

Commencé le	mardi 20 février 2024, 08:30
État	Terminé
Terminé le	mardi 20 février 2024, 10:30
Temps mis	1 heure 59 min
Note	11,36 sur 20,00 (56,79%)

#### Description

## INF2705 : Infographie (Hiver 2022)

### Contrôle périodique

- L'examen dure 2h et est sur 20 points.
- Assurez-vous d'enregistrer vos réponses tout au long de l'examen pour éviter les problèmes d'enregistrement des réponses.
- Vous avez accès aux diapositives du cours sur MoodleExamen. Aucune autre documentation permise.
- Aucune calculatrice permise (vous avez la calculatrice Windows et une console Python sur votre poste).
- Vous pouvez utiliser une feuille brouillon.
- En cas de problème technique majeur, veuillez contacter le chargé de cours en communiquant avec les surveillants.
- Il y a une partie pratique à l'examen (du code à compléter dans Visual Studio). Allez voir les consignes dans la dernière question.

Bonne chance!

#### Question 1

Correct

Note de 0,25 sur 0,25

Depuis le début de la session, on parle souvent de *caméra* ou d'*observateur* qui est positionné dans la scène indépendamment des objets. Dans un programme OpenGL **moderne** tel que vu dans le cours, sélectionnez le choix suivant qui décrit mieux la nature de la caméra synthétique.

- ☐ a. Une configuration de caméra (position et orientation) conservée par OpenGL dans son état global
- ☐ b. Une partie de la matrice de modèle-vue conservée par OpenGL dans son état global
- ☐ c. Une construction sociale
- ☒ d. Une matrice de transformation passée uniformément aux nuanceurs ✓
- ☐ e. Une matrice de transformation passée comme attribut aux nuanceurs

Votre réponse est correcte.

Les réponses correctes sont :

Une matrice de transformation passée uniformément aux nuanceurs,

Une construction sociale

**Question 2**

Correct

Note de 0,50 sur 0,50

Soient les interactions suivantes dans un jeu vidéo 3D. Selon le pipeline de transformations vu en cours, quel élément (si applicable) doit-on principalement manipuler pour effectuer chacune des interactions?

Observer la scène en suivant le personnage à quelques mètres derrière son épaule : Matrice de visualisation ✓

Changer l'angle de champs de vision (FOV) dans les paramètres du jeu : Matrice de projection ✓

Allonger l'épée tenue par le personnage : Matrice de modélisation ✓

Garder la caméra alignée avec l'orientation du personnage : Matrice de visualisation ✓

Votre réponse est correcte.

La réponse correcte est :

Soient les interactions suivantes dans un jeu vidéo 3D. Selon le pipeline de transformations vu en cours, quel élément (si applicable) doit-on principalement manipuler pour effectuer chacune des interactions?

Observer la scène en suivant le personnage à quelques mètres derrière son épaule : [Matrice de visualisation]

Changer l'angle de champs de vision (FOV) dans les paramètres du jeu : [Matrice de projection]

Allonger l'épée tenue par le personnage : [Matrice de modélisation]

Garder la caméra alignée avec l'orientation du personnage : [Matrice de visualisation]

**Question 3**

Correct

Note de 1,25 sur 1,25

Suivant le modèle de pipeline programmable OpenGL, complétez les descriptions suivantes. Supposez un GPU discret (pas intégré au CPU avec mémoire partagée).

Un VBO contient des données de sommet ✓ et est conservé dans la mémoire du GPU ✓.

Un VAO contient l'état de configuration de données de sommets ✓.

En OpenGL moderne (4.x), on peut avoir au moins un ✓ VAO par programme pour afficher des primitives.

Ce qu'on appelle communément un *EBO* ou *IBO* (Element/Index Buffer Object) est en fait un VBO

✓ qui est utilisé comme tableau de connectivité ✓.

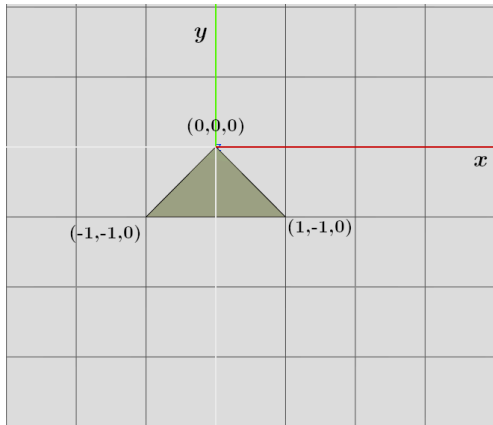
Les données de sommets sont passées en variables d'entrée ✓ au nuanceur de sommets ✓.

**Question 4**

Incorrect

Note de 0,00 sur 1,50

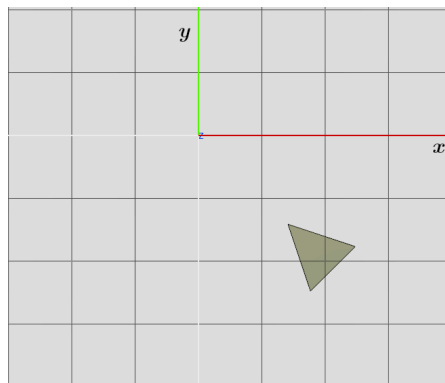
Soit le triangle suivant tel qu'il est chargé en mémoire (dans les images, chaque carré de la grille a une largeur de 1 unité et le z positif pointe vers vous) :



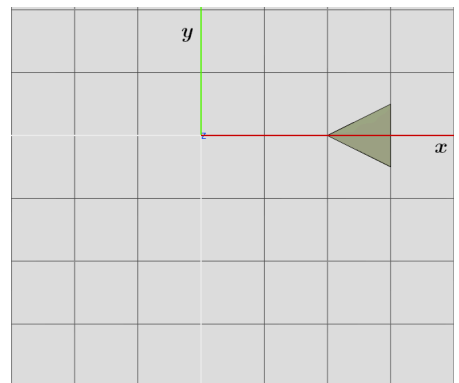
On veut que le triangle tourne autour de l'axe des z selon un angle  $\alpha$  qui change à chaque trame (mis à jour par le programme principal) et une distance de 2 unités de l'axe z. Le triangle est aussi écrasé de 50% sur sa largeur.



Trame 1

 $\alpha = 0^\circ$ 

Trame 2

 $\alpha = 45^\circ$ 

Trame 3

 $\alpha = 90^\circ$ 

On peut exprimer la **matrice de modélisation** à chaque trame par une matrice 4x4 qui est le produit de trois matrices élémentaires.

Parmi les choix ci-dessous, quelles seront les valeurs des matrices 4x4 qui correspondent aux transformations à multiplier pour obtenir la matrice de modélisation? Notez qu'il n'y a que  $\alpha$  qui change à chaque trame, ça doit donc être la seule variable dans vos matrices.

Matrice modélisation =  × ·  × ·  ×

Choix possibles :

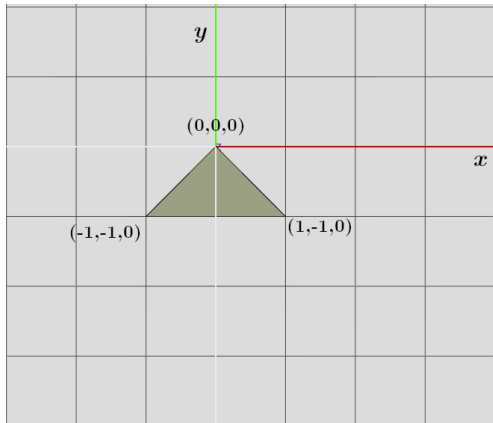
$$M_1 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_2 = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_3 = \begin{vmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_4 = \begin{vmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$M_5 = \begin{vmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_6 = \begin{vmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_7 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_8 = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Votre réponse est incorrecte.

La réponse correcte est :

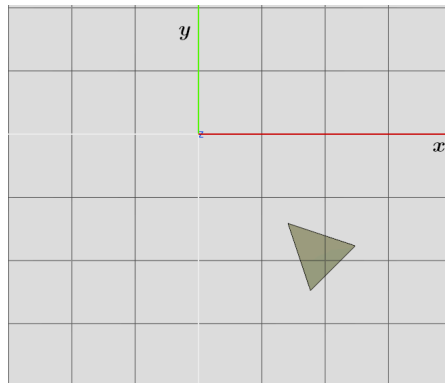
Soit le triangle suivant tel qu'il est chargé en mémoire (dans les images, chaque carré de la grille a une largeur de 1 unité et le z positif pointe vers vous) :



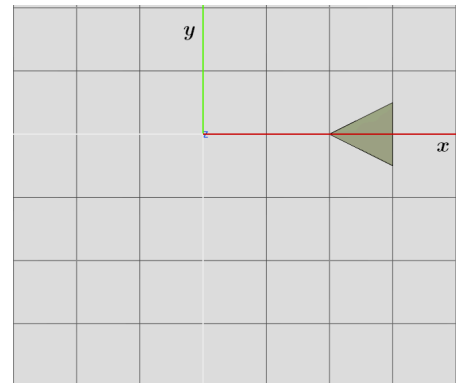
On veut que le triangle tourne autour de l'axe des z selon un angle  $\alpha$  qui change à chaque trame (mis à jour par le programme principal) et une distance de 2 unités de l'axe z. Le triangle est aussi écrasé de 50% sur sa largeur.



Trame 1  
 $\alpha = 0^\circ$



Trame 2  
 $\alpha = 45^\circ$



Trame 3  
 $\alpha = 90^\circ$

On peut exprimer la **matrice de modélisation** à chaque trame par une matrice 4x4 qui est le produit de trois matrices élémentaires.

Parmi les choix ci-dessous, quelles seront les valeurs des matrices 4x4 qui correspondent aux transformations à multiplier pour obtenir la matrice de modélisation? Notez qu'il n'y a que  $\alpha$  qui change à chaque trame, ça doit donc être la seule variable dans vos matrices.

Matrice modélisation = [M6] · [M7] · [M3]

Choix possibles :

$$M_1 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_2 = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_3 = \begin{vmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_4 = \begin{vmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$M_5 = \begin{vmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_6 = \begin{vmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_7 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad M_8 = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

**Question 5**

Incorrect

Note de 0,00 sur 0,50

Dans un programme simple selon le pipeline de transformation vu dans le cours, dans quelle(s) situation(s) parmi les suivantes est-il nécessaire d'initialiser ou de mettre à jour la matrice de projection? On veut si possible un nombre **minimal** de mises à jour de cette matrice tout en gardant la projection désirée et sans déformer l'affichage.

- ☐ a. Redimensionnement de la fenêtre
- ☒ b. Début de chaque affichage de trame ✖
- ☐ c. Avant l'affichage de chaque objet dans la scène
- ☐ d. Une fois avant la première trame
- ☒ e. Changement d'orientation de la caméra synthétique ✖
- ☒ f. Changement de position de la caméra synthétique ✖

Votre réponse est incorrecte.

Les réponses correctes sont :

Redimensionnement de la fenêtre,

Une fois avant la première trame

Question 6

Correct

Note de 1,00 sur 1,00

Les deux fonctions ci-dessous permettent de définir une projection perspective, mais n'utilisent pas les mêmes paramètres.

```
void frustum(float left, float right, float bottom, float top, float nearVal, float farVal);
```

```
void perspective(float fovy, float aspect, float zNear, float zFar);
```

Où *fovy* est un angle en degrés, *aspect* est le rapport x/y.

Y a-t-il une fonction plus générale que l'autre? C'est-à-dire, est-ce qu'une des fonction permet de définir certaines projections perspectives que l'autre ne permet pas?

Si oui, donnez un exemple d'appel à la fonction la plus générale qui ne peut pas être converti en un appel à la fonction la moins générale.

Sinon, mettez le commentaire `/* Non */` dans la boîte de réponse (elle ne peut pas être laissée vide ou inchangée).

Notez que le test unitaire ne vous dit pas si vous avez la bonne réponse, juste que votre réponse est enregistrée.

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 // Si oui,
2 //   perspective(...);
3 //   Ou
4 //   frustum(...);
5 // Sinon, remplacez par un commentaire quelconque
6 frustum(-5, 0, -10, 5, 3, 10);
```

	Test	Résultat attendu	Résultat obtenu	
✓	// Votre réponse est enregistrée std::cout << true;	true	true	✓

Tous les tests ont été réussis ! ✓

► Montrer / masquer la solution de l'auteur de la question (Cpp)

Correct

Note pour cet envoi : 1,00/1,00.

**Description**

Soit le prisme trapézoïdal ci-contre centré en  $(0, 0, 0)$ . Une application OpenGL affiche cet objet dans une fenêtre à l'écran de 200 x 200 pixels. Cette application fait appel à `perspective()` et `lookAt()` de la façon suivante (on s'épargne les détails d'implémentation):

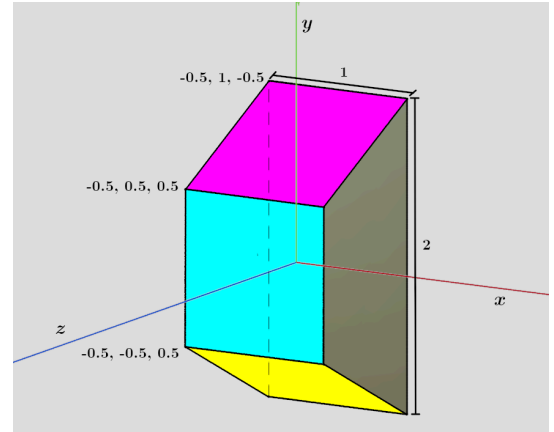
```
glViewport(0, 0, 200, 200);

matrModel.loadIdentity();

//      lookAt(oeil,      cible,      up)
matrVisu.lookAt({0, 0, 8}, {0, 0, 0}, {0, 1, 0});

//      perspective(fovy, aspect, avant, arrière)
matrProj.perspective(25, 0.5, 3, 9);

// Les détails pour mettre à jour et afficher...
```

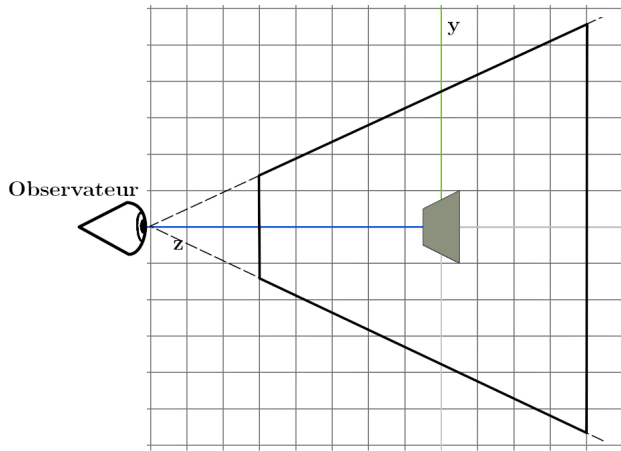
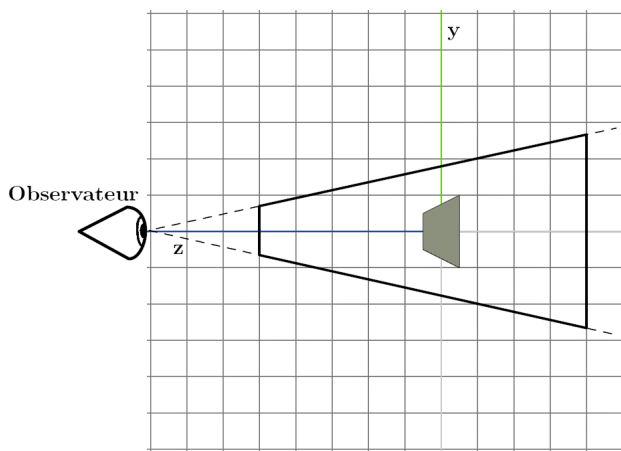
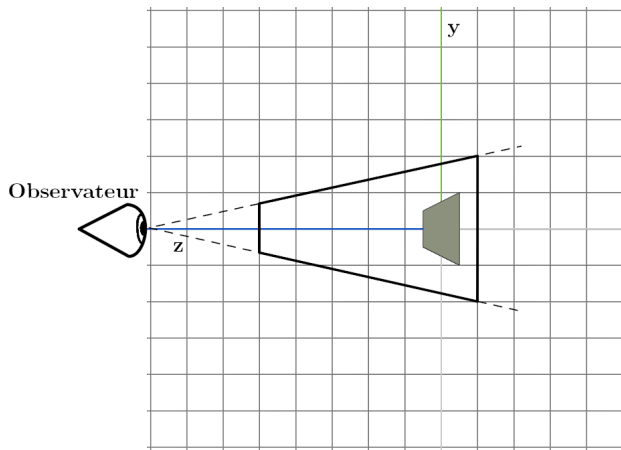


**Question 7**

Partiellement correct

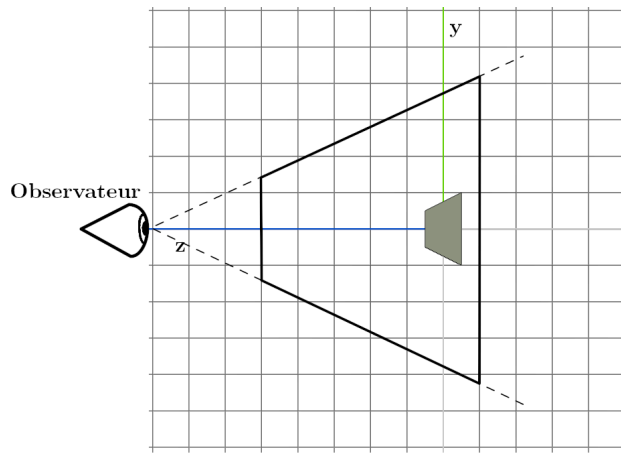
Note de 0,38 sur 0,75

Choisissez la vue de côté qui trace bien le volume de visualisation spécifié par l'appel à `perspective()` dans ce contexte. Chaque carré de la grille a une largeur de une unité et les lignes noires pleines sont les limites du volume, les pointillés ne sont qu'un guide.

☐ a.☐ b.☐ c.

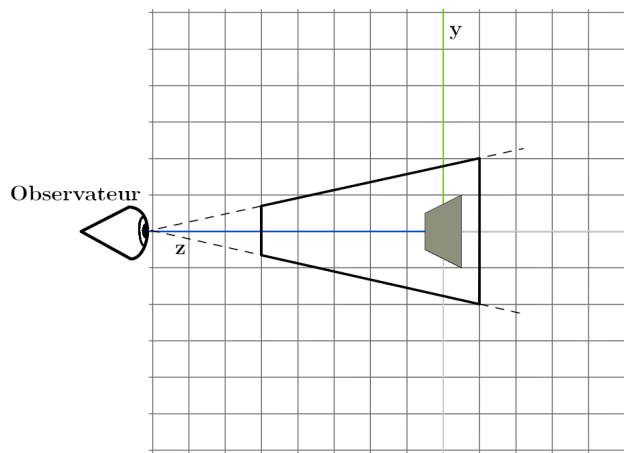


☐ d.



Votre réponse est partiellement correcte.

La réponse correcte est :

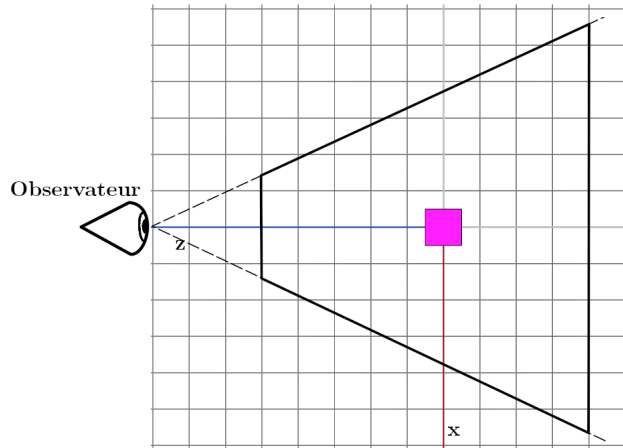
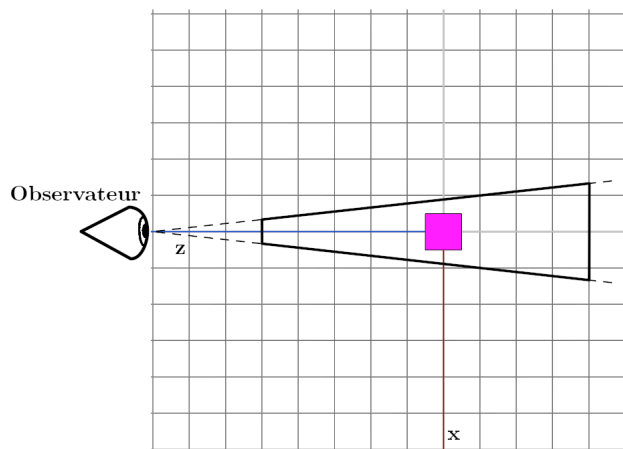
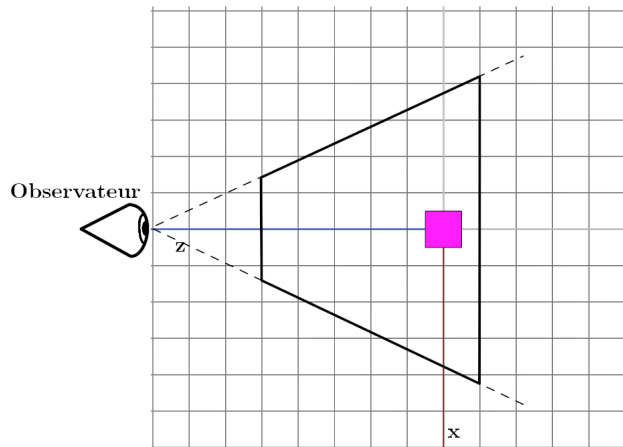


**Question 8**

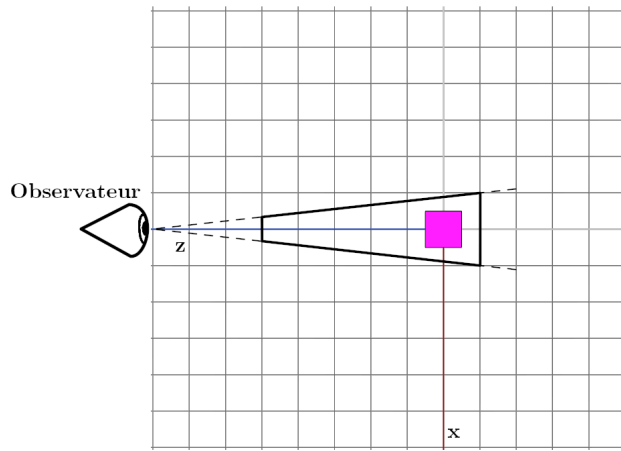
Partiellement correct

Note de 0,38 sur 0,75

Choisissez la vue de haut qui trace bien le volume de visualisation spécifié par l'appel à `perspective()` dans ce contexte. Chaque carré de la grille a une largeur de une unité et les lignes noires pleines sont les limites du volume, les pointillés ne sont qu'un guide.

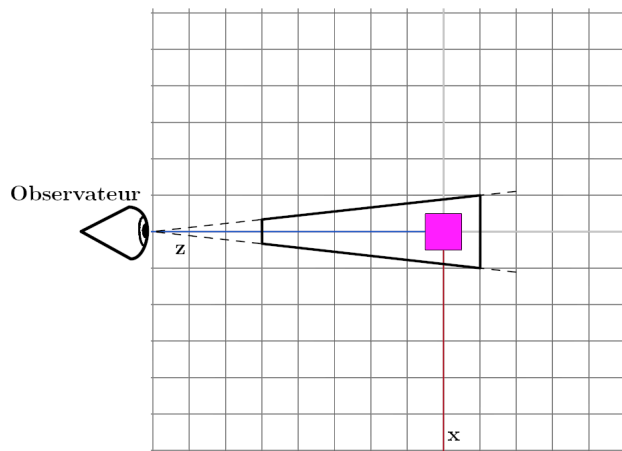
☐ a.☐ b.☒ c.

☐ d.



Votre réponse est partiellement correcte.

La réponse correcte est :



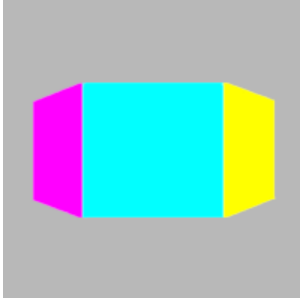
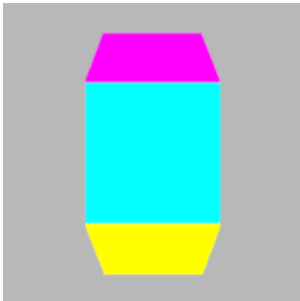
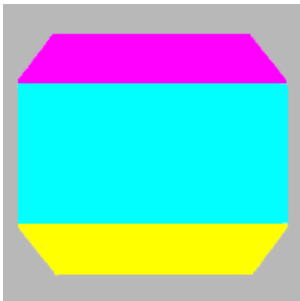
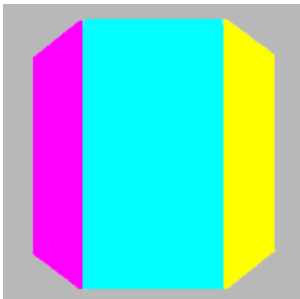
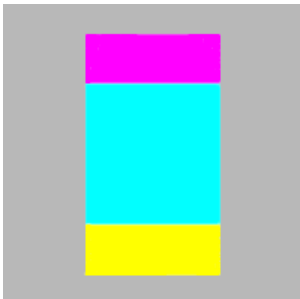
**Question 9**

Partiellement correct

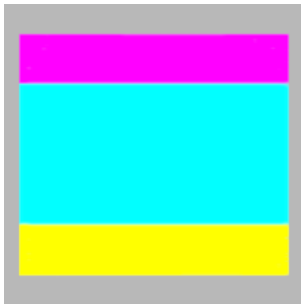
Note de 0,25 sur 0,50

Considérant la clôture et la projection utilisées, comment la face avant de l'objet apparaît-elle à l'écran?

Notez que dans les images, certains angles sont exagérés pour aider à faire la distinction.

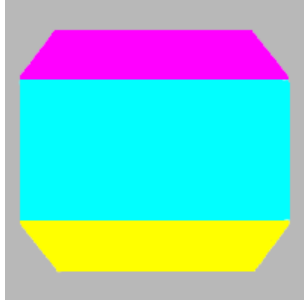
☐ a.☒ b.☐ c.☐ d.☐ e.

☐ f.



Votre réponse est partiellement correcte.

La réponse correcte est :



**Question 10**

Partiellement correct

Note de 0,25 sur 0,50

Encore avec le même prisme trapézoïdal et le même appel à `lookAt()`, y a-t-il moyen de changer l'appel à `perspective()` de façon à ce que les faces haut et bas (magenta et jaune) ne soit **plus du tout** visibles? Autrement dit, la face avant (cyan) cache **complètement** les autres faces. On suppose bien sûr qu'aucun des sommets de l'objet ne sorte du volume de visualisation (pas de *clipping*).



Serait-ce possible avec une projection orthogonale?



Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.

La réponse correcte est :

Encore avec le même prisme trapézoïdal et le même appel à `lookAt()`, y a-t-il moyen de changer l'appel à `perspective()` de façon à ce que les faces haut et bas (magenta et jaune) ne soit **plus du tout** visibles? Autrement dit, la face avant (cyan) cache **complètement** les autres faces. On suppose bien sûr qu'aucun des sommets de l'objet ne sorte du volume de visualisation (pas de *clipping*).

[Non, il faut aussi rapprocher la caméra]

Serait-ce possible avec une projection orthogonale?

[Non, ce n'est pas possible]

**Question 11**

Partiellement correct

Note de 2,33 sur 3,50

Toutes les sous-questions qui suivent présentent différentes situations pour lesquelles on souhaite déterminer le résultat d'un « test unitaire » pour tester l'effet de certains énoncés OpenGL. Chaque sous-question est indépendante des autres.

Pour chaque test, on vous donne les valeurs des attributs du fragment courant et les valeurs présentes dans les différents tampons (profondeur, couleur, stencil) et on vous demande de donner, selon les énoncés OpenGL, les valeurs subséquentes qui seront présentes dans les différents tampons. Écrivez vos valeurs avec au plus deux décimales et faites attention de contraindre vos valeurs entre 0 et 1.

**1)**

	profondeur (z)	couleur (RGBA)	stencil
valeurs des attributs du fragment	0.3	0.9, 0.9, 0.9, 0.6	-
valeurs déjà présentes dans les tampons	0.7	0.4, 0.4, 0.4, 0.3	-

```
glDepthFunc(GL_ALWAYS);
glEnable(GL_DEPTH_TEST);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
```

	profondeur (z)	couleur (RGBA)	stencil
valeurs subséquentes dans les tampons	0,3 ✓	0,7 ✓ , 0,7 ✓ , 0,7 ✓ , 0,48 ✓	-

**2)**

	profondeur (z)	couleur (RGBA)	stencil
valeurs des attributs du fragment	0.3	0.9, 0.9, 0.9, 0.6	-
valeurs déjà présentes dans les tampons	0.7	0.4, 0.4, 0.4, 0.3	-

```
glDepthFunc(GL_LESS);
glEnable(GL_DEPTH_TEST);
glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
```

	profondeur (z)	couleur (RGBA)	stencil
valeurs subséquentes dans les tampons	0,7 ✗	0,4 ✗ , 0,4 ✗ , 0,4 ✗ , 0,3	-

**3)**

	profondeur (z)	couleur (RGBA)	stencil
valeurs des attributs du fragment	0.3	0.9, 0.9, 0.9, 0.6	-
valeurs déjà présentes dans les tampons	0.7	0.4, 0.4, 0.4, 0.3	-

```
glDepthFunc(GL_NEVER);
glEnable(GL_DEPTH_TEST);
glBlendFunc(GL_ONE, GL_ZERO);
glEnable(GL_BLEND);
```

	profondeur (z)	couleur (RGBA)	stencil
valeurs subséquentes dans les tampons	0,7 ✓	0,4 ✓ , 0,4 ✓ , 0,4 ✓ , 0,3	-

4)

	profondeur (z)	couleur (RGBA)	stencil
valeurs des attributs du fragment	0.3	0.9, 0.9, 0.9, 0.6	-
valeurs déjà présentes dans les tampons	0.7	0.4, 0.4, 0.4, 0.3	5

```
glDepthFunc(GL_GREATER);
glEnable(GL_DEPTH_TEST);
glDisable(GL_BLEND);
glStencilFunc(GL_GEQUAL, 3, 0xFF);
glStencilOp(GL_REPLACE, GL DECR, GL_INCR);
glEnable(GL_STENCIL_TEST);
```

	profondeur (z)	couleur (RGBA)	stencil
valeurs subséquentes dans les tampons	0,7 ✓	0,4 ✓ , 0,4 ✓ , 0,4 ✓ , 0,3 ✓	3 ✓

5)

	profondeur (z)	couleur (RGBA)	stencil
valeurs des attributs du fragment	0.3	0.9, 0.9, 0.9, 0.6	-
valeurs déjà présentes dans les tampons	0.7	0.4, 0.4, 0.4, 0.3	5

```
glDepthFunc(GL_LESS);
glEnable(GL_DEPTH_TEST);
glDisable(GL_BLEND);
glStencilFunc(GL_NEVER, 3, 0xFF);
glStencilOp(GL_REPLACE, GL DECR, GL_INCR);
glEnable(GL_STENCIL_TEST);
```

	profondeur (z)	couleur (RGBA)	stencil
valeurs subséquentes dans les tampons	0,7 ✓	0,4 ✓ , 0,4 ✓ , 0,4 ✓ , 0,3 ✓	3 ✓

6)

	profondeur (z)	couleur (RGBA)	stencil
valeurs des attributs du fragment	0.3	0.9, 0.9, 0.9, 0.6	-
valeurs déjà présentes dans les tampons	0.7	0.4, 0.4, 0.4, 0.3	5

```
glDepthFunc(GL_LESS);
glEnable(GL_DEPTH_TEST);
glDisable(GL_BLEND);
glStencilFunc(GL_LESS, 3, 0xFF);
glStencilOp(GL_REPLACE, GL DECR, GL_INCR);
glEnable(GL_STENCIL_TEST);
```

	profondeur (z)	couleur (RGBA)	stencil
valeurs subséquentes dans les tampons	0,7 ✗	0,4 ✗ , 0,4 ✗ , 0,4 ✗ , 0,3 ✗	4 ✗

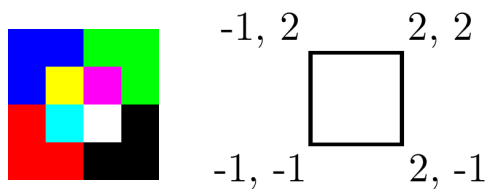


**Question 12**

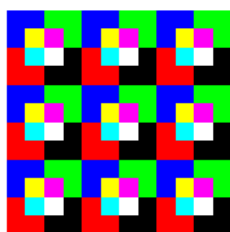
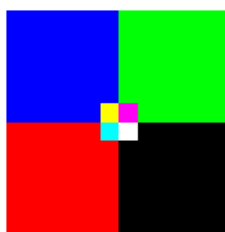
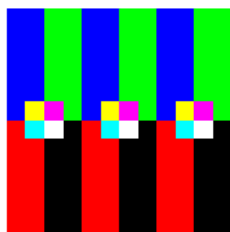
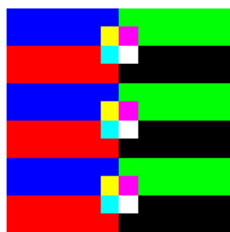
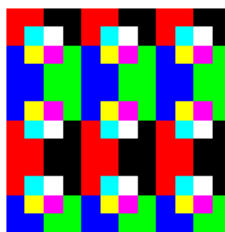
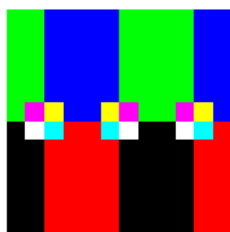
Correct

Note de 2,00 sur 2,00

Soit une texture et un quadrilatère avec ses coordonnées de texture pour chacun de ses sommets (on assume qu'il est affiché à l'écran tel-quel).



Indiquez pour chacun des appels à `glTexParameter` lequel des affichages suivants serait généré.

**A****B****C****D****E****F****G****H****I**

F



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
```

C



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

E



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

G



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

B




```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
```

Votre réponse est correcte.

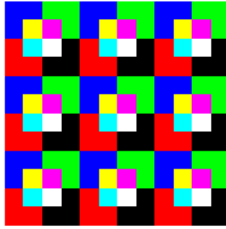
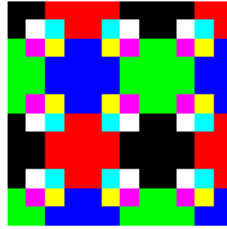
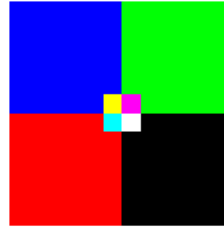
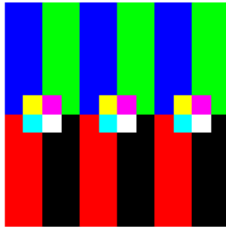
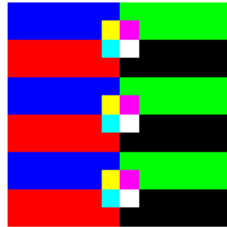
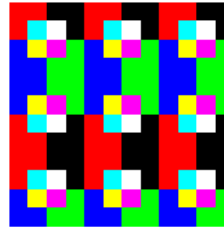
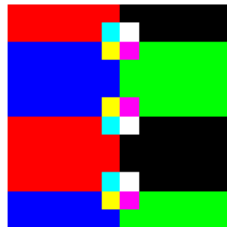
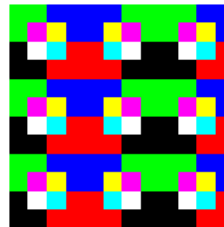
La réponse correcte est :

Soit une texture et un quadrilatère avec ses coordonnées de texture pour chacun de ses sommets (on assume qu'il est affiché à l'écran tel-quel).



-1, 2                      2, 2  
  
-1, -1                      2, -1

Indiquez pour chacun des appels à `glTexParameter` lequel des affichages suivants serait généré.

**A****B****C****D****E****F****G****H****I**

[F] `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);`  
`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);`

[C] `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);`  
`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);`

[E] `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);`  
`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`

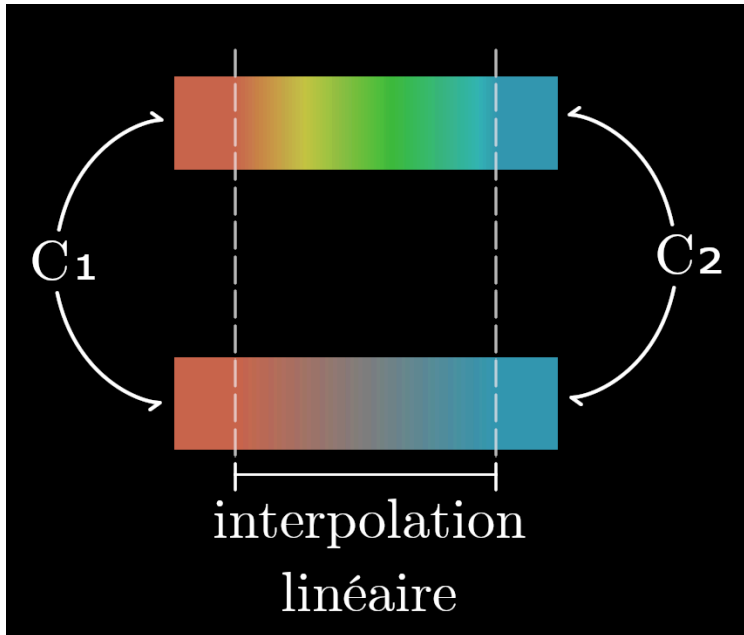
[G] `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);`  
`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);`

[B] `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);`  
`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);`

Commentaire :

## Description

Soit deux couleur  $C_1$  et  $C_2$  où  $C_1$  est  $\text{rgb}(0.8, 0.4, 0.3)$  et  $C_2$  est  $\text{rgb}(0.2, 0.6, 0.7)$ . On constate que  $C_2 = 1 - C_1$  en RGB. C'est une façon un peu bête d'approximer le *négatif* d'une couleur. Dans l'image ci-dessous, on observe des gradients entre ces deux couleurs (le gradient se produit entre les lignes pointillées).



## Question 13

Correct

Note de 0,75 sur 0,75

Quelle est la couleur au point milieu du gradient de la bande inférieure (en valeurs dans  $[0, 1]$ )?

R=  ✓ , G=  ✓ , B=  ✓ .

Serait-ce la même valeur pour n'importe quel gradient entre une couleur  $C$  et  $1 - C$ ?  ✓


**Question 14**

Partiellement correct

Note de 0,38 sur 0,75

Nous voulons écrire une application OpenGL simple qui génère un gradient similaire au gradient supérieur entre  $C_1$  et  $C_2$ . On dessine à l'écran une primitive ligne (GL\_LINES) entre deux points  $P_1$  et  $P_2$ . On assume le code habituel pour passer les positions de sommet et appliquer les transformations.

Parmi les choix suivants, quelle serait la solution fonctionnelle la plus simple pour obtenir ce résultat?

- ☐ a. Passer  $C_1$  et  $C_2$  en coordonnées RGB aux nuanceurs comme attribut des sommets  $P_1$  et  $P_2$  respectivement.  
Dans le nuanceur de sommets, affecter cet attribut tel-quel à une variable de sortie pour le nuanceur de fragments.  
Dans le nuanceur de fragments, convertir la variable d'entrée en HSV et l'affecter en sortie du fragment.
- ☐ b.
- ☐ c. Passer  $C_1$  et  $C_2$  en coordonnées HSV aux nuanceurs comme attribut des sommets  $P_1$  et  $P_2$  respectivement.  
Dans le nuanceur de sommets, affecter cet attribut tel-quel à une variable de sortie pour le nuanceur de fragments.  
Dans le nuanceur de fragments, convertir la variable d'entrée de HSV à RGB et l'affecter en sortie.
- ☒ d. Passer  $C_1$  et  $C_2$  en coordonnées HSV aux nuanceurs comme variables uniformes.   
Passer 0 et 1 comme attribut des sommets  $P_1$  et  $P_2$  respectivement.  
Dans le nuanceur de sommets, affecter la valeur d'entrée à une variable de sortie pour le nuanceur de fragments.  
Dans le nuanceur de fragment, interpoler (fonction *mix*) entre  $C_1$  et  $C_2$  selon la variable d'entrée puis convertir en RGB pour affecter en sortie du fragment.

Votre réponse est partiellement correcte.

La réponse correcte est :

Passer  $C_1$  et  $C_2$  en coordonnées HSV aux nuanceurs comme attribut des sommets  $P_1$  et  $P_2$  respectivement.

Dans le nuanceur de sommets, affecter cet attribut tel-quel à une variable de sortie pour le nuanceur de fragments.

Dans le nuanceur de fragments, convertir la variable d'entrée de HSV à RGB et l'affecter en sortie.

**Question 15**

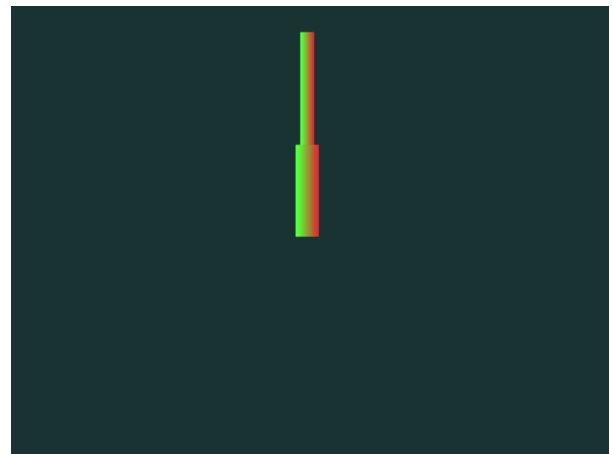
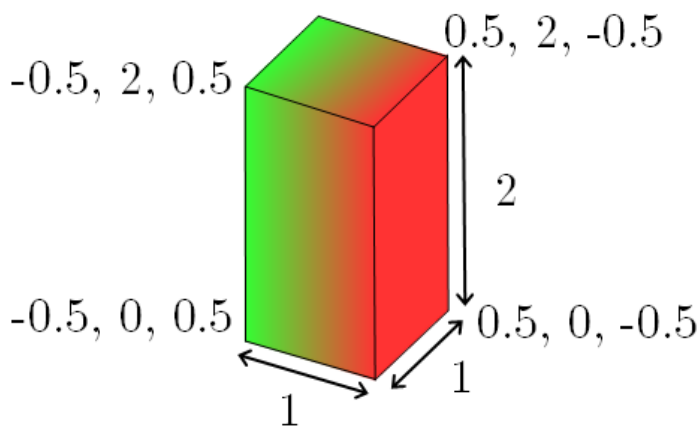
Terminé

Note de 1,65 sur 5,50

**TRÈS IMPORTANT AVANT DE COMMENCER** : dans la boîte de texte ci-dessous, inscrivez votre numéro de poste (votre machine devrait avoir une étiquette). Ça nous prend cette information pour qu'on puisse retrouver votre code source et l'associer à votre session d'examen pour la correction.

## Description de la tâche

Vous devez écrire une application OpenGL qui affiche à l'écran deux prismes rectangulaires. Le premier prisme (plus épais) a sa base à l'origine (0,0,0) de la scène. Le deuxième prisme (plus mince) est connecté au premier et bouge avec lui. Les deux prismes partagent la même source de données mais sont dessinés avec des transformations différentes. Les données des sommets (positions, couleurs, indices de faces) sont déclarées pour vous dans *donnees.hpp*. Voici le prisme en question tel qu'il est défini dans l'entête ansi qu'une courte démo de l'exécution :



Les touches W et S font tourner le premier prisme autour de l'axe des z. A et D font tourner le premier prisme sur lui-même autour de son axe longitudinale (sur sa hauteur si vous voulez). Les flèches ← et → font tourner le deuxième prisme sur lui-même autour de son axe longitudinale (comme pour le premier prisme). ↑ et ↓ font tourner le deuxième prisme sur son joint avec le premier prisme. Pensez à un couteau de poche qui replie sa lame.

Pour comprendre exactement le résultat voulu, vous pouvez exécuter une application (*Intra\_H24\_Solution.exe*) qui est en fait le solutionnaire de la question. Ça vous permet d'interagir avec le programme et de comprendre concrètement ce que chaque touche de clavier fait et comment est fait l'affichage. Il y a un raccourci sur le bureau, sinon vous pouvez télécharger les fichiers de départ avec le lien fourni à la fin des consignes. Votre application devrait avoir un comportement identique à celui du solutionnaire qui vous est fourni.

## Environnement et code fourni

Lorsque vous vous êtes assis à votre poste, il devait y avoir une solution Visual Studio (*Intra\_H24.sln*) déjà ouverte. Sinon, il y a un raccourci sur le bureau. Vous devez compléter les *TODO* dans les fichiers *main.cpp*, *vert.glsl* et *frag.glsl*. **Vous ne devez pas modifier d'autres fichiers que ceux-ci**, seul ces trois fichiers seront tenus en compte pour la correction. Vous n'avez pas à remettre vos fichiers dans Moodle, faites juste bien sauvegarder les fichiers dans Visual Studio sur votre poste.

Les cadriciel fourni utilise la classe *TransformStack* comme dans les exemples faits en cours. Rappel des méthodes de cette classe :

- *top()* : La matrice sur le dessus de la pile.
- *push()* : Copier et empiler le *top()*.
- *push(m)* : Empiler *m*.
- *pushIdentity()* et *loadIdentity()* : Empiler ou charger une matrice identité.
- *pop()* : Dépiler le dessus de la pile.
- *scale(v)*, *translate(v)*, *rotate(a, v)* : Les transformations usuelles. Notez que les angles sont en degrés.
- *lookAt(oeil, cible, up)* : Même appel qu'avec glm.
- *frustum(plane)* : Les membres du plan sont dans le même ordre que l'appel de glm, donc on fait *matr.frustum({L, R, B, T, N, F})*.
- *perspective(fovy, aspect, proche, loin)* : Même appel qu'avec glm, mais les angles sont en degrés.

Si vous avez effacé par mégarde certains des fichiers fournis : [lien pour télécharger les fichiers de départ](#)

Votre numéro de poste (de la forme L-66XX-XX) : L-6611-25

Commentaire :

Élément	Note	Max
VBO positions	0.20	0.25
VBO couleurs	0.25	0.25
EBO	0.20	0.25
Position caméra	0.25	0.50
Prisme 1 : transformations		1.00
Prisme 1 : traçage		0.25
Prisme 2 : transformations		1.00
Prisme 2 : traçage		0.25
appliquerPerspective()		0.75
Nuanceur sommets : attributs	0.25	0.25
Nuanceur sommets : calculs	0.25	0.50
Nuanceur fragments	0.25	0.25
TOTAL	1.65	5.50