



# INF2705 Infographie

## Spécification des requis du système

### Travail pratique 3

### *Illumination, textures et tessellation*

Département de génie informatique et génie logiciel  
École polytechnique de Montréal  
Automne 2018

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	But . . . . .	2
1.2	Portée . . . . .	2
1.3	Remise . . . . .	2
<b>2</b>	<b>Description globale</b>	<b>3</b>
2.1	But . . . . .	3
2.2	Travail demandé . . . . .	3
<b>3</b>	<b>Exigences</b>	<b>9</b>
3.1	Exigences fonctionnelles . . . . .	9
3.2	Exigences non fonctionnelles . . . . .	9
3.3	Rapport . . . . .	10
<b>4</b>	<b>Liste des commandes</b>	<b>10</b>
<b>5</b>	<b>Figures supplémentaires</b>	<b>11</b>
<b>6</b>	<b>Apprentissage supplémentaire</b>	<b>11</b>
<b>7</b>	<b>Formules utilisées</b>	<b>12</b>
7.1	Modèles de réflexion spéculaire de Phong et de Blinn . . . . .	12
7.2	Modèles de spot inspirés d'OpenGL ou de Direct3D . . . . .	13

# 1 Introduction

Ce document décrit les exigences fonctionnelles et non fonctionnelles du TP3 « *Illumination, textures et tessellation* » du cours INF2705 Infographie.

## 1.1 But

Le but des travaux pratiques est de permettre à l'étudiant d'appliquer directement les notions vues en classe.

## 1.2 Portée

Chaque travail pratique permet à l'étudiant d'aborder un sujet spécifique.

## 1.3 Remise

Faites la commande « `make remise` » afin de créer l'archive « **INF2705\_remise\_TPn.zip** » que vous déposerez ensuite dans Moodle. (Moodle ajoute automatiquement vos matricules ou le numéro de votre groupe au nom du fichier remis.)

Ce fichier zip contient le fichier Rapport.txt et tout le code source du TP (`makefile`, `*.h`, `*.cpp`, `*.glsl`, `*.txt`).

## 2 Description globale

### 2.1 But

Le but de ce TP est de permettre à l'étudiant mettre en pratique l'illumination des objets, l'utilisation d'un spot et d'appliquer des textures en utilisant des nuanceurs en GLSL. Ce travail pratique permet aussi de se familiariser avec les nuanceurs de tessellation.

### 2.2 Travail demandé

#### Partie 1 : l'illumination des objets

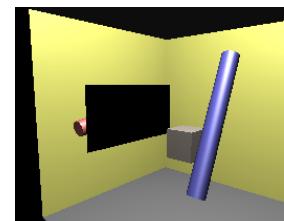
On demande de calculer l'illumination sur divers objets en utilisant deux sources de lumière (figure 1). Comme vu au cours, l'effet des deux sources de lumière est cumulatif et augmente les réflexions aux endroits doublement éclairés. Pour tous les objets, les modèles d'illumination de Gouraud et de Phong (figure 2) seront mis en œuvre et on pourra aussi choisir, pour le calcul de la réflexion spéculaire, entre le modèle de Phong ou celui de Blinn (figure 3).

Vous observerez que l'illumination de Gouraud sur le cube montre bien les limites de cette approche. Les couleurs sont calculées aux sommets de chaque face du cube et interpolées sur toute la face à partir de ces valeurs aux coins. Mais puisqu'il n'y a que deux triangles par face, la réflexion spéculaire n'est jamais très « belle », voire même inexiste : ça dépend s'il y a ou non de la réflexion spéculaire aux coins du cube. Et s'il a réflexion spéculaire au coin, alors l'intensité est alors simplement interpolée linéairement sur la face. (Modifiez le point de vue pour bien constater cet effet ... pas très souhaitable.) Ce problème de rendu dû à la faible résolution des faces sur le cube sera résolu, à la partie 4 du TP, par l'utilisation de nuanceurs de tessellation afin de subdiviser les faces du cube (figure 10). En attendant d'utiliser la tessellation, on acceptera ce défaut d'affichage sur le cube.

Pour démarrer, on pourra utiliser les nuanceurs des exemples du cours :

[cours.polymtl.ca/inf2705/exemples/06-IlluminationMiroir/](http://cours.polymtl.ca/inf2705/exemples/06-IlluminationMiroir/) et

[cours.polymtl.ca/inf2705/exemples/06-Illumination/](http://cours.polymtl.ca/inf2705/exemples/06-Illumination/)



Note : Ce TP utilise des « *Uniform Buffer Object* » (UBO) afin de transférer en bloc les variables uniformes aux nuanceurs. L'usage des UBO est très semblable aux VBO et permet surtout que le passage des valeurs des variables uniformes aux nuanceurs soit beaucoup plus efficace. Dans ce TP, on utilise ainsi quatre UBO qui correspondent aux quatre blocs de variables uniformes utilisés dans les nuanceurs : LightSource, FrontMaterial, LightModel et varsUnif. Les trois premiers blocs contiennent les variables uniformes servant à l'illumination, tandis que le dernier bloc contient les variables uniformes de l'application. (Les noms des variables sont inspirés de OpenGL 2.x.)

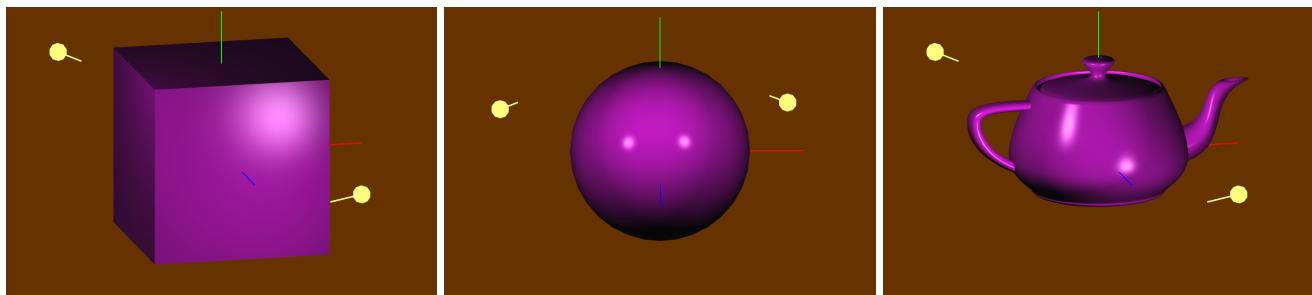


FIGURE 1 – Divers objets illuminés.

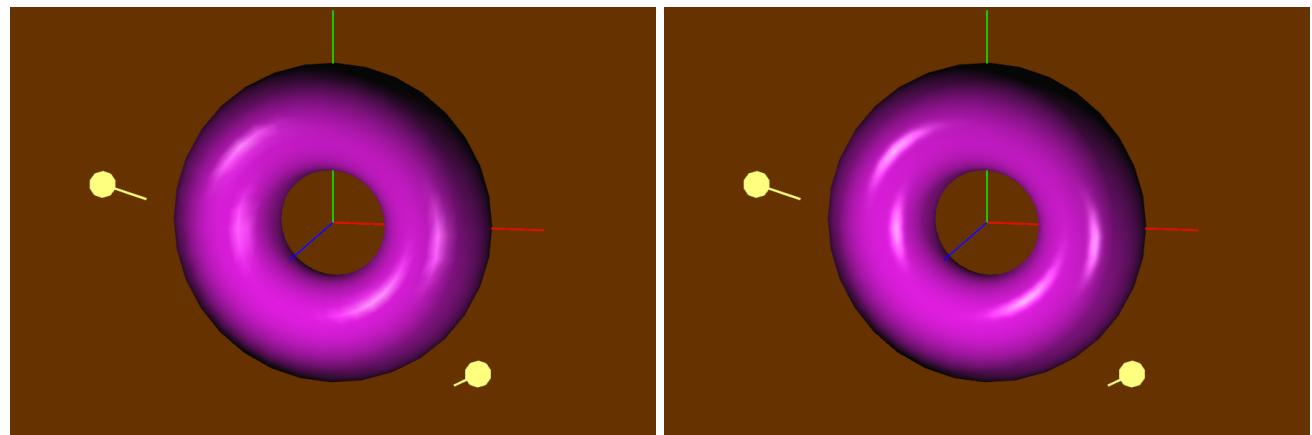


FIGURE 2 – Rendu avec réflexion spéculaire de Phong : a) illumination de Gouraud, b) illumination de Phong.

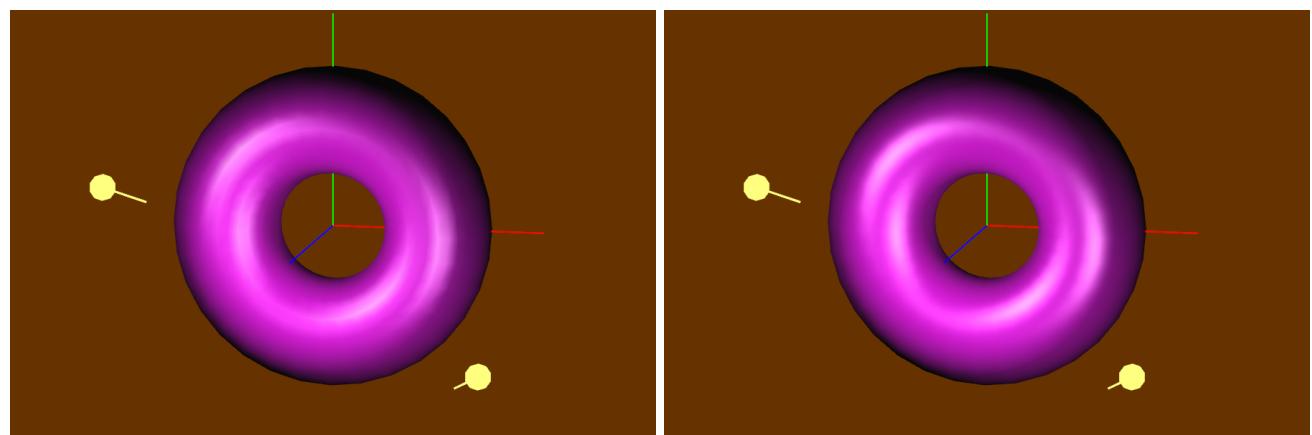


FIGURE 3 – Rendu avec réflexion spéculaire de Blinn : a) illumination de Gouraud, b) illumination de Phong.

## Partie 2 : l'utilisation d'un spot

Le programme permettra aussi d'utiliser un éclairage de type « spot » selon une définition semblable à celle d'OpenGL ou à celle de Direct3D (figure 4). La section 7 décrit les formules applicables. Les propriétés du spot, sa position, son angle maximum (`spotAngleOuverture`) et son exposant (`spotExponent`) sont modifiables en cours de l'exécution et seront correctement utilisés.

Lorsqu'on change le point de vue, la source de lumière se déplace avec le modèle et c'est toujours la même région qui est illuminée peu importe la position de la caméra. Bien sûr, la réflexion spéculaire changera selon la position de la caméra, mais pas la réflexion diffuse, ni l'angle du spot.

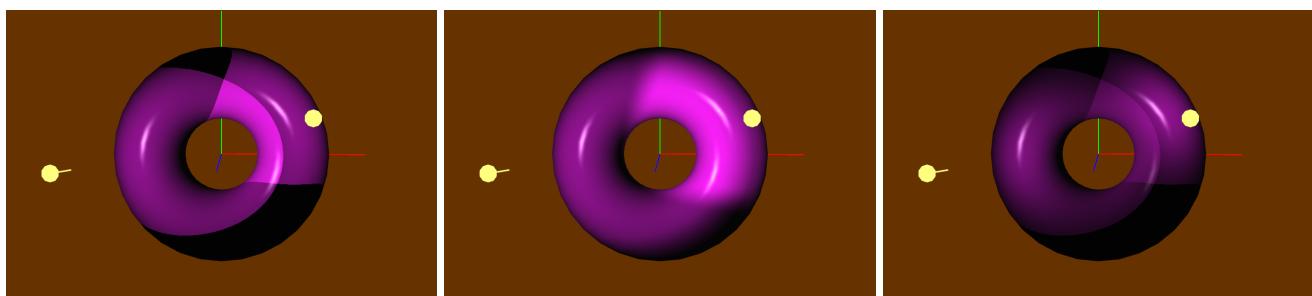


FIGURE 4 – Rendu avec le spot a) avec le modèle du style OpenGL, b) avec le modèle du style Direct3D, c) avec un plus grand exposant

## Partie 3 : l'application de textures

Le logiciel permettra d'afficher des textures (figure 11) sur les objets illuminés.

- Le dé à jouer sur le cube : on spécifiera les coordonnées de texture afin de montrer sur le cube un dé à jouer. La texture contenant toutes les faces du cube est fournie et elle sera utilisée sans la subdiviser en 6 textures différentes (figure 5).
- Le patron d'échiquier sur le cube : on spécifiera les coordonnées de texture afin de montrer l'échiquier répété 3 fois dans les deux directions sur chaque face (figure 6.)
- Les autres textures sur le cube : on spécifiera les coordonnées de texture afin de montrer la texture directement sur chaque face ( figure 6).
- Les autres objets (tore, sphère, théière, cube, cylindre, cône) définissent eux-mêmes leurs coordonnées de texture.

L'utilisateur pourra choisir son type de rendu en tenant compte de la couleur de base de l'objet ou d'une couleur grise uniforme `grisUniforme = vec4(0.7, 0.7, 0.7, 1.0)` (semblable à la figure 6). Enfin, on pourra choisir que les texels foncés de la texture ( $rgb < 0.5$ ) soient affichés normalement, colorés (la moyenne de la couleur de la texture et de la couleur de l'objet) ou complètement transparents (figures 5, 7 et 8).

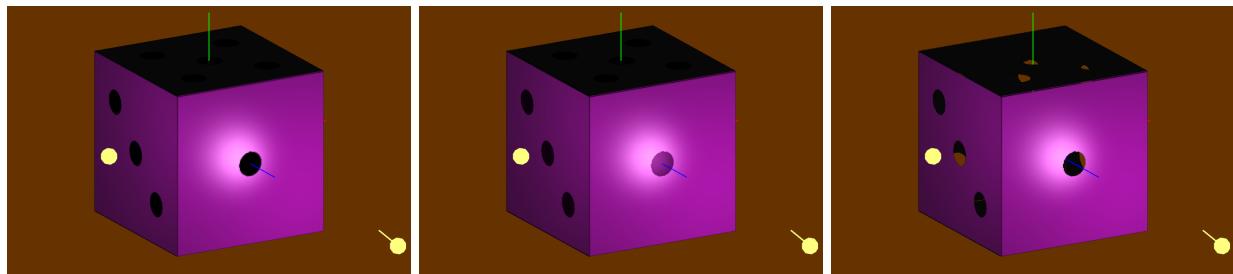


FIGURE 5 – Texture appliquée sur le dé en 3D (texels foncés opaques, colorés ou transparents)

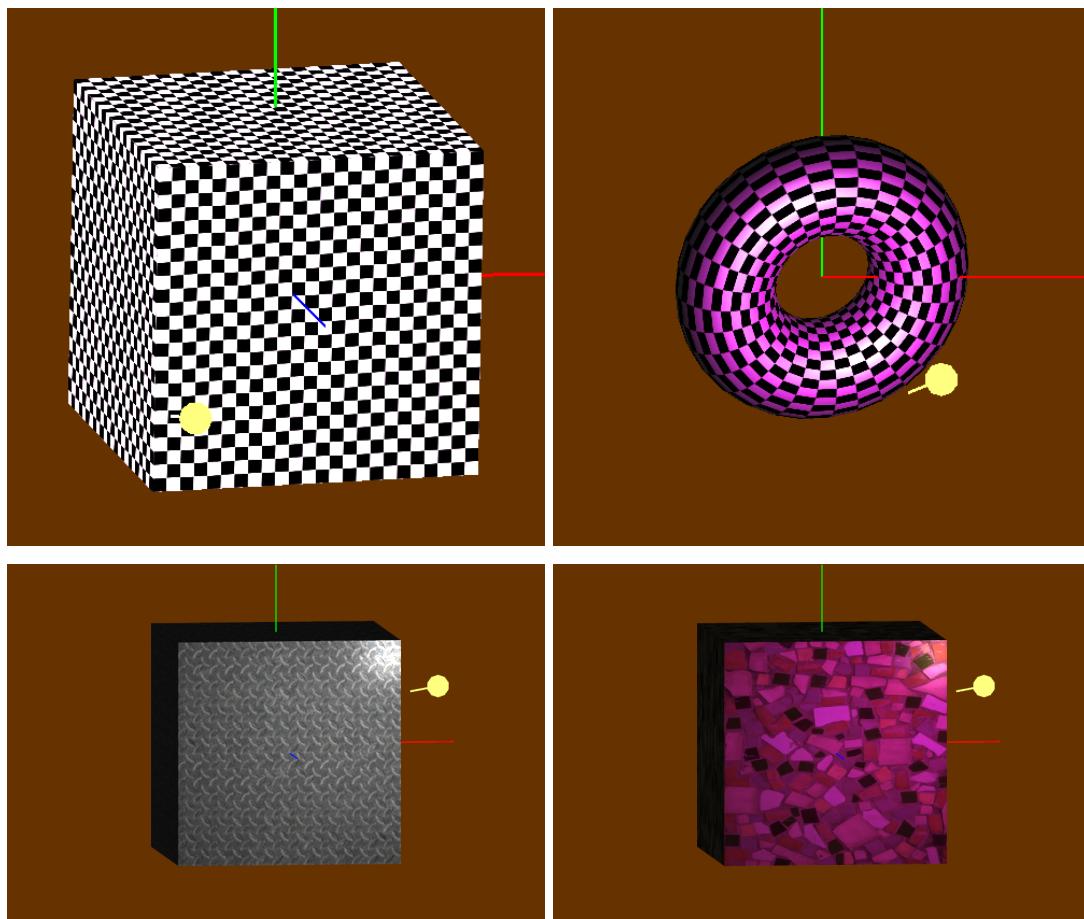


FIGURE 6 – Texture échiquier appliquée sur le cube, sur le tore et sur la théière

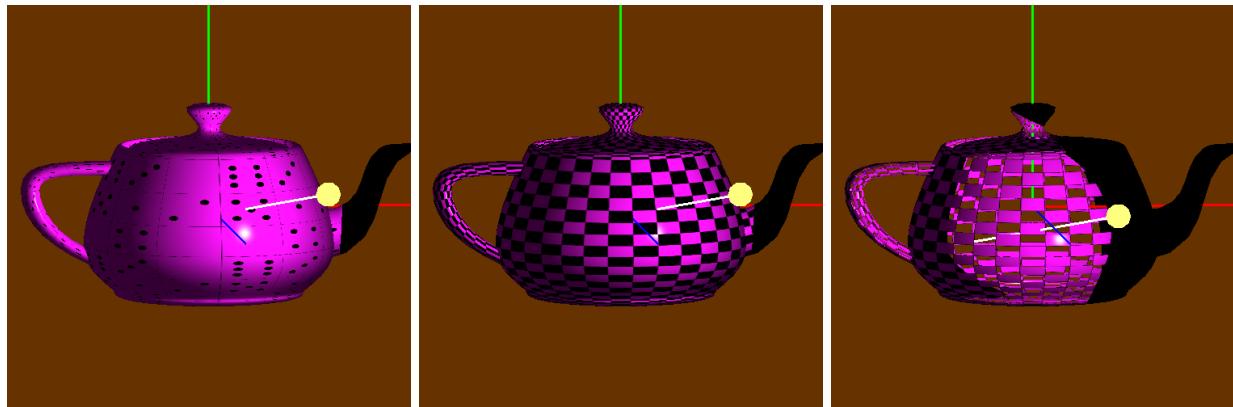


FIGURE 7 – La théière texturée (texels foncés opaques ou transparents)

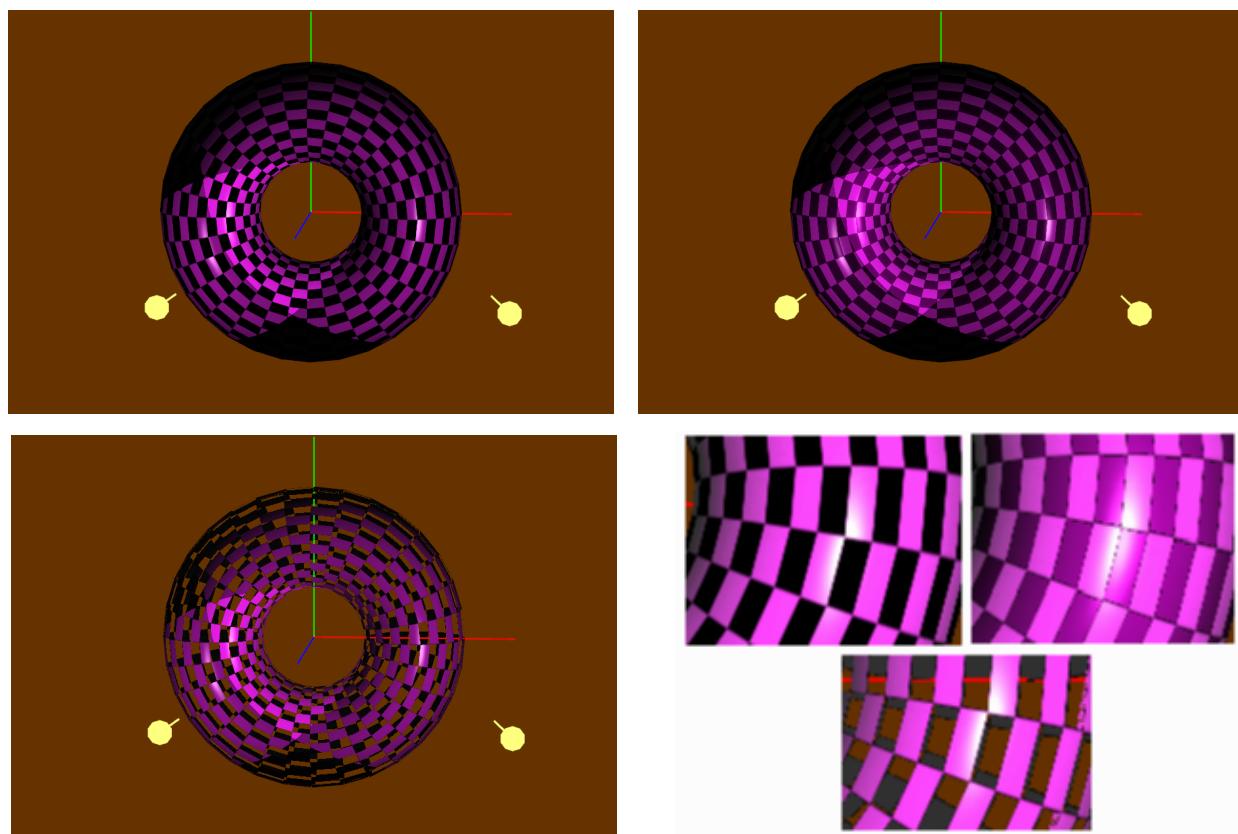


FIGURE 8 – Les texels foncés peuvent être affichés normalement, colorés ou complètement transparents

#### Partie 4 : l'utilisation des nuanceurs de tessellation

On aura remarqué que le modèle de Gouraud n'offre pas un très beau rendu graphique lorsque la surface du cube n'est composée que seulement de deux triangles. Afin de corriger cet effet, on utilisera des nuanceurs de tessellation pour subdiviser chaque face du cube et fournir un plus beau rendu lorsque l'algorithme de Gouraud est utilisé. Il faudra d'abord activer les nuanceurs de tessellation dans le programme principal et ensuite utiliser des GL\_PATCHES afin d'afficher les faces du cube qui seront alors subdivisées en un nombre variable de sous-triangles (figures 9 et 10).

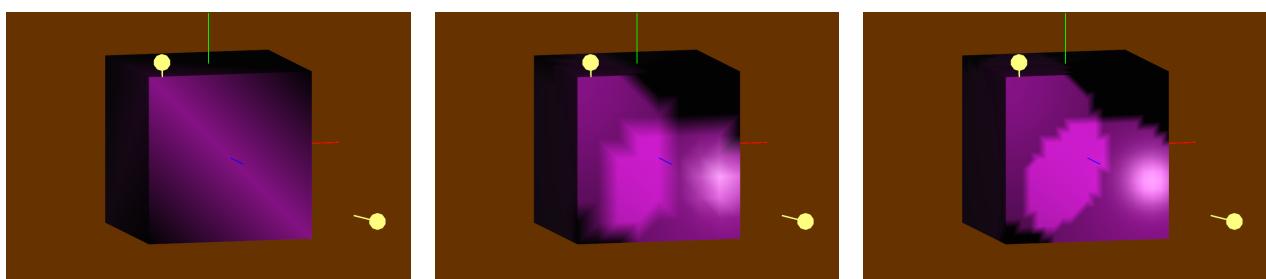


FIGURE 9 – La face du cube avec des niveaux de tessellation de 1, 8 ou 20 (plein)

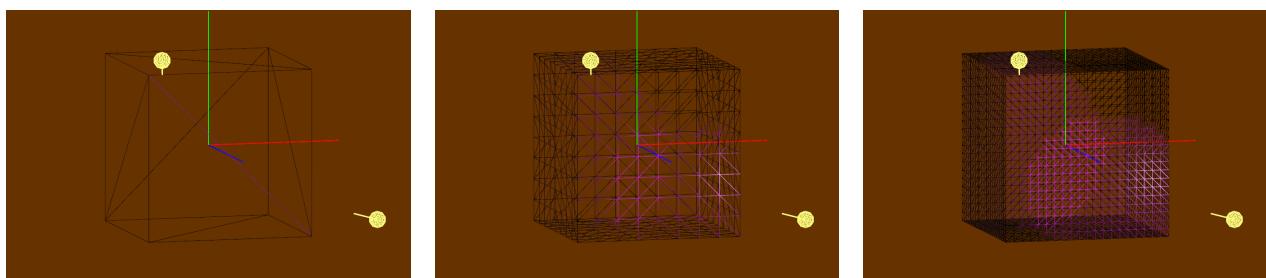


FIGURE 10 – La face du cube avec des niveaux de tessellation de 1, 8 ou 20 (fil de fer)

## 3 Exigences

### 3.1 Exigences fonctionnelles

Partie 1 :

- E1. Le modèle d'illumination de Phong est implanté comme demandé.
- E2. Le modèle d'illumination de Gouraud est implanté comme demandé.
- E3. Les modèles de réflexion de Phong et de Blinn sont implémentés comme demandé.

Partie 2 :

- E4. Le modèle de spot inspiré d'OpenGL est implémenté comme demandé.
- E5. Le modèle de spot inspiré de Direct3D est implémenté comme demandé.
- E6. Les modifications des propriétés du spot sont visibles (ex. : position, orientation, taille du cône).

Partie 3 :

- E7. Les paramètres des textures sont bien initialisés et elles sont correctement affichées sur les objets.
- E8. Les objets texturés sont correctement illuminés.
- E9. Le cube est affiché correctement avec la texture du dé (figure 5).
- E10. Le cube et les autres objets sont affichés correctement avec la texture de l'échiquier (figure 6).
- E11. Les texels foncés sur l'objet sont affichés normalement, mi-colorés ou transparents (figure 8).
- E12. Permettre l'affichage de l'objet texturé avec ou sans sa couleur (ex. : figures 6).

Partie 4 :

- E13. Des nuanceurs de tessellation sont utilisés pour corriger le rendu illuminé avec le modèle de Gouraud.

### 3.2 Exigences non fonctionnelles

Notez bien que, normalement, on chargerait des nuanceurs différents pour chaque cas d'utilisation afin d'augmenter la performance en évitant les énoncés conditionnels à la valeur d'une variable uniforme.

Toutefois, dans le contexte de TP, on utilisera sciemment de tels énoncés conditionnels aux valeurs des variables uniformes (modifiables interactivement) : le modèle d'illumination, le modèle de spot, si on veut des pixels transparents, etc. Ceci permettra de plus facilement contrôler le type de rendu, en plus de faciliter votre développement... et la correction !

Dans vos nuanceurs, on pourra donc voir des énoncés semblables à ceux-ci :

```
if ( variableUniforme == ... ) ... else ... ;
```

ou, mieux encore :

```
( variableUniforme == ... ) ? ... : ...
```

### 3.3 Rapport

Vous devez répondre aux questions dans le fichier Rapport.txt qui sera inclus dans la remise. Vos réponses doivent être complètes et suffisamment détaillées. (Quelqu'un pourrait suivre les instructions que vous avez écrites sans avoir à ajouter quoi que ce soit.)

## 4 Liste des commandes

Touche	Description
q	Quitter l'application
x	Activer/désactiver l'affichage des axes
9	Permuter l'utilisation des nuanceurs de tessellation
v	Recharger les fichiers des nuanceurs et recréer le programme
i	Augmenter le niveau de tessellation interne
k	Diminuer le niveau de tessellation interne
o	Augmenter le niveau de tessellation externe
l	Diminuer le niveau de tessellation externe
u	Augmenter les deux niveaux de tessellation
j	Diminuer les deux niveaux de tessellation
p	Permuter la projection : perspective ou orthogonale
w	Alterner entre le modèle d'illumination : Gouraud, Phong
r	Alterner entre le modèle de réflexion spéculaire : Phong, Blinn
s	Alterner entre le modèle de spot : OpenGL, Direct3D
a	Incrémenter l'angle d'ouverture du cône du spot
z	Décrémenter l'angle d'ouverture du cône du spot
d	Incrémenter l'exposant du spot
e	Décrémenter l'exposant du spot
y	Incrémenter le coefficient de brillance
h	Décrémenter le coefficient de brillance
m	Choisir le modèle affiché : cube, tore, sphère, théière, cylindre, cône
0	Replacer Caméra et Lumière afin d'avoir une belle vue
t	Choisir la texture utilisée : aucune, dé, échiquier, métal, mosaique
c	Changer l'affichage de l'objet texturé avec couleur ou sans couleur
f	Changer l'affichage des texels foncés (normal, mi-coloré, transparent)
POINT	Augmenter l'effet du déplacement
BARREOBLIQUE	Diminuer l'effet du déplacement
g	Permuter l'affichage en fil de fer ou plein
n	Utiliser ou non les normales calculées comme couleur (pour le débogage)
ESPACE	Permuter la rotation automatique du modèle
BOUTON GAUCHE	Tourner l'objet
BOUTON MILIEU	Modifier l'orientation du spot
BOUTON DROIT	Déplacer la lumière
Molette	Changer la taille du spot

## 5 Figures supplémentaires

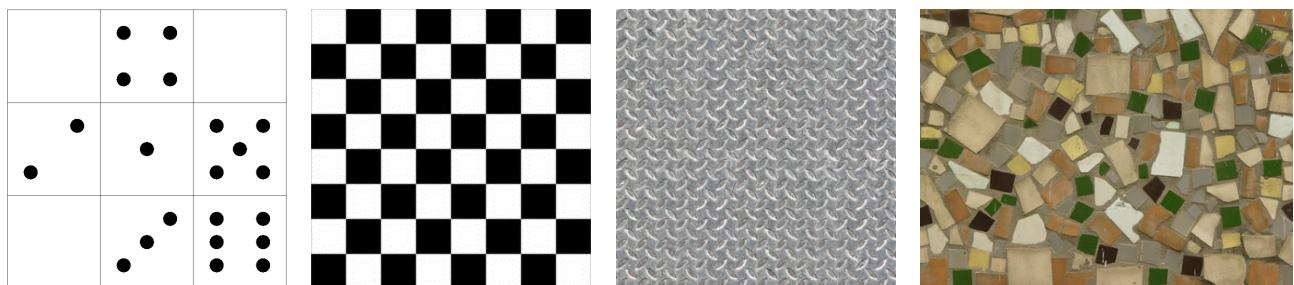


FIGURE 11 – Les textures fournies

## 6 Apprentissage supplémentaire

Partie 1 :

1. Valider les réflexions individuelles de chaque composante de la lumière (ambiante, diffuse, spéculaire) en commentant sélectivement les lignes qui calculent la contribution de chaque composante dans le nuanceur.
2. Comparer visuellement les différences entre les modèles de Phong et Blinn, en particulier pour une lumière rasante.
3. Comment ferez-vous pour implanter le modèle d'illumination de Lambert ?
4. Dans un nuanceur, utilisez des couleurs de lumière diffuse différentes pour chacun des sources lumineuses. Comment se mélangent les couleurs à l'écran ?
5. Afficher une sphère au lieu du cube. (Vous devrez alors calculer et spécifier ses normales.)

Partie 2 :

6. Utiliser une atténuation en fonction de la distance comme prévu dans le modèle d'illumination.
7. Plutôt que de mettre en noir les fragments non directement éclairés par le spot, diminuez simplement leur intensité par un facteur de 2.
8. Définissez un autre modèle de spot (d'autres paramètres et fonction de calcul) qui vous semblerait intéressant.

Partie 3 :

9. Modifier les coordonnées de texture pour constater l'effet sur le résultat visuel.
10. Définir les bonnes coordonnées de texture pour un objet plus complexe composé de triangles.
11. Modifier les coordonnées de texture afin de déplacer la texture en fonction du temps.

Partie 4 :

12. Rapetisser légèrement chaque triangle afin de voir à l'intérieur du cube entre les triangles.
13. Déformer la surface du cube différemment, par exemple pour la convertir en ellipsoïde.

## 7 Formules utilisées

### 7.1 Modèles de réflexion spéculaire de Phong et de Blinn

Le calcul de la réflexion spéculaire fait intervenir un produit scalaire entre deux vecteurs. La différence entre les modèles de Phong et de Blinn réside dans le choix des deux vecteurs utilisés :

- Phong utilise :  $\vec{R} \cdot \vec{O} = \text{reflect}(-\vec{L}, \vec{N}) \cdot \vec{O}$
- Blinn utilise :  $\vec{B} \cdot \vec{N} = \text{bissectrice}(\vec{L} \text{ et } \vec{O}) \cdot \vec{N} = \text{normalise}(\vec{L} + \vec{O}) \cdot \vec{N}$

où :

- $\vec{N}$  : normale à la surface
- $\vec{L}$  : direction du point vers la source lumineuse
- $\vec{R}$  : direction du rayon réfléchi  $= \text{reflect}(-\vec{L}, \vec{N})$ , si  $\vec{L}$  et  $\vec{N}$  sont unitaires.
- $\vec{O}$  : direction du point vers l'observateur
- $\vec{B}$  : bissectrice entre les vecteurs  $\vec{L}$  et  $\vec{O}$   $= \text{normalise}(\vec{L} + \vec{O})$ , si  $\vec{L}$  et  $\vec{O}$  sont unitaires.

Tous les calculs d'illumination se font dans le repère de la caméra en GLSL.

- Le calcul de la direction vers l'observateur ( $\vec{O}$ ) :
- (un vecteur qui pointe vers le (0,0,0), c'est-à-dire vers la caméra)

```
AttribsOut.obsVec = normalize(-pos); // = (0 - pos)
```

- La direction de la lumière ( $\vec{L}$ ) :

```
AttribsOut.lumiDir[j] = (matrVisu * LightSource.position[j]).xyz - pos;
```

- Le calcul du vecteur de la direction du spot ( $\vec{D}$ ) :
- (on doit le multiplier par la transposée de l'inverse de matrVisu)

```
AttribsOut.spotDir[j] = transpose(inverse(mat3(matrVisu))) *  
-LightSource.spotDirection[j];
```

## 7.2 Modèles de spot inspirés d'OpenGL ou de Direct3D

Un spot n'éclaire qu'à l'intérieur d'un cône, c'est-à-dire a une influence seulement si l'angle  $\gamma$  entre la direction du spot et la direction vers le point à éclairer est plus petit que l'angle d'ouverture  $\delta$  du spot. Lorsque c'est le cas, on a «  $\gamma < \delta$  » et mais on vérifiera plutôt si «  $\cos(\gamma) > \cos(\delta)$  » en évaluant des *produits scalaires* entre les vecteurs appropriés.

Pour correctement déterminer la direction du spot dans le repère de la caméra, on peut calculer l'inverse et la transposée du vecteur direction directement dans le nuanceur :

```
AttribsOut.spotDir[j] = transpose(inverse(mat3(matrVisu))) *
                        -LightSource.spotDirection[j];
```

La différence entre les modèles inspirés d'OpenGL et de Direct3D que nous utiliserons réside dans la formule pour calculer le facteur qui multiplie l'intensité lumineuse du spot à l'intérieur du cône :

- OpenGL utilise le facteur :  $\text{fact} = (\cos(\gamma))^c$
- Direct3D utilise le facteur :  $\text{fact} = \text{smoothstep}(\cos(\theta_{outer}), \cos(\theta_{inner}), \cos(\gamma))$

où :

$\cos(\delta)$ :	cosinus de l'angle d'ouverture	$= \cos(\text{LightSource.spotAngleOuverture})$
$\vec{L}_n$ :	direction du spot	$= \text{LightSource.spotDirection}[j]$
$c$ :	exposant du spot	$= \text{LightSource.spotExponent}$
$\cos(\gamma)$ :	est obtenue par	$(\vec{L} \cdot \vec{L}_n)$
$\cos(\theta_{inner})$ :	est remplacé <i>dans ce TP</i> par	$\cos(\delta)$
$\cos(\theta_{outer})$ :	est remplacé <i>dans ce TP</i> par	$(\cos(\delta))^{1.01+c/2}$