALLOC" environment variable is set for debugging purposes and it obviously slows down running time. The following tests are being made on every block allocated,and messages are printed to stderr as needed:

1. Overwriting beyond the end of the allocated block.
2. Overwriting below the beginning of the allocated block.
3. Freeing an unallocated pointer.
4. Freeing the same pointer twice.
5. If "IRIT_MALLOC_PTR" environment variable is set to an address, this address is being search for during allocation (IritMalloc) and announced when detected.

5.2.32 IritWarningError (misc_lib/irit_wrn.c:24)

error trap

```
void IritWarningError(char *Msg)
```

Msg: Error message to print.

Description: Default trap for IRIT programs for irit warning errors. This function just prints the given error message.

5.2.33 MatAddTwo4by4 (misc_lib/genmat.c:289)

transformations

matrix addition

```
void MatAddTwo4by4(MatrixType MatRes, MatrixType Mat1, MatrixType Mat2)
```

MatRes: Result of matrix addition.

Mat1, Mat2: The two operand of the matrix addition.

Description: Routine to add 2 4by4 matrices. MatRes may be one of Mat1 or Mat2.

5.2.34 MatGenMatRotX (misc_lib/genmat.c:138)

transformations

rotation

```
void MatGenMatRotX(RealType CosTeta, RealType SinTeta, MatrixType Mat)
```

SinTeta, CosTeta: Amount of rotation, given as sine and cosine of Teta.

Mat: Matrix to initialize as a rotation matrix.

Description: Routine to generate a 4*4 matrix to Rotate around the X axis by Teta, given the sin and cosine of Teta

5.2.35 MatGenMatRotX1 (misc_lib/genmat.c:114)

transformations

rotation

```
void MatGenMatRotX1(RealType Teta, MatrixType Mat)
```

Teta: Amount of rotation, in radians.

Mat: Matrix to initialize as a rotation matrix.

Description: Routine to generate a 4*4 matrix to Rotate around the X axis by Teta radians.

5.2.36 MatGenMatRotY (misc_lib/genmat.c:186)

transformations

rotation

```
void MatGenMatRotY(RealType CosTeta, RealType SinTeta, MatrixType Mat)
```

SinTeta, CosTeta: Amount of rotation, given as sine and cosine of Teta.

Mat: Matrix to initialize as a rotation matrix.

Description: Routine to generate a 4*4 matrix to Rotate around the Y axis by Teta, given the sin and cosine of Teta

5.2.37 MatGenMatRotY1 (misc_lib/genmat.c:162)

transformations

rotation

```
void MatGenMatRotY1(RealType Teta, MatrixType Mat)
```

Teta: Amount of rotation, in radians.

Mat: Matrix to initialize as a rotation matrix.

Description: Routine to generate a 4*4 matrix to Rotate around the Y axis by Teta radians.

5.2.38 MatGenMatRotZ (misc_lib/genmat.c:234)

transformations

rotation

```
void MatGenMatRotZ(RealType CosTeta, RealType SinTeta, MatrixType Mat)
```

SinTeta, CosTeta: Amount of rotation, given as sine and cosine of Teta.

Mat: Matrix to initialize as a rotation matrix.

Description: Routine to generate a 4*4 matrix to Rotate around the Z axis by Teta, given the sin and cosine of Teta

5.2.39 MatGenMatRotZ1 (misc_lib/genmat.c:210)

transformations

rotation

```
void MatGenMatRotZ1(RealType Teta, MatrixType Mat)
```

Teta: Amount of rotation, in radians.

Mat: Matrix to initialize as a rotation matrix.

Description: Routine to generate a 4*4 matrix to Rotate around the Z axis by Teta radians.

5.2.40 MatGenMatScale (misc_lib/genmat.c:72)

transformations

scaling

```
void MatGenMatScale(RealType Sx, RealType Sy, RealType Sz, MatrixType Mat)
```

Sx, Sy, Sz: Scaling factors requested.

Mat: Matrix to initialize as a scaling matrix.

Description: Routine to generate a 4*4 matrix to Scale x, y, z in Sx, Sy, Sz amounts.

5.2.41 MatGenMatTrans (misc_lib/genmat.c:50)

transformations

translation

```
void MatGenMatTrans(RealType Tx, RealType Ty, RealType Tz, MatrixType Mat)
```

Tx, Ty, Tz: Translational amounts requested.

Mat: Matrix to initialize as a translation matrix.

Description: Routine to generate a 4*4 matrix to translate in Tx, Ty, Tz amounts.

5.2.42 MatGenMatUnifScale (misc_lib/genmat.c:94)

transformations

scaling

```
void MatGenMatUnifScale(RealType Scale, MatrixType Mat)
```

Scale: Uniform scaling factor requested.

Mat: Matrix to initialize as a scaling matrix.

Description: Routine to generate a 4*4 matrix to uniformly scale Scale amount.

5.2.43 MatGenUnitMat (misc_lib/genmat.c:24)

transformations

unit matrix

```
void MatGenUnitMat(MatrixType Mat)
```

Mat: Matrix to initialize as a unit matrix.

Description: Routine to generate a 4*4 unit matrix:

5.2.44 MatInverseMatrix (misc_lib/genmat.c:434)

transformations

matrix inverse

```
int MatInverseMatrix(MatrixType M, MatrixType InvM)
```

M: Original matrix to invert.

InvM: Inverted matrix will be placed here.

Returns: TRUE if inverse exists, FALSE otherwise.

Description: Routine to compute the INVERSE of a given matrix M which is not modified. The matrix is assumed to be 4 by 4 (transformation matrix). Return TRUE if inverted matrix (InvM) do exists.

5.2.45 MatMultTwo4by4 (misc_lib/genmat.c:258)

transformations

matrix product

```
void MatMultTwo4by4(MatrixType MatRes, MatrixType Mat1, MatrixType Mat2)
```

MatRes: Result of matrix product.

Mat1, Mat2: The two operand of the matrix product.

Description: Routine to multiply 2 4by4 matrices. MatRes may be one of Mat1 or Mat2 - it is only updated in the end.

5.2.46 MatMultVecby4by4 (misc_lib/genmat.c:364)

transformations

vector matrix product

```
void MatMultVecby4by4(VectorType VRes, VectorType Vec, MatrixType Mat)
```

VRes: Result of vector - matrix product.

Vec: Vector to transfrom using Matrix.

Mat: Transformation matrix.

Description: Routine to multiply an XYZ Vector by 4by4 matrix: The Vector has only 3 components (X, Y, Z) and it is assumed that W = 1 VRes may be Vec as it is only updated in the end.

5.2.47 MatMultWVecby4by4 (misc_lib/genmat.c:403)

transformations

vector matrix product

```
void MatMultWVecby4by4(RealType VRes[4], RealType Vec[4], MatrixType Mat)
```

VRes: Result of vector - matrix product.

Vec: Vector to transfrom using Matrix.

Mat: Transformation matrix.

Description: Routine to multiply a WXYZ Vector by 4by4 matrix: The Vector has only 4 components (W, X, Y, Z). VRes may be Vec as it is only updated in the end.

5.2.48 MatScale4by4 (misc_lib/genmat.c:338)

transformations

matrix scaling

```
void MatScale4by4(MatrixType MatRes, MatrixType Mat, RealType *Scale)
```

MatRes: Result of matrix scaling.

Mat: The two operand of the matrix scaling.

Scale: Scalar value to multiple matrix with.

Description: Routine to scale a 4by4 matrix. MatRes may be Mat.

5.2.49 MatSubTwo4by4 (misc_lib/genmat.c:313)

transformations

matrix subtraction

```
void MatSubTwo4by4(MatrixType MatRes, MatrixType Mat1, MatrixType Mat2)
```

MatRes: Result of matrix subtraction.

Mat1, Mat2: The two operand of the matrix subtraction.

Description: Routine to subtract 2 4by4 matrices. MatRes may be one of Mat1 or Mat2.

5.2.50 PQCompFunc (misc_lib/priorque.c:97)

priority queue

```
void PQCompFunc(PQCompFuncType NewCompFunc)
```

NewCompFunc: A comparison function to used on item in the queue.

Description: Sets (a pointer to) the function that is used in comparing two items in the queue. This comparison function will get two item pointers, and should return >0, 0, <0 as comparison result for greater than, equal, or less than relation, respectively.

5.2.51 PQDelete (misc_lib/priorque.c:196)

priority queue

```
VoidPtr PQDelete(PriorQue **PQ, VoidPtr OldItem)
```

PQ: To delete OldItem from.

OldItem: Old element in priority queue PQ to remove from.

Returns: Removed OldItem if found, NULL otherwise.

Description: Deletes an old item from the queue, using comparison function CompFunc. Returns a pointer to Deleted item if was fould and deleted from the queure, NULL otherwise.

5.2.52 PQEmpty (misc_lib/priorque.c:75)

priority queue

```
int PQEmpty(PriorQue *PQ)
```

PQ: Priority queue to test for containment.

Returns: TRUE if not empty, FALSE otherwise.

Description: returns TRUE iff PQ priority queue is empty.

5.2.53 PQFind (misc_lib/priorque.c:277)

priority queue

```
VoidPtr PQFind(PriorQue *PQ, VoidPtr OldItem)
```

PQ: To search for OldItem at.

OldItem: Element to search in PQ.

Returns: Found element or othewise NULL.

Description: Finds old item on the queue, PQ, using the comparison function CompFunc. Returns a pointer to item if was fould, NULL otherwise.

5.2.54 PQFirst (misc_lib/priorque.c:117)

priority queue

```
VoidPtr PQFirst(PriorQue **PQ, int Delete)
```

PQ: To examine/remove first element from.

Delete: If TRUE first element is being removed from the queue.

Returns: A pointer to the first element in the queue.

Description: Returns the first element in the given priority queue, and delete it from the queue if Delete is TRUE. returns NULL if empty queue.

5.2.55 PQFree (misc_lib/priorque.c:389)

priority queue

```
void PQFree(PriorQue *PQ, int FreeItem)
```

PQ: Priority queue to release.

FreeItem: If TRUE, elements are being freed as well.

Description: Frees the given queue. The elelents are also freed if FreeItems is TRUE.

5.2.56 PQFreeFunc (misc_lib/priorque.c:417)

priority queue

```
void PQFreeFunc(PriorQue *PQ, void (*FreeFunc)(VoidPtr))
```

PQ: Priority queue to release.

FreeFunc: "Printing function".

Description: Frees the given queue. The elelents are also freed by invoking FreeFunc onall of them as FreeFunc's only argument.

5.2.57 PQInit (misc_lib/priorque.c:56)

priority queue

```
void PQInit(PriorQue **PQ)
```

PQ: To initialize.

Description: Initializes the priority queue.

5.2.58 PQInsert (misc_lib/priorque.c:153)

priority queue

```
VoidPtr PQInsert(PriorQue **PQ, VoidPtr NewItem)
```

PQ: To insert a new element to.

NewItem: The new element to insert.

Returns: An old element NewItem replaced, or NULL otherwise.

Description: Insert a new element into the queue (NewItem is a pointer to new element) using given compare function CompFunc (See PQCompFunc). Insert elelemnt will always be a leaf of the constructed tree. Returns a pointer to old element if was replaced or NULL if the element is new.

5.2.59 PQNext (misc_lib/priorque.c:318)

priority queue

```
VoidPtr PQNext(PriorQue *PQ, VoidPtr CmpItem, VoidPtr LargerThan)
```

PQ: To examine.

CmpItem: To find the smallest item in PQ that is larger than it.

LargerThan: The item that is found larger so far.

Returns: The samllest item in PQ that is larger than CmpItem or NULL if no found.

Description: Returns the smallest element in PQ that is larger than given element CmpItem. PQ is not modified. Return NULL if none was found. LargerThan will allways hold the smallest Item Larger than current one.

5.2.60 PQPrint (misc_lib/priorque.c:363)

priority queue

```
void PQPrint(PriorQue *PQ, void (*PrintFunc)(VoidPtr))
```

PQ: Pritority queue to traverse.

PrintFunc: "Printing function".

Description: Scans the priority queue in order and invokes the "printing" routine, PrintFunc on every item in the queue as its only argument.

5.2.61 getcwd (misc_lib/xgeneral.c:481)

```
char *getcwd(char *s, int Len)
```

s: Where to save current working direction.

Len: Length of s.

Returns: Same as s.

Description: Get current working directory - BSD4.3 style. *

5.2.62 movmem (misc_lib/xgeneral.c:300)

copy

```
void movmem(VoidPtr Src, VoidPtr Dest, int Len)
```

Src: Of block to copy.

Dest: Of block to copy.

Len: Of block to copy.

Description: Routine to move a block in memory. Unlike memcpy/bcopy, this routine should support overlaying blocks. This stupid implemetation will copy it twice - to a temporary block and back again. The temporary block size will be allocated by demand.

5.2.63 searchpath (misc_lib/xgeneral.c:325)

```
char *searchpath(char *Name)
```

Name: Of file to search for.

Returns: Complete file name of Name.

Description: RRoutine to search for a given file name.

5.2.64 stricmp (misc_lib/xgeneral.c:404)

```
int stricmp(char *s1, char *s2)
```

s1, s2: The two strings to compare.

Returns: <0, 0, >0 according to the relation between s1 and s2.

Description: Routine to compare two strings, ignoring case.

5.2.65 strnicmp (misc_lib/xgeneral.c:363)

```
int strnicmp(char *s1, char *s2, int n)
```

s1, s2: The two strings to compare.

n: maximum number of characters to compare.

Returns: <0, 0, >0 according to the relation between s1 and s2.

Description: Routine to compare two strings, ignoring case, up to given length.

5.2.66 strstr (misc_lib/xgeneral.c:450)

```
char *strstr(char *s, char *Pattern)
```

s: To search for Pattern in.

Pattern: To search in s.

Returns: Address in s where Pattern was first found, NULL otherwise.

Description: Routine to search for a Pattern (no regular expression) in s. Returns address in s of first occurance of Pattern, NULL if non found.

Chapter 6

Prsr Library, prsr_lib

6.1 General Information

This library provides the data file interface for IRIT. Functions are provided to read and write data files, both compressed (on unix only, using *compress*), and uncompressed, in binary and/or ascii text modes. This library is also used to exchange data between the IRIT server and the display devices' clients. Several header files can be found for this library:

Header (include/*.h)	Functionality
allocate.h	High level dynamic allocation of objects
attribut.h	High level attributes for objects
ip_cnvrt.h	Freeform to polygon and polyline high level conversion
iritprsr.h	Main interface to reading and writing data
irit_soc.h	Socket communication for data exchange

6.2 Library Functions

6.2.1 AttrGetObjAttrib (prsr_lib/attribut.c:414)

attributes

```
IPObjectStruct *AttrGetObjAttrib(IPAttributeStruct *Attrs, char *Name)
```

Attrs: Attribute list to search for requested attribute.

Name: Name of requested attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute.

6.2.2 AttrGetObjectColor (prsr_lib/attribut.c:48)

attributes

color

```
int AttrGetObjectColor(IPObjectStruct *PObj)
```

PObj: For which we would like to know the color of.

Returns: Color of PObj or IP_ATTR_NO_COLOR if no color set.

Description: Routine to return the color of an object.

6.2.3 AttrGetObjectIntAttrib (prsr_lib/attribut.c:157)

attributes

```
int AttrGetObjectIntAttrib(IPObjectStruct *PObj, char *Name)
```

PObj: Object from which to get an ingeter attribute.

Name: Name of integer attribute.

Returns: Found attribute, or IP_ATTR_BAD_INT if not found.

Description: Routine to get an integer attribute from an object.

6.2.4 AttrGetObjectObjAttrib (prsr_lib/attribut.c:395)

attributes

```
IPObjectStruct *AttrGetObjectObjAttrib(IPObjectStruct *PObj, char *Name)
```

PObj: Object from which to get a object attribute.

Name: Name of object attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get an object attribute from an object.

6.2.5 AttrGetObjectPtrAttrib (prsr_lib/attribut.c:205)

attributes

```
VoidPtr AttrGetObjectPtrAttrib(IPObjectStruct *PObj, char *Name)
```

PObj: Object from which to get a pointer attribute.

Name: Name of pointer attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a pointer attribute from an object.

6.2.6 AttrGetObjectRGBColor (prsr_lib/attribut.c:106)

attributes

color

rgb

```
*****************************************************************************/
int AttrGetObjectRGBColor(IPObjectStruct *PObj,
                          int *Red,
                          int *Green,
                          int *Blue)
```

PObj: Object to get its RGB color.

Red, Green, Blue: Component of color to initialize.

Returns: TRUE if PObj does have an RGB color attribute, FALSE otherwise.

Description: Routine to return the RGB color of an object.

6.2.7 AttrGetObjectRealAttrib (prsr_lib/attribut.c:253)

attributes

```
RealType AttrGetObjectRealAttrib(IPObjectStruct *PObj, char *Name)
```

PObj: Object from which to get a real attribute.

Name: Name of real attribute.

Returns: Found attribute, or IP_ATTR_BAD_REAL if not found.

Description: Routine to get a real attribute from an object.

6.2.8 AttrGetObjectStrAttrib (prsr_lib/attribut.c:301)

attributes

```
char *AttrGetObjectStrAttrib(IPObjectStruct *PObj, char *Name)
```

PObj: Object from which to get a string attribute.

Name: Name of string attribute.

Returns: Found attribute, or NULL if not found.

Description: Routine to get a string attribute from an object.

6.2.9 AttrSetObjAttrib (prsr_lib/attribut.c:361)

attributes

```
void AttrSetObjAttrib(IPAttributeStruct **Attrs,
                      char *Name,
                      IPObjectStruct *Data,
                      int CopyData)
```

Attrs: Attribute list where to place new attribute.

Name: Name of thely introducedattribute

Data: Pointer attribute to save.

CopyData: If TRUE, object Data is duplicated first.

Description: Routine to set an object attribute.

6.2.10 AttrSetObjectColor (prsr_lib/attribut.c:29)

attributes

color

```
void AttrSetObjectColor(IPObjectStruct *PObj, int Color)
```

PObj: Object to set its color to Color.

Color: New color for PObj.

Description: Routine to set the color of an object.

6.2.11 AttrSetObjectIntAttrib (prsr_lib/attribut.c:129)

attributes

```
void AttrSetObjectIntAttrib(IPObjectStruct *PObj, char *Name, int Data)
```

PObj: To add an integer attribute for.

Name: Name of attribute.

Data: Content of attribute.

Description: Routine to set an integer attribute for an object.

6.2.12 AttrSetObjectObjAttrib (prsr_lib/attribut.c:325)

attributes

```
void AttrSetObjectObjAttrib(IPObjectStruct *PObj,
                            char *Name,
                            IPObjectStruct *Data,
                            int CopyData)
```

PObj: To add an object attribute for.

Name: Name of attribute.

Data: Content of attribute.

CopyData: If TRUE, Data object is duplicated first.

Description: Routine to set an object attribute for an object.

6.2.13 AttrSetObjectPtrAttrib (prsr_lib/attribut.c:177)

attributes

```
void AttrSetObjectPtrAttrib(IPObjectStruct *PObj, char *Name, VoidPtr Data)
```

PObj: To add a pointer attribute for.

Name: Name of attribute.

Data: Content of attribute.

Description: Routine to set a pointer attribute for an object.

6.2.14 AttrSetObjectRGBColor (prsr_lib/attribut.c:79)

attributes color rgb

```
void AttrSetObjectRGBColor(IPObjectStruct *PObj, int Red, int Green, int Blue)
```

PObj: Object to set its RGB color.

Red, Green, Blue: Component of color.

Description: Routine to set the RGB color of an object.

6.2.15 AttrSetObjectRealAttrib (prsr_lib/attribut.c:225)

attributes

```
void AttrSetObjectRealAttrib(IPObjectStruct *PObj, char *Name, RealType Data)
```

PObj: To add a real attribute for.

Name: Name of attribute.

Data: Content of attribute.

Description: Routine to set a real attribute for an object.

6.2.16 AttrSetObjectStrAttrib (prsr_lib/attribut.c:273)

attributes

```
void AttrSetObjectStrAttrib(IPObjectStruct *PObj, char *Name, char *Data)
```

PObj: To add a string attribute for.

Name: Name of attribute.

Data: Content of attribute.

Description: Routine to set a string attribute for an object.

6.2.17 BspCrvReadFromFile (prsr_lib/bsp_read.c:30)

files read

```
CagdCrvStruct *BspCrvReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the Bspline curve from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read curve, or NULL if an error occured.

Description: Reads from a file Bspline curve(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.18 BspCrvReadFromFile2 (prsr_lib/bsp_read.c:93)

files read stream

```
CagdCrvStruct *BspCrvReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[CURVE BSPLINE" has been read, FALSE otherwise.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read curve, or NULL if an error occured.

Description: Reads from a file a Bspline curve. If NameWasRead is TRUE, it is assumed prefix "[CURVE BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of curve. If no error is detected *ErrStr is set to NULL.

6.2.19 BspCrvWriteToFile (prsr_lib/bsp_wrt.c:37)

files

write

```
int BspCrvWriteToFile(CagdCrvStruct *Crvs,
                      char *FileName,
                      int Indent,
                      char *Comment,
                      char **ErrStr)
```

Crvs: To write to file FileName.

FileName: Name of file to open so we can write Crvs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bspline curve(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.20 BspCrvWriteToFile2 (prsr_lib/bsp_wrt.c:82)

files

write

```
int BspCrvWriteToFile2(CagdCrvStruct *Crvs,
                       int Handler,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

Crvs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bspline curve(s) list into file. Returns TRUE if succesful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.21 BspSrfReadFromFile (prsr_lib/bsp_read.c:261)

files

read

```
CagdSrfStruct *BspSrfReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the Bspline surface from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read surface, or NULL if an error occured.

Description: Reads from a file Bspline surface(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.22 BspSrfReadFromFile2 (prsr_lib/bsp_read.c:324)

files

read

stream

```
CagdSrfStruct *BspSrfReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[SURFACE BSPLINE" hasbeen read, FALSE otherwise.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read surface, or NULL if an error occured.

Description: Reads from a file a Bspline surface. If NameWasRead is TRUE, it is assumed prefix "[SURFACE BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of surface. If no error is detected *ErrStr is set to NULL.

6.2.23 BspSrfWriteToFile (prsr_lib/bsp_wrt.c:171)

files

write

```
int BspSrfWriteToFile(CagdSrfStruct *Srfs,
                      char *FileName,
                      int Indent,
                      char *Comment,
                      char **ErrStr)
```

Srfs: To write to file FileName.

FileName: Name of file to open so we can write Srfs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bspline surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.24 BspSrfWriteToFile2 (prsr_lib/bsp_wrt.c:216)

files

write

```
int BspSrfWriteToFile2(CagdSrfStruct *Srfs,
                       int Handler,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

Srfs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bspline surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.25 BzrCrvReadFromFile (prsr_lib/bzr_read.c:30)

files

read

```
CagdCrvStruct *BzrCrvReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the Bezier curve from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read curve, or NULL if an error occured.

Description: Reads from a file Bezier curve(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.26 BzrCrvReadFromFile2 (prsr_lib/bzr_read.c:93)

files

read

stream

```
CagdCrvStruct *BzrCrvReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[CURVE BEZIER" hasbeen read, FALSE otherwise.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read curve, or NULL if an error occured.

Description: Reads from a file a Bezier curve. If NameWasRead is TRUE, it is assumed prefix "[CURVE BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of curve. If no error is detected *ErrStr is set to NULL.

6.2.27 BzrCrvWriteToFile (prsr_lib/bzr_wrt.c:37)

files

write

```
int BzrCrvWriteToFile(CagdCrvStruct *Crvs,
                      char *FileName,
                      int Indent,
                      char *Comment,
                      char **ErrStr)
```

Crvs: To write to file FileName.

FileName: Name of file to open so we can write Crvs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bezier curve(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.28 BzrCrvWriteToFile2 (prsr_lib/bzr_wrt.c:82)

files

write

```
int BzrCrvWriteToFile2(CagdCrvStruct *Crvs,
                       int Handler,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

Crvs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bezier curve(s) list into file. Returns TRUE if succesful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.29 BzrSrfReadFromFile (prsr_lib/bzr_read.c:217)

files

read

```
CagdSrfStruct *BzrSrfReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the Bezier surface from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read surface, or NULL if an error occured.

Description: Reads from a file Bezier surface(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.30 BzrSrfReadFromFile2 (prsr_lib/bzr_read.c:280)

files

read

stream

```
CagdSrfStruct *BzrSrfReadFromFile2(int Handler,
                                   CagdBType NameWasRead,
                                   char **ErrStr,
                                   int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: TRUE if "[SURFACE BEZIER" hasbeen read, FALSE otherwise.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read surface, or NULL if an error occured.

Description: Reads from a file a Bezier surface. If NameWasRead is TRUE, it is assumed prefix "[SURFACE BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of surface. If no error is detected *ErrStr is set to NULL.

6.2.31 BzrSrfWriteToFile (prsr_lib/bzr_wrt.c:154)

files

write

```
int BzrSrfWriteToFile(CagdSrfStruct *Srfs,
                      char *FileName,
                      int Indent,
                      char *Comment,
                      char **ErrStr)
```

Srfs: To write to file FileName.

FileName: Name of file to open so we can write Srfs in.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bezier surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.32 BzrSrfWriteToFile2 (prsr_lib/bzr_wrt.c:199)

files

write

```
int BzrSrfWriteToFile2(CagdSrfStruct *Srfs,
                       int Handler,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

Srfs: To write to open stream.

Handler: A handler to the open stream.

Indent: Primary indentation. All information will be written from the column specified by Indent.

Comment: Optional, to describe the geometry.

ErrStr: If an error occurs, to describe the error.

Returns: TRUE if succesful, FALSE otherwise.

Description: Writes Bezier surface(s) list into file. Returns TRUE if succesful, FALSE otherwise. The file descriptor is not closed. If Comment is NULL, no comment is wrriten, if "" only internal comment is written.

6.2.33 CagdCrvReadFromFile (prsr_lib/cagdread.c:27)

files

read

```
CagdCrvStruct *CagdCrvReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the curve from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read curve, or NULL if an error occured.

Description: Reads from a file curve(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.34 CagdCrvReadFromFile2 (prsr_lib/cagdread.c:150)

files

read

stream

```
CagdCrvStruct *CagdCrvReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in strea of the error, if occured.

Returns: The read curve, or NULL if an error occured.

Description: Reads from a stream a curve. It is assumed prefix "[CURVE" has already been read. This is useful for a global parser which invokes this routine to read from a stream several times as a parent controller. For exactly this reason, the given stream descriptor is NOT closed in the end. If error is found in reading the stream, ErrStr is set to a string describing it and ErrLine to line it occured in stream relative to begining of curve. If no error is detected *ErrStr is set to NULL.

6.2.35 CagdCrvWriteToFile (prsr_lib/cagd_wrt.c:39)

files

write

```
int CagdCrvWriteToFile(CagdCrvStruct *Crvs,
                       char *FileName,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

Crvs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.36 CagdCrvWriteToFile2 (prsr_lib/cagd_wrt.c:92)

files

write

```
int CagdCrvWriteToFile2(CagdCrvStruct *Crvs,
                        int Handler,
                        int Indent,
                        char *Comment,
                        char **ErrStr)
```

Crvs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.37 CagdCrvWriteToFile3 (prsr_lib/cagd_wrt.c:146)

files

write

```
int CagdCrvWriteToFile3(CagdCrvStruct *Crvs,
                        FILE *f,
                        int Indent,
                        char *Comment,
                        char **ErrStr)
```

Crvs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.38 CagdCrvWriteToFile3 (prsr_lib/triv_wrt.c:134)

files

write

```
int TrivTVWriteToFile3(TrivTVStruct *TVs,
                       FILE *f,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

TVs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write curve(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.39 CagdSrfReadFromFile (prsr_lib/cagdread.c:86)

files read

```
CagdSrfStruct *CagdSrfReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the surface from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read surface, or NULL if an error occured.

Description: Reads from a file surface(s). If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.40 CagdSrfReadFromFile2 (prsr_lib/cagdread.c:190)

files read stream

```
CagdSrfStruct *CagdSrfReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in stream of the error, if occured.

Returns: The read surface, or NULL if an error occured.

Description: Reads from a stream a surface. It is assumed prefix "[SURFACE" has already been read. This is useful for a global parser which invokes this routine to read from a stream several times as a parent controller. For exactly this reason, the given stream descriptor is NOT closed in the end. If error is found in reading the stream, ErrStr is set to a string describing it and ErrLine to line it occured in stream relative to begining of surface. If no error is detected *ErrStr is set to NULL.

6.2.41 CagdSrfWriteToFile (prsr_lib/cagd_wrt.c:178)

files write

```
int CagdSrfWriteToFile(CagdSrfStruct *Srfs,
                       char *FileName,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

Srfs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write surface(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.42 CagdSrfWriteToFile2 (prsr_lib/cagd_wrt.c:232)

files write stream

```
int CagdSrfWriteToFile2(CagdSrfStruct *Srfs,
                        int Handler,
                        int Indent,
                        char *Comment,
                        char **ErrStr)
```

Srfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write surface(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.43 CagdSrfWriteToFile3 (prsr_lib/trim_wrt.c:143)

files

write

```
int TrimWriteTrimmedSrfToFile3(TrimSrfStruct *TrimSrfs,
                               FILE *f,
                               int Indent,
                               char *Comment,
                               char **ErrStr)
```

TrimSrfs: To be saved in stream.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trimmed surface(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.44 CagdSrfWriteToFile3 (prsr_lib/cagd_wrt.c:286)

files

write

```
int CagdSrfWriteToFile3(CagdSrfStruct *Srfs,
                        FILE *f,
                        int Indent,
                        char *Comment,
                        char **ErrStr)
```

Srfs: To be saved in file f.

f: File descriptor where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write surface(s) to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.45 CopyObject (prsr_lib/allocate.c:1259)

copy

```
IPObjectStruct *CopyObject(IPObjectStruct *Dest,
                           IPObjectStruct *Src,
                           int CopyAll)
```

Dest: Destination object, possibly NULL.

Src: Source object.

CopyAll: Do we want a complete identical copy?

Returns: Duplicate of Src, same as Dest if Dest != NULL.

Description: Routine to create a whole new copy of an object Src into Dest. If Dest is NULL, new object is allocated, otherwise Dest itself is updated to hold the new copy. If CopyAll then all the record is copied, otherwise, only its invariant elements arebeen copied (i.e. no Name/Pnext copying).

6.2.46 CopyObjectList (prsr_lib/allocate.c:1370)

copy

```
IPObjectStruct *CopyObjectList(IPObjectStruct *PObjs, int CopyAll)
```

Pobjs: Source objects.

CopyAll: Do we want a complete identical copy?

Returns: Duplicated list of PObjs.

Description: Routine to create a new copy of an object list.

6.2.47 CopyPolygonList (prsr_lib/allocate.c:1401)

copy

```
IPPolygonStruct *CopyPolygonList(IPPolygonStruct *Src)
```

Src: A polygon list to copy.

Returns: Duplicated list of polygons.

Description: Routine to create a new copy of an object polygon list.

6.2.48 CopyVertexList (prsr_lib/allocate.c:1442)

copy

```
IPVertexStruct *CopyVertexList(IPVertexStruct *Src)
```

Src: A vertex list to copy.

Returns: Duplicated list of vertices.

Description: Routine to create a new copy of a polygon vertices list.

6.2.49 GenCRVObject (prsr_lib/allocate.c:696)

allocation

```
IPObjectStruct *GenCRVObject(CagdCrvStruct *Crv)
```

Crv: Curves to place in object.

Returns: A newly created curve object.

Description: Creates one curve object.

6.2.50 GenCTLPTObject (prsr_lib/allocate.c:902)

allocation

```
IPObjectStruct *GenCTLPTObject(CagdPointType PtType,
                               CagdRType *CagdCoords,
                               RealType *Coords)
```

PtType: Point type of created control point (E2, P3, etc.).

CagdCoords: If specified, used as coefficients of new control point.

Coords: If specified, used as coefficients of new control point.

Returns: A newly created control point object.

Description: Creates one control point object. Only one of CagdCoords/Coords should be specified.

6.2.51 GenCrvObject (prsr_lib/allocate.c:672)

allocation

```
IPObjectStruct *GenCrvObject(char *Name,
                             CagdCrvStruct *Crv,
                             IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Crv: Curves to place in object.

Pnext: Entry into the object structure.

Returns: A newly created curve object.

Description: Creates one curve object.

6.2.52 GenCtlPtObject (prsr_lib/allocate.c:861)

allocation

```
IPObjectStruct *GenCtlPtObject(char *Name,
                               CagdPointType PtType,
                               CagdRType *CagdCoords,
                               RealType *Coords,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
PtType: Point type of created control point (E2, P3, etc.).
CagdCoords: If specified, used as coefficients of new control point.
Coords: If specified, used as coefficients of new control point.
Pnext: Entry into the object structure.

Returns: A newly created control point object.

Description: Creates one control point object. Only one of CagdCoords/Coords should be specified.

6.2.53 GenMATObject (prsr_lib/allocate.c:1210)

allocation

```
IPObjectStruct *GenMATObject(MatrixType Mat)
```

Mat: Matrix to initialize with.

Returns: A newly created matrix object.

Description: Creates one matrix object.

6.2.54 GenMatObject (prsr_lib/allocate.c:1183)

allocation

```
IPObjectStruct *GenMatObject(char *Name, MatrixType Mat, IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
Mat: Matrix to initialize with.
Pnext: Entry into the object structure.

Returns: A newly created matrix object.

Description: Creates one matrix object.

6.2.55 GenNUMObject (prsr_lib/allocate.c:946)

allocation

```
IPObjectStruct *GenNUMObject(RealType *R)
```

R: Numeric value to place in object.

Returns: A newly created numeric object.

Description: Creates one numeric object.

6.2.56 GenNUMValObject (prsr_lib/allocate.c:964)

allocation

```
IPObjectStruct *GenNUMValObject(RealType R)
```

R: Numeric value to place in object.

Returns: A newly created numeric object.

Description: Creates one numeric object.

6.2.57 GenNumObject (prsr_lib/allocate.c:922)

allocation

```
IPObjectStruct *GenNumObject(char *Name, RealType *R, IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
R: Numeric value to place in object.
Pnext: Entry into the object structure.

Returns: A newly created numeric object.

Description: Creates one numeric object.

6.2.58 GenPLANEObject (prsr_lib/allocate.c:1163)

allocation

```
IPObjectStruct *GenPLANEObject(RealType *Plane0,
                               RealType *Plane1,
                               RealType *Plane2,
                               RealType *Plane3)
```

Plane0, Plane1, Plane2, Plane3: Coefficients of point.

Returns: A newly created plane object.

Description: Creates one plane object.

6.2.59 GenPOLYObject (prsr_lib/allocate.c:650)

allocation

```
IPObjectStruct *GenPOLYObject(IPPolygonStruct *Pl)
```

Pl: Polygon(s) to place in object.

Returns: A newly created polygonal object.

Description: Creates one polygonal object.

6.2.60 GenPTObject (prsr_lib/allocate.c:1014)

allocation

```
IPObjectStruct *GenPTObject(RealType *Pt0, RealType *Pt1, RealType *Pt2)
```

Pt0, Pt1, Pt2: Coefficients of point.

Returns: A newly created point object.

Description: Creates one point object.

6.2.61 GenPlaneObject (prsr_lib/allocate.c:1133)

allocation

```
IPObjectStruct *GenPlaneObject(char *Name,
                               RealType *Plane0,
                               RealType *Plane1,
                               RealType *Plane2,
                               RealType *Plane3,
                               IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
Plane0, Plane1, Plane2, Plane3: Coefficients of point.
Pnext: Entry into the object structure.

Returns: A newly created plane object.

Description: Creates one plane object.

6.2.62 GenPolyObject (prsr_lib/allocate.c:625)

allocation

```
IPObjectStruct *GenPolyObject(char *Name,
                              IPPolygonStruct *Pl,
                              IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
Pl: Polygon(s) to place in object.
Pnext: Entry into the object structure.

Returns: A newly created polygonal object.

Description: Creates one polygonal object.

6.2.63 GenPtObject (prsr_lib/allocate.c:988)

allocation

```
IPObjectStruct *GenPtObject(char *Name,
                            RealType *Pt0,
                            RealType *Pt1,
                            RealType *Pt2,
                            IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
Pt0, Pt1, Pt2: Coefficients of point.
Pnext: Entry into the object structure.

Returns: A newly created point object.

Description: Creates one point object.

6.2.64 GenSRFObject (prsr_lib/allocate.c:742)

allocation

```
IPObjectStruct *GenSRFObject(CagdSrfStruct *Srf)
```

Srf: Surfaces to place in object.

Returns: A newly created surface object.

Description: Creates one surface object.

6.2.65 GenSTRObject (prsr_lib/allocate.c:1108)

allocation

```
IPObjectStruct *GenSTRObject(char *Str)
```

Str: The string.

Returns: A newly created strtor object.

Description: Creates one string object.

6.2.66 GenSrfObject (prsr_lib/allocate.c:718)

allocation

```
IPObjectStruct *GenSrfObject(char *Name,
                             CagdSrfStruct *Srf,
                             IPObjectStruct *Pnext)
```

Name: Name of polygonal object.
Srf: Surfaces to place in object.
Pnext: Entry into the object structure.

Returns: A newly created surface object.

Description: Creates one surface object.

6.2.67 GenStrObject (prsr_lib/allocate.c:1084)

allocation

```
IPObjectStruct *GenStrObject(char *Name, char *Str, IPObjectStruct *Pnext)
```

Name: Name of string object.

Str: The string.

Pnext: Entry into the object structure.

Returns: A newly created strtor object.

Description: Creates one string object.

6.2.68 GenTRIMSRFObject (prsr_lib/allocate.c:788)

allocation

```
IPObjectStruct *GenTRIMSRFObject(TrimSrfStruct *TrimSrf)
```

TrimSrf: Trimmed surfaces to place in object.

Returns: A newly created trimmed surface object.

Description: Creates one surface object.

6.2.69 GenTRIVARObject (prsr_lib/allocate.c:834)

allocation

```
IPObjectStruct *GenTRIVARObject(TrivTVStruct *Triv)
```

Triv: Trivariates to place in object.

Returns: A newly created trivariate object.

Description: Creates one trivariate object.

6.2.70 GenTrimSrfObject (prsr_lib/allocate.c:764)

allocation

```
IPObjectStruct *GenTrimSrfObject(char *Name,
                                 TrimSrfStruct *TrimSrf,
                                 IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

TrimSrf: Trimmed surfaces to place in object.

Pnext: Entry into the object structure.

Returns: A newly created trimmed surface object.

Description: Creates one surface object.

6.2.71 GenTrivarObject (prsr_lib/allocate.c:810)

allocation

```
IPObjectStruct *GenTrivarObject(char *Name,
                                TrivTVStruct *Triv,
                                IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Triv: Trivariates to place in object.

Pnext: Entry into the object structure.

Returns: A newly created trivariate object.

Description: Creates one trivariate object.

6.2.72 GenVECObject (prsr_lib/allocate.c:1064)

allocation

```
IPObjectStruct *GenVECObject(RealType *Vec0, RealType *Vec1, RealType *Vec2)
```

Vec0, Vec1, Vec2: Coefficients of vector.

Returns: A newly created vector object.

Description: Creates one vector object.

6.2.73 GenVecObject (prsr_lib/allocate.c:1038)

allocation

```
IPObjectStruct *GenVecObject(char *Name,
                             RealType *Vec0,
                             RealType *Vec1,
                             RealType *Vec2,
                             IPObjectStruct *Pnext)
```

Name: Name of polygonal object.

Vec0, Vec1, Vec2: Coefficients of vector.

Pnext: Entry into the object structure.

Returns: A newly created vector object.

Description: Creates one vector object.

6.2.74 IPAllacPolygon (prsr_lib/allocate.c:183)

allocation

```
IPPolygonStruct *IPAllocPolygon(ByteType Count,
                                ByteType Tags,
                                IPVertexStruct *V,
                                IPPolygonStruct *Pnext)
```

Count: Entry into new vertex structure.

Tags: Entry into new vertex structure.

V: Entry into new vertex structure.

Pnext: Entry into new vertex structure.

Returns: A new allocated polygon structure.

Description: Allocates one Polygon Structure.

6.2.75 IPAllocObject (prsr_lib/allocate.c:238)

allocation

```
IPObjectStruct *IPAllocObject(char *Name,
                              IPObjStructType ObjType,
                              IPObjectStruct *Pnext)
```

Name: Name of newly allocated object.

ObjType: Object type of newly allocated object.

Pnext: Entry into new object structure.

Returns: A new allocated object structure.

Description: Allocates one Object Structure.

6.2.76 IPAllocVertex (prsr_lib/allocate.c:126)

allocation

```
IPVertexStruct *IPAllocVertex(ByteType Count,
                              ByteType Tags,
                              IPPolygonStruct *PAdj,
                              IPVertexStruct *Pnext)
```

Count: Entry into new vertex structure.

Tags: Entry into new vertex structure.

PAdj: Entry into new vertex structure.

Pnext: Entry into new vertex structure.

Returns: A new allocated vertex structure.

Description: Allocates one Vertex Structure.

6.2.77 IPFreeObject (prsr_lib/allocate.c:332)

allocation

```
void IPFreeObject(IPObjectStruct *O)
```

O: To free.

Description: Frees one Object Structure.

6.2.78 IPFreeObjectList (prsr_lib/allocate.c:428)

allocation

```
void IPFreeObjectList(IPObjectStruct *OFirst)
```

OFirst: To free.

Description: Free a list of Object structures.

6.2.79 IPFreePolygon (prsr_lib/allocate.c:311)

allocation

```
void IPFreePolygon(IPPolygonStruct *P)
```

P: To free.

Description: Frees one Polygon Structure.

6.2.80 IPFreePolygonList (prsr_lib/allocate.c:395)

allocation

```
void IPFreePolygonList(IPPolygonStruct *PFirst)
```

PFirst: To free.

Description: Free a list of Polygon structures.

6.2.81 IPFreeVertex (prsr_lib/allocate.c:290)

allocation

```
void IPFreeVertex(IPVertexStruct *V)
```

V: To free.

Description: Frees one Vertex Structure.

6.2.82 IPFreeVertexList (prsr_lib/allocate.c:363)

allocation

```
void IPFreeVertexList(IPVertexStruct *VFirst)
```

VFirst: To free.

Description: Free a, possibly circular, list of Vertex structures.

6.2.83 IritCurve2CtlPoly (prsr_lib/ip_cnvrt.c:146)

conversion

```
IPPolygonStruct *IritCurve2CtlPoly(CagdCrvStruct *Crv)
```

Crv: To extract its control polygon as a polyline.

Returns: A polyline representing Crv's control polygon.

Description: Routine to convert a single curve's control polygon into a polyline.

6.2.84 IritCurve2Polylines (prsr_lib/ip_cnvrt.c:85)

conversion

approximation

```
IPPolygonStruct *IritCurve2Polylines(CagdCrvStruct *Crv,
                                     int SamplesPerCurve,
                                     int Optimal)
```

Crv: To approximate as a polyline .

SamplesPerCurve: Number of samples to compute on the polyline.

Optimal: If FALSE samples are uniform in parametric space, otherwise attempt is made to sample optimally.

Returns: A polyline approximating Crv.

Description: Routine to convert one curve into a polyline with SamplesPerCurve samples. If Optimal, attempt is made to optimally distribute the sampled points.

6.2.85 IritCurvesToCubicBzrCrvs (prsr_lib/ip_cnvrt.c:723)

conversion

approximation

```
CagdCrvStruct *IritCurvesToCubicBzrCrvs(CagdCrvStruct *Crvs,
                                        IPPolygonStruct **CtlPolys,
                                        CagdBType DrawCurve,
                                        CagdBType DrawCtlPoly,
                                        CagdRType MaxArcLen)
```

Crvs: To approximate as cubic Bezier curves.

CtlPolys: If we want control polygons as well (DrawCtlPoly == TRUE) they will be placed herein.

DrawCurve: Do we want to draw the curves?

DrawCtlPoly: Do we want to draw the control polygons?

MaxArcLen: Tolerance for cubic Bezier approximation. See function BzrApproxBzrCrvAsCubics.

Returns: The cubic Bezier approximation, or NULL if DrawCurve is FALSE.

Description: Approximates an arbitrary list of curves into cubic Beziers curves.

6.2.86 IritPrsrAppendObjLists (prsr_lib/iritprs2.c:1123)

linked lists

```
IPObjectStruct *IritPrsrAppendObjLists(IPObjectStruct *OList1,
                                       IPObjectStruct *OList2)
```

OList1, OList2: Two lists to append.

Returns: Appended list.

Description: Appends two object lists together.

6.2.87 IritPrsrAppendPolyLists (prsr_lib/iritprs2.c:1048)

linked lists

```
IPPolygonStruct *IritPrsrAppendPolyLists(IPPolygonStruct *PList1,
                                         IPPolygonStruct *PList2)
```

PList1, PList2: Two lists to append.

Returns: Appended list.

Description: Appends two poly lists together.

6.2.88 IritPrsrAppendVrtxLists (prsr_lib/iritprs2.c:974)

linked lists

```
IPVertexStruct *IritPrsrAppendVrtxLists(IPVertexStruct *VList1,
                                        IPVertexStruct *VList2)
```

VList1, VList2: Two lists to append.

Returns: Appended list.

Description: Appends two vertex lists together.

6.2.89 IritPrsrClntAcceptConnect (prsr_lib/soc_clnt.c:545)

```
int IritPrsrClntAcceptConnect(int *PrgmInput, int *PrgmOutput)
```

PrgmInput: A handler to the Program's input channel.

PrgmOutput: A handler to the Program's output channel.

Returns: TRUE, if succesful, FALSE otherwise.

Description: Accept a connection. Must be called by clients at the beginning.

6.2.90 IritPrsrClntCloseConnect (prsr_lib/soc_clnt.c:598)

```
int IritPrsrClntCloseConnect(int PrgmInput, int PrgmOutput)
```

PrgmInput: A handler to the Program's input channel.

PrgmOutput: A handler to the Program's output channel.

Returns: TRUE, if succesful, FALSE otherwise.

Description: Close a connection. Must be called by clients at the end.

6.2.91 IritPrsrCloseStream (prsr_lib/iritprs1.c:176)

files

stream

parser

```
void IritPrsrCloseStream(int Handler, int Free)
```

Handler: A handler to the open stream.

Free: If TURE, release content.

Description: Close a data file for read/write.

6.2.92 IritPrsrCoerceCommonSpace (prsr_lib/coerce.c:28)

coercion

```
CagdPointType IritPrsrCoerceCommonSpace(IPObjectStruct *PtObjList, int Type)
```

PtObjList: List of points.

Type: Point type that we must span its space as well.

Returns: Point type that spans the space of point type Type as well as all points in PtObjList.

Description: Given a set of points, returns the list's common denominator that spans the space of all the points, taking into account type Type.

6.2.93 IritPrsrCoerceObjectTo (prsr_lib/coerce.c:117)

coercion

```
IPObjectStruct *IritPrsrCoerceObjectTo(IPObjectStruct *PObj, int NewType)
```

PObj: Object to coerce.

NewType: New type which can be object type like IP_OBJ_VECTOR or point type like E2.

Returns: Newly coerced object.

Description: Coerces an object to a new object. Points, vectors, control points and planes can always be coerced between themselves using this routine by specifying the new object type desired such as IP_OBJ_PLANE or control point type like CAGD_PT_E4_TYPE. Control points of curves and surfaces may be coerced to a new type by prescribing the needed point type as NewType, such as CAGD_PT_P2_TYPE.

6.2.94 IritPrsrCoercePtsListTo (prsr_lib/coerce.c:71)

coercion

```
CagdPointType IritPrsrCoercePtsListTo(IPObjectStruct *PtObjList, int Type)
```

PtObjList: Coerce points/vectors/control points in this list to Type.

Type: Minimum space type to coerce to in PtObjList.

Returns: The type coercion actualy took place with in PtObjList.

Description: Coerces a list of objects to Type.

6.2.95 IritPrsrConcatFreeForm (prsr_lib/iritprs1.c:1606)

conversion

```
IPObjectStruct *IritPrsrConcatFreeForm(IritPrsrFreeFormStruct *FreeForms)
```

FreeForms: Freeform geometry to process.

Returns: concatenated linked list.

Description: Concatenate all freeform objects in FreeForms into a single list.

6.2.96 IritPrsrFatalError (prsr_lib/ip_fatal.c:27)

error handling

```
void IritPrsrFatalError(char *Msg)
```

Msg: Error message.

Description: Default trap for Irit parser errors. This function prints the provided error message anddies.

6.2.97 IritPrsrFlattenTree (prsr_lib/iritprs1.c:605)

```
IPObjectStruct *IritPrsrFlattenTree(IPObjectStruct *PObj)
```

PObj: Object(s) to flatten out.

Returns: Flattened hierarchy.

Description: Flattens out a tree hierarchy of objects into a linear list, in place.

6.2.98 IritPrsrGetBinObject (prsr_lib/iritprsb.c:208)

files

parser

```
IPObjectStruct *IritPrsrGetBinObject(int Handler)
```

Handler: A handler to the open stream.

Returns: Read object.

Description: Routine to read one object from a given binary file, directly. Objects may be recursively defined (as lists), in which case all are read in this single call.

6.2.99 IritPrsrGetDataFiles (prsr_lib/iritprs1.c:320)

files

parser

```
IPObjectStruct *IritPrsrGetDataFiles(char **DataFileNames,
                                     int NumOfDataFiles,
                                     int Messages,
                                     int MoreMessages)
```

DataFileNames: Array of strings (file names) to process.

NumOfDataFiles: Number of elements in DataFileNames.

Messages: Do we want error messages?

MoreMessages: Do we want informative messages?

Returns: Objects read from all files.

Description: Reads data from a set of files specified by file names. Messages and MoreMessages controls the level of printout to stderr. Freeform geometry read in is handed out to a call back function named IritPrsrProcessFreeForm before it is returned from this routine. This is done so applications that do not want to deal with freeform shapes will be able to provide a call back that processes the freeform shapes into other geometry such as polygons.

6.2.100 IritPrsrGetLastObj (prsr_lib/iritprs2.c:1077)

linked lists

last element

```
IPObjectStruct *IritPrsrGetLastObj(IPObjectStruct *OList)
```

OList: A list of objects.

Returns: Last object in list OList.

Description: Returns a pointer to last object of a list.

6.2.101 IritPrsrGetLastPoly (prsr_lib/iritprs2.c:1003)

linked lists

last element

```
IPPolygonStruct *IritPrsrGetLastPoly(IPPolygonStruct *PList)
```

PList: A list of polygons.

Returns: Last polygon in list PList.

Description: Returns a pointer to last polygon/line of a list.

6.2.102 IritPrsrGetLastVrtx (prsr_lib/iritprs2.c:925)

linked lists

last element

```
IPVertexStruct *IritPrsrGetLastVrtx(IPVertexStruct *VList)
```

VList: A list of vertices

Returns: Last vertex in VList.

Description: Returns a pointer to last vertex of a list.

6.2.103 IritPrsrGetObjects (prsr_lib/iritprs1.c:377)

files

parser

```
IPObjectStruct *IritPrsrGetObjects(int Handler)
```

Handler: A handler to the open stream.

Returns: Read object, or NULl if failed.

Description: Routine to read the data from a given file. Returns NULL if EOF was reached or error occured. See also functions IritPrsrSetPolyListCirc, IritPrsrSetFlattenObjects, and IritPrsrSetReadOneObject.

6.2.104 IritPrsrGetPrevObj (prsr_lib/iritprs2.c:1097)

previous element

linked lists

```
IPObjectStruct *IritPrsrGetPrevObj(IPObjectStruct *OList, IPObjectStruct *O)
```

OList: A list of objects.

O: For which the previous object in OList is pursuit.

Returns: Previous object to O in OList if found, NULL otherwise.

Description: Returns a pointer to previous object in OList to O.

6.2.105 IritPrsrGetPrevPoly (prsr_lib/iritprs2.c:1024)

previous element

linked lists

```
IPPolygonStruct *IritPrsrGetPrevPoly(IPPolygonStruct *PList,
                                     IPPolygonStruct *P)
```

PList: A list of polygons.

P: For which the previous polygon in PList is pursuit.

Returns: Previous polygon to P in PList if found, NULL otherwise.

Description: Returns a pointer to previous polygon in PList to P.

6.2.106 IritPrsrGetPrevVrtx (prsr_lib/iritprs2.c:945)

previous element

linked lists

```
IPVertexStruct *IritPrsrGetPrevVrtx(IPVertexStruct *VList, IPVertexStruct *V)
```

VList: A list of vertices.

V: For which the previous vertex in VList is pursuit.

Returns: Previous vertex to V in VList if found, NULL otherwise.

Description: Returns a pointer to previous vertex in VList to V.

6.2.107 IritPrsrInputUnGetC (prsr_lib/iritprs1.c:1076)

files

parser

```
void IritPrsrInputUnGetC(int Handler, char c)
```

Handler: A handler to the open stream.

c: Character to unget.

Description: Routine to unget a single character from input stream.

6.2.108 IritPrsrIsConvexPolygon (prsr_lib/iritprs2.c:455)

convexity

files

parser

```
int IritPrsrIsConvexPolygon(IPPolygonStruct *Pl)
```

Pl: To test for convexity.

Returns: TRUE if PL convex, FALSE otherwise.

Description: Routine to test if the given polygon is convex (by IRIT definition) or not. Algorithm: The polygon is convex iff the normals generated from cross products of two consecutive edges points to the same direction. The same direction is tested by a positive dot product.

6.2.109 IritPrsrObjListLen (prsr_lib/iritprs2.c:1196)

length

linked lists

```
int IritPrsrObjListLen(IPObjectStruct *O)
```

O: Object list to compute its length.

Returns: Number of elements in O list.

Description: Returns the length of a list of objects.

6.2.110 IritPrsrOpenDataFile (prsr_lib/iritprs1.c:80)

files

parser

```
int IritPrsrOpenDataFile(char *FileName, int Read, int Messages)
```

FileName: To try and open.

Read: If TRUE assume a read operation, otheriwse write.

Messages: Do we want error/warning messages to stderr?

Returns: A handler to the open file, -1 if error.

Description: Open a data file for read/write. Data file can be either Ascii IRIT data file or binary IRIT data file. A binary data file must have a ".bdt" (for Binary DaTa) file type. Under unix, file names with the psotfix ".Z" are assumed compressed and treated accordingly. See also functions IritPrsrSetPolyListCirc, IritPrsrSetFlattenObjects, and IritPrsrSetReadOneObject.

6.2.111 IritPrsrOpenStreamFromFile (prsr_lib/iritprs1.c:219)

```
int IritPrsrOpenStreamFromFile(FILE *f, int Read, int IsBinary, int IsPipe)
```

f: A handle to the open file.

Read: TRUE for reading from f, FALSE for writing to f.

IsBinary: Is it a binary file?

IsPipe: Is it a pipe?

Returns: A handle on the constructed stream.

Description: Converts an open file into a stream.

6.2.112 IritPrsrOpenStreamFromSocket (prsr_lib/iritprs1.c:248)

```
int IritPrsrOpenStreamFromSocket(int Soc, int Read, int IsBinary)
```

Soc: A handle to the open socket.

Read: TRUE for reading from f, FALSE for writing to f.

IsBinary: Is it a binary file?

Returns: A handle on the constructed stream.

Description: Converts an open socket into a stream.

6.2.113 IritPrsrParseError (prsr_lib/iritprs2.c:334)

error handling

files

parser

```
int IritPrsrParseError(int LineNum, char **ErrorMsg)
```

LineNum: Line number of error, in file/stream.

ErrorMsg: To be updated with latest error to have happened in parser.

Returns: TRUE if error occured since last call, FALSE otherwise.

Description: Returns TRUE if error has happend since last call to this function during data read or write, FALSE otherwise. If error, then ErrorMsg is updated to point on static str describing it.

6.2.114 IritPrsrPolyListLen (prsr_lib/iritprs2.c:1174)

length

linked lists

```
int IritPrsrPolyListLen(IPPolygonStruct *P)
```

P: Polygon list to compute its length.

Returns: Number of elements in P list.

Description: Returns the length of a list of polygons.

6.2.115 IritPrsrProcessFreeForm (prsr_lib/ip_procs.c:30)

conversion

```
IPObjectStruct *IritPrsrProcessFreeForm(IritPrsrFreeFormStruct *FreeForms)
```

FreeForms: Freeform geometry to process.

Returns: Processed freeform geometry. This function simply returns what it got.

Description: Default processor of read freeform geometry. This routine does not process the freeform geometry in any way. Other programs can, for example, convert the freeform shapes to polygons or polylines using the call back function or purge the freeform data if it is not supported.

6.2.116 IritPrsrProcessReadObject (prsr_lib/iritprs1.c:458)

files

parser

```
IPObjectStruct *IritPrsrProcessReadObject(IPObjectStruct *PObj)
```

PObj: Object to process.

Returns: Processed object, in place.

Description: Process a read object, in place, before returning it to the caller. List objects of zero or one elements are eliminated. Attributes are propagated throughout the hierarchy. If FlattenTree mode (see IritPrsrSetFlattenObjects) hierarchy is flattened out.

6.2.117 IritPrsrPropagateAttrs (prsr_lib/iritprs1.c:549)

attributes

files

parser

```
void IritPrsrPropagateAttrs(IPObjectStruct *PObj, IPAttributeStruct *Attrs)
```

PObj: To propagate down Attrs attributes.

Attrs: Attributes to propagate.

Description: Propagate attributes from list objects down into their elements.

6.2.118 IritPrsrPutBinObject (prsr_lib/iritprsb.c:834)

files

parser

```
void IritPrsrPutBinObject(int Handler, IPObjectStruct *PObj)
```

Handler: A handler to the open stream. *

PObj: Object to write.

Description: Routine to write one object to a given binary file, directly. Objects may be recursively defined, as lists of objects.

6.2.119 IritPrsrPutObjectToFile (prsr_lib/iritprs2.c:548)

files

```
void IritPrsrPutObjectToFile(FILE *f, IPObjectStruct *PObj)
```

f: Output stream.

PObj: Object to put on output stream.

Description: Routine to print the data from given object into given file FileName. If FileName is NULL or empty, print using IritPrsrPrintFunc. See function IritPrsrSetPrintFunc, IritPrsrSetFloatFormat.

6.2.120 IritPrsrPutObjectToHandler (prsr_lib/iritprs2.c:587)

files

```
void IritPrsrPutObjectToHandler(int Handler, IPObjectStruct *PObj)
```

Handler: A handler to the open stream.

PObj: Object to put on output stream.

Description: Routine to print the data from given object into given file FileName. If FileName is NULL or empty, print using IritPrsrPrintFunc. See function IritPrsrSetPrintFunc, IritPrsrSetFloatFormat.

6.2.121 IritPrsrReverseObjList (prsr_lib/iritprs2.c:215)

reverse

files

parser

```
*****************************************************************************/
IPObjectStruct *IritPrsrReverseObjList(IPObjectStruct *PObj)
```

PObj: A list of objects to reverse.

Returns: Reverse list of objects, in place.

Description: Reverses a list of objects, in place.

6.2.122 IritPrsrReverseVrtxList (prsr_lib/iritprs2.c:248)

reverse

files

parser

```
void IritPrsrReverseVrtxList(IPPolygonStruct *Pl)
```

Pl: A polygon to reverse its vertex list, in place.

Description: Reverses the vertex list of a given polygon. This is used mainly to reverse polygons such that cross product of consecutive edges which form a convex corner will point in the polygon normal direction.

6.2.123 IritPrsrSenseBinaryFile (prsr_lib/iritprs1.c:157)

```
int IritPrsrSenseBinaryFile(char *FileName)
```

FileName: File to sense.

Returns: TRUE if binary, FALSE if text.

Description: Senses if a given file (name) is a binary or a text file.

6.2.124 IritPrsrSetFlattenObjects (prsr_lib/iritprs1.c:506)

files

parser

```
int IritPrsrSetFlattenObjects(int Flatten)
```

Circ: If TRUE, list objects will be flattened out to a long linear list. If FALSE, read object will be unchanged.

Returns: Old value of flatten state.

Description: Controls the hierarchy flattening of a read object.

6.2.125 IritPrsrSetFlattenObjects (prsr_lib/iritprs1.c:530)

files

parser

```
void IritPrsrSetReadOneObject(int OneObject)
```

Circ: If TRUE, only next object will be read by IritPrsrGetObjectst. If FALSE, objects will be read until EOF is detected and placed in a linked list.

Description: Controls the way the Ascii parser handle multiple objects in a file.

6.2.126 IritPrsrSetFloatFormat (prsr_lib/iritprs2.c:1236)

files

```
void IritPrsrSetFloatFormat(char *FloatFormat)
```

FloatFormat: A printf style floating point printing format string.

Description: Sets the floating point printing format.

6.2.127 IritPrsrSetPolyListCirc (prsr_lib/iritprs1.c:487)

files

parser

```
void IritPrsrSetPolyListCirc(int Circ)
```

Circ: If TRUE, vertex lists of polygons will be circular. If FALSE, the lists will be NULL terminated.

Description: Controls vertex list in polygons. Do we want it circular?

6.2.128 IritPrsrSetPrintFunc (prsr_lib/iritprs2.c:1218)

files

```
void IritPrsrSetPrintFunc(IritPrsrPrintFuncType PrintFunc)
```

PrintFunc: A function that gets a single string it should print.

Description: Sets the printing function to call if needs to redirect printing.

6.2.129 IritPrsrSrvrExecAndConnect (prsr_lib/soc_srvr.c:565)

```
int IritPrsrSrvrExecAndConnect(char *Program,
                               int *PrgmInput,
                               int *PrgmOutput,
                               int IsBinary)
```

Program: Name of program to execute. Name can be NULL, in which the user is prompt to execute the program manually.

PrgmInput: A handler to the Program's input channel.

PrgmOutput: A handler to the Program's output channel.

IsBinary: If TRUE sets channels to binary, if FALSE channels are text.

Returns: TRUE, if succesful, FALSE otherwise.

Description: Executes the given program and connect to it io ports.

6.2.130 IritPrsrSrvrKillAndDisConnect (prsr_lib/soc_srvr.c:669)

```
int IritPrsrSrvrKillAndDisConnect(int Kill, int PrgmInput, int PrgmOutput)
```

Kill: If TRUE, send a KILL message to the other process.

PrgmInput: A handler to the Program's input channel.

PrgmOutput: A handler to the Program's output channel.

Returns: TRUE, if succesful, FALSE otherwise.

Description: Optionally kill and close channels to another process.

6.2.131 IritPrsrStderrObject (prsr_lib/iritprs2.c:527)

files

```
void IritPrsrStderrObject(IPObjectStruct *PObj)
```

PObj: To be put out to stderr.

Description: Routine to print the data from given object into stderr.

6.2.132 IritPrsrStdoutObject (prsr_lib/iritprs2.c:509)

files

```
void IritPrsrStdoutObject(IPObjectStruct *PObj)
```

PObj: To be put out to stdout.

Description: Routine to print the data from given object into stdout.

6.2.133 IritPrsrUpdatePolyPlane (prsr_lib/iritprs2.c:81)

files

parser

```
void IritPrsrUpdatePolyPlane(IPPolygonStruct *PPoly)
```

PPoly: To update its normal/plane equation.

Description: Routine to update the Plane equation of the given polygon by the order of the most robust three vertices of that polygon to define the normal.

6.2.134 IritPrsrUpdatePolyPlane2 (prsr_lib/iritprs2.c:157)

files

parser

```
void IritPrsrUpdatePolyPlane2(IPPolygonStruct *PPoly, VectorType Vin)
```

PPoly: To update its normal/plane equation.

Vin: A vertex to be considered in the inside, respective to PPoly.

Description: Routine to update the Plane equation of the given polygon such that the Vin vertex will be in the positive side of it.

6.2.135 IritPrsrUpdateVrtxNrml (prsr_lib/iritprs2.c:186)

files

parser

```
void IritPrsrUpdateVrtxNrml(IPPolygonStruct *PPoly, VectorType DefNrml)
```

PPoly: Polygon to update normal information.

DefNrml: Normal tp use in update.

Description: Routine to update all vertices in polygon to hold a default normal if have none already.

6.2.136 IritPrsrVrtxListLen (prsr_lib/iritprs2.c:1152)

length

linked lists

```
int IritPrsrVrtxListLen(IPVertexStruct *V)
```

V: Vertex list to compute its length.

Returns: Number of elements in V list.

Description: Returns the length of a list of vertices.

6.2.137 IritSetCurvesToCubicBzrTol (prsr_lib/ip_cnvrt.c:694)

approximation

```
void IritSetCurvesToCubicBzrTol(RealType Tolerance)
```

Tolerance: Of approximation to use.

Description: Sets the tolerance that is used by the Bezier to Cubic Bezier conversion routines IritCurvesToCubicBzrCrvs and IritSurfacesToCubicBzrCrvs

6.2.138 IritSurface2CtlMesh (prsr_lib/ip_cnvrt.c:240)

conversion

```
IPPolygonStruct *IritSurface2CtlMesh(CagdSrfStruct *Srf)
```

Srf: To extract its control mesh as a polylines.

Returns: A polylines object representing Srf's control mesh.

Description: Routine to convert a single surface's control mesh into a polylines object.

6.2.139 IritSurface2Polygons (prsr_lib/ip_cnvrt.c:292)

approximation

conversion

```
IPPolygonStruct *IritSurface2Polygons(CagdSrfStruct *Srf,
                                      int FourPerFlat,
                                      RealType FineNess,
                                      int ComputeUV,
                                      int Optimal)
```

Srf: To approximate using polygons.

FourPerFlat: If TRUE, four triangle per flat surface patch are created, otherwise only two.

FineNess: Fineness control on polygonal approximation. The larger this number is the finer the approximation becomes. 10 is a good compromise when Optimal is FALSE.

ComputeUV: Do we want UV parameter values with the vertices of the triangles?

Optimal: If FALSE (0) then parametric space of Srf is sampled

uniformely. Otherwise: If the tenths digit of Optimal is equal to 1. Subdivide the surface alternatively in U and V. 2. Subdivide the surface in the direction that minimizes the error of the approximation If the units digit of Optimal is equal to 0. No real error analysis. The fastest way. 1. Use curvature surface analysis to decide where to subdivide. Much slower than 0. 2. Use a bilinear surface fit to estimate error. Somewhat slower than 0. 3. Combine 1 and 2 for a bilinar fit error measure with curvature analysis. Very slow.

Returns: Resulting polygons that approximates Srf.

Description: Routine to approximate a single surface by polygons. FourPerFlat cause four triangles from each flat patch, two otherwise. FineNess is an estimate on number of polygons for each side of mesh if Optimal = 0, otherwise a FineNess curvature control on the maximal distance between the surface and its polygonal approximation. Optimal is a two digits number where the units hold the subdivision strategy and the tens the termination criteria.

6.2.140 IritSurface2Polylines (prsr_lib/ip_cnvrt.c:178)

conversion

approximation

```
IPPolygonStruct *IritSurface2Polylines(CagdSrfStruct *Srf,
                                       int NumOfIsolines[2],
                                       int SamplesPerCurve,
                                       int Optimal)
```

Srf: To approximate as a polyline .

NumOfIsolines: Number of isocurves to extract, in each direction.

SamplesPerCurve: Number of samples to compute on the polyline.

Optimal: If FALSE samples are uniform in parametric space, otherwise attempt is made to sample optimally.

Returns: A polylines object approximating Srf.

Description: Routine to convert a single surface into polylines with SamplesPerCurve samples, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it. If Optimal, attempt is made to optimally distribute the sampled points.

6.2.141 IritSurfacesToCubicBzrCrvs (prsr_lib/ip_cnvrt.c:796)

conversion

approximation

```
CagdCrvStruct *IritSurfacesToCubicBzrCrvs(CagdSrfStruct *Srfs,
                                          IPPolygonStruct **CtlMeshes,
                                          CagdBType DrawSurface,
                                          CagdBType DrawMesh,
                                          int NumOfIsolines[2],
                                          CagdRType MaxArcLen)
```

Srfs: To approximate as cubic Bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawSurface: Do we want to draw the surfaces?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function BzrApproxBzrCrvAsCubics.

Returns: The cubic Bezier approximation, or NULL if DrawSurface is FALSE.

Description: Approximates an arbitrary list of surfaces into cubic Beziers curves.

6.2.142 IritTrimSrf2CtlMesh (prsr_lib/ip_cnvrt.c:503)

conversion

```
IPPolygonStruct *IritTrimSrf2CtlMesh(TrimSrfStruct *TrimSrf)
```

TrimSrf: To extract its control mesh as a polylines.

Returns: A polylines object representing TrimSrf's control mesh.

Description: Routine to convert a single trimmed surface's control mesh into a polylines object.

6.2.143 IritTrimSrf2Polylines (prsr_lib/ip_cnvrt.c:436)

conversion

approximation

```
IPPolygonStruct *IritTrimSrf2Polylines(TrimSrfStruct *TrimSrf,
                                       int NumOfIsolines[2],
                                       int SamplesPerCurve,
                                       int Optimal,
                                       int TrimmingCurves,
                                       int IsoParamCurves)
```

TrimSrf: To approximate as a polyline .

NumOfIsolines: Number of isocurves to extract, in each direction.

SamplesPerCurve: Number of samples to compute on the polyline.

Optimal: If FALSE samples are uniform in parametric space, otherwise attempt is made to sample optimally.

TrimmingCurves: Do we want the trimming curves as well.

IsoParamCurves: Do we want trimmed isoparametric curves.

Returns: A polylines object approximating TrimSrf.

Description: Routine to convert a single trimmed surface into polylines with SamplesPerCurve samples, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it. If Optimal, attempt is made to optimally distribute the sampled points.

6.2.144 IritTrimSrfsToCubicBzrCrvs (prsr_lib/ip_cnvrt.c:861)

conversion

approximation

```
CagdCrvStruct *IritTrimSrfsToCubicBzrCrvs(TrimSrfStruct *TrimSrfs,
                                          IPPolygonStruct **CtlMeshes,
                                          CagdBType DrawTrimSrf,
                                          CagdBType DrawMesh,
                                          int NumOfIsolines[2],
                                          CagdRType MaxArcLen)
```

TrimSrfs: To approximate as cubic Bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawTrimSrf: Do we want to draw the surfaces?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function BzrApproxBzrCrvAsCubics.

Returns: The cubic Bezier approximation, or NULL if DrawSurface is FALSE.

Description: Approximates an arbitrary list of trimmed surfaces into cubic Beziers curves. Only isoparametric curves are extarcted (no trimming curves).

6.2.145 IritTrivar2CtlMesh (prsr_lib/ip_cnvrt.c:671)

conversion

```
IPPolygonStruct *IritTrivar2CtlMesh(TrivTVStruct *Trivar)
```

Trivar: To extract its control mesh as a polylines.

Returns: A polylines object representing Trivar's control mesh.

Description: Routine to convert a single trivariate's control mesh into a polylines object.

6.2.146 IritTrivar2Polygons (prsr_lib/ip_cnvrt.c:533)

approximation

conversion

```
IPPolygonStruct *IritTrivar2Polygons(TrivTVStruct *Trivar,
                                     int FourPerFlat,
                                     RealType FineNess,
                                     int ComputeUV,
                                     int Optimal)
```

Trivar: To approximate using polygons.

FourPerFlat: See IritSurface2Polygons.

FineNess: See IritSurface2Polygons.

ComputeUV: See IritSurface2Polygons.

Optimal: See IritSurface2Polygons.

Returns: Resulting polygons that approximates Srf.

Description: Routine to approximate a single trivariate by polygons. Six faces of the trivariate are extracted as six surfaces that are displayed.

6.2.147 IritTrivar2Polylines (prsr_lib/ip_cnvrt.c:591)

conversion

approximation

```
IPPolygonStruct *IritTrivar2Polylines(TrivTVStruct *Trivar,
                                      int NumOfIsolines[3],
                                      int SamplesPerCurve,
                                      int Optimal)
```

Trivar: To approximate as a polyline .

NumOfIsolines: Number of isocurves to extract, in each direction.

SamplesPerCurve: Number of samples to compute on the polyline.

Optimal: If FALSE samples are uniform in parametric space, otherwise attempt is made to sample optimally.

Returns: A polylines object approximating Trivar.

Description: Routine to convert a single trivariate function into polylines with SamplesPerCurve samples, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it. If Optimal, attempt is made to optimally distribute the sampled points.

6.2.148 IritTrivarToCubicBzrCrvs (prsr_lib/ip_cnvrt.c:930)

conversion

approximation

```
CagdCrvStruct *IritTrivarToCubicBzrCrvs(TrivTVStruct *Trivar,
                                        IPPolygonStruct **CtlMeshes,
                                        CagdBType DrawSurface,
                                        CagdBType DrawMesh,
                                        int NumOfIsolines[2],
                                        CagdRType MaxArcLen)
```

Trivar: To approximate as a set of cubic bezier curves.

CtlMeshes: If we want control meshes as well (DrawMesh == TRUE) they will be placed herein.

DrawTrivar: Do we want to draw the trivariate function?

DrawMesh: Do we want to draw the control meshes?

NumOfIsolines: Number of isocurves to extract, in each direction.

MaxArcLen: Tolerance for cubic Bezier approximation. See function BzrApproxBzrCrvAsCubics.

Returns: A list of curves approximating Trivar.

Description: Routine to convert a single trivariate function into polylines with SamplesPerCurve samples, NumOfIsolines isolines into a polylines object. If NumOfIsolines has negative value, its absolute value is heuristically used to derive a new NumOfIsolines number for it. If Optimal, attempt is made to optimally distribute the sampled points.

6.2.149 ListObjectFind (prsr_lib/allocate.c:482)

linked lists

find

```
int ListObjectFind(IPObjectStruct *PObjList, IPObjectStruct *PObj)
```

PObjList: To search for PObj in.

PObj: The element to search in PObjList.

Returns: TRUE if PObj was found in PObjList, FALSE otherwise.

Description: Returns TRUE if PObj is an object in list PObjList or in a sublist of PObjList, recursively.

6.2.150 ListObjectGet (prsr_lib/allocate.c:544)

lists

find

```
IPObjectStruct *ListObjectGet(IPObjectStruct *PObj, int Index)
```

PObj: A list object to extract one object from.

Index: Index of object to extractfrom PObj.

Returns: Index object in list PObj, or NULL if no such thing.

Description: Returns the object number Index in list of PObjList object.

6.2.151 ListObjectInsert (prsr_lib/allocate.c:518)

lists

insert

```
void ListObjectInsert(IPObjectStruct *PObj,
                      int Index,
                      IPObjectStruct *PObjItem)
```

PObj: A list of objects to insert PObjItem into.

Index: Index where PObjItem should enter PObj.

PObjItem: Eleent to insert into the list PObj.

Description: Insert an object PObjItem at index Index into a list of object,PObj.

6.2.152 ListObjectLength (prsr_lib/allocate.c:452)

linked lists

length

```
int ListObjectLength(IPObjectStruct *PObj)
```

PObj: A list of objects to find its length.

Returns: Resulting length of list PObj.

Description: Returns the length of a list, given a list of objects.

6.2.153 ReallocNewTypeObject (prsr_lib/allocate.c:1229)

allocation

```
void ReallocNewTypeObject(IPObjectStruct *PObj, IPObjStructType ObjType)
```

PObj: Object to reallocated as a new object of type ObjType.

ObjType: New type for object PObj.

Description: Routine to reallocate as necessary and object to a new object type.

6.2.154 SocClientCloseSocket (prsr_lib/soc_clnt.c:463)

ipc

```
void SocClientCloseSocket(int Handler)
```

Handler: The socket info handler index. *

Description: Close the client's socket.

6.2.155 SocClientCreateSocket (prsr_lib/soc_clnt.c:286)

ipc

```
int SocClientCreateSocket(int BinaryIPC, int Read)
```

BinaryIPC: Do we want to communicate text or binary?

Read: Is this socket for read (TRUE) or write (FALSE).

Returns: Non negative handler if successful, -1 otherwise.

Description: Opens the client's socket. Returns handler if succesful.

6.2.156 SocEchoInput (prsr_lib/soc_clnt.c:89)

ipc

```
void SocEchoInput(int Handler, int EchoInput)
```

Handler: The socket info handler index.

EchoInput: TRUE to echo every character read in.

Description: Sets echo printing of read input.

6.2.157 SocReadCharNonBlock (prsr_lib/soc_clnt.c:108)

ipc

```
int SocReadCharNonBlock(int Handler)
```

Handler: The socket info handler index.

Returns: Read character or EOF if none found.

Description: Non blocking read of a single character. Returns EOF if no data is found.

6.2.158 SocReadLineNonBlock (prsr_lib/soc_clnt.c:242)

ipc

```
char *SocReadLineNonBlock(int Handler)
```

Handler: The socket info handler index.

Returns: Read line, or NULL if unavailable.

Description: Non blocking read of a single line. Returns NULL if no line is available.

6.2.159 SocReadOneObject (prsr_lib/soc_clnt.c:502)

ipc

```
IPObjectStruct *SocReadOneObject(int Handler)
```

Handler: The socket info handler index. *

Returns: An object if read one, NULL otherwise.

Description: Attempts to read (non blocking) an object from socket. If read is successful the object is returned, otherwise NULL is returned.

6.2.160 SocServerAcceptConnection (prsr_lib/soc_srvr.c:355)

ipc

```
int SocServerAcceptConnection(int Handler)
```

Handler: The socket info handler index.

Returns: TRUE if succesful, FALSE otherwise.

Description: Accepts a client socket connection. This function blocks until a connection with a client is established.

6.2.161 SocServerActive (prsr_lib/soc_srvr.c:470)

ipc

```
int SocServerActive(int Handler)
```

Handler: The socket info handler index.

Returns: TRUE if connection to client exists.

Description: Returns TRUE if connection is alive and active.

6.2.162 SocServerCloseSocket (prsr_lib/soc_srvr.c:420)

ipc

```
void SocServerCloseSocket(int Handler)
```

Handler: The socket info handler index.

Description: Closes a server socket/connection.

6.2.163 SocServerCreateSocket (prsr_lib/soc_srvr.c:208)

ipc

```
int SocServerCreateSocket(int BinaryIPC, int Read)
```

BinaryIPC: Do we want to communicate text or binary?

Read: Is this socket for read (TRUE) or write (FALSE).

Returns: Non negative handler if successful, -1 otherwise.

Description: Creates a server socket. Returns handler if successful. Also sets the IRIT_SERVER_PORT environment variable to the allocated socket.

6.2.164 SocWriteChar (prsr_lib/soc_srvr.c:99)

ipc

```
void SocWriteChar(int Handler, char c)
```

Handler: The socket info handler index.

c: Character to write.

Description: Writes a single char to client's socket.

6.2.165 SocWriteLine (prsr_lib/soc_srvr.c:150)

ipc

```
void SocWriteLine(int Handler, char *Line, int LineLen)
```

Handler: The socket info handler index.

Line: Line to write.

LineLen: Length of line to write.

Description: Writes a single line of line length characters.

6.2.166 SocWriteOneObject (prsr_lib/soc_srvr.c:525)

ipc

```
void SocWriteOneObject(int Handler, IPObjectStruct *PObj)
```

Handler: The socket info handler index.

PObj: Object to write to the client's socket.

Description: Attempts to write an object to a socket.

6.2.167 TrimWriteTrimmedSrfToFile (prsr_lib/trim_wrt.c:32)

files

write

```
int TrimWriteTrimmedSrfToFile(TrimSrfStruct *TrimSrfs,
                              char *FileName,
                              int Indent,
                              char *Comment,
                              char **ErrStr)
```

TrimSrfs: To be saved in stream.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write a trimmed surface to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.168 TrimWriteTrimmedSrfToFile2 (prsr_lib/trim_wrt.c:74)

files

write

stream

```
int TrimWriteTrimmedSrfToFile2(TrimSrfStruct *TrimSrfs,
                               int Handler,
                               int Indent,
                               char *Comment,
                               char **ErrStr)
```

TrimSrfs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trimmed surface(s) to the given stream. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.169 TrivBspTVReadFromFile (prsr_lib/trivread.c:314)

files

read

trivariates

```
TrivTVStruct *TrivBspTVReadFromFile(char *FileName,
                                    char **ErrStr,
                                    int *ErrLine)
```

FileName: To read the trivariate from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trivariate, or NULL if an error occured.

Description: Reads from a file Bspline trivariates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.170 TrivBspTVWriteToFile2 (prsr_lib/triv_wrt.c:330)

files

write

trivariates

```
int TrivBspTVWriteToFile2(TrivTVStruct *TVs,
                          int Handler,
                          int Indent,
                          char *Comment,
                          char **ErrStr)
```

TVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bspline trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.171 TrivBzrTVReadFromFile (prsr_lib/trivread.c:124)

files

read

trivariates

```
TrivTVStruct *TrivBzrTVReadFromFile(char *FileName,
                                    char **ErrStr,
                                    int *ErrLine)
```

FileName: To read the trivariate from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trivariate, or NULL if an error occured.

Description: Reads from a file Bezier trivariates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.172 TrivBzrTVReadFromFile2 (prsr_lib/trivread.c:187)

files

read

trivariates

```
TrivTVStruct *TrivBzrTVReadFromFile2(int Handler,
                                     CagdBType NameWasRead,
                                     char **ErrStr,
                                     int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trivariate, or NULL if an error occured.

Description: Reads from a file a Bezier trivariate. If NameWasRead is TRUE, it is assumed prefix "[TRIVAR BEZIER" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of trivariate. If no error is detected *ErrStr is set to NULL.

6.2.173 TrivBzrTVReadFromFile2 (prsr_lib/trivread.c:377)

files read trivariates

```
TrivTVStruct *TrivBspTVReadFromFile2(int Handler,
                                     CagdBType NameWasRead,
                                     char **ErrStr,
                                     int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trivariate, or NULL if an error occured.

Description: Reads from a file a Bezier trivariate. If NameWasRead is TRUE, it is assumed prefix "[TRIVAR BSPLINE" has already been read. This is useful for a global parser which invokes this routine to read from a file several times as a parent controller. For exactly this reason, the given file descriptor is NOT closed in the end. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of trivariate. If no error is detected *ErrStr is set to NULL.

6.2.174 TrivBzrTVWriteToFile (prsr_lib/triv_wrt.c:288)

files write trivariates

```
int TrivBspTVWriteToFile(TrivTVStruct *TVs,
                         char *FileName,
                         int Indent,
                         char *Comment,
                         char **ErrStr)
```

TVs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bspline trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.175 TrivBzrTVWriteToFile (prsr_lib/triv_wrt.c:167)

files write trivariates

```
int TrivBzrTVWriteToFile(TrivTVStruct *TVs,
                         char *FileName,
                         int Indent,
                         char *Comment,
                         char **ErrStr)
```

TVs: To be saved in file.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bezier trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.176 TrivBzrTVWriteToFile2 (prsr_lib/triv_wrt.c:209)

files

write

trivariates

```
int TrivBzrTVWriteToFile2(TrivTVStruct *TVs,
                          int Handler,
                          int Indent,
                          char *Comment,
                          char **ErrStr)
```

TVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write Bezier trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.177 TrivReadTrimmedSrfFromFile2 (prsr_lib/trimread.c:89)

files

read

trivariates

```
TrimSrfStruct *TrimReadTrimmedSrfFromFile2(int Handler,
                                           CagdBType NameWasRead,
                                           char **ErrStr,
                                           int *ErrLine)
```

Handler: A handler to the open stream.

NameWasRead: If FALSE, also reads the TRIVAR BEZIER prefix.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trimmed surface, or NULL if an error occured.

Description: Reads from a file a trimmed surface. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of trivariate. Assumes the [TRIMSRF prefix was read if NameWasRead. If no error is detected *ErrStr is set to NULL.

6.2.178 TrivTVReadFromFile (prsr_lib/trivread.c:28)

files

read

trivariates

```
TrivTVStruct *TrivTVReadFromFile(char *FileName, char **ErrStr, int *ErrLine)
```

FileName: To read the trivariate from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trivariate, or NULL if an error occured.

Description: Reads from a file trivariates. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.179 TrivTVReadFromFile (prsr_lib/trimread.c:30)

files

read

trivariates

```
TrimSrfStruct *TrimReadTrimmedSrfFromFile(char *FileName,
                                          char **ErrStr,
                                          int *ErrLine)
```

FileName: To read the trimmed surface from.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trimmed surface, or NULL if an error occured.

Description: Reads from a file trimmed surfaces. If error is detected in reading the file, ErrStr is set to a string describing the error and Line to the line it occured in file. If no error is detected *ErrStr = NULL.

6.2.180 TrivTVReadFromFile2 (prsr_lib/trivread.c:88)

files

read

trivariates

```
TrivTVStruct *TrivTVReadFromFile2(int Handler, char **ErrStr, int *ErrLine)
```

Handler: A handler to the open stream.

ErrStr: Will be initialized if an error has occured.

ErrLine: Line number in file FileName of the error, if occured.

Returns: The read trivariate, or NULL if an error occured.

Description: Reads from a file a trivariate. If error is found in reading the file, ErrStr is set to a string describing it and ErrLine to line it occured in file relative to begining of trivariate. If no error is detected *ErrStr is set to NULL.

6.2.181 TrivTVWriteToFile (prsr_lib/triv_wrt.c:32)

files

write

trivariates

```
int TrivTVWriteToFile(TrivTVStruct *TVs,
                      char *FileName,
                      int Indent,
                      char *Comment,
                      char **ErrStr)
```

TVs: To be saved in file f.

FileName: File name where output should go to.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.182 TrivTVWriteToFile2 (prsr_lib/triv_wrt.c:83)

files

write

trivariates

```
int TrivTVWriteToFile2(TrivTVStruct *TVs,
                       int Handler,
                       int Indent,
                       char *Comment,
                       char **ErrStr)
```

TVs: To be saved in stream.

Handler: A handler to the open stream.

Indent: Column in which all printing starts at.

Comment: Optional comment to describe the geometry.

ErrStr: If failed, ErrStr will be set to describe the problem.

Returns: TRUE if succesful, FALSE otherwise.

Description: Generic routine to write trivariates to the given file. If Comment is NULL, no comment is printed, if "" only internal comment.

6.2.183 _IPGetCloseParenToken (prsr_lib/iritprs1.c:779)

```
void _IPGetCloseParenToken(int Handler)
```

Handler: A handler to the open stream.

Description: Routine to get close paren token from FILE f. This function invokes the parser's abort routine, if no close paren.

6.2.184 _IPGetToken (prsr_lib/iritprs1.c:1230)

```
IPTokenType _IPGetToken(int Handler, char *StringToken)
```

Handler: A handler to the open stream.

StringToken: String token will be placed herein.

Returns: Token as a numeral.

Description: Routine to get the next token out of the input file f as token number. StringToken must be allocated before calling this routine!

6.2.185 _IPParserAbort (prsr_lib/iritprs2.c:310)

error handling

files

parser

```
void _IPParserAbort(IritPrsrErrType ErrNum, char *Msg)
```

ErrNum: Type of error that had occured.

Msg: A message to accompany the error number.

Description: Routine to abort parsing operation and save error reported. See also function IritPrsrParseError.

6.2.186 _IPSkipToCloseParenToken (prsr_lib/iritprs1.c:800)

```
int _IPSkipToCloseParenToken(int Handler)
```

Handler: A handler to the open stream.

Returns: TRUE, if found close paren.

Description: Routine to skip to the next closed parenthesis.

6.2.187 _IPUnGetToken (prsr_lib/iritprs1.c:1053)

```
void _IPUnGetToken(int Handler, char *StringToken)
```

Handler: A handler to the open stream.

StringToken: Token to unget

Description: Routine to unget one token (on stack of UNGET_STACK_SIZE levels!)

Chapter 7

Symbolic Library, symb_lib

7.1 General Information

This library provides a rich set of function to symbolically manipulate freeform curves and surfaces. This library heavily depends on the cagd library. Functions are provided to low level add, subtract, and multiply freeform curves and surfaces, to compute fields such as curvature, and to extract singular points such as extremums, zeros, and inflections. High level tools to metamorph curves and surfaces, to compute layout (prisa) of freeform surfaces, to compute offset approximations of curves and surfaces, and to compose curves and surfaces are also provided.

The interface of the library is defined in *include/symb_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **SymbFatalError**.

Globals in this library have a prefix of **Symb** for general symbolic routines. Prefix of **Bzr** is used for Bezier routines, and prefix of **Bsp** for Bspline specific routines.

7.2 Library Functions

7.2.1 BspCrvDeriveRational (symb_lib/bsp_sym.c:359)

derivatives

```
CagdCrvStruct *BspCrvDeriveRational(CagdCrvStruct *Crv)
```

Crv: Bspline curve to differentiate.

Returns: Differentiated rational Bspline curve.

Description: Given a rational Bspline curve - computes its derivative curve (Hodograph) using the quotient rule for differentiation.

7.2.2 BspCrvMult (symb_lib/bsp_sym.c:56)

product

```
CagdCrvStruct *BspCrvMult(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curves to multiply.

Returns: The product Crv1 * Crv2 coordinatewise.

Description: Given two Bspline curves - multiply them coordinatewise. The two curves are promoted to same point type before the multiplication can take place. See also BspMultInterpFlag.

7.2.3 BspMultInterpFlag (symb_lib/bsp_sym.c:32)

product

```
int BspMultInterpFlag(int BspMultUsingInter)
```

BspMultUsingInter: If TRUE, Bspline product iscomputed by setting an interpolation problem. Otherwise, by decomposing the Bspline geometry to Bezier geometry.

Returns: Previous setting.

Description: Sets method of Bspline product computation.

7.2.4 BspSrfDeriveRational (symb_lib/bsp_sym.c:453)

derivatives

```
CagdSrfStruct *BspSrfDeriveRational(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: Bspline surface to differentiate.

Dir: Direction of Differentiation. Either U or V.

Returns: Differentiated rational Bspline surface.

Description: Given a rational Bspline surface - computes its derivative surface in direction Dir, using the quotient rule for differentiation.

7.2.5 BspSrfMult (symb_lib/bsp_sym.c:221)

product

```
CagdSrfStruct *BspSrfMult(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: The two surfaces to multiply.

Returns: The product Srf1 * Srf2 coordinatewise.

Description: Given two Bspline surfaces - multiply them coordinatewise. The two surfaces are promoted to same point type before multiplication can take place. See also BspMultInterpFlag.

7.2.6 BzrApproxBzrCrvAsCubicPoly (symb_lib/bzr_sym.c:482)

approximation

conversion

```
CagdCrvStruct *BzrApproxBzrCrvAsCubicPoly(CagdCrvStruct *Crv, CagdRType Tol2)
```

Crv: To approximate using cubic Bezier curves.

Tol2: Accuracy control.

Returns: List of cubic Bezier curves approximating Crv.

Description: Given a Bezier curve with order larger than cubic, approximate it using piecewise cubic curves. A C^1 continuous approximation is computed so that the approximation is at most sqrt(Tol2) away from the given curve. Input curve can be rational, although output is always polynomial.

7.2.7 BzrApproxBzrCrvAsCubics (symb_lib/bzr_sym.c:407)

conversion

approximation

```
CagdCrvStruct *BzrApproxBzrCrvAsCubics(CagdCrvStruct *Crv,
                                       CagdRType Tol,
                                       CagdRType MaxLen,
                                       CagdBType NoRational)
```

Crv: To approximate using cubic Bezier polynomials.

Tol: Accuracy control.

MaxLen: Maximum arc length of curve.

NoRational: Do we want to approximate rational curves as well?

Returns: A list of cubic Bezier polynomials approximating Crv.

Description: Given a Bezier curve - convert it to (possibly) piecewise cubic. If the curve is

1. A cubic - a copy if it is returned.
2. Lower than cubic - a degree raised (to a cubic) curve is returned.
3. Higher than cubic - a C^1 continuous piecewise cubic approximation is computed for Crv.

In case 3 a list of polynomial cubic curves is returned. Tol is then used for the distance tolerance error measure for the approximation. If, however, NoRational is set, rational curves of any order will also be approximated using cubic polynomials. Furthermore if the total length of control polygon is more than MaxLen, the curve is subdivided until this is not the case.

7.2.8 BzrComposeCrvCrv (symb_lib/composit.c:230)

composition

```
CagdCrvStruct *BzrComposeCrvCrv(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curve to compose together.

Returns: The composed curve.

Description: Given two Bezier curves, Crv1 and Crv2, computes their composition Crv1(Crv2(t)). Crv2 must be a scalar curve completely contained in Crv1's parametric domain. See: "Freeform surfcae analysis using a hybrid of symbolic and numeric computation" by Gershon Elber, PhD thesis, University of Utah, 1992.

7.2.9 BzrComposeSrfCrv (symb_lib/composit.c:426)

composition

```
CagdCrvStruct *BzrComposeSrfCrv(CagdSrfStruct *Srf, CagdCrvStruct *Crv)
```

Srf, Crv: The curve and surface to compose.

Returns: The resulting composition.

Description: Given a Bezier curve Crv and a Bezier surface Srf, computes their coomposition Srf(Crv(t)). Crv must be a two dimensional curve completely contained in the parametric domain of Srf. See: "Freeform surfcae analysis using a hybrid of symbolic and numeric computation" by Gershon Elber, PhD thesis, University of Utah, 1992.

7.2.10 BzrCrvDeriveRational (symb_lib/bzr_sym.c:209)

derivatives

```
CagdCrvStruct *BzrCrvDeriveRational(CagdCrvStruct *Crv)
```

Crv: Bezier curve to differentiate.

Returns: Differentiated rational Bezier curve.

Description: Given a rational Bezier curve - computes its derivative curve (Hodograph) using the quotient rule for differentiation.

7.2.11 BzrCrvMult (symb_lib/bzr_sym.c:24)

product

```
CagdCrvStruct *BzrCrvMult(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curves to multiply.

Returns: The product Crv1 * Crv2 coordinatewise.

Description: Given two Bezier curves - multiply them coordinatewise. The two curves are promoted to same point type before the multiplication can take place.

7.2.12 BzrCrvMultList (symb_lib/bzr_sym.c:88)

product

```
CagdCrvStruct *BzrCrvMultList(CagdCrvStruct *Crv1Lst, CagdCrvStruct *Crv2Lst)
```

Crv1Lst: First list of Bezier curves to multiply.

Crv2Lst: Second list of Bezier curves to multiply.

Returns: A list of product curves

Description: Given two Bezier curve lists - multiply them one at a time. Return a Bezier curve lists representing their products.

7.2.13 BzrSrfDeriveRational (symb_lib/bzr_sym.c:301)

derivatives

```
CagdSrfStruct *BzrSrfDeriveRational(CagdSrfStruct *Srf, CagdSrfDirType Dir)
```

Srf: Bezier surface to differentiate.

Dir: Direction of Differentiation. Either U or V.

Returns: Differentiated rational Bezier surface.

Description: Given a rational Bezier surface - computes its derivative surface in direction Dir, using the quotient rule for differentiation.

7.2.14 BzrSrfMult (symb_lib/bzr_sym.c:126)

product

```
CagdSrfStruct *BzrSrfMult(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: The two surfaces to multiply.

Returns: The product Srf1 * Srf2 coordinatewise.

Description: Given two Bezier surfaces - multiply them coordinatewise. The two surfaces are promoted to same point type before multiplication can take place.

7.2.15 SymbAdapIsoExtract (symb_lib/adap_iso.c:101)

adaptive isocurves

```
CagdCrvStruct *SymbAdapIsoExtract(CagdSrfStruct *Srf,
                                  CagdSrfStruct *NSrf,
                                  SymbAdapIsoDistSqrFuncType AdapIsoDistFunc,
                                  CagdSrfDirType Dir,
                                  CagdRType Eps,
                                  CagdBType FullIso,
                                  CagdBType SinglePath)
```

Srf: To compute adaptive isocurve coverage form

NSrf: Normal vector field defining the normals of Srf.

AdapIsoDistFunc: Optional function to invoke with the two adjacent isoparametric curves of the coverage to evaluate the distance between them.

Dir: Direction of adaptive isocurve extraction. Either U or V.

Eps: Tolerance of adaptive isocurve cuverage. For every point P on Srf there will be a point Q in one of the extracted isocurves such the |P - Q| < Eps.

FullIso: Do we want all isocurves to span the entire domain?

SinglePath: Do we want a single curve through them all?

Returns: A list of curves representing the computed adaptive isocurve coverage for surface Srf. If normal field, NSrf, is prescribed, normal curves are concatenated alternatingly in this list.

Description: Extracts a valid coverage set of isolines from the given surface in the given direction and epsilon. If FullIso is TRUE, all extracted isocurves are spanning the entire parametric domain. If SinglePath is TRUE, the entire coverage is going to be a single curve. If NSrf != NULL, every second curve will be a vector field curve representing the unnormalized normal for the previous Euclidean curve. This mode disable the SinglePath mode. See also function SymbSetAdapIsoExtractMinLevel.

7.2.16 SymbAllPrisaSrfs (symb_lib/prisa.c:52)

layout

prisa

```
CagdSrfStruct *SymbAllPrisaSrfs(CagdSrfStruct *Srfs,
                                int SamplesPerCurve,
                                CagdRType Epsilon,
                                CagdSrfDirType Dir,
                                CagdVType Space)
```

Srfs: To approximate and faltten out.

SamplesPerCurve: During the approximation of a ruled surface as a developable surface.

Epsilon: Accuracy control for the piecewise ruled surface approximation.

Dir: Direction of ruled/developable surface approximation. Either U or V.

Space: A vector in the XY plane to denote the amount of translation from one flattened out surface to the next.

Returns: A list of planar surfaces denoting the layout (prisa) of the given Srfs to the accuracy requested.

Description: Computes a piecewise ruled surface approximation to a given set of surfaces with given Epsilon, and lay them out "nicely" onto the XY plane, by approximating each ruled surface as a developable surface with SamplesPerCurve samples. Dir controls the direction of ruled approximation, SpaceScale and Offset controls the placement of the different planar pieces. Prisa is the hebrew word for the process of flattening out a three dimensional surface. I have still to find an english word for it.

7.2.17 SymbComposeCrvCrv (symb_lib/composit.c:35)

composition

```
CagdCrvStruct *SymbComposeCrvCrv(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curve to compose together.

Returns: The composed curve.

Description: Given two curves, Crv1 and Crv2, computes the composition Crv1(Crv2(t)). Crv2 must be a scalar curve completely contained in Crv1's parametric domain.

7.2.18 SymbComposeSrfCrv (symb_lib/composit.c:313)

composition

```
CagdCrvStruct *SymbComposeSrfCrv(CagdSrfStruct *Srf, CagdCrvStruct *Crv)
```

Srf, Crv: The curve and surface to compose.

Returns: The resulting composition.

Description: Given a curve Crv and surface Srf, computes the composition Srf(Crv(t)). Crv must be a two dimensional curve completely contained in the parametric domain of Srf.

7.2.19 SymbCrv2DCurvatureSign (symb_lib/curvatur.c:333)

curvature

```
CagdCrvStruct *SymbCrv2DCurvatureSign(CagdCrvStruct *Crv)
```

Crv:

Returns:

Description: Computes a scalar curve representing the curvature sign of a planar curve. The given curve is assumed to be planar and only its x and y coordinates are considered. Then the curvature sign is equal to

```
     .  ..    .  ..
s =  X  Y  -  Y  X
```

7.2.20 SymbCrv2DCurvatureSqr (symb_lib/curvatur.c:34)

curvature

```
CagdCrvStruct *SymbCrv2DCurvatureSqr(CagdCrvStruct *Crv)
```

Crv: To compute the square of teh curvature field for.

Returns: The square of the curvature field of Crv.

Description: Computes a scalar curve representing the curvature of a planar curve. The given curve is assumed to be planar and only its x and y coordinates are considered. Then the curvature k is equal to

```
     .  ..    .  ..
     X  Y  -  Y  X
k =  -------------
        .2    .2  3/2
     (  X  +  Y   )
```

Since we cannot represent k because of the square root, we compute and represent k^2.

7.2.21 SymbCrv2DExtremCrvtrPts (symb_lib/curvatur.c:491)

curvature

```
CagdPtStruct *SymbCrv2DExtremCrvtrPts(CagdCrvStruct *Crv, CagdRType Epsilon)
```

Crv: To find all int extrem curvature locations.

Epsilon: Accuracy control.

Returns: A list of parameter values on Crv that have extrem curvature values.

Description: Given a planar curve, finds all its extreme curvatrue points by finding the set of extreme locations on the curvature function of Crv.

7.2.22 SymbCrv2DInflectionPts (symb_lib/curvatur.c:463)

curvature

inflection points

```
CagdPtStruct *SymbCrv2DInflectionPts(CagdCrvStruct *Crv, CagdRType Epsilon)
```

Crv: To find all its inflection points.

Epsilon: Accuracy control.

Returns: A list of parameter values on Crv that are inflection points.

Description: Given a planar curve, finds all its inflection points by finding the zero set of the sign of the curvature function of the curve.

7.2.23 SymbCrv2Polyline (symb_lib/symbpoly.c:193)

piecewise linear approximation

polyline

```
CagdPolylineStruct *SymbCrv2Polyline(CagdCrvStruct *Crv,
                                     int SamplesPerCurve,
                                     CagdBType Optimal,
                                     CagdBType OptiLin)
```

Crv: To approximate as a polyline.

SamplesPerCurve: Number of samples to approximate with.

Optimal: If TRUE, yse optimal approximation of isocurves. Otherwise, the curve is sampled using equally spaced samples in parametric space.

OptiLin: If TRUE, optimize linear curves.

Returns: A polyline representing the piecewise linear approximation from, or NULL in case of an error.

Description: Routine to approx. a single curve as a polyline with SamplesPerCurve samples. Polyline is always E3 CagdPolylineStruct type. NULL is returned in case of an error, otherwise CagdPolylineStruct.

7.2.24 SymbCrv3DCurvatureNormal (symb_lib/curvatur.c:251)

curvature

```
CagdCrvStruct *SymbCrv3DCurvatureNormal(CagdCrvStruct *Crv)
```

Crv: To compute the normal curvature field.

Returns: Computed normal curvature field.

Description: Computes a vector field curve representing the curvature of a curve, in the normal direction, that is kN.

```
                 .  ..       .       .  ..     .
                 C x C       C     ( C x C  ) x C
kN = kB x T =   -----  x   ----- = --------------
                   .  3      .            .  4
                 | C |     | C |        | C |
```

7.2.25 SymbCrv3DCurvatureSqr (symb_lib/curvatur.c:128)

curvature

```
CagdCrvStruct *SymbCrv3DCurvatureSqr(CagdCrvStruct *Crv)
```

Crv: To compute vector field curvatrue for,

Returns: Computed vector field curvature of Crv

Description: Computes a vector field curve representing the curvature of a curve, in the binormal direction, that is kB, and square it:

```
       .   ..                          .   ..  2
       C x C                         ( C x C )
kB =  -----            and      k^2 = --------
         .  3                            .  6
       | C |                          | C |
```

7.2.26 SymbCrv3DRadiusNormal (symb_lib/curvatur.c:192)

curvature

```
CagdCrvStruct *SymbCrv3DRadiusNormal(CagdCrvStruct *Crv)
```

Crv: To compute the normal field with radius as magnitude.

Returns: Computed normal field with 1 / k as magnitude.

Description: Computes a vector field curve representing the radius (1/curvature) of a curve, in the normal direction, that is N / k:

```
                 .   ..    .       .  6       .   ..    .     .  2
         k N     (C x C ) x C    | C |      ( (C x C ) x C ) | C |
N / k =  ----- = ------------ . --------- = ----------------------
          2          .  4        .   .. 2          .   .. 2
          k        | C |        (C x C )          (C x C )
```

7.2.27 SymbCrvAdapOffset (symb_lib/offset.c:426)

offset

```
CagdCrvStruct *SymbCrvAdapOffset(CagdCrvStruct *OrigCrv,
                                 CagdRType OffsetDist,
                                 CagdRType OffsetError,
                                 SymbOffCrvFuncType OffsetAprxFunc,
                                 CagdBType BezInterp)
```

OrigCrv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

OffsetError: Tolerance control.

OffsetAprxFunc: A function that can be used to approximate an offset of a curve. If NULL SymbCrvOffset function is selected.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within OffsetError.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction. This function computes an approximation to the offset using OffsetAprxFunc, measure the error and use it to refine and decrease the error adaptively. Bezier curves are promoted to Bsplines curves. See also: Gershon Elber and Elaine Cohen, "Error Bounded Variable Distance Offset Operator for Free Form Curves and Surfaces". International Journal of Computational Geometry & Applications, Vol. 1, Num. 1, March 1991, pp 67-78.

7.2.28 SymbCrvAdapOffsetTrim (symb_lib/offset.c:553)

offset

```
CagdCrvStruct *SymbCrvAdapOffsetTrim(CagdCrvStruct *OrigCrv,
                                     CagdRType OffsetDist,
                                     CagdRType OffsetError,
                                     SymbOffCrvFuncType OffsetAprxFunc,
                                     CagdBType BezInterp)
```

OrigCrv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

OffsetError: Tolerance control.

OffsetAprxFunc: A function that can be used to approximate an offset of a curve. If NULL SymbCrvOffset function is selected. Third parameter of SymbOffCrvFuncType is optional.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within OffsetError.

Description: Same function as CagdCrvAdapOffset, but trims the self intersection loops. See also: Gershon Elber and Elaine Cohen, "Error Bounded Variable Distance Offset Operator for Free Form Curves and Surfaces". International Journal of Computational Geometry & Applications, Vol. 1, Num. 1, March 1991, pp 67-78.

7.2.29 SymbCrvAdd (symb_lib/symbolic.c:34)

addition

symbolic computation

```
CagdCrvStruct *SymbCrvAdd(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to add up coordinatewise.

Returns: The summation of Crv1 + Crv2 coordinatewise.

Description: Given two curves - add them coordinatewise. The two curves are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.

7.2.30 SymbCrvArcLen (symb_lib/arc_len.c:385)

arc length

```
CagdRType SymbCrvArcLen(CagdCrvStruct *Crv, CagdRType Epsilon)
```

Crv: Curve to compute a tight approximation on arc length.

Epsilon: Accuracy control.

Returns: The approximated arc length of the given curve Crv.

Description: Computes a tight approximation to the arc length of a curve. Estimates the arc length scalar field of Crv using SymbCrvArcLenCrv and evaluate the estimate on the curve's domain boundary.

7.2.31 SymbCrvArcLenCrv (symb_lib/arc_len.c:354)

arc length

```
CagdCrvStruct *SymbCrvArcLenCrv(CagdCrvStruct *Crv, CagdRType Epsilon)
```

Crv: To approximate its arc length scalar field.

Epsilon: Accuracy of approximating.

Returns: A scalar field approximating Crv arc length.

Description: Computes a scalar curve approximating the arc length of given curve Crv. Arc length is astimated by computing the square of Crv first derivative approximating its square root and integrating symbolically.

7.2.32 SymbCrvArcLenPoly (symb_lib/arc_len.c:69)

arc length

```
CagdRType SymbCrvArcLenPoly(CagdCrvStruct *Crv)
```

Crv: To bound its length.

Returns: An upper bound on the curve Crv length as the length of Crv's control polygon.

Description: Computes a bound on the arc length of a curve by computing the length of its control polygon.

7.2.33 SymbCrvArcLenSteps (symb_lib/arc_len.c:417)

arc length

```
CagdPtStruct *SymbCrvArcLenSteps(CagdCrvStruct *Crv,
                                 CagdRType Length,
                                 CagdRType Epsilon)
```

Crv: Curve to compute constant arc Length steps.

Length: The step size.

Epsilon: Accuracy control.

Returns: List of parameter values to march along Crv with arc Length between them.

Description: Computes parameter values to move steps of Length at a time on curve Crv. Returned is a list of parameter values to move along.

7.2.34 SymbCrvConstSet (symb_lib/symbzero.c:147)

constant set

zero set

symbolic computation

```
CagdPtStruct *SymbCrvConstSet(CagdCrvStruct *Crv,
                              int Axis,
                              CagdRType Epsilon,
                              CagdRType ConstVal)
```

Crv: To compute its constant set.

Axis: The axis of Crv to compute constant set for.

Epsilon: Tolerance control.

ConstVal: The value at which to compute the constant set.

Returns: List of parameter values form which Crv has an value of ConstVal in axis Axis.

Description: Computes the constant set of a given curve, in the given axis (1-3 for X-Z). Returned is a list of the constant set points holding the parameter values at Pt[0] of each point.

7.2.35 SymbCrvCrossProd (symb_lib/symbolic.c:297)

product

cross product

symbolic computation

```
CagdCrvStruct *SymbCrvCrossProd(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply and compute a cross product for.

Returns: A scalar curve representing the cross product of Crv1 x Crv2.

Description: Given two curves - computes their cross product. Returned curve is a scalar curve representing the cross product of the two given curves.

7.2.36 SymbCrvDotProd (symb_lib/symbolic.c:186)

product

dot product

symbolic computation

```
CagdCrvStruct *SymbCrvDotProd(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply and compute a dot product for.

Returns: A scalar curve representing the dot product of Crv1 . Crv2.

Description: Given two curves - computes their dot product. Returned curve is a scalar curve representing the dot product of the two given curves.

7.2.37 SymbCrvEnclosedArea (symb_lib/symbolic.c:561)

area

symbolic computation

```
CagdCrvStruct *SymbCrvEnclosedArea(CagdCrvStruct *Crv)
```

Crv: A curve to compute area filed curve for.

Returns: The area field curve.

Description: Given a planar curve, compute its enclosed area field curve. This has little meaning unless Crv is closed, in which by evaluation the resulting area filed curve at the end points, the area enclosed by Crv can be computed.

7.2.38 SymbCrvExtremSet (symb_lib/symbzero.c:67)

extremum set

symbolic computation

```
CagdPtStruct *SymbCrvExtremSet(CagdCrvStruct *Crv, int Axis, CagdRType Epsilon)
```

Crv: To compute its extremum set.

Axis: The axis of Crv to compute extremum set for.

Epsilon: Tolerance control.

Returns: List of parameter values form which Crv has an extremum value in axis Axis.

Description: Computes the extremum set of a given curve, in given axis (1-3 for X-Z). Returned is a list of the extreme set points holding the parameter values at Pt[0] of each point. One could compute the derivative of the curve and find its zero set. However, for rational curves, this will double the degree and slow down the computation considerably.

7.2.39 SymbCrvInvert (symb_lib/symbolic.c:112)

division

symbolic computation

reciprocal value

```
CagdCrvStruct *SymbCrvInvert(CagdCrvStruct *Crv)
```

Crv: A scalar curve to compute a reciprocal value for.

Returns: A rational scalar curve that is equalto the reciprocal value of Crv.

Description: Given a scalar curve, returns a scalar curve representing the reciprocal values, by making it rational (if was not one) and flipping the numerator and the denominator.

7.2.40 SymbCrvLeastSquarOffset (symb_lib/offset.c:725)

offset

```
CagdCrvStruct *SymbCrvLeastSquarOffset(CagdCrvStruct *Crv,
                                       CagdRType OffsetDist,
                                       int NumOfSamples,
                                       int NumOfDOF,
                                       int Order,
                                       CagdRType *Tolerance)
```

Crv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

NumOfSamples: Number of samples to sample the offset curve at.

NumOfDOF: Number of degrees of freedom on the newly computed offset approximation. This is thesame as the number of control points the new curve will have.

Order: Of the newly constructed offset approximation. If equal to zero, the order of Crv will be used.

Tolerance: To return an error estimate in the L-infinity norm.

Returns: An approximation to the offset curve.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by least square fitting a curve to samples taken on the offset curve. Resulting curve of order Order (degree of Crv if Order == 0) will have NumOfDOF control points that least sqaure fit NumOfSamples samples on the offset curve. Tolerance will be updated to hold an error distance measure.

7.2.41 SymbCrvMergeScalar (symb_lib/symbolic.c:1248)

merge

symbolic computation

```
CagdCrvStruct *SymbCrvMergeScalar(CagdCrvStruct *CrvW,
                                  CagdCrvStruct *CrvX,
                                  CagdCrvStruct *CrvY,
                                  CagdCrvStruct *CrvZ)
```

CrvW: The weight component of new constructed curve, if have any.

CrvX: The X component of new constructed curve.

CrvY: The Y component of new constructed curve, if have any.

CrvZ: The Z component of new constructed curve, if have any.

Returns: A new curve constructed from given scalar curves.

Description: Given a set of scalar curves, treat them as coordinates into a new curve. Assumes at least CrvX is not NULL in which a scalar curve is returned. Assumes CrvX/Y/Z/W are either E1 or P1 in which the weights are assumed to be identical and can be ignored if CrvW exists or copied otheriwse.

7.2.42 SymbCrvMult (symb_lib/symbolic.c:74)

product

symbolic computation

```
CagdCrvStruct *SymbCrvMult(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply coordinatewise.

Returns: The product of Crv1 * Crv2 coordinatewise.

Description: Given two curves - multiply them coordinatewise. The two curves are promoted to same point type before the multiplication can take place.

7.2.43 SymbCrvMultScalar (symb_lib/symbolic.c:232)

product

symbolic computation

```
CagdCrvStruct *SymbCrvMultScalar(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to multiply.

Returns: A curve representing the product of Crv1 and Crv2.

Description: Given two curves - a vector curve Crv1 and a scalar curve Crv2, multiply all Crv1's coordinates by the scalar curve Crv2. Returned curve is a curve representing the product of the two given curves.

7.2.44 SymbCrvMultiResCompos (symb_lib/multires.c:235)

multi resolution

least square decomposition

```
CagdCrvStruct *SymbCrvMultiResCompos(SymbMultiResCrvStruct *MRCrv)
```

MRCrv: A multi resolution decomposition of a curve.

Returns: A curve that adds up all components of the multi resolution decomposition MRCrv.

Description: Given a multi resolution decomposition of a Bspline curve, computes the regular Bspline curve out of it. *

7.2.45 SymbCrvMultiResComposAtT (symb_lib/multires.c:264)

multi resolution

least square decomposition

```
CagdCrvStruct *SymbCrvMultiResComposAtT(SymbMultiResCrvStruct *MRCrv,
                                        CagdRType T)
```

MRCrv: A multi resolution decomposition of a curve.

T: A mult resolution hierarcy level to compute curve for.

Returns: A curve that adds up all components of the multi resolution decomposition MRCrv up to and including level T.

Description: Given a multi resolution decomposition of a Bspline curve, computes a regular Bspline curve out of it representing the decomposed curve at the multi resolution hierarchy level of T. Although decomposition is discrete, T can be any real number between these discrete levels and a linear interpolation of adjacent levels is exploited.

7.2.46 SymbCrvMultiResCopy (symb_lib/multires.c:573)

multi resolution

least square decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResCopy(SymbMultiResCrvStruct *MRCrvOrig)
```

MRCrv: A multi resolution decomposition of a curve to copy.

Returns: A duplicated structure of MRCrv.

Description: Given a multi resolution decomposition of a Bspline curve, copy it.

7.2.47 SymbCrvMultiResDecomp (symb_lib/multires.c:35)

multi resolution

least square decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResDecomp(CagdCrvStruct *Crv, int Discont)
```

Crv: To compute a least square multi resolution decomposition for.

Discont: Do we want to preserve discontinuities?

Returns: A multi resolution curve structure hold the multi resolution decomposition of Crv.

Description: Given a Bspline curve, computes a hierarch of Bspline curves, each being represented using a subspace of the previous, upto a curve with no interior knots (i.e. a polynomial Bezier). However, if Discont == TRUE, then C1 discontinuities are preserved through out the hierarchy decomposition. Each level in hierarchy has approximately half the number of control points of the previous one. Least square curve fitting is used to build the hierarchy.

7.2.48 SymbCrvMultiResEdit (symb_lib/multires.c:323)

multi resolution

least square decomposition

```
void SymbCrvMultiResEdit(SymbMultiResCrvStruct *MRCrv,
                         CagdRType t,
                         CagdVType TransDir,
                         CagdRType Level,
                         CagdRType FracLevel)
```

MRCrv: A multi resolution decomposition of a curve to edit it in place.

t: Parameter value at which to modify MRCrv.

TransDir: Directional tranlation transformation to apply.

Level: Of multi resolution hierarchy to edit.

FracLevel: The fraction level to edit - will blend two neighboring levels.

SpanDiscont: Are we allowed to cross over discontinuities?

Description: Given a multi resolution decomposition of a Bspline curve, edit it by modifying its Level'th Level according to the TransDir of Position at parametr t. Level can be a fraction number between the discrete levels of the decomposition denoting a linear blend of two neighboring discrete levels. Editing is performed in place.

7.2.49 SymbCrvMultiResFree (symb_lib/multires.c:512)

multi resolution

least square decomposition

```
void SymbCrvMultiResFree(SymbMultiResCrvStruct *MRCrv)
```

MRCrv: A multi resolution decomposition of a curve to free.

Description: Given a multi resolution decomposition of a Bspline curve, free it.

7.2.50 SymbCrvMultiResNew (symb_lib/multires.c:542)

multi resolution

least square decomposition

```
SymbMultiResCrvStruct *SymbCrvMultiResNew(int Levels, CagdBType Periodic)
```

Levels: Number of levels to expect in the decomposition.

Periodic: Is the curve periodic?

Returns: A structure to hold a multi resolution decomposition of a curve of Levels levels.

Description: Allocates a data structure for multi resolution decomposition of a Bspline curve of Levels levels and possiblt periodic.

7.2.51 SymbCrvMultiResRefineLevel (symb_lib/multires.c:430)

multi resolution

least square decomposition

```
CagdRType *SymbCrvMultiResRefineLevel(SymbMultiResCrvStruct *MRCrv,
                                      CagdRType T,
                                      int SpanDiscont)
```

MRCrv: A multi resolution decomposition of a curve, to refine in place.

T: Parameter value at which to refine MRCrv.

SpanDiscont: Do we want to refine beyond discontinuities?

Returns: A pointer to an array of two real numbers holding the domain in MRCrv that was refined.

Description: Given a multi resolution decomposition of a Bspline curve, refine it at neighborhood of parameter value t, in place.

7.2.52 SymbCrvOffset (symb_lib/offset.c:35)

offset

```
CagdCrvStruct *SymbCrvOffset(CagdCrvStruct *Crv,
                             CagdRType OffsetDist,
                             CagdBType BezInterp)
```

Crv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction.

7.2.53 SymbCrvPosNegWeights (symb_lib/symbzero.c:367)

symbolic computation

```
CagdBType SymbCrvPosNegWeights(CagdCrvStruct *Crv)
```

Crv: To examine for same sign weights, if any.

Returns: TRUE if no weights or all of same sign.

Description: Returns TRUE iff the Crv is not rational or rational with weights that are entirely positive or entirely negative.

7.2.54 SymbCrvRtnlMult (symb_lib/symbolic.c:412)

product

symbolic computation

```
CagdCrvStruct *SymbCrvRtnlMult(CagdCrvStruct *Crv1X,
                               CagdCrvStruct *Crv1W,
                               CagdCrvStruct *Crv2X,
                               CagdCrvStruct *Crv2W,
                               CagdBType OperationAdd)
```

Crv1X: Numerator of first curve.

Crv1W: Denominator of first curve.

Crv2X: Numerator of second curve.

Crv2W: Denominator of second curve.

OperationAdd: TRUE for addition, FALSE for subtraction.

Returns: The result of Crv1X Crv2W +/- Crv2X Crv1W.

Description: Given two curves - multiply them using the quotient product rule:

```
X = X1 W2 +/- X2 W1
```

All provided curves are assumed to be non rational scalar curves. Returned is a non rational scalar curve (CAGD_PT_E1_TYPE).

7.2.55 SymbCrvScalarScale (symb_lib/symbolic.c:156)

scaling

symbolic computation

```
CagdCrvStruct *SymbCrvScalarScale(CagdCrvStruct *Crv, CagdRType Scale)
```

Crv: A curve to scale by magnitude Scale.

Scale: Scaling factor.

Returns: A curves scaled by Scale compared to Crv.

Description: Given a curve, scale it by Scale.

7.2.56 SymbCrvSplitScalar (symb_lib/symbolic.c:1196)

split

symbolic computation

```
void SymbCrvSplitScalar(CagdCrvStruct *Crv,
                        CagdCrvStruct **CrvW,
                        CagdCrvStruct **CrvX,
                        CagdCrvStruct **CrvY,
                        CagdCrvStruct **CrvZ)
```

Crv: Curve to split.

CrvW: The weight component of Crv, if have any.

CrvX: The X component of Crv.

CrvY: The Y component of Crv, if have any.

CrvZ: The Z component of Crv, if have any.

Description: Given a curve splits it to its scalar component curves. Ignores all dimensions beyond the third, Z, dimension.

7.2.57 SymbCrvSqrtScalar (symb_lib/arc_len.c:242)

square root

```
CagdCrvStruct *SymbCrvSqrtScalar(CagdCrvStruct *OrigCrv, CagdRType Epsilon)
```

OrigCrv: Scalar curve to approximate its square root function.

Epsilon: Accuracy of approximation.

Returns: A curve approximating the square root of OrigCrv.

Description: Computes the curve which is a square root approximation to a given scalar curve, to within epsilon.

7.2.58 SymbCrvSub (symb_lib/symbolic.c:54)

subtraction

symbolic computation

```
CagdCrvStruct *SymbCrvSub(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: Two curve to subtract coordinatewise.

Returns: The difference of Crv1 - Crv2 coordinatewise.

Description: Given two curves - subtract them coordinatewise. The two curves are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.

7.2.59 SymbCrvSubdivOffset (symb_lib/offset.c:161)

offset

```
CagdCrvStruct *SymbCrvSubdivOffset(CagdCrvStruct *Crv,
                                   CagdRType OffsetDist,
                                   CagdRType Tolerance,
                                   CagdBType BezInterp)
```

Crv: To approximate its offset curve with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Tolerance: Accuracy control.

BezInterp: If TRUE, control points are interpolated when the curve is reduced to a Bezier form. Otherwise, control points are translated OffsetDist amount only, under estimating the Offset.

Returns: An approximation to the offset curve, to within Tolerance.

Description: Given a curve and an offset amount OffsetDist, returns an approximation to the offset curve by offseting the control polygon in the normal direction. If resulting offset is not satisfying the required tolerance the curve is subdivided and the algorithm recurses on both parts.

7.2.60 SymbCrvUnitLenScalar (symb_lib/arc_len.c:110)

unit vector field

```
CagdCrvStruct *SymbCrvUnitLenScalar(CagdCrvStruct *OrigCrv,
                                    CagdBType Mult,
                                    CagdRType Epsilon)
```

OrigCrv: Curve to approximate a unit size for.

Mult: Do we want to multiply the computed scalar curve with Crv?

Epsilon: Accuracy required of this approximation.

Returns: A scalar curve to multiply OrigCrv so a unit size curve will return if Mult is FALSE, or the actual unit size vector field curve, if Mult.

Description: Normalizes the given vector field curve to be a unit length curve, by computing a scalar curve to multiply with this vector field curve. Returns the multiplied curve if Mult, or otherwise just the scalar curve.

7.2.61 SymbCrvZeroSet (symb_lib/symbzero.c:41)

zero set

symbolic computation

```
CagdPtStruct *SymbCrvZeroSet(CagdCrvStruct *Crv, int Axis, CagdRType Epsilon)
```

Crv: To compute its zero set.

Axis: The axis of Crv to compute zero set for.

Epsilon: Tolerance control.

Returns: List of parameter values form which Crv is zero in axis Axis.

Description: Computes the zero set of a given curve, in the given axis (1-3 for X-Z). Returned is a list of the zero set points holding the parameter values att Pt[0] of each point.

7.2.62 SymbDescribeError (symb_lib/symb_err.c:65)

error handling

```
char *SymbDescribeError(SymbFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this symb library as well as other users. Raised error will cause an invokation of SymbFatalError function which decides how to handle this error. SymbFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

7.2.63 SymbDistCrvLine (symb_lib/distance.c:164)

curve line distance

```
CagdRType SymbDistCrvLine(CagdCrvStruct *Crv,
                          CagdLType Line,
                          CagdBType MinDist,
                          CagdRType Epsilon)
```

Crv: The curve to find its nearest (farest) point to Line.

Line: The line to find the nearest (farest) point on Crv to it.

MinDist: If TRUE nearest points is needed, if FALSE farest.

Epsilon: Accuracy of computation.

Returns: Parameter value in the parameter space of Crv of the nearest (farest) point to line Line.

Description: Given a curve and a line, finds the nearest point (if MinDist) or the farest location (if MinDist FALSE) from the curve to the given line. Returned is the parameter value of the curve. Let Crv be (x(t), y(t)). By substituting x(t) and y(t) into the line equation, we derive the distance function. Its zero set, combined with the zero set of its derivative provide the needed extreme distances.

7.2.64 SymbDistCrvPoint (symb_lib/distance.c:45)

curve point distance

```
CagdRType SymbDistCrvPoint(CagdCrvStruct *Crv,
                           CagdPType Pt,
                           CagdBType MinDist,
                           CagdRType Epsilon)
```

Crv: The curve to find its nearest (farest) point to Pt.

Pt: The point to find the nearest (farest) point on Crv to it.

MinDist: If TRUE nearest points is needed, if FALSE farest.

Epsilon: Accuracy of computation.

Returns: Parameter value in the parameter space of Crv of the nearest (farest) point to point Pt.

Description: Given a curve and a point, finds the nearest point (if MinDist) or the farest location (if MinDist FALSE) from the curve to the given point. Returned is the parameter value of the curve. Computes the zero set of (Crv(t) - Pt) . Crv'(t).

7.2.65 SymbExtremumCntPtVals (symb_lib/symbolic.c:1536)

extremum values

symbolic computation

```
CagdRType *SymbExtremumCntPtVals(CagdRType **Points,
                                 int Length,
                                 CagdBType FindMinimum)
```

Points: To scan for extremum values.

Length: Length of each vector in Points.

FindMinimum: TRUE for minimum, FALSE for maximum.

Returns: A vector holding PType point with the extremum values of each axis independently.

Description: Given a controlpolygon/mesh, computes the extremum values of them all.

7.2.66 SymbFatalError (symb_lib/symb_ftl.c:25)

error handling

```
void SymbFatalError(SymbFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Description: Trap Symb_lib errors right here. Provides a default error handler for the symb library. Gets an error description using SymbDescribeError, prints it and exit the program using exit.

7.2.67 SymbLclDistCrvLine (symb_lib/distance.c:235)

curve line distance

```
CagdPtStruct *SymbLclDistCrvLine(CagdCrvStruct *Crv,
                                 CagdLType Line,
                                 CagdRType Epsilon)
```

Crv: The curve to find its nearest (farest) point to Line.

Line: The line to find the nearest (farest) point on Crv to it.

Epsilon: Accuracy of computation.

Returns: A list of parameter values of extreme distance locations.

Description: Given a curve and a line, finds the local extreme distance points on the curve to the given line. Returned is a list of parameter value with local extreme distances. Let Crv be (x(t), y(t)). By substituting x(t) and y(t) into the line equation, we derive the distance function. Its zero set, combined with the zero set of its derivative provide the needed extreme distances.

7.2.68 SymbLclDistCrvPoint (symb_lib/distance.c:114)

curve point distance

```
CagdPtStruct *SymbLclDistCrvPoint(CagdCrvStruct *Crv,
                                  CagdPType Pt,
                                  CagdRType Epsilon)
```

Crv: The curve to find its extreme distance locations to Pt.

Pt: The point to find the extreme distance locations from Crv.

Epsilon: Accuracy of computation.

Returns: A list of parameter values of extreme distance locations.

Description: Given a curve and a point, find the local extremum distance points on the curve to the given point. Returned is a list of parameter value with local extremum. Computes the zero set of (Crv(t) - Pt) . Crv'(t).

7.2.69 SymbLimitCrvArcLen (symb_lib/arc_len.c:29)

arc length

```
CagdCrvStruct *SymbLimitCrvArcLen(CagdCrvStruct *Crv, CagdRType MaxLen)
```

Crv: To subdivide into curves, each with control polygon length less than MaxLen.

MaxLen: Maximum length of control polygon to allow.

Returns: List of subdivided curves from Crv, each with control polygon size of less than MaxLen.

Description: Subdivides the given curves to curves, each with size of control polygon less than or equal to MaxLen. Returned is a list of curves.

7.2.70 SymbMakePosCrvCtlPolyPos (symb_lib/curvatur.c:396)

refinement

```
CagdCrvStruct *SymbMakePosCrvCtlPolyPos(CagdCrvStruct *OrigCrv)
```

OrigCrv: To refine until all its control points are non negative.

Returns: Refined positive curve with positive control points.

Description: Given a scalar curve that is positive, refine it until all its control points has positive coefficients. Always returns a Bspline curve.

7.2.71 SymbMeshAddSub (symb_lib/symbolic.c:1142)

addition

subtraction

symbolic computation

```
void SymbMeshAddSub(CagdRType **DestPoints,
                    CagdRType **Points1,
                    CagdRType **Points2,
                    CagdPointType PType,
                    int Size,
                    CagdBType OperationAdd)
```

DestPoints: Where addition or difference result should go to.

Points1: First control polygon/mesh.

Points2: Second control polygon/mesh.

PType: Type of points we are dealing with.

Size: Length of each vector in Points1/2.

OperationAdd: TRUE of addition, FALSE for subtraction.

Description: Given two control polygons/meshes - add them coordinatewise. If mesh is rational, weights are assumed identical and are just copied.

7.2.72 SymbPiecewiseRuledSrfApprox (symb_lib/prisa.c:120)

layout

prisa

ruled surface approximation

```
CagdSrfStruct *SymbPiecewiseRuledSrfApprox(CagdSrfStruct *Srf,
                                           CagdBType ConsistentDir,
                                           CagdRType Epsilon,
                                           CagdSrfDirType Dir)
```

Srf: To approximate using piecewise ruled surfaces.

ConsistentDir: Do we want parametrization to be the same as Srf?

Epsilon: Accuracy of piecewise ruled surface approximation.

Dir: Direction of piecewise ruled surface approximation. Either U or V.

Returns: A list of ruled surfaces approximating Srf to within Epsilon in direction Dir.

Description: Constructs a piecewise ruled surface approximation to the given surface, Srf, in the given direction, Dir, that is close to the surface to within Epsilon. If ConsitentDir then ruled surface parametrization is set to be the same as original surface Srf. Otherwise, ruling dir is always CAGD_CONST_V_DIR. Surface is assumed to have point types E3 or P3 only.

7.2.73 SymbPrisaRuledSrf (symb_lib/prisa.c:360)

layout

prisa

```
CagdSrfStruct *SymbPrisaRuledSrf(CagdSrfStruct *Srf,
                                 int SamplesPerCurve,
                                 CagdRType Space,
                                 CagdVType Offset)
```

Srf: A ruled surface to layout flat on the XY plane.

SamplesPerCurve: During the approximation of a ruled surface as a developable surface.

Space: Increment on Y on the offset vector, after this surface was placed in the XY plane.

Offset: A vector in the XY plane to denote the amount of translation for the flatten surface in the XY plane.

Returns: A planar surface in the XY plane approximating the falttening process of Srf.

Description: Layout a single ruled surface, by approximating it as a set of polygons. The given ruled surface might be non-developable, in which case approximation will be of a surface with no twist. The ruled surface is assumed to be constructed using CagdRuledSrf and that the ruled direction is consisnt and is always CAGD_CONST_V_DIR.

7.2.74 SymbPrmtSclrCrvTo2D (symb_lib/symbolic.c:1437)

promotion

conversion

symbolic computation

```
CagdCrvStruct *SymbPrmtSclrCrvTo2D(CagdCrvStruct *Crv,
                                   CagdRType Min,
                                   CagdRType Max)
```

Crv: Scalar curve to promose to a two dimensional one.

Min: Minimum of new monotone X axis.

Max: Maximum of new monotone X axis.

Returns: A two dimensional curve.

Description: Promote a scalar curve to two dimensions by moving the scalar axis to be the Y axis and adding monotone X axis.

7.2.75 SymbPrmtSclrSrfTo3D (symb_lib/symbolic.c:1483)

promotion

conversion

symbolic computation

```
CagdSrfStruct *SymbPrmtSclrSrfTo3D(CagdSrfStruct *Srf,
                                   CagdRType UMin,
                                   CagdRType UMax,
                                   CagdRType VMin,
                                   CagdRType VMax)
```

Srf:

UMin: Minimum of new monotone X axis.

UMax: Maximum of new monotone X axis.

VMin: Minimum of new monotone Y axis.

VMax: Maximum of new monotone X axis.

Returns: A three dimensional surface.

Description: Promote a scalar surface to three dimensions by moving the scalar axis to be the Z axis and adding monotone X and Y axes.

7.2.76 SymbSetAdapIsoExtractMinLevel (symb_lib/adap_iso.c:55)

adaptive isocurves

```
void SymbSetAdapIsoExtractMinLevel(int MinLevel)
```

MinLevel: At least that many subdivision will occur.

Description: Sets minimum level of subdivision forced in the adaptive iso extraction.

7.2.77 SymbSrf2Curves (symb_lib/symbpoly.c:153)

curves

isoparametric curves

```
CagdCrvStruct *SymbSrf2Curves(CagdSrfStruct *Srf, int NumOfIsocurves[2])
```

Srf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct.

7.2.78 SymbSrf2DDeterminant (symb_lib/curvatur.c:612)

determinant

```
CagdSrfStruct *SymbSrf2DDeterminant(CagdSrfStruct *Srf11,
                                    CagdSrfStruct *Srf12,
                                    CagdSrfStruct *Srf21,
                                    CagdSrfStruct *Srf22)
```

Srf11, Srf12, Srf21, Srf22: The four factors of the determinant.

Returns: A scalar field representing the determinant computation.

Description: Computes the expression of Srf11 * Srf22 - Srf12 * Srf21, which is a determinant of a 2 by 2 matrix.

7.2.79 SymbSrf2OptPolysBilinPolyError (symb_lib/symbsply.c:288)

polygonization

surface approximation

```
CagdRType SymbSrf2OptPolysBilinPolyError(CagdSrfStruct *Srf,
                                         CagdSrfDirType Dir)
```

Srf: To estimate curvature for.

Dir: Currently not used.

Returns: Curvature estimated.

Description: Routine to estimate the curvature of the patch using a bilinear approx.

7.2.80 SymbSrf2OptPolysCurvatureError (symb_lib/symbsply.c:199)

polygonization

surface approximation

```
CagdRType SymbSrf2OptPolysCurvatureError(CagdSrfStruct *Srf,
                                         CagdSrfDirType Dir)
```

Srf: To estimate curvature for.

Dir: Currently not used.

Returns: Curvature estimated.

Description: Routine to estimate the curvature of the patch using k1^2 + k2^2. Assumes the availability of the GlblCrvtrSqrSrf for Srf. This estimate is too loose and in fact is not recommended!

7.2.81 SymbSrf2OptPolysCurvatureErrorPrep (symb_lib/symbsply.c:172)

polygonization

surface approximation

```
void SymbSrf2OptPolysCurvatureErrorPrep(CagdSrfStruct *Srf)
```

Srf: To compute the curvature bound for as an optional preprocess for function SymbSrf2OptPolysCurvatureError.

Description: Routine to compute the scalar field of k1^2 + k2^2 (k1, k2 are principal curvatures) for the surface Srf, into GlblCrvtrSqrSrf. This scalar field is used by SymbSrf2OptPolysCurvatureError function.

7.2.82 SymbSrf2OptPolysIsoDirCurvatureErrorPrep (symb_lib/symbsply.c:355)

polygonization

surface approximation

```
void SymbSrf2OptPolysIsoDirCurvatureErrorPrep(CagdSrfStruct *Srf)
```

Srf: To compute the curvature bound in the isoparametric direction.

Description: Routine to precompute the scalar field of kn^u and kn^v (the normal curvatures in the iso parametric directions). These scalar fields are used to determined the prefered subdivision location of Srf.

7.2.83 SymbSrf2OptimalPolygons (symb_lib/symbsply.c:400)

approximation

conversion

```
CagdPolygonStruct *SymbSrf2OptimalPolygons(CagdSrfStruct *Srf,
                               CagdRType Tolerance,
                               SymbPlSubdivStrategyType SubdivDirStrategy,
                               SymbPlErrorFuncType SrfPolyApproxErr,
                               CagdBType ComputeNormals,
                               CagdBType FourPerFlat,
                               CagdBType ComputeUV)
```

Srf: To convert and approximate using triangles.

Tolerance: Accuracy control.

SubdivDirStrategy: Alternatively in U and V, direction that minimizes the error, etc.

SrfPolyApproxErr: Using bilinear curvature estimate, k1^2 + k2^2 estimate, etc.

ComputeNormals: Do we want normals to be computed as well?

FourPerFlat: If TRUE, four triangle per flat surface patch are created, otherwise only two.

ComputeUV: Do we want UV parameter values with the vertices of the triangles?

Returns: Resulting polygons that approximates Srf.

Description: Routine to convert a single surface to a set of triangles approximating it. FineNess is controlled via a function that returns an error measure SrfPolyApproxError that is guaranteed to be less than Tolerance.

7.2.84 SymbSrf2Polygons (symb_lib/symbpoly.c:49)

polygonization

surface approximation

```
CagdPolygonStruct *SymbSrf2Polygons(CagdSrfStruct *Srf,
                                    int FineNess,
                                    CagdBType ComputeNormals,
                                    CagdBType FourPerFlat,
                                    CagdBType ComputeUV)
```

Srf: To approximate into triangles.

FineNess: Control on accuracy, the higher the finer.

ComputeNormals: If TRUE, normal information is also computed.

FourPerFlat: If TRUE, four triangles are created per flat surface. If FALSE, only 2 triangles are created.

ComputeUV: If TRUE, UV values are stored and returned as well.

Returns: A list of polygons with optional normal and/or UV parametric information. NULL is returned in case of an error.

Description: Routine to convert a single surface to set of triangles approximating it. FineNess is a finess control on result and the larger it is more triangles may result. A value of 10 is a good starting value. NULL is returned in case of an error, otherwise list of CagdPolygonStruct.

7.2.85 SymbSrf2Polylines (symb_lib/symbpoly.c:101)

polylines

isoparametric curves

```
CagdPolylineStruct *SymbSrf2Polylines(CagdSrfStruct *Srf,
                                      int NumOfIsocurves[2],
                                      int SamplesPerCurve,
                                      int Optimal)
```

Srf: Srf to extract isoparametric curves from.

NumOfIsocurves: To extarct from Srf in each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Optimal: Use optimal approximation of isocurves.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.

7.2.86 SymbSrfAdd (symb_lib/symbolic.c:634)

addition

symbolic computation

```
CagdSrfStruct *SymbSrfAdd(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to add up coordinatewise.

Returns: The summation of Srf1 + Srf2 coordinatewise.

Description: Given two surfaces - add them coordinatewise. The two surfaces are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.

7.2.87 SymbSrfCrossProd (symb_lib/symbolic.c:850)

product

cross product

symbolic computation

```
CagdSrfStruct *SymbSrfCrossProd(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to multiply and compute a cross product for.

Returns: A scalar surface representing the cross product of Srf1 x Srf2.

Description: Given two surfaces - computes their cross product. Returned surface is a scalar surface representing the cross product of the two given surfaces.

7.2.88 SymbSrfCurvatureUpperBound (symb_lib/curvatur.c:648)

curvature

```
CagdSrfStruct *SymbSrfCurvatureUpperBound(CagdSrfStruct *Srf)
```

Srf: Surface to compute curvature bound for.

Returns: A scalar field representing the curvature bound.

Description: Computes curvature upper bound as Xi = k1^2 + k2^2, where k1 and k2 are the principal curvatures. Gij are the coefficients of the first fundamental form and Lij are of the second, using non unit normal n,

```
      ( G11 L22 + G22 L11 - 2 G12 L12 )^2 - 2 |G| |L|
 Xi = -----------------------------------------------
                          |G|^2 ||n||^2
```

See: "Second Order Surface Analysis Using Hybrid of Symbolic and Numeric Operators", By Gershon Elber and Elaine Cohen, Transaction on graphics, Vol. 12, No. 2, pp 160-178, April 1993.

7.2.89 SymbSrfDistCrvCrv (symb_lib/distance.c:300)

curve curve distance

```
CagdSrfStruct *SymbSrfDistCrvCrv(CagdCrvStruct *Crv1, CagdCrvStruct *Crv2)
```

Crv1, Crv2: The two curves, Crv1(u) and Crv2(v), to form their distance function square between them as a bivariate function.

Returns: The distance function square d2(u, v) of the distance from Crv1(u) to Crv2(v).

Description: Given two curves, creates a bivariate scalar surface representing the distance function square, between them.

7.2.90 SymbSrfDistFindPoints (symb_lib/distance.c:365)

curve curve distance

curve curve intersection

```
CagdPtStruct *SymbSrfDistFindPoints(CagdSrfStruct *Srf,
                                    CagdRType Epsilon,
                                    CagdBType SelfInter)
```

Srf: A bivariate function that represent the distance square function between two curves.

Epsilon: Accuracy control.

SelfInter: Should be consider self intersection? That is, is Srf between a curve to itself!?

Returns: A list of parameter values of both curves, at all detected intersection locations.

Description: Given a scalar surface representing the distance function square between two curves, finds the zero set of the distance surface, if any, and returns it. The given surface is a non negative surface and zero set is its minima. The returned points will contail the two parameter values of the two curves that intersect in the detected zero set points.

7.2.91 SymbSrfDotProd (symb_lib/symbolic.c:804)

product

dot product

symbolic computation

```
CagdSrfStruct *SymbSrfDotProd(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to multiply and compute a dot product for.

Returns: A scalar surface representing the dot product of Srf1 . Srf2.

Description: Given two surfaces - computes their dot product. Returned surface is a scalar surface representing the dot product of the two given surfaces.

7.2.92 SymbSrfFff (symb_lib/curvatur.c:544)

first fundamental form

```
void SymbSrfFff(CagdSrfStruct *Srf,
                CagdSrfStruct **DuSrf,
                CagdSrfStruct **DvSrf,
                CagdSrfStruct **FffG11,
                CagdSrfStruct **FffG12,
                CagdSrfStruct **FffG22)
```

Srf: Do compute the coefficients of the FFF for.

DuSrf: First derivative of Srf with respect to U goes to here.

DvSrf: First derivative of Srf with respect to V goes to here.

FffG11: FFF G11 scalar field.

FffG12: FFF G12 scalar field.

FffG22: FFF G22 scalar field.

Description: Computes coefficients of the first fundamental form of given surface Srf.

7.2.93 SymbSrfFff (symb_lib/curvatur.c:578)

second fundamental form

```
void SymbSrfSff(CagdSrfStruct *DuSrf,
                CagdSrfStruct *DvSrf,
                CagdSrfStruct **SffL11,
                CagdSrfStruct **SffL12,
                CagdSrfStruct **SffL22,
                CagdSrfStruct **SNormal)
```

DuSrf: First derivative of Srf with respect to U.

DvSrf: First derivative of Srf with respect to V.

SffL11: SFF L11 scalar field returned herein.

SffL12: SFF L12 scalar field returned herein.

SffL22: SFF L22 scalar field returned herein.

SNormal: Unnormalized normal vector field returned herein.

Description: Computes coefficients of the first fundamental form of given surface Srf. These coefficients are using non normalized normal that is also returned.

7.2.94 SymbSrfInvert (symb_lib/symbolic.c:721)

division

symbolic computation

reciprocal value

```
CagdSrfStruct *SymbSrfInvert(CagdSrfStruct *Srf)
```

Srf: A scalar surface to compute a reciprocal value for.

Returns: A rational scalar surface that is equalto the reciprocal value of Srf.

Description: Given a scalar surface, returns a scalar surface representing the reciprocal values, by making it rational (if was not one) and flipping the numerator and the denominator.

7.2.95 SymbSrfIsoDirNormalCurvatureBound (symb_lib/curvatur.c:724)

curvature

```
CagdSrfStruct *SymbSrfIsoDirNormalCurvatureBound(CagdSrfStruct *Srf,
                                                 CagdSrfDirType Dir)
```

Srf: To compute normal curvature in an isoparametric direction Dir.

Dir: Direction to compute normal curvature. Either U or V.

Returns: A scalar field representing the normal curvature square of Srf in dirction Dir.

Description: Computes normal curvature bound in given isoparametric direction. This turns out to be (L11 . n) / G11 for u and (L22 . n) / G22 for v. Herein the square of these equations is computed symbolically and returned.

7.2.96 SymbSrfMergeScalar (symb_lib/symbolic.c:1371)

merge

symbolic computation

```
CagdSrfStruct *SymbSrfMergeScalar(CagdSrfStruct *SrfW,
                                  CagdSrfStruct *SrfX,
                                  CagdSrfStruct *SrfY,
                                  CagdSrfStruct *SrfZ)
```

SrfW: The weight component of new constructed surface, if have any.

SrfX: The X component of new constructed surface.

SrfY: The Y component of new constructed surface, if have any.

SrfZ: The Z component of new constructed surface, if have any.

Returns: A new surface constructed from given scalar surfaces.

Description: Given a set of scalar surfaces, treat them as coordinates into a new surface. Assumes at least SrfX is not NULL in which a scalar surface is returned. Assumes SrfX/Y/Z/W are either E1 or P1 in which the weights are assumed to be identical and can be ignored if SrfW exists or copied otheriwse.

7.2.97 SymbSrfMult (symb_lib/symbolic.c:675)

product

symbolic computation

```
CagdSrfStruct *SymbSrfMult(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to multiply coordinatewise.

Returns: The product of Srf1 * Srf2 coordinatewise.

Description: Given two surfaces - multiply them coordinatewise. The two surfaces are promoted to same point type before the multiplication can take place.

7.2.98 SymbSrfNormalSrf (symb_lib/symbolic.c:998)

normal

symbolic computation

```
CagdSrfStruct *SymbSrfNormalSrf(CagdSrfStruct *Srf)
```

Srf: To compute an unnormalized normal vector field for.

Returns: A vector field representing the unnormalized normal vector field of Srf.

Description: Given a surface - compute its unnormalized norrmal vectorfield surface, as the cross product if this partial derivatives.

7.2.99 SymbSrfOffset (symb_lib/offset.c:232)

offset

```
CagdSrfStruct *SymbSrfOffset(CagdSrfStruct *Srf, CagdRType OffsetDist)
```

Srf: To approximate its offset surface with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Returns: An approximation to the offset surface.

Description: Given a surface and an offset amount OffsetDist, returns an approximation to the offset surface by offseting the control mesh in the normal direction.

7.2.100 SymbSrfRtnlMult (symb_lib/symbolic.c:954)

product

symbolic computation

```
CagdSrfStruct *SymbSrfRtnlMult(CagdSrfStruct *Srf1X,
                               CagdSrfStruct *Srf1W,
                               CagdSrfStruct *Srf2X,
                               CagdSrfStruct *Srf2W,
                               CagdBType OperationAdd)
```

Srf1X: Numerator of first surface.

Srf1W: Denominator of first surface.

Srf2X: Numerator of second surface.

Srf2W: Denominator of second surface.

OperationAdd: TRUE for addition, FALSE for subtraction.

Returns: The result of Srf1X Srf2W +/- Srf2X Srf1W.

Description: Given two surfaces - multiply them using the quotient product rule:

```
X = X1 W2 +/- X2 W1
```

All provided surfaces are assumed to be non rational scalar surfaces. Returned is a non rational scalar surface (CAGD_PT_E1_TYPE).

7.2.101 SymbSrfScalarScale (symb_lib/symbolic.c:765)

scaling

symbolic computation

```
CagdSrfStruct *SymbSrfScalarScale(CagdSrfStruct *Srf, CagdRType Scale)
```

Srf: A surface to scale by magnitude Scale.

Scale: Scaling factor.

Returns: A surfaces scaled by Scale compared to Srf.

Description: Given a surface, scale it by Scale.

7.2.102 SymbSrfSplitScalar (symb_lib/symbolic.c:1313)

split

symbolic computation

```
void SymbSrfSplitScalar(CagdSrfStruct *Srf,
                        CagdSrfStruct **SrfW,
                        CagdSrfStruct **SrfX,
                        CagdSrfStruct **SrfY,
                        CagdSrfStruct **SrfZ)
```

Srf: Surface to split.

SrfW: The weight component of Srf, if have any.

SrfX: The X component of Srf.

SrfY: The Y component of Srf, if have any.

SrfZ: The Z component of Srf, if have any.

Description: Given a surface splits it to its scalar component surfaces. Ignores all dimensions beyond the third, Z, dimension.

7.2.103 SymbSrfSub (symb_lib/symbolic.c:655)

subtraction

symbolic computation

```
CagdSrfStruct *SymbSrfSub(CagdSrfStruct *Srf1, CagdSrfStruct *Srf2)
```

Srf1, Srf2: Two surface to subtract coordinatewise.

Returns: The difference of Srf1 - Srf2 coordinatewise.

Description: Given two surfaces - subtract them coordinatewise. The two surfaces are promoted to same point type before the multiplication can take place. Furthermore, order and continuity are matched as well.

7.2.104 SymbSrfSubdivOffset (symb_lib/offset.c:324)

offset

```
CagdSrfStruct *SymbSrfSubdivOffset(CagdSrfStruct *Srf,
                                   CagdRType OffsetDist,
                                   CagdRType Tolerance)
```

Srf: To approximate its offset surface with distance OffsetDist.

OffsetDist: Amount of offset. Negative denotes other offset direction.

Tolerance: Accuracy control.

Returns: An approximation to the offset surface, to within Tolerance.

Description: Given a surface and an offset amount OffsetDist, returns an approximation to the offset surface by offseting the control mesh in the normal direction. If resulting offset is not satisfying the required tolerance the surface is subdivided and the algorithm recurses on both parts.

7.2.105 SymbTwoCrvsMorphing (symb_lib/morphing.c:51)

morphing

```
CagdCrvStruct *SymbTwoCrvsMorphing(CagdCrvStruct *Crv1,
                                   CagdCrvStruct *Crv2,
                                   CagdRType Blend)
```

Crv1, Crv2: The two curves to blend.

Blend: A parameter between zero and one

Returns: Crv2 * Blend + Crv1 * (1 - Blend).

Description: Given two compatible curves (See function CagdmakeCrvsCompatible), computes a convex blend between them according to Blend which must be between zero and one. Returned is the new blended curve.

7.2.106 SymbTwoCrvsMorphingCornerCut (symb_lib/morphing.c:115)

morphing

```
CagdCrvStruct *SymbTwoCrvsMorphingCornerCut(CagdCrvStruct *Crv1,
                                            CagdCrvStruct *Crv2,
                                            CagdRType MinDist,
                                            CagdBType SameLength,
                                            CagdBType FilterTangencies)
```

Crv1, Crv2: The two curves to blend.

MinDist: Minimal maximum distance between adjacent curves to make sure motion is visible. The curves will move at most twice that much in their maximal distance (roughly).

SameLength: If TRUE, length of curves is preserved, otherwise BBOX is.

FilterTangenices: If TRUE, attempt is made to eliminate the intermediate line representation.

Returns: The blended curve.

Description: Given two compatible curves (See function CagdMakeCrvsCompatible), computes a morph between them using corner cutting approach. Returned is the new blended curve.

7.2.107 SymbTwoCrvsMorphingMultiRes (symb_lib/morphing.c:293)

morphing

```
CagdCrvStruct *SymbTwoCrvsMorphingMultiRes(CagdCrvStruct *Crv1,
                                           CagdCrvStruct *Crv2,
                                           CagdRType BlendStep)
```

Crv1, Crv2: The two curves to blend.

BlendStep: A step size of the blending.

Returns: A list of blended curves.

Description: Given two compatible curves (See function CagdMakeCrvsCompatible), computes a morph between them using multiresolution decomposition. Returned is a list of new blended curves.

7.2.108 SymbTwoSrfsMorphing (symb_lib/morphing.c:725)

morphing

```
CagdSrfStruct *SymbTwoSrfsMorphing(CagdSrfStruct *Srf1,
                                   CagdSrfStruct *Srf2,
                                   CagdRType Blend)
```

Srf1, Srf2: The two surfaces to blend.

Blend: A parameter between zero and one

Returns: Srf2 * Blend + Srf1 * (1 - Blend).

Description: Given two compatible surfaces (See function CagdmakeSrfsCompatible), computes a convex blend between them according to Blend which must be between zero and one. Returned is the new blended surface.

Chapter 8

Trimmed surfaces Library, trim_lib

8.1 General Information

This library provides a set of function to manipulate freeform trimmed Bezier and/or NURBs surfaces. This library heavily depends on the cagd library. Functions are provided to create, copy, and destruct trimmed surfaces to extract isoparametric curves, to evaluate, refine and subdivide, to read and write trimmed surfaces, degree raise, and approximate using polygonal representations.

The interface of the library is defined in *include/trim_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **TrimFatalError**.

All globals in this library have a prefix of **Trim**.

8.2 Library Functions

8.2.1 TrimCrvCopyList (trim_lib/trim_gen.c:213)

copy

```
TrimCrvStruct *TrimCrvCopyList(TrimCrvStruct *TrimCrvList)
```

TrimCrvList: To be copied.

Returns: A duplicated list of trimming curves.

Description: Allocates and copies a list of trimming curve structures.

8.2.2 TrimCrvFree (trim_lib/trim_gen.c:242)

allocation

```
void TrimCrvFree(TrimCrvStruct *TrimCrv)
```

TrimCrv: A trimming curve to free.

Description: Deallocates a trimming curve structure.

8.2.3 TrimCrvFreeList (trim_lib/trim_gen.c:262)

allocation

```
void TrimCrvFreeList(TrimCrvStruct *TrimCrvList)
```

TrimCrvList: A list of trimming curve to free.

Description: Deallocates a list of trimming curve structures.

8.2.4 TrimCrvNew (trim_lib/trim_gen.c:160)

allocation

```
TrimCrvStruct *TrimCrvNew(TrimCrvSegStruct *TrimCrvSegList)
```

TrimCrvSegList: List of trimming curve segments forming the trimming curve.

Returns: A trimmig curve.

Description: Allocates a trimming curve structure.

8.2.5 TrimCrvNew (trim_lib/trim_gen.c:186)

allocation

```
TrimCrvStruct *TrimCrvCopy(TrimCrvStruct *TrimCrv)
```

TrimCrv: A trimming curve to duplicate.

Returns: A trimming curve structure.

Description: Duplicates a trimming curve structure.

8.2.6 TrimCrvSegCopyList (trim_lib/trim_gen.c:85)

copy

```
TrimCrvSegStruct *TrimCrvSegCopyList(TrimCrvSegStruct *TrimCrvSegList)
```

TrimCrvSegList: To be copied.

Returns: A duplicated list of trimming curve segments.

Description: Allocates and copies a list of trimming curve segment structures.

8.2.7 TrimCrvSegFree (trim_lib/trim_gen.c:114)

allocation

```
void TrimCrvSegFree(TrimCrvSegStruct *TrimCrvSeg)
```

TrimCrvSeg: A trimming curve segment to free.

Description: Deallocates a trimming curve segment structure.

8.2.8 TrimCrvSegFreeList (trim_lib/trim_gen.c:135)

allocation

```
void TrimCrvSegFreeList(TrimCrvSegStruct *TrimCrvSegList)
```

TrimCrvSegList: A list of trimming curve segments to free.

Description: Deallocates a list of trimming curve segment structures.

8.2.9 TrimCrvSegNew (trim_lib/trim_gen.c:26)

allocation

```
TrimCrvSegStruct *TrimCrvSegNew(CagdCrvStruct *UVCrv, CagdCrvStruct *EucCrv)
```

UVCrv: A UV curve. Must be an E2 curve.

EucCrv: Optional Euclidean curve. Must be an E3 curve.

Returns: A trimming curve segment structure.

Description: Allocates a trimming curve segment structure.

8.2.10 TrimCrvSegNew (trim_lib/trim_gen.c:55)

allocation

```
TrimCrvSegStruct *TrimCrvSegCopy(TrimCrvSegStruct *TrimCrvSeg)
```

TrimCrvSeg: A trimming curve segment to duplicate.

Returns: A trimming curve segment structure.

Description: Duplicates a trimming curve segment structure.

8.2.11 TrimCrvs2Polylines (trim_lib/trim_aux.c:377)

trimming curves

```
CagdPolylineStruct *TrimCrvs2Polylines(TrimSrfStruct *TrimSrf,
                                       CagdBType ParamSpace,
                                       int SamplesPerCurve,
                                       int Optimal)
```

TrimSrf: To extract isoparametric curves from.

ParamSpace: TRUE for curves in parameteric space, FALSE of 3D Euclidean space.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Optimal: Use optimal approximation of isocurves.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert the trimming curves of a trimmed surface to polylines. Polyline are always E3 of CagdPolylineStruct type. NULL is returned in case of an error, otherwise list of CagdPolylineStruct.

8.2.12 TrimDbg (trim_lib/trim_dbg.c:24)

debugging

```
void TrimDbg(void *Obj)
```

Obj: A trimmed surface - to be printed to stderr.

Description: Prints trimmed surfaces to stderr. Should be linked to programs for debugging purposes, so trimmed surfaces may be inspected from a debugger.

8.2.13 TrimDescribeError (trim_lib/trim_err.c:42)

error handling

```
char *TrimDescribeError(TrimFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this trim library as well as other users. Raised error will cause an invokation of TrimFatalError function which decides how to handle this error. TrimFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

8.2.14 TrimEvalTrimCrvToEuclid (trim_lib/trim_aux.c:411)

```
CagdCrvStruct *TrimEvalTrimCrvToEuclid(TrimSrfStruct *TrimSrf,
                                       CagdCrvStruct *UVCrv)
```

TrimSrf: To compute the Euclidean UVCrv for.

UVCrv: A curve in the parametric space of TrimSrf.

Returns: A Euclidean curve in TrimSrf, following UVCrv.

Description: Computes the composed Euclidean curve of TrimSrf(UVCrv). The resulting curve is either computed using a piecewise linear approximation or by symbolically composing it onto the surface.

8.2.15 TrimFatalError (trim_lib/trim_ftl.c:25)

error handling

```
void TrimFatalError(TrimFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Description: Trap Trim_lib errors right here. Provides a default error handler for the trim library. Gets an error description using TrimDescribeError, prints it and exit the program using exit.

8.2.16 TrimGetTrimmingCurves (trim_lib/trim_aux.c:318)

trimming curves

```
CagdCrvStruct *TrimGetTrimmingCurves(TrimSrfStruct *TrimSrf,
                                     CagdBType ParamSpace)
```

TrimSrf: Trimmed surface to extract trimming curves from.

ParamSpace: TRUE for curves in parameteric space, FALSE of 3D Euclidean space.

Returns: List of trimming curves of TrimSrf.

Description: Extracts the trimming curves of the given trimmed surface.

8.2.17 TrimSetEuclidComposedFromU (trim_lib/trim_aux.c:466)

```
int TrimSetEuclidComposedFromUV(int EuclidComposedFromUV)
```

EuclidComposedFromUV: Do we want symbolic composition for Euclidean curves, or should we piecewise linear sample the UV trimming curves.

Returns: Old value of way of Euclidean curve's computation

Description: Sets the way Euclidean trimming curves are computed from parametric trimming curves. Either by symbolic composition (TRUE) or by piecewise linear approximation of trimming curves (FALSE).

8.2.18 TrimSetTrimCrvLinearApprox (trim_lib/iso_crvs.c:505)

```
int TrimSetTrimCrvLinearApprox(int UVSamplesPerCurve,
                               int UVSamplesOptimal)
```

UVSamplesPerCurve: Piecewise linear approximation of high order trimming curves - number of samples per curve.

UVSamplesOptimal: Do we want optimal sampling (see CagdCrv2Polyline).

Returns: Old number of samples per curve.

Description: Sets the tolerances to use when approximating higher order trimming curves using piecewise linear approximation, for intersection computation.

8.2.19 TrimSrf2Curves (trim_lib/iso_crvs.c:112)

curves

isoparametric curves

```
CagdCrvStruct *TrimSrf2Curves(TrimSrfStruct *TrimSrf, int NumOfIsocurves[2])
```

TrimSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

Returns: List of extracted isoparametric curves. These curves inherit the order and continuity of the original Srf. NULL is returned in case of an error.

Description: Routine to extract from a trimmed surface NumOfIsoline isocurve list in each param. direction. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdCrvStruct. As the isoparametric curves are trimmed according to the trimming curves the resulting number of curves is arbitrary.

8.2.20 TrimSrf2Polylines (trim_lib/iso_crvs.c:74)

isoparametric curves

```
CagdPolylineStruct *TrimSrf2Polylines(TrimSrfStruct *TrimSrf,
                                      int NumOfIsocurves[2],
                                      int SamplesPerCurve,
                                      int Optimal)
```

TrimSrf: To extract isoparametric curves from.

NumOfIsocurves: In each (U or V) direction.

SamplesPerCurve: Fineness control on piecewise linear curve approximation.

Optimal: Use optimal approximation of isocurves.

Returns: List of polylines representing a piecewise linear approximation of the extracted isoparamteric curves or NULL is case of an error.

Description: Routine to convert a single trimmed surface to NumOfIsolines polylines in each parametric direction with SamplesPerCurve in each isoparametric curve. Polyline are always E3 of CagdPolylineStruct type. Iso parametric curves are sampled equally spaced in parametric space. NULL is returned in case of an error, otherwise list of CagdPolylineStruct. Attempt is made to extract isolines along C1 discontinuities first.

8.2.21 TrimSrfCopyList (trim_lib/trim_gen.c:410)

copy

```
TrimSrfStruct *TrimSrfCopyList(TrimSrfStruct *TrimSrfList)
```

TrimSrfList: To be copied.

Returns: A duplicated list of trimming surfaces.

Description: Allocates and copies a list of trimming surface structures.

8.2.22 TrimSrfDegreeRaise (trim_lib/trim_aux.c:81)

degree raising

```
TrimSrfStruct *TrimSrfDegreeRaise(TrimSrfStruct *TrimSrf, CagdSrfDirType Dir)
```

TrimSrf: To raise its degree.

Returns: A surface with same geometry as Srf but with one degree higher.

Description: Returns a new trimmed surface representing the same surface as TrimSrf but with its degree raised by one.

8.2.23 TrimSrfDomain (trim_lib/trim_aux.c:35)

domain

parametric domain

```
void TrimSrfDomain(TrimSrfStruct *TrimSrf,
                   CagdRType *UMin,
                   CagdRType *UMax,
                   CagdRType *VMin,
                   CagdRType *VMax)
```

TrimSrf: To get its parametric domain.

UMin: Where to put the minimal U domain's boundary.

UMax: Where to put the maximal U domain's boundary.

VMin: Where to put the minimal V domain's boundary.

VMax: Where to put the maximal V domain's boundary.

Description: Returns the parametric domain of a trimmed surface.

8.2.24 TrimSrfDomain (trim_lib/trim_gen.c:290)

allocation

```
TrimSrfStruct *TrimSrfNew(CagdSrfStruct *Srf,
                          TrimCrvStruct *TrimCrvList,
                          CagdBType HasTopLvlTrim)
```

Srf: Surface to make into a trimmed surface.

TrimCrvList: A list of trimming curves.

HasTopLvlTrim: Do we have a top level outer most trimming curve?

Returns: The trimmed surface.

Description: Constructor for a trimmed surface.

8.2.25 TrimSrfDomain (trim_lib/trim_gen.c:350)

allocation

```
TrimSrfStruct *TrimSrfNew2(CagdSrfStruct *Srf,
                           CagdCrvStruct *TrimCrvList,
                           CagdBType HasTopLvlTrim)
```

Srf: Surface to make into a trimmed surface.

CagdCrvStruct: A list of trimming curves, as regular curves.

HasTopLvlTrim: Do we have a top level outer most trimming curve?

Returns: The trimmed surface.

Description: Constructor for a trimmed surface.

8.2.26 TrimSrfEval (trim_lib/trim_aux.c:61)

evaluation

```
CagdRType *TrimSrfEval(TrimSrfStruct *TrimSrf, CagdRType u, CagdRType v)
```

TrimSrf: To evaluate at the given parametric location (u, v).

u, v: The parameter values at which TrimSrf is to be evaluated.

Returns: A vector holding all the coefficients of all components of surface TrimSrf's point type. If, for example, TrimSrf's point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Given a trimmed surface and parameter values u, v, evaluate the surface at (u, v).

8.2.27 TrimSrfFree (trim_lib/trim_gen.c:439)

allocation

```
void TrimSrfFree(TrimSrfStruct *TrimSrf)
```

TrimSrf: A trimmed surface to free.

Description: Deallocates a trimmed surface structure.

8.2.28 TrimSrfFreeList (trim_lib/trim_gen.c:460)

allocation

```
void TrimSrfFreeList(TrimSrfStruct *TrimSrfList)
```

TrimSrfList: A list of trimmed surface to free.

Description: Deallocates a list of trimmed surface structures.

8.2.29 TrimSrfMatTransform (trim_lib/trim_gen.c:527)

Trimariates

```
void TrimSrfMatTransform(TrimSrfStruct *TrimSrf, CagdMType Mat)
```

TV: Trimariate to transform.

Mat: Homogeneous transformation to apply to TV.

Description: Transforms, in place, the given TV as specified by homogeneous matrix Mat.

8.2.30 TrimSrfNew (trim_lib/trim_gen.c:382)

allocation

```
TrimSrfStruct *TrimSrfCopy(TrimSrfStruct *TrimSrf)
```

TrimSrf: A trimming surface to duplicate.

Returns: A trimming surface structure.

Description: Duplicates a trimming surface structure.

8.2.31 TrimSrfRefineAtParams (trim_lib/trim_aux.c:206)

refinement

subdivision

```
TrimSrfStruct *TrimSrfRefineAtParams(TrimSrfStruct *TrimSrf,
                                     CagdSrfDirType Dir,
                                     CagdBType Replace,
                                     CagdRType *t,
                                     int n)
```

TrimSrf: To refine.

Dir: Direction of refinement. Either U or V.

Replace: If TRUE, t holds knots in exactly the same length as the length of the knot vector of Srf and t simply replaces the knot vector.

t: Vector of knots with length of n.

n: Length of vector t.

Returns: A refined surface of TrimSrf after insertion of all the knots as specified by vector t of length n.

Description: Given a trimmed surface - refines it at the given n knots as defined by vector t. If Replace is TRUE, the values in t replaces current knot vector. Returns pointer to refined surface (Note a Bezier surface will be converted into a Bspline surface).

8.2.32 TrimSrfRegionFromTrimSrf (trim_lib/trim_aux.c:135)

regions

subdivision

```
TrimSrfStruct *TrimSrfRegionFromTrimSrf(TrimSrfStruct *TrimSrf,
                                        CagdRType t1,
                                        CagdRType t2,
                                        CagdSrfDirType Dir)
```

TrimSrf: To extract a sub-region from.

t1, t2: Parametric domain boundaries of sub-region.

Dir: Direction of region extraction. Either U or V.

Returns: Sub-region extracted from TrimSrf from t1 to t2.

Description: Given a trimmed surface - extracts a sub-region within the domain specified by t1 and t2, in the direction Dir.

8.2.33 TrimSrfReverse (trim_lib/trim_aux.c:274)

reverse

```
TrimSrfStruct *TrimSrfReverse2(TrimSrfStruct *TrimSrf)
```

TrimSrf: To be reversed.

Returns: Reversed surface of TrimSrf.

Description: Returns a new trimmed surface that is the reversed surface of Srf by flipping the U and the V directions of the surface, as well as flipping them in the trimming curves. See also BspKnotReverse.

8.2.34 TrimSrfReverse (trim_lib/trim_aux.c:228)

reverse

```
TrimSrfStruct *TrimSrfReverse(TrimSrfStruct *TrimSrf)
```

TrimSrf: To be reversed.

Returns: Reversed surface of TrimSrf.

Description: Returns a new trimmed surface that is the reversed surface of TrimSrf by reversing the control mesh and the knot vector (if Bspline surface) of TrimSrf in the U direction, as well as its trimming curves. See also CagdSrfReverse and BspKnotReverse.

8.2.35 TrimSrfSubdivAtParam (trim_lib/trim_aux.c:110)

subdivision

```
TrimSrfStruct *TrimSrfSubdivAtParam(TrimSrfStruct *TrimSrf,
                                    CagdRType t,
                                    CagdSrfDirType Dir)
```

TrimSrf: To subdivide at the prescibed parameter value t.

t: The parameter to subdivide the curve Crv at.

Dir: Direction of subdivision. Either U or V.

Returns: The subdivided surfaces. Usually two, but can have only one, if other is totally trimmed away.

Description: Given a trimmed surface - subdivides it into two sub-surfaces at given parametric value t in the given direction Dir. Returns pointer to a list of two trimmed surfaces, at most. It can very well may happen that the subdivided surface is completely trimmed out and hence nothing is returned for it.

8.2.36 TrimSrfTransformd (trim_lib/trim_gen.c:489)

```
void TrimSrfTransform(TrimSrfStruct *TrimSrf,
                      CagdRType *Translate,
                      CagdRType Scale)
```

TrimSrf: Trimmed surface to transform.

Translate: Translation factor.

Scale: Scaling factor.

Description: Linearly transforms, in place, given trimmed surface as specified by Translate and Scale.

Chapter 9

Trivariate Library, triv_lib

9.1 General Information

This library provides a rich set of function to manipulate freeform Bezier and/or NURBs trivariate. This library heavily depends on the cagd library. Functions are provided to create, copy, and destruct trivariates, to extract isoparametric surfaces, to evaluate, refine and subdivide, to read and write trivariates, to differentiate, degree raise, make compatible and approximate iso-surface at iso values using polygonal representations.

The interface of the library is defined in *include/triv_lib.h*.

This library has its own error handler, which by default prints an error message and exit the program called **TrivFatalError**.

All globals in this library have a prefix of **Triv**.

9.2 Library Functions

9.2.1 MCThresholdCube (triv_lib/mrchcube.c:66)

marching cubes

```
MCPolygonStruct *MCThresholdCube(MCCubeCornerScalarStruct *CCS,
                                 RealType Threshold)
```

CCS: The cube's dimensions/information.

Threshold: Iso surface level.

Returns: List of polygons (not necessarily triangles), or possibly NULL.

Description: Given 8 cube corner values (scalars), compute the polygon(s) in this cube along the isosurface at Threshold. if CCS has gradient information, it is used to approximate normals at the vertices.

```
         7            K           6
          ***********************
        * +                   * *
    L *   +                 *   *
    *     +     I         * J   *
4 *********************** 5     *
  *       +             *       *
  *       +             *       * G
  *       + H           *       *
  *       +             *       *
  *       +             * F     *
E *       +       C     *       *
  *       ++++++++++++++*+++++++* 2
  *   D + 3             *     *
  *   +                 *   * B
  * +                   * *
  ***********************
 0           A           1
```

```
Vertices 0 - 7
Edges    A - L
```

9.2.2 TrivBspNew (triv_lib/triv_gen.c:91)

trivariates

allocation

```
TrivTVStruct *TrivBspTVNew(int ULength,
                           int VLength,
                           int WLength,
                           int UOrder,
                           int VOrder,
                           int WOrder,
                           CagdPointType PType)
```

ULength: Number of control points in the U direction.
VLength: Number of control points in the V direction.
WLength: Number of control points in the W direction.
UOrder: Order of trivariate in the U direction.
VOrder: Order of trivariate in the V direction.
WOrder: Order of trivariate in the W direction.
PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform trivariate Bspline.

Description: Allocates the memory required for a new Bspline trivariate.

9.2.3 TrivBspTVDegreeRaise (triv_lib/trivrais.c:173)

degree raising

```
TrivTVStruct *TrivBspTVDegreeRaise(TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To raise it degree by one.
Dir: Direction of degree raising. Either U, V or W.

Returns: A trivariate with one degree higher in direction Dir, representing the same geometry as TV.

Description: Returns a new Bspline surface, identical to the original but with one degree higher, in the requested direction Dir.

9.2.4 TrivBspTVDerive (triv_lib/triv_der.c:152)

trivariates

```
TrivTVStruct *TrivBspTVDerive(TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.
Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir. A Bspline trivariate.

Description: Given a Bspline trivariate, computes its partial derivative trivariate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)) / (Kv(i + k) - Kv(i + 1)), i = 0 to k-2.
```

9.2.5 TrivBspTVKnotInsertNDiff (triv_lib/triv_ref.c:84)

trivariates

```
TrivTVStruct *TrivBspTVKnotInsertNDiff(TrivTVStruct *TV,
                                       TrivTVDirType Dir,
                                       int Replace,
                                       CagdRType *t,
                                       int n)
```

TV: Trivariate to refine according to t in direction Dir.

Replace: If TRUE t is a knot vector exaclt in the length of the knot vector in direction Dir in TV and t simply replaces than knot vector. If FALSE, the knot vector in direction Dir in TV is refined by adding all the knots in t.

t: Knot vector to refine/replace the knot vector of TV in direction Dir.

n: Length of vector t.

Returns: The refined trivariate. A Bspline trivariate.

Description: Given a Bspline trivariate, inserts n knots with different values as defined by t. If, however, Replace is TRUE, the knot are simply replacing the current knot vector in the prescribed direction.

9.2.6 TrivBzrNew (triv_lib/triv_gen.c:135)

trivariates

allocation

```
TrivTVStruct *TrivBzrTVNew(int ULength,
                           int VLength,
                           int WLength,
                           CagdPointType PType)
```

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

PType: Type of control points (E2, P3, etc.).

Returns: An uninitialized freeform trivariate Bezier.

Description: Allocates the memory required for a new Bezier trivariate.

9.2.7 TrivBzrTVDegreeRaise (triv_lib/trivrais.c:67)

degree raising

```
TrivTVStruct *TrivBzrTVDegreeRaise(TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To raise it degree by one.

Dir: Direction of degree raising. Either U, V or W.

Returns: A surface with one degree higher in direction Dir, representing the same geometry as TV.

Description: Returns a new Bezier trivariate, identical to the original but with one degree higher, in the requested direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
                      i           k-i
Q(0) = P(0), Q(i) = --- P(i-1) + (---) P(i), Q(k) = P(k-1).
                      k            k
```

This is applied to all rows/cols of the trivariate.

9.2.8 TrivBzrTVDerive (triv_lib/triv_der.c:64)

trivariates

```
TrivTVStruct *TrivBzrTVDerive(TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir. A Bezier trivariate.

Description: Given a Bezier trivariate, computes its partial derivative trivariate in direction Dir. Let old control polygon be P(i), i = 0 to k-1, and Q(i) be new one then:

```
Q(i) = (k - 1) * (P(i+1) - P(i)), i = 0 to k-2.
```

9.2.9 TrivCnvrtBezier2BsplineTV (triv_lib/triv_gen.c:505)

conversion

trivariate

```
TrivTVStruct *TrivCnvrtBezier2BsplineTV(TrivTVStruct *TV)
```

TV: A Bezier trivariate to convert to a Bspline TV.

Returns: A Bspline trivariate representing the same geometry as the given Bezier TV.

Description: Converts a Bezier trivariate into a Bspline trivariate by adding two open end uniform knot vectors to it.

9.2.10 TrivCoerceTVTo (triv_lib/trivcoer.c:23)

coercion

```
TrivTVStruct *TrivCoerceTVTo(TrivTVStruct *TV, CagdPointType PType)
```

TV: To coerce to a new point type PType.

PType: New point type for TV.

Returns: A new trivariate with PType as its point type.

Description: Coerces a trivariate to point type PType.

9.2.11 TrivDbg (triv_lib/triv_dbg.c:23)

debugging

```
void TrivDbg(void *Obj)
```

Obj: A trivariate - to be printed to stderr.

Description: Prints trivariates stderr. Should be linked to programs for debugging purposes, so trivariates may be inspected from a debugger.

9.2.12 TrivDescribeError (triv_lib/triv_err.c:57)

error handling

```
char *TrivDescribeError(TrivFatalErrorType ErrorNum)
```

ErrorNum: Type of the error that was raised.

Returns: A string describing the error type.

Description: Returns a string describing a the given error. Errors can be raised by any member of this triv library as well as other users. Raised error will cause an invokation of TrivFatalError function which decides how to handle this error. TrivFatalError can for example, invoke this routine with the error type, print the appropriate message and quit the program.

9.2.13 TrivFatalError (triv_lib/triv_ftl.c:25)

error handling

```
void TrivFatalError(TrivFatalErrorType ErrID)
```

ErrID: Error type that was raised.

Description: Trap Triv_lib errors right here. Provides a default error handler for the triv library. Gets an error description using TrivDescribeError, prints it and exit the program using exit.

9.2.14 TrivInterpTrivar (triv_lib/trinterp.c:24)

interpolation

```
TrivTVStruct *TrivInterpTrivar(TrivTVStruct *TV)
```

TV: Trivariate to interpolate its control mesh.

Returns: The interpolating trivariate.

Description: Interpolates control points of given trivariate, preserving the order and continuity of the original trivariate.

9.2.15 TrivMakeTVsCompatible (triv_lib/trivcmpt.c:48)

compatibility

```
CagdBType TrivMakeTVsCompatible(TrivTVStruct **TV1,
                                TrivTVStruct **TV2,
                                CagdBType SameUOrder,
                                CagdBType SameVOrder,
                                CagdBType SameWOrder,
                                CagdBType SameUKV,
                                CagdBType SameVKV,
                                CagdBType SameWKV)
```

TV1, TV2: Two surfaces to be made compatible, in place.

SameUOrder: If TRUE, this routine make sure they share the same U order.

SameVOrder: If TRUE, this routine make sure they share the same order.

SameWOrder: If TRUE, this routine make sure they share the same W order.

SameUKV: If TRUE, this routine make sure they share the same U knot vector and hence continuity. *

SameVKV: If TRUE, this routine make sure they share the same knot vector and hence continuity.

SameWKV: If TRUE, this routine make sure they share the same W knot vector and hence continuity.

Returns: TRUE if successful, FALSE otherwise.

Description: Given two trivariates, makes them compatible by:

1. Coercing their point type to be the same.
2. Making them have the same curve type.
3. Raising the degree of the lower one to be the same as the higher.
4. Refining them to a common knot vector (If Bspline and SameOrder).

Note 3 is performed if SameOrder TRUE, 4 if SameKV TRUE. Both trivariates are modified IN PLACE.

9.2.16 TrivParamInDomain (triv_lib/triv_aux.c:80)

trivariates

```
CagdBType TrivParamInDomain(TrivTVStruct *TV, CagdRType t, TrivTVDirType Dir)
```

TV: To make sure t is in its Dir domain.

t: Parameter value to verify.

Dir: Direction. Either U or V or W.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a tri-variate and a domain - validate it.

9.2.17 TrivParamsInDomain (triv_lib/triv_aux.c:116)

trivariates

```
CagdBType TrivParamsInDomain(TrivTVStruct *TV,
                             CagdRType u,
                             CagdRType v,
                             CagdRType w)
```

TV: To make sure (u, v, w) is in its domain.

u, v, w: To verify if it is in TV's parametric domain.

Returns: TRUE if in domain, FALSE otherwise.

Description: Given a tri-variate and a domain - validate it.

9.2.18 TrivPlaneFrom4Points (triv_lib/geomat4d.c:42)

hyper plane

plane

```
int TrivPlaneFrom4Points(TrivPType Pt1,
                         TrivPType Pt2,
                         TrivPType Pt3,
                         TrivPType Pt4,
                         TrivPlaneType Plane)
```

Pt1, Pt2, Pt3, Pt4: The four points the plane should go through.

Plane: Where the result should be placed.

Returns: TRUE if successful, FALSE otherwise.

Description: Computes a hyperplane in four space through the given four points. Based on a direct solution in Maple of:

```
with(linalg);
readlib(C);



d := det( matrix( [ [x - x1, y - y1, z - z1, w - w1],
                    [x2 - x1, y2 - y1, z2 - z1, w2 - w1],
                    [x3 - x2, y3 - y2, z3 - z2, w3 - w2],
                    [x4 - x3, y4 - y3, z4 - z3, w4 - w3]] ) );
coeff( d, x );
coeff( d, y );
coeff( d, z );
coeff( d, w );
```

9.2.19 TrivSrfFromMesh (triv_lib/triveval.c:361)

trivariates

```
CagdSrfStruct *TrivSrfFromMesh(TrivTVStruct *TV,
                               int Index,
                               TrivTVDirType Dir)
```

TV: Trivariate to extract a bivariate surface out of its mesh.

Index: Index of row/column/level of TV's mesh in direction Dir.

Dir: Direction of isosurface extraction. Either U or V or W.

Returns: A bivariate surface which was extracted from TV's Mesh. This surface is not necessarily on TV.

Description: Extract a bivariate surface out of the given trivariate's mesh. The provided (zero based) Index specifies which Index to extract.

9.2.20 TrivSrfFromTV (triv_lib/triveval.c:182)

trivariates

```
CagdSrfStruct *TrivSrfFromTV(TrivTVStruct *TV,
                             CagdRType t,
                             TrivTVDirType Dir)
```

TV: To extract an isoparametric surface from at parameter value t in direction Dir.

t: Parameter value at which to extract the isosurface.

Dir: Direction of isosurface extraction. Either U or V or W.

Returns: A bivariate surface which is an isosurface of TV.

Description: Extract an isoparametric surface out of the given tensor product trivariate. Operations should favor the CONST_W_DIR, in which the extraction is somewhat faster, if it is possible.

9.2.21 TrivSrfToMesh (triv_lib/triveval.c:490)

trivariates

```
void TrivSrfToMesh(CagdSrfStruct *Srf,
                   int Index,
                   TrivTVDirType Dir,
                   TrivTVStruct *TV)
```

Srf: Surface to substitute into the trivariate TV.
Index: Index of row/column/level of TV's mesh in direction Dir.
Dir: Direction of isosurface extraction. Either U or V or W.
TV: Trivariate to substitute a bivariate surface into its mesh.

Description: Substitute a bivariate surface into a given trivariate's mesh. The provided (zero based) Index specifies which Index to extract.

9.2.22 TrivTV2CtrlMesh (triv_lib/trivmesh.c:22)

control mesh

```
CagdPolylineStruct *TrivTV2CtrlMesh(TrivTVStruct *Trivar)
```

Srf: To extract a control mesh from.

Returns: The control mesh of Srf.

Description: Extracts the control mesh of a surface as a list of polylines.

9.2.23 TrivTVBBox (triv_lib/triv_aux.c:217)

bbox

bounding box

```
void TrivTVBBox(TrivTVStruct *Trivar, CagdBBoxStruct *BBox)
```

Trivar: To compute a bounding box for.
BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a trivariate freeform function.

9.2.24 TrivTVCopy (triv_lib/triv_gen.c:162)

trivariates

```
TrivTVStruct *TrivTVCopy(TrivTVStruct *TV)
```

TV: Trivariate to duplicate

Returns: Duplicated trivariate.

Description: Allocates and duplicates all slots of a trivariate structure.

9.2.25 TrivTVCopyList (triv_lib/triv_gen.c:223)

trivariates

```
TrivTVStruct *TrivTVCopyList(TrivTVStruct *TVList)
```

TVList: List of trivariates to duplicate.

Returns: Duplicated list of trivariates.

Description: Duplicates a list of trivariate structures.

9.2.26 TrivTVDegreeRaise (triv_lib/trivrais.c:33)

degree raising

```
TrivTVStruct *TrivTVDegreeRaise(TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: To raise its degree.
Dir: Direction of degree raising. Either U, V or W.

Returns: A surface with same geometry as TV but with one degree higher.

Description: Returns a new trivariate representing the same curve as TV but with its degree raised by one.

9.2.27 TrivTVDerive (triv_lib/triv_der.c:33)

trivariates

```
TrivTVStruct *TrivTVDerive(TrivTVStruct *TV, TrivTVDirType Dir)
```

TV: Trivariate to differentiate.

Dir: Direction of differentiation. Either U or V or W.

Returns: Differentiated trivariate in direction Dir.

Description: Given a trivariate, computes its partial derivative trivariate in direction Dir.

9.2.28 TrivTVDomain (triv_lib/triv_aux.c:31)

trivariates

```
void TrivTVDomain(TrivTVStruct *TV,
                  CagdRType *UMin,
                  CagdRType *UMax,
                  CagdRType *VMin,
                  CagdRType *VMax,
                  CagdRType *WMin,
                  CagdRType *WMax)
```

TV: Trivariate function to consider.

UMin, UMax: U Domain of TV will be placed herein.

VMin, VMax: V Domain of TV will be placed herein.

WMin, WMax: W Domain of TV will be placed herein.

Description: Given a tri-variate, returns its parametric domain.

9.2.29 TrivTVEval (triv_lib/triveval.c:43)

evaluation

trivariates

```
CagdRType *TrivTVEval(TrivTVStruct *TV, CagdRType u, CagdRType v, CagdRType w)
```

TV: To evaluate at given (u, v, w) parametric location.

u, v, w: Parametric location to evaluate TV at.

Returns: A vector holding all the coefficients of all components of the trivariate's point type. If for example trivariate point type is P2, the W, X, and Y will be saved in the first three locations of the returned vector. The first location (index 0) of the returned vector is reserved for the rational coefficient W and XYZ always starts at second location of the returned vector (index 1).

Description: Evaluates the given tensor product trivariate at a given point, by extracting an isoparamteric surface along w and evaluating (u,v) in it. Not very efficient way to evaluate the tri-variate...

```
      +-----------------------+
    W  /                     /|
   /  /                     / |
  /  /     U -->           /  |
    +-----------------------+  |  Control tri-variate mesh orientation.
 V | |P0              Pi-1|   +
   v |Pi             P2i-1|  /
     |                    | /
     |Pn-i            Pn-1|/
     +-----------------------+
```

9.2.30 TrivTVFree (triv_lib/triv_gen.c:252)

trivariates

```
void TrivTVFree(TrivTVStruct *TV)
```

TV: Trivariate to free.

Description: Deallocates and frees all slots of a trivariate structure.

9.2.31 TrivTVFreeList (triv_lib/triv_gen.c:288)

trivariates

```
void TrivTVFreeList(TrivTVStruct *TVList)
```

TVList: Trivariate list to free.

Description: Deallocates and frees a list of trivariate structures.

9.2.32 TrivTVFromSrfs (triv_lib/trivstrv.c:31)

trivar constructors

```
TrivTVStruct *TrivTVFromSrfs(CagdSrfStruct *SrfList, int OtherOrder)
```

SrfList: List of surfaces to consturct a trivariate with.

OtherOrder: Other, third, order of trivariate.

Returns: Constructed trivariate from surfaces.

Description: Constructs a trivariate using a set of surfaces. Surfaces are made to be compatible and then each is substituted into the new trivariate's mesh as a row. If the OtherOrder is less than the number of curves, number of curves is used. A knot vector is formed with uniform open end for the other direction, so it interpolates the first and last surfaces. Note, however, that only the first and the last surfaces are interpolated if OtherOrder is greater than 2.

9.2.33 TrivTVListBBox (triv_lib/triv_aux.c:244)

bbox

bounding box

```
void TrivTVListBBox(TrivTVStruct *TVs, CagdBBoxStruct *BBox)
```

Trivars: To compute a bounding box for.

BBox: Where bounding information is to be saved.

Description: Computes a bounding box for a list of trivariate freeform function.

9.2.34 TrivTVMatTransform (triv_lib/triv_gen.c:473)

trivariates

```
void TrivTVMatTransform(TrivTVStruct *TV, CagdMType Mat)
```

TV: Trivariate to transform.

Mat: Homogeneous transformation to apply to TV.

Description: Transforms, in place, the given TV as specified by homogeneous matrix Mat.

9.2.35 TrivTVNew (triv_lib/triv_gen.c:33)

trivariates

allocation

```
TrivTVStruct *TrivTVNew(TrivGeomType GType,
                        CagdPointType PType,
                        int ULength,
                        int VLength,
                        int WLength)
```

GType: Type of geometry the curve should be - Bspline, Bezier etc.

PType: Type of control points (E2, P3, etc.).

ULength: Number of control points in the U direction.

VLength: Number of control points in the V direction.

WLength: Number of control points in the W direction.

Returns: An uninitialized freeform trivariate.

Description: Allocates the memory required for a new trivariate.

9.2.36 TrivTVRefineAtParams (triv_lib/triv_ref.c:39)

trivariates

```
TrivTVStruct *TrivTVRefineAtParams(TrivTVStruct *TV,
                                   TrivTVDirType Dir,
                                   CagdBType Replace,
                                   CagdRType *t,
                                   int n)
```

TV: Trivariate to refine according to t in direction Dir.

Dir: Direction of refinement. Either U or V or W.

Replace: If TRUE t is a knot vector exaclt in the length of the knot vector in direction Dir in TV and t simply replaces than knot vector. If FALSE, the knot vector in direction Dir in TV is refined by adding all the knots in t.

t: Knot vector to refine/replace the knot vector of TV in direction Dir.

n: Length of vector t.

Returns: The refined trivariate. Always a Bspline trivariate.

Description: Given a trivariate, refines it at the given n knots as defined by the vector t. If Replace is TRUE, the values replace the current knot vector. Returns pointer to refined TV (Note a Bezier trivariate will be converted into a Bspline trivariate).

9.2.37 TrivTVRegionFromTV (triv_lib/triv_aux.c:145)

trivariates

```
TrivTVStruct *TrivTVRegionFromTV(TrivTVStruct *TV,
                                 CagdRType t1,
                                 CagdRType t2,
                                 TrivTVDirType Dir)
```

TV: To extract asub-region from.

t1, t1: Domain to extract from TV, in parametric direction Dir.

Dir: Direction to extract the sub-region. Either U or V or W.

Returns: A sub-region of TV from t1 to t2 in direction Dir.

Description: Given a tri-variate, returns a sub-region of it.

9.2.38 TrivTVSubdivAtParam (triv_lib/triv_sub.c:27)

trivariates

```
TrivTVStruct *TrivTVSubdivAtParam(TrivTVStruct *TV,
                                  CagdRType t,
                                  TrivTVDirType Dir)
```

TV: Trivariate to subdivide.

t: Parameter to subdivide at.

Dir: Direction of subdivision.

Returns: A list of two trivariates, result of the subdivision.

Description: Given a tri-variate, subdivides it at parameter value t in direction Dir.

9.2.39 TrivTVTransform (triv_lib/triv_gen.c:441)

trivariates

```
void TrivTVTransform(TrivTVStruct *TV, CagdRType *Translate, CagdRType Scale)
```

TV: Trivariate to transform.

Translate: Translation factor.

Scale: Scaling factor.

Description: Linearly transforms, in place, given TV as specified by Translate and Scale.

9.2.40 TrivTriangleCopy (triv_lib/triv_gen.c:336)

```
TrivTriangleStruct *TrivTriangleCopy(TrivTriangleStruct *Triangle)
```

TrivTriangleStruct *: Triangle to duplicate.

Returns: Duplicated triangle.

Description: Allocates and duplicates all slots of a triangle structure.

9.2.41 TrivTriangleCopyList (triv_lib/triv_gen.c:363)

```
TrivTriangleStruct *TrivTriangleCopyList(TrivTriangleStruct *TriangleList)
```

TriangleList: List of triangle to duplicate.

Returns: Duplicated list of triangle.

Description: Duplicates a list of triangle structures.

9.2.42 TrivTriangleFree (triv_lib/triv_gen.c:392)

```
void TrivTriangleFree(TrivTriangleStruct *Triangle)
```

Triangle: Triangle to free.

Description: Deallocates and frees all slots of a triangle structure.

9.2.43 TrivTriangleFreeList (triv_lib/triv_gen.c:414)

```
void TrivTriangleFreeList(TrivTriangleStruct *TriangleList)
```

TriangleList: Triangle list to free.

Description: Deallocates and frees a list of triangle structures.

9.2.44 TrivTriangleNew (triv_lib/triv_gen.c:312)

allocation

```
TrivTriangleStruct *TrivTriangleNew(void)
```

Returns: An uninitialized triangle.

Description: Allocates the memory required for a new triangle.

Chapter 10

Extra Library, xtra_lib

10.1 General Information

This library is not an official part of IRIT and contains public domain code that is used by routines in IRIT.
The interface of the library is defined in *include/extra_fn.h*.

10.2 Library Functions

10.2.1 BzrCrvInterp (xtra_lib/bzrintrp.c:204)

interpolation

```
void BzrCrvInterp(RealType *Result, RealType *Input, int Size)
```

Result: Where the interpolated control points will be placed.

Input: Points to interpolate at node parameter values.

Size: Of control polygon.

Description: Blends the input points using the Interp array and puts the resulted blended point in Result array. Input and Result array are of size Size.

10.2.2 SvdLeastSqr (xtra_lib/nure_svd.c:276)

singular value decomposition

linear systems

```
void SvdLeastSqr(RealType *A, RealType *x, RealType *b, int NData, int Nx)
```

A: The matrix of size Nx by NData.

x: The vector of sought solution of size Nx.

b: The vector of coefficients of size NData.

NData, Nx: Dimensions of input.

Description: Least square solves A x = b. The vector X is of size Nx, vector b is of size NData and matrix A is of size Nx by NData. Uses singular value decomposition. If A != NULL is SVD decomposition is computed, otherwise (A == NULL) a solution is computed for the given b and is placed in x.

Chapter 11

Programming Examples

This chapter describes several simple examples of C programs that exploits the libraries of IRIT. All external function are defined in the *include* subdirectory of IRIT and one can 'grep' there for the exact include file that contains a certain function name. All C programs in all C files should include 'irit_sm.h' as their first include file of IRIT, before any other include file of IRIT. Header files are set so C++ code can compile and link to it without any special treatment.

11.1 Setting up the Compilation Environment

In order to compile programs that uses the libraries of IRIT, a makefile has to be constructed. Assuming IRIT is installed in */usr/local/irit*, here is a simple makefile that can be used (for a unix environment):

```
IRIT_DIR = /usr/local/irit

include $(IRIT_DIR)/makeflag.unx

OBJS = program.o

program: $(OBJS)
$(CC) $(CFLAGS) -o program $(OBJS) $(LIBS) -lm $(MORELIBS)
```

The simplicity of this makefile is drawn from the complexity of makeflag.unx. The file makeflag.unx sets the CC, CFLAGS, LIBS, and MORELIBS for the machined using among other things. Furthermore, makeflag.unx also sets the default compilation rules from C sources to object files. The file makeflag.unx had to be modified once, when IRIT was installed on this system. If the used system is not a unix environment, then the file makefile.unx will have to be replaced with the proper makeflag file. In an OS2 environment, using the emx gcc compiler, the makefile is even simpler since the linking rule is also defined in makeflag.os2:

```
IRIT_DIR = \usr\local\irit

include $(IRIT_DIR)\makeflag.os2

OBJS = program.o

program.exe: $(OBJS)
```

Finally, here is a proper makefile for Windows NT:

```
IRIT_DIR = \usr\local\irit

include $(IRIT_DIR)\makeflag.wnt

OBJS = program.obj

program.exe: $(OBJS)
        $(IRITCONLINK) -out:program.exe $(OBJS) $(LIBS) $(W32CONLIBS)
```

11.2 Simple C Programs using IRIT

Now that we have an idea how to compile C code using IRIT, here are several examples to read, manipulate and write IRIT data files. You will be able to find all these examples in the doc/cexample directory.

11.2.1 Compute Area of a Polygonal Model (polyarea.c)

Here is a simple program to compute the total area of all polygons in the given data file. The program expects one argument on the command line which is the name of the file to read, and it prints out one line with the total computed area.

```
#include "irit_sm.h"
#include "iritprsr.h"
#include "allocate.h"
#include "geomvals.h"

void main(int argc, char **argv)
{
    int Handler;

    if (argc == 2) {
        if ((Handler = IritPrsrOpenDataFile(argv[1], TRUE, TRUE)) >= 0) {
            IPObjectStruct
                *PObj = IritPrsrGetObjects(Handler);

            /* Done with file - close it. */
            IritPrsrCloseStream(Handler, TRUE);

            /* Process the geometry - compute the accumulated area. */
            if (IP_IS_POLY_OBJ(PObj) && IP_IS_POLYGON_OBJ(PObj))
                fprintf(stderr, "Area of polyhedra is %lf\n",
                        PolyObjectArea(PObj));
            else
                fprintf(stderr, "Read object is not a polyhedra.\n");

            IPFreeObject(PObj);
        }
        else {
            fprintf(stderr, "Failed to open file \"%s\"\n", argv[1]);
            exit(1);
        }
    }
    else {
        fprintf(stderr, "Usage: PolyArea geom.dat\n");
        exit(2);
    }

    exit(0);
}
```

11.2.2 Converts a Freeform Surface into Polygons (polygons.c)

This true filter reads a single surface from stdin and dumps out a polygonal approximation of it to stdout. They are several parameters that controls the way a surface is approximated into polygons and in this simple filter they are being held fixed in a set of integer variables.

```
#include "irit_sm.h"
#include "cagd_lib.h"
#include "iritprsr.h"
#include "ip_cnvrt.h"
#include "allocate.h"

void main(int argc, char **argv)
{
    int Handler,
        FourPerFlat = TRUE, /* Settable parameters of IritSurface2Polygons. */
        FineNess = 20,
        ComputeUV = FALSE,
        Optimal = FALSE;
```

```
    if ((Handler = IritPrsrOpenDataFile("-", TRUE, TRUE)) >= 0) {
        IPObjectStruct
            *PObj = IritPrsrGetObjects(Handler);

        /* Done with file - close it. */
        IritPrsrCloseStream(Handler, TRUE);

        /* Process the surface into polygons. */
        if (IP_IS_SRF_OBJ(PObj)) {
            IPPolygonStruct
                *PPoly = IritSurface2Polygons(PObj -> U.Srfs, FourPerFlat,
                                              FineNess, ComputeUV, Optimal);
            IPObjectStruct
                *PObjPoly = GenPOLYObject(PPoly);

            IritPrsrStdoutObject(PObjPoly);

            IPFreeObject(PObjPoly);
        }
        else
            fprintf(stderr, "Read object is not a surface.\n");

        IPFreeObject(PObj);
    }
    else {
        fprintf(stderr, "Failed to read from stdin\n");
        exit(1);
    }

    exit(0);
}
```

11.2.3 Linear Transformations' Filter (transfrm.c)

This little more complex program transfroms all the geometry in the read data which can be any number of files according to the specified transformations on the command line. The command line is parsed via **GAGetArgs** and its associated functions. The transformation matrix is then computed with the aid of the matrix package and applied to the read geometry at once.

```
#include "irit_sm.h"
#include "allocate.h"
#include "iritprsr.h"
#include "geomat3d.h"
#include "genmat.h"
#include "getarg.h"

static char *CtrlStr =
#ifdef DOUBLE
    "Transfrm x%-Degs!F y%-Degs!F z%-Degs!F t%-X|Y|Z!F!F!F s%-Scale!F h%- DFiles!*s";
#else
    "Transfrm x%-Degs!f y%-Degs!f z%-Degs!f t%-X|Y|Z!f!f!f s%-Scale!f h%- DFiles!*s";
#endif /* DOUBLE */

void main(int argc, char **argv)
{
    int NumFiles, Error,
        RotXFlag = FALSE,
        RotYFlag = FALSE,
        RotZFlag = FALSE,
        TransFlag = FALSE,
        ScaleFlag = FALSE,
        HelpFlag = FALSE;
```

```
char **FileNames;
RealType RotXDegrees, RotYDegrees, RotZDegrees, TransX, TransY, TransZ,
    Scale;
MatrixType Mat1, TransMat;
IPObjectStruct *PObjs, *PObjsTrans, *PObj;

if ((Error = GAGetArgs(argc, argv, CtrlStr,
                       &RotXFlag, &RotXDegrees,
                       &RotYFlag, &RotYDegrees,
                       &RotZFlag, &RotZDegrees,
                       &TransFlag, &TransX, &TransY, &TransZ,
                       &ScaleFlag, &Scale,
                       &HelpFlag,
                       &NumFiles, &FileNames)) != 0) {
    GAPrintErrMsg(Error);
    GAPrintHowTo(CtrlStr);
    exit(1);
}

if (HelpFlag) {
    fprintf(stderr, "This is Transform...\n");
    GAPrintHowTo(CtrlStr);
    exit(0);
}

if (NumFiles == 0) {
    fprintf(stderr, "No data files to process.\n");
    exit(2);
}

/* Construct the transformation matrix: */
MatGenUnitMat(TransMat);
if (RotXFlag) {
    MatGenMatRotX1(DEG2RAD(RotXDegrees), Mat1);
    MatMultTwo4by4(TransMat, TransMat, Mat1);
}
if (RotYFlag) {
    MatGenMatRotY1(DEG2RAD(RotYDegrees), Mat1);
    MatMultTwo4by4(TransMat, TransMat, Mat1);
}
if (RotZFlag) {
    MatGenMatRotZ1(DEG2RAD(RotZDegrees), Mat1);
    MatMultTwo4by4(TransMat, TransMat, Mat1);
}
if (TransFlag) {
    MatGenMatTrans(TransX, TransY, TransZ, Mat1);
    MatMultTwo4by4(TransMat, TransMat, Mat1);
}
if (ScaleFlag) {
    MatGenMatUnifScale(Scale, Mat1);
    MatMultTwo4by4(TransMat, TransMat, Mat1);
}

/* Get all the data from all the input files. */
PObjs = IritPrsrGetDataFiles(FileNames, NumFiles, TRUE, TRUE);

/* Apply the transformation to all geometry in input file(s) and dump */
/* the transformed geometry to stdout.                                 */

PObjsTrans = GMTransformObjectList(PObjs, TransMat);
for (PObj = PObjsTrans; PObj != NULL; PObj = PObj -> Pnext)
    IritPrsrStdoutObject(PObj);
```

```
    IPFreeObjectList(PObjs);
    IPFreeObjectList(PObjsTrans);

    exit(0);
}
```

Here is the result of running 'transfrm -h':

```
This is Transform...
Usage: Transfrm [-x Degs] [-y Degs] [-z Degs] [-t X Y Z] [-s Scale] [-h] DFiles
```

When you are considering the usefulness of this tool remember that the transformations are applied to the geometry in an internal order which is different from the command line order. That is,

```
transfrm -x 30 -y 30 geometry.dat > tgeometry.dat
```

will compute the exact same transfrom as,

```
transfrm -y 30 -x 30 geometry.dat > tgeometry.dat
```

This, while rotation is not a commutative operation. Nonetheless, you may split the operations. That is:

```
transfrm -y 30 geometry.dat | transfrm -x 30 - > tgeometry.dat
```

or

```
transfrm -x 30 geometry.dat | transfrm -y 30 - > tgeometry.dat
```

will do exactly what one expects.

11.2.4 Least sqaures curve fitting and process communication (lst_sqrs.c)

This example creates a Bspline curve that least squares approximates a given set of data points. In the default values, a quadratic Bspline curve of 10 control points least sqaures approximates a set of 100 three dimensional points. Also shown is a polyline of the original (100) points. Arbitrary number of curves will be displayed every, one for every ¡return¿ keystroke until Q¡return¿ is typed.

```
#include "irit_sm.h"
#include "iritprsr.h"
#include "allocate.h"
#include "attribut.h"
#include "cagd_lib.h"
#include "irit_soc.h"
#include "iritgrap.h"
#include "getarg.h"

static char *CtrlStr =
    "Lst_Sqrs n%-#Pts!d d%-Degree!d f%-DOF!d p%-PrgmName!s h%-";

void main(int argc, char **argv)
{
    int i, Error, PrgmInput, PrgmOutput,
        NumOfPoints = 100,
        NumOfPtsFlag = FALSE,
        Degree = 3,
        DegreeFlag = FALSE,
        NumOfDOF = 10,
        NumOfDOFFlag = FALSE,
        PrgmFlag = FALSE,
        HelpFlag = FALSE;
    char *Err,
        *Program = "x11drvs -s-";

    if ((Error = GAGetArgs(argc, argv, CtrlStr,
                           &NumOfPtsFlag, &NumOfPoints,
                           &DegreeFlag, &Degree,
```

```
                      &NumOfDOFFlag, &NumOfDOF,
                      &PrgmFlag, &Program,
                      &HelpFlag)) != 0) {
        GAPrintErrMsg(Error);
        GAPrintHowTo(CtrlStr);
        exit(1);
    }

    if (HelpFlag) {
        GAPrintHowTo(CtrlStr);
        exit(0);
    }
    if (IritPrsrSrvrExecAndConnect(Program, &PrgmInput, &PrgmOutput, TRUE)) {
        char Line[LINE_LEN];
        IPObjectStruct
            *PClrObj = GenStrObject("command_", "clear", NULL);

        do {
            CagdPtStruct
                *PtList = NULL;
            IPPolygonStruct
                *PPoly = IPAllocPolygon(0, 0, NULL, NULL);
            CagdCrvStruct *Crv;
            IPObjectStruct *PCrvObj, *PPolyObj;

            for (i = 0; i < NumOfPoints; i++) {
                int j;
                IPVertexStruct *V;
                CagdPtStruct
                    *Pt = CagdPtNew();

                if (i == 0) {
                    for (j = 0; j < 3; j++)
                        Pt -> Pt[j] = IritRandom(-1.0, 1.0);
                }
                else {
                    for (j = 0; j < 3; j++)
                        Pt -> Pt[j] = PtList -> Pt[j] + IritRandom(-0.1, 0.1);
                }

                V = IPAllocVertex(0, 0, NULL, PPoly -> PVertex);
                for (j = 0; j < 3; j++)
                    V -> Coord[j] = Pt -> Pt[j];
                PPoly -> PVertex = V;

                LIST_PUSH(Pt, PtList);
            }

            Crv = BspCrvInterpPts(PtList, Degree + 1,
                                  NumOfDOF, CAGD_UNIFORM_PARAM);
            CagdPtFreeList(PtList);

            CagdCrvWriteToFile3(Crv, stdout, 0, "This is from LstSqrs", &Err);

            /* Generate objects out of the geometry and set proper attrs. */
            PCrvObj = GenCRVObject(Crv);
            AttrSetObjectColor(PCrvObj, IG_IRIT_GREEN);

            PPolyObj = GenPOLYObject(PPoly);
            IP_SET_POLYLINE_OBJ(PPolyObj);
            AttrSetObjectColor(PPolyObj, IG_IRIT_YELLOW);
```

```
            /* Clear old data and display our curve and data. */
            SocWriteOneObject(PrgmInput, PClrObj);
            SocWriteOneObject(PrgmInput, PCrvObj);
            SocWriteOneObject(PrgmInput, PPolyObj);

            IPFreeObject(PCrvObj);
            IPFreeObject(PPolyObj);

            gets(Line);
        }
        while (Line[0] != 'q' && Line[0] != 'Q');

        IritPrsrSrvrKillAndDisConnect(TRUE, PrgmInput, PrgmOutput);
    }

    exit(0);
}
```

Index