Chapter 35. Classes Qa thru Rz

Topic: Ignore

rb_blend_spl_sur

Class: Blending, SAT Save and Restore

Purpose: Implements the constant radius rolling ball blend surface.

Derivation: rb_blend_spl_sur : blend_spl_sur : spl_sur : subtrans_object : subtype_object : ACIS_OBJECT : –

SAT Identifier: “rbblnsur”

Filename: kern/kernel/kerngeom/splsur/rb_spl.hxx

Description: This is a straightforward derivation of blend_spl_sur. The ball rolls on two support entities, which may be either curves, surfaces or points. The point-point case is not included because this is always a sphere. The surface-surface case is equivalent to the pipe surface.

Limitations: None

References: None

Data:

None

Constructor:

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const curve& left_crv,   // left curve
    const curve& right_crv,  // right curve
    bs2_curve left_bs2,      // defining curve
    bs2_curve right_bs2,     // defining curve
    const curve& def_crv,    // defining curve
    SPAinterval v_range,     // v range
    double left_off,         // left off
    double right_off,        // right off
    closed_forms u_closure   // u closure
        = OPEN,
    closed_forms v_closure   // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates an edge-vertex blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const curve& left_crv,   // left curve
    const curve& right_crv,  // right curve
    const curve& def_crv,    // defining curve
    SPAinterval v_range,     // v parameter range
    double left_off,         // left offset
    double right_off,        // right offset
    closed_forms u_closure   // u closure
        = OPEN,
    closed_forms v_closure   // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates an edge-edge blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const curve& left_crv,      // left curve
    const SPAposition& right_pt,// right point
    const curve& def_crv,       // defining curve
    SPAinterval v_range,        // v range
    double left_off,            // left off
    double right_off,           // right off
    closed_forms u_closure      // u closure
        = OPEN,
    closed_forms v_closure      // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates an edge-vertex blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const curve& left_crv,      // left curve
    const surface& right_srf,   // right surface
    bs2_curve left_bs2,         // defining curve
    bs2_curve right_bs2,        // defining curve
    const curve& def_crv,       // defining curve
    SPAinterval v_range,        // v range
    double left_off,            // left off
    double right_off,           // right off
    closed_forms u_closure      // u closure
        = OPEN,
    closed_forms v_closure      // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates an edge-vertex blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const curve& left_crv,      // left curve
    const surface& right_srf,   // right surface
    bs2_curve right_bs2,        // right bs2
    const curve& def_crv,       // defining curve
    SPAinterval v_range,        // v range
    double left_off,            // left off
    double right_off,           // right off
    closed_forms u_closure      // u closure
        = OPEN,
    closed_forms v_closure      // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates an edge-face blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const curve& zero_crv,  // zero curve
    const curve& def_crv,   // defining curve
    SPAinterval v_range,    // v range
    double offset,          // offset
    closed_forms v_closure  // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates a vertex-vertex blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const SPAposition& left_pt,// left point
    const curve& right_crv, // right curve
    const curve& def_crv,   // defining curve
    SPAinterval v_range,    // v range
    double left_off,        // left off
    double right_off,       // right off
    closed_forms u_closure  // u closure
        = OPEN,
    closed_forms v_closure  // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates a vertex-edge blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const SPAposition& left_pt, // left point
    const surface& right_srf,   // right surface
    bs2_curve right_bs2,        // right bs2
    const curve& def_crv,       // defining curve
    SPAinterval v_range,        // v range
    double left_off,            // left off
    double right_off,           // right off
    closed_forms u_closure      // u closure
        = OPEN,
    closed_forms v_closure      // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates a vertex-face blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const surface& left_srf,// left surface
    const curve& right_crv, // right curve
    bs2_curve left_bs2,     // defining curve
    bs2_curve right_bs2,    // defining curve
    const curve& def_crv,   // defining curve
    SPAinterval v_range,    // v range
    double left_off,        // left off
    double right_off,       // right off
    closed_forms u_closure  // u closure
        = OPEN,
    closed_forms v_closure  // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates an edge-vertex blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const surface& left_srf,// left surface
    const curve& right_crv, // right curve
    bs2_curve left_bs2,     // left bs2
    const curve& def_crv,   // defining curve
    SPAinterval v_range,    // v range
    double left_off,        // left off
    double right_off,       // right off
    closed_forms u_closure  // u closure
        = OPEN,
    closed_forms v_closure  // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates a face-edge blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const surface& left_srf,    // left surface
    const SPAposition& right_pt,// right point
    bs2_curve left_bs2,         // left bs2
    const curve& def_crv,       // defining curve
    SPAinterval v_range,        // v range
    double left_off,            // left off
    double right_off,           // right off
    closed_forms u_closure      // u closure
        = OPEN,
    closed_forms v_closure      // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates a face-vertex blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

```
public: rb_blend_spl_sur::rb_blend_spl_sur (
    const surface& left_srf,     // left surface
    const surface& right_srf,    // right surface
    bs2_curve left_bs2,          // left bs2
    bs2_curve right_bs2,         // right bs2
    const curve& def_crv,        // defining curve
    SPAinterval v_range,         // v range
    double left_off,             // left off
    double right_off,            // right off
    closed_forms u_closure       // u closure
        = OPEN,
    closed_forms v_closure       // v closure
        = CLOSURE_UNSET
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Creates a face-face blend. The constructor doesn't copy anything, but assumes ownership of the data that is passed to it by pointer—namely the blend_supports, radius functions and cross sections.

Destructor:

```
public: virtual
    rb_blend_spl_sur::~rb_blend_spl_sur ();
```

C++ destructor, deleting a rb_blend_spl_sur.

Methods:

```
public: virtual int
    rb_blend_spl_sur::accurate_derivs (
    SPApar_box const&        // parameter box
        = * (SPApar_box* ) NULL_REF
    ) const;
```

Return the number of derivatives which evaluate can find precisely (and fairly directly), rather than by finite differencing, over the given portion of the surface. If there is no limit to the number of accurate derivatives, returns the value ALL_SURFACE_DERIVATIVES, which is large enough to be more than anyone could reasonably want.

```
public: double rb_blend_spl_sur::blend_angle (
    SPAunit_vector& Tan,     // tangent vector
    SPAvector const& R0,     // 1st radius vector
    SPAvector const& R1,     // 2nd radius vector
    double& rr_sina          // radius rule sin angle
        = * (double* ) NULL_REF,
    double& rr_cosa          // radius rule cos angle
        = * (double* ) NULL_REF
    ) const;
```

Find the angle between two radius vectors at def_cvec, according to the rule:

1. 0 <= ang < two pi – SPAresnor
2. nb cvec provides tangent to complete coordinate system

```
public: double rb_blend_spl_sur::blend_total_angle (
    SPAposition& P,          // pipe position
    SPAunit_vector& Tan,     // tangent vector
    SPAvector const& R0,     // 1st radius vector
    SPAvector const& R1,     // 2nd radius vector
    double& rr_sina          // radius rule sin angle
        = * (double* ) NULL_REF,
    double& rr_cosa          // radius rule cos angle
        = * (double* ) NULL_REF
        ) const;
```

Find the angle between perpendiculars to supports at def_cvec, according to the rules:

1. pipes return two pi
2. singularities return 0
3. 0 <= ang <= two pi
4. nb cvec provides tangent to complete coordinate system

```
public: virtual SPAbox rb_blend_spl_sur::bound (
    SPApar_box const&        // bounding box
    );
```

Bounding box. Normally the default is OK. Occasionally we make these surfaces without bs3_surfaces just to support offset SSIs, so we can do something more sensible here.

```
public: virtual void
    rb_blend_spl_sur::calculate_disc_info ();
```

Calculates the discontinuity information for the rb_blend_spl_sur.

```
public: virtual void
    rb_blend_spl_sur::compute_section (
    double v,                // v parameter
    int spine_nder,          // # required spine
                             // derivatives
    int spring_nder,         // number of required
                             // spring derivs
    logical xcrv_norm,       // whether to fill in
                             // xcurve normal
    blend_section& section,  // all output in here
    int                      // evaluation location:
        = 0                  // 1 => above,
                             // –1 => below,
                             // 0 => don’t care
    ) const;
```

A form of evaluation specific to blend_spl_surs (certain numerical algorithms used by blending need this function). Evaluates spine, defining curve, contact points and their derivatives at the given *v*-parameter, according to the blend_section class declaration as above. We may specify exactly how may spine and spring curve derivatives we require. As the two are typically connected you may get more than you asked for, but you are guaranteed to get at least what you ask for. Implementations of this should also ensure it does no more than is necessary. Finally the logical flag indicates whether you require the cross curve normal filled in; again this may (will) have implications on the amount of other stuff you get back, but if passed as TRUE then this is guaranteed to be returned. Note that calling this with for example –1, –1 and TRUE is valid

```
public: virtual subtrans_object*
    rb_blend_spl_sur::copy () const;
```

Construct a duplicate in free store of this object but with zero use count.

```
public: virtual void rb_blend_spl_sur::debug (
    char const*,            // loader line
    logical,                // brief output flag
    FILE*                   // file
    ) const;
```

Debug printout. The virtual function debug prints a class-specific identifying line, then calls the ordinary function debug_data to put out the details. It is done this way so that a derived class' debug_data can call its parent's version first, to put out the common data. Indeed, if the derived class has no additional data it need not define its own version of debug_data and use its parent's instead. A string argument provides the introduction to each displayed line after the first, and a logical sets brief output (normally removing detailed subsidiary curve and surface definitions).

```
public: virtual spl_sur*
    rb_blend_spl_sur::deep_copy (
    pointer_map*  pm        // list of items within
        = NULL              // the entity that are
                            // already  deep copied
    ) const;
```

Creates a copy of an item that does not share any data with the original. Allocates new storage for all member data and any pointers. Returns a pointer to the copied item.

```
public: virtual int rb_blend_spl_sur::evaluate (
    SPApar_pos const&,              // given param
                                    // value
    SPAposition&,                   // evaluated
                                    // position
    SPAvector**                     // derivative
        = NULL,                     // vectors
    int                             // # derivatives
        = 0,                        // requested
    evaluate_surface_quadrant       // eval. surface
        = evaluate_surface_unknown  // quadrant
    ) const;
```

The evaluate function calculates derivatives, of any order up to the number requested, and stores them in vectors provided by the user. It returns the number it was able to calculate; this will be equal to the number requested in all but the most exceptional circumstances. A certain number will be evaluated directly and (more or less) accurately; higher derivatives will be automatically calculated by finite differencing; the accuracy of these decreases with the order of the derivative, as the cost increases.

```
public: virtual void
    rb_blend_spl_sur::eval_prin_curv (
    SPApar_pos const& uv,    // uv parameter position
    SPAunit_vector& u1,      // first axis direction
    double& c1,              // curvature in first
                             // direction
    SPAunit_vector& u2,      // second axis direction
    double& c2               // curvature in second
                             // direction
    ) const;
```

Find the principal axes of curvature of the surface at a point with given parameter values, and the curvatures in those directions.

```
public: static int rb_blend_spl_sur::id ();
```

Returns the ID for the rb_blend_spl_sur list.

```
public: virtual SPApar_pos rb_blend_spl_sur::param (
    SPAposition const&,      // position
    SPApar_pos const&        // parameter position
        = * (SPApar_pos* ) NULL_REF
    ) const;
```

Returns the parameter.

```
public: virtual void rb_blend_spl_sur::point_perp (
    SPAposition const& point,   // point
    SPAposition& foot,          // foot
    SPAunit_vector& norm,       // normal
    surf_princurv& curv,        // curve
    SPApar_pos const& uv_guess  // uv guess
        = * (SPApar_pos* ) NULL_REF,
    SPApar_pos& uv_actual       // uv actual
        = * (SPApar_pos* ) NULL_REF,
    logical f_weak              // weak flag
        = FALSE
    ) const;
```

Find the point on the surface nearest to the given point, iterating from the given parameter values (if supplied). Return the found point, the normal to the surface at that point and the parameter values at the found point.

```
public: double rb_blend_spl_sur::radius () const;
```

Returns the radius.

```
public:logical rb_blend_spl_sur::relax (
    SPAposition const& point,   // position
    SVEC& sv                    // svec
    );
```

A version of point_perp to support SVEC::relax – it doesn't pull the relaxed parameters back to range.

```
public: virtual int rb_blend_spl_sur::type () const;
```

Returns the type of rb_blend_spl_sur.

```
public: virtual char const*
    rb_blend_spl_sur::type_name () const;
```

Returns the string "rbblnsur".

Internal Use: full_size

Related Fncs:

restore_rb_blend_spl_sur

restore_def

Class: SAT Save and Restore

Purpose: Records an entity type name, a pointer to a restore_data routine for that type of entity, and a link pointer.

Derivation: restore_def : ACIS_OBJECT : –

SAT Identifier: None

Filename: kern/kernel/kerndata/savres/savres.hxx

Description: This class records an entity type name, a pointer to a restore_data routine for that type of entity, and a link pointer. A constructor for the class links instances of the class into a chain at initialization.

Limitations: None

References: None

Data:

None

Constructor:

```
public: restore_def::restore_def (
    char const*,                    // ext. entity name
    int&,                           // entity type code
    ptr_to_restore_routine_type,// sub-class head_ptr
    restore_def* const*             // ptr to associated
        = NULL                      // entities
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

```
public: restore_def::restore_def (
    restore_def*&,                  // owning class
                                    // head_ptr
    char const*,                    // ext. entity name
    int&,                           // entity type code
    ptr_to_restore_routine_type,// sub-class head_ptr
    restore_def* const*             // ptr to associated
        = NULL                      // entities
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

Constructs the sub-class of a main class or sub-class of restore_def, with some possible subclasses.

Destructor:

```
public: restore_def::~restore_def();
```

C++ destructor for restore_def which deallocates memory.

Methods:

```
public: int restore_def::get_ent_code () const;
```

Returns the integer identifier of objects of this class.

```
public: char const*
    restore_def::get_ent_name () const;
```

Returns the string describing objects of this class.

```
public: ptr_to_restore_routine_type
    restore_def::get_restore_routine () const;
```

Returns a pointer to the routine to restore objects of this class.

```
public: restore_def*
    restore_def::get_sub_classes () const;
```

Returns a pointer to a list of restore_defs for all classes immediately derived from this one.

```
public: restore_def* restore_def::next () const;
```

Returns a pointer to the next restore_def in the list.

```
public: static logical
    restore_def::remove_from_list (
    restore_def** sub_classes_list, // list of
                                    // classes
    restore_def* object_to_remove   // object to
                                    // remove
    );
```

Removes an object from the restore list.

Related Fncs:

copy_body_from_body, copy_entity_from_entity, find_entity_code, get_save_file_version, get_savres_file_interface, read_array, restore_body_from_file, restore_entity_from_file, restore_entity_list_from_file, restore_some_entities, save_body_on_file, save_entity_list_on_file, save_entity_on_file, set_save_file_version

rot_spl_sur

Class: Construction Geometry, SAT Save and Restore

Purpose: Represents a surface of rotation.

Derivation: rot_spl_sur : spl_sur : subtrans_object : subtype_object : ACIS_OBJECT : –

SAT Identifier: “rotsur”

Filename: kern/kernel/kerngeom/splsur/rot_spl.hxx

Description: This class represents a surface of rotation. The surface is defined by an axis of rotation and a curve. The curve must not intersect with the axis, except possibly at its ends, and must not be tangential to a circle centered on the axis and perpendicular to it (i.e., at no point on the curve can the tangent direction be the same as that of a circle that is centered on the axis of revolution, perpendicular to it, and through the point). The parameter ranges defining the surface are the *u*-direction along the curve following its parameterization, and the *v*-direction clockwise around the axis, with the given curve as the v=0 parameter line.

Limitations: None

References:

KERN	curve
BASE	SPAposition, SPAunit_vector

Data:

None

Constructor:

```
public: rot_spl_sur::rot_spl_sur(
    const rot_spl_sur&      // rot_spl_sur
    );
```

C++ copy constructor requests memory for this object and populates it with the data from the object supplied as an argument.

```
public: rot_spl_sur::rot_spl_sur (
    curve const&,             // given curve
    SPAposition const&,       // axis point
    SPAunit_vector const&,    // axis direction
    SPAinterval const&        // u-parameter range
        = * (SPAinterval* ) NULL_REF,
    SPAinterval const&        // v-parameter range
        = * (SPAinterval* ) NULL_REF
    );
```

C++ initialize constructor requests memory for this object and populates it with the data supplied as arguments.

The *u*-parameter range defaults to the full given curve, or it returns an error if the curve is unbounded. The *v*-parameter range defaults to a full circle, 0 to 2pi.

Destructor:

None

Methods:

```
public: virtual spl_sur* rot_spl_sur::deep_copy (
    pointer_map*  pm          // list of items within
        = NULL                // the entity that are
                              // already  deep copied
    ) const;
```

Creates a copy of an item that does not share any data with the original. Allocates new storage for all member data and any pointers. Returns a pointer to the copied item.

```
public: virtual curve*
    rot_spl_sur::get_path () const;
```

Returns the path used for the rotation.

```
public: virtual sweep_path_type
    rot_spl_sur::get_path_type () const;
```

Returns the path type used used for the rotation operation.

```
public: virtual curve*
    rot_spl_sur::get_profile (
    double                    // parameter
    ) const;
```

Returns sweep information for the curve.

```
public: virtual law* rot_spl_sur::get_rail () const;
```

Returns the rail used for profile orientation in the rotation operation.

```
public: static int rot_spl_sur::id ();
```

Returns the ID for the rot_spl_sur list.

```
protected: virtual void rot_spl_sur::make_approx (
    double fit,                  // fit tolerance
    const spline& spl            // pointer to output
        = * (spline*) NULL_REF,  // spline approx.
    logical force                // flag for forcing
        = FALSE
    ) const;
```

Makes or remakes an approximation of the rot_spl_sur, within the given tolerance.

```
private: void rot_spl_sur::restore_data ();
```

Restores the information for a rot_spl_sur from a save file.

```
restore_curve                              Save the underlying curve
read_position                              axis root
read_unit_vector                           axis direction
if ( restore_version_number < APPROX_SUMMARY_VERSION )
    read_interval                          u range
    read_interval                          v range
    if ( restore_version_number >= DISCONTINUITY_VERSION )
        discontinuity_info::restore        u discontinuity information.
        discontinuity_info::restore        v discontinuity information.
else
    spl_sur::restore_common_data           Save the common data.
```

```
public: virtual void rot_spl_sur::save_data () const;
```

Saves the information for a rot_spl_sur to a save file.

```
public: virtual int rot_spl_sur::type () const;
```

Returns the integer representing the type of this spl_sur.

```
public: virtual char const*
   rot_spl_sur::type_name () const;
```

Returns the type of “rotsur”.

Internal Use: full_size

Related Fncs:

restore_rot_spl_sur