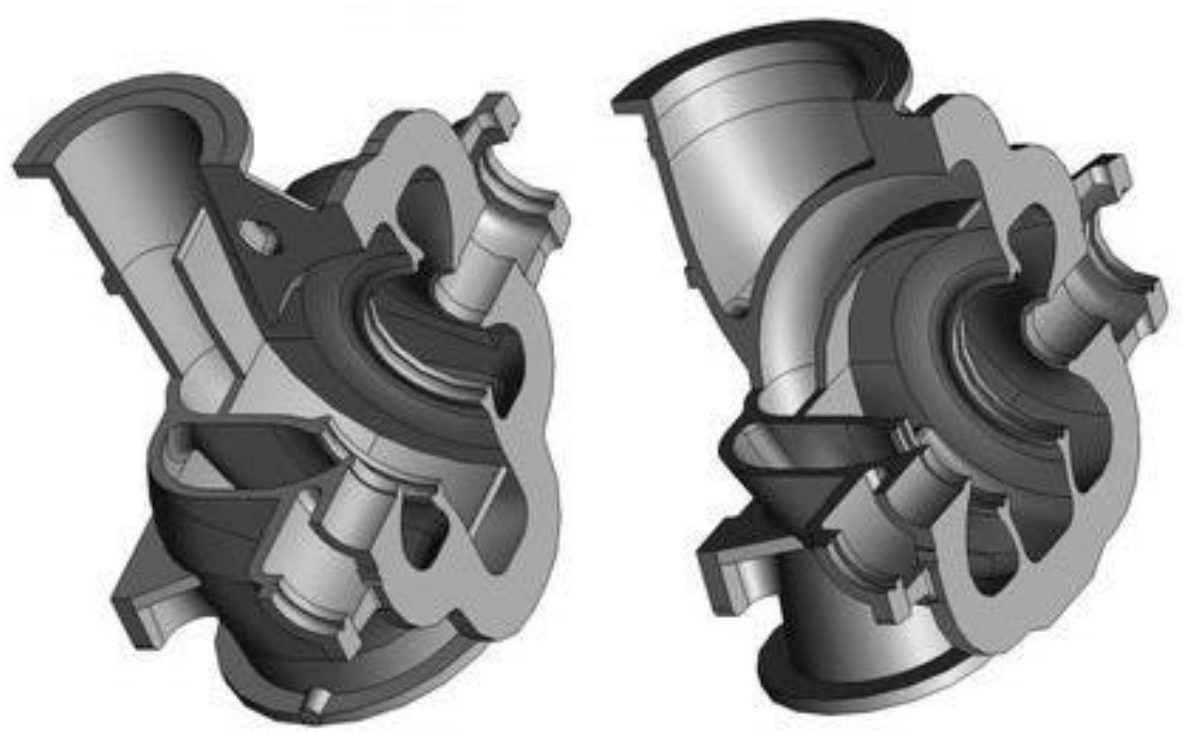
# Overview of Python Libraries and Functionality for working with 3D models

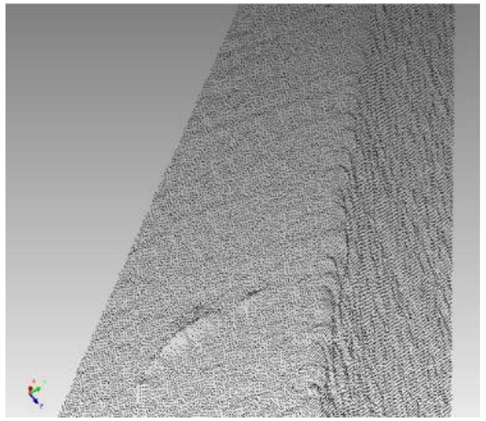
# Shape is fundamental to digital manufacturing

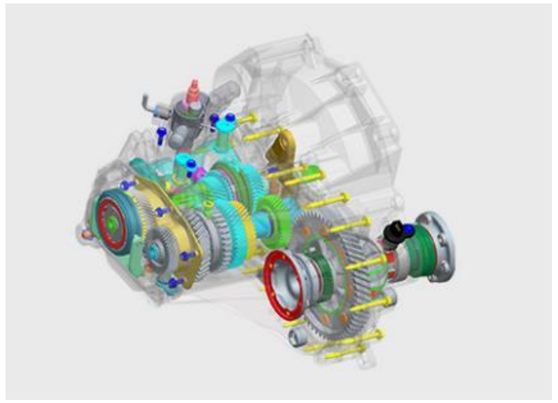
Digital technologies are used to create and manipulate shapes for design and manufacturing applications. This very quick overview will introduce the main geometric representations and give examples of the python libraries used to support their use.

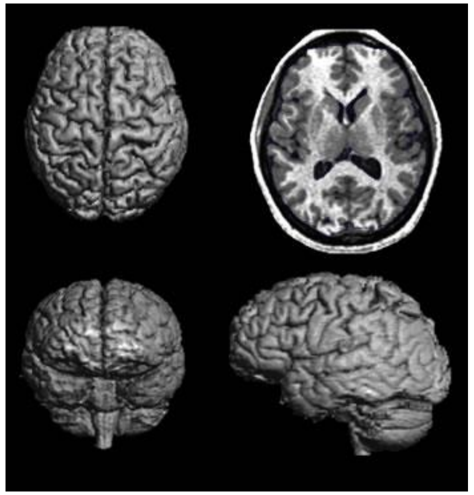


# Digital Representation of 3D shape?

The answer varies with the application?

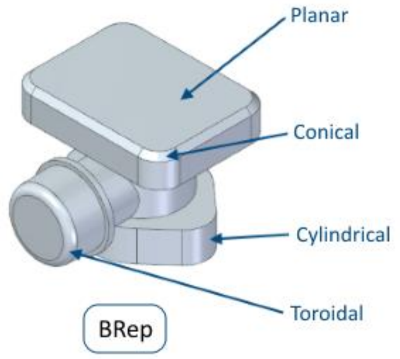




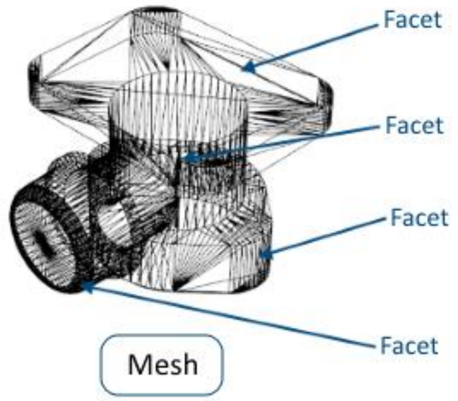


Point Cloud

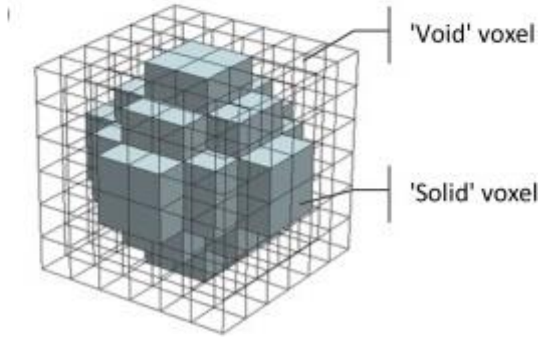
Physical Measurement



Precision



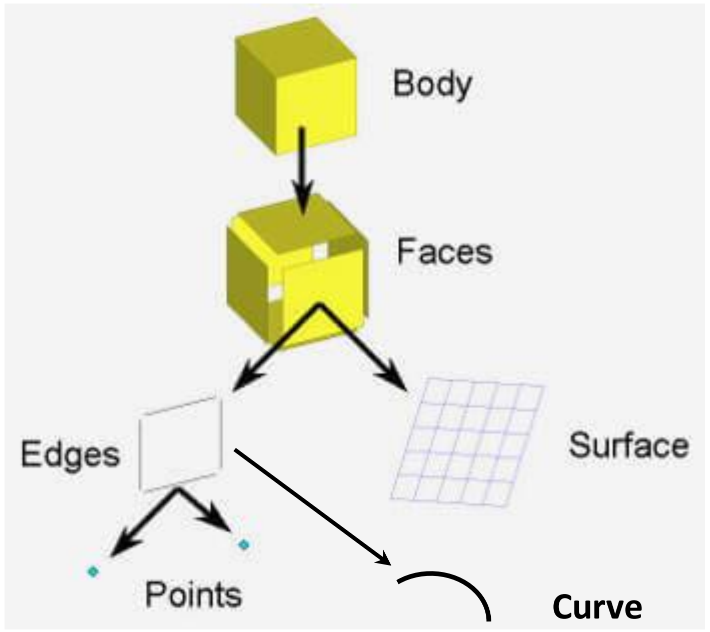
External Appearance



Voxel

Internal Composition

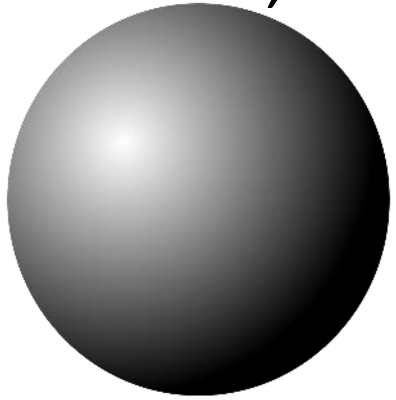
# **Boundary Representation (B-Rep)** defines the boundary of a shape into terms of analytical expression ..exclusive to Mechanical CAD systems (SolidWorks, etc)



The boundary is defined by connected collections of faces, edges and vertices that together define both a object's topology and geometry

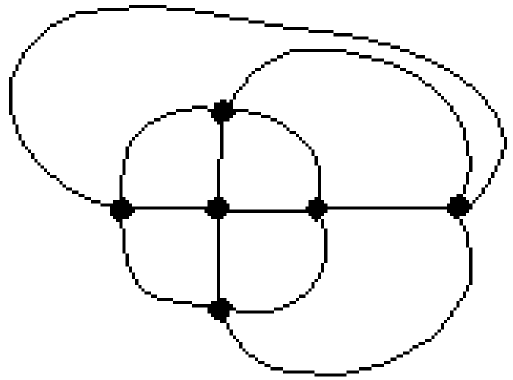
There are Python Libraries for working with BReps: **pythonOCC** provides a **python** wrapper for the **OpenCASCADE** C++ technology.
But needs some effort to install.

Geometry: what is shape of something. Example the equation of a sphere



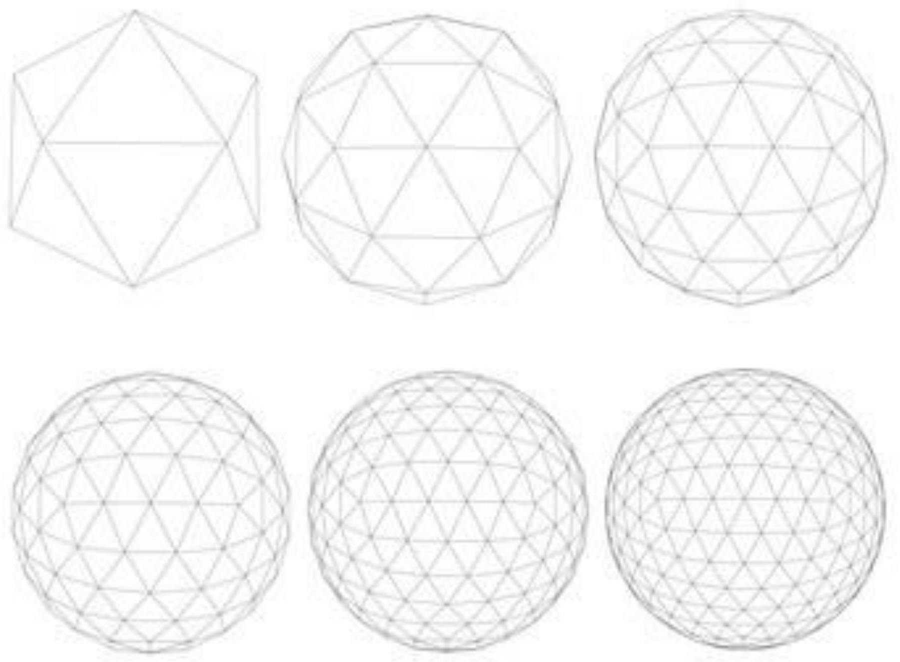
Analytical definition of a sphere is $x^2 + y^2 + z^2 = r^2$

Topology: What is connected to what. Example the Face-Edge graph of a cube.

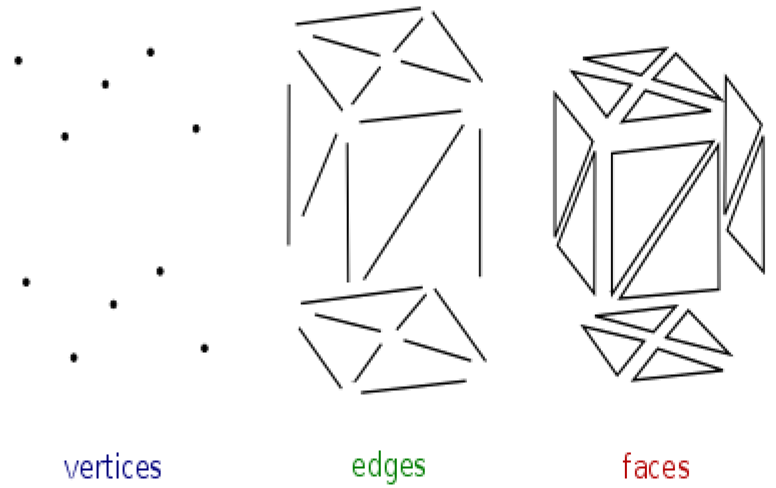


Dots are Faces, Lines are Edges

## **Meshes** also represent 3D objects as Brep but usually only have one or two types of planar face (i.e. triangles or polygons)



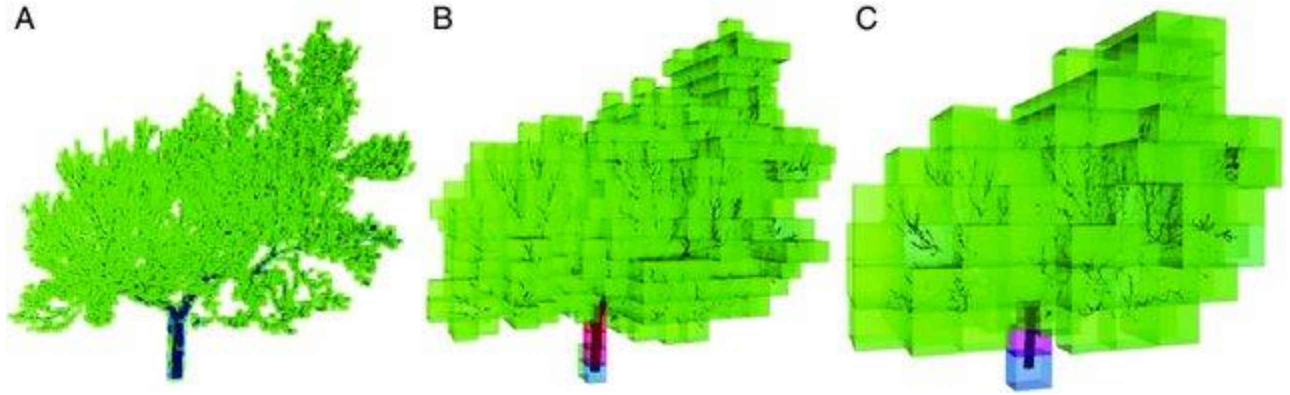
Mesh is only an approximation but the quality of the approximation can be increased until its "good enough"

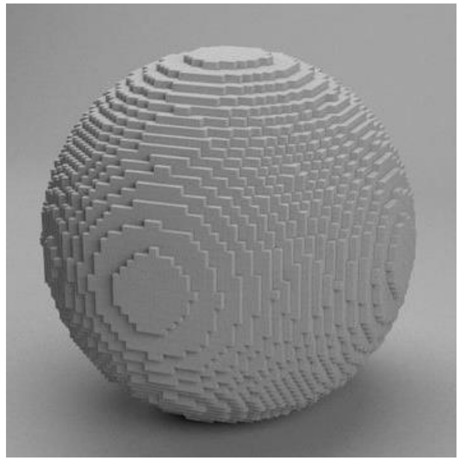


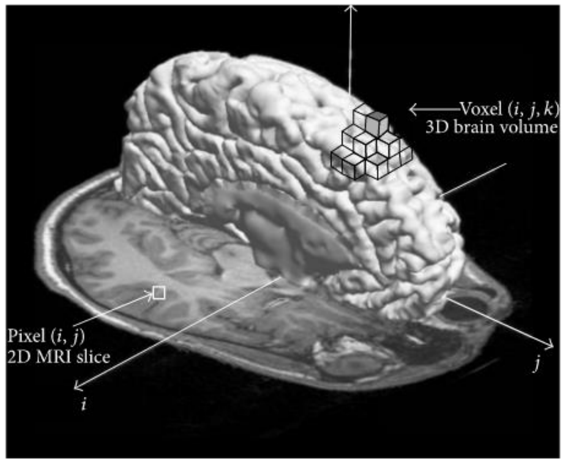
Because there are only triangles dedicated hardware (ie Graphics boards) can be used to render images with millions of facets at high speed.

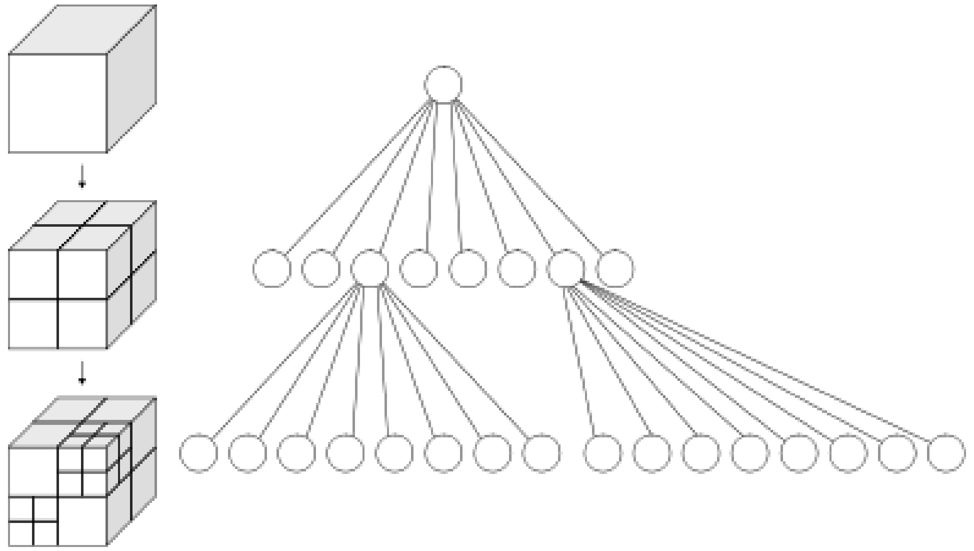
# Voxel models define the volume of space occupied by a shape

Voxel models can be represented internally as an Octree Tree. An octree is **a tree data structure in which each internal node has exactly eight children.** Octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants.

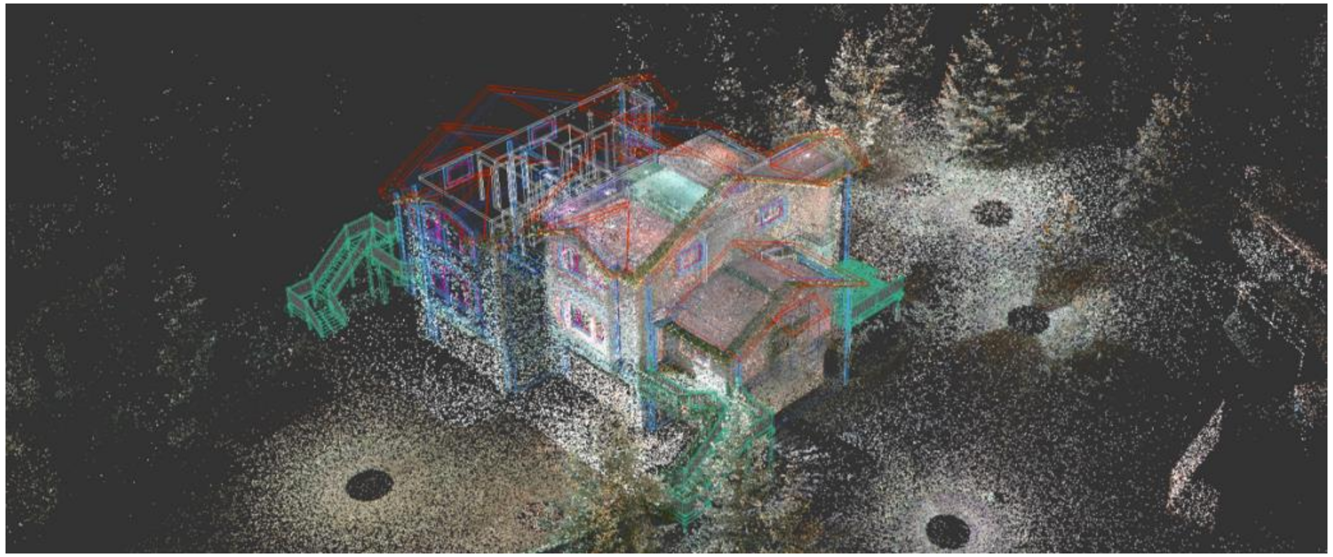




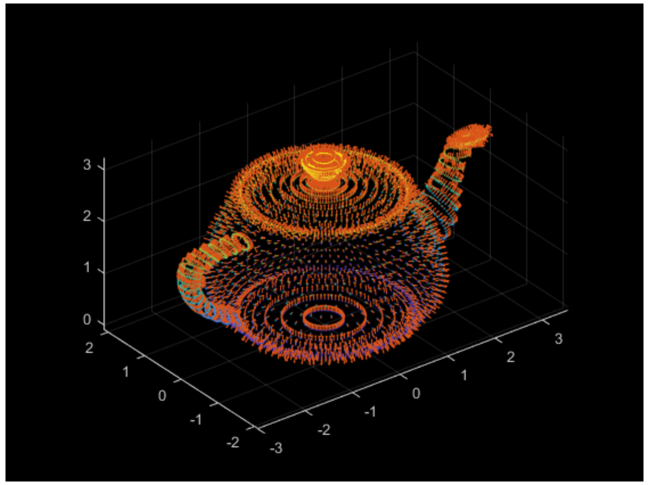


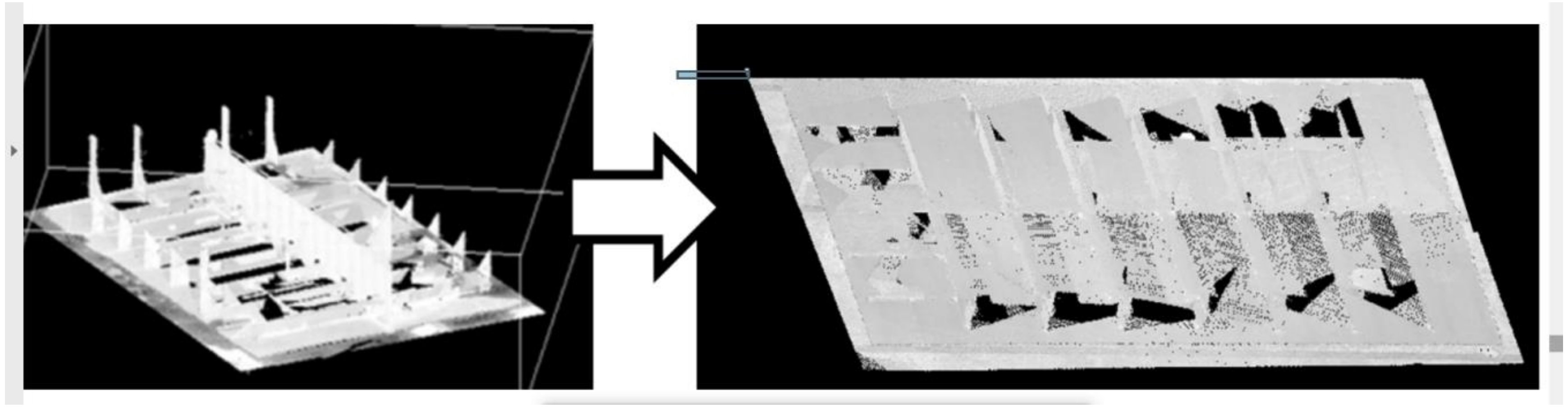


## **Point Clouds** : Are unordered lists of points

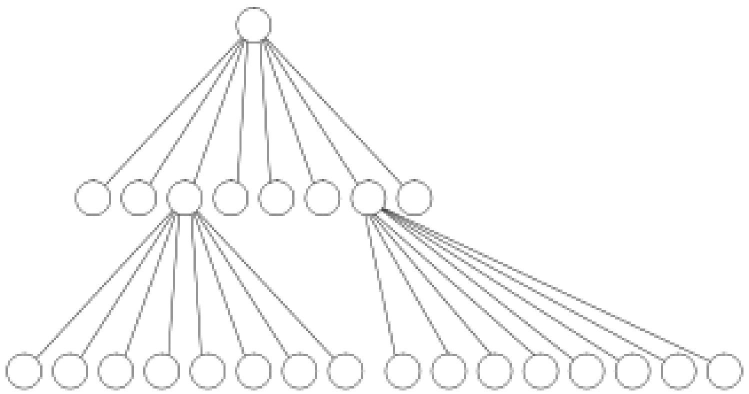


- Point clouds are difficult to work with directly so they are usually used to generate mesh or voxel models.



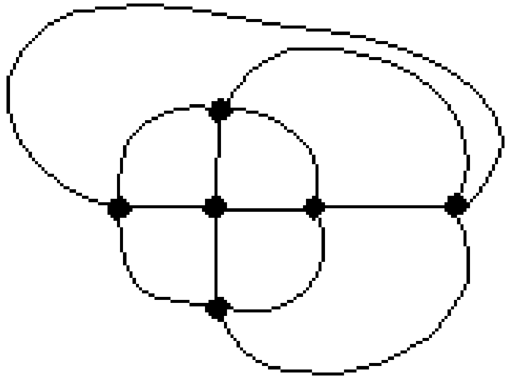


# Internally in the computer uses different data structures to represent the shape of objects



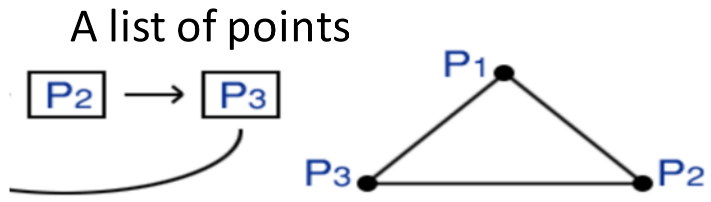
Trees

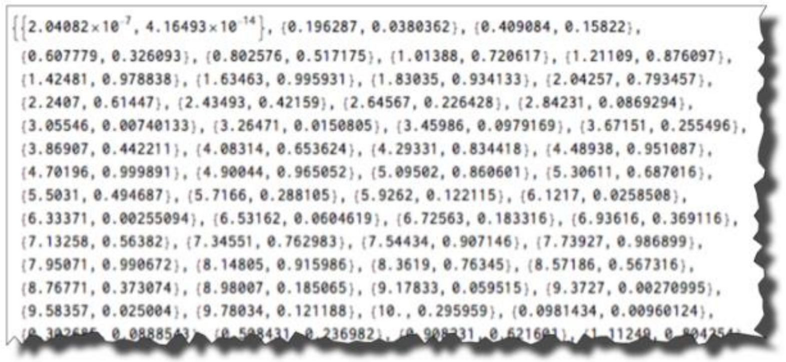
(Voxels)



Graph

(Brep)



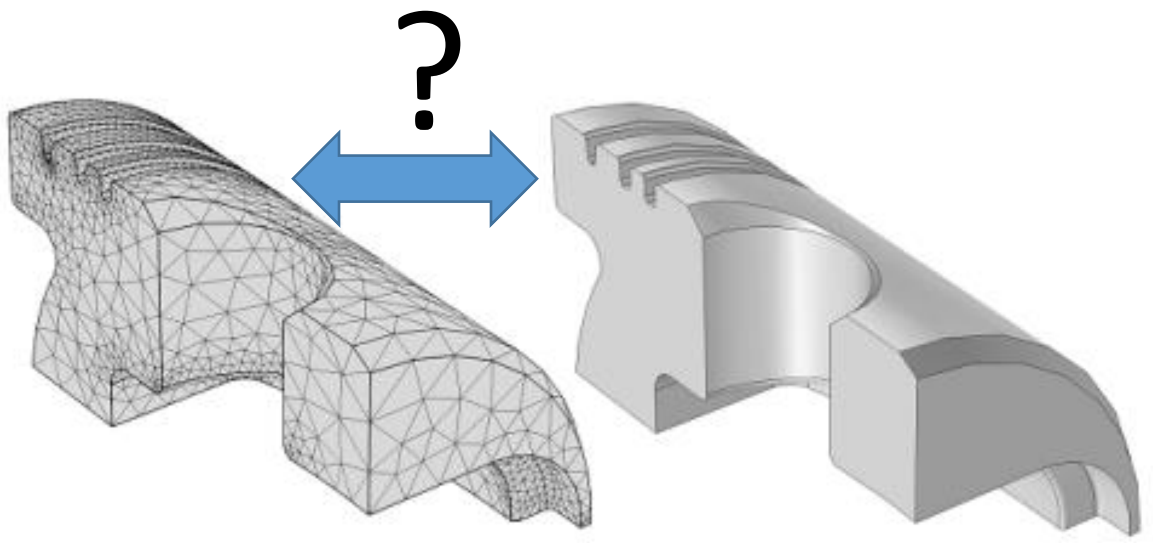


```
{{2.04082×10^-7, 4.16493×10^-14}, {0.196287, 0.0380362}, {0.409084, 0.15822},
 {0.607779, 0.326093}, {0.802576, 0.517175}, {1.01388, 0.720617}, {1.21109, 0.876097},
 {1.42481, 0.978838}, {1.63463, 0.995931}, {1.83035, 0.934133}, {2.04257, 0.793457},
 {2.2407, 0.61447}, {2.43493, 0.42159}, {2.64567, 0.226428}, {2.84231, 0.0869294},
 {3.05546, 0.00740133}, {3.26471, 0.0150805}, {3.45986, 0.0979169}, {3.67151, 0.255496},
 {3.86907, 0.442211}, {4.08314, 0.653624}, {4.29331, 0.834418}, {4.48938, 0.951087},
 {4.70196, 0.999891}, {4.90044, 0.965052}, {5.09502, 0.860601}, {5.30611, 0.687016},
 {5.5031, 0.494687}, {5.7166, 0.288105}, {5.9262, 0.122115}, {6.1217, 0.0258508},
 {6.33371, 0.00255094}, {6.53162, 0.0604619}, {6.72563, 0.183316}, {6.93616, 0.369116},
 {7.13258, 0.56382}, {7.34551, 0.762983}, {7.54434, 0.907146}, {7.73927, 0.986899},
 {7.95071, 0.990672}, {8.14805, 0.915986}, {8.3619, 0.76345}, {8.57186, 0.567316},
 {8.76771, 0.373074}, {8.98007, 0.185065}, {9.17833, 0.059515}, {9.3727, 0.00270995},
 {9.58357, 0.025004}, {9.78034, 0.121188}, {10., 0.295959}, {0.0981434, 0.00960124},
 {0.302686, 0.0888543}, {0.508431, 0.236982}, {0.908231, 0.621601}, {1.11249, 0.80425
```

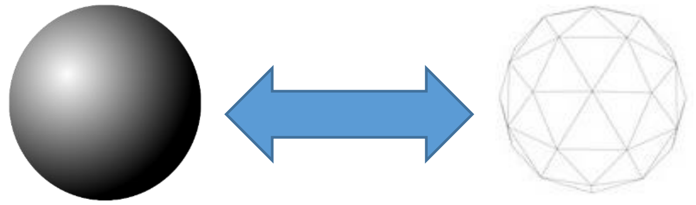
Lists
(Mesh)

Although conceptually they appear very different in practice they can be implemented in similar ways (e.g. trees are a special form of graph and graphs can be represented by lists)!

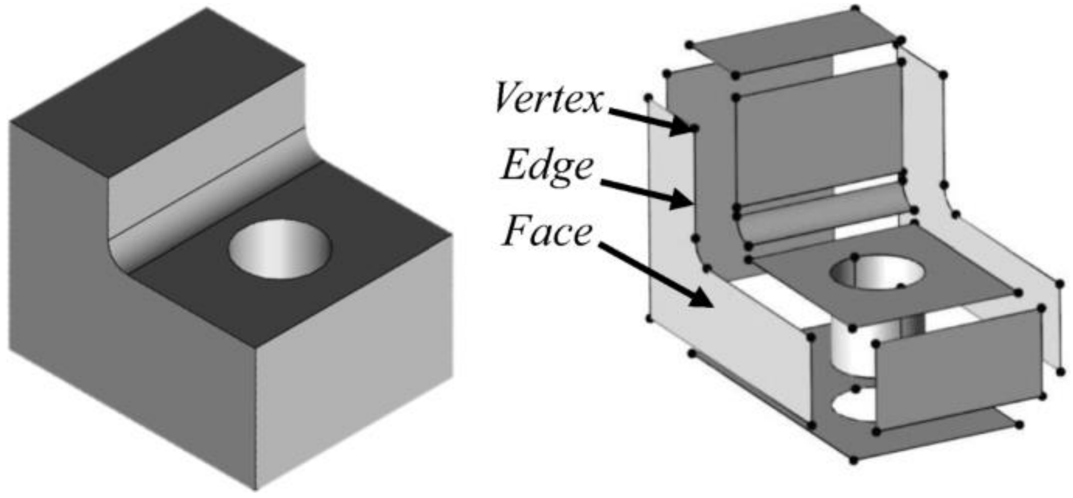
# Does the Representation matter, surely you can just translate between them



Analytical surfaces have to fitted to collections of facets (i.e. triangles). Difficult to do this reliably.

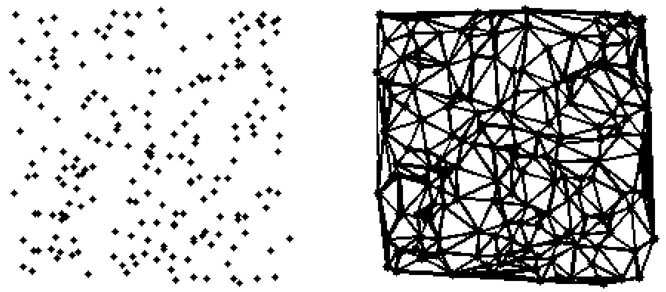


$x^2 + y^2 + z^2 = r^2$

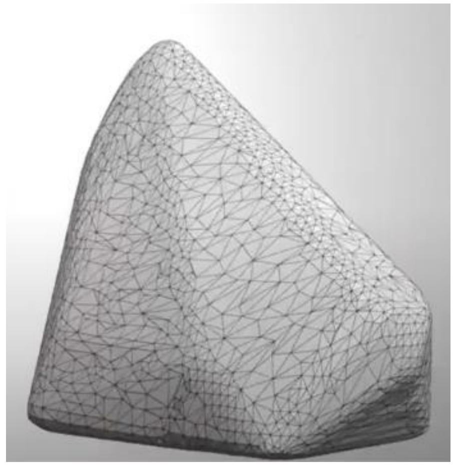


The core of mechanical CAD Systems use exact, analytic definitions of geometry. Equations of planes and circles so shapes are defined to high engineering tolerances

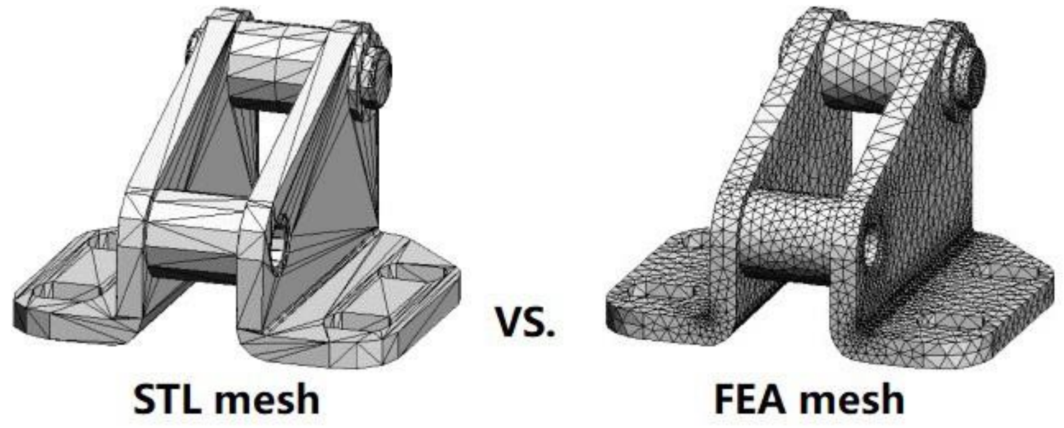
# Point Cloud => Mesh : Not always..because not all meshes are created equal!

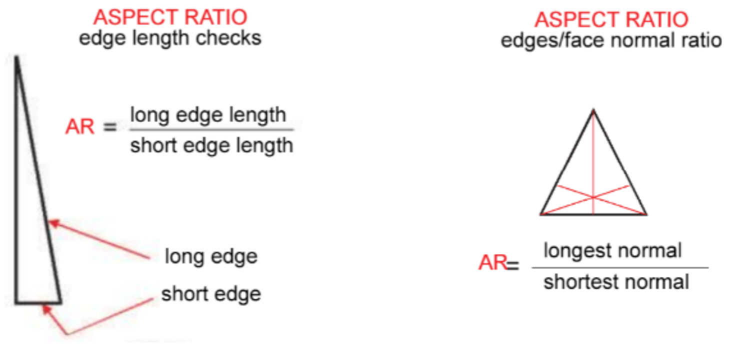


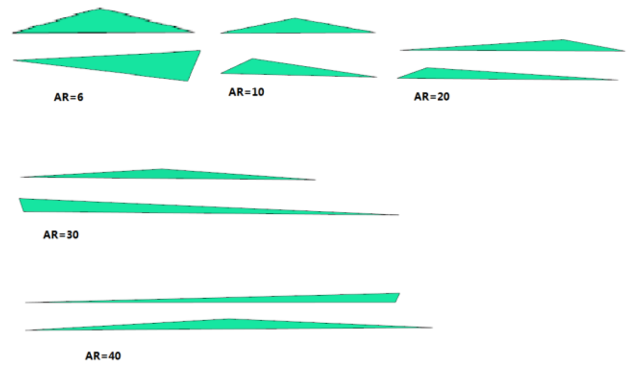
2D Points -> 2D  Mesh



3D Mesh







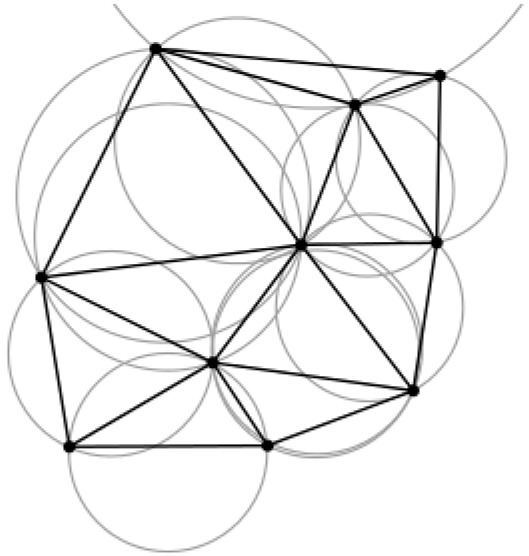
The presence of High Aspect Ratio Triangles in a mesh can cause computational problems

# Delaunay triangulation: Points to Triangles

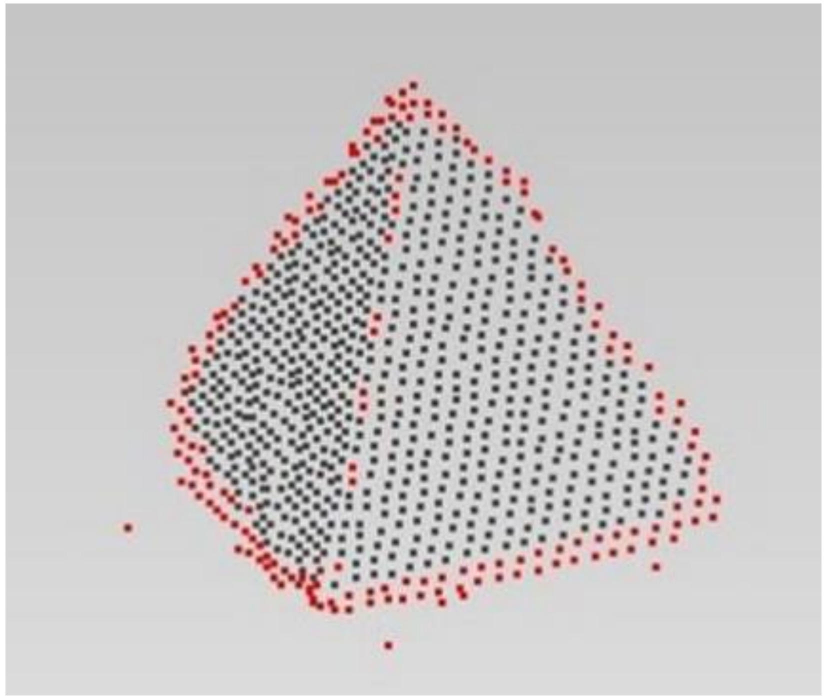
**Boris Delaunay**

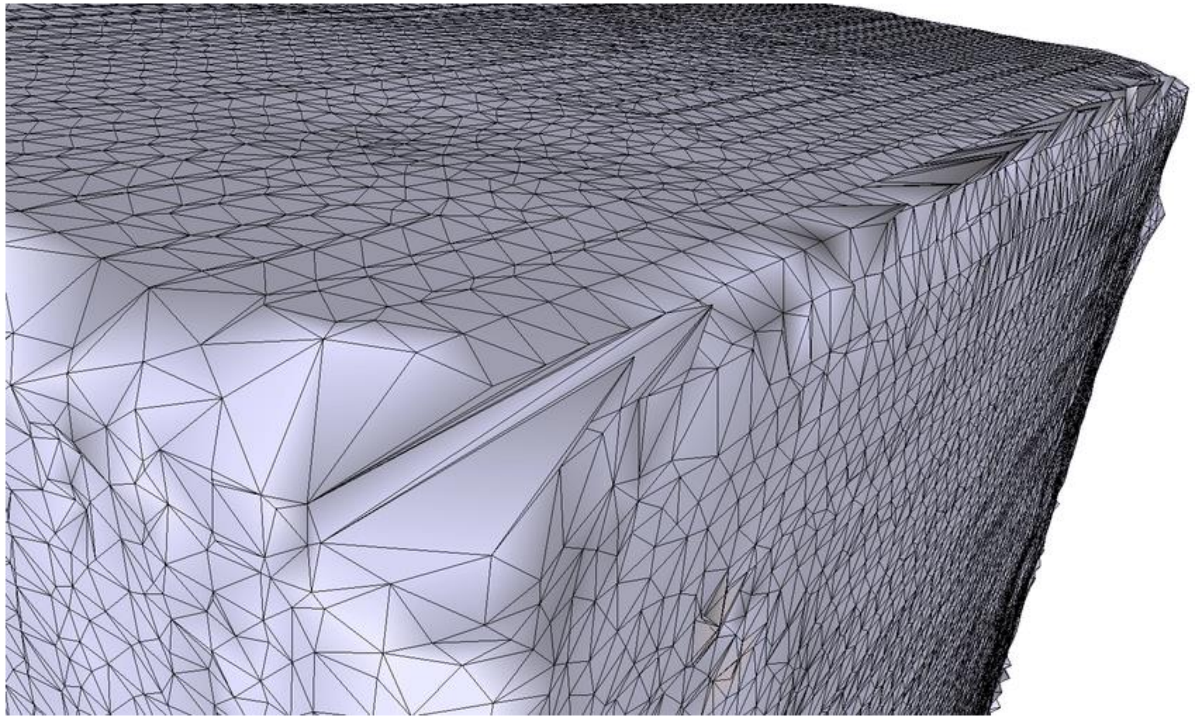
A Delaunay triangulation (also known as a Delone triangulation) for a given set P of discrete points in a general position is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). **Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation**; they tend to avoid sliver triangles. The triangulation is named after Boris Delaunay for his work on this topic from 1934

However the triangulation of the mesh is not the only problem……there are 3 main problems


A Delaunay triangulation in the plane with circumcircles shown

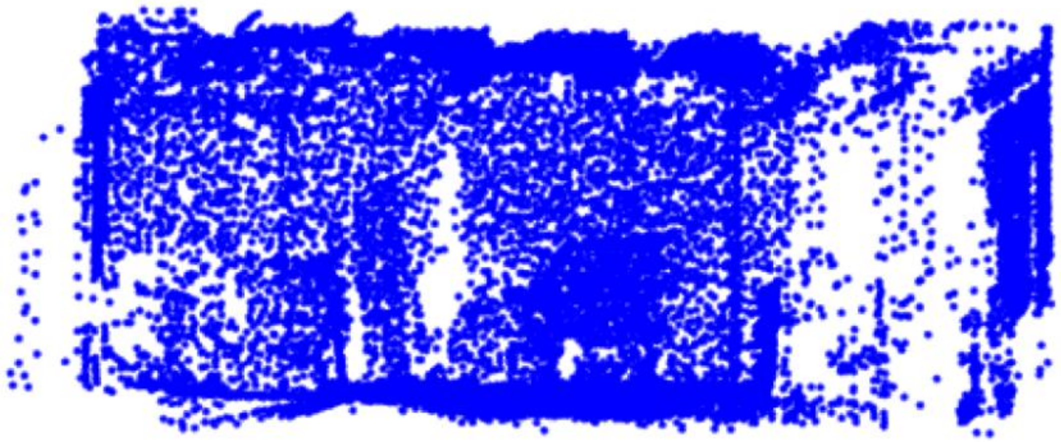
# 1) Problems Defining Sharp Edges and Corners

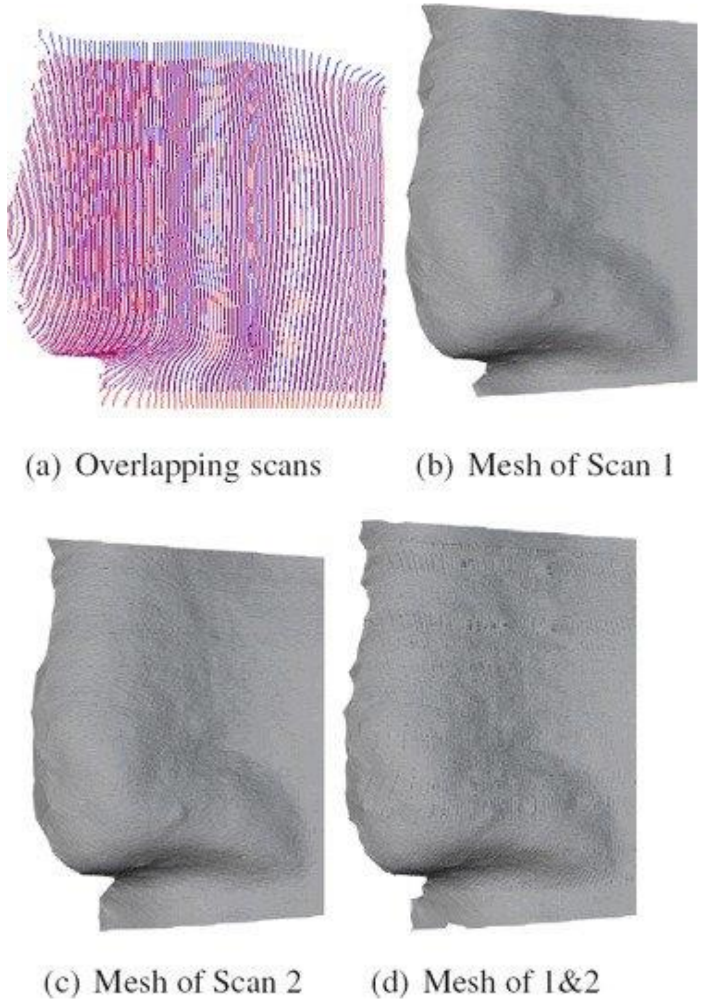




Points in Cloud will not lie exactly "on" an object's Edges or Corners

## 2) Problems arise from the distribution and “depth” of points in the cloud will often vary

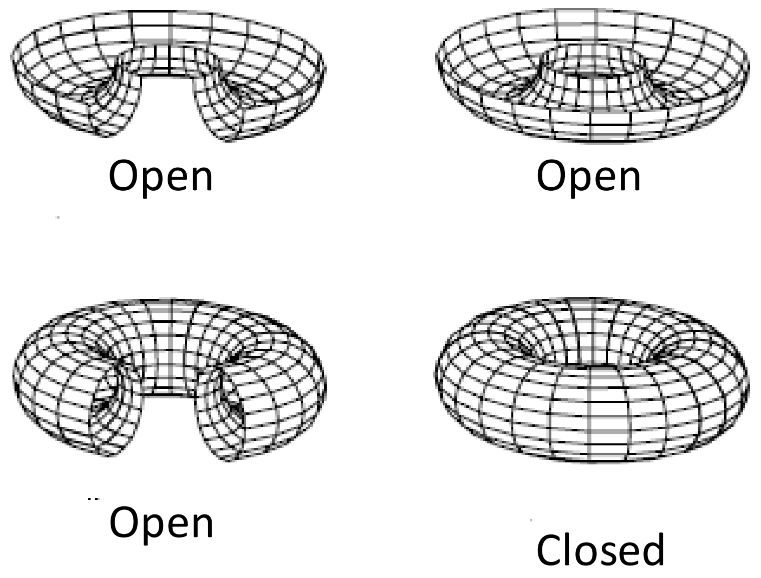


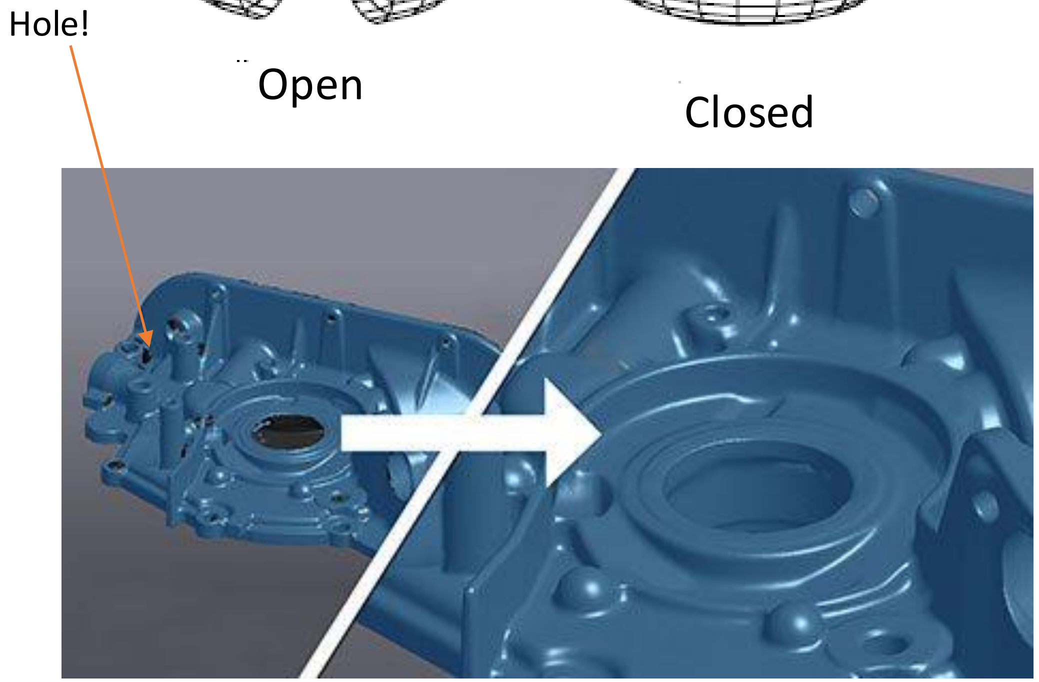


Variable speed and angle when scanning can causing the point cloud to be irregular and the resulting mesh to be rough and the worse case self-intersecting!

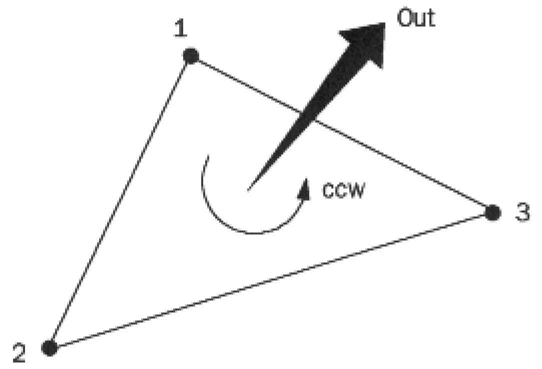
So often need to reduce the number of points in a point set

# 3) Problems with Watertight (Closed) Meshes





Watertight or closed meshes usually describe **meshes consisting of one closed surface**. In this sense, watertight meshes do not contain holes and have a **clearly defined inside.**



Meshes **can only be 3D printed** if they are Watertight (closed) meshes.

Every triangle in a closed mesh has a vector (the normal) associated with-it that points outward so the system knows which side of the triangle is material and which is air.

## Part 1 of today’s notebook does some point-cloud processing:

1. It uses the **PyntCloud** library to load a pointcloud from a .ply
2. The pointcloud is coverted to a mesh using the **Trimesh** library and checked to see if it is “closed”
3. The mesh is “open” (it has holes in it)!
4. The hole is fixed (filled) using functions from the **pymeshfix** library
5. The **open3D** library is used to reduce the number of triangles to simplify the now closed mesh

# Trimesh

Trimesh is a **pure Python** library for loading and using triangular meshes with an emphasis on watertight meshes.

See: **trimesh.org**

**Trimesh** represents a 3D object with a mesh (or network) of triangular faces (facets) using python lists

**Features**

- Import meshes from binary/ASCII STL, Wavefront OBJ, ASCII OFF, binary/ASCII PLY, GLTF/GLB 2.0, 3MF, XAML, 3DXML, etc.
- Import and export 2D or 3D vector paths from/to DXF or SVG files
- Import geometry files using the GMSH SDK if installed (BREP, STEP, IGES, INP, BDF, etc)
- Export meshes as binary STL, binary PLY, ASCII OFF, OBJ, GLTF/GLB 2.0, COLLADA, etc.
- Export meshes using the GMSH SDK if installed (Abaqus INP, Nastran BDF, etc)
- Preview meshes using pyglet or in- line in jupyter notebooks using three.js
- Automatic hashing of numpy arrays for change tracking using MD5, zlib CRC, or xxhash
- Internal caching of computed values validated from hashes
- Calculate face adjacencies, face angles, vertex defects, etc.
- Calculate cross sections, i.e. the slicing operation used in 3D printing
- Slice meshes with one or multiple arbitrary planes and return the resulting surface
- Split mesh based on face connectivity using networkx, graph-tool, or scipy.sparse
- Calculate mass properties, including volume, center of mass, moment of inertia, principal components of inertia vectors and components
- Repair simple problems with triangle winding, normals, and quad/tri holes

# Trimesh represents a 3D Point (i.e. Vertex) as a lists of three number (ie. X,Y,Z)

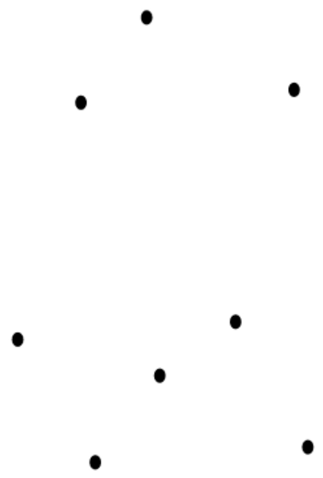
- Define pt1 = [12.45, 32.00, 23.55]
- A set of points (e.g. a point cloud) can be defined by a list of pts.
- In other words a **list of list**
- vertices = [

```
            [12.45, 32.00, 23.55]
            [24.51, 20.03, 35.52]
            [45.12, 00.32, 55.23]
            [51.24, 3.20, 52.35]
             ]
```

Point with Index number = 0

Point with Index number = 3

Each point can be identified by its index number (i.e. its position in the list



Points or Vertices

# Trimesh represents Edges (i.e. straight lines between vertices) as a list of two points

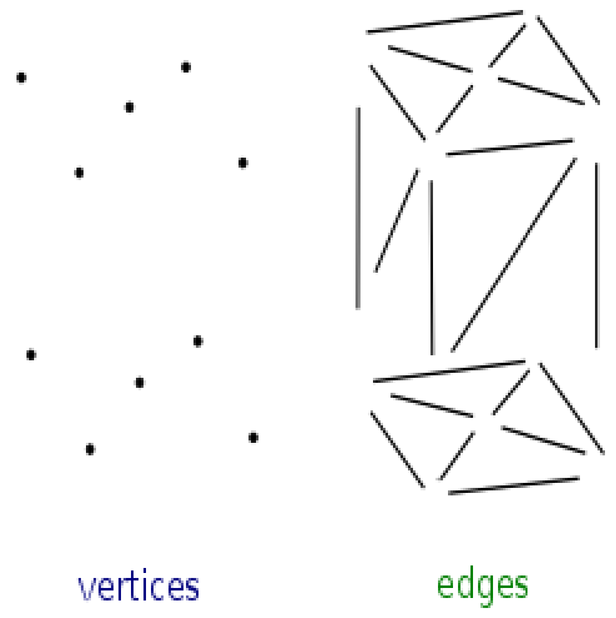
- Each edge can be represents  a list of two points (e.g. e1 = [pt1, pt2] )
- The points (e.g. pt1) can be defined by their position (i.e. index) in a list vertices.
- All the edge in a 3D mesh can be defined as a list of edges. In other words a **list of lists**
- edges = [

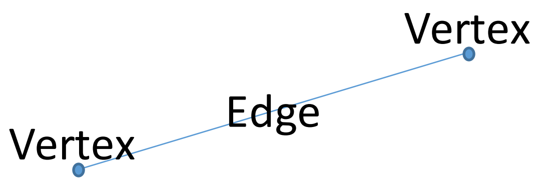
Vertex
Index
Numbers

[0, 1]
[1,2]
[2,3]
[2,0]
……
]

- vertices = [

Index = 0 → [12.45, 32.00, 23.55]
Index = 1 → [24.51, 20.03, 35.52]
Index = 2 → [45.12, 00.32, 55.23]
Index = 3 → [51.24, 3.20, 52.35]
……………
]





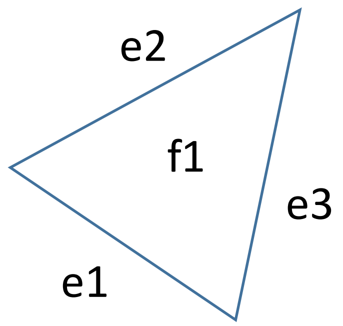
# Trimesh represents Faces as a list of the edges that surround them

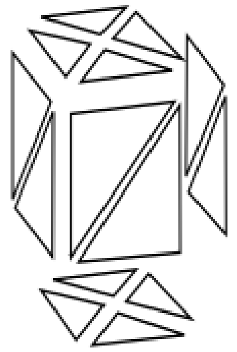
- f1 = [2,5,0]

  Edge index numbers (i.e. positions in the list of edges)

- edges = [ *// list of edges*

```
[0, 1] ← index number = 0
[1,2]  ← index number = 1
[2,3]  ← index number = 2
[2,0]  ← index number = 3
 ……
 ]
```



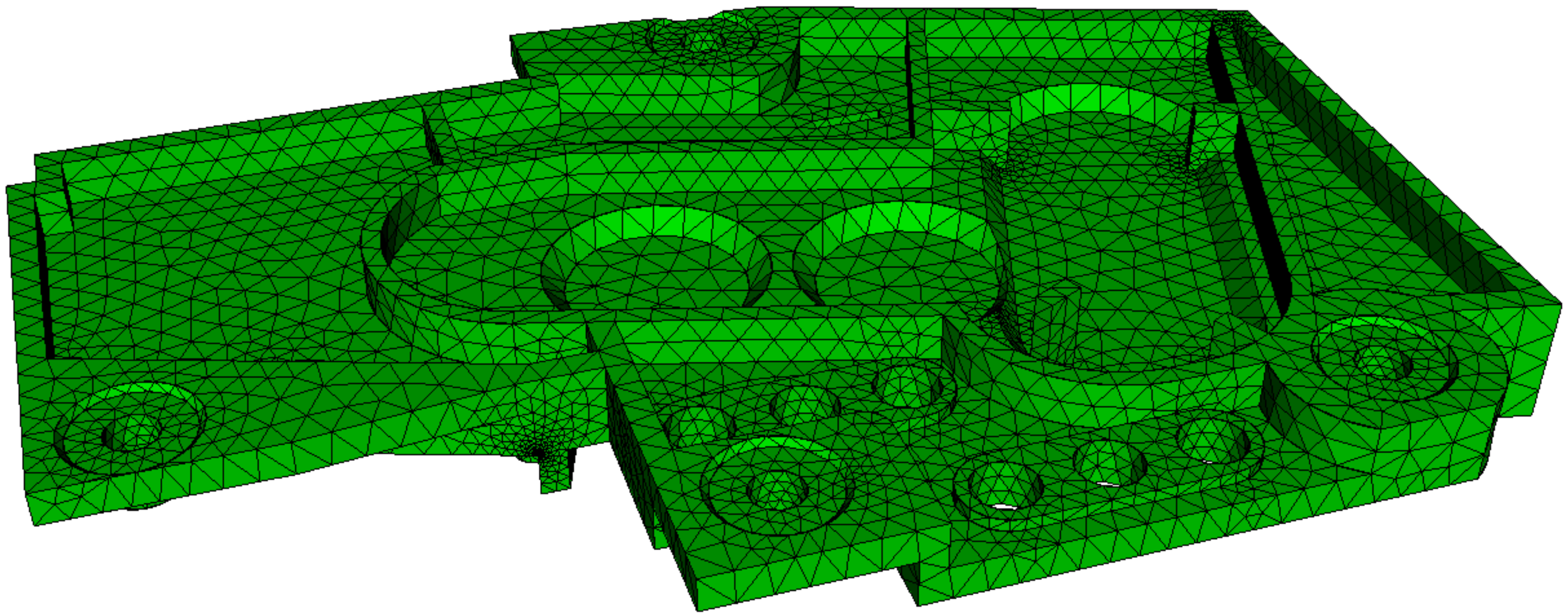


Each face can be defined by the three edges that define its boundary

f1 = [e1, e2, e3]  = a list of three edges

Each edge is defined by its index (ie position in the list edges)

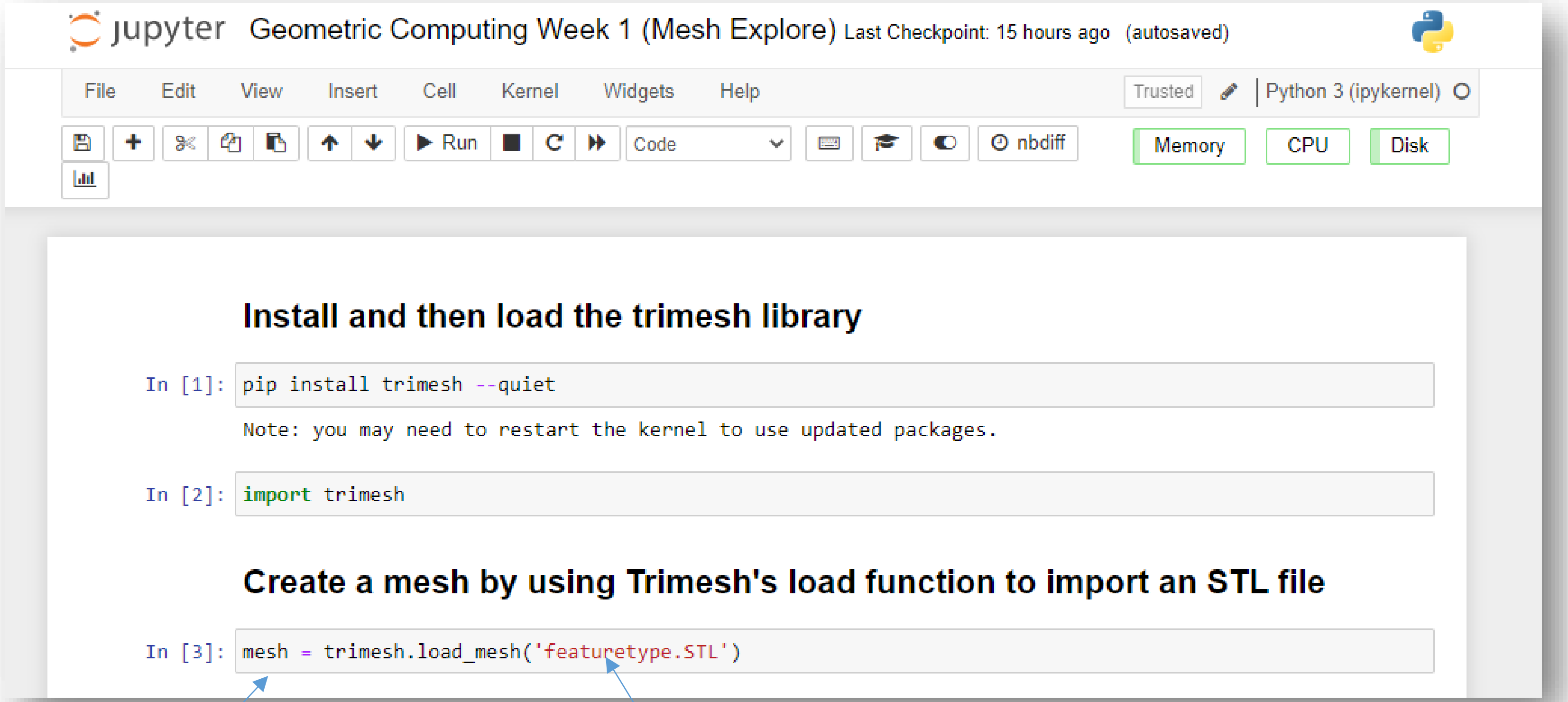
## Lets see how a faceted mesh is implemented in the trimesh library



# Using trimesh functions

Here are some simple examples of using a few of the trimesh library functions:

- load_mesh(..)
- show()
- vertices
- edges
- faces


jupyter Geometric Computing Week 1 (Mesh Explore) Last Checkpoint: 15 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
nbdiff
Memory
CPU
Disk

**Install and then load the trimesh library**

```
In [1]: pip install trimesh --quiet
        Note: you may need to restart the kernel to use updated packages.

In [2]: import trimesh
```

**Create a mesh by using Trimesh's load function to import an STL file**

```
In [3]: mesh = trimesh.load_mesh('featuretype.STL')
```

“mesh” is a variable name that could equally well be, say, “myMesh” or “FirstMesh” ….etc

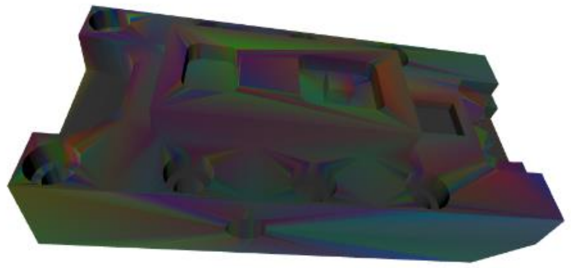
Name of the file being loaded (you can also load many other types of mesh format such as *ply and *obj files)

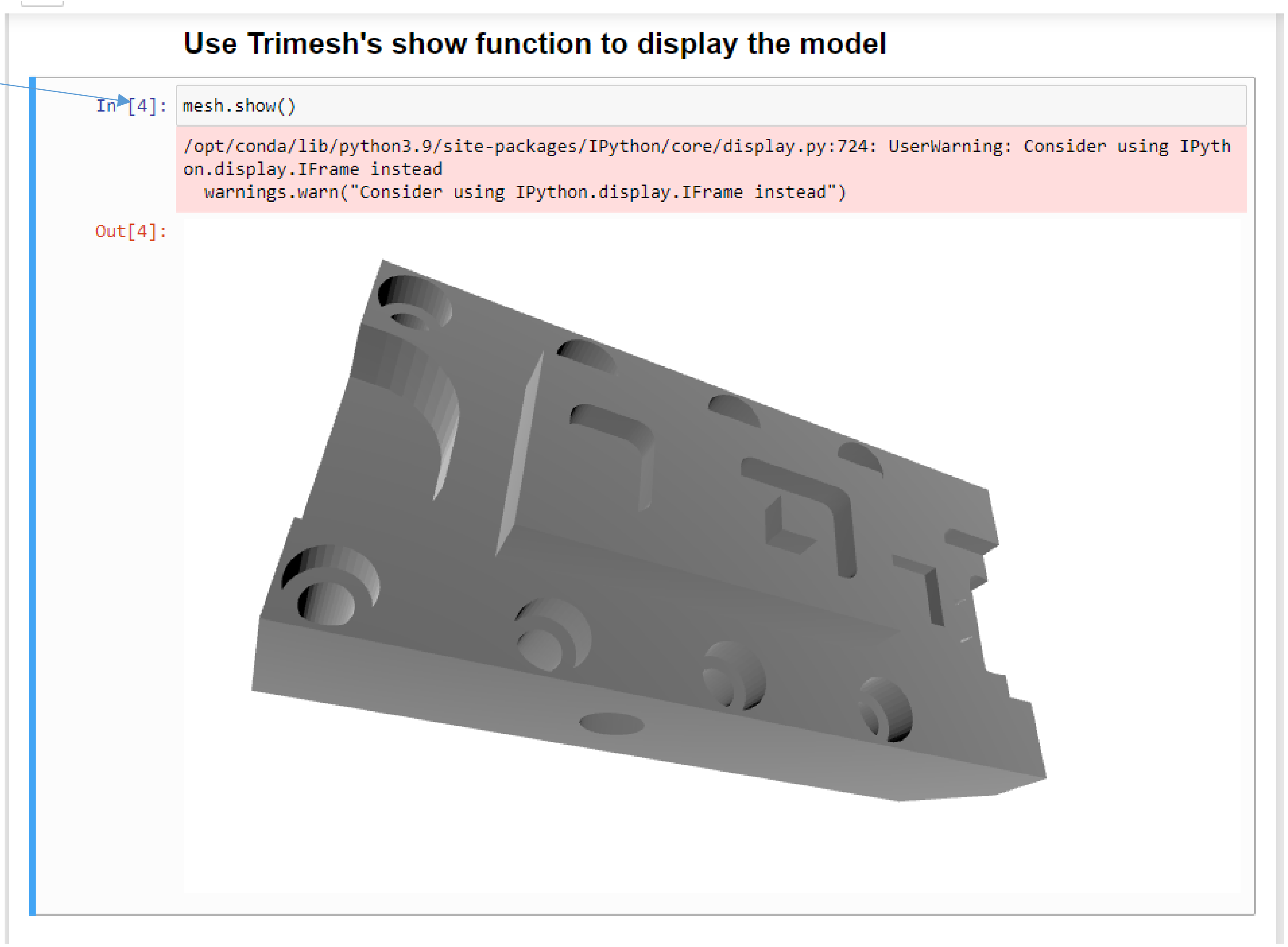
## show()

Because we are using python in a notebook it is not straight forward to use many of the 3D display methods which create separate display windows.

However Trimesh's show command provide a limited inline visualization that responds to mouse input to rotate and zoom the model.
It is also possible to color individual faces, for example:

```
for face in mesh.faces:
   mesh.visual.face_colors[face] = trimesh.visual.random_color()

mesh.show()
```





**Use Trimesh's show function to display the model**

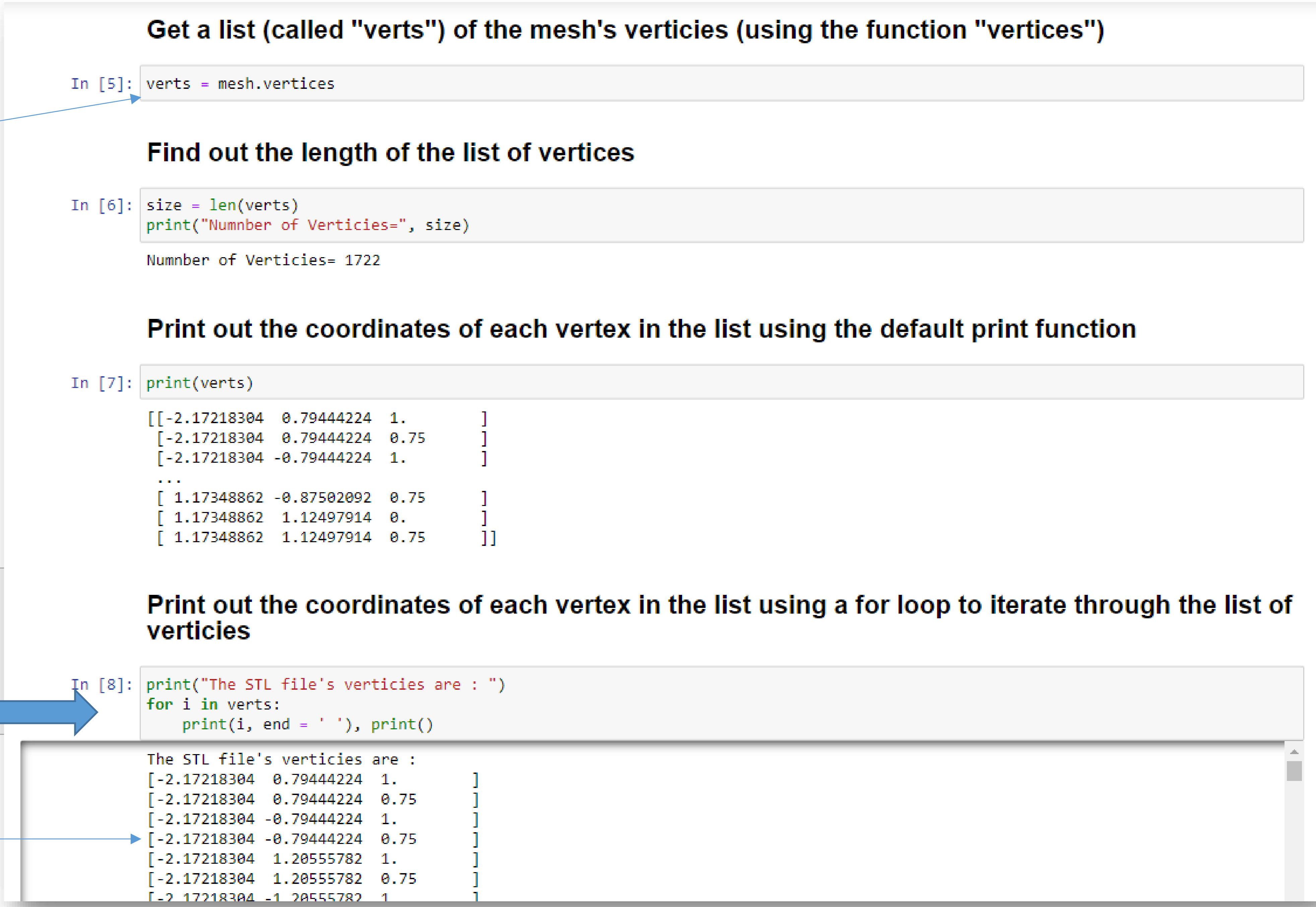
```
In [4]: mesh.show()

/opt/conda/lib/python3.9/site-packages/IPython/core/display.py:724: UserWarning: Consider using IPyth
on.display.IFrame instead
  warnings.warn("Consider using IPython.display.IFrame instead")

Out[4]:
```

verts is a python "list" containing the coordinates of all the points (i.e. vertices) in the STL file.

We can iterate (i.e. step) through the list using a "for" loop

```
# Given a list of numbers
list = [1, 3, 5, 7, 9]

# Using for loop to get each item
# in the list
for i in list:
    print(i)
```

Each "i" is a list of three numbers (i.e. x,y,z)



### Get a list (called "verts") of the mesh's verticies (using the function "vertices")

```
In [5]: verts = mesh.vertices
```

### Find out the length of the list of vertices

```
In [6]: size = len(verts)
        print("Numnber of Verticies=", size)

        Numnber of Verticies= 1722
```

### Print out the coordinates of each vertex in the list using the default print function

```
In [7]: print(verts)

        [[-2.17218304  0.79444224  1.        ]
         [-2.17218304  0.79444224  0.75      ]
         [-2.17218304 -0.79444224  1.        ]
         ...
         [ 1.17348862 -0.87502092  0.75      ]
         [ 1.17348862  1.12497914  0.        ]
         [ 1.17348862  1.12497914  0.75      ]]
```

### Print out the coordinates of each vertex in the list using a for loop to iterate through the list of verticies

```
In [8]: print("The STL file's verticies are : ")
        for i in verts:
            print(i, end = ' '), print()

        The STL file's verticies are :
        [-2.17218304  0.79444224  1.        ]
        [-2.17218304  0.79444224  0.75      ]
        [-2.17218304 -0.79444224  1.        ]
        [-2.17218304 -0.79444224  0.75      ]
        [-2.17218304  1.20555782  1.        ]
        [-2.17218304  1.20555782  0.75      ]
        [-2.17218304 -1.20555782  1.        ]
```

## Part 2 of today’s notebook does combines trimesh and the library to create a layout for 3D printing

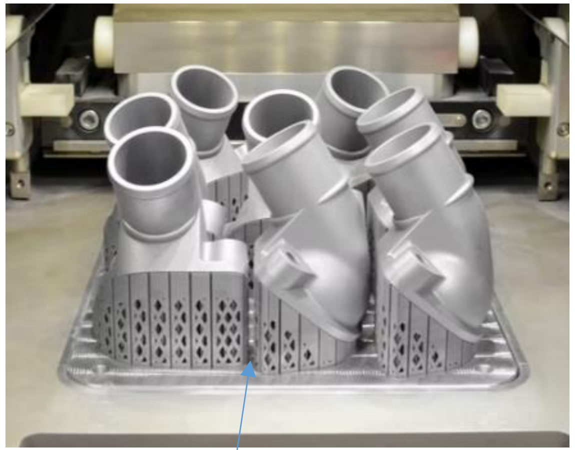
It uses Trimesh to load example shapes from *.ply files

The 3DBP library is used to packing these shapes into a cubic volume so they can be efficiently 3D Printed

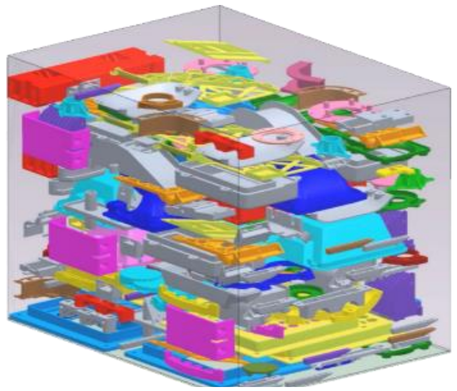
A few words about 3D packing

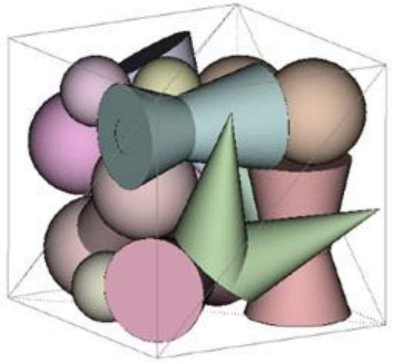
# Manufacturing Application of Geometric Computing: Packing for Additive Manufacture

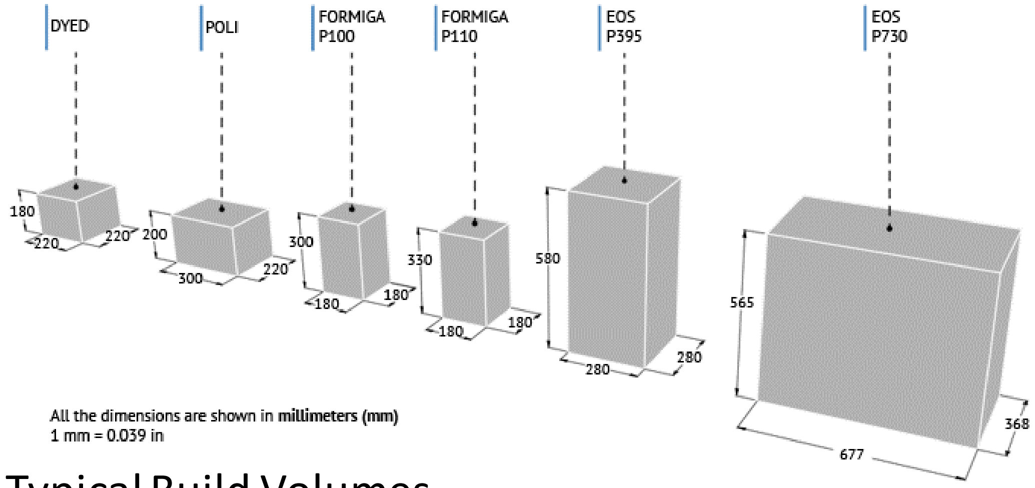
- Last Week shape packing problems in 2 and 3D were introduced.
- This week we consider similar problems but in the context of additive manufacture



Parts can be very close, **but its important they don't touch**







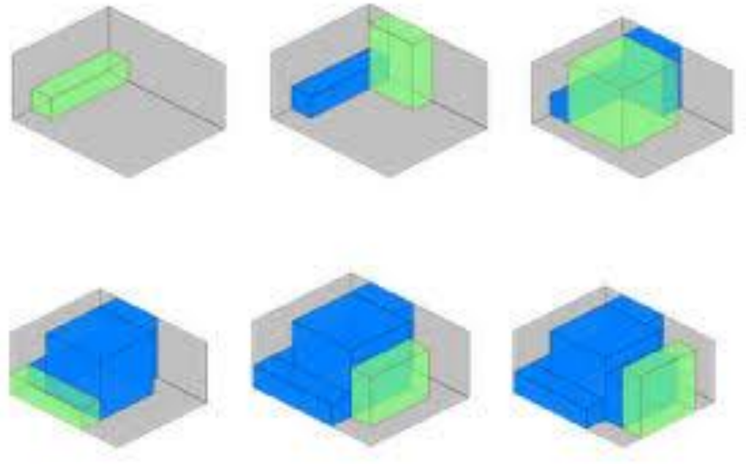
Typical Build Volumes

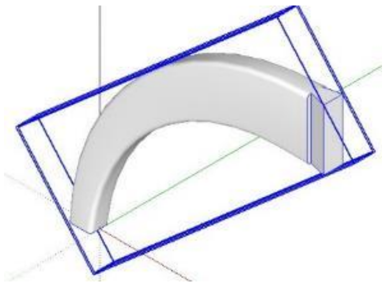
How should components be positioned in the build volume?

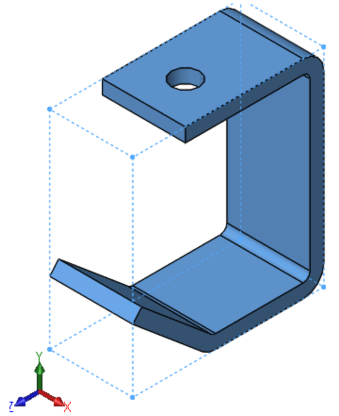
# How to arrange the build volume ?

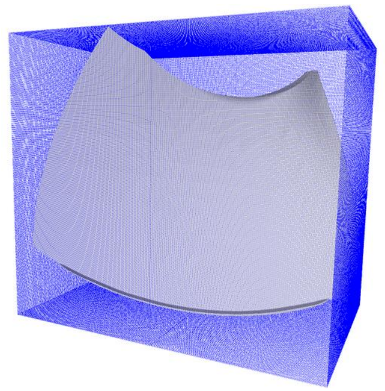
One Method:

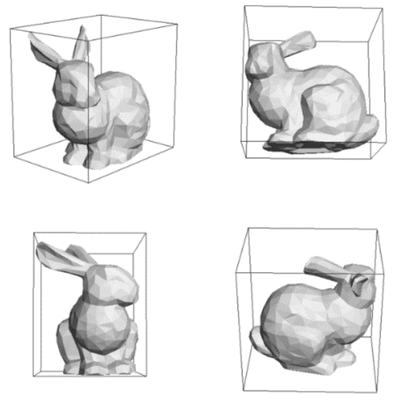
- Generate Bound Box approximations for every 3D part and use standard box packing algorithm
- What functions would you need a 3D modeller to provide to implement your algorithm?



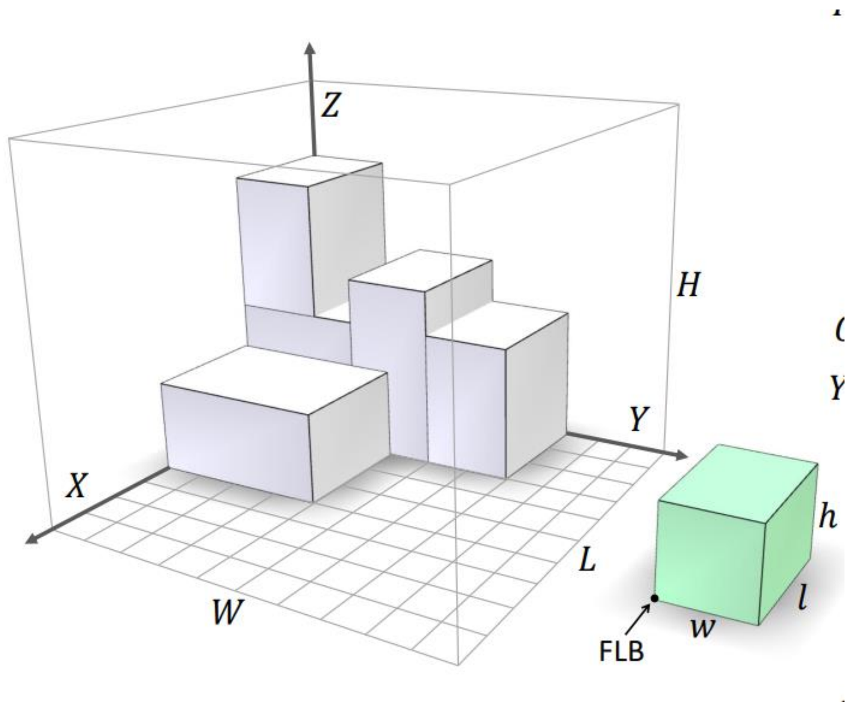








# 3DBP : 3D Bin Packing : Overview

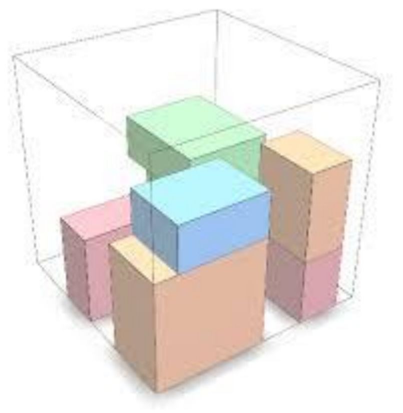


Inputs

- Information about the bins (i.e. Containers)
  - Number and size of the bins
  - Maximum weight each bin can hold
- Information about the Items (i.e. Packages)
  - Number, size and mass of each parcel

Outputs

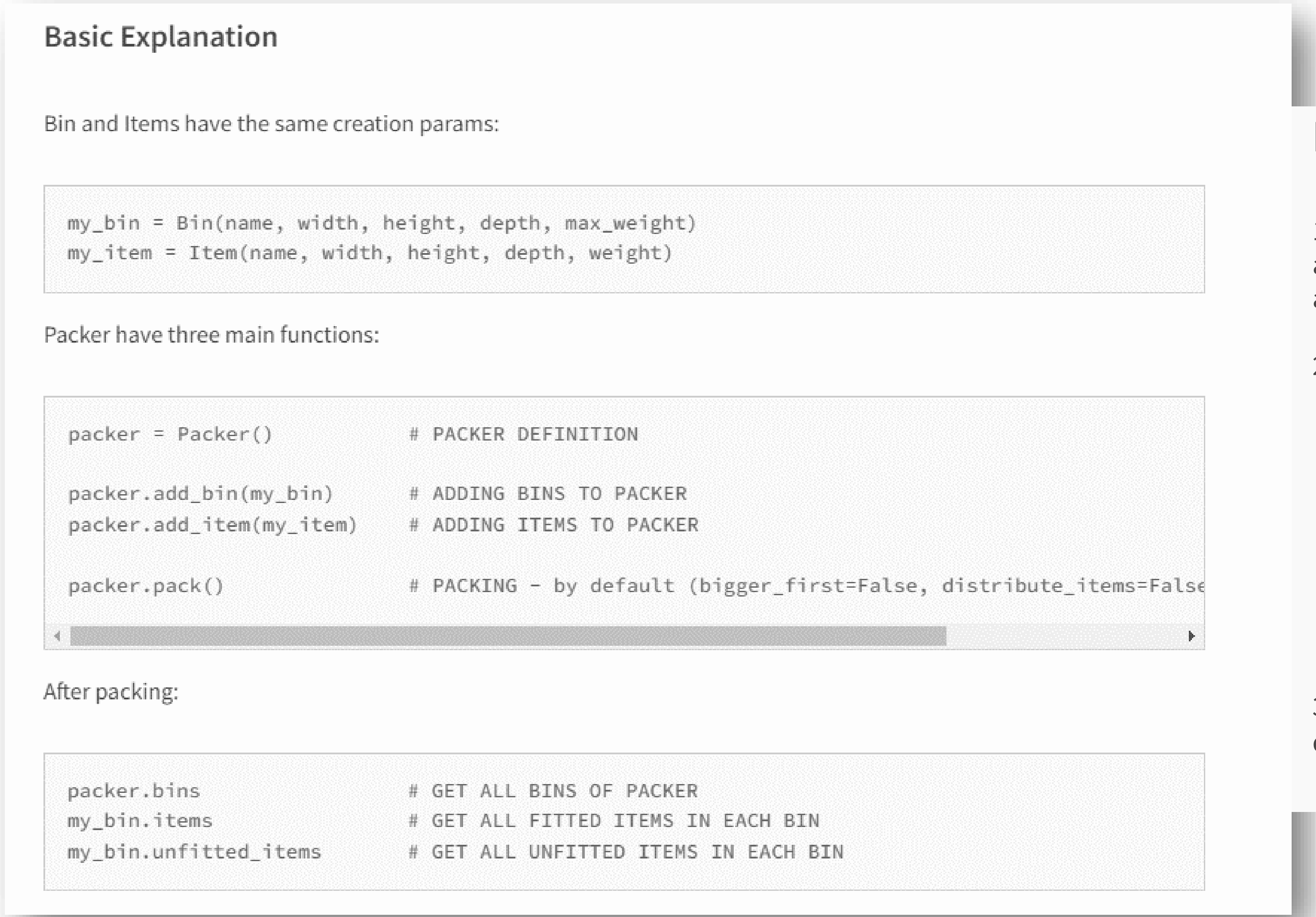
- The best (or at least good) arrangement of Items in the bins
- List of any items not in the bins



To apply 3DBP **for additive manufacture**:

- Only one bin: the 3D printers build area
- Mass or Weight information irrelevant

# 3DBP : 3D Bin Packing : Library



## Basic Explanation

Bin and Items have the same creation params:

```
my_bin = Bin(name, width, height, depth, max_weight)
my_item = Item(name, width, height, depth, weight)
```

Packer have three main functions:

```
packer = Packer()            # PACKER DEFINITION

packer.add_bin(my_bin)       # ADDING BINS TO PACKER
packer.add_item(my_item)     # ADDING ITEMS TO PACKER

packer.pack()                # PACKING - by default (bigger_first=False, distribute_items=False
```

After packing:

```
packer.bins                  # GET ALL BINS OF PACKER
my_bin.items                 # GET ALL FITTED ITEMS IN EACH BIN
my_bin.unfitted_items        # GET ALL UNFITTED ITEMS IN EACH BIN
```

## Kwargs!

1.Sorting Bins and Items: `[bigger_first=False/True]` By default all the bins and items are sorted from the smallest to the biggest, also it can be vice versa, to make the packing in such ordering.

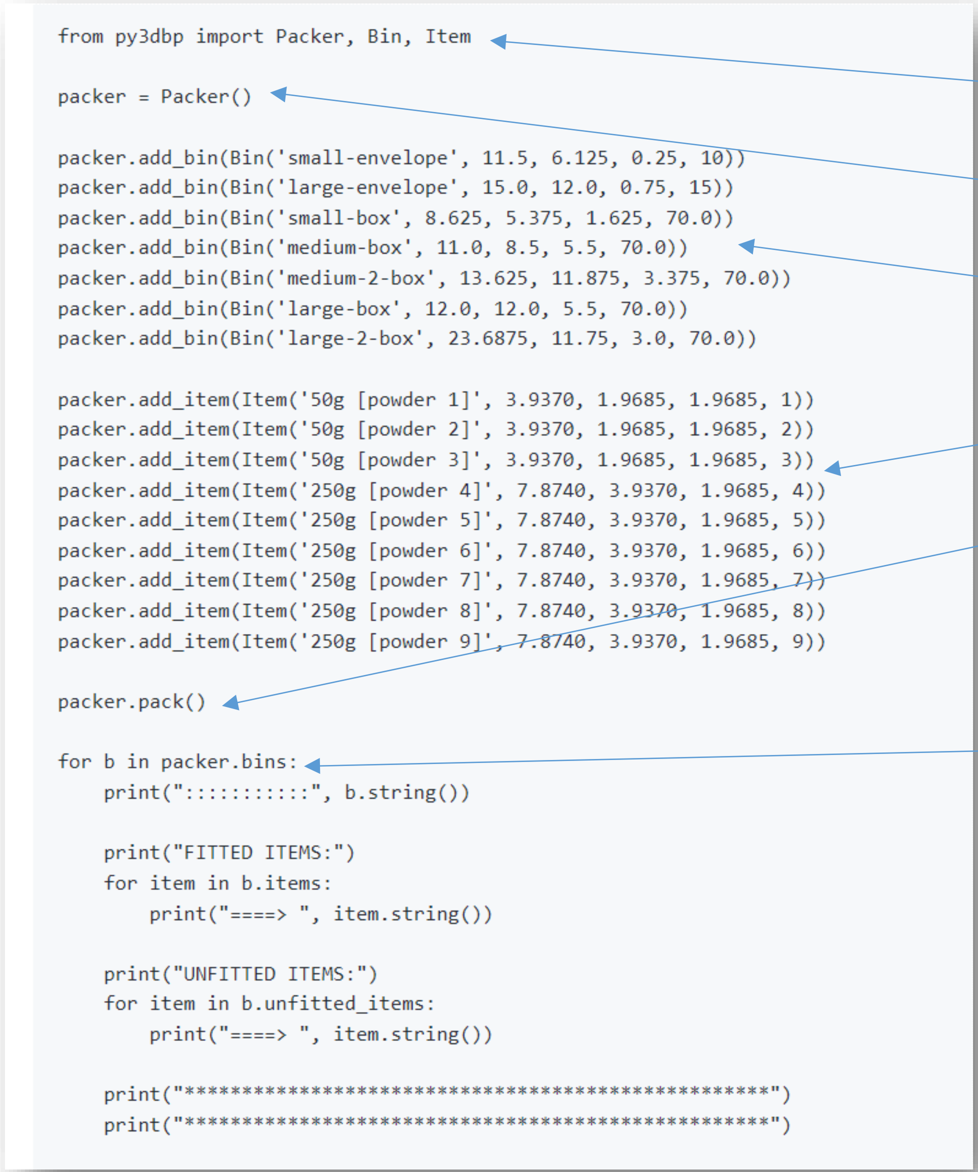
2.Item Distribution:

- `[distribute_items=True]` From a list of bins and items, put the items in the bins that at least one item be in one bin that can be fitted. That is, distribute all the items in all the bins so that they can be contained.

- `[distribute_items=False]` From a list of bins and items, try to put all the items in each bin and in the end it show per bin all the items that was fitted and the items that was not.

3.Number of decimals: `[number_of_decimals=X]` Define the limits of decimals of the inputs and the outputs. By default is 3.

# 3DBP : 3D Bin Packing : Example



```python
from py3dbp import Packer, Bin, Item

packer = Packer()

packer.add_bin(Bin('small-envelope', 11.5, 6.125, 0.25, 10))
packer.add_bin(Bin('large-envelope', 15.0, 12.0, 0.75, 15))
packer.add_bin(Bin('small-box', 8.625, 5.375, 1.625, 70.0))
packer.add_bin(Bin('medium-box', 11.0, 8.5, 5.5, 70.0))
packer.add_bin(Bin('medium-2-box', 13.625, 11.875, 3.375, 70.0))
packer.add_bin(Bin('large-box', 12.0, 12.0, 5.5, 70.0))
packer.add_bin(Bin('large-2-box', 23.6875, 11.75, 3.0, 70.0))

packer.add_item(Item('50g [powder 1]', 3.9370, 1.9685, 1.9685, 1))
packer.add_item(Item('50g [powder 2]', 3.9370, 1.9685, 1.9685, 2))
packer.add_item(Item('50g [powder 3]', 3.9370, 1.9685, 1.9685, 3))
packer.add_item(Item('250g [powder 4]', 7.8740, 3.9370, 1.9685, 4))
packer.add_item(Item('250g [powder 5]', 7.8740, 3.9370, 1.9685, 5))
packer.add_item(Item('250g [powder 6]', 7.8740, 3.9370, 1.9685, 6))
packer.add_item(Item('250g [powder 7]', 7.8740, 3.9370, 1.9685, 7))
packer.add_item(Item('250g [powder 8]', 7.8740, 3.9370, 1.9685, 8))
packer.add_item(Item('250g [powder 9]', 7.8740, 3.9370, 1.9685, 9))

packer.pack()

for b in packer.bins:
    print(":::::::::::", b.string())

    print("FITTED ITEMS:")
    for item in b.items:
        print("====> ", item.string())

    print("UNFITTED ITEMS:")
    for item in b.unfitted_items:
        print("====> ", item.string())

    print("****************************************************")
    print("****************************************************")
```

After: pip install 3dbp

Create a “Packer” object (data & functions)

Define & Add **Bins** to the packer

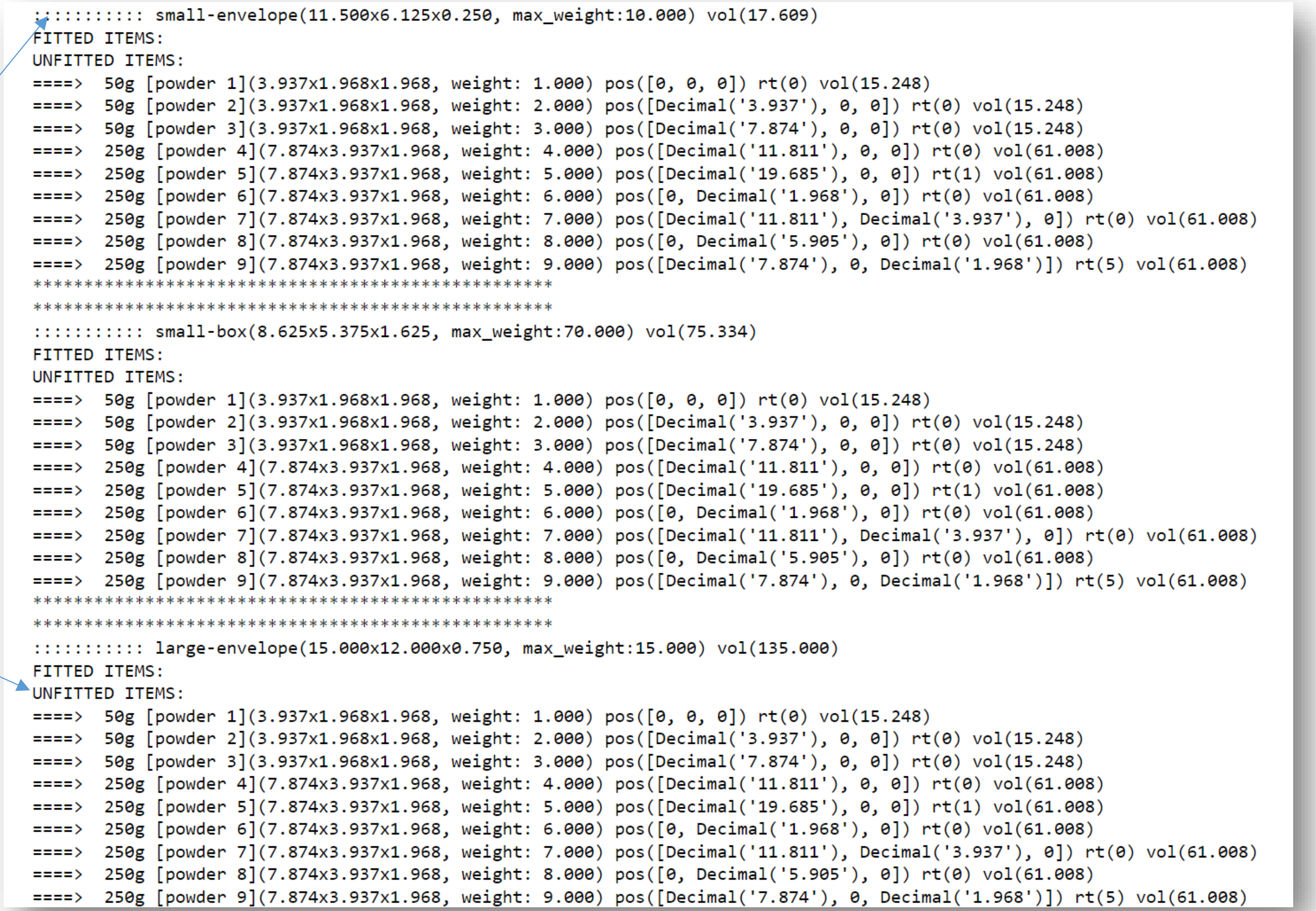
Define & Add **Items** to the packet

Get the packer to do the packing!

Print out the results one bin at a time (i.e. what items are in what bins in what positions)

# Results show details of each bin

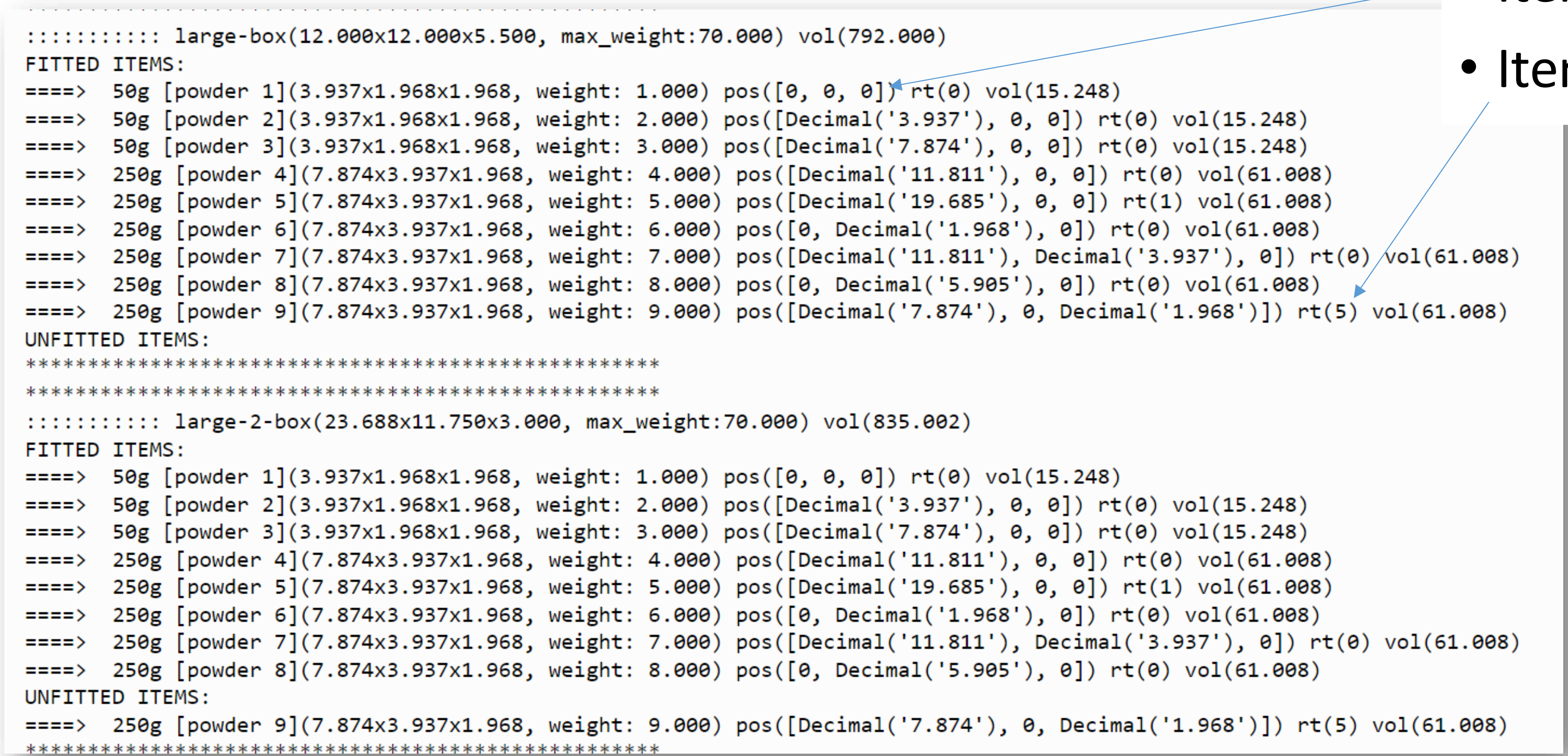
- Bin Name and Details
- Lists of items in the bin
- List of items not in the bin



```
::::::::::: small-envelope(11.500x6.125x0.250, max_weight:10.000) vol(17.609)
FITTED ITEMS:
UNFITTED ITEMS:
====>  50g [powder 1](3.937x1.968x1.968, weight: 1.000) pos([0, 0, 0]) rt(0) vol(15.248)
====>  50g [powder 2](3.937x1.968x1.968, weight: 2.000) pos([Decimal('3.937'), 0, 0]) rt(0) vol(15.248)
====>  50g [powder 3](3.937x1.968x1.968, weight: 3.000) pos([Decimal('7.874'), 0, 0]) rt(0) vol(15.248)
====>  250g [powder 4](7.874x3.937x1.968, weight: 4.000) pos([Decimal('11.811'), 0, 0]) rt(0) vol(61.008)
====>  250g [powder 5](7.874x3.937x1.968, weight: 5.000) pos([Decimal('19.685'), 0, 0]) rt(1) vol(61.008)
====>  250g [powder 6](7.874x3.937x1.968, weight: 6.000) pos([0, Decimal('1.968'), 0]) rt(0) vol(61.008)
====>  250g [powder 7](7.874x3.937x1.968, weight: 7.000) pos([Decimal('11.811'), Decimal('3.937'), 0]) rt(0) vol(61.008)
====>  250g [powder 8](7.874x3.937x1.968, weight: 8.000) pos([0, Decimal('5.905'), 0]) rt(0) vol(61.008)
====>  250g [powder 9](7.874x3.937x1.968, weight: 9.000) pos([Decimal('7.874'), 0, Decimal('1.968')]) rt(5) vol(61.008)
****************************************************
****************************************************
::::::::::: small-box(8.625x5.375x1.625, max_weight:70.000) vol(75.334)
FITTED ITEMS:
UNFITTED ITEMS:
====>  50g [powder 1](3.937x1.968x1.968, weight: 1.000) pos([0, 0, 0]) rt(0) vol(15.248)
====>  50g [powder 2](3.937x1.968x1.968, weight: 2.000) pos([Decimal('3.937'), 0, 0]) rt(0) vol(15.248)
====>  50g [powder 3](3.937x1.968x1.968, weight: 3.000) pos([Decimal('7.874'), 0, 0]) rt(0) vol(15.248)
====>  250g [powder 4](7.874x3.937x1.968, weight: 4.000) pos([Decimal('11.811'), 0, 0]) rt(0) vol(61.008)
====>  250g [powder 5](7.874x3.937x1.968, weight: 5.000) pos([Decimal('19.685'), 0, 0]) rt(1) vol(61.008)
====>  250g [powder 6](7.874x3.937x1.968, weight: 6.000) pos([0, Decimal('1.968'), 0]) rt(0) vol(61.008)
====>  250g [powder 7](7.874x3.937x1.968, weight: 7.000) pos([Decimal('11.811'), Decimal('3.937'), 0]) rt(0) vol(61.008)
====>  250g [powder 8](7.874x3.937x1.968, weight: 8.000) pos([0, Decimal('5.905'), 0]) rt(0) vol(61.008)
====>  250g [powder 9](7.874x3.937x1.968, weight: 9.000) pos([Decimal('7.874'), 0, Decimal('1.968')]) rt(5) vol(61.008)
****************************************************
****************************************************
::::::::::: large-envelope(15.000x12.000x0.750, max_weight:15.000) vol(135.000)
FITTED ITEMS:
UNFITTED ITEMS:
====>  50g [powder 1](3.937x1.968x1.968, weight: 1.000) pos([0, 0, 0]) rt(0) vol(15.248)
====>  50g [powder 2](3.937x1.968x1.968, weight: 2.000) pos([Decimal('3.937'), 0, 0]) rt(0) vol(15.248)
====>  50g [powder 3](3.937x1.968x1.968, weight: 3.000) pos([Decimal('7.874'), 0, 0]) rt(0) vol(15.248)
====>  250g [powder 4](7.874x3.937x1.968, weight: 4.000) pos([Decimal('11.811'), 0, 0]) rt(0) vol(61.008)
====>  250g [powder 5](7.874x3.937x1.968, weight: 5.000) pos([Decimal('19.685'), 0, 0]) rt(1) vol(61.008)
====>  250g [powder 6](7.874x3.937x1.968, weight: 6.000) pos([0, Decimal('1.968'), 0]) rt(0) vol(61.008)
====>  250g [powder 7](7.874x3.937x1.968, weight: 7.000) pos([Decimal('11.811'), Decimal('3.937'), 0]) rt(0) vol(61.008)
====>  250g [powder 8](7.874x3.937x1.968, weight: 8.000) pos([0, Decimal('5.905'), 0]) rt(0) vol(61.008)
====>  250g [powder 9](7.874x3.937x1.968, weight: 9.000) pos([Decimal('7.874'), 0, Decimal('1.968')]) rt(5) vol(61.008)
```

# The “large-box” can hold everything

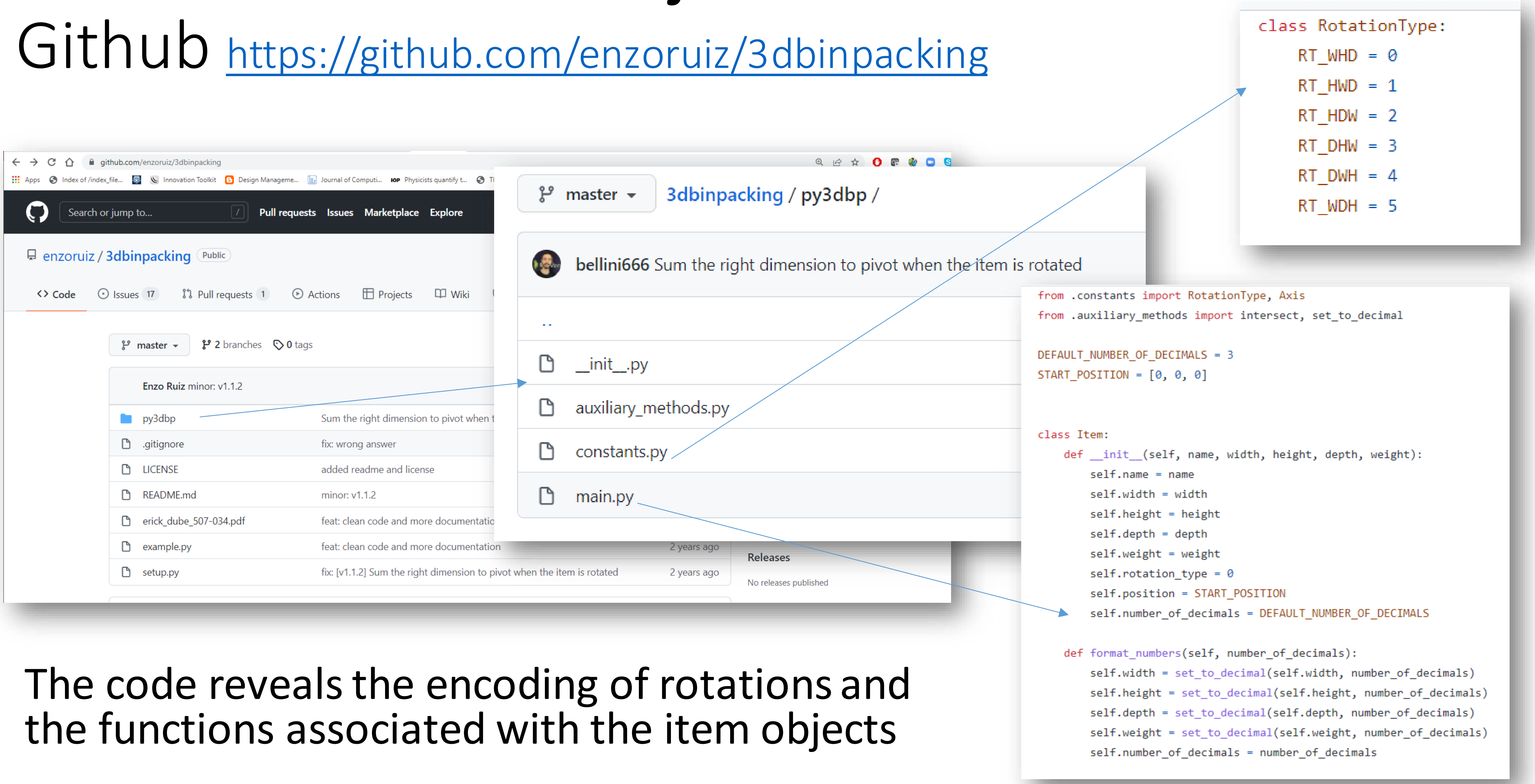
- Item Position
- Item Orientation



```
:::::::::: large-box(12.000x12.000x5.500, max_weight:70.000) vol(792.000)
FITTED ITEMS:
====>  50g [powder 1](3.937x1.968x1.968, weight: 1.000) pos([0, 0, 0]) rt(0) vol(15.248)
====>  50g [powder 2](3.937x1.968x1.968, weight: 2.000) pos([Decimal('3.937'), 0, 0]) rt(0) vol(15.248)
====>  50g [powder 3](3.937x1.968x1.968, weight: 3.000) pos([Decimal('7.874'), 0, 0]) rt(0) vol(15.248)
====>  250g [powder 4](7.874x3.937x1.968, weight: 4.000) pos([Decimal('11.811'), 0, 0]) rt(0) vol(61.008)
====>  250g [powder 5](7.874x3.937x1.968, weight: 5.000) pos([Decimal('19.685'), 0, 0]) rt(1) vol(61.008)
====>  250g [powder 6](7.874x3.937x1.968, weight: 6.000) pos([0, Decimal('1.968'), 0]) rt(0) vol(61.008)
====>  250g [powder 7](7.874x3.937x1.968, weight: 7.000) pos([Decimal('11.811'), Decimal('3.937'), 0]) rt(0) vol(61.008)
====>  250g [powder 8](7.874x3.937x1.968, weight: 8.000) pos([0, Decimal('5.905'), 0]) rt(0) vol(61.008)
====>  250g [powder 9](7.874x3.937x1.968, weight: 9.000) pos([Decimal('7.874'), 0, Decimal('1.968')]) rt(5) vol(61.008)
UNFITTED ITEMS:
***************************************************
***************************************************
:::::::::: large-2-box(23.688x11.750x3.000, max_weight:70.000) vol(835.002)
FITTED ITEMS:
====>  50g [powder 1](3.937x1.968x1.968, weight: 1.000) pos([0, 0, 0]) rt(0) vol(15.248)
====>  50g [powder 2](3.937x1.968x1.968, weight: 2.000) pos([Decimal('3.937'), 0, 0]) rt(0) vol(15.248)
====>  50g [powder 3](3.937x1.968x1.968, weight: 3.000) pos([Decimal('7.874'), 0, 0]) rt(0) vol(15.248)
====>  250g [powder 4](7.874x3.937x1.968, weight: 4.000) pos([Decimal('11.811'), 0, 0]) rt(0) vol(61.008)
====>  250g [powder 5](7.874x3.937x1.968, weight: 5.000) pos([Decimal('19.685'), 0, 0]) rt(1) vol(61.008)
====>  250g [powder 6](7.874x3.937x1.968, weight: 6.000) pos([0, Decimal('1.968'), 0]) rt(0) vol(61.008)
====>  250g [powder 7](7.874x3.937x1.968, weight: 7.000) pos([Decimal('11.811'), Decimal('3.937'), 0]) rt(0) vol(61.008)
====>  250g [powder 8](7.874x3.937x1.968, weight: 8.000) pos([0, Decimal('5.905'), 0]) rt(0) vol(61.008)
UNFITTED ITEMS:
====>  250g [powder 9](7.874x3.937x1.968, weight: 9.000) pos([Decimal('7.874'), 0, Decimal('1.968')]) rt(5) vol(61.008)
***************************************************
```

But this information is in the form of text strings, to use the program with Trimesh we need to be able to get the numerical values and lists

# Details of each **Item object** can be found on Github [https://github.com/enzoruiz/3dbinpacking](https://github.com/enzoruiz/3dbinpacking)



| | | |
|---|---|---|
| Enzo Ruiz minor: v1.1.2 | | |
| py3dbp | Sum the right dimension to pivot when t | |
| .gitignore | fix: wrong answer | |
| LICENSE | added readme and license | |
| README.md | minor: v1.1.2 | |
| erick_dube_507-034.pdf | feat: clean code and more documentatio | |
| example.py | feat: clean code and more documentation | 2 years ago |
| setup.py | fix: [v1.1.2] Sum the right dimension to pivot when the item is rotated | 2 years ago |

bellini666 Sum the right dimension to pivot when the item is rotated

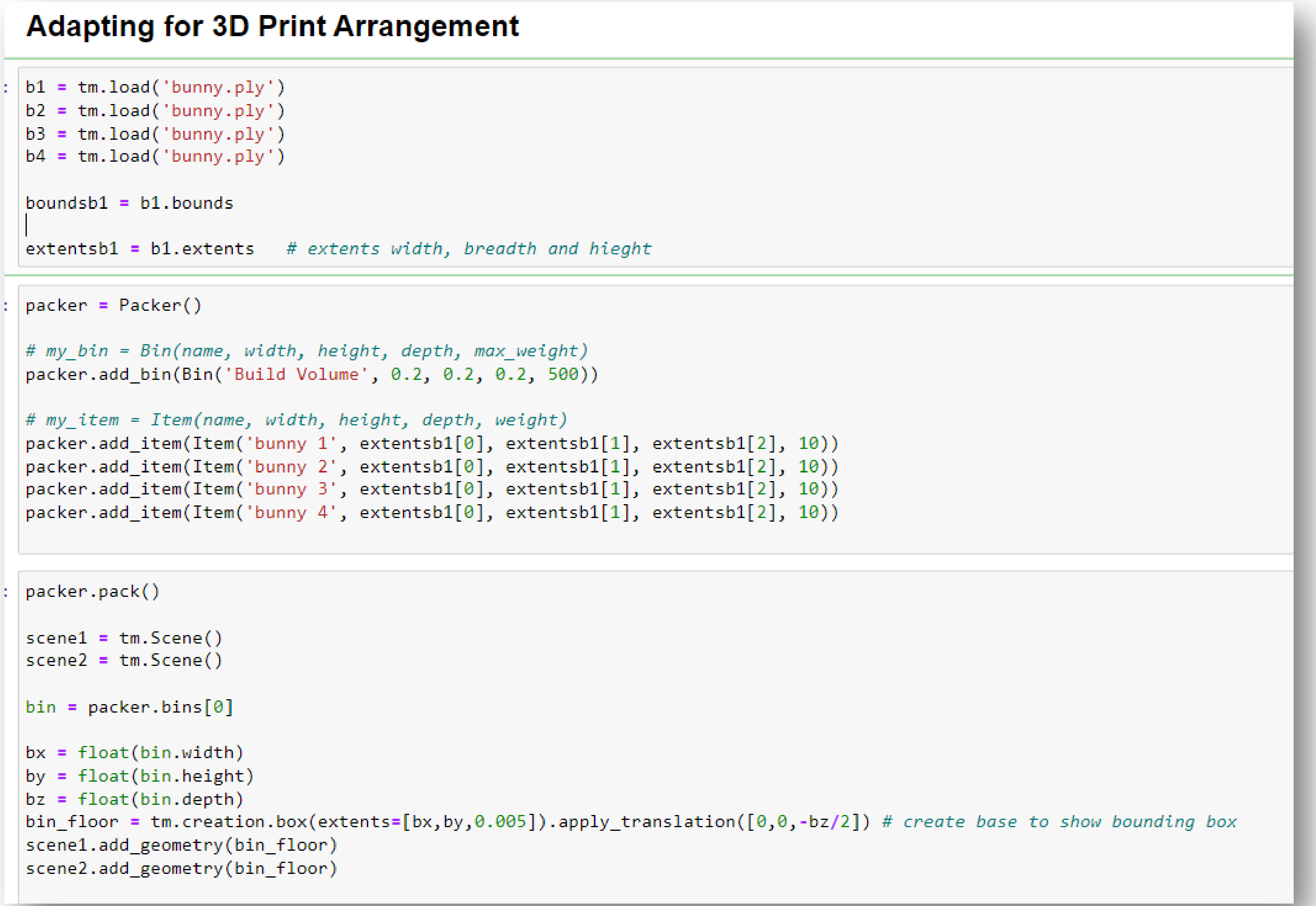
```
..
__init__.py
auxiliary_methods.py
constants.py
main.py
```

```
class RotationType:
    RT_WHD = 0
    RT_HWD = 1
    RT_HDW = 2
    RT_DHW = 3
    RT_DWH = 4
    RT_WDH = 5
```

```
from .constants import RotationType, Axis
from .auxiliary_methods import intersect, set_to_decimal

DEFAULT_NUMBER_OF_DECIMALS = 3
START_POSITION = [0, 0, 0]


class Item:
    def __init__(self, name, width, height, depth, weight):
        self.name = name
        self.width = width
        self.height = height
        self.depth = depth
        self.weight = weight
        self.rotation_type = 0
        self.position = START_POSITION
        self.number_of_decimals = DEFAULT_NUMBER_OF_DECIMALS

    def format_numbers(self, number_of_decimals):
        self.width = set_to_decimal(self.width, number_of_decimals)
        self.height = set_to_decimal(self.height, number_of_decimals)
        self.depth = set_to_decimal(self.depth, number_of_decimals)
        self.weight = set_to_decimal(self.weight, number_of_decimals)
        self.number_of_decimals = number_of_decimals
```

The code reveals the encoding of rotations and the functions associated with the item objects

# Our notebook to use this packing program for planning 3D printing jobs



**Adapting for 3D Print Arrangement**

```
b1 = tm.load('bunny.ply')
b2 = tm.load('bunny.ply')
b3 = tm.load('bunny.ply')
b4 = tm.load('bunny.ply')

boundsb1 = b1.bounds

extentsb1 = b1.extents   # extents width, breadth and hieght
```

```
packer = Packer()

# my_bin = Bin(name, width, height, depth, max_weight)
packer.add_bin(Bin('Build Volume', 0.2, 0.2, 0.2, 500))

# my_item = Item(name, width, height, depth, weight)
packer.add_item(Item('bunny 1', extentsb1[0], extentsb1[1], extentsb1[2], 10))
packer.add_item(Item('bunny 2', extentsb1[0], extentsb1[1], extentsb1[2], 10))
packer.add_item(Item('bunny 3', extentsb1[0], extentsb1[1], extentsb1[2], 10))
packer.add_item(Item('bunny 4', extentsb1[0], extentsb1[1], extentsb1[2], 10))
```

```
packer.pack()

scene1 = tm.Scene()
scene2 = tm.Scene()

bin = packer.bins[0]

bx = float(bin.width)
by = float(bin.height)
bz = float(bin.depth)
bin_floor = tm.creation.box(extents=[bx,by,0.005]).apply_translation([0,0,-bz/2]) # create base to show bounding box
scene1.add_geometry(bin_floor)
scene2.add_geometry(bin_floor)
```

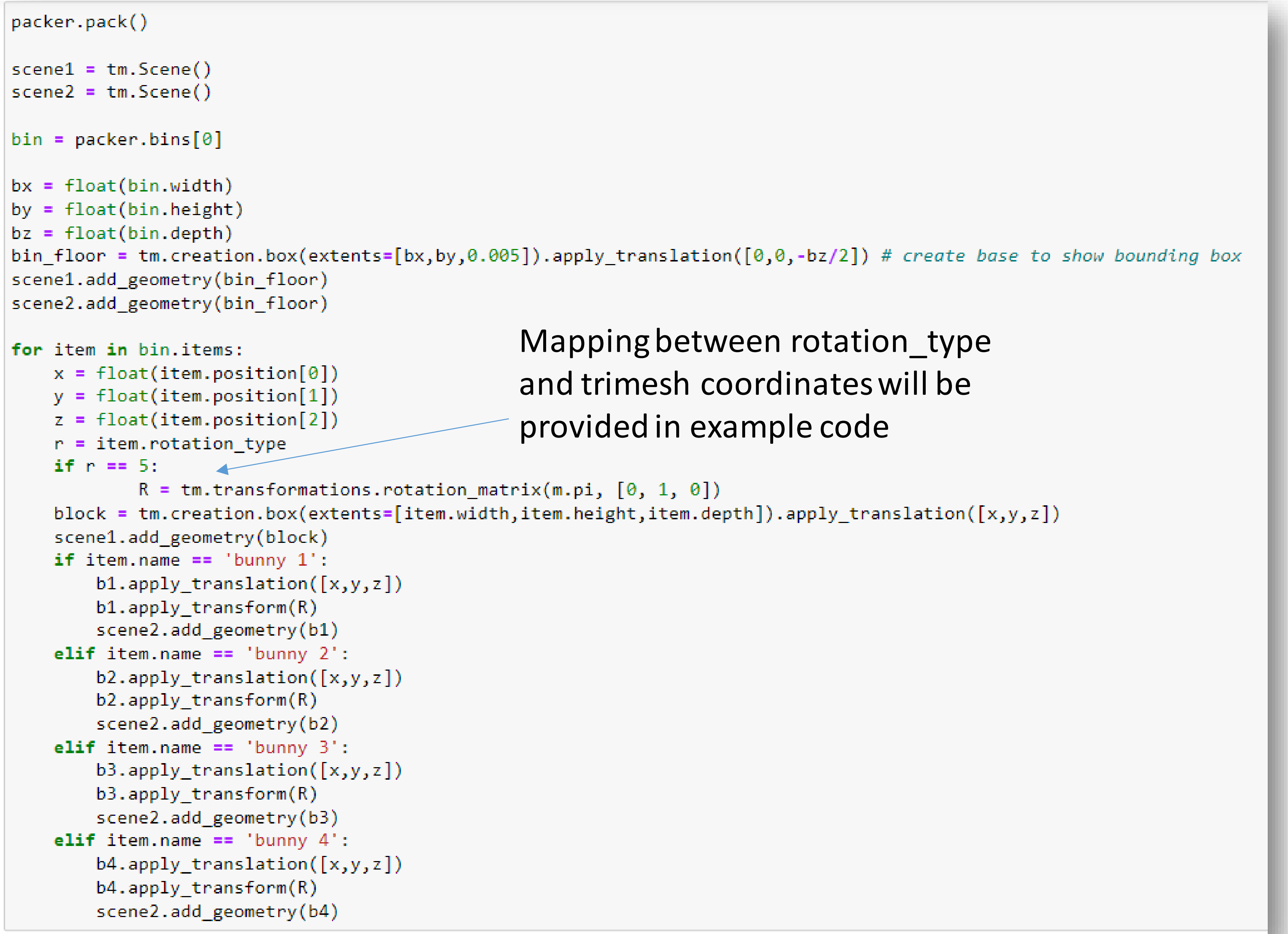
Loading 3D models

Getting their size

Adding their size to the "Packer"

Difficult to display the build volume in wire frame so just add its base to the scene

- After packing use direct interface to get the location and orientation of parts



```
packer.pack()

scene1 = tm.Scene()
scene2 = tm.Scene()

bin = packer.bins[0]

bx = float(bin.width)
by = float(bin.height)
bz = float(bin.depth)
bin_floor = tm.creation.box(extents=[bx,by,0.005]).apply_translation([0,0,-bz/2]) # create base to show bounding box
scene1.add_geometry(bin_floor)
scene2.add_geometry(bin_floor)

for item in bin.items:
    x = float(item.position[0])
    y = float(item.position[1])
    z = float(item.position[2])
    r = item.rotation_type
    if r == 5:
            R = tm.transformations.rotation_matrix(m.pi, [0, 1, 0])
    block = tm.creation.box(extents=[item.width,item.height,item.depth]).apply_translation([x,y,z])
    scene1.add_geometry(block)
    if item.name == 'bunny 1':
        b1.apply_translation([x,y,z])
        b1.apply_transform(R)
        scene2.add_geometry(b1)
    elif item.name == 'bunny 2':
        b2.apply_translation([x,y,z])
        b2.apply_transform(R)
        scene2.add_geometry(b2)
    elif item.name == 'bunny 3':
        b3.apply_translation([x,y,z])
        b3.apply_transform(R)
        scene2.add_geometry(b3)
    elif item.name == 'bunny 4':
        b4.apply_translation([x,y,z])
        b4.apply_transform(R)
        scene2.add_geometry(b4)
```

Mapping between rotation_type and trimesh coordinates will be provided in example code

Lots more functionality for using and displaying 3D models available on Github!