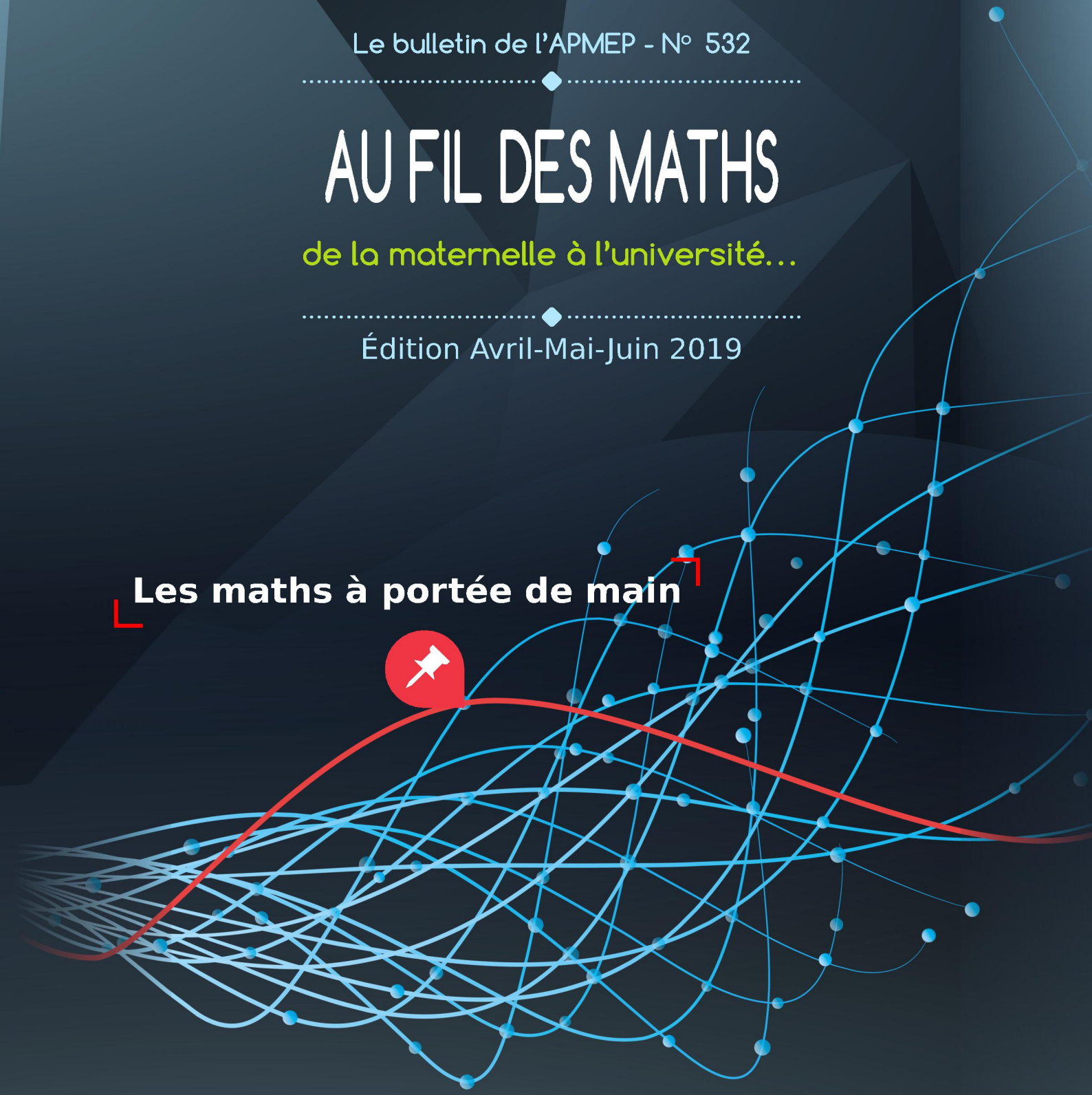
Le bulletin de l'APMEP - N° 532

AU FIL DES MATHS

de la maternelle à l'université...

Édition Avril-Mai-Juin 2019

Les maths à portée de main



APMEP

Association des Professeurs de Mathématiques de l'Enseignement Public

ASSOCIATION
DES PROFESSEURS DE MATHÉMATIQUES
DE L'ENSEIGNEMENT PUBLIC

26 rue Duméril, 75013 Paris
Tél. : 01 43 31 34 05 - Fax : 01 42 17 08 77
Courriel : secretariat-apmep@orange.fr - Site : https://www.apmep.fr
Présidente d'honneur : Christiane Zehren

Au fil des maths, c'est aussi une revue numérique augmentée :
https://afdm.apmep.fr
version réservée aux adhérents. Pour y accéder connectez-vous à votre compte *via* l'onglet *Au fil des maths* (page d'accueil du site) ou *via* le QRcode, ou suivez les logos.

Si vous désirez rejoindre l'équipe d'*Au fil des maths* ou bien proposer un article, écrivez à aufildesmaths@apmep.fr

Annonceurs : pour toute demande de publicité, contactez Mireille Génin mcgenin@wanadoo.fr

À ce numéro est joint le BGV
nº 206 spécial « Journées Nationales »

ÉQUIPE DE RÉDACTION

Directrice de publication : *Alice Ernoult.*

Responsable coordinatrice de l'équipe : *Lise Malrieu.*

Rédacteurs : *Vincent Beck, Marie-Astrid Bézard, François Boucher, Richard Cabassut, Séverine Chassagne-Lambert, Frédéric De Ligt, Mireille Génin, Cécile Kerboul, Valérie Larose, Lise Malrieu, Jean-Marie Martin, Vincent Pantaloni, Daniel Vagost, Christine Zelty.*

« Fils rouges » numériques : *Gwenaëlle Clément, Nada Dragovic, Laure Étévez, Marianne Fabre, Robert Ferréol, Adrien Guinemer, Christophe Romero, Jacques Vallois.*

Illustrateurs : *Pol Le Gall, Olivier Longuet, Jean-Sébastien Masset.*

Équipe TEXnique : *François Couturier, Isabelle Flavier, Anne Heam, François Pétiard, Olivier Reboux, Guillaume Seguin, Sébastien Soucaze, Michel Suquet.*

Relations avec le Bureau national : *Jean Toromanoff.*

Votre adhésion à l'APMEP vous abonne automatiquement à *Au fil des maths*.
Pour les établissements, le prix de l'abonnement est de 60 € par an.
La revue peut être achetée au numéro au prix de 15 € sur la boutique en ligne de l'APMEP.

Mise en page : Olivier Reboux
Dépôt légal : Juin 2019
Impression : Imprimerie Corlet.
ZI, rue Maximilien Vox BP 86, 14110 Condé-sur-Noireau ISSN : 2608-9297

AU FIL DES MATHS

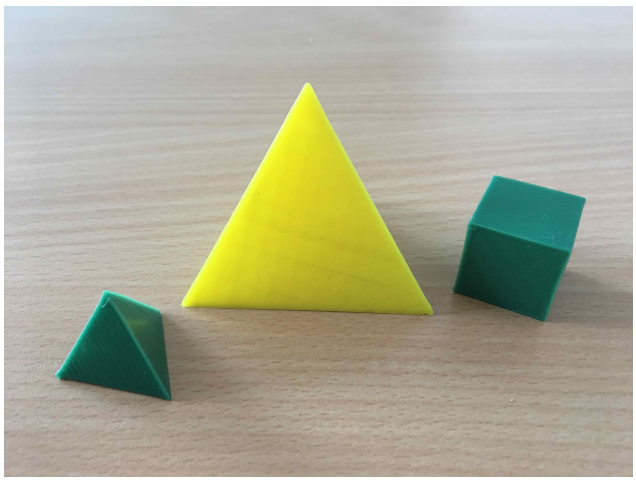
Triangulation et impression 3D

Imprimer en 3D cache en réalité pas mal de mathématiques : essentiellement de la géométrie dans l'espace ; mais en arrière-plan, on voit poindre quelques résultats de la topologie des surfaces. L'utilisateur ne voit pas les mathématiques car ce sont les logiciels qui les prennent en charge. Dans cet article qui complète celui de la revue papier, Aurélien Alvarez se propose de nous les dévoiler en nous présentant tous les calculs nécessaires pour imprimer un tétraèdre.

Dans l'article *Triangulation et impression 3D* [1], nous expliquons le principe de l'impression 3D et le lien avec la triangulation des surfaces.

Nous proposons dans cet article d'écrire un programme permettant de générer un fichier STL contenant les données d'impression d'un tétraèdre régulier.

Ces deux articles complémentaires peuvent être lus indépendamment l'un de l'autre, ce dernier ayant pour but de permettre aux plus enthousiastes de lancer concrètement des impressions 3D des polyèdres réguliers.



Nous commençons par rappeler le principe général de l'impression 3D et ce qu'est le format STL.

Très souvent, on utilise un logiciel de modélisation à partir duquel on exporte un fichier STL du modèle que l'on souhaite imprimer.

Pour autant, peut-on écrire le fichier STL du modèle que l'on souhaite imprimer sans passer par un logiciel de modélisation ?

La complexité d'un modèle peut rapidement rendre la tâche particulièrement fastidieuse mais, sur des exemples simples comme les polyèdres, en particulier le tétraèdre qui ne comporte que quatre faces triangulaires, il s'agit d'un exercice très instructif, nous semble-t-il.

Nous avons choisi d'écrire nos programmes en Swift mais aucune réelle spécificité de ce langage n'est utilisée, de sorte que chacun est libre de réécrire les lignes de code que nous allons présenter dans son langage de programmation préféré.

Impression 3D et format STL

Le principe général de l'impression 3D

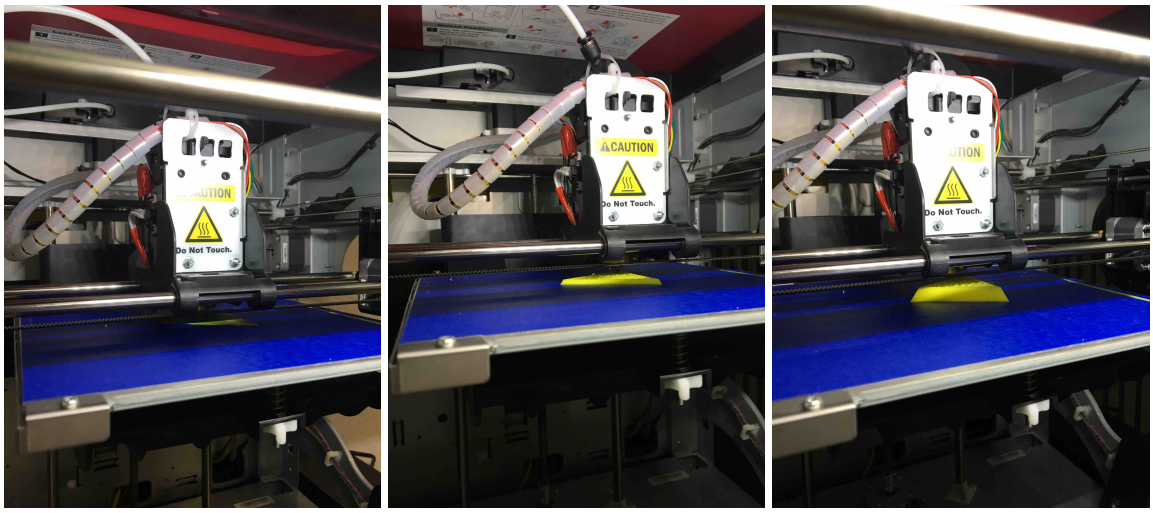
Le principe général du fonctionnement d'une imprimante 3D est de fabriquer des volumes solides par ajout progressif de matière.

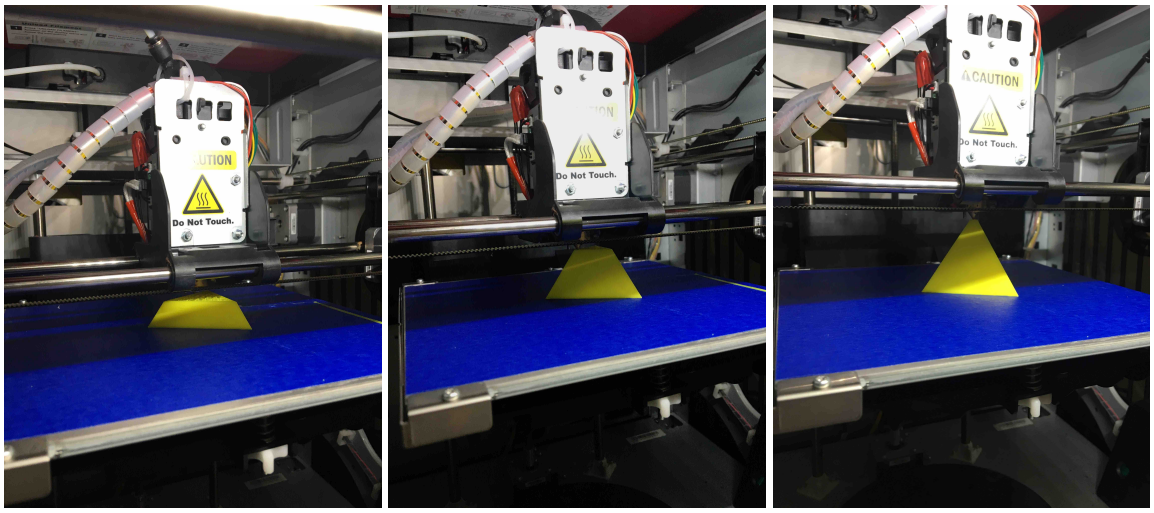
Avant de pouvoir lancer l'impression d'un objet, il faut d'abord disposer d'un modèle numérique 3D de celui-ci.

Le fichier du modèle 3D est ensuite analysé par le logiciel d'impression de l'imprimante qui est en charge d'organiser le découpage en tranches des différentes couches de matière qui seront déposées les unes sur les autres afin de fabriquer l'objet.

Le logiciel d'impression de l'imprimante convertit donc le modèle 3D en instructions précises que la machine est capable de comprendre et d'exécuter.

L'imprimante dépose finalement la matière couche après couche, lentement mais sûrement, jusqu'à obtenir le volume souhaité.





À leurs débuts, les imprimantes 3D utilisaient bien souvent des résines, alors que depuis les années 2010, d'autres matériaux sont plébiscités, en particulier certains plastiques comme le PLA ou l'ABS.

C'est d'ailleurs le cas de l'imprimante da Vinci 1.0 Pro 3in1 que nous avons utilisée ici.

- Le PLA est biodégradable, issu de matériaux recyclés, et il est assez courant de l'utiliser dans l'industrie alimentaire (comme emballage !). Il ramollit autour de $50\,°C$, commence à fondre à $160\,°C$ et peut être réellement travaillé à partir de $180\,°C$. En plus d'être sensible à l'humidité, le PLA a tendance à casser facilement si on le contraint trop fortement.
- L'ABS quant à lui est un polymère thermoplastique que l'on retrouve dans les appareils électroménagers ou dans les jouets Lego. Il ramollit autour de $90\,°C$, commence à fondre à $180\,°C$ et peut être réellement travaillé vers $230\,°C$. Il est donc plus résistant à la chaleur que le PLA et se plie facilement sans rompre.

Le PLA et l'ABS ont chacun leurs avantages et leurs inconvénients mais il est en général plus facile de réaliser des impressions avec le PLA qu'avec l'ABS car ce dernier résiste mal aux chocs de température.

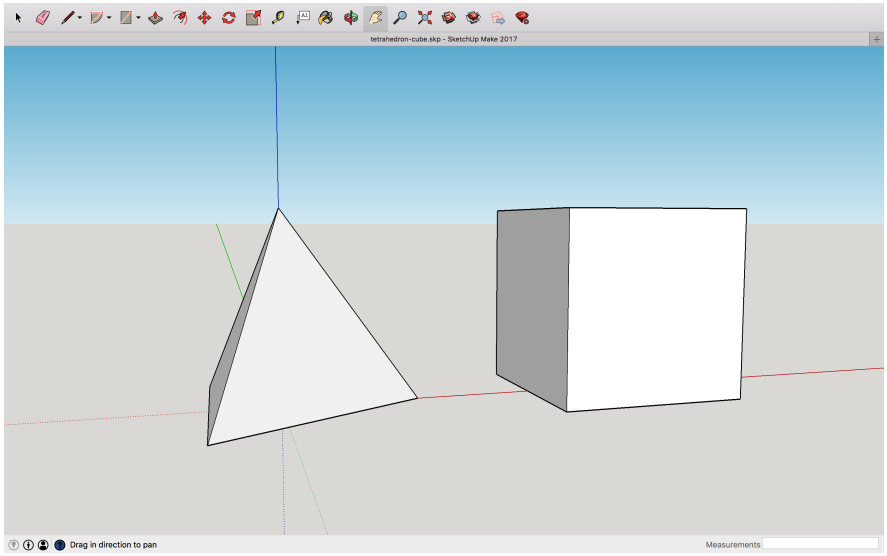
Bien choisir entre le PLA ou l'ABS selon l'objet que l'on souhaite imprimer est davantage une question d'expérience que de science...

Le format STL

De nombreux logiciels de modélisation sont disponibles, y compris des logiciels gratuits et multiplate-formes.

Ces logiciels permettent de modéliser des formes géométriques complexes à l'aide d'outils de base très simple, comme par exemple l'outil d'extrusion.

Une fois le logiciel pris en main et avec un peu d'habitude, on peut modéliser un tétraèdre et un cube en quelques clics, comme illustré sur l'image suivante extraite du logiciel SketchUp.



Pour imprimer le tétraèdre ci-dessus, il nous suffit alors de demander à notre logiciel de modélisation d'exporter le modèle 3D vers un fichier au format STL qui sera traité par le logiciel d'impression de l'imprimante.

Le temps d'impression dépend essentiellement de la taille de l'objet à imprimer et du taux de remplissage de celui-ci : cela peut donc aller de quelques minutes à de longues heures.

Pendant ce temps, avec un simple éditeur texte, rien n'interdit de jeter un œil au fichier STL en question.

On découvre alors quelque chose comme ceci.

```
solid tetrahedron
facet normal 0.0 0.942809 0.333333
  outer loop
    vertex 70.7107 72.4745 -70.7107
    vertex -70.7107 72.4745 -70.7107
    vertex 0.0 31.6497 44.7594
  endloop
endfacet
facet normal 0.816497 -0.471404 0.333333
  outer loop
    vertex 70.7107 72.4745 -70.7107
    vertex 0.0 31.6497 44.7594
    vertex 0.0 -50.0 -70.7107
  endloop
endfacet
facet normal -0.0 0.0 -1.0
  outer loop
    vertex 70.7107 72.4745 -70.7107
    vertex 0.0 -50.0 -70.7107
    vertex -70.7107 72.4745 -70.7107
  endloop
```

```
endfacet
facet normal -0.816497 -0.471404 0.333333
  outer loop
    vertex -70.7107 72.4745 -70.7107
    vertex 0.0 -50.0 -70.7107
    vertex 0.0 31.6497 44.7594
  endloop
endfacet
endsolid tetrahedron
```

Le format STL est très répandu de nos jours, son nom provient du mot *stéréolithographie* (STL pour STereo-Lithography).

D'après Wikipédia :

> La stéréolithographie est une technique, dite de prototypage rapide, qui permet de fabriquer des objets solides à partir d'un modèle numérique.
> L'objet est obtenu par superposition de tranches fines de matière.
> Le développement industriel de cette technique date des années 1980 et fut initié aux États-Unis par Charles W. Hull.

Notons que le format de fichier STL ne décrit que la géométrie de surface d'un solide alors qu'un modèle numérique de conception assistée par ordinateur contient bien souvent des informations complémentaires comme la couleur, la texture, etc.

Regardons de plus près ce fichier.

La première ligne contient le mot réservé `solid` suivi du nom que l'on a choisi pour l'objet au moment de l'exportation (ici `tetrahedron`) et la dernière ligne contient ce même nom après le mot réservé `endsolid`.

Le reste du fichier est composé de blocs commençant par le mot réservé `facet` et terminant par le mot réservé `endfacet`.

```
facet normal ni nj nk
  outer loop
    vertex v1x v1y v1z
    vertex v2x v2y v2z
    vertex v3x v3y v3z
  endloop
endfacet
```

À l'intérieur d'un bloc, les coordonnées de trois sommets sont indiquées, ainsi que les coordonnées de la normale au triangle formé par ces trois sommets et pointant vers l'extérieur du volume.

Les coordonnées de la normale sont précédées par le mot réservé `normal` et celles de chacun des sommets par le mot réservé `vertex`.

Dans notre exemple, on lit quatre blocs `facet`-`endfacet` correspondant chacun à l'une des quatre faces du tétraèdre.

On vérifie également que les quatre normales indiquées dans le fichier sont bien des vecteurs de norme 1.

En résumé, un fichier STL contient essentiellement les données de triangulation de la surface de l'objet que l'on souhaite imprimer.

En effet, tous les objets que l'on imprime, même s'ils peuvent sembler lisses quand on les prend en main, sont en fait décrits par une multitude (parfois des dizaines de milliers) de petits triangles plats.

Écrire un programme qui génère un fichier STL

L'espace affine euclidien $\mathbb{R}^3$

Nous allons utiliser le type `Float` pour représenter les nombres réels ; un vecteur de $\mathbb{R}^3$ est alors la donnée d'un triplet de `Float`.

La structure standard d'espace vectoriel est encodée par l'opération d'addition $+$ sur les vecteurs et par la multiplication externe $*$.

```
struct Vect {
  let x, y, z: Float
  init(x: Float, y: Float, z: Float) {
    self.x = x
    self.y = y
    self.z = z
  }
  static func + (v1: Vect, v2: Vect) -> Vect {
    return Vect(x: v1.x + v2.x, y: v1.y + v2.y, z: v1.z + v2.z)
  }
  static func * (s: Float, v: Vect) -> Vect {
    return Vect(x: s * v.x, y: s * v.y, z: s * v.z)
  }
}
```

Nous munissons à présent l'espace vectoriel $\mathbb{R}^3$ de sa structure euclidienne usuelle (la norme d'un vecteur est la racine carrée de la somme des carrés de ses coordonnées) et définissons une fonction qui normalise un vecteur.

```
extension Vect {
  var norm: Float {
    get {
      return sqrt(self.x * self.x +
                  self.y * self.y + self.z * self.z)
    }
  }
  func normalize() -> Vect {
    return (1.0 / self.norm) * self
  }
}
```

Enfin nous introduisons la structure affine de $\mathbb{R}^3$: un point de $\mathbb{R}^3$ est repéré par ses trois coordonnées et il est possible d'opérer sur tout point une translation par un vecteur quelconque ou une homothétie centrée en l'origine O.

```
struct Vertex {
  var x, y, z: Float
  init(x: Float, y: Float, z: Float) {
    self.x = x
    self.y = y
    self.z = z
  }
  func translate(v: Vect) -> Vertex {
    return Vertex(x: x + v.x, y: y + v.y, z: z + v.z)
  }
  func homothety(s: Float) -> Vertex {
    return Vertex(x: s * x, y: s * y, z: s * z)
  }
}
```

Une autre manière de préciser la structure affine est d'indiquer la fonction `affineStruct` : `Vertex` $\times$ `Vertex` $\longrightarrow$ `Vect` qui, étant donné deux points A et B de $\mathbb{R}^3$, calcule l'unique vecteur $\overrightarrow{v_{AB}}$ tel que $B = A + \overrightarrow{v_{AB}}$, ce qui est une façon équivalente de dire que l'espace vectoriel $\mathbb{R}^3$ opère librement et transitivement dans l'ensemble $\mathbb{R}^3$.

```
func affineStruct(A: Vertex, B: Vertex) -> Vect {
  return Vect(x: B.x - A.x, y: B.y - A.y, z: B.z - A.z)
}
```

Des sommets, des arêtes, des faces et des polyèdres

Après avoir défini la notion de point, ou plutôt de *sommet* (`Vertex`), nous définissons à présent la notion d'*arête* qui est spécifiée par un sommet origine et un sommet terminal ; la méthode `opp()` permet de calculer l'opposée d'une arête et la méthode `toVect()` transforme une arête en un vecteur.

```
struct Edge {
  let origin, terminal: Vertex
  init(origin: Vertex, terminal: Vertex) {
    self.origin = origin
    self.terminal = terminal
  }
  func opp() -> Edge {
    return Edge(origin: terminal, terminal: origin)
  }
  func toVect() -> Vect {
    return affineStruct(A: origin, B: terminal)
  }
}
```

La fonction suivante calcule le produit vectoriel de deux vecteurs de $\mathbb{R}^3$.

```
func wedge(v1: Vect, v2: Vect) -> Vect {
  let nx = v1.y * v2.z - v1.z * v2.y
  let ny = v1.z * v2.x - v1.x * v2.z
  let nz = v1.x * v2.y - v1.y * v2.x
  return Vect(x: nx, y: ny, z: nz)
}
```

Nous définissons maintenant la notion de *face* par la donnée de trois sommets ordonnés ; la méthode `normal()` calcule le vecteur normal unitaire à la face et la méthode `opp()` calcule la face opposée[1].

Sans difficulté, on vérifie d'ailleurs que le vecteur normal unitaire de la face opposée est l'opposé du vecteur normal unitaire de la face de départ.

Autrement dit, pour tout `f` de type `Face`, on a

$$\texttt{f.opp().normal()} = -\texttt{f.normal()}.$$

```
struct Face {
  let vertex_1, vertex_2, vertex_3: Vertex
  init(vertex_1: Vertex, vertex_2: Vertex, vertex_3: Vertex) {
    self.vertex_1 = vertex_1
    self.vertex_2 = vertex_2
    self.vertex_3 = vertex_3
  }
  func normal() -> Vect {
    let v1 = affineStruct(A: self.vertex_1, B: self.vertex_2)
    let v2 = affineStruct(A: self.vertex_1, B: self.vertex_3)
    let normal = wedge(v1: v1, v2: v2)
    return normal.normalize()
    }
  func opp() -> Face {
    return Face(vertex_1: vertex_1, vertex_2: vertex_3,
       vertex_3: vertex_2)
  }
}
```

Nous terminons cette série de définitions avec la notion de polyèdre : ce sera pour nous n'importe quelle structure qui contient une liste de sommets, une liste d'arêtes et une liste de faces.

Du point de vue mathématique, il conviendrait d'être plus précis ici en spécifiant que les sommets de la liste d'arêtes appartiennent à la liste des sommets, et de même pour les arêtes des faces.

Mais ce ne sera pas nécessaire pour notre besoin car nous appliquerons nos définitions dans des cas très précis comme celui du tétraèdre régulier dont nous connaissons bien la combinatoire.

1. Puisque l'on convient d'appeler *face* la donnée de trois sommets ordonnés, la face *opposée* est la donnée de ces trois mêmes sommets ordonnés dans l'autre sens. Concrètement, si ABC est une face, CBA est la face opposée.

```
protocol Polyhedron {
  var vertices: [Vertex] { get }
  var edges: [Edge] { get }
  var faces: [Face] { get }
}
```

De la triangulation au STL

Pour mémoire, rappelons qu'il s'agit de générer un fichier dont le motif général est le suivant.

```
facet normal ni nj nk
  outer loop
    vertex v1x v1y v1z
    vertex v2x v2y v2z
    vertex v3x v3y v3z
  endloop
endfacet
```

La fonction suivante `STLwrite: [Face]` $\times$ `Float` $\longrightarrow$ `Void` prend en entrée une liste de faces, un facteur d'échelle et, pour chaque face de la liste, imprime à l'écran le motif général rappelé ci-dessus avec les coordonnées des normales unitaires aux faces, ainsi que les coordonnées des sommets des faces.

```
func STLwrite(faces: [Face], scale: Float) {
  for face in faces {
    let normal = face.normal()
    print("facet normal \(normal.x) \(normal.y) \(normal.z)")
    print(" outer loop")
    let face_vertices: [Vertex] =
       [face.vertex_1.homothety(s: scale),
          face.vertex_2.homothety(s: scale),
             face.vertex_3.homothety(s: scale)]
    for vertex in face_vertices {
       print(" vertex \(vertex.x) \(vertex.y) \(vertex.z)")
    }
    print(" endloop")
    print("endfacet")
  }
}
```

Le tétraèdre régulier en STL

Voici à présent la définition de la structure polyédrale de tétraèdre : quatre sommets `P`, `Q`, `R`, `S`, six arêtes orientées `PQ`, `PR`, `PS`, `QR`, `RS`, `SQ`, et quatre faces orientées `PQR`, `PRS`, `PSQ`, `QSR`.

```
struct Tetrahedron: Polyhedron {
  let P, Q, R, S: Vertex
  init(P: Vertex, Q: Vertex, R: Vertex, S: Vertex) {
```

```
    self.P = P
    self.Q = Q
    self.R = R
    self.S = S
  }
  var PQ: Edge { get { return Edge(origin: P, terminal: Q) } }
  var PR: Edge { get { return Edge(origin: P, terminal: R) } }
  var PS: Edge { get { return Edge(origin: P, terminal: S) } }
  var QR: Edge { get { return Edge(origin: Q, terminal: R) } }
  var RS: Edge { get { return Edge(origin: R, terminal: S) } }
  var SQ: Edge { get { return Edge(origin: S, terminal: Q) } }
  var PQR: Face { get { return Face(vertex_1: P, vertex_2: Q,
                                    vertex_3: R) } }
  var PRS: Face { get { return Face(vertex_1: P, vertex_2: R,
                                    vertex_3: S) } }
  var PSQ: Face { get { return Face(vertex_1: P, vertex_2: S,
                                    vertex_3: Q) } }
  var QSR: Face { get { return Face(vertex_1: Q, vertex_2: S,
                                    vertex_3: R) } }
  var vertices: [Vertex] { get { return [P, Q, R, S] } }
  var edges: [Edge] { get { return [PQ, PR, PS, QR, RS, SQ] } }
  var faces: [Face] { get { return [PQR, PRS, PSQ, QSR] } }
}
```

Interlude mathématique

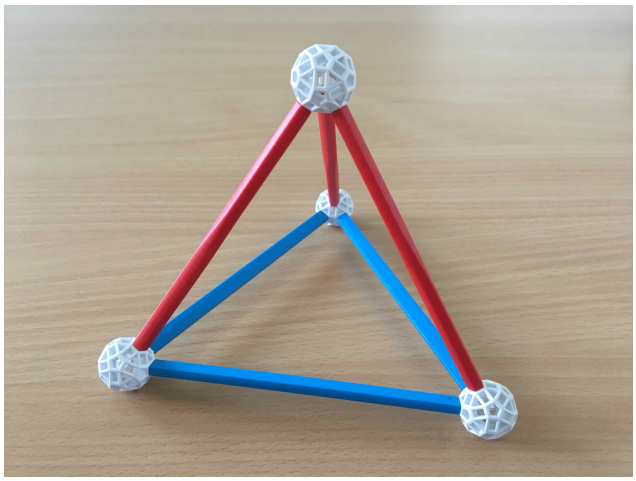
Nous détaillons à présent la construction du tétraèdre régulier et faisons quelques calculs afin de déterminer des coordonnées explicites pour ses sommets.

On désigne par $\mathbb{R}^4$ l'espace affine euclidien de dimension 4 muni du produit scalaire usuel et rapporté au repère canonique $(O, (\vec{e_1}, \vec{e_2}, \vec{e_3}, \vec{e_4}))$.

Le tétraèdre régulier est par définition l'enveloppe convexe des quatre points $P = O + \vec{e_1}$, $Q = O + \vec{e_2}$, $R = O + \vec{e_3}$ et $S = O + \vec{e_4}$.

L'isobarycentre de ces quatre points, c'est-à-dire le centre du tétraèdre régulier, est le point $O' = \frac{1}{4}(P + Q + R + S)$.

En coordonnées, on a donc $P = (1, 0, 0, 0)$, $Q = (0, 1, 0, 0)$, $R = (0, 0, 1, 0)$, $S = (0, 0, 0, 1)$ et $O' = \frac{1}{4}(1, 1, 1, 1)$.



Les quatre points P, Q, R, S, et donc le tétraèdre régulier en entier, appartiennent à l'hyperplan affine euclidien $\mathscr{H}$ d'équation $x_1 + x_2 + x_3 + x_4 = 1$.

Un vecteur normal unitaire à cet hyperplan est le vecteur $\vec{n} = \frac{1}{2}(1,1,1,1)$.

On vérifie facilement que $(O', (\vec{u_1}, \vec{u_2}, \vec{u_3}))$ est un repère orthonormal de l'hyperplan $\mathscr{H}$, où $\vec{u_1}$, $\vec{u_2}$ et $\vec{u_3}$ sont, dans la base orthonormée $(\vec{e_1}, \vec{e_2}, \vec{e_3}, \vec{e_4})$, les vecteurs de coordonnées $\vec{u_1} = \frac{1}{\sqrt{2}}(1,-1,0,0)$, $\vec{u_2} = \frac{1}{2}(1,1,-1,-1)$ et $\vec{u_3} = \frac{1}{\sqrt{2}}(0,0,1,-1)$.

Le calcul du déterminant des quatre vecteurs $(\vec{u_1}, \vec{u_2}, \vec{u_3}, \vec{n})$ donne 1, ce qui fait de cette base une base directe.

Des calculs élémentaires permettent désormais d'exprimer les coordonnées des points P, Q, R, S dans le repère orthonormal $(O', (\vec{u_1}, \vec{u_2}, \vec{u_3}))$ de $\mathscr{H}$.

On trouve $P = \frac{1}{2}(\sqrt{2}, 1, 0)$, $Q = \frac{1}{2}(-\sqrt{2}, 1, 0)$, $R = \frac{1}{2}(0, -1, \sqrt{2})$ et $S = \frac{1}{2}(0, -1, -\sqrt{2})$.

Le centre de la face PQR est l'isobarycentre de ses trois sommets, c'est-à-dire le point $C = \frac{1}{3}(P + Q + R)$ de coordonnées $\frac{1}{6}(0, 1, \sqrt{2})$.

On en déduit qu'un vecteur normal à cette face dirigé vers l'extérieur du tétraèdre est le vecteur $\overrightarrow{SC} = \frac{2}{3}(0, 1, \sqrt{2})$.

Le calcul du produit vectoriel $\overrightarrow{PQ} \wedge \overrightarrow{PR}$ donne le vecteur $\frac{3}{2}\overrightarrow{SC}$, ce qui assure que la face PQR est bien orientée.

On en déduit finalement la liste des quatre faces orientées, à savoir les faces PQR, PRS, PSQ et QSR.

Le fichier STL du tétraèdre régulier

Nous reprenons les calculs des coordonnées des sommets du tétraèdre régulier que nous avons faits précédemment, ce qui nous permet de définir quatre points P, Q, R et S qui seront les sommets de notre tétraèdre.

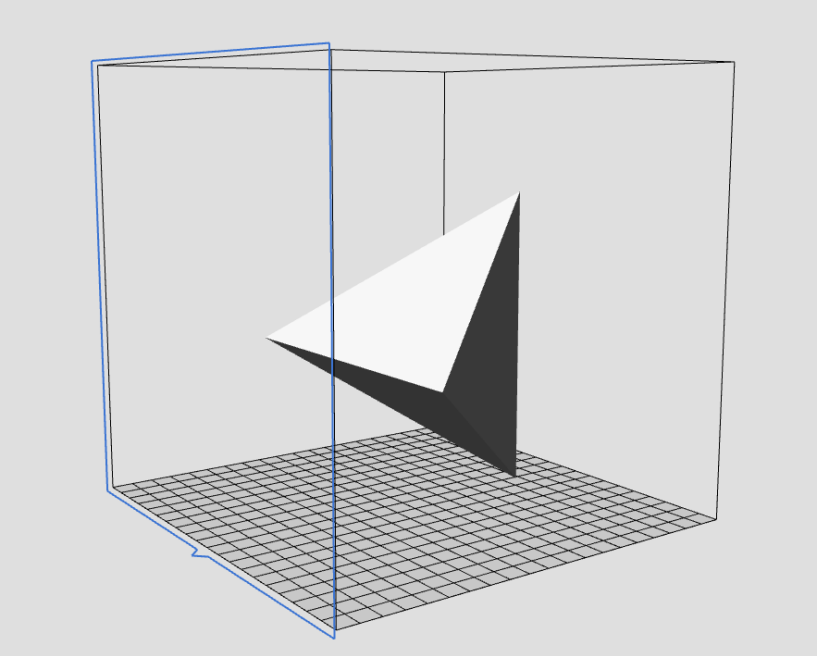
En choisissant un facteur d'échelle de 100, nous nous préparons à imprimer un tétraèdre de côté $100 \times \sqrt{2}\,\text{mm}$, soit environ $14\,\text{cm}$.

```
let P = Vertex(x: sqrt(2)/2.0, y: 0.5, z: 0.0)
let Q = Vertex(x: -sqrt(2)/2.0, y: 0.5, z: 0.0)
let R = Vertex(x: 0.0, y: -0.5, z: sqrt(2)/2.0)
let S = Vertex(x: 0.0, y: -0.5, z: -sqrt(2)/2.0)

let tetrahedron = Tetrahedron(P: P, Q: Q, R: R, S: S)

print("solid tetrahedron")
STLwrite(faces: tetrahedron.faces, scale: 100.0)
print("endsolid tetrahedron")
```

Si nous importons le code STL généré ci-dessus dans le logiciel d'impression de notre imprimante, nous constatons que le tétraèdre ne se présente pas dans une position propice à l'impression[2]...



En regardant attentivement les coordonnées des sommets du tétraèdre, on comprend que le sommet qui touche le plateau de l'imprimante est S (puisque sa coordonnée en z est la plus petite des quatre), et ce sont les points P et Q qui sont sur la même horizontale (puisqu'ils ont leurs coordonnées en z égales, en l'occurrence nulle).

On identifie sur la figure l'axe (Ox) et l'axe (Oy) en notant que les coordonnées en y de P et Q sont égales (leurs coordonnées en x étant opposées).

On en déduit qu'il faudrait appliquer une rotation de notre modèle d'un certain angle θ suivant l'axe (Ox).

2. Bien que cela paraisse évident, c'est une erreur que l'on commet souvent au début : on ne peut pas déposer de la matière dans le vide ! Ainsi, si l'on souhaite imprimer une table toute simple, constituée d'un plateau et de quatre pieds, il faut l'imprimer à l'envers, de sorte que la machine commencera d'abord par le plateau et finira par les quatre pieds.

Un calcul élémentaire de trigonométrie permet de voir qu'il s'agit en fait d'une rotation d'angle θ tel que $\tan(\theta) = -\frac{\sqrt{2}}{2}$, ou encore $\cos(\theta) = \sqrt{\frac{2}{3}}$ et $\sin(\theta) = -\sqrt{\frac{1}{3}}$.

Plutôt que de demander à notre logiciel d'impression d'appliquer cette rotation, il est presque aussi simple de reprendre notre programme et d'étendre la structure `Vect` avec une méthode qui opère comme une rotation d'angle `theta` suivant l'axe des x.

```
extension Vect {
  func rotation_x(cos_theta: Float, sin_theta: Float) -> Vect {
    return Vect(x: self.x,
      y: cos_theta * self.y - sin_theta * self.z,
      z: sin_theta * self.y + cos_theta * self.z)
  }
}
```

De là, nous définissons une fonction de rotation globale du tétraèdre : plus précisément, il s'agit d'une rotation d'angle `theta` dont l'axe de rotation est dirigée suivant l'axe (Ox) et passe par S.

```
func tetrahedron_rotation(tetrahedron: Tetrahedron,
      cos_theta: Float, sin_theta: Float) -> Tetrahedron {
  let S_rot_SP = tetrahedron.S.translate(v: tetrahedron.PS.opp()
      .toVect().rotation_x(cos_theta: cos_theta,
            sin_theta: sin_theta))
  let S_rot_SQ = tetrahedron.S.translate(v: tetrahedron.SQ
      .toVect().rotation_x(cos_theta: cos_theta,
            sin_theta: sin_theta))
  let S_rot_SR = tetrahedron.S.translate(v: tetrahedron.RS.opp()
      .toVect().rotation_x(cos_theta: cos_theta,
            sin_theta: sin_theta))
  return Tetrahedron(P: S_rot_SP, Q: S_rot_SQ, R: S_rot_SR,
      S: tetrahedron.S)
}
```

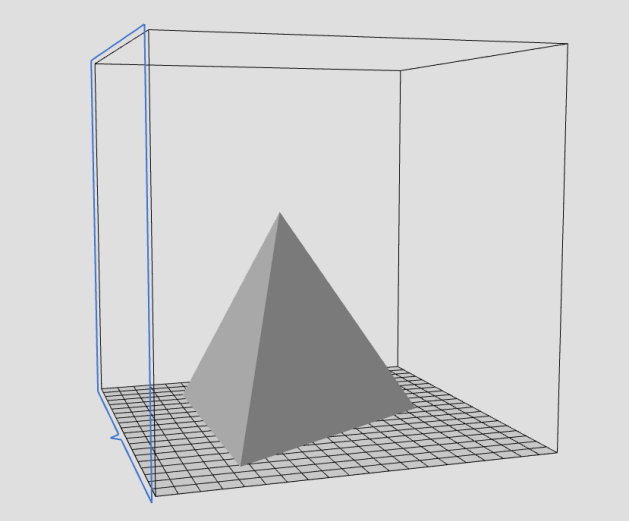
. . . puis nous retentons notre chance !

```
let cos_theta: Float = sqrt(2.0/3.0)
let sin_theta: Float = -sqrt(1.0/3.0)

let tetrahedron2 = tetrahedron_rotation(tetrahedron:
      Tetrahedron(P: P, Q: Q, R: R, S: S),
            cos_theta: cos_theta, sin_theta: sin_theta)

print("solid tetrahedron")
STLwrite(faces: tetrahedron2.faces, scale: 100.0)
print("endsolid tetrahedron")
```

Cette fois, le fichier STL généré correspond à un tétraèdre qui se présente parfaitement bien pour l'impression puisque l'une de ses faces, à savoir la face PSQ, est horizontale : il ne reste plus cette fois-ci qu'à lancer l'impression !



Nous laissons en exercice le cas du cube, objet un peu plus complexe puisque constitué de six faces carrées que l'on peut chacune trianguler avec deux triangles en choisissant une diagonale dans chaque face, ou bien par quatre triangles en rajoutant un sommet au centre de chaque face.

Fichiers sources

Tous les codes sources discutés dans cet article sont disponibles dans ce dépôt GitHub :

- le fichier SketchUp de modélisation `tetrahedron-cube.skp` ;
- un playground Swift `tetrahedron.playground` qui permet de générer les deux fichiers STL ;
- le STL du tétraèdre avant rotation ;
- le STL du tétraèdre après rotation .

Remerciements

Ce travail d'impression 3D a été réalisé au FabLab du laboratoire d'informatique (LIFO) de l'université d'Orléans.

Merci aux collègues qui fréquentent régulièrement ce lieu pour leur enthousiasme et leurs conseils avisés.

Référence

[1] Aurélien Alvarez. « Triangulation et impression 3D ». In : *Au fil des maths* n° 532 (2019).

Aurélien Alvarez est enseignant-chercheur à l'Institut Denis Poisson, le laboratoire de mathématiques des universités d'Orléans et de Tours, et formateur au centre national de la Fondation *La main à la pâte*.

Il participe à de nombreux projets de diffusion de mathématiques et est le rédacteur en chef de la revue en ligne *Image des mathématiques*.

aurelien.alvarez@univ-orleans.fr

SOMMAIRE

FIL ROUGE ..... **Les maths à portée de main**

CultureMATH

APMEP

www.apmep.fr