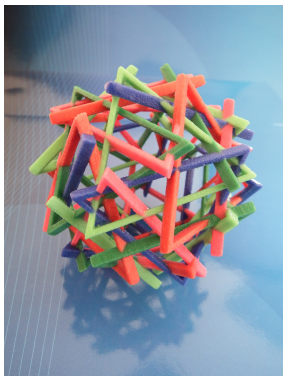
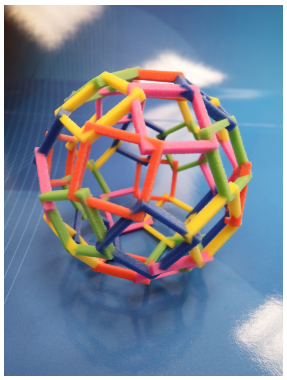
Impression 3D : une première approche

Francesco De Comité
CRIStAL
Université des Sciences de Lille
francesco.de-comite@univ-lille1.fr





Résumé

On parle souvent de l'impression 3D et des prouesses qu'elle permet d'accomplir dans les domaines médicaux ou scientifiques par exemple. Cette technique semble un peu magique, promettant de produire n'importe quel objet facilement et à moindre coût. Le but des manipulations que nous vous proposons est de vous faire comprendre la chaîne complète qui va de l'idée initiale de l'objet à construire, puis passe par sa modélisation mathématique, sa traduction en programme informatique et finalement son impression.

Ce document, ainsi que les fichiers auquel il fait référence peuvent être chargés à partir de l'adresse : http://www.cristal.univ-lille.fr/~decomite/download/downloadF.html

Première partie

Introduction

La chaîne de conception

Pour produire un objet, on utilise d'abord un logiciel de modélisation 3D dans lequel on définira de façon virtuelle cet objet. Lorsque l'objet correspond à ce qu'on veut, on le sauvegarde dans un fichier (dans un format de fihier compris par le logiciel de l'imprimante), puis on l'envoie à cette imprimante. Un programme informatique envoie alors aux moteurs de l'imprimante les commandes qui permettent de créer l'objet. Un aperçu des différentes techniques est esquissé en annexe.

Le logiciel de modélisation

Le logiciel de modélisation que nous allons utiliser s'appelle Blender[1] : c'est un logiciel ouvert, gratuit, très complet, en développement constant et disponible sur toutes les plateformes. Blender est assez compliqué à apprendre et nous nous limiterons à ce qu'il est nécessaire de savoir pour le but qui est le nôtre : définir un objet 3D.

Dans Blender, on peut définir des objets soit directement à la main, en sculptant une forme de manière interactive, soit par l'intermédiaire d'un programme : on peut écrire des scripts en Python pour construire des objets. Cela nous ouvre des perspectives nouvelles : nous pouvons définir des objets mathématiques, les programmer, et leur donner forme, passer de la définition abstraite à la réalisation pratique.

Plan de la manipulation

La manipulation se divise en plusieurs phases : le prochain chapitre vous apprendra les rudiments de Blender. La partie suivante décrira un format de représentation d'objets 3D. Les deux dernières parties se concentreront sur la construction d'objets par l'intermédiaire de la programmation en Python, incluse dans Blender.

1. Disponible à l'adresse blender.org

Deuxième partie

Les bases de Blender pour l'impression 3D

Mise en route

Blender est présent par défaut sur toutes les distributions Linux, il existe des installateurs automatiques pour Windows et Mac. De nouvelles versions apparaissent souvent (en moyenne deux fois par an). Nous utiliseront cette année (2018) des variantes de la version 2.7 (les illustrations dans ce document proviennent de la version 2.76).

Après avoir lancé Blender, vous obtenez l'écran de la figure 1 page 20. Cet écran contient quatre zones principales :

- La vue 3D qui contient la scène en cours de définition. Cette scène est composée des objets suivants :
  - Un cube, au centre de l'écran. Il est entouré d'un liseré orange : il a été sélectionné, c'est un objet actif (celui sur lequel on peut appliquer des actions).
  - Une caméra (la pyramide à gauche de la vue) : elle définit ce qui sera vu lorsqu'on fera un rendu (une photo) de la scène.
  - Une lampe, hors de vue pour le moment, qui définit l'éclairage de la scène pour le rendu.
- A gauche, une fenêtre avec plusieurs onglets, regroupant les outils les plus utilisés.
- En haut à droite, la liste des objets présents dans la scène. On y retrouve le cube, la lampe, la caméra et le monde (couleur du ciel, propriétés de la lumière ...)
- En dessous de cette liste, la fenêtre des propriétés qui permettent d'agir sur la définition de la scène de beaucoup de manières différentes.

La plupart des menus et des contenus des fenêtres sont contextuels : ils dépendent des objets sélectionnés et des actions en cours.

La vue 3D

Quelques notions de base sur la navigation dans la fenêtre Vue 3D :

- Assurez-vous que le curseur de la souris soit dans cette fenêtre (la fenêtre active est toujours celle où se trouve la souris).
- En bougeant la souris tout en maintenant le bouton central enfoncé, on effectue une rotation dans l'espace. Tourner la molette centrale de la souris permet de zoomer en avant ou en arrière.
- SHIFT-tourner la molette centrale déplace la scène verticalement.
- CTRL-tourner la molette centrale déplace la scène horizontalement.
- Cliquer droit sur un objet le sélectionne.
- La touche A sélectionne ou désélectionne tous les objets.
- Les touches du pavé numérique modifie l'angle de vue et le type de perspective. En particulier, la touche '0' correspond à ce que voit la caméra. La touche '5'

permet de passer du mode perspective au mode 'ortho' et vice-versa. Essayez...
— La combinaison de touches CTRL-ALT-Q scinde la vue 3D en quatre fenêtres correspondant à quatre points de vue différents. La même combinaison permet de revenir à une seule fenêtre.
— CTRL-flèche vers le haut met la fenêtre active (celle où se trouve la souris) en mode plein écran.
— CTRL-flèche vers le bas restitue la taille initiale de la fenêtre.
— CTRL-Z annule la dernière commande.
— CTRL-U sauvegarde la configuration actuelle de Blender (forme et disposition des fenêtres, composition de la scène ...).

Les objets

Manipulation

Quelques notions de base pour la manipulation des objets :
— Assurez-vous que votre souris est bien dans la fenêtre vue 3D, et sélectionnez le cube (seulement le cube : d'abord, désélectionnez tout (A) puis cliquez droit sur le cube).
— La touche 'G' (pour Grab : empoigner) permet de déplacer l'objet, soit en mode libre, dans une fenêtre en perspective, soit dans un plan (fenêtres ortho), soit selon une des directions X,Y ou Z.
  — Les points de vue 'ortho' ou 'perspective' se choisissent avec le pavé numérique. Rappel : la touche '5' permet de passer d'un mode à l'autre.
  — Quelque soit la fenêtre, la commande 'G' toute seule permet de déplacer l'objet dans n'importe quelle direction.
  — Les commandes 'G X', 'G Y, 'G Z' restreignent les mouvements de l'objet à une seule direction. Les directions des axes sont indiquées en bas à gauche de la vue 3D.
— La touche 'R' fait de même pour les rotations de l'objet.
— La touche 'N' ouvre une fenêtre à droite de la vue 3D, qui contient toutes les informations concernant l'objet sélectionné (ou concernant le dernier objet à avoir été sélectionné) : position, orientation, taille..., ainsi que des informations plus générales sur la scène (angle de prise de vue ...). Appuyer une deuxième fois sur 'N' referme ce volet.

Comment sont définis les objets

Avant d'aller plus loin et de construire nos propres objets, voyons comment Blender les décrit et les définit. Dans la fenêtre Vue 3D, sélectionnez le cube. En bas à gauche de la fenêtre, une case contient le texte `Object Mode` (figure 2).

Cliquez sur cette case : dans le menu déroulant qui apparaît, choisissez Edit Mode.

Remarquez comment la barre qui contient ce bouton a changé (figures 2 et 3).

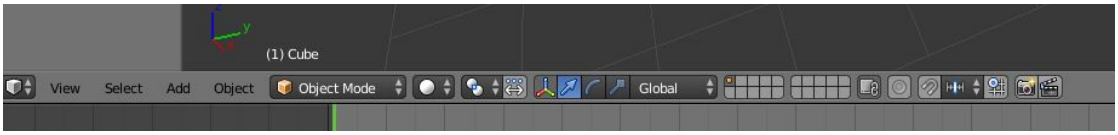


FIGURE 2 – Le bandeau inférieur en mode objet

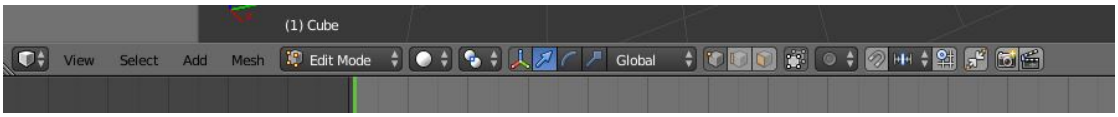


FIGURE 3 – Le bandeau inférieur en mode édition

En mode édition, on a accès aux composants primitifs de l'objet : les sommets (vertex/vertices), les arêtes (edges) et les faces. La définition d'un objet 3D peut souvent se ramener à la description de ces composants : un objet est une union de faces, chaque face est limitée par des arêtes, chaque arête est définie par deux sommets. En mode édition, la barre du bas de la vue 3D contient trois boutons (figure 4) qui permettent de choisir si l'on veut travailler au niveau de l'arête, du sommet ou de la face.

FIGURE 4 – Les trois choix en mode édition

Une fois choisi le niveau de contrôle (sommet, arête, face), on peut sélectionner les composants de l'objet :

— Les uns à la suite des autres : sélectionnez le premier (cliquer droit) puis maintenez la touche CTRL enfoncée en sélectionnant le suivants.
— par bloc : Appuyez sur la touche B, puis en cliquant sur le bouton central de la souris, et en le déplaçant tout en le maintenant enfoncé, définissez un rectangle. Tout ce qui se trouvera dans ce rectangle sera sélectionné. Il est important de choisir la vue la plus pratique pour cette sélection.

Outre les transformations décrites plus haut (déplacements G et rotations R), on peut maintenant étirer un composant avec la commande E (Extrude : extrusion). C'est de cette façon qu'on peut sculpter des objets, avec un peu de patience et de savoir-faire [2].

2. Il existe aussi dans Blender une option Sculpt Mode, au même endroit qu'Object Mode et

Troisième partie

Coder un objet 3D

Problématique

Divers programmes permettent soit de concevoir, soit de représenter des objets tridimensionnels. Ces programmes ont souvent besoin de se transmettre les objets (par exemple un programme de modélisation 3D comme Blender construit un objet, puis le transmet à un programme de rendu graphique comme Povray, ou bien à une imprimante 3D, ou bien un concepteur veut envoyer la description de l'objet qu'il a créé à un collègue ...). Pour cela, on a besoin de définir des standards de descriptions de fichiers adaptés. Les différents formats de fichiers décrivant des objets tridimensionnels partagent des caractéristiques communes :

— Un objet est défini comme un ensemble de *faces*.
— Chaque face est décrite par la suite des *sommets* qui la composent.
— Chaque sommet est défini par ses *coordonnées* dans l'espace à trois dimensions.

Les différences entre les différents formats peuvent apparaître :

— Au niveau de la forme possible des faces (triangles, quadrilatères, polygones quelconques ...)
— Informations sur la couleur éventuelle de l'objet (objet complet monochrome, couleur par face, couleur par sommet ...)
— codage compact des informations (format texte, binaire ...)

Blender, comme les autres programmes de modélisation 3D, est capable de charger des fichiers de différents types, et de sauvegarder un objet dans n'importe lequel de ces formats. Dans cette partie de la manipulation, nous allons nous intéresser à un format largement utilisé dans la conception et l'impression 3D, le format STL.

Le format STL

Le format STL (pour **ST**éréo-**L**ithographie) permet de décrire un objet comme une collection de triangles. Chaque triangle est défini comme dans la figure 5 :

— Le mot-clé *facet* suivi de la direction du vecteur normal à la surface (utile pour calculer la réflection des rayons lumineux lors des rendus).
— Le mot-clé *outer loop* qui ne sert jamais à rien.
— Les trois coordonnées des sommets (*vertex*) du triangle.
— La fin de loop, et la fin de facet.

Chaque triangle génère donc sept lignes dans le fichier. Un objet un peu fin et compliqué peut contenir des millions de triangles élémentaires : il existe une variante du format STL où seuls les 12 nombres réels sont utilisés pour décrire une face, soit 50 octets par

Edit Mode

triangle (un nombre réel est représenté sur 32 bits, soit 4 octets× 12 plus deux octets de contrôle).

```
facet normal -0.249707 0.335139 0.908476
 outer loop
  vertex 29.226992 18.707151 44.539494
  vertex 46.159988 16.625916 49.961536
  vertex 36.518944 26.357403 43.721592
 endloop
endfacet
```

FIGURE 5 – La définition d'un triangle en STL

Un peu de pratique

Allez chercher le fichier `lapin.stl` à l'adresse :

http://www.cristal.univ-lille.fr/~ decomite/download/downloadF.html

Relancez Blender ou bien nettoyez-le de tous ses objets. Dans le menu File, choisissez Import puis Stl et charger `lapin.stl`. Un lapin énorme apparaît dans la fenêtre, vous pouvez le manipuler, le modifier comme n'importe quel objet créé dans Blender (sélectionner des arêtes, des faces, les extruder ...). Passez en mode édition pour visualiser les triangles du fichier `lapin.stl`, que vous pouvez lire dans n'importe quel éditeur de texte. De combien de triangles est composé cet objet ? (vous avez deux façons indépendantes de trouver la réponse à cette question).

Illustrons les capacités de Blender en affinant cet objet (le rendre plus arrondi, moins rugueux). Pour ce faire, nous allons utiliser un modificateur : c'est un algorithme qui augmentera le nombre de triangles en les orientant de façon à arrondir les angles (au sens propre).

- Placez-vous en Object mode et sélectionnez le lapin.
- Dans le menu Properties (fenêtre en haut à droite, sous la composition de la scène), choisissez `Modifiers` (la petite clé anglaise), puis `Add Modifiers`
- Un menu de quatre colonnes s'ouvre alors. Choisisser `Subdivision surface` (dans le bas de la deuxième colonne). La fenêtre `View` vous permet de choisir le nombre d'itérations qui seront appliquées (une itération divise un triangle en plusieurs triangles plus petits). Attention, le nombre de faces croît très vite.
- Lorsque le résultat vous convient, n'oubliez pas d'appliquer réellement la transformation (pour le moment, vous n'en aviez qu'une prévisualisation).
- Vous pouvez tester l'effet des autres transformations...

Une fois que vous avez adouci l'objet, vous pouvez faire l'opposé de ce que vous

avez fait au début : sauvegarder le résultat sous forme d'un fichier STL. Allez dans le menu File, Export et choisissez STL. Si Blender est capable de lire les fichiers STL au format texte, il ne les écrit que sous format STL binaire, et donc vous n'arriverez pas à le visualiser dans un éditeur de texte.

STL et impression 3D

Tous les types d'imprimantes 3D construisent l'objet qu'on leur demande de synthétiser couche par couche (exactement comme des courbes de niveau en topographie). Le format STL est tout à fait adapté à cette manière de faire.

- Une courbe de niveau est le contour de l'objet coupé par un plan donné.
- Tous les triangles ne sont pas concernés pour un plan donné.
- Seuls les triangles qui ont deux points d'un côté du plan, et un point de l'autre côté seront coupés.
- Chaque triangle coupé par le plan engendrera un segment de droite, dont les extrémités se situent sur les côtés du triangle reliant le sommet solitaire aux deux autres.
- Un peu de géométrie élémentaire permet de calculer les coordonnées des extrémités de ce segment.
- Comme ces segments sont connectés les uns aux autres, ils forment le contour à imprimer.

Le lapin en carton de la figure 6 a été construit de cette façon, en dessinant les contours à partir du fichier `lapin.stl`.

FIGURE 6 – Une impression 3D sans imprimante 3D

Une autre façon de définir un objet dans Blender, c'est de calculer ses sommets, arêtes et faces (plus généralement, l'ensemble des triangles élémentaires qui le composent), et de fournir ces données à Blender, qui se chargera de l'afficher dans la vue 3D. Qui plus est, Blender nous offre la possibilité d'écrire des programmes (en Python) pour définir des objets, de les exécuter, et d'observer l'objet produit dans la fenêtre 3D. C'est ce que nous allons faire dans la suite de cette manipulation.

Quatrième partie

Programmer un objet

Lire et charger le programme

Il nous faut un programme qui calculera l'objet et l'affichera dans la fenêtre Vue 3D. Ce programme doit être chargé dans la fenêtre de script (ou Text Editor), qu'il nous faut donc ouvrir.

- Placez-vous sur le bord supérieur de la fenêtre Vue 3D. Lorsque le curseur de la souris devient une flèche à deux pointes, cliquez droit et choisissez `split area` (scinder la zone). Choisissez une scission horizontale en utilisant la touche de tabulation.
- Faites glisser la souris jusqu'à obtenir une fenêtre de taille convenable. On pourra ajuster cette taille plus tard.
- Dans le menu déroulant en bas à gauche de cette nouvelle fenêtre, choisissez `Text Editor`.
- Ouvrez le menu `Text` du bandeau inférieur de cette fenêtre.
- Choisissez `Open Text Block`, puis allez chercher le fichier `icosaedre.py`. C'est un programme Python qui décrit un icosaèdre à faces colorées. Ce programme est reproduit en annexe à la page 16.

Voyons ce code en détail :

Ligne 1 à 3 : Importation des modules Python (les fonctions mathématiques et les fonctions reliant Python à Blender)

Ligne 5 à 4 : Construction d'un icosaèdre. Ce polyèdre se compose de 12 sommets et de 20 faces : nous n'auront pas besoin des arêtes. Plus précisément :

Ligne 6 : Création d'une structure de données 'Mesh' (un maillage), vide pour l'instant. Elle contiendra la liste des sommets et des faces.

Lignes 7 et 8 : Définitions des deux listes qui contiendront les sommets et les faces.

Ligne 9 : Le nombre d'or apparaît dans la définition des coordonnées des sommets de l'icosaèdre.

Ligne 10 : Initialisation de la liste des sommets.

Lignes 11 à 22 : Les coordonnées des sommets dans $\mathbb{R}^3$. L'ordre des sommets dans la liste est important, car la définition des faces utilise l'indice des sommets dans ce tableau.

Lignes 23 à 42 : Une face est une liste de trois sommets. Chaque sommet est définit par sa place dans la liste `coords`. Ces lignes ajoutent une à une chacune des faces à la liste de toutes les faces.

Ligne 43 : On fournit à Blender les listes des sommets et des faces, on lui demande d'en faire un `mesh`.

Ligne 44 : Fin de la fonction, retour au programme principal en fournissant un objet icosaèdre.

Ligne 46 : Récupération de la scène courante.

Ligne 48 : Appel de la fonction construisant un icosaèdre.

Ligne 50 : Prévenir Blender qu'il existe un nouvel objet, nommé `icoz`, qui est un `mesh`

Ligne 51 : Intégration de l'objet à la scène courante.

Lignes 53 à 68 : Ce code sert à fixer les couleurs des faces. Les couleurs sont disposées de telle façon qu'on n'a jamais deux faces adjacentes de même couleur.

Exécuter le programme

— Dans la fenêtre Vue 3D, placez-vous en Object Mode, sélectionnez le cube, et supprimez-le (touche `Suppr`).

— Dans la fenêtre `Text Editor`, qui contient le programme, cliquez droit et choisissez `Run Script` pour exécuter le programme.

La fenêtre Vue 3D contient maintenant l'icosaèdre défini par le programme. Pour voir les couleurs, choisissez le mode `Vertex Paint` au lieu de `Object Mode`. Ou bien, dans le menu à droite du menu `Object Mode`, choisissez `Texture`.

Vous pouvez maintenant éditer l'icosaèdre, extruder une face, une arête, un sommet. Changer son échelle dans une ou plusieurs directions ('N' pour afficher le menu `Propriétés`, puis `Scale`). Vous pouvez aussi le modifier à partir du programme (changez la valeur de phi, ou la valeur et l'ordre des couleurs).

On peut aussi vérifier les propriétés de l'objet. Par exemple, chaque face a un côté interne et un côté externe. Pour que les logiciels qui manipuleront cet objet fonctionnent correctement (en particulier ceux qui devront l'imprimer en trois dimensions), il faut que tous les côtés externes de l'objet pointent vers l'extérieur. Pour cela, nous allons vérifier l'orientation des normales à chaque face. La normale à une surface est un vecteur orienté qui pointe vers l'extérieur. Pour afficher les normales des faces de l'icosaèdre, passez en `Edit Mode`, tapez 'N' dans la Vue 3D pour faire apparaître la fenêtre des propriétés, et dans le sous-menu `Mesh Display`, indiquez que vous voulez afficher les normales aux faces, avec une longueur 1 (figure 7). Si tout se passe bien, chaque face est maintenant munie d'un vecteur bleu en son centre pointant vers l'extérieur.

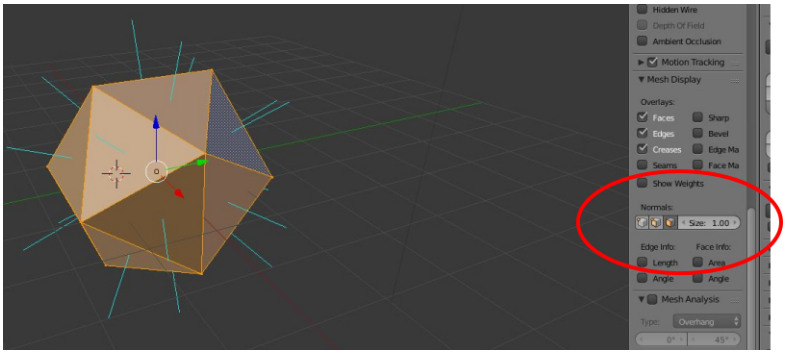


FIGURE 7 – Vérifier les normales

Que se passe-t-il si ce n'est pas le cas ? Effacez l'icosaèdre, et dans le programme, changez l'ordre des indices des sommets dans la définition d'une des faces, puis relancez le script. Passez en `Edit Mode` et affichez les normales. La face qui a été modifiée n'affiche plus de vecteur pointant vers l'extérieur. Cela risquerait de poser des problèmes lors de l'impression. Mais ce problème peut se régler facilement :

Toujours en `Edit Mode`, l'objet étant sélectionné, tapez CTRL-N pour faire pointer toutes les normales vers l'extérieur.

Prendre une photo

Vérifier que l'icosaèdre est bien dans le champ de la caméra (c'est le cas si l'objet et la caméra n'ont pas été déplacés) en choisissant le point de vue de la caméra (touche '0' du pavé numérique). Cliquez sur le menu `Render` dans le menu principal, puis `Render Image`. La 'photo' apparaît à la place de la Vue 3D, mais en gris . . .On peut repasser en Vue 3D en utilisant la touche `Echap` du clavier. Pour voir les couleurs sur la photo, il faut :

- Dans la fenêtre propriétés, à droite de l'écran, choisir le menu `materials` (figure 8)
- cliquez sur `new`
- cochez `Vertex Color Paint` dans les `Options`.

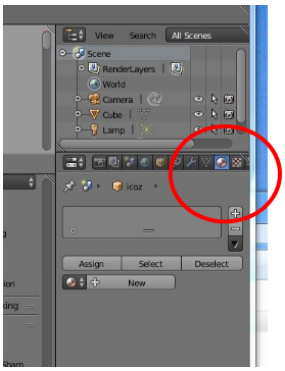


FIGURE 8 – L'onglet Material

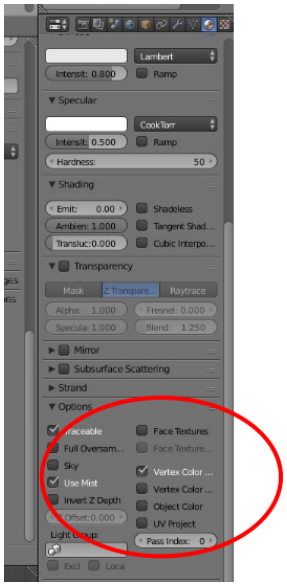


FIGURE 9 – La case Vertex Color Paint

Vous pouvez relancer le rendu, qui montrera maintenant les couleurs définies dans le programme. Obtenir un joli rendu est un travail à part entière, il faut jouer avec les éclairages et la position de la caméra etc...

Le menu `Image`, en bas à gauche de la fenêtre de rendu vous permet de sauvegarder l'image. Les dimensions et la qualité de cette image peuvent se régler dans le menu `Render` de la fenêtre des propriétés.

Exporter l'objet pour l'impression 3D

Les objets en couleur doivent avoir des parois d'au moins trois millimètres d'épaisseur pour pouvoir être imprimés sans problème par les imprimantes à sable coloré actuelles. On peut vérifier les dimensions de l'objet en ouvrant la fenêtre de ses propriétés ('N' dans la Vue 3D). On voit que ses dimensions dans les trois directions sont 3.236 (millimètres ?) Comme l'objet est plein, il n'y a pas de problème d'épaisseur des parois. Sinon, on pourrait modifier son échelle en X, Y, Z dans cette même fenêtre des propriétés de l'objet.

Pour exporter l'objet :
— Passez en `Object Mode`
— Sélectionnez l'objet.
— Dans le menu `File` du menu principal, choisissez `Export`.
— Pour une version couleur, choisissez le format x3D (le format STL suffit pour une version monochrome)

Le fichier sauvegardé peut être vérifié dans des programmes comme Meshlab (disponible en salle IVI) ou Netfabb (dont il existe une version d'évaluation gratuite). Vous

pouvez aussi charger le fichier sur un site d'impression en ligne (Sculpteo, Shapeways) qui vérifiera votre fichier, et vous indiquera son prix.

Cinquième partie

Un objet plus compliqué

Afficher l'objet

Un icosaèdre, c'est bien, mais tout le monde sait en faire un avec du papier et de la colle.Mais tout ce qui a été fait jusqu'à maintenant ne l'a pas été en vain, les manipulations apprises à cette occasion vont encore servir.

— Quittez Blender et relancez-le.
— Charger le programme `dodecaedre.py`, comme vous aviez chargé `icosaedre.py`.
— Téléchargez le fichier `donnees.txt`, et copiez-le dans le répertoire `\tmp`. Ce fichier contient des informations qui seront lues par le programme.
— Lancez le script.

L'objet obtenu est lui aussi coloré : reportez-vous aux chapitres précédents pour savoir comment faire apparaître les couleurs.

Cet objet est composé de 30 cadres carrés disposés régulièrement en suivant les arêtes d'un dodécaèdre régulier. Les couleurs sont choisies pour que deux carrés de la même couleur ne se touchent jamais. Les photos du début de ce document représentent plusieurs variantes de ce solide de base, qui ont été obtenues par la présente manipulation.

Le programme est écrit en Python. Donnons-en les grandes lignes : le programme construit 30 polygones creux, qu'il place par rotation et translation, à trente positions différentes dans l'espace. On s'arrangera pour que ces trente polygones s'intersectent, afin de construire un objet d'un seul tenant.

Les parties principales du programme sont les suivantes :

Lignes 1 à 3 : Importation des modules Python.

Lignes 4 à 12 : Définition des couleurs et de l'ordre des couleurs pour les polygones.

Lignes 13 à 41 : Définition d'une fonction déplaçant un polygone (3 rotations et une translation).

Lignes 43 à 79 : Définition d'un cadre à n côtés. Comme pour l'icosaèdre, on définit les sommets et les faces, et on regroupe le tout dans un *mesh*.

Lignes 81 à 91 : Une fonction pour colorier un objet (déjà vue dans `icoasedre.py`)

Ligne 94 : Définition de la scène courante.

Lignes 95 à 113 : Lecture des caractéristiques des transformations dans le fichier `donnees.txt`.

Lignes 115 à 132 : Création, colorisation et positionnement des polygones.

Lignes 133 à 135 : S'arranger pour que tous les polygones soient groupés (afin de pouvoir les manipuler comme un seul objet).

Modifier l'objet

Le programme `dodecaedre.py` peut être modifié à plusieurs endroits, ce qui permet d'explorer un grand nombre de formes différentes. Pour bien visualiser l'effet de chaque modification sur le résultat final, n'oubliez pas d'effacer l'objet calculé précédemment.

Changer les couleurs

Une couleur est définie en codage RGB : on fournit une liste contenant les valeurs respectives de Rouge, Vert (Green) et Bleu entrant dans sa composition (des nombres réels compris entre 0 et 1). Les couleurs utilisées pour le modèle sont définies aux lignes 5 à 9.

Changer l'ordre des couleurs dans la liste de la ligne 10 est un peu plus hasardeux : on risque de se retrouver avec deux polygones de même couleur qui se touchent.

Modifier les transformations

Les transformations (définies entre les lignes 13 et 41) peuvent être modifiées de manière homogène en changeant le coefficient 35 apparaissant à la ligne 20. Une valeur plus importante agrandira la sphère (virtuelle) qui contient tous les polygones. Une valeur plus petite la resserrera. On contrôle ainsi la taille globale du modèle.

Modifier les polygones de base

Les polygones sont définis par la fonction aux lignes 43 à 79, mais les modifications se feront lors de l'appel de cette fonction, à la ligne 119 du programme. La fonction `polygone` est appelée avec quatre paramètres :

— Le rayon du cercle circonscrit, et donc la taille du polygone.

— L'épaisseur et la largeur du cadre. Mieux vaut ne pas modifier ces paramètres : plus petits ils rendront le modèle trop fragile, et plus grands le modèle devient trop massif et trop coûteux.

— Le nombre de côtés du polygone : on peut modifier ce paramètre (en laissant au moins trois côtés...)

Plusieurs types de polygones

En plaçant une boucle autour de la ligne 119 (`myMesh=polygone(14,4,4,4)`), on peut engendrer des polygones de formes différentes, en mixant par exemple des triangles et des carrés...

Modifier l'orientation des polygones

A la ligne 121 (`transfo(myMesh,translat[ind],alpha[ind],beta[ind],0)`), en remplaçant le dernier paramètre (0) par un angle en degrés, on fait tourner les polygones sur eux-même avant de les placer. Essayez ...

Toutes ces modifications sont simples et rapides, ce qui montre que Blender est un bon outil pour la conception d'objets 3D. En vous référant à ce qui a été expliqué au dernier paragraphe du chapitre précédent (page 12), vous pouvez maintenant exporter votre objet sous forme d'un fichier imprimable. Les caractéristiques de cet objet en font un bon candidat pour les imprimantes à poudre collée (version polychrome), ou poudre fondue au laser (version monochrome) : les dimensions peuvent dans ce cas être réduites (épaisseur minimale : 1mm). Par contre, l'impression sur une machine à fusion de filament devrait s'avérer plus difficile, en particulier à cause des surplombs qui risquent de s'affaisser durant l'impression.

Quelques pistes

Normalement, on efface l'essai précédent avant de lancer une nouvelle expérience, mais rien ne nous y oblige. On peut très bien imaginer définir un objet, puis recommencer avec un objet plus petit, qui se retrouvera à l'intérieur du précédent ! Si l'ensemble de ces deux objets est exporté (ne pas oublier de les sélectionner), il pourra être imprimé sans problème particulier. On peut aussi fabriquer plusieurs copies du même objet, en les décalant les uns par rapport aux autres, tout en les laissant imbriqués les uns dans les autres...

Définition d'un icosaèdre

```
from math import *
from mathutils import *
import bpy
#Creation d'un icosaedre avec des aretes de longueur 2
def icosaedre():
    me=bpy.data.meshes.new('icosaedre')
    coords=[]
    faces=[]
    phi=(1+sqrt(5))/2
    coords=[[0,0,0] for i in range(12)]
    coords[0]=[-phi,0,1]
    coords[1]=[phi,0,1]
    coords[2]=[-phi,0,-1]
    coords[3]=[phi,0,-1]
    coords[4]=[0,1,phi]
    coords[5]=[0,1,-phi]
    coords[6]=[0,-1,phi]
    coords[7]=[0,-1,-phi]
    coords[8]=[1,phi,0]
    coords[9]=[-1,phi,0]
    coords[10]=[1,-phi,0]
    coords[11]=[-1,-phi,0]
    faces.append([4,1,8])
    faces.append([10,1,6])
    faces.append([4,0,6])
    faces.append([6,1,4])
    faces.append([8,1,3])
    faces.append([3,1,10])
    faces.append([4,8,9])
    faces.append([2,0,9])
    faces.append([9,0,4 ])
    faces.append([11,10,6 ])
    faces.append([11,0,2])
    faces.append([6,0,11])
    faces.append([5,8,3])
    faces.append([9,8,5])
    faces.append([5,2,9])
    faces.append([3,10,7,])
    faces.append([7,10,11])
    faces.append([11,2,7])
    faces.append([7,2,5])
    faces.append([5,3,7])
    me.from_pydata(coords,[],faces)
    return me
#la scene courante
scn=bpy.context.scene
#construire un icosaedre
myMesh=icosaedre()
# en faire un objet et le mettre en scene
ob = bpy.data.objects.new('icoz', myMesh)
bpy.context.scene.objects.link(ob)
#definir les couleurs des faces
myMesh.vertex_colors.new()
vertexColor = myMesh.vertex_colors[0].data
indice=0
rouge=[1,0,0]
bleu=[0,0,1]
vert=[0,1,0]
jaune=[1,1,0]
orange=[1,0.55,0]
valeur=[rouge,bleu,orange,jaune,orange,vert,vert,rouge,bleu,rouge,jaune,vert,
        bleu,jaune,orange,jaune,orange,bleu,vert,rouge]
j=0
for poly in myMesh.polygons:
    for idx in poly.loop_indices:
        vertexColor[indice].color=valeur[j]
        indice=indice+1
    j=j+1
```

Définition d'un groupe de carrés

```
import bpy
from mathutils import *
from math import *
# Listes des couleurs
rouge=[1,0,0]
bleu=[0,0,1]
vert=[0,1,0]
jaune=[1,1,0]
orange=[1,0.55,0]
couleurs=[rouge,bleu,vert,orange,bleu,vert,jaune,vert,bleu,rouge,
          orange,jaune,orange,rouge,vert,jaune,rouge,jaune,bleu,orange,
          rouge,jaune,bleu,vert,orange,bleu,orange,jaune,rouge,vert]
# les transformations
# Parametres
# me : le solide a transformer
# vectrans : la translation depuis le centre
# angalp et angbet : les deux angles de rotation (en degres)
# autoangle : rotation sur lui-meme (en degres)
def transfo(me,vectrans,angalp,angbet,autoangle):
  transly=Matrix.Translation(35*Vector(vectrans))
  rotax=Matrix.Rotation(pi/2,4,'X')
  me.transform(rotax)
  rotawa=Matrix.Rotation(autoangle/180*pi,4,'Z')
  me.transform(rotawa)
  rotaz=Matrix.Rotation(-angalp/180*pi,4,'Y')
  rotay=Matrix.Rotation(-angbet/180*pi,4,'Z')
  victor=Vector((0,1,0))
  victor.rotate(rotay)
  victor.rotate(rotaz)
  hector=Vector((0,0,1))
  hector.rotate(rotay)
  hector.rotate(rotaz)
  provy=hector.angle(vectrans)
  if vectrans[1]>0:
      provy=-provy
  rota3=Matrix.Rotation(provy,4,victor)
  me.transform(rotay)
  me.transform(rotaz)
  me.transform(rota3)
  me.transform(transly)
  return

#fonction creant un polygone
# R : rayon du cercle circonscrit
# e1 : epaisseur du polygone
# e2 : hauteur du polygone
# n : nombre de faces du polygone
def polygone(R,e1,e2,n):
 me=bpy.data.meshes.new('polygone')
 coords=[]
 faces=[]
 for i in range(4*n):
  coords.append([0,0,0])
 # definition des sommets
 for i in range(n):
  coords[i]=[R*cos(2*i*pi/n),e2/2,R*sin(2*i*pi/n)]
  coords[i+n]=[(R-e1)*cos(2*i*pi/n),e2/2,(R-e1)*sin(2*i*pi/n)]
  coords[i+2*n]=[R*cos(2*i*pi/n),-e2/2,R*sin(2*i*pi/n)]
  coords[i+3*n]=[(R-e1)*cos(2*i*pi/n),-e2/2,(R-e1)*sin(2*i*pi/n)]
 # definition des faces
 # face superieure
 for i in range(n):
  faces.append([(i+1)%n,i,n+(i+1)%n])
  faces.append([i,n+i,n+(i+1)%n])
 # face  inferieure
 for i in range(n):
  faces.append([2*n+i,2*n+(i+1)%n,3*n+(i+1)%n])
  faces.append([3*n+i,2*n+i,3*n+(i+1)%n])
 # face verticale exterieure
 for i in range(n):
  faces.append([i,(i+1)%n,2*n+(i+1)%n])
  faces.append([2*n+i,i,2*n+(i+1)%n])
 # face verticale interieure
 for i in range(n):
  faces.append([n+(i+1)%n,n+i,3*n+i%n])
  faces.append([n+(i+1)%n,3*n+i,3*n+(i+1)%n])
 me.from_pydata(coords,[],faces)
 me.update(calc_edges=True)
 return me

```

```
81  def color(me,col):
82   # Create new 'Col' Vertex Color Layer
83   me.vertex_colors.new()
84   # Vertex colour data
85   vertexColor = me.vertex_colors[0].data
86   #Fixer la couleur de tous les sommets d'un polygone
87   i = 0
88   for face in me.polygons:
89    for idx in face.loop_indices:
90     vertexColor[i].color = col
91     i += 1
92
93
94  scn=bpy.context.scene
95  # On lit les informations concernant les transformations dans un fichier
96  # (valeurs calculees dans un autre programme)
97  fichier=open('/tmp/donnees.txt')
98
99  # Premiere ligne du fichier : nombre de transformations (nombre de polygones)
100 premiereLigne=fichier.readline().split("␣");
101 nbPoly=int(premiereLigne[0]);
102 # Deux tableaux pour ranger les angles lus dans le fichier
103 alpha=[0 for i in range(nbPoly)]
104 beta=[0 for i in range(nbPoly)]
105 # Un tableau pour ranger les translations lues dans le fichier
106 translat=[[0,0,0] for i in range(nbPoly)]
107 # Lire le fichier et remplir les tableaux
108 for i in range(nbPoly):
109      ligne1=fichier.readline().split("␣") # coefficients de la translation
110      ligne2=fichier.readline().split("␣")   # angles alpha et beta
111      translat[i]=[float(ligne1[0]),float(ligne1[1]),float(ligne1[2])]
112      alpha[i]=float(ligne2[0])
113      beta[i]=float(ligne2[1])
114
115 # Construction de l'objet
116 first=1
117 for ind in range(nbPoly):
118  # Creer un polygone
119  myMesh=polygone(14,4,4,4)
120  # Tourner et deplacer le polygone
121  transfo(myMesh,translat[ind],alpha[ind],beta[ind],0)
122  # colorer le polygone
123  color(myMesh,couleurs[ind])
124
125  if(first==1):
126   ob = bpy.data.objects.new('polygon'+str(ind), myMesh)
127   bpy.context.scene.objects.link(ob)
128   bpy.context.scene.objects.active = ob
129    first=0
130  else:
131   localOb = bpy.data.objects.new('polygon'+str(ind), myMesh)
132   scn.objects.link(localOb)
133 #selectionner tous les polygons et les lier (joindre)
134 bpy.ops.object.select_pattern(extend=False, pattern="polygon*", case_sensitive=False)
135 bpy.ops.object.join()
```

L'impression 3D

Il existe plusieurs techniques d'impression 3D, mais toutes obéissent au même principe de base : un fichier décrivant la géométrie de l'objet à imprimer est transmis à un logiciel spécifique qui découpe cet objet en tranches, comme des courbes de niveau sur une carte géographique. L'objet est ensuite construit couche par couche, en suivant ces courbes de niveau.

Les différentes techniques sont :

— La fusion de filament : une buse chauffante dépose du plastique fondu en suivant les courbes de niveau. C'est le principe des machines bon marché, de l'ordre de 500 à 1000 euros (RepRap, Makerbot, Zortrax). La précision est faible, on ne peut pas créer de formes emboîtées, ni de parties en surplomb (sauf à calculer des échafaudages qui seront imprimés, et qu'il faudra couper à la fin).

— La poudre fondue : Pour chaque couche, la machine dépose une épaisseur de poudre. On passe ensuite un laser qui solidifie cette poudre selon les courbes de niveau. A la fin de l'impression, l'objet est enfoui à l'intérieur d'un bloc de poudre, qu'on aspire et qu'on récupère. La précision est excellente (de l'ordre du dixième de millimètre), les surplombs et les imbrications ne posent pas de problème. On peut même imaginer des dispositifs mécaniques avec des pièces mobiles ! On peut remplacer la poudre plastique par de la poudre métallique.

— Le sable collé : C'est le même principe que la poudre fondue : une couche de sable uniforme est déposée, et la tête d'écriture dépose de la colle le long des courbes de niveau. On en profite aussi pour colorer le contour de ces courbes de niveau, ce qui permet d'obtenir des objets polychromes. Comme pour les autres machines à poudre, on peut avoir des surplombs et des objets imbriqués et articulés. Les machines utilisant cette technique et la précédente coûtent plusieurs dizaines de milliers d'euros et ont la taille de deux machines à laver. Plusieurs compagnies vous permettent d'utiliser ces machines en envoyant vos fichiers sur leur site (Sculpteo, Shapeways).

— La résine solidifiée : un bac contient une résine qui durcit à la chaleur. Une tête laser parcourt la surface du liquide et solidifie la courbe de niveau en cours de traitement. C'est la technique utilisée par les machines de Formlabs. Ces machines coûtent environ 3000 euros.

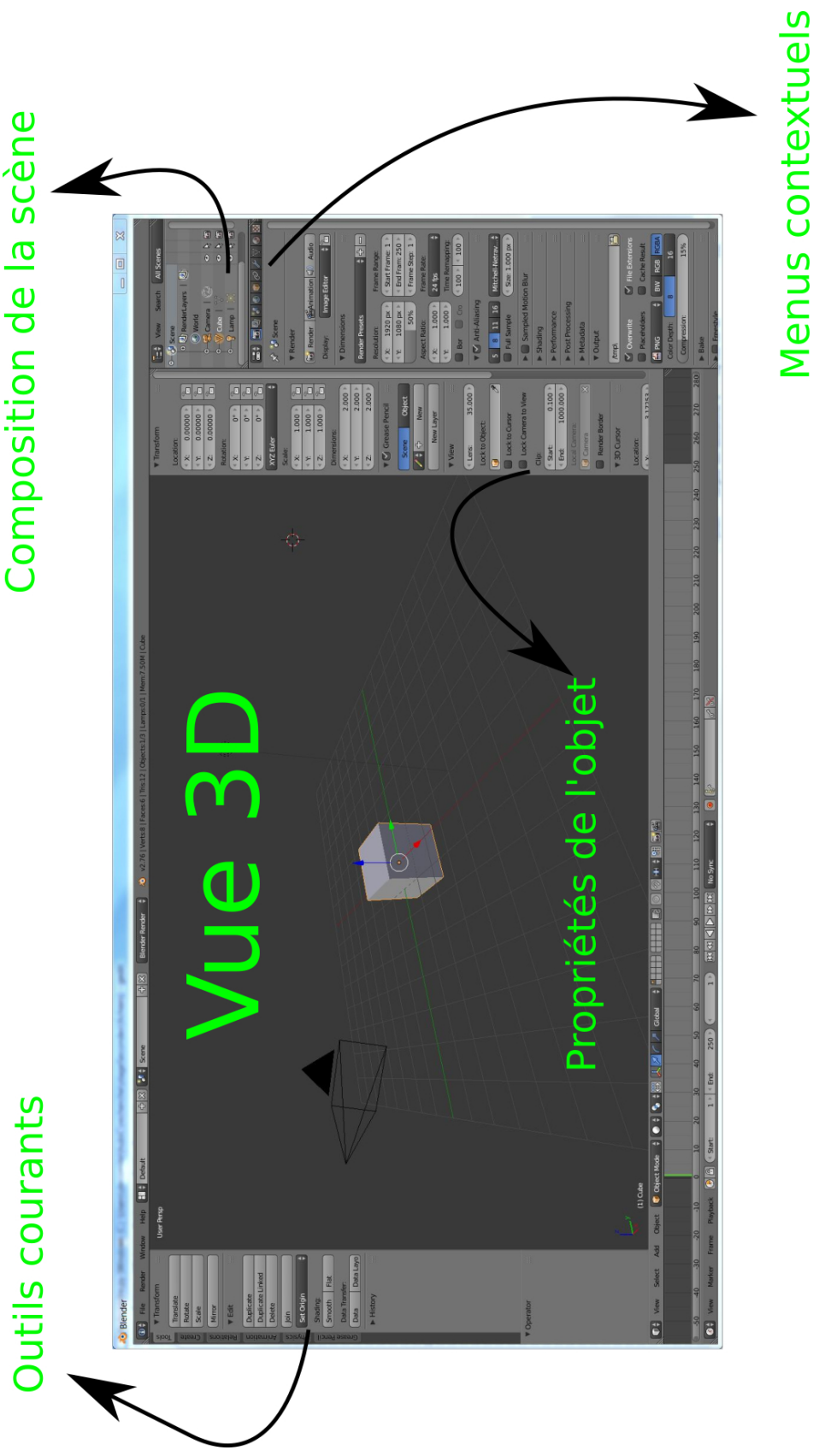


FIGURE 1 – L'écran de Blender