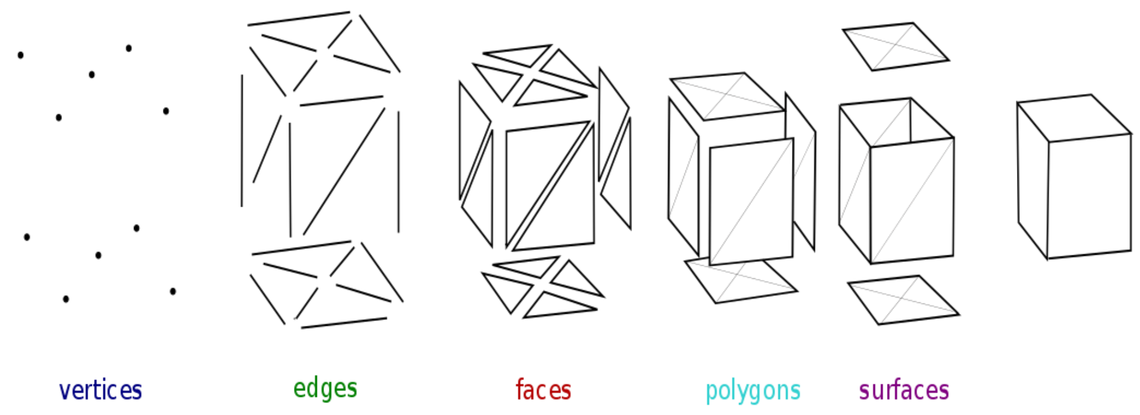
Internship

Polygon Mesh

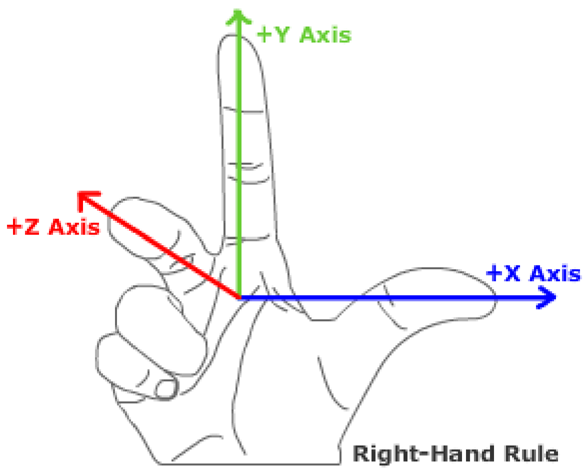
A mesh is a set of vertices, edges and faces, saved in certain ways, according to the necessary specifications, with which it is possible to represent the surface of real objects on a digital domain.

There are both surface meshes, which represent an object only through their surface but whose interior is empty, and volume meshes, which also represent the interior.



Transformations on Mesh

As a rule, the right hand rule as in in the image is taken into account for axis positioning.



So to rotate a mesh with respect to a point in space you can apply the transformation to all the vertices of the mesh, going to modify the coordinates of each of them.
In case the edges are explicitly determined by the vertex pairs, it will not be necessary to go to modify them

Scaling

$$\begin{bmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} SX \cdot x \\ SY \cdot y \\ SZ \cdot z \\ 1 \end{pmatrix}$$

Rotation

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

$$T_{\mathbf{v}}\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = \mathbf{p} + \mathbf{v}$$

Ubuntu Work Space Setup

Download Qt-Creator + Cinolib:

```
sudo apt install build-essential -y
sudo apt install qtcreator -y
sudo apt install qt5-default -y
sudo apt install qt5-doc qt5-doc-html qtbase5-doc-html qtbase5-examples -y
git clone --recursive https://github.com/mlivesu/cinolib.git
```

Cinolib library approach

All meshes inherit from the Abstract Mesh.

Apply Rotation, Translation and Scaling to one of the examples.

Using the '*01_base_app_trimesh*' example
To compile comment all lines referring to vtk in the **.pro** file.
The mesh is initialized in a DrawableTrimesh and pushed onto the canvas.
Before being, to move the mesh to the point (0,0,0) we take its bbox.

Manifold Concept

A space is n-manifold if and only if the outline at each of its points resembles a Euclidean space of the same size.

A 1-manifold space must resemble a size 1 space for each point.

Mesh subdivision and inside cluster non-manifoldness cases identification.

The problem of non manifold topology in a cluster after the subdivision will be useful in a larger pipeline for a conversion of tetrahedral mesh to hexahedral mesh.
Manifoldness is key for a correct conversion and the subsequent reconnection of the hex mesh clusters.

Problem workflow:

- Cluster subdivision
- Non manifold vertex identification
- Edge Split incident to the vertex
- Re apply cluster Cluster subdivision

Beginning from a tetrahedral mesh might be a little much. So let-s start with the Stanford bunny.

First step is the clusterization, and since I the bunny is a pretty triangulated mesh and not many vertexes will result non manifold, I'll subdivide the mesh around the 8 spatial spaces created at the intersection of the axis.

So the mesh needs to be translated in the origin :

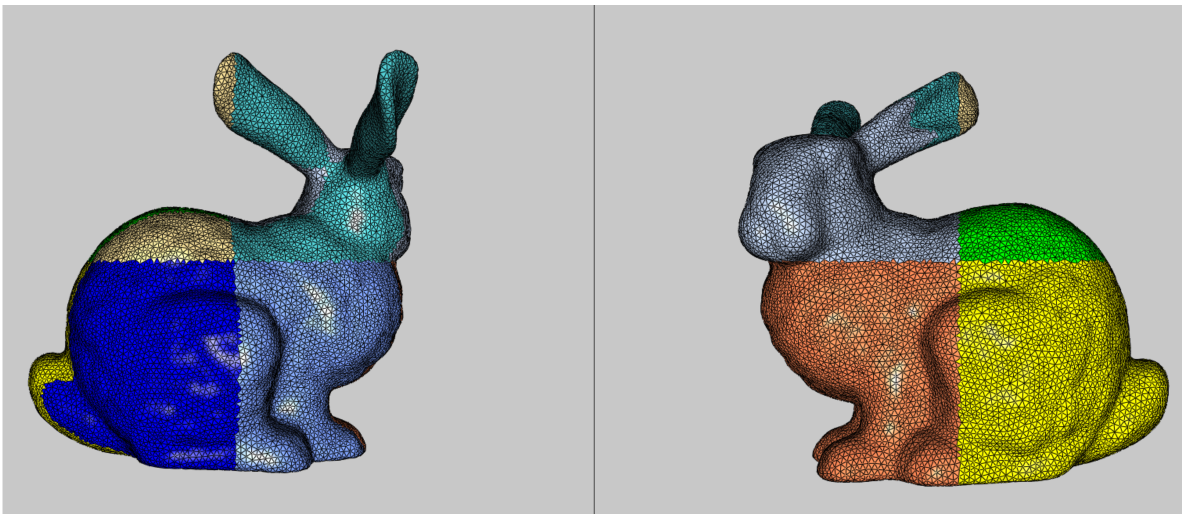
```
    m.translate( m.bbox().center() );
```

Result of cluster subdivision around the origin point

The cluster subdivision is done in a loop over all vertices:

```
    double x,y,z;
    for(uint pid=0; pid < m.num_polys(); ++pid){
        x = m.poly_centroid(pid).x();
        y = m.poly_centroid(pid).y();
        z = m.poly_centroid(pid).z();
        if(x >= 0 && y >= 0 && z >= 0) m.poly_data(pid).color = Color::GREEN();
        if(x >= 0 && y >= 0 && z <= 0) m.poly_data(pid).color =
Color::PASTEL_YELLOW();
        if(x >= 0 && y <= 0 && z >= 0) m.poly_data(pid).color = Color::YELLOW();
        if(x >= 0 && y <= 0 && z <= 0) m.poly_data(pid).color = Color::BLUE();
        if(x <= 0 && y >= 0 && z >= 0) m.poly_data(pid).color =
Color::PASTEL_PINK();
        if(x <= 0 && y >= 0 && z <= 0) m.poly_data(pid).color =
Color::PASTEL_CYAN();
        if(x <= 0 && y <= 0 && z >= 0) m.poly_data(pid).color =
Color::PASTEL_ORANGE();
```

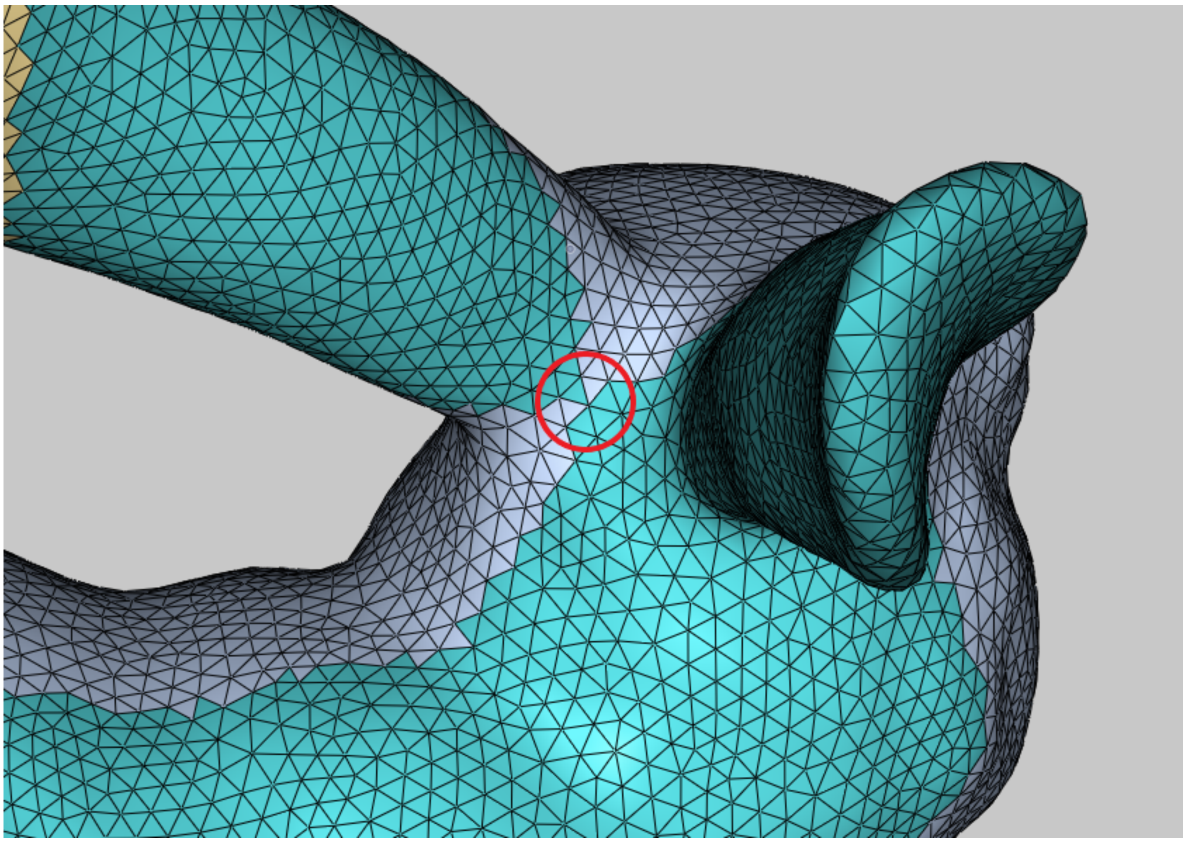
```
        if(x <= 0 && y <= 0 && z <= 0) m.poly_data(pid).color =
Color::PASTEL_VIOLET();
    }
    m.updateGL(); //Always update after transforms on mesh
```



Non Manifold vertexes in same Cluster

The only position where a non manifold vertex can appear on this nicely done mesh is over the saddle surface between the ears of the bunny.
After fiddling with rotations, the job of finding a detectable case is done by the following code:

```
m.rotate(vec3d(0,1,0),0.001);
```



Non manifold vertex identification

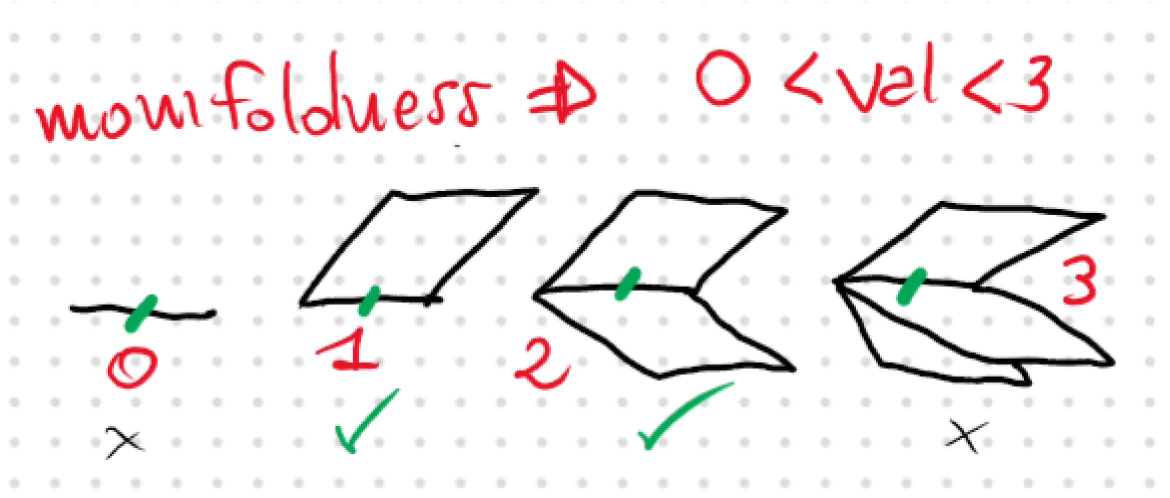
The cinolib library contains a function for detecting non manifold vertexes or edges of a mesh.

```
bool AbstractPolygonMesh<M,V,E,P>::edge_is_manifold(const uint eid)
bool AbstractPolygonMesh<M,V,E,P>::vert_is_manifold(const uint vid)
```

Edge manifoldness

Edge manifoldness analysis is determined by counting how many polygons a given edge is adjacent to.

This number, called *valence*, needs to be equal to either 1 or 2 for the edge to be manifold. Otherwise the edge is a single line or there are multiples faces adjacent to it, in which case the 2-manifoldness satisfied.



In cinolib the the edge-faces adjacencies is a vector and the valence number is therefore its size. Consequently edge manifoldness has O(1) complexity.

Vertex manifoldness

Analysing if vertex is manifold requires to check if every component of its edge_link is connected or not.

The function consist in a Breadth-first search over the edge_link of a vertex.

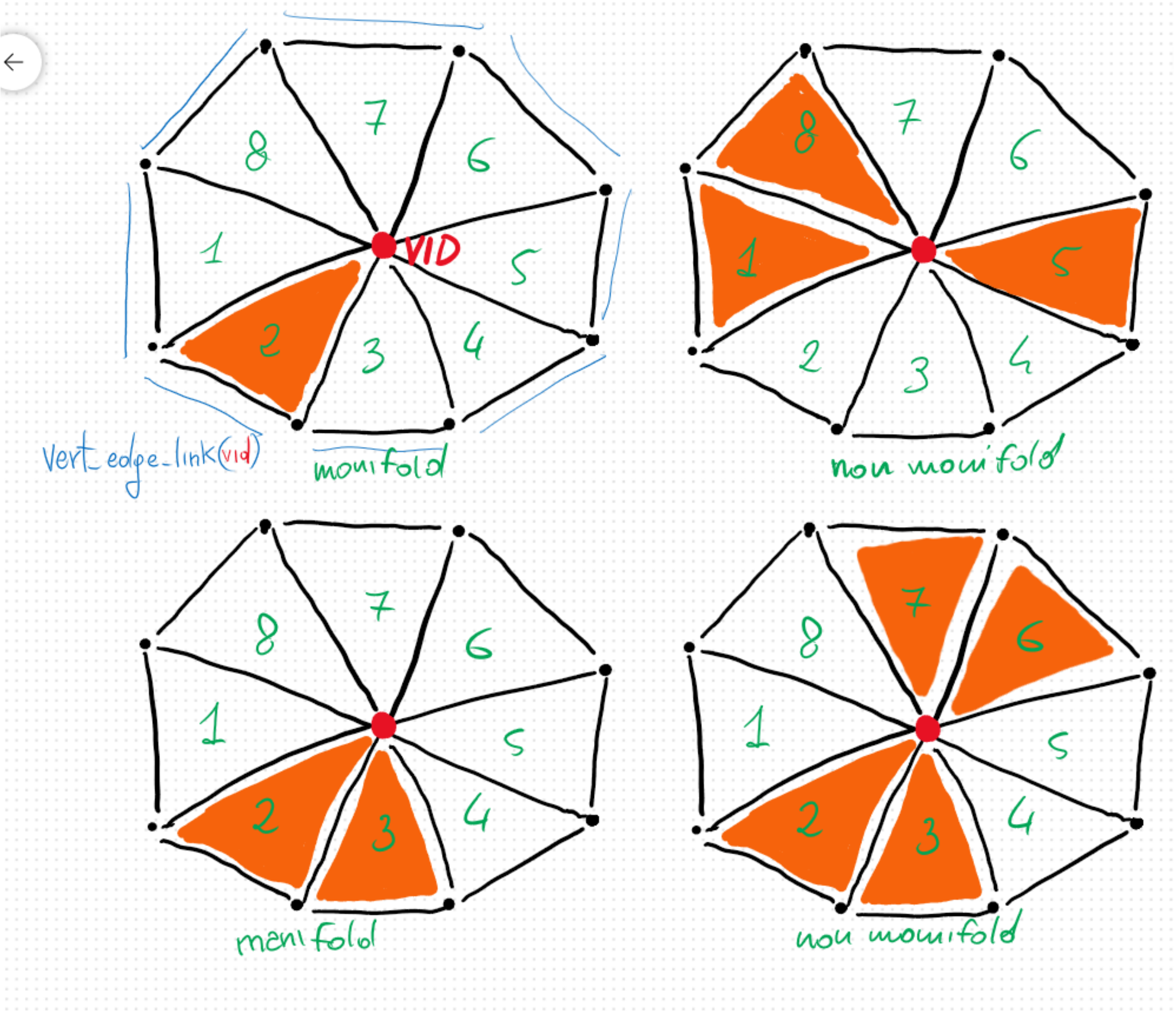
```
std::vector<uint> e_link = this->vert_edges_link(vid);
std::unordered_set<uint> edge_set(e_link.begin(), e_link.end());

    std::queue<uint> q;
    q.push(e_link.front());

    std::unordered_set<uint> visited;
    visited.insert(e_link.front());

    while(!q.empty())
    {
        uint curr = q.front();
        q.pop();

        assert(CONTAINS(visited, curr));

        for(uint nbr : this->adj_e2e(curr))
        {
            // still in the link of vid, but not visited yet
            if(CONTAINS(edge_set, nbr) && !CONTAINS(visited, nbr) )
            {
                visited.insert(nbr);
                q.push(nbr);
            }
        }
    }

    return (visited.size() == e_link.size());
```

If the number of visited edges corresponds to the number of edges in the edge_link, the vertex is manifold.
Vertex manifoldness detection has O(V + E) complexity.

How to detect a non manifold vertex inside a cluster ?

This two library algorithms search over the entire mesh surface.

They will not detect the two non manifold red vertices on the right side of the image, as they are so just in relation to the orange cluster of the mesh.



On the bottom right poly star around the red vertex there's no 2d space path which can allow an ant to walk from the portion 2-3 to the 6-7.