

# nTopology Modeling Technology

George Allen, nTop Fellow

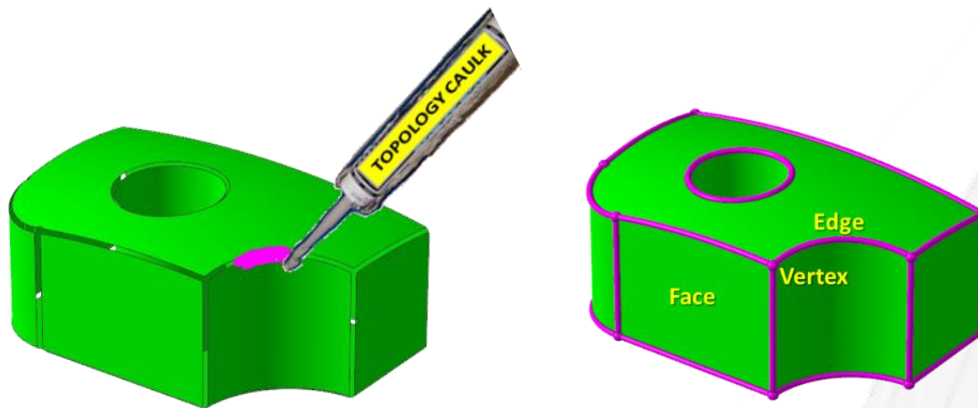


# 1 Introduction

This document describes the modeling technology used in the nTopology software (or just “nTop”, for short), and explains how this differs from the approach used in current CAD systems. As we will see, nTop uses a completely different approach to solid modeling, which delivers large and sustainable advantages in reliability, speed, and scalability.

## 2 The Current Generation: B-Reps

Current CAD systems (NX, Catia, Creo, SolidWorks, and others) all use **boundary representations** (b-reps) to express the shapes of solid objects. As the name implies, a boundary representation is a collection of faces that form the boundary (the outer skin) of the object. The faces are glued together by “topology” information that describes connectivity, such as which edges lie on each face, which faces meet at each vertex, and so on, as shown here:



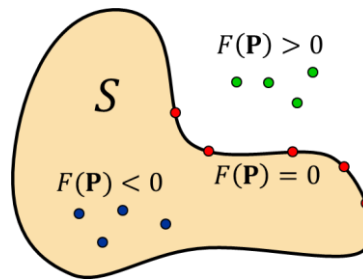
In CAD systems, the surfaces are usually curved; they might be cylinders, or cones, or NURBS surfaces, for example. In computer graphics applications, on the other hand, people often use boundary representations in which all faces are planar. These are much simpler, of course, but the principle is the same — we are again representing the object by its external skin.

It’s natural to ask whether we also need some information about the interior of the object, not just its skin. Current systems assume that the interior of the object is homogeneous, in which case the boundary alone does provide us with enough information to fully describe the object. However, given the capabilities of modern 3D printers, the assumption of homogeneity is no longer valid (if it ever was), but that’s a topic for another day.

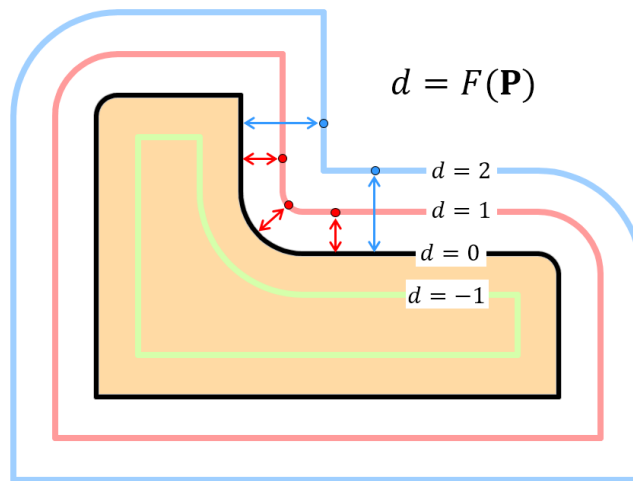
Boundary representations first appeared in a system called BUILD, which was developed by [Ian Braid and his colleagues](#) at the University of Cambridge in the mid-1970s. Later, people from this group played a leading role in the development of the Romulus, Parasolid, and ACIS modelers, so the ideas from BUILD have had a major influence on current technology — many of the ideas used today are very similar to the ones originally developed in BUILD. Some new surface types have been added, and topological structures have been generalized, but the fundamentals of b-rep modeling today are largely the same as they were 40 years ago. Computers and manufacturing have changed enormously in the past four decades, of course. At nTopology, we believe it’s time for the basic modeling technology to change, too, to keep pace.

### 3 The nTopology Approach: Implicit Modeling

The nTopology system does not use boundary representations (or, not much, anyway). Instead, we use a technology called **implicit modeling**, which simply means modeling based on **implicit functions**. An implicit function (i-function, for short) is a mathematical function that returns a value at each point in 3D space. The key property is that an i-function of a solid object is negative at points that are inside the object, and positive outside. Or, in mathematical terms, if  $F$  is an i-function of a solid object  $S$ , then  $F(\mathbf{P}) < 0$  when the point  $\mathbf{P}$  is inside  $S$ , and  $F(\mathbf{P}) > 0$  when  $\mathbf{P}$  is outside  $S$ . And, of course  $F(\mathbf{P}) = 0$  when  $\mathbf{P}$  is on the boundary of  $S$ .



In addition to this positive/negative property, the value  $F(\mathbf{P})$  often gives us some additional information: it provides some measure of the distance from the point  $\mathbf{P}$  to the boundary of  $S$ . So, the sign of  $F(\mathbf{P})$  tells us whether we are inside or outside the object, and its magnitude tells how far outside (or inside) we are. An i-function that measures distance in this way is called a **signed distance function**. To understand the behavior of an i-function, it is often useful to draw the curves where  $F(\mathbf{P})$  is constant, like the blue, pink, and green curves in the picture below:



For further visualizations of i-functions, please see [this blog post](#), or the last part of [this video](#).

With implicit modeling, there are no topological entities like edges and vertices. Or, we might say that objects can have any topology at all, and it can change however you wish as your design evolves. The phrases “no topology” and “any topology” became **nTopology** when we were choosing a name for our company.

We at nTopology did not invent the implicit modeling approach; it has been used in the film, animation, and gaming industries since the 1980s. But we believe we are the first company to apply the technology to engineering and manufacturing on a broad scale.

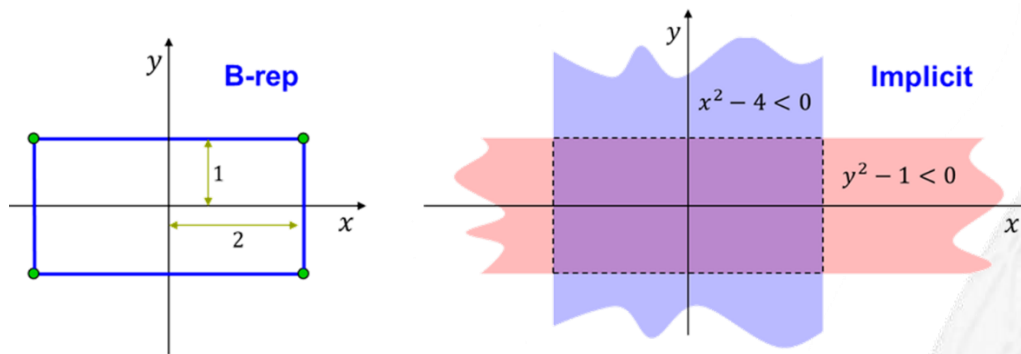
### 3.1 Some More Terminology

In this area of technology, people use an assortment of different terminology. Implicit modeling is sometimes called functional modeling. The object representations are then called functional representations, or f-reps, to contrast with b-reps.

Also, some people use the term **signed distance field**, rather than signed distance function. These two terms mean exactly the same thing, and luckily the acronym SDF covers both of them.

### 3.2 A 2D Example

A simple 2D example might serve to make the implicit modeling idea more clear. Suppose we wanted to store some description of a 2D rectangular region. The b-rep approach would be to store the outside boundary of the rectangle, which consists of four lines. The implicit modeling approach is to store an i-function that is negative inside the rectangular region, and positive outside. To be specific, let's consider the rectangular region whose opposite corners are at the points  $(-2, -1)$  and  $(2, 1)$ .



In a b-rep, we would store the four blue lines shown in the left-hand image above. The implicit approach is shown in the right-hand image. We see that a point lies in our rectangle if it lies in both the pink region and the blue region. A point  $(x, y)$  lies in the pink region if  $y^2 - 1 < 0$ , and it lies in the blue region if  $x^2 - 4 < 0$ . So it lies in the rectangle if **both**  $y^2 - 1$  and  $x^2 - 4$  are less than zero, which means that  $\max\{x^2 - 4, y^2 - 1\}$  is less than zero. So, one possible i-function for the rectangle is the function  $F(x, y) = \max\{x^2 - 4, y^2 - 1\}$ . To convince yourself of this, you can try checking the value of  $F$  at a few points; you will find that  $F$  gives you negative values inside the rectangle, and positive values outside. Notice that i-functions are not unique: the blue region above could be represented by any of the i-functions  $x^2 - 4$ ,  $|x| - 2$ ,  $x^4 - 16$ , or numerous others. Actually, of the three listed, the function  $|x| - 2$  is the best choice, since it accurately measures signed distance for the blue region.

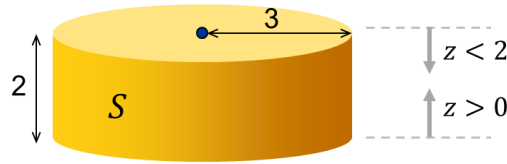
Note that the b-rep has no explicit representation of the interior of the rectangle, and the implicit approach involves no explicit representation of the four edges. The two representation schemes are completely different.

### 3.3 Some 3D Examples

A simple 3D example is a spherical solid of radius 3 centered at the origin. One possible i-function is given by  $F(x, y, z) = x^2 + y^2 + z^2 - 9$ . Then  $F(x, y, z) < 0$  means that  $\sqrt{x^2 + y^2 + z^2} < 3$ , and this happens precisely when the point  $(x, y, z)$  lies inside the sphere. Actually, the i-function  $F(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 3$  would work a little better, since it most accurately measures true signed distance.

Similarly, the sphere of radius 5 centered at the point (2,7,1) can be represented by the i-function  $F(x, y, z) = (x - 2)^2 + (y - 7)^2 + (z - 1)^2 - 25$ .

As a final example, consider a cylinder with height 2 and radius 3, with its base at the origin.



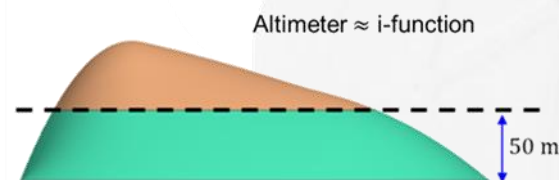
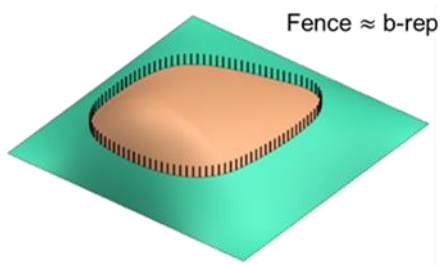
The point  $(x, y, z)$  will lie inside this cylinder if  $z > 0$ ,  $z < 2$ , and  $x^2 + y^2 < 9$ , so one possible i-function is  $F(x, y, z) = \max\{x^2 + y^2 - 9, -z, z - 2\}$ . Again, you can convince yourself that this is correct by checking a few points.

## 4 Fences and Altimeters

The description in the previous section was a little mathematical, so here's a less technical way to see that b-reps and implicit modeling are completely different. Suppose you discovered a new island, and you wanted to claim the high ground as your personal property. How will you indicate which portion of the island is yours, and how can you tell whether you're standing on your land or not?

The old-fashioned way is to build a fence around your property. This is troublesome because you have to buy fence-posts, dig holes, and so on. Also, with this approach, deciding whether or not a given point is on your land is quite complicated. What you have to do is walk in some fixed direction (say due west), and count how many times you cross your fence. If the number of crossings is odd, then the original point is on your property. But lots of things can go wrong. If there's a gap in the fence, you'll make big mistakes. Or, what if your walking path just grazes the fence-line? Does that count as one crossing, or two, or zero? The issues are discussed in more detail in section 5.1 below, but the key point is that the decision process is fundamentally fragile and prone to error.

There's a better way: you could just declare that you own all the land that's more than 50 meters above sea level. You don't need a fence, so no buying posts and no digging holes. Then, to determine whether or not a given point lies on your property, all you need to do is stand at the point and look at an altimeter. No walking, and no counting. If you're near the boundary, and your altimeter is not very accurate, you might make small mistakes, but nothing as disastrous as the ones caused by gaps in a fence. The altimeter is a lot less expensive, and it delivers in/out decisions that are faster and more reliable than any fence.



B-reps are like fences, and i-functions are like altimeters. The implicit modeling approach that we use in nTopology isn't just a new kind of fence, it's something entirely different.

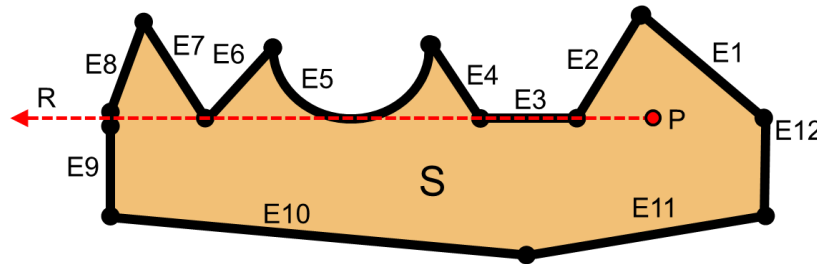


## 5 Example Operation: In/Out Testing

One of the most fundamental operations in any solid modeling system is deciding whether a given point is inside or outside a given solid body. In solid modeling jargon, this is often called point membership classification, or PMC, for short. We touched on the subject of PMC in the previous section, but let's take another look here. We'll use a 2D example, for simplicity; the situation in 3D is similar, conceptually, although the computations are much more difficult.

### 5.1 In/Out Testing in a B-Rep System

Let's consider the shape  $S$  below, bounded by 12 edge curves, as shown:



We have a point  $P$  and we want to decide whether it's inside or outside our shape  $S$ . To do this in a b-rep system, we shoot a "ray"  $R$  (an infinite line) from  $P$  in some fixed direction, and count how many times this ray intersects the boundary of  $S$ . If that number is odd, then  $P$  is inside  $S$ ; if even, then  $P$  is outside  $S$ . However, there are many things that can go wrong with this algorithm. Look at edge  $E3$ , for example: it lies on the ray  $R$ , so how many intersections should we count? Also,  $E5$  is tangent to the ray, so how many intersections does that imply? Similar problems arise where  $R$  crosses the junction of edges  $E6$  and  $E7$ . At the junction of edges  $E8$  and  $E9$ , there is a small gap, which the ray might pass through, if we're unlucky. In contemporary modelers, the computations will be done using floating-point arithmetic, which always introduces tiny round-off errors, so it's very difficult to get the counting correct in cases like the one shown above. Of course, this is a fabricated example, intended to illustrate the issues, but situations like this do arise in real models, and they lead to reliability problems.

To do in/out testing in 3D, we fire a ray and count its intersections with the faces of the object we're considering. Unfortunately, b-rep systems inevitably have small "cracks" between adjacent faces — people say that the models are not "water-tight". Unless great care is taken, the rays we fire can pass through these cracks, and in/out results will be wrong, leading to errors in higher-level applications.

In the worst case, the b-rep modeler is simply unreliable: we'll get the wrong in/out answer for a point like  $P$  above, even though it's nowhere near the boundary of  $S$ . In the best b-rep modelers, problems like these are handled (mostly) because they have been beaten into submission by decades of careful engineering. However, the code required is extremely complex and full of special-case branches, which makes it entirely unsuitable for GPU implementation.

### 5.2 In/Out Testing in an Implicit Modeling System

With the implicit modeling approach, we have an  $i$ -function  $F$  that is negative inside  $S$  and positive outside. This function would be constructed using the same basic ideas used in the examples in section 3.2. Many further examples are described [here](#) and [here](#).

Then, given the point  $P$ , we do the in/out testing merely by looking at the sign of  $F(P)$ : if  $F(P) < 0$ , then  $P$  is inside the shape  $S$ . Of course, we'll be using floating point arithmetic, again,

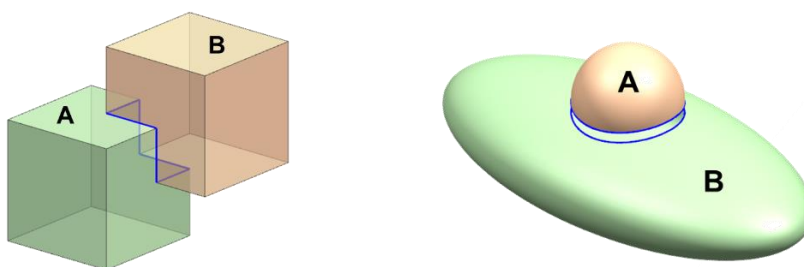
so numerical errors might lead to  $F(\mathbf{P})$  being positive even though  $\mathbf{P}$  is inside  $S$ . But this will happen only if  $\mathbf{P}$  is very close to the boundary of  $S$ ; we will never make wrong classifications of points that are far from the boundary, as we did with the b-rep-based ray shooting algorithm. Calculating  $F(\mathbf{P})$  might require a lot of computation, but it is simple, direct, and free from decision logic, which makes it suitable for GPU implementation. More on this topic in section 13.

## 6 Example Operation: Boolean Unite

Most solid modeling systems support boolean operations (unite, subtract, and intersect). Let's see how these operations can be implemented using b-reps and implicit modeling. We will use the unite operation as an example; subtract and intersect are very similar.

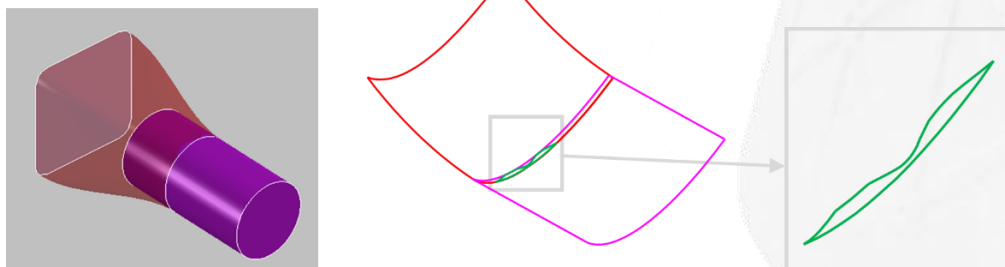
### 6.1 Boolean Unite in a B-Rep System

When using b-reps, the main step in a boolean unite (or in many modeling operations, actually) is computation of the new edges that are formed where existing faces intersect.



So, if we want to unite two cubes, A and B, as shown above, we have to compute the blue intersection curves. Of course, the operation of intersecting two planar faces is pretty easy, so the only problem is how often we have to do this. Each cube has 6 faces, so, in the worst case, we have to compute  $6 \times 6 = 36$  intersections. More generally, to unite an object with  $m$  faces and an object with  $n$  faces, we have to perform  $mn$  intersections. If  $m$  and  $n$  are large, then  $mn$  will be very large, and all these intersections might take a very long time. The example on the right shows a different problem — we only have to compute one face-face intersection, but it's a fairly difficult one that results in a complex 3D spline curve.

The computations get really nasty when faces are tangent, or nearly tangent.



Intersecting nearly-tangent surfaces is what mathematicians call an “ill-conditioned” problem. This just means that a tiny change in the input can lead to much larger changes in the output, so getting reliable answers is very difficult. As we noted above, b-rep systems have been under development for decades, so an enormous amount of effort has been expended trying to make

difficult boolean operations work reliably. Modern systems are pretty good, but they still make intersection errors from time to time. And unfortunately, the boolean algorithms are fragile: if one face-face intersection fails, then the entire boolean operation will fail.

For developers of b-rep systems, there is a sad irony here: the intersections that are most difficult to calculate are the ones that are least useful to the user. For example, the strangely-shaped green curve shown on the right in the image above is useless to the end user, but it has to be computed correctly, or else the entire b-rep will fall apart. Years of software development time and precious seconds of computing time are expended to produce a curve that has no real value in engineering or manufacturing.

## 6.2 Boolean Unite in an Implicit Modeling System

In an implicit modeling system like nTop, the boolean unite operation is entirely different. There are no edges, so we don't spend any time on the arduous task of trying to compute them.

Suppose we have two objects A and B in an implicit modeling system, and we want to unite them to form  $C = A \cup B$ . All we need to do is construct an i-function of C from the known i-functions of A and B. To make things more specific, let's suppose that A and B have i-functions  $F_A$  and  $F_B$  respectively. Consider the following sequence of equivalent statements. Don't worry about the  $\Leftrightarrow$  symbol; it's just mathematical shorthand for "if and only if" or "means the same thing as".

The point  $\mathbf{P}$  lies inside  $C = A \cup B$

$\Leftrightarrow \mathbf{P}$  lies in A or in B (or both)

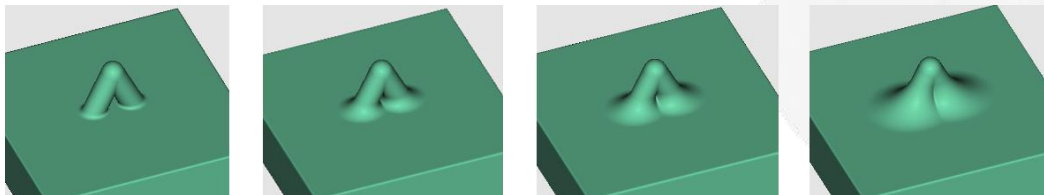
$\Leftrightarrow F_A(\mathbf{P}) < 0$  or  $F_B(\mathbf{P}) < 0$  (or both)

$\Leftrightarrow \min\{F_A(\mathbf{P}), F_B(\mathbf{P})\} < 0$

So, we see that the points  $\mathbf{P}$  that lie inside  $A \cup B$  are precisely those points where the function  $G(\mathbf{P}) = \min\{F_A(\mathbf{P}), F_B(\mathbf{P})\}$  is negative. But this is just another way of saying that  $G$  is a suitable i-function for  $A \cup B$ . So, in an implicit modeling system, doing a boolean unite is trivial: to get an i-function for the united body, we just take the minimum of the i-functions of the two original bodies. There are no significant computations to perform, so it's very fast and there's no chance of anything going wrong. So, boolean operations in nTop are 100% reliable; they **never** fail.

## 7 Blending/Filleting/Rounding

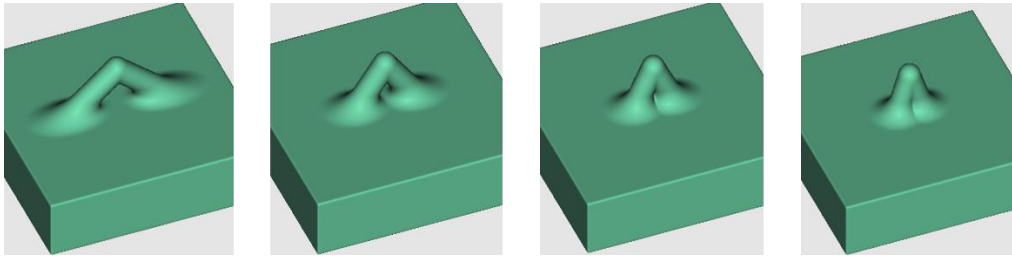
Blending, filleting, and rounding are modeling operations that replace sharp edges by smooth transitions. In b-rep systems, the edges of fillets need to be calculated, so problems tend to arise when fillets touch or overlap each other. In the example shown below, there are fillets where the cylindrical beams join the base plane, and we are gradually increasing the fillet radius as we move from left to right:



The left two fillets would work in most b-rep-based systems, but the two on the right would probably fail.

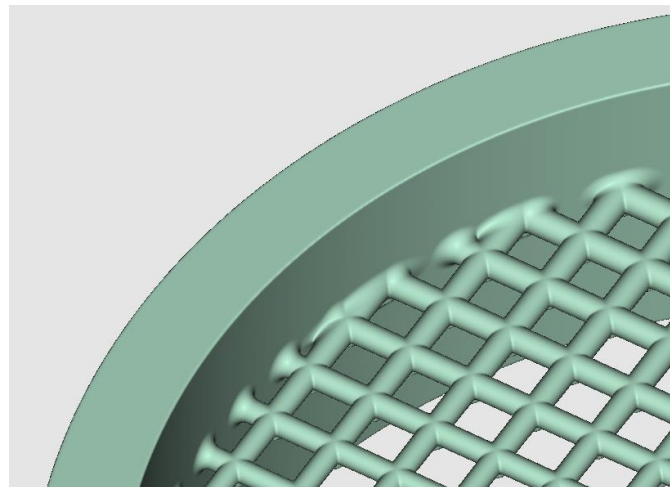


Similar problems would arise if we narrowed the angle between the two cylinders, since this would again cause the fillets to overlap each other:



The first two fillets would work in most b-rep systems, but the third and fourth ones would probably fail. In nTop, filleting is simple, because we can obtain an i-function for the filleted shape very easily from the i-function of the original one, as explained in [this video](#). There are no difficult computations to perform, so filleting in nTop is 100% reliable. In particular, all of the fillets shown above work in nTop (in fact, nTop was used to make the pictures).

Some more realistic filleting examples are discussed in [this blog post](#). One example is illustrated in the picture below, which shows the fillets between a lattice and an enclosing shell. As you can see, there are many overlapping fillets, which would probably cause failures in most b-rep-based systems:

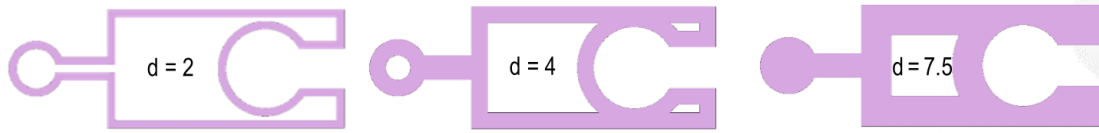


## 8 Offset and Shelling

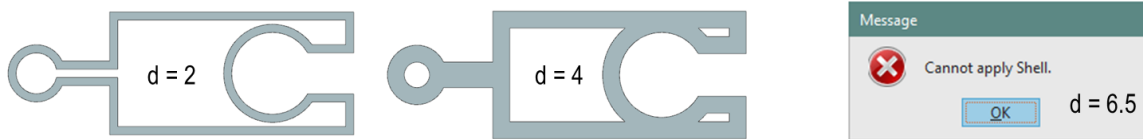
Offsetting and shelling cause problems in b-rep systems because the offset shape and the original one might have radically different topology. In particular, an offset shape might have far fewer faces and edges than the original, especially if the offset distance is quite large. So, a b-rep system has the difficult task of figuring out how the offset faces intersect with each other. The following is a 2D example that illustrates what might happen. Suppose we want to “shell” this object (i.e. turn it into a thin-walled shape with a hollowed-out interior):



If we let  $d$  denote the shell wall thickness, then nTop's shelling results for various values of  $d$  are shown in the illustration below:



Offsetting is simple in an implicit modeling system because it is very easy to derive an i-function for the offset shape from an i-function of the original. In nTop, offsetting and shelling operations never fail. However, b-rep systems find the computations more and more difficult as the offset distance increases, so failures are common, as shown here:



## 9 Model Sizes

Models in nTop are smaller than the same designs in traditional CAD systems by a factor of roughly 60. The data below describe the sizes on disk of some lattice models composed of cylindrical beams:

Cells	125	1,000	8,000	125,000	1,000,000
<b>nTop</b>					
<b>Disk</b>	31KB	120KB	940KB	16MB	127MB

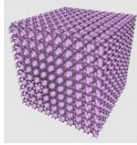
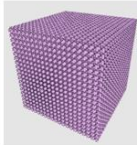


Cells	125	1,000	8,000	125,000	1,000,000
<b>B-rep</b> (Parasolid)				?	?
<b>Disk</b>	1.1MB	8.1MB	66MB	1 GB*	8 GB*

<b>Factor</b>	<b>35x</b>	<b>67x</b>	<b>70x</b>	<b>62x*</b>	<b>63x*</b>
---------------	------------	------------	------------	-------------	-------------

The numbers marked by asterisks are estimates, since I didn't have the patience to build these models in a traditional CAD system.

For triply-periodic shapes like gyroids, nTop models are amazingly small. The following table shows the sizes (in memory and on disk) of gyroid lattices with various numbers of cells

				
Array size	10x10x10	20x20x20	50x50x50	100x100x100
Cells	1,000	8,000	125,000	1,000,000
Memory*	280 MB	320 MB	340 MB	360 MB
Disk	6 KB	6 KB	6 KB	6 KB

The size on disk is only 6 KB in every case, because nTop just stores the gyroid equation:

$$F(x, y, z) = \cos x \sin y + \cos y \sin z + \cos z \sin x$$

A more realistic case is this heat exchanger that has a gyroid lattice structure with 10,000 cells.



	nTop	B-rep system
Time to generate	2 minutes	6 days
Disk space	1.3 MB (*)	1.5 GB

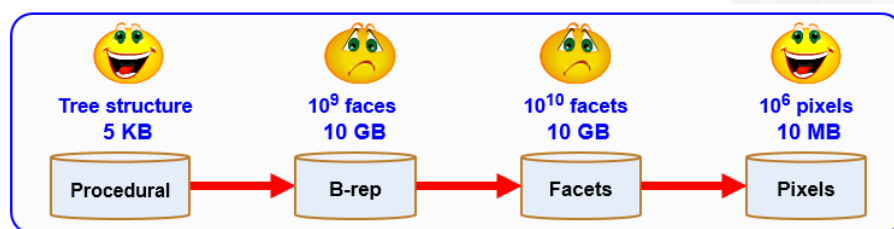
(\*) In fact, most of this 1.3 MB is an imported surface. The gyroid data itself is only about 10 KB.

## 10 The B-Rep Bottleneck

The enormous size of b-reps causes a bottleneck in a number of computations. In this section, we look at generation of shaded images as an example of this b-rep bottleneck effect.

### 10.1 Display in B-Rep Systems

In b-rep systems, the process of generating a shaded image of a model on your computer screen proceeds as in the diagram below:

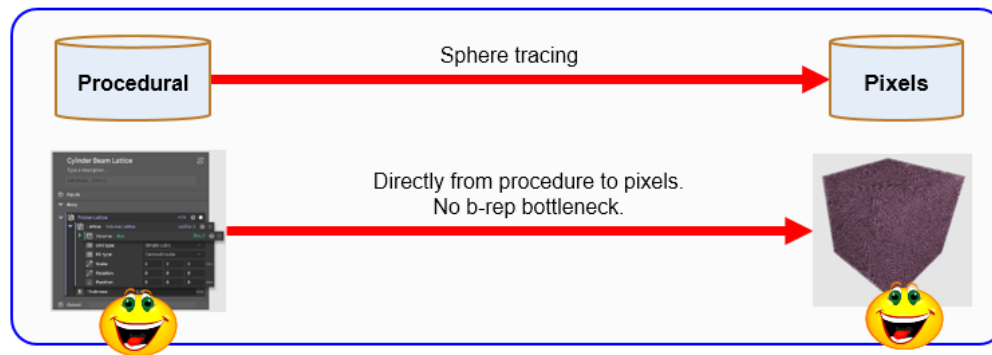


On the left, we start out with a procedural representation of the object to be displayed. This is the feature tree that is familiar to most CAD users. It's called a procedural representation because it contains a "workflow" or procedure for building the model, rather than the results of this building process. These procedural representations are very small; even a very large model might have a feature tree that can be stored in a few kilobytes. But then the next step is to

generate a b-rep, which might be enormous. For example, a lattice structure with  $1000 \times 1000 \times 1000$  cells might easily have  $10^9$  faces, and might require 10 GB of storage. Then, to prepare for display, the faces are divided up into triangular facets. Triangles are simpler than b-rep faces, but there will be more of them, so again the storage requirements might be around 10 GB. Finally, the facets are passed to the display subsystem, which uses the triangles to draw pixels on your screen. If your screen has 3 million pixels, and each pixel uses 24 bits for color, then the total image requires about 10 MB of data. So, we started out with 5 KB of data, and ended up with 10 MB, but the intermediate steps produced gigabytes of b-rep data, making the process impractical. This is the b-rep bottleneck, and it's the reason that b-rep systems can't display fine-grain lattices and other highly complex parts.

## 10.2 Display in Implicit Modeling Systems

The diagram below shows how nTop generates a shaded image:



The implicit representation in nTop is again a procedural one; in fact it's quite similar to the feature tree found in CAD systems. The big difference is that nTop's algorithms use this representation directly, without first producing any b-rep. For example, the pixels in a shaded image are computed directly from nTop's implicit representation using a technique called [sphere tracing](#), which is a more sophisticated variant of the well-known "ray-tracing" approach to generating images. B-reps are never produced, so we avoid the b-rep bottleneck.

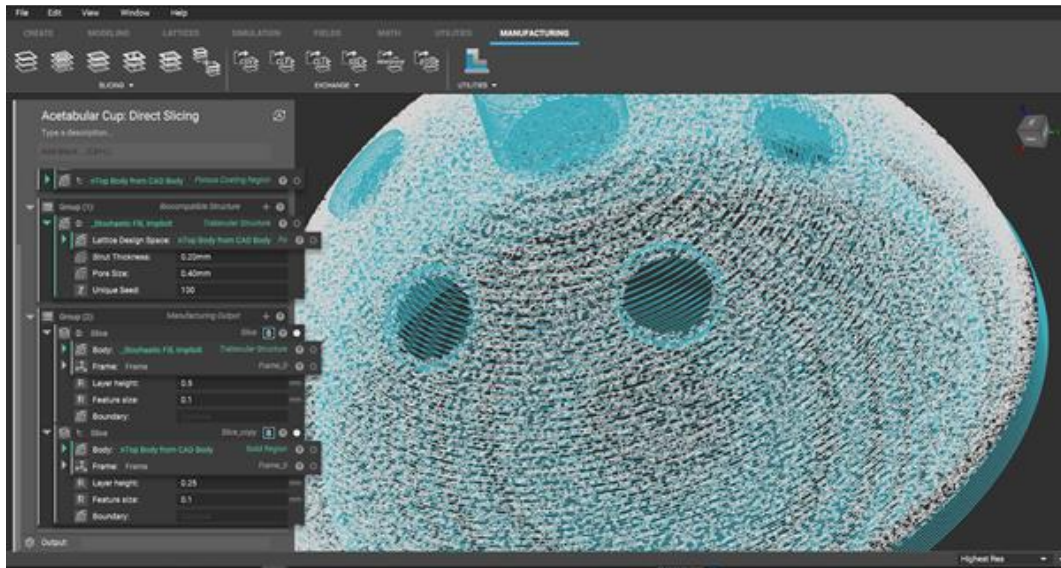
Producing images by ray tracing (or sphere tracing) involves a great deal of computation, so traditionally it has been slow, and has been used only for very high quality rendering. But the algorithms are also highly parallel, so are well suited to modern CPUs and GPUs, and the nTop implementation is fast enough for interactive display. Use of ray tracing in games is increasing, and so the latest graphics cards have special hardware to support this. Display performance in nTop will continue to improve because the game industry is driving rapid hardware advances.

## 10.3 Other Applications

Display is just one example, and there are many other important algorithms that suffer from the same b-rep bottleneck. In fact, in contemporary CAD systems, almost every application is driven by b-reps. So, typical CAD systems need a complete b-rep of the object you're designing, and they need it all the time, or else nothing works. Another interesting example application is producing "slices" of a model to drive a 3D printer. In most CAD systems, the slicing algorithm begins by producing a faceted representation of the model, which is then used to drive the slicing process. But, if the faceted representation is too large (as it often is), this process doesn't work.

In the nTop system, slices are produced directly from the implicit model; no b-reps are produced, so there's no bottleneck, and nTop can slice extremely complex models, like the one shown here:





## 10.4 Avoiding the Bottleneck

It's interesting to consider whether traditional CAD systems could avoid the b-rep bottleneck. Most of them have some sort of procedural representation of the model (the feature tree), so why can't they use this directly in applications, the way nTop does? Well, they could, of course, but it wouldn't be easy. CAD feature trees were not designed with computation in mind; their sole purpose is to support interactive editing and drive the generation of b-reps. Developers would have to write entirely new applications algorithms that have little in common with the old b-rep-based ones. Also, major user interface changes would be needed, because current CAD interaction techniques rely heavily on b-reps. The only way to eliminate the b-rep bottleneck is to eliminate b-reps, and doing this in a traditional CAD system would be a very big change.

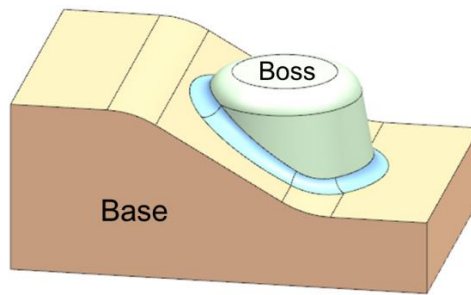
# 11 Stability of Links

In a parametric CAD system, the model is a tree structure composed of features. Each feature knows how to recompute itself when its parents (i.e. its inputs) are changed. By walking through the tree, recomputing each feature in turn, the CAD system can regenerate the entire model. The process is called replay, or rebuild, or regeneration, or update.

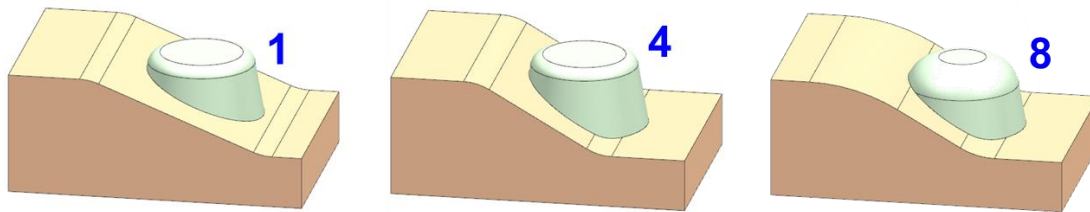
The replay process is error-prone for a number of reasons. First of all, it is a sequential process in which each step depends on the preceding ones: a given feature can be regenerated only after all of its parents have been. So, if any one feature fails to replay, the whole replay process fails. As models get more complex, the chances of failure increase merely as a result of the laws of probability. This is true in both nTop and b-rep systems.

However, in a b-rep system, there is another source of failures: the parent-child relationships between features are often prone to breakage because they involve edges. So, for example, we might define a fillet by specifying a particular edge to be filleted, together with some radius value. However, the next time the model is regenerated, that particular edge might no longer be present (because the two adjacent faces no longer intersect), so the fillet feature will fail, and the whole regeneration process will grind to a halt. More generally, the number of edges that need to be filleted can change as the model is modified. For example, suppose we want to fillet the junction where the boss meets the base part in this model:





The number of edges that need to be filleted can be 1, 2, 4, 6, or 8, depending on the shapes and relative positions of the base and the boss. Three examples are shown here:

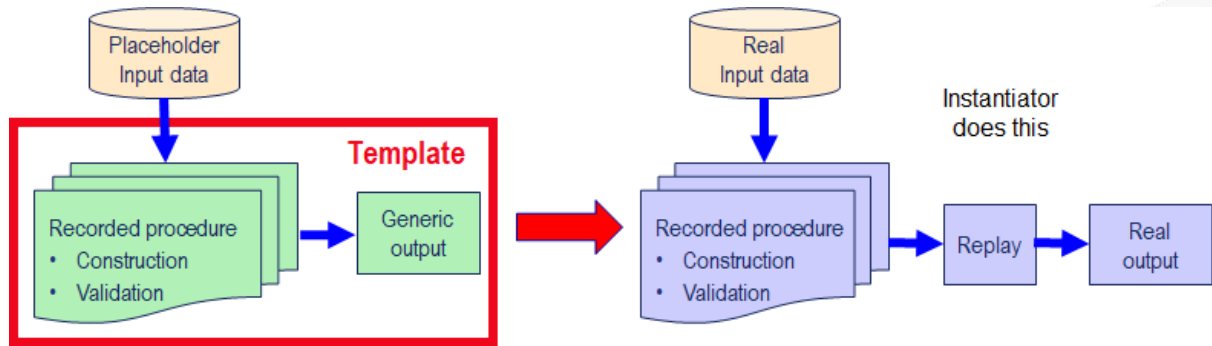


In nTop, we simply say that we want to fillet the junction of the base and the boss, and this is the information that's remembered in the filleting operation. This operation will replay successfully no matter how the model changes. So, replay in nTop is more reliable for two reasons: first, the basic modeling operations are more reliable, as explained earlier, and secondly there are no fragile references to edges (because nTop doesn't have edges).

## 12 Automating Engineering Workflows

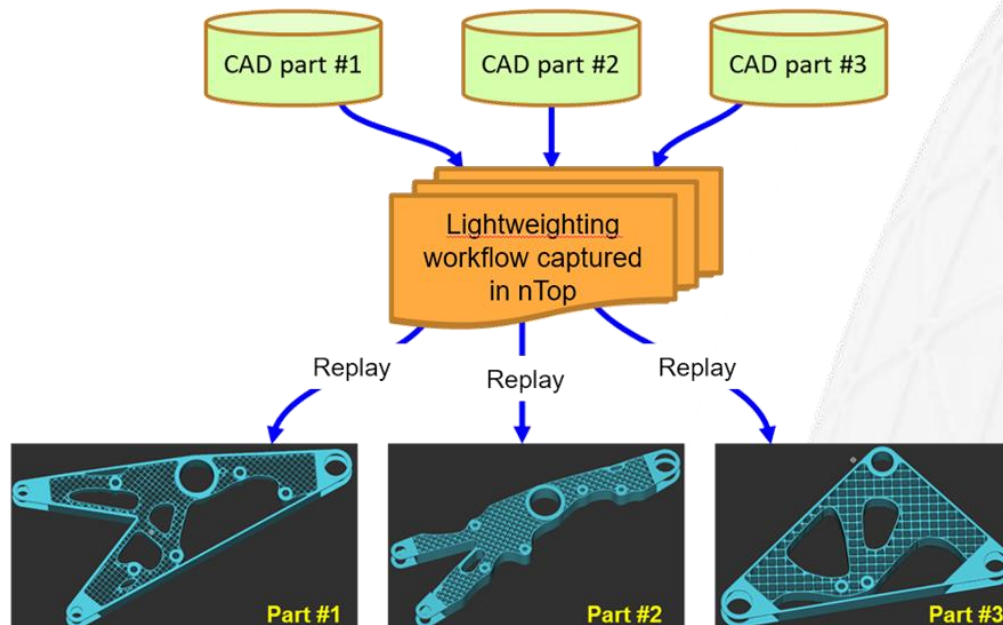
In parametric feature-based CAD systems, people sometimes try to build a model that captures the sequence of operations of an engineering workflow in a repeatable/replayable form. The model is built with fake "placeholder" inputs, which later get replaced by real ones. After this replacement, the model can be replayed, which means that all the operations are re-executed automatically to produce a new design. In some sense, the model has captured the "intelligence" of the engineering workflow. Ideally, the model embodies the process used by the company's most expert designers. In traditional CAD systems, these sorts of models are often called "template models". When this idea works, it delivers enormous benefits. Engineering work is automated, productivity and quality are vastly improved, best practices are captured and institutionalized, and so on.

The idea is illustrated in the following diagram:



Unfortunately, the idea often does not work, because it is extremely difficult to build template models that work reliably when the inputs are changed significantly. If the template model fails during replay, the person using it typically has no idea what went wrong, and usually resorts to building his own model from scratch, so all the benefits are lost.

We can use nTop models to capture engineering workflows in exactly the same way. However, because nTop's modeling operations are more reliable, and its parent-child links are more stable, replay failures are much less likely, even when the inputs change dramatically. So, nTop is able to actually deliver the automation benefits that were expected from parametric CAD systems. An example is shown below:



We have used nTop to capture a light-weighting workflow that uses a given CAD part as input. The light-weighting process might have been developed by the best engineer in the company, but, once captured in nTop, anyone can re-use it simply by providing a new CAD model as input. As you can see, the process works reliably on all three of these models, even though they are quite different.

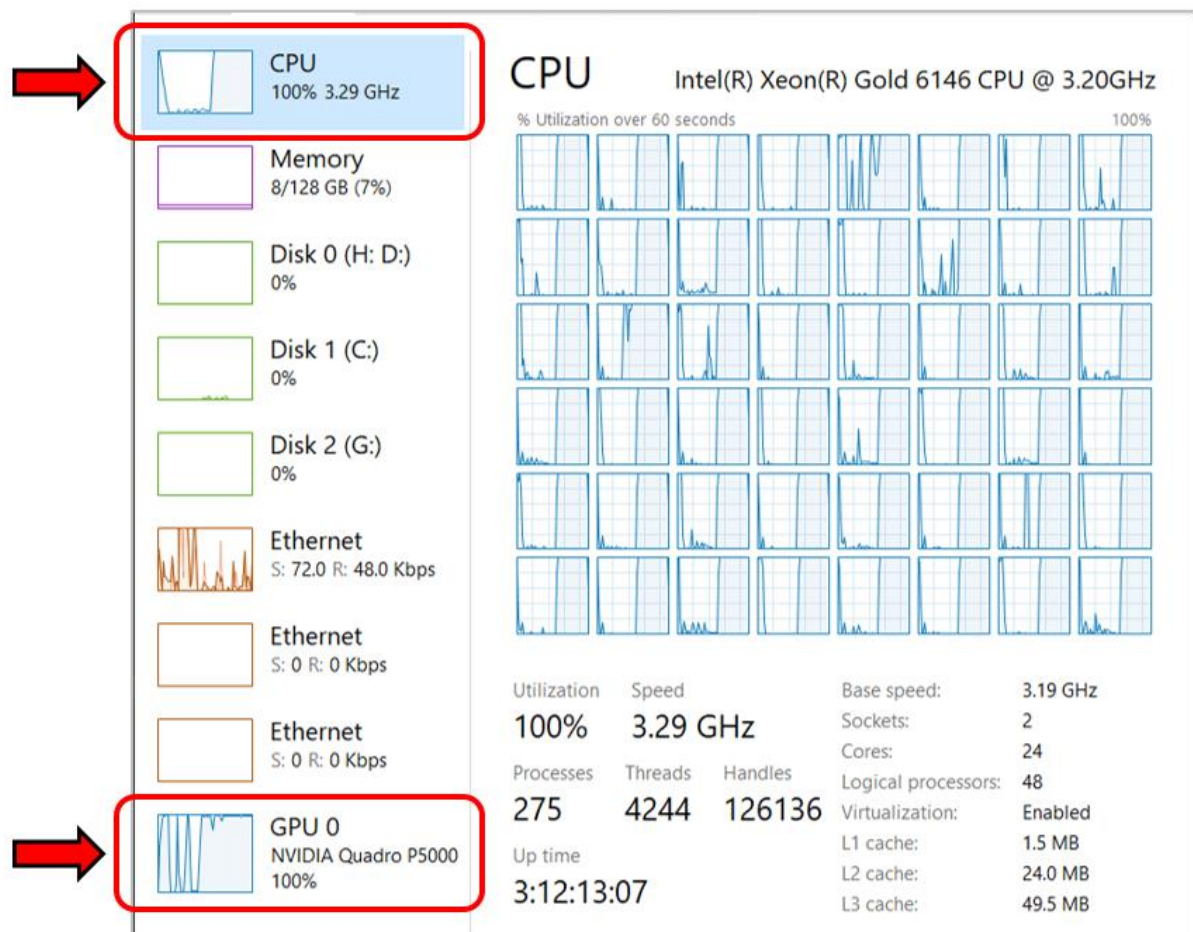
# 13 Modern Hardware & Modern Algorithms

Computers have changed dramatically since the time when b-rep systems were first developed in the 1970s. Specifically, they now have multiple CPUs, and enormously powerful graphics processing units (GPUs) that can be used for general-purpose computations, not just graphics. The nTop software is designed to take advantage of these modern computer architectures.

On the other hand, many b-rep computations are difficult to parallelize, often because they were designed in an era when parallel processing was a rarity and no-one thought much about it.

nTop's use of GPU computing is especially significant, because it means that the performance of our algorithms benefits from the rapid advances in GPU technology. The algorithms used in b-rep systems often include many special-case branches, as we saw in section 5.1, which makes them unsuitable for GPU implementation. GPUs are very good at computation, but they are very poor at decision logic, so they work best with simple direct algorithms that have no branching.

The image below shows nTop making good use of almost all the power available in a 24-core CPU, and also using the GPU for modeling computations:



## 14 Summary

In the discussion above, we have explained that ...

- nTop is a **different** kind of modeling kernel: it does not use boundary representations (b-reps), it uses implicit representations based on signed-distance functions
- To perform a modeling operation, we simply have to construct a new i-function from old ones. **No fragile calculations** (like intersecting surfaces) are needed.
- So, operations like booleans, filleting and shelling are all **instantaneous and 100% reliable**.
- Because it does not use b-reps, nTop can handle **extremely complex models**
- Because modeling operations are reliable, and we do not use edge selections (e.g. to define fillets), there are **no replay failures**.
- Reliable replay enables **workflow automation**.
- nTop is modern software that makes full use of **modern computers** (CPUs and GPU)

So, in summary, we have seen that **nTop uses a radically different approach to solid modeling that delivers large gains in speed, reliability, and scalability. Similar gains will be difficult to achieve in b-rep-based systems.**

## 15 Further Reading

If you want to know more about the topics discussed in this document, please consult the references listed below. There is lots of available material that describes how implicit modeling technology can be applied in computer graphics; applications in engineering and manufacturing are harder to find, because few people focused on this until nTopology emerged. The documents below are arranged roughly in order of increasing difficulty: the first few should be fairly easy to understand, but the later ones might require some effort, for many people.

- [nTopology blog: Implicits and fields for beginners](#)
- [nTopology blog: Implicit modeling for mechanical design](#)
- [nTopology blog: Understanding the basics of b-reps and implicits](#)
- [nTopology blog: How implicits succeed where b-reps fail](#)
- [nTopology blog: Automated filleting that won't break your model](#)
- [Introduction to Implicit Surfaces, by Jules Bloomenthal](#)
- [Signed Distance Fields, by Alan Zucconi](#)
- [2D distance functions, by Inigo Quilez](#)
- [3D distance functions, by Inigo Quilez](#)
- [Rounding Corners, by Inigo Quilez](#)
- [Ray marching and signed distance functions, by Jamie Wong](#)
- [Enhanced Sphere Tracing, by Keinert, Schäfer, and Korndörfer](#)
- [Representation and Rendering of Implicit Surfaces, by Christian Sigg](#)
- [Implicit Surface Modeling, by Hongxin Zhang and Jieqing Feng](#)
- [3D Distance Fields: A Survey of Techniques and Applications, by Jones, Bærentzen, and Sramek](#)