

# Examen d'Informatique Graphique - seconde session

Licence Informatique 3ème année

12 juin 2023 - durée 2 heures

*Les document papier sont autorisés - matériels électroniques (calculatrices, téléphones portables, tablettes, ordinateurs personnels, etc.) interdits - aucun périphérique ne peut être connecté aux machines durant la présence des étudiants dans la salle d'examen.*

## Préliminaires

Cet examen est composé de deux exercices, pour lesquels vous disposez de fichiers source nommés respectivement `exo1.html` et `exo2.html`. Ils sont accompagnés de scripts, placés dans le dossier `js`. Le travail que vous aurez réalisé devra se trouver dans ces fichiers, qui seront récupérés automatiquement à la fin de l'examen.

**Rappel :** Pour pouvoir charger des données locales depuis vos scripts (pour les deux exercices), il sera nécessaire :

- d'ouvrir une console, de se déplacer dans le dossier où se trouvent les fichiers html de l'examen et de lancer un serveur web local via la commande : `python3 -m http.server` ;
- lors du chargement du fichier html pour vos tests, il faudra indiquer l'adresse `localhost:8000/exo1.html` ou `localhost:8000/exo2.html`.

## Exercice 1 (10 points)

L'objectif de cet exercice est de vous faire construire et animer un polygone représentant une croix, avec différents matériaux : couleur uniforme, couleurs différentes pour chaque triangle composant le polygone, texture (voir figure 1b).



(a) Vue initiale de l'application de `exo1.html`, avec affichage de 3 cubes colorés.

(b) Vue finale à obtenir : les 3 cubes sont remplacés par 3 croix.

FIGURE 1 – Rendu initial et rendu final pour le premier exercice.

Le fichier `exo1.html` contient le code nécessaire à l'affichage de ces différentes versions du polygone, sachant que le code fourni au départ affiche à la place de ceux-ci le cube coloré vu en TP (voir figure 1a). Ne modifiez rien dans ce fichier en ce qui concerne la position et l'affichage des objets. **On précise que la question 4 est indépendante des 3 premières questions et peut être résolue même si vous n'avez pas réussi celles-ci. Dans ce cas il faudra conserver l'affichage des cubes.**

## Question 1

Complétez la fonction `croix01` qui se trouve dans le fichier `js/exo1.js` ; cette fonction prend un paramètre (`cote`) qui représente la taille du côté dans lequel la croix est inscrite. Toutes les dimensions du contour de cette croix sont données dans la figure 2 en fonction de la valeur de ce paramètre. On précise que le contour de la croix doit se trouver dans le plan  $0xy$  (donc  $z = 0$  pour chacun de ses points) et être centré par rapport à l'origine. La figure indique également les 6 triangles qui doivent composer le polygone représentant cette croix. On précise que dans la version du code fourni, cette fonction retourne un cube coloré, qui devra bien évidemment être remplacé par le code demandé.

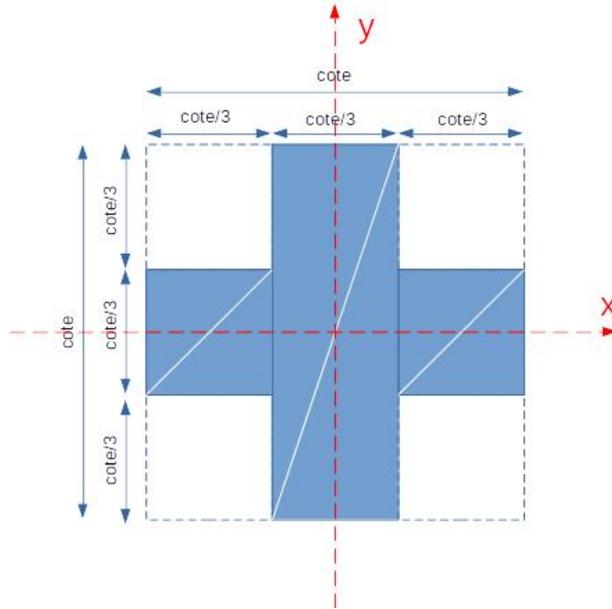


FIGURE 2 – Aperçu de la croix à modéliser, avec les différentes côtes exprimées par rapport au paramètre de la fonction. Le polygone représentant le contour doit être centré par rapport à l'origine du repère et placé dans le plan  $z = 0$ . Les diagonales de couleur blanche indiquent le découpage en triangles qui devra être appliqué.

### Question 2

Complétez le code de la fonction `croix02` qui se trouve dans le fichier `js/exo1.js`, de telle sorte qu'elle construise la même croix que celle de la question 1, en affectant une couleur différente à chaque triangle qui compose le polygone (voir figure 3).

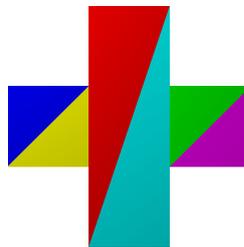


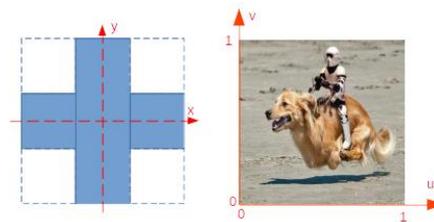
FIGURE 3 – Croix de la question 2, dont les 6 triangles apparaissent avec une couleur différente.

### Question 3

Complétez le code de la fonction `croix03` qui se trouve dans le fichier `js/exo1.js`, de telle sorte qu'elle construise la même croix que celle de des questions précédentes, en lui associant une texture. La texture devra être positionnée de telle sorte que l'on obtienne le résultat de la figure 4a. La figure 4b illustre les systèmes de coordonnées à considérer.



(a) Croix de la question 3, sur laquelle est appliquée une texture.



(b) Représentation des espaces de coordonnées pour la géométrie (à gauche) et les textures (à droite).

FIGURE 4 – Schémas pour l'application des textures.

## Question 4

On considère à présent le fichier `exo1.html` et les deux fonctions `onDocumentMouseDown()` et `animer()` qui s'y trouvent. Complétez ces deux fonctions de telle sorte que lorsque l'on clique sur l'une des croix, celle-ci se mette à tourner autour de son axe  $0x$ . Un nouveau clic stoppe la rotation, et ainsi de suite. Chaque croix doit pouvoir tourner indépendamment l'une de l'autre. On précise que chaque croix, représentée dans le code par les variables `c1`, `c2` et `c3`, s'est vue ajouter un attribut booléen nommé `animation`, initialisé à `false`, permettant de mémoriser le fait que l'objet correspondant doit être animé ou pas.

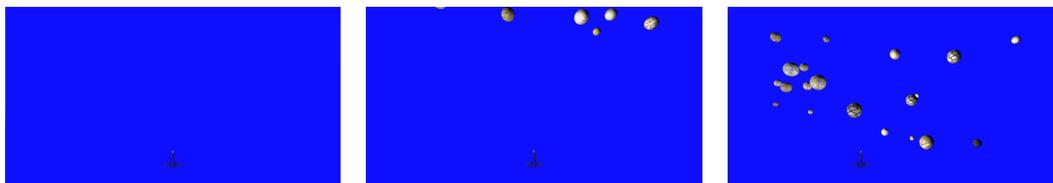
**Vous ne devez modifier que le code de ces deux fonctions et n'appliquer aucune autre modification au fichier `exo1.html`.**

## Exercice 2 (10 points)

Dans cet exercice, vous allez compléter le script `exo2.html` de telle sorte que vous puissiez jouer à un petit jeu dans lequel un vaisseau spatial doit éviter des astéroïdes.

### Question 1

Lors du chargement du fichier `exo2.html` par le navigateur, vous devez obtenir l'image figurant sur la figure 5a : l'écran est vide, à l'exception d'un petit vaisseau spatial qui se trouve positionné en bas et au centre de l'écran.



(a) Aspect initial de l'application. (b) Aspect de l'affichage après création des météorites. (c) Aspect de l'affichage durant l'animation des météorites.

FIGURE 5 – Différents aperçus de l'application durant son développement.

Dans cette question, on vous demande de compléter la fonction `deplacerVaisseau(event)` qui se trouve à la fin du fichier `exo2.html`, de telle sorte qu'elle permette de déplacer le vaisseau de 0.1 unités sur la droite ou sur la gauche selon que l'utilisateur appuie sur la flèche droite ou la flèche gauche. On précise que le vaisseau ne doit pas dépasser les coordonnées  $x = -4$  et  $x = 4$ .

### Question 2

Le script `exo2.html` contient une partie de code permettant de créer des météorites, de les placer dans l'espace et de calculer leur boîtes englobantes. Chaque météorite est créée par un appel à la fonction `creerMeteorite` qui se trouve dans le fichier `js/exo2.js`. Une fois créée, elle est ajoutée dans le tableau `tabMeteorite` et sa boîte englobante est calculée et ajoutée dans le tableau `tabBB`.

Compléter le code de la fonction `creerMeteorite` de telle sorte qu'elle crée une sphère d'un rayon aléatoire compris entre 0.05 et 0.25. Cette sphère devra être texturée aléatoirement par l'une des 7 images `pierre1.jpg` à `pierre7.jpg` situées dans le dossier `Images`. On rappelle qu'un appel à la fonction `Math.random()` retourne une valeur réelle comprise entre 0 et 1. A l'issue de cette question, vous devez obtenir une image similaire à celle de la figure 5b.

### Question 3

Compléter la partie de code indiquée dans la fonction `animer`, qui aura pour objectif de déplacer verticalement vers le bas les différentes météorites. Le déplacement se fera à raison de 0.1 unités à chaque appel. Si la position d'une météorite dépasse la position  $y = -3$ , alors sa position est réinitialisée de la même manière que lors de sa création. Vous penserez à mettre à jour la position des boîtes englobantes des météorites lors de leur déplacement. La figure 5c illustre l'affichage obtenu durant l'animation des météorites.

## Question 4

Compléter le code de la fonction `collision(boxVaisseau, tabBox)` qui se trouve dans le fichier `js/exo2.js`, de telle sorte qu'elle détermine s'il existe une collision entre la boîte englobante du vaisseau et l'une des boîtes englobantes des météorites. En cas de collision, le code fournit assure que l'animation des météorites sera stoppé.