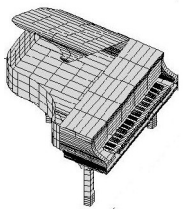
3D Crade : Manuel utilisateur

"La fantaisie et la liberté d'imagination ne s'acquièrent pas comme ça, il y faut du temps, de l'obstination, de la sévérité, de la rigueur, des mathématiques et de la raison"

Philippe Sollers



Jyce3d (2003)

Table des matières

Introduction

La construction de 3D-Crade relève d'un rêve de gosse. J'ai réellement commencé à m'intéresser à la 3D durant ma période (crise) d'adolescence dans le début des années 90. En cette époque un clip musical tel que « Let's get rock » de Def Lepard faisait figure de

révolution. Aujourd'hui le rendu utilisé passerait pour de l'amateurisme de plus bas niveau.
Au départ mes piètres connaissances mathématiques (issues du non intérêt que je portais à cette matière, le tout renforcé par quelques mois de grèves scolaires inutiles, puisque l'enseignement ne s'est visiblement pas amélioré depuis) ne me permirent pas d'évoluer très loin de la compréhension des bases de la 3D.
Par la suite, mes études d'ingénieur allaient me donner les fondements utiles, sans pour autant me donner un support explicite de la 3D. Toutefois les cours d'algèbre matriciel, de géométrie dans l'espace et d'optique donnaient toutes les réponses quantitatives voulues aux questions que je m'étais posées lorsque j'étais adolescent. Restait à remettre tout cela en ordre, et surtout à trouver le temps de le faire.
Le but de mes pérégrinations d'adolescent étaient de produire de la 3D sur ordinateur, (durant la Renaissance, certains artistes faisaient de la 3D à la main, mais le travail et le temps demandé était conséquent, temps dont je ne disposais pas). Fallait-il encore trouver le temps et les connaissances pour réaliser ce type d'application informatique.
Vers 1996, je me suis ré-intéressé quelque peu au problème, mais pour des raisons personnelles et professionnelles je n'ai pu reprendre cette recherche que dans les années 2000(fin 2003 pour être précis) pour produire 3D-Crade.

3D-Crade n'est pas une application optimisée pour faire de la 3D performante, ce n'est pas non plus un must de convivialité, toutefois elle présente des avantages intéressants sur le plan pédagogique. Que ce soit au niveau des bases de la programmation que des bases de la 3D.

En effet, un lecteur assidu et motivé prendra sans doute la peine de lire les « Aspects théoriques de base de la 3D ». Ce lecteur comprendra très vite que 3D-Crade n'est que la transposition pratique à l'informatique des principes évoqués dans ce document.

Le gros reproche que l'on peut faire à 3D-Crade est sa lenteur. Ce fait est incontestable puisque 3D-Crade n'utilise que du calcul matriciel pour afficher les scènes. Le but n'était pas de tester quel fabricant de carte graphiques était le plus habile à réaliser ces opérations au niveau de l'électronique digitale, mais bien d'implémenter concrètement et simplement des principes de bases de 3D sur le plan informatique.

Sémantique :

Objet fait référence à un objet tridimensionnel, visualisable dans le fenêtre Viewer. Il existe un certain rapport entre ces objets et les objets de la programmation orientée objet. Toutefois, ces rapports n'ont pas besoin d'être connus pour pouvoir comprendre ce document.

Scène : L'espace tridimensionnelle représenté par un repaire orthonormé composé de trois axes, dans lequel les objets 3D peuvent être ajoutés, modifiés ou supprimés. Les unités utilisées dans ce repaire sont le mètre [m]. Il faut noter que certains objets pourront être regroupé entre eux et donc apparaître dans des blocs ou sous-scènes.

Commande : Ordre que l'on peut donner au système via la fenêtre de Command, la commande possède un nom de commande et des paramètres. La commande ne renvoit pas de résultat sauf dans la partie « output» de la fenêtre de commande.

Primitive : Commande particulière qui permet de dessiner une forme. Le résultat est renvoyé dans la fenêtre Viewer.

Polygone : Un polygone est une forme géométrique fermée constituée de plusieurs côtés. En 3D, il s'agit du plus petit élément de surface pouvant être définit. Chaque surface est constituée d'une multitude de polygones.
Les polygones sont très utiles pour d'une part définir, par le calcul de leur moyenne (barycentre), l'ordre d'apparition des constituants de la surface (déterminer quelles seront les faces cachées et les faces visibles de la surface).
D'autre part l'utilisation de polygone s'avère très utile en raytracing[1] pour déterminer l'absorption de la lumière par le polygone.
Généralement les application 3D utilisent des polygones triangulaires : les avantages étant d'une part une plus grande précision quand à la représentation de la surface. Les polygones triangulaires sont simples à dessiner (on évite l'effet de papillonnement lors du tracé).
Les inconvénients majeurs étant la gourmandise au niveau mémoire.
C'est le manque crucial de mémoire sur la machine de développement qui m'a fait choisir l'utilisation des polygones rectangulaires pour 3D-Crade.
Retenez toutes fois, que toute forme tridimensionnelle sera toujours polygonalisée pour être affichée par un modeleur.
Le résultat de cette transformation en polygones d'un volume ou d'un surface, s'appelle le maillage.

MayeSurface: Surface maillée construite sur un ensemble de polygones. Chaque polygone représentant une maille de la surface. Le terme « Maye » à une origine apocryphe assez obscure voir amusante : en effet, il vient du terme Maille en français traduit par Mesh en anglais, toutefois je trouvais qu'utliser maillesurface faisait trop franglais. Il fait aussi référence au mot japonais « mae » qui signifie devant et qui est assez souvent utilisé en Karaté pour désigner tout ce qui se donne vers l'avant : « mae geri », signifie coup de pied avant, par exemple. Mais je me suis dit que beaucoup de gens prononceraient maé plutôt que « maille » et des maéé surface, ça ne le faisait pas non plus. De plus, le gestionnaire de surfaces maillées fut développé en 2004, au moment où Ozone/Haiduci avaient sorti le célèbre tube « Dragostea din tei ». J'ignorais à l'époque comment pouvait s'écrire la première partie du refrain, mais j'ai estimé que l'orthographe « maye » pouvait être appropriée, ressemblant au mae du japonais mais renforcé par l'emploi du « y ». Suite à une telle inspiration lamentable, je décidai d'utiliser par

[1] Le raytracing est une technique permettant de générer des rendus de type image de synthèses à partir d'une scène tridimensionnelle constituées de polygones. La technique consiste à calculer l'éclairement produit par une gerbe de rayons lumineux provenant d'une source sur les polygone. Les algorithmes évolués peuvent tenir compte de la réflexion, ou de la réfraction (cas de surface translucide) des rayons lumineux pour donner un résultat plus réaliste.

convention le terme « mayesurface » pour désigner les surfaces maillées, et ce en référence au français, au Karaté et à bien sûr à « Dragostea din teï » d'Haiduci.
Bon je referme là cette parenthèse inutile car je doute que même dans 3000 ans, quelqu'un s'amuse à faire de l'exégèse sur 3D-Crade et les mayesurfaces.

Partie I : Installation et Présentation de l'interface

Cette partie n'est autre qu'une approche qui vous permettra de vous familiariser quelque peu avec les bases de 3D-Crade, les opérations présentées ici sont donc très limitées. L'exploitation de l'application sera développée dans la seconde partie du document : Les arcanes de 3D-Crade.

Installation de 3D-Crade

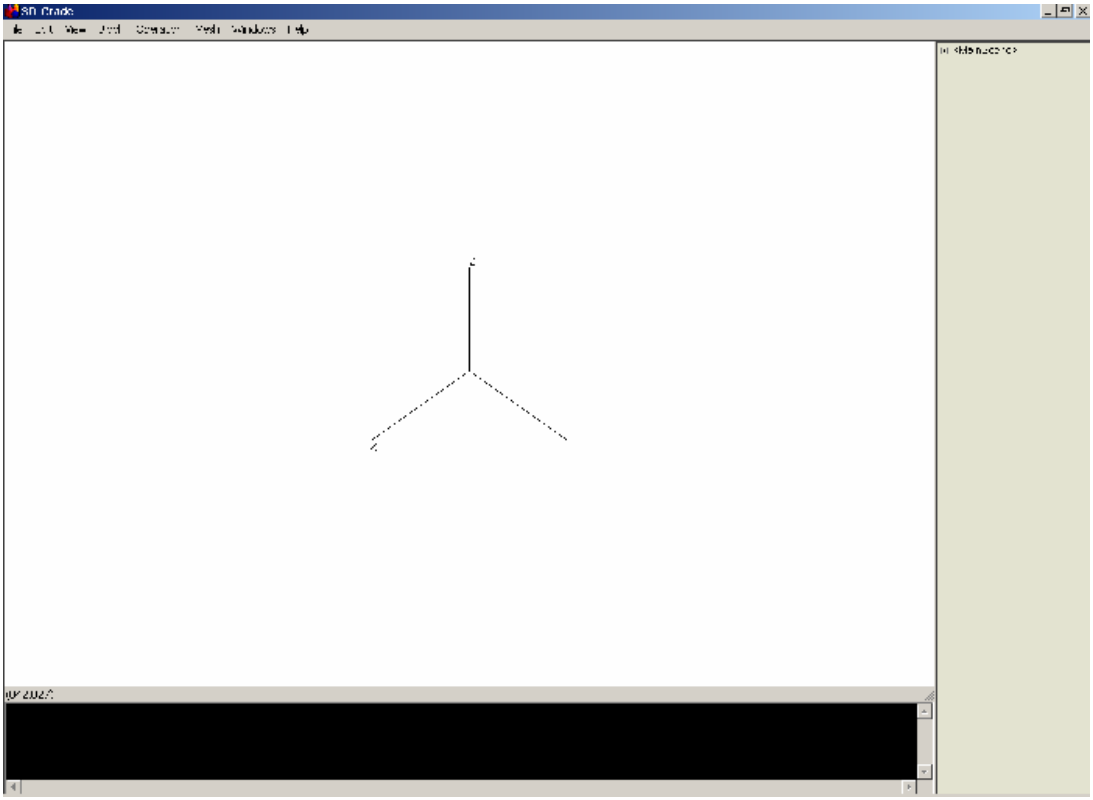
Dézipper le fichier 3D-CradeInst.zip dans un répertoire au choix (C:\ ou C:\Program Files).
L'arborescence suivante est automatiquement crée :
..\3D-Crade : contient les deux raccourcis que vous pouvez copier sur le bureau pour accéder aux éxécutables.
..\3D-Crade\bin : contient les deux fichiers exécutables : 3D-Crade.exe et BodySculptor.exe.
..\3D-Crade\Scenes : contient des exemples de scènes réalisées avec 3D-Crade.
..\3D-Crade\BodySculptor : contient des exemples de personnages réalisés avec BodySculptor.
..\3D-Crade\Scripts : contient des scripts d'exemple qui peuvent être chargés avec 3D-Crade.

3D-Crade est un système relativement léger, il n'utilise aucunes librairies additionnelles, exception faite de celles fournies en standard avec Windows. 3D-Crade n'utilise pas non plus DirectX et peut fonctionner sur n'importe quelle configuration que vous possédiez une carte graphique 3D ou pas. Les inconvénients majeurs sont sa gourmandise mémoire et sa lenteur. En effet toutes les projections sont calculées par le processeur sans aucune aide accélératrice auxiliaire, ce qui peut s'avérer ruineux en terme de rapidité.

Présentation de l'interface

Cliquer sur le raccourci 3D-Crade. Une fois l'application lancée la fenêtre principale suivante apparaît.
L'écran se décompose en quatre parties principales :

- les menus,
- la fenêtre viewer située au centre de l'écran. Dans laquelle se trouve par défaut le répère orthonormé tridimensionnel.
- La fenêtre d'objets 3D, localisée à droite.
- La fenêtre de commandes localisées en bas.



Chacune de ces sections sera décrite plus en détail dans le chapitre suivant.

Premiers pas

Nous allons débuter avec un exemple très simple : Nous allons forcer 3D-Crade à dessiner une sphère.
Pour ce faire, nous allons avoir recours à la Command Window (la fenêtre noire située en bas de l'écran).

La « Command Window » est composée de deux parties :

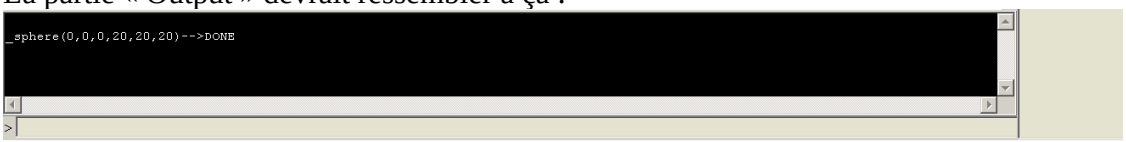
- L' « output » : sur fond noir
- L' « input » : la zone de saisie blanche localisée après le signe >

Pour commencer amener le pointeur de la souris dans cette zone de saisie et saisissez au clavier la commande suivante :

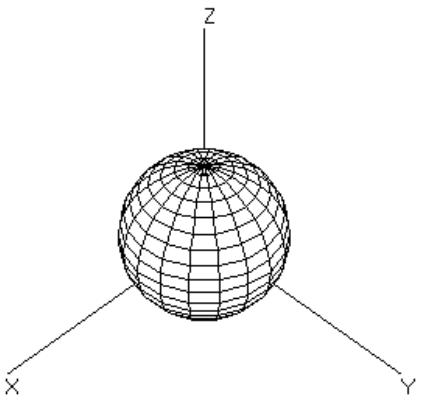
```
_sphere(0,0,0,20,20,20)
```

N'ommettez pas le « _ » et pressez « Return » pour valider la commande une fois celle-ci entrée.

Après avoir pressé « Return »
La partie « Output » devrait ressembler à ça :



La Fenêtre Viewer quant à elle devrait ressembler à ça :



Voici, votre première réalisation en 3D-Crade. Cet exemple est évidemment très simple, toutefois, il vous donne déjà une idée de ce qui vous attend.
L'utilisation des commandes peut paraître lourde et rébarbatives, toutefois, sachez que d'une part, cette technique présente un avantage insoupçonnable lorsque vous écrirez vos scripts 3D-Crade. De plus, des fenêtres de Wizard sont déjà disponibles pour certaines fonctions afin de compléter les commandes à votre place.
La plupart des commandes supportent bon nombre de paramètres optionnels qui offre une assez grande flexibilité dans les réalisations pratiques.

Les menus

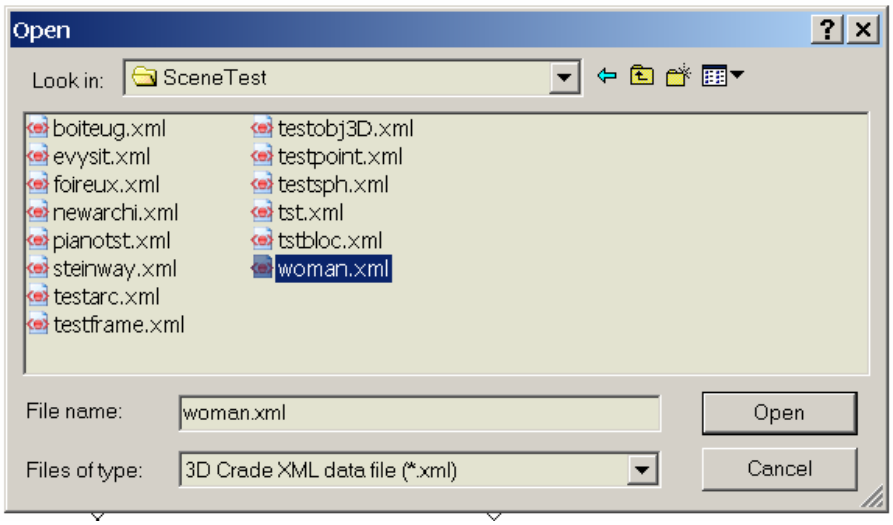
Le menu File

Les entrées de ce menu permettent de manipuler les fichiers générés par 3D-Crade. Globalement c'est ici que vous pourrez charger et sauver vos œuvres. Voyons de plus près l'utilité de chacune des entrées.

- **New** : permet de créer une nouvelle scène de travail : *Cette fonctionalité ne marche pas dans cette version.*

- **Open :** permet de charger et d'afficher un fichier de donnée scène au format .xml stocké au format 3D-Crade. Les fichiers ouvrables par le menu open ont été auparavant sauvé au moyen de l'entrée Save.

Un click sur open fait apparaître la boîte de dialogue suivante, qui vous permet de choisir un fichier.

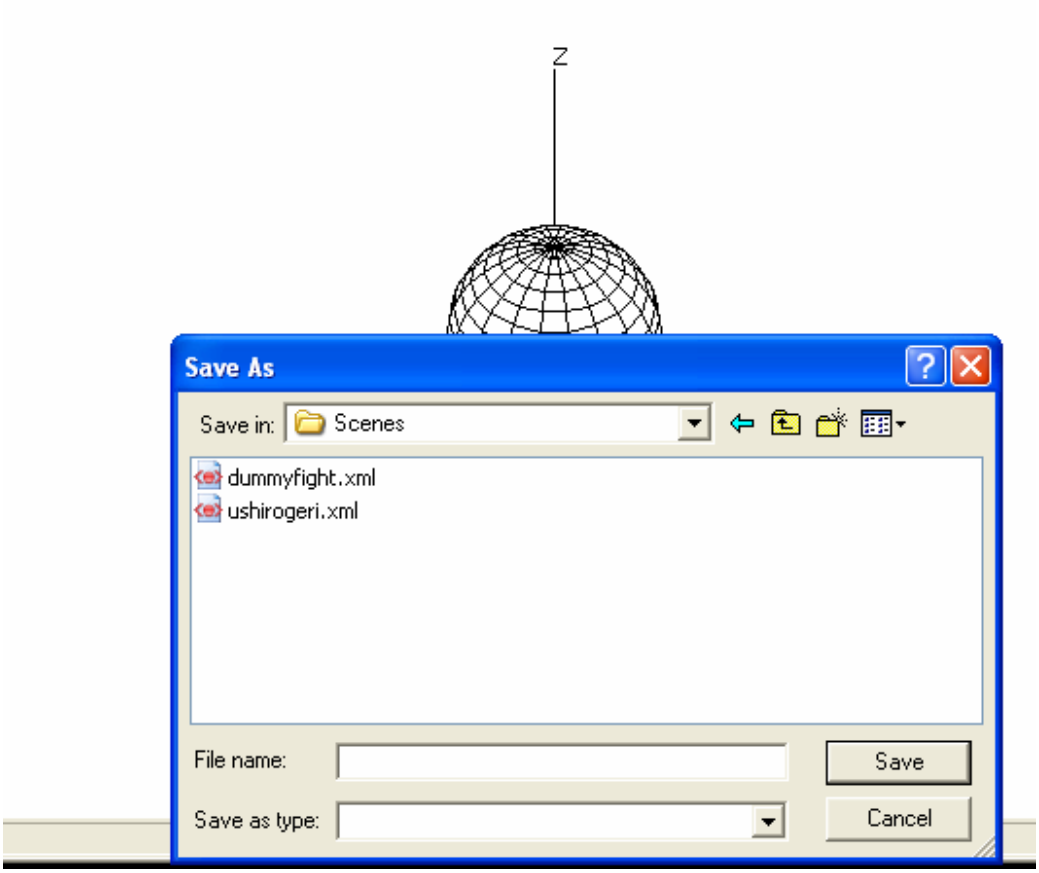


Cliquer sur open pour ouvrir votre scène.

Attention : 3D-Crade utilise un format XML pour sauvegarder et manipuler ses données brutes sur des fichiers. Toutefois, il s'agit d'un fichier XML particulier, tous les fichiers XML ne sont pas compatibles avec 3D-Crade.

Cliquer sur Cancel pour annuler l'opération.

- **Save :** permet de sauvegarder un fichier de scène sous la forme d'un fichier .XML répondant au format défini par 3D-Crade. L'entrée de menu Save fait aussi apparaître une boîte de dialogue.



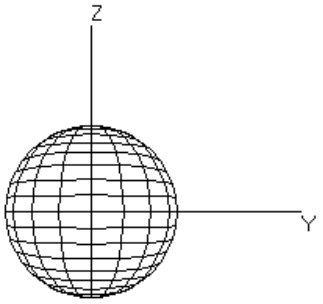
N'omettez pas le .xml à la fin du nom de fichier, si vous voulez être apte à le recharger par la suite.
Cliquer sur le bouton Save pour sauvegarder le fichier. Cliquer sur le bouton Cancel pour annuler l'opération.
Run Script : permet d'exécuter un fichier texte contenant un script 3D-Crade. Un script est un ensemble de commande 3D-Crade se suivant séquentiellement, ou répondant à certaines structures de branchements ou de répétitions, sauvegardées dans un fichier texte ayant pour extension .3ds. *3D-Studio utilisant la même extension pour ces fichiers de données, cette extension sera changée en .3dc, voir en .3dw*.
Le piano steinway (steinway.3ds) présenté sur le site est un exemple assez typique de scripts que vous pourrez être amené à réaliser avec 3D-Crade. Les scripts seront vu plus en détail dans une section ultérieure qui leur est entièrement consacrée.

Le menu Edit

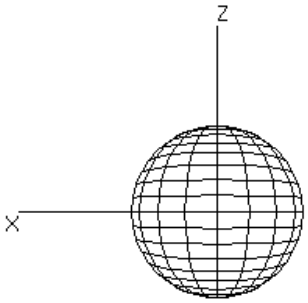
Ce menu n'est pas encore utilisé pour l'instant.

Le menu View

Ce menu permet de changer les vues de la Fenêtre viewer, quatre vues prédéfinies ont été retenues :
Face : permet d'afficher la scène en vue de face, l'appui sur la touche <F1>, donne le même résultat.

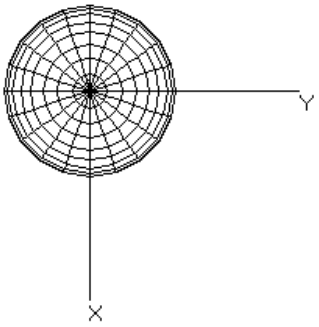


Left :permet d'afficher la scène suivant la vue de gauche, l'appui sur la touche <F2> donne le même résultat. (Dans le cas d'une sphère ce n'est pas très différent)



Vous remarquerez toutefois, qu'ici c'est l'axe des X qui est apparent, ce qui signifie que l'œil est positionné suivant l'axe des Y.

Up : Il s'agit de la vue du dessus de la scène. L'œil est placé sur l'axe des Z. Un appui sur la touche <F3> donne le même résultat :



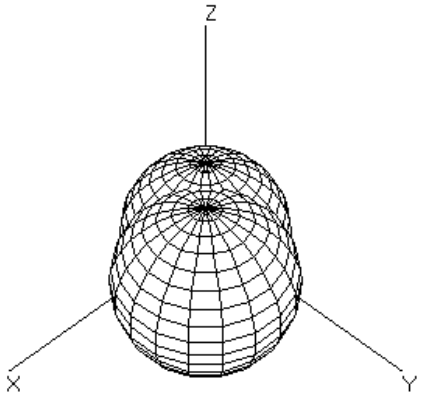
Isometric : Il s'agit ici de la vue par défaut fournie à l'ouverture de 3D-Crade. L'œil est placé à 45 degré par rapport au plan XY et à 45 degré par rapport à l'axe des Z. Nous verrons que 3D-Crade offre des mécanismes bien plus performant pour contrôler le point de vue de l'utilisateur. Qu'il s'agisse de raccourcis claviers ou de modification de variables systèmes.

Le menu Block

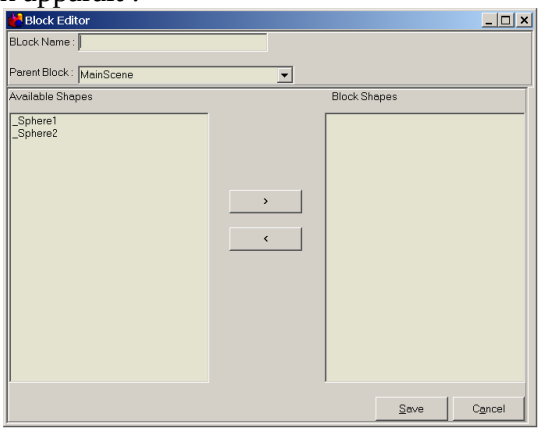
Les blocks (encore appelés sous-scènes) sont un mécanisme assez puissant de 3D-Crade permettant le développement et surtout la ré-utilisation d'objet 3D d'une scène à l'autre de 3D-Crade.

Les blocks sont crées, soit par l'interface graphique au moyen des menus. Ou directement dans les scripts 3D-Crade.
L'exportation quant à elle, ne peut se faire que via les menus.
Pour comprendre l'utilité des blocks : récréons une seconde sphère au moyen de la commande :

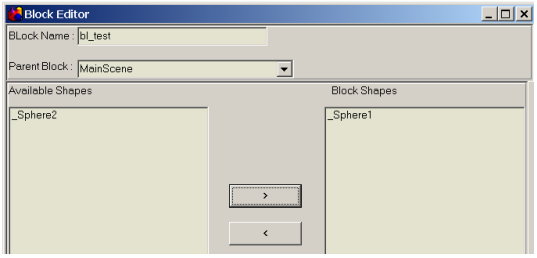
```
_sphere(10,10,0,20,20,20)
```



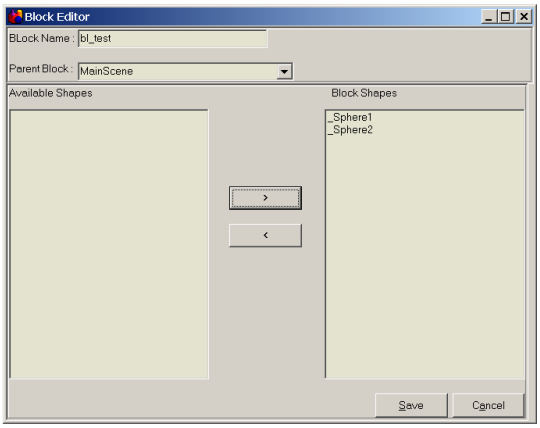
Les blocks vont me permettre de reformer une seule entité avec ces deux sphères. Je pourrais alors les manipuler comme un seul objet.
New : Permet de créer un nouveau bloc à partir d'objets 3D-Crade. Après un click sur New, la fenêtre de block apparaît :



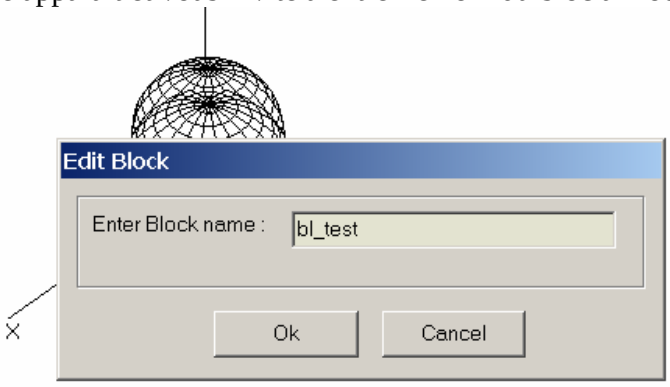
Après avoir entré un nom dans Blockname : par exemple bl_test,
Sélectionnez la première sphère en cliquant sur _Sphere1, ensuite cliquez sur le bouton « > » pour ajouter cet élément au bloc.



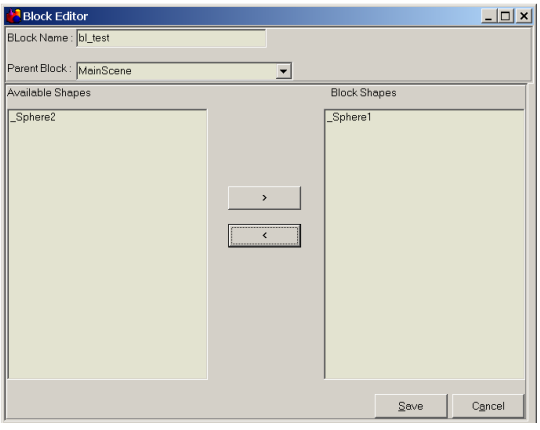
Refaites la même opération avec la seconde sphère, puis cliquez sur le bouton « Save » pour construire le nouveau bloc : bl_test qui sera composé de deux sphères.



La combo box Parent Block : permet d'associer un bloc à un bloc parent. Dans ce cas de figure, le bloc parent sera la scène principale. (3D-Crade ne fait pas de différences entre les blocs et les scènes, même si nous faisons cette différence de manière conceptuelle.).
Nous verrons par la suite dans l'étude de la fenêtre Objects, comment utiliser ces blocs à bon escient.
Modify : il vous est loisible de modifier les blocs crées. Lorsque vous cliquez cette entrée, une fenêtre apparaît et vous invite à entrer le nom du bloc à modifier.



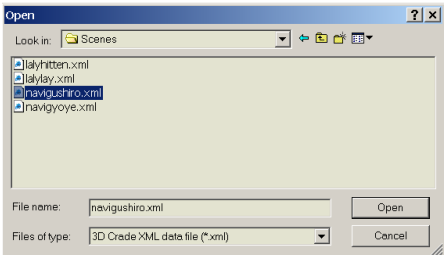
La fenêtre d'édition ressemble très fort à la fenêtre de création.
Vous pouvez à présent sélectionner l'objet _Sphere2 dans la fenêtre de droite Block Shapes, et cliquer sur « < ». L'objet _Sphere2 revient dans la fenêtre de gauche : Available Shapes.



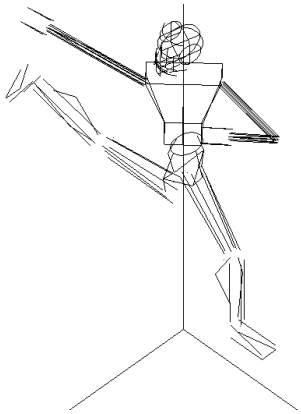
Cliquer ensuite sur Save. La _Sphere2 a retrouvé sa liberté et appartient à nouveau à la scène principale, toutefois le bloc bl_test existe toujours et contient toujours l'objet _Sphere1 :

L'éditeur ne permet pas de détruire des blocs, il vous faut pour cela utiliser la commande Explode() dans l'éditeur de commande. La syntaxe exacte, ainsi que les paramètres de cette commande seront décrits plus loin.

Import : Cette commande permet d'importer un bloc (une scène 3D-Crade, sauvegardée au format XML) provenant d'une autre application compatible avec 3D-Crade. Pour l'instant la seule application compatible avec 3D-Crade est Body-Sculptor.
Un click sur l'item : Import vous propose une fenêtre d'ouverture de fichier, vous permettant de sélectionner le bloc désiré :



Cliquer sur Open pour charger et afficher le bloc. Sachez que lors de l'ouverture des blocs aucun paramètre de scène (voir partie 2 : les arcanes de 3D-Crade) ne sera modifié par le bloc. Le bloc (sous-scène) sera affiché en se basant sur les paramètres courants, ce qui peut parfois être perturbant au niveau des échelles.

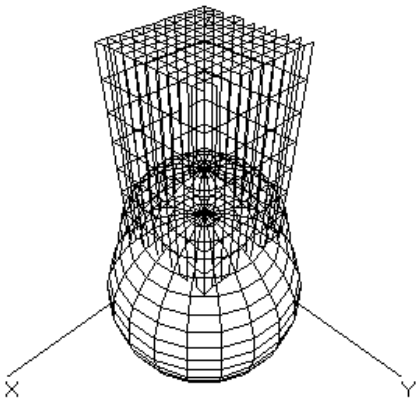


Résultat obtenu après adaptation de l'échelle.
La fonctionnalité Import sera développée plus en profondeur dans la partie consacrée à l'utilisation de Body Sculptor.

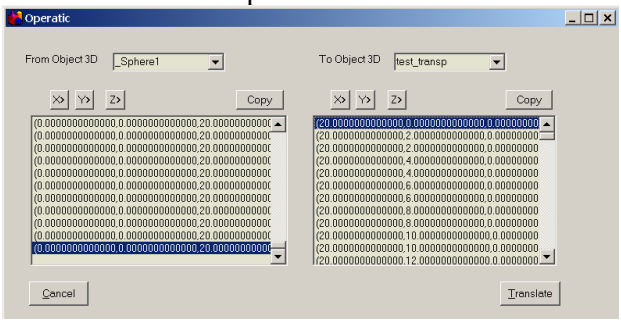
Le menu Operation

Translate : Cette commande permet d'effectuer un translation des coordonnées d'un objet 3D vers les coordonnées d'un autre objet 3D.
La scène ici bas contient un bloc « bl_test », constitué des deux sphères mentionnées précédemment, ainsi qu'un parallélépipède rectangle transparent obtenu au moyen de la

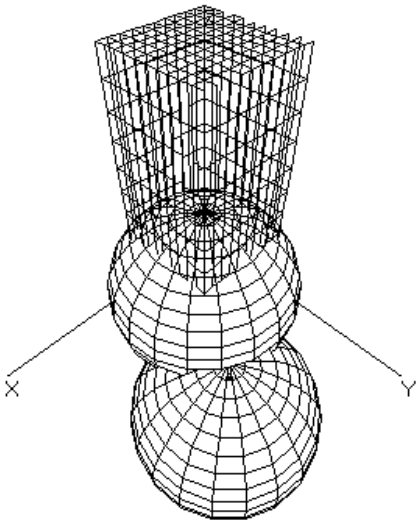
commande :
`_framebox(10,10,25,10,10,25,10,10,0,"_default",1,"test_transp")`
Cette commande sera détaillée dans une section ultérieure.



Un click sur l'item Translate du menu opération nous conduit à la fenêtre suivante :



Cette fenêtre permet de choisir les coordonnées de l'objet 3D-Source (From Object 3D) ou les coordonnées de l'objet 3D cible (To Object 3D).
Les boutons X>, Y> et Z> permettent de trier par ordre croissant (le symbole > signifie croissant) les coordonnées respectives, X, Y ou Z de tous les points des segments (côtés) des polygones constituant l'objet choisi, qu'il s'agisse de l'objet source ou de l'objet cible.
Il existe aussi des boutons Copy, qui permettent de copier les 3 coordonnées choisies dans le clipboard de Windows sous la forme « (X,Y,Z) ».
Une fois les coordonnées cibles et sources sélectionnées, un click sur le bouton translate génèrera la commande de translation, permettant de translater la source, aux coordonnées de la cible choisie.
Les composantes du vecteur de translation seront bien sûr données par (|ciblex-sourcex|,|cibley-sourcey|,|ciblez-sourcez|)
Cette opération peut s'avérer très utile pour calculer automatiquement le vecteur (les valeurs) de translation pour projeter un objet sur un autre.
La sélection montrée dans cet exemple nous donnera la commande suivante :
`translate(20,-6.75439053982486E-33,-20,"_Sphere1")`
Et le résultat suivant :



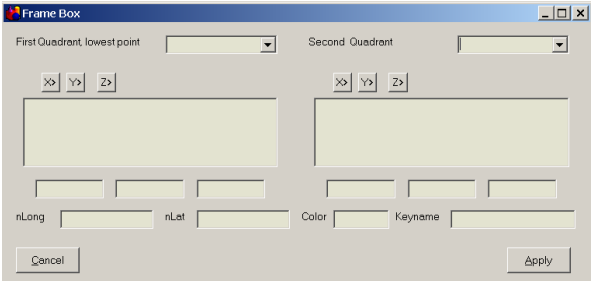
Après validation de la commande.
Remarque 1 : Les détails de cette commande seront donnés dans une section ultérieure.
Remarque 2 : Même si cet outil de calcul de coordonnées de translation ne permet pas de tenir compte des blocs (sous-scènes), la commande de translation, elle, permet de tenir compte de ces sous-scènes.

Le menu Mesh

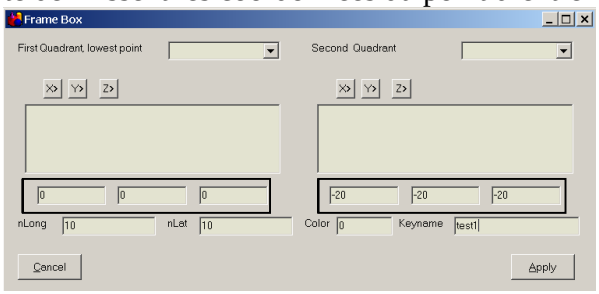
Avertissement : Ce menu est toujours en construction. A l'heure qu'il est seul l'item Framebox est en état de fonctionnement.
Le but de ce menu est d'offrir un certain nombre de fonctionnalités permettant de simplifier l'utilisation des primitives de construction d'objets 3D ou 2D.
L'un des concepts développé est de simplifier la jointure entre objets par d'autres objets.

Framebox : permet de construire un parallélépipède rectangle, en indiquant son point d'origine et son point d'extrémité.
(Il faut remarquer que la commande/primitive **framebox** ne fonctionne pas de la sorte, elle utilise un centre et 3 rayons pour définir le parallélépipède, ceci sera développé dans la partie suivante).

Nous allons découvrir deux méthodes permettant de construire un parallélépipède rectangle : Clickons tout d'abord sur l'item **FrameBox** du menu **Mesh**. La boîte de dialogue suivante apparaît :

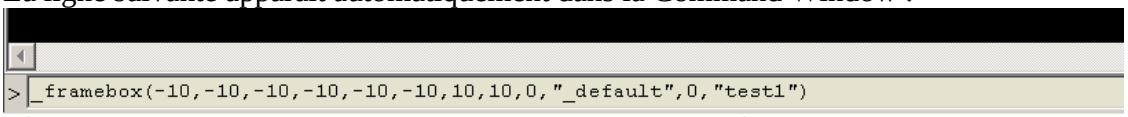


La première méthode consiste à donner des valeurs directes au point d'origine et au point d'extrémité.
Les trois textbox situées à gauche définissent les coordonnées du point d'origine, les trois textbox situées à droite définissent les coordonnées du point d'extrémité.

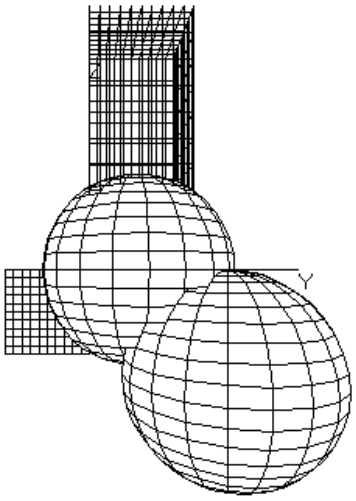


nlong et nLat représente le nombre de polygones que l'on disposera en longueur/largeur et en hauteur.
La notion de couleur sera définie par la suite. Retenez simplement que 0 correspond au noir.
Le keyname permet d'identifier l'objet créer afin de pouvoir le manipuler au moyen de l'application (que ce soit l'incorporer dans un bloc, effectuer une translation, ou encore l'utiliser comme paramètre dans les primitives de 3D-Crade, voir partie suivante).

Une fois toutes les informations encodées, il suffit de clicker le bouton « Apply ».
La ligne suivante apparaît automatiquement dans la Command Window :



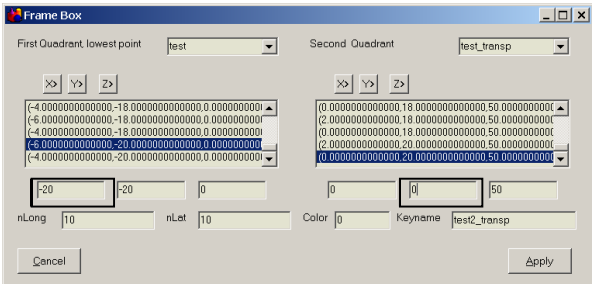
_framebox(-10,-10,-10,-10,-10,-10,10,10,0, «_default »,0, « test »)
Placer le curseur dans la fenêtre command et pressez enter.
Pressez ensuite <F1> pour obtenir ce résultat :



Nous apercevons dès à présent, un carré remplit situé juste derrière, et décalé en bas sur la gauche de la sphère centrale.

La seconde méthode de construction de parallélépipède se base sur des structures déjà existantes. Nous allons créer une boîte prenant appui sur le petit cube et ayant pour hauteur le grand parallélépipède rectangle.

- Clicker Mesh/FrameBox



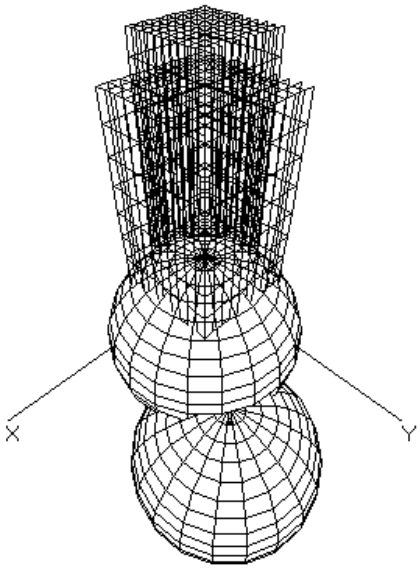
- Pour First Quandrant Lowest point : Choisir « test »
- Pour Second Quandrant Choisir « test_transp »
- Cliquer le Z> de gauche afin de trier par ordre croissant les coordonnées en Z.
- Cliquer ensuite sur la scroll bar de gauche pour atteindre la coordonnée Z la plus élevée (dans ce cas de figure zéro).
- Cliquer sur les coordonnées, comme montré sur l'exemple, les coordonnées minimales sont copiées automatiquement dans les text box du bas.
- Remplacer manuellement la valeur –6 par –20 dans la textbox.
- Cliquer sur le Z> de droite
- Cliquer la scrollbar pour choisir la plus grande valeur de Z.
- Remplacer le –20 par 0 pour la coordonnée Y.
- Entrer les valeurs de nLong et nLat
- La couleur ainsi qu'un nom de forme.
- Cliquer Apply.
- Placer le curseur sur la ligne suivante, quand elle s'inscrit dans la fenêtre Command.

```
_framebox(-10,-10,25,10,10,25,10,10,0,"_default",0,"test2_transp")
```

- Remplacer la valeur 0 en surbrillance par 1 pour activer la transparence

```
_framebox(-10,-10,25,10,10,25,10,10,0,"_default",1,"test2_transp")
```

- Presser enter
- Puis <F4> pour obtenir ceci :



Les personnes ayant déjà travaillé avec des logiciels tels qu'AutoCad, comprendront que l'outil de recherche de coordonnées de segment permet d'émuler (très mal) des commandes telles que Début d'un objet, Fin d'un objet ou encore milieu d'un objet.
De tels outils sont assez difficiles à utiliser à partir d'une interface graphique, c'est pourquoi je n'ai pas beaucoup développé cette partie aux autres formes géométriques.

Le menu Windows
Ce menu n'est plus utilisé dans cette version, il devra être supprimé.

Le menu Help
A terme ce menu devra contenir une aide sur l'utilisation de 3D-Crade.
Le menu contient aussi des informations sur 3D-Crade et notamment sur le site web :
http://users.skynet.be/jyce3d
où 3D-Crade est téléchargeable, ainsi que des objets 3D ou scènes.
L'idée de départ étant de stocker des éléments 3D afin de les échanger gratuitement entre utilisateurs.

La fenêtre Command

Le rôle de la fenêtre de commande est d'exécuter immédiatement des primitives dans 3D-Crade. Les commandes sont exécutées une à une (il est impossible de définir un script), et la nature de ces commandes est diverse. Il peut s'agir de primitives graphiques, d'affectation de valeur pour des variables systèmes, de command de gestion des objets graphiques, …

La fenêtre de commande se décompose en deux parties :
la fenêtre d'output :

Comme son nom l'indique la fenêtre d'output affiche les messages de sorties envoyées par l'exécution des primitives.
En général il s'agit d'une copie de la primitive et de ces paramètres, avec à côté l'apparition d'un statut d'exécution. Qui peut prendre les valeurs DONE, pour indiquer que la commande à été exécutée avec succès ou FAILED, qui indique que l'exécution de la commande n'a pas fonctionné.
Le fait de disposer en sortie de la liste des commandes effectuées avec succès peut s'avérer utile pour le développement de script. La fonctionnalité de copier/coller étant activée dans la fenêtre de sortie. Il vous est loisible de copier/coller les commandes les unes à la suite des autres dans un fichier texte pour former un script.
Une fois sauvé, le script pourra être ré-exécuter par 3D-Crade pour reproduire le dessin voulu. (Voir Partie II section Scripts pour plus d'information).

la fenêtre d'input :

La fenêtre d'input vous permet d'entrer des commandes à exécuter, il suffit de positionner le curseur à l'intérieur de la zone de saisie, d'entrer une commande et de valider en pressant <Enter> pour que la commande s'exécute.

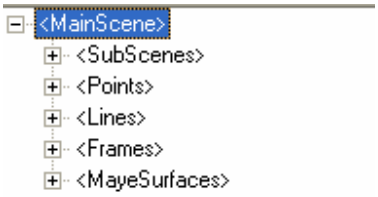
Cette partie de la fenêtre est également utilisée comme sortie par les menus d'opérations ou de Mesh. Le résultat d'une translation ou la création d'un parallélépipède sera affiché à cet endroit. Il suffira à l'utilisateur de cliquer dans la fenêtre d'input pour y amener le curseur et de presser <Enter> pour valider la commande.

La fenêtre Viewer

C'est dans cette fenêtre que la scène contenant tous les objets graphiques crées par les commandes ou les menus sont rendus.
Cette fenêtre peut afficher de manière paramétrable les trois axes représentant le repère orthonormé tridimensionnel. Ces axes peuvent être rendus invisibles.
Le point de vue de l'utilisateur déterminera également la manière dont la scène sera rendue. Ce point de vue est contrôlé par un certain nombre de variables, ou via des raccourcis clavier (cfr La fenêtre d'Objet du paragraphe suivant).
Les variables systèmes utiles seront décrites dans la seconde partie : « Les arcanes de 3D-Crade ».
Cette fenêtre permet également un possibilité d'édition (assez limitée) des objets de la scène en utilisant des raccourcis claviers. Ceci sera détaillé dans la fenêtre d'objet.
3D-Crade ne dispose pas de capacité ray-tracing, si l'on peut utiliser des couleurs différentes pour les faces, il est toutefois impossible de réaliser un éclairement de la scène pour obtenir un rendu.
Au bas de la fenêtre se trouvent les coordonnées en pixels[2] de la position du curseur.

La fenêtre Objects

La fenêtre « objets » permet d'afficher les noms et certaines propriétés des éléments contenus dans la scène principale. Ces éléments sont les points, les lignes, les frames, les surfaces maillées ainsi que certaines sous scènes liées à la scène principale.
Cette scène principale est notée MainScene. Pour des raisons d'économie d'espace mémoire, seuls les sous-scènes dont le parent est la scène principale (MainScene) sont affichées dans la liste.
Chaque type élément est représenté par un nœud qui une fois ouvert donne un certain nombre de propriété sur les objets tridimensionnels. Considérons une MainScene contenant tous les types d'éléments. Après avoir étendu le nœuds MainScene, la fenêtre d'objet devrait se présenter comme suit :



SubScenes : contient la liste des nœuds sous-scènes (bloc) ayant comme parent la scène principale. Pour chacune de ces sous-scènes seul le nom de l'objet sous-scène est disponible, si l'utilisateur étend le nœud sous-scène correspondant.
Points : contient l'ensemble des nœuds points
Pour chaque nœud point, les propriétés suivantes sont disponibles, si le nœud a été étendu :

- *Keyname* : nom de l'objet point

[2] Pixel : contraction de picture element. Le pixel représente la plus petite portion d'écran que l'on peut éclairer. Les dimensions du pixel dépendent de la résolution utilisée (640*480,1024*768,…).

- *Hide* : Propriété qui indique que le point est caché ou visible : (1 caché, 0 visible)
- *Color* : Couleur du point. Le point est le seul objet ayant un comportement particulier, en effet sa couleur n'est pas une couleur prédéfinie dans une table de couleur comme pour les autres surfaces. Mais bien les composantes RGB[3] de la couleur du point à afficher.
- *Type* : Type de trait (non utilisé pour les points)
- *Parent* : Indique la scène ou la sous-scène (bloc) à laquelle appartient le point.
- *x,y,z* : représente les coordonnées du point exprimées en mètre par rapport au centre du repère orthonormé.

Lines : Contient l'ensemble des nœuds line. Ces objets de type Line sont de simple trait, définit par une origine et une extrémité. Ces objets ne sont pas constitués de mailles et ne peuvent être utilisé dans des commandes d'extrusion, de révolution, … Leur utilisation est donc déconseillée en pratique.
Une fois une nœud line étendu il affiche les propriétés suivantes.

- *Keyname* : nom de l'objet ligne.
- *Hide* : Propriété qui indique que la ligne est caché ou visible : (1 caché, 0 visible)
- *Color* : Couleur de la ligne. Cette couleur correspond à un index correspondant à une couleur prédéfinie dans la table des couleurs. (Cfr Partie 2)
- *Type* : Type de trait soit normal ou pointillé. (Crf Partie 2)
- *Parent* : Indique la scène ou la sous-scène (bloc) à laquelle appartient la ligne.
- *x1,y1,z1* : coordonnées du point, exprimées en mètre, d'origine du segment
- *x2,y2,z2* : coordonnées exprimées en mètre, du point d'extrémité du segment.

Frame : Contient l'ensemble des nœuds « Frame ». Les Frame sont définis ici comme des cubes transparents ne disposant pas de mailles.

- *Keyname* : nom de l'objet Frame.
- *Hide* : Propriété qui indique que le Frame est caché ou visible : (1 caché, 0 visible)
- *Color* : Couleur du Frame. Cette couleur correspond à un index correspondant à une couleur prédéfinie dans la table des couleurs. (Cfr Partie 2)
- *Type* : Type de trait soit normal ou pointillé. (Crf Partie 2)
- *Parent* : Indique la scène ou la sous-scène (bloc) à laquelle appartient le Frame.
- *<Points>* : Ensemble des points coordonnées des points formant les arrêtes du cube.

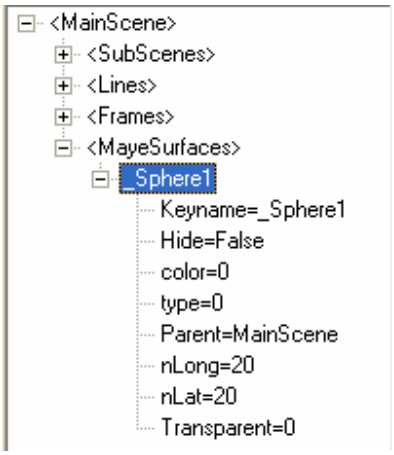
MayeSurfaces : Contient l'ensemble des nœuds MayeSurface. Les MayeSurfaces représentent des surfaces plus ou moins complexes, définies par des polygones. Ces surfaces sont sensibles à la suppression des faces cachées et certaines d'entre elles peuvent être utilisée dans des opérations d'extrusions et de révolutions (Cfr Partie II : Les arcanes de 3D-Crade). Certaines Mayesurfaces peuvent être bidimensionnelles (comme les arcs, les lignes, les ellipses,…) ou tridimensionnelles comme les framebox, les sphères, les ellipses, les arcoid, …
Les propriétés présentées ici, sont celles qui ont été définies au moment de la création de la MayeSurface.

[3] RGB : Composantes fondamentales de Rouge, de Vert et de Bleu d'une couleur.

- *Keyname* : nom de l'objet MayeSurface.
- *Hide* : Propriété qui indique que la MaySurface est cachée ou visible : (1 caché, 0 visible)
- *Color* : Couleur de la MayeSurface. Cette couleur correspond à un index correspondant à une couleur prédéfinie dans la table des couleurs. (Cfr Partie 2)
- *Type* : Type de trait soit normal ou pointillé. (Crf Partie 2)
- *Parent* : Indique la scène ou la sous-scène (bloc) à laquelle appartient la MayeSurface.
- *nLong* : nombre de polygones qui ont été définis dans pour le plan XY
- *nLat :* nombre de polygones qui ont été définis par rapport à l'axe des Z.
- *Transparent* : Indique que tous les polygones sont transparents, seul leur contour apparaîtront.

Liens entre la fenêtre Objects et la fenêtre Viewer

La fenêtre Objects peut être utilisée pour permettre la sélection d'objets 3D. Une fois sélectionné ces éléments pourront être déplacés dans la fenêtre Viewer au moyen de raccourcis claviers appropriés.
Pour sélectionner un objet dans la fenêtre Objects, il faut cliquer sur le nœud qui le représente, ce nœud apparaîtra en surbrillance.



Attention : La sélection des nœuds <Subscenes>,<Lines>,<Frames><MayeSurfaces> n'a aucun effet. Il s'agit juste de titre informatif de regroupement. Les raccourcis claviers ne s'appliquent pas si vous sélectionnez l'un de ces éléments. Par contre vous pouvez sélectionner les nœuds des éléments se trouvant en dessous ou même au dessus (le nœud MainScene) est sélectionnable et est affecté par l'utilisation des raccourcis claviers.

Une fois un nœud Objet 3D sélectionné, cliquez sur la fenêtre Viewer pour lui rendre le « focus » (la remettre en avant plan).

Vous pourrez ensuite utiliser les touches suivantes : quelque soit la vue employée.

Raccourcis claviers concernant les objets sélectionnés :

Touches directionnelles du clavier :

Gauche	Translation de -1 mètre par rapport à l'axe des Y
Droite	Translation de +1 mètre par rapport à l'axe des Y
Haut	Translation de -1 mètre par rapport à l'axe des X
Bas	Translation de +1 mètre par rapport à l'axe des X

Touches standards :

+ ou (=)	Translation de +1 mètre par rapport à l'axe des Z
-	Translation de -1 mètre par rapport à l'axe des Z

Remarque Importante : La valeur initiale de 1 mètre est en réalité stockée dans la variable système _CursorInc. Si cette valeur est changée le pas de translation sera changé pour les prochains déplacements : (Cfr Partie II : Les arcanes de 3D-Crade).

Remarque 2 : Si l'élément sélectionné est une scène contenant un niveau imbriqué de sous-scènes, toutes les sous-scènes seront-elles même affectées par le changement de position.

Racourcis claviers généraux de 3D-Crade

Touche	Action
Gauche	Translation de -1 mètre par rapport à l'axe des Y de toute la scène et ses composants si l'objet <MainScene> est sélectionné.
Droite	Translation de +1 mètre par rapport à l'axe des Y de toute la scène et ses composants si l'objet <MainScene> est sélectionné.
Haut	Translation de -1 mètre par rapport à l'axe des X de toute la scène et ses composants si l'objet <MainScene> est sélectionné.
Bas	Translation de +1 mètre par rapport à l'axe des X de toute la scène et ses composants si l'objet <MainScene> est sélectionné.
+ ou (=)	Translation de +1 mètre par rapport à l'axe des Z de toute la scène et ses composants si l'objet <MainScene> est sélectionné.
-	Translation de -1 mètre par rapport à l'axe des Z de toute la scène et ses composants si l'objet <MainScene> est sélectionné.
CTRL + ou (=)	Rotation de +pi/10 radian du point de vue en azimut si l'objet <MainScene> est sélectionné.
CTRL -	Rotation de –pi/10 radian du point de vue en azimut si l'objet <MainScene> est sélectionné.
CTRL SHIFT + ou (=)	Rotation de +pi/10 radian du point de vue en colatitude si l'objet <MainScene> est sélectionné.
CTRL SHIFT -	Rotation de -pi/10 radian du point de vue en colatitude si l'objet <MainScene> est sélectionné.

Remarque : Le pas de rotation est fixé à pi/10 en réalité il provient de la variable système : _RadianInc, qui si elle est modifiée permettra une rotation du point de vue en accord avec les nouvelles valeurs (Cfr Partie II : Les arcanes de 3D-Crade).

Touche	Action
F1	Vue de gauche
F2	Vue de face
F3	Vue du dessus
F4	Vue isométrique
F9	Ouvre une boîte de dialogue permettant de charger un script 3D-Crade : fichier d'extension .3dc
CTRL O	Ouvrir une scène 3D-Crade au format xml reconnut par 3D-Crade.
CTRL S	Permet de sauvegarder une scène au format xml de 3D-Crade.

Partie II : Les arcanes de 3D-Crade

Cette partie est de loin la plus intéressante mais sans doute la plus ardue, pour les non initiés aux rudiments de programmation informatique.

3D-Crade n'est rien d'autre qu'une interface sur un système de projection conique qui tire sa source de l'application d'une matrice mathématique sur des quadruplets de coordonnées et de quelques formules de trigonométries connexes. Toutefois l'application n'aurait que peu d'intérêt, si elle ne permettait pas à l'utilisateur de contrôler tous ces aspects d'affichage ou de création d'objets 3D.

Tout l'environnement d'affichage est contrôlé par des variables systèmes, qui peuvent être modifiées à loisir par l'utilisateur. L'utilisateur pourra également créer ses propres variables personnelles et même les supprimer, toutefois, il ne pourra en aucun cas effacer des variables systèmes. Cette option est interdite car le programme se planterait immédiatement (cela reviendra à scier la branche sur laquelle il est assis).
Toutes ces variables sont stockées dans une entité informatique que l'on appelle le Contexte de 3D-Crade.
Le contexte peut être modifié ou mis à jour par des commandes bien précises, ces commandes sont analysées par 3D-Crade. Remarquez que l'analyseur syntaxique de 3D-Crade ne gère pas que des commandes de contexte, il peut aussi gérer un certain nombre de primitives liées à la création d'objets tridimensionnels.

3D-Crade comprend donc dans ses arcanes, une matrice de projection 3D vers 2D, un contexte contenant des variables systèmes intimement liées à cette fameuse projection, à l'affichage. Un analyseur d'expression, ainsi qu'un analyseur de scripts (ensemble d'expressions).

Il faut noter que l'unité de base utilisée par 3D-Crade pour définir des objets tridimensionnels est le mètre [m], tel qu'il est défini dans le système MKSA[4].
L'analyseur de syntaxe utilise toujours le '.' comme séparateur décimal, quelque soit les regional settings définis dans Windows.
Si par exemple vous voulez définir une distance de 8 centimètres dans une primitive de 3D-Crade, vous utiliserez la notation suivante : « 0.08 », puisque par défaut 3D-Crade référence tout par rapport au mètre. Notez que vous pourrez utiliser sans mal la notation « 8/100 » toutefois ceci reste quelque peu compliqué, puisque le système devra effectuer une opération de division (généralement lente) pour obtenir un résultat identique.
Il est important de conserver des échelles réalistes lorsque vous réalisez des scènes ou des objets tridimensionnels afin que ceux-ci puissent être réutilisés dans n'importe quel autre cas d'utilisation.

[4] MKSA : Système international d'unités de physique : mètre [m], kilo [k], seconde [s], Ampère [A].

Toutes les mesures se rapportant à des angles sont exprimées en radian, qui somme tout est la manière la plus logique d'exprimer l'ouverture d'un angle. (Du moins c'est mon point de vue personnel)

Dans ce premier chapitre nous allons voir comment manipuler le contexte, dans le second comment utiliser les primitives de l'analyseur syntaxique pour réaliser nos premières scènes, dans le troisième chapitre le lecteur apprendra à rédiger ses premiers scripts, ou ces premiers programmes informatiques servant à dessiner de la 3D. Enfin le quatrième chapitre nous livrera tous les secrets de la sauvegarde des scènes 3D-Crade. Equipé de ce format XML, un programmeur confirmé pourra générer avec n'importe quel programme des scènes lisibles par 3D-Crade, en générant les éléments XML à partir d'une source externe par exemple.

Chapitre I Le Contexte

Les variables

Une variable sera définie dans notre cas d'utilisation, comme un « être », une entité informatique qui peut contenir une valeur pouvant changer au cours de l'utilisation et par ce fait, au cours du temps. La variable est identifiée par un nom, appelé aussi identificateur. Une variable peut contenir différentes sortes de données, c'est pourquoi en général on lui attribue un type.
Le contexte de 3D-Crade ne supporte à l'heure actuelle que les données de type réel. Toutefois il supporte aussi un de manière limitée un type texte, ce type est très utile pour nommer les objets 3D que nous créons dans 3D-Crade. Chaque valeur de ce type, doit être entourée de guillemets, pour faire comprendre au contexte qu'il s'agit d'une valeur texte. Les variables de type texte ne sont pas du tout bien supportées par 3D-Crade.

La liste des variables :
Avant de travailler avec des variables il est intéressant de savoir à tout moment quelles sont les variables mises à notre disposition. La liste des variables peut s'obtenir au moyen de la commande :
`list_var()`[5]
Il suffit d'entrer cette commande dans la section « input » de la fenêtre de commande comme suit et de valider par <ENTER> :

La liste des variables apparaît dans la fenêtre d'output. L'ascenseur est disponible pour naviguer à travers toutes l es variables.

[5] Remarquez que dans 3D-Crade toutes les commandes ou primitives sont considérées comme des fonctions ne renvoyant rien. D'où la présence obligatoire des deux paranthèses contenant ou ne contenant pas de paramètres.

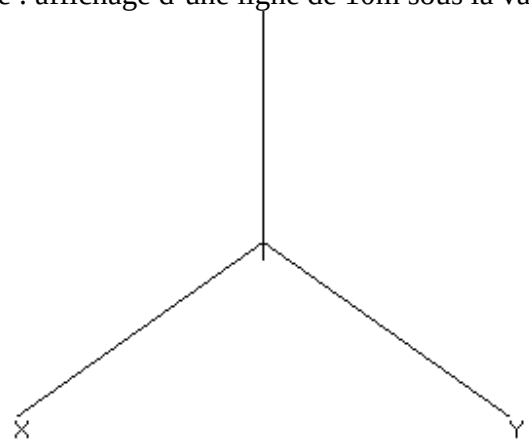
Les variables sont affichées dans la zone d'« output » en respectant les critères suivants :

Attributs	Définition
Nom de la variable	Identificateur qui permet au Contexte et aux utilisateurs de retrouver la variable. La plupart des caractères sont autorisés : [Aa-Zz], [0-9], [_ !,@,$,%] Eviter toutefois d'utiliser des caractères symbolique tels que les espace, guillemets, quotes, ou encore des caractères opérateurs tels que +,-,/,*,(,),#. En général les variables précédées du caractères '_' indique qu'il s'agit de variables systèmes. Une exception est toutefois faites pour la variable pi (3.14159265358…). Vous pouvez toutefois créer vos propres variables en les faisant préfixer de « _ » mais vous risquez d'apporter plus de confusion que de visibilité.
Type	FLOAT : indique que la valeur peut contenir des valeurs numériques, entières ou réelles.
Valeur	Valeur courante de la variable
Status	Information complémentaire sur la nature de la variable : *protected* indique que la variable est une variable système et qu'elle ne peut en aucun cas être retirée sans démolir 3D-Crade.

Affecter une valeur à un variable :
Pour changer la valeur d'une variable vous pouvez utiliser la commande :

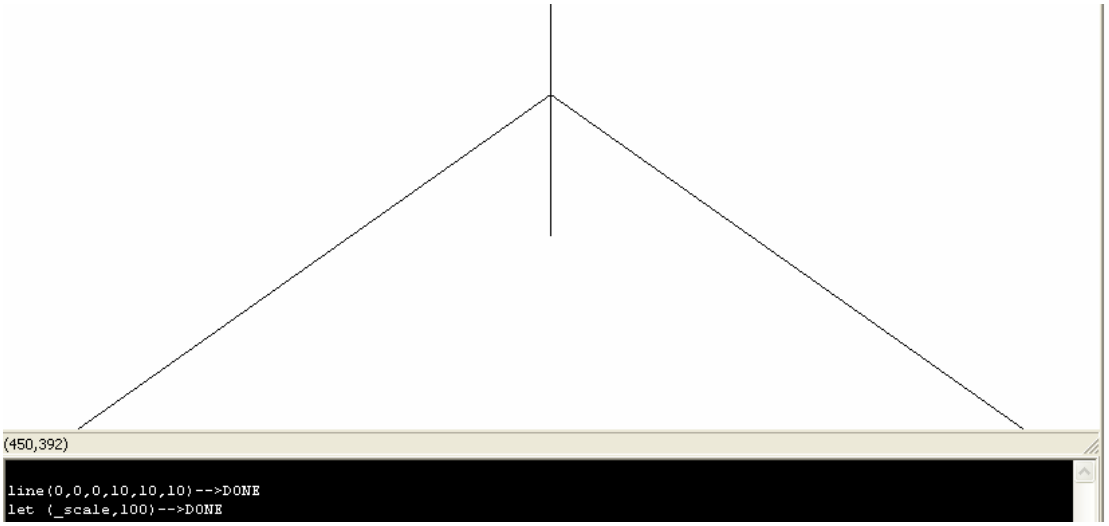
`let (<nom_de_variable>,<valeur>)`

Cette fonction prend deux arguments :
Le premier étant le nom d'une variable de la liste contenue dans le contexte, le second argument représente la nouvelle valeur à affecter à cette variable.
Exemple : affichage d'une ligne de 10m sous la valeur d'échelle courante.



Entrer la commande suivante :

```
let (_scale,100)
```



Le tracé d'une ligne de 10 mètres après agrandissement de l'échelle par modification de la variable système de réglage d'échelle : _scale.

Ajouter une nouvelle variable :
Pour une utilisation personnelle il est parfois intéressant d'avoir sous la main une variable contenant une valeur remarquable. Il est loisible à l'utilisateur de créer ses propres variables et de les affecter d'une valeur :
Pour créer une nouvelle variable : entrer la commande suivante dans la fenêtre de Commandes :

`set (<nom_de_variable>,FLOAT)`

Par exemple :

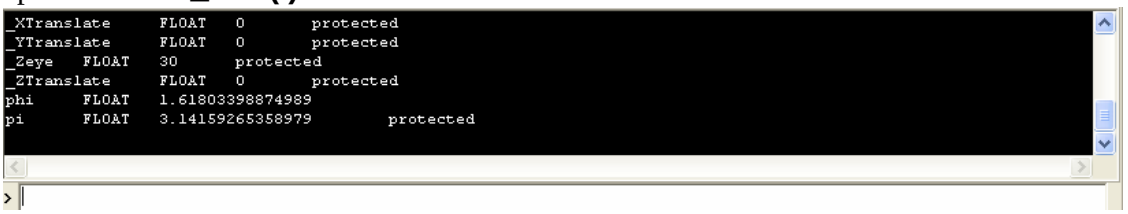
`set (phi,float)`

Après avoir effectué un **`list_var()`**, nous obtenons ceci :

La variable `phi` est définie mais ne contient aucune valeur, de plus elle n'est pas affublée du status *protected*.
Reste à donner une valeur à `phi`, par exemple la valeur approchée du nombre d'or :

`let (phi,(1+5^0.5)/2)`

après un **`list_var()`**



La variable `phi` contient à présent la valeur approchée du nombre d'or[6].
Remarque : Vous ne pouvez pas créer une variable portant le nom d'une variable déjà existante.

Supprimer une variable ajoutée

Si les rectangles et les spirales d'or ne vous intéressent pas, vous pouvez vous débarrasser de cette variable inutile en utilisant la commande :
`kill(phi)`
Comme son nom l'indique cette commande permet de tuer, une, plusieurs ou toutes les variables non systèmes contenues dans le contexte.
`kill()`
Détruira toutes les variables crées par l'utilisateur, sans distinction et sans confirmation. Soyez donc prudent quand vous utilisez cette commande. Les variables systèmes (*protected*) ne seront pas inquiétées par la commande.

Expressions analytiques et opérateurs mathématiques du Contexte :

Il est possible d'affecter une valeur directe à une variable mais aussi le résultat d'une expression analytique, ou plus communément le résultat d'une fonction.

Les opérateurs du Contexte et leur ordre de priorité :
L'ordre indique la priorité d'interprétation de l'opérateur, plus le numéro d'ordre est petit plus la priorité de l'opérateur est grande.

Ordre	Symbole	Description
1	()	Permette de regrouper une expression analytique et de la traiter en priorité.
2	^	Mise en exposant (exemple : `let (a,2^10)` affectera la valeur 1024 à la variable a)
3	*,/	* est l'opérateur de multiplication : `let (a,2*5)` / est l'opérateur de division : `let (a,2/5)`
4	+,-	+ opérateur d'addition entre deux réels (float) ou opérateur de concaténation entre deux textes. `let (a,a+1)` «`name `»+ « `_1` » donnera pour résultat « `name_1` » - opérateur de soustraction entre deux réels.
5	<,>, <=,	Opérateurs de comparaisons :

[6] Le nombre d'or (Phi) est considéré comme le rapport de la divine proportion : on le retrouve dans les proportions des corps humains bien faits, ainsi que dans bon nombre d'éléments naturels (coquille d'escargots, forme des galaxies,…). Beaucoup d'artiste s'en sont d'ailleurs allègrement inspirés pour composer leur tableau, les bâtisseurs de cathédrale s'en seraient aussi inspirés.
Phi s'obtient de différentes manière la plus simple consiste à le définir comme : phi=(1+sqrt(5))/2, en fait il s'agit de la solution positives de l'équation x^2-x-1=0.
Une autre méthode plus précise définis phi comme étant le rapport de U_n/U_{n-1}, pour n tendant vers l'infini. U_n étant la suite de Fibonacci définie par : $U_{n+1}=U_n+U_{n-1}$

	>=, !=, ==	< : strictement inférieur > : strictement supérieur <= : inférieur ou égal >= : supérieur ou égal != : différent == : égalité stricte
6	!	Opérateur NON (inverse le sens des opérateurs de comparaisons)
7	#	Opérateur de concaténation entre un texte et un réel « Nom »#1 donnera pour résultat : « Nom1 ».

Les fonctions internes supportées par le Contexte :

sin(x)	sinus d'un angle.
cos(x)	cosinus d'un angle
tan(x)	tangente d'un angle
asin(x)	arc sinus d'une grandeur
acos(x)	arc cosinus d'une grandeur
atan(x)	arc tengente d'une grandeur
exp(x)	exponentielle d'une grandeur
ln(x)	logrithme népérien d'une grandeur
abs(x)	valeur absolue d'une grandeur
neg(x)	oppose d'une grandeur

Les variables du système

La manipulation des variables du système demande une certaine prudence, car mal affectées, elles peuvent conduire à de sérieux problèmes d'affichage. L'une des particularités de 3D-Crade est de donner un maximum de contrôle sur le rendu du modeleur, pour donner un maximum de liberté à l'utilisateur. L'effet pervers est qu'il permet aussi de détraquer très facilement ce rendu.

Variables d'affichage de la scène 3D dans la fenêtre Viewer :
Ces variables gèrent essentiellement la manière dont la projection est effectuée :_Zeye. Mais aussi le point de vue qui intervient aussi dans cette projection. Le point de vue est défini par trois coordonnées polaires : l'angle d'azimuth, l'angle de colatitude, et la distance du point de vue.

Nom	Description
_Azimutheye	Angle d'azimuth (plan XY), exprimé en radian du point de vue.
_Colatiteye	Angle de colatitude (plan Z->XY), exprimé en radian du point de vue.
_Rhoeye	Distance du point de vue en mètre du centre (0,0,0) du repère.
_Scale	Facteur de grossissement.
_Zeye	Position en mètre du plan de projection par rapport à l'œil.

[Insérer le schéma de projection quelconque]

Variables de gestion du rendu de la scène après projection.

Nom	Description
_Polygonframes	Cette variable permet d'afficher ou non les contours des polygones des surfaces maillées. Exemple : let (_Polygonframes,1) Et let (_Polygonframes,0) [Insérer un exemple pour 0 et un exemple pour 1]
_XTranslate	Translation de la scène en X
_YTranslate	Translation de la scène en Y
_ZTranslate	Translation de la scène en Z, ces variables permettent à l'utilisateur de naviguer dans la scène.
_ViewAxes	Permet d'afficher les trois axes du repère : let (_Viewaxes,1) Permet d'enlever les trois axes du repère : let (_Viewaxes,0)

Les variables systèmes d'éditions
Combinées aux raccourcis claviers, ces variables permettent de définir l'espace de déplacement ainsi que le pas de rotation en radian (lorsque l'on fait pivoter la scène).
Revoir le tableau contenant la définition des raccourcis liés aux touches : Gauche, droite, haut, bas, CTRL (+|=), CTRL-, CTRL SHIT +|= et CTRL SHIFT -.

Nom	Description
_CursorInc	Permet de fixer le pas de déplacement lorsque l'on utilise les raccourcis clavier pour déplacer des objets. La valeur par défaut est 1 (mètre).
_RadianInc	Permet de fixer le pas de rotation lorsque l'on utilise les raccourcis clavier pour faire pivoter le point de vue. La valeur par défaut est pi/10.

Les variables systèmes inutilisées

Nom	Description
_RulerDiv	Non implémentée : fixer le nombre de division de la règle
_ViewRuler	Affichage de la règle (inutilisé).

Divers

Nom	Description
Pi	Constante mathématique : 3.14159265358…

Chapitre II L'analyseur d'expressions (Parser)

L'analyseur d'expressions est la partie active (voire centrale) de 3D-Crade, il complète et étend fortement les opérations et les fonctionnalités présentes dans les menus de 3D-Crade. (La plupart des actions des menus s'appuient presque entièrement sur l'analyseur d'expression).
Pour résumer simplement l'analyseur d'expression permet de manipuler les variables du contexte, en donnant la possibilité de réaliser des assignations, additions, soustractions, multiplications, … Il permet aussi d'appliquer des fonctions mathématiques plus ou moins évoluées. L'analyseur d'expression est aussi capable d'interpréter et d'exécuter les primitives graphiques.

Les commandes de bases

Les commandes du contexte :

La plupart des commandes du contexte ont déjà été vue lors de la présentation du contexte et des variables, nous en donneront ici une liste formelle.

Nom	Description
Set	Permet de créer une nouvelle variable utilisateurs dans le contexte: Entrée : - Identificateur de la variable : il s'agit d'un texte pouvant contenir les caractères suivants : [Aa-Zz], [0-9], [_ !,@,$,%] - le type : pour l'instant seul le type *float* est supporté, il s'agit du type réel. Exemple : set (ma_variable,float)
Let	Permet de d'affecter une valeur ou une expression algébrique à une variable utilisateur ou à une variable système. Une valeur est généralement une valeur réelle, une expression algébrique est composée de valeurs, ou de fonctions dépendantes de valeurs ou d'autres variables. L'expression algébrique est liée au moyen d'opérateur et regroupée au moyen de parenthèses. Entrée : - Identificateur de la variable que l'on veut affecter - Valeur ou expression algébrique Exemple : let(ma_variable,4*atn(1)+(pi/2)) Atn(1) est la fonction arctangente Pi est une variable système.
List_var()	Affiche la liste des variables systèmes ou utilisateurs définies au sein du contexte : Les informations suivantes sont affichées :
Kill()	Détruit toutes les variables utilisateurs (ne disposant pas du status protected). Il est possible de détruire, sélectivement, une ou plusieurs variables spécifiques, il suffit de placer entre les parenthèses et

	séparés par des virgules le nom d'identificateur des variables que l'on veut supprimmer. Ex : kill(phi,rho,teta) NB:Soyez prudent avec l'utilisation de kill(). Cela peut conduire à de drôles de surprises.

Les primitives graphiques

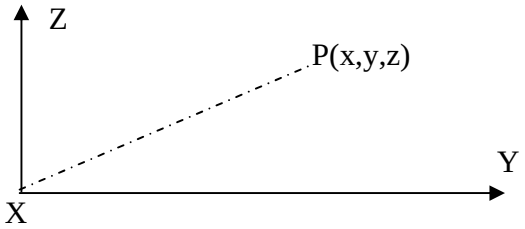
Avant de nous lancer dans la description des primitives graphiques de 3D-Crade, il est bon de donner quelques informations sur la manière dont sont codées les couleurs, les coordonnées et les types de traits.

Le codage des couleurs :
Tout d'abord 3D-Crade n'utilise pas du bitmap réel en 24bits. Ce choix n'est pas très judicieux, toutefois il s'accordait avec la philosophie de l'application qui consistait à créer de la 3D basique, de plus, comme 3D-Crade ne gère pas les éclairages (et encore moins les ombrages), il semblait peut intéressant de fournir une palette énorme de couleurs puisque celle-ci n'aurait pas été représentée comme il le fallait. 3D-Crade dispose donc d'une table de conversion de 16 entrées entières, pour chacune de ces entrées entières correspond une couleur de base.

Codage	Couleur
0	Blanc
1	Rouge
2	Vert
3	Bleu
4	Jaune
5	Pourpre
6	Vert foncé
7	Cyan
8	Noir
9	Gris très foncé
10	Gris
11	Gris moins foncé
12	Gris clair
13	Gris encore plus clair
14	Gris léger
15	Gris presque blanc.

Codage des coordonnées
Les coordonnées s'encodent toujours en mètre [m]. De plus, le repère orthonormé est un repaire dit main droite (l'axe de Z positif est dirigé vers le haut, l'axe des X positif est dirigé vers l'utilisateur, l'axe des Y positifs est dirigé vers la droite).

[7]



La coordonnée Z représente donc ici la hauteur et la coordonnée X la profondeur.

Le type de traits

_default : permet de construire les tri-face au moyen de lignes pleines

_thin : permet de construire les tri-face au moyen de lignes pointillées

Les noms d'objet (Keyname)
Tous les objets 3D sont référencés ou reconnus par 3D-Crade au moyen d'un nom qui leur est propre, ce nom est appelé communément le « Keyname ». Le nom étant une valeur alphanumérique de type texte doit toujours se trouver entourrés de guillemets :
Exemple :
`Translate(1,1,1, « tabourret »)`
Commandes diverses sur les objets 3D

Nom	Description
Block(Keyname)	Crée un nouveau bloc (sous-scène) portant le nom Keyname. Pour indiquer la fin du bloc on utilise la commande Blend.
Copy(KeynameSource, KeynameTarget)	Copie un objet (Source) en un objet cible (Target). L'objet cible est créé (en fonction du type de l'objet source) et assigné avec des valeurs identiques à celles de l'objet source. Exemple : **`Copy("Sphere1","Sphere2").`**
Explode(Keyname)	Permet d'éclater un bloc (sous-scène) existant et référencé au moyen de Keyname. La scène parente à tous les objets du bloc éclaté redevient la scène principale (MainScene).
List_3D()	Permet d'obtenir la liste des objets 3D contenus dans la scène principale.
Remove(Keyname)	Permet de supprimer un objet 3D de la scène, il suffit d'indiquer son nom. On ne peut supprimer qu'objet par objet.

[7] Ndrl : Si vous travaillez un jour en DirectX 3D, eux utilisent un système de coordonnée main gauche. Ce qui signifie que l'axe des Y représente la hauteur et l'axe des Z la profondeur. Toutefois la convention mathématique habituelle utilise le système main droite, et comme 3D-Crade a été construit sur base des cours de géométrie dans l'espace, il utilise donc le système main droite, plus classique.

Les commandes et primitives[8] d'opérations

Nom	Description
_ExtrudeX(SourceKeyname, length, nLong, CoordType, TargetKeyName)	Extrude une mayesurface simple (ellipse ou arc) de "length" mètre, en suivant l'axe des X. SourceKeyname : Nom du bloc à extruder. length:longueur en mètre d'extrusion nLong:Nombre de points (nLong-1) polygones qui définiront l'extrusion. CoordType:(0-relatif|1-absolu) TargetKeyname:Nom de la nouvelle mayesurface après extrusion
_ExtrudeY(SourceKeyname, length, nLong, CoordType, TargetKeyName)	Extrude une mayesurface simple (ellipse ou arc) de "length" mètre, en suivant l'axe des Y. SourceKeyname : Nom du bloc à extruder. length:longueur en mètre d'extrusion nLong:Nombre de points (nLong-1) polygones qui définiront l'extrusion. CoordType:(0-relatif|1-absolu) TargetKeyname:Nom de la nouvelle mayesurface après extrusion
_ExtrudeZ(SourceKeyname, length, nLong, CoordType, TargetKeyName)	Extrude une mayesurface simple (ellipse ou arc) de "length" mètre, en suivant l'axe des Z. SourceKeyname : Nom du bloc à extruder. length:longueur en mètre d'extrusion nLong:Nombre de points (nLong-1) polygones qui définiront l'extrusion. CoordType:(0-relatif|1-absolu) TargetKeyname:Nom de la nouvelle mayesurface après extrusion
_RevolutionX(SourceKeyname, phi0, phi1, nLong, TargetKeyName)	Construit une mayesurface complexe à partir d'une mayesurface simple (arc, ellipse) en effectuant une révolution autours de l'axe des X SourceKeyname : Nom du bloc à révolutionner. Phi0:angle d'origine en radian Ph1:angle d'extrémité en radian nLong:Nombre de points (nLong-1) polygones qui définiront la révolution . TargetKeyname:Nom de la nouvelle mayesurface après révolution
_RevolutionY(SourceKeyname, phi0, phi1, nLong,	Construit une mayesurface complexe à partir d'une mayesurface simple (arc, ellipse) en effectuant une

[8] On entend par primitive, ce qui crée un objet graphique de type mayesurface, en général les primitives se reconnaissent par le fait qu'elle commence souvent par le caractère « _ », il existe toutefois une exception : la primitive Copy, qui elle n'a pas héritée de l'underscore pour rester plus proche de la commande copy du MS-DOS, qui fait la même chose sur des fichiers.

TargetKeyName)	révolution autours de l'axe des Y SourceKeyname : Nom du bloc à révolutionner. Phi0:angle d'origine en radian Ph1:angle d'extrémité en radian nLong:Nombre de points (nLong-1) polygones qui définiront la révolution . TargetKeyname:Nom de la nouvelle mayesurface après révolution
_RevolutionZ(SourceKeyname, phi0, phi1, nLong, TargetKeyName)	Construit une mayesurface complexe à partir d'une mayesurface simple (arc, ellipse) en effectuant une révolution autours de l'axe des Z SourceKeyname : Nom du bloc à révolutionner. Phi0:angle d'origine en radian Ph1:angle d'extrémité en radian nLong:Nombre de points (nLong-1) polygones qui définiront la révolution . TargetKeyname:Nom de la nouvelle mayesurface après révolution
RotateX(phi, Keyname)	Fait effectuer une rotation de phi radian autours de l'axe des X à l'objet désigné par Keyname. Exemple : **`RotateX(pi/3,"MyObject3D")`**
RotateY(phi, Keyname)	Fait effectuer une rotation de phi radian autours de l'axe des Y à l'objet désigné par Keyname. Exemple : **`RotateY(pi/3,"MyObject3D")`**
RotateZ(phi, Keyname)	Fait effectuer une rotation de phi radian autours de l'axe des Z à l'objet désigné par Keyname. Exemple : **`RotateZ(pi/3,"MyObject3D")`**
Translate(tx, ty, tz, Keyname)	Fait effectuer une translation de tx,ty,tz aux coordonnées (x,y,z) d'un objet désigné par Keyname. Exemple : **`Translate(0.5,0.3,0.2,"MyObject3D")`**

Les commandes graphiques

Convention : les paramètres placés entre crochets sont optionnels. Ces commandes graphiques ne produisent pas de surfaces ou d'objet 3D contenant des polygones, il est donc impossible d'utiliser des opérations d'extrusion et/ou de révolution sur ceux-ci.

Nom	Description
Frame(x, y, z, width, length, height[, color, type, Keyname])	Affiche un parallélépipède rectangle transparent. • Le triplet (x,y,z) localise le centre de la forme ; • width est la demi longueur en x; • length représente la demi longueur en y; • height la demi longueur en z. Toutes ces grandeurs sont définies en mètres. • Color : représente la couleur • Keyname=nom de l'objet (facultatif)

Line (x1, y1 ,z1 ,x2 ,y2 ,z2 [,color, type, Keyname])	Affiche une droite dans l'espace de la scène • (x1,y1,z1)et (x2,y2,z2) représentent les coordonnées des deux points formant la droite. • Keyname=nom de l'objet (facultatif)
Point (x, y, z [, color, Keyname])	Affiche un point dans l'espace de la scène. • Le triplet (x,y,z) représente les coordonnées du point en mètres. • Keyname=nom de l'objet (facultatif)

Les primitives graphiques

Nom	Description
_Arc(x, y, z, a, b, nLong, phi0, phi1[, color, type, Transparent, keyname])	Affiche un arc d'ellipse dans l'espace de la scène. • Le triplet (x, y, z) représente l'arc d'ellipse. • Le doublet (a,b) représente les rayons respectifs de l'ellipse ; • nLong indique le nombre de pseudo-polygones qui composeront l'arc d'ellipse; • phi0 indique l'angle de départ exprimé en radian ; • phi1 indique l'angle de fin exprimé en radian. Ces deux valeurs représentent l'ouverture de l'angle définissant l'arc en longitude ; • Transparent indique si l'objet apparait en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatifs)
_Arcoid(x, y, z, rx, ry, rz, phi0, phi1, teta0, teta1[, color, type, Transparent, keyname])	Affiche un arc d'ellipsoïde maillé en polygones dans l'espace de la scène. • Le triplet (x,y,z) localise le centre de l'ellipsoïde complet. • Les valeurs a,b,c représentent les rayons, en x,y et z de l'ellipsoïde; • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude (z); • phi0 indique l'angle de longitude de départ exprimé en radians ; • phi1 indique l'angle de longitude de fin exprimé en radians. Ces deux valeurs représentent l'ouverture de l'angle définissant l'arc en longitude ; • teta0 représente l'angle de latitude de départ exprimé en radians; • teta1 l'angle de latitude de départ exprimé en radians. Ces deux valeurs définissent l'ouverture de l'angle de latitude qui permet de définir l'arc d'ellipsoide. • Transparent indique si l'objet apparait en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatif)
_Cylinder(x, y, z, a, b, height,	Affiche un cylindre dans l'espace de la scène. • Le triplet (x,y,z) représente le centre de l'ellipse de la base du

nLong, nLat[, color, type, Transparent, Keyname])	cylindre. • Les valeurs a et b représentent ses rayons respectifs. • Height représente la hauteur du cylindre (en mètres). • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude(z); • Transparent indique si l'objet apparait en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatif)
_Ellipse(x, y , z, a, b, nLong[, color, type, Transparent, Keyname])	Affiche une ellipse dans l'espace de la scène. • Le triplet (x,y,z) représente le centre de l'ellipse ; • Le doublet (a,b) représente les rayons respectifs de l'ellipse ; • nLong indique le nombre de pseudo-polygones qui composeront l'ellipse; • Transparent indique si l'objet apparait en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatifs)
_Ellipsoid(x, y, z, rx, ry, rz, nLong, nLat, phi0, phi1, teta0, teta1[, color, type, Transparent, Keyname])	Affiche un ellipsoïde maillé en polygones dans l'espace de la scène. • Le triplet (x,y,z) localise le centre de l'ellipsoïde. • Les valeurs a,b,c représentent les rayons, en x,y et z de l'ellipsoïde; • phi0 est le point d'ouverture de l'angle en azimuth, • ph1 est le point de fermeture de l'angle en azimuth (sens positif : X vers le point); • teta0 est le point d'ouverture de l'angle en latitude ; • teta1 est le point de fermeture de l'angle en latitude (sens positif de Z vers le point d'ouverture); • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude (z); • Transparent indique si l'objet apparaît en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatif)
_Framebox(x, y, z, width, length, height, nLong, nLat[, color, Type, Transparent, Keyname])	Affiche un parallélépipède rectangle maillé en polygones. • Le triplet (x,y,z) localise le centre de la forme: • width est la demi longueur en x; • length représente la demi longeur en y; • height la demi longueur en z. Toutes ces grandeurs sont définies en mètres. • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude (z); • Transparent indique si l'objet apparait en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatif)
_Line(x1, y1, z1, x2, y2, z2,	Affiche une ligne maillée en polygons dans l'espace de la scène. • Le triplet (x1,y1,z1) localise le point de départ de la ligne

nLong[, color, type, keyname])	• Le triplet (x2,y2,z2) localise le point final de la ligne • nLong représente le nombre de polygones qui composeront la ligne. • Keyname=nom de l'objet (facultatif) Rmk : Cette fonction est très utile pour les extrusions ou les révolutions.
_Sphere(x, y, z, radius, nLong, nLat[, color, type, Transparent, Keyname])	Affiche une sphère maillée en polygones dans l'espace de la scène. • Le triplet (x,y,z) localise le centre de la sphère. • Radius représente le rayon de la sphère; • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude(z); • Transparent indique si l'objet apparait en transparence, ou solide [0-Solide,1-Transparent]; • Keyname=nom de l'objet (facultatif)

Primitives Architecturales

Ces primitives réalisent des constructions de murs, de cloisons et de coins, toutefois elles sont extrêmement gourmandes en ressource mémoire et ressource processeur, il n'est donc pas conseillé de les utiliser trop fréquemment, sauf si vous possédez une très bonne machine.

Nom	Description
_Wall(length, height, nLong, nLat, color, Keyname)	Affiche un mur de construction, comprenant la brique, l'isolant le bloc et le plafonnage. • Length : Longueur du mur • Height : Hauteur du mur • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude(z); • color représente la couleur • Keyname le nom de l'objet.
_WallCorner(height, nLat, color, Keyname)	Permet d'afficher un coin de mur de construction • Height : représente la hauteur du coin de mur • nLat représente le nombre de polygones en latitude(z); • color représente la couleur • Keyname le nom de l'objet.
_WallFence(length, height, nLong, nLat, color, Keyname)	Affiche une cloison en brique avec plafonnage. • Length : Longueur du mur • Height : Hauteur du mur • nLong représente le nombre de polygones en longitude (x,y); • nLat représente le nombre de polygones en latitude(z); • color représente la couleur • Keyname le nom de l'objet.

Les fonctions du contexte

Nom	Description	Retour
Let (y,sin(x))	Sinus de l'angle x en radian (FLOAT)	FLOAT
Let (y,cos(x))	Cosinus de l'angle x en radian (FLOAT)	FLOAT
Let (y,tan(x))	Tangente de l'angle x en radian (FLOAT)	FLOAT
Let (y,asin(x))	Arcsinus de x (FLOAT)	FLOAT
Let (y,acos(x))	Arccosinus de x (FLOAT)	FLOAT
Let (y,atan(x))	ArcTangente de x (FLOAT)	FLOAT
Let (y,exp(x))	Exponentielle (base e) de x (FLOAT)	FLOAT
Let (y,ln(x))	Logarithme népérien de x (FLOAT) avec x>0	FLOAT
Let (y,abs(x))	Valeur absolue de x (FLOAT)	FLOAT
Let (y,neg(x))	Opposé de x (FLOAT)	FLOAT

Chapitre III Les scripts

"Un langage de programmation est censé être une façon conventionnelle de donner des ordres à un ordinateur. Il n'est pas censé être obscur, bizarre et plein de pièges subtils, ça se sont les attributs de la magie."

Dave Small

L'idée fondamentale qui m'a motivée à concevoir ce système de script fut avant tout un retour aux sources de la programmation. En effet, dans les années 80, la limitation des possibilités offertes par les machines permirent de véritable succès pour des langages tels que le BASIC (Beginners All purposes Symbolic Instruction Code : Code d'instruction symbolique et générique pour débutants). Tout possesseur de micro-ordinateur à cette époque se devait de maîtriser un minimum ce langage BASIC, s'il voulait faire la moindre chose avec sa machine. Que ce soit lancer un jeu, écrire du texte, …
Bien que le BASIC ait été décrié à juste titre pour sa propension à la création de code spaghetti, il faut lui reconnaître qu'il à rendu la programmation accessible à tous. En effet, un enfant de 8 ou 10 ans pouvait déjà commencer à programmer en BASIC et en comprendre les tenants et aboutissants. En témoigne des revues telles que Hebdogiciel ou anciennement MicroMsx qui publiaient des dizaines de pages de listing en BASIC à recopier.
Aujourd'hui dans ce monde informatique fait de classes, d'objets, d'instances, de polymorphisme, de composants COM ou CORBA, d'assemblies en code managés aux méthodes comprenant des paramètres incompréhensibles au commun des mortels, il est très difficile pour un néophyte de retrouver son chemin et ce même pour réaliser la moindre chose.
3D-Crade Scriptor contient un nombre de primitives limitées, des paramètres simples et propose peu d'évolutivité objet. Toutefois, il permettra au non initié de construire très vite des programmes simples et de voir immédiatement les résultats fournis. Et ce, sans devoir se lancer dans la maîtrise de structures de paramètres compliquées, d'options de compilation diverses, d'installation lourdes et rébarbative.
En bref, mon but était de renouer avec cette programmation des années 80, celle que même les « grands » utilisaient, tout comme moi qui n'était qu'un gamin à l'époque. Reste à espérer que certains se laisseront prendre au jeu et qui sait une fois les bases acquises, l'ordinateur peut s'avérer être un outil très créatif.
Si le principe même du langage repose sur le BASIC, il n'en est pas moins structuré, d'une part le célèbre GOTO du BASIC n'existe pas. Les numéros de ligne que certains auraient pu connaître n'existent pas non plus, ni même les labels.
Les éléments conditionnels sont confinés dans des blocs de réalisation et de non réalisation.
Il existe également une structure répétitive que les programmeurs PASCAL ou C connaissent très bien.
Le but étant d'arriver à réaliser des choses visuellement fortes de la manière la plus simpliste possible.

Un script 3D-Crade est concrètement un fichier texte que l'on génère au moyen de NOTEPAD ou de n'importe quel éditeur de texte produisant les mêmes résultats. Veillez simplement à ce que le résultat produit soit un fichier ANSI ou UNICODE (mais l'ANSI est préféré). Si vous utilisez MS-Word comme éditeur de texte, vous avez de forte chance que ça ne fonctionne pas.
De plus les fichiers doivent être sauvegardés avec l'extension .3ds[9] pour être reconnu et exécuté en tant que script par 3D-Crade.

I. Les instructions séquentielles :
Il s'agit de n'importe quelle commande du Contexte ou de n'importe quelle primitive graphique.
Le séparateur d'instruction séquentielle est le retour de chariot (Enter). On ne peut pas placer plus d'une instruction par ligne, on ne peut pas non plus étaler une instruction sur plusieurs lignes.

II. Les remarques
Les remarques permettent de placer des commentaires dans le script. Le symbole utilisé est le suivant :
//
Toute ligne commençant pas ce symbole est ignorée
Si un commentaire est placé derrière une instruction (commande ou primitive), il sera également ignoré.

III. Les branchements conditionnels
Les conditions : Une expression conditionnelle comporte des opérateurs conditionnels :
Le ET logique (&)
Le OU logique (|)
Le non (!)

Chacun de ces opérateurs compare deux expressions booléennes et renvoient un résultat booléen.
Les opérateurs de type :

- Egalité : ==
- Différence : !=
- Plus petit que :<=
- Plus grand que : >=
- Strictement plus petit que : <
- Strictement plus grand que : >

comparent deux expressions de type **FLOAT** et renvoient un résultat booléen.

Pour des raisons techniques et surtout pratiques, il n'y a pas de comparaison possible entre des variables de type TEXT ou entre des variables de type TEXT et FLOAT[10].

[9] .3ds devait signifier 3D-Crade Script, toutefois c'était un mauvais choix puisque 3D-Studio utilise déjà cette extension. Evidemment, je n'avais pas pensé à cela vu que je n'ai jamais utilisé 3D-Studio, merci à Gérard, pour le tuyau.

De même il n'est pas possible de comparer un TEXT et un FLOAT (même si certain langage de script le permettent sur le fond, cela revient à comparer des pommes et des poires, ce qui n'est pas un raisonnement sain).

Les instructions conditionnelles permettent d'exécuter un ensemble d'instructions si la condition est vraie ou (un autre bloc ou rien du tout) si la condition est fausse. 3D-Crade utilise la structure conditionnelle suivante
If (condition)
…// ensemble d'instructions à exécuter
Else
…// ensemble d'instruction à exécuter
Endif
Ou encore,
If (condition)
…// ensemble d'instruction à éxécuter
Endif

Exemple :

```
set(cp,float)
Let(cp,0)
if(cp==1)
 _line(0,0,0,1,1,1)
endif
```

III. Les boucles

Les boucles permettent de répéter un ensemble d'instruction un certain nombre de fois ou une infinité de fois. La méthode est simple, on définit une condition pour répéter une série d'instructions, tant que la condition est vraie l'ensemble d'instruction est répété, une fois que la condition devient fausse, la boucle s'interrompt et la suite du programme s'exécute séquentiellement.
3D-Crade supporte une instruction de boucle appelée **while**. Tant que la condition est vraie on répète l'ensemble d'instruction qui s'étend jusqu'au **wend.**
Il faut noter qu'il est aisé d'imbriquer des boucles les unes dans les autres. Si la condition n'est jamais vraie la boucle s'exécute à l'infini, on dit alors que le programme se plante, à bon entendeur d'entendre.

Exemple :

```
Set (cp,float)
Let (cp,1)
while (cp<8)
 copy(« bloc», « bloc »#cp)
 translate(x+cp*20, y, z, "bloc"#cp)
```

[10] En 3ve-Walker toutefois, il est possible d'utiliser les opérateur == et != entre deux textes, ou variable de type texte. Les comparateurs (<=,>=,<,>) ne sont pas des opérateurs supportés par le type TEXT en 3ve-Walker.

```
 let (cp,cp+1)
wend
```

Le bloc graphique sera copié 7 fois et chaque fois translaté de la valeur de cp*20.
Remarque :Si vous oubliez l'instruction let(cp,cp+1) vous aurez un bel exemple de plantage !

V. Les blocs graphiques

La structure de bloc graphique de 3D-Crade vous permet de créer une nouvelle sous scène dans laquelle vont automatiquement se placer les résultats des primitives graphiques qui seront définies dans ce bloc. Cette méthode est très pratique pour définir des objets complexes pour ensuite les copier et les déplacer ailleurs.

3D-Crade utilise les instructions suivantes pour créer une nouvelle scène :
Block(Keyname) : Keyname représente le nom de la nouvelle scène
… //Primitive graphique qui auront pour scène principale Keyname
Blend
//Les instructions suivantes ne feront plus partie du bloc

Remarque : Il est bien sûr possible d'imbriquer les blocs les uns dans les autres, comme on peut imbriquer des boucles les unes dans les autres[11].
Exemple :

```
let (_scale,300)

set (i,float)
set (i1,float)

block("tabouret1")
 _Ellipse(0,0,0.68,0.16,0.16,20,0,"_default","el_el1")
 _extrudez("el_el1",0.08,2,0,"sf_tabou_upper")

_cylinder(0,0,0,0.02,0.02,0.68,10,10,0,"_default",0,"sf_tabo
u_shaft")
 _Ellipse(0,0,0.20,0.16,0.16,20,0,"_default","el_el3")
 _extrudez("el_el3",0.02,2,0,"sf_tabou_upper3")
 _Line(0,0,0.76,0.16,0,0.76,10,0,"_default","li_tabou")
 _RevolutionZ("li_tabou",0,2*pi,3,"sf_tabou_top")

 _arc(0,0,0,0.20,0.20,10,0,pi/2,0,"_default","pod_b1")
 RotateY(-pi/2,"pod_b1")
 _arc(0,0,0,0.22,0.22,10,0,pi/2,0,"_default","pod_u1")
 RotateY(-pi/2,"pod_u1")

 let (i,1)
 while (i<9)
```

[11] La récursivité ce n'est pas fait pour les chiens ;-)

```
  let (i1,i+1)
  copy("pod_b" # i,"pod_b" #i1)
  copy("pod_u" # i,"pod_u" #i1)
  let (i,i+1)
  rotateZ(pi/4,"pod_b" #i1)
  rotateZ(pi/4,"pod_u" #i1)
 wend
blend
```

Ce script permet de créer un tabouret qui sera placé dans le bloc (sous-scène) « tabouret1 ».
Si je veux afficher un second tabouret, il me suffira d'utiliser les commandes suivantes :

```
Copy("tabouret1 ", " tabouret2 ")
Translate(0,2,0, " tabouret2 ")
```

Ceci m'évite de faire un copier coller double du code pour définir mon second tabouret.

VI. Limitation

Ce langage de script est extrêmement limité puisqu'il ne contient qu'un seul type de boucle et que de plus, il ne contient pas de possibilité de créer ses propres procédures ou fonctions. Toutefois, comme je l'ai cité précédemment le but n'est pas de créer un langage complexe, mais simplement un petit langage qui nous permette de dessiner, avant tout, de la 3D. Les lecteurs plus intéressé par la programmation peuvent se référer au Pascal Objet, au C# ou C++. Ces langages sont extrêmement puissants et permettent de réaliser des choses irréalisables avec 3D-Crade, toutefois leur maîtrise peut s'avérer un peu plus ardue pour un débutant.

Exemple de scripts :

La section « download » du site de jyce3d regorge d'exemple de script 3D-Crade, certains sont très simples à comprendre d'autres légèrement plus compliqués mais toutes les instructions et commandes employées dans ces scripts sont détaillées dans ce chapitre. Avec un peu de patience et d'obstination vous devriez être très vite à même de créer vos propres scriptes.

Lorsque je dis que cette partie est quelque peu ardue, ce n'est pas parce qu'elle est longue, au contraire elle peut se lire très vite, mais les concepts présentés ouvrent pas mal de possibilités (il s'agit tout de même de la description d'un langage de scripting, même si il est simple dans sa définition, ça n'en reste pas moins un langage de programmation). Donc, c'est un peu comme si je vous expliquait la manière dont on peut déplacer les pions dans un jeu d'échec sans vous donner d'indice de stratégie, ça peut vite devenir casse tête, par ailleurs les scripts d'exemple devraient vous aider à appréhender ce côté plus programmatique.

Chapitre IV : Les fichiers de scène en XML

Une scène 3D-Crade est généralement générée au moyen d'un script composé d'instructions (commande ou primitive). Le résultat de ce script est une série d'objet 3D graphique affichable dans la fenêtre du Viewer. Ces entités sont stockées dans la mémoire vive de l'ordinateur et peuvent être manipulées par l'utilisateur. Il semble donc judicieux de pouvoir stocker de manière persistante le contenu de la scène en mémoire. Pour ce faire, 3D-Crade utilise une représentation XML des données qu'il peut sauver dans un document XML. Tout document XML répondant au schéma qui sera présenté à la suite pourra être représenté par 3D-Crade. L'extension de ces fichiers est simplement l'extension .xml[12].

Note : Il est assumé que le lecteur de cette question ait un minimum de connaissance de la norme XML. La plupart des utilisateurs lambda ne seront que rarement confronté à cette partie (moi-même, je ne l'utilise pratiquement jamais), toutefois cette spécification peut-être utile à toute personne désireuse de réaliser des conversions d'un format 3D quelconque vers 3D-Crade ou l'inverse. Tout ceci pour avertir le lecteur que les notions de bases d'XML telles que le modèle DOM, les nœuds, les paramètres, … sont considérées comme connues.

La structure des documents XML de 3D-Crade.

L'entête :
Le fichier XML commence par la définition de la version de XML supportée :
<?xml version="1.0"?>

Les nœuds globaux
<Doc3DCrade></Doc3DCrade> signature du fichier 3D Crade. Ce noeud contient deux sous noeuds :

```
<Doc3DCrade>
    <Parameters />
    <Objects />
</Doc3DCrade>
```

Le nœud <Parameters> contient les paramètres propres à l'affichage de la scène dans 3D Crade.
Le nœud <Objects> Contient la définition des objets de la scène, ainsi que les références entre les objets 3D et les sous scènes, ainsi que les références entre les sous-scènes et les sous-scènes elles même.

Les attributs de <Parameters>

[12] Ndlr :Je sais que j'aurais pu inventer un autre type d'extension de fichier comme 3dx ou n'importe quoi d'autre, mais sur le coup j'ai pensé que .xml serait suffisant (après tout pour être informaticien, il faut être fainéant, maintenant tous les fainéants ne sont pas forcément informaticiens).

Nom	Description
Rhoeye	Distance entre l'oeil et le centre de la scène [mètre]
Azimutheye	angle longitudinale de la position de l'oeil [radian] : azimutheye est compris entre [0,2pi[
Colatiteye	angle de colatitude de la position de l'oeil [radian] : colatiteye est compris entre [0, pi[
Zeye	distance entre l'oeil et le plan de projection [mètre]
Scale	echelle : 1m=x pixel (à revoir)

Le nœud <Objects>

Cette section contient deux principales parties :

- La définition des Objets 3D : Cette partie contient l'énumération des propriétés de chacun des objets, les objets peuvent être des points, des lignes, des frame, des blocs (sous-scènes) proprement dit ou encore des mayesurfaces dans ce cas, tous les mayetrifaces définissant la surface maillées seront définis.
- Les références de Scènes : il s'agit des liens régissant l'arbre des scènes parents et des scènes filles.

La partie définition des Objets

- La liste des points :

```
<Points3D> liste des points
     <Point3D
          Keyname nom unique de l'objet
          Color Couleur
          Hide "false" : visible; "true": caché
          ParentKeyname nom de la scène parent.
          x coordonnée x
          y coordonnée y
          z coordonnée z
  />
</Points3D>
```

- La liste des lignes

```
<Lines3D> liste des lignes
     <Line3D
          Keyname nom unique de l'objet
          Color Couleur
          Hide "false" : visible; "true": caché
          ParentKeyname nom de la scène parent.
          x1 coordonnée x du premier point de la droite
          y1 coordonnée y du premier point de la droite
          z1 coordonnée z du premier point de la droite
          x2 coordonnée x du second point de la droite
          y2 coordonnée y du second point de la droite
          z2 coordonnée z du second point de la droite
```

```
     />
</Lines3D>
```

- La liste des frame :

```
<Frames3D> Liste des parallélépipèdes
     <Frame3D
     Keyname nom unique de l'objet
     Color Couleur
     Hide "false" : visible; "true": caché
     ParentKeyname nom de la scène parent.
     >
     <Points> Liste des point du Frame, le Frame en contient
toujours 8 (sommets).
          <Point
               x coordonnée x du sommet
               y coordonnée y du sommet
               z coordonnée z du sommet
          />
          </Points>
     </Frame3D>
</Frames3D>
```

- La liste des Mayesurface

```
<MayeSurface3D> Liste des MayeSurface3D
     <MayeSurface3D
          Keyname nom unique de l'objet
          Color Couleur
          Hide "false" : visible; "true": caché
          ParentKeyname nom de la scène parent.
          nLong nombre de points en longitude lors de la
     définition de la surface
          nLat nombre de points en latitude lors de la
     définition de la surface
          Transparent 1 - affichage en transparence; 0
     affichage en "dûr-té"
     >
          <MayeTriFaces3D> Liste des polygones.
               <MayeTriFace3D Chaque "MayeTriFace" contient
                des edges (sommets ou points) : Trois pour
                les triangles, quatre pour les
                quadrilatères. Les deux sont, en théorie,
                supportés par 3D Crade mais sachez qu'en
                mode "polygone" triangulaire, il vous faudra
                deux fois plus de polygones qu'en mode
                "polygone" quadrilatère.
               Keyname nom unique de l'objet
               Color Couleur
               Hide "false" : visible; "true": caché
               ParentKeyname nom de la scène parent.
```

/>

<Edges> Liste des sommets : 2,3 ou 4 suivant le type de polygones choisi. Remarquez que pour une même Mayesurface, il n'est pas conseillé de mixer les types de polygones (ça n'a en tout cas pas été testé, donc un utilisateur avertit en vaut 1).

<Edge

x coordonnée x du sommet

y coordonnée y du sommet

z coordonnée z du sommet

/>

</Edges>...

</MayeTriFace3D>

…

</MayeTriFaces3D>

</MayeSurface3D>

</MayeSurfaces3D>

- Liste des scènes :

<Scenes3D> Liste des scènes. Les scènes sont des entités disposant de listes de pointeurs (au sens informatique du terme) ou de référence (pour ceux qui aiment Java) sur des objets concrets.

<Scene3D La scène contient quatre listes de pointeurs, l'une sur les points <Points3DRef>, une autre sur les lignes <Lines3DRef>, une autre sur les Frame <Frames3DRef> et enfin une dernière sur les <Mayesurfaces3DRef>. De plus, la scène dispose des paramètres standards habituels.

Keyname nom unique de l'objet

Color Couleur

Hide "false" : visible; "true": caché

ParentKeyname nom de la scène parent. Si le nom de parent ne contient c'est qu'il s'agit de la scène principale. Il ne peut jamais y avoir qu'une seule scène principale.

>

<Points3DRef>

<Point3DRef la référence à l'objet est donnée par le Keyname qui est unique

Keyname référence à l'objet

/>

...

</Points3DRef>

<Lines3DRef>

```
                    <Line3DRef la référence à l'objet est
               donnée par le Keyname qui est unique
                         Keyname référence à l'objet
                    />
               ...
               </Lines3DRef>
               <Framess3DRef>
                    <Frame3DRef la référence à l'objet est
               donnée par le Keyname qui est unique
                         Keyname référence à l'objet
                    />
               ...
               </Frames3DRef>
               <MayeSurfaces3DRef>
                    <MayeSurface3DRef la référence à l'objet
               est donnée par le Keyname qui est unique
                         Keyname référence à l'objet
                    />
                    ...
               </MayeSurfaces3DRef>
     </Scene3D>
</Scenes3D>
```

La partie de définition des références entre les scènes

Cette partie contient les références nécessaires à la construction de l'arbre des scènes. L'arbre contient des références sur ses noeuds (scènes) qui elles même contiennent des références sur des objets 3D. C'est en quelque sorte des "pointeurs de pointeurs", pour les amateurs de programmation. L'arbre commence par le noeud racine contenu dans le tag de liste <Scenes3DRef>. Ensuite viennent les noeuds enfants, les noeuds petits enfants, …

```
<Scenes3DRef>
     <Scene3DRef
          Keyname keyname de la scène sur laquelle pointe la
     Scene3DRef.
     >
          <Scene3DChildsRef> Scenes enfants de la scène
     courante
               <Scene3DRef
                    Keyname keyname de la scène sur laquelle
               pointe la Scene3DRef.
               />
               ...
          </Scene3DChildsRef>
     </Scene3DRef>
</Scenes3DRef>
```

Remarque : on peut avoir autant de scènes enfants que l’on veut et les scènes enfants peuvent elles même contenir des scènes enfants. C’est ce que l’on appelle dans notre jargon de la récursivité !

Remerciements :

À mon père pour avoir pris la peine de relire, ce document qui a du lui paraître ô combien chiant ;-).