

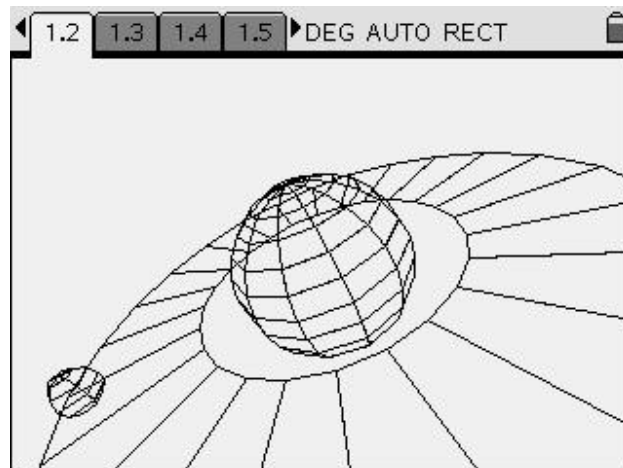
Make3D! v7

Programme de représentation tridimensionnelle pour TI Nspire CAS (ou non CAS)

Quoi de neuf depuis la version 4 ?

- Ajout d'une librairie de création
- Ajout de l'affichage des axes (vecteurs unitaires)
- Optimisation du moteur de rendu (optimisation ultérieure possible)
- Calcul approximatif du temps de rendu

Tutoriel d'utilisation basique



Note de l'auteur :

Je vous remercie de consulter ce manuel d'utilisation, prêtez-y attention et entraînez-vous à manipuler le moteur de rendu ainsi que les librairies de création. Le programme est complexe et nécessite un niveau d'utilisation avancé, ce tutoriel vous demandera une attention particulière, ne lâchez pas !

Il faut tout d'abord savoir que le concepteur de ce programme est un élève de lycée passionné de 3D en outre sur Blender, et qu'il est tout à fait possible qu'il y ait des mégardes, dans le tutoriel mais aussi dans le classeur : si le cas suivant ce produit je vous remercie de bien vouloir me contacter à l'adresse E-mail suivante :

levak@ifrance.com

Levak

Introduction

Bienvenue dans ce tutoriel d'utilisation basique du classeur Make3D! pour TI nSpire et TI nSpire CAS.

Make 3D! est un classeur assez complexe, de part sa réalisation, mais a également une approche très simple de la 3D ! En effet, contrairement aux autres traceurs de fonctions 3D qui existent sur d'autres calculatrices, Make3D! affiche le modèle à partir d'une matrice de points. L'utilité réside dans le fait que n'importe qui peut modifier ou créer des matrices de points rien qu'en connaissant les coordonnées tridimensionnelles de ce dernier !

Pour ne pas vous assommer d'exemples que certains ne comprendrons pas, je vais vous raconter l'histoire de Make3D, puis nous étudierons la liste des programmes ainsi que leurs spécificités, puis nous évaluerons les possibilités du classeur à travers quelques exemples.

Make3D! : toute une histoire !

Le projet insensé de réaliser un moteur de rendu sur TI nSpire qui, rappelons-le a de sérieuses lacunes au niveau programmation (absence de fonction Input, Menu, Output...), m'est venu pendant la réalisation de mon TPE :

« Modèle et Simulation 3D ». En effet, lorsque mon professeur de Mathématiques nous a confié, mon binôme et moi, que le fait d'utiliser un logiciel 3D (Blender) ne permettait en aucun cas de faire un rapprochement des lois mathématiques, j'ai décidé de prouver que nous n'étions pas de simples utilisateurs, et que nous avons bien assimilé les règles et théorèmes qui existent au sein de la géométrie dans l'espace.

J'ai été motivé par la popularité de mon premier jeu, mais aussi premier programme graphique sur TI nSpire : **Puissance 4!** . En effet ce jeu permet d'afficher une grille interactive qui se remplit au fur et à mesure que l'on joue avec son adversaire ! (IA en cours de développement)

Ainsi dit, ainsi fait, je me lançai donc dans la réalisation d'un moteur de rendu qui reprenait exactement les formules incluses dans mon TPE, qui m'ont permise de représenter un espace, au début « uni-visionnaire ». Grâce à l'aide de TI-Bank, j'ai réussi à améliorer l'algorithme et à rajouter des fonctions très utiles, ainsi que la possibilité de se mouvoir dans l'espace tridimensionnel.

Spécificités, Programmes et Options :

Make3D! utilise des matrices de points et des matrices de faces :

Le format de la matrice de points doit être de 4 lignes et p colonnes, où p est le nombre de points. Chaque colonne doit finir par 1, où x , y et z sont les coordonnées des points suivant les abscisses, ordonnées et profondeur. Ex :

$$mat := \begin{bmatrix} x1 & x2 & x.. \\ y1 & y2 & y.. \\ z1 & z2 & z.. \\ 1 & 1 & 1 \end{bmatrix}$$

Le nom de la matrice des lignes doit être identique à celle des points et suffixée par « `_edge` ». C'est une matrice de m lignes et 3 ou 4 colonnes, où m est le nombre de faces de l'objet. Suivant que l'objet comporte des triangles ou des quadrilatère, il y aura 3 ou 4 colonnes. Ex :

$$mat_edge := [1 \ 2 \ 3 \ 4]$$

Cette matrice de face va tracer 4 lignes (polygone fermé) entre les points 1 et 2 puis 2 et 3 puis 3 et 4 et enfin 4 et 1, pour former un quadrilatère. L'ordre est important !

Donc la matrice $carre := \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ et $carre_edge := [1 \ 2 \ 3 \ 4]$

traceront un carré grâce à la commande `make3d('carre')` !

Le modèle est visible dans la partie graphique précédant la page de calcul prévue pour accueillir les commandes.

Il est à noter que le temps de rendu est exponentiel, plus il y a de points plus le temps de rendu sera long (2 minutes pour 290 points) car la calculatrice n'est pas un ordinateur et les fonctions qui lui sont demandées sont prévues pour des plate formes plus performantes !

Make3D! comprend 11 programmes, 31 variables, 4 listes et 2 chaînes de caractères.

- **make3D**(*ChaîneDeCaractère*) : Le programme principal, c'est par lui qu'il faut lancer le programme, en mettant entre guillemets le nom de la matrice de points
- **projection**(*Matrice*) : Algorithme de projection 3D
- **segment**(*Matrice*) : Algorithme de tracé de lignes
- **ligne**(*a,b*) : trace une ligne entre le point a et b ; dépend de **line**(*x,y,i*)
- **line**(*x,y,i*) : trace la ligne dans la partie graphique dans un nuage de points.
- **plot**(*x,y,i*) : trace un point dans la partie graphique dans un nuage de points.
- **Initiate**() : RAZ du nuage de point.
- **rot_axes**(*xa,ya,za*) actualise la disposition des axes pour éviter le rendu.
- Outils de transformation :
 - **translate**(*ChaîneDeCaractère, x, y, z*)
 - **scale**(*ChaîneDeCaractère, x, y, z*)
 - **rotation**(*ChaîneDeCaractère, x, y, z*)

La chaîne de caractères correspond au nom de la matrice de points.

Les paramètres de Make3D! Sont disponibles dans les 2 tableurs :

- **Offset** : Distance entre l'œil de l'observateur et la « fente » de la caméra.
Remarque : lorsque *zoff* est > 50, la vue se met automatiquement orthonormale
- **Rotation** : Rotation de l'objet suivant le repère global.
- **Échelle** : Homothétie de l'objet suivant le repère global.
- **Position** : Translation de l'objet suivant le repère global.
- **Ne pas afficher...**: valeur à 1 masque les commentaires, à 0, les affiche.

- **Modifier...**: Écrase la matrice de points d'origine par la matrice après transformation. (*déconseillé*)
- **Objet actif : NE PAS MODIFIER LA VALEUR** sauf en cas de bug, si le modèle actif ne correspond pas au modèle demandé.
- **Sens des normales** : le mode « solid » permet de cacher les faces non visibles d'un objet convexe, il est possible que le sens soit inversé. La valeur doit être négative ou positive, mais non nulle.
- **Mod wire/solid** : Affichage en mode fil de fer ou en mode solide (seulement pour les objets convexes)
- **Afficher les axes** : valeur à 1 affiche les vecteurs unitaires servant d'axe global à la scène, à 0 les retirent.
- **Proportion des axes** : Définit la taille unitaire des axes.

La librairie nommée `mk3D_lib.tns` est à placer dans le dossier `MyLibs` de l'unité nomade, et il faut donc actualiser les librairies depuis le classeur `Make3D.tns`

- **Duplicate_mesh("Obj", "Mesh", x, y, z)** duplique les coordonnées de *Mesh* translattée de *x*, *y* et *z*, pour la stocker dans *Obj*.
- **Duplijoin_mesh("Obj", "Mesh", x, y, z)** duplique les coordonnées de *Mesh* et d'une copie de *Mesh* translattée de *x*, *y* et *z*, pour les stocker dans *Obj*.
- **Flip_Normals("Obj")** Inverse les normales de l'objet *Obj*.
- **Join_Mesh("Obj", "Mesh1", "Mesh2", x, y, z)** concatène les coordonnées de *Mesh1* et celles de *Mesh2* appliquées d'une translation *x*, *y* et *z*, dans *Obj*.
- **Mk_cube("Obj", L)** Crée un cube de longueur d'arêtes *L* dans *Obj*.
- **Mk_Func("Obj", Gz(x,y), {-x,x}, {-y,y}, {-z,z}, div, dim)**
Enregistre la fonction définie sur *Df* dans *Obj*, d'un pas égal à *div*, sur une grille de longueur *Dim*.
- **Mk_Quad("Obj", h, P, Def)** Crée un objet composé de quadrilatères, de hauteur *h*, et dont les deux extrémités ont *P* nombre de points, avec comme résolution *Def*, pour le stocker dans *Obj*

- **Mk_Tri**("Obj", h , P , Def) Crée un objet composé de Triangles, de hauteur h , et dont la base a P nombre de points, avec comme résolution Def , pour le stocker dans *Obj*.
- **Mk_UVsphère**("Obj", R , P , C , Def) Crée une USphère de rayon R , composée de C nombre de cercles divisés en P nombre de points dans *Obj*.
- **QuadToTri**("Obj") convertit les matrices de faces $4n$ (quadrilatères) en matrices $3n$ (triangles).
- **TriToQuad**("Obj") convertit les matrices de faces $3n$ (Triangles) en matrices $4n$ (Quadrilatères).

Activités :

- **Tracer un Cube**

Grâce à notre librairie, tracer un cube revient au plus simple utilisateur d'appréhender correctement l'espace 3D : Rentrez donc, dans une partie calcul, cette commande :

```
mk3d_lib\mk_cube("cube",2):make3D("cube")
```

Pour rentrer un **underscore** utilisez la séquence de touches suivante :

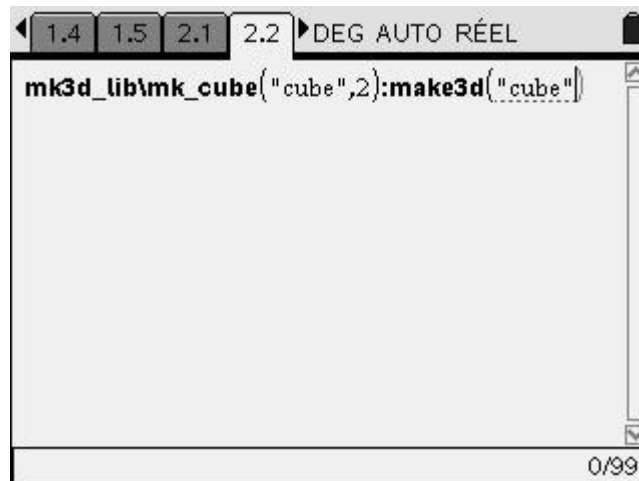
[Ctrl]+espace

Pour rentrer un **anteslash** utilisez la séquence de touches suivante :

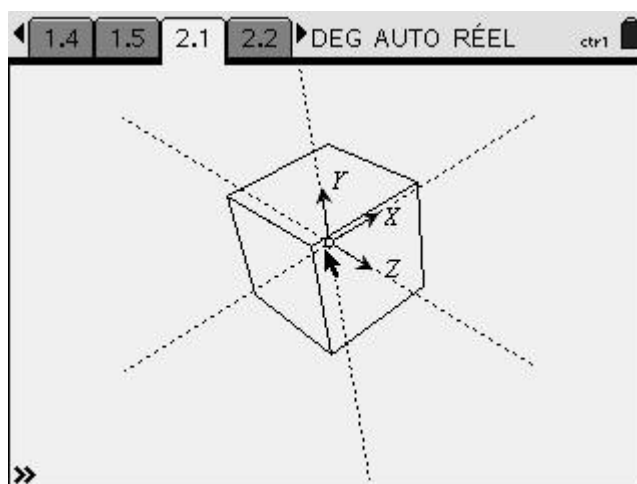
[Maj]+[÷]

Analysons la commande :

- **mk3d_lib\mk_cube("cube",2)** stocke dans la variable *cube* une matrice contenant les coordonnées des points et dans une variable *cube_edge* une matrice contenant les correspondances points/faces.
- **:** sépare deux commandes, comme en BASIC.
- **make3D("cube")** lance le rendu de l'objet *cube*.



Il ne vous reste plus qu'à valider la commande pour observer le modèle 3D dans la partie graphique qui, à la version 7, se situe en page 2.1.

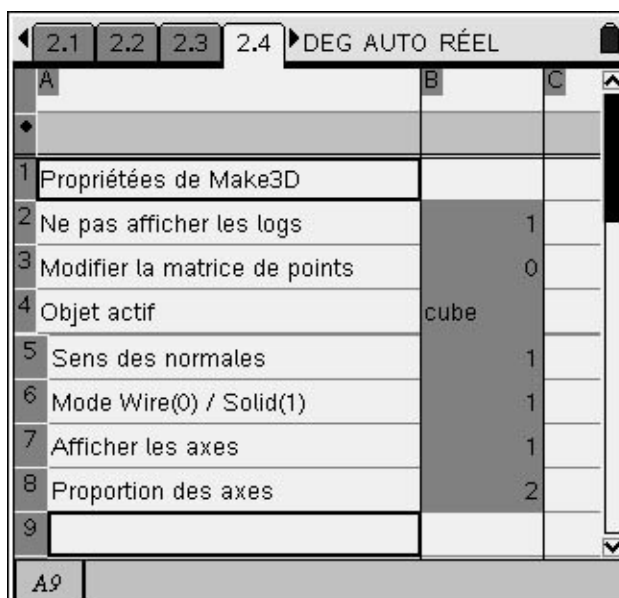


Vous remarquerez, si vous n'avez pas touché au classeur auparavant, que les axes s'affichent et que le mode par défaut est SOLID, voyons comment changer cela...

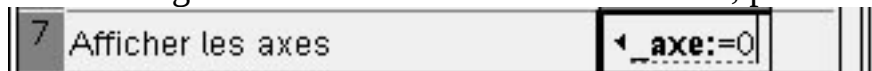
- **Changer les paramètres de Make3D!**

Nous ne voulons pas des axes, comment faire ?

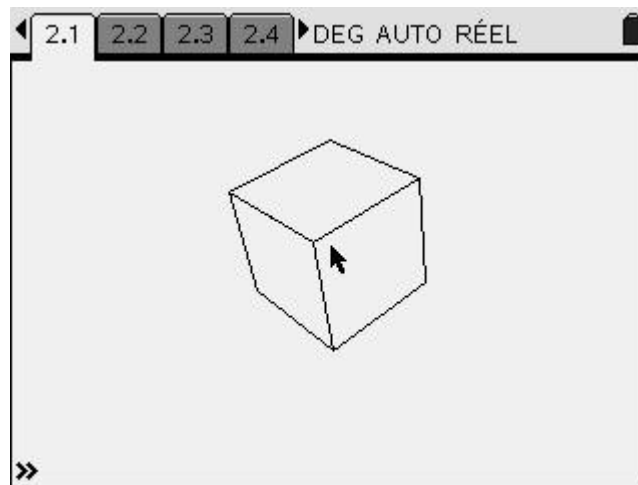
Tout d'abord reportons-nous à la page de classeur des paramètres de Make3D!, c'est à dire la page 2.4.



Nous allons changer la valeur « Afficher les axes » à 0, pour dire « false ».



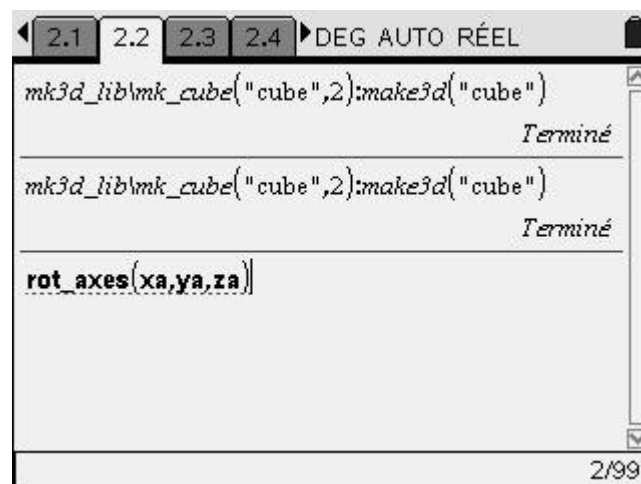
En retournant sur la partie commande et en **exécutant à nouveau la commande**, nous devrions voir les axes disparaître...



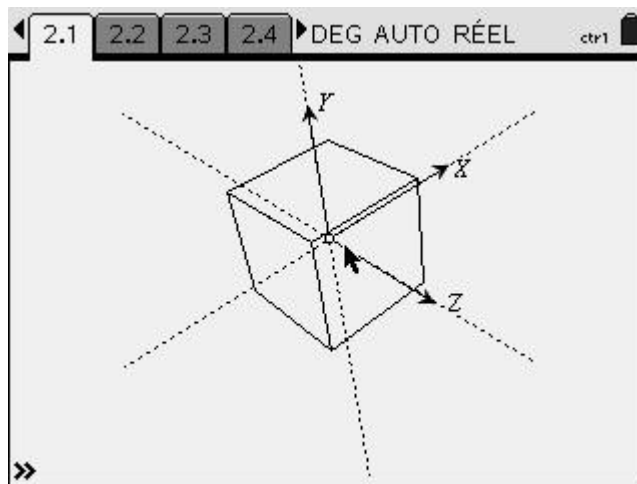
En fait, nous ne voulions pas faire disparaître les axes, mais seulement les agrandir, quelle étourderie ! Pas de problème, changeons la valeur « Proportion des Axes » en mettant une valeur plus grande, soit ici 5. *Assurez-vous de bien remettre la valeur précédente à 1 !*

7	Afficher les axes	1
8	Proportion des axes	← <u>axe</u> :=5

Puis allons dans la partie calcul et cette fois si, nous ne voulons pas rendre le modèle à nouveau pour des question d'optimisation, seulement actualiser les axes unitaires, pour cela pas de problème ! Il nous suffis d'utiliser la commande suivante : **rot_axes(xa,ya,za)**
Ne changez pas les variables par des valeurs, là est tout l'intérêt !



Et vous devriez avoir ce qui suit en partie graphique ...



ps : Vous remarquerez que si vous exécutez la commande de `make3D()` à nouveau, les paramètres seront perdus, pourquoi donc ? Dû au fait que `make3D!` prend en compte l'échelle de l'objet pour les transposer dans les paramètres de la taille unitaire des axes, histoire de toujours garder les même proportions objet/axes.

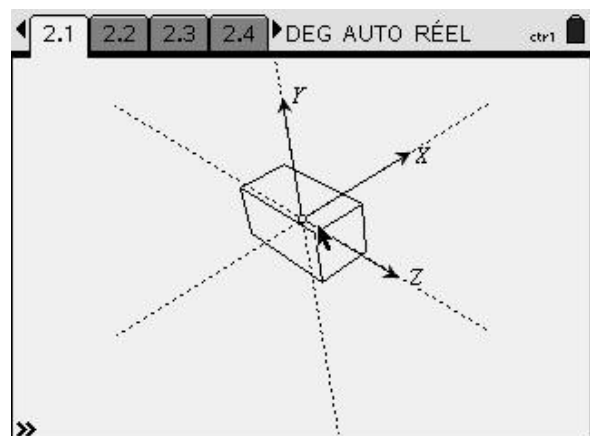
- **Changer les paramètres de l'objet actif**

Vous désirez tourner, bouger, ou réduire l'objet actif ? Rendez-vous page 2.3.

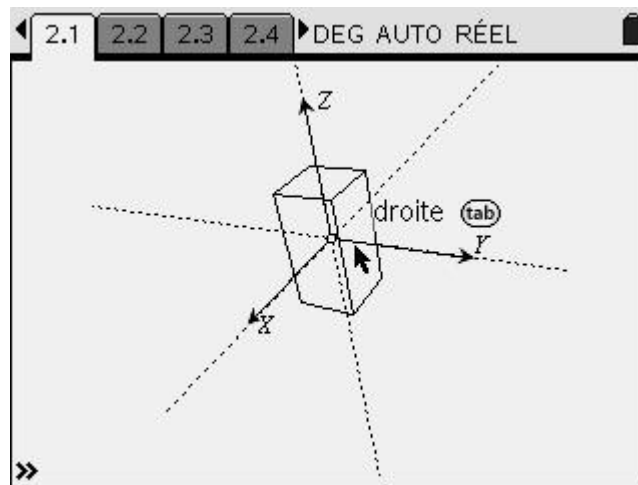
	A	B	C	D
1	Propriétés de l'Objet cube			
2	Offset sur...			
3	X		0	
4	Y		0	
5	Z		20	
6	Rotation sur...			
7	X		30	-60
8	Y		-40	10
9	Z		10	-120
10	Affinité (échelle) sur...			
11	X		2	
12	Y		2	
13	Z		2	
14	Position sur...			
15	X		0	
16	Y		0	
17	Z		0	
18				
19				
	A19			

Il y a à votre disposition 4 ensembles de paramètres, l'Offset, la Rotation, l'échelle, et la Position.

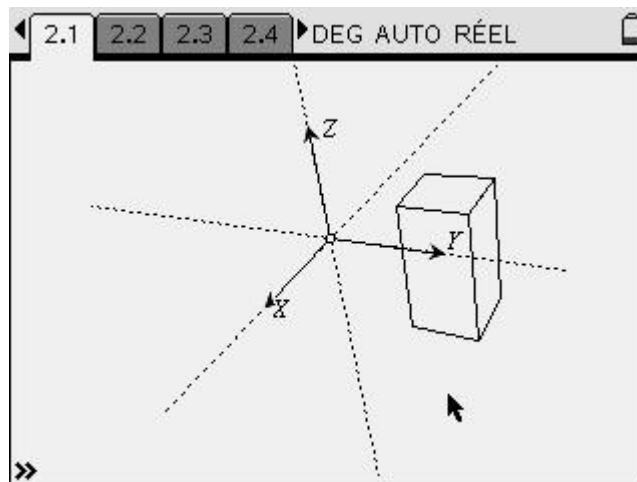
Changez par exemple, l'échelle en la mettant à 1 sur X et Y seulement. En régénérant le rendu avec la commande `make3D("cube")`, vous verrez les changements occasionnés :



Nous pouvons également tourner dans l'espace 3D :
x: -60 ; y: 10 ; z: -120



Ou encore, le translater
x: 0 ; y: 4 ; z: 0



Nous pouvons donc remarquer que l'utilisation du tableur est très pratique pour faire des transformations paramétrés. Seulement, si l'on désire opérer directement des modifications à la matrice en étant toutefois sûr de ses actes, il est possible de rentrer directement les commande relatives aux transformations à savoir :

- **translate**(*ChaîneDeCaractère*, x, y, z)
- **scale**(*ChaîneDeCaractère*, x, y, z)
- **rotation**(*ChaîneDeCaractère*, x, y, z)

Avec *ChaîneDeCaractère* le nom de l'objet entre guillemets.

- **Réaliser des transformations directes sur un objet quelconque.**

Nous venons de voir qu'il était possible de réaliser des transformations sans exécuter le moteur Make3D! qui va calculer par ce biais le rendu.

Reprenons notre scène de départ du cube, le mieux est de sortir de make3D sans enregistrer le classeur pour revenir dedans.

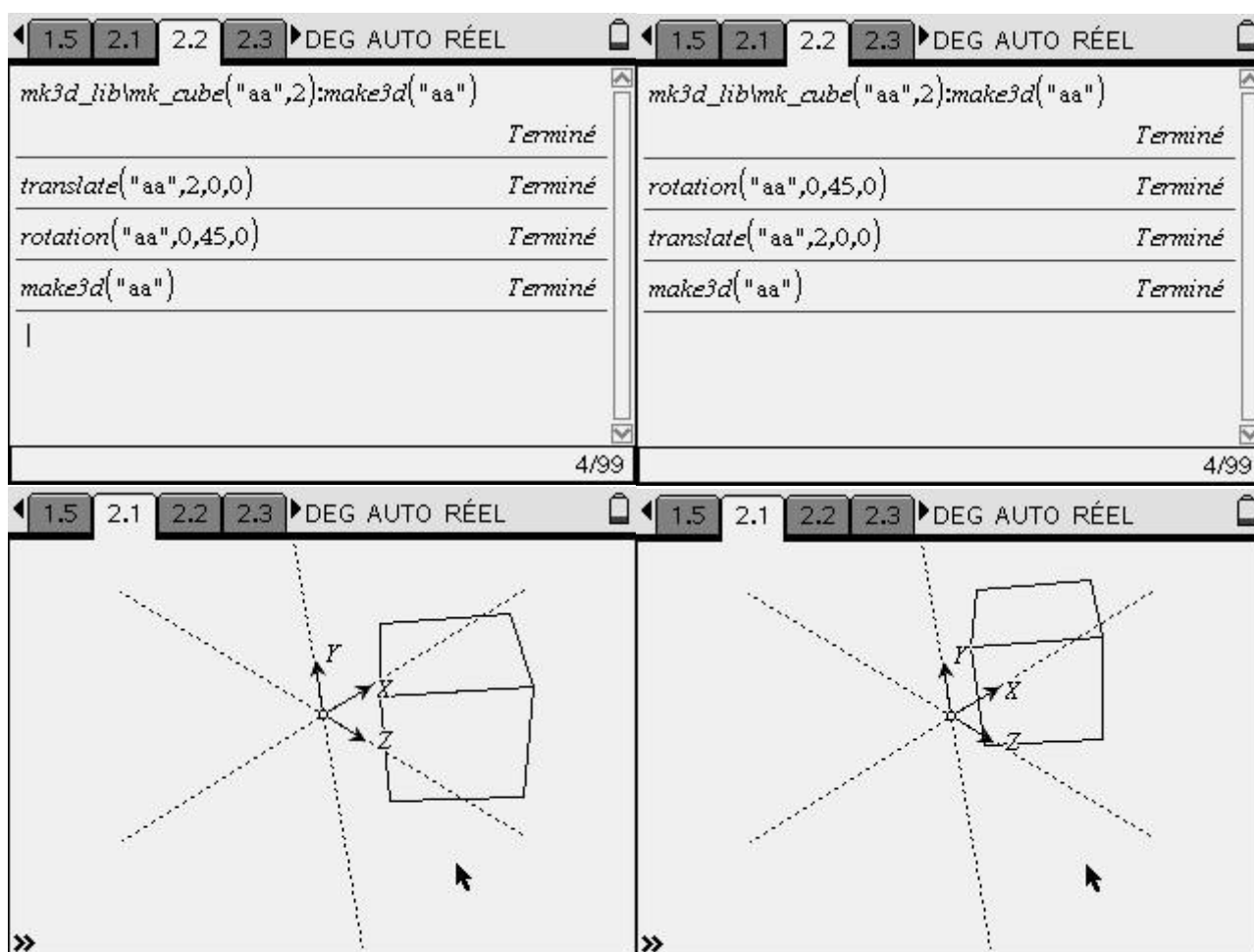
Dans une partie calcul, copier cette fonction :

scale("aa", 1, 1, 2):make3d("aa")

En d'autres termes, cela fait exactement la même chose que précédemment, sauf que (subtilité), nous pouvons exécuter autant de fois cette commande afin de réaliser diverses combinaisons de transformation.

Il vous sera donc possible de créer des modèles composés de plusieurs objets translétés, tournés et mis à l'échelle, et ce autant de fois que vous le voulez, contrairement aux tableurs qui gardent leur utilité pour le positionnement de l'observateur par rapport à l'objet observé !

Attention : l'ordre des commandes est important !



Exemple de différence de rendu avec le changement de séquence

- **Utiliser les bibliothèques de création.**