

### Question 1 Courbes paramétriques polynomiales [9 points]

a) [4 points] Les courbes paramétriques sont définies par la relation  $\sum_{i=1}^n N_{i,k}(t)G_i$  où la forme des fonctions de base  $N(t)$  diffère d'un type de courbe à l'autre. Associez les fonctions de base ci-dessous à un des types de courbe que nous avons vus en classe.

i)  $N_{i,1}(t) = 1 \text{ si } u_i \leq t \leq u_{i+1}, 0 \text{ sinon}$   
 $N_{i,k}(t) = \frac{t-u_i}{u_{i+k-1}-u_i} N_{i,k-1}(t) + \frac{u_{i+k}-t}{u_{i+k}-u_{i+1}} N_{i+1,k-1}(t)$

B-Spline

ii)  $N_{i,k=n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$

Bézier

iii)  $N_{i,k}^*(t) = \frac{h_i N_{i,k}(t)}{\sum_{j=1}^n h_j N_{j,k}(t)}$

NURBS

iv)  $N_{i,k=4}(t) = H_i(t)$  où  $H_1(t) = 2t^3 - 3t^2 + 1$   
 $H_2(t) = -2t^3 + 3t^2$   
 $H_3(t) = t^3 - 2t^2 + t$   
 $H_4(t) = t^3 - t^2$

Spline cubique

b) [2 points] Supposez deux B-Splines et leur enveloppe convexe respective formée des points de contrôle de chaque courbe.

i) Si les deux enveloppes convexes des courbes n'ont aucune région en commun, peut-on conclure que les deux B-Splines ne s'intersectent pas ? Pourquoi ?

On ne peut conclure que les deux B-splines ne s'intersectent pas.

ii) Si les deux enveloppes convexes des courbes ont une région en commun, peut-on conclure que les deux B-Splines s'intersectent dans cette région ? Pourquoi ?

on peut conclure que les deux s'intersectent dans cette région si  $i, k$  sont pareils ✗ non

c) [3 points] Soit les 6 points de contrôle  $\{\vec{P}_1, \vec{P}_2, \vec{P}_3, \vec{P}_4, \vec{P}_5, \vec{P}_6\}$ .

i) Écrivez l'équation de la courbe de Bézier construite à partir de ces points de contrôle. ( $\vec{C}(t) = \dots$ )

$$\cancel{n=6} \quad K=5 \quad (n+1)=6 \quad n=5 \quad K=4$$

*note: DÉTAILLER*

$$\vec{c}(t) = \underline{\vec{P}_1(1-t)^5} + \underline{5t^4(1-t)^4} \vec{P}_2 + \underline{5t^2(1-t)^3} \vec{P}_3 + \cancel{5t^3(1-t)^2} \vec{P}_4 + \cancel{5t^4(1-t)^1} \vec{P}_5 + t^5 \vec{P}_6$$

$n=6 \quad K=5 \quad n+K=11$  noeuds

$$(0001234555)$$

$$\left(0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\right)$$

ii) Par lequel ou lesquels des six points de contrôle la courbe de Bézier passe-t-elle certainement ?

La courbe de Bézier passe par  $\times$

iii) Quelle contrainte devrait-on imposer aux points de contrôle pour que la courbe de Bézier passe par les six points ?

Pour qu'il passe par les 6 points faudrait écrire le 1<sup>er</sup> noeud et le dernier noeud  $K$  fois.

INF2705

**Question 2 Notions de base [11 points]**

- a) [1 point] Le langage PostScript est un langage graphique très populaire, mis au point par Adobe, qui a servi de base pour le format PDF utilisé aujourd’hui par la grande majorité des imprimantes. Dans ces langages graphiques, quelle est l’unité de mesure de base du système de coordonnées ?

l’unité est le point ✓       $72 \text{ points} = 1 \text{ pouce}$

- b) [1 point] Nous avons vu en classe un *algorithme du point milieu* qui est utilisé depuis longtemps par les cartes graphiques. Au milieu de quoi est ce « point milieu » ?

au milieu du pixel (fragment)  
non

- c) [2 points] Dans votre TP3, vous avez utilisé des nuanceurs de tessellation afin d’améliorer l’illumination du modèle de Gouraud. Donnez deux autres cas d’utilisation où l’emploi de nuanceurs de tessellation dans une application d’infographie (autre que ce TP3) serait utile et pertinent.

1. le nuanceur de tessellation est utilisé dans une application d’infographie pour le ~~élimination~~ de l’ombrage de phong  
X déjà au pixel près
2. Pour l’utilisation de l’illumination du modèle de lambert  
X constant  
OK

d) [2 points] Lorsque le test de profondeur est utilisé (`glEnable(GL_DEPTH_TEST)`), est-ce que le rendu à l'écran est toujours la même, *peu importe l'ordre* dans lequel les primitives sont tracées (en supposant bien sûr qu'on garde le même point de vue et la même projection)? Pourquoi?

0 Non, le rendu n'est pas le même car les primitives sont tracées de manière qui sont liées  $\times$

e) [5 points] Lors du rendu d'une scène composée de plusieurs triangles avec OpenGL, vous constatez que beaucoup de triangles tracés par le logiciel sont visibles à l'écran (pixels allumés), tandis que d'autres triangles ne le sont pas du tout. Ce n'est pourtant pas une erreur et on peut même dire que c'est un résultat courant dans un programme graphique.

Donnez *cinq raisons distinctes* qui peuvent faire que certains triangles ne soient absolument pas visibles à l'écran. Utilisez des termes reliés à la librairie OpenGL.

1. `glClear(GL_DEPTH_BUFFER_BIT)` n'est pas appelé  $\times$
2. `glPosition = normalized(glPosition)` par ~~que ce n'est pas normal~~ normalize tous les 4 condamnés  $\times$
3. `glDrawElements()`  $\times$
4. ~~glVertexAttribPointer~~  $\times$
5. `glBindBuffer`  $\times$

### Question 3 Illumination [20 points]

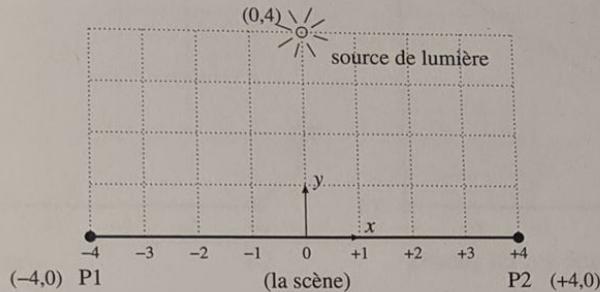
a) [8 points] Comme nous l'avons vu en classe, le modèle de réflexion de la lumière en un point proposé par Phong contient trois termes distincts de réflexion : *ambiante*, *diffuse* et *spéculaire*.

Identifiez les réflexions de la lumière que chaque élément ci-dessous influence ou contrôle.  
Cochez un seul choix par ligne. (-1.0 point par erreur afin de pénaliser les choix aléatoires!)

	aucune	ambi. seulement	diff. seulement	spéc. seulement	ambi. et diff.	ambi. et spé.	diff. et spé.	ambi., diff., spé.
( - )	a	d	s	a+d	a+s	d+s	a+d+s	
i) La position de l'observateur influence la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ii) Les positions des autres objets de la scène influencent la ou les réflexions ...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
iii) Le coefficient de brillance influence la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
iv) La position de la source lumineuse influence la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
v) L'intensité de la source lumineuse influence la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
vi) Les propriétés de matériau de l'objet influencent la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
vii) Le vecteur normal à la surface influence la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
viii) La présence d'un autre objet situé entre l'objet éclairé et la source lumineuse influence la ou les réflexions ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

INP2705

**b) [12 points]** Nous avons vu que le modèle d'illumination de Gouraud interpole des intensités, tandis que le modèle d'illumination de Phong interpole des normales pour la coloration d'une primitive. En utilisant une simple scène 2D composée d'un unique segment de droite entre deux sommets P1 et P2 (comme illustré ci-dessous), on souhaite comparer le comportement de la lumière diffuse sur ce segment selon les deux modèles.



On suppose que :

- le segment est éclairé par une source de lumière positionnelle située au-dessus du segment en (0,4), à égale distance des deux sommets P1 et P2 situés en (-4,0) et (+4,0).
- il n'y a aucun sommet intermédiaire entre P1 et P2 ;
- aucune lumière ambiante ou spéculaire n'est émise par la source de lumière ;
- la source de lumière n'émet que de la lumière diffuse avec une intensité maximale ;
- le matériau du segment réfléchit toute la lumière diffuse reçue ;
- il n'y a aucune atténuation en fonction de la distance ;
- les normales N1 et N2 aux sommets P1 et P2 sont celles spécifiées à chacune des sous-questions.

Pour chacune des sous-questions suivantes, tracez deux courbes qui montrent l'intensité de l'illumination en fonction de  $x$  de P1 à P2 : une première courbe pour le modèle d'illumination de Gouraud et une seconde courbe pour le modèle d'illumination de Phong.

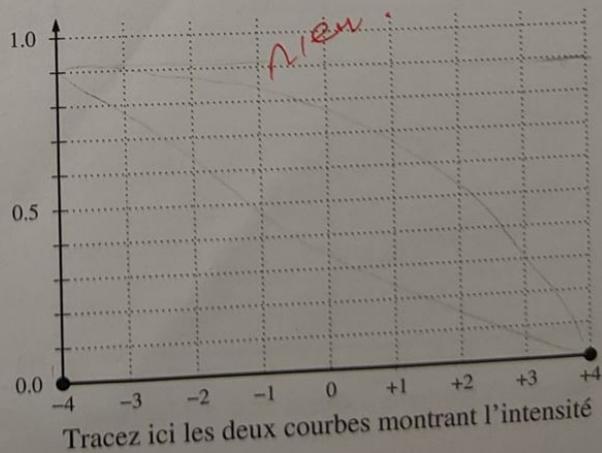
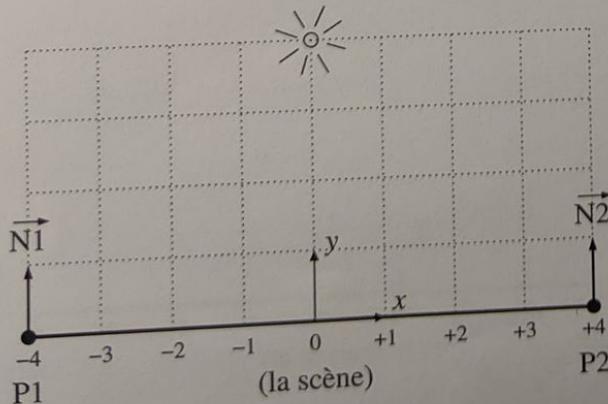
Prenez soin de bien montrer les positions où chaque courbe débute ou se termine, les endroits où l'intensité est la plus forte et où elle est la plus faible, de même que l'allure générale de la courbe entre ces positions.  
⇒ Il n'est pas nécessaire de faire de longs calculs pour tracer ces courbes. Un peu de réflexion suffira !

ambiant

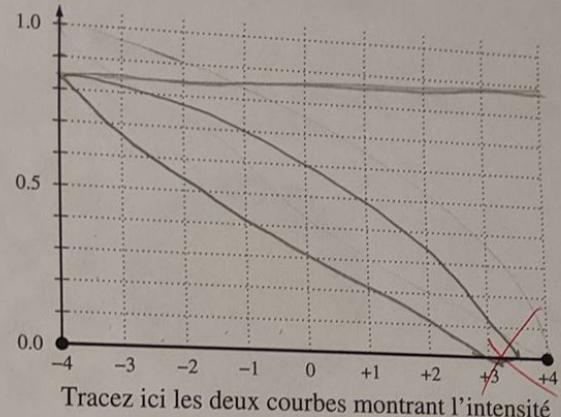
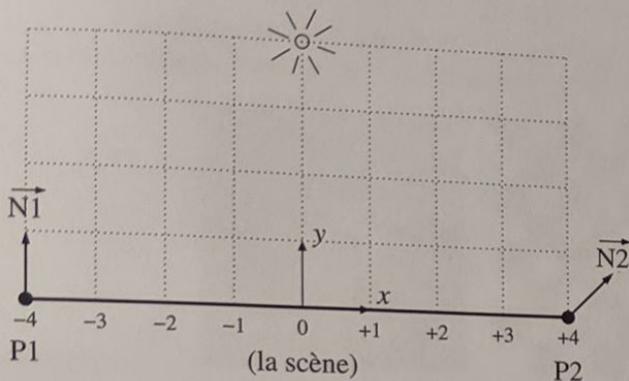
$$I_a = K_a I_{lumi}$$

$$\text{diffuse: } I_d = K_d I_{lumi} \cos(\theta)$$

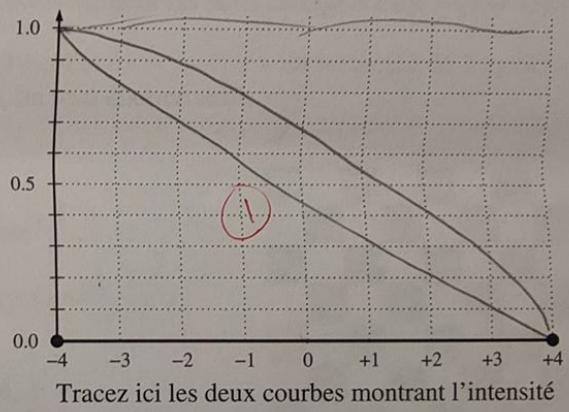
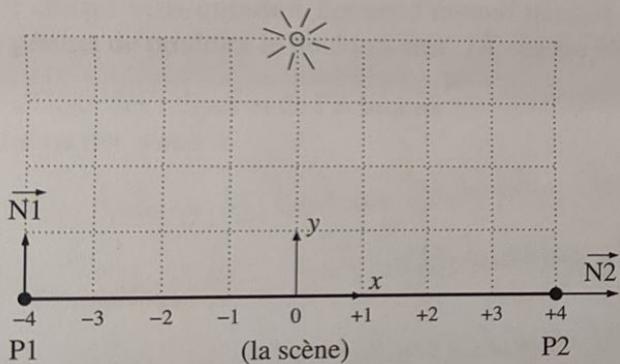
i) Les normales sont  $N1 = (0, 1)$  et  $N2 = (0, 1)$ .



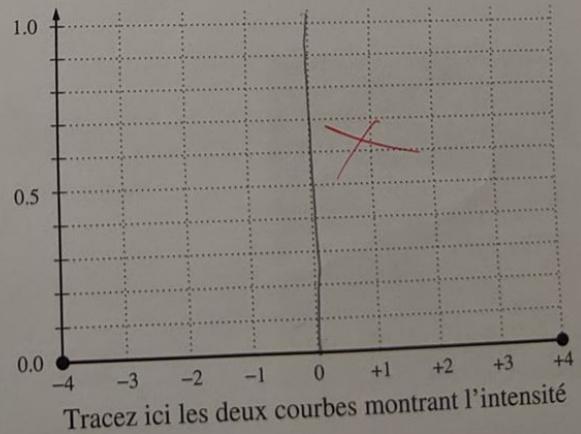
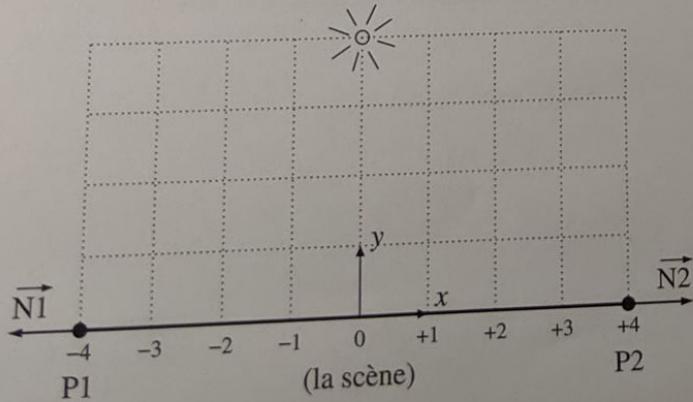
ii) Les normales sont  $N_1 = (0, 1)$  et  $N_2 = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$ .



iii) Les normales sont  $N_1 = (0, 1)$  et  $N_2 = (1, 0)$ .

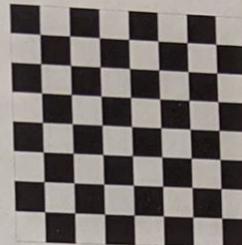
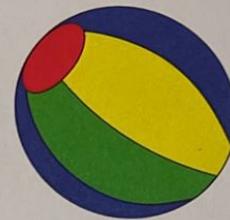


iv) Les normales sont  $N_1 = (-1, 0)$  et  $N_2 = (1, 0)$ .



#### Question 4 Utilisation des textures et opérations en GLSL [10 points]

Le programme listé à l'annexe A charge les images `Tulipes.bmp`, `Echiquier.bmp` et `Ballon.bmp` ci-dessous dans trois textures distinctes. Ce programme affiche un carré (deux triangles) qu'on souhaite texturer en utilisant ces trois textures afin de montrer l'effet de divers énoncés GLSL.

`Tulipes.bmp``Echiquier.bmp``Ballon.bmp`

Le programme de l'annexe A utilise les mêmes fonctions utilitaires que dans vos travaux pratiques et il n'a besoin d'aucune modification. Ce programme charge en mémoire les trois images, crée les trois textures, charge les nuanceurs donnés en fin d'annexe et dessine correctement les deux triangles dans le plan Z=0.

Pour chaque sous-question, écrivez l'énoncé modifié « `FragColor = ... ;` » du nuanceur de fragments qui permet de produire le rendu montré. (À chaque fois, un seul énoncé suffit.)

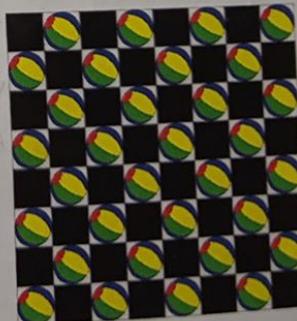
i) mélange des tulipes et de l'échiquier :

```
void main( void )
{
    Var Tulipes = texture(@textureTulipes, TexCoord);
    Var Echiquier = texture(@textureEchiquier, TexCoord);
    FragColor = mix(Tulipes, Echiquier) * 0,5;
    non
    ≠ mélange
}
```



ii) mélange des ballons et de l'échiquier :

```
void main( void )
{
    Var Ballon = texture(@textureBallon, TexCoord);
    Var Echiquier = texture(@textureEchiquier, TexCoord);
    FragColor = mix(Echiquier, Ballon) * 0,5
}
```



iii) affichage d'une partie des tulipes dans un cercle entouré de gris :

```
void main( void )
```

*TexCoord contient Texture coordonnée en cercle*

O  $FragColor = \text{Texture}(\text{laTextureTulipes}, \text{TexCoord}) + \text{Rgba}(\text{gris}),$



}

iv) mélange linéaire avec du blanc, semblable à un flash inopportun :

```
void main( void )
```

{

O  $FragColor = \text{color}.a + (\text{texture}(\text{laTextureTulipes}, \text{TexCoord}), 0, 0, 0),$

*X pas de blanc*



}

v) inversion du rouge et bleu pour de nouvelles couleurs de tulipes :

```
void main( void )
```

{

O  $FragColor = \text{mix}([\text{texTulipes}, \text{texCoord}], [(-r, g, -b, a)], 0.5)$



}



Bon printemps !

Cet examen comprend 4 questions sur 13 pages pour un total de 50 points.  
Benoît Ozell

## Annexe A

(Si vous le souhaitez, vous pouvez détacher et ne pas remettre cette annexe.)

Le programme principal et les nuanceurs utilisés pour la question 4 (page 8) :

```
#include "inf2705.h"

GLuint prog;
GLint locVertex, locTexCoord;
GLint locmatrModel, locmatrVisu, locmatrProj;
GLint loclaTextureTulipes, loclaTextureEchiquier, loclaTextureBallon;

// matrices de du pipeline graphique
MatricePipeline matrModel, matrVisu, matrProj;

// la fonction habituelle!
void chargerNuanceurs()
{
    // créer le programme
    prog = glCreateProgram();
    // attacher le nuanceur de sommets
    const GLchar *chainesSommets = ProgNuanceur::lireNuanceur("nuanceurSommets.glsl");
    if ( chainesSommets != NULL )
    {
        GLuint nuanceurObj = glCreateShader( GL_VERTEX_SHADER );
        glShaderSource( nuanceurObj, 1, &chainesSommets, NULL );
        glCompileShader( nuanceurObj );
        glAttachShader( prog, nuanceurObj );
        delete [] chainesSommets;
    }
    // attacher le nuanceur de fragments
    const GLchar *chainesFragments = ProgNuanceur::lireNuanceur("nuanceurFragments.glsl");
    if ( chainesFragments != NULL )
    {
        GLuint nuanceurObj = glCreateShader( GL_FRAGMENT_SHADER );
        glShaderSource( nuanceurObj, 1, &chainesFragments, NULL );
        glCompileShader( nuanceurObj );
        glAttachShader( prog, nuanceurObj );
        delete [] chainesFragments;
    }
    // faire l'édition des liens du programme
    glLinkProgram( prog );
    ProgNuanceur::afficherLogLink( prog );

    // demander la "Location" des variables
    locVertex = glGetAttribLocation( prog, "Vertex" );
    locTexCoord = glGetAttribLocation( prog, "TexCoord" );
    locmatrModel = glGetUniformLocation( prog, "matrModel" );
    locmatrVisu = glGetUniformLocation( prog, "matrVisu" );
    locmatrProj = glGetUniformLocation( prog, "matrProj" );
    loclaTextureTulipes = glGetUniformLocation( prog, "laTextureTulipes" );
    loclaTextureEchiquier = glGetUniformLocation( prog, "laTextureEchiquier" );
    loclaTextureBallon = glGetUniformLocation( prog, "laTextureBallon" );
}
}
```

```

bool chargerTexture( std::string fichier, GLuint &texture )
{
    GLsizei largeur, hauteur;
    unsigned char *pixels;
    if ( ( pixels = ChargerImage( fichier, largeur, hauteur ) ) != NULL )
    {
        glGenTextures( 1, &texture );
        glBindTexture( GL_TEXTURE_2D, texture );
        glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB,
                      largeur, hauteur, 0, GL_RGBA, GL_UNSIGNED_BYTE, pixels );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
        glBindTexture( GL_TEXTURE_2D, 0 );
        delete[] pixels;
    }
    return true;
}

void FenetreTP::initialiser()
{
    // charger les trois textures et retourner son identificateur dans maTexture...
    GLuint maTextureTulipes=0, maTextureEchiquier=0, maTextureBallon=0;
    chargerTexture( std::string("20191-tulipes.bmp"), maTextureTulipes );
    chargerTexture( std::string("20191-echiquier.bmp"), maTextureEchiquier );
    chargerTexture( std::string("20191-ballon.bmp"), maTextureBallon );

    // assigner chaque image dans une unité de texture différente
    glEnableTexture( GL_TEXTURE0 ); // l'unité de texture 0
    glBindTexture( GL_TEXTURE_2D, maTextureTulipes );
    glEnableTexture( GL_TEXTURE1 ); // l'unité de texture 1
    glBindTexture( GL_TEXTURE_2D, maTextureEchiquier );
    glEnableTexture( GL_TEXTURE2 ); // l'unité de texture 2
    glBindTexture( GL_TEXTURE_2D, maTextureBallon );

    // charger les nuanceurs (tous vos fichiers de nuanceurs seront chargés ici)
    chargerNuanceurs();

    // Le modèle
    const GLfloat coords[] = { -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0 };
    const GLfloat texcoo[] = { 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0 };
    const GLuint connec[] = { 0, 1, 2, 2, 3, 0 };
    // allouer les objets OpenGL
    GLuint vao[1]; glGenVertexArrays( 1, vao );
    GLuint vbo[3]; glGenBuffers( 3, vbo );
    // initialiser le VAO
    glBindVertexArray( vao[0] );
    // charger le VBO pour les sommets
    glBindBuffer( GL_ARRAY_BUFFER, vbo[0] );
    glBufferData( GL_ARRAY_BUFFER, sizeof(coords), coords, GL_STATIC_DRAW );
    glVertexAttribPointer( locVertex, 2, GL_FLOAT, GL_FALSE, 0, 0 );
    glEnableVertexAttribArray(locVertex);
    // charger le VBO pour la connectivité
    glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, vbo[1] );
    glBufferData( GL_ELEMENT_ARRAY_BUFFER, sizeof(connec), connec, GL_STATIC_DRAW );
}

```

```

// charger le VBO pour les coordonnées de texture
glBindBuffer( GL_ARRAY_BUFFER, vbo[2] );
glBufferData( GL_ARRAY_BUFFER, sizeof(texcoo), texcoo, GL_STATIC_DRAW );
glVertexAttribPointer( locTexCoord, 2, GL_FLOAT, GL_FALSE, 0, 0 );
 glEnableVertexAttribArray(locTexCoord);
}

void FenetreTP::afficherScene()
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    // assigner les variables uniformes
    glUseProgram( prog );
    glUniform1i( loclaTextureTulipes, 0 ); // unité de texture 0
    glUniform1i( loclaTextureEchiquier, 1 ); // unité de texture 1
    glUniform1i( loclaTextureBallon, 2 ); // unité de texture 2

    // transformation de projection orthogonale
    matrProj.Ortho( -1.0, 1.0, -1.0, 1.0, -1.0, 1.0 );
    glUniformMatrix4fv( locmatrProj, 1, GL_FALSE, matrProj );

    // transformation de visualisation
    matrVisu.LookAt( 0.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 1.0, 0.0 );
    glUniformMatrix4fv( locmatrVisu, 1, GL_FALSE, matrVisu );

    // transformation de modélisation
    matrModelLoadIdentity();
    glUniformMatrix4fv( locmatrModel, 1, GL_FALSE, matrModel );

    // afficher les deux triangles qui forment le carré (6 indices dans connec[])
    glDrawElements( GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0 );

    // permuter tampons avant et arrière
    swap();
}

...
int main( int argc, char *argv[] )
{
    // créer une fenêtre
    FenetreTP fenetre( "texture", 500, 500 );

    // allouer des ressources et définir le contexte OpenGL
    fenetre.initialiser();

    bool boucler = true;
    while ( boucler )
    {
        // affichage
        fenetre.afficherScene();
        // récupérer les événements et appeler la fonction de rappel
        boucler = fenetre.gererEvenement();
    }
    return 0;
}

```

*Le nuanceur de sommets "nuanceurSommets.gls1" utilisé dans le programme principal :*  
#version 410

```
uniform mat4 matrModel, matrVisu, matrProj;  
in vec4 Vertex;  
in vec2 TexCoord;  
out vec2 texCoord;  
  
void main( void )  
{  
    // transformation standard du sommet  
    gl_Position = matrProj * matrVisu * matrModel * Vertex;  
    // transmettre au nuanceur de fragments les coordonnées de texture reçues  
    texCoord = TexCoord;  
}
```

*Le nuanceur de fragments "nuanceurFragments.gls1" utilisé dans le programme principal :*  
#version 410

```
uniform sampler2D laTextureTulipes, laTextureBallon, laTextureEchiquier;  
// in vec4 gl_FragCoord; // variable pré définie qui contient la position du fragment  
in vec2 texCoord;  
out vec4 FragColor;  
  
void main( void )  
{  
    // afficher la texture des tulipes (À MODIFIER)  
    FragColor = texture( laTextureTulipes, texCoord );  
}
```