Easy, the Elliminator

Jens K. Becker
Uni Tübingen
Sigawrtstr. 10
72072 Tübingen

Libraries

✔ Python

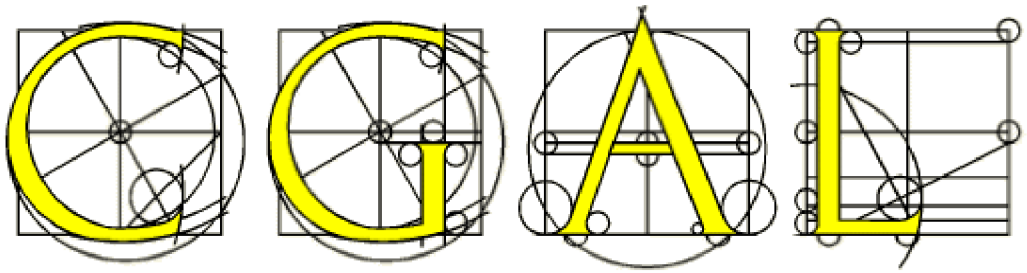
Python® is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days.

✔ CGAL

The Computational Geometry Algorithms Library (CGAL), offers data structures and algorithms like triangulations (2D constrained triangulations and Delaunay triangulations in 2D and 3D), Voronoi diagrams (for 2D and 3D points, 2D additively weighted Voronoi diagrams, and segment Voronoi diagrams), Boolean operations on polygons and polyhedra, arrangements of curves, mesh algorithms.....

✔ VTK

The Visualization ToolKit (VTK) is an open source, freely available software system for 3D computer graphics, image processing, and visualization used by thousands of researchers and developers around the world. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python.



3 dimensions and precision: field and ring number types

Some programming internals:

Type	Bits	Values	Digits
float	32	1.5E-45 .. 3.4E38	7 – 8
double	64	5.0E-324 .. 1.7E308	15 – 16
long double	80	1.9E-4951 .. 1.1E4932	19 – 20

This means that using any of these, the following is not true:

3*1/3=1

because (using e.g. a float) 1/3=0.3333333 and 3*0.3333333=0.9999999
This means any of these is a field number type

The solution to this are ring number types: Ring numbers do not actually calculate 1/3, they store the calculation of 1/3 (they store the nominator and denominator). This is nifty but awkward to get used to because that also means that with ring number types there is no such thing as a division, only +, - and * are allowed (and of course “higher” mathematical functions such as square root etc.)

Mathematical errors in computational geometry

```
double result=1.0;
for(int i=0;i<20;i++)
      result*=0.000016;
for(int i=0;i<20;i++)
{
      result/=0.2;
      result/=0.00008;
}
```

multiply content of result 20 time with 0.000016

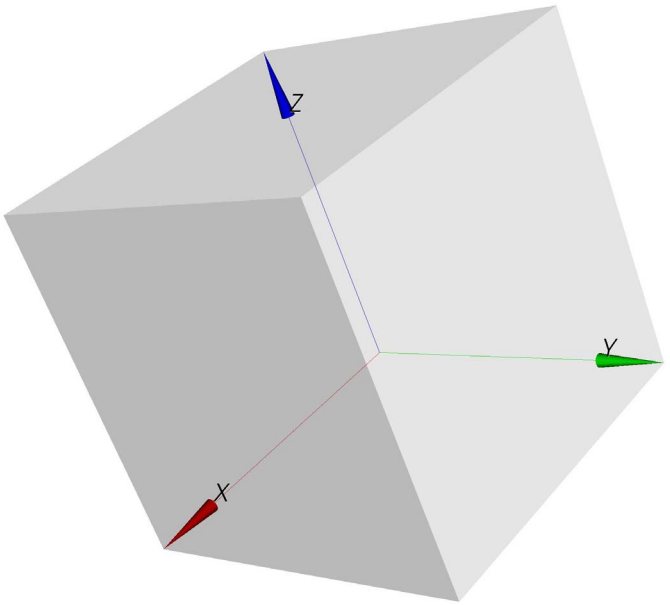
divide content of result first by 0.2, then by 0.00008. do that 20 times

Using a pocket-calculator the result should be 1.
Using a computer the result is 0.9999999999999678.

Now do calculations like the one above on a polyhedron with 500 vertices for 100000 time steps.....

Coordinate systems: Cartesian vs. Homogeneous

Everybody knows what the Cartesian coordinate system is: three values, one for x, y and z coordinates. Homogeneous coordinates expand this triplet to include one more coordinate: w

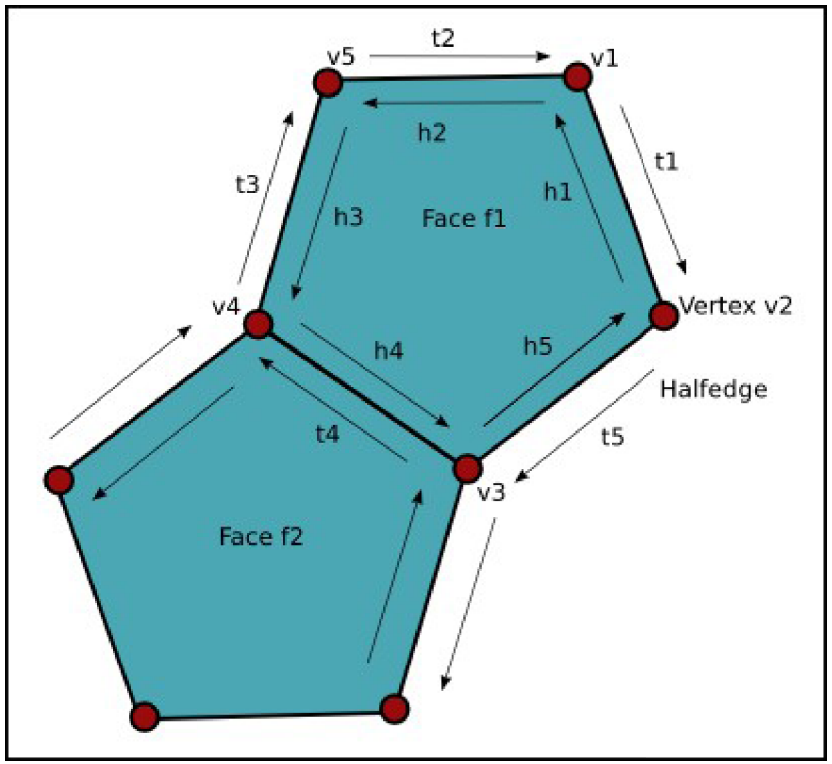


Looking at the following example it is easy to understand how this works:

Cartesian:	Homogeneous (all integers!):
X=1.5	X=3
Y=3.5	Y=7
Z=4	Z=8
	W=2

Ring number types and homogeneous coordinates are easy to understand on their own, combining these two concepts it suddenly gets very tricky....

The data structure: a doubly connected halfedge list

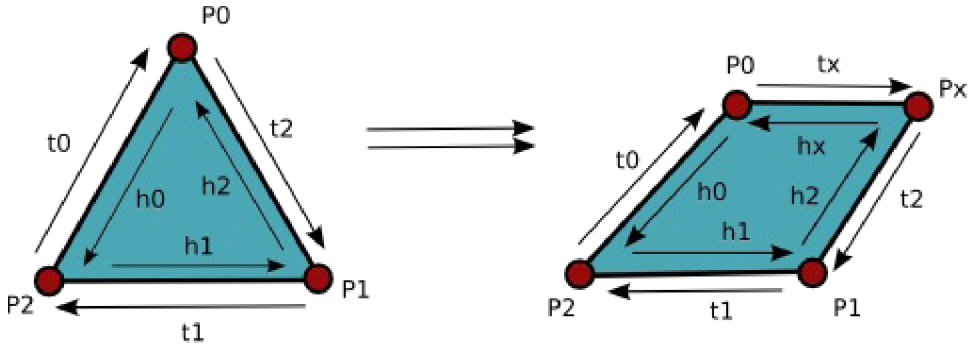


Each polyhedron has a list of

- facets
  - Facets store a pointer to a halfedge
- halfedges
  - Halfedges have a pointer to a vertex, to the next, previous and twin halfedges and to a facet
- vertices
  - Vertices have a pointer to a halfedge

With this information it is possible to traverse through all vertices, all facets, all halfedges of one or all facets etc.. And it is fast. But it comes at a price.....

Operations with a dchl: Adding a single vertex



Halfedge	Direction	Origin	Twin	Next	Prev
h0	P2P0	*P0	*t0	*h1	*h2
t0	P0P2	*P2	*h0	*t2	*t1
h2	P0P1	*P1	*t2	*h0	*h1
t2	P1P0	*P0	*h2	*t1	*t0

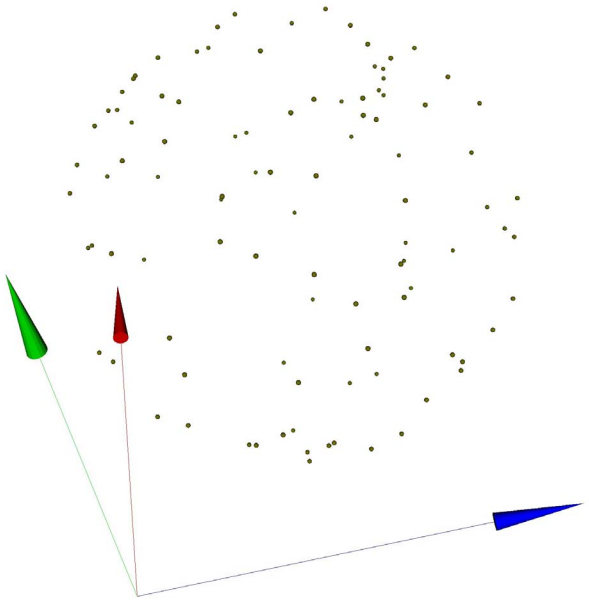
It is easy (!) enough to do but needs a lot of operations:

Original structure

Halfedge	Direction	Origin	Twin	Next	Prev
h0	P2P0	*P0	*t0	*h1	*hx
t0	P0P2	*P2	*h0	*tx	*t1
h2	PxP1	*P1	*t2	*hx	*h1
t2	P1Px	*Px	*h2	*t1	*tx
hx	P0Px	*Px	*tx	*h0	*h2
tx	PxP0	*P0	*hx	*t2	*t0

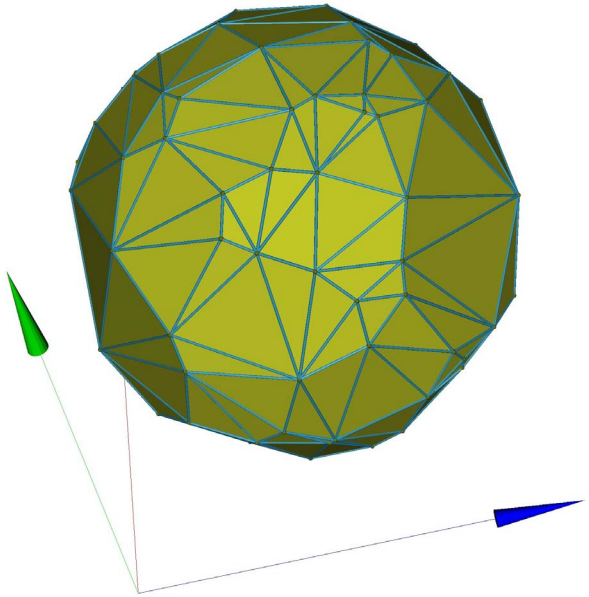
Structure after adding a single vertex

Constructing a polyhedron

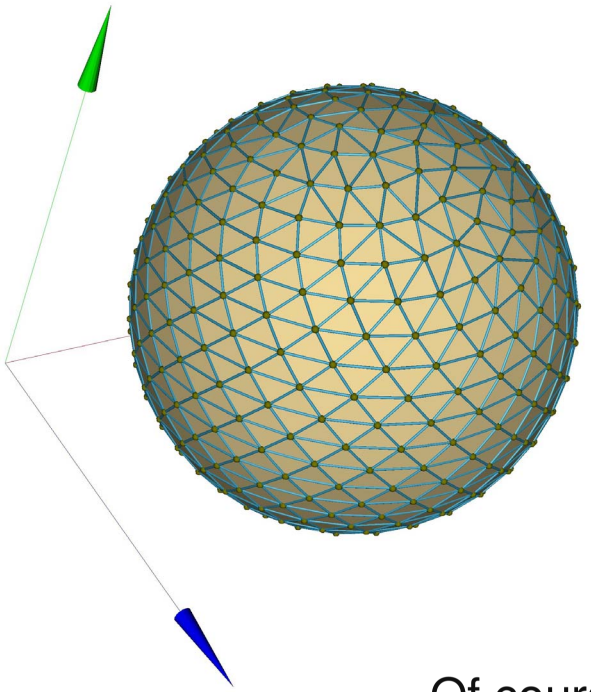


To construct a single polyhedron, I just generate random coordinates on e.g. a sphere and compute the convex hull of these points.

This gives a nice polyhedron, but the angles of facets can be very small. This is not good for numerical calculations, they should all be (more or less) the same.

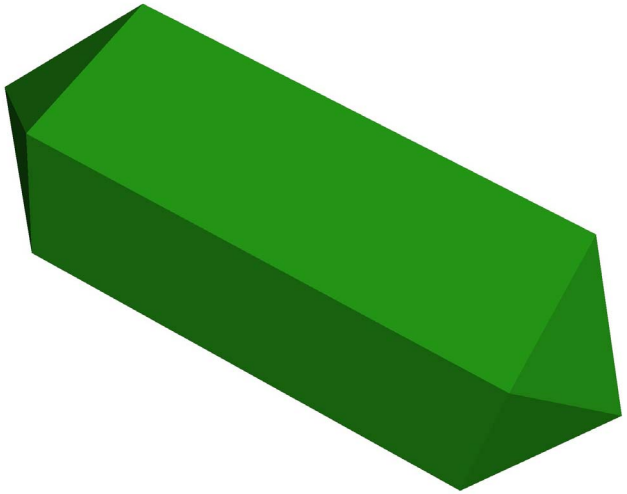


Construct arbitrary shapes



“Regular distribution of points using the “golden section spiral”.

Of course other shapes are possible, but will they be used? Do we need something that constructs polyhedra from e.g. HKL-Indices?



Saving structures

So far I use an XML-structure, but there might be something better suited?

The main elements are:

```
<POLYHEDRON>
    <ID></ID>
    <DOMINATOR></DOMINATOR>
    <MINERAL></MINERAL>
    <OFF></OFF>
    <BOUNDING_BOX></BOUNDING_BOX>
    <LIST_OF_ATTRIBUTES>
        <FACETS>
            <NAME></NAME>
            <ID></ID>
            <VALUES></VALUES>
        </FACETS>
        <VERTICES>
            ...
        </VERTICES>
    </LIST_OF_ATTRIBUTES>
</POLYHEDRON>
```

Index of polyhedron

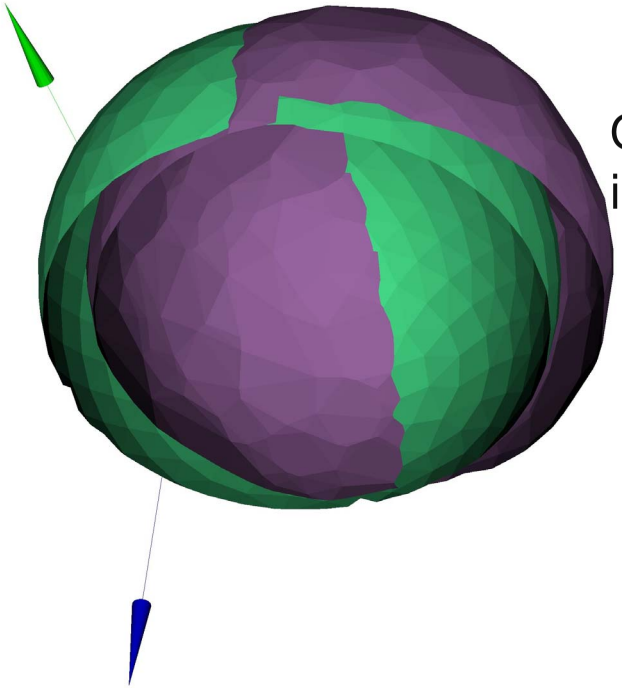
Number that can be used to define which polyhedron is substracted from which

Coordinates of points and list of which point belongs to which facet

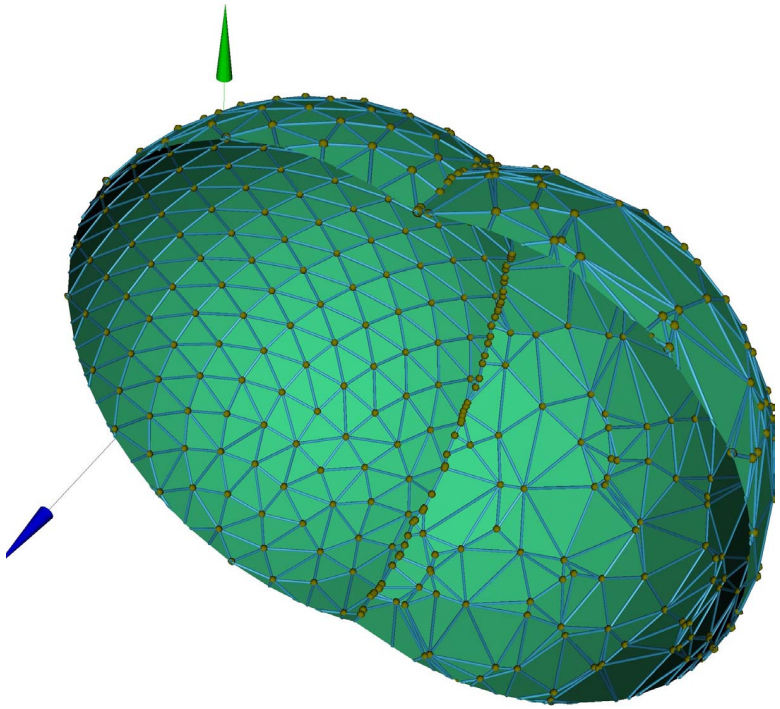
An attribute can have one or more values

Operations with polyhedra

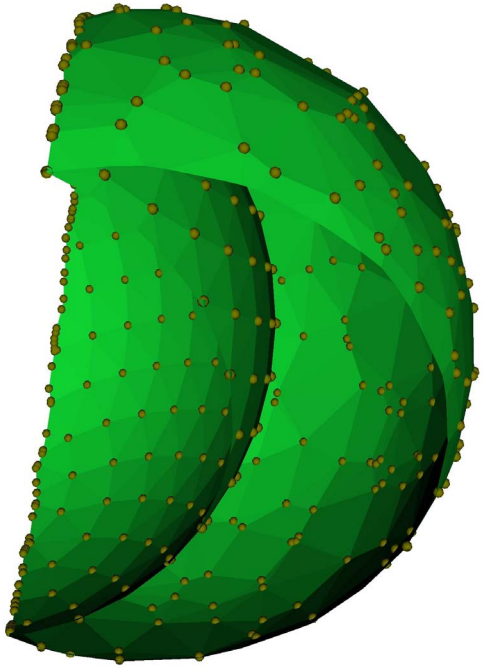
Two operations on polyhedra are possible:



Original structure with two intersecting polyhedra

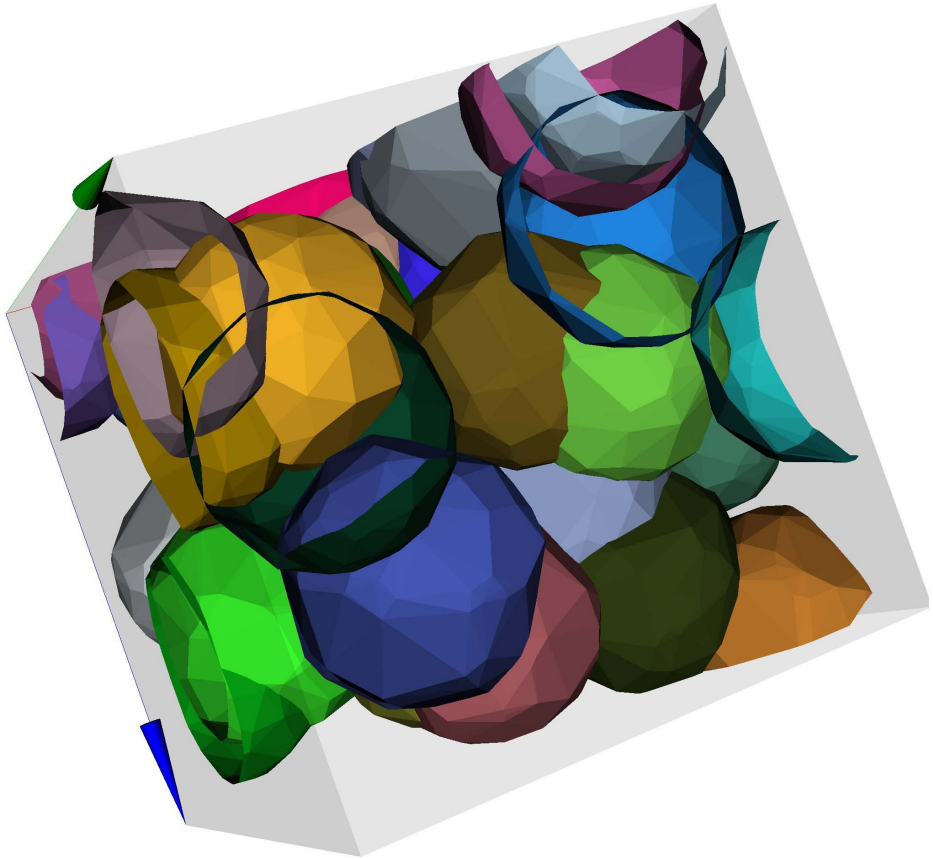


Merged polyhedra

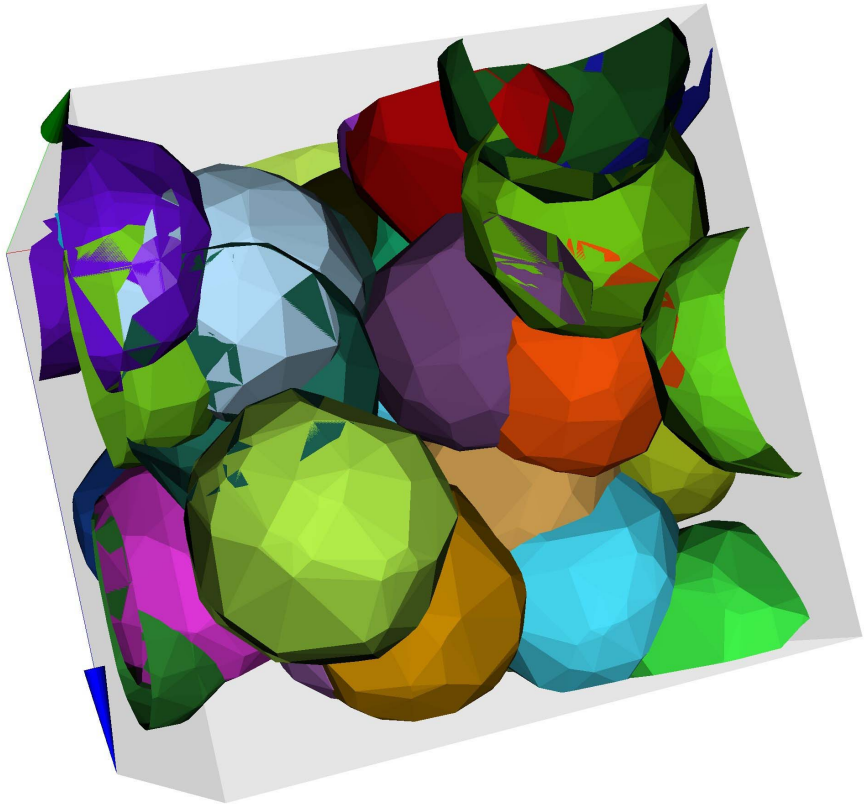


Substract polyhedra

Calculating intersections of a whole “structure”



Before calculating intersections and substracting polyhedra



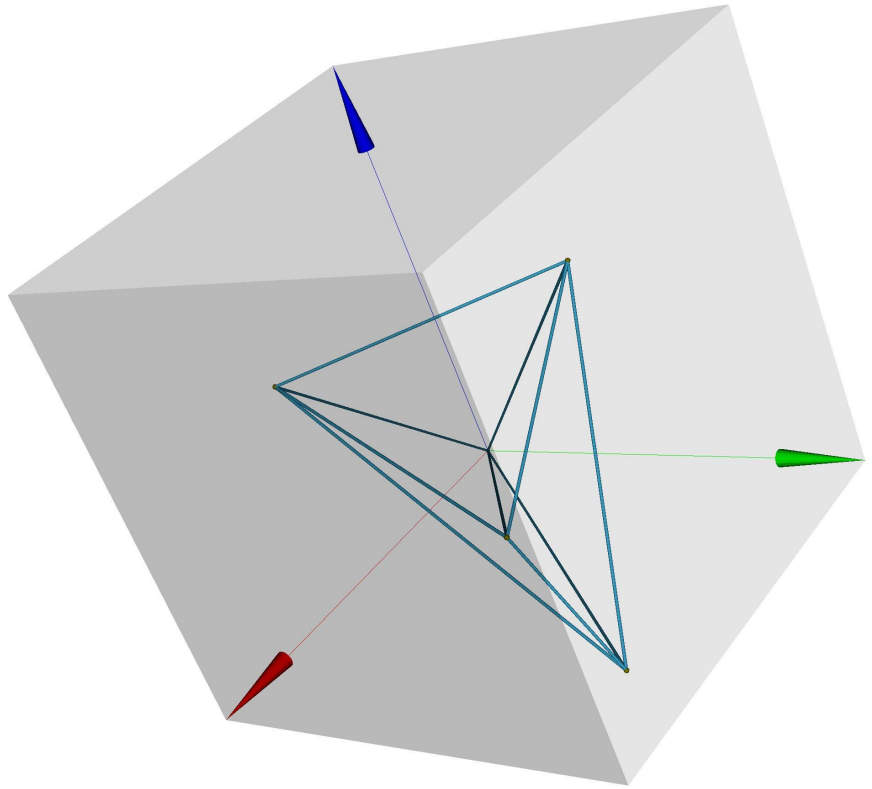
After

Triangulations

Unfortunately triangulations will be very important in Easy. Unfortunately because they are not easy in 3D.

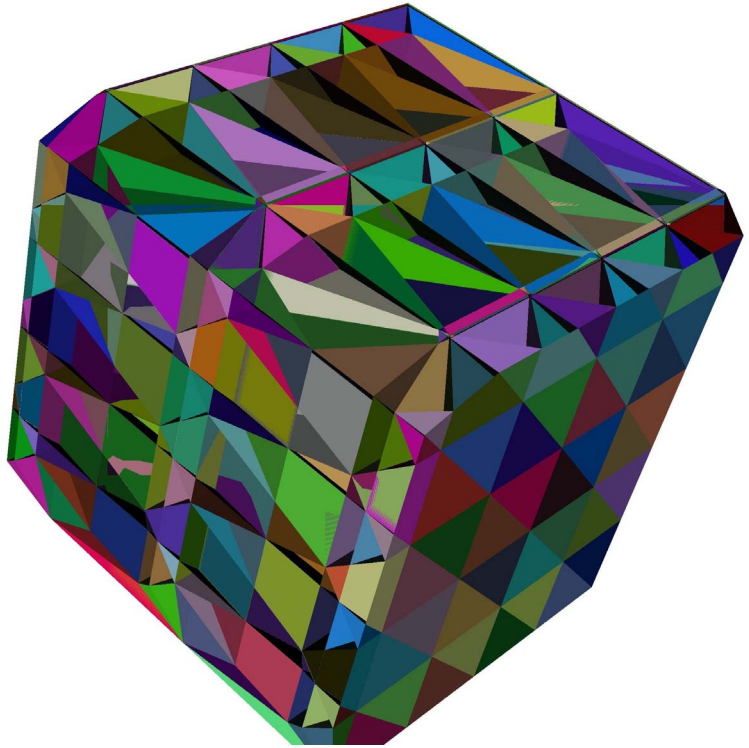
Triangulations are needed for:

- Construction of polyhedra
- Volume calculations
- Increasing (or decreasing) the resolution of the structure
- Find center of polyhedra
- Construction of whole microstructures
- etc.
- etc.
- etc.

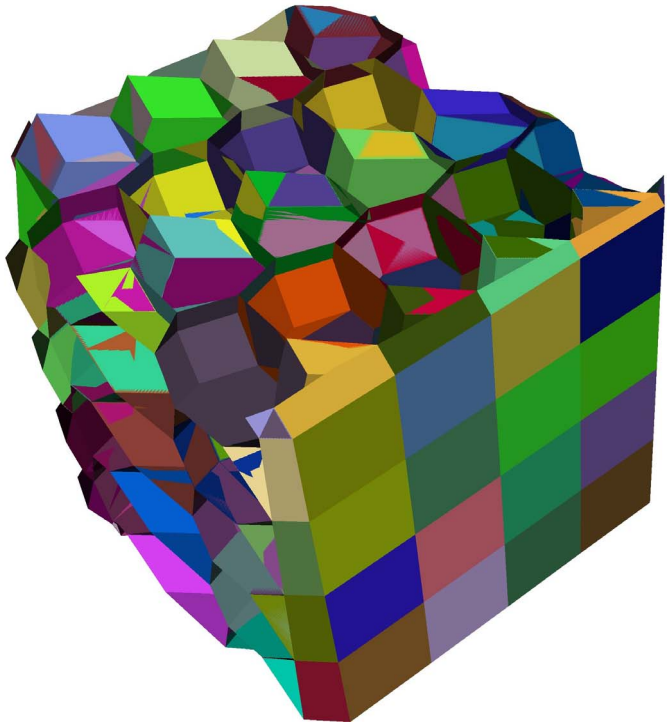


Incremental Delaunay triangulation using 4 vertices.
Results in 3 tetrahedrons, 7 facets, 10 halfedges and 5 vertices.

Triangulations (2)

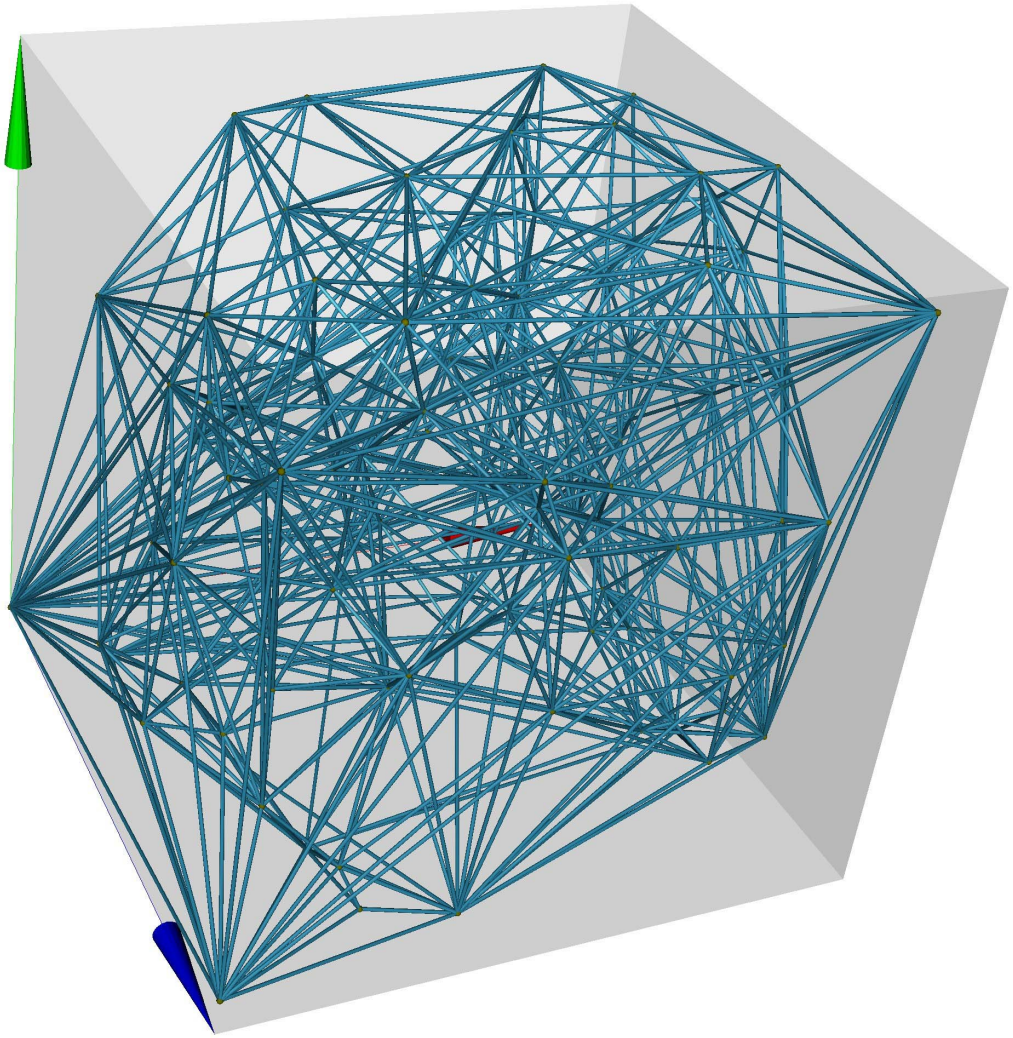


Delaunay triangulation of 125 points



Voronoi triangulation of the same points.

Triangulations (3)



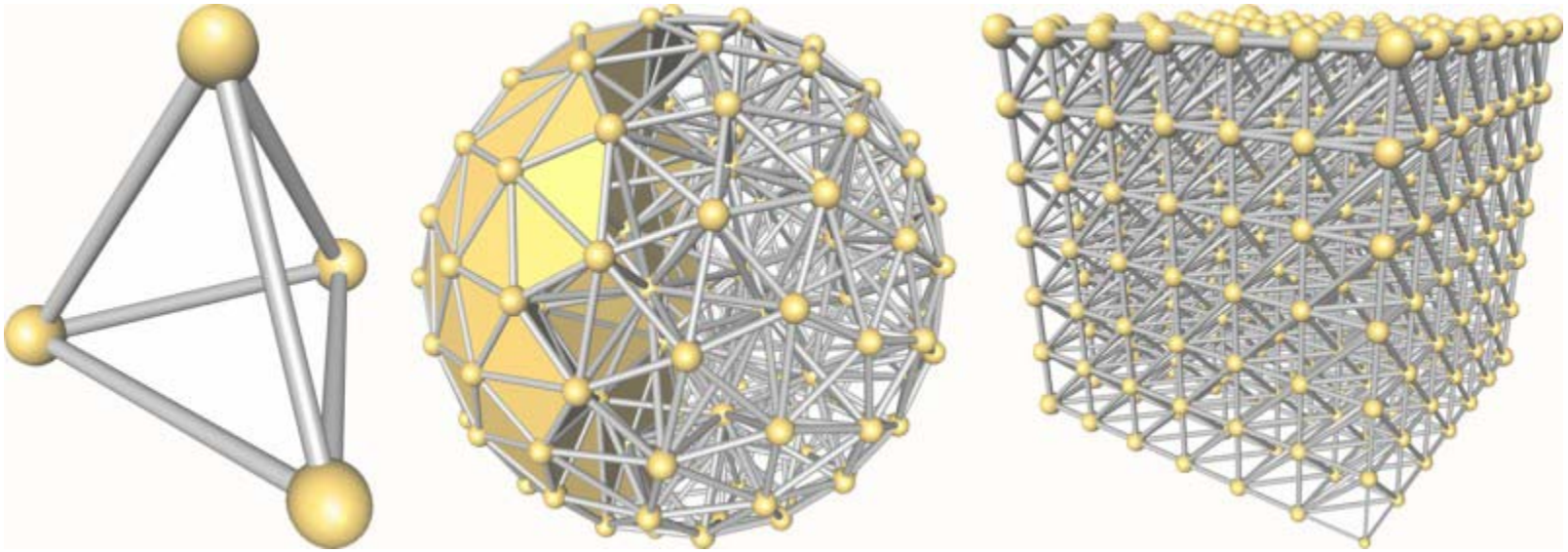
Incremental Delaunay triangulation using 100 points. This results in 575 tetrahedra, lots of vertices and even more halfedges.

This is not easy to handle... How do I test if it is a valid triangulation? Cant do it graphically.....

Constrained regular triangulations

What I really need in the end are constrained regular triangulations. This means that:

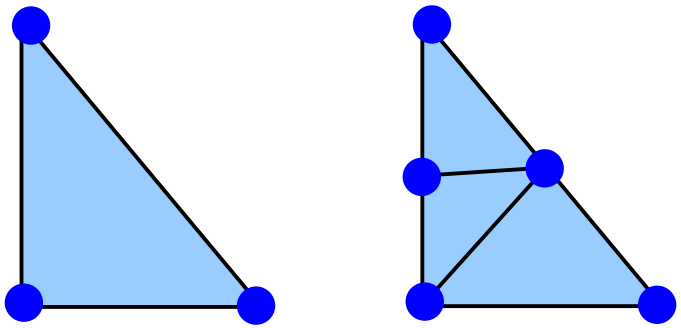
- triangulations should be confined in a certain volume (inside a polyhedron)
- the tetrahedrons should have the same volume if possible (the outer once will of course be irregular)



How to triangulate a facet

2D

Generate a grid of points in the facet and use this as triangulation points. Insert the new facets back in the polyhedron.

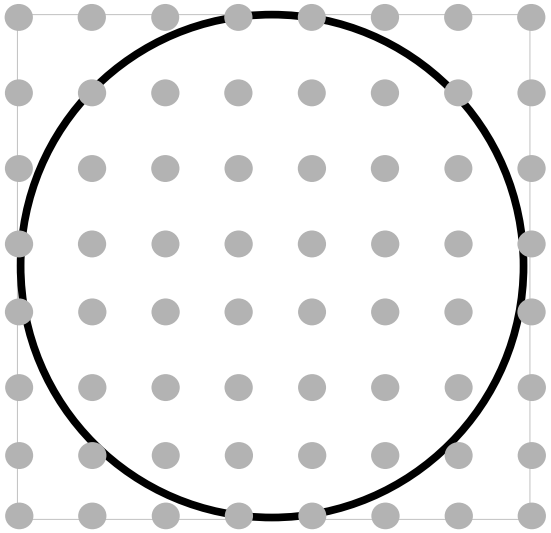


3D

This will probably not happen often (if ever) because in that case the facet has not been triangular in the first place. If it does happen, the facet has to be divided into triangles first.

Generation and triangulation of a polyhedron

As stated earlier, generating random points on a sphere does result in a nice looking polyhedron but for numerical simulations it might not be that good at all because of the small angles of facets. Therefore, it is desirable to construct a polyhedron from a regular triangulation or make sure that the points on the sphere are evenly distributed. And that is a problem known for centuries (e.g. Föppl L. 1912: Stabile Anordnungen von Elektronen im Atom (Stable distribution of electrons in an atom). J. Reine und Angew. Math. Vol. 141, pp. 251-302). **Solution: It has been solved of course but includes a lot of complicated calculations and analyses. Is there an easy way?**

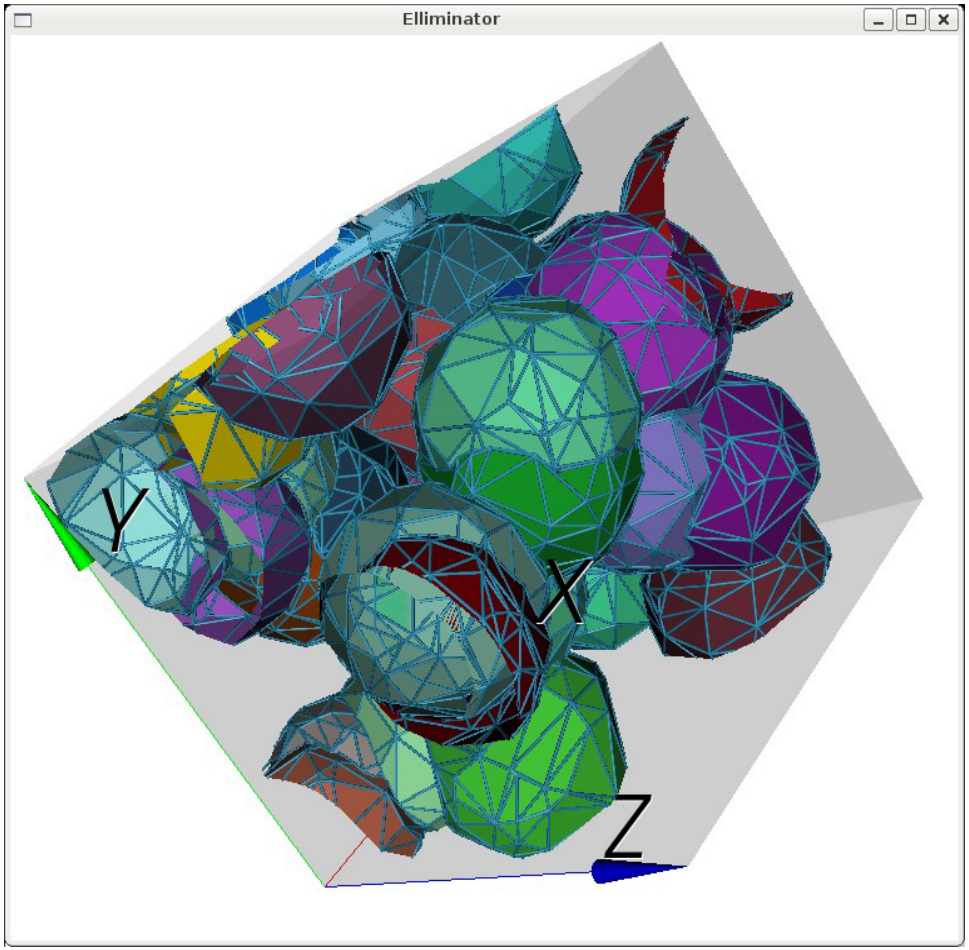


3D regular triangulation: the “Just getting it to work”-way

- Calculate the bounding box, calculate a regular grid of vertices in the box
- Cut (substract) the grid against the polyhedron, the remaining points are inside the polyhedron.
- Add the vertices defining the polyhedron to the grid
- Use this list for an incremental triangulation

EasyView

A very basic program to display the results is available. There are a lot of things that need to be done but for now it is good enough to visually debug structures. The speed is surprisingly good (I think) but depends of course on the complexity of the structure.



So far you can:

- Move around the structure
- Zoom in and out
- Display vertices, halfedges and facets (or a combination thereof)
- Save a picture of the current view
- Cut out parts of the structure
- Show axes and labels (or not)
- Show bonding box

Python, a scripting language

So far, Easy is written in C++, EasyView is written in python (I will translate it into C++ at one point). But the combination of python and C++ is very powerful. The plan is to have a python wrapper for all public functions in Easy (and EasyView). This makes developing very very fast and easy because there is no compiling of anything (that is done on the fly with execution) and even people who do not know C++ can program and use Easy.

C

```
#include <stdio.h>
int main()
{
        printf("Hello World!")
        return 1;
}
```

Python

```
print "Hello World!"
```

Remaining problems:

Programming problems:

- How to interlink nodes and facets of different polyhedra
- Wrapping
- Triangulations
- Self intersections of a polyhedron
- Do we need attributes other than doubles (the class attribute is a template but how do we access attributes of different types, overloading?)

Technical/logical problems:

- Debugging is very hard because of the high level of templating and wrapped typedefs
- Logical debugging is also very hard (e.g. if I move this node here, what happens to all the facets surrounding it etc.)
- Graphical debugging .....

Gimmick: real fake 3D