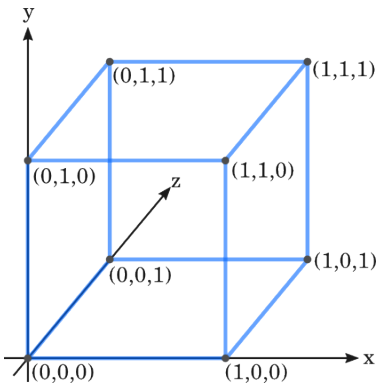
Examen Graphique 3D 2018-2019

L'examen dure 2 heures. Une feuille A4 rector verso autorisée. Aucun matériel électronique ou communicant autorisé. Justifier les réponses en donnant des détails, en nommant précisément et justifiant les techniques utilisées, et au besoin en dessinant un shéma. Le maximum de point est accordé uniquement pour les réponses justes ET précises.

1 Pipeline graphique (3pts)

Nous effectuons le rendu d'un cube illustré en Figure. 1 (gauche) avec les shaders associés (droite), dans une fenêtre de taille 512x512. Le maillage correspondant est défini par un index array, chaque face y est décomposée en 2 triangles. Notez aussi que le « backface culling » et le z-buffering ne sont pas utilisés (les faces avant et arrières d'un polygone sont toutes affichées).




```glsl
// vertex shader
layout(location = 0) in vec3 position;
out vec4 color;

void main() {
  gl_Position = vec4(position,1);
  color = vec4(vec3(1,1,0)-position,1);
}

// fragment shader
in vec4 color;
out vec4 outColor;

void main() {
  outColor = color;
}
```

FIGURE 1 – Gauche : le cube dessiné. Droite : les shaders utilisés pour le rendu.

1. Combien de fois le vertex shader sera-t-il exécuté ?
2. Combien de fois le fragment shader sera-t-il exécuté ?
3. Quelle sera la couleur affichée au centre de l'écran (i.e. en coordonnée 256, 256) ?
4. Quelle sera la couleur affichée au 3/4 en haut à droite de l'écran (i.e. en coordonnée 384, 384) ?

2 Shading (4pts)

Les shaders utilisés pour le rendu du modèle de singe (Figure 3) sont donnés en Figure 4. Chaque rendu correspond à la sortie obtenue d'une des variables c[numéro].

1. Donnez la correspondance entre ces variables et le numéro du rendu associé dans la figure.
2. Quelle expression correspond au modèle d'illumination diffus / Lambertien ?
3. Quelle expression correspond au modèle d'illumination de Phong ?

3 Textures (3pts)

Nous souhaitons mettre en place un sol texturé, illustré en Figure 5 (3,4).

1. Notre premier but est de créer la texture elle-même, telle que montrée dans la Figure 5.1. En supposant déjà disposer d'une fonction *perlin(u,v)* qui retourne une valeur de bruit $\in [0,1]$, proposez un pseudo-algorithme pour reproduire cette texture, en utilisant seulement les coordonnées (u, v) en paramètres d'entrée.
2. Une fois la texture créée, nous souhaitons la plaquer sur une grille régulière de polygones (quadrilatères) dans le plan $y = 0$, afin de créer des formes semblables à des diamants (Figure 5.2). Quelles doivent être les coordonnées (u, v) pour chaque point $P1, P2, P3, P4$ de la figure pour obtenir le résultat illustré dans celle-ci ?
3. Quelle est la méthode d'extension de la texture en dehors de $[0,1]$ pour obtenir le rendu de la figure ?
4. Quelles méthodes/paramètres de filtrage ont été utilisés pour obtenir les rendus des Figure 5.3 and 5.4 respectivement ?

4 Cas d'étude : jeu de voitures

Nous souhaitons réaliser un petit jeu de voiture illustré en Figure 2.

4.1 Géometrie et modélisation (3pts)

1. Donnez les définitions des matrices de projection, de vue (view) et modèle (model).
2. La rangée d'arbres de la Figure 2.3 est constituée d'arbres tous alignés selon une direction avec la même coordonnée $x = x_t$, et espacés régulièrement de s unités du repère monde, avec x_t et s des constantes prédéfinies. Le modèle d'arbre utilisé pour tous les arbres est supposé défini dans un repère dont l'origine $(0, 0, 0)$ correspond à la base du tronc de l'arbre au sol. Donnez la forme générale de la matrice model du i-ème arbre en fonction de x_t et s.
3. Les rendus de notre jeu de voiture seront obtenus du point de vue subjectif de la voiture, avec une caméra purement horizontale paramétrée par une rotation d'angle θ autour de l'axe y, une translation 2D x, z pour le placer au sol (le plan du sol étant en $y = 0$), et une hauteur constante y_v par rapport au sol. Donnez la forme de la matrice de vue (view) en fonction des paramètres mentionnés.

4.2 Animation (4pts)

Nous souhaitons animer les véhicules des Figures 2.(1, 2) ainsi que les arbres du jeu pour améliorer le réalisme.

1. Les trajectoires des véhicules autres que celui du joueur doivent être générées automatiquement par le jeu. Quelle technique peut être utilisée pour générer leurs trajectoires et comment peut-on les paramétrer ?
2. Nous souhaitons détecter lorsque le véhicule du jouer heurte un autre véhicule du jeu. Quelle technique pourrait-on utiliser pour détecter précisément ces collisions ? Comment la rendrait-on rapide ?
3. Si deux véhicules entrent en collision dans le jeu nous décidons d'animer la collision comme un simple écrasement unidimensionnel dans l'axe de la collision accompagné d'un changement de trajectoire. Comment paramétrer et animer cet effet ?
4. Proposez un modèle d'animation pour les arbres. Décrivez quelles primitives supporteraient l'animation, comment celle-ci est paramétrée, et comment l'état est mis à jour à chaque nouvel affichage du jeu.

4.3 Effets avancés / composites (3pts)

1. Décrivez une technique pour calculer les ombres projetées par les objets sur le plan du sol observées dans la Figure 2. Expliquez précisément ce qui doit être calculé et/ou peut être précalculé.
2. Nous souhaitons améliorer le réalisme du ciel dans le jeu. Décrivez et expliquez une technique qui irait dans ce sens. Expliquez comment cette technique affecterait la géométrie, le rendu ou les animations de la scène, quels aspects sont calculés en temps réel, et ceux pouvant être précalculés.
3. Quelle technique peut-on utiliser pour améliorer l'aspect des bâtiments dans la Figure 2.1 ?

1 2 3

FIGURE 2 – Trois vues du jeu de voitures envisagé. *(Tirées de Stamminger et Drettakis, Siggraph 2002)*

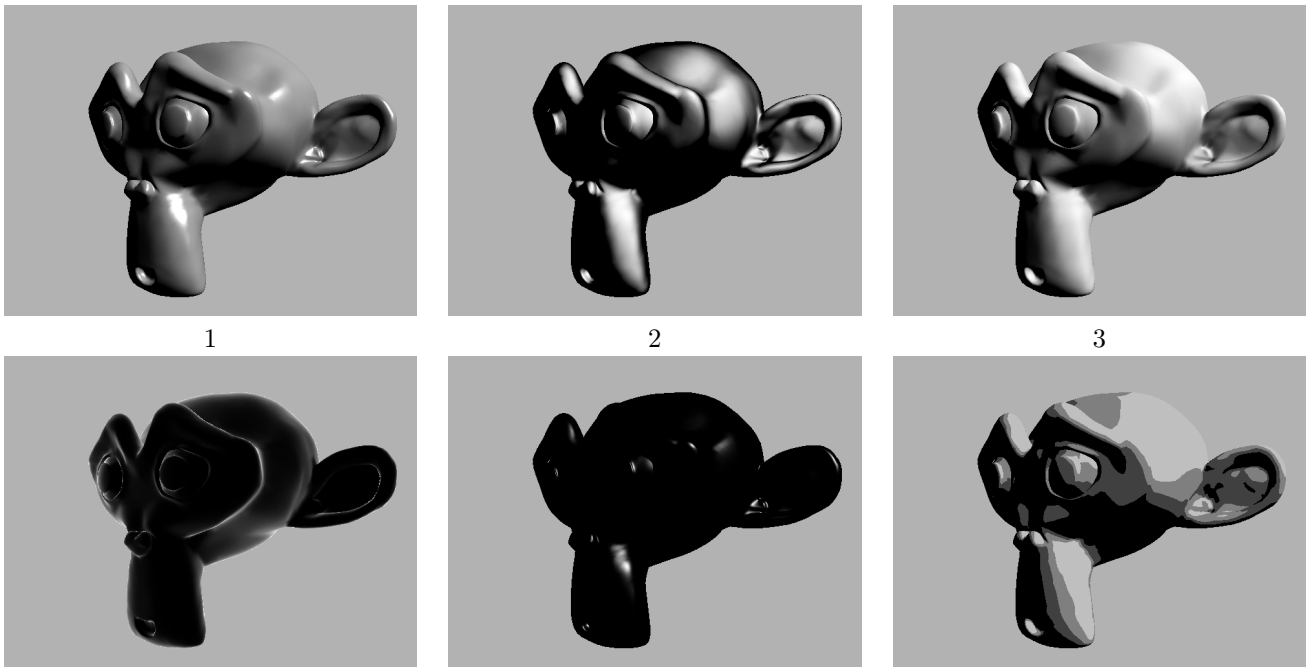
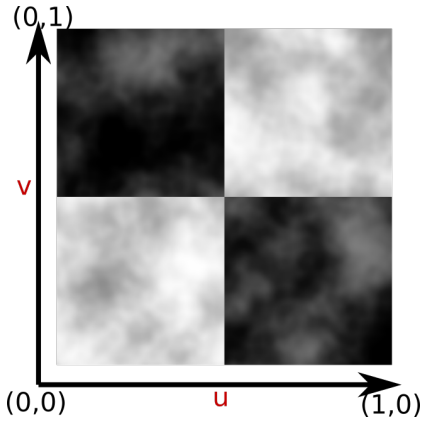


FIGURE 3 – Rendus obtenus avec les shaders de la Figure 4.

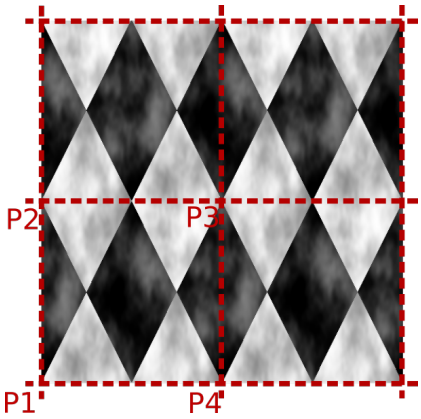
```glsl
#version 330 core
uniform mat4 projection, view, model;

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;

out vec3 wPosition, wNormal;

void main(){
  vec4 wPosition4 =  model * vec4(position, 1.0);
  wPosition = wPosition4.xyz / wPosition4.w;
  gl_Position = projection * view * wPosition4;
  wNormal = (model * vec4(normal, 0.0)).xyz;
}
```

```glsl
#version 330 core
uniform vec3 wCameraPosition;
uniform vec4 lightPosition;

in vec3 wPosition, wNormal;

out vec4 outColor;

void main() {
  vec3 col = vec3(1);
  float s = 20.0;

  vec3 L = normalize(lightPosition.xyz);
  vec3 N = normalize(wNormal);
  vec3 V = normalize(wCameraPosition - wPosition);
  vec3 R = reflect(-L, N);

  float v1 = max(dot(N, L), 0);
  float v2 = max(dot(R, V), 0);
  float v3 = pow(v2, s);
  float v4 = floor(v1*4.0)/4.0;
  float v5 = pow(1-max(dot(V, N), 0),3);

  vec3 c1 = v1 * col;
  vec3 c2 = v2 * col;
  vec3 c3 = v3 * col;
  vec3 c4 = c1*0.7 + c3;
  vec3 c5 = v4 * col;
  vec3 c6 = v5 * col;

  outColor = vec4(c***,1);
}
```

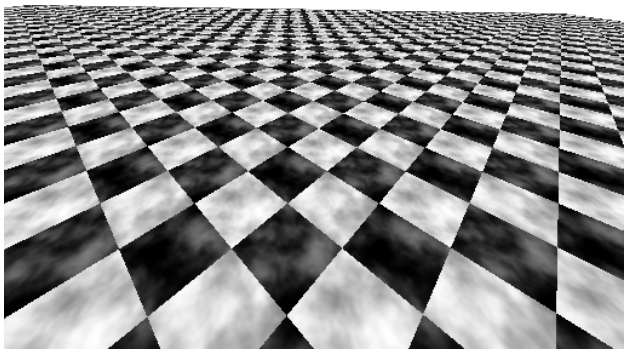
FIGURE 4 – Gauche : vertex shader. Droite : fragment shader.



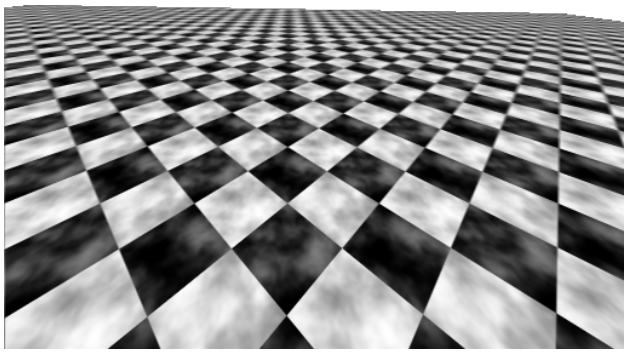
1



2



3



4

FIGURE 5 – En haut à gauche : la texture à créer. En haut à droite : plaquage sur le plan $y = 0$. Bas : Résultats avec différents filtres.