SGDL-Scheme: a high level algorithmic language for projective solid modeling programming

Jean-François Rotgé
GRCAO / University of Montreal
http://www.sgdl.com

Abstract

The SGDL/Scheme language, which is an absolutely pure functional language for volume programming, is a geometric extension of Scheme. It achieves a natural integration of the notions of volume and programming in a powerful algorithmic system. This capability of phrasing volume expressions by way of arithmetic expressions is the foundation of this new solid modeling system.

1 Introduction

Traditionally, geometric modeling systems base their algorithmic strategies upon Euclidean geometry, and topology or set theory, ([2]). On the other hand, when they are language-oriented they obey to a 3D object description scheme using euclidean surfaces as primitives and grammatical production rules as structural relationships descriptors,([1], [7],[8]). The basic principles used by SGDL, ([9],[10]),clearly break with those of conventional systems. The primitive recursive functions,([6], [3], [11]), denoted by PRF in the following, form the basis of the function-theoretic models and they make the link between the geometric modeling of the projective primitives and the combinatorics of volumes.

In particular, they allow us to unify projective geometry and logic through arithmetic. But another remarkable feature of the SGDL language allows the user to program volume objects in a programming environment stemming from the PRF theory. The SGDL/Scheme language belongs in fact to the class of functional programming languages based upon lambda calculus integrating the PRF, ([4],[5]).

2 SGDL Programming

2.1 Principles

The SGDL language allows you to describe complex volume objects as well as the methodologies associated with their design and construction. In this approach, the basic idea is that complex solids can be represented as compositions of the primitive by means of arithmetic and logical operators. The use of primitive recursive functions allows us to transform the notion of volumes into an arithmetic without division only involving integers based upon completely rigorous algorithmic procedures. The implicit nature of this arithmetic liberates the system from any explicit topological considerations and hence from the associated problems with special cases inherent to the implementation. According to [9], the key points of the SGDL system are:

- Modeling of geometric forms is realized by means of general algebraic surfaces and the modeling of polyhedral forms by degenerate surfaces.
- The surfaces are represented as implicitely given free-form surfaces and for a given degree the representation controls the general implicit equation of this degree as well as all "special" cases of lower degree.
- All SGDL geometric algorithms are genuine projective and any Euclidean or affine problem is treated as a specialization of a projective problem. This holds in particular for all deformations or geometric constraints.
- The construction of volume objects is done by arithmetic combinations of the implicit free-form surfaces establishing an equivalence between volume objects and arithmetic expressions.

2.2 Comparison with other solid modeling systems

There are obviously other solid modeling systems which make use of the Scheme language (e.g. 3D-Scheme or AutoCad) as a programming interface to their industrial solid modeling kernels. Most often these systems combine the amazing potential and simplicity of the Scheme language, which is vital and ideal for non-experts in programming, with the geometric kernel they use, without any matching implied or needed with the solid modeling system's theoretic principles. At the same time, the historical precedence of geometry and topology versus algorithmic and data computing theory is assured hence clearly separating the engineering process and the programming process.

But as far as performance is concerned, Scheme can not keep up with the theoretical problems encountered by conventional solid modeling kernels: whether it is combinatorial expansion of graphs or inadequate use of set-theory to solve some topological problems. On the other hand, SGDL has a completely different approach. With SGDL, it has been proven ([10]) that solid modeling can be defined as a geometrical branch of functionnal algorithmic theory which principles (e.g. integer arithmetic, recursivity, high order function) can thus be used during the modeling process.

The new SGDL-Scheme language, based on a new kernel that provides some purely functionnal and recursively defined volumic operators, turns the Scheme language into a pure solid modeling language. Due to the good performance of the SGDL-Scheme language and kernel, we can even think of migrating to real-time systems for geometric simulations.

2.3 The example of the cube.

The deformation of cube into a triple sphere is a good example of SGDL volume programming principles: it highlights the advantages and characteristics of the SGDL-Scheme language compared with conventionnal solid modeling systems.

In SGDL, three spheric Forms may be deformed in a continuous fashion into three intersecting plane quadrics. The figure Fig.1 illustrates the combination of these three spheric Forms combined with the help of f, a SGDL arithmetic operator. The concept underlying this operation is the logical concept of conjunction. The cube is obtained by setting $(SLcubsph\ a\ a\ f)$ and the sphere by setting
$(SLcubsph\ (/\ (*\ r\ (sqrt\ 2))\ 2)\ r\ f)$.

The $SLcubsph$ procedure is interpreted by SGDL-Scheme which evaluates all Scheme expressions and returns a procedure; this evaluated procedure in turn is in the Scheme syntax but only with SGDL volumic expressions: it is the volumic expression of our 3D model.

Once again, the evaluated procedure is then passed to the SGDL compiler for optimization of all real-time oriented processing. In our example, the compiler generates a pure arithmetic expression with operator f (which is passed on to *SLcubsph*; f may be *DLint* or *DLmul* or any other SGDL volumic operator) and its three operands corresponding to each $DLformz2$ (each operand being evaluated as a characteristic function of the homogeneous polynomial with integer coefficients).

The volumic expression can be evaluated for any point in space (each point being defined by its homogeneous coordinates). With this evaluation we can test if the point is inside the volume (the cube in our example), or on it, or outside it. Processing can be done with integer arithmetic, evaluating total functions defined by means of arithmetic PRF.

Figure 1: Algorithmic definition of cube

```
(define SLcubsph
  (lambda (a r f)
   (let
       (
        (X0 (vector 1 0 0 0))
        (X1 (vector 0 1 0 0))
        (X2 (vector 0 0 1 0))
        (numa (numerator a))
        (dena (denominator a))
        (numr (numerator r))
        (denr (denominator r))
        )
     (f
      (DLformz2
        (vector
         (vector (- numr) 0 0 denr)
         (vector numr 0 0 denr)
         X1 X2
         (vector numa numa 0 dena)
         (vector numa 0 numa dena)))
      (DLformz2
       (vector
        (vector 0 0 (- numr) denr)
        (vector 0 0 numr denr)
        X0 X1
        (vector numa 0 numa dena)
        (vector 0 numa numa dena)))
      (DLformz2
       (vector
        (vector 0 (- numr) 0 denr)
        (vector 0 numr 0 denr)
        X2 X0
        (vector 0 numa numa dena)
        (vector numa numa 0 dena)))
))))
```

3 Projective and Synthetic Programming

From a purely geometric point of view, projective geometry occupies a privileged place in the classification of the different geometries: Projective $\supset$ Affine $\supset$ Euclidean $\supset$ Metric and Projective $\supset$ Elliptic or Hyperbolic.

This classification suggests the possibility from a programming point of view, of algorithmic solutions to all problems in the lower level geometries using only the projective axioms.

In this case, we will be able to substitute a projective type of programming for the classical, essentially metric, programming, eliminating at the same time most of the special cases stemming from the lower geometries.

With the SGDL language, the projective programming of the design should enable us to describe the omnipresent algorithmic intelligence in the phases of construction of the technical and geometric design.

To recover and exploit this intelligence it is indispensable to provide functionalities capable of transcribing geometric notions into computer language, such as drawing a line from one point to another, recovering the intersection point of two lines, or finding the parallel to 2 lines through a point. This approach that restores the mechanism of the construction phases (for example, translation of a figure by means of a ruler with parallel edges), in contrast to the phases of a purely algebraic calculus (for example, a matrix translation of a figure) constitutes synthetic geometry programming. Combined with algebraic or analytic geometry programming this type of programming will provide the tools which are indispensable for the computerization of the process of volume design.

4 Unconditional Programming

4.1 The Geometric Aspect

This type of approach to programming may be defined as the search for a programming without "IF" that we call unconditional programming. From a purely geometric point of view, certain projective functions naturally avoid special cases. This is true, for example, for all functions concerning the incidences of points, lines and planes. These functions have been written in a hyperspatial projective coordinate system, the Grassman-Plcker system, that allows us to obtain a unique and systematic return for every function. For example the function for the intersection of a line and a plane will always return a quadruple of coordinates that can be unambiguously interpreted.

Thus, in the case of a plane and a line which do not coincide, the function will always return a projective point of which the 4th coordinate will be 0 if the line is in the affine space parallel to the plane. If the line coincides with the plane then all the coordinates of the point will be zero, showing that in this case it is undetermined. We could use this lack of determination to test whether or not a line lies in a plane, and avoid the need for error message and "IF". You have already had the opportunity to check the "disappearance of IFs" from a geometric point of view, when utilizing SGDL. We shall now direct your attention to the processing of numerical particularities.

4.2 The Numerical Aspect

The computer processing of all geometric algorithms uses either a symbolic formalism or a numerical formalism of the geometric data. For various reasons, such as increasing speed and reducing algorithms complexity, the conventional modelers adopt a numerical approach. The algorithms are based upon the most general analytic solutions. These solutions themselves may be the result of computerized symbolic calculations, made with special software.

One of the major problems in the numerical formalism compared with the symbolic formalism is the loss or lack of control of the precision of the computations, which may lead to disastrous solutions, devoid of any logical meaning. One of the main reasons for this problem is the representation of floating points in the machine, which leads to the use of a computer arithmetic differing in particular from the traditional arithmetic as soon as the size of the numbers manipulated escape the reserved memory capacity. Computer arithmetic with floating points has established a compromise between precision and memory space, which is acceptable in CAD, but strongly incompatible with most volume problems or geometric applications related to artificial intelligence.

An interesting compromise between the symbolic formalism and the numerical floating point formalism is the numerical integer formalism. In fact, as long as the calculations deal with rational numbers, projective geometry allows us to pass to integers via the projective coordinate systems. Then there is full precision in the calculations and the obtained results are comparable to those from the symbolic calculus. When the calculations involve irrational numbers, the way to integers can pass through an approximation of these irrationals by rationals. The results obtained have no longer the same precision as in symbolic calculus if exact simplifications of the irrationals are possible. On the other hand, compared to a numbering with floating point numbers, the places in the code where the approximations happen are precisely localized since they depend upon the programmer and not on the system. The control of coherence between algorithms and numbering remains possible.

The precision of the algorithms can thus be built on a unifying approach to number theory (rational arithmetic) and hyperspatial projective geometry. SGDL makes it possible to handle data with numbering by integers and this will guarantee, once the approximations or conversions from reals to integers have been made, that the precision of the calculations is assured. The basic principle allowing us to go from real numbers to integers, without any loss of precision, comes from the use of projective homogeneous coordinates instead of Euclidean coordinates. There are considerable advantages because there is no accumulation of rounding errors that might lead to meaningless results, and no reversing of logical decisions based upon numerical tests on approximate values. Moreover, this approach allows us to use only projective geometric functions even when the initial context of the geometric data is Euclidean. Thus, the example in Fig.2 in 3 dimensions that follows, illustrates one of the (non optimized) possibilities for passing from real Euclidean coordinates to integral affine coordinates that may then "feed" the projective geometry functions.

```
0 -> [6.5 1.75 .6]        Euclidean real number starting coordinates
1 -> [65/10 175/100 6/10] Conversion to rational decimal numbers
2 -> [13/2 7/4 3/5]        Euclidean gcd on each coordinate
3 -> [260/40 70/40 24/40] Reduction to a common denominator
4 -> [260 70 24 40]        Passing to homogeneous coordinates
5 -> [130 35 12 20]        Euclidean gcd on all the coordinates
```

Figure 2: from Euclidean to integral affine coordinates

Recall once again that the notion of infinity at the geometric level can be dealt with in an elegant way by using homogeneous coordinates. At the same time, this avoids the problems inherent in the machine infinity. A point at infinity, for example, will have its last coordinate equal to zero. It will be localized and it will be possible to reuse it as such.

5 Conclusion

The SGDL modeling system is integrated into a complete software for volume programming and rendering around the Scheme kernel. Libraries of projective, affine and euclidean geometry are available as well as libraries of predefined volume objects. These functions are accessible through the SGDL-Scheme language. Projective volume rendering is realized with the real-time oriented SGDLscript software.

References

[1] (1978) GONZALEZ, R.C. and M.G. THOMASON. *Syntactic Pattern Recognition: An introduction.* Addison-Wesley, London, 1978.

[2] (1989) HOFFMANN, C.M. *Geometric & Solid Modeling.* Morgan Kaufmann Publishers, Inc., San Mateo, 1989.

[3] (1971) KLEENE, S.C. *Introduction to Metamathematics.* Wolters-Noordhoff Publishing, Groningen, 1971.

[4] (1960) McCARTHY, J. "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I.". *Communications of the ACM*, April, 1960.

[5] (1961) McCARTHY, J. A basis for a mathematical theory of computation. In P.BRAFFORT and D.HIRSCHBERG, editors, *Computer Programming and Formal Systems.* North-Holland Publishing Company. Amsterdam, 1967.

[6] (1951) PETER, R. *Rekursive Funktionen.* Akademiai kiado. Akademisher Verlag, Budapest, 1951.

[7] (1989) PRUSINKIEWICZ, P. and J. HANAN. *Lindenmayer Systems, Fractals, and Plants.* Springer-Verlag, New York, 1990.

[8] (1990) PRUSINKIEWICZ, P. and A. LINDENMAYER. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.

[9] (1996) ROTGÉ, J. F. Principles of solid geometry design logic. In *CSG-96, Set-theoretic Solid Modelling Techniques and Applications*, pages 233–254. Information Geometers Ltd. Winchester. UK, 1996.

[10] (1997) ROTGÉ, J. F. *L'arithmétique des Formes: une introduction à la logique de l'espace.* PhD thesis, Faculté d'aménagement. Université de Montréal, Montréal, Canada, 1997.

[11] (1988) SUDKAMP, T. A. *Languages and Machines.* AddisonWesley Publishing Company, 1988.