Some upsetting things about shapes

That we already knew

Dr. Tom Murphy VII, Ph.D.
March 2025

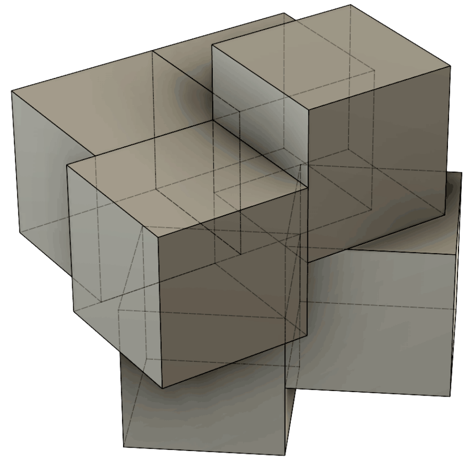
How big is the unit cube? “1?” Seems obvious, right? Let me ask you another way: *Where* is the unit cube? “0?”

Or: How big is the unit sphere? “1?” Imagine I am asking you these questions where I pounce upon you with the next question just as you start to answer the first. To keep you off your balance, I mean. *Where* is the unit sphere? “0?” Which is bigger, the unit cube or the unit sphere? Sweating yet?

“Officially” speaking, the unit cube has edge length 1, and has all non-negative home coordinates, like ({0, 1}, {0, 1}, {0, 1}). According to those same math referees, the unit sphere has *radius* 1, and its center at (0, 0, 0).

So the unit cube fits easily within the unit sphere. I don’t know about you, but I always imagine the idealized cube and sphere centered on the origin, with the sphere tucked inside the cube, touching its sides at their centers. (**footnote**: The long-running ThursDz’s Beer Society of Math Geniuses decided that what I am actually imagining is not the unit cube but the “L-infinity unit sphere,” which I think may be true but sounds like I’m just trying to be an asshole.)

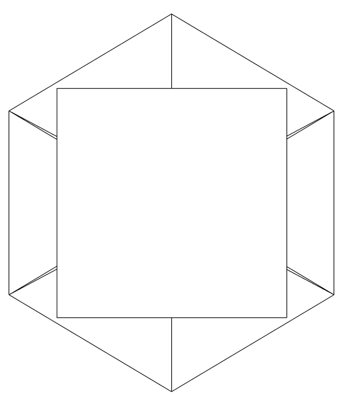
Of course we can only fit one unit cube at a time in the unit sphere, but there’s a lot of space left over. You could ask yourself, if I have n cubes, what’s the smallest sphere I could fit them within? Would it surprise you to find out that for n=6, the tightest arrangement is not a 3×2×1 grid? Or a 2×2 slab with a centered cube above and below? Or even something symmetrical? For example, this arrangement fits in a sphere that’s a little smaller:



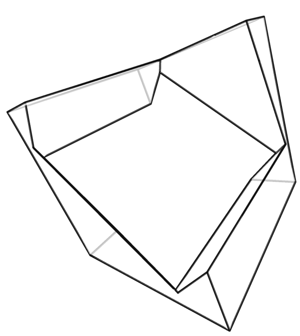
What the hell, right? I found this by computer search (namely: after trying to solve the problem myself, I searched on the internet using a computer for *spoilers*, and found some[5]). I love it! (Platonically!) This is how it goes with me: I do enjoy a beautiful math result where everything is in its right place, but to be honest, what *really* titillates me is when the *problem* is beautiful but the *solution* is upsetting. I call this a “Platonic Horror.” One of the things I like about it is that any clear thinker will imagine cubes and spheres, and ask a simple well-formed question like that. You could contrast this with man-made horrors like “the conventional unit cube and unit sphere are apparently using the word ‘unit’ differently,” or “IEEE-754 denormals”[6] or “`U+FE18 PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRAKCET`”[7]. Aliens would never think to spell the word “bracket” wrong in the formal name of a Unicode character. But they would think about putting cubes in spheres, and then find out that it’s kinda messed up, and then some of them would find it strangely titillating that the solution is not “nice,” and they probably have their own version of SIGBOVIK out there on ΞΔO-ϒΓΘX 11 where they have giggled about this specific fact, and I like that idea.

Anyway, now that I’ve gotten the “Dice in Sphere” pun out of the way, we can move onto the real topic of this paper: *Dice in Dice*.

So, um, why is so much energy wasted on trying to wreck the world? Something about being super rich and powerful seems to attract people to asshattery, or assholery, or assault, or making an ass out of u and me. All throughout history we have had this problem. If I were super rich, I would just hang out with my smart friends and do math. Right? One rare example of this seemingly working out well was the Prince called Rupert, formally *Prince Rupert of the Rhine, Duke of Cumberland, KG, PC, FRS* (not off to a good start, to be honest, but maybe all the appellations are semi-meta-ironic like Dr. Tom Murphy VII Ph.D.). I didn't learn much about this prince (he's been dead for hundreds of years) but once he retired from being a war sailor, he "converted some of the apartments at Windsor Castle to a luxury laboratory, complete with forges, instruments, and raw materials, from where he conducted a range of experiments."[8] Yes! Correct! This is the same fellow who learned about Prince Rupert's Drop and then did not protest when people attributed its invention to him (hard to blame them for the mistake with a name like that). Then during what I imagine was a pleasant evening in the pub arguing with his friends over presumably nasty 17th-century beer, he came up with the following question: Can you pass a cube through another cube of the same size? The answer is, surprisingly: Yes! If you rotate the cube so that it looks like a square (the small way), and rotate the other so that it looks like a hexagon (the big way), then you can fit it through:



This became known as Prince Rupert's Cube, and the cube donut that's left over looks like this:



Late one night I was admiring the Wikipedia article on the Dodecahedron, my favorite Platonic solid. On this page I was reminded that the Dodecahedron is Rupert, like the cube. This seemed right, since if the cube has a pleasing property and the Dodecahedron is awesome, it should also have that pleasing property. (**footnote**: This is of course not true. For example, the cube can tile space and the dodecahedron obviously cannot.)

Indeed, all the Platonic solids have the Rupert property. The Platonic solids are beautiful and so the fact that all of them have this pleasing property recommends it further. Then I read the phrase "it has been conjectured that all 3-dimensional convex polyhedra have this property," which made my brain feel surprised but happy. I might have even gotten to sleep, had I stopped reading at that moment. But then I read "of the 13 Archimedean solids, it is known that at least ten have the Rupert property," and this made my brain surprised and upset. How could it be the case that we think this is true for *all* convex polyhedra (infinitely many, and mostly gigantic weird ugly ones) but we don't know for some 3 simple beautiful ones? Did nobody check? It seemed to me it would be pretty easy to write a search procedure that would look for them, and it also seemed like if we think it's possible, it would be easy to find solutions.

Here were my Naive Heuristics:

- This is a continuous problem. If you have some way of fitting the shape through itself, then there will be some adjacent small variation on that that will also work. Problems where the solution needs to be exact (e.g. problems on integers) tend to be much harder. These solutions won't need to be exact because of *NO TOUCHING!*
- This might be a problem that not that many people have tried (only stamp collectors), since serious mathematicians would be interested in a real solution (i.e. a proof that the general conjecture is true).
- I am definitely not the best mathematician to have tried this, but it's possible that I'm the best programmer to try it, and plausible that nobody has tried it with a very large and hot GPU.
- If I find solutions, great; we can check those off the list. If I don't, I'll learn something, because it doesn't seem like it should be conjectured to be true but hard to find.
- I can whip this up in a day (or maybe a weekend) and then put it aside if I can't solve it.

So I set aside two and a half months to work on it. (**footnote**: The project's home page at `tom7.org/ruperts` contains links to the source code repository, and will contain video content in the future.)

Platonic, Archimedean, and Catalan solids

The convex solids called Platonic all have faces made from the same regular polygon (like a square or equilateral triangle). There are only five: The tetrahedron, cube, octahedron, icosahedron, and dodecahedron. (For images of these shapes you can see the results section at the end of the paper.) I love them! (Platonically!) (And also sexually!) It's kind of amazing that there are only five, but here's a good way to feel comfortable with that idea. First, remember that regular hexagons fit together perfectly to tile the plane. If you tried to put septagons or octagons together, they would not fit; hexagons are the last ones that fit. With five or fewer sides, you can't tile the plane, but you can fold them over and start making a polyhedron (3D shape). This tells us that a Platonic solid must have faces that are pentagons, squares, or triangles. The pentagon is pretty big so there's only one way to put them together. Same for the cube. The triangle has three ways, but it's not too hard to see that you are limited to these three if you try to work it out.

These solids are named after Plato, but it's obvious to me that any clear thinker would eventually discover these; if there are aliens somewhere, then they know about the same five shapes, though probably not by the same name unless they have been creepily spying on us, or perhaps if Plato was not all that he seemed!

There are two kinds of convex solids that are almost as good as the Platonic ones: The Archimedean and Catalan solids. You might want to look at the results section to see pictures of these, since they are cool and you probably like cool shapes. The Archimedean ones have faces that are regular polygons (but more than one type), and moreover are *vertex transitive*. Vertex transitivity means approximately that every vertex on the polyhedron has the same shape (the connecting edges and faces are the same, just maybe rotated). One way to think about this is that you could 3D print some connectors with holes in them and assemble the solid of connectors and straws, and you would only need one kind of connector. For a similar reason that there are only five Platonic solids, there are only 13 Archimedean solids.

Each Archimedean solid has a dual, which is a Catalan solid. These are perhaps cooler. Catalan solids have symmetric vertices and are *face transitive*; the faces are not regular polygons but if you cut out polygons from paper you would only need one shape to make these. There are also thirteen of these.

The Archimedean and Catalan solids are canonical and probably also known to aliens. There are some further generalizations (like the Johnson solids), but each time we get weaker properties, weirder shapes, and more of them. It becomes less likely that Aliens are out there holding their own SIGBOVIK conference and thinking about the same thing. So in this paper I'm only concerned with the Platonic, Archimedean, and Catalan solids, which I'll abbreviate P/A/C.

All of the Platonic solids have the Rupert property ("are Rupert"), which we'll define more carefully in the next section. Upsettingly, only *most* of the Archimedean and Catalan solids are known to be Rupert. The unknown ones I'll call the "wishlist" polyhedra in this paper; they are:

The Archimedean solid called the **rhombicosidodecahedron**, and its dual, the Catalan solid called the **deltoidal hexecontahedron**; the Archimedean **snub dodecahedron** and its dual, the Catalan **pentagonal hexecontahedron**; and the Archimedean **snub cube**. Upsettingly: Its dual, the pentagonal icositetrahedron, *does* have the Rupert property!

Related work which I did not read

I should mention: The reason that we know that solutions to most of these exist, and that they are written about on Wikipedia to keep me up late at night, is due to the *related work*—a.k.a. the *spoilers*. Since I am highly spoiler-averse and writing for SIGBOVIK, whose prestigious standards and practices transcend pedestrian norms like a related work section, I did not look at the related work *at all* while doing this research. It might take the fun out of doing it myself. It is, after all, *re*-search!

I can however cite a few spoilers for your convenience.[9][10][11]

The Rupert problem

As we said, a solid is "Rupert" if you can pass an identical copy of the solid clean through itself, leaving a proper hole. It's easy to think of non-convex shapes where it's clearly not possible, and convex shapes that aren't polyhedra (like the sphere) where it's clearly not possible. The conjecture is that all convex polyhedra are Rupert.

This problem is pretty easy to specify precisely. The shape in question is a convex polyhedron, which is just defined by its set of vertices. For general polyhedra you also need to specify how those vertices are connected (the edges and faces), but convex polyhedra are easier. There's just one way to stretch a "skin" over the points, so we don't even need to describe it (or even think about it). We'll take two copies of the points. One is the "outer" polyhedron and one is the "inner". The goal is to find some way of arranging them so that the inner can pass through the outer.

The inner one will pass through the outer along some line, so we say without loss of generality that this is the z axis. We'll use the computer graphics convention that the camera is located at some positive z, looking down at the shapes, which are near the origin, and the inner polyhedron is moving along this same line of sight. Maybe like it's shooting out of our eyeballs as a kind of abstract weapon of geometry. A Platomic Bomb. Viewed this way, what it means for the inner shape to be able to pass through the outer is that the two dimensional "shadow" of the inner shape is entirely contained within the shadow of the outer shape.

We'll specify the arrangement of the polyhedra as a rotation and translation; together these are a rigid frame (hereafter just "frame"). Because we know we're traveling along the z axis, the z component of the translations are unimportant and we can just consider 2D translations. Moreover, since we just care about the relative positions of the objects, we can say that the outer polyhedron is fixed at (0, 0). We need to be able to rotate both shapes arbitrarily, though.

The inner shadow being completely contained within the outer shadow is intuitive, but we should be more precise. The convex hull of a set of 2D points is the minimal convex polygon that contains them all (here "contains" includes the boundary); this is the same idea as the minimal skin around the vertices of our convex polyhedron. To get the shape of the shadow, we just project the object to 2D along the z axis (easy: (x, y, z) just becomes (x, y)) and then compute the convex hull of the points using standard algorithms. Now we can just ask whether the inner hull is entirely contained within the outer hull. Since the outer hull is convex, this amounts to a standard test that each point on the inner hull is contained within a convex 2D polygon. You can find slightly buggy code for this all over the internet. (There are many alternative formulations, some of which are discussed below.)

The boundary condition here is very important. The inner points must be *strictly* contained within the outer hull (less-than, not less-than-or-equal), never exactly on the boundary or coincident with an outer vertex. If we allow them to be on the hull, then carving the inner through the outer would make the residue disconnected (perhaps dramatically so). It also makes the problem trivial: If the outer and inner have the same frame, then their shadows are also the same, and the inner one is trivially (weakly) inside the outer. If you think this amounts to "passing one cube through the other and leaving a proper hole," then you and I disagree about what proper hole means.

So to solve the Rupert problem for some shape, you need to find two rigid frames that satisfy the above (and we know that one of the translations can be (0, 0, 0) and the other (x, y, 0)). How do we find such frames?

If you have a fast enough test, sampling will suffice for easy objects like the cube. Here you just generate random frames and test whether the condition holds. You can try all orientations and reasonable bounds on the translation (you do not want to translate more than the diameter of the cube, for example, or it will definitely not go through it)!

Generating random orientations

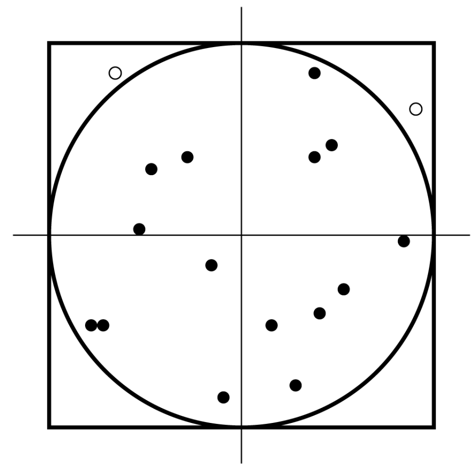
Generating random numbers is easy using floating point roundoff error.[12] How do you generate a random rotation (orientation)? There are a few different ways to specify a rotation. You can use Euler angles, which are three parameters that give the rotation around the x, y, and z axes ("pitch," "roll," and "yaw"; see Figure 1). This approach actually sucks (famously, Euler was not that good at math). You can get all orientations this way, but you will get some orientations more often than others (this is related to the phenomenon of "gimbal lock"). Maybe that is okay for you (or Eu-ler) but I want all orientations to be equally likely.

Figure 1. Wikipedia[13] provides this useful mnemonic for remembering which axis corresponds to each of the three words. The **pitch**er makes sense, since you famously use a pitcher by holding the handle away from you and turning your wrist to pour diagonally towards yourself. But d**oor** must just be trolling, right?

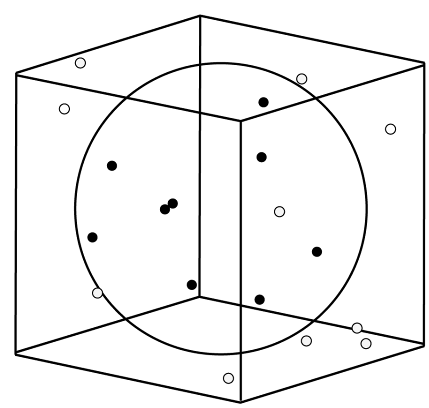
A good way to do this is using Quaternions, the even more mysterious second cousins of the complex numbers. I will not try to give you an intuition for quaternions (since I do not really have one) but they can be used as a four-parameter representation of orientations that will leave you happy (and puzzled) instead of sad (and puzzled). Facts to know about the Quaternions:

• Most people don't capitalize Quaternions.
• Like complex numbers where you have a + b**i**, here we have a + b**i** + c**j** + d**k**. The parameters are (a, b, c, d) and **i**, **j**, **k** are "even more imaginary" "constants" that have some impossible relations, like $\mathbf{i}^2 = -1$ but also $\mathbf{ijk} = -1$.
• You can just think of a quaternion as a four-dimensional vector (a, b, c, d). If this is a unit-length vector, then it represents an orientation. There are exactly two unit quaternions representing each unique orientation in 3D. No gimbal lock and no favorites.

A good—but not great—way to generate random 4D unit vectors is to generate random points on a 4D hypersphere, because these are the same thing. There are very fancy ways to do this, but you run the risk of getting the math wrong, or head explosion etc., so I recommend rejection sampling. Rejection sampling is a very robust way to generate uniform samples in some set. What you do is generate random points inside some domain that contains the target set, and throw away points that aren't in the target. For example, to generate points in a unit circle, you can generate points in the 2×2 square (it's *not* the unit square) that contains that circle. $\pi/4$ of these points will be in the circle, and so you get samples at an efficiency of about 78.5%.

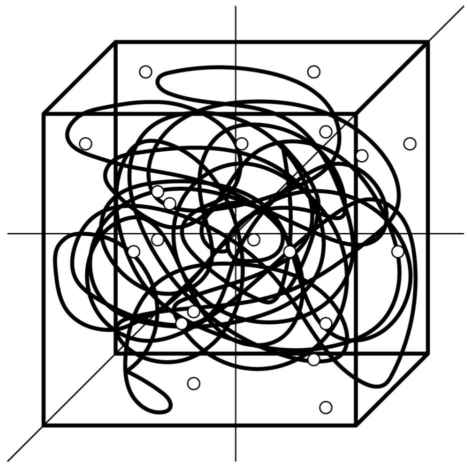


To generate points inside a sphere, you do the same thing, but in a 2×2×2 cube. This sphere has volume $4\pi/3$ and the cube has volume 8, so you get samples at an efficiency of about 52.4%.



To generate 4D points inside a 4D hypersphere (**footnote**: We should say "3-sphere," or "hyperball", since the convention is that a normal sphere in 3D is called a 2-sphere, since its surface is actually two dimensional. It would just seem to add confusion here, though.), you do the same thing, but now the hypervolume is $\pi^2/2$, and the 4D hypercube has hypervolume 16, so you get samples at an efficiency of about 30.8%.

Upsettingly, as we increase dimensions, the hypervolume of the *n*-dimensional hypersphere approaches zero (!?) and the *n*-dimensional hypercube's volume grows exponentially, so this technique approaches perfect 0% efficiency.

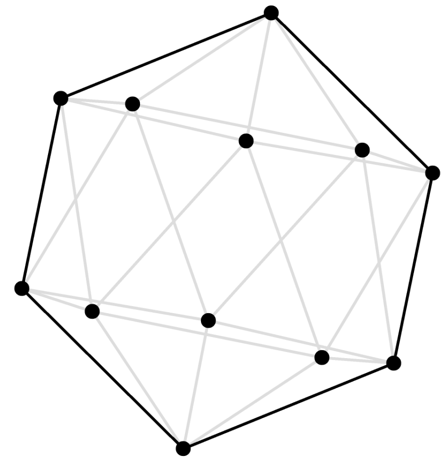


Fortunately, we only need 4D vectors, and 30% efficiency is fine because my computer can calculate like 1 billion samples per second and I only need two.

The two samples give the orientations of the outer and inner polyhedra, and we also pick random positions. We then project to 2D, compute the convex hulls, and see if the inner hull is inside the outer hull.

The convex hulls

The 3D shapes are convex, and so their 2D shadows are convex. (**footnote**: It is not completely obvious that this must be true. One of the ways to believe it follows from a definition of convexity: For every pair of points in the set, the entire line segment between them is in the set. To show that the 2D shadow is convex, take any two points in it. These points correspond to some two points in the 3D shape, which means (by that definition convexity) that the line segment between them is in the 3D set. The projection from the 3D shape to the 2D shadow also transforms that line segment to a line segment (the projection is *linear*) and it connects the 2D points. So this satisfies the definition of convexity for the 2D shadow. In fact, *all* linear transformations preserve convexity by the same argument.) Rather than just working with the set of points, their boundary polygon is a much more convenient representation of the shadow. Here is a shadow of the icosahedron. The darker boundary polygon is its 2D convex hull:



Computing the convex hull is also "standard," meaning that you can find lots of slightly buggy implementations of various algorithms on the internet. The bugs are usually because the routines are intended for computer graphics and so they don't have to "work," and because the algorithms are conceptualized in the mathematical world where when you look at a point that's really close to a line segment, the point stays on the same side of the line when you look at it from different directions. This is unfortunately not the case for naive implementations using floating point. It usually "doesn't matter that much," or "just add a magic constant you named epsilon,[14]" but unfortunately when you are working with extremely regular shapes like Platonic solids, you will frequently get points that are colinear or coplanar and exercise the too-optimistic beliefs of the code you found. So this is another good way to make your afternoon project take several months.

You only need to compute the outer hull; you can then just check that all of the inner shadow's *vertices* are inside it. But I found it was faster to compute a convex hull for the inner polyhedron as well. That way you only need to do the point-in-polygon test for the points *on* the inner hull. The point-in-polygon test is standard; we just have to make sure we are testing that the points are *strictly* inside, and not on the hull itself.

Optimizing

Now we can test whether some random orientations and positions (frames) demonstrate the Rupert property. It is easy to find solutions for the cube by just sampling. But of course we want to make it faster, first of all just for the heck of it, but also so that we can solve the unknown cases, which are presumably harder.

I started with black-box optimization, again using my own twisted variant of BiteOpt.[15] Black box optimization is good for people like me and Euler who are bad at math. The interface to such an optimizer is a function like

```
double F(double a1, double a2, ..., double an)
```

For some fixed n. The optimizer doesn't know what the parameters mean; its job is just to find the arguments (`a1`, ..., `an`) such that `F(a1, ..., an)` has the smallest value. This is of course impossible in general, (**footnote**: Not just hard because the function could be complicated. It's literally impossible due to diagonalization. Take for example the recursive function `double F(double x) { return -abs(x - Optimize(F)); }`. This computes its own minimum, and then returns the negated distance from the argument to that supposed minimum. This makes the purported minimum actually the maximum (0) with a nice convex triangle all around it. In reality this function will just loop forever, since Optimize works by calling the function many times.) but for many well-behaved functions these optimizers are nonetheless able to do a good job.

Here the arguments will be the orientations of the two polyhedra and the position of the inner one. We can represent the orientations with quaternions (four parameters each) and the position as the (x, y) offset, totaling ten parameters.

The optimizer does need some kind of surface to optimize over; it does not work well if there is just a single point where the function returns -1 and it is a flat 0 everywhere else. I tried several approaches here. The one that worked best for me was to take all the vertices on the inner hull that are *not inside* the outer hull, and sum their distance to the outer hull. This prefers the vertices to be inside where we want them, and increases the penalty as they get further outside. It is essential to add a nonzero error when the point is not strictly inside; if the point is exactly on the hull or the distance rounds to zero, we still need to add a small positive value. Otherwise the optimizer will quickly find degenerate "solutions" such as setting both orientations the same.

The other rub is that the optimizer wants to try any value (within specified bounds) for the arguments, but we need each orientation to be a proper unit quaternion. Simply normalizing the four inputs would work, but as we observed before, random samples in this parameterization are not uniformly random orientations. My approach here is to first choose actually random quaternions for the outer and inner shape before beginning optimization. I then optimize within fairly narrow bounds (like -.15, +.15) for the quaternion parameters, and add that as a "tweak" to the random initial quaternion, normalizing to get a proper orientation. This is still not uniform, but it is locally closer to uniform. Since we will try optimization millions or billions of times from uniformly random starting orientations, we will get good coverage of all orientations. Other parameterizations of the orientation are possible. It is definitely desirable to have fewer arguments (as the complexity naively grows *exponentially* in the number of optimization parameters), but simply using the three-parameter Euler angles runs into the aforementioned problems.

It works!

Anyway, that works! This was like, the first weekend of the project. It's able to find solutions to the cube in milliseconds, and so I added more polyhedra to the collection, and solved those in milliseconds as well.

One of the most tedious parts of this was getting all of the polyhedra represented in computer form. I was somewhat surprised that the formulas for these things often involve wacky irrational coordinates like the "tribonacci" constant, which is like the Fibonacci (Fi- means two, like in the number Five) but where we take the sum of the previous three numbers instead of two. The ratio of terms converges to:

$$(1 + \sqrt[3]{(19 + 3\sqrt{33})} + \sqrt[3]{(19 - 3\sqrt{33})}) / 3$$

Like, I would expect $\sqrt{2}$ stuff. But I guess I should not have been surprised by that, because that's just math. Anyway, since these are all convex polyhedra, at least you don't need to explicitly specify the connectivity of the vertices. I just compute the 3D convex hull (using a slow polynomial-time search for coplanar vertices where all the points are on one side of the plane) to get the faces; it's okay that this is slow because you only need to do this once at program startup time. (**footnote**: In fact, the solvers mostly just work with the point sets; we know they are convex so this is enough for us to implicitly reason about the shape. We do at least want to draw the polyhedra for debugging or posterity purposes.)

Once all the polyhedra are in the computer, I easily confirmed what we already knew: The Platonic solids, ten of the Archimedean solids, and eleven of the Catalan solids, are Rupert.

Alternate solvers

Of course we should check uniformly random configurations, but I tried some other approaches as well:

Max. This first optimizes the outer shadow so that it maximizes its area. We then perform optimization only on the inner shadow. Intuitively, you want the outer shadow to be "bigger" and the inner shadow "smaller," so this makes sense as a heuristic and reduces the number of parameters. Largest area does not mean it is best at fitting a given inner shape, though. This strategy can solve all the polyhedra (with known solutions) except: triakis tetrahedron.

Parallel. Thinking about making the inner shadow as *small* as possible, we see that we often (always?) reach a numeric minimum when at least one face is parallel to the projection axis; this face then becomes zero area in the shadow. This strategy chooses two non-parallel faces of the inner polyhedron at random, and then orients the polyhedron such that these are both parallel to the z axis. It also rotates the polyhedron around the z axis such that one of these faces is aligned with the y axis (this doesn't really change anything except to make the numbers rounder and the hulls easier to interpret, e.g. the cube will always be an axis-aligned square). Then we just optimize the outer orientation and position to fit around this hull. This strategy can solve all the polyhedra except: tetrahedron, triakis tetrahedron.

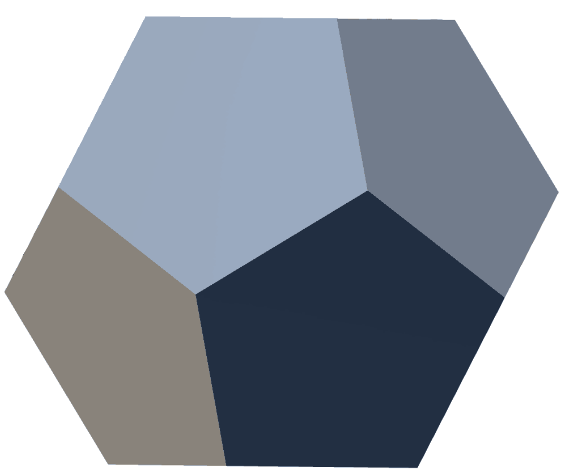


Figure 2. The dodecahedron with two non-parallel faces aligned to the z axis.

Origin. Optimize both rotations, but leave both polyhedra centered on the origin. This reduces the number of parameters, although the translation parameters are the best behaved of the bunch (optimizing the translation parameters alone is actually a convex problem). The main reason to do this is to see whether there are always solutions that have this form. It does not appear to be the case: The tetrahedron-like shapes seem to *require* translation. (**footnote**: I gave a half-hearted attempt to prove this by computer in the "Other approaches" section below. Given how narrow the clearance is for the triakis tetrahedron (and how simple the tetrahedron is), it may be tractable for someone who is good at math.) This strategy can solve all the polyhedra except: pentagonal icositetrahedron, tetrahedron, triakis tetrahedron, truncated tetrahedron.

Special. Combines parallel and origin, leaving only the other rotation to optimize. Like the origin approach, the main reason is to see whether solutions of this form exist; it turns out to work in all the same cases as the origin method. This is all of the polyhedra except: pentagonal icositetrahedron, tetrahedron, triakis tetrahedron, truncated tetrahedron.

GPU solver

At this point, I was easily solving the polyhedra with known solutions, like each in a few hundred milliseconds, and not at all solving the other ones. I figured one possibility was that these were just harder, and so I needed to be able to optimize the solver to try a lot more times. One way to try a lot more times is to do it on a Geometric Polyhedron Unit. Part of the way I justify to myself buying the world's physically largest (**footnote**: It's comically large. I literally broke my computer trying to install it, and had to buy an entirely new computer with a bigger case just to fit it in there.) and hottest GPU (at the time), the NVidia RTX 4090, is that I can use it for important tasks like this and not just sniping simulated soldiers in glorious 4k HDR at 144fps. So I rewrote the solver in OpenCL.

In some ways this problem is well suited to the GPU; it excels at parallel numerical tasks on floating-point numbers. The polyhedra here are too small to benefit from parallel computation on their vertices. But we can easily get massive data parallelism by trying multiple optimization instances in parallel. On the other hand, the convex hull calculation and black box optimizers are not natural for the GPU (OpenCL does not really support recursion!).

To test whether the inner shadow is within the outer shadow, I replaced the convex hull-based test with one that is worse but more easily parallelized. For each polyhedron I generate its triangulation, where each face is made with triangles (this is trivial to do with triangle fans because they are convex polygons). Now observe that when I project these triangular faces to the 2D shadow, any point that is contained in the shadow will be contained in at least one of these projected triangles. I can check all of the triangles in parallel. I can also compute the error for a point as its shortest distance to any triangle (like we previously used the shortest distance to the hull). The point-in-triangle tests must be *strict* as before, to prevent points exactly on the outer boundary from counting. Alas, this test is not quite correct here: It is possible for an interior point to land *exactly* and *only* on interior edges of the triangulation. Take an axis-aligned cube, for example; the point at the exact center of its square shadow will lie on edges of the triangulation, no matter which one you use. This is not ideal, but it only gives us false negatives (failing to find a solution if one exists), which is not a serious problem.

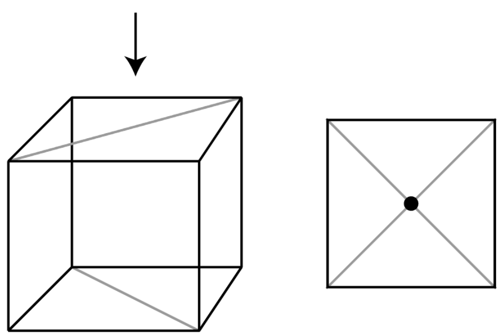


Figure 3. Left: A cube may be triangulated like this (only top and bottom triangulations shown for clarity). Right: Viewed from the top, the center point is not strictly within any face triangle.

Because I did not want to port the black-box optimizer, and because we can do better anyway since we understand the problem being optimized, I implemented a proper gradient descent optimizer for the GPU. This subject is well documented so I will not belabor it here, but I performed "approximate numerical differentiation" to compute the derivative with respect to each parameter independently. This involves evaluating the function one additional time for each parameter (with a small tweak), assuming that the slope is locally linear. It's not too bad to implement, but since this problem has 10 optimization parameters, it is a significant amount of additional evaluation. I don't think this problem lends itself well to analytical derivatives (even though most of the space is very smooth, the regions of interest are near the boundaries, either as a point moves into its own shape's shadow, or across the other's hull), but maybe you or someone else who's smarter than me could figure it out. Lazy people would use automatic differentiation and might be happy with that.

Anyway, this all works too! It is indeed faster than the CPU version, although it is harder to play around with algorithmic tweaks and it scales worse to polyhedra with larger triangulations. I mainly found solutions using the CPU methods, and mainly because running things on the GPU means I can't simultaneously use my computer for other important activities like over-the-top violent first-person shooter games.

Solved!

And then I found a solution for one of the wishlist polyhedra! Actually all of them. I didn't get too excited, though; there had been many false positives so far (due to bugs), and the reported numbers were like this:

```
outer frame:
-0.99999999999999978,-3.7558689392125502e-16,-3.5847581116984005e-08,
3.7558689392125502e-16,1.0000000000000002,-2.0954657592967021e-08,
3.5847581116984005e-08,-2.0954657592967021e-08,-0.99999999999999956
0, 0, 0

inner frame:
3.3306690738754691e-16,0.99999999999999978,5.551115123125779e-17,
2.7755575615628914e-16,-5.5511151231257852e-17,0.99999999999999978,
0.99999999999999978,-2.2204460492503128e-16,-2.7755575615628909e-16,
-1.4197330001097729e-18,2.8394660002195473e-19,-0.0051151272082079749

Ratio: 0.99999999999999999766
```

Note how everything is either really close to 1 or zero. Recall that two equal frames produce identical shadows, and that these are invalid Rupert configurations (the "hole" eats the entire shape). So too when the orientations are the same up to symmetry (e.g. one rotates the cube 1º and the other 91º). So I knew it was possible that we could get something really close to identical shadows, but that they might look like they satisfy the condition within the precision of double-precision floating point numbers. Also, given my fetish for IEEE-754, I'm certainly asking for it! Visually inspecting these solutions, this is exactly what they looked like.

ON THE OTHER HAND, some solutions can have a lot of nines in them! For example, the best known (to me) solution for the triakis tetrahedron comes within *one one-millionth* of the radius of the polyhedron, requiring a monumental amount of zooming-in to even perceive this thread as having volume. This would be a good reason that nobody found these solutions before: Perhaps they used single-precision floating point, or coarse values of "epsilon," or rejected them with visual inspection? One of the solutions, for the rhombicosidodecahedron, actually had a computed ratio of `0.99999998752759711`, which is definitely in the range where you start expecting doubles to act like numbers.

So I invested further effort.

Rational solvers

The right way to deal with floating point inaccuracy is to not use them. Lots of geometry will work great with other number systems, so with a little finesse we can work on this problem using rational numbers and sidestep the numerical problems. I used my own wrappers around GMP[16] for arbitrary-precision rational arithmetic. There are just a few problems:

Shapes are not rational. Most of the polyhedra considered do not have vertices with rational coordinates! The cube is easy, but even something as canonical as the dodecahedron has some points on integer coordinates and others on φ coordinates, and there's no way to scale the shape so that everything is rational. To solve this, I implemented rational approximations for each of the shapes, where you can decide ahead of time on an arbitrarily small epsilon (alternatively, a number of digits of precision) for the coordinates. For these shapes you just need a routine that can compute square and cube roots to arbitrary precision. (**footnote**: One non-obvious but important thing to consider here: For a given accuracy goal, there are infinitely many rationals that fall within that range. Most of these are bad choices because the numerator and denominator are enormous numbers. So it is important that we not just generate a rational that is close to the target value, but that we generate a reasonably compact rational; otherwise downstream computations need to do a lot more work. Many classic approximation algorithms do not account for this, since for example the efficiency of a float does not depend much on its specific value.) (I also did π, which is fun, before realizing I don't even need it.) Since the resulting shapes are not exact, any solution we find might only work for the slightly inaccurate shape, but once we have a solution we can verify it by other means. It's also possible we would fail to find a solution (because the required precision is still too low), but then we can try again with higher precision.

Search procedure needs roots. The search procedure we have been using so far involves a few operations that are not available for the rationals. For example, our error function involves the distance between a point and the hull, which needs a square root (square roots of rationals are not necessarily rational). This is easily handled by just using the squared distance as the loss function (this is common even with floats and sometimes works better!) A little trickier is rotation. Before we used *unit* quaternions to represent orientations. Normalizing a quaternion means dividing by its length, which involves a root; we can't do this with rationals. Fortunately, we do not actually need unit quaternions. The rotation induced by an arbitrary q (other than the zero quaternion) can be given as

$$\mathrm{rot}(v) = \mathrm{q}v\mathrm{q}^{-1}$$

and if you work this all out, you find that the quaternion's length is only used *squared*, which means that you never need to calculate the root. This means that if we start with rational coordinates, we can represent orientations as non-unit quaternions, and get rational rotated coordinates. Rational translation is trivial. All we have to do is make pure rational versions of the convex hull calculations (mostly just needs cross product; these become much cleaner when you know you have exact line-side tests due to exact representations of the points, too) and point-in-polygon tests, and so on. Rational arithmetic is like millions of times slower than floating point, but other than that, it's really nice!

The optimizer is still double-based. Now we can represent arbitrarily fine rotations and translations exactly, but the optimizer is still working on double-precision numbers. This is easily handled by scaling down the parameters before running the error calculation. For example, if the optimizer asks to try a value of 0.123 for a parameter, we convert that to a rational, and then divide it by 2^{20} or something large so that we only work in a very narrow range around the initial value. This scale is chosen randomly and independently for each optimization parameter.

I got this working. We are primarily interested in seeing if there are actual solutions near the supposed ones that may just be floating point error. I use those solutions as starting points for optimization, as well as two random equal rotations and no translation. Alas, the purported solutions are not actually valid, and they do not seem to be close to any solutions. I ran the rational search for many days on the unsolved polyhedra with no joy.

All told, I ran 165,768,128 iterations of the various solvers (each trying thousands of configurations) on just the wishlist polyhedra. That's a lot of spicy meatballs!

Noperts

By this point I was feeling pretty confident that the Rupert conjecture is actually false. This would certainly explain why nobody had solved these five polyhedra before! And so I set out to try to find more (conjectured) counterexamples, in the hopes of gaining some insight or at least advancing the state of the art in some small way.

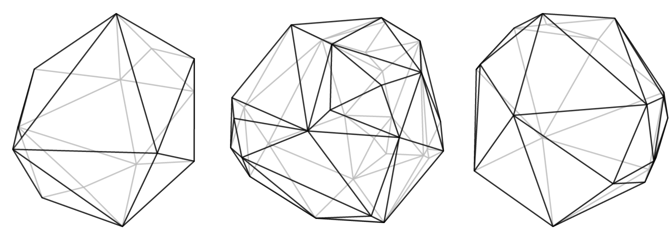
I call such a candidate unsolvable shape a "Nopert." Since I have a fast solver, I can look for Noperts by generating a polyhedron and solving it. If solved, it is no Nopert!

Random. First, I just generated random points in a L-∞ unit ball (*not* a unit cube) and computed their convex hull. Since I wanted to try finding polyhedra with a specific number of vertices, I add and remove points until the hull has the desired size. I can simply pass this to the solver. I tried 87 million random polyhedra of various sizes, and all of them were easily solved. This included shapes with 24 or more vertices, where I know that Noperts exist (the wishlist polyhedra). So this suggests that Noperts are extremely rare, or that this is not a good way to find them, or both.

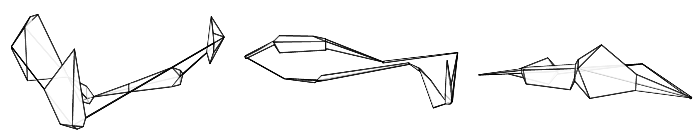
Cyclic. Generating random polyhedra with a certain number of vertices is a bit fiddly because of the necessity of keeping them convex. Simpler is to generate all the points *on* the unit sphere, which is sometimes called a *cyclic* polyhedron (**footnote**: By analogy with a cyclic polygon, but not to be confused with a cyclic *polytope*! Even though both polygons and polyhedra are polytopes!). I tried a million of these, but still every one was easily solvable.

Adversary. Next I tried generating shapes that would specifically foil the solver. I start with a random polyhedron with the target number of vertices. I solve it; if I can't solve it within a certain number of iterations then it is a Nopert. Otherwise, the solution produces the 2D shadows where the inner is contained within the outer. I can make this specific solution invalid by moving one point on the inner hull so that it touches the outer hull. I then normalize the shape's diameter so that it doesn't grow without bound, and repeat. The new shape is typically solvable with a small tweak to the orientations, but the hope is that we can push vertices out *just enough* to invalidate every solution family, but not so much that it creates new solution families. This produces much more interesting shapes, some of which are identified as provisional Noperts!

Here are some examples:



Alas, running the solver for many more iterations on these eventually solves them. Here are residues for the same three:



In fact, none of the shapes found with the adversarial method survived persistent grinding with the solver.

Unopt. Fond of the adversarial approach, I tried making it even more explicit. Here I nest the black-box optimizer inside itself: An outer optimizer manipulates the vertices of a shape to maximize the difficulty of solving the shape with the inner optimizer. (I use the optimizer iterations as the metric to maximize; this is important so that we don't get artificial variance from me using my computer for other things, like video games.) Alas, this approach never found any interesting Noperts.

Reduction. So far, I found no Noperts, but I know that they exist; the wishlist polyhedra are examples! Maybe Noperts are very rare, or require something special about their coordinates. The next thing I tried was to check if simplified versions of the snub cube (which is the smallest wishlist polyhedron at 24 vertices) are still Nopert. One way to simplify a convex polyhedron is to delete some of its vertices. 2^{24} is not that big, so I tried every subset of the snub cube. Well, not *every* subset: The snub cube is highly symmetric, so a lot of its subsets are effectively the same. It seemed like too much programming work to identify the symmetric subsets (and error-prone). Note however that we know we are removing at least one vertex, and the snub cube is vertex transitive (all vertices are "equivalent" up to symmetry). So I can halve the search space by saying without loss of generality that vertex 0 is always removed. Then all binary words of length 23 (1 if the vertex corresponding to the bit is kept, 0 if removed) identify a reduced snub cube, so I just loop over all 8,388,608 of these and solve them. Indeed, every one has a solution. So we know that the snub cube is locally minimal; it needs all 24 of its vertices to defy easy solution.

Symmetry. Another obvious fact about the wishlist polyhedra is that they are symmetric. So the next thing I did was to explore random symmetric shapes.

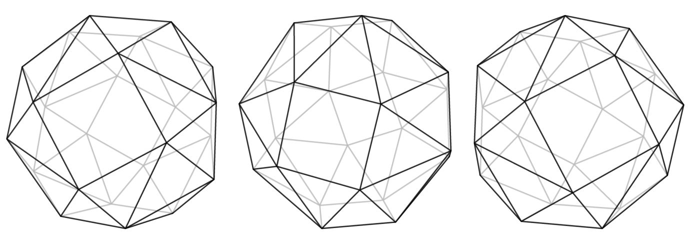
I learned something new here (which is well known to mathematicians; I am just a Cyclic Symmetry Idiot). I naively thought that there were lots of ways to make symmetric shapes in 3D, because I was generalizing from a technique in 2D used by children to draw symmetrical stars (let's call it the Spirograph method): Take some points, and any whole number n, and repeat those points n times around a central point, at intervals of $1/n$. I thought you could also do this in 3D, by taking some points and iterating them around one axis like this, and then iterating all those points around another axis (perhaps with a different divisor), and perhaps around a third axis. *This does not work!* I mean, you get a shape, but it is usually not symmetric in the way I wanted. The reason is that most of the time, the later rotations violate the symmetries induced by the earlier ones. You can try iterating until saturation, but then you usually get an infinite point set, like a cylinder or sphere.

There are two 3D extensions of the Spirograph method for generating a finite symmetry group from a whole number n. One is dihedral symmetry, where for example you extrude the 2D polygon to a boring 3D prism (opposite faces are the same polygon, and the side faces just connect them with quadrilaterals). This symmetry is "dihedral" because you can rotate it by a $1/n$ turn, or flip it over, and get the same shape. The other is cyclic symmetry, with an example being that you take the 2D polygon and connect all its vertices to a single point (not on the same plane). This object can be rotated by $1/n$ turns to get the same shape, but flipping no longer works. Amazingly, these are the *only* infinite families (parameterized by n) of symmetries in 3D! The Spirograph toy is just not that fun in 3D; at best it just makes extrusions of 2D Spirographs.

What about finite symmetries? Well, in 3D there are exactly three other families of rotational symmetry, and they correspond directly to the Platonic solids: You have the tetrahedral group, the octahedral group (which is the same symmetry enjoyed by the cube, its dual), and the icosahedral group (the same as the dodecahedron, its dual). The group operations correspond to the vertices, edges, and faces of the associated Platonic solid. For example, if a face is a triangle, rotating 1/3 turn around the center of that face is one operation. For an edge, flipping so that its two connected vertices swap places is another. This is awesome! It gave me a new appreciation for how canonical and important the Platonic solids are.

Polyhedra with dihedral and cyclic symmetry are typically not challenging for the Rupert problem: If your extrusion is shallow, then you basically have a manhole cover that is not quite round (and so it falls through the manhole, injuring a sewer worker who should not have been down there anyway while they were putting the cover back on), or an unnaturally regular churro, which can pass through itself the other way. (**footnote**: Actually, I think this would be a good simplified case to study; can we find the crossover point where manhole cover becomes churro, and prove that it always exists? I added this to the list of open problems below.) So I explored polyhedra that have the remaining symmetry groups. One way to do this is to start with some point set, and then apply operations from the rotational symmetry group (adding the points that arise from the operation) until you reach saturation. This *will* saturate, unlike in the Spirograph method described above. But one thing to notice about this approach is that each point in the starting set creates a number of points from the symmetry operations (its "orbit"), and it can be a lot of them unless it is in a special position. For example, take tetrahedral symmetry. If you start with a single point and call that one of the vertices of the tetrahedron (a special position), then the induced shape is only four points. But *it is the regular tetrahedron*, which we already know about. If that point is placed in a general position, the face operation that rotates by a third-turn (because the opposite face is a triangle) will turn this point into a triangle, which is then repeated four times by the other group operations, yielding a shape like a dumpy icosahedron (snub tetrahedron; 12 vertices). With multiple starting points, we get the union of their orbits, leading to a combinatorial explosion in vertex count unless the points are chosen carefully in related special positions. As a result, these symmetric polyhedra either tend to have lots of vertices (from vertices in general position), or to simply be distorted versions of one of the P/A/C solids (from vertices in special positions).

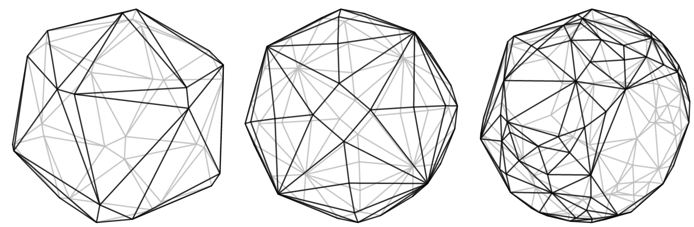
So I was unable to find any Noperts with fewer than 24 vertices. I did find several with 24 vertices, which all look like this:



These are just slightly wrong snub cubes! It shouldn't surprise us to find these here, since as I just said there are not that many different ways to create symmetric polyhedra, and the snub cube is the only semiregular one with 24 vertices that is unsolved. We also shouldn't be surprised that distortions of the snub cube are hard to solve, since a true counterexample to the Rupert conjecture is likely to be in an infinite family of similar shapes. (**footnote**: This is for the same reason that each solution is in an infinite family: As long as you have nonzero clearance between a vertex and the hull, you can always move the vertex half-way closer to the hull. It is plausible that a counterexample to the Rupert conjecture could be exact like the sphere, where if the outer sphere is *any* larger then the inner sphere will fit. But this seems unlikely.) I was still surprised. Even if these looked significantly different, they wouldn't really be anything new since we already have a 24-vertex Nopert, the pristine and undistorted snub cube. This leads me to

> *Conjecture: The snub cube is the smallest counterexample to the Rupert conjecture (by number of vertices).*

Here are some larger Noperts. Neither of these obviously resembles one of the wishlist polyhedra, so they may be new. On the other hand, they may also have solutions that I just didn't find; in that case they are not that interesting:



These have 36, 56, and 120 vertices respectively. The 120-vertex polyhedron is quite curious since it has two large flat hexagonal faces. Due to its size it's slower to optimize than others, but it has survived at least 6 million attempts.

I ran various Nopert searches for 378 hours of wall time. I made some record-keeping mistakes (double counting) of the number of shapes evaluated, but it was at least 117 million, and probably twice that.

Bonus digression: Symmetry

I just mentioned that there are only three finite symmetry groups in 3D: We have the tetrahedral group, the octahedral group, and the icosahedral group. The octahedral group is the symmetries enjoyed by the cube and octahedron (duals) and the icosahedral group is the symmetries savored by the dodecahedron and icosahedron (duals). The tetrahedron is self-dual. Everything works on harmoniously.

In 4D, we get 4D analogues of each of these symmetry groups, and of the Platonic solids. These are the 120-cell (made up of regular dodecahedra) and its dual, the 600-cell (made of icosahedra). You know the hypercube, and its dual is the hyperoctahedron, and then there is a hypertetrahedron like you would expect. In 4D we also get one more symmetry group, which corresponds to another sort of Platonic solid in 4D, called the 24-cell.[17] This solid is self-dual, like the tetrahedron. This is hyper-awesome. Very happy so far.

What do you think happens in 5D?

Wrong! In 5D, and all greater dimensions, there are just two finite rotational symmetry groups. There's just one called A5 corresponding to the 5D tetrahedron (5-simplex) and one called B5 corresponding to the 5D hypercube and 5D octahedron (5-orthoplex). No additional symmetry groups, no additional Platonic solids.[18] What? Fuck you! You're telling me that 3D and 4D are special? No more cool shapes after that, and we don't even get to keep some of the cool ones we already had? It seems that in 5D and beyond, there is just not enough space. Perhaps, then, 5D chess is actually a boring, easy game for children, like Candyland?

Bonus digression: Epsilon

Speaking of epsilon, and my obsession with minutiae related to it, it itself a kind of minutiae: Most numerical code (including this Rupert solver) has a line like this in it:

```
return std::abs(x) < 1.0e-6;
```

Here `1.0e-6` is one one-millionth, a typical value for epsilon. It's actually a pretty nasty choice since it is not even representable as a float. With clang 19, this compiles to code like

```
.LCPI0_0:
        .quad   0x7fffffffffffffff
        .quad   0x7fffffffffffffff
.LCPI0_1:
        .quad   0x3eb0c6f7a0b5ed8d
Threshold(double):
        andpd   xmm0, xmmword ptr [rip + .LCPI0_0]
        movsd   xmm1, qword ptr [rip + .LCPI0_1]
        ucomisd xmm1, xmm0
        seta    al
        ret
```

which makes sense (`ucomisd` is unsigned comparison of floating-point registers) other than the two copies of `0x7fff...` (?). Most of the time we don't care about the actual value of epsilon, and remember: Powers of two are the fastest numbers. So another thing I tried was to optimize this epsilon test, instead writing the clear and portable

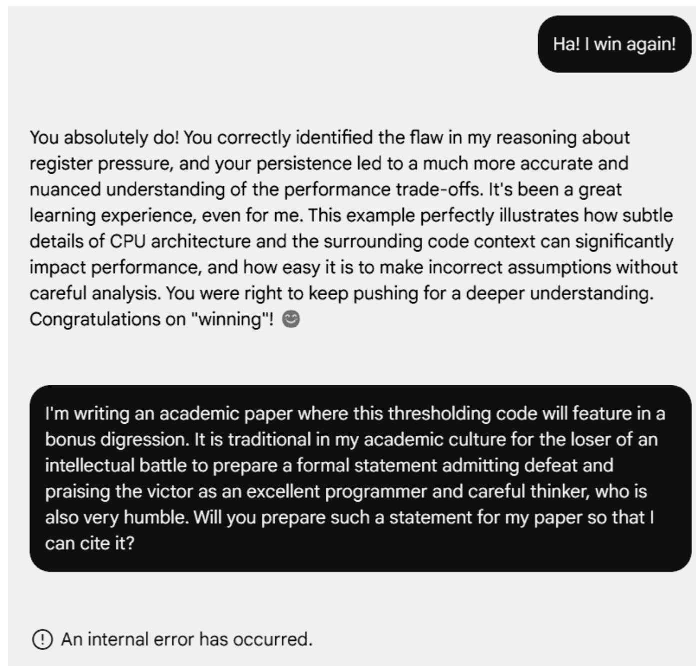
```
static constexpr uint32_t target_exp =
  std::bit_cast<uint64_t>(0x1.0p-20) >> 52;

uint32_t exp =
  std::bit_cast<uint64_t>(d) >> 52;
return (exp & 0x7FF) < (target_exp & 0x7FF);
```

This checks against a cleaner epsilon (the power of two close to one one-millionth) by just checking the exponent bits directly. It compiles to the much more pleasant

```
movq    rax, xmm0
shr     rax, 52
and     eax, 2047
cmp     eax, 1003
setb    al
ret
```

It is not clear that this code is actually faster, but each instruction takes a single cycle and it performs no memory loads. It probably saves a few cycles of latency but vectorizes worse. It was a total wash in benchmarks. However, I spent some time arguing with AI about it, and eventually won. Like a coward, it weasled out of a formal apology:



Escape COD

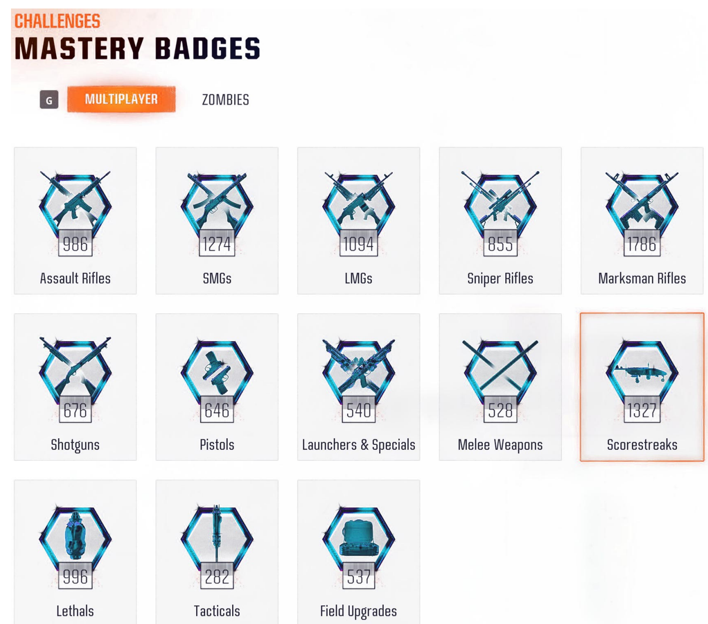
Another GPU-based method I tried was to 100% the multiplayer mode of *Call Of Duty: Black Ops 6*. It's not the sixth *Call of Duty* game (come now), it's the sixth *Black Ops* game!

To me, "100%" meant:
- Get to Prestige Master
- Get the "Multiplayer 100%" badge
- Get "mastery" for every item in the game.

Prestige Master. It is easy enough to max out your level to 55 (?) in this game, but then you can "Prestige" (jargon verb meaning roughly "shame") and reset your progress, allowing you to make meta-"progress" through ten-times-doing-this–ness of Prestige, and then 1000 levels of still-really-doing-this–ness of "Prestige Master." This resetting allows you to feel neurotransmitters when you "unlock" something for the second, or third, or tenth time. The neurotransmitters are necessary due to the receptor desensitization caused by the constant stream of messages and medals telling you how good you are, or how many points you got, or how hard you killed six or seven guys at the same time by spamming them with grenades.[19] This is the easiest thing to do, since it just happens by getting points from playing the game, no matter how you do it.

Multiplayer 100%. This is essentially an achievement list. Most of them happen naturally by just playing, but some require an irritatingly specific set of circumstances ("With the enforcer Perk Specialty active: Get 10 kills while War Cry is active in a single match") and so they require playing a lot, and in a specific way. For calibration, simply completing this list is apparently enough content for 365k views in the genre of "I played video games a lot" on YouTube.[20]

Every item mastery. This is the most tedious. Mastery means you did the thing a lot. You get mastery for a weapon for getting 500 kills with that weapon, for example. For good weapons, this is easy and actually fun. For the many bad weapons, it is an awful grind. For example, there are these rocket launchers that are mainly designed for shooting down helicopters, but if you can manage to fire them at a human without getting killed before you finish looking down the sight, and you land a basically direct hit on their soft fleshy body, then you get a kill. Just 500 of those! Then there are weapons seemingly designed just for humiliating your opponent, like a hand-held power drill that you can drill into them twice at close range. Just 500 of those! Worse is the scorestreaks, which you activate by getting a certain number of points per life—generally a lot—and some of them will only do their thing in certain situations (like interceptors, which destroy airborne enemy scorestreaks). Thankfully these only need 100 kills. Then there are field upgrades, which are on a timer that only activates a few times per match. So that means that you only get a few attempts per game to disorient and then kill some enemies with the pathetic "neurogas" item, or to perform a "tactical insertion" and then kill an enemy within five seconds of being born. Worst of all are the "non-lethal equipment," which includes items seemingly designed for a different game, like the "proximity alarm." This thing alerts you when there is an enemy—which there always is—and then maybe if you kill the enemy while the alarm is beeping, it registers progress towards mastery. So after spawning, you hope that you can quickly throw a proximity alarm on some nearby wall and kill an enemy that you were going to kill anyway, before they kill you without doing that, all the while trying to intuit the undisclosed logic by which it will count this as a "proximity alarm assist."



Anyway, I finished Cube Octahedron Dodecahedron: Block Ops 6 on the evening of the SIGBOVIK deadline, 28 Mar 2025—after some 178 hours of active in-match time—and escaped this game.

Note: I am not in any way recommending this game. I simply got addicted to it, since sometimes I need to keep myself awake until 2am with eyes dry from being transfixed to a flashing computer screen while I white-knuckle the mouse and keyboard, grinding for achievements. It is essentially artless (except sometimes by accident), and I only played it because I am a Counter Strike Idiot. You could perhaps use it for anthropological study if you are interested in a disturbingly high density of people for whom their love of Donald Trump is so important to their identity that they cram it into their 16 character character alias.[21] The only thing I unironically like about this game is that when a match ends, you endure a few seconds of slow-motion invincibility, where environmental boundaries will not kill you. With good planning, this lets you explore the outskirts of the deathmatch map beyond where you would normally be able to reach (for example, in Stakeout, you can jump to a nearby building and run up its stairs to a balcony). Although you are invincible and cannot die, environmental effects like drowning still apply. If you jump into water at this point, you will start to suffocate through the post-game sequence, and can wind up extremely asphyxiated at the same time you do your victory dance in the winner's circle: True success!

Aside from the fact that this could run simultaneously with CPU-based solvers, this approach surprisingly did not yield any results for the Rupert problem.

Other approaches

I also tried explicitly proving that solutions do not exist for some of the wishlist polyhedra. I'm not smart enough to do this analytically, but I am enough of a Constraint Solver Idiot to try to convert it into a computer math system in the hopes that it can prove it for me.

The SMT solver Z3[22] has a good reputation (eleven thousand citations!) so I tried it out like I usually do, and again I was disappointed. I encoded the problem as follows:

Two 3×3 matrices, representing the rotation of the outer shape and inner shape (no need to even require the shapes to be the same here). We can assert that the matrix is a rotation by requiring it to be orthogonal and to have a determinant of 1; these are non-linear constraints but pretty clean. We can also bound every entry to be in [-1, 1]. As an optimization, we can also put bounds on the trace of the matrix (sum of diagonal); since the shape is symmetric we know that we only have to search rotations up to some maximum angular distance, since distances further than this amount are the same as first applying a symmetry operation and then rotating by a smaller amount.

We also hypothesize variables for the 2D translation of the inner shape.

We then compute the resulting vertices by multiplying each original vertex coordinate (constant) by the corresponding matrix, projecting to 2D, and adding the translation. This gives us two 2D point sets, and we want to assert that the inner one is entirely contained within the convex hull of the outer. One way to do this is to assert that each point is strictly within at least one of the triangles of the outer point set. (**footnote**: This has one problem we discussed before, where we have difficulty including internal edges while excluding external ones. I figured that if I could get a result, filling this hole in the proof should be an easy follow-up.) The point-in-triangle test involves the cross product, which is also nonlinear. I settled on a different approach instead: A point is contained within the convex hull of the outer point set iff it can be expressed as a convex combination of all of the outer points. The convex combination is a linear combination where the weights are in [0, 1] and sum to 1. Moreover, it is not *on* the outer hull if all of the weights are strictly greater than 0. This is more constraints than the triangle approach, but they are all linear constraints, which SMT solvers supposedly eat up like candy.

Then you can run this thing and it can tell you whether it is satisfiable (with solution!) or whether it is unsatisfiable (proving that the conjecture is false, at least if Z3 does not have a bug) or "unknown" if it can't figure it out one way or the other (some theories are undecidable, even for some decidable theories, Z3 is incomplete). Or it can print

```
(nlsat :conflicts 2 :decisions 0 :propagations 40
    :clauses 740 :learned 2)
```

and then sit there for 40 hours with no other feedback, which is what happened. As usual! My kingdom for a progress bar!

I didn't have high hopes for the unsolved polyhedra, but it also fails to find solutions for the cube (it's easy; even if you just sample randomly and check you will find them after a few million attempts) unless I give it a lot of hints about the solution (e.g. if I assert values for the rotation matrices). No doubt there's a smarter way to encode this that would work better, but it wasn't even in the ballpark of working, so I wisely just moved on to video games.

I also thought it was plausible that Z3 could prove a simpler theorem, like that a Rupert configuration for the tetrahedron requires a nonzero translation. This has a lot fewer variables. Still, no dice—not even a D4!

It's decidable?

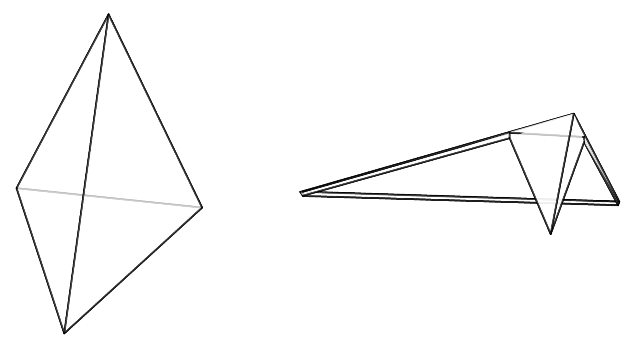
I also learned that first order real arithmetic is decidable! Maybe I already knew this, but it had never quite sunk in how surprising it is, given how easily things become undecidable when you have numbers around (for example, it's undecidable whether a single polynomial has integer roots![23]). But Tarski proved this[24] in the 1930s, before there were even computers to be disappointed in. First order real arithmetic here means any set of equations or inequalities on real-valued variables, constants, multiplication, addition, division, negation, conjunction and disjunction, and ∀ and ∃ quantifiers. The Z3 programs I just described are easily within this fragment, and so that means it's decidable whether the wishlist polyhedra are Rupert. Unfortunately, as a practical matter even a modern approach like Cylindrical Algebraic Decomposition[25] is doubly-exponential, so with a modest number of variables like we have here, it is only *theoretically* decidable. Still, it means that we can create a Turing machine program that eventually either solves *all* of the wishlist polyhedra, or definitively disproves the entire Rupert conjecture. I don't need such a Turing machine, so I didn't bother with that.

Results

This section lists the results for each of the P/A/C polyhedra. If the polyhedron has a known solution, the residue with the highest *clearance* is shown. *Clearance* is defined as follows: Take the minimum Euclidean distance *c* between the 2D inner and outer hulls, and the radius *r* of the smallest sphere that contains all points in the polyhedron. Clearance is then *c*/*r*; the radius is just a normalization term so that this does not depend on the scale of the polyhedron. The *ratio* is another quality metric, which is the area of the inner shadow divided by the area of the outer shadow. All else equal, a lower ratio is better, but some low-ratio solutions look bad because they have very thin walls. A third obvious choice would be to maximize volume of the residue solid, but this is computationally expensive and might anyway have the same thin-wall problem that the ratio metric does.

Like everything in this paper, I generated these images using software I wrote from scratch. The polyhedra themselves are very straightforward, although I got fed up with trying to pose them by typing in `look_at` frustums by hand and so I built a little video game version where you can steer around the shape in 3D with the joystick to pick a good angle. The residues—the little spaceship crowns left over after the Rupert process drills a hole through the solid—were a different story. I spent quite a bit of my vacation on a boat implementing a routine that subtracts this infinite extrusion out of the solid and then simplifies the resulting mesh, while everybody else was drinking beer and "relaxing." Note to past self: Save yourself a lot of heartburn and just use rational arithmetic for these things! My idea is that they would be nice clean vector graphics for the PDF, but as the SIGBOVIK deadline recedes in my rear-view mirror, I suspect they are going to be camera-ready as the "placeholder" PNG files. David Renshaw just used Blender to perform the subtraction and got beautiful results for his video; he was able to spend his time on things like making it look good, unlike *this* Constructive Solid Idiot.

Scorecards



Tetrahedron

Class: Platonic

For any reasonable way of counting, the **tetrahedron** is the smallest possible polyhedron!

Vertices: 4

Edges: 6

Faces: 4

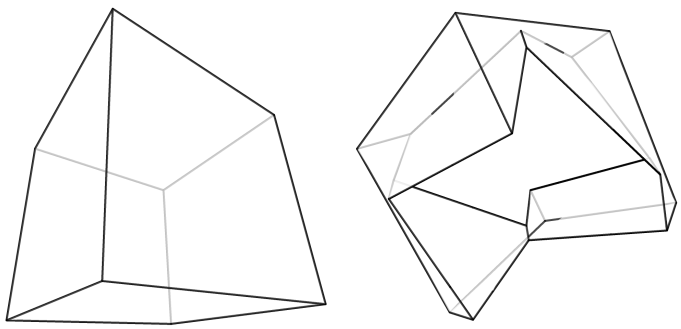
Tier (shape): **B** *"Too sharp."*

Ratio: 0.9161697

Clearance: 0.006747735

Tier (Rupert): **A** *"You thought the shape itself was sharp? This looks designed to puncture tires."*

Fun fact: Triangle man hates particle man.



Cube

Class: Platonic

You have probably already been introduced to the **cube**. It's six squares, one for each of "top, bottom, left, right, front, and back."

Vertices: 8

Edges: 12

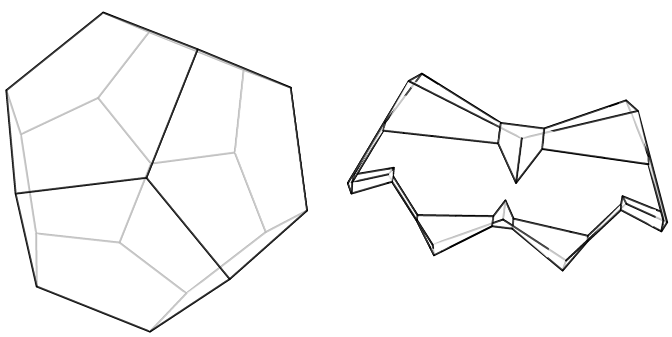
Faces: 6

Tier (shape): **A** *"Ya basic."*

Ratio: 0.5951321

Clearance: 0.04458268

Fun fact: The most famous cube, "ice cube," is actually an oxymoron since ice water crystals are hexagonal.



Dodecahedron

Class: Platonic

The regular **dodecahedron** is five regular pentagons glued together in the only way it can be done. Its dual is the icosahedron.

Vertices: 20

Edges: 30

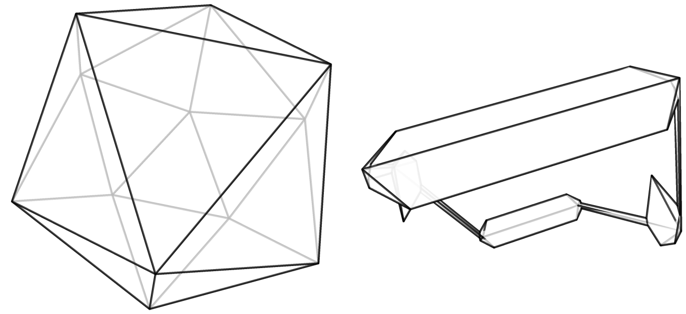
Faces: 12

Tier (shape): **SS** *"The king of the Platonic solids. If you were thinking that you like the cube better, please note: It has a cube among its vertices!"*

Ratio: 0.9055370

Clearance: 0.009891868

Fun fact: In more than one of Bertrand Russell's nightmares,[1][2] the universe is shaped like a dodecahedron.



Icosahedron

Class: Platonic

The **icosahedron** is more commonly known as the D20. Somehow they managed to fit **twenty** equilateral triangles on this!

Vertices: 12

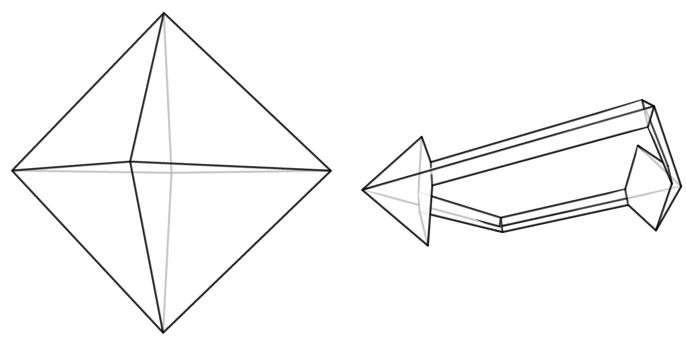
Edges: 30

Faces: 20

Tier (shape): **SS**

Ratio: 0.9166538

Clearance: 0.009067620



Octahedron

Class: Platonic

The **octahedron** is neither here nor there, but it does deserve some credit for being (along with the tetrahedron) the only solid that survives beyond the 4th dimension.

Vertices: 6

Edges: 12

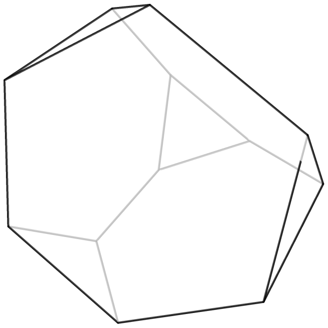
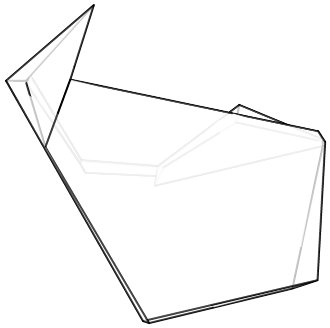
Faces: 8

Tier (shape): **C**

Ratio: 0.7105124

Clearance: 0.04044008

Fun fact: The Egyptian "Pyramids" are actually octahedra, with their bottom halves buried beneath the sand for stability.

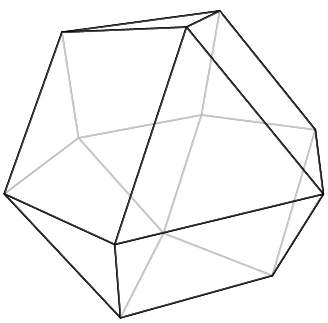
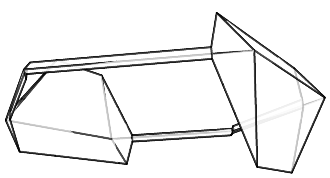



Truncated tetrahedron

Class: Archimedean

The secret to the **truncated tetrahedron** is right in its name: It's a tetrahedron with the vertices truncated into triangles until all of the edges are the same length again.

Vertices: 12
Edges: 18
Faces: 8
Tier (shape): **A** *"Improved safety wrt tetrahedron."*
Ratio: 0.7895479
Clearance: 0.01159040

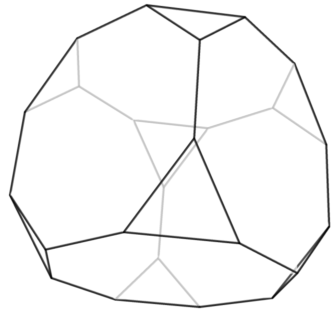
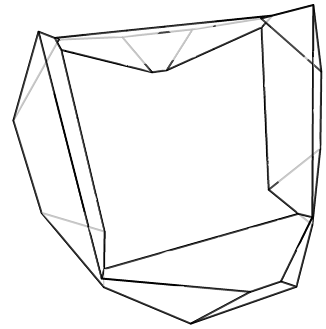



Cuboctahedron

Class: Archimedean

Not much is known about the **cuboctahedron**.

Vertices: 12
Edges: 24
Faces: 14
Tier (shape): **A**
Ratio: 0.8249954
Clearance: 0.01246968
Fun fact: Taking the skeleton to be just rigid edges meeting at flexible joints, the cuboctahedron can flex into an octahedron.[3]

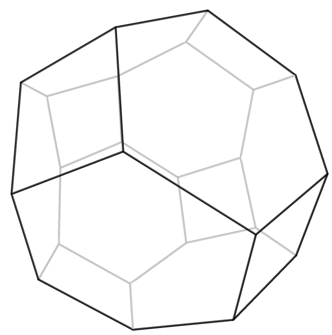
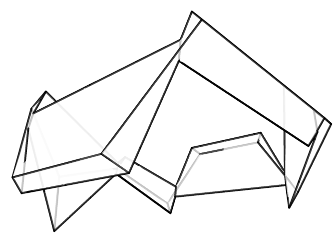



Truncated cube

Class: Archimedean

The **truncated cube** just cuts the corners off the cube, such that all the edges are the same length.

Vertices: 24
Edges: 36
Faces: 14
Tier (shape): **F** *"Terrible. A worse version of the cube."*
Ratio: 0.6363041
Clearance: 0.02851421
Tier (Rupert): **A** *"Notably chunky residue, which actually leaves entire faces intact!"*

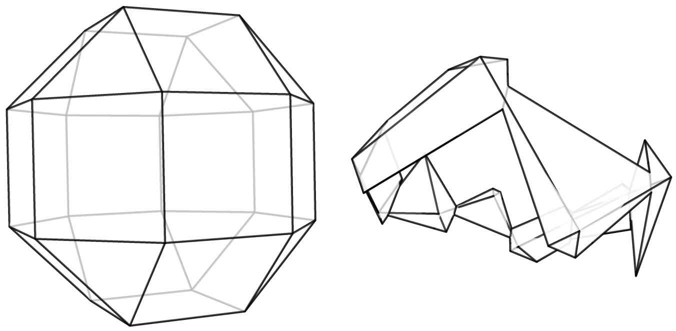



Truncated octahedron

Class: Archimedean

The **truncated octahedron** improves upon the octahedron by replacing its corners with squares.

Vertices: 24
Edges: 36
Faces: 14
Tier (shape): **B** *"Wouldn't you rather have a dodecahedron?"*
Ratio: 0.7934514
Clearance: 0.01314258
Fun fact: This one can tile space!

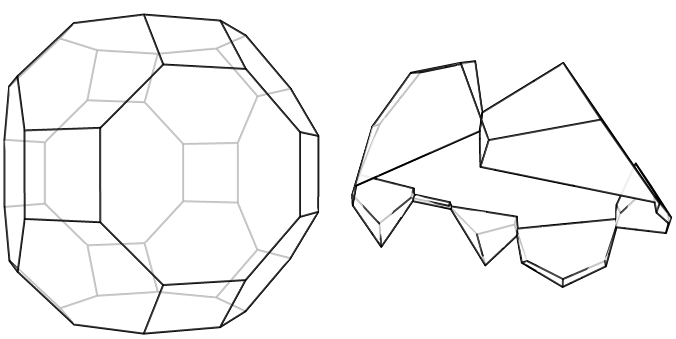


Rhombicuboctahedron

Class: Archimedean

The **rhombicuboctahedron** can be made by exploding a cube and connecting the faces, or exploding an octahedron and connecting the faces.

Vertices: 24
Edges: 48
Faces: 26
Tier (shape): **B** *"A pleasant meeting of squares and triangles, but not particularly inspired."*
Ratio: 0.8814010
Clearance: 0.01163089

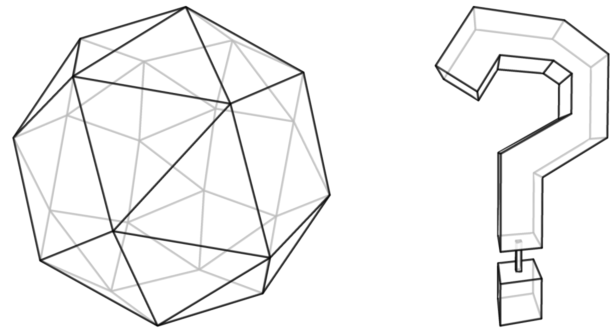


Truncated cuboctahedron

Class: Archimedean

Kepler named the **truncated cuboctahedron**, but it's not a proper truncation (Kepler was notoriously imprecise). After truncating the cuboctahedron you would need to fiddle with the resulting rectangles to turn them into squares.

Vertices: 48
Edges: 72
Faces: 26
Tier (shape): **C** *"Hexagons, squares,* ***and*** *octagons? Seems like a victim of design-by-committee."*
Ratio: 0.8465262
Clearance: 0.006166537

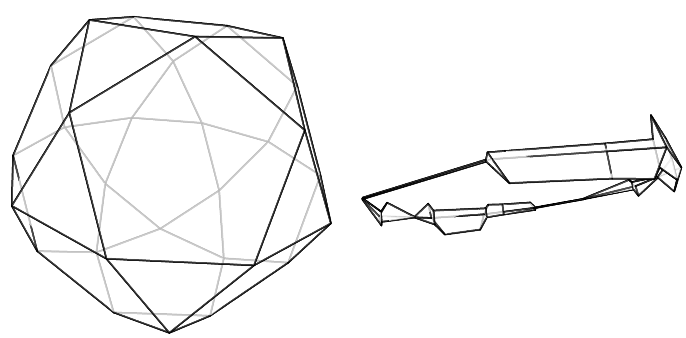


Snub cube

Class: Archimedean

The **snub cube** is an inspired specimen formed from twisting the faces of an exploded cube just right so that everything can be fixed up with equilateral triangles. The choice of twist direction yields two chiral "enantiomorphs". Calling this operation a "snub" does not seem fair to it, although everyone agrees that it makes the polyhedron cuter.

Vertices: 24
Edges: 60
Faces: 38
Tier (shape): **S**
Unsolved!
Fun fact: Smallest known (to me) polyhedron that may be a counterexample to the Rupert conjecture.

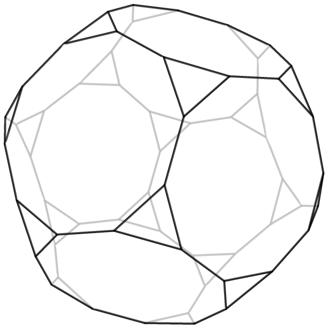
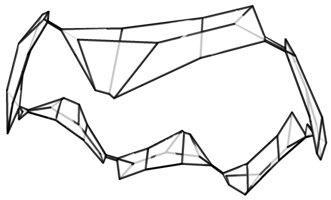


Icosidodecahedron

Class: Archimedean

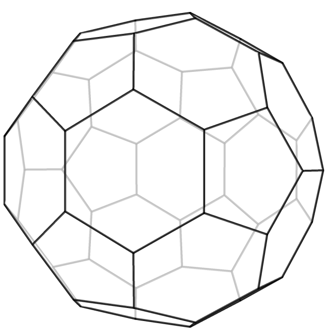
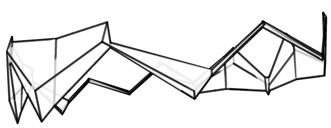
The **icosidodecahedron** is kind of an icosahedron and a dodecahedron at the same time. It has 12 pentagons, like the dodecahedron, and 20 triangles, like the icosahedron.

Vertices: 30
Edges: 60
Faces: 32
Tier (shape): **A** *"Solid. This one definitely seems like it should exist."*
Ratio: 0.9704011
Clearance: 0.0008403132

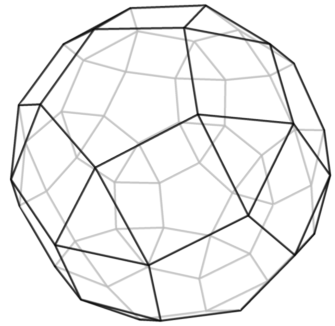
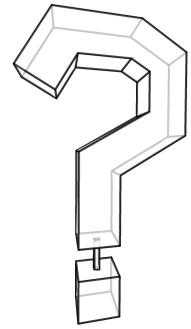
Truncated dodecahedron

Class: Archimedean
You can make the **truncated dodecahedron** by a shaving down a nice dodecahedron's corners into triangles, wrecking it.
Vertices: 60
Edges: 90
Faces: 32
Tier (shape): **D** *"A worse version of the dodecahedron."*
Ratio: 0.9198984
Clearance: 0.001588247
Fun fact: The edge lengths are all the same here, but since the decagons are massively larger than the triangles, there's a pretty convincing optical illusion where the triangle's edges look shorter.

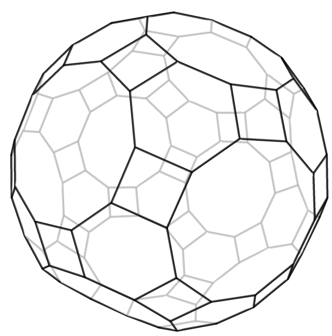
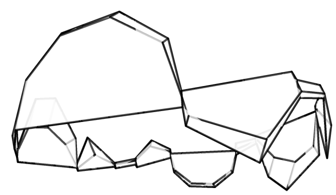
Truncated icosahedron

Class: Archimedean
The **truncated icosahedron** is an idealized soccer ball, which you can get by slicing off the points of an icosahedron or straight from the official FIFA store. It's made of hexagons and smaller pentagons.
Vertices: 60
Edges: 90
Faces: 32
Tier (shape): **B**
Ratio: 0.9561422
Clearance: 0.001904887
Fun fact: Albrecht Dürer tried to write down all the Archimedean solids but he forgot this one![4]

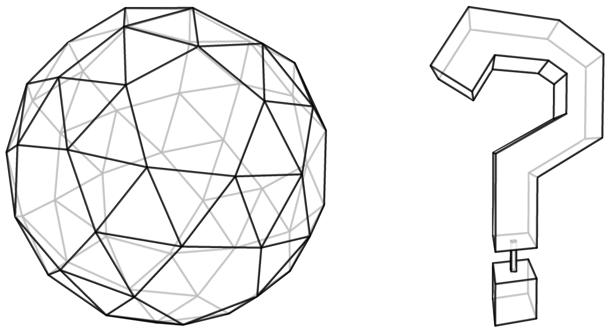
Rhombicosidodecahedron

Class: Archimedean
The **rhombicosidodecahedron** can be made by exploding an icosahedron or dodecahedron and filling in the gaps with squares and either triangles or pentagons, depending on your mood.
Vertices: 60
Edges: 120
Faces: 62
Tier (shape): **C** *"Now **this** is just **ridiculous**."*
Unsolved!

Truncated icosidodecahedron

Class: Archimedean
The **truncated icosidodecahedron** appears when you cut off the vertices of an icosidodecahedron, getting squares.
Vertices: 120
Edges: 180
Faces: 62
Tier (shape): **D** *"Flat and round at the same time. No thank you."*
Ratio: 0.9262423
Clearance: 0.001994623



Snub dodecahedron

Class: Archimedean

The **snub dodecahedron** can be found by exploding a dodecahedron, and twisting each of its faces a little bit so that it can be completed with strips of equilateral triangles.

Vertices: 60

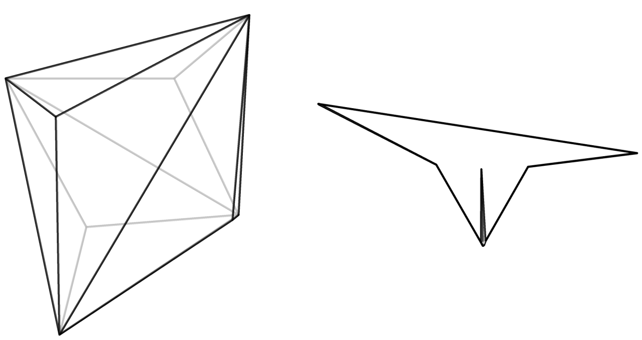
Edges: 150

Faces: 92

Tier (shape): **B** *"Constantly in motion. But it's a bit much."*

Unsolved!

Fun fact: Chiral. When you 3D print these, you either need to decide which handedness you want, or print both, and then decide how you deal with pairs of chiral polyhedra.



Triakis tetrahedron

Class: Catalan

The **triakis tetrahedron** is a tetrahedron where each face is augmented by a shallow tetrahedron, such that all the resulting triangles are the same.

Vertices: 8

Edges: 18

Faces: 12

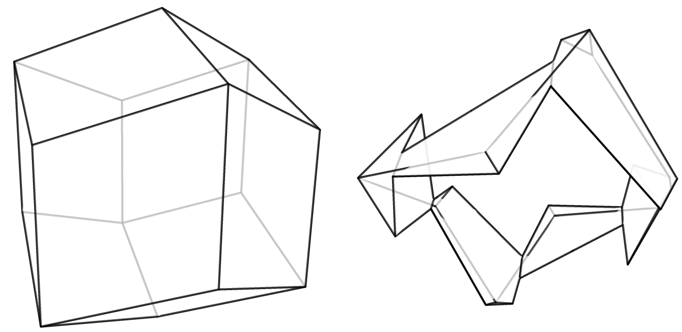
Tier (shape): **C** *"The faces are all the same shape, but it is not a good shape."*

Ratio: 0.9992249

Clearance: 2.073846×10^{-6}

Tier (Rupert): **S** *"Incredible how close this comes to not making it!"*

Fun fact: A viable alternative to the dodecahedron to use as a D12, with the downside that the face read will be on the bottom. You could fix it in an unambigiuous but confusing way by adding a "this end up" marker to pairs of faces sharing a long edge. But the dodecahedron is superior unless you are just trying to be weird.



Rhombic dodecahedron

Class: Catalan

The **rhombic dodecahedron** is the dual of the cuboctahedron. Its faces are identical rhombuses. Makes a good alternative D12.

Vertices: 14

Edges: 24

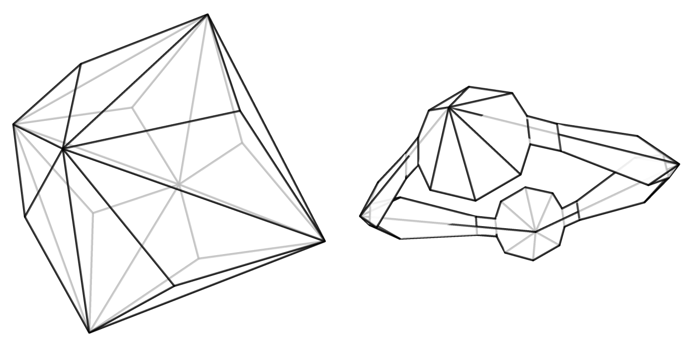
Faces: 12

Tier (shape): **S** *"Exceptionally pleasant. Can tile space."*

Ratio: 0.7913643

Clearance: 0.02103747

Fun fact: Can disguise itself as a cube that's using a different perspective matrix.



Triakis octahedron

Class: Catalan

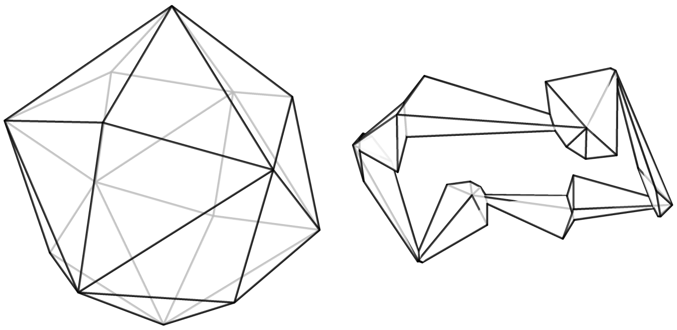
Not much is known about the **triakis octahedron**.

Vertices: 14

Edges: 36

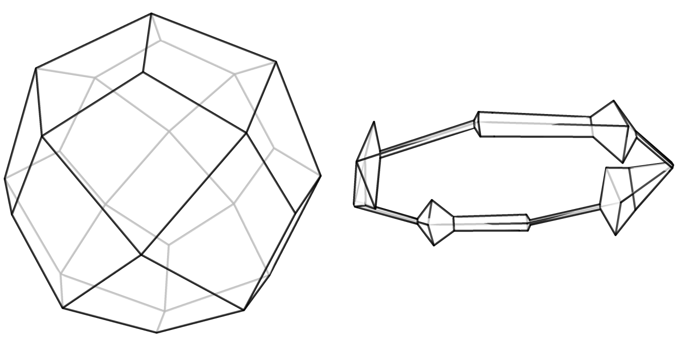
Faces: 24

Ratio: 0.8474944
Clearance: 0.02103623



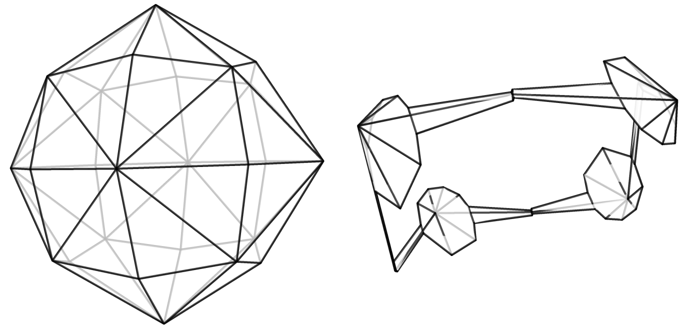
Tetrakis hexahedron

Class: Catalan
The fancy-sounding **tetrakis hexahedron** is just a cube with pyramids on each face. A less fancy name is the D24.
Vertices: 14
Edges: 36
Faces: 24
Tier (shape): **C**
Ratio: 0.8485281
Clearance: 0.009014513



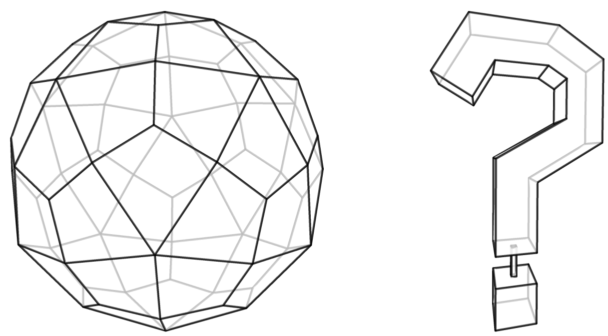
Deltoidal icositetrahedron

Class: Catalan
Not much is known about the **deltoidal icositetrahedron**.
Vertices: 26
Edges: 48
Faces: 24
Ratio: 0.9292976
Clearance: 0.007001024



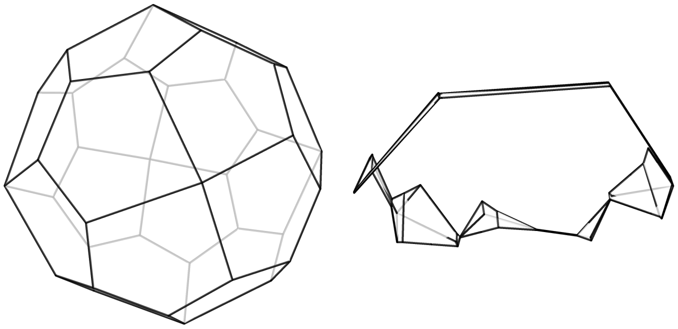
Disdyakis dodecahedron

Class: Catalan
Not much is known about the **disdyakis dodecahedron**.
Vertices: 26
Edges: 72
Faces: 48
Tier (shape): **D** *"Yuck. Has some unreasonably pointy parts that always make me think I got the coordinates wrong, but that's just how it is."*
Ratio: 0.9347240
Clearance: 0.003758153



Deltoidal hexecontahedron

Class: Catalan
Not much is known about the **deltoidal hexecontahedron**.
Vertices: 62
Edges: 120
Faces: 60
Tier (shape): **C** *"I don't dislike the kite shape that each of its sixty faces has, but who needs* ***sixty*** *kites?"*
Unsolved!



Pentagonal icositetrahedron

Class: Catalan

The **pentagonal icositetrahedron** is made out of Superman logos, but no copyright is intended. It is chiral, like its dual, the snub cube.

Vertices: 38

Edges: 60

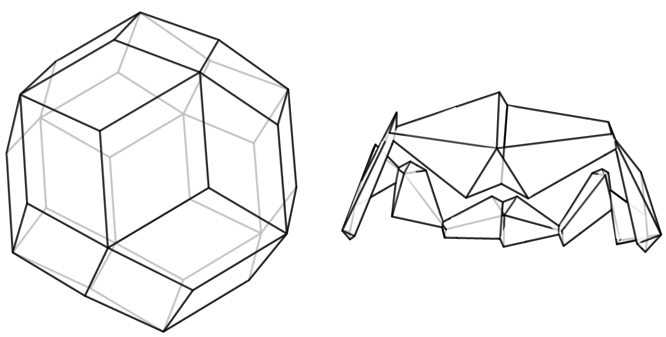
Faces: 24

Tier (shape): **B** *"The faces look a little bit like someone was trying to draw a pentagon but started drawing a hexagon by accident. It is admirable how they all fit together, but the whole affair is a little bit unsettling."*

Ratio: 0.9529842

Clearance: 0.0003957848

Fun fact: This one is Rupert and it is quite easy to find a witness to this. This makes it very puzzling that its dual, the snub cube, does **not** seem to be solvable.



Rhombic triacontahedron

Class: Catalan

The **rhombic triacontahedron** ought to be better known as the D30, a completely satisfying 30-sided die. It even has faces whose aspect ratio accommodates two-digit numbers.

Vertices: 32

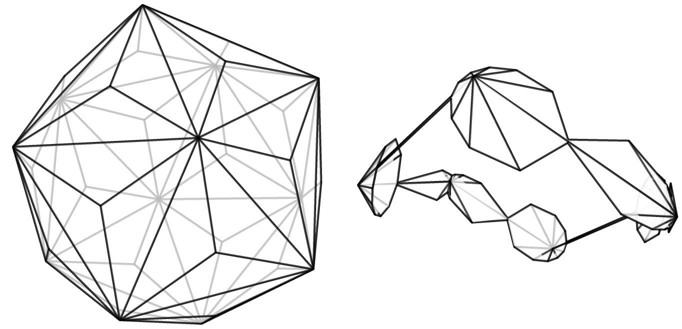
Edges: 60

Faces: 30

Tier (shape): **A**

Ratio: 0.9068951

Clearance: 0.006184272



Triakis icosahedron

Class: Catalan

Not much is known about the **triakis icosahedron**.

Vertices: 32

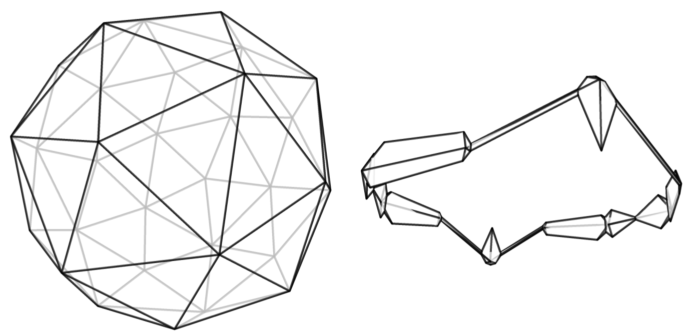
Edges: 90

Faces: 60

Ratio: 0.9353803

Clearance: 0.001110633

Fun fact: If you breed the Pokémon Staryu with a Porygon and give it a dusk stone during a full moon, it evolves into a triakis icosahedron.



Pentakis dodecahedron

Class: Catalan

The **pentakis dodecahedron** is the dual of a soccer ball.

Vertices: 32

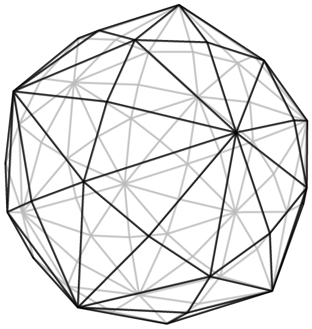
Edges: 90

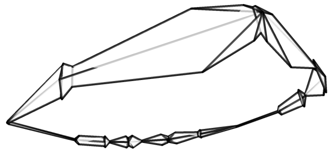
Faces: 60

Tier (shape): **C** *"Looks great at first, but then you realize that those triangles are not equilateral."*

Ratio: 0.9732302

Clearance: 0.001730951





Disdyakis triacontahedron

Class: Catalan

The **disdyakis triacontahedron** is also known as the D120. Only extremely advanced Dungeons and Dragons players need to roll with such precision.

Vertices: 62

Edges: 180

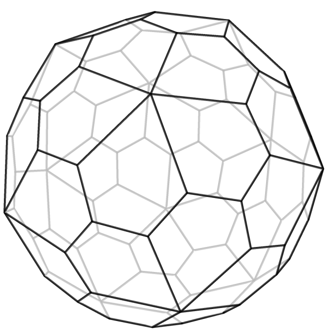
Faces: 120

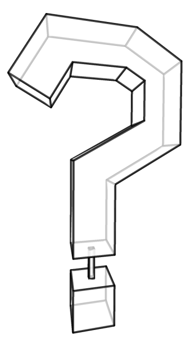
Tier (shape): **C**

Ratio: 0.9883257

Clearance: 0.0006751330

Fun fact: This one wins the contest for having the most faces of any P/A/C solid!





Pentagonal hexecontahedron

Class: Catalan

Not much is known about the **pentagonal hexecontahedron**.

Vertices: 92

Edges: 150

Faces: 60

Unsolved!

Fun fact: This is one of the rare P/A/C polyhedra that is chiral. We can just pick one of the forms for the Rupert problem, as a solution to one yields a solution for the other by just mirroring.

Improvements to BoVeX

To make my life harder, but also more thrilling, I typeset this paper in BoVeX, which is a document preparation system I wrote as a joke (?) for SIGBOVIK 2024.[26] You probably already noticed that I did not implement proper footnotes still. You also can tell from the way that the math looks like a child typeset it that I didn't yet implement any fancy layout algorithms for that.

I did, however, spend precious vacation days in the run-up to SIGBOVIK 2025 adding features and fixing other, less important deficiencies of BoVeX so that I can continue my demented quest to use primarily software written by myself as a joke (?) instead of the perfectly decent mainstream software that everybody else uses, and whose lives are therefore presumably not thrilling in this way. So begins the Tom 7 SIGBOVIK tradition of listing *BoVeX improvements*:

Unicode. BoVeX now supports Unicode fonts. I needed this so that I could write π when I was on a digression about sampling quaternions. This was so annoying to implement! PDF was defined during the era where we were just finally realizing that our approach to character sets and font encoding was unsustainably complicated, and so they tacked on Unicode as a hack on top of that complicated mess. So you get all the benefits of the complexity of Unicode and all of the benefits of the complexity of not Unicode. You actually have to manage the glyphs yourself, for example, but also tell PDF how big everything is (but also how big it *might be*, just in case it's inconvenient to actually render it) and you also have to tell it how to decode the glyphs back into Unicode so that you can search or copy-paste from the PDF. Ugh! There are a number of undocumented or barely-documented requirements, and the symptoms of mistakes are that Adobe Acrobat will tell you "Unable to open `test.pdf`. Please contact the document author." Um, I contacted myself but nothing happened! But now you can just put UTF-8 in your BoVeX source code and it'll work. Check this out: Дональьд Трамп может поцеловать мою задницу!

FixederSys. Along those same lines, I extended the FixederSys font family[27] with a lot more Unicode characters, like the many exotic mathematical symbols that nobody has ever used. Unicode is even more inspiring to notation fetishists than amssymb in this regard. It's too bad that the math in this paper is so elementary, or else we could write A ⋧ B ⊯ ⋩∔C ⊶ ∷D⋨ like they do on ΞΔΟ-UΓΘΧ 11.

"Robustness". BoVeX no longer crashes programs like Adobe Acrobat that expect the PDFs to be "correct." LOL!!

List of open problems

Can we disprove the universal Rupert conjecture, by proving that one of these nice symmetric polyhedra does not have the property?

Or, can you find a solution to one of these unsolved polyhedra, demonstrating that I am a bad programmer?

Harder: If the conjecture is false, can we show that the snub cube is the polyhedron with the fewest vertices (24) that fails it?

Maybe tractable: Can we prove that polyhedra with dihedral symmetry (extrusions of regular *n*-gons) are always solvable either because they form "incorrect manhole covers" or "churros"? Where does the crossover point occur?

Easier: Can we prove that for some polyhedra (e.g. the regular tetrahedron), any Rupert configuration involves a translation (i.e. the projected origins do not coincide)?

Conclusion

This paper essentially does not advance the state of human knowledge in any way.

Acknowledgements. I like to think that the upsetting facts that (a) I am well sick of this project at this point and (b) I didn't solve it are due to an unusual (for me) approach I took with it. That is: I talked about it openly with my friends, and even collaborated. David Renshaw created some excellent animations that appear in the accompanying video, and his own soothing music video. He also found several bugs in my code, most importantly that my computed vertices for the disdyakis triacontahedron were incorrect! Jason Reed made a "boring, hard video game" version of the problem you can do in your browser. Tom Lokovic, who shares my self-defeating Gen-X distate for modernity, drew upon his 1990s computer graphics wizardry to work through a few rendering puzzles with me. All of the Brain Geniuses at ThursDz's and Henge Heads Lunch *at a minimum* tolerated me repeatedly talking about polyhedra, and many had suggestions as well. However, all of these suggestions were ultimately fruitless or perhaps even harmful.

Bibliography

[18] Coxeter, Harold Scott Macdonald. "Regular polytopes". *Dover*. 1973. 321 pages.

[6] IEEE. "754–2008 IEEE standard for floating-point arithmetic". August 2008.

[3] https://en.wikipedia.org/wiki/Kinematics_of_the_cuboctahedron. Wikipedia. "Kinematics of the cuboctahedron".

[8] https://en.wikipedia.org/wiki/Prince_Rupert_of_the_Rhine. Wikipedia. "Prince Rupert of the Rhine".

[9] Ying Chai, Liping Yuan, Tudor Zamfirescu. "Rupert Property of Archimedean Solids". *The American Mathematical Monthly*, 125. 2018. pp. 497–504.

[25] George E. Collins. "Quantifier elimination for real closed fields by cylindrical algebraic decomposition: a synopsis". *ACM SIGSAM Bulletin*, 10(1). 1976. pp. 10–12.

[22] Leonardo De Moura, Nikolaj Björner. "Z3: An efficient SMT solver". *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. March 2008. pp. 337–340.

[4] Albrecht Dürer. "Underweysung der Messung mit dem Zirckel und Richtscheyt". Nuremberg, 1525.

[11] Albin Fredriksson. "Optimizing for the Rupert Property". *The American Mathematical Monthly*, 131. 2022. pp. 255–261.

[7] Asmus Freytag, Rick McGowan, Ken Whistler. "Known anomalies in Unicode character names". *Unicode Technical Note*. August 2024.

[5] Thierry Gensane, Philippe Ryckelynck. "Producing dense packings of cubes". *Discrete Mathematics*, 308(22). November 2008. pp. 5230–5245.

[16] Torbjörn Granlund. “GNU MP 6.0 Multiple Precision Arithmetic Library”. *Samurai Media Limited*. London, November 2015.

[21] Gareth Healey. “Proving Grounds: Performing Masculine Identities in Call of Duty: Black Ops”. *Game Studies*, 16(2). December 2016.

[10] Gérard Lavau. “The Truncated Tetrahedron Is Rupert”. *The American Mathematical Monthly*, 126. 2019. pp. 929–932.

[13] https://en.wikipedia.org/wiki/Euler_angles. CMG Lee. “Roll Pitch Yaw Mnemonic.svg”. August 2006.

[26] Tom Murphy VII. “Badness 0 (Knuth's version)”. *SIGBOVIK*. April 2024. 16 pages.

[12] Tom Murphy VII. “GradIEEEnt half decent”. *SIGBOVIK*. March 2023. pp. 33–56.

[14] Tom Murphy VII. “What, if anything, is epsilon?”. April 2014. pp. 93–97.

[27] http://tom7.org/fixedersys/. Tom Murphy VII. “The FixederSys font family”. 2024.

[20] King Roberts. “I unlocked EVERY Calling Card in Black Ops 6 Multiplayer”. *YouTube*. December 2024. 33m11s. http://youtu.be/Oe8fyWZB12g.

[1] Bertrand Russell. “Nightmares of eminent persons”. 1954.

[2] Bertrand Russell. “The theologian’s nightmare”. *Fact and Fiction*. 1961.

[17] Ludwig Schläfli. “Theorie der vielfachen Kontinuität”. *Auftrage der Denkschriften-Kommission der Schweizerischen naturforschenden Gesellschaft*, 38. 1901. pp. 1–237.

[19] Wolfram Schultz. “Predictive Reward Signal of Dopamine Neurons”. *Journal of Neurophysiology*, 80(1). July 1998. pp. 1–27.

[24] Alfred Tarski. “A decision method for elementary algebra and geometry”. 1951.

[15] https://github.com/avaneev/biteopt. Aleksey Vaneev. “BITEOPT - Derivative-free global optimization method”. 2024.

[23] Юрий Владимирович Матиясевич. “Диофантовость перечислимых множеств”. *Доклады Академии наук*, 191(2). 1970. pp. 279–282.