TP05 : textures

Initialisation

Terminez le TP précédent de sorte à ce que l'algorithme du ray-tracing soit bien en place.

L'objectif de ce TP est d'implémenter et tester différentes textures pour faire varier les matériaux des objets et l'éclairement de la scène.

Texture simple

Commencez par combiner le matériau assigné au sol avec une texture simple que vous choisirez. Dans l'exemple ci-dessus, une texture de type checkerboard a été utilisée pour faire varier les propriétés spéculaire de la surface. Vous pouvez utiliser n'importe quelle fonction mathématique pour créer votre propre texture. Rappelez vous aussi que toutes les propriétés des matériaux peuvent éventuellement être affectées (couleur diffuse, couleur spéculaire ou rugosité).

Gradient noise

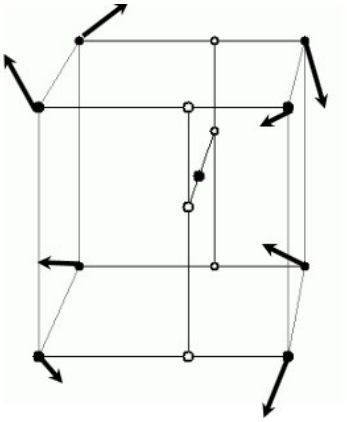
Ecrivez une fonction implémentant le gradient noise comme vu en cours. Pour rappel, l'algorithme est le suivant pour calculer le bruit en un point p(x,y,z) de l'espace 3D :

1. déterminer la position des 8 points du cube dans lequel se trouve p. La fonction floor(p) détermine la position du point en bas à gauche. Il suffit ensuite d'ajouter 1 dans les directions x, y, et z pour déterminer les 7 autres coins.
2. Déterminer des gradients (vec3) définis de manière pseudo aléatoire sur chacun de ces 8 points. La fonction suivante pourra être utilisée pour cela :

```
vec3 hash(vec3 p) {
        p = vec3(dot(p,vec3(127.1,311.7, 74.7)),
                 dot(p,vec3(269.5,183.3,246.1)),
                 dot(p,vec3(113.5,271.9,124.6)));

        return -1.0 + 2.0*fract(sin(p)*43758.5453123);
}
```

Elle assure que pour une position donnée en entrée, le gradient pseudo aléatoire renvoyé sera toujours le même (pratique donc lorsque qu'on va passer d'un cube à l'autre). On obtient donc un vecteur sur chacun des coins du cube :



3. Pour calculer la valeur du bruit en chaque coin du cube, il nous faut calculer le produit scalaire entre le gradient et la différence entre p et le coin considéré. On peut donc résumer ces 3 premières étapes avec la formulation ci-desssous. Pour calculer la valeur du bruit dans le coin en bas à gauche du cube, il suffit donc de faire :

   float v000 = dot(hash(i + vec3(0.0,0.0,0.0)), f - vec3(0.0,0.0,0.0))

   dans cet exemple, i = floor(p) et représente la position du coin de la grille, et f = fract(p) représente la partie fractionnaire de p dans la cellule de la grille (position relative de p dans la grille car cette valeur varie entre 0 et 1).

   La valeur de bruit est donc à calculer sur les 8 sommets de la cellule (il suffit pour cela de faire varier les vec3 inclus à l'intérieur de cette instruction).

4. Une fois les 8 valeurs de bruit calculées, il suffit d'effectuer les interpolations de ces valeurs pour obtenir la valeur de bruit au point p. Comme on peut le voir sur le shéma, il faut effectuer 4 interpolations sur l'axe des X, puis 2 interpolations sur l'axe des Y et enfin une dernière sur l'axe des Z. Le vecteur f = fract(p) peut servir pour effectuer cette interpolation car il est composé de valeurs variant entre 0 et 1. Néanmoins, on va tout d'abord le reparamétrer pour assurer une interpolation la plus lisse possible entre les données. On calculera d'abord le vecteur vec3 u = f*f*(3.0-2.0*f) que l'on pourra ensuite directement utiliser comme paramètre pour l'interpolation linéaire. Par exemple, le point interpolé le plus bas dans le shéma ci-dessus pourra être obtenu comme ceci : vec3 a1 = mix(v000,v100,u.x)

Une fois votre bruit obtenu, essayez le sur un ou plusieurs objets pour vous assurer qu'il fonctionne bien.

Fractal gradient noise

Ecrivez une fonction permettant d'accumuler du bruit à mutliples octaves (fréquence X 2 à chaque nouveau bruit) pour obtenir un bruit fractal. Cette fonction doit être facilement controlable : il doit être possible de modifier facilement l'amplitude, la fréquence, la persistence ainsi que le nombre d'octaves utilisées pour générer le bruit final. Pour rappel, l'amplitude et la fréquence du bruit peuvent être controlé comme ceci : amplitude * gradientNoise(p * frequency). Le cumul de l'octave suivant se fera avec une fréquence multipliée par 2 et une amplitude multipliée par la persistence (en général <= 1 de sorte à ce que l'amplitude du bruit diminue dans les hautes fréquences).

Une fois fait, l'objectif sera de manipuler ces bruits pour concevoir des effets différents. Le bruit fractal peut être utilisé tel quel pour simuler des effets de vieillissement ou de rouille. Le marbre peut être obtenu avec la fonction sin(p.x + fractalNoise(...)). Les turbulence (ou l'ajout de discontinuitées) peuvent être obtenues en prenant la valeur absolue des bruits calculés. Bien entenu, ces bruit peuvent être associés à des colormap et / ou peuvent modifier n'importe quelle propriété des matériaux pour obtenir les effets souhaités (il faut tester).

Image based lighting

Pour finir, on souhaite utiliser un environnement lumieux pour rendre la scène plus réaliste (en plus de notre source de lumière ponctuelle). Il s'agit donc simplement de retourner la couleur de l'environnement dans les régions où les rayons n'auront rien touché dans la boucle du ray-tracing.

Pour cela, ajouter un noeud imgLoader dans votre scène et chargez l'environement. Ajoutez aussi une entrée à votre noeud pour y connecter l'environnement.

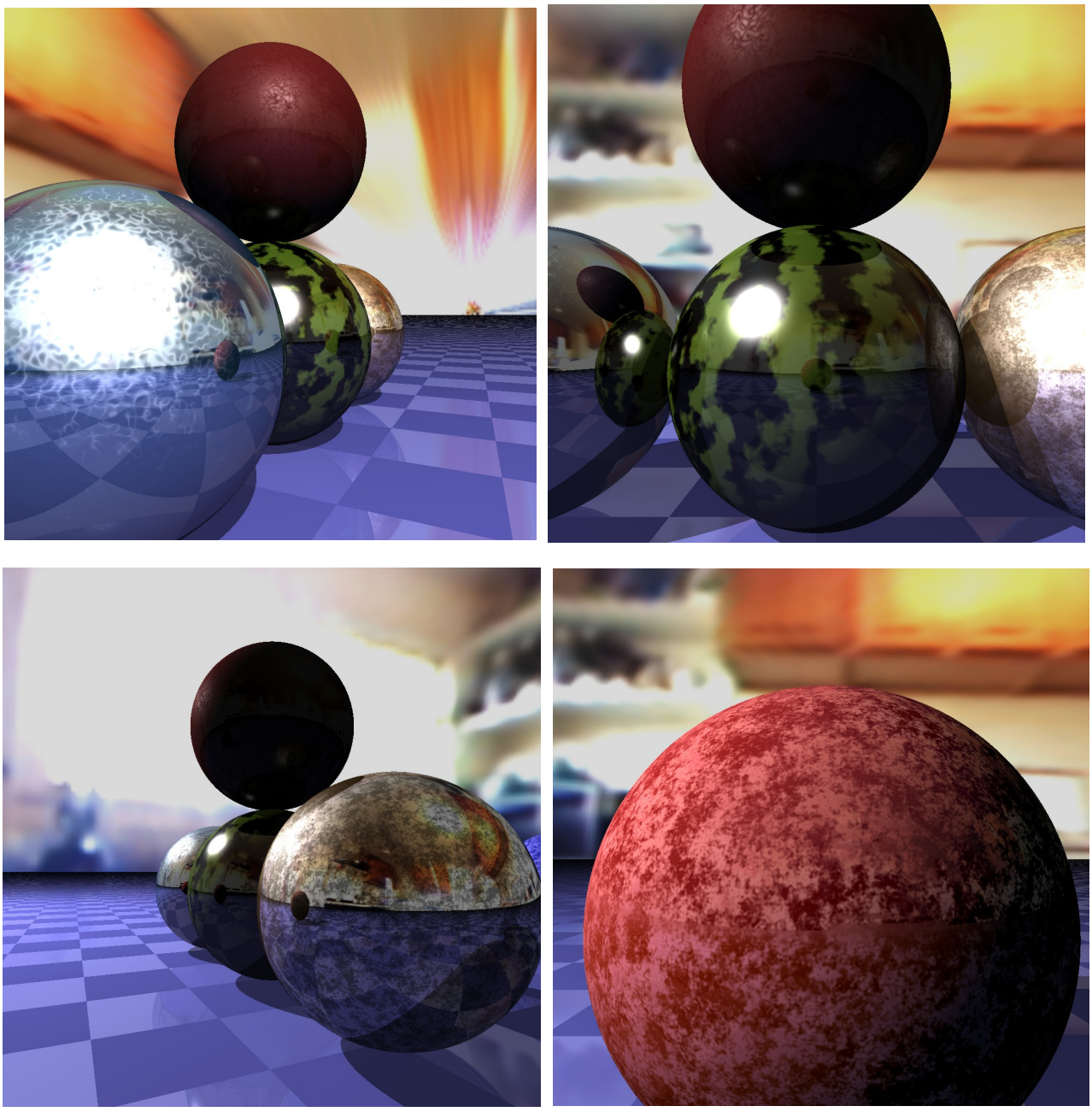
L'environement est stocké dans une image sous forme de latitude-longitude. Il vous faut donc une fonction pour convertir un vecteur 3D (qui définit une direction sur la sphère, ie. La direction du rayon) en une coordonnée 2D. Les fonctions suivantes pourront être utilisées pour cela :

```
#define PI 3.14159265359
#define PIO2 1.5707963267948
#define PIT2 6.2831853071795

// latitude-longitude mapping using the atan2 function
float atan2(in float y,in float x) {
	if(x>0.0) return atan(y/x);
	if(y>=0.0 && x<0.0) return PI+atan(y/x);
	if(y<0.0 && x<0.0) return -PI+atan(y/x);
	if(y>0.0 && x==0.0) return PIO2;
	if(y<0.0 && x==0.0) return -PIO2;
	return 0.0;
}

vec2 envMapCoord(in vec3 v) {
	return vec2((atan2(v.x,v.z)+PI)/PIT2,acos(-v.y)/PI);
}
```

Affinez votre scène pour obtenir des effets / matériaux les plus réalistes / variés possibles. Quelques exemples ci-dessous :



En haut à gauche : des turbulences controlent la rugosité (la taille du lobe) du matériau.

En haut à droite : une texture de marbre controlent la couleur diffuse du matériau.

En bas à gauche : le bruit fractal controle la couleur spéculaire pour obtenir un effet rouillé.

En bas à droite : un objet très matte modifié par un bruit fractal (une planète rouge).

A rendre

Enregistrer et nommer votre pipeline prenom-nom-tp05.gra.

Envoyer votre pipeline à benoit.arbelot@gmail.com (titre du mail : [GICAO SIA] TP05)

Dans le mail, expliquer brièvement les étapes que vous avez effectuées et les problèmes rencontrés.