COUTAREL Allan **2020-2021**
COUDERT-BOUDET Nicolas

L2 – Projet d'Info3B
Rapport

Le projet a été réalisé en binôme.

Introduction

Le fichier principal du projet est nommé **echec.pov**. C'est lui qui relie toutes les composantes du projet. Il paramètre l'environnement (lumières et caméra), importe les pièces, puis gère leur placement.

Tous les modèles de pièces sont isolés dans leur fichier qui se trouve dans le dossier **pieces.**

Le code a été entièrement commenté afin de faciliter sa compréhension.

SOMMAIRE

I – Utilitaires

1. Importation des pièces

Afin de créer un projet le plus lisible possible, nous avons choisi de créer un fichier .pov pour chaque pièce du jeu d'échecs (pion, fou, cavalier, dame, roi, tour). Il faut donc importer chacun de ces fichiers dans le fichier **echec.pov** qui gère la modélisation de l'ensemble du jeu d'échecs, des caméras, des lumières… Ainsi, pour créer un objet correspondant à la pièce voulue, il suffit dans le fichier **echec.pov** de créer un objet et d'appeler la macro permettant de modéliser la pièce contenue dans le fichier de la pièce idoine.

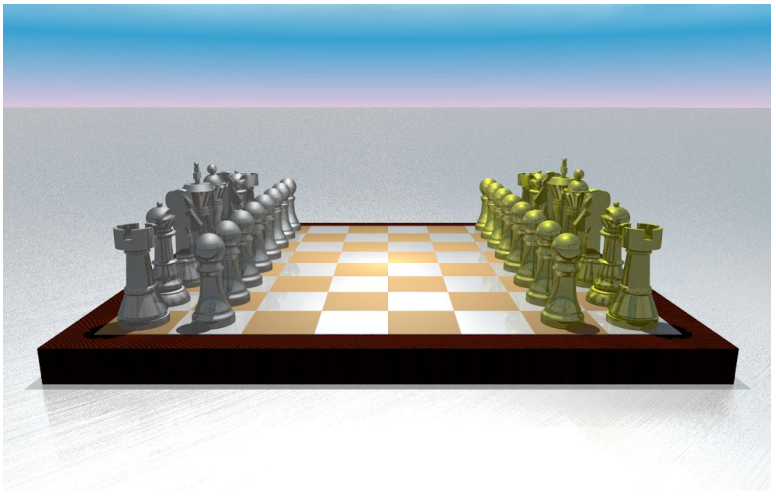
2. Caméra, lumières, ciel et plan

La caméra de la scène est positionnée de manière à avoir une vue symétrique sur chaque camp du jeu. La caméra subit également un déplacement lors de l'animation que nous expliquerons par la suite.

Pour éclairer la scène, nous avons mis en place deux light_source, l'une éclaire globalement le jeu d'échecs car elle est de type *spotlight,* alors que la deuxième , placée proche du jeu permet d'améliorer l'éclairage des pièces, que nous avons jugé trop sombre sans cette dernière.

Pour le ciel, nous avons choisi de charger le *S_Cloud2* sur une sky_sphere pour un effet couché de soleil.

Nous avons ensuite créé un plan d'équation -z = 0.51. C'est sur ce plan que va reposer l'échiquier. Nous lui avons appliqué une texture *Brushed_Aluminum* pour améliorer l'esthétique de celui-ci.



Rendu du fichier echec.pov

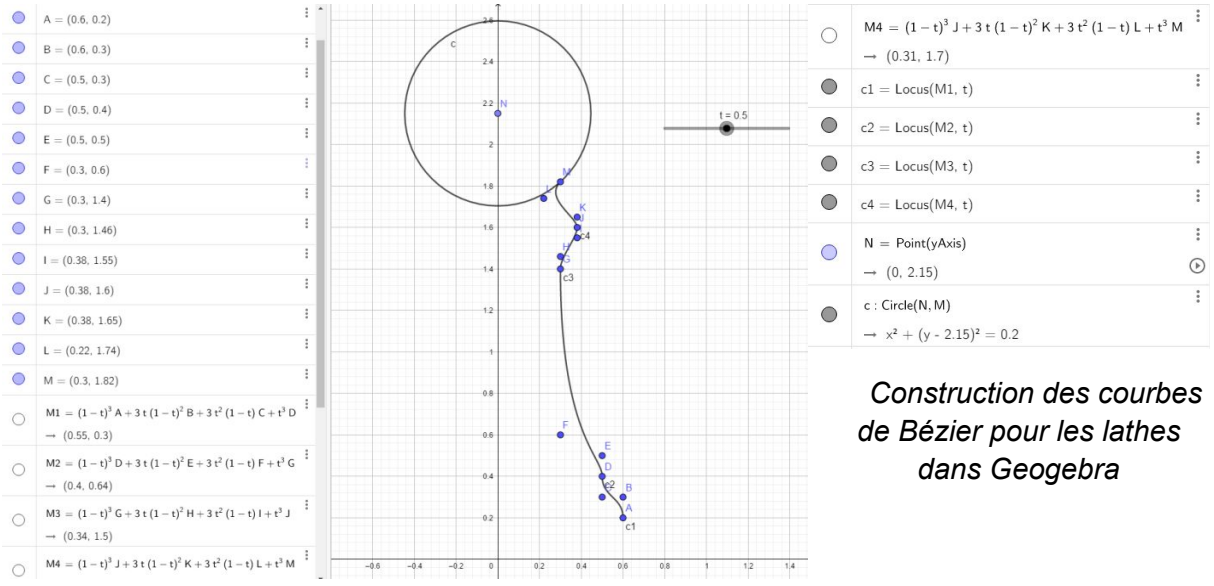
II – Modélisation des pièces

1. Fou

Le fou a été réalisé avec un unique blob. La difficulté de cette pièce était de faire la petite brèche sur le haut de la pièce. Pour cela, nous avons utilisé une sphère très aplatie avec une force négative afin de créer un vide générant la brèche à l'endroit où elle entre en collision avec le reste de la pièce. Cette pièce est donc une suite de sphères plus ou moins aplaties combinées à deux cylindres afin de former le corps de la pièce. Nous avons également utilisé une sphère avec une force négative afin de créer le bas de la pièce et faire comme une coupe, pour que le pied de la pièce soit plat. Les paramètres de la macro permettant de créer le fou sont *transl* pour placer l'objet dans la scène ou le déplacer lors d'une animation via une translation, *rot* pour effectuer une rotation de l'objet et ainsi l'orienter dans la direction souhaitée et *textr* pour lui appliquer une texture. Un scale est également utilisé pour respecter les proportions avec les autres pièces et l'échiquier.

2. Pion

Le pion a été réalisé à l'aide d'un cylindre, 4 lathes et une sphère. La difficulté de cette pièce provient des raccords G1 entre chaque surface. Le cylindre permet de modéliser la base du pion. On retrouve ensuite un ensemble de 4 lathes se raccordant entre elles via des raccords G1. Enfin, la sphère correspond à la sphère située sur le haut du pion. Les paramètres de la macro permettant de créer le pion sont *transl* pour placer l'objet dans la scène ou le déplacer lors d'une animation via une translation et *textr* pour lui appliquer une texture. Un scale est également utilisé pour respecter les proportions avec les autres pièces et l'échiquier.



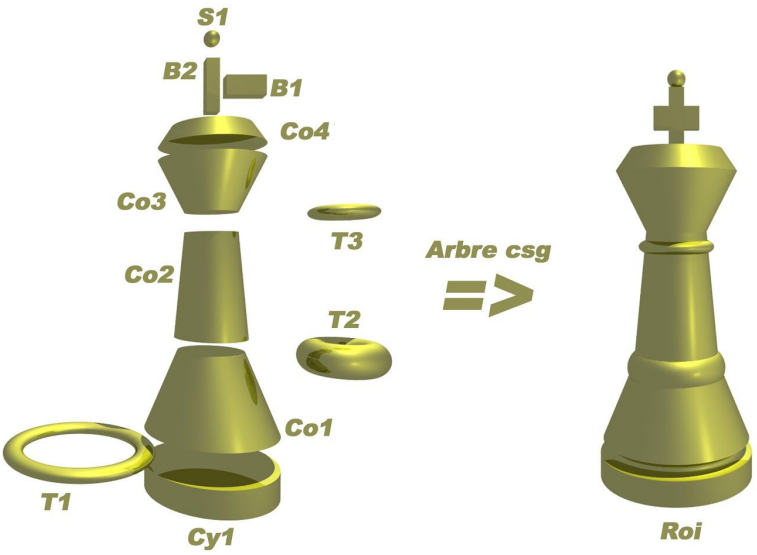
Construction des courbes de Bézier pour les lathes dans Geogebra

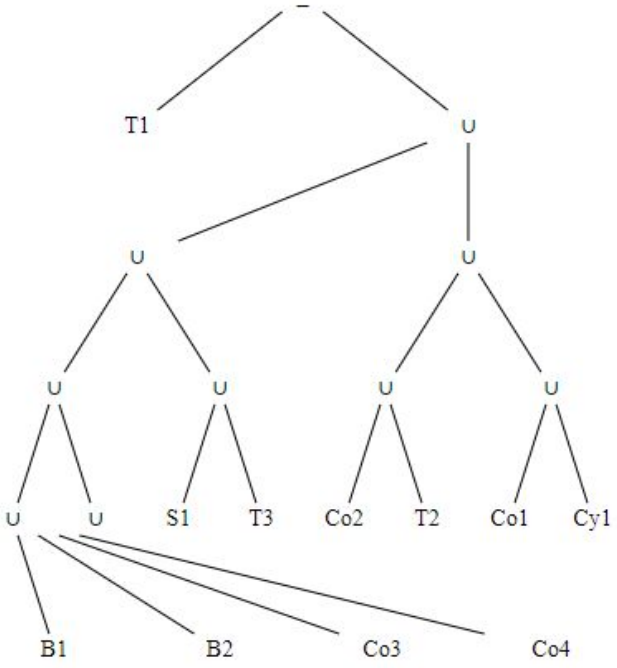
Pour définir la forme de nos 4 lathes, nous avons choisi de les définir via des courbes de Bézier. Nous avons donc placé notre premier point de contrôle A en (0.6,0.2), c'est-à-dire sur le bord de la surface supérieure du cylindre formant la base du pion pour assurer le raccord entre le cylindre et la première lathe. Nous avons ensuite construit 4 courbes de Bézier de la forme que nous voulions en bougeant leurs points de contrôle tout en s'assurant que celles-ci se raccordent entre-elles via des raccords G1. Nous avons également placé sur notre schéma 2D un cercle de rayon 0.45 et de centre (0,2.15) pour représenter la sphère du pion et ainsi s'assurer d'un raccord parfait entre la 4ème lathe et cette sphère. Une fois les courbes de Bézier recherchées obtenues, il ne reste donc plus qu'à reporter les points de contrôle de chaque courbe de Bézier dans une courbe de type *bezier_spline* dans nos lathes. Ainsi, nous obtenons le corps de notre pion. Le raccord des lathes avec le cylindre et la sphère est donc également assuré par le choix de nos points de contrôle comme expliqué précédemment. L'union de ces surfaces de révolution forme alors notre pion.

3. Roi

Le roi est construit à partir de primitives usuelles via un arbre csg. Il est composé d'un cylindre, 4 cônes de révolution, une sphère, 3 torus et 2 boxs. Les paramètres de la macro permettant de créer le roi sont *transl* pour placer l'objet dans la scène ou le déplacer lors d'une animation via une translation et *textr* pour lui appliquer une texture. Un scale est également utilisé pour respecter les proportions avec les autres pièces et l'échiquier. Voici donc une vue éclatée de notre pièce roi permettant de visualiser chacune des primitives que nous avons utilisées. Pour connaître les propriétés de chacune d'entre elles, leur translation, etc., il suffit de se référer au code; chaque primitive est associée au nom qui lui correspond dans le schéma ci-dessous :

Vue éclatée du Roi





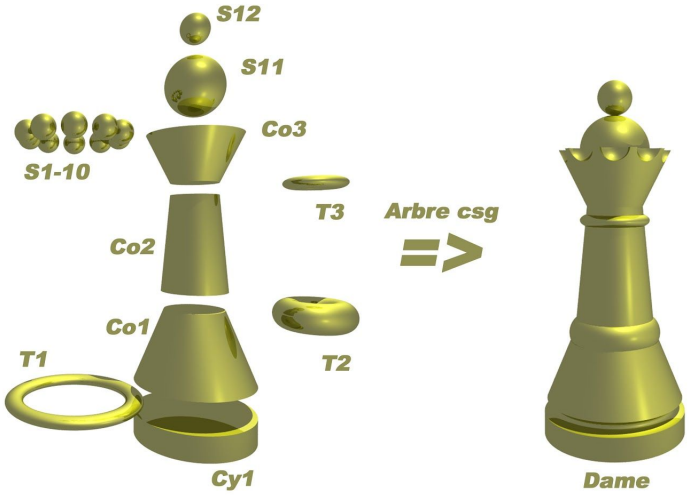
Les différentes valeurs utilisées lors de la construction des primitives et des translations de celles-ci, éventuellement de leur changement de taille et l'implémentation de celles-ci via l'arbre csg ci-contre nous a permis d'obtenir notre roi.

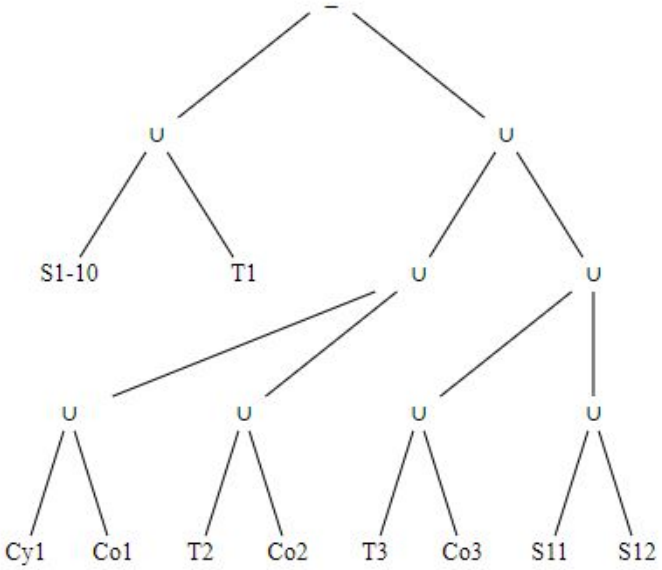
Arbre csg du Roi

4. Dame

La dame est construite à partir de primitives usuelles via un arbre csg. Elle est composée d'un cylindre, 3 cônes de révolution, 12 sphères et 3 torus. Les paramètres de la macro permettant de créer la dame sont *transl* pour placer l'objet dans la scène ou le déplacer lors d'une animation via une translation et *textr* pour lui appliquer une texture. Un scale est également utilisé pour respecter les proportions avec les autres pièces et l'échiquier. Voici donc une vue éclatée de notre pièce dame permettant de visualiser chacune des primitives que nous avons utilisées. Pour connaître les propriétés de chacune d'entre elles, leur translation, etc., il suffit de se référer au code; chaque primitive est associée au nom qui lui correspond dans le schéma ci-dessous :

Vue éclatée de la Dame





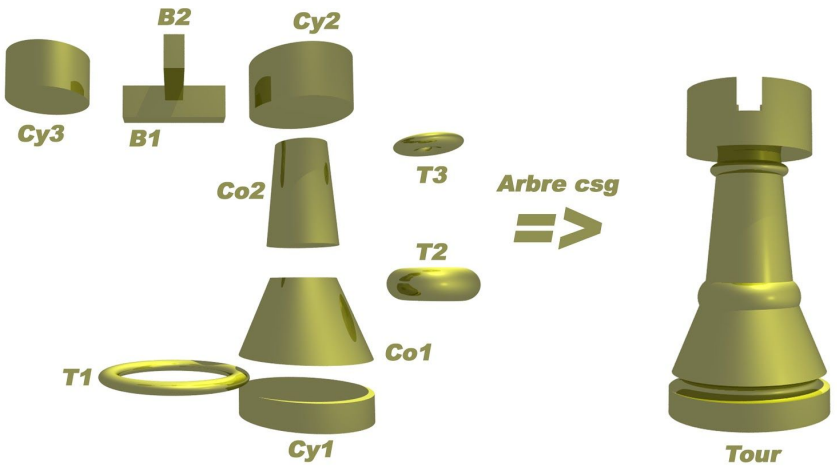
Arbre csg de la Dame

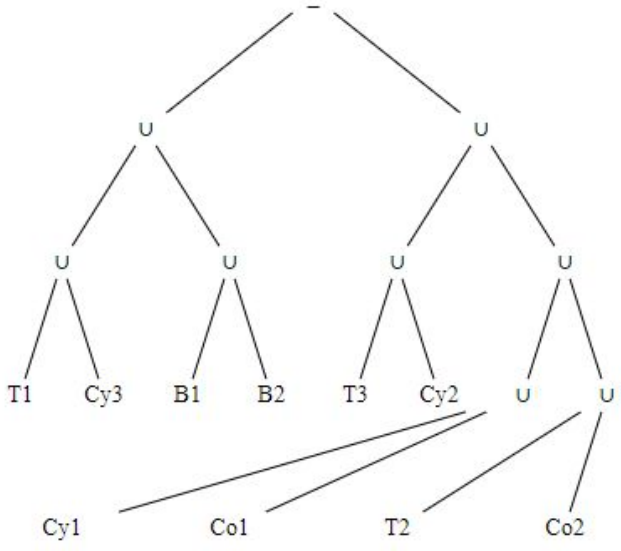
Les différentes valeurs utilisées lors de la construction des primitives et des translations de celles-ci, éventuellement de leur changement de taille et l'implémentation de celles-ci via l'arbre csg ci-contre nous a permis d'obtenir notre dame. La difficulté était de créer la couronne sur le cône le plus haut. Pour cela, nous avons opté pour faire la différence entre le cône et 10 sphères créées et directement placées à la bonne place via une boucle *for*.

5. Tour

La tour est construite à partir de primitives usuelles via un arbre csg. Elle est composée de 3 cylindres, 2 cônes de révolution, 2 boxs et 3 torus. Les paramètres de la macro permettant de créer la tour sont *transl* pour placer l'objet dans la scène ou le déplacer lors d'une animation via une translation et *textr* pour lui appliquer une texture. Un scale est également utilisé pour respecter les proportions avec les autres pièces et l'échiquier. Voici donc une vue éclatée de notre pièce tour permettant de visualiser chacune des primitives que nous avons utilisées. Pour connaître les propriétés de chacune d'entre elles, leur translation, etc., il suffit de se référer au code; chaque primitive est associée au nom qui lui correspond dans le schéma ci-dessous :

Vue éclatée de la Tour



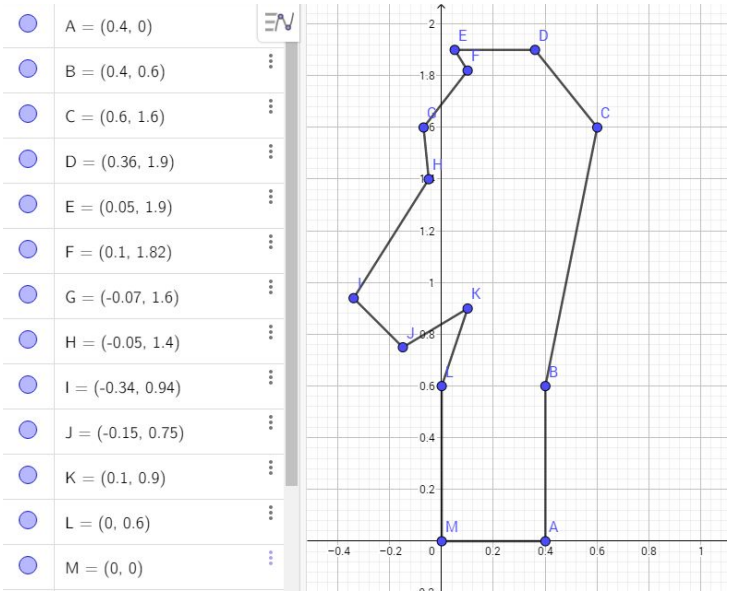


Arbre csg de la Tour

Les différentes valeurs utilisées lors de la construction des primitives et des translations de celles-ci, éventuellement de leur changement de taille et l'implémentation de celles-ci via l'arbre csg ci-contre nous a permis d'obtenir notre tour. La difficulté était de créer la "tête" de la tour. Pour cela, nous avons creusé le cylindre supérieur avec un autre cylindre de taille inférieure et avec deux boxs placées en croix traversant de part et d'autre le cylindre pour créer 4 créneaux.

6. Cavalier

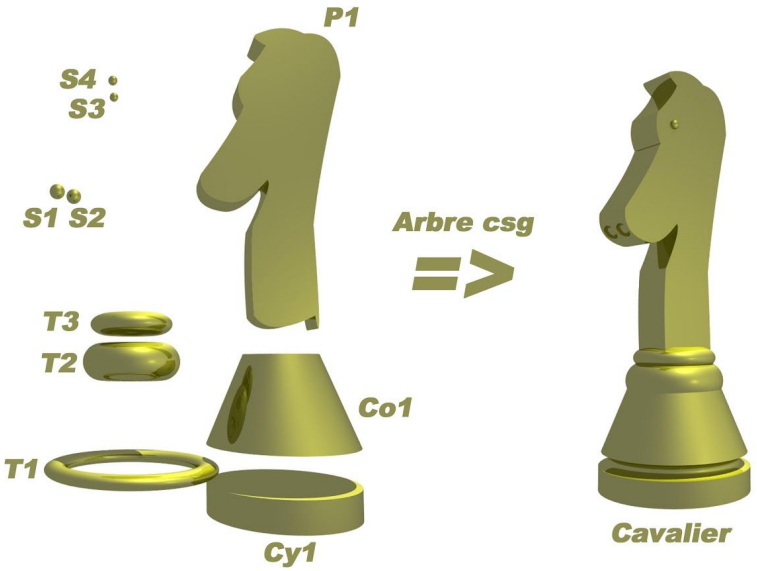
Le cavalier est construit à partir de primitives usuelles via un arbre csg. Le socle de la pièce a été réalisé grâce à 1 cylindre, 1 cône et 3 torus. Pour modéliser le haut de la pièce, nous avons utilisé un prisme avec une disposition quadratique. Nous avons d'abord placé nos points sur Geogebra pour visualiser correctement la forme de la pièce que nous allions faire.



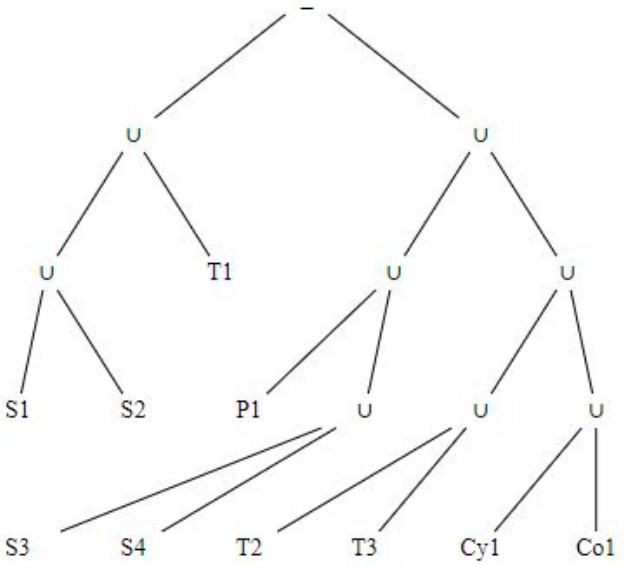
La modélisation de la tête du cheval est donc réalisée dans un premier temps en 2D. Nous avons placé suffisamment de points pour créer une tête de cheval en lignes brisées.

Capture d'écran du schéma 2D de la tête du cheval

Tous les points nous permettant de créer notre tête de cheval sont ensuite utilisés pour créer notre prisme. Pour lisser les contours du prisme, nous définissons notre prisme via une *quadratric_spline* ayant pour points de contrôle les points trouvés précédemment. Ensuite, nous avons rajouté deux sphères pour former les yeux, et fait la différence avec deux autres sphères afin de créer les narines du cavalier.



Vue éclatée du Cavalier



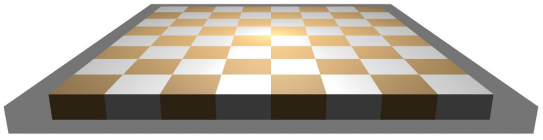
Les différentes valeurs utilisées lors de la construction des primitives et des translations de celles-ci, éventuellement de leur changement de taille et l'implémentation de celles-ci via l'arbre csg ci-contre nous a permis d'obtenir notre cavalier.

Arbre csg du Cavalier

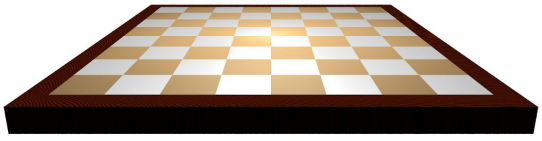
7. Plateau

Pour modéliser les 64 cases de l'échiquier avec le moins de code possible, nous avons choisi de créer l'union de 2 boxs. Ainsi la première box permet de modéliser le contour en bois de l'échiquier ainsi que le dessous de celui-ci en lui appliquant la texture *DMFWood3.* Cette box a pour cote de départ -0.51, de sorte à ce qu'elle soit au même niveau que le plan pour donner une impression que l'échiquier est "posé" sur le plan. La deuxième box permet de modéliser les 64 cases de l'échiquier. Pour créer les 64 cases, il suffit alors d'appliquer une texture à carreaux à la deuxième box et de choisir deux textures différentes pour le quadrillage, nous avons choisi *New_Brass* et *Chrome_Metal*. La taille de la box a été choisie pour avoir 64 cases, c'est-à-dire que nous avons opté pour une box d'une longueur sur l'axe x et y de 8. Ainsi, 1 carreau est de taille 1x1 et 64 carreaux sont donc modélisés sur la box. La cote supérieure de la box contenant les 64 cases est 0, ce qui nous permettra lors du placement des pièces sur les cases de ne pas créer de translation sur l'axe z, puisque le bas de chaque pièce est tel que z=0. Ainsi, chaque pièce sera comme posée sur l'échiquier.

1ère box avec la texture DMFWood3



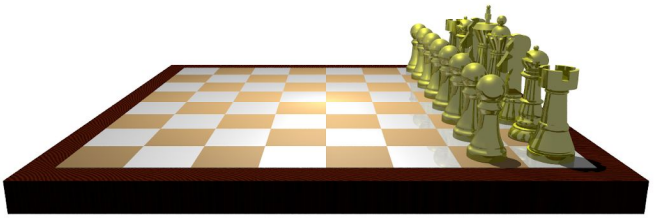
2ème box permettant de modéliser les 64 cases de l'échiquier (1ère box en transparence)



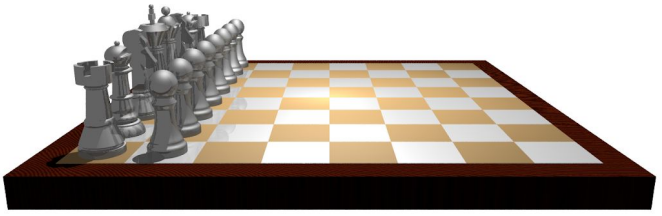
Union des deux boxs => rendu final de l'échiquier

8. Placements des pièces sur l'échiquier

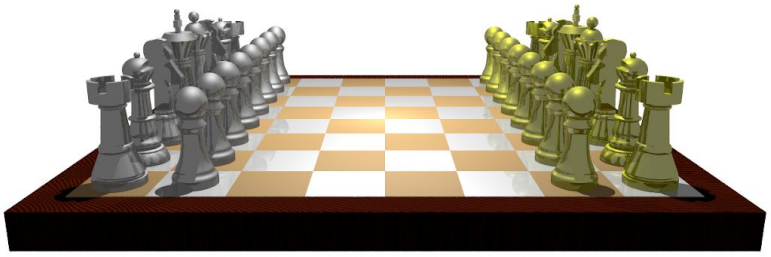
Pour placer toutes les pièces sur l'échiquier, nous avons choisi de créer deux camps; le camp A et B.



Le camp A est l'union de toutes les pièces du premier joueur. On crée donc un objet pour chaque pièce nécessaire auquel on applique une translation sur l'axe x et y pour le placer sur la bonne case de l'échiquier. Comme dit précédemment, un carreau étant de taille 1x1, il est alors facile de centrer les objets puisque le centre de la base de chaque objet est x = 0 et y = 0. Le centre du premier carreau est alors x = 0.5, y = 0.5, il suffit donc d'appliquer une translation de 0.5 en x et 0.5 en y à la première tour pour la centrer sur la première case, et ainsi de suite avec les autres pièces. Comme précisé dans l'explication sur l'échiquier, il n'y a pas de translation à réaliser sur l'axe y pour placer les objets. On applique également une texture *Gold_Metal* à chaque objet créé afin d' identifier les pièces du camp A.



Le camp B suit exactement la même logique de modélisation que le camp A. On adapte bien entendu les translations de chaque objet et les rotations de certains objets pour obtenir le symétrique du camp A de l'autre côté de l'échiquier. On applique une texture *Chrome_Metal* sur chaque objet afin de bien distinguer les deux camps.



Le camp B à gauche et le camp A à droite

III – Animation

1. L'horloge

Le projet a été fait sur Linux. A cause du confinement, nous ne savions pas s'il fallait le faire pour Linux, ou pour Windows, dans le doute nous avons fait les deux. Le projet est par défaut compatible avec Linux. Nous avons laissé les instructions pour le rendre compatible sous Windows directement dans le fichier **echec.pov**.

Sous Linux, pour générer les images du projet il faut utiliser l'instruction : **$ povray -W1280 -H720 +A +KFI1 +KFF186 echec.pov**

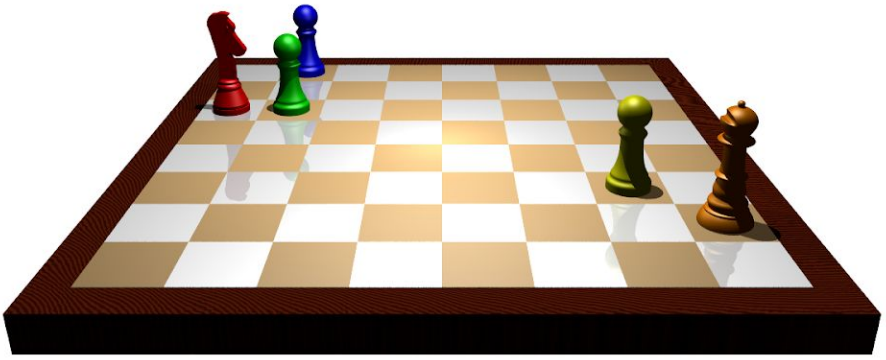
Puis, pour les convertir en gif : **.gif** : **$ convert -loop 1 -delay 40 echec*.png echec.gif**

Pour réaliser l'animation, nous avons d'abord déclaré notre variable myclock qui varie entre 0 et 6.2 en fonction de la variable clock, définie par défaut entre 0 et 1. Nous avons choisi l'intervalle [0;6.2] pour myclock de sorte à pouvoir réaliser tous nos déplacements en facilitant pour chacun d'eux les calculs de translations des objets dépendants de la variable myclock.

Nous avons ensuite réalisé 5 déplacements sur 5 pièces différentes : deux déplacements rectilignes, un déplacement en L, un déplacement horizontal et un déplacement en forme d'un arc de parabole via une courbe de Bézier.

2. Les déplacements des objets

Chaque déplacement d'un objet est une translation. Pour déplacer nos pièces sur l'échiquier, nous avons donc déclaré une variable par déplacement (d1, d2, d3, d4, d5), variable qui est placée en 1er paramètre de la macro modélisant la pièce devant subir un déplacement lors de la créations des deux camps. Dans la macro, le 1er paramètre permet de définir les valeurs de la translation de l'objet créé.



Voici les 5 pièces à déplacer. Nommons-les cavalierRouge, pionVert, pionBleu, pionJaune et fouOrange.

pionBleu subit une translation de valeur d1.

pionJaune subit une translation de valeur d2.

pionVert subit une translation de valeur d3.

fouOrange subit une translation de valeur d4.

cavalierRouge subit une translation de valeur d5.

Les 5 variables d1, d2, d3, d4 et d5 sont initialisées par la position initiale sur l'échiquier de l'objet déplacé correspondant. Ces variables vont ensuite être modifiées en fonction de l'horloge myclock pour créer un déplacement progressif.

Chaque déplacement suit la même logique, dans un intervalle de myclock différent, pour ainsi déplacer les objets un à un.

Si l'horloge myclock est comprise entre 0 et 1, pionBleu doit subir un déplacement rectiligne. Une fois myclock à 1, pionBleu doit avoir avancé de 2 cases, c'est-à-dire que d1 doit diminuer de 2 en x, pionBleu étant initialement placé à 6.5 en x.

Cette diminution de la valeur en x de d1 doit s'effectuer en fonction de myclock pour créer un déplacement progressif, myclock étant comprise entre 0 et 1 à ce moment là, il suffit de faire 6.5-(myclock)*2 pour la valeur en x de d1 pour obtenir le déplacement recherché.

Si l'horloge myclock est comprise entre 1 et 2, pionJaune doit subir un déplacement rectiligne. Une fois myclock à 2, pionJaune doit avoir avancé de 2 cases, c'est-à-dire que d2 doit augmenter de 2 en x, pionJaune étant initialement placé à 1.5 en x.

Myclock étant comprise entre 1 et 2 à ce moment là, il suffit de faire 1.5+(myclock-1)*2 pour la valeur en x de d2 pour obtenir le déplacement recherché.

Si l'horloge myclock est comprise entre 2 et 3, pionVert subit un déplacement en L. Une fois myclock à 3, pionVert doit avoir avancé d'une case en avant et d'une sur sa droite, c'est-à-dire que d3 doit diminuer de 1 en x et augmenter de 1 en y, pionVert étant initialement placé à 6.5 en x et à 2.5 en y.

Quand myclock est comprise entre 2 et 2.5, il suffit de faire 6.5-(myclock-2)*2 pour la valeur en x de d3 pour obtenir l'avancée d'une case de pionVert. Puis ensuite quand

myclock est comprise entre 2.5 et 3, il suffit de faire 2.5+(myclock-2.5)*2 pour la valeur en y de d3 pour obtenir le décalage d'une case de pionVert sur sa droite.

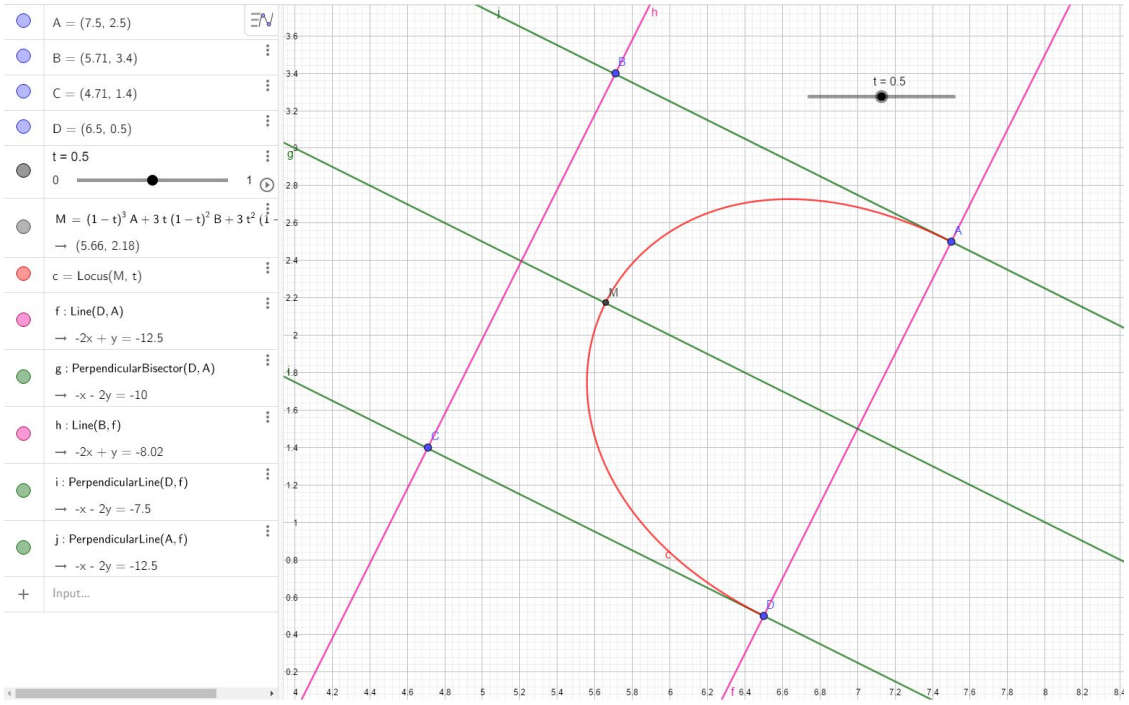
Si l'horloge myclock est comprise entre 3 et 5, fouOrange subit un déplacement vertical. Une fois myclock à 5, fouOrange doit avoir avancé d'une case en avant et d'une sur sa droite, c'est-à-dire que d4 doit augmenter de 1 en x et diminuer de 1 en y, fouOrange étant initialement placé à 0.5 en x et à 6.5 en y. fouOrange doit toutefois monter verticalement, effectuer son déplacement puis descendre verticalement.

Quand myclock est comprise entre 3 et 3.5, il suffit de faire 0+(myclock-3)*4.2 pour la valeur en z de d4 pour faire monter fouOrange de 2.1.

Puis ensuite quand myclock est comprise entre 3.5 et 4.5, il suffit de faire 0.5+(myclock-3.5) pour la valeur en x de d4 et 6.5-(myclock-3.5) pour la valeur en y de d4 pour obtenir le décalage d'une case en avant et sur la droite de fouOrange.

Enfin quand myclock est comprise entre 4.5 et 5, il suffit de faire 2.1-((myclock-4.5)*4.2 pour la valeur en z de d4 pour faire redescendre fouOrange sur l'échiquier.

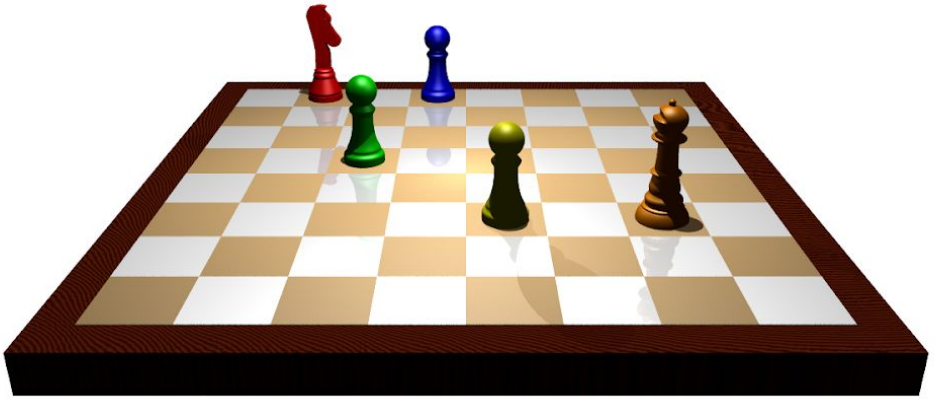
Si l'horloge myclock est comprise entre 5 et 6, cavalierRouge subit un déplacement en forme d'arc de parabole via une courbe de Bézier. Une fois myclock à 6, cavalierRouge doit avoir avancé d'une case en avant et de 2 cases sur sa gauche, c'est-à-dire que d5 doit diminuer de 1 en x et diminuer de 2 en y, cavalierRouge étant initialement placé à 7.5 en x et à 2.5 en y.



Pour réaliser un tel déplacement en forme d'arc de parabole via une courbe de Bézier, on commence par placer 4 points de contrôle A, B, C et D dans geogebra. On place A en (7.5,2.5) soit la position initiale de cavalierRouge et on place D en (6.5,0.5) soit la position d'arrivée de cavalierRouge. On trace ensuite la droite AD ainsi que sa parallèle passant par B. On replace B tel que B appartienne à la droite perpendiculaire à AD passant par A. On trace la perpendiculaire à AD passant par le milieu de [AD] et on place C tel que C soit le symétrique de B par rapport à cette perpendiculaire. On obtient alors une courbe de Bézier en forme d'arc de parabole (courbe rouge) ayant pour points de contrôle A, B, C et D.

Pour utiliser cette courbe en tant que valeur de translation, il faut définir une *spline* de type *cubic_spline* (car povray n'accepte pas les courbes de type *bezier_spline* dans ce cas là) dans laquelle on renseigne la forme paramétrique de la courbe de bézier obtenue auparavant. Pour simuler le curseur t allant de 0 à 1, on le remplace par (myclock-5) puisque myclock se trouve dans l'intervalle [5;6] lorsque cette courbe est définie. On utilise les points de contrôle A, B, C et D calculés précédemment et on obtient la courbe de Bézier en forme d'arc de parabole recherchée. Quand l'horloge myclock est comprise entre 5 et 6, la variable d5 est donc égale au point parcourant cette courbe de Bézier en fonction de myclock, ce qui permet d'obtenir le déplacement de cavalierRouge voulu.

Si l'horloge myclock est comprise entre 6 et 6.2, les pièces restent à leur position d'après déplacements. Cela permet de générer quelques images lors du rendu de l'animation avec les pièces dans leur position finale, pour bien montrer le résultat de toutes ces translations.



Positions des pièces après déplacements

Toutes ces conditions sont simplifiées à l'aide d'un switch, qui permet de définir les instructions à réaliser suivant l'intervalle dans lequel est comprise la valeur de myclock.

Il faut toutefois penser à fixer la position d'arrivée d'un objet lorsque son déplacement est terminé pour éviter que celui-ci ne retourne à sa position initiale.

Nous avons également ajouté un déplacement de la caméra de telle sorte à créer un effet de travelling vertical autour du point <4, 4, 0>, c'est-à-dire le milieu de la surface de l'échiquier, en modifiant la valeur z de la position de la caméra en fonction de l'horloge myclock tel que la position de la caméra soit *location <4, 16, 3+(myclock/2)>,* soit une augmentation de la position de la caméra de 3.1 en z à la fin de l'horloge par rapport à sa position initiale.