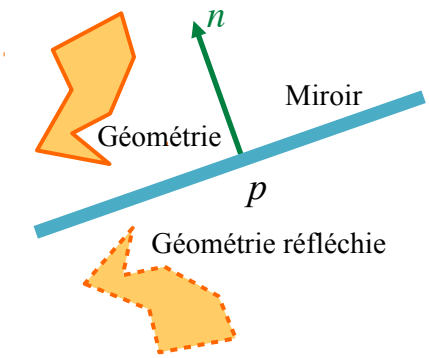
université de BORDEAUX	**ANNEE UNIVERSITAIRE 2015 / 2016** **SESSION DE PRINTEMPS 2016** **MENTION** : INFORMATIQUE **Code UE :** IN8M12EX **Intitulé de l'épreuve :** Programmation 3D Haute-Performance **Date :** 04/05/2016 **Heure :** 8h30 **Durée :** 1h30 Documents : non autorisés Épreuve de messieurs : Guennebaud et Bénard	**Collège Sciences et technologies** **Masters**

Vos réponses doivent être claires et concises (respect des maxima indiqués). L'examen est long mais noté sur 22, il n'est donc pas indispensable de répondre à toutes les questions.

A. Transformations de l'espace [4 pts]

Dans cet exercice nous supposerons que nous avons accès à une fonction *rot(θ,a)* qui retourne une matrice de rotation d'angle θ autour de l'axe a, ainsi qu'une fonction *trans(t)* qui retourne une matrice effectuant une translation d'un vecteur t et une fonction *scale*(s_x, s_y, s_z) qui retourne une matrice de mise à l'échelle. Ces trois fonctions retournent des matrices 4×4.



A.1. Pourquoi utiliser une matrice 4×4 pour transformer des données 3D ?

A.2. Donnez la matrice de rotation retournée par *rot* (θ, $[0,1,0]^T$).

A.3. On veut effectuer la réflexion planaire d'un objet par rapport à un miroir centré au point p et orienté selon la normale n, donnez la combinaison et les paramètres des fonctions *rot*, *trans* et *scale* permettant d'arriver à ce résultat.

B. Lancer de rayon [3 pts]

B.1. Décrivez le principe général du lancer de rayon récursif. *(10 lignes max + 1 schéma)*

B.2. Voici un exemple de code calculant l'intersection entre un rayon **r** et un maillage triangulaire représenté sous la forme d'une liste de triangles :

```
1 : Hit intersect_mesh(const std::vector<Triangle>& mesh, Ray r) {
2 :   for(int i=0 ; i<mesh.size() ; ++i) {
3 :     float t = intersect_triangle(mesh[i], r);
4 :     if(t>=0)
5 :       return Hit(t, mesh[i]); // retourne le paramètre d'intersection
6 :                               // et le triangle intersecté
7 :   }
8 :   return InvalidHit; // retourne un objet Hit invalide (pas d'intersection trouvée)
9 : }
```

La fonction `intersect_triangle` retourne une valeur négative s'il n'y a pas d'intersection, et le paramètre **t** de l'intersection le long du rayon le cas échéant. Expliquez pourquoi l'image produite par lancer de rayon avec cette fonction sera incorrecte. Proposez une version corrigée du code ci-dessus.

C. Rendu temps-réel / OpenGL [6 pts]

L'objectif de cet exercice est de calculer l'éclairement d'un objet par une source lumineuse de type spot, via des shaders. Le spot est défini dans le repère global par :

- une position L_p,
- une direction d'éclairement L_d,
- un angle d'ouverture θ.

On note la fonction calculant l'intensité de lumière re-émise pour une surface **diffuse** définie à la position *p*, orientée selon une normale *n*, le tout exprimé dans le même repère :

```
float computeLighting(vec4 Lp, vec4 Ld, float theta, vec4 p, vec3 n){ }
```

Note : la valeur d'intensité retournée doit être comprise dans l'intervalle [0,1]

C.1. Proposez une implémentation de cette fonction.

Cette fonction peut être utilisée pour calculer l'éclairement par sommet, comme décrit dans les vertex et fragment shaders ci-dessous :

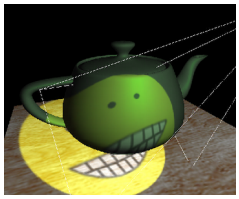
Vertex shader	Fragment shader

`in vec3 vtx_position, vtx_normal;`
 	
`uniform vec4 Lp, Ld;`
`uniform float theta;`
`uniform mat4 model_view_matrix;`
`uniform mat3 normal_matrix;`
`uniform mat4 projection_matrix;`
 	
`out float intensity;`
 	
`void main(void)`
`{`
`  vec4 global_p = model_view_matrix *`
`  vec4(position,1.);`
`  vec3 global_n = normal_matrix * vtx_normal;`
 	
`  intensity = computeLighting(Lp, Ld, theta,`
`  global_p, global_n);`
 	
`  gl_Position = projection_matrix * global_p;`
`}`
	`in float intensity;`
	
	`out vec4 out_color;`
	
	`void main(void)`
	`{`
	
	`  out_color = vec4(vec3(intensity),1.0);`
	`}`

C.2. Modifiez ce code pour réaliser le calcul d'éclairement par fragment.

C.3. Quelles sont les avantages et inconvénients des deux solutions ? *(3 lignes max)*

C.4. Proposez une méthode pour transformer ce spot en un projecteur comme illustré par la figure ci-contre. *(5 lignes max)*

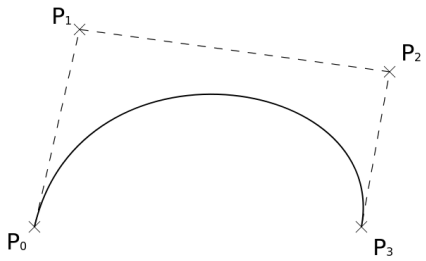


D. Géométrie et animation [5 pts]

D.1. Donnez l'équation implicite d'une sphère de rayon *r* et de centre *c*.

D.2. Une courbe cubique de Bézier $B_1(u), u\in[0,1], B_1:\mathbb{R}\rightarrow\mathbb{R}^3$ définie par les quatre points de contrôle p_0, p_1, p_2, p_3 peut être exprimée de la manière suivante :

$$B_1(u)=(1-t)^3 p_0+3t(1-t)^2 p_1+3t^2(1-t) p_2+t^3 p_3$$



D.2.1. En déduire une expression simple des vecteurs tangents t_0 et t_3 aux extrémités p_0 et p_3 .

D.2.2. Soit $B_2(u), u\in[0,1]$ une seconde courbe de Bézier définie par les points de contrôle p_4, p_5, p_6, p_7 . Nous souhaitons connecter les deux courbes en $B_1(1)$ et $B_2(0)$. Déduire de la question précédente les conditions nécessaires et suffisantes pour avoir une continuité C^0 ? C^1 ? et enfin G^1 ?

D.3. Explicitez la différence entre cinématique directe et inverse. Donnez un exemple de leur utilisation respective. *(4 lignes max)*

E. CUDA [4 pts]

Nous disposons d'un grand nombre n de points $\mathbf{p}_i$ auxquels sont associés des poids w_i. L'objectif est de calculer le centre de masse $\mathbf{c} = \sum_{i=0}^{n-1} w_i \mathbf{p}_i$ en utilisant CUDA. Nous supposerons que les sommets $\mathbf{p}_i$ et les poids w_i sont déjà stockés en mémoire GPU dans deux tableaux `p` et `w` :

```
float *p, *w ;
```

E.1. Écrivez un noyau CUDA réalisant cette opération. Vous donnerez également le code C++ appelant le noyau sans vous soucier des allocations ni des copies mémoires.

E.2. De votre point de vu, le code que vous avez proposé est-il optimal ? Si oui justifiez, sinon donnez des pistes d'amélioration possibles. *(5 lignes max)*