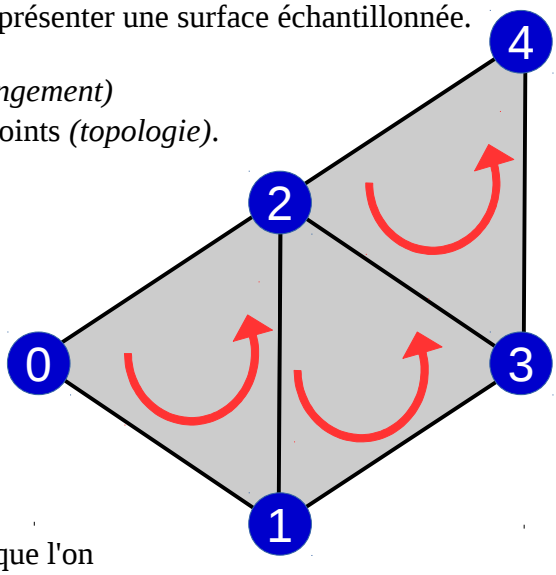
Maillages (triangulaires) et révolution.

Le but de ce TP est de manipuler des maillages (surfaciques triangulaires) avec comme application principale l'algorithme de révolution.

Maillages (triangulaires)

Un maillage est une structure de donnée permettant de représenter une surface échantillonnée.
Il est composé de 2 parties:

- Les sommets définis par leur position (x,y,z) *(plongement)*
- Les faces (triangles) définies par les indices des points *(topologie)*.



Dans l'exemple ci-contre on a:

- 5 points indicés de 0 à 4
- 3 triangles:
  - 0 1 2
  - 2 1 3
  - 4 2 3

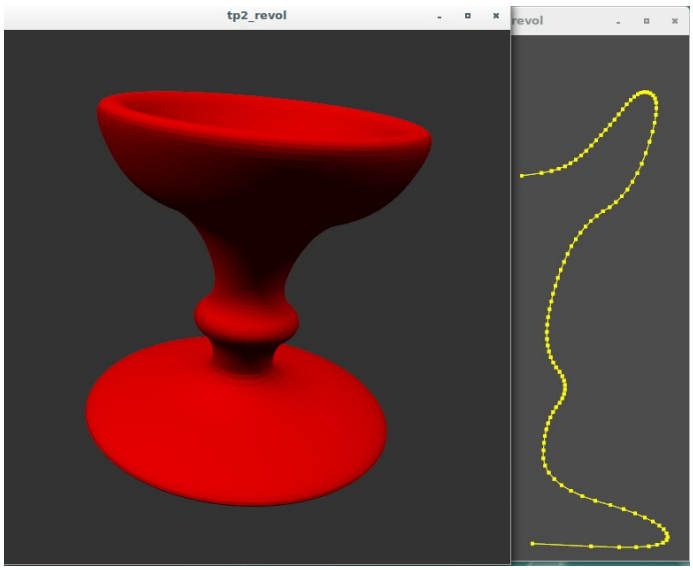
Remarques:

- Tous les triangles ont la même orientation, c.a.d que l'on tourne toujours dans le même sens (trigo/anti-horaire)
- En moyenne un sommet est partagé par ~ 6 triangles.
- On utilise souvent les triangles car :
  - ils sont plans !
  - OpenGL ne sait tracer que des triangles

Révolution

L'algorithme de révolution consiste à faire tourner une courbe autour d'un axe pour engendrer la surface correspondante.

On peut faire l'analogie avec les parallèles qui balayent la surfaces du globe.



Encore un peu de C++

Vector (structure de donnée, tableau auto-redimensionnable)

```
#include <vector>
std::vector<Vec3> vertices;       // déclare un vector de Vec3
vertices.push_back(P);            // ajoute un élément à la fin
vertices.pop_back();              // retire le dernier élément
Vec3 v2 = vertices[0];            // accès: comme un tableau
int nb = vertices.size();         // renvoit la taille
vertices.clear();                 // vide le tableau
```

http://www.cplusplus.com/reference/vector/vector/
http://en.cppreference.com/w/cpp/container/vector

GLM opérations sur les vecteurs (Vec2, Vec3, Vec4):

```
glm::dot(u,v);          // produit scalaire
glm::cross(u,v);        // produit vectoriel
glm::length(u);         // longueur (norme) d'un vecteur
glm::normalize(u)       // calcul le vecteur normalizée
```

TP à réaliser:

Dans la classe **MeshTri**, les maillages sont stockés sous la forme de trois tableau (std::vector)

```
std::vector<Vec3> m_points;
std::vector<Vec3> m_normals;
std::vector<int>  m_indices;
```

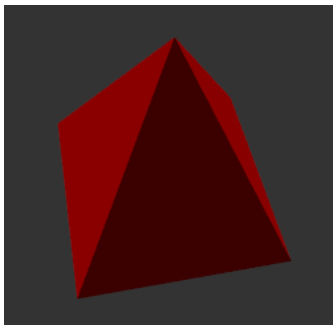
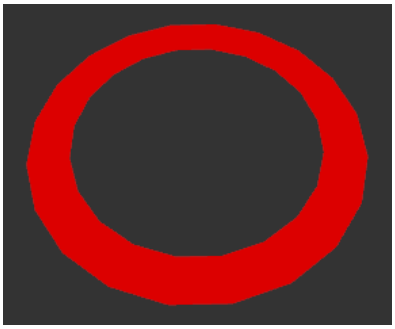
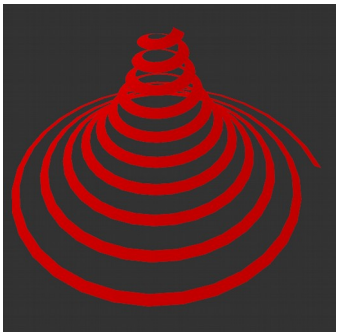
m_points pour les sommets, m_normals pour les vecteurs orthogonaux à la surface en chaque sommet (optionel) et m_indices pour les indices de triangles (à ajouter et lire par paquets de trois)

implanter les méthodes suivantes:

- *add_vertex*
- *add_normal*
- *add_tri*
- *add_quad*

qui vous permetteront de réaliser facilement:

- *create_pyramide* (1 quad + 4 triangles)
- *create_anneau*
- *create_spirale*

La classe **PolygonEditor** permet de saisir un polygone grace à l'interface View2D.

Les points saisis à la souris ont leur coordonnées dans l'intervalle -1,+1.
On générera le polygone dans le plan XY (Z=0)

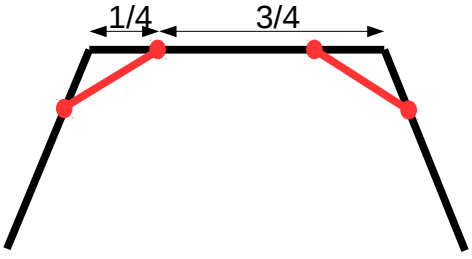
Compléter les méthodes suivantes:

- *add_vertex*
- *remove_last_vertex*
- *clear*
- Interdire les croisements dans add_vertex
  - faire une fonction intermédiaire qui teste l'intersection de 2 segments du plan xy
    - on utilisera le produit vectoriel et on comparera les signes de la composante z
    - remarque la fonction std::signbit(z) renvoit vrai si z est négatif faux sinon

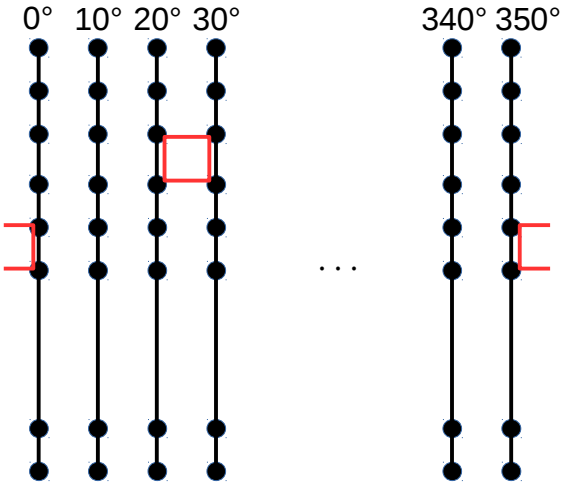
Bonus:

- *lisse*: lisse le polygone par le schéma subdivision de Chaikin, qui consiste à remplacer chaque points intermédiaire par 2 points placé au ¾ des segments adjacents.

  On créera un nouveau vector que l'on échangera avec l'original (méthode swap de vector)



Implanter *revolution* dans **MeshTri**:

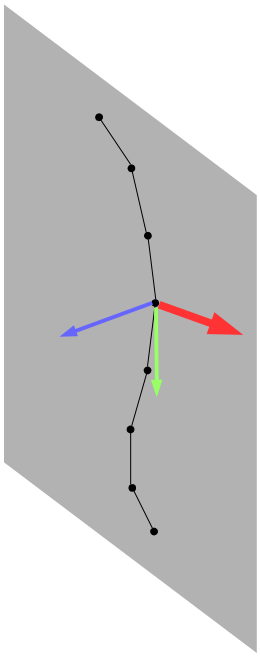


- on ajoutera d'abord les points (colonne par colonne, polygone par polygone) en utilisant une matrice de rotation pour les tourner.

- on ajoutera ensuite les quads qui relient ces points. Attention à la dernière rangée qui finit sur la première rangée de points.

  Même remarque que sur les triangles, tous les quads doivent tourner dans le sens trigo!

Bonus:

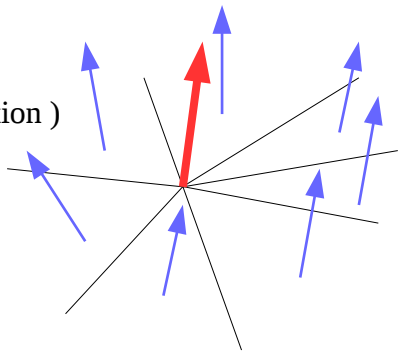


- ajouter les normales aux sommets dans révolution:
  Ici la normale est orthogonale à la courbe polygonale dans le plan du polygone.

  Remarque: si on trouve facilement 2 vecteurs non colinéaires et orthogonaux au vecteur recherché c'est gagné.

  en vert le vecteur tangent à la courbe et en bleu la normale au plan Le vecteur bleu peut être directement obtenue depuis la matrice de rotation.

  Rappel: une matrice de transformation est un repère local: (Nx)(Ny)(Nz)(O)



- implanter *compute_normals* (calcule les normales par approximation)

  La normale a un sommet peut être approximée comme la somme normalisée des vecteurs normaux des faces qui le contiennent.

Algo en 3 passes:
- initialisation de toutes les normales a (0,0,0)
- pour tous les triangles:
    - calcul de la normale au triangles Nf
    - ajout de Nf au 3 normales des 3 sommets
- normalisation de toutes les normales

- fermer les trous du maillage (haut et bas) avec des ombrelles de triangles.
  Remarque: il faut ajouter 2 sommets. Il y a plusieurs façons de les choisir, plus ou moins simple à implanter.