

Djinn

Application programmers interface (API) for solid modelling

Adrian Bowyer, Stephen Cameron, Graham Jared, Ralph Martin,
Alan Middleditch*, Malcolm Sabin, John Woodwark

Acknowledgements

The Djinn *application programmers interface* (API) is being developed by a group of geometric modelling researchers (the authors of this report) and another group of applications researchers. Support for the project is provided by the Control, Design and Production Group of the UK Engineering and Physical Sciences Research Council. We are also grateful to EDS Ltd. for complimentary copies of the Parasolid modeller and associated documentation. Free documentation has also been received from the US Army Ballistics Research Laboratory and from Perspective Design Ltd..

Introduction

This paper describes the need for the Djinn project, the philosophy to be followed during interface design, the domain of the interface and some initial thoughts on functionality. It is a summary of [Bowyer 1995].

Background

Solid modelling research has been in progress for at least twenty-five years. Early arguments about the need for modellers ... given those excellent drafting systems ... already sound authentically medieval. Solid modelling is now a key technology without which effective CAD/CAM systems could not be built.

Despite previous research, many research issues remain. For example, conversion of boundary models (a.k.a. B-reps) to set-theoretic models (a.k.a. CSG) [Shapiro93], problems due to degeneracy and inaccuracy [Fortune93], and mechanisms for variational modelling [Hoffmann93]. In addition, the technology is being extended to include tolerance modelling [Boyer91, Juster92, Turner91], multi-dimensional [Parry95] and cellular [Rossignac90, Middleditch92, Paoluzzi93] modelling, feature modelling and recognition [Shah91], and physically based modelling [Essa93].

Kernel modelling systems

The early solid modelling research groups made modellers available to applications researchers, e.g. TIPS [Okino73], BUILD [Braid80] and PADL [PAP74], but a modeller acquired on these terms can become a liability after research on the modeller itself has ceased. The alternative use of a commercial modeller, in the hope of more permanent support, was unattractive because commercial modellers were integral parts of CAD systems and they could not be linked to other programs. Eventually, some commercial modellers were written in a form suitable for incorporation in more than one CAD system, e.g. Romulus [Faux83]. More recently, some modellers have been explicitly conceived as *kernel modellers*, their functionality packaged beneath an API, e.g. the boundary modellers ACIS [Braid89] and Parasolid [EDS95], and the set-theoretic modeller Svlis [Bowyer94].

Unfortunately, current kernel modellers can make applications researchers dependent on a particular vendor, a particular modelling paradigm and a particular programming language. Even systems written in object-oriented languages such as C++, whose major advantage is the ability to *hide* unnecessary detail from the user, usually allow access to data classes and thus paradoxically commit users to a particular data-structure. With applications researchers locked into a particular vendor, it is increasingly difficult to get modelling research ideas tested in practice. Even new functionality that is extremely relevant to a particular

* Principal Investigator: Alan E Middleditch, Director, Centre for Geometric Modelling and Design, Brunel University, Uxbridge UB8 3PH, U.K.

application is rejected because applications researchers are not prepared to make the painful transition from a commercially supported modeller to a research modeller with dubious support. Thus, progress in applications research is impeded and modelling research results remain untested.

Some modeller independent APIs have been specified, e.g. AIS (Application Interface Specification) originally developed for CAM-I and now a draft ANSI standard [Ranyak94]. Operations such as set-theoretic (a.k.a. Boolean) combinations, can be performed on either boundary or set-theoretic models, but a number of operations address the boundary data-structure or set-operator tree directly. It is not possible for an implementation to support the entire interface without internal representation conversions, but boundary to set-theoretic conversion techniques remain elusive despite heroic efforts [Shapiro93]. The STEP product data exchange standard (ISO 10303) [Owen93] has a related API called SDAI [ISO94] that also accommodates both boundary and set-theoretic models, but again only in parallel and support for set-theoretic models is rather perfunctory. Inertia probably also renders STEP unsuitable as a research interface [Eastman94].

Djinn objectives

In contrast to AIS and SDAI, the aim of the Djinn¹ project is to ignore existing modelling paradigms in favour of more abstract geometric data types and operations. The Djinn API will be defined in terms of those abstractions, but implementations can employ either the boundary representation or set-theoretic paradigm, or any other representation that unambiguously defines solid objects, e.g. voxels. Applications will manipulate geometric abstractions, not boundary graphs, set-expressions, oct-trees, or any other implementation specific structure. Implementations could employ representation conversion, but it is not mandated by the interface specification. It would be foolish to define an API predicated on the availability of a general solution to the historic obstacle of boundary to set-theoretic conversion and conversions to and from voxel modellers is even more challenging. Such quantised solids are often used as approximations from which many of the original characteristic entities such as surfaces have been eliminated.

The Djinn API will also be independent of specific programming languages. Further, there is no plan for any direct support for graphics, although operations that provide data in an appropriate form for generating graphics will be included.

Although implementations of the Djinn API are intended as tools for applications research, the Djinn project is itself research. It is not immediately obvious that a practical abstraction exists in which operations can be specified independent of a particular modelling paradigm. As far as the authors are aware, this approach is new.

Properties of the Djinn API

Conformance ... Although it should have a major influence on future standardisation efforts, the Djinn interface is not intended to become a standard, because Djinn implementations will inevitably be incomplete and provide additional functions in order to make the results of modelling research available to applications researchers. Since a Djinn-compliant implementation supports all Djinn functions, lack of functionality is reflected by Djinn specified error parameters. Thus, a trivial Djinn-compliant implementation is a set of functions that return consistent error messages and has no geometric functionality. A Djinn-compliant implementation usually provides a *coherent* subset of Djinn functionality. A modeller which only allows primitives orthogonal to the axes cannot logically support a general rotation function, and a modeller using spatial-enumeration representations cannot support functions that access faces because it has no useful face information.

¹ Djinn: a demon of Arabian mythology, able to assume many forms, and to become "invisible at pleasure" [*Brewer's Dictionary of Phrase and Fable*].

Function definitions ... The Djinn API is primarily the specification of a set of functions, defined in terms of pre- and post-conditions. Formal methods have been considered, but resource constraints on the project do not allow a fully fledged formal approach.

Djinn functions are defined in terms of the point-set abstraction . Set-theory has been strongly associated with ‘CSG’ modelling, but all entities manipulated by boundary modellers can be also defined as point sets. Boundary modellers have typically exploited topological concepts such as Euler operators, but correct topology follows from solidity, not the other way around.

Data types ... The Djinn interface employs three categories of data-type:

- *Visible data-types*, such as points. All fields of the data is accessible by both the application and the Djinn implementation.
- *Hidden data-types*, such as the implementation-dependent object representations that result from Djinn modelling operations. Access to the data is only possible through Djinn interface functions.
- *External data-types*, i.e. references to data of externally defined types. The data is not directly accessible to the Djinn implementation and possibly not to the program calling the Djinn interface. Implementations interrogate and manipulate such external data using external functions.

It is not obvious whether the representations of some geometric entities should be *visible* or *hidden* .

State ... An implementations of the Djinn API may have state, i.e. data that persists within that implementation between function invocations. However, there will be no externally perceived state, except a single state variable provided to permit initialisation. An initialisation function is necessary for the creation of efficient implementations in some languages.

Side-effects ... Djinn functions will use only their argument values in their computations, and the results will effect only the values of their arguments. Arguments will be defined as input, output or input/output parameters and input parameters will not be modified. Data present in output parameters when a function is called may be overwritten.

Errors ... Djinn functions return error indications of the following types:

- An indication of the accuracy of numerical results. Djinn implementations will inevitably have very different capabilities in this regard.
- Special values of geometric types, such as ‘empty object’ or ‘unknown object’.
- Hidden error representations to indicate that the primary operation of a function has not been achieved. Djinn provides functions to interrogate such errors.

Approximate models ... Modelling systems sometimes represent exact geometry by an approximation, e.g. a quadric-surface might be approximated by a set of facets. An approximate model may be used to assist computations on an exact model, such as a voxel model used as a localising spatial segmentation. Such approximations cause larger numerical errors than those inherent in ‘exact’ models using floating-point approximations for real numbers, but the form of geometric results is unaffected.

Approximate models may be used also as the primary representation of exact geometry. In this case, when a geometric entity is manipulated, it could be an entity of the approximate model, an approximation to the corresponding entity in the exact model, or a reconstruction of that exact entity. For instance, a face of a voxel approximation to a quadric-surfaced solid could be a voxel face, a set of voxels corresponding to an exact face, or a reconstruction of the face, made by some fitting process. There is no distinction from the application viewpoint, because all these data-types are hidden, but the application must be able to control the accuracy of approximation appropriately for the actual computation performed.

The accuracy of approximation could be specified whenever the application requests Djinn to create a piece of geometry. Subsequent interrogations for numerical properties such as volume would reflect this approximation. The alternative taken by Djinn is to allow the application to specify the desired interrogation accuracy. This may require a model to be recreated to a greater accuracy by the implementation, but it hides the approximation and provides individual accuracy control.

Labels ... Because no boundary representation or set-operator tree is accessible through the Djinn interface, pieces of geometry are accessed using unique labels. Attaching labels to geometry as it is instantiated is relatively straightforward, but modelling operations destroy geometry. Djinn will accept labels associated with defunct geometry and provide an appropriate error message.

Attributes ... In addition to the labels by which geometric entities are identified, it is necessary to handle application dependent and thus externally defined attributes of those entities. The attributes attached to Djinn entities are thus references to application data.

Persistent models ... The Djinn API will provide input and output facilities to retain models between modelling sessions. Since the output data will inevitably reflect internal representations and thus will be implementation dependent, it cannot form part of the Djinn specification. Unfortunately, archiving of this sort precludes a file created by one implementation being read by another.

Djinn geometry

Dimensionality ... While acknowledging geometric modelling research in dimensions greater than three [Parry95], Djinn will support entities of three or fewer dimensions, i.e. points, curves, surfaces and solids, all embedded in three-dimensional space. That excludes both higher-dimensional entities and 'drafting' constructions. However, facilities will be provided to export two dimensional results of sectioning operations and to import two-dimensional geometry.

Motion ... Djinn will not provide explicit support for objects in motion nor for parameterised objects. Initial discussions with applications researchers indicate that in many applications where moving or variable solids are important, the required geometric computations can be adequately performed on static and invariant models (e.g. using sweeps or by sampling).

Geometric domain ... Every geometric modeller is limited in the range of shapes that it can manipulate. Djinn will support natural quadrics, tori, probably cyclides and solid helices (spring-shapes), together with analogous two dimensional entities.

Free-form surfaces ... Although current commercial modellers favour B-splines, and specifically NURBS, very many more types of parametric and implicit curves and surfaces are of current research interest. Therefore, the Djinn interface supports free-form curves and surfaces generically to enable users to 'plug in' proprietary geometric types and types common in their own industries. This is done by the import of references to external data through the primary Djinn interface and geometric interrogation by the Djinn implementation through a secondary interface defined as a part of Djinn. The latter defines functions to compute properties such as surface normals and ray intersections. These functions can be provided by a surface geometry package.

Cellular models ... Djinn will provide support for cellular and non-manifold objects, and for hierarchies of features. Cellular models will be represented as collections of point-sets [Rossignac91, Middleditch92].

Geometric operations

Modification ... Such operations will be defined in terms of point-sets. This is natural for both the regularised and non-regularised set-operations that are necessary [Middleditch94]. Unfortunately, operations such as blends [Vida94, Woodwark87] and 'local operations' such

as the tweaks [Grayer80] usually associated with boundary models will need considerable reformulation because they do not reliably map one point-set to another.

Transforms ... There is a large range of transforms applicable to geometric entities. Rigid-body and scaling transforms will be supported, and more general affine and projective linear transforms will be considered, as will non-linear transforms such as Möbius transforms that map spheres to spheres.

Sweeps ... Djinn will provide a range of sweep operations to create objects defined by the region covered during the motion of another object. The Minkowski sum (also called set sum or vector sum) [Middleditch88] provides a convenient formalism to define sweeps in terms of point-sets. Unfortunately, since the Minkowski sum cannot define all useful sweep operations (e.g. a profile rotating along a spine curve), alternative definitions must be developed.

Properties ... Djinn will provide support for a range of intrinsic properties of point-sets, e.g. connectedness, genus and integral properties such as volume. Some support will also be provided for more complicated properties that are naturally stated in terms of point-sets, e.g. the Voronoi diagram [Lavender92], even though they do not necessarily map easily onto existing modelling technologies.

Relations ... Djinn will provide functions to compute properties of one object relative to another. These include set-membership tests and boundary connectivity relationships, as well as interference and distance properties. Some of these properties are more easily computed by set-theoretic methods and others by boundary methods, but all such properties are geometric and therefore computable in principle using either paradigm.

Conclusions

This document gives an overview of the topics that the authors have been considering as they have started to define the Djinn application programmers interface for solid modelling. Our next publication will be a book containing the Djinn interface specification and the rationale for its functionality.

References

- A. Bowyer, Svlis: Introduction and User Manual, Information Geometers 1994.
- A. Bowyer et al, Introducing Djinn - A geometric interface for solid modelling, Information Geometers Ltd, 1995.
- M. Boyer and N.F. Stewart, "Modelling spaces for toleranced objects", International Journal of Robotics Research 10,5 (pp 570--582), October 1991.
- I.C. Braid, "Notes on a geometric modeller", Cambridge University Computer Laboratory, Document 101, June 1980.
- I.C. Braid, "Improving product models and kernel modellers", Organisation of Engineering Knowledge for Product Modelling in Computer-Integrated Manufacturing, ed. T. Sata (pp 275--299), Elsevier, 1989.
- C.M. Eastman, "Out of STEP?", Computer-Aided Design 26,5 (pp 338--340), May 1994.
- I.A. Essa, S. Sclaroff and A.P. Pentland, "Physically based modelling for graphics and vision", Chapter 5 of Directions in Geometric Computing, ed. R.R. Martin (pp 161--218), Information Geometers, 1993.
- S. Fortune, "Progress in computational geometry", Chapter 3 of Directions in Geometric Computing, ed. R.R. Martin (pp 81--127), Information Geometers, 1993.
- A.R. Grayer, "Alternative approaches in geometric modelling", Computer-Aided Design 1,4 (pp 189--192), May 1980.
- C.M. Hoffmann, "On the semantics of generative geometry representations", Advances in Design Automation/ (ASME DE Vol.~65), 1993.

- N.P. Juster, "Modelling and representation of dimensions and tolerances: a survey", *Computer-Aided Design* 24,1 (pp 3--17), 1992.
- D.A. Lavender, A. Bowyer, J.L. Davenport, A.F. Wallis and J.R. Woodwark, "Voronoi diagrams of set-theoretic solid models", *IEEE Computer Graphics and Applications* 12,5 (pp 69--77), September 1992.
- A.E. Middleditch "Application of vector sum operator", *Computer-Aided Design* 20,4 (pp 183--188), May 1988.
- A.E. Middleditch, "Cellular models of mixed dimension", Brunel University Centre for Geometric Modelling and Design, Technical Report BRU/CAE/92:3, April 1992.
- A.E. Middleditch, "'The bug' and beyond", *Set-theoretic Solid Modelling: Techniques and Applications* (pp 1--16), Information Geometers, April 1994.
- N. Okino, H. Kakazu and H. Kubo, "TIPS-1: technical information processing system for computer-aided design, drawing and manufacturing", *Computer Languages for Numerical Control (Proceedings of the PROLAMAT 73 Conference, Budapest)*, ed. J. Hatvany, North-Holland (pp 141--150), 1973.
- J. Owen, *STEP: An Introduction*, Information Geometers, 1993.
- A. Paoluzzi, F. Bernardini, C. Cattani and V. Ferrucci, "Dimension-independent modelling with simplicial complexes", *ACM Transactions on Graphics* 12,1 (pp 56--102), January 1993.
- S.J. Parry-Barwick and A. Bowyer, "Multidimensional set-theoretic feature recognition", *Computer-Aided Design*.
- P. Ranyak "Application Interface Specification (AIS): Volume I: Functional Specification", CAM-I Report R-94-PM-01 (Version 2.1), 1994.
- J.R. Rossignac and M. O'Connor, "SGC: a dimension-independent model for point sets with internal structures and incomplete boundaries", *Geometric Modeling for Product Engineering*, eds. M.J. Wozny, J.U. Turner and K. Preiss, (Proceedings of IFIP WG 5.2 Working Conference, Rensselaerville, September 1988) (pp 145--180), North-Holland, 1990.
- J.R. Rossignac and A.A.G. Requicha, "Constructive non-regularised geometry", *Computer-Aided Design* 23,1 (pp 21--32), January/February 1991.
- J.J. Shah, "Assessment of features technology", *Computer-Aided Design* 23,5 (pp 331--343), June 1991.
- V. Shapiro and D.L. Vossler, "Separation for boundary to CSG conversion", *ACM Transactions on Graphics* 12,1 (pp 35--55), January 1993.
- J.U. Turner and A.B. Gangoiti, "Tolerance analysis approaches in commercial software", *Concurrent Engineering* 1,2 (pp 11--23), 1991.
- J. Vida, R.R. Martin and T. Varady, "A survey of blending methods that use parametric surfaces", *Computer-Aided Design* 26,5 (pp 341-365), May 1994.
- J.R. Woodwark, "Blends in geometric modelling", *The Mathematics of Surfaces II*, ed. R.R. Martin (pp 225--297), Oxford University Press, 1987.
- PAP74, "An Introduction to PADL", University of Rochester Production Automation Project, Report TM 24, December 1974.
- ISO94, "Industrial automation systems and integration ... Product data representation and exchange ... Part 22: Implementation methods: Standard data access interface specification", ISO TC 184/SC4 N280 (committee draft), 1994.
- EDS95, "Overview of Parasolid", Parasolid Business Unit, EDS, March 1995.

Faux83, "Romulus", Proceedings of a Geometric Modelling Seminar, Robinson College Cambridge, 12--14 December 1983, ed. I.D. Faux, CAM-I Report P-83-GM-01, CAM-I Inc., Arlington, Texas, 1983.