

Ten Questions that Arose in Designing the Djinn API for Solid Modelling

Adrian Bowyer
School of Mechanical Engineering
University of Bath
Bath BA2 7AY UK
a.bowyer@bath.ac.uk

Stephen Cameron
Computing Laboratory
University of Oxford
Parks Road, Oxford OX1 3QD UK
stephen.cameron@comlab.ox.ac.uk

Graham Jared
School of Industrial and Manufacturing Sciences
Cranfield University
Cranfield, Bedford MK43 0AL UK
g.jared@cranfield.ac.uk

Ralph Martin
Department of Computer Science
University of Wales, Cardiff
Cardiff CF2 3XF UK
ralph@cs.cf.ac.uk

Alan Middleditch
Centre for Geometric Modelling and Design
Brunel University
Uxbridge UB8 3PH UK
alan.middleditch@brunel.ac.uk

Malcolm Sabin
Numerical Geometry Ltd
26 Abbey Lane, Lode
Cambridge CB5 9EP UK
malcolm@geometry.demon.co.uk

John Woodwark
Information Geometers Ltd
47 Stockers Avenue
Winchester SO22 5LB UK
jrw@inge.com

Abstract

Djinn is an API for solid modelling which is defined in the language of point-sets and is thus independent of any particular modelling data-structure (e.g. boundary representations or CSG trees). In designing this API, several significant and interesting questions have arisen, including: the basic feasibility of the approach; deciding how to support cellular models; providing facilities for navigation without traditional data-structures; addressing the problems of 'tweaks' and blends; ensuring that labels are preserved under geometric operations; permitting regions in which changes to a model have taken place to be identified; coping with approximate models; deciding whether to support variational models; keeping sweeps and transforms within a reasonable domain; and providing facilities to handle the many types of parametric surface. These questions are not all fully answered, but progress continues towards a published interface and C++ language binding.

1. Introduction

One of the strongest themes in computing in recent years has been the *encapsulation* of software, including the increasing popularity of object-orientation and the rise of various 'standard' interfaces, some of which are more standard than others. Such interfaces may start life attached to a particular piece of software, and then be used more generally; or they may be specified in a free-standing way, in the hope that they will find favour with system writers. Both sorts of interface can be found in software for computer-aided design and manufacture and, in particular, there are now a number of published interfaces through which solid modellers may be accessed.

However, one thing that all existing modelling interfaces have in common is that they acknowledge and refer directly to the data-structures of the underlying

ing solid modellers: typically, boundary models (a.k.a. B-reps), set-theoretic models (a.k.a. CSG), or approximate models such as oct-trees and facets. Certain interfaces have, indeed, addressed more than one data-structure (e.g. [5]), but in a portmanteau sense, by including separate sets of functions appropriate to each structure.

The Djinn project is quite simply an attempt to specify a usable interface that supports a discourse at a level above specific modelling data-structures (see also [8, 11]). It has only been possible to conceive of one language in which such a discourse can be conducted: the language of point-sets. And so the Djinn project boils down to an attempt to specify a solid modelling API in terms of point-sets.

The project started in October 1994 and will be completed next year. As well as the seven authors of this paper, there is a Djinn Application Group, representing research users of solid modelling systems. This group provides balance with a strong advocacy of useful functionality, even when it is difficult to describe in terms of point-sets.

So far, only an early analysis of the aims of the project has been published [1]. Next year, there should be a book-length report and some papers expounding different aspects in more detail. In the meantime, a statement of some of the most significant questions that have been encountered in designing Djinn may be of interest. Not all of them have complete answers, and may not have, even when the project is complete. Each unanswered question may be seen as a potential research project.

2. Contents of Djinn

In order to make sense of the rest of the paper, the contents of the Djinn interface are now briefly reviewed.

Djinn consists of a data abstraction and a set of functions acting on that abstraction, which are outlined below. In translating these concepts into an API that can be implemented, a number of programming aspects must be considered; and some of these are also mentioned.

2.1 Data abstraction

The data abstraction hides the modelling data used by a particular implementation so that the calling program is not aware of the data-structure being used under the API. The abstraction is the domain of point-sets in zero, one, two and three dimensions, augmented by cells [3, 6] and persistent labels [2], which together al-

low subsets of objects or surrounding space to be identified. By default, an object will consist of two cells: its boundary and its interior. Cells may be enabled and disabled for the purposes of 'counting' operations such as the calculation of integral properties.

2.2 Functions

In Djinn, objects are created and modified by geometric and set functions, and interrogated by geometric and topological functions. Overall, their effect is to support the range of operations that are commonly included in solid modellers. For the purposes of this paper, it is not necessary to do more than itemize some of the more important types of function:

- Creation of objects: generation of hidden variables from visible ones. These functions encapsulate the creation of the usual modelling primitives, such as the quadrics etc.
- Set-operators: with optional regularization [7].
- Navigation: expressed in terms of bounding relations between cells.
- Labelling: creation and maintenance of cell labels.
- Sweeps: translation, rotation and other common sweeps.
- Transformations: rigid-body and other common transforms.
- External geometry: allowing Djinn objects to contain pieces of geometry of which the representation is not known by the modeller, and which are accessed through a 'bottom interface' using geometrical queries.
- Import and export: allowing the exchange of model data with other software.

2.3 Programming aspects

In software engineering terms, Djinn tries to follow an object-oriented paradigm. In particular, problems with state and side-effects are avoided as far as possible.

- State is data that is perceived externally to persist within the implementation between procedure invocations. It must reside in Djinn objects, not in the underlying modeller.
- A side-effect changes the data represented by a variable input to a procedure. Occasionally, side-effects must be permitted for efficiency reasons, but such cases are limited and well-defined.

A C++ language binding is being developed and will be published as part of next year's report.

3. The ten questions

Q1. *How can geometry be made available through an API without any data-structures? Surely that doesn't make sense?*

A1. It's quite difficult to ignore representations that everyone knows and loves. The Djinn project has sometimes seemed like a game of snooker on a table with several holes in the middle: of which the two largest are marked 'B-rep' and 'CSG'. Just keeping the balls in play requires concentration. Arguably, the entire project is a single exercise in object-oriented design: trying to identify a class of objects and the enquiries that must be made about them. By focusing on this paradigm, most of the snooker balls can be induced to stay on the table.

The important thing is to keep the data abstract. In Djinn, even points and transforms are hidden data types. To create a point, a visible point coordinate vector is passed to Djinn, and a hidden point is returned. Treating points in this way may seem unnecessarily complex, but points at infinity and indeterminate points may arise as intermediate data and, if they are hidden, they can be supported in a consistent manner.

Note that, as a result of using point-sets as the universe of discourse, most topological and dimensional problems can be kept in abeyance, and the somewhat baroque questions of indexing topological components that have enthralled the boundary-modelling community are never manifested across the interface.

Because of the definition in terms of point-sets, an extension to arbitrary dimensionality would be feasible. However, this would increase the complexity of the API, without much benefit, as most underlying modellers would not be able to support higher dimensionalities.

Q2. *Early on, it was decided that Djinn would need to support cellular models. How can cells be generated and accessed without reinventing existing data-structures?*

A2. It is possible to create a cellular structure incrementally through the Djinn API, using set-operations to partition objects. But there are certain predictable cases in which quicker methods need to be supported. Two of the most obvious are the segmentation of a object into connected components and the partitioning of one- and two-dimensional objects into connected cells,

each having some specified degree of derivative continuity. Note that Djinn is strict about the latter method, and correct implementations will *not* allow 'artefacts', such as patch boundaries, to surface through the API, unless of course such boundaries happen to correspond to genuine derivative discontinuities.

Once a cellular structure has been constructed, it can be navigated using labels, or by interrogations which identify bounding relations (i.e. adjacencies).

Q3. *People know how to get about the existing types of models, including faceted and oct-tree approximations. How can these structures be hidden without making models impossibly difficult to use?*

A3. Traditional model data-structures allow some operations to be done blindingly fast, while others are awkward. Djinn provides navigation methods based on adjacency and bounding relations between cells; they are intended to cover a range of requirements, all with reasonable efficiency. Of course, the underlying modeller may well *implement* navigation operations (e.g. traversing the boundary of a solid) logically rather than geometrically.

Q4. *Some shape modelling techniques rely on specific representations. What about constructions that cannot easily be specified in terms of point-sets, such as 'tweaks' to boundary models and many types of blend?*

A4. This is certainly not an easy question. Many useful operations on solid models—especially boundary models—are not formulated in terms of point-sets [13], because this has never been necessary to get working CAD systems out of the door. However, if they cannot be formulated as changes to point-sets, it is probably difficult to show that such operations are robust. In fact, many are obviously not: the classic example is the overdone 'tweak' that pulls one part of the boundary of a solid through another part.

There are some blending techniques—those usually associated with CSG models—which can be formulated in terms of point-sets [14], and Djinn includes calls to accommodate them. But it has to be admitted that the blends used in commercial systems are largely boundary-based. Here is a gap in the underpinnings of solid modelling, which will only be closed when blend operations are better understood and robustness issues addressed.

Note that the inclusion of 'CSG-type' blends does not force implementations to support them, as partial implementations of Djinn are explicitly allowed—indeed will be the norm. Partial implementations will sup-

port all functions, but ‘missing’ functions will return only error messages.

An interim solution to the blends problem is to treat blend surfaces as external geometry.

Q5. *Without a navigable structure, the labelling of geometry becomes particularly important. How can labels persist through geometric operations which may create, destroy or partition geometry?*

A5. Label maps are returned with the results of operations, as component parts of Djinn objects. These maps must be prepared by the modeller underlying Djinn, and indicate the relationship between the label-spaces before and after an operation.

Q6. *How can the calling program tell what parts of a model have been changed by a given operation, without searching the whole of it?*

A6. This is an urgent concern of the Djinn Application Group, and another tricky problem. Using explicit models, changes can be identified as lists of items in the data-structure (boundary elements, CSG sub-trees, octs etc.) that have been altered. Of course, this approach cannot be taken in Djinn.

Another existing approach to this problem is to implement ‘callbacks’, in which the modeller under the API makes calls back to the program above the API to notify it as each change is made. This is not used either, because of the difficulty of ensuring that the arguments to any further function calls that might be made during the callback are valid; the ‘hidden’ data in some objects is likely to be in an intermediate, implementation-dependent state.

Two ways of restricting the search for changed parts of a model are incorporated into Djinn. The label map is an indication of change, and an update box is available to indicate a region of space within which the changes are contained.

Q7. *How can the API cope when the underlying model will not be exact and may be a gross approximation (e.g. facets or an oct-tree)?*

A7. The usual accuracy problems are exacerbated when the nature of the underlying model is unknown; and there are qualitative issues associated with grossly approximate models. For instance, what is an edge of an oct-tree model? Is it the edge of an oct, or a set of octs approximating one of the ‘original’ edges?

In essence, Djinn pretends that its hidden geometry is perfect. However, interrogations necessarily provide

inaccurate results and an estimate of precision is associated with them. In other words, the calling program receives results appropriate for the data that it put in, irrespective of the underlying model, which contributes only to the inaccuracy of those results. Obviously, a particular implementation should have a modeller under Djinn that provides the right accuracy for the application.

The actual, inaccurate models under the interface only see the light of day if they are converted into visible geometry for export.

Q8. *Is there any support for parametric or variational modelling in the API?*

A8. No. It has been difficult enough to try to recast existing modelling operations strictly in terms of point-sets. When it comes to variable point-sets, it is not easy even to decide into what formalism to *attempt* to cast the modelling operations. Quite deliberately, Djinn does not even retain any history of the ‘static’ model.

However, it is recognized that commercial modellers increasingly use parametric and variational facilities, combined with shape features. Nothing prevents parametric, constraint-, or feature-based modelling from using Djinn to represent *instances* of a variable model, and some work in this direction is already taking place [4].

One problem with parameterized and variational models (like CSG trees) is that a lot of the model structure is invisible when one looks at a picture of the object. So changes to an old model can often produce unexpected effects, depending on the ‘history’ of how the model was created. At least one commercial system (Hewlett-Packard’s SolidDesigner) seems to be in rebellion against the history-file concept, apparently for just this reason of unpredictability and the resulting problems of editing old models.

In effect, a CSG model *is* its history, and there are problems with boundary modellers too. For instance, if a piece of boundary is deleted, it can no longer be recreated by ‘replaying’ a previous Boolean: a thoroughly undefined hole appears in the model, and the system is forced to interpolate it, as robustly as possible, from adjoining surfaces. Djinn does not face this problem, because Djinn functions are all well-defined point-set operations. However, making such operations reflect a user’s intent efficiently is likely to be a continuing challenge.

New formalisms in the area of variational modelling would be welcomed (and see [10]).

Q9. *Is it possible to include a set of sweeps and transforms that does not put impossible requirements on the domain of the underlying modeller? What happens to the cellular structure under sweeps and transforms?*

A9. A 'reasonable' set of sweeps and transforms is included in the interface, based on discussions with the Application Group. Maintaining the cellular structure is not a problem for factorizable sweeps (those where every point in the result can have a unique ordered pair identified in directrix and generator). It may be possible to include other sweeps too, but Djinn is structured so that the underlying modeller has the option of signalling that it cannot cope with maintaining the cell structure under a given sweep.

Q10. *How can parametric surfaces be accessed and used to bound solids?*

A10. Parametric surfaces present two problems for Djinn. Firstly, in the research world at least, there are so many formulations that it would not be feasible to provide a call for every possible operation on every possible structure; although NURBS are a *de facto* standard which it may be convenient to support explicitly. Secondly, while parametric surfaces are well-defined point-sets, they are not half-spaces, and so cannot be combined into a solid by simple set-operations.

The first problem is addressed by support for external geometry. The second problem requires the specification of a 'fill' operation [12], which recognizes connected components of free space, separated by point-sets (which may well be parametric surfaces, but could be objects of any dimensionality). Implementation of the 'fill' operation would depend on the underlying modeller. For CSG modellers, implementors would need to call on recent research results [9]. But it would be a simpler operation for, say, oct-trees; and a nominal one for boundary modellers, although ensuring real robustness might not be so trivial.

4. Conclusions

Why bother? It is already apparent that the point-set approach does not meet universal acclaim from the commercial world. But this is, surely, the fate facing anyone who seeks to introduce 'unnecessary' formalism into some area of software where increasing functionality has, for years, been the overriding objective.

Existing solid modellers are almost all used within a computer-aided design context, with the emphasis on

the 'aided'. Progress in other areas of computing indicates that more and more responsibility will inevitably be thrown on to the machine. As that happens, and there is no human user inventively working around inaccurate results and failed operations, then matters of robustness and the need for precise definitions of functions and data abstractions will inevitably bubble to the top of even the commercial agenda.

In the meantime, researchers may care to consider the benefits of a neutral interface, allowing the latest modeller to be plugged into the latest application, and thus facilitating a closer link between experiments in solid modelling and experiments in its application. This is an important shorter-term objective of the Djinn project.

5. Disclaimer

There is likely to be some disparity between the partial solutions briefly reviewed here, and the contents of the final Djinn report and language binding.

6. Acknowledgements

The authors gratefully acknowledge support for the Djinn project by the Control, Design and Production Group of the UK Engineering and Physical Sciences Research Council.

We are also very grateful for the participation of the Djinn Applications Group: Cecil Armstrong, Jonathan Corney, Pat Fothergill, Tony Medland, Frank Mill, Jonathan Salmon and Ken Swift.

7. References

1. A. Bowyer, S.A. Cameron, G.E.M. Jared, R.R. Martin, A.E. Middleditch, M.A. Sabin and J.R. Woodwark, *Introducing Djinn: a Geometric Interface for Solid Modelling*, Information Geometers, 1995.
2. J. Kripac, "Topological identification system: a mechanism for persistently naming topological entities in history-based parametric solid models", PhD Thesis, Czech Technical University, Prague, 1994.

3. A.E. Middleditch, "Cellular models of mixed dimension", Brunel University Department of Computer Science Report BRU/CAE/92:3, April 1992.
4. A.E. Middleditch, "A kernel for geometric features", *submitted to Solid Modeling '97: Fourth ACM Symposium on Solid Modeling and Applications*, Atlanta, May 1997.
5. P. Ranyak, "Application interface specification (AIS): Volume I: functional specification", CAM-I Report R-94-PM-01 (Version 2.1), 1994.
6. J.R. Rossignac and M.A. O'Connor, "SGC: a dimension-independent model for point-sets with internal structure and incomplete boundaries", in *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) (145–180), North-Holland, 1990.
7. J.R. Rossignac and A.A.G. Requicha, "Constructive non-regularized geometry", *Computer-Aided Design* **23**,1 (21–32), January 1991.
8. J.R. Rossignac, "MAGISET: architecture and programming interface for a universal modeler", in *Theory and Practice of Geometric Modelling* (Proceeding of the Second Blaubeuren Conference, October 14–18, 1996, Tübingen), 1996.
9. V. Shapiro and D.L. Vossler, "Separation for boundary to CSG conversion", *ACM Transactions on Graphics* **12**,1 (33–55), January 1993.
10. V. Shapiro, "What is a parametric family of solids?", Proceedings of the Third ACM Symposium on Solid Modeling and Applications, Salt Lake City, eds. C.M. Hoffmann and J.R. Rossignac (43–54), May 17–19, 1995.
11. V. Shapiro, "Maintenance of geometric representations through space decompositions", *to appear, International Journal of Computational Geometry & Applications*
12. S.W. Thomas, "Set-operations on a sculptured solid", PhD Thesis, University of Utah (University of Utah Computer Science Report UUCS-84-009), June 1984.
13. J. Vida, R.R. Martin and T. Varady, "A survey of blending methods that use parametric surfaces", *Computer-Aided Design* **26**,5 (341–365), May 1994.
14. J.R. Woodwark, "Blends in geometric modelling", in *The Mathematics of Surfaces II*, ed. R.R. Martin (Proceedings of the IMA Mathematics of Surfaces Conference, Cardiff, September 1986) (255–297), Clarendon Press, 1987.