2011 Automne intra

Question 1 Notions de base

a) En classe, nous avons vu l'algorithme du point milieu.

i) À quoi sert cet algorithme ?

ii) Au milieu de quoi est ce « point milieu » ?

[3 points]

b) Qu'est-ce qu'un anaglyphe ? **[2 points]**

c) L'oeil humain est-il plus sensible aux variations d'*intensité* de la lumière ou aux variations de *fréquence* de la lumière ? Pourquoi ? **[2 points]**

d) Nous avons vu quelques modèles de couleur en classe.

i) Identifiez *trois* modèles différents en nommant les axes du système de coordonnées de chaque modèle.

ii) Pour chacun, dessinez le *volume* représentant l'espace 3D des couleurs définies par le modèle en indiquant la position des axes.

[4 points]

Question 2 Librairies graphiques et OpenGL

a) Nommez *deux* systèmes de librairies graphiques antérieurs à OpenGL. **[2 points]**
(Notez que Direct3D [1995] est plus récent qu'OpenGL [1992], ce n'est donc pas une réponse possible !)

b) Avec les librairies graphiques, on parle souvent de sommets, de fragments et de pixels.

i) Distinguez les concepts de sommet et de fragment.

ii) Comment sont créés les fragments ?

[3 points]

c) Une sphère pleine est tracée en OpenGL avec une projection perspective et elle est entièrement visible à l'écran.

i) Combien de point(s) de fuite peut-on facilement identifier s'il n'y a que cette sphère ? Pourquoi ?

ii) Quelle serait votre réponse si, au lieu d'une sphère, la scène contenait plutôt un cylindre ? Pourquoi ?

[2 points]

d) On affiche une scène 3D à l'écran en utilisation la fonction glFrustum d'OpenGL :

```
void glFrustum( GLdouble Gauche, GLdouble Droit, GLdouble Bas, GLdouble Haut,
                GLdouble PlanAvant, GLdouble PlanArrière );
```

i) Nommez deux conséquences d'une modification à la valeur du paramètre `PlanArrière` sur l'image vue à l'écran. **[1 point]**

ii) Nommez deux conséquences d'une modification à la valeur du paramètre `PlanAvant` sur l'image vue à l'écran. **[1 point]**

Question 3 Transformations affines

La librairie GLUT fournit la fonction `glutWireSphere(1.0,15,15)` qui permet de tracer une sphère de rayon 1.0 centrée en $(0, 0, 0)$ tel qu'illustré à la figure 1. En utilisant cette fonction avec ces arguments, ainsi que d'autres fonctions de base d'OpenGL comme `glRotatef()`, `glScalef()` et `glTranslatef()`, écrivez la méthode qui trace l'ellipsoïde de la figure 2 *à la position* et aussi *dans l'orientation* illustrées. Observez l'orientation des lignes en fil de fer !

Notez que le rayon de l'ellipsoïde selon l'axe Y est 2.0, tandis que le rayon selon les axes X et Z est 1.0. (N'utilisez pas de transformations inutiles. Ne faites pas compliqué lorsque vous pouvez faire simple !)

[4 points]

```
void traceEllipsoide( )
{
  ...
  glutWireSphere( 1.0, 15, 15 );
  ...
}
```

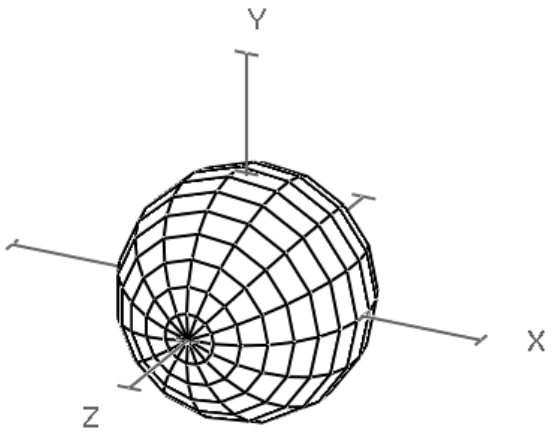


Figure 1 : sphère

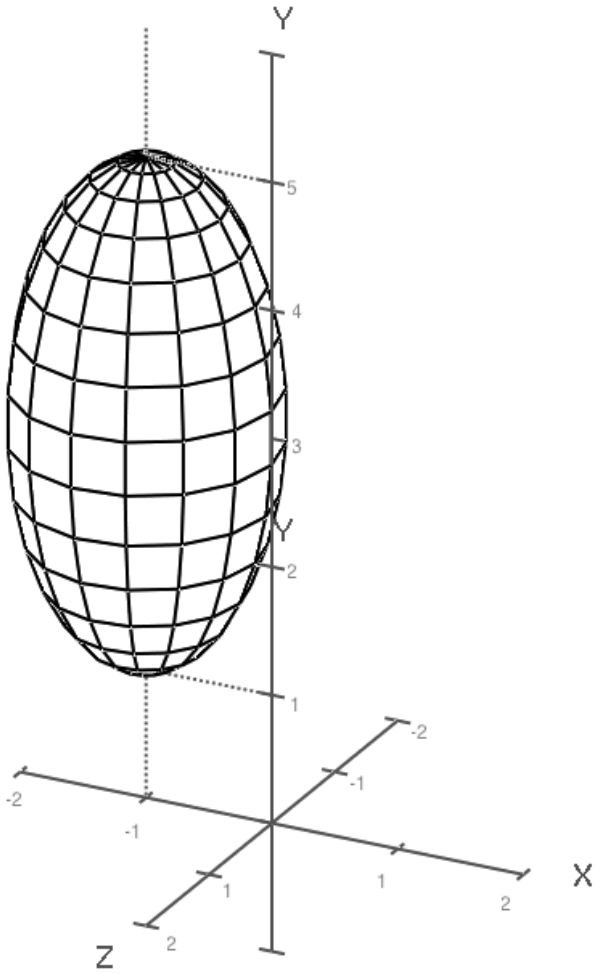
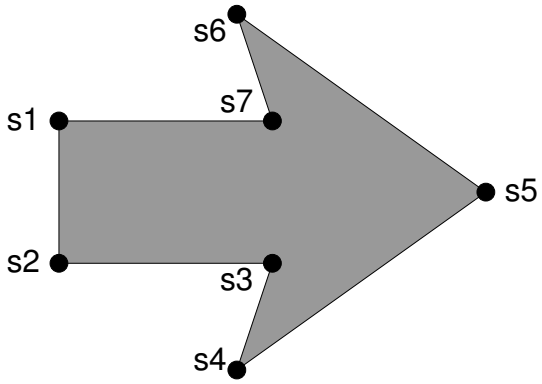


Figure 2 : ellipsoïde

Question 4 Primitives OpenGL

Considérez cette forme pleine pouvant être tracée par différentes primitives OpenGL :

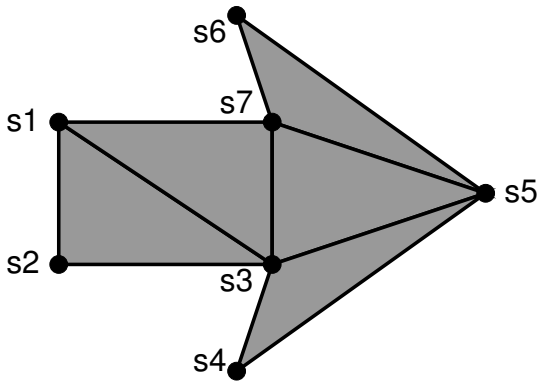


Dans les sous-questions suivantes, vous devez écrire les énoncés `glBegin()`, `glVertex*()` et `glEnd()` nécessaires pour tracer exactement et correctement la forme ci-dessus.

- Vous ne devez pas utiliser d'autres sommets que ceux existants (s1 à s7).
- Vous ne devez pas remplir ou tracer par-dessus ce qui est déjà rempli ou tracé.
- Vous devez utiliser un nombre minimal de paires `glBegin() ... glEnd()`, mais vous devez bien sûr les répéter lorsque nécessaire.
- Pour faire plus court dans la réponse, utilisez simplement « `S1` » au lieu de « `glVertex*(s1);` ».

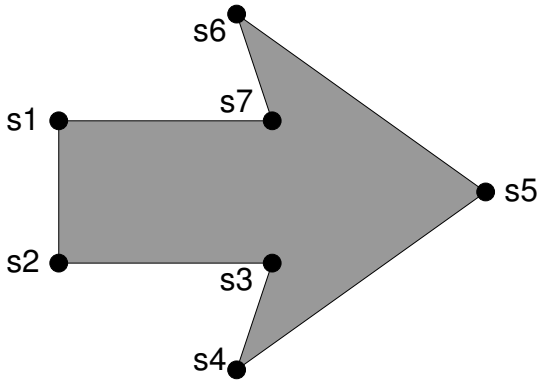
EXEMPLE :

i) **Tracez** les triangles et ii) **écrivez** les énoncés OpenGL si on utilise la primitive `GL_TRIANGLES`.

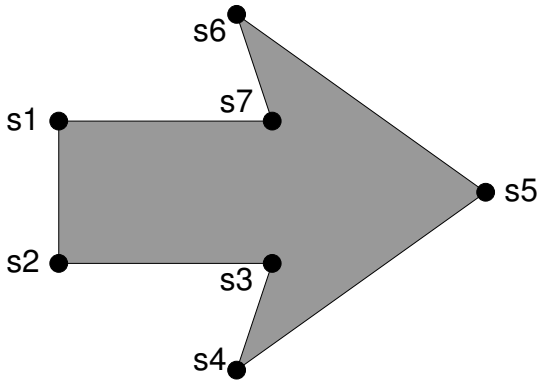



```
glBegin( GL_TRIANGLES );
  s1 s2 s3
  s1 s3 s7
  s3 s4 s5
  s3 s5 s7
  s5 s6 s7
glEnd( );
```

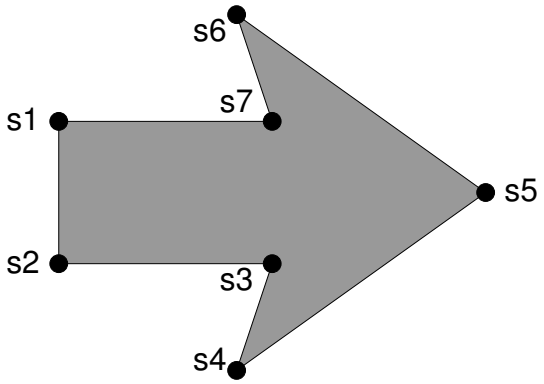
a) i) **Tracez** les quadrilatères et ii) **écrivez** les énoncés OpenGL si on utilise la primitive `GL_QUADS`. **[2 points]**



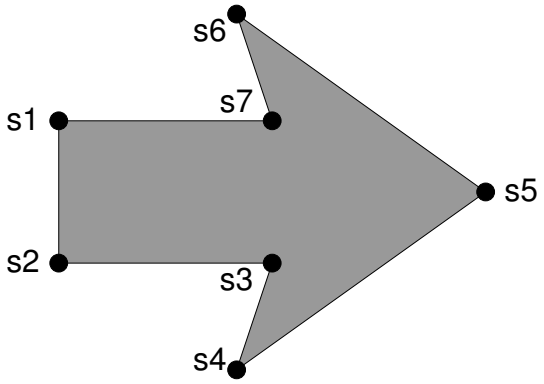
b) i) **Tracez** les quadrilatères et ii) **écrivez** les énoncés OpenGL si on utilise la primitive `GL_QUAD_STRIP`. **[2 points]**



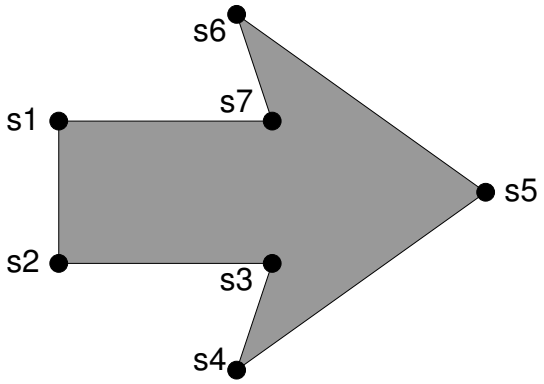
c) i) **Tracez** les triangles et ii) **écrivez** les énoncés OpenGL si on utilise la primitive `GL_TRIANGLE_STRIP`. **[2 points]**



d) i) **Tracez** les triangles et ii) **écrivez** les énoncés OpenGL si on utilise la primitive `GL_TRIANGLE_FAN`. **[2 points]**



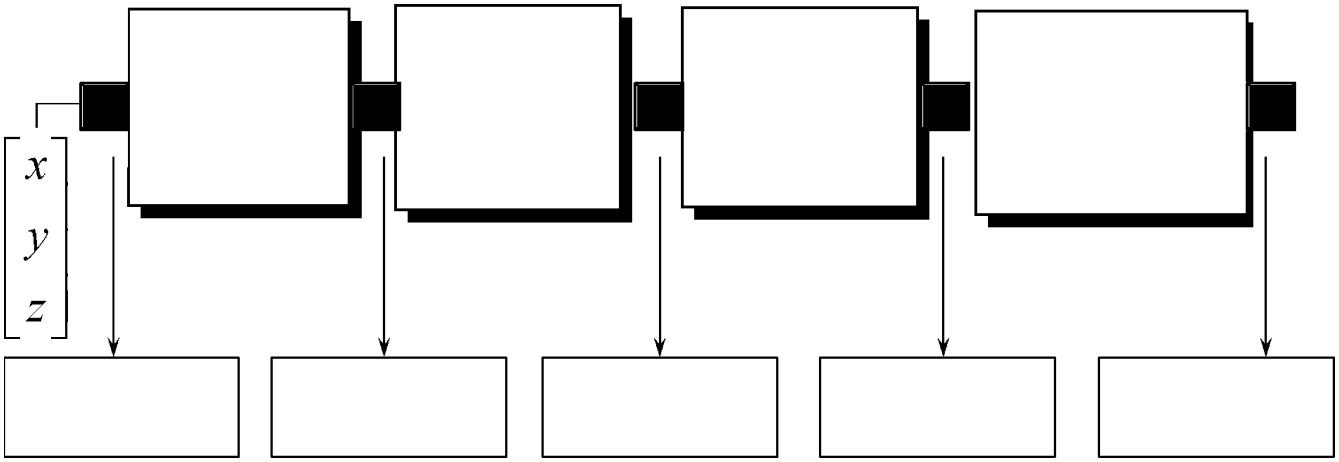
e) i) **Tracez** les polygones et ii) **écrivez** les énoncés OpenGL si on utilise la primitive `GL_POLYGON`. **[2 points]**



f) Pour l'affichage de cette forme, quelle est la primitive la plus efficace ? Pourquoi ? **[1 point]**

Question 5 Pipeline graphique

a) La figure suivante illustre le pipeline OpenGL de la transformation des sommets.



i) Pour chacune des neuf cases ci-dessus, choisissez l'expression la plus appropriée dans la liste suivante et inscrivez la lettre correspondante dans la case. **[4 points]**
(Une seule expression par case. Il n'est évidemment pas nécessaire d'utiliser toutes les expressions !)

- **a** : système de coordonnées
- **b** : coordonnées d'affichage
- **c** : coordonnées d'appareil normalisées
- **d** : coordonnées d'appareil
- **e** : coordonnées d'objet
- **f** : coordonnées de primitive
- **g** : coordonnées de couleur
- **h** : coordonnées de découpage
- **i** : coordonnées polaires
- **j** : coordonnées de stencil
- **k** : coordonnées de visualisation
- **l** : coordonnées normalisées
- **m** : coordonnées universelles
- **n** : matrice de modélisation
- **o** : matrice de profondeur
- **p** : matrice de projection
- **q** : matrice de forme
- **r** : matrice de stencil
- **s** : mémoire de trame
- **t** : transformation de clôture
- **u** : transformation de conversion
- **v** : transformation de normalisation
- **w** : transformation de couleur
- **x** : tampon de couleur
- **y** : tampon de stencil
- **z** : tampon de profondeur

ii) Pour chacune des quatre expressions suivantes, choisissez la case la plus appropriée et inscrivez-y la lettre correspondante. **[4 points]**

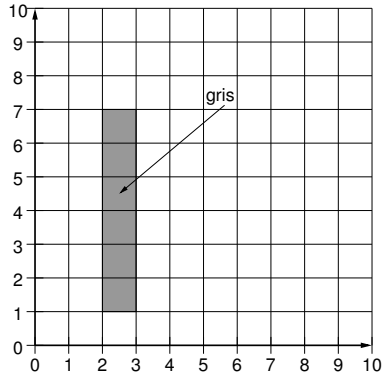
- **A** : volume de visualisation
- **B** : caméra synthétique
- **C** : point de fuite
- **D** : Cohen-Sutherland

iii) Pour chacun des sept groupes de fonctions suivants, choisissez la ou les cases les plus appropriées (selon ce que chacun de ces groupes peut contrôler) et inscrivez-y le chiffre correspondant. **[4 points]**

- **1** : `glFrustum()`, `gluPerspective()`
- **2** : `glLoadIdentity()`
- **3** : `glLoadMatrix()`, `glMultMatrix()`
- **4** : `glOrtho()`, `gluOrtho2D()`
- **5** : `glTranslate()`, `glScale()`, `glRotate()`
- **6** : `glViewport()`
- **7** : `gluLookAt()`

Question 6 Opérations sur les fragments

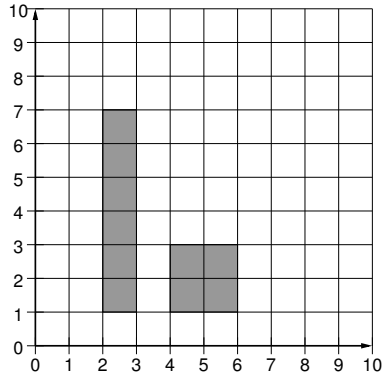
Considérez la méthode OpenGL présentée à l'annexe A. On vous demande ici dessiner le contenu de la mémoire de trame et du stencil aux différents jalons identifiés (a), (b), (c), (d) et (e) dans cette méthode. Par exemple, à la ligne notée « EXEMPLE ... », la mémoire de trame contient ceci :



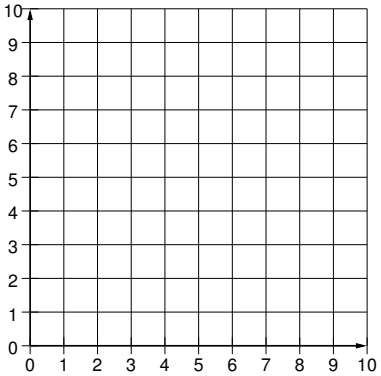
mémoire de trame à la ligne EXEMPLE

Notes :
- L'affichage est de bonne résolution et on peut donc considérer les trames comme des régions continues.
- Dans vos réponses pour la mémoire de trame, assurez-vous d'indiquer la couleur de chaque région.
- Dans vos réponses pour le stencil, inscrivez la valeur stockée dans le stencil pour chaque région.
- À moins d'avis contraire, le *brouillon* à droite est un brouillon qui ne sera pas corrigé.

a) Dessinez le contenu de la mémoire de trame. **[2 points]**

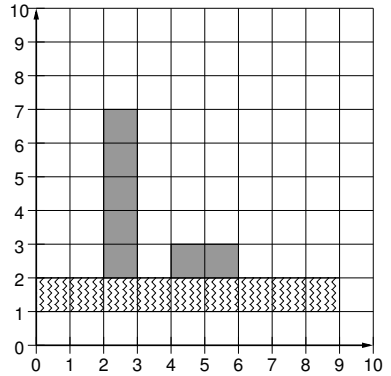


mémoire de trame à ligne (**a)**

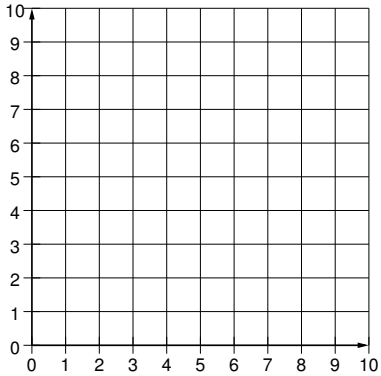


(brouillon)

b) Dessinez le contenu de la mémoire de trame. **[2 points]**

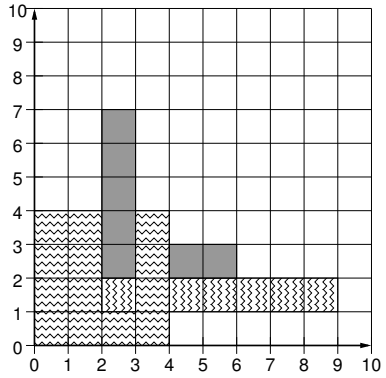


mémoire de trame à ligne (**b)**

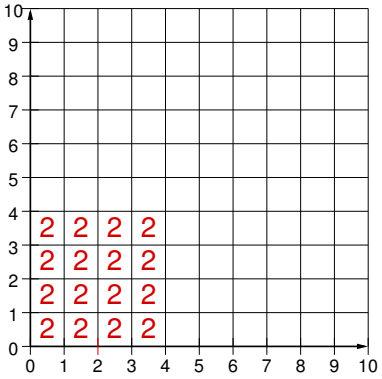


(brouillon)

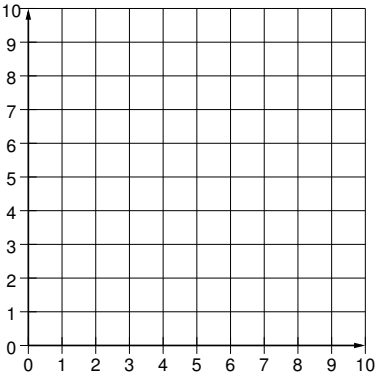
c) Dessinez le contenu de la mémoire de trame et le contenu du stencil. **[3 points]**



mémoire de trame à ligne (**c)**

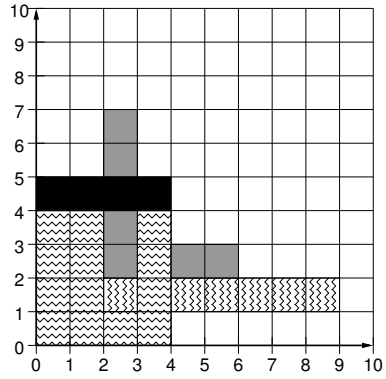


stencil à ligne (**c)**

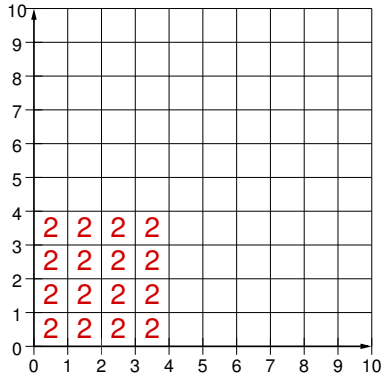


(brouillon)

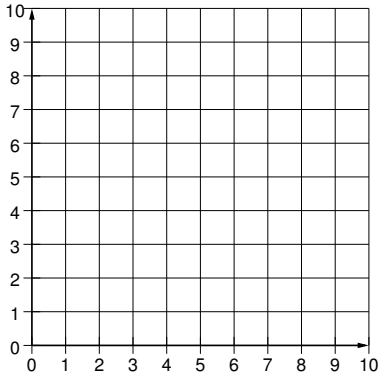
d) Dessinez le contenu de la mémoire de trame et le contenu du stencil. **[3 points]**



mémoire de trame à ligne (**d)**

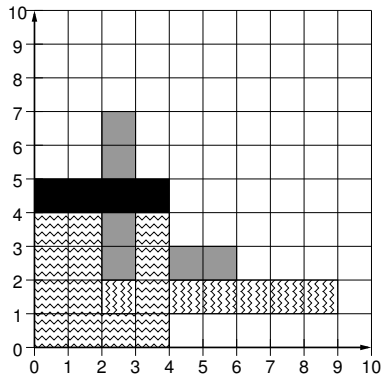


stencil à ligne (**d)**

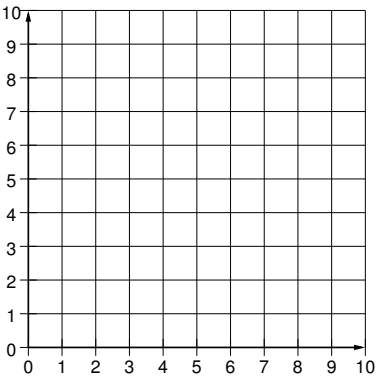


(brouillon)

e) Dessinez le contenu de la mémoire de trame. **[3 points]**



mémoire de trame à ligne (**e)**



(brouillon)

Annexe A - pour la question 6 (page 8)

(Vous pouvez détacher cette annexe et ne pas la remettre.)

```
void Scene::dessine( )
{
   // initialiser le pipeline graphique
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity( );
   glOrtho( 0, 10,  0, 10,  0, 10 );
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity( );
   gluLookAt( 0, 0, 10,  0, 0, 0,  0, 1, 0 );

   // initialiser certaines variables d'état
   // void glBlendFunc( GLenum sourcefactor, GLenum destfactor )
   glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
   glDisable( GL_BLEND );
   glEnable( GL_DEPTH_TEST );
   glClearColor( 1.0, 1.0, 1.0, 1.0 );     // blanc

   // initialiser les différents tampons et tracer un premier rectangle
   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
   glBegin( GL_QUADS ); {
      glColor4f( 0.5, 0.5, 0.5, 1.0 );
      glVertex3i( 3, 1, 6 );
      glVertex3i( 2, 1, 6 );
      glVertex3i( 2, 7, 6 );
      glVertex3i( 3, 7, 6 );
   } glEnd();

   // EXEMPLE DU DESSIN DU CONTENU LA MÉMOIRE DE TRAME

   glEnable( GL_BLEND );
   glBegin( GL_QUADS ); {
      glColor4f( 0.0, 0.0, 0.0, 0.5 );
      glVertex3i( 6, 1, 5 );
      glVertex3i( 6, 3, 5 );
      glVertex3i( 4, 3, 5 );
      glVertex3i( 4, 1, 5 );
   } glEnd();

   // (a) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME

   glDisable( GL_DEPTH_TEST );
   glBegin( GL_QUADS ); {
      glColor3f( 1.0, 1.0, 1.0 );
      glVertex3i( 0, 1, 4 );
      glVertex3i( 0, 2, 4 );
      glVertex3i( 9, 2, 4 );
      glVertex3i( 9, 1, 4 );
   } glEnd();
   glEnable( GL_DEPTH_TEST );

   // (b) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME
```

```
   // void glStencilFunc( GLenum func, GLint ref, GLuint mask )
   // void glStencilOp( GLenum fail, GLenum zfail, GLenum zpass )

   glEnable( GL_STENCIL_TEST );
   glStencilFunc( GL_ALWAYS, 2, 2 );
   glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );

   glBegin( GL_QUADS ); {
      glColor3f( 1.0, 1.0, 1.0 );
      glVertex3i( 0, 0, 3 );
      glVertex3i( 4, 0, 3 );
      glVertex3i( 4, 4, 3 );
      glVertex3i( 0, 4, 3 );
   } glEnd();

   // (c) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME ET DU STENCIL

   glStencilFunc( GL_NOTEQUAL, 2, 2 ) ;
   glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );

   glBegin( GL_QUADS ); {
      glColor3f( 0.0, 0.0, 0.0 );
      glVertex3i( 0, 0, 7 );
      glVertex3i( 4, 0, 7 );
      glVertex3i( 4, 5, 7 );
      glVertex3i( 0, 5, 7 );
   } glEnd();

   glDisable( GL_STENCIL_TEST );

   // (d) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME ET DU STENCIL

   glDisable( GL_DEPTH_TEST );

   glBegin( GL_QUADS ); {
      glColor4f( 0.5, 0.5, 0.5, 0.0 );
      glVertex3i( 0, 3, 9 );
      glVertex3i( 3, 3, 9 );
      glVertex3i( 3, 5, 9 );
      glVertex3i( 0, 5, 9 );
   } glEnd();

   glDisable( GL_BLEND );

   // (e) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME

   glutSwapBuffers();
   return;
}
```

2011 Hiver intra

Question 1 Notions de base

a) i) Pour les transformations géométriques 3D, pourquoi utilise-t-on une matrice 4X4 au lieu d'une 3X3 ? ii) Comment se nomme le système de coordonnées ainsi créé ? **[2 points]**

b) Si la matrice M ci-contre est utilisée comme matrice de transformation affine dans le pipeline graphique pour multiplier les sommets des primitives OpenGL, expliquez ce qui arrive si le terme $m16$ est nul ? **[2 points]**

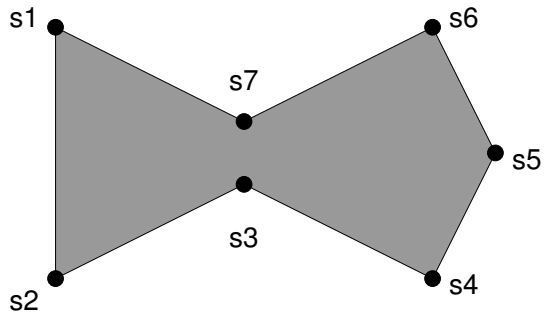
$$M = \begin{bmatrix} m1 & m5 & m9 & m13 \\ m2 & m6 & m10 & m14 \\ m3 & m7 & m11 & m15 \\ m4 & m8 & m12 & m16 \end{bmatrix}$$

c) L'oeil humain est-il plus sensible aux variations d'intensité de la lumière ou aux variations de fréquence de la lumière ? Pourquoi ? **[2 points]**

d) Quelle est la différence entre un sommet et un fragment ? Comment sont créés les fragments ? **[3 points]**

Question 2 Primitives OpenGL

Considérez cette forme pleine pouvant être tracée par différentes primitives OpenGL :

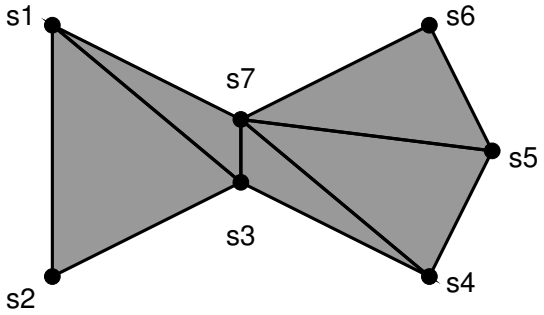


Dans les sous-questions suivantes, on vous demande d'écrire les énoncés `glBegin`, `glVertex2iv` et `glEnd` nécessaires pour tracer exactement et correctement la forme ci-dessus.
Pour faire plus court dans la réponse, utilisez « `S1` » au lieu de « `glVertex2iv(s1);` ».

Les règles à respecter sont :
- vous ne devez pas utiliser d'autres sommets que ceux existants (s1 à s7) ;
- vous devez utiliser un nombre minimal de paires `glBegin() ... glEnd()`.
- vous ne devez pas remplir ou tracer par-dessus ce qui est déjà rempli ou tracé.

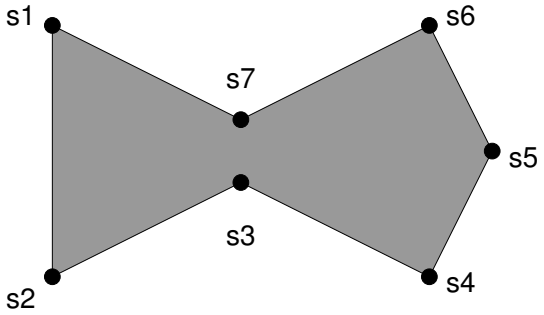
EXEMPLE :

i) Tracez les triangles et ii) donnez les énoncés OpenGL si on utilise la primitive `GL_TRIANGLES` ?

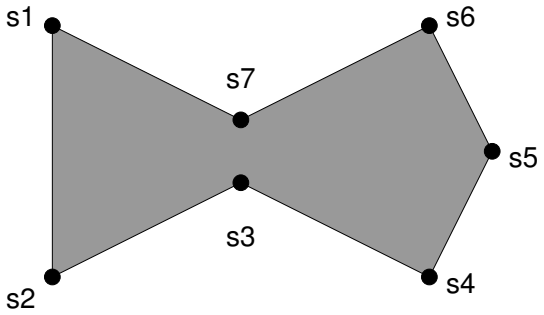



```
glBegin( GL_TRIANGLES );
  s1 s2 s3
  s7 s1 s3
  s3 s4 s7
  s7 s4 s5
  s5 s6 s7
glEnd( );
```

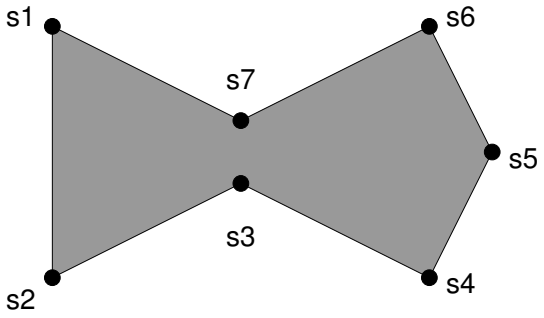
a) i) Tracez les quadrilatères et ii) donnez les énoncés OpenGL si on utilise la primitive `GL_QUADS` ? **[2 points]**



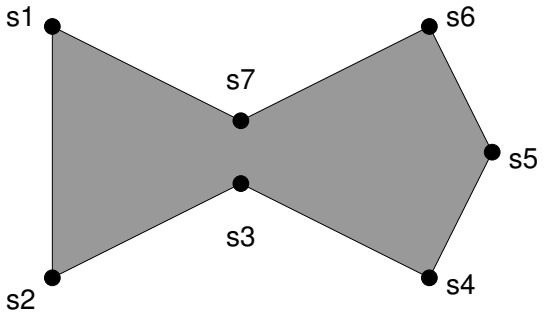
b) i) Tracez les quadrilatères et ii) donnez les énoncés OpenGL si on utilise la primitive `GL_QUAD_STRIP` ? **[2 points]**



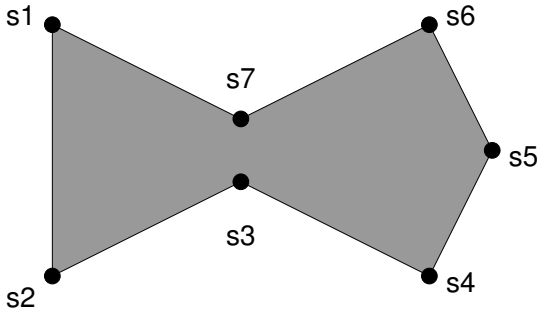
c) i) Tracez les triangles et ii) donnez les énoncés OpenGL si on utilise la primitive `GL_TRIANGLE_STRIP` ? **[2 points]**



d) i) Tracez les triangles et ii) donnez les énoncés OpenGL si on utilise la primitive `GL_TRIANGLE_FAN` ? **[2 points]**



e) i) Tracez les polygones et ii) donnez les énoncés OpenGL si on utilise la primitive `GL_POLYGON` ? **[2 points]**



f) Pour l'affichage de cette forme, quelles sont les primitives les plus efficaces ? Pourquoi ? **[1 point]**

Question 3 Pipeline graphique

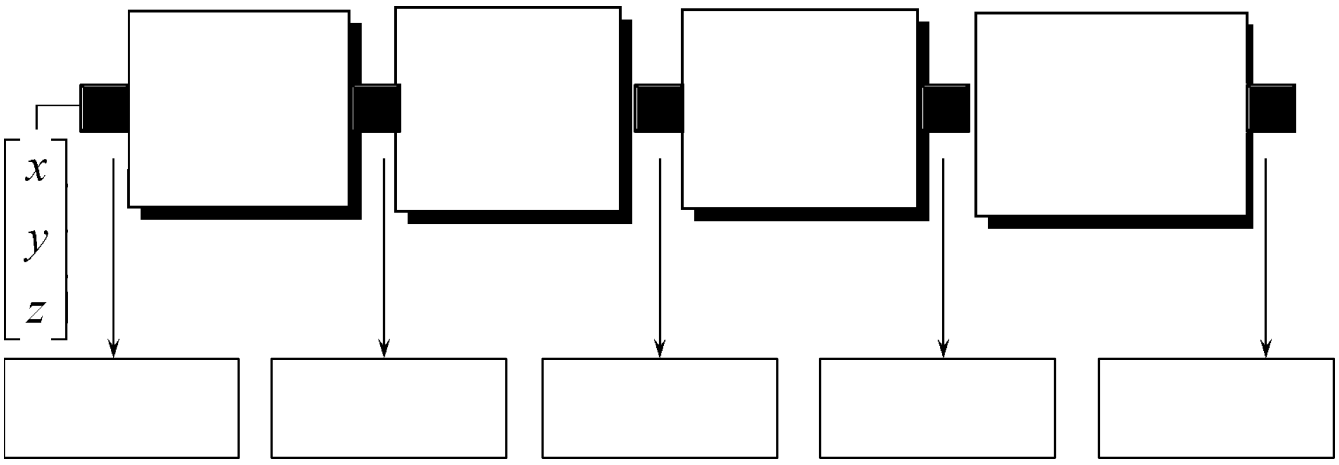
a) Trois sphères de rayon $r = 1$ sont positionnées à ces coordonnées 2D : $(3, 0)$, $(4, -5)$ et $(-2, 2)$. Donnez le coin inférieur gauche (xmin, ymin) et le coin supérieur droit (xmax, ymax) du plus petit rectangle qui englobe ces trois sphères. **[1 point]**

(xmin, ymin) = (_____ , _____) (xmax, ymax) = (_____ , _____)

b) Quel problème peut se produire si on utilise directement la boîte englobante trouvée ci-dessus comme fenêtre virtuelle sans se soucier des valeurs courantes de la clôture ? **[1 point]**

c) Si la clôture a les coordonnées (imin,jmin) = $(100, 0)$ et (imax,jmax) = $(400, 600)$, calculez alors les coordonnées (xmin, ymin) et (xmax, ymax) de la plus petite fenêtre virtuelle permettant d'englober les trois sphères sans produire le problème identifié ci-dessus. **[4 points]**

d) La figure suivante illustre le pipeline OpenGL de la transformation des sommets.



i) Pour chacune des neuf cases, choisissez l'expression la plus appropriée dans la liste suivante et inscrivez cette expression (ou la lettre correspondante) dans la case. **[4 points]**
(Il n'est évidemment pas nécessaire d'utiliser toutes les expressions !)

- `a` : « coordonnées d'affichage »
- `b` : « coordonnées d'appareil normalisées »
- `c` : « coordonnées d'appareil »
- `d` : « coordonnées d'objet »
- `e` : « coordonnées de couleur »
- `f` : « coordonnées de découpage »
- `g` : « coordonnées de stencil »
- `h` : « coordonnées de visualisation »
- `i` : « coordonnées normalisées »
- `j` : « coordonnées universelles »
- `k` : « matrice de modélisation »
- `l` : « matrice de profondeur »
- `m` : « matrice de projection »
- `n` : « matrice de stencil »
- `o` : « mémoire de trame »
- `p` : « tampon de profondeur »
- `q` : « tampon de stencil »
- `r` : « transformation de clôture »
- `s` : « transformation de conversion »

– `t` : « transformation de normalisation »

ii) Inscrivez aussi les quatre expressions suivantes (ou les lettres) dans les cases appropriées. **[4 points]** *(Ici, chaque expression doit trouver sa place dans une case !)*

– `A` : « caméra »
– `B` : « glFrustum »
– `C` : « Cohen-Sutherland »
– `D` : « point de fuite »

Question 4 Modèles de couleur

a) Combien de triplets du modèle HSV peuvent représenter chacune des couleurs suivantes ? **[3 points]**

rouge : _____ vert : _____ bleu : _____ noir : _____ blanc : _____

b) Donnez les coordonnées de la couleur magenta dans le modèle CMY et dans le modèle RGB. **[2 points]**

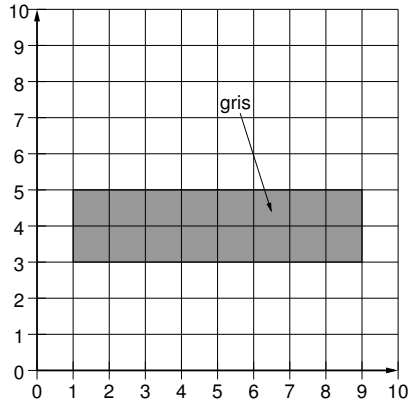
CMY : _____ RGB : _____

c) Dans une imprimante couleur, pourquoi y-a-t-il une cartouche d'encre noire en plus des cartouches d'encre cyan, d'encre magenta et d'encre jaune ? **[1 point]**

d) Quelle couleur est perçue par l'oeil lorsqu'on éclaire une feuille imprimée avec de l'encre jaune avec une lumière bleue ? Expliquez pourquoi. **[2 points]**

Question 5 Opérations sur les fragments

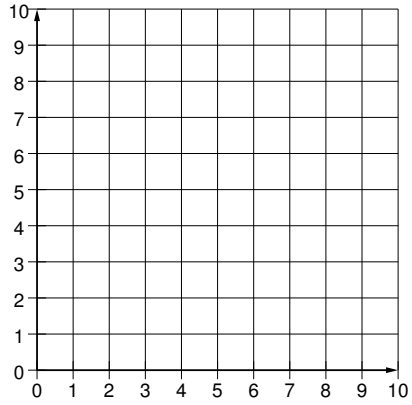
On vous demande de dessiner le contenu de la mémoire de trame et du stencil aux différents jalons identifiés (a), (b), (c), (d) et (e) dans le programme OpenGL présenté à l'annexe A. (L'effet des énoncés est évidemment cumulatif.) Ainsi, à la ligne notée « EXEMPLE ... », la mémoire de trame contient ceci :

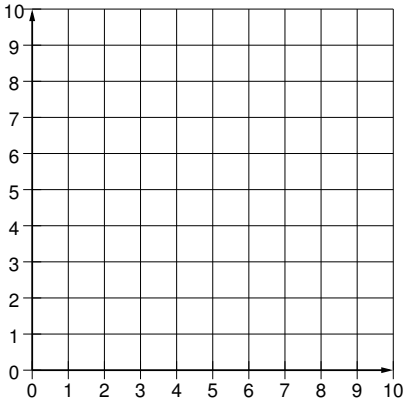


mémoire de trame à la ligne EXEMPLE

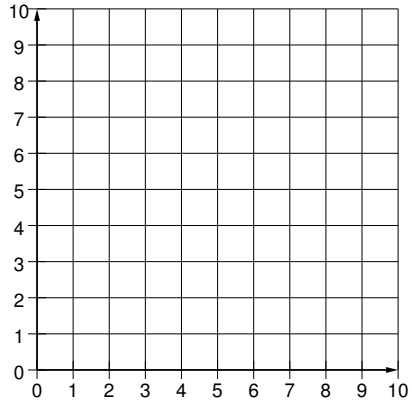
- L'affichage est de bonne résolution et on peut donc considérer les trames comme des régions continues.
- Dans vos réponses pour la mémoire de trame, assurez-vous d'indiquer clairement la couleur de chaque région.
- Dans vos réponses pour le stencil, inscrivez la valeur stockée dans le stencil pour chaque région.
- À moins d'avis contraire, le *brouillon* tout à droite est un brouillon qui ne sera pas corrigé.

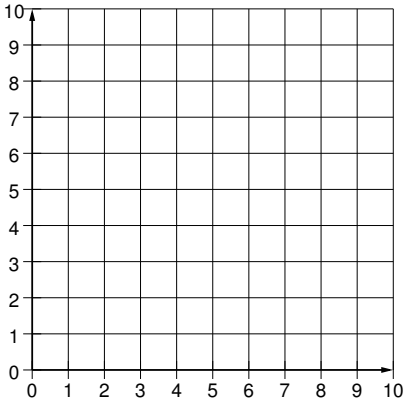
a) Dessinez le contenu de la mémoire de trame. **[1 point]**


mémoire de trame à ligne (**a)**

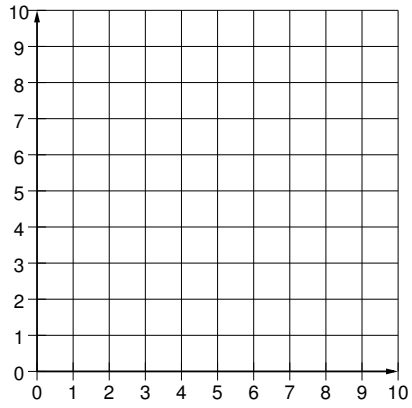

(brouillon)

b) Dessinez le contenu de la mémoire de trame. **[3 points]**

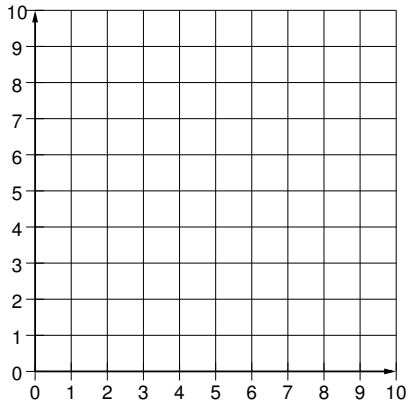

mémoire de trame à ligne (**b)**


(brouillon)

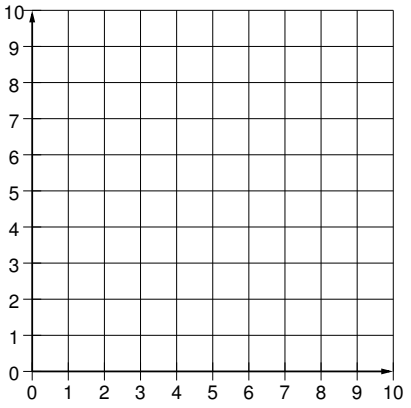
c) Dessinez le contenu de la mémoire de trame et le contenu du stencil. **[3 points]**



mémoire de trame à ligne (**c)**

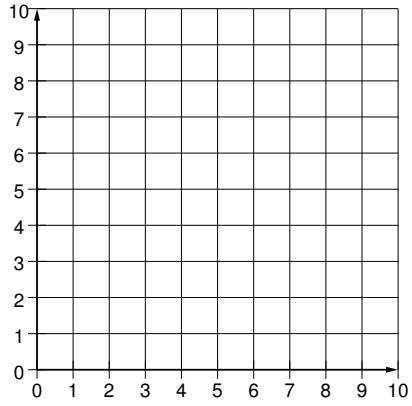


stencil à ligne (**c)**

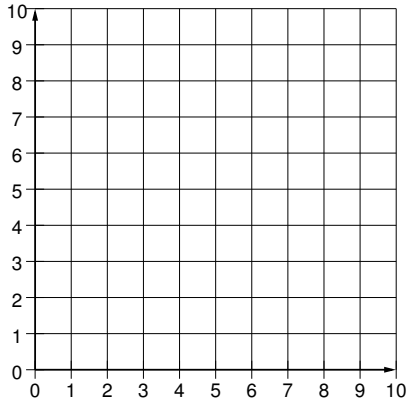


(brouillon)

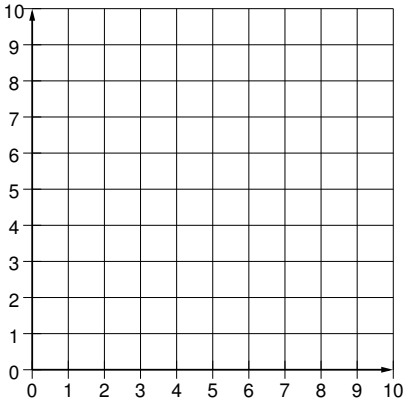
d) Dessinez le contenu de la mémoire de trame et le contenu du stencil. **[3 points]**



mémoire de trame à ligne (**d)**

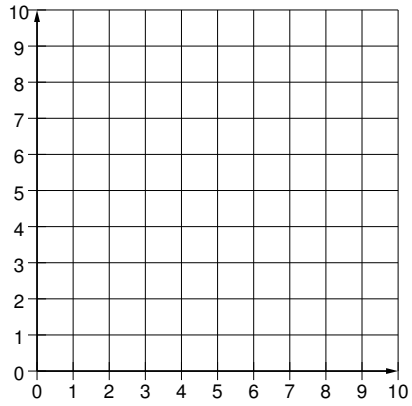


stencil à ligne (**d)**

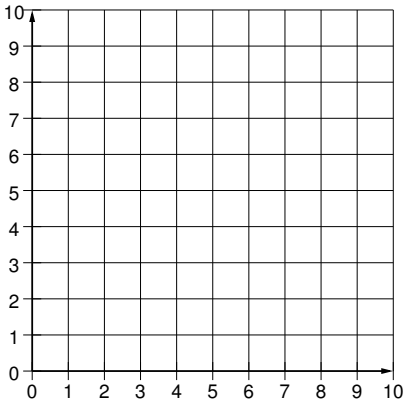


(brouillon)

e) Dessinez le contenu de la mémoire de trame. **[3 points]**



mémoire de trame à ligne (**e)**



(brouillon)

Annexe A - pour la question 5 (page 16)

(Vous pouvez détacher cette annexe et ne pas la remettre.)

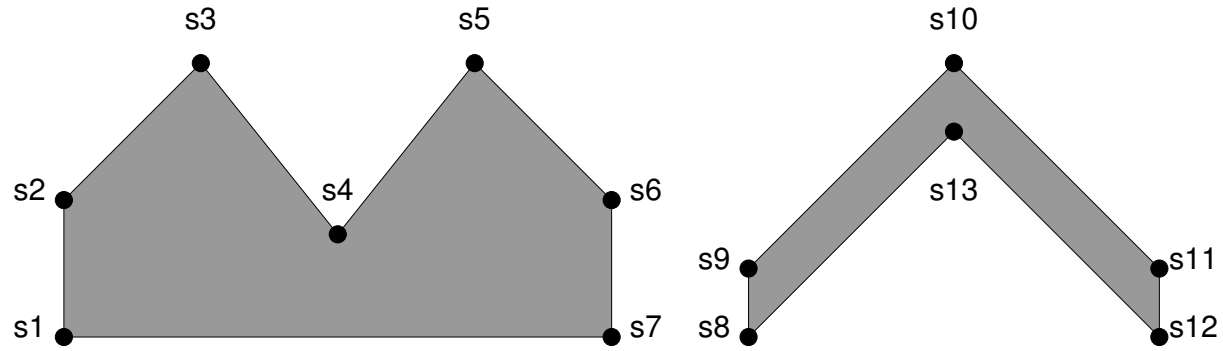
```
static void dessine( )
{
   // Initialisation du pipeline graphique
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity( );
   glOrtho( 0, 10,  0, 10,  -10, 10 );

   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity( );

   // fixer la couleur de l'arrière-plan à "blanc"
   glClearColor( 1.0, 1.0, 1.0, 1.0 );

   // initialiser les différents tampons
   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );

   glEnable( GL_DEPTH_TEST );

   glBegin( GL_QUADS ); {
      glColor4f( 0.5, 0.5, 0.5, 1.0 );
      glVertex3i( 1, 5, -5 );
      glVertex3i( 1, 3, -5 );
      glVertex3i( 9, 3, -5 );
      glVertex3i( 9, 5, -5 );
   } glEnd();

// EXEMPLE DE DESSIN DU CONTENU DE LA MÉMOIRE DE TRAME

   glBegin( GL_QUADS ); {
      glColor4f( 0.5, 0.5, 0.5, 0.5 );
      glVertex3i( 7, 6, -7 );
      glVertex3i( 7, 9, -7 );
      glVertex3i( 4, 9, -7 );
      glVertex3i( 4, 6, -7 );
   } glEnd();

// (a) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME

   glDisable( GL_DEPTH_TEST );

   glBegin( GL_QUADS ); {
      glColor3f( 0.0, 0.0, 0.0 );
      glVertex3i( 5, 0, -6 );
      glVertex3i( 6, 0, -6 );
      glVertex3i( 6, 9, -6 );
      glVertex3i( 5, 9, -6 );
   } glEnd();

// (b) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME
```

```
   // void glStencilOp( GLenum fail, GLenum zfail, GLenum zpass )
   // void glStencilFunc( GLenum func, GLint ref, GLuint mask )

   glEnable( GL_STENCIL_TEST );
   glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );
   glStencilFunc( GL_NEVER, 1, 1 );

   glBegin( GL_QUADS ); {
      glColor3f( 0.5, 0.5, 0.5 );
      glVertex3i( 0, 0, -2 );
      glVertex3i( 3, 0, -2 );
      glVertex3i( 3, 4, -2 );
      glVertex3i( 0, 4, -2 );
   } glEnd();

// (c) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME ET DU STENCIL

   glStencilFunc( GL_NOTEQUAL, 1, 1 ) ;
   glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );

   glBegin( GL_QUADS ); {
      glColor3f( 0.0, 0.0, 0.0 );
      glVertex3i( 0, 0, -3 );
      glVertex3i( 4, 0, -3 );
      glVertex3i( 4, 5, -3 );
      glVertex3i( 0, 5, -3 );
   } glEnd();

   glDisable( GL_STENCIL_TEST );

// (d) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME ET DU STENCIL

   // void glBlendFunc( GLenum sourcefactor, GLenum destfactor )

   glEnable( GL_DEPTH_TEST );
   glEnable( GL_BLEND );
   glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

   glBegin( GL_QUADS ); {
      glColor4f( 0.0, 0.0, 0.0, 0.0 );
      glVertex3i( 7, 2, 0 );
      glVertex3i( 9, 2, 0 );
      glVertex3i( 9, 6, 0 );
      glVertex3i( 7, 6, 0 );
   } glEnd();

   glDisable( GL_BLEND );

// (e) DESSINEZ LE CONTENU DE LA MÉMOIRE DE TRAME

   glutSwapBuffers();
   return;
}
```

2009 Automne intra

Question 1 Primitives OpenGL

Considérez ces formes pleines tracées par des primitives OpenGL :



Dans les sous-questions suivantes, on vous demande d'écrire les énoncés `glBegin`, `glVertex2iv` et `glEnd` nécessaires pour tracer exactement et correctement les formes ci-dessus en utilisant les primitives d'OpenGL. Les règles à respecter sont :

- vous ne devez pas utiliser d'autres sommets que ceux existants (s1 à s13) ;
- vous devez utiliser un nombre minimal de paires `glBegin() ... glEnd()`.
- vous ne devez pas remplir ou tracer par-dessus ce qui est déjà rempli ou tracé.

Exemple :
Quels sont les énoncés OpenGL si on utilise la primitive `GL_TRIANGLES` *?*

```
glBegin( GL_TRIANGLES );
  S1 S7 S4   S1 S4 S2   S2 S4 S3   S7 S6 S4   S6 S5 S4
glEnd( );
glBegin( GL_TRIANGLES );
  S9 S8 S13   S9 S13 S10   S13 S12 S11   S13 S11 S10
glEnd( );
```

où, pour faire plus court dans la réponse, « `S1` » *remplace* « `glVertex2iv(s1);` ».

a) Quels sont les énoncés OpenGL si on utilise la primitive `GL_QUADS` ? **[2 points]**

b) Quels sont les énoncés OpenGL si on utilise la primitive `GL_QUAD_STRIP` ? **[2 points]**

c) Quels sont les énoncés OpenGL si on utilise la primitive `GL_TRIANGLE_STRIP` ? **[2 points]**

d) Quels sont les énoncés OpenGL si on utilise la primitive `GL_TRIANGLE_FAN` ? **[2 points]**

e) Quels sont les énoncés OpenGL si on utilise la primitive `GL_POLYGON` ? **[2 points]**

f) Pour l'affichage de ces formes, quelles sont les primitives les plus efficaces ? Pourquoi ? **[2 points]**

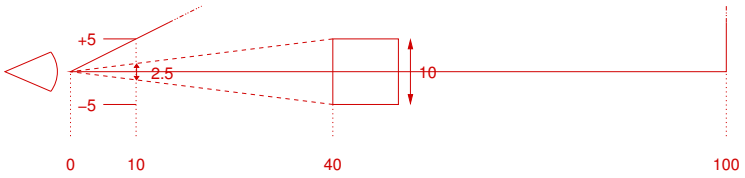
Question 2 Visualisation 3D

Une application ouvre une fenêtre OpenGL de 400 x 200 pixels. Cette application utilise `glFrustum` pour générer une matrice de projection en utilisant les paramètres suivants :

```
// glFrustum( GLdouble Gauche, GLdouble Droit,
//            GLdouble Bas, GLdouble Haut,
//            GLdouble PlanAvant, GLdouble PlanArrière );

   glMatrixMode( GL_PROJECTION );
   glLoadIdentity( );
   glFrustum(  -10.0, 10.0,  -5.0, 5.0,  10.0, 100.0  );
```

Un cube de 10 x 10 x 10 unités est positionné à 40 unités de l'observateur, directement dans l'axe de visée, au centre du volume de visualisation. La normale de la face avant du cube pointe directement vers l'observateur et celui-ci voit ainsi une forme bien carrée.

a) Quelles sont les dimensions de la face avant du cube une fois projetée sur le plan avant du volume de visualisation ? Illustrez votre raisonnement et vos calculs avec un diagramme. **[3 points]**



La face du cube projetée sur le plan avant est obtenue par une relation de proportion :
10 / 40 = Dim / 10 $\Longrightarrow$ Dim = 100 / 40 = ***2.5*** *[1pt] (ou* ***100 x 100*** *pixels).*

b) Quelles modifications doit-on apporter à Gauche, Droit, Bas et Haut (sans modifier PlanAvant ou PlanArrière) de manière à cadrer ce cube, c'est-à-dire de manière à ce que la face avant du cube occupe le plus d'espace possible dans la fenêtre ? Expliquez vos calculs. (N'oubliez pas de respecter le rapport d'aspect !) **[2 points]**

On pourrait avoir bêtement : Gauche = -1.25, Droite = 1.25, Bas = -1.25, Haut = 1.25 ;
pour respecter le rapport d'aspect, on prend ***Gauche = -2.5, Droit = 2.5, Bas = -1.25, Haut = 1.25***

c) Est-il possible de cadrer le cube en ne modifiant que PlanAvant ? Si oui, donnez la valeur à utiliser. Sinon, expliquez pourquoi. **[2 points]**

Oui. On doit amener le PlanAvant exactement à la position de la surface avant du cube : ***PlanAvant = 40****.*

d) Est-il possible de cadrer le cube en ne modifiant que PlanArrière ? Si oui, donnez la valeur à utiliser. Sinon, expliquez pourquoi. **[2 points]**

Non. Le plan arrière ne sert qu'au découpage du volume et n'affecte pas l'ouverture de la pyramide de visualisation.

Question 3 Notions de base

a) Une mise à l'échelle est fondamentalement une opération de *multiplication*, tandis qu'une translation est, au contraire, une opération d'*addition*. Pourtant, la librairie OpenGL cumule toutes ces opérations dans une seule matrice de transformation courante qui n'est utilisée, elle, que pour *multiplier* des coordonnées. Comment est-ce possible ? (Expliquez brièvement le fonctionnement.) **[2 points]**

b) Distinguez les concepts de sommet, de fragment et de pixel. **[3 points]**

c) Durant le cours, nous avons présenté trois modèles d'affichage.

i) Identifiez ces trois modèles d'affichage.

ii) Pour chacun, décrivez brièvement en quel format l'information graphique est conservée.
(Une image est décomposée en ...)

iii) Donnez un cas exemple où l'utilisation de chacun est appropriée.

[4 points]

d) Nous avons vu quelques modèles de couleur en classe.

i) Nommez trois modèles de couleur que nous avons vus.

ii) Pour chacun, décrivez les *trois* axes qui le définissent

iii) Dessinez le *volume* contenant les couleurs que le modèle définit.

[4 points]

e) Nous avons aussi parlé de modèles de couleur additif et soustractif.

i) Expliquez ce qu'est un modèle de couleur soustractif (par opposition à un modèle additif).

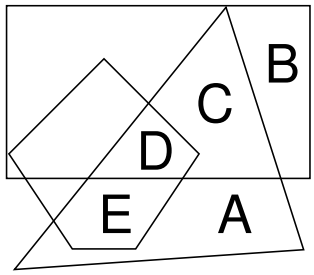
ii) Nommez une application courante de ce type de modèle.

[2 points]

f) Convertissez la couleur CMY (0.4, 0.75, 0.1) en couleur RGB. **[1 point]**

Question 4 Fragments

a) Trois primitives graphiques sont dessinées l'une à la suite de l'autre dans un programme utilisant les énoncés OpenGL ci-dessous pour tracer les primitives illustrées dans la figure ci-contre. La première primitive est un triangle, la seconde est un rectangle et la troisième est un pentagone. La fusion de couleurs OpenGL (`GL_BLEND`) étant activée, il faut déterminer, pour chaque région, quelle est la couleur source et quelle est la couleur destination afin de bien calculer la couleur résultante dans chaque région A, B, C, D, E dans le rendu final.




```
glPolygonMode( GL_FRONT_AND_BACK, GL_FILL );
glDisable( GL_DEPTH_TEST );
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA ); // les valeurs de défaut!
glClearColor( 0.2, 0.2, 0.2, 1.0 );
glClear( GL_COLOR_BUFFER_BIT );
glColor4f( 0.7, 0.6, 0.6, 1.0 );
glCallList( leTriangle );  // la liste ne contient aucun appel à glColor()
glColor4f( 1.0, 0.2, 0.2, 0.5 );
glCallList( leRectangle );  // la liste ne contient aucun appel à glColor()
glCallList( lePentagone );  // la liste ne contient aucun appel à glColor()
```

i) Quelle sera la formule qui permet de calculer la couleur résultante selon ces énoncés ? **[2 points]**

Il faut utiliser la formule suivante pour la fusion de couleurs :
$RGBA_{final} = RGBA_{source} * A_{source} + RGBA_{destination} * (1 - A_{source})$

(Les valeurs de A doivent être calculées parce qu'elle pourrait être utilisées si on dessine autre chose par-dessus.)

ii) Lorsque la première primitive est tracée dans la région A, quelles sont les valeurs (R,G,B,A) de la couleur source, celles de la couleur destination et celles de la couleur résultante ? **[1 point]**

S : (0.7, 0.6, 0.6, 1.0) = la couleur du triangle
D : (0.2, 0.2, 0.2, 1.0) = la couleur du fond
*R : (0.7, 0.6, 0.6, 1.0) * (1.0) + (0.2, 0.2, 0.2, 1.0) * (1-1.0)*
*⇒ **(0.7, 0.6, 0.6, 1.0)***

iii) Lorsque la seconde primitive est tracée dans la région B quelles sont les valeurs (R,G,B,A) de la couleur source, celles de la couleur destination et celles de la couleur résultante ? **[2 points]**

S : (1.0, 0.2, 0.2, 0.5) = la couleur du rectangle
D : (0.2, 0.2, 0.2, 1.0) = la couleur du fond

*R : (1.0, 0.2, 0.2, 0.5) * (0.5) + (0.2, 0.2, 0.2, 1.0) * (1-0.5)*
= (0.5, 0.1, 0.1, 0.25) + (0.1, 0.1, 0.1, 0.5)
*⇒ **(0.6, 0.2, 0.2, 0.75)***

iv) Lorsque la seconde primitive est tracée dans la région C quelles sont les valeurs (R,G,B,A) de la couleur source, celles de la couleur destination et celles de la couleur résultante ? **[2 points]**

S : (1.0, 0.2, 0.2, 0.5) = la couleur du rectangle
D : (0.7, 0.6, 0.6, 1.0) = la couleur de A (ii)
*R : (1.0, 0.2, 0.2, 0.5) * (0.5) + (0.7, 0.6, 0.6, 1.0) * (1-0.5)*
= (0.5, 0.1, 0.1, 0.25) + (0.35, 0.3, 0.3, 0.5)
*⇒ **(0.85, 0.4, 0.4, 0.75)***

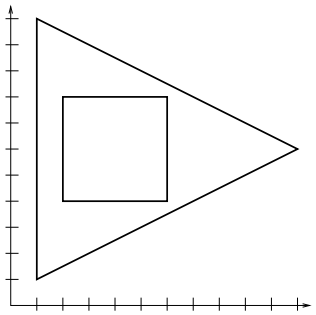
v) Lorsque la troisième primitive est tracée dans la région D quelles sont les valeurs (R,G,B,A) de la couleur source, celles de la couleur destination et celles de la couleur résultante ? **[2 points]**

S : (1.0, 0.2, 0.2, 0.5) = la couleur du pentagone
D : (0.85, 0.4, 0.4, 0.75) = la couleur de C (iv)
*R : (1.0, 0.2, 0.2, 0.5) * (0.5) + (0.85, 0.4, 0.4, 0.75) * (1-0.5)*
= (0.5, 0.1, 0.1, 0.25) + (0.425, 0.2, 0.2, 0.375)
*⇒ **(0.925, 0.3, 0.3, 0.625)***

vi) Lorsque la troisième primitive est tracée dans la région E quelles sont les valeurs (R,G,B,A) de la couleur source, celles de la couleur destination et celles de la couleur résultante ? **[1 point]**

S : (1.0, 0.2, 0.2, 0.5) = la couleur du pentagone
D : (0.7, 0.6, 0.6, 1.0) = la couleur de A (ii)
*R : (1.0, 0.2, 0.2, 0.5) * (0.5) + (0.7, 0.6, 0.6, 1.0) * (1-0.5)*
= (0.5, 0.1, 0.1, 0.25) + (0.35, 0.3, 0.3, 0.5)
*⇒ **(0.85, 0.4, 0.4, 0.75)***
La même couleur qu'en C (iv).

b) Les énoncés OpenGL suivants sont utilisés pour dessiner un quadrilatère plein et un triangle plein (voir ci-contre). La caméra est placée en (0, 0, 10) et regarde vers l'origine.



```
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_LESS ); // la valeur de défaut!
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
glOrtho( -6.0, 6.0, -6.0, 6.0, 0.0, 20.0 );
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
gluLookAt( 0.0, 0.0, 10.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 );
glBegin(GL_TRIANGLES);
  glColor3f( 0.0, 0.5, 1.0 );
  glVertex3f(  1.0,  1.0, 8.0 );
```

```
  glVertex3f( 11.0,  6.0, 8.0 );
  glVertex3f(  1.0, 11.0, 8.0 );
glEnd();
glBegin(GL_QUADS);
  glColor3f( 1.0, 0.5, 0.0 );
  glVertex3f(  2.0,  4.0, 3.0 );
  glVertex3f(  6.0,  4.0, 3.0 );
  glVertex3f(  6.0,  8.0, 3.0 );
  glVertex3f(  2.0,  8.0, 3.0 );
glEnd();
```

i) Quelle est la valeur de Z (entre 0.0 et 1.0) des fragments du triangle après la transformation de projection ? Expliquez votre réponse. **[1 point]**

La caméra se trouve en (0, 0, 10) et le volume de visualisation est profond de 20 unités. Le plan avant est à 10 et le plan arrière est à -10. Après la transformation de projection, la profondeur en Z du triangle est de ***(10-8) / 20 = 2 / 20 = 0.1****.*

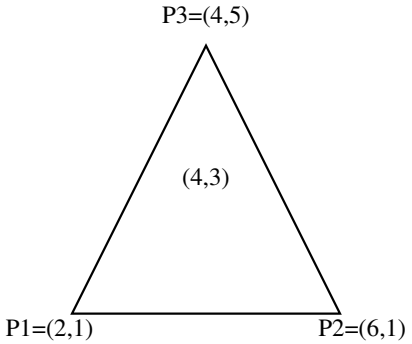
ii) Quelle est la valeur de Z (entre 0.0 et 1.0) des fragments du quadrilatère après la transformation de projection ? Expliquez votre réponse. **[1 point]**

Pour sa part, la profondeur en Z du quadrilatère est de ***(10-3) / 20 = 7 / 20 = 0.35****.*

iii) Quelle est la comparaison faite par le test de profondeur qui est exécuté lors de l'affichage du quadrilatère ? Est-ce que le quadrilatère sera visible dans le rendu final ? **[1 point]**

Le test est : ***0.35 < 0.1*** *? $\Rightarrow$ NON... donc rejet du quadrilatère.*

Question 5 Listes d'affichage



a) Considérez le triangle plein décrit par trois sommets dont les coordonnées sont : P1=(2,1), P2=(6,1) et P3=(4,5). Donnez tous les énoncés OpenGL nécessaires pour créer correctement la liste d'affichage qui affichera ce triangle aux coordonnées indiquées. (Si on appelle la liste d'affichage sans aucune autre transformation, le triangle s'affiche à ces coordonnées.) On stockera le numéro de la liste d'affichage dans la variable `triLA`. **[3 points]**

```
// Création de la liste d'affichage
GLuint triLA = ...
...
```

b) On souhaite maintenant créer une séquence d'animation où ce triangle tournera sur lui-même, autour du point (4,3), dans le plan Z=0. Pour tester l'animation, seulement ce triangle tournant sur lui-même sera d'abord tracé et animé. Le programme principal appellera la fonction `affiche` qui utilise la liste d'affichage créée précédemment pour afficher le triangle, en tournant à chaque fois le triangle d'un angle supplémentaire de 1 degré autour du point (4,3). Donnez les énoncés qu'il faut mettre dans cette fonction, sachant qu'elle reçoit en argument le numéro de la liste d'affichage et qu'elle appelle la liste précédemment créée en (a). **[4 points]**

```
// Affichage de la scène
static void affiche( const GLuint triLA )
{
  static GLfloat angleCourant = 0; // en degrés
  angleCourant += 1.0;
  glClear( GL_COLOR_BUFFER_BIT );
  ...
}
```

c) Dans une compagnie où vous venez d'être embauché, un de vos nouveaux collègues soutient que les listes d'affichage ne servent à rien. Pour le convaincre du contraire, donnez deux arguments militant en faveur de l'utilisation des listes d'affichage. Expliquez brièvement et clairement chacun. **[3 points]**

2008 Automne intra

Question 1 Concepts théoriques

a) Durant le cours, nous avons présenté trois modèles d'affichage.

i) Identifiez ces trois modèles d'affichage.

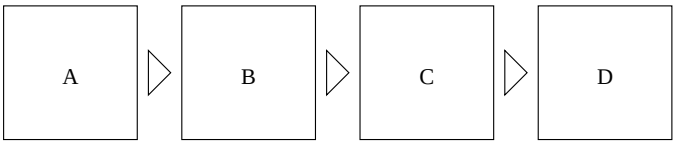
ii) Pour chacun, décrivez brièvement en quel format l'information graphique est conservée.
(Une image est décomposée en ...)

iii) Donnez un cas exemple où l'utilisation de chacun est appropriée.

[4 points]

b) Quelle est l'utilité d'une représentation en coordonnées homogènes des sommets dans les primitives graphiques ? **[2 points]**

c) Vous reconnaissez certainement le pipeline graphique d'OpenGL dans le schéma ci-dessous. Chaque case identifie une étape de transformation faite par OpenGL sur les sommets passent dans le pipeline.



i) Donnez les noms assignés à chaque étape (A-B-C-D) du pipeline.

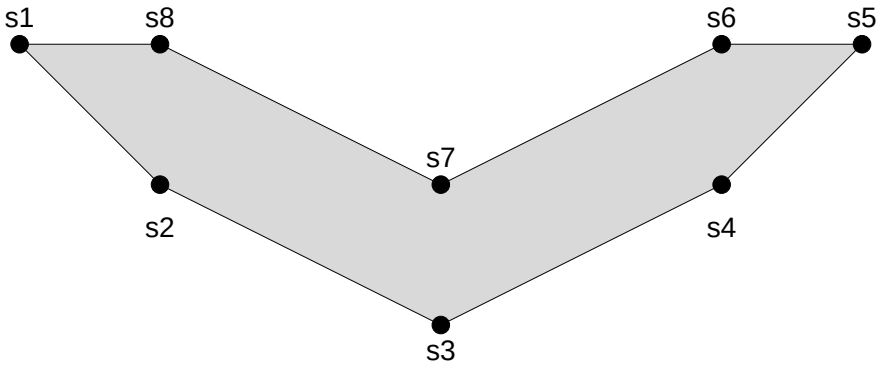
ii) Nommez les 12 fonctions qui permettent de contrôler les transformations appliquées dans ce pipeline.

iii) Identifiez quelles fonctions s'appliquent à chaque étape du pipeline. (Notez que certaines fonctions peuvent s'appliquer à plus d'une étape !)

[6 points]

Question 2 Primitives OpenGL

Considérez cette forme pleine tracée avec les énoncés OpenGL qui suivent :




```
glBegin( GL_TRIANGLES );
  glVertex2iv( s1 ); glVertex2iv( s2 ); glVertex2iv( s8 );
  glVertex2iv( s2 ); glVertex2iv( s7 ); glVertex2iv( s8 );
  glVertex2iv( s2 ); glVertex2iv( s3 ); glVertex2iv( s7 );
  glVertex2iv( s3 ); glVertex2iv( s4 ); glVertex2iv( s7 );
  glVertex2iv( s7 ); glVertex2iv( s4 ); glVertex2iv( s6 );
  glVertex2iv( s4 ); glVertex2iv( s5 ); glVertex2iv( s6 );
glEnd( );
```

Dans les sous-questions suivantes, on vous demande de donner les énoncés `glVertex2iv` nécessaires pour tracer exactement et correctement la même forme, mais en utilisant d'autres primitives OpenGL. Quelques règles à respecter :

- vous ne devez pas utiliser d'autres sommets que ceux existants (s1 à s8) ;
- vous devez utiliser le nombre minimal de paires `glBegin() ... glEnd()`.
- vous ne devez pas remplir ou tracer par-dessus ce qui est déjà rempli ou tracé.

Pour faire une réponse plus courte, plutôt que d'écrire l'énoncé `glVertex2iv(`s_i`)` au complet, vous pouvez n'indiquer que le numéro du sommet s_i. Ainsi, au lieu des énoncés donnés en exemple, on peut écrire :

```
glBegin( GL_TRIANGLES );
  s1 s2 s8   s2 s7 s8   s2 s3 s7   s3 s4 s7   s7 s4 s6   s4 s5 s6
glEnd( );
```

a) Quel doit être l'ordre des énoncés si on utilise la primitive `GL_QUADS` ? **[2 points]**

b) Quel doit être l'ordre des énoncés si on utilise la primitive `GL_QUAD_STRIP` ? **[2 points]**

c) Quel doit être l'ordre des énoncés si on utilise la primitive `GL_TRIANGLE_STRIP` ? **[2 points]**

d) Quel doit être l'ordre des énoncés si on utilise la primitive `GL_TRIANGLE_FAN` ? **[2 points]**

e) Quel doit être l'ordre des énoncés si on utilise la primitive `GL_POLYGON` ? **[2 points]**

f) Pour l'affichage de cette forme, quelles sont les primitives les plus efficaces ? Pourquoi ? **[1 point]**

Question 3 Listes d'affichage

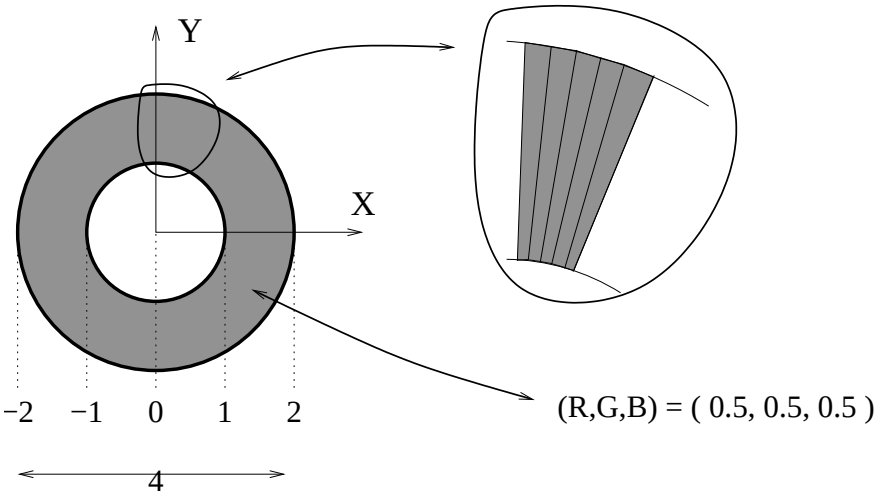


Figure 1

a) On souhaite tracer un anneau plein en approximant sa forme par des quadrilatères minces (voir Fig. 1). On utilisera une seule primitive OpenGL (une seule paire `glBegin() ... glEnd()` et aucune fonction des librairies GLU ou GLUT) et plusieurs `glVertex`.
Écrivez les énoncés C++ qui créent une liste d'affichage qui tracera cet anneau *centré à l'origine*, de la *taille* et de la *couleur* indiquées dans la figure. (On veut voir les appels aux fonctions OpenGL avec leurs arguments, de même que la boucle `for` ($n \approx 50$) qui calcule les coordonnées des sommets.) **[3 points]**

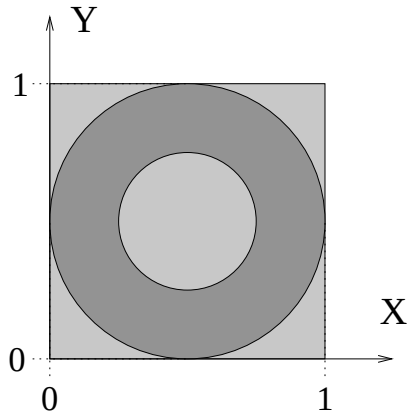


Figure 2

b) On souhaite maintenant tracer un carré unitaire plein (voir Fig. 2) où est inscrit l'anneau précédent. Notez la position de l'origine ! À nouveau, on utilisera une seule primitive OpenGL (une seule paire `glBegin() ... glEnd()` et aucune fonction des librairies GLU ou GLUT).
Écrivez les énoncés C++ qui créent une seconde liste d'affichage qui tracera cette figure à la *position* et de la *taille* indiquées dans la figure et en faisant appel à la liste d'affichage créée en (a) pour inscrire l'anneau

dans le carré. (On veut voir les appels aux fonctions OpenGL avec leurs arguments, incluant l'appel à la liste d'affichage de l'anneau.) La couleur de fond du carré sera spécifiée par ailleurs. **[4 points]**

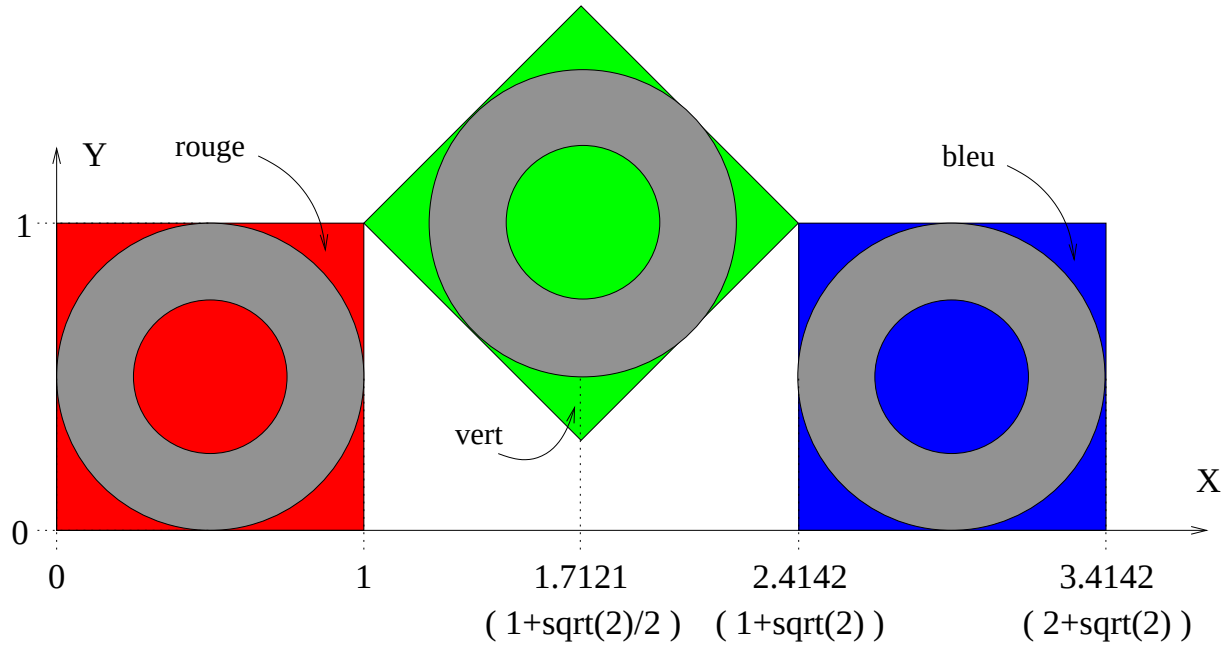
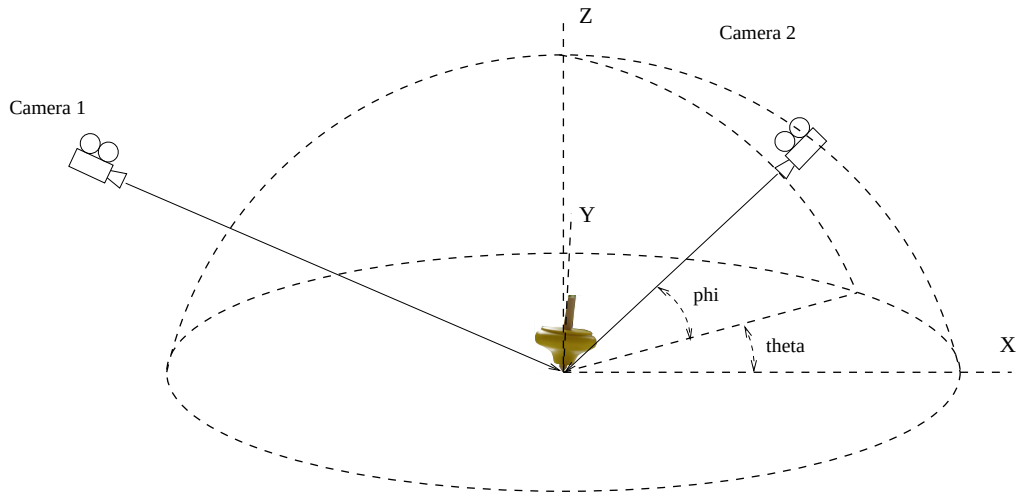


Figure 3

c) On souhaite maintenant utiliser la liste d'affichage définie en (b) pour tracer la forme illustrée ci-dessus (voir Fig. 3).
Écrivez les énoncés C++ qui tracent cette figure à la *position*, de la *taille* et de la *couleur* indiquées dans la figure et en faisant appel à la liste d'affichage créée en (b). (On veut voir les appels aux fonctions OpenGL avec leurs arguments, incluant la spécification des couleurs de fond des carrés.) **[4 points]**

Question 4 Visualisation 3D et caméra synthétique



Vous concevez un nouveau jeu de toupies et vous voulez suivre les mouvements d'une toupie avec des

caméras en proposant deux vues possibles :

1. La vue « 3^e personne » où la caméra 1 suit la toupie et imite ses mouvements, tout en restant éloignée d'une certaine distance et toujours positionnée le long du vecteur (-1,-1,+1) par rapport à la toupie.
2. La vue sphérique où la caméra 2 peut tourner librement autour de la position de la toupie (sur la surface marquée en pointillés dans le diagramme ci-dessus). Les mouvements de cette caméra sont toutefois limités : la caméra ne peut se déplacer que dans l'hémisphère situé au-dessus du plan X-Y et la caméra ne peut être directement au-dessus de la toupie.

La distance de la caméra par rapport à la toupie est contrôlable pour les deux points de vue.

La signature de la fonction `PositionnerCamera` est :

```
void PositionnerCamera( const int numero, const float distance,
                        const float theta, const float phi,
                        const float toupieX, const float toupieY )
{ ... }
```

où :

- `numero` : est le numéro de la caméra (1 ou 2) ;
- `distance` : est la distance de la caméra à la toupie, valide pour les deux caméras ;
- `theta` et `phi` : sont les angles (en degrés) pour positionner la caméra 2 (inutiles pour la caméra 1) ;
- `toupieX` et `toupieY` : sont la position de la toupie dans le plan $Z = 0$.

Implémentez la fonction `PositionnerCamera` qui permet de positionner la caméra par rapport à la toupie à chaque rendu, en tenant compte des contraintes indiquées. Votre fonction doit mettre à jour toutes les matrices nécessaires à l'affichage, incluant la matrice de projection, en appelant les fonctions OpenGL pertinentes. (Vous pouvez choisir la perspective qui vous semble la plus appropriée, mais une ouverture de 90 degrés serait correcte.) Suite à l'appel de la fonction `PositionnerCamera`, il ne reste plus qu'à effectuer l'affichage de la toupie (que vous n'avez pas à faire ici mais plutôt dans votre TP2 !). **[9 points]**

2008 Hiver intra

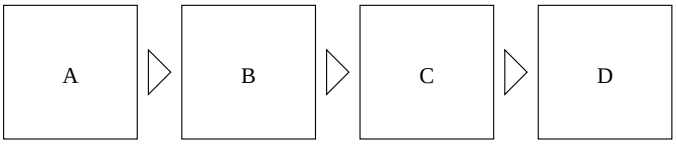
Question 1 Concepts théoriques

a) Dites à quoi sert chacune des fonctions OpenGL suivantes. Autant que possible, soyez bref et précis, tout en vous assurant de bien montrer l'utilité de la fonction. S'il y a lieu, faites référence aux arguments de cette fonction. Par exemple :
glPointSize ... spécifie le diamètre, en pixels, des points affichés à l'écran ;
glColor3f ... définit les nouvelles intensités de rouge, vert et bleu de la couleur courante.

i. glFrustum ...
ii. glGetFloatv ...
iii. glClear ...
iv. glIsEnabled ...
v. glMatrixMode ...
vi. glMultMatrixd ...
vii. glPolygonMode ...
viii. glPopMatrix ...
ix. glScalef ...
x. glPushAttrib ...
xi. gluLookAt ...
xii. glLoadIdentity ..
xiii. glVertex2d ...
xiv. gluOrtho2D ...
xv. glEnd ...

[15 points]

b) Vous reconnaissez le pipeline graphique d'OpenGL dans le schéma ci-dessous. Chaque case identifie une étape de transformation faite par OpenGL sur les sommets donnés au début du pipeline.



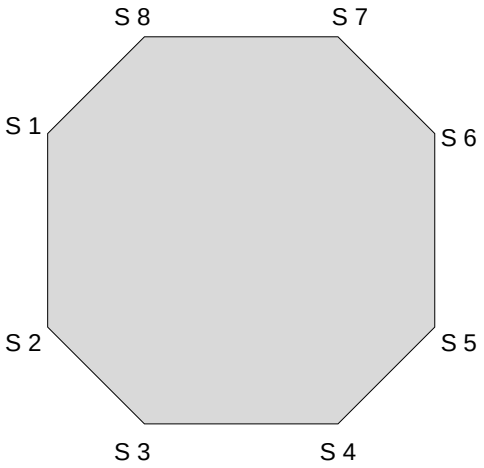
i) Donnez les noms assignés à chaque étape (A-B-C-D) du pipeline.

ii) Pour chacune des quatre étapes, nommez toutes les fonctions qui permettent de contrôler la transformation effectuée.

[6 points]

Question 2 Transformations affines et primitives OpenGL

Considérez les sommets de l'octogone illustré, qu'on peut tracer avec les énoncés OpenGL ci-dessous :




```
glBegin( GL_POLYGON );
  glVertex2iv( s1 );
  glVertex2iv( s2 );
  glVertex2iv( s3 );
  glVertex2iv( s4 );
  glVertex2iv( s5 );
  glVertex2iv( s6 );
  glVertex2iv( s7 );
  glVertex2iv( s8 );
glEnd( );
```

Dans les sous-questions suivantes, on vous demande de donner les énoncés `glVertex2iv` nécessaires pour tracer exactement et correctement la même surface, mais en utilisant d'autres primitives OpenGL. Quelques règles à respecter : i) vous ne devez pas utiliser d'autres sommets que s1 à s8 ; ii) vous ne devez pas remplir / tracer par-dessus ce qui est déjà rempli / tracé. Pour faire une réponse plus courte, vous pouvez n'indiquer que le numéro du sommet s_i plutôt que d'écrire l'énoncé `glVertex2iv(`s_i`)` au complet.

a) Que doit être l'ordre des énoncés si on utilise la primitive GL_QUADS ? **[2 points]**

b) Que doit être l'ordre des énoncés si on utilise la primitive GL_QUAD_STRIP ? **[2 points]**

c) Que doit être l'ordre des énoncés si on utilise la primitive GL_TRIANGLES ? **[2 points]**

d) Que doit être l'ordre des énoncés si on utilise la primitive GL_TRIANGLE_STRIP ? **[2 points]**

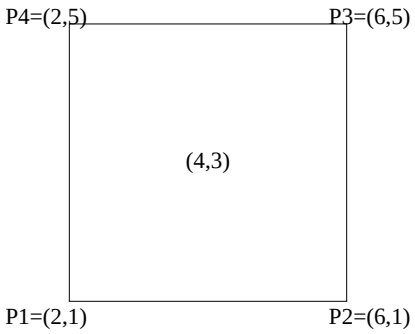
e) Que doit être l'ordre des énoncés si on utilise la primitive GL_TRIANGLE_FAN ? **[2 points]**

f) Pour l'affichage de cet octogone, quelle(s) primitive(s) semble(nt) être la(les) plus efficace(s) ? Pourquoi ? **[2 points]**

Question 3 Listes d'affichage

Considérez le rectangle décrit par quatre sommets dont les coordonnées sont : P1=(2,1), P2=(6,1), P3=(6,5) et P4=(2,5). On souhaite créer une séquence d'animation où ce rectangle doit tourner sur lui-même, autour

de son centre de gravité, dans le plan 2D. Bien sûr, on voudra utiliser des listes d'affichage afin d'être efficace !



a) Donnez tous les énoncés OpenGL nécessaires pour créer la liste d'affichage qui contient ce rectangle. On stockera le numéro de la liste d'affichage dans la variable `rectN`. **[3 points]**

```
// Création de la liste d'affichage
GLuint rectN = ...
...
```

b) Pour tester l'animation, seulement ce rectangle tournant sur lui-même sera d'abord tracé et animé. Le programme principal appelera la fonction `affiche` qui utilise la liste d'affichage créée précédemment pour afficher le rectangle, en tournant à chaque fois le rectangle d'un angle supplémentaire de 1 degré autour de son centre de gravité situé en (4,3). Donnez les énoncés qu'il faut mettre dans cette fonction, sachant qu'elle reçoit en argument le numéro de la liste d'affichage. **[4 points]**

```
// Affichage de la scène
static void affiche( const GLuint rectN )
{
  static GLfloat angleCourant = 0; // en degrés
  angleCourant += 1.0;
  glClear( GL_COLOR_BUFFER_BIT );
  ...
}
```

c) Donnez deux arguments militant en faveur de l'utilisation des listes d'affichage. Expliquez brièvement et clairement chacun. **[3 points]**

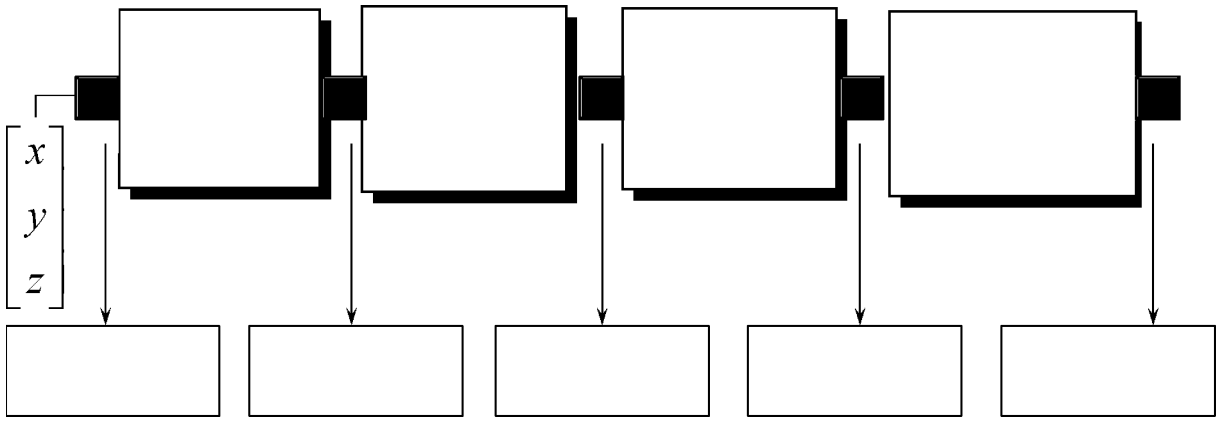
2007 Hiver intra

Question 1 Notions de base

a) Pour les transformations géométriques 3D, pourquoi utilise-t-on une matrice 4X4 au lieu de 3X3 ? Comment s'appelle le système de coordonnées ainsi créé ? **[2 points]**

b) Comparez les deux approches suivantes au niveau de leurs avantages et inconvénients respectifs : Mode indexé avec table de couleurs de 256 entrées (avec composantes R, G et B codées sur 8 bits) VS mode direct « trueColor » codé sur 24 bits. **[2 points]**

c) Remplissez correctement les cases vides de ce schéma du pipeline openGL de la transformation des sommets avec les neuf concepts suivants : coordonnées normalisées, matrice de projection, coordonnées de visualisation, transformation de clôture, coordonnées de découpage, transformation de normalisation, coordonnées d'objet, coordonnés d'affichage, matrice de modélisation. **[3 points]**



Question 2 Concepts théoriques

a) Durant le cours, nous avons présenté trois modèles d'affichage.

i) Identifiez ces trois modèles d'affichage

ii) Décrivez brièvement comment l'information est conservée dans chacun

iii) Donnez au moins un exemple d'utilisation appropriée pour chacun.

[3 points]

b) Durant le cours, nous avons aussi présenté le contexte graphique.

i) Quelle est l'utilité de définir un contexte graphique dans une application ?

ii) Donnez au moins un exemple où l'utilisation est clairement avantageuse.

[3 points]

c) Parmi les objets dessinés par une certaine application graphique, il y a un quelconque objet sphérique.

i) Si cette application utilise une projection orthograhique, est-ce que la silhouette de l'objet sphérique sera toujours un cercle ? Dites pourquoi.

ii) Si cette application utilise une projection perspective, est-ce que la silhouette de l'objet sphérique sera toujours un cercle ? Dites pourquoi.

[1 point]

Question 3 Listes d'affichage et hiérarchie

À l'occasion du nouvel an chinois qui aura lieu le 7 février 2008 prochain, un ami vous demande de lui créer un dragon virtuel en openGL qu'il pourra intégrer dans son application le moment venu afin d'épater les enfants.

Pour ce faire, vous devez lui écrire les fonctions suivantes :

```
void afficherDragon(float theta1, float theta2, float theta3, float posX, float posY);
void compilerTete( );
void compilerCorps( );
void compilerQueue( );
```

La fonction `afficherDragon` se chargera d'afficher le dragon tout entier à la position virtuelle (posX, posY), qui correspond au milieu de la tête du dragon. La fonction devra *impérativement* faire usage des listes d'affichages et des transformations matricielles, et mettre de l'avant la hiérarchisation des listes d'affichage afin de dessiner le dragon de façon performante. Vous devez aussi utiliser dans votre code les arguments d'entrées `theta1`, `theta2`, `theta3`, `posX` et `posY`.

Les fonctions `compilerTete`, `compilerCorps` et `compilerQueue` se chargeront de compiler en listes d'affichage distinctes le graphisme des segments correspondants.

L'unique axe de rotation de la tête et de la queue se situe en (0,0), alors que le segment de corps possède deux axes de rotation en (0,0) et en (-4,0). Le graphisme des différentes parties du dragon vous est fourni en coordonnées d'objet dans la figure 1.

Tel q'indiqué sur la figure 2, le dragon est composé (dans l'ordre) d'une « tête » jaune, d'un premier segment de « corps » vert, d'un deuxième segment de « corps » bleu et d'une « queue » rouge, articulés entre eux selon les angles θ_1, θ_2 et θ_3. La position du dragon en coordonnées universelles se situe en (posX, posY), point qui correspond à la commissure des lèvres de la tête du dragon (voir figure 1).

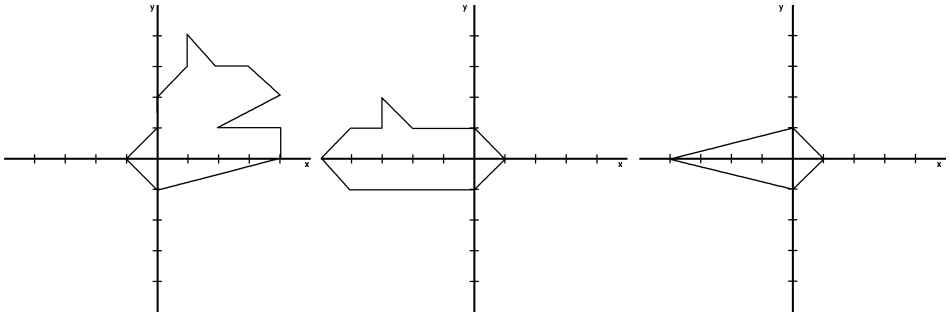


Figure 1 : Définition en coordonnées d'objet de la tête (gauche), d'un segment de corps (milieu) et de la queue (droite).

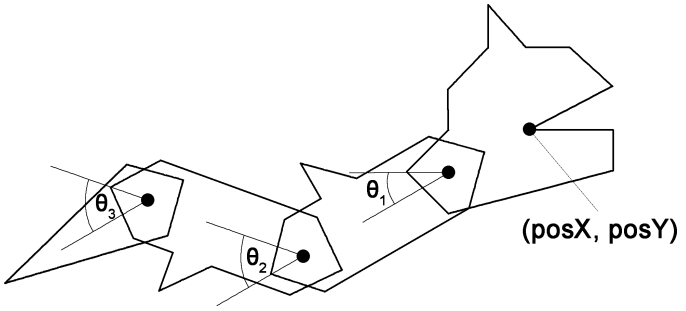


Figure 2 : Constitution du dragon. Le dragon est constitué d'une tête, de deux segments corps (l'un inversé, l'autre normal) et d'une queue.

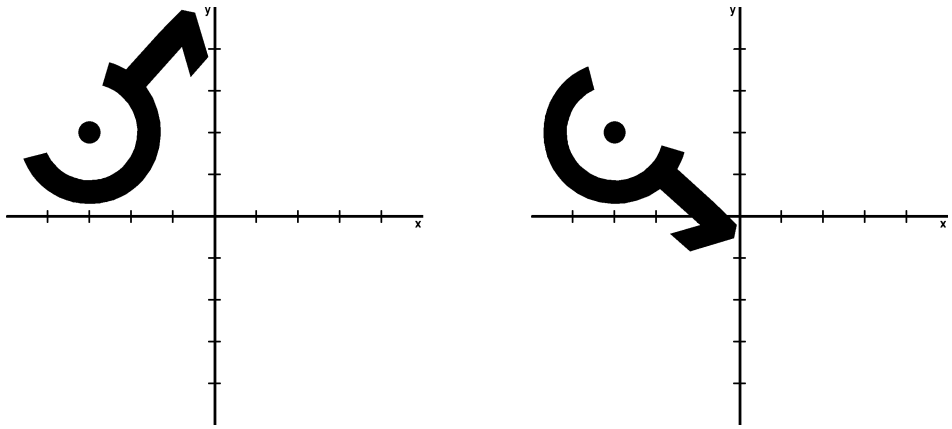
Écrivez les quatre fonctions demandées. **[10 points]**

Question 4 Transformations affines 2D

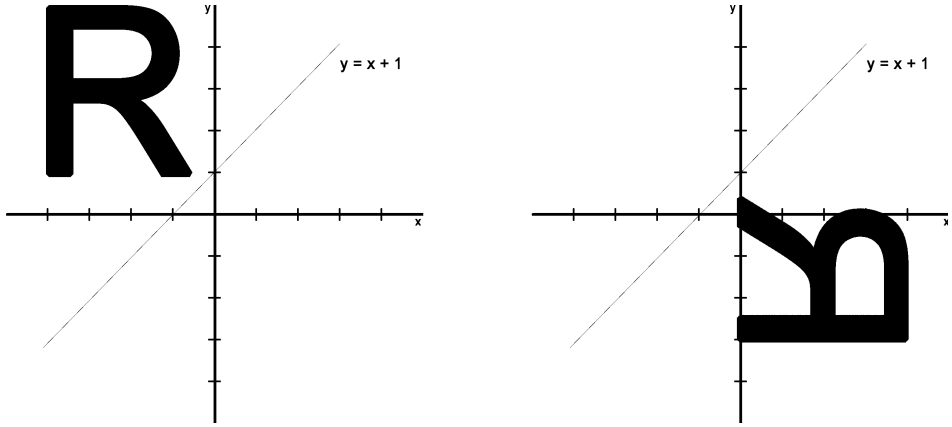
Voici un ensemble de 2 patrons définis en coordonnées d'objet dans l'image de gauche. Chacun de ces patrons ont subi une série de n transformations élémentaires affines 2D dont le résultat final est affiché dans l'image de droite. Ces transformations élémentaires peuvent être de type translation $T(d_x, d_y)$, rotation par rapport à l'origine $R(\theta)$ ou mise à l'échelle $S(s_x, s_y)$.

Pour chaque patron :
- Décrivez textuellement les n différentes étapes de transformation (en donnant également la valeur des paramètres) ;
- Écrivez les matrices 3x3 correspondantes à chacune des n étapes ; et
- Écrivez l'équation matricielle $p' = Mp$ en détaillant M en fonction des matrices $T(d_x, d_y)$, $R(\theta)$ et $S(s_x, s_y)$ de façon à faire ressortir leur ordre matriciel.

a) Patron 1 : $n = 3$



b) Patron 2 : $n = 5$



Question 5 Visualisation 3D et caméra synthétique

Pour mieux visiter la planète Mars, on pourrait vouloir mettre en orbite une navette autour de la planète. Pour simuler cette possibilité, on souhaite écrire un programme en OpenGL qui montrera la vue que les astronautes auront à partir de la navette.

a) L'orbite de la navette passe d'un pôle à l'autre à une hauteur de $2 * 10^6 m$ au-dessus du sol. On convient que l'origine de notre repère orthogonal est placée au centre de Mars, que l'axe des Z passe par les pôles et que l'axe des X passe par un point bien connu des astronautes (le cratère Airy-0 qui définit le méridien 0). Considérant que le rayon moyen de Mars est d'environ $3.4 * 10^6 m$, le rayon total de l'orbite est donc de $5.4 * 10^6 m$ ($5\ 400\ 000\ m$). L'angle θ (en degrés) donne la position courante de la navette sur son orbite (dans le plan YZ). Notez que le plancher de la navette demeure toujours parallèle à la surface du sol (figure 1). Écrivez la fonction `DefinirCamera` avec les énoncés OpenGL nécessaires en indiquant bien la valeur des paramètres qu'il faut mettre afin d'initialiser la matrice de visualisation qui donne le point de vue d'un astronaute assis aux commandes de la navette et qui regarde droit devant lui (sans voir la planète). (N'oubliez pas de faire appel à `glMatrixMode` !) **[5 points]**

```
void DefinirCamera( const GLdouble theta ) { ... }
```

b) Pour bien fonctionner, notre logiciel de visualisation doit aussi définir le contenu de la matrice de projection. L'astronaute est assis aux commandes de la navette et regarde droit devant lui, vers le centre d'une fenêtre rectangulaire qui est perpendiculaire à la direction de son regard (figure 2). Il est intéressé par ce qui est visible à l'extérieur de la navette, dans l'espace autour de la planète. Son siège est positionné à $3\ m$ du centre de la fenêtre et celle-ci mesure $4\ m$ en largeur et $1.6\ m$ en hauteur. Écrivez la fonction `DefinirProjection` avec les énoncés OpenGL nécessaires en indiquant bien la valeur des paramètres qu'il faut mettre. (N'oubliez pas `glMatrixMode` !) **[4 points]**

```
void DefinirProjection( void ) { ... }
```

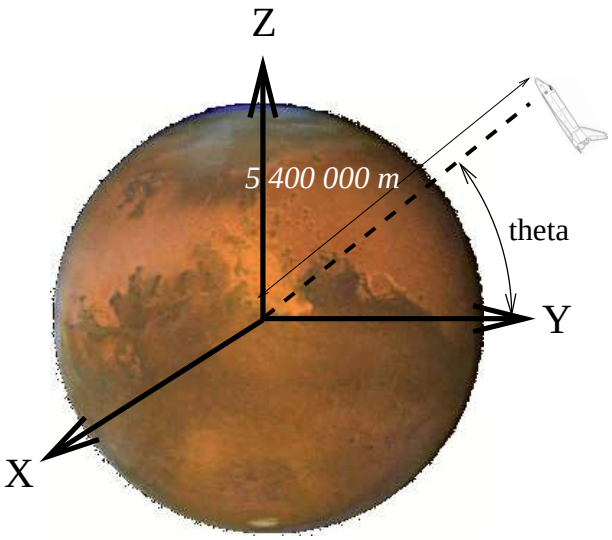


figure 1

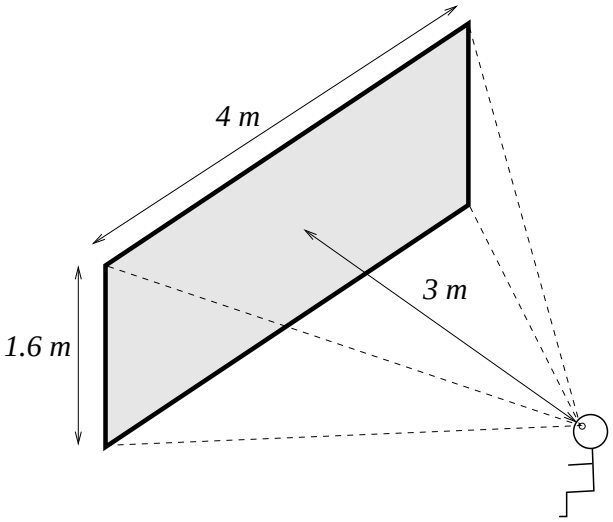


figure 2