A New Era for Mechanical CAD

TIME TO MOVE FORWARD FROM DECADES-OLD DESIGN

JESSIE FRAZELLE

Computer-aided design (CAD) has been around since the 1950s. The first graphical CAD program, called Sketchpad, came out of MIT [designworldonline.com]. Since then, CAD has become essential to designing and manufacturing hardware products. Today, there are multiple types of CAD. This column focuses on mechanical CAD, used for mechanical engineering.

Digging into the history of computer graphics reveals some interesting connections between the most ambitious and notorious engineers. Ivan Sutherland, who won the Turing Award for Sketchpad in 1988, had Edwin Catmull as a student. Catmull and Pat Hanrahan won the Turing award for their contributions to computer graphics in 2019. This included their work at Pixar building RenderMan [pixar.com], which was licensed to other filmmakers. This led to innovations in hardware, software, and GPUs. Without these innovators, there would be no mechanical CAD, nor would animated films be as sophisticated as they are today. There wouldn't even be GPUs.

Modeling geometries has evolved greatly over time. Solids were first modeled as wireframes by representing the object by its edges, line curves, and vertices. This evolved into surface representation using faces, surfaces, edges, and vertices. Surface representation is valuable in robot path planning as well. Wireframe and surface

representation contains only geometrical data. Today, modeling includes topological information to describe how the object is bounded and connected, and to describe its neighborhood. (A *neighborhood* of a point consists of a set of points containing that point where one can move some distance in any direction away from that point without leaving the set.)

OpenCascade, Parasolid, and ACIS are all boundary-representation (B-rep) kernels. A B-rep model is composed of geometry and topology information. The topology information differs depending on the program used. B-rep file formats include STEP (Standard for the Exchange of Product Model Data), IGES (Initial Graphics Exchange Specification), NX's prt, Solid Edge's par and asm, Creo's prt and asm, SolidWorks' sldprt and sldasm, Inventor's ipt and iam, and AutoCAD's dwg.

Visual representation (vis-rep) models tend to be much smaller in data size than B-rep models. This is because they do not contain as much structural or product management information. Vis-rep models are approximations of geometry and are composed of a mass of flat polygons. Vis-rep file formats include obj, STL, 3D XML, 3D PDF, COLLADA, and PLY.

CAD programs tend to use B-rep models, while animation, game development, augmented reality, and virtual reality tend to use vis-rep models. However, the two are interchanged frequently. For example, if you were using a B-rep model for manufacturing but wanted to load it into Apple's ARKit for some animations, you would first convert it to COLLADA, a vis-rep file format. The file should already be a bit smaller from dropping all the CAD data, but if you

wanted to make it even smaller, you could tweak the polygon counts on each of the meshes for the various parts.

The tools used to build with today are supported on the shoulders of giants, but a lot could be done to make them even better. At some point, mechanical CAD lost some of its roots of innovation. Let's dive into a few of the problems with the CAD programs that exist today and see how to make them better.

I am sure there is a lot to learn from these codebases, but as someone who has seen many old codebases, I know this can lead down a dangerous path.

SINGLE THREADED

Since most CAD kernels are built on cores from the '80s, they are not meant for modern systems. Even the latest CPU or GPU won't do much to help the performance since most of these programs are single threaded, or have single-threaded aspects, and have no awareness of a GPU. OpenSCAD and everything built on CGAL (Computational Geometry Algorithms Library) are single threaded. Sure, some of these kernels have been updated since the '80s, but their roots are still tied to their predecessors. (I am sure there is a lot to learn from these codebases, but as someone who has seen many old codebases, I know this can lead down a dangerous path.)

This does not mean that all CAD kernels are *entirely* single threaded. Parasolid is multithreaded, but that still means if you are importing or exporting to a file format other than Parasolid, you might have just switched back to a single-threaded process. Another example of a multithreaded kernel is ImplicitCAD [implicitcad.org], which is written in Haskell.

One problem with making a whole CAD program multithreaded is the different file formats. For example,

a STEP file, whose format dates back to the '80s [iso.org], pretty much mandates the need for a single-threaded process. (Additionally, a STEP file cannot be read sequentially; it must be loaded into memory and then resolved.) Most parametric CAD operations are single threaded; however, the open-source project SolveSpace [solvespace.com], which uses NURBS (nonuniform rational basis splines), has some parallel operations.

DUPLICATION

In software development, a pointer is used to get the contents of a memory address. This allows users to reference that same content over and over again without the expense of copying the content itself.

Some products built using CAD may never duplicate a part of their model—lucky for them! For people who do have multiple similar parts in their CAD designs, most CAD programs are creating very expensive copies of these parts.

For example, imagine a model of a server rack. The *default* method of copying a part (using copy and paste) in SolidWorks, as well as many other industrywide CAD programs, is to copy the entire contents of a child model to a new model. So, if you have 32 sleds in the server rack and use the default copy method in SolidWorks, you have 32 of the exact same model in individual copies. This is very expensive. Each sled has many more models inside, and then those models have child models as well. This exponentially increases the workload on the kernel and on your program to load your model in the first place, since the program does not know these are all the same thing.

Taking a lesson from software development, what you

really want is a form of pointer to the model. In the CAD world, these are called *instances*. Then you can have one copy of the model stored, and all the other instances are actually just references to the original copy. This also saves the user a bunch of time. Imagine having to update parts of models in 32 different locations when a part in a sled changes. A wise person once said, "The definition of insanity is doing the same thing over and over again but expecting different results."

SolidWorks does offer another option that is more in line with how pointers operate, but since this is not the default, most users might not even realize there is a better way. The default path should lead to the least amount of pain. Instead of having two methods for copying, products should have just one. They should make the default method act more like a pointer (or instance) *until* the geometry, surfaces, or topology of the copy (not the main) has changed. In this case the user should be warned that this will now act like a unique part aside from the main copy. Or the user might have mistakenly meant to apply those changes to all the copies, in which case the changes should be applied to the main copy.

There is another huge problem with this. Each CAD program has its own way of implementing and referencing instances. If you export your design from one CAD program to another, you will likely still have 32 individual sleds rather than one sled and 31 references to the original with only the xyz coordinates changed. Some programs offer ways to import instances, but they all rely on the file format being imported and whether they have the support for that format.

Even if you are using instances, you are still at the mercy of the single-threaded kernel, and none of the copies are likely to render in parallel.

VERSION CONTROL

For software teams that are accustomed to using git, being able to diff, fix merge conflicts, and work as a team in parallel on the same file is a huge time saver. A number of startups are working to bring this ability to mechanical CAD.

Instead of reinventing version control for CAD, those who use git today want to continue to use git and not have to add another tool to their workflow. Today there is no way to push a CAD file to a git repo, have several people modify the file, and resolve merge conflicts. (Well, maybe it could be done, but it would be the opposite of fun.) For all the startups working to solve version control for mechanical CAD, this is why they had to reinvent the wheel.

Instead of reinventing version control for CAD, those who use git today want to continue to use git and not have to add another tool to their workflow.

In a world where a kernel can fully utilize a modern CPU and GPU, can you not also use a file format that is human-readable and would allow for resolving merge conflicts? When you ask, "What is human-readable and works well with git?" the first answer that comes to mind is a programming language.

The other great win from using a programming language is this: Even if you don't use or want to use git, there are already many different options for version control of human-readable files. Additionally, integrations with GitHub and other version-control tools could be extended with wasm (WebAssembly) support so that diffs could be visualized as renders as well.

PROGRAMMABLE

Think back to the example of the rack of servers. If part of the rack contained complex math that you were calculating in a program such as Mathematica, you would have to reevaluate the math continuously in another program and update it in your model. If, instead, you could program in the CAD product itself, then you could do all your calculations in one place and the model would update if anything in the equations changed.

Each sled in the rack of servers has network cables that connect to the back of the sled. Using the GUI, you would have difficulty making these align perfectly with the connector on the sled. Someone would have to sit with the model for an hour or so just tweaking each cable to be perfectly aligned—a huge waste of time. Instead, if you could program the alignment of the cables, you could ensure each was perfectly aligned with the connector.

The need for programming becomes even more acute if you want to do mesh or topology optimizations. Unfortunately, most optimizations are implemented through GUI click interfaces, and given their complexity to define, can often be more trouble than they are worth. Today, some programs allow for scripting, but their APIs are COM (Component Object Model) based and, as you can imagine, built in the '90s. It's great they even offer this, however. (Thank you, AutoCAD, for being the first CAD command-line interface I ever used.)

For the modern world, it would be great to generate SDK clients for the CAD program in every language, much the way API clients are generated. This would allow anyone to program in any language. It would lower the barrier to

entry since learning a new language would not be required. This would allow for complex math to be done in the CAD program itself rather than using Mathematica, MATLAB, or Wolfram Alpha.

A few scriptable CAD programs exist today and are paving the way for this transition: ImplicitCAD [implicitcad.org], libfive Studio [libfive.com], OpenSCAD [openscad.org], CadQuery [github.com], FreeCAD [freecadweb.org], and ruckus [github.com]. Blender [blender.org] has a great console interface. Three.js [threejs.org], while not CAD oriented, is also another great example of a 3D programming language. Jonathan Blow's Jai [oxide.computer] is for writing systems-level code and a great example of creating a language thinking heavily about performance. (This is not yet open to the public, but he has talked about it extensively.)

Most of the mechanical engineering community is tied to the GUI, so generating code from GUI interactions would be necessary. This is quite similar to an HTML point-and-click GUI that generates code on the back end. This allows people who want to script to script, and others who want to click can click. Both worlds can be happy—code on the left side, render on the right, just like a markdown editor.

If there is an SDK client for the CAD program and underlying kernel, you can imagine a rich ecosystem of plug-ins and tools emerging, much like the ecosystem that surrounds VSCode, Vim, and Emacs. Most CAD editors used for products are closed off and don't allow for this type of community-based development and sharing. Plug-ins could be written for any use case: for example,

mesh/topology optimizations and supply-chain system integrations. This includes the functionality for finding parts, creating BOMs (bills of materials), and computing lead times for parts of the model. Today, this is usually done in separate programs or even spreadsheets.

Plug-ins that support a command+P function would be welcome. In most programs, when you want to print something, you hit command+P. (Creo probably has the closest thing to this functionality but lacks an open ecosystem.) For mechanical CAD, when you want to print your model the underlying program should discover all the 3D printers and machines on the local network (or plugged directly into your machine) and send the parts of your model that are compatible with each machine to be printed. This could even be taken a step further—in a fully automated factory with robots, the program should set up and start the assembly for the model and all the parts.

Speaking of 3D printing, let's look at the STL file format. This format was defined in 1987, and its namesake comes from stereolithography, the first method of additive manufacturing. STL files represent geometry in a series of triangular surfaces. Since STL is a vis-rep format, it does not hold any data about internal structure, color, texture, or any other CAD data that a B-rep format would contain. Modern 3D printers have innovated past the simplicity of the STL format. For example, to print a full-color model, users need a VRML (Virtual Reality Modeling Language) file, or an STL file associated with textures in order for the printer to add color and texture to the object. Plug-ins can ensure that the printer gets the correct data for the specific model to be printed,

without the pain of conversion and ensuring that no materials or textures are dropped.

TESTING

The test flow of CAD models usually consists of running simulations. Let's use airflow and thermals as examples.

In the software world, after pushing your code updates, typically a CI (continuous integration) is run on the changes, letting you and your teammates know if you broke anything or if your code is safe to merge. CAD should work the same way. If you make changes to a model, your simulations should run in a CI to let your teammates know if your code is safe to merge. Most of these simulations are compute intensive, so being able to offload the simulations to the cloud or remote servers would also be ideal.

Much as VSCode and other editors have nice plug-ins for offloading tests to other computers, a modern CAD program should have the same.

USER EXPERIENCE AND DESIGN

After trying many different industry CAD programs, I have found that most have one characteristic in common: a user interface that looks like it is from the '90s. It is a bit ironic that a tool used for mechanical design has not considered the design and experience of its user interface. Most CAD programs are in need of a makeover, though there are a couple of outliers that do interface design well: Shapr3D [shapr3d.com], an iPad app, has a great design and intuitive interface; SketchUp [sketchup.com] has an intuitive and beautiful design.

Additionally, CAD applications need to be native on

MacOS, Linux, and Windows. Native applications built for their specific platform perform better than ones built with Electron and the like. (That being said, VSCode is a nice Electron app.) Especially for a program as graphics heavy as CAD, using the underlying operating-system graphics mechanisms helps achieve the best performance possible. Today, a CAD program can be used only on the operating system that is supported by that specific program. Additionally, most use archaic GUI frameworks that truly show their age.

Onshape [onshape.com] changed the mold by offering a SaaS (software-as-a-service) CAD program. This allows expensive compute processes to be easily offloaded to the cloud. This was a truly revolutionary idea, but it limits the user's ability to work offline. In contrast, native apps can work offline but also have the ability to offload workloads to the cloud when connected to the network.

If CAD programs can focus on an intuitive design without falling into a trap of complexity, both new users and professionals should be productive. Just as I would use Vim for side projects as well as professional jobs, I would expect my CAD tool to work just as well for building a toy for fun as it would for a complex project. A lot of this capability comes down to the interface design and extensibility through plug-ins.

A BETTER TOMORROW

Developers of new CAD programs need to think through each of these aspects. No existing CAD program has solved all of these problems.

The world owes so much of the amazing innovation

of computer graphics to brilliant people such as Ivan Sutherland, Pat Hanrahan, Ed Catmull, John Carmack, and many others. I can only hope some truly revolutionary changes are headed to the world of computer-aided design in the same way that computer graphics pioneers paved the way for rendering, animations, and virtual reality.

The hardware industry is desperate for a modern way to do mechanical design. A new CAD program created for the modern world would lower the barrier to building hardware, decrease the time of development, and usher in a new era of building.

Jessie Frazelle *is the cofounder and chief product officer of the Oxide Computer Company. Before that, she worked on various parts of Linux, including containers, as well as the Go programming language.*

CONTENTS