Nom : [redacted] *Prénom* [redacted]

École polytechnique de Montréal
Département de génie informatique et génie logiciel

INF2705: Infographie (Hiver 2019)
Contrôle périodique

Notes:
- Toute documentation interdite, calculatrice programmable et ordinateur portatif interdits. Calculatrice non programmable permise.
- Cet examen comprend 5 questions sur 9 pages pour un total de 40 points.

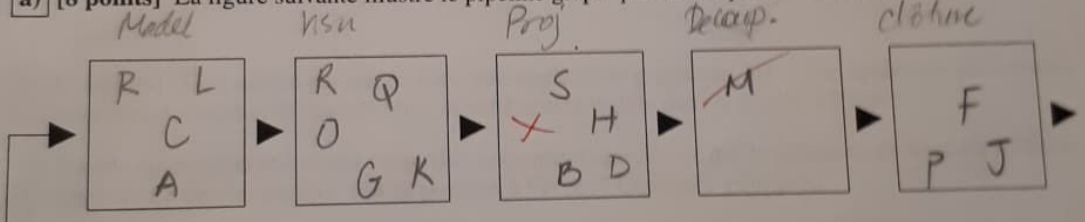
⇒ *Répondez aux questions directement sur le questionnaire.*

Q1 : 7.5
Q2 : 9
Q3 : 7
Q4 : 8
Q5 : 6

37.5

Question 1 Pipeline graphique 3D [8 points]

a) [8 points] La figure suivante illustre le pipeline graphique 3D tel que vu au cours.



non pertinent: ~~E~~ I N

Pour chacun des groupes de mots ou énoncés suivants, inscrivez la lettre correspondante ci-dessus, soit dans le carré approprié (selon ce à quoi il est associé ou ce qu'il peut contrôler), soit sur la ligne intitulée « *non pertinent* » (s'il n'est pas associé à ce pipeline). *(-0.5 point par lettre manquante dans chaque case.)*

- ~~A~~ : *rotation 3D autour d'un axe arbitraire*
- ~~B~~ : *projection orthographique*
- ~~C~~ : *matrice de modélisation*
- ~~D~~ : *point de fuite*
- ~~E~~ : *fusion de couleurs*
- ~~F~~ : *clôture*
- ~~G~~ : *caméra synthétique*
- ~~H~~ : *angle d'ouverture*
- ~~I~~ : *multi-échantillonnage*
- ~~J~~ : *dimensions de la fenêtre à l'écran*
- ~~K~~ : *pivoter la caméra*
- ~~L~~ : *transformations de modélisation*
- ~~M~~ : *volume de visualisation*
- ~~N~~ : *transformation de stencil*
- ~~O~~ : *positionner l'observateur*
- ~~P~~ : `glViewport()`
- ~~Q~~ : `MatricePipeline::LookAt()`
- ~~R~~ : `MatricePipeline::Rotate()`
- ~~S~~ : `MatricePipeline::Frustum()`

me remettre

Question 2 Concepts théoriques en infographie [10 points]

a) [2 points] Nous avons vu que pour représenter les transformations géométriques 3D en infographie, on utilise généralement une matrice 4X4 au lieu d'une matrice 3X3.

i) Quelle est la principale raison de cet apparent ajout de complexité ?

Pour faire une translation, il faut faire une addition/soustraction et non pas un produit, c'est pourquoi on a besoin d'une matrice 4x4

1

ii) Comment se nomme la coordonnée supplémentaire ainsi ajoutée en 4D ?

b) [3 points] Une sphère est tracée dans une application graphique 3D. Cette sphère est entièrement contenue dans le volume de visualisation et 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, quelque soit sa position à l'écran ? Expliquez pourquoi.

Non, si le rapport d'aspect du viewport n'est pas respecté, la sphère sera déformée ✓

3

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

Non, si le rappat d'aspect du viewport n'est pas respecté, la sphère sera déformée. ✓

c) [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 ?

La face avant est celle où les sommets sont numérotés en ordre anti-horaire et la face arrière en sens horaire.

d) [3 points] Les applications d'infographie, comme celles que vous avez développées avec la librairie OpenGL, utilisent généralement un tampon de profondeur qui est « effacé » au besoin par un appel à `glClear( GL_DEPTH_BUFFER_BIT )`.

i) Que mesure chaque « *profondeur* » que contient ce tampon ? *(C'est la distance entre ...)*

La distance normalisée entre le fragment et le plan avant du volume de visualisation.

ii) Pourquoi ce tampon de profondeur est-il utile ?

Pour comparer la profondeur des différents objets et afficher ceux qui sont plus près de l'observateur par dessus ceux qui sont plus éloignés et qui sont partiellement ou totalement cachés.

iii) Dans l'application, à quel moment (dans l'exécution du programme) doit-on effacer ce tampon avec l'appel ci-dessus ?

Au début de chaque affichage

Prénom : Maude

de Montréal
et génie logiciel

7.5

Question 3 Visualisation 3D [7 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*)

7

L'une de ces deux fonctions est en fait plus générale que l'autre puisqu'elle permet de définir certaines projections perspectives que l'autre ne permet pas.

a) [1 point] Laquelle des deux fonctions est la plus générale ?

Frustrum

b) [1 point] Quelle est la caractéristique d'une projection perspective qui peut être définie par la fonction la plus générale, mais qui ne peut pas être définie par la fonction la moins générale ?

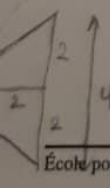
Avec Frustrum la projection du point où se trouve la caméra sur le plan avant ne doit pas forcément être centrée alors qu'avec perspective elle est toujours centrée

c) [2 points] Donnez un exemple d'appel (avec des valeurs pour les paramètres) à la fonction la plus générale qui ne peut pas être converti en un appel à la fonction la moins générale.

Frustrum (-5, 0, -2, 3, 1, 10)

d) [3 points] Considérez les paramètres de cet énoncé : `Perspective( 90.0, 0.5, 2.0, 4.0 );`. Est-il possible d'utiliser la fonction `Frustum()` pour définir exactement la même projection ? Si oui, écrivez les paramètres dans l'appel. Sinon, dites pourquoi.

`Frustum(` -1 , 1 , -2 , 2 , 2 , 4 `);`



me nommer

Maude

Question 4 Opérations sur les fragments [8 points]

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 d'écrire 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	0.2	(1.0, 0.9, 0.8, 0.7)
- 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 "
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons	0.2	(1.0, 0.9, 0.8, 0.7)

b)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)

```
glDisable( GL_STENCIL_TEST );
glDisable( GL_DEPTH_TEST ); // Désactiver le test fait en sorte que le
                            // tampon de profondeur ne sera pas modifié
glDepthFunc( GL_GREATER ); // " > "
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons	0.7	(1.0, 0.9, 0.8, 0.7)

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

c)

	profondeur (z)	*couleur (r, g, b, a)*	*stencil*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)	-
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	5

```
glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_LEQUAL, 3, 7 ); // " <= "
glStencilOp( GL_DECR, GL_REPLACE, GL_INCR );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_LEQUAL ); // " <= "
glDisable( GL_BLEND );
```

- valeurs subséquentes dans les tampons | 0.2 | (1.0, 0.9, 0.8, 0.7) | 6

d)

	profondeur (z)	*couleur (r, g, b, a)*	*stencil*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)	-
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	5

```
glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_LESS, 3, 7 ); // " < "
glStencilOp( GL_DECR, GL_REPLACE, GL_INCR );
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) | 3

e)

	profondeur (z)	*couleur (r, g, b, a)*	*stencil*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)	-
- valeurs présentes dans les tampons	0.7	(0.6, 0.6, 0.6, 0.6)	5

```
glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_GEQUAL, 3, 7 ); // " >= "
glStencilOp( GL_DECR, GL_REPLACE, GL_INCR );
glEnable( GL_DEPTH_TEST );
glDepthFunc( GL_LESS ); // " < ", la valeur de défaut
glDisable( GL_BLEND );
```

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

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

f)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)
- 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 ); // " >= "
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA ); // " SrcA, 1-SrcA "
```

NON parce pas on s'arrête ici

- valeurs subséquentes dans les tampons | 0,7 | (0.6, 0.6, 0.6, 0.6)

g)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)
- 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_ZERO, GL_ONE ); // " 0, 1 "
```

OK

- valeurs subséquentes dans les tampons | 0.2 | (0.6, 0.6, 0.6, 0.6)

h)

	profondeur (z)	*couleur (r, g, b, a)*
- valeurs des attributs du fragment	0.2	(1.0, 0.9, 0.8, 0.7)
- 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_ONE, GL_ONE_MINUS_SRC_ALPHA ); // " 1, 1-SrcA "
```

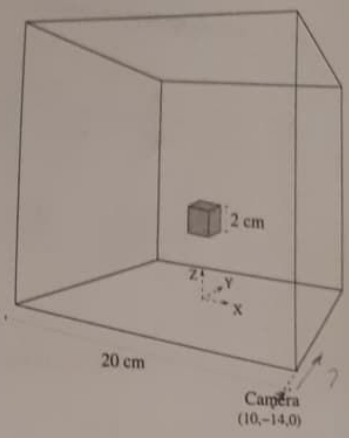
OK

1.0 0.9 0.8 0.7
+ 0.3·0.6 0.3·0.6 0.3·0.6 0.3·0.6

- valeurs subséquentes dans les tampons | 0.2 | (1.0, 1.0, 0.98, 0.88)

Question 5 Le « poisson » cubique [7 points]

Dans votre TP2, les poissons qui se déplaçaient dans l'aquarium étaient cylindriques. Si on simplifie le corps de l'animal en lui donnant un petit corps cubique sans yeux, l'affichage en perspective serait semblable à la figure ci-contre.



- Le corps de ce nouvel animal simplifié est un cube dont chaque arête mesure 2 cm.

- Cet animal se déplace dans un aquarium aussi cubique dont chaque arête mesure 20 cm.

- L'origine du repère du monde est au sol, au centre de l'aquarium, et le corps de l'animal ne dépasse jamais les frontières de l'aquarium. (L'étrange animal entre souvent en collision avec les parois de l'aquarium en se déplaçant, mais le *centre* de son corps cubique de 2 cm est toujours à au moins 1 cm de la paroi.)

- On positionne la caméra synthétique à l'extérieur de l'aquarium, au niveau du plancher à (10, -14, 0). La caméra regarde dans le sens positif de l'axe des Y et son « *vecteur up* » est orienté vers le haut, dans l'axe des Z.

- Enfin (et surtout), on veut utiliser une projection orthogonale qui définit un volume de visualisation correspondant exactement à l'espace intérieur de l'aquarium où peut se déplacer l'animal.

Le programme principal de l'application graphique utilise la fonction ci-dessous pour afficher la scène :

```
void FenetreTP::afficherScene()
{
  glUseProgram( prog );
  glClear( ... );
  glViewport( 0, 0, 400, 400 );   // le rapport d'aspect est fixe et respecté
  MatricePipeline matrModel, matrVisu, matrProj;
  matrVisu.LookAt( ... );
  glUniformMatrix4fv( locmatrVisu, 1, GL_FALSE, matrVisu );
  matrProj.Ortho( ... );  // projection orthogonale
  glUniformMatrix4fv( locmatrProj, 1, GL_FALSE, matrProj );
  matrModel.LoadIdentity();
  matrModel.Translate( positionAnimal.x, positionAnimal.y, positionAnimal.z );
  glUniformMatrix4fv( locmatrModel, 1, GL_FALSE, matrModel );
  afficherAnimal(); // un cube
}
```

Dans le nuanceur de sommets utilisé, on trouve ces lignes (parmi d'autres) :

```
vec4 pos = matrModel * Vertex;
vec4 posVisu = matrVisu * pos;
gl_Position = matrProj * posVisu;
```

Considérant que ce logiciel graphique est utilisé comme décrit ci-dessus, remplissez les espaces soulignés à la page suivante.

a) **[3 points]** Dans ces deux énoncés de `afficherScene()`, les paramètres appropriés sont :

```
matrVisu.LookAt( 10, -14, 0,  10, 0, 0,  0, 0, 1 );
//       LookAt( obsx, ...y, ...z, ptViséx, ...y, ...z, upx, ...y, ...z );

matrProj.Ortho( -20, 0, 0, 20, 4, 24 );
//       Ortho( gauche, droite, bas, haut, avant, arrière );
```

3

b) **[4 points]** Dans le nuanceur de sommets utilisé : *(-0.5 point par erreur.)*

– Les valeurs de `Vertex.x` sont toujours entre ___-1___ et ___1___ (inclusivement).

3 – Les valeurs de `Vertex.y` sont toujours entre ___-1___ et ___1___ (inclusivement).

– Les valeurs de `Vertex.z` sont toujours entre ___-1___ et ___1___ (inclusivement).

Model = Vertex

– Les valeurs de `pos.x` sont toujours entre ___-10___ et ___10___ (inclusivement).

– Les valeurs de `pos.y` sont toujours entre ___-10___ et ___10___ (inclusivement).

– Les valeurs de `pos.z` sont toujours entre ___0___ et ___20___ (inclusivement).

Visu = pos

– Les valeurs de `posVisu.x` sont toujours entre ___-20___ et ___0___ (inclusivement).

– Les valeurs de `posVisu.y` sont toujours entre ___4___ et ___24___ (inclusivement). ✗

– Les valeurs de `posVisu.z` sont toujours entre ___0___ et ___20___ (inclusivement). ✗

Cet examen comprend 5 questions sur 9 pages pour un total de 40 points.
Benoît Ozell