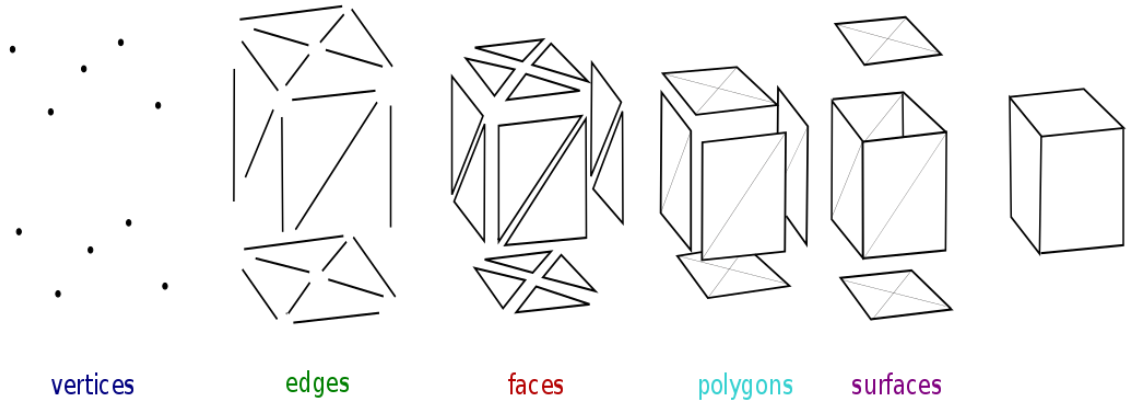
Internship

```
- [ ] Converte starting paragraphs in english
- [ ] rework the images
- [ ] add images for the non manifoldness workflow
```

Concetto di Polygon Mesh

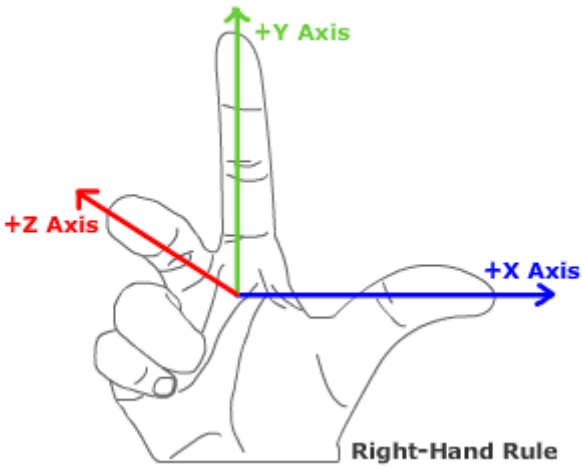
Una mesh è un insieme di vertici, spigoli e facce, salvate in determinati modi, a seconda delle specifiche necessarie, con la quale e possibili rappresentare la superficie di oggetti reali su un dominio digitale.

Esistono sia mesh di superficie, che rappresentano un oggetto solo attraverso la loro superficie ma il cui interno e vuoto, sia le mesh di volume, che rappresentano anche la parte interna.



Trasformazioni Mesh

Come norma, per il posizionamento degli assi, si tiene conto della regola della mano destra come in nell'immagine.



Quindi per ruotare una mesh rispetto a un punto dello spazio e possibile applicare la trasformazione a tutti i vertici della mesh, andando a modificare le coordinate di ognuno di essi. Nel caso gli spigoli siano determinati esplicitamente dalle coppie di vertici non sarà necessario andarli a modificare

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

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

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

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

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

Concetto di Manifold

Uno spazio è n-manifold se e solo se il contorno a ogni suo punto assomiglia a uno spazio euclideo di stessa dimensione.

Uno spazio 1-manifold deve assomigliare per ogni punto a uno spazio di dimensione 1.
Ubuntu WorkSpace Setup Download Qt-Creator + Cinolib:

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

Week 1**Approccio alla libreria Cinolib**
Tutte le mesh ereditano dal Abstract Mesh.
Applicare Rotazione, Traslazione e scalatura ad uno degli esempi**Utilizzando l'esempio *'01_base_app_trimesh'***
Per compilare commentare tutte le righe che fanno riferimento a vtk nel file .pro**.
La mesh viene inizializzata in un DrawableTrimesh e pushato sulla canvas.
Prima di essere, per traslare la mesh nel punto (0,0,0) prendiamo il suo bbox.

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 around vertex
- Re apply cluster Cluster subdivision

///TODO

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.

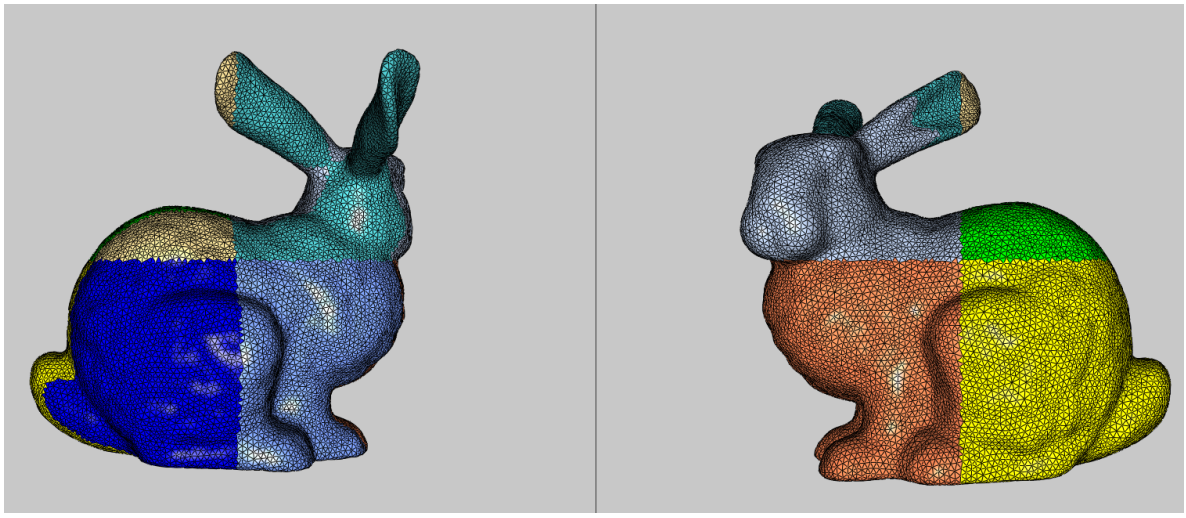
So the mesh needs to be translated in the origin :

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

Result of cluster subdivision around the origin point

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

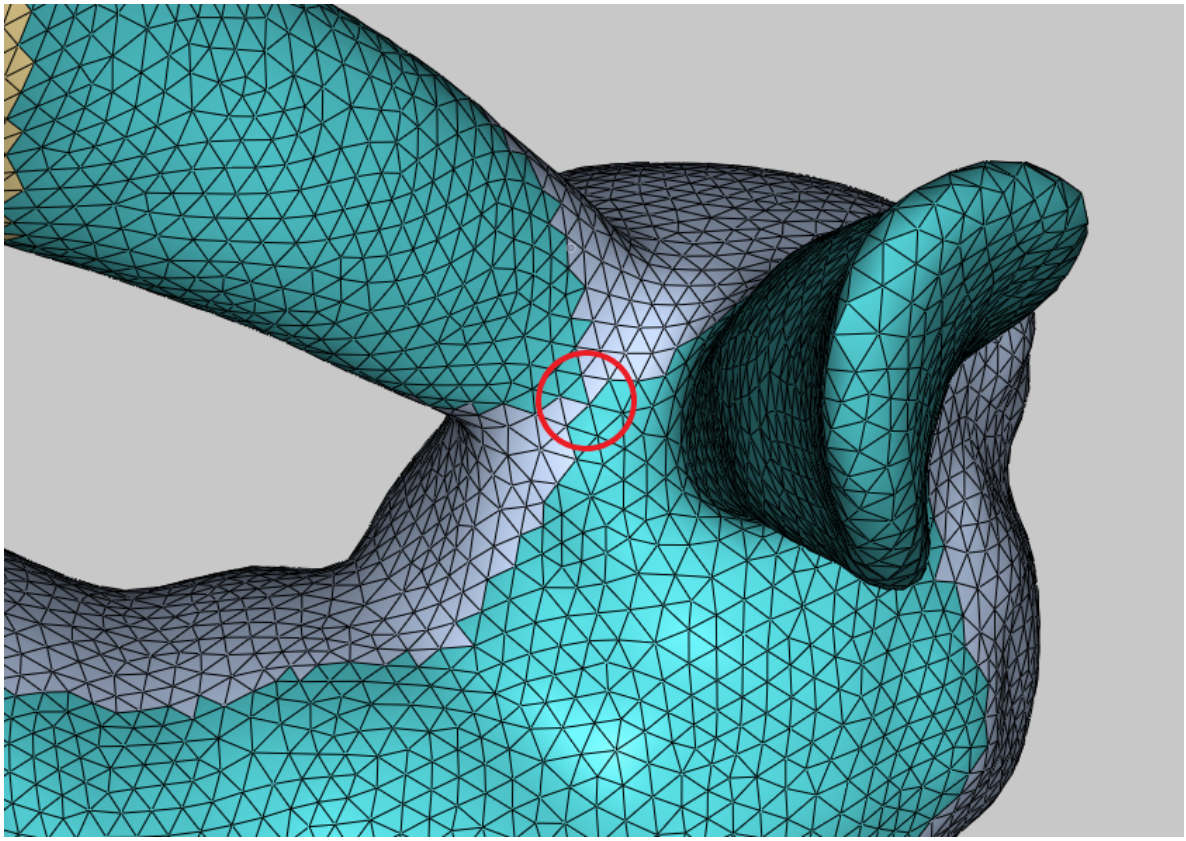
```
    double x,y,z;
    for(uint pid=0; pid < m.num_polys(); ++pid){
        x = m.poly_centroid(pid).x();
        y = m.poly_centroid(pid).y();
        z = m.poly_centroid(pid).z();
        if(x >= 0 && y >= 0 && z >= 0) m.poly_data(pid).color = Color::GREEN();
        if(x >= 0 && y >= 0 && z <= 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>::vert_is_manifold(const uint vid)
bool AbstractPolygonMesh<M,V,E,P>::edge_is_manifold(const uint eid)
```

These, however cannot detect the manifoldness of the vertex with connect the two parts the blue cluster.

It will not, however, detect non manifoldness of a vertex inside one of the clusters created.