Geometric Modeling Problems in Industrial CAD/CAM/CAE

George Allen
Chief Technologist & Technical Fellow
Siemens PLM Software, Shanghai

Introduction

This paper is a broad discussion of some of the geometric problems that arise in the real-world use of commercial CAD/CAM/CAE systems. The goal is to help the research community understand what problems exist in commercial environments today, which ones are important, and how they can help address them via future work. The focus will be on the design and manufacturing of mechanical devices (as opposed to electrical devices, buildings, or other types of products). First we briefly survey some of the business problems faced by companies who design and manufacture mechanical products, so that we can try to relate these business problems to possible future innovations. Next we describe some necessary technology background, and then three sample problems. One problem (filleting) is fairly narrow and specific, while the other two (history-based modeling and performance) are broader and more general. For comparison, we classify research papers from a recent conference, to assess their alignment with the needs and interests of the CAD/CAM/CAE business.

Business Background

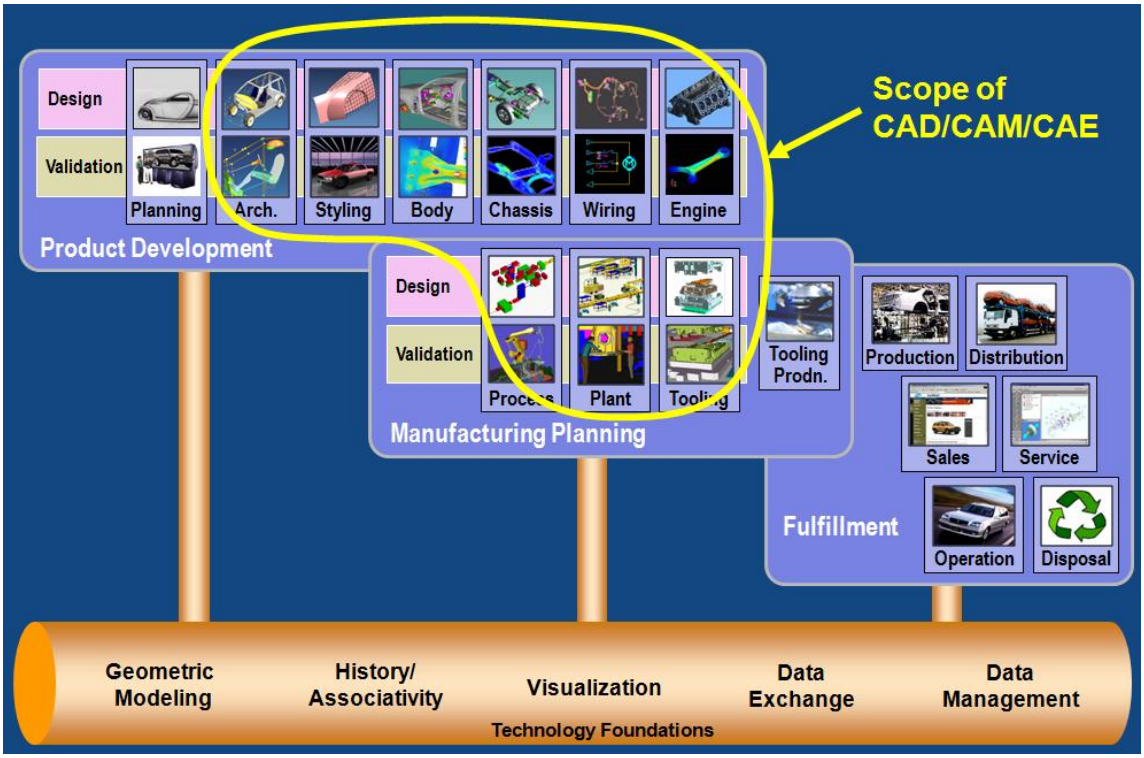
First, some background on our company and our industry might be helpful. My company is Siemens PLM Software, formerly known as UGS, formerly known as Unigraphics Solutions, and so on. We develop software for Product Lifecycle Management (PLM). This includes the following software categories and applications:

Category	Applications
CAD/CAM	NX Unigraphics, I-deas, Solid Edge
CAE	Nastran, Femap
Data Management	Teamcenter
Digital Manufacturing	Tecnomatix product suite
Components	Parasolid, D-Cubed components

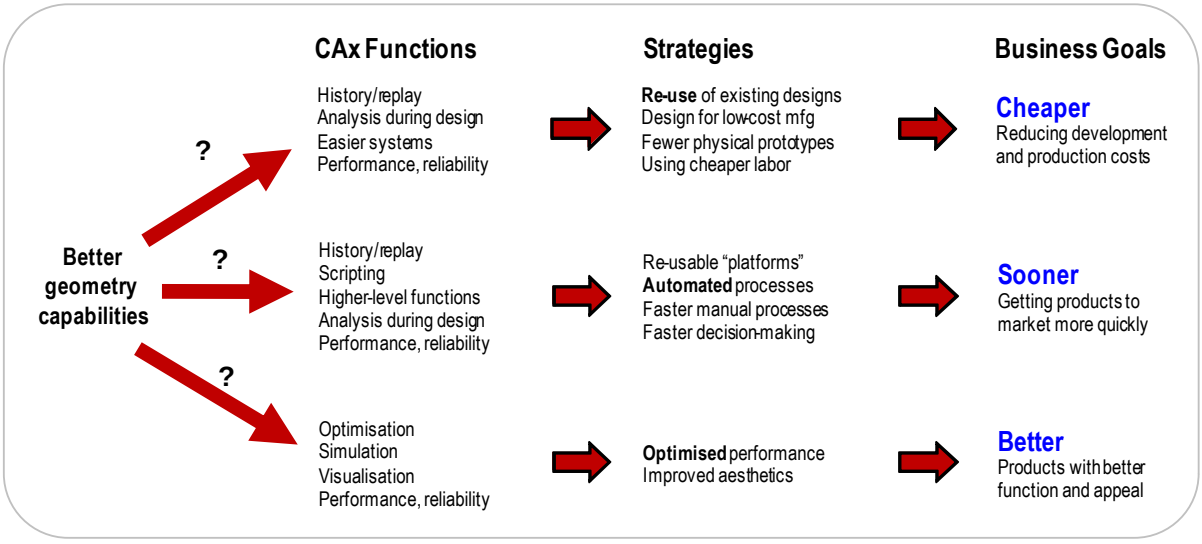
This software is developed by a team of around 2000 people in 30 development centers around the world. Our annual revenues are roughly $1B, and our R&D budget is roughly $200M. This is fairly typical – most of our major competitors are companies of similar size and make-up. Our customers are manufacturing companies in a variety of industries, including General Motors, Boeing, General Electric, Nissan, Canon, and thousands of others. In addition, we sell software components (like our Parasolid geometry kernel) to other CAD/CAM companies, including our competitors.

Our CAD/CAM/CAE software, NX, has about 30 million of lines of code, which has been developed over the course of around 30 years. We have been able to incorporate many new technologies over the years, but fundamental and far-reaching changes are sometimes very expensive to implement, so they must also be very beneficial, in order to be justified.

Our focus in this paper is CAD/CAM/CAE, which is about half of our business. The following picture shows where CAD/CAM/CAE systems are used within the overall product lifecycle, using the case of an automobile as an example.



The following diagram tries to explain why our customers use our software. It is grossly simplified, of course, but the fundamental ideas are correct. Basically, our customers want to develop products that are cheaper, sooner, and better, and our software must help them do this.



So, when assessing a software development project (or a research problem) it is important to ask how the results will contribute to these three goals. The introductory paragraphs of a research paper often state that the topic of the paper is "an important problem in CAD/CAM/CAE". If this is really true, then it should be possible to trace the connection between the research results and the business benefits of "cheaper", "sooner", or "better". This is not an easy task, and few authors (or reviewers) are able to do it effectively, so the true relevance of research is often poorly understood, unfortunately.

Technology Background

This section provides a very brief introduction to some of the basic technologies used in modern CAD/CAM/CAE systems, with an emphasis on the geometric modeling areas.

Geometry

The geometric objects used in our software are shown in the table below:

Surfaces	Curves
Plane Circular cylinder Circular cone Sphere Torus NURB surface Surface of revolution Extruded surface Offset surface [*] Rolling ball fillet surface [*]	Line Circle Conic (ellipse, parabola, hyperbola) NURB curve Parameter-space curve Intersection curve [*]

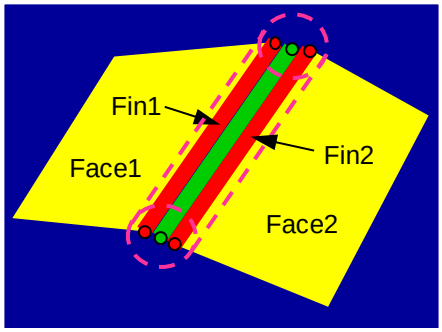
The objects marked [*] are procedural in nature – they are not spline approximations. So, for example, an offset surface is represented via a reference to its base surface, plus an offset distance, not by a NURB surface.

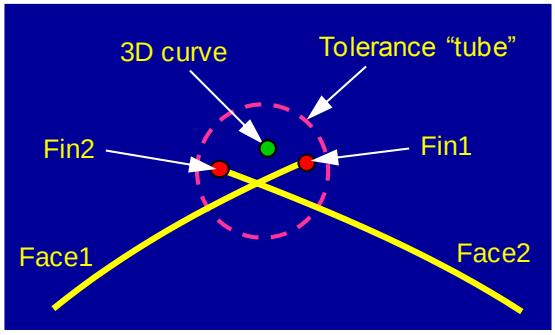
The range of geometric objects in any other CAD/CAM/CAE would be roughly the same. In fact, since many systems use our Parasolid geometry kernel, the objects would be exactly the same. Exchange standards such as IGES and STEP also contain roughly the same range of object types.

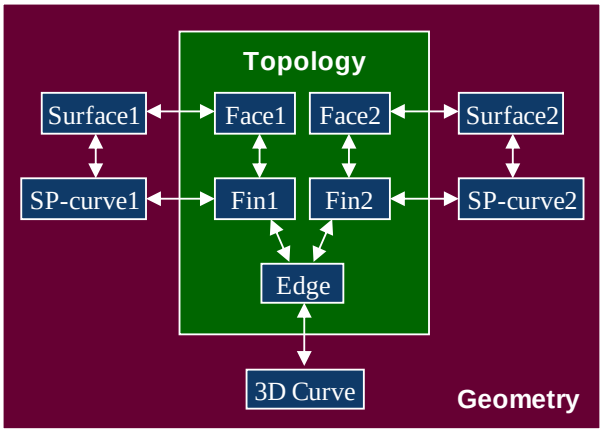
This set of geometric object types has not changed for around 15 years, and it is not likely to change very much in the future, either. These objects seem to be sufficient to support all the CAD/CAM/CAE functions that are of interest to us, and adding new object types is very expensive. Note that subdivision surfaces and triangular surface patches (two popular research topics) are not used in our systems, or in any other major CAD/CAM/CAE system.

Trimming and Topology

The surfaces are trimmed, and then connected together using topological data structures to form bodies. An edge where two faces meet is a fairly complex object consisting of three somewhat separate representations. The topological data consists of an edge and two fins (sometimes called half-edges or co-edges, or edge-uses). There is a 3D curve attached to the edge, and a surface-parameter-space curve (sp-curve) attached to each fin, as shown in the pictures below. The 3D curve might be a simple analytic one, a procedural intersection curve, or a spline curve. The sp-curves are typically low-degree spline curves defined in the parameter spaces of the owning surfaces. The three curves are required to lie inside a certain tolerance “tube” in order for the model to be considered valid. The data structures are illustrated in the following pictures:

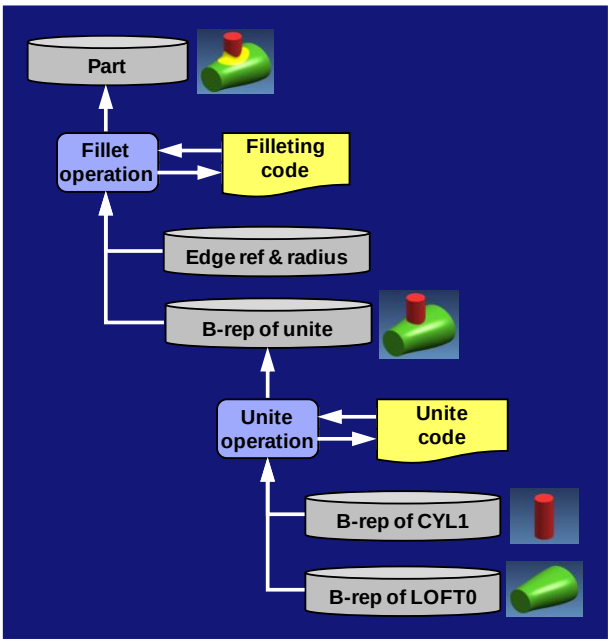






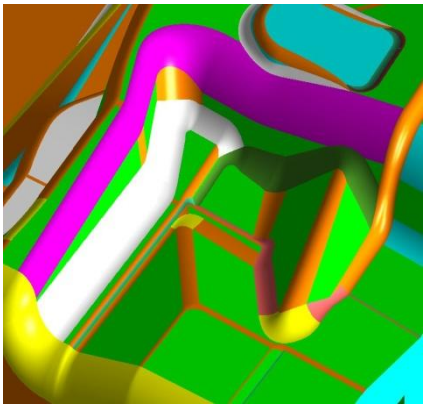
History-Based Models

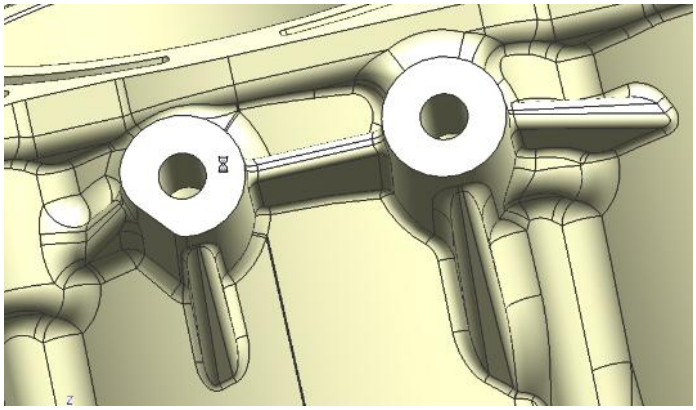
Most contemporary systems store the sequence of operations that was used to construct a model in a graph-like data structure. The sequence of operations is called the "history" of the model, and the data structure is called the "history tree". This sequence of operations can then be replayed with new inputs. This provides an extremely powerful editing facility. But, perhaps more important, it allows the creation of "template" models that can easily be "morphed" and re-used in new design situations.



Problem #1: Filleting

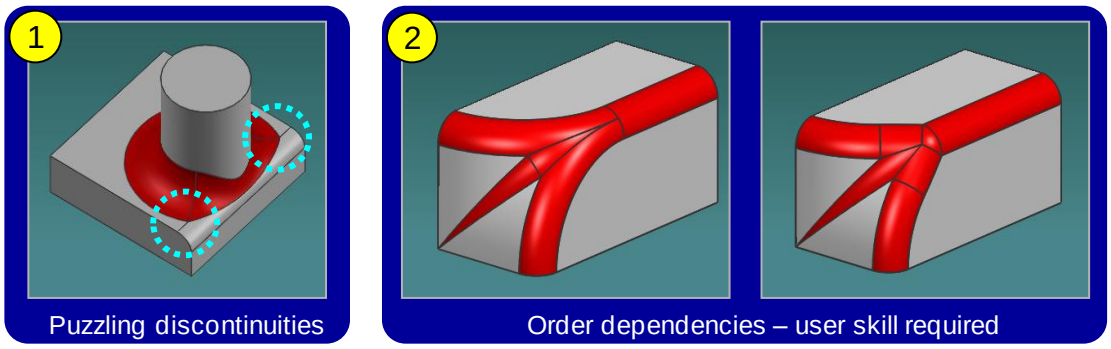
In some manufacturing processes, sharp corners cannot be produced, so all sharp corners in the CAD model have to be "rounded" or "filleted". The typical approach uses computations that mimic the earlier manual methods. The fillet surface is formed as the envelope of a sphere that rolls in simultaneous contact with two sets of surfaces. In simple cases, the fillet surfaces produced are cylinders, tori, and spheres. For more complex cases, some systems used specialized surface types, and some systems just approximate the fillet surfaces with NURBs. Here are two example models – one from an automotive body panel, and one from a casting





The filleting problem is important because it consumes a great deal of modeling time – typically as much as 40% in parts like castings, forgings, and sheet metal stampings. The fillets are often omitted in the early stages of design, but they have to be present in order to do detailed structural analysis, and they are obviously needed in the manufacturing of tooling (moulds, dies, and casting patterns). Sadly, in some companies, the workload is doubled because the filleting is done twice – once in the stress analysis department and once in the tooling department. But, even when it's only done once, it is a bottleneck that consumes considerable time in the product development process.

To make matters worse, the filleting functions in CAD systems are often unpredictable, counter-intuitive, and prone to failure. So, producing the desired results often requires considerable user skill, which means that the task can not be done effectively by low-priced inexperienced workers. Two of the oddities are illustrated in the following pictures.

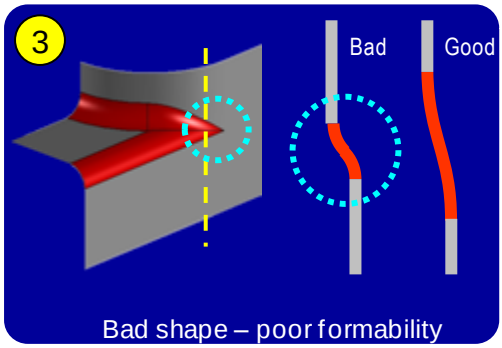


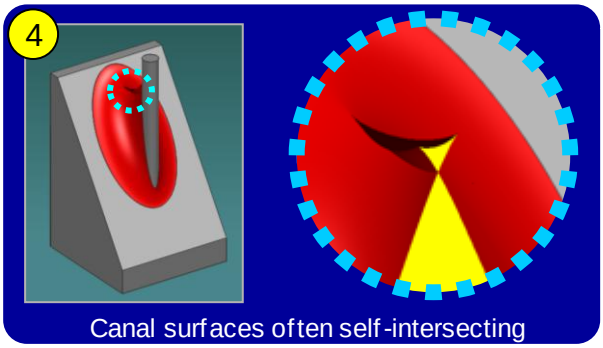
.

As picture #1 shows, the contact curves (where the rolling sphere touches the underlying surfaces) often have puzzling corners. In fact, if the underlying surfaces are not G2, then the contact curves will not be G1. The corners are often surprising for users, and they then waste time trying to figure out what they did wrong. The algorithm is producing an answer that is "correct" (assuming we use the rolling sphere approach) but is often undesirable.

Picture #2 shows that dramatically different results can be obtained by filleting edges in different orders. So, again, some skill and experience is required to get the desired results. The situation shown in picture #3 is even worse. Again, different results are obtained by filleting the edges in two different orders, but they might both be

undesirable. If this is a sheet metal part, the rapid reversals of curvature produced by rolling-ball fillets would lead to formability problems.





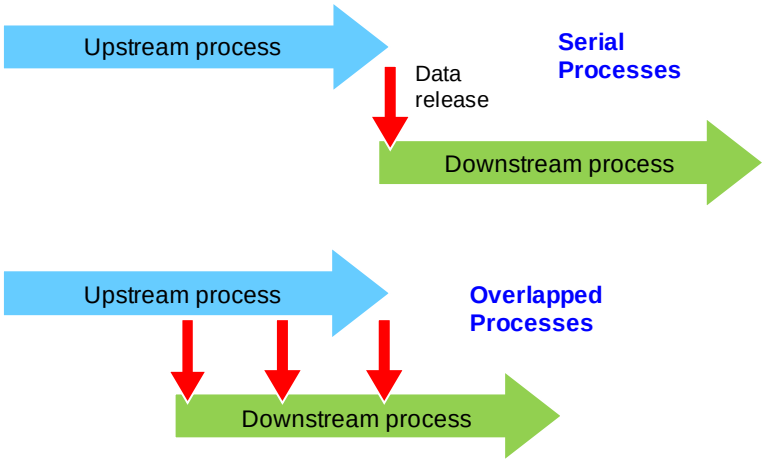
Finally, a more fundamental problem is shown in picture #4. The fillet surfaces formed by sweeping a sphere often have unexpected self-intersections. If these self-intersections are removed in a naïve way, then the surface is left with an unacceptable crease. So, in situations like this, the rolling-sphere paradigm simply does not work.

It may well be that we are over-using the rolling sphere idea. It has to be used for simple fillets, because it is what users expect, and it is the shape that would be produced by the obvious NC milling process. But in more complex situations, the results are unpredictable and often undesirable, so perhaps some other approach is needed.

Problem #2: History-Based Models

History-based modeling is important because, when it works well, it delivers huge productivity benefits. Smart companies build “template” models that are designed to be morphed and re-used. When they start a new project, they simply find a suitable template model, adjust its inputs, and replay it. This is not likely to produce a finished design, but it often provides a very good starting point for subsequent work, which significantly reduces schedules. In addition, the template model records an approved modeling process, so it is an effective way to capture the company’s valuable design process knowledge.

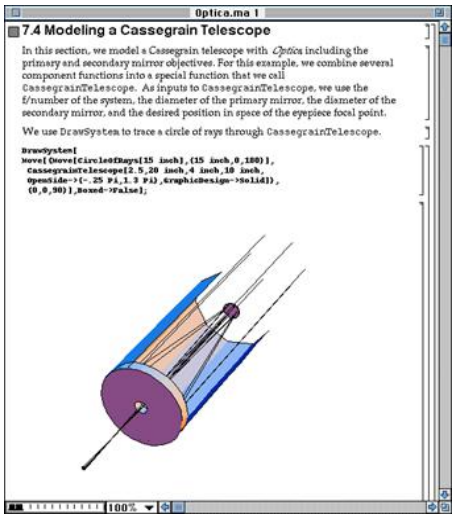
Another benefit is the increased overlapping of tasks. Since the replay process makes it easy to adapt to modified inputs, downstream tasks can be started earlier, using preliminary data releases, before upstream tasks are complete. The overlapping obviously compresses schedules.

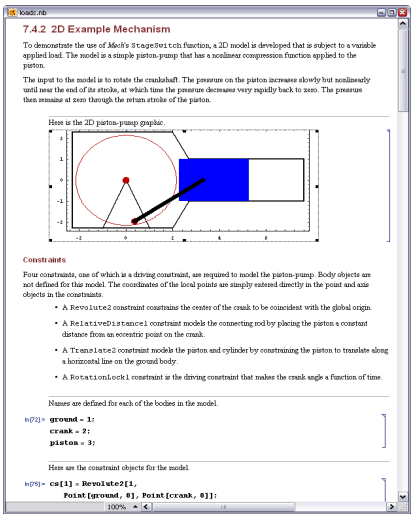


But, on the other hand, there are also several problems. Firstly, the replay process sometimes fails, especially if the new inputs are significantly different from the previous ones. When this happens, the user must “debug” the model. He has to understand the sequence of steps that was used to build it, and find out which of these steps are failing, and why.

The process is very similar to the debugging of programs -- in fact, in a sense, a history-based model **is** a program. But, unfortunately, the debugging tools are very primitive compared to those available for debugging programs. As a result, people often just give up and rebuild the model from scratch.

One possible approach is to apply the concepts of "literate programming" to the programs embodied in history-based models. The basic idea is to reverse the role of code and comments -- the program is regarded as a document to be read by people, but with executable code embedded in it, rather than the other way around. Today, an engineering design processes is often described in a "design handbook" – the handbook outlines the "recipe" that the engineer should follow to develop the design. Conceivably, these handbooks could be made executable (by embedding code within them) without reducing their readability or clarity. Mathematica notebooks, as shown below, provide one example of this sort of approach.





To some extent, understanding history-based models is not a geometry problem -- it's a computer science or user-interface problem. But geometric modeling improvements could certainly help. Specifically, if we could develop modeling operations that are more general, and less sensitive to changes in their inputs, failures would be much less common. In the old days of the 1980s, the only modeling operations were Boolean set operations, and these worked fine because their outputs are always well-defined, regardless of their inputs. We need some new modeling functions with this same sort of universality.

Another issue is the performance of the replay process, but this is the topic of Problem #3 below.

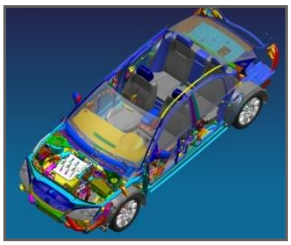
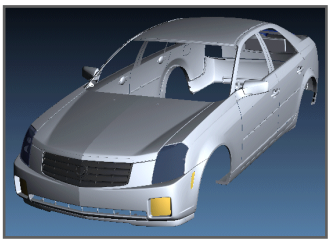
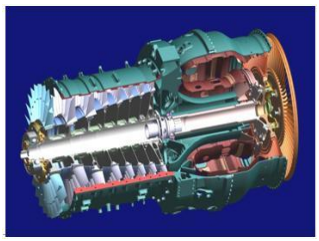
Given the problems with history-based models, there is a strong industry trend towards modeling approaches that do not rely on history. Examples are the Synchronous Technology functions from Siemens, Inventor Fusion from Autodesk, and the new SpaceClaim system.

Problem #3: Performance

There are two quite distinct types of performance problems in modern systems. The first and most obvious one is that some computations take minutes or even hours to complete, and this delays work and reduces user productivity. One of the most troublesome operations is replay of a large history-based model. A typical modeling operation might take only around a second. But a large model might well have a thousand or more operations, and 1000 seconds is around 17 minutes. Waiting 17 minutes for a model to update every time you make a change is obviously unacceptable. Other time-consuming operations are generation and verification of complex NC toolpaths, and generation of finite-element models.

Part of the problem, of course, is the fact that our users are dealing with enormous models. A motor vehicle, for example, will typically have around 30,000 parts, and overall data size is likely to be around 15 or 20 gigabytes. The largest parts are complex castings like the engine block and complex sheet metal parts like the floor pan. Perhaps surprisingly, the highly sculptured outer surfaces of an automobile (as shown in the middle picture below) typically occupy very little space, and are relatively easy to handle.

In other industries, also, models that are dozens of gigabytes in size are not unusual, and sometimes users want to work with the whole model all at once. The bunny and Buddha models that are often used as examples in the research community are hopelessly unrealistic and could easily lead to erroneous research directions and conclusions.


Another less obvious problem is that some operations take a few seconds, but users really need the computations to be done in real time (in other words, in around 1/30th of a second). Lack of real-time response makes some exploratory functions unusable, and this impairs user creativity.

In either case, we need performance that is roughly 100x better than we have today, so clearly small incremental refinements of our current approaches will not be sufficient.

Some promising opportunities are arising through fundamental changes in computer architectures. Only a few years ago, the computational power of a typical computer came solely from a single CPU. Today, the typical computer has several CPUs plus an enormously powerful graphics card (or "GPU") that can also be used for computational tasks. Our challenge is to find ways to leverage all this power. Unfortunately, this is not easy. The algorithms and data structures we are using were generally created without parallelism in mind, and may need to be completely redesigned. Geometric modeling algorithms, in particular, often include vast amounts of branching logic and special case code, which makes them unsuitable for GPU implementations.

It may be that we need some approach that is radically different from current ones. Perhaps some simpler and more universal geometry type would be appropriate, instead of the large collection of different types shown earlier. The obvious approach is to abandon curved surfaces altogether and use polygons for everything. This might seem like a step backwards, but, in fact, a great many of the algorithms in contemporary systems are already based on polygonal (facetted) models. Even NC milling of sculptured surfaces is essentially a facet-base process in most systems. Perhaps some simple low-degree patch would be a reasonable compromise between the austerity of a purely polygonal world and the combinatorial complexity of our current one. A change like this would require us to rewrite millions of lines of code, and it would significantly impact our ability to exchange data with other systems (unless they made the same change), so it's not likely to happen anytime soon.

Summary of Problems

The following table summarizes the three problems discussed, and their relationships to the overall business goals of cheaper, sooner, and better.

Problem	Cheaper	Sooner	Better
Filleting	Allows use of less skilled (lower-cost) people	Reduces time spent on a critical-path activity	
History-based modeling	Allows use of less skilled (lower-cost) people	Overlapping of tasks, and reduction of manual work	Template models capture and promote best practices for design tasks
Performance		General improvements in user productivity	Real-time response promotes exploration, creativity, innovation.

Improvements in the reliability and understandability of history-based models would probably be the most valuable of the three. But, as noted earlier, this is not solely a geometry problem.

Comparison with Current Research Topics

It is interesting to examine the trends of current CAGD research, to see whether or not they are aligned with problems in commercial CAD/CAM/CAE. As a small step in this direction, I took the papers from a recent CAGD conference, and divided them into the categories shown below. An analysis based on a single conference cannot provide any definitive conclusions, of course, but it's better than nothing.

Research Area	% of Papers	Impact on Commercial CAD/CAM/CAE
Calculations (intersections, root finding, etc.)	20%	Heavily used, but current techniques work OK
New geometry (subdiv surfs, implicit surfs, etc)	20%	Irrelevant, since we don't use these types of geometry
Meshing	18%	Important, but not my area of expertise
Approximation	17%	Heavily used, but current techniques work OK
Point clouds processing (reverse engineering)	11%	A fairly small niche. Little impact.
Applications (pumps, composites, sound, etc.)	11%	Interesting, but narrow impact
Hardware leverage (especially GPUs)	5%	Very important

At this particular conference, three of the most popular topics were geometric calculations, (including surface-surface intersection, root finding, and surface trimming), approximation, and meshing. Together, these three topic areas accounted for almost 60% of the papers. Obviously these sorts of algorithms are very heavily used in commercial CAD/CAM/CAE systems. However, our current functions have been refined through many years of customer usage and incremental improvement, and they now work fairly reliably. Meshing is especially interesting, because of its relationship to finite-element methods, which are an increasingly important means of improving the performance and safety of product designs. Meshing algorithms are also used to perform tessellation (polygonalisation), which is the first step in many of our applications, including graphical rendering, clearance/interference analysis, and even NC toolpath generation.

Another very popular topic was the study of particular classes of geometry, such as subdivision surfaces, algebraic surfaces, and implicit surfaces. Subdivision surfaces are used in some animation and game design systems, but, as noted earlier, they are not used in any of the mainstream CAD/CAM/CAE systems. Also, we only use implicit equations for a few calculations involving quadric surfaces. The cost of incorporating fundamentally new surface types into our systems is enormous, and difficult to justify, so these research areas are of little interest to us.

The area of point-cloud processing includes computations, registration, and surface fitting for reverse engineering. Interest in this area is growing as scanner technology improves and their prices drop. But, as of right now, point-cloud processing is not a significant activity, except in a few niche areas. Also, the trend is to use point cloud (polygonal) models directly in applications, rather than fitting them with surfaces.

The topic of parallel and GPU-based computing is extremely important, in my view. Computer architectures are changing in fundamental ways, and it is important for us to change our geometric modeling methods to match. The hardware is improving at a very rapid pace, and it can give us significant performance gains, if we can invent ways to use it effectively.

Conclusions

We have tried to explain how improvements in geometric modeling technology software have connections to business benefits in the manufacturing industries. When we launch a software project ourselves, we use these connections and benefits to assess the value of the project, so this is how we judge the relevance of research, too. Using these ideas, we analyzed three sample problems, plus some research papers, to assess their relevance.

We have not discussed the issue of whether research **should** be relevant. One could certainly argue that the purpose of research is to gather and codify knowledge, and it does not need to be applicable to any industrial activity, including the CAD/CAM/CAE software business, manufacturing, or anything else.

But my sense is that many people in the CAGD field want to do research that is applicable to industrial problems, either directly or indirectly, so they need some way of assessing the relevance and applicability of research projects. Our suggestion is that the connection to end-user business benefits is the right way to make this assessment, and our hope is that the discussions in this paper might make it easier to trace these connections. So, then, the next time a researcher writes “this is an important problem in CAD/CAM/CAE”, there will be a better foundation for this statement.