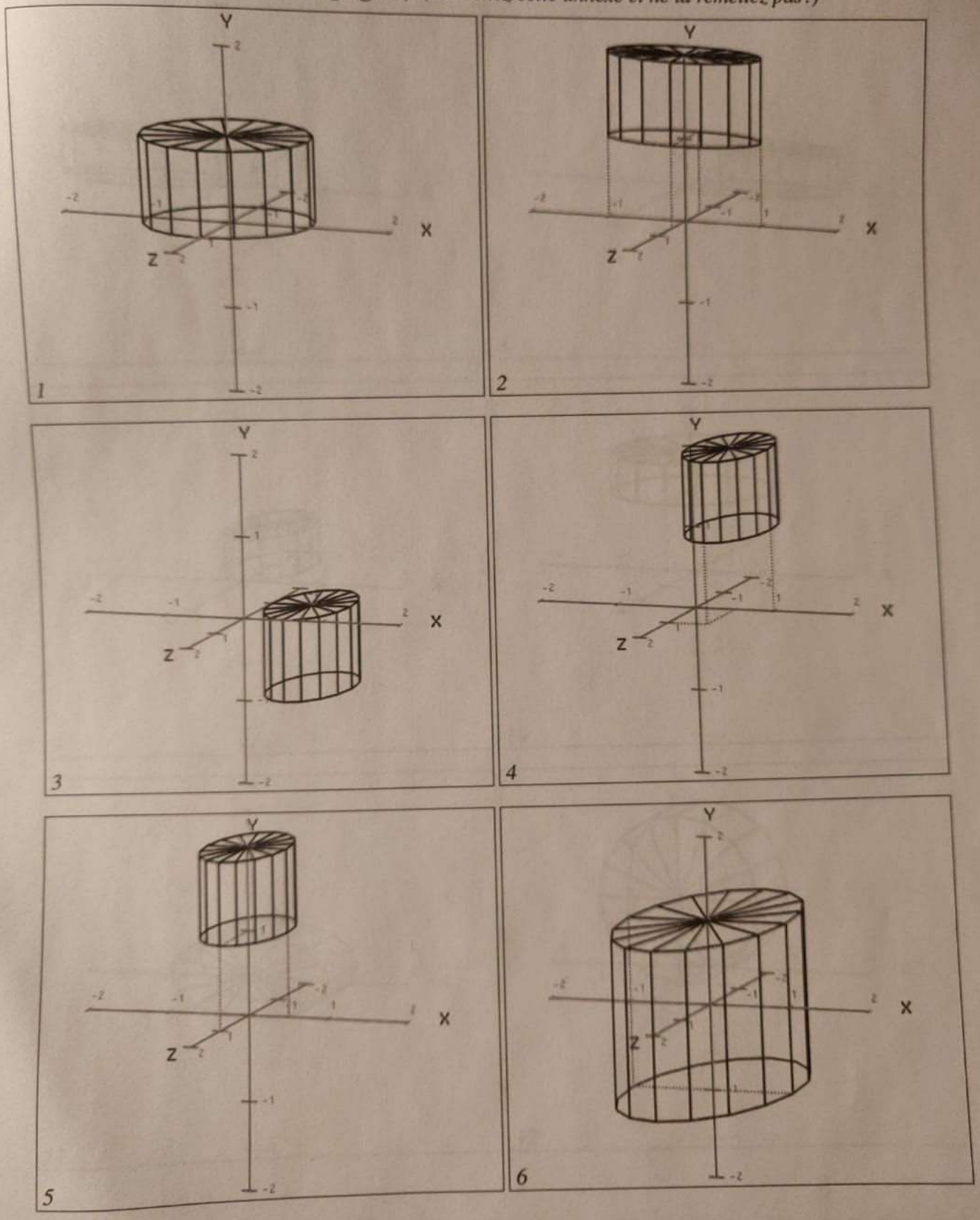
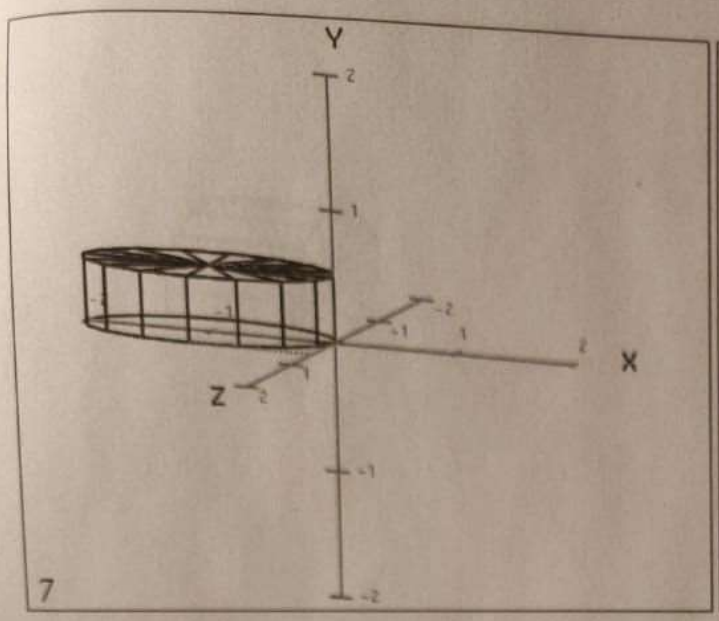
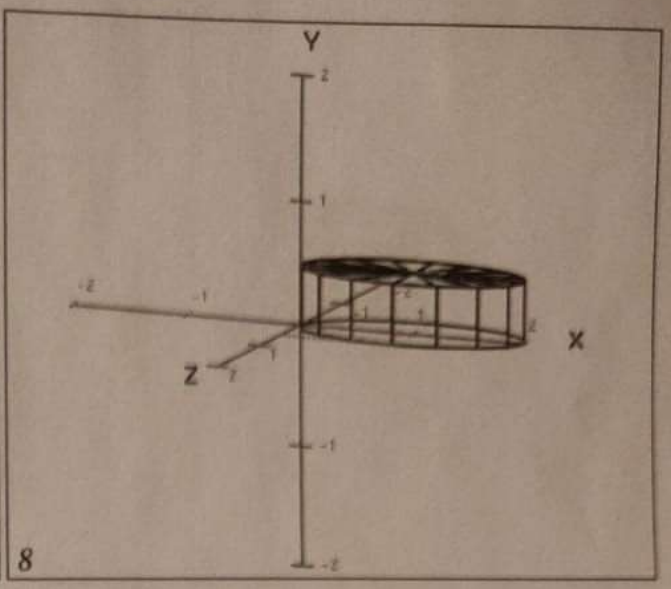
Annexe A (pour question 2, page 3) *(Détachez cette annexe et ne la remettez pas !)*

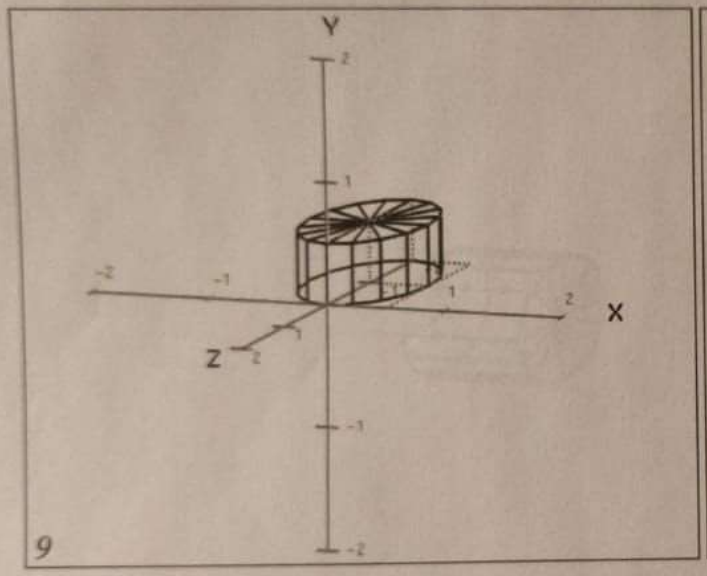




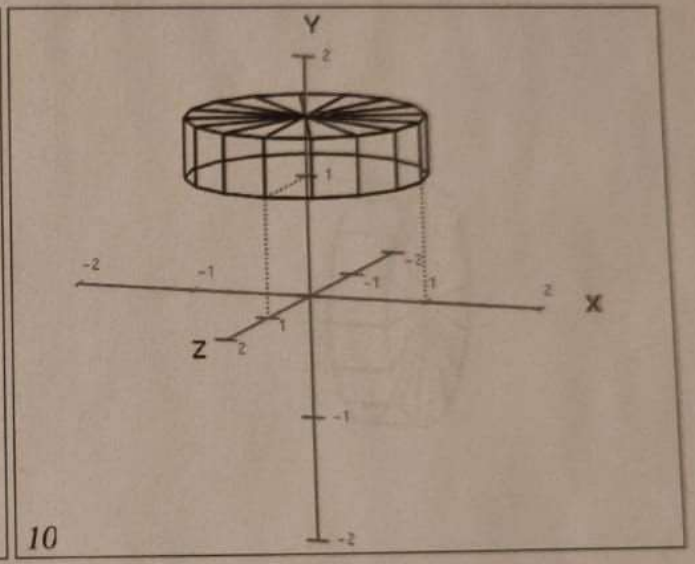
7



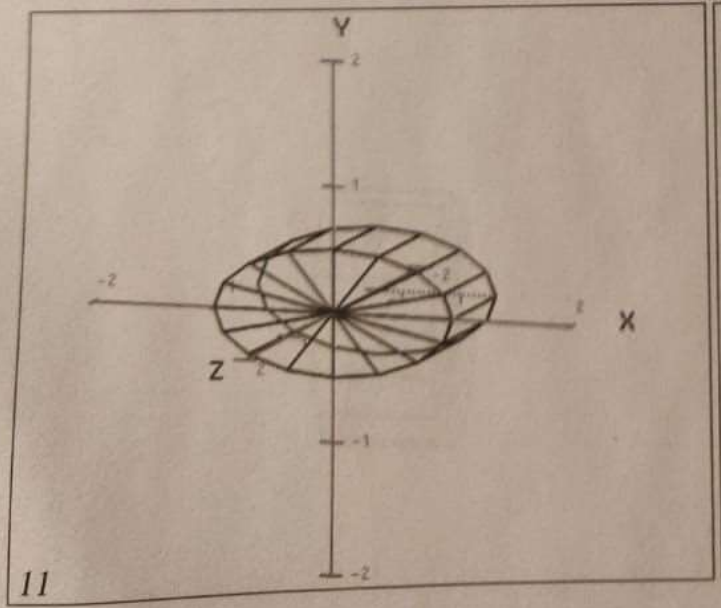
8



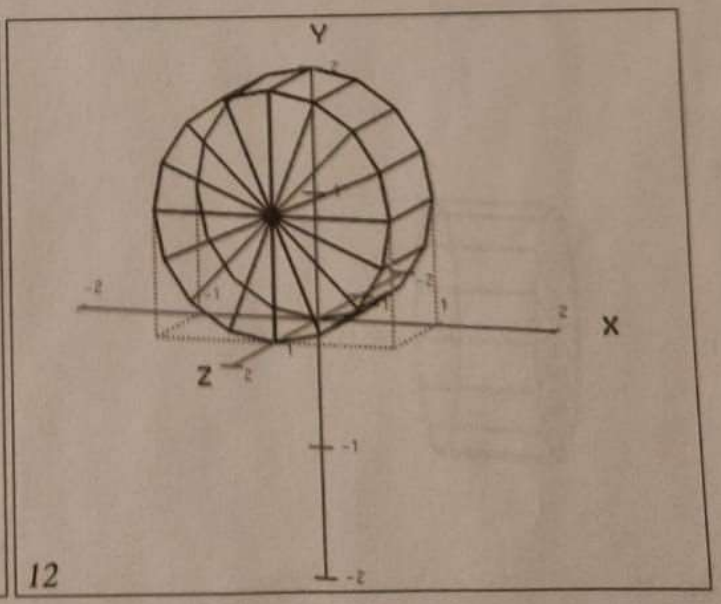
9



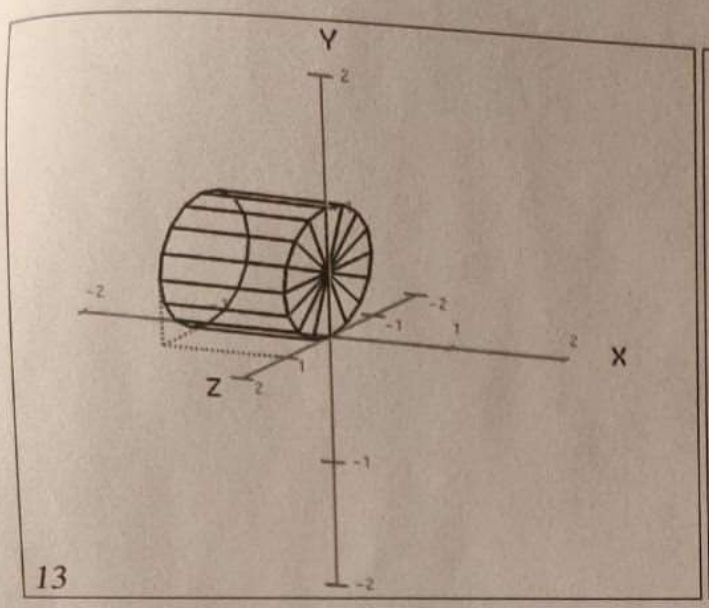
10



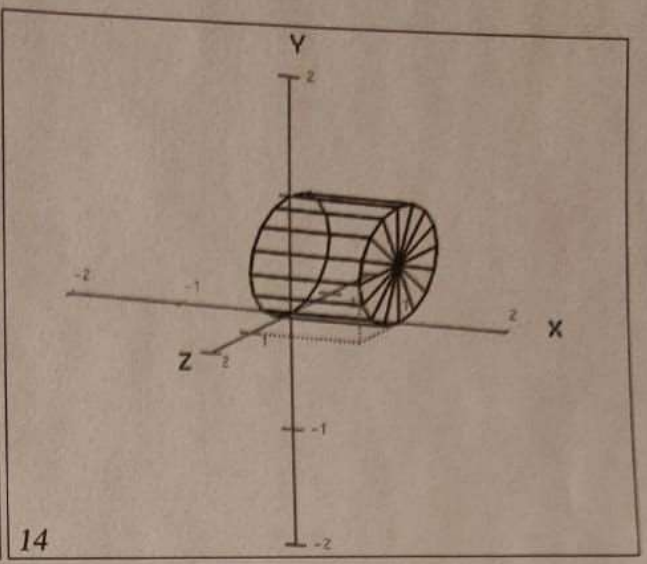
11



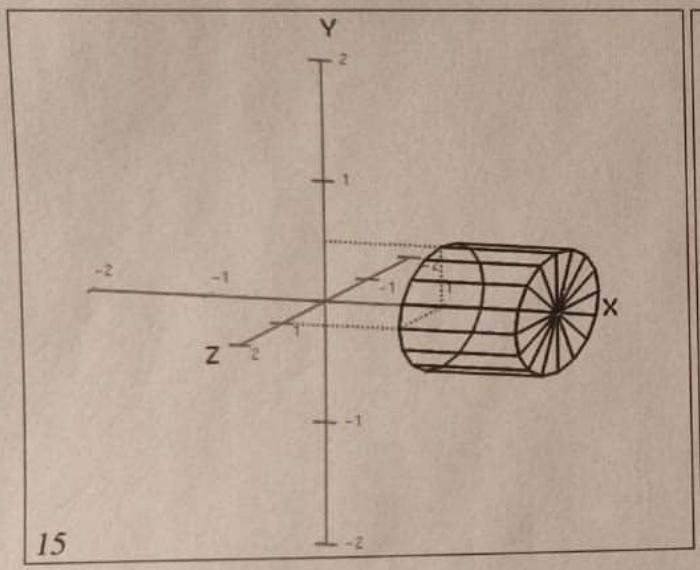
12



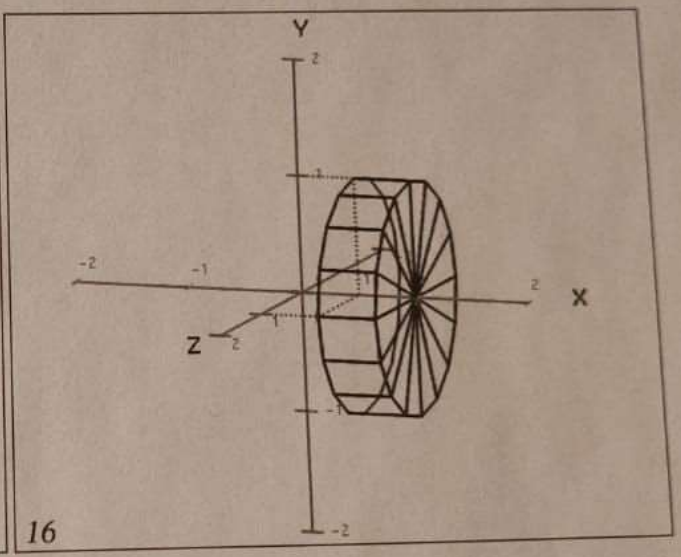
13



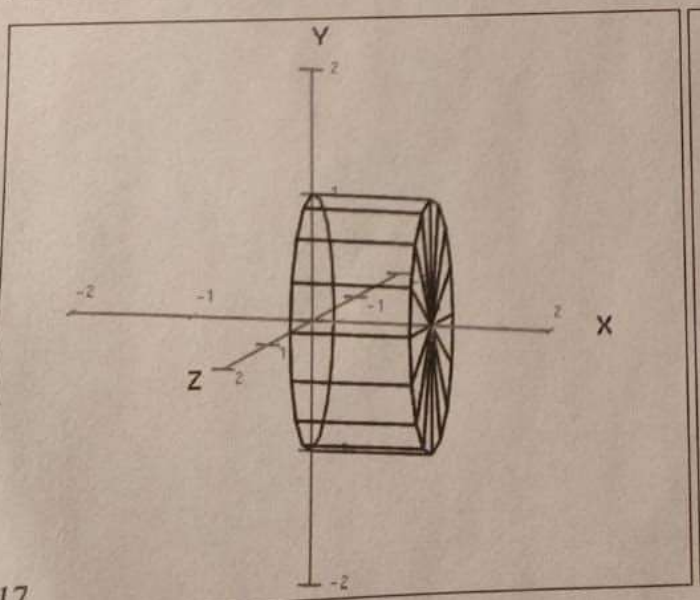
14



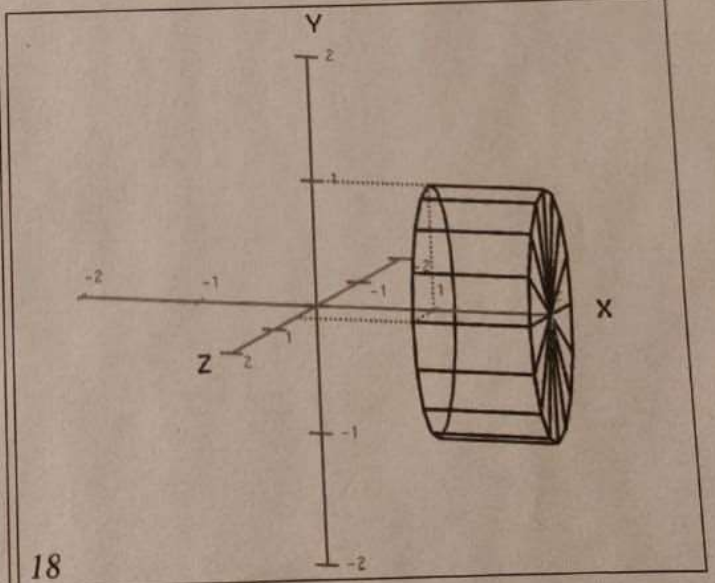
15



16



17



18

Département de génie informatique et génie logiciel

325/40

INF2705: Infographie (Automne 2019)
Contrôle périodique

Notes:

- Toute documentation interdite, calculatrice programmable et ordinateur portatif interdits. Calculatrice non programmable permise.
- Cet examen comprend 4 questions sur 11 pages pour un total de 40 points.

⇒ *Répondez aux questions directement sur le questionnaire.*

Q1 : 7
Q2 : 8
Q3 : 10
Q4 : 7.5

Question 1 Concepts théoriques en infographie [10 points]

a) [1 point] Lorsque la carte graphique traite les primitives graphiques pour l'affichage, qu'est-ce qui est produit par l'étape du *tramage* ?

1 Des fragments sont produit lors du tramage. ✓

b) [2 points] On fait souvent mention de la face avant ou de la face arrière (`GL_FRONT` ou `GL_BACK`) d'une primitive OpenGL. Quelle convention utilise-t-on pour déterminer la face avant ou la face arrière d'une primitive ?

2 La face avant est celle dont les sommets sont ~~qui est dans le sens horaire~~ anti-horaire (de la main droite) ✓

c) [2 points] Une mise à l'échelle est fondamentalement une opération de *multiplication*, tandis qu'une translation est, au contraire, une opération d'*addition*. Pourtant, nos applications graphiques avec OpenGL cumulent toutes ces opérations dans une seule matrice de modélisation qui n'est utilisée, elle, que pour *multiplier* des coordonnées. Comment est-ce possible ? Expliquez succinctement l'artifice mathématique utilisé.

2 On octroie une coordonnée homogène à vecteur qu'on veut translater. Ainsi, on aura une matrice 4x4 qui lors de la multiplication avec le vecteur position donnera l'équation de translation

d) [3 points] Les deux fonctions ci-dessous permettent de définir une projection perspective, mais n'utilisent pas les mêmes paramètres.

```
void Frustum( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
              GLdouble near, GLdouble far );

void Perspective( GLdouble fovy, GLdouble aspect,
                  GLdouble near, GLdouble far );
```

(`fovy` *est un angle en y;* `aspect` *est un rapport dx/dy*)

Considérez les paramètres de cet énoncé : `Perspective( 90.0, 0.5, 10.0, 20.0 );`.

Est-il possible de définir exactement le même volume de projection en utilisant la fonction `Frustum()` ? Si oui, écrivez les paramètres dans l'appel ci-dessous. Sinon, dites pourquoi.

`Frustum(` 10.0 , 20.0 , 20.0 , 40.0 , 10.0 , 20.0 `);`

$\frac{x}{z} = \tan 45 \Rightarrow x = 2z$

$\frac{x}{z} = z$

e) [2 points] Une sphère est tracée dans une application graphique 3D. Cette sphère est entièrement contenue dans un volume de visualisation qui respecte le rapport d'aspect et elle est entièrement visible dans la fenêtre à l'écran.

i) Si cette application utilise une projection *orthographique*, est-ce que la silhouette de la sphère à l'écran sera *toujours* un cercle, quelle que soit sa position à l'écran ? Expliquez pourquoi.

Oui, car la projection orthographique s'effectue parallèlement aux axes

ii) Si cette application utilise une projection *perspective*, est-ce que la silhouette de la sphère à l'écran sera *toujours* un cercle, quelle que soit sa position à l'écran ? Expliquez pourquoi.

Oui, car lors de la projection perspective, la silhouette se déforme pour répondre aux besoins de la perspective mais conserve le même aspect (forme)

Question 2 Transformations affines [8 points]

La fonction ci-dessous produit la figure 1 de l'annexe A (un cylindre de rayon 1 et de hauteur 1).

```
void afficher_exemple( )
{
   afficherObjet( );
}
```

Dans chacune des sous-questions suivantes, inscrivez quelle figure de l'annexe A est le résultat visuel de la fonction donnée ou, inscrivez « Aucune » si aucune figure n'y correspond.

```
void afficher_i( )
{
  matrModel.Rotate( 90, 0, 1, 0 );
  matrModel.Translate( 0, 0, -1 );
  matrModel.Scale( .5, .5, 1 );
  afficherObjet();
}
```

i) produit la figure 7 ✓

```
void afficher_ii( )
{
  matrModel.Rotate( 90, 0, 1, 0 );
  matrModel.Rotate( 90, 1, 0, 0 );
  matrModel.Translate( 0, 1, 0 );
  matrModel.Scale( 1, 1, .5 );
  afficherObjet();
}
```

ii) produit la figure 15 ✓

```
void afficher_iii( )
{
  matrModel.Rotate( 90, 0, 1, 0 );
  matrModel.Translate( 0, 1, 1 );
  matrModel.Scale( 1, 1, 2 );
  afficherObjet();
}
```

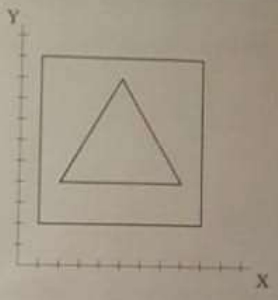
iii) produit la figure Aucun ✓

```
void afficher_iv( )
{
  matrModel.Rotate( 90, 1, 0, 0 );
  matrModel.Translate( 0, 0, -1 );
  afficherObjet();
}
```

iv) produit la figure 12 ✓

Question 3 Visualisation 3D [14 points]

Considérez les énoncés OpenGL ci-dessous utilisés pour afficher un triangle plein et un quadrilatère plein, tel qu'illustré dans le schéma ci-contre.




```
void FenetreTP::initialiser( )
{
   // les coordonnées du quadrilatère
   GLfloat cooq[] = { 1.0,  2.0, 2.0,     // sommet 1 carré
                      9.0,  2.0, 2.0,     // sommet 2 carré
                      9.0, 10.0, 2.0,     // sommet 3 carré
                      1.0, 10.0, 2.0 };   // sommet 4 carré
   glBindVertexArray( vao[0] );
   glBindBuffer( ... );
   glBufferData( ... );
   ...

   // les coordonnées du triangle
   GLfloat coot[] = { 2.0,  4.0, 0.0,     // sommet 1 triangle
                      8.0,  4.0, 0.0,     // sommet 2 triangle
                      5.0,  9.0, 0.0 };   // sommet 3 triangle
   glBindVertexArray( vao[1] );
   glBindBuffer( ... );
   glBufferData( ... );
   ...
}


void FenetreTP::afficherScene( )
{
   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
   glEnable( GL_DEPTH_TEST );
   glDepthFunc( GL_LESS ); // la valeur de défaut

   matrModel.LoadIdentity();

   // void LookAt( obsX,obsY,obsZ, versX,versY,versZ, upX,upY,upZ );
   matrVisu.LookAt( 0.0, 0.0, 6.0,  0.0, 0.0, 0.0,  0.0, 1.0, 0.0 );

   // void Ortho( gauche,droite, bas,haut, planAvant,planArriere );
   matrProj.Ortho( -2.0, 12.0, -2.0, 12.0, 0.0, 8.0 );

   // informer les nuanceurs des matrices courantes avec glUniformMatrix4fv( )
   ...

   // tracer la scène
   glBindVertexArray( vao[0] );
   tracerQuadrilatere( );  // tracer le quadrilatère
   glBindVertexArray( vao[1] );
   tracerTriangle( );      // tracer le triangle
}
```

(suite page suivante)

a) **[2 points]** Quelle est la valeur de Z de la position de la caméra dans le repère de *visualisation*? Expliquez succinctement votre réponse.

$z = 0$ car dans le repere de visualisation tous les objets sont positionnés par rapport à la caméra

b) **[2 points]** Quelle est la valeur de Z des fragments du quadrilatère dans le repère de *visualisation*, c'est-à-dire après la transformation de *visualisation*? Expliquez succinctement vos calculs.

$z = -$ (distance entre caméra et quadrilatere) $= -4$

$z = -4$

$z < 0$ car le quadrilatere se trouve en $\underset{\text{partie négative}}{\text{l'axe}}$ des z

c) **[2 points]** Quelle est la valeur de Z des fragments du triangle dans le repère de *visualisation*, c'est-à-dire après la transformation de *visualisation*? Expliquez succinctement vos calculs.

$z = -$ (distance entre caméra et le triangle) $= -6$

$z = -6$ car le triangle se trouve a une distance de 6 de la caméra et que les objets regardés sont dans la partie negative de l'axe z

d) **[2 points]** Quelle est la valeur de Z (entre -1.0 et 1.0) des fragments du quadrilatère dans le repère de *projection*, c'est-à-dire après la transformation de *projection*? Expliquez succinctement vos calculs.

$$\frac{z-0}{(1.0+1.0)} = \frac{-4-0}{8.0-0} \Rightarrow \frac{z}{2} = \frac{-1}{2} \Rightarrow z = -1 \Rightarrow \boxed{z = -1}$$ ✗

e) **[2 points]** Quelle est la valeur de Z (entre -1.0 et 1.0) des fragments du triangle dans le repère de *projection*, c'est-à-dire après la transformation de *projection*? Expliquez succinctement vos calculs.

$$\frac{z-0}{1.0-(-1.0)} = \frac{-6.0}{8.0-0} \Rightarrow \frac{2}{2} = \frac{-3}{4} \Rightarrow z = \frac{-6}{4} = -1.5 \Rightarrow \boxed{z = -1.5}$$ ✗

f) **[2 points]**
i) Écrivez la comparaison faite par le test de profondeur qui est exécuté lors de l'affichage du triangle. (Écrivez la valeur de profondeur Z, la comparaison et l'autre valeur de profondeur Z.)

Valeur de profondeur = -1
Comparaison : $-1.5 < -1$ → Oui alors on écrit dans le Tampon de profondeur
Valeur de profondeur nouvelle = -1.5

1

ii) Est-ce que le triangle sera visible dans le rendu final ?

Oui

1

g) **[2 points]** Dans les énoncés de la page précédente, on affiche le quadrilatère suivi ensuite du triangle. Si on traçait d'abord le triangle et ensuite le quadrilatère, est-ce que le triangle serait alors visible dans le rendu final ? Pourquoi ?

Oui, car les fragments du quadrilatère ne passeraient pas le test de profondeur aux endroits où sont affichés le triangle

2

Question 4 Fragments [8 points]

4.5

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, selon les énoncés OpenGL, on vous demande de donner les valeurs subséquentes qui seront présentes dans les différents tampons.

Note : `void glDepthFunc( GLenum func );`

a)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.3	(0.9, 0.7, 0.5, 0.3)
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)

```
glDisable( GL_STENCIL_TEST );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_GEQUAL ); // " >= ", l'inverse de la valeur par défaut
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	✓

Note :
```
void glStencilFunc( GLenum func, GLint ref, GLuint mask );
void glStencilOp( GLenum sfail, GLenum zfail, GLenum pass );
```

b)

	profondeur (z)	*couleur (r, g, b, a)*	*stencil*
- valeurs des attributs du fragment	0.1	(0.9, 0.7, 0.5, 0.3)	-
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	5

```
glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_LEQUAL, 1, 7 ); // " <= "
glStencilOp( GL_INCR, GL_DECR, GL_REPLACE );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_LEQUAL ); // " <= "
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons	0.1	(0.9, 0.7, 0.5, 0.3)	1

c)

	profondeur (z)	*couleur (r, g, b, a)*	*stencil*
- valeurs des attributs du fragment	0.1	(0.9, 0.7, 0.5, 0.3)	-
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	5

```
glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_LESS, 1, 7 ); // " < "
glStencilOp( GL_INCR, GL_DECR, GL_REPLACE );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_GREATER ); // " > "
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	4

d)

	profondeur (z)	*couleur (r, g, b, a)*	*stencil*
- valeurs des attributs du fragment	0.1	(0.9, 0.7, 0.5, 0.3)	-
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	5

```
glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_GREATER, 1, 7 ); // " > "
glStencilOp( GL_INCR, GL_DECR, GL_REPLACE );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_LESS ); // " < "
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	6

Note : `void glBlendFunc( GLenum sfactor, GLenum dfactor );`

e)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.1	(0.9, 0.7, 0.5, 0.3)
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)

```
glDisable( GL_STENCIL_TEST );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_ALWAYS ); // " toujours "
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
```

- valeurs subséquentes dans les tampons | 0.1 | (0.69, 0.63, 0.57, 0.51)

f)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.1	(0.9, 0.7, 0.5, 0.3)
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)

```
glDisable( GL_STENCIL_TEST );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_LESS ); // " < "
glEnable( GL_BLEND );
glBlendFunc( GL_ONE, GL_ONE_MINUS_SRC_ALPHA );
```

- valeurs subséquentes dans les tampons | 0.1 | (1.32, 1.12, 0.92, 0.72)

g)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.1	(0.9, 0.7, 0.5, 0.3)
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)

```
glDisable( GL_STENCIL_TEST );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_NEVER ); // " jamais "
glEnable( GL_BLEND );
glBlendFunc( GL_ONE, GL_ZERO );
```

- valeurs subséquentes dans les tampons | 0.7 | (0.6, 0.6, 0.6, 0.6)

Cet examen comprend 4 questions sur 11 pages pour un total de 40 points.