# Generative Modeling: A Symbolic System for Geometric Modeling

John M. Snyder
James T. Kajiya
California Institute of Technology
Pasadena, CA 91125

## Abstract

This paper discusses a new, symbolic approach to geometric modeling called generative modeling. The approach allows specification, rendering, and analysis of a wide variety of shapes including 3D curves, surfaces, and solids, as well as higher-dimensional shapes such as surfaces deforming in time, and volumes with a spatially varying mass density. The system also supports powerful operations on shapes such as "reparameterize this curve by arclength", "compute the volume, center of mass, and moments of inertia of the solid bounded by these surfaces", or "solve this constraint or ODE system". The system has been used for a wide variety of applications, including creating surfaces for computer graphics animations, modeling the fur and body shape of a teddy bear, constructing 3D solid models of elastic bodies, and extracting surfaces from magnetic resonance (MR) data.

Shapes in the system are specified using a language which builds multidimensional parametric functions. The language is based on a set of symbolic operators on continuous, piecewise differentiable parametric functions. We present several shape examples to show how conveniently shapes can be specified in the system. We also discuss the kinds of operators useful in a geometric modeling system, including arithmetic operators, vector and matrix operators, integration, differentiation, constraint solution, and constrained minimization. Associated with each operator are several methods, which compute properties about the parametric functions represented with the operators. We show how many powerful rendering and analytical operations can be supported with only three methods: evaluation of the parametric function at a point, symbolic differentiation of the parametric function, and evaluation of an inclusion function for the parametric function.

Like CSG, and unlike most other geometric modeling approaches, this modeling approach is closed, meaning that further modeling operations can be applied to any results of modeling operations, yielding valid models. Because of this closure property, the symbolic operators can be composed very flexibly, allowing the construction of higher-level operators without changing the underlying implementation of the system. Because the modeling operations are described symbolically, specified models can capture the designer's intent without approximation error.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – curve, surface, solid, and object representations; geometric algorithms, languages, and systems

**Additional Key Words:** geometric modeling, parametric shape, sweep

## 1 Introduction

One way of representing a limited class of shapes uses sweeps. A sweep represents a shape by moving an object (called a generator) along a trajectory through space. The simplest sweeps are extrusions and surfaces of revolution, which sweep 2D curves. Sweeps whose generator can change size, orientation, or shape are called general sweeps. General sweeps that use 2D curve generators are called generalized cylinders [BINF71].

Several researchers have studied sweeps [GOLD83,CARL82b,WANG86, COQU87]. Barr's *spherical product* [BARR81], is an example of a sweep that uses a constant 2D curve generator with translation and scaling. Carlson [CARL82b] introduced the idea of varying the sweep generator. Wang and Wang [WANG86] explored sweeps of surfaces for use in manipulating numerically controlled milling machine cutter paths. Sweeps have been used in solid modeling systems for many years (e.g., GMSolid, ROMULUS). Lossing and Eshleman [LOSS74] developed a system using sweeps of constant 2D curves. Alpha_1, a modeling system developed at the University of Utah, has a much more sophisticated sweeping facility [COHE83].

One of the advantages of sweeps is their naturalness, compactness, and controllability in representing a large class of man-made objects. For example, an airplane wing is naturally viewed as an airfoil cross section which is translated from the root to the tip of the wing. At the same time its thickness is modified, it is twisted, swept back, and translated vertically according to other schedules. Two crucial questions remain concerning how sweeps fit into a general shape design and manipulation program:

- how can sweeps be specified by the human designer in a general and powerful way?
- what tools are appropriate to allow swept shapes to be rendered and simulated?

The generative modeling approach presented here extends the kinds of sweeps that can be conveniently specified, and provides high-level tools for their rendering and simulation. The approach specifies sweeps procedurally, in a fashion similar to other procedural specification methods in computer graphics: shade trees [COOK84], Perlin's texturing language [PERL85], and the POSTSCRIPT language [ADOB85].

A prototype system called GENMOD has been developed implementing these ideas, which includes a C interpreter, a curve editor, methods for several dozen primitive symbolic operators, and a multidimensional visualization library. While each piece of the system is fairly simple, we have found that combining all the pieces into a single system produces an extremely powerful geometric modeling tool.

## 2 Generative Modeling Overview

A *generative model* is a shape generated by the continuous transformation of a shape called the *generator*. As an example, consider a curve generator $\gamma(u)\colon \mathbf{R}^1 \rightarrow \mathbf{R}^3$, and a parameterized transformation, $\delta(p,v)\colon \mathbf{R}^3 \times \mathbf{R} \rightarrow \mathbf{R}^3$, that acts on points $p \in \mathbf{R}^3$ given a parameter $v$. A generative surface, $S(u,v)$, may be formed consisting of all the points generated by the transformation $\delta$ acting on the curve $\gamma$, i.e.,

$$S(u,v) = \delta(\gamma(u), v)$$

A cylinder is an example of a generative model. The generator, a circle in the *xy* plane, is translated along the *z* axis. The set of points generated as the circle is translated yield a cylinder. Mathematically, the generator and

transformation for a cylinder are

$$\gamma(u) = \begin{pmatrix} \cos(2\pi u) \\ \sin(2\pi u) \\ 0 \end{pmatrix} \qquad \delta(p, v) = \begin{pmatrix} p_1 \\ p_2 \\ p_3 + v \end{pmatrix}$$

yielding the surface

$$S(u, v) = \delta(\gamma(u), v) = \begin{pmatrix} \cos(2\pi u) \\ \sin(2\pi u) \\ v \end{pmatrix}$$

## 2.1 Parametric Functions and the Closure Property

If a generator is expressed as a parametric function, then a generative model built by transforming this generator is also a parametric function. Generalizing from the cylinder example, let a generator be represented by the parametric function

$$F(x): \mathbf{R}^l \rightarrow \mathbf{R}^m$$

A continuous set of transformations can be represented as a parameterized transformation

$$T(p; q): \mathbf{R}^m \times \mathbf{R}^k \rightarrow \mathbf{R}^n$$

where $p \in \mathbf{R}^m$ is a point to be transformed, and $q \in \mathbf{R}^k$ is an additional parameter that defines a continuous set of transformations. The generative model is the parametric function [1]

$$T(F(x); q): \mathbf{R}^{l+k} \rightarrow \mathbf{R}^n$$

The ability to use a generative model as a generator in another generative model will be called the *closure property* of the generative modeling representation. The use of parametric generators and transformations yields closure because transformation of a generator can be expressed as a simple composition of parametric functions, resulting in another parametric function. In fact, the use of parametric generators and transformations blurs the distinction between generator and transformation. Both are parametric functions; the domain of a generator must be completely specified, while the domain of a transformation is partly specified and partly determined as the image of a generator.

## 2.2 Terminology

Let $F: \mathbf{R}^n \rightarrow \mathbf{R}^m$ be a parametric function with scalar variables $x_1, x_2, \ldots, x_n$, called the *parametric variables* or *parametric coordinates*. The number of parametric coordinates on which $F$ depends, $n$, is called the *input dimension* of the parametric function. The number of components in the result of $F$, $m$, is called the *output dimension* of the parametric function. In this work, the domain of $F$ is a rectilinear region of $\mathbf{R}^n$, called a *hyper-rectangle*, of the form:

$$[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$$

Hyper-rectangles are convenient for sampling and integration of the parametric functions in a computer implementation. The image of $F$ over a specified hyper-rectangle defines the shape of interest.

## 2.3 Operators and Methods

One way of specifying parametric functions is by selecting a set of *operators*. An operator is a function that takes parametric functions as input and produces a parametric function as output. For example, addition is an operator that acts on two parametric functions $f$ and $g$, and produces a new parametric function, $f + g$. The addition operator is recursive, in that we can continue to use it on its own results or on the results of other operators, in order to build more complicated parametric functions (e.g., $(f + g) + h$).

Like the addition operator, all operators in the system are recursive; their results can be used as inputs to other operators. [2] Together with the closure property of parametric generators, this recursive nature of operators yields a modeling system with closure. That is, the designer is not prevented from using any reasonable combination of operations to specify shapes. For example, the addition operator can be applied to parametric functions of any input dimension ( e.g., curves or surfaces). It can also be applied to parametric functions of any output dimension, to perform vector addition, as long as the output dimension of its two arguments is identical.

Of course, it is not enough to represent parametric functions; we must also be able to compute properties about the parametric functions for rendering and analysis. Such computations can be implemented by defining a set of *methods* for each operator. One method evaluates the parametric function at a point in its parameter space. Other methods include symbolic differentiation of the parametric function and evaluation of an inclusion function (see [SNYD92a] for a discussion of inclusion functions). Section 3.2 discusses methods in more detail.

# 3 Symbolic Operators

## 3.1 Specific Operators

In this section, we examine specific operators that form a basis for a flexible variety of shapes. This set of operators will be used in Section 4 to show the capability of the generative modeling approach for combining such operators to build interesting shapes.

**Elementary Operators** Elementary operators include constants, parametric coordinates, arithmetic operators, square root, trigonometric functions, exponentiation, and logarithm. [3] The constant operator represents a parametric function with a real, constant value, such as $f(x) = 2.5$. The parametric coordinate operator represents a particular parametric coordinate, such as $f(x) = x_2$ where $x_2$ is the second component of the parametric domain, in a global coordinate system. Arithmetic operators are addition, subtraction, multiplication, division, and negation of parametric functions. They are useful for such geometric operations as scaling and interpolation, and in many other more complicated operations. They can also be combined to represent bicubic patches, NURBS, and other parametric polynomials.

Other elementary operators are useful in special circumstances. The square root operator, for example, is useful to compute the distance between points. The sine and cosine operators are useful in building parametric circles and arcs.

**Vector and Matrix Operators** Vector operators are projection, cartesian product, vector length, dot product, and cross product. Projection and cartesian product allow extraction and rearrangement of coordinates of parametric functions. Vector length, dot product, and cross product find many applications in defining geometric constraints on parameterized shapes.

Vector operator analogs of the arithmetic operators are also useful for geometric modeling. These operators include addition and subtraction of vectors, and multiplication and division of vectors by scalars. Matrix operators include multiplication and addition of matrices, matrix determinant, and inverse. Matrix multiplication is especially useful to define affine transformations, which are used extensively in simple sweeps (see Section 4.2). While these operators can be defined in terms of simple projection, cartesian product, and arithmetic operators, they are included as primitive operators for the sake of efficiency.

**Differentiation and Integration Operators** The differentiation operator returns the partial derivative of a parametric function with respect to one of its parametric coordinates. This is useful, for example, in finding tangent or normal vectors on curves and surfaces.

The integration operator integrates a parametric function with respect to one of its parametric coordinates, given two parametric functions representing the upper and lower limits of integration. For example, the function

$$\int_{b(u)}^{a(u,v)} s(v, \tau) d\tau$$

[1] More precisely, the generative model is the set of points in the image of $T(F(x); q)$ over a domain $U \subset \mathbf{R}^{l+k}$.

[2] It should be noted that the result of an operator can not always be used as input to another operator. Operators may constrain the output dimension of their arguments (e.g., an operator may accept only a scalar function as an argument and prohibit the use of functions of higher output dimension). In special circumstances, it may be desirable to constrain other properties of operator arguments. For example, the inversion operator expects its argument to be a monotonic scalar function. In this context, closure of the set of operators implies that an operator not arbitrarily prohibit any "reasonable" arguments, given the nature of the operator.

[3] GENMOD contains many more simple operators like these, listed in [SNYD92b].

can be formed by the integration operator applied to three parametric functions, where $s(v, \tau)$ is the integrand, $a(u, v)$ the upper limit of integration, and $b(u)$ the lower limit of integration. In general, parametric functions having any number of input parameters can be used as the integrand, or limits of integration. Integration can be used to compute arclength of curves, surface area of surfaces, and volumes and moments of inertia of solids.

**Indexing and Branching Operators** A useful operation in geometric modeling is concatenation, the piecewise linking together of a collection of shapes. For example, the concatenation of the set of $n$ curves $\gamma_1(u), \gamma_2(u), \ldots, \gamma_n(u)$, each defined over the parametric variable $u \in [0, 1]$, may be defined as

$$\gamma(u) = \begin{cases} \gamma_1(nu) & u \in [0, 1/n] \\ \gamma_2(nu - 1) & u \in (1/n, 2/n] \\ \vdots & \\ \gamma_n(nu - (n-1)) & u \in ((n-1)/n, 1] \end{cases}$$

The concatenation of surfaces or functions with many parameters can be defined similarly, where the concatenation is done with respect to one of the coordinates. This kind of concatenation is *uniform* concatenation, because each concatenated segment is defined in an interval of equal length $(1/n)$ in parameter space. It is commonly used in defining piecewise cubic curves such as B-splines.

Uniform concatenation is implemented using an *indexing operator*, which takes as input an array of parametric functions and an index function that controls which function is to be evaluated. Given the same $\gamma_i(u)$ curves used in the previous example, and an index function $q(x)$, the index operator is defined as

$$\text{index}(q(x), \gamma_1(u), \ldots, \gamma_n(u)) = \gamma_{\lfloor q(x) \rfloor}(u)$$

where $q(x) = nu$ results in the uniform concatenation of the $\gamma_i$ functions. In addition to the indexing operator, it is also useful to have a *substitution operator* to define uniform concatenation. The substitution operator symbolically substitutes a given parametric function for one of the parametric coordinates of another parametric function. For example, this can be used to represent $\gamma_i(nu-(i-1))$ given $\gamma_i(u)$, by substituting the function $nu-(i-1)$ for the parametric coordinate $u$.

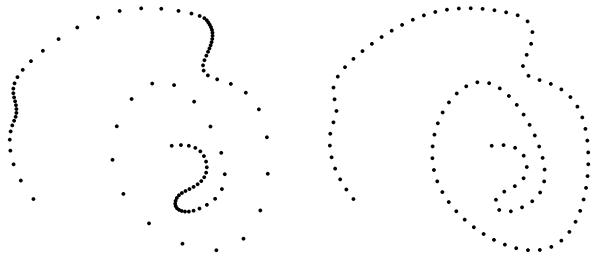
The index operator is a special case of a *branching operator*, an operator that takes as input a sequence of conditional functions and evaluation functions. The result of the branching operator is the result of the first evaluation function whose corresponding conditional is true. This multiway branch operator can be used to define a *nonuniform* concatenation of parametric functions where each concatenated segment need not be defined on an equally sized interval. Branching operators are also useful for finding the minimum and maximum of a pair of functions, for defining deformations that act only on certain parts of space, and for detecting error conditions (e.g., taking the square root of a negative number, or normalizing a zero length vector).

**Relational and Logical Operators** In order to support the definition of useful conditional expressions for the branching operators (and the constraint solution operator to be presented), we include the standard mathematical relational operators such as equality, inequality, greater than, etc., and the logical operators (such as "and", "or", and "not").

**Curve and Table Operators** Curve and table operators allow shapes to be specified from data produced outside the system. The curve operator specifies continuous curves such as piecewise cubic splines, produced using an interactive curve editor. The table operator is used to specify an interpolation of a multidimensional data set (GENMOD implements both linear and bicubic interpolation). For example, a simulation program may produce data defined over a discrete collection of points on a solid. The table operator interpolates this data to yield a continuous parametric function.

**Inversion Operator** Inversion of monotonic functions can be used, for example, to reparameterize a curve by arclength, as shown in Figure 1. Let $\gamma(t)$ be a continuous curve specifying the object's trajectory, starting at $t = 0$ and ending at $t = 1$. The arclength along $\gamma$, $\gamma_{\text{arc}}(t)$ is given by

$$\gamma_{\text{arc}}(t) = \int_0^t ||\gamma'(\tau)|| d\tau$$



**original** **reparameterized by arclength**

Figure 1: A parametric curve is reparameterized by arclength. Each dot represents a point on the curve along uniform increments of the curve's input parameter.

The integration and differentiation operators mentioned previously serve to define $\gamma_{\text{arc}}$. The reparameterization of $\gamma$ by arclength, $\gamma_{\text{new}}$, is then given by[4]

$$\gamma_{\text{new}}(s) = \gamma\left(\gamma_{\text{arc}}^{-1}(s\ \gamma_{\text{arc}}(1))\right)$$

This reparameterization involves the inversion of the monotonic arclength function, $\gamma_{\text{arc}}$.

Many other useful operations can also be formulated in terms of the inversion of monotonic functions, including the reparameterizing of curves and surfaces so that their parameters are matched by arclength, polar angle, or output coordinate to some other curve or surface. Inversion of monotonic functions in a single variable may be computed using fast algorithms, such as Brent's method [PRES86].

**Constraint Solution Operator** The constraint solution operator takes a parametric function representing a system of constraints, and produces a solution to the constrained system or an indication that no solution exists.[5] Two forms of solution are useful: finding any point that solves the system, or finding all points that solve it, assuming there is a finite set of solutions.[6] The operator also requires a parametric function specifying the hyper-rectangle in which to solve the constraints.

For example, the constraint solution operator can be used to find an intersection between two planar curves. Let $\gamma^1(s)$ and $\gamma^2(t)$ be two curves in $\mathbf{R}^2$. These curves could be represented using the curve operator of Section 3.1, or any of the other operators. The appropriate constraint is

$$F(s, t) \equiv (\gamma^1(s) = \gamma^2(t))$$

which can be represented using the equality relational operator. The constraint solution operator applied to $F$ produces a constant function representing a point, $(s, t)$, where the two curves intersect. Such an operation can be used to define boolean operations on planar areas bounded by parametric curves, which we will use in the screwdriver tip example of Section 4.4.

The constraint system can also be solved over a subset of its parameters, to yield a non-constant parametric function. For example, the constraint system $\gamma^1(r, s) = \gamma^2(t)$ can be solved over $s$ and $t$, resulting in a function that depends on $r$. The user therefore specifies not only a parametric function representing the constraint system, but also which parametric coordinates the system should be solved over, and which coordinates parameterize the system.

Constraint solution has application to problems involving intersection, collision detection, and finding appropriate parameters for parameterized shapes. A robust algorithm for evaluating this operator uses interval analysis, and is described in [SNYD92a].

[4] The $s$ parameter of $\gamma_{\text{new}}$ actually represents "normalized" arclength, in that $s$ varies between 0 and 1 to traverse the original curve $\gamma$, and equal distances in $s$ represent equal distances in arclength on the curve.

[5] Note that inversion operator of the previous section is a special case of the constraint solution operator.

[6] One form of the constraint solution operator produces a single solution, with an output dimension equal to the number of coordinates over which the constraint is solved. The other form returns the number of solutions as one output coordinate, followed by the solution points. The concatenated array of solution points is padded to some maximum length, $n$, specified by the user. Padding is done because parametric functions in GENMOD always have a fixed output dimension. The second form thus has output dimension $n + 1$.

**Constrained Minimization Operator** The constrained minimization operator takes two parametric functions representing a system of constraints and an objective function, and produces a point that globally minimizes the objective function, subject to the constraints. The operator also requires a parametric function specifying a hyper-rectangle in which to perform the minimization. The minimization operator has many applications to geometric modeling, including

- finding intersections of rays with surfaces
- finding the point on a shape closest to given point
- finding the minimum distance between shapes
- finding whether a point is inside or outside a region defined with parametric boundaries

A robust algorithm for evaluating parametric functions defined with the minimization operator uses interval analysis, and is described in [SNYD92a].

**ODE Solution Operator** The ODE operator solves a first order, initial value ordinary differential equation. It is useful for defining limited kinds of physical simulations within the modeling environment. For example, we can simulate rigid body mechanics, or find flow lines through vector fields. Figure 12 illustrates the results of the ODE operator for a simple simulation specified entirely in GENMOD.

Let $f$ be a specified parametric function of the form

$$f(t, y_1, y_2, \ldots, y_n): \mathbf{R}^{n+1} \rightarrow \mathbf{R}^n$$

The ODE operator returns the solution $y(t)$ to the system of $n$ first order equations

$$\frac{dy}{dt} = f(t, y)$$

with the initial condition

$$y(t_0) = y_0$$

Parameterized ODEs, in which $f$ and $y_0$ (and thus the result $y$) depend on an additional $m$ parameters $x_1, \ldots, x_m$, are also allowed. The user supplies the ODE operator with an indication of which parametric coordinates of $f$ are the $t$ and $y_i$ variables, and which are the additional parameters $x_i$.

GENMOD implements the ODE operators using a Numerical Algorithms Group(NAG) ODE solver. Similar operators, for solution of boundary value problems and PDEs, are also useful in a geometric modeling environment, but have not been implemented in the present GENMOD system.

## 3.2 Operator Methods

Let $P$ be an operator that takes $n$ parametric functions as inputs and produces the parametric function $p = P(f_1, \ldots, f_n)$. A method for $P$ is a function that can be evaluated by evaluating similar methods for the functions $f_1, \ldots, f_n$. A method on parametric functions is called *locally recursive for P* if its result on $p$ is completely determined by the set of its results on each of the $n$ parametric functions $f_1, \ldots, f_n$. Thus, a method to evaluate a parametric function at a point in parameter space is locally recursive for the addition operator because $f + g$ can be evaluated by evaluating $f$, evaluating $g$, and adding the result. A method to symbolically integrate a parametric function is not locally recursive for the division operator, because $\int f/g$ can not be computed given only $\int f$ and $\int g$. Generally, a locally recursive method can be simply implemented and efficiently computed.

We now examine specific methods useful in a geometric modeling system.

**Evaluation at a Point** Computation of points on a shape is necessary to approximate the shape for visualization and simulation. A method to evaluate a parametric function at a point in parameter space is locally recursive for most of the operators discussed previously. Several operators are exceptions: the integration, inversion, and ODE solution operators. [7] All three of these operators require their input parametric functions to be evaluated repeatedly over many domain points. For example, evaluation of the integration operator can be computed numerically using Romberg integration [PRES86, pages 123–125], which adds evaluations of the integrand over many points in its domain.

Two forms of the evaluation method have proved useful: evaluation at a single, specified point in parameter space and evaluation over a multidimensional, rectilinear lattice of points in parameter space. Evaluation of a parametric function over a rectilinear lattice gives information about how the function behaves over a whole domain, and is useful in "quick and dirty" rendering schemes. Although evaluation over a rectilinear lattice can be implemented by repeated evaluation at specified points, much greater computational speed can be achieved with a special method, as we will see in the Appendix.

The evaluation methods return an error condition as well as a numerical result. The error condition signifies whether the parametric function has been evaluated at an invalid point in its domain (e.g., $f/g$ where $g$ evaluates to 0, or $\sqrt{h}$ where $h < 0$). A failure error condition is also returned when the constraint solution or constrained minimization operators are evaluated in a domain in which there are no solutions.

**Differentiation** The differentiation method is used to implement the differentiation operator introduced in Section 3.1. The differentiation method computes a parametric function that is the partial derivative of a given parametric function with respect to one of the parametric coordinates. The partial derivative is computed symbolically; that is, the partial derivative result is represented using the set of symbolic operators. For example, the partial derivative with respect to $x_1$ of the parametric function $x_1 + \sqrt{x_1 x_2}$ yields the parametric function $1 + x_2/(2\sqrt{x_1 x_2})$, which is represented with the addition, multiplication, division, square root, constant, and parametric coordinate operators.

Although the differentiation method is not locally recursive for most operators discussed previously, it is still relatively easy to compute. For example, the partial derivative of the parametric function $h = \cos(f)$ depends not only on the partial derivative of $f$, but also on $f$ itself, since

$$\frac{\partial h}{\partial x_i} = -\sin(f)\frac{\partial f}{\partial x_i}$$

The differentiation method is therefore not locally recursive for the cosine operator, but may be computed simply if a sine operator exists. Similar situations arise for many of the other operators. Fortunately, it is a simple matter to extend a set of operators such that the set is closed with respect to the differentiation method, meaning that any partial derivative may be represented in terms of available operators. [8]

**Evaluation of an Inclusion Function** An inclusion function computes a hyper-rectangular bound for the range of a parametric function, given a hyper-rectangular domain. It is used in interval analysis algorithms to evaluate parametric functions defined with the constrained minimization and constraint solution operators. It is also useful to approximate shapes to user-defined tolerances, and compute CSG and offset operations. The uses and implementation of inclusion functions are fully discussed in [SNYD92a, SNYD92b].

Although an inclusion function computes a global property of a parametric function, it can often be computed using locally recursive methods. For example, an inclusion function method for the multiplication operator can be computed using interval arithmetic on the results of the inclusion functions for its parametric function multiplicands.

**Other Methods** Another useful method determines whether a parametric function is continuous or differentiable to a specified order over a given hyper-rectangle. Many times, algorithms for rendering and analysis require differentiability of input functions (e.g., multidimensional root finding methods). The differentiability operator can therefore be used to select whether an algorithm that assumes differentiability is appropriate, or if a more robust and slower algorithm must be used instead.

The differentiability/continuity method is locally recursive for most of the operators discussed previously, but there are exceptions. For example, the differentiability method for the division operator can not simply check that the two parametric functions being divided are differentiable. It must

[7] The derivative operator, and the constraint solution and constrained minimization operators are also exceptions. As we will discuss later, the evaluation method for the differentiation operator depends on the differentiation method, while the evaluation method for the constraint solution and constrained minimization operators uses the inclusion function method.

[8] For example, this implies that if the cosine operator is included in the set of primitive operators, then the sine operator must be included as well. Some operators, such as the constrained minimization operator, do not have analytically expressible partial derivatives. For these operators, the partial derivative must be computed numerically.

also check whether the denominator is 0 in the given domain. This can be accomplished using an inclusion function method.

Other operator methods, whose implementation is still a research issue, include determining whether a function $f: \mathbf{R}^n \rightarrow \mathbf{R}^n$ is one-to-one over a hyper-rectangle. A similar method is *degree*, defined as

$$d(f, D, p) = \text{cardinality } \{x \in D \mid f(x) = p\}$$

where $D \subseteq \mathbf{R}^n$.

### 3.3 Operator Libraries

While the primitive operators described in Section 3.1 form a powerful basis for a shape representation, they do not always match the operations the designer wishes to perform. In these cases, the designer can employ operators formed by composition of the primitive operators. The GENMOD system includes operator libraries which predefine hundreds of such higher level operators. The definitions of these operators are loaded from interpreted files when the program is first run, and can be dynamically modified and added to by the user.

For example, a simple but useful non-primitive operator is the linear interpolation operator, `m_interp`, whose GENMOD definition is [9]

```
MAN m_interp(MAN h,MAN f,MAN g)
{
    return f + h*(g-f);
}
```

The `MAN` type (for *manifold*) is the basic data structure in GENMOD, representing a parametric function. The `+`, `-`, and `*` operators have been overloaded to perform addition, subtraction, and multiplication of manifolds.

The `m_interp` operator takes three parametric functions as input: $f$ and $g$ are functions to be interpolated, and $h$ is the interpolation variable. The parametric functions $f$ and $g$ can be of any input or output dimension, as long as they have equal output dimension. This allows linear interpolation between two curves, surfaces, or even higher dimensional shapes. [10]

The closure property of the generative modeling approach means that such non-primitive operators can be very powerful. For example, the `m_arc_2pt_height` non-primitive operator used in the next section forms a circular arc connecting two 2D points and having a specified height above their line of connection. The 2D points supplied as arguments to this operator need not be constants but can depend on parameters, allowing convenient definition of the spoon of Section 4.3.

## 4 Examples

This section presents examples of generative shapes and their specification in GENMOD. It is meant to show how the generative modeling approach leads a designer to think about shape, and the size of the domain of shapes that can be represented. Many other examples can be found in [SNYD92b].

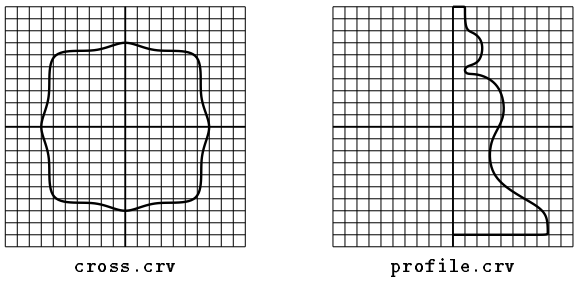
### 4.1 Lamp Bases and Profile Products

A profile product [BARR81] is perhaps the simplest nontrivial generative surface. It is formed by scaling and translating a 2D cross section according to a 2D profile. More precisely, a profile product surface, $S(u, v)$, is defined using a cross section curve, $\gamma(u) = (\gamma_1, \gamma_2)$, and a profile curve, $\delta(v) = (\delta_1, \delta_2)$, where

$$S(u, v) = \begin{pmatrix} \gamma_1(u)\delta_1(v) \\ \gamma_2(u)\delta_1(v) \\ \delta_2(v) \end{pmatrix}$$

A profile product may be defined in the GENMOD language as follows:

```
MAN cross = m_crv("cross.crv",m_x(0));
MAN profile = m_crv("profile.crv",m_x(1));
MAN lampbase = m_profile(cross,profile);
```



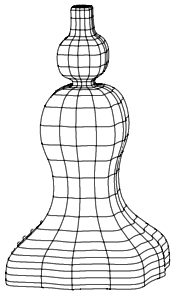



Figure 2: Lamp base example — A lamp base shape is represented by a profile surface. The GENMOD definition of a lamp base is shown, followed by graphs of the two curves (plotted between -1 and 1 in $x$ and $y$) used in the definition, and a wire frame image of the shape.

```
MAN m_profile(MAN cross,MAN profile)
{
    return @(cross[0]*profile[0],
             cross[1]*profile[0],
             profile[1]);
}
```

The `@()` operator, a C extension in GENMOD's language, is the cartesian product operator, which, in this case, combines three scalar functions into a 3D point. The `[]` operator returns a single output coordinate of a parametric function. In keeping with C language convention (and unlike the mathematical notation used in the definition of $S(u, v)$), coordinate indexing is done starting with index 0 for the first coordinate, rather than index 1.

Figure 2 presents an example of a profile product surface for a lamp base shape. It uses the `m_profile` operator defined above, and the primitive curve operator `m_crv`. The curve operator takes the name of a file, produced using a curve editor program, and creates a parametric curve that is evaluated over the parametric function specified as its second argument. In this case, the shape of the cross section curve is specified in the file `cross.crv`, and is evaluated over `m_x(0)`, representing parametric coordinate $x_0$. The profile curve is evaluated over parametric coordinate $x_1$ (`m_x(1)`).

### 4.2 Impeller Blades and Affine Transformations

An affine transformation shape uses a 2D or 3D curve generator and a transformation represented by a linear transformation and a translation. Let $\gamma(u)$ be a 3D curve, $M(v)$ be a linear transformation on 3D space, and $T(v)$ be another 3D curve. An affine transformation surface, $S(u, v)$, is given by

$$S(u, v) = M(v)\gamma(u) + T(v)$$

One method of representing affine transformations is to use $4 \times 4$ matrices (homogeneous transformations), allowing the composition of affine transformations using simple matrix multiplies.

Figure 3 presents an example of an affine transformation representing the impeller blade of a centrifugal compressor. The `m_transform3d` non-primitive GENMOD operator takes a vector and applies an affine transformation to it. Note that because the matrix transforms the cross section by

[9] GENMOD's language is based on ANSI C, with several extensions. The extensions allow overloading of the C operators, in order to more naturally express parametric functions. Several additional operators were also added.

[10] The binary arithmetic operators in GENMOD can be used in two modes. If the two parametric function arguments have the same output dimension, the operation is performed separately for each component on the corresponding components of the two arguments. If the output dimension of one argument is 1, and the other greater than 1, then the operation is performed on each component of the multicomponent argument with the same value of the scalar argument. Thus, $f + g$ denotes vector addition of $f$ and $g$ when $f$ and $g$ have the same output dimension, but $f * 2$ scales each component of $f$ by a factor of 2.

```
MAN u = m_x(0);
MAN v = m_x(1);
MAN cross = m_crv("bladecros.crv",u);
MAN blade = m_transform3d(@(cross,0),
    m_transz(m_interp(v,-1,1)) *
    m_transx(-0.5) *
    m_rotz(pi*m_crv("bladerot.crv",v)[1]) *
    m_transx(0.5) *
    m_scalex(m_crv("bladexscl.crv",v)[1]) *
    m_scaley(m_crv("bladeyscl.crv",v)[1])
);
```

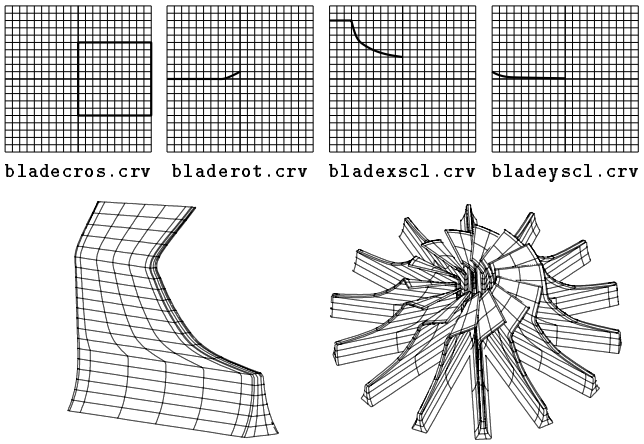



Figure 3: Impeller blade example — An impeller blade surface is represented using an affine transformation. A square cross section in the *xy* plane, which forms the bottom of the blade, is scaled separately in *x* and *y*, translated in *x*, rotated around *z*, translated back in *x*, and translated up the *z* axis.

premultiplying it, transformations that affect the cross section first must appear last in the list of multiplied transformations. The `m_transz`, `m_rotz`, `m_scalex`, and `m_scaley` are non-primitive operators that produce $4 \times 4$ matrices representing translation along *z*, rotation around *z*, and scaling of the *x* and *y* axes, respectively. They are multiplied together to define the complete affine transformation applied to a square cross section.

## 4.3 Spoons and Closed Offsets

Curve offsetting can also be used to define a cross section with a given thickness that surrounds a given non-closed curve (see Figure 4). An offset curve of radius $r$ around a 2D curve $\gamma(t)$ is given by

$$\gamma(t) + rn(t)$$

where $n(t)$ is the unit normal to the curve. The closed offset of a 2D curve $\gamma(t)$ of radius $r$ can therefore be defined as the uniform concatenation of 4 curve segments: the offset curve of $\gamma$ of radius $r$, the reversed offset curve of $\gamma$ of radius $-r$, and two semicircles of radius $r$ with centers at $\gamma(0)$ and $\gamma(1)$. The non-primitive GENMOD operator `m_ closed_offset` creates the closed offset to a 2D curve (first argument), of a given radius (second argument).

Figure 5 shows a spoon whose cross section is formed using this technique. In this case, the curve that is offset is a circular arc whose end points and radius are varied.

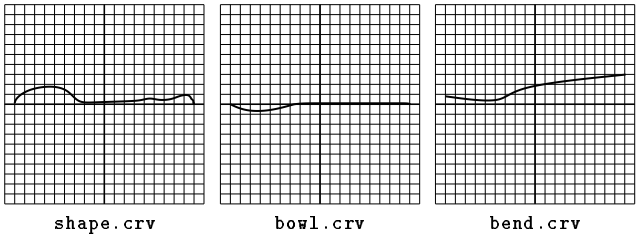
## 4.4 Screwdriver Tips and CPG

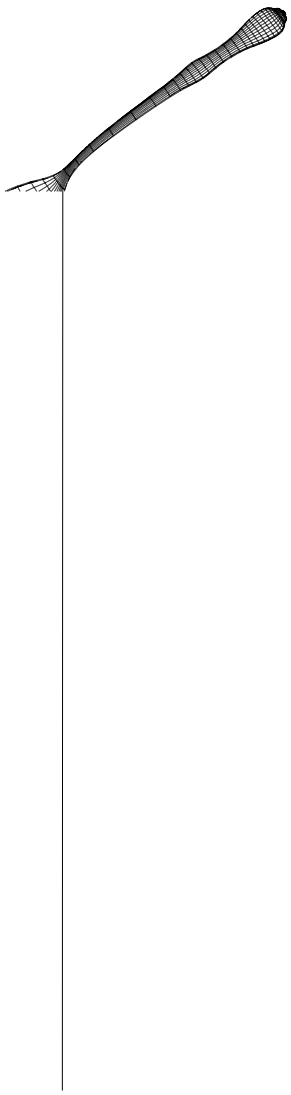
*Constructive planar geometry* (CPG) is the analog of constructive solid geometry for 2D areas. It is a modeling operation that uses Boolean set operations on closed planar areas to produce new planar areas. Figure 6 shows some examples of CPG operations.

Many objects can be represented as surfaces where each cross section is a Boolean set subtraction of one closed area from another. The fact that

Figure 4: Defining a cross section using offsets and circular end caps — A closed cross section may be defined in terms of a non-closed curve by concatenating two offset curves and two circular end caps.

```
MAN u = m_x(0), v = m_x(1);
MAN shape = m_crv("shape.crv",v);
MAN bowl = m_crv("bowl.crv",v);
MAN bend = m_crv("bend.crv",v);
MAN p1 = @(-shape[1],0);
MAN p2 = @(shape[1],0);
MAN arc = m_arc_2pt_height(p1,p2,bowl[1],u);
MAN closed = m_closed_offset(arc,0.01);
MAN spoon = @(shape[0],closed[0],closed[1]+bend[1]);
```



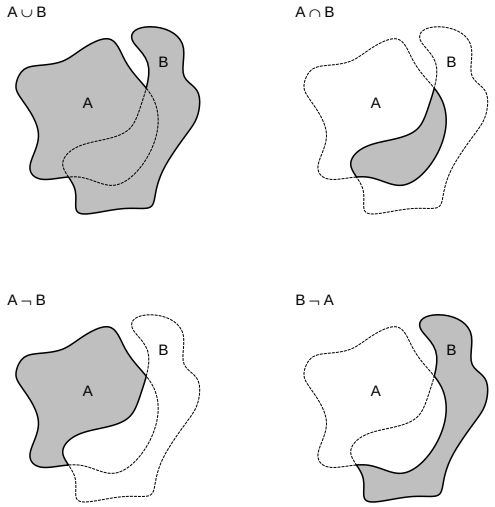



Figure 6: Constructive planar geometry – Two planar regions, A and B, are used in four binary CPG operations. We can compute the boundary of the result of a CPG operation by computing the intersections of the boundaries of the regions, dividing the boundaries into segments at these intersections, and concatenating appropriate segments.

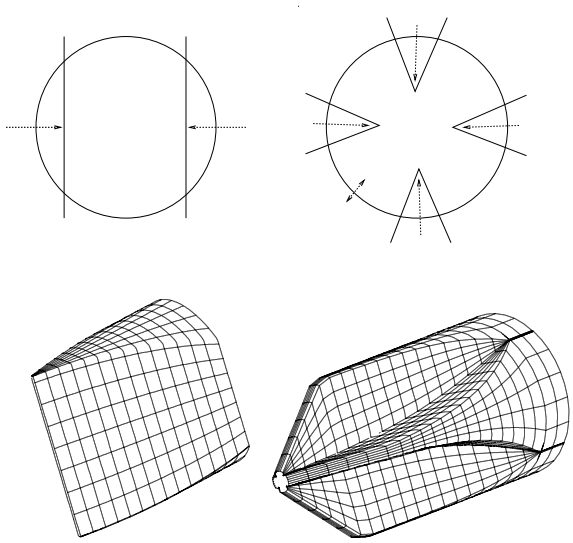


Figure 7: Screwdriver example – The tips of two screwdriver blades are constructed using CPG. The regular screwdriver on the left is generated using a cross section formed by subtracting two half-plane regions from a circle. The two half-planes are gradually moved toward each other as the cross section is translated to the tip of the screwdriver. The Phillips screwdriver on the right has a cross section formed by subtracting four wedge shaped regions from a circle. In this case, the wedge shaped regions are moved toward the circle's center as the cross section is translated to the tip of the screwdriver. while the circle is scaled down near the tip to yield a pointed blade.

## 5.1 Sampling

Uniform sampling of a parametric function involves evaluating the function over a rectilinear lattice of domain points. For each parametric coordinate $x_i$, we pick a number of samples, $N_i$. The parametric function $S$ is then evaluated over the $\prod_{i=1}^{n} N_i$ samples given by

$$\left(a_1 + \frac{i_1(b_1 - a_1)}{N_1 - 1}, \ldots, a_n + \frac{i_n(b_n - a_n)}{N_n - 1}\right)$$

where $a_i$ and $b_i$ define the hyper-rectangular domain of the parametric function. Each of the indices $i_j$ independently ranges from 0 to $N_j - 1$. This evaluation is done by calling the uniform evaluation method of $S$ (from Section 3.2). Uniform evaluation can be optimized so that it computes much faster than simple evaluation at each point in the rectilinear lattice of domain points, as discussed in the Appendix.

Adaptive sampling can be used to generate approximations that satisfy criteria [VONH87], where the sampling density varies over the parameter space. Robust approximation techniques that use inclusion functions are discussed in [SNYD92b]. The simple "evaluation at a specified point" method is used to compute the samples. Such evaluation can be optimized using caching, as discussed in the Appendix.

## 5.2 Interactive Visualization

A *visualization method* takes a shape and produces a renderable object, or produces a transformation that can be applied to a renderable object. There are four kinds of interactively renderable objects in GENMOD: points, curves, planar areas, and surfaces. A point is rendered as a dot in 2D or 3D space. A curve is rendered as a sequence of line segments. A planar region is rendered as a single polygon formed by the interior of an approximated curve.[11] A surface is rendered as a collection of triangles. A transformation can be applied to any of the other renderable objects, transforming it via the 4x3 affine transformation

$$p \rightarrow Mp + T$$

where $M$ is a $3 \times 3$ matrix and $T$ is a 3D vector.

Each of the visualization methods expects a shape of a given output dimension (e.g., a function $S(u, v)$ must have output dimension three to be used as input to the surface visualization method). Each visualization method also expects an input dimension at least as large as the intrinsic input dimension of the shape. For example, a function $C(t)$: $\mathbf{R} \rightarrow \mathbf{R}^3$ can be used in the curve visualization method, as can $D(t, s)$: $\mathbf{R}^2 \rightarrow \mathbf{R}^3$, since $C$ and $D$ have input dimension at least 1. On the other hand, a constant function is not appropriate for the curve method, nor is a function of a single coordinate appropriate for the surface method. The following table shows the number of intrinsic input parameters and output parameters of GENMOD's visualization methods:

| name | intrinsic dim. | output dim. |
|---|---|---|
| point | 0 | 2 or 3 |
| curve | 1 | 2 or 3 |
| planar area | 1 | 2 or 3 |
| surface | 2 | 3 |
| transformation | 0 | 12 |

Functions that have an input dimension greater than the visualization method's intrinsic dimension (e.g., a surface that deforms in time) are still valid input to the visualization method. The extra input coordinates, called *variable input parameters*, can be visualized with two techniques: *animation* or *superimposition*. The shapes are first sampled at various points in the variable input parameter space. Superimposition combines these shape instances in a single image, while animation renders the instances one at a time, according to the values of graphics input devices.

As an example, consider a parameterized family of 3D lines, $L(t, u, v)$ defined as

$$L(t, u, v) = S(u, v) + tV(u, v)$$

where $S(u, v)$ represents the line origin, and $V(u, v)$, the line direction. The $t$ parameter is the intrinsic parameter of the line; $u$ and $v$ are variable input parameters. This family of lines can be visualized by superimposition as in Figure 8, resulting in an image containing a 2D family of line segments. Alternatively, the $u$ and $v$ parameters can be animated, resulting in an image of a single line segment which interactively changes as the user controls, say, two dials. The user could also superimpose the $u$ parameter and animate $v$, resulting in a 1D family of line segments that changes in response to a single dial. Visualization methods therefore require an argument specifying which of the variable input coordinates are to be superimposed, and which are to be animated.

[11] The curve must not self intersect, and must lie in a plane. Planar regions are convenient for forming end caps of generalized tubes, where the tube cross-section is bounded by an arbitrary planar curve. Surfaces can also be used for this purpose, but are less convenient, since they require a 2D parameterization of the region's interior, rather than a simple boundary curve.