BUTTERWORTH HEINEMANN

0010-4485(95)00003-8

Computer-Aided Design, Vol. 27, No. 11, pp. 821–832, 1995

0010-4485/95 $10.00 + 0.00

Relational geometric synthesis: Part 1—framework

John S Letcher Jr, D Michael Shook and Simon G Shepherd

A novel object-orientated framework is proposed for 3D computer-aided design. Geometric objects including points, lines, curves, surfaces and solids are defined and constructed in a system which captures and retains many dependency relationships between objects, in a data structure equivalent to a directed graph. The primary advantage is automatic updating of the model, preserving relationships, when an underlying object is changed. Multiple types of parametric curves and surfaces and multiple levels of dependency are supported through a recursive program structure. Difficulties of surface–surface intersection and trimming are largely avoided, by provision of novel entities and relationships which permit construction of accurate and durable joins between surface objects.

Keywords: surface modelling, relational geometry, computer-aided geometric design

Geometric definition is an essential element in the design of practically any object to be manufactured. In CAD, mathematical representations of an object's geometry are stored in computer memory and manipulated by the computer user. The end product of a CAD design can be scale drawings; NC instructions for automated production of the object; or visualization or pictorial rendering of the object may be the entire objective. The CAD representation may also serve as a basis for analysis and evaluation of the design aside from visual aspects, e.g. finite-element method (FEM) stress analysis or computational fluid dynamic (CFD) flow analysis.

We open by noting three areas of deficiency in conventional CAD:

1 failure to capture or retain relationships which are important to the designer;
2 limitations on the variety of curve and surface types which are commonly supported; and
3 difficulties associated with surface–surface intersections and trimming of surfaces along such intersections.

AeroHydro, Inc., PO Box 684, Southwest Harbor, ME 04679, USA
Paper received: 21 June 1994. Revised: 21 November 1994

We are introducing a formal framework — 'relational geometric synthesis', or RGS — which addresses these three issues, and furnishes substantial solutions for them, with few negative side effects that we are aware of. Indeed, in our experience many other advantages accrue from the logical consistency of RGS, which make it easy to learn, to use, and to remember.

In operation of a CAD program, each geometric object is created in sequence, either by execution of a user command, or as a result of reading data from a file. Almost every new object is positioned, orientated or shaped in some deliberate relationship to one or more already existing objects. For example, line B might be created in such a way that one of its endpoints is at one end of an existing line A. However, this relationship is not retained by the CAD system; so if in some later revision line A is displaced, then line B will no longer connect to line A. A conventional CAD representation of geometry therefore consists of a large number of essentially independent simple objects, whose relationships are incidental to the manner and order in which they were created, but are not known to the program.

If design always proceeded in a forward direction, loss of relationship information might be of little importance. One would start a project, add objects until the design was complete, and save the results. However, it is well known that engineering design is only rarely a linear forward process. It is far more commonly an iteration: design is carried forward to some stage, then analysed and evaluated; problems are identified; then the designer has to retreat to some earlier stage, change some elements, and work forward again. It is typical that many iterative cycles are required, depending on the skills of the designer, the difficulty of the design specifications, and whatever constraints and optimization objectives may be present. In each forward stage, the designer will have to repeat many operations he previously performed (updating), in order to restore relationships disrupted by the revision of earlier design elements.

Revision of an existing design to meet new requirements is a common situation where similar problems are encountered. A change that alters an early stage of the design process requires at least one forward pass through all the subsequent design stages to restore disrupted relationships. If the relationships and the

sequence of design stages to achieve them have been lost, the updating process can be very difficult, error-prone and time-consuming.

Some partial solutions to this problem are known. In most CAD programs, lines A and B can be created together as part of a 'polyline' entity; then their connectivity will be automatically maintained if any of their endpoints, including their common point, are moved. Himelstein[1] has disclosed the concept of a 'sticky' attribute which causes selected lines to remain connected to objects they are deliberately attached to. Draney[2] has disclosed a 'relative point' entity which is specified by its positional relationship (coordinate offsets) from a selected point. Oosterholt[3] has disclosed the concept of giving names to geometric solids, and positioning each solid in relationship to at most one other solid, in a tree structure of positional dependency. The graphics standard PHIGS[4] provides similarly for optional naming of geometric structures, and use of structure names for positioning other structures. Ota *et al.*[5] have disclosed a CAD system in which some curve objects have names, and are used by name in the construction of some surface objects. In 'constraint-based modelling'[6] some kinds of relationships between objects are represented by constraint equations which are automatically satisfied during updates of the model.

Sutherland's Sketchpad project[7] utilized stored references to point objects to establish durable connections between lines. Some elements of RGS may underlie AGPS (aerodynamics grid and panelizing system), a proprietary grid generation system at Boeing[8–10]. AGPS has at least some entities which reference other objects for their data, achieving at least one level of automatic updating. Reference to objects by name or address, and geometry languages, were also present in the APT tool programming system[11], and in Armit's Multipatch, Mulitobject, ICON and TAG projects[12,13]. Snyder[14] presents a procedural-language approach to CAD which supports rapid regeneration following changes in underlying objects. Each of these systems allows a script to be replayed in order to regenerate a complex geometry with parametric variations.

Further, there are some examples of storing a multilevel relational structure of interobject dependencies, permitting selective updating of affected objects. Armit[13] discusses this capability in a surface modeller, TAG. PADL-2[15] is a CSG solid modelling framework with this feature, and Gossard *et al.*[16] describe a dimension-driven polygonal solid modeller.

In RGS every geometric object has a name, and a rich system of entities (primitives or abstract object types) is provided which reference other objects *by name* for their instantiation. The pervasive use of references to objects by name or identity allows the capture of many kinds of qualitative and quantitative relationships between objects, in a logical data structure having the form of a *relation* or *directed graph*. Using this structure, a program can automatically enforce all captured relationships during updates of the model, correctly propagating changes through the precise set of objects that are actually affected. This capability dramatically transforms geometric design from a linear, disjoint process in which many design elements necessarily become frozen in non-optimal states, to a fluid, coherent process permitting changes in any part of the model at any time, thus greatly facilitating refinement and optimization of the entire design.

Many different curve and surface entities are valuable for different purposes in CAD. Circular arcs and surfaces of revolution are useful because they are natural elements of, and easily manufactured by, rotating machinery. B-spline curves and surfaces are useful because they can be shaped into such a variety of forms, and because of the many useful and intuitive relationships they bear to their control points. Interpolating splines are useful in other circumstances, when we need a free-form curve to pass through specified points. Standard airfoil curves are useful because foils made from them have predictable and desirable aerodynamic characteristics[17]. Developable surfaces are useful because of the ease with which they can be formed from flat sheet materials. Numerous other useful curve and surface entities could be added to this list.

On the other hand it is burdensome for a CAD program to support a large number of curve and surface types, because their behaviours and data requirements vary so widely. It is generally accepted that the programming cost of supporting N entities increases much more rapidly than in proportion to N. In particular, it has been suggested[18] that support of N surface types requires implementation of $N(N+1)/2$ surface–surface intersection routines. Even the handful of useful curves and surfaces mentioned in the preceding paragraph have not previously been supported by any single CAD system known to the authors.

One known partial solution to this problem is to support only a single surface type, which has sufficient degrees of flexibility to encompass a useful set of simpler surfaces as special cases. Nonuniform rational B-spline (NURBS) surfaces have often been proposed to fill this role[18], since they encompass the earlier Bézier and B-spline formulations, and by special choice of knots and weights they can exactly represent arcs of conic sections, especially circles. Disadvantages of this approach include the obscure relationship between the selection of order, knots, weights and control points to achieve a desired curve; the large quantity of data required to define even a simple surface such as a circular cylinder; non-uniformity of resulting parameterizations; and the continuing usefulness of some curves and surfaces which lie outside the compass of NURBS.

A second main objective of relational geometric synthesis is therefore to provide a CAD surface modelling environment in which a wide variety of curve and surface types are supported for utility, convenience and flexibility. This is accomplished by a combination of polymorphism and recursion.

In CAD surface and solid modelling systems, intersections between surfaces often account for much of the complexity in both the program and the user interface, and are a major source of unreliability. In a typical application, surface Y is constructed, then surface Z is constructed in such a way that it deliberately intersects surface Y. The next step is usually to find the curve of intersection of Y and Z, so portions of Y and/or Z which extend beyond the intersection may be discarded (trimmed).

The problem of intersection of two surfaces is inherently difficult, for several reasons. Finding any single

point of intersection requires solution of three simultaneous, usually non-linear, equations. These equations will be ill-conditioned if the intersection is at a low angle. The intersection may consist of isolated points, simple arcs, closed curves, self-intersecting curves, or multiple combinations of these elements. The surfaces might indeed coincide over some finite area. Once a curve of intersection is found, it is often difficult to indicate correctly which portions of which surfaces are to be discarded. After trimming, a parametric surface patch may no longer be topologically quadrilateral, so it can no longer be conveniently parameterized.

A third main objective of RGS is therefore to provide a CAD surface modelling environment in which the difficulties of intersecting and trimming of surfaces are largely avoided, by providing convenient ways to construct surfaces which join one another accurately in the first place, with joins that are automatically maintained during updates of the model. This is accomplished by providing transfinite surface entities which interpolate arbitrary edge curves; by use of object names for points and curves so two surfaces can reference the same curve or set of points as edge data; and by providing a class of curve entities embedded in surfaces ('snakes').

It is noted that the novel features of RGS are the subject of patent applications in several countries. The PC-based surface modeller MultiSurf (R) from Aero-Hydro, Inc. is an initial commercial implementation of RGS. MultiSurf currently supports 20 point entities, 18 curves, 13 snakes, and 22 surface entities, in a highly interactive and visual 3D environment.

RELATIONAL GEOMETRY

For purposes of this paper, a 3D design space is contemplated, utilizing Cartesian coordinates (x, y, z) for the location of points. Our focus is primarily on surface rather than solid modelling. Of course, the modelling of surfaces, and particularly of accurately conjoined multiple surface patches, relates strongly to B-rep solid modelling, so we believe RGS has a large potential in that area. At this time we see little relation of RGS to CSG solid modelling. RGS does, however, suggest an alternative approach to solid modelling which will be developed in Part 2.

Definition: An *entity* is an abstract type of geometric object supported by the system, and requiring a specific set of data for its actualization.

Common CAD entities are the point, the line, the arc, the Bézier patch. An entity can be thought of as a blank form, with spaces to fill in data such as name, colour, names of supporting objects, etc.

Definition: An *object* is an instance of an entity; for example an absolute point, colour yellow, located at $(1, 2, -3)$. An object can be thought of as a completely filled-in entity form.

Comment: In CAD usage, in general, we find the words 'object' and 'entity' applied fairly interchangeably to *both* of the two concepts that we are distinguishing. Dictionaries likewise make little distinction between the words. We early encountered a need to have distinct names for the two separate concepts, and our original choice (as stated above) stuck. We are aware of precedent for use of the word 'primitive' for what we are calling an 'entity'.

The use of an object name as data for another object establishes a dependency relationship between the two objects. In this relationship, the object whose name is used is called the *supporting object* or *support*; the object which uses the name as data is called the *dependent object* or *dependent*. The dependency relationship is directional. The complete set of dependency relationships between the objects in a model has the logical form of a *relation* or *directed graph* (digraph)[19]. Each node of the dependency digraph represents an object; each directed edge indicates the dependency of one object on another (see *Figure 2* below).

To date, implementation of RGS are restricted to acyclic directed graphs. Bidirectional dependencies (for example, mutual tangency between two surface objects) would open some interesting possibilities, and complications, which we have not explored. Most forms of cyclic dependency we have contemplated result in a singular model; i.e. either no solutions or an infinite number of solutions.

The dependency relationship can take many forms. For example, a relative point depends on one other point object for its location. A B-spline curve depends on each of the point objects which are its control points. A lofted surface depends on each of the curve objects through which it is lofted, and it also depends, in turn, on each point object used in the definition of those curves. Dependency can extend to many levels. An object can depend on many other objects, and can in turn support many other objects.

Some objects may be defined in an absolute sense, having no dependency on any others. For example, an absolute point is specified solely by its coordinates x, y, z.

Use is made of curve and surface parameters as part of the data for some entities. For example, a point constrained to lie on a curve (a *bead*) can be located by naming the curve and giving a specific value for t. A point constrained to lie on a surface (a *magnet*) can be located by naming the surface and a specific pair of parameter values for u, v. A snake is a parametric curve in the two-dimensional u, v parameter space of its supporting surface, mapped onto the supporting surface.

Definition: A *logical model* is any valid collection of objects, i.e. a set of valid objects in which all required dependencies are satisfied, without cyclic dependencies.

Definition: An *absolute model* is a geometric representation computed from a logical model, in which all objects are located by their absolute coordinates.

The calculation of absolute models from logical models is the primary computational task in RGS.

Those qualitative and quantitative properties of a model which are captured and enforced by utilizing the data structure of dependencies are referred to as *durable* properties.

CLASSIFICATION OF RGS ENTITIES

It is useful to classify and define entities first in terms of their *dimensionality* and *embedding*, and second in terms of their primary dependencies:
Points are zero-dimensional objects.

An absolute point depends on nothing.
A relative point depends on another point.
A bead is a point embedded in a curve.
A magnet is a point embedded in a surface.
A ring is a point embedded in a snake.

Curves are one-dimensional objects.

A line depends on two points.
An arc depends on three points.
A B-spline curve depends on two or more points.
A C-spline curve interpolates two or more points.

Snakes are one-dimensional objects, parametric curves embedded in a parametric surface. Any snake depends on its surface; in addition:

A line snake depends on two magnets or rings.
An arc snake depends on three magnets or rings.
A B-spline snake depends on two or more magnets or rings.
A C-spline snake interpolates two or more magnets or rings.

Surfaces are two-dimensional objects.

A translation surface depends on two curves.
A ruled surface depends on two curves.
A revolution surface depends on one curve and one line.
A C-lofted surface interpolates two or more curves.
A blended surface depends on four curves.
A B-spline surface depends on an array of points.

The above list of entities is intended to be illustrative, but by no means complete. Additional entities can easily be suggested for any of these classes.

A different useful classification may be made in terms of the *support role* each entity class can fulfil:

When a point is required, any point object will serve.
When a bead is required, only a bead will serve.
When a magnet is required, a magnet or ring will serve.
When a ring is required, only a ring will serve.
When a line is required, only a line will serve.
When a curve is required, any point, line, curve or snake will serve.
When a snake is required, any snake, magnet or ring will serve.
When a surface is required, only a surface will serve.

These rules are based on two precepts:

1 A supporting object must have the requisite properties to durably fulfil its role. Thus, a point which currently happens to lie on a surface does not qualify as a magnet, because in some future revision it may no longer be on that surface. A curve or surface cannot serve as a point, because it generally will consist of multiple points.
2 Degenerate objects are permitted; e.g. a point used as a curve simply returns the same coordinates for any value of parameter, t.

A SIMPLE EXAMPLE

One useful form of representation of a logical model is a text file having one record for each object. This file maps one-to-one into the program's internal data storage. The file format is introduced here, and elaborated in a later section. An object record includes the entity type, the object name, various object attributes such as colour and visibility, and any other variable data required to actualize the object, presented in a predefined order peculiar to the entity. For example, the prototypes or forms for the absolute point entity and the line entity are, respectively:

AbsPoint *name colour vis x y z*
Line *name colour vis ndt relabel point1 point2*

where

colour = a colour index
vis = visibility index (0 means invisible)
ndt = number of divisions in t for wireframe displa
relabel = provision for reparameterization
('*' means default parameterization)

The following set of five records is an RGS solution to the 'line A–line B' problem discussed above (*Figure 1*):

```
AbsPoint  A1     14  1  1.  1.  3.;
AbsPoint  A2     14  1  2.  1.  3.;
Line      line_A 13  1  1   *   A1  A2;
AbsPoint  B2     14  1  2.  3.  3.;
Line      line_B 13  1  1   *   A2  B2;
```

This model contains five objects: three absolute points (named 'A1', 'A2', 'B2') and two lines (named 'line_A', 'line_B'). This data clearly records the intention that 'line_B' start where 'line_A' ends, viz. at point 'A2'. The dependency of both Lines on point 'A2' creates the durable connection.

The following record adds one more object to this example: AbsBead bead_B 12 1 line_B 0.7. This creates a visible point, of colour 12, constrained to remain

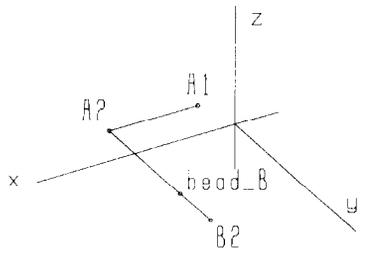


Figure 1 A simple relational model consisting of four points and two lines

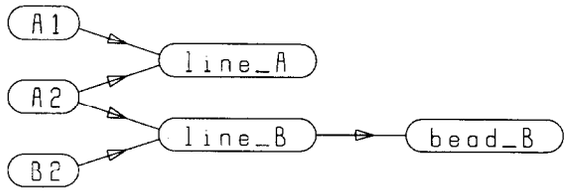


Figure 2 The directed graph of dependencies for the model of *Figure 1*

on 'line–B', at a parameter value of 0.7, i.e. 70% of the way from point 'A2' to point 'B2'. Following any change in 'line–A', 'bead–B' will still lie on 'line–B', in the same relative location. The dependency of 'bead–B' on 'line–B' creates the durable relationship. The digraph for this model is shown in *Figure 2*.

For purposes of output or display, an absolute model will be computed from the logical model. For this example, the absolute model would consist of:

a point, colour 14 at (1.,1.,1.)

a point, colour 14, at (2.,1.,1.)

a line, colour 13, from (1.,1.,1.) to (2.,1.,1.)

a point, colour 14, at (2.,3.,1.)

a line, colour 13, from (2.,1.,1.) to (2.,3.,1.)

a point, colour 12, at (2.,2.4,1.)

Now suppose that the example model is changed by moving point 'A2' to a new position (2.,1.,2.). This is accomplished by changing one element in one record of the logical model:

```
AbsPoint  A2  14  1  2.  1.  2.;
```

Following this change, the updated absolute model (*Figure 3*) would consist of:

a point, colour 14 at (1.,1.,1.)

a point, colour 14, at (2.,1.,2.)

a line, colour 13, from (1.,1.,1.) to (2.,1.,2.)

a point, colour 14, at (2.,3.,1.)

a line, colour 13, from (2.,1.,2.) to (2.,3.,1.)

a point, colour 12, at (2.,2.4,1.3)

The connection of 'line–A' and 'line–B' has been automatically maintained; 'bead–B' is still located on 'line–B', and in the same proportional position, i.e. at 70% of the length of 'line–B'. This brief example illustrates the automatic updating of the model that is made possible by utilization of the data structure of dependency relations.

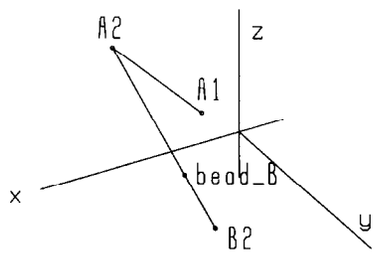


Figure 3 The model of *Figure 1*, following a change in position of point 'A2'

LOGICAL MODEL FILE

The concept of the logical model file was introduced in the preceding section; further details are presented here.

Each object description begins with a keyword specifying the entity, followed by the object name. Most objects have a colour attribute, currently selecting one colour from a palette of 16. Most objects have a visibility attribute; this is a byte in which the bits have different significance for different classes of entities, as follows:

points:
  bit 1: point is visible
curves, snakes:
  bit 1: polyline is visible
  bit 2: tick-marks displayed at uniform parameter intervals
  bit 3: polygon connecting control points or magnets is visible
surfaces:
  bit 1: parameter lines in u-direction are visible
  bit 2: parameter lines in v-direction are visible
  bit 3: (reserved)
  bit 4: boundary is visible
  bit 5: net connecting control points is visible

The following is a representative set of entity descriptions, sufficient for the example developed at the end of the paper. Many other entities will be detailed or proposed in Part 2 of this paper.

Point class

Absolute point
AbsPoint *name colour vis x y z.*

x, y, z are the absolute coordinates of the point.

Relative point
RelPoint *name colour vis point dx dy dz.*

dx, dy, dz are the coordinate offsets from *point*.

Absolute bead
AbsBead *name colour vis curve t.*

t is an absolute parameter value on *curve*.

Relative bead
RelBead *name colour vis bead dt.*

dt is the parameter offset from *bead*.

Absolute magnet
AbsMagnet *name colour vis surface u v.*

u, v are the absolute parameters on *surface*.

Relative magnet
RelMagnet *name colour vis magnet du dv.*

du, *dv* are the parameter offsets from *magnet*.

Absolute ring
AbsRing *name colour vis snake t.*

t is an absolute parameter value on *snake*.

Relative ring
RelRing *name colour vis ring dt.*

dt is a parameter offset from *ring*.

Curve class

All curves are parameterized from 0 to 1. All curve prototypes include *ndt*, specifying the number of divisions in *t* for wireframe display, and a 'relabel object' which permits reparameterization of the curve, as detailed in Part 2. Some curves additionally require specification of 'type' parameters.

Line
Line *name colour vis ndt relabel point1 point2.*

The line is a straight line from *point1* ($\mathbf{x}_1$) to *point2* ($\mathbf{x}_2$): $\mathbf{x}(t) = (1-t)\mathbf{x}_1 + t\mathbf{x}_2$.

Circular arc
Arc *name colour vis ndt relabel type point1 point2 point3.*

For all types, the curve is a circular arc lying in the plane of the three points. In some cases the arc can degenerate to a line (e.g. Type 1 with collinear points) or to a point (e.g. Type 2 with *point1* and *point2* coincident). In all cases parameterization is uniform with respect to arc length.

Type 1: The arc interpolates the three points in sequence. (Error if any two points coincide.)

Type 2: The arc starts at $\mathbf{x}_1$, has centre at $\mathbf{x}_2$, and ends on the line from $\mathbf{x}_2$ to $\mathbf{x}_3$. (Error if the points are collinear.)

Type 3: The arc is a full circle starting and ending at $\mathbf{x}_1$, with centre at $\mathbf{x}_2$; the direction is such that the closest point to $\mathbf{x}_3$ has $t < .5$. (Error if the points are collinear).

Type 4: The arc runs from $\mathbf{x}_1$ to $\mathbf{x}_3$, and at $\mathbf{x}_1$ is tangent to the line from $\mathbf{x}_1$ to $\mathbf{x}_2$. (Error if points 1 and 2 coincide).

Type 5: The arc runs from $\mathbf{x}_1$ to $\mathbf{x}_3$, and at $\mathbf{x}_3$ is tangent to the line from $\mathbf{x}_2$ to $\mathbf{x}_3$. (Error if points 2 and 3 coincide).

B-spline curve
BCurve *name colour vis ndt relabel type {point1 point2... pointN}.*

type specifies the spline order *k*: $k = type + 1$, i.e. 1 = linear, 2 = quadratic, etc.

The named points are the control points in sequence. (Braces are used to contain a variable-length list of supports.) A point is evaluated by using B-splines of the specified order as weights applied to the *N* control points[20,21]:

$$\mathbf{x}(t) = \sum_{i=1}^{N} \mathbf{x}_i B_i(t)$$

C-spline curve
CCurve *name colour vis ndt relabel type {point1 point2... pointN}.*

type specifies the spline order *k*: $k = type + 1$, i.e. 1 = linear, 2 = quadratic, etc.

The named control points are interpolated in sequence by the curve. The curve is a parametric spline with chord-length parameterization, knots at the data points (odd *type*) or midway between data points (even *type*), and not-a-knot end conditions[20]. An error occurs if two consecutive control points are coincident.

Sub-curve
SubCurve *name colour vis ndt relabel bead1 bead2.*

The subcurve is the portion of the basis curve $\mathbf{x}_c(.)$ between the two beads, reparameterized to the interval [0, 1]:

$$\mathbf{x}(t) = \mathbf{x}_c[(1-t)t_1 + t\,t_2]$$

Relative curve
RelCurve *name colour vis ndt relabel curve point1 point2.*

The relative curve is a copy of the basis curve $\mathbf{x}_c(.)$, linearly mapped to span the two endpoints $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\mathbf{x}(t) = \mathbf{x}_c(t) + (1-t)\,[\mathbf{x}_1 - \mathbf{x}_c(0)] + t[\mathbf{x}_2 - \mathbf{x}_c(1)]$$

Snake class

All snakes are parameterized from 0 to 1. A snake is evaluated by first locating a point $\mathbf{w} = \{u, v\}$ in the parameter space of the surface, then evaluating the surface with those parameter values. In general, the supporting surface for the snake is deduced from the supporting magnets or other surface-based objects supporting the snake. It is an error condition if these supporting objects do not all lie on the same surface. Like curves, all snake prototypes include *ndt*, specifying the number of divisions in *t* for wireframe display, and a 'relabel object' which permits reparameterization of the snake. Some snakes additionally require specification of 'type' or 'order' parameters. Snake prototypes are in fact completely parallel to the corresponding curve prototypes.

Line snake
LineSnake *name colour vis ndt relabel magnet1 magnet2.*

The LineSnake is a straight line in *u*, *v* parameter space from *magnet1* ($\mathbf{w}_1 = \{u_2, v_1\}$) to *magnet2* ($\mathbf{w}_2 = \{u_2, v_2\}$):

$$\mathbf{w}(t) = (1-t)\mathbf{w}_1 + t\mathbf{w}_2$$

Arc snake
ArcSnake *name colour vis ndt type relabel magnet1 magnet2 magnet3.*

Types 1 to 5 are the same as for Arc, above. The circular arc is drawn in the u, v parameter space and then mapped onto the surface.

B-spline snake
BSnake *name colour vis ndt relabel type {magnet1 magnet 2... magnetN}.*

type specifies the B-spline order: 1 = linear, 2 = quadratic, etc. The named magnets are the control points in sequence. A point is evaluated by using B-splines of the specified order as weights applied to the control magnets[20,21]:

$$\mathbf{w}(t) = \sum_{i=1}^{N} \mathbf{w}_i B_i(t)$$

Figure 4 illustrates a BSnake object, dependent on a surface object and multiple magnet objects.

C-spline snake
CSnake *name colour vis ndt relabel type {magnet1 magnet2... magnetN}.*

type specifies the spline order: 1 = linear, 2 = quadratic, etc. The named magnets are interpolated in sequence. The snake is a parametric spline of the specified type in the u, v parameter space with chord-length parameterization, knots at the data points (odd type) or midway between data points (even type), and not-a-knot end conditions[20]. An error occurs if consecutive magnets are coincident.

Sub-snake
SubSnake *name colour vis ndt relabel ring1 ring2.*

The subsnake is the portion of the basis snake $\mathbf{w}_s(.)$ between the two rings, reparameterized to the interval [0,1]:

$$\mathbf{w}(t) = \mathbf{w}_s[(1-t)\, t_1 + t\, t_2]$$

Surface class

All surfaces are parameterized from 0 to 1 in both u and v directions. All surface prototypes include four subdivision parameters which control the display of a wireframe mesh:

ndu = no. of divisions in u-direction
nsu = no. of subdivisions in u-direction
ndv = no. of divisions in v-direction
nsv = no. of subdivisions in v-direction

and one normal orientation parameter:

ior = 0 or 1

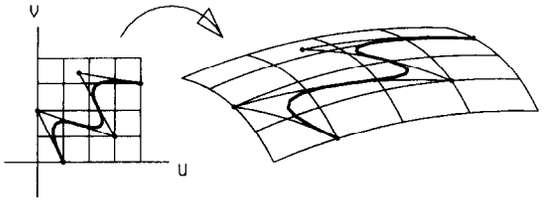


Figure 4 A B-spline snake supported by magnets embedded in a parametric surface object

(With *ior* = 0, the unit normal $\mathbf{n}$ is oriented so that the scalar triple product $|\partial\mathbf{x}/\partial u \;\; \partial\mathbf{x}/\partial v \;\; \mathbf{n}|$ is positive; *ior* = 1 reverses the normal orientation. This provision anticipates applications in which the inside/outside orientation of a surface is significant.)

There are several species of 'lofted surface' which form a subclass with important common properties. Each derives its name and behaviour from a type of curve. Each is supported by two or more curves $\mathbf{x}_i(t)$, $i = 1,\ldots,N$ called *master curves*. A surface point $\mathbf{x}(u,v)$ on a lofted surface is obtained in three stages: (1) from each 'master curve' i take the point $\mathbf{x}_i(u)$; (2) form the 'longitudinal' curve of specified type which uses the $\mathbf{x}_i(u)$ in sequence for its data; (3) evaluate the longitudinal curve at parameter v. Each ruled surface interpolates its first and last master curves, along the $v = 0$ and $v = 1$ edges, respectively.

Ruled surface
RuledSurf name *colour vis ndu nsu ndv nsv ior curve1 curve2.*

The surface is formed from the two curves $\mathbf{y}_1(t)$, $\mathbf{y}_2(t)$ by linear interpolation:

$$\mathbf{x}(u,v) = (1-v)\,\mathbf{y}_1(u) + v\mathbf{y}_2(u)$$

The ruled surface belongs to the subclass of lofted surfaces, and could as well be called a 'line-lofted surface'.

Arc-lofted surface
ALoftSurf *name colour vis ndu nsu ndv nsv ior type curve1 curve2 curve3.*

A surface point $\mathbf{x}(u,v)$ is obtained in three stages: (1) from each 'master curve' i take the point $\mathbf{x}_i(u)$; (2) form the circular arc curve of specified *type* which uses the $\mathbf{x}_i(u)$ as data points; (3) evaluate the arc curve at parameter v. An error will occur in stage (2) if the three points are invalid data for an arc curve.

B-lofted surface

BLoftSurf *name colour vis ndu nsu ndv nsv ior type {curve1 curve2... curveN}.*

A surface point $\mathbf{x}(u,v)$ is obtained in three stages: (1) from each master curve i take the point $\mathbf{x}_i(u)$l (2) form the B-spline curve of specified order which uses the $\mathbf{x}_1(u)$ in sequence as control points; (3) evaluate the B-spline curve at parameter v.

When the master curves are all B-spline curves of the same order, the same number of vertices, and uniform knots, the B-lofted surface is identical to the tensor-product B-spline surface[20,21] using the same array of control points. Otherwise, the B-lofted surface is a useful transfinite generalization of the B-spline surface.

C-lofted surface
CLoftSurf name *colour vis ndu nsu ndv nsv ior type {curve1 curve2... curveN}.*

A surface point $\mathbf{x}(u,v)$ is obtained in three stages: (1) from each 'master curve' i take the point $\mathbf{x}_i(u)$; (2) form the C-spline curve of specified type which inter-

polates the $\mathbf{x}_i(u)$ in sequence; (3) evaluate the C-spline at parameter v. An error condition will occur in stage (2) if $\mathbf{x}_i(u) = \mathbf{x}_{i+1}(u)$ for any i.

Developable surface

DevSurf name *colour vis ndu nsu ndv nsv ior curve1 curve2.*

A developable surface is formed spanning the two basis curves $\mathbf{y}_1(t_1)$, $\mathbf{y}_2(t_2)$ by finding ruling lines which join the two edges and satisfy the additional condition[22,23]:

$$f(t_1, t_2) = |\, d\mathbf{y}_1/dt_1 \;\; d\mathbf{y}_2/dt_2 \;\; \mathbf{y}_1(t_1) - \mathbf{y}_2(t_2)| = 0$$

The v parameter is the fractional distance along any ruling from edge 1 to edge 2.

Translation surface

TranSurf *name colour vis ndu nsu ndv nsv ior curve1 curve2.*

The surface is formed from the two curves $\mathbf{y}(t)$, $\mathbf{z}(t)$ by addition:

$$\mathbf{x}(u, v) = \mathbf{y}(u) + \mathbf{z}(v) - \mathbf{z}(0)$$

i.e. a copy of $\mathbf{z}(v)$ is translated along $\mathbf{y}(u)$.

Revolution surface

RevSurf *name colour vis ndu nsu ndv nsv ior curve axis angle1 angle2.*

axis has to be the name of a line object.

The surface point at u, v is constructed by taking a point $\mathbf{y}(u)$ from *curve*, then rotating it through an angle $\theta = (1 - v)\theta_1 + v\theta_2$ about the axis line.

Blended surface

BlendSurf *name colour vis ndu nsu ndv nsv ior curve1 curve2 curve3 curve4.*

The surface is a bilinear Coons patch[21] constructed from the four curves. The equation for locating a surface point is:

$$\begin{aligned}\mathbf{x}(u, v) = {} & (1 - v)\mathbf{x}_1(u) + v\mathbf{x}_3(1 - u) \\ & + (1 - u)\mathbf{x}_4(1 - v) + u\mathbf{x}_2(v) \\ & - (1 - u)(1 - v)[\mathbf{x}_1(0) + \mathbf{x}_4(1)]/2 \\ & - u(1 - v)[\mathbf{x}_1(1) + \mathbf{x}_2(0)]/2 \\ & - uv[\mathbf{x}_2(1) + \mathbf{x}_3(0)]/2 \\ & - (1 - u)v[\mathbf{x}_3(1) + \mathbf{x}_4(0)]/2\end{aligned}$$

The four basis curves nominally join end-to-end in a closed loop. If the curves do not all meet in this fashion, a surface patch is still formed, but it does not interpolate all its edges. Triangular patches can be formed by having one degenerate side, i.e. one of the four curves is just a point.

Sub-surface

SubSurf *name colour vis ndu nsu ndv nsv ior snake1 snake2 snake3 snake4.*

The sub-surface is a bilinear Coons blending of four snakes lying on a surface $\mathbf{x}_s(.)$ (*Figure 5*). The equation for locating a surface point is:

$$\mathbf{x}(u, v) = \mathbf{x}_s(\mathbf{w}), \text{ where}$$

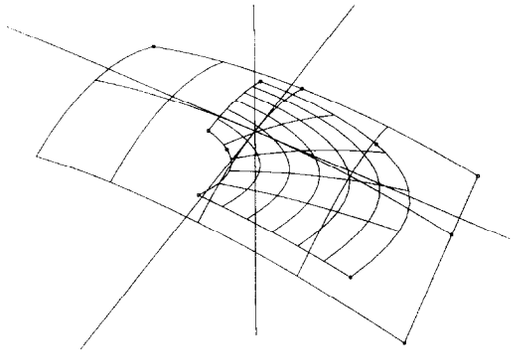


Figure 5 A sub-surface bounded by four snakes embedded in a surface object

$$\begin{aligned}\mathbf{w}(u, v) = {} & (1 - v)\mathbf{w}_1(u) + v\mathbf{w}_3(1 - u) \\ & + (1 - u)\mathbf{w}_4(1 - v) + u\mathbf{w}_2(v) \\ & - (1 - u)(1 - v)[\mathbf{w}_1(0) + \mathbf{w}_4(1)]/2 \\ & - u(1 - v)[\mathbf{w}_1(1) + \mathbf{w}_2(0)]/2 \\ & - uv[\mathbf{w}_2(1) + \mathbf{w}_3(0)]/2 \\ & - (1 - u)v[\mathbf{w}_3(1) + \mathbf{w}_4(0)]/2\end{aligned}$$

The four basis snakes nominally join end-to-end in a closed loop. If the curves do not all meet in this fashion, a surface patch is still formed, but it does not interpolate all its edges. Triangular subsurface patches can be formed by having one degenerate side, i.e. one of the four snakes is just a magnet.

In a logical model file, each object is represented by a single text record conforming to the format specified in the above entity definitions. White-space characters separate tokens, but are not otherwise significant. An object record is terminated by a semicolon. The text file is terminated by the keyword 'EndModel'. Remarks can be included in the text file by use of the keyword 'Rem'. Any object must be defined before it can be used as support for another object.

Internal to the program, objects may be referenced by serial numbers corresponding to their sequence in the input data file, or sequence of creation, or other unique index numbers, or by pointers to memory. Requiring that all references be to previously defined objects (lower serial number) is a simple way to exclude circular dependencies (digraph cycles). The organization of internal storage of the logical model may include a data structure representing the dependency digraph, to be used to control updating of the absolute model.

Because model files store only the logical model, requiring only about 30–60 characters per object, they are very compact. A model of 200 objects, which we currently view as moderately complex, would probably have a file size under 10 kB. In binary form, the same data would be about 50% smaller again.

The logical model file syntax can be viewed as a procedural geometric programming language. The entity keyword is a command to create an object of specified type (invocation of a function), and the variable data items that follow are function arguments. This view of the model file will be further explored in Part 2.

SUPPORT OF MULTIPLE CURVE AND SURFACE TYPES

The second principal objective of RGS is achieved by polymorphism and recursive program structure. The object orientation of RGS suggests an object-orientated implementation, but in fact all operational implementations to date have been via procedural languages.

To make these ideas more specific, we focus on the support of multiple curve types in a procedural implementation. First, in RGS all curves are members of a class which embodies many common properties, e.g. a common range of parameterization, 0 to 1. All curve objects are accessed through a single 'generic' function 'Curve', whose arguments include the identity (or memory location) of a curve object and a list of parameter values, and which returns a list of 3D points sampled from the specified curve at the specified parametric positions. From the point of view of a routine that needs to use points taken from a curve, all curves are equivalent; the routine calling 'Curve' does not need to know what kind of curve it is interrogating. 'Curve' must examine the data of the specified curve object, determine what kind of curve it is, and branch to the routine for evaluating this particular kind of curve from its particular data. LaBozzetta *et al.* have previously employed this polymorphic approach to supporting multiple surface types in I3G, a geometry system for CFD gridding[24].

Second, to allow for the possibility that objects used to support a curve are other curves, or depend on other curves, it is essential that the generic 'Curve' routine be programmed recursively, so it can call itself, or be called by routines which it has called. In similar fashion, generic recursive 'Point' and 'Surface' routines must be provided, as well as generic recursive routines for evaluating some other entity classes we have not yet introduced.

The three 'generic' class-level or primary procedures named 'Point', 'Curve' and 'Surface' are the interface to any application program requiring absolute geometric information from the model. These have input and output arguments as follows:

- Point
  in: a point object
  out: absolute coordinates x, y, z
- Curve
  in: a point, line, curve or snake object
  list of t parameter values
  out: list of absolute point coordinates x, y, z
- Surface
  in: a surface object
  list of u parameter values
  list of v parameter values
  out: array of absolute point coordinates x, y, z

The object passed to these procedures could be specified by name, index, serial number, or address pointer; or could be passed 'by value', i.e. a data structure which includes the entity type. Any input list of parameter values could have only a single entry, if only one point on the object is to be evaluated.

Three other class-level primary procedures return parameter values which are required in the operation of Point, Curve and Surface for some entities:

- Bead
  in: a bead or ring object
  out: supporting curve or snake;
  parameter value t
- Magnet
  in: a magnet or ring object
  out: supporting surface; parameter values u, v
- Snake
  in: a snake, magnet or ring object
  list of t parameter values
  out: supporting surface;
  array of parameter values u, v

Module 'Point' determines what kind of point object it is evaluating, and branches to the appropriate procedure for evaluating that type of point. Similarly, 'Curve', 'Snake' and 'Surface' are basically branches to their constituent secondary routines, one for each supported entity in the class.

'Line', 'LineSnake' and 'RuledSurface' routines share a common 'LineMath' routine; similarly, the other curves, snakes and lofted surfaces share common secondary math routines, able to operate with either 2D data (when called for a snake) or 3D data (when called for a curve). It is obvious now how easily a new parametric curve, snake, or surface entity can be added to the system; it requires only:

1 defining the prototype for the new entity;
2 addition of one secondary routine implementing the new entity; and
3 slight modification of one primary procedure, adding a branch to the new secondary routine.

Clearly, the effort of supporting N curves or surfaces is only $O(N)$.

Evaluation of most entities requires calls to other primary procedures; for example, a bead requires evaluation of a curve; a snake requires evaluation of a surface.

Support of 'relative' entities of all types evidently opens the possibility of recursion. For example, to locate a relative point, the program first needs to locate the basis point, no matter what kind of point object the basis point is. Thus, 'RelativePoint' must be able to call 'Point'; 'RelativeBead' must be able to call 'Bead', etc.

Other potential instances of recursion arise during evaluation of curves and surfaces. For example, 'Line', 'Arc', 'BCurve' and 'CCurve' all need to evaluate their supporting points, by a series of calls to 'Point'. Any such point object could be a bead on another curve. 'LineSnake', 'ArcSnake', 'BSnake' and 'CSnake' need to evaluate u, v parameters of each of their supporting magnet objects, by a series of calls to 'Magnet'. Such a supporting magnet could be a ring on another snake. The several surface routines need to evaluate various point, curve, snake or surface supports, according to their individual constitutions; these are all done through calls to the primary procedures.

Further cycles of recursion occur when, for example, one curve supporting the surface being evaluated is a snake on another surface. In this case the call sequence passes through 'Surface' twice. It is easy to think up

cases with arbitrarily long chains of dependency. All such recursive possibilities are accommodated by the program structure outlined above. Without recursion, the program complexity and size would grow extremely rapidly with the allowable depth of dependency; with recursion, only stack space is required to indefinitely extend the permitted depth of dependency.

The same polymorphism can be accomplished in an object-orientated language by subclassing. In C++, the abstract class Curve has a pure virtual member function Evaluate which receives a curve object and returns a tabulation of point coordinates. Each specific curve entity, implemented as a subclass of curve, has its own peculiar set of data members, and provides its own version of the Evaluate member function.

AVOIDANCE OF SURFACE–SURFACE INTERSECTION AND TRIMMING

The third principal objective of RGS is achieved by utilizing the dependency relationships outlined above, and providing certain generally useful snake and surface entities. Two distinct problems are addressed here: (1) coercing two surface objects Y and Z to accurately share a common edge, and (2) coercing one surface object Z to end accurately on another surface object Y, along a curve which is not necessarily an edge of Y.

Durable common edges between surfaces may be achieved by using common data to define the adjoining edges (*Figure 6*). For example, if the two surfaces are blended surfaces, whose data includes boundary curves, all that is required is to use the same curve object for the corresponding edges of the two surfaces. Two lofted surfaces of the same type will accurately join in the loft direction if their corresponding master curves have common endpoints along the edges where they adjoin. The subcurve entity allows construction of common edges where commonality does not extend along a complete edge of either or both surfaces.

Alternatively, a surface Z having an edge which accurately and durably lies on another surface Y can be achieved by defining a snake S on Y, then using S for an edge curve in the subsequent specification of Z (*Figure 7*). This arrangement also provides an alternative solution for common edges, when S is specified to be a linesnake lying along all or part of one edge of Y.

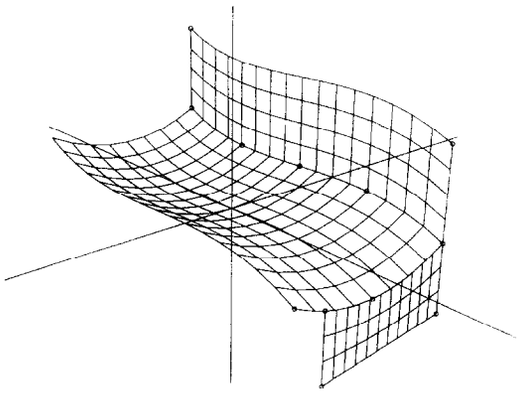


Figure 6 Surfaces accurately and durably joined by dependency on common edge curves

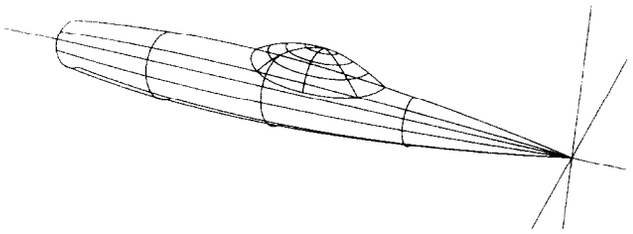


Figure 7 A surface accurately and durably joined to a snake embedded in another surface

It may be noted that two of these methods of forming conjoined surfaces are in the pattern of designing the curve of intersection first, then attaching the surfaces to it. This is the inverse of the usual intersection-trimming problem, and it should be no surprise that it is far simpler to execute.

DETAILED EXAMPLE

Table 1 is a text representation of an example logical model utilizing a variety of point, curve and snake objects, and six interconnected surface objects of various types, as defined and outlined above. *Figure 8* is a wireframe representation of the resulting absolute model. The example comprises hull, deck and cabin surfaces for a 30-foot sailing yacht design.

The example model has six surface objects: 'hull' and 'deck' are C-lofted surfaces; 'cabin_fwd', 'cabin_side', and 'cabin_aft' are ruled surfaces; and 'cabin_top' is a blended surface. The surfaces all have visibility 1, which causes only the parameter lines u = constant to be displayed. Eleven transverse sections through the model are also displayed for purposes of visualizing the shapes.

'Hull' is a C-lofted surface with three B-spline master curves 'MCA', 'MCB', 'MCC', each having four absolute points as vertices. 'Deck' also has three master curves; the first is the single point 'A1', the other two are three-vertex B-spline curves 'deck_beam' and 'transom'. The join between 'hull' and 'deck' is accurate and durable because the C-splines at the adjoining edges on each surface use the same data points, viz. 'A1', 'B1', 'C1', and therefore are identical curves.

The three ruled surfaces 'cabin_fwd', 'cabin_side', 'cabin_aft' are constructed in a similar fashion to one another; each uses a snake on 'deck' as one edge, providing an accurate and durable join to the 'deck' surface, and a relative curve dependent on that snake as the second (upper) edge. The three snakes on 'deck' join each other accurately and durably because they share common endpoint data, viz. magnets 'dm3' and 'dm5'. The three relative curves 'top_fwd', 'top_side', 'top_aft' also join each other accurately and durably because they are constructed using common end points, viz. relative points 'rp3' and 'rp5'. 'cabin_side' joins the other two surfaces accurately because its end rulings are the lines 'dm3' - 'rp3' and 'dm5' - 'rp5', which are identical to end rulings on the adjoining surfaces.

The blended surface 'cabin_top' joins the three ruled surfaces accurately because it uses their upper edge curves 'top_fwd', 'top_side', 'top_aft' as data. Its fourth side is a three-point C-spline 'top_ctr', which lies accurately in the centre plane because each of its vertices has a zero y coordinate.

Table 1 Text representation of logical model for the example application

```
Rem 3 × 4 cloft hull with deck and cabin for CAD example
AbsPoint  A1  14  1  0.00  0.00    3.60;
AbsPoint  A2  14  1  1.00  0.00    1.41;
AbsPoint  A3  14  1  2.50  0.00   −0.84;
AbsPoint  A4  14  1  3.00  0.00   −0.90;
BCurve  MCA  12  1  20  *  2  {A1  A2  A3  A4};
AbsPoint  B1  14  1  15.00 5.84    2.64;
AbsPoint  B2  14  1  15.00 6.00    0.54;
AbsPoint  B3  14  1  15.00 3.90   −1.20;
AbsPoint  B4  14  1  15.00 0.00   −1.44;
BCurve  MCB  12  1  20  *  2  {B1  B2  B3  B4};
AbsPoint  C1  14  1  30.00 3.50  2.76;
AbsPoint  C2  14  1  30.90 3.50  1.41;
AbsPoint  C3  14  1  31.70 2.50  0.22;
AbsPoint  C4  14  1  31.70 0.00  0.22;
BCurve  MCC  12  1  20  *  2  {C1  C2  C3  C4};
CLoftSurf hull  10  2  10  2  10  3  0  3  {MCA  MCB  MCC};
AbsPoint  transom0  14  1  29.80  0.00  3.00;
AbsPoint  transom1  14  1  29.80  1.75  3.00;
BCurve  transom  10  1  10  *  2  {C1  transom1  transom0};
AbsPoint  deck_ctr  14  1  15.00  0.00  3.45;
AbsPoint  deck_mid  14  1  15.00  2.70  3.45;
BCurve  deck_beam  10  1  10  *  2  {B1  deck_mid  deck_ctr};
CLoftSurf  deck7  2  4  2  10  3  0  3  {A1  deck_beam  transom};
AbsMagnet  dm1  11  1  deck  1.00  0.27;
RelMagnet  dm2  11  1  dm1  −0.37  0.00;
RelMagnet  dm3  11  1  dm1  −0.65  0.03;
RelMagnet  dm6  11  1  dm1  0.00  0.43;
RelMagnet  dm5  11  1  dm6  −0.80  0.00;
RelMagnet  dm4  11  1  dm5  0.00  −0.20;
BSnake  fp_fwd  11  1  10  *  2  {dm1  dm2  dm3};
BSnake  fp_side  11  1  20  *  2  {dm3  dm4  dm5};
LineSnake  fp_aft  11  1  10  *  dm5  dm6;
RelPoint  rp1  11  1  dm1  2.00  0.00  1.30;
RelPoint  rp3  11  1  dm3  2.00  0.00  1.10;
RelPoint  rp5  11  1  dm5  −0.20  −0.50  1.40;
RelPoint  rp6  11  1  dm6  −0.30  0.00  1.80;
Line lin1  6  1  1  *  rp1  rp6;
AbsBead  e1  10  1  lin1  0.500;
RelPoint  rp7  11  1  e1  0.00  0.00  0.20;
RelCurve  top_fwd  11  1  10  fp_fwd  rp1  rp3;
RelCurve  top_side  11  1  20  fp_side  rp3  rp5;
RelCurve  top_aft  11  1  10  fp_aft  rp5  rp6;
RuledSurf  cabin_fwd  11  2  10  1  1  1  f_fwd  top_fwd;
RuledSurf  cabin_side  11  2  20  1  1  1  fp_side  top_side;
RuledSurf  cabin_aft  11  2  10  1  1  1  fp_aft  top_aft;
CCurve  top_ctr  11  1  10  *  2  {rp1  rp7  rp6};
BlendSurf cabin_top  14  2  4  2  5  2  0  {top_fwd  top_side  top_aft  top_ctr};
XContours  stations  12  1  0  10  1.950  2.850;
EndModel
```

The example model as now defined can easily be transformed, with preservation of its topology, into an extremely wide variety of alternative shapes by changing the coordinates of absolute points, the offsets of relative points, and the parameters of magnets. An example modification which affects all six surfaces is to increase the *y* coordinate of 'B1'. Following this change, the connectivity and relative positioning of the several surfaces is automatically preserved as the absolute model is update (*Figure 9*). A further example modification is to drag AbsMagnet 'dm1' to a new position (*Figure 10*). Because the other 'dm?' magnets are relative to this one, the entire cabin is transported to a new location where it continues to join the deck precisely, following update. The update and redisplay for these example modifications takes approximately 3 s on a 486-50 PC.

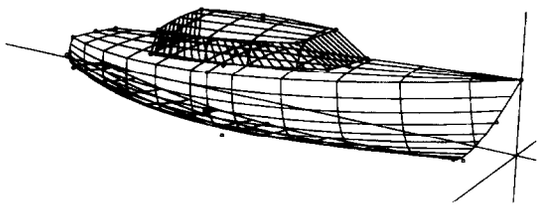


Figure 8 Example of model of *Table 1*, comprising six accurately conjoined surface objects

CONCLUSIONS

We have presented the outline for a new logical framework for CAD, offering several major advantages over conventional CAD systems.

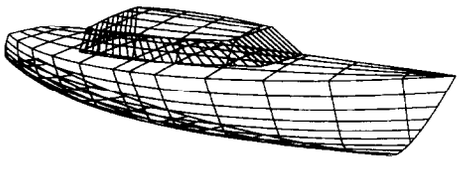


Figure 9 Example model following a change in the *Y*-coordinate of point 'B1'

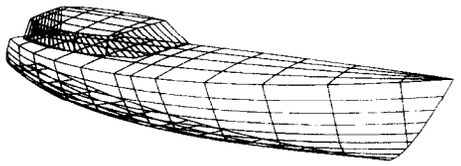


Figure 10 Example model following change in the v parameter of magnet 'dm1'

The most important advantage is the capture of qualitative and quantitative relationships between design elements, permitting rapid automatic update of a design following changes in an underlying object. This can greatly reduce design cycle time and labour, enabling a much more systematic and thorough refinement and optimization of the complete design.

RGS provides a framework in which many curve and surface types can comfortably coexist in a design system, with programming effort increasing only linearly with the number of entities supported.

To a considerable extent, RGS circumvents the complications of surface–surface intersection and trimming, by providing simple ways to build surfaces whose junctions are accurate by construction, and are automatically maintained through updates as qualitative relationships between the surface elements.

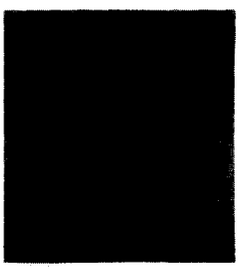
RGS also provides an extremely compact representation of its geometric models, including dependency relationships, in a text format which may be viewed as a geometry programming language.

RGS is readily extensible in many ways which invite exploration; some of these will be developed in Part 2.

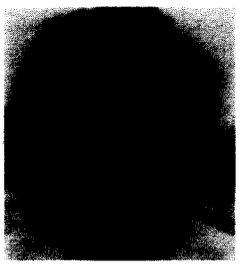
REFERENCES

1 Himelstein, C S US patent 4,633,616 (1986)
2 Draney, M R US patent 4,829,446 (1989)
3 Oosterholt, R US patent 4,868,766 (1989)
4 Howard, T L J, Hewitt, W T, Hubbold, R J and Wyrwas, K M *A Practical Introduction to PHIGS and PHIGS PLUS* Addison-Wesley, Reading (1991)
5 Ota, Y and Misato, A US patent 5,003,498 (1991)
6 MacKrell, J 'Making sense of a revolution', *Comput. Graphics World* Vol 16 No 11 (1993)
7 Sutherland, I E 'Sketchpad, a man–machine graphical communication system. *Dissertation* Massachusetts Inst. of Technology (1963)
8 Snepp, D K and Pomeroy R C *A Geometry System for Aerodynamic Design* (AIAA-87-2902) (1987)
9 Capron, W K and Smit, K L *Advanced Aerodynamic Applications of an Interactive Geometry and Visualization System* (AIAA-91-0800) (1991)
10 Gentry A E *Requirements for a Geometry Programming Language for CFD Applications* (NASA CP-3143) (1992)
11 Ross, D T 'Origins of the APT language for automatically programmed tools': History of Programming Languages Conference, *ACM SIGPLAN Notices*, Vol 13 No 8 (1978) pp 59–99
12 Armit, A P 'Computer systems for interactive design: The description of multipatch, multiobject and other existing systems' *in* Piegl, L (Ed.) *Fundamental Developments of Computer-Aided Geometric Modelling* Academic Press, Boston, MA (1993)
13 Armit, A 'TAG — A high performance interactive 3D graphics system' *Computers Indust.* Vol 3 (1982) pp 117–123
14 Snyder, J M *Generative Modelling for Computer Graphics and* CAD Academic Press, New York (1992)
15 Brown, C M 'PADL-2: A Technical Summary' *IEEE CG&A* Vol 2 No 2 (March 1982) pp 69–84
16 Gossard, D C, Zuffante, R P and Sakurai, H 'Representing dimensions, tolerances and features in MCAE systems' *IEEE CG&A* Vol 8 No 2 (March 1988) pp 69–84
17 Abbott, I H and von Doenhoff, A E *Theory of Wing Sections* Dover, New York (1959)
18 Tiller, W 'Rational B-splines for curve and surface representation' *IEEE Trans. Comput. Graph. Appl.* (1983) pp 61–69
19 Tenenbaum, A M, Langsaam, Y and Augenstein, M J *Data Structures Using C* Prentice-Hall, Englewood Cliffs, NJ (1990)
20 de Boor, C A *Practical Guide to Splines* Springer-Verlag, New York (1978)
21 Farin, G *Curves and Surfaces for Computer Aided Geometric Design* Academic Press, Boston, MA (1988)
22 Kreyszig, E *Differential Geometry* University of Toronto Press, Toronto (1959)
23 Nolan, T 'Computer-aided design of developable hull surfaces' *Marine Technol.* Vol 8 No 2 (1971) pp 233–242
24 LaBozzetta, W F, Cole, P E and Born, K E *Interactive Graphics for Geometry Generation — A Program with a Contemporary Design* (AIAA-84-2389) (1984)

John Letcher is president and founder of AeroHydro, Inc. in Southwest Harbor, Maine, USA. He received his BS (1963) in physics and PhD (1966) in aeronautics and applied maths from the California Institute of Technology. He also holds a MS (1984) in naval architecture from the University of Michigan. AeroHydro, Inc. is a leading developer of CAD *applications for marine design and shipbuilding, including geometric design, hydrostatic and hydrodynamic analysis, and performance prediction.*



Michael Shook is Vice-President for Engineering at AeroHydro, Inc. in Southwest Harbor, Maine, USA. He received a Bachelor of Liberal Studies (interdisciplinary) from Boston University in 1978. His interests include computer languages, computer graphics, geometric modelling, application frameworks, and component technologies.



Simon Shepherd is a second-year graduate student in Physics at Dartmouth College. He received his BA in physics from Middlebury College in 1989, and his MS in mechanical engineering from the University of Washington in 1991. From 1991 to 1993 he was employed on the MultiSurf project at AeroHydro, Inc. His current interests are plasma physics, fluid dynamics, numerical methods and Lagrangian dynamics.