Licensing

MIT license ✔

Implementation language

~~TypeScript~~ Rust ✔

Todo list

AFAIK thoroughly out of date

Currently working on:

- A Polytope.prototype.circumcenter function.
- A function to build a cyclic polygon from its edge lengths and orientations.
- A Polytope.prototype.petrial function.

Short term:

- Allow exporting into other formats, such as STL or GGB.
- Add duals and petrials.
- Add various render types: projection, wireframe, expansion, spherical (?), as well as further options (fill type, projection type, view distance, …).
- Render vertices and edges, or at least have an option to render them.
- Create functions for some polyhedron operators: truncating, stellating, faceting, …
- Expand on the Scene class, make a basic HTML interface.
- Render only on demand.
- Implementing triangulating algorithms for different fillings, to allow for general polyhedra to be displayed.

Medium term:

- Generate convex polytopes from their vertices (we'll probably port QHull's code for this).
- Create compressed file formats to efficiently store polytopes as files.
- Generate simple polytopes from Coxeter diagrams.
- Import some localization library
- Generate incidence matrices from polytopes for structural verification.

Long term:

- Create an extensive polytope library, including functionality for the infinite families, and with as many of the following as is reasonable built in:
    - Regulars/Uniforms

    - Scaliforms
    - Semiuniforms
    - CRFs
- Also allow generation of infinite families of interest, like
    - Simplexes, orthoplexes, hypercubes
    - Products of polytopes
    - Antiprisms
    - Pyramids, bipyramids
    - Cupolas
    - Orthohemihypercubes (aka thah and friends), and other uniforms that generalize to higher dimensions
    - Demihypercubes.

Petitions

- Ability to rotate polytope and not only view.

Backend stuff

Potential representation: Symmetry group, flags in each element of symmetry group, realization of the symmetry group, realization of each flag class
Another representation: Vertex positions, abstract polytope
Variant of the first representation: Symmetry group, elements in each domain and generators under which they're invariant, realization of the symmetry group, realization of each element class
Advantages: Element count is generally lower than flag count
Disadvantages: Some operations are more awkward
Groups can be represented as rewriting systems for strings of generators or matrices
Strings of generators:
Pros: Works for Grünbaumians easily, easy to enumerate elements
Cons: Slow to generate, requires manual handling of starry groups
Matrices:
Pros: Easy to generate, straightforward handling of starry groups
Cons: Grünbaumians are kinda awkward (but doable), hard to enumerate elements AFAIK
At any rate, we can use both so yeah
Representing compounds with the first format requires the ability to have multiple root flags, or just assume that all flags are reachable which might require awkward messing around with the symmetry group
We really need to work out our data structures somewhere
Algorithms for the stuff in the big list below

- Faceting: I don't really care, figure it out yourself

- Semiuniform morphs: No changes to the abstract structure, just change the lengths of the edges and solve, numerically or otherwise (note that the vertex may cross mirror planes, for polyhedra where symmetries always have (supersymmetries with) reflectional domains AFAIK you can just bounce off the plane, but in 4D+ some symmetries don't work like that). For non-snub Wythoffians, choosing the vertex position for particular edge lengths is straightforward.
- Truncation rotation: The right half is moving the point along the original polytope's edge, the left half is continuing past the end of the original edge but projecting back onto the original cube so the point doesn't go to infinity at 9 o'clock (it would probably be simpler to just move the point along a great circle parallel to an edge)
- Tribe generation: If you just want to be able to move freely around the configuration space, then that's fairly straightforward. If you want to draw a picture of all teepee representatives like in the glossary, then you need to find the ridges between teepees, which can probably be done by generating subtribes (?) from setting one edge length to 0.
- Generating polytope from verf: ~~Dual, fit the resulting shapes together somehow, dual. Of course, the hard part is figuring out how the verf duals fit together, so good luck with that. It's much easier if you already have a construction of another regiment member since then you already know how copies of the dual of the verf fit together.~~ Better ways exist.
- List valid verf facetings: Recursively I guess
- Honeycombs: Instead of generating/drawing all elements, only draw elements close to the camera. This might also be a good idea for highly complex spherical polytopes.
- Generate polytope from Wythoff triangle/Coxeter diagram: Generating the domain is easy, but determining the flags in each domain isn't. I think it's easier to find centers of elements that intersect with the domain and then build the flags from those. Any mirror that is not incident with the vertex generates an edge. (Including edges from mirrors incident with the vertex produces degenerate edges, which might be useful for some stuff) Place this edge's center on the domain facet corresponding to this mirror. (Exact location doesn't matter since we're building the abstract polytope.) For each mirror incident with the vertex, if said mirror is perpendicular to the mirror being used to generate an edge, move the center to be incident with this mirror. A similar procedure can be used for generating faces and higher elements, except you start with an edge/higher element center and after the initial placement of the center, you must align the new center to be incident with the mirror(s) that generated the edge/higher element. After generating centers of all elements, the flags of this domain can be generated by tracing all paths from the nulloid to the full polytope using only elements within this domain. The physical vertex position can then be derived using the normals of the mirrors, which can be computed from reflection matrices as the eigenvectors with eigenvalue -1. For snubs, do the construction for the corresponding non-snub, abstractly holoalternate like in the Grünbaumian thing, delete degenerate elements, and readjust the vertex position numerically.
- Generate polytope from a more general fundamental domain: There are probably ways to do this similar to the above, but they're probably more complicated (eg directly

building snub polyhedra requires a pentagonal domain with various types of connectivity between edges)

- Deleting degenerate elements: For this, you need to select two dimensions: the dimension of the element to delete and the dimension of elements to merge. For example, when deleting digonal faces from an alternated cube, dimension to delete=2 and dimension to merge=1 since we're deleting faces and merging the incident edges. For each flag class of the element to delete, modify the flag class reached from a change (element of dimension to delete) operation to go to the flag obtained from changing (element of dimension to merge) and then (element of dimension to delete) from the chosen representative of the element to delete. Then, delete the degenerate flag classes. Since we're removing only degenerate elements the only modifications to the realization are making sure the realization doesn't reference any classes that no longer exist.
- Blending: This is equivalent to making an overlapping compound, finding overlapping elements, rearranging the classes in the overlapping elements into degenerate elements, and deleting those degenerate elements. Compounding and removing degenerates are discussed elsewhere, so here I'll focus on finding overlapping elements of compounds and rearranging them into degenerate elements. Finding overlapping elements is just looking for flags with identical physical realizations excluding the components themselves (which in most cases should be identical to flags such that the vertex, edge center, face center, etc are identical, for a suitable choice of center), and to rearrange the overlaps into degenerates, you need to change the (facet-of-overlapping-element)-change operation of each overlapping flag to point to the flag it overlaps. This changes the overlapping elements into ditopes, which can be safely eliminated. If you want to mess with fissary ridges/etc, instead of only ignoring the full-dimension components when finding flags, ignore any elements of dimension higher than the dimension of elements you would like to modify. If there's more than two overlapping elements, choice of overlapping elements to merge doesn't really matter as long as you're consistent within an element. Something else that should work for facet elimination is instead of rearranging the overlapping facets into degenerate facets, disconnect them from the rest of the polytope. This can be done by replacing the (overlapping-element-dimension)-change operation instead of the (facet-of-overlapping-element)-change operation. Then you can simply eliminate the relevant flag classes.
- Compounding: The two interesting types of compounding (AFAIK) are compounds of same-symmetry polytopes and compounds of lower-symmetry polytopes with higher symmetry. The first one is probably trivial so I'll describe the algorithm for the second one here. First, apply the monomorphism from the smaller symmetry group to the larger symmetry group to the group elements in the description of the compound component. Then, add root flags for each pair of root flags from the initial polytope and representatives of the left (I think) cosets of the smaller symmetry group in the larger symmetry group. This might not be necessary depending on how it's implemented. For example, applying this to o4x2x using the monomorphism from o4o2o to o4o3o gives the

compound of 3 cubes, and doing something similar using s4s2x and s4s2o to o4o3o gives the compound of 6 cubes with rotational freedom.

- Decreasing symmetry: To decrease the symmetry of a polytope, first choose representatives of the left (I think) cosets of the smaller symmetry group in the larger symmetry group. Then, for each old flag class, create (order of larger group)/(order of smaller group) new classes, where each one corresponds to a member of the old flag class in a domain in one of the cosets. Connect the new classes according to that and copy the concrete realizations of the vertices (if necessary) in a similar way.
- Matching vertices: Again, dunno what this is supposed to be.
- Measures of polytopes: Someone else do this one I don't really care about it
- General truncation operators: The abstract half is just applying the algorithm in the "Generate polytope from Wythoff triangle/Coxeter diagram" section to get a mapping from flags of the original to flags of the result, then applying this to a general polytope. For most polytopes, I'm pretty sure you can assume that all flags have linear CD diagrams, but just to be on the safe side you might want to assume a complete CD diagram. For the concrete half, for any particular truncation operator it's fairly easy to find the new location of each vertex as a linear combination of the relevant vertex, edge center, face center, etc. if you don't mind messed-up edge lengths. If you do, then that probably leads to a bunch of messy equations that you might as well just do numerically somehow.
- Duals: Taking the dual of an abstract polytope in the flag format is fairly trivial. For the concrete polytope, place the new vertices at the centers of the old facets and rescale if you care about that.
- Element counts: Number of elements of a particular type=Symmetries of the polytope/symmetries of one of the elements, in the context of the whole polytope (aka how Spirit says element symmetries should work). Note that symmetries of one of the elements in context=symmetries of that element alone/number of classes of flag used to represent it in the full polytope.
- Incidence matrices: Diagonal entries are simply element counts. There's probably some easy way to get the rest of the entries but I can't seem to figure it out so yeah idk ask Klitzing.
- Circumradius: For uniform polytopes, you can apply a simple formula to the verf circumradius to get the full polytope's circumradius. (I'm pretty sure this also works for isogonal polytopes with multiple edge lengths.) Of course, we already know where the vertices are, so the verf thing is only really useful if we want an exact answer.
- Vertex coordinates: Start with each vertex within a domain and reflect it around. If you want to compress sets of vertices that are equivalent under certain subgroups of hypercubic symmetry ("[even] permutations and [even] sign changes of ..."), it's more complicated.
- Generate troops of a regiment: Starting from a collection of flag classes corresponding to the relevant facet regiment members, with every connection except "change facet" already defined, link the classes that correspond to adjacent facets.

- Fundamental domain information: Determining the measures of general spherical/Euclidean/hyperbolic simplices is quite difficult and I dunno if there's a general method for that.
- Symmetry group information: For Coxeter groups, we can determine the order quickly using the hypervolume of the fundamental domain, taking into account the fact that starry Coxeter groups will multiply cover the sphere. The density of a starry Coxeter group is not always obvious, however. For other groups, the best thing I can think of is to try to describe it as an extension of a cyclic group, which probably isn't too hard to check and if possible reduces the order significantly. Once you reach a group with no cyclic quotients, it's probably not that easy to do better than just enumerating all the elements. However, for most groups in 6 dimensions or less, it should be safe to simply enumerate all elements, or doing so after removing diagram-automorphism extensions, splitting prismatics, and taking the chiral subgroup (after those, I think the largest possible group is order-25920 chiral E6). Actually, if we use a rewriting-system group representation, then orders can actually be computed without checking every element of the group by keeping track of the number of elements of a particular length that end with a prefix of one of the LHSes of the rewriting system and no longer prefix. To compute the number of elements of length (n+1) from a table of numbers of elements of length (n) in each suffix class, for each suffix class, append a generator and repeatedly delete the first generator of the suffix until it becomes a prefix of an LHS of a rewrite rule of the system. If this prefix is an LHS of a rewrite rule, then these elements are or will be counted elsewhere, so don't count them. Otherwise, do count them. Of course, this is still one of those places where matrix representations suck.
- Exporting media files: I don't know and I don't care. There's probably guides online for this kind of stuff.
- Slices: Slicing a 1D polytope is trivial, everything higher's kinda tough
- Polytope filling: Really even figuring out how to actually render nonconvex polytopes in a not completely shitty way is somewhat nontrivial. Other people probably already know 1D and lower elements are trivial, and for everything higher you can just compute which facet planes you'd hit if you went in a random direction from the point you're checking, check how many of those facets you actually collide with, and react accordingly, but this is probably kinda slow. URL posted an algorithm for this on the server. It's probably not that fast in higher dimensions but it should work, at least for solid/binary filling.
- Rendering as a sphere tessellation: Yeah I dunno
- Generate nets for model making

Frontend stuff

1. Faceting
    a. Faceting diagrams
    b. Faceting criteria: tame, feral, wild, orientable, isohedral, semiuniform, same edges etc.

    c. Manual faceting from faceting diagrams
2. Semiuniform morph stuff
    a. Truncation rotation
    b. Generating tribes
3. Generating polytope from verf
    a. List out valid verfs
4. Rendering for Euclidean and hyperbolic polytope possibly with multiple projections.
5. Generate polytope from wythoff triangle, coxeter diagram, etc.
6. Easier blending, and matching vertices
7. Detailed information for each polytope, such as exact expressions for measures, their symmetry groups, etc.
8. Support for general truncation operators.
9. Duals
10. Research features (for higher dimensional solids and stuff that cant be displayed too)
    a. Element counts
    b. Metrics
        i. Circumradii
        ii. Vertex coordinates
    c. Generate troops of a regiment
11. Information on fundamental polytopes (fundamental domains) and symmetry groups.
12. Export to image, video, and 3d files (transparent backgrounds)
13. Slices
14. Multiple filling methods
15. Polytopes can be spheres.
16. Generate nets from polytope.
17. REMEMBER SETTINGS!
18. Stellation (include internal structure).
19. Change symmetry of a polytope (including chiral ones, so like chiral cube)
20. Show elements and flags
21. Augment and excavate polytopes.
22. Use minimal surfaces for skew polytopes. (nah, way too hard)
23. Generate conjugate if a polytope.