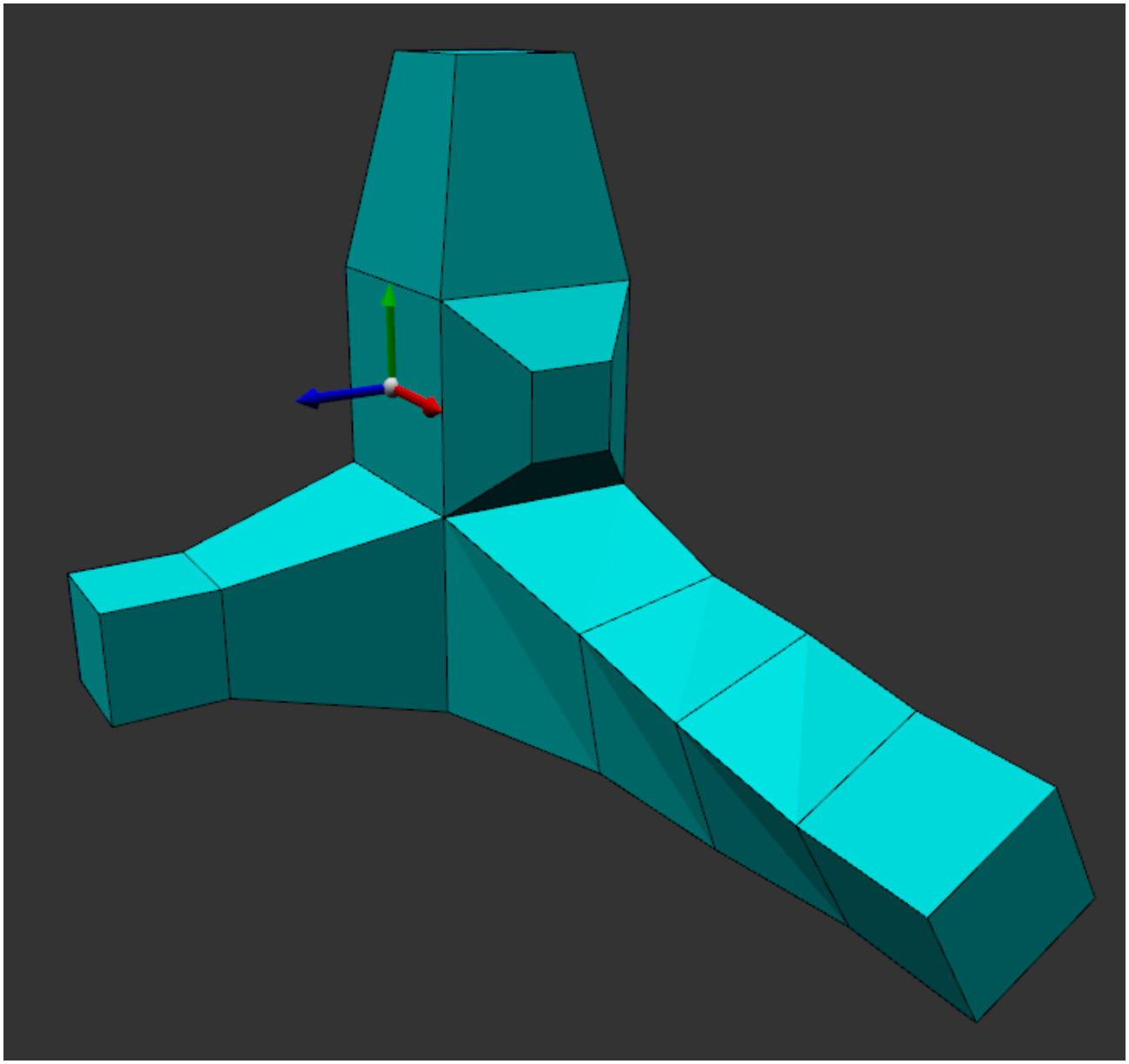
Projet: Maillages Quadrangulaires et extrusion.

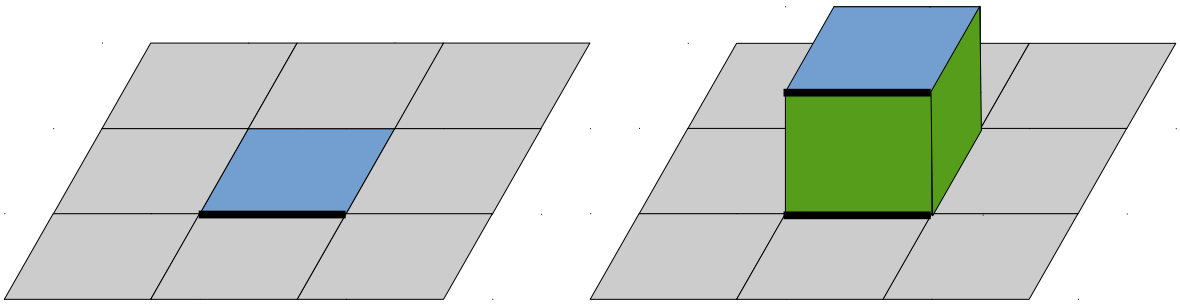
Le but de ce projet est de faire une application interactive de modélisation par extrusion de faces.

On travaillera ici uniquement avec des faces quadrangulaires (4 côtés, “quad”) car elles sont plus naturelles à manipuler. A partir d'un cube on pourra sélectionner une face, l'extruder, changer sa taille, la tourner.
On pourra ensuite continuer avec les nouvelles faces ajoutées, et ainsi de suite.
La face sélectionnée sera repérée par un repère (voir 1er TP), le Z correspondant à la normale à la face.



L'extrusion de face consite a remplacer une face à n cotés par un prisme à n cotés (avec ici n=4)

On peut aussi considérer que l'on sépare et déplace la face (bleu) dans la direction de sa normale et que l'on insère une face quadrangulaire (verte) entre chaque paire d'arêtes séparées (noir)



Pour réaliser tout cela on implantera dans la classe **MeshQuad** les méthodes suivantes:

Remarque: dans MeshQuad on stocke des indices de quads (paquet de 4, pas comme dans MeshTri !)

- **clear**: vide le maillage
- **add_vertex**: ajoute un sommet
- **add_quad**: ajoute un quad
- **convert_quads_to_tris**: convertit le tableau d'indices de quad en tableau d'indices de triangles. On découpera chaque quad en deux triangles. Cette méthode est nécessaire à l'affichage, car OpenGL ne gère que les triangles
- **convert_quads_to_edge**: convertit le tableau d'indices de quad en tableau d'indices d'arêtes. Cette méthode est nécessaire à l'affichage des arêtes. Si possible on ne créera pas les arêtes en double ! (remarque la solution est très simple et très rapide car ici une arête est toujours partagée par deux faces).
- **bounding_sphere**: calcul le centre et le rayon de la scène
- **create_cube**: creation d'un cube
- **normal_of_quad**: calcul de la normale moyenne à un quad
- **area_of_quad**: calcul de l'aire d'un quad
- **is_point_in_quad**: est-ce qu'un points est dans un quad, le points étant déjà dans le même plan que le quad (considéré comme plan)
- **intersect_ray_quad**: calcul de l'intersection entre un rayon et un quad. Le rayon est donné par un point et un vecteur (générés au click souris dans viewer.cpp), utilise is_point_in_quad
- **intersected_visible**: trouve le quad visible (le plus proche de la caméra) du maillage qui est intersecté par un rayon.
- **local_frame**: calcule la matrice de transformation qui définit un repère local au milieu de la face, avec Z aligné sur la normale et X aligné sur la première arête du quad.
- **extrude_quad**: extrude une face quadrangulaire le long de sa normale, on la décalera d'une

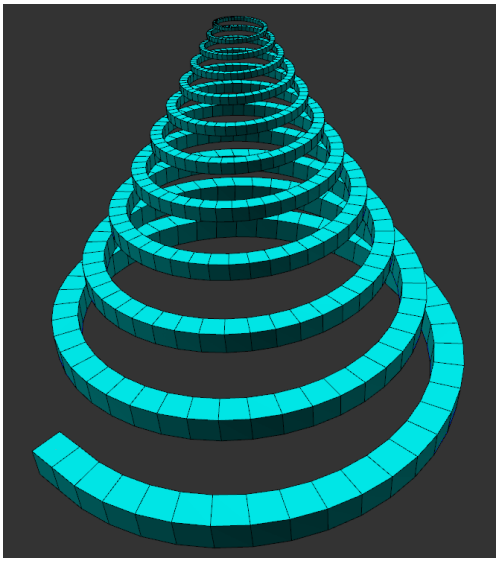
distance proportionnelle à la racine carré de son aire. La face extrudée reste à sa place dans le tableau d'indices (si c'est la $i^{ème}$ face, elle reste la $i^{ème}$ face)

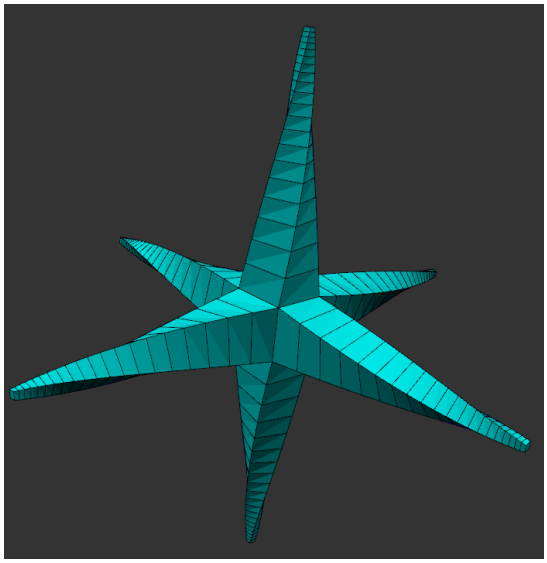
- **decal_quad**: décale la face suivant la normale d'une distance proportionnelle à la racine carré de son aire
- **shrink_quad**: effectue une homothétie sur la face (centrée sur le centre de la face)
- **tourne_quad**: tourne la face autour de sa normale
- pour les trois dernières on pourra utiliser une fonction intermédiaire **transform_quad** qui applique une transformation *locale* a un quad.

Les paramètres sont détaillés dans les commentaires de meshquad.h. Certains algorithmes complexes sont détaillés dans le code des méthodes dans meshquad.cpp

Pour ajouter des fonctionnalités à l'interface vous devrez modifier *keyPressEvent et mousePressEvent* dans la classe Viewer.

Vous pouvez ensuite facilement réaliser des opérations de plus haut niveau réalisant des objets comme ceux présentés-ci dessous.





Ou même avec un peu de récursivité: