OS04 : Application au lancé de rayons

1. Concernant l'implication au sein du groupe, nous nous sommes répartis les tâches au sein du groupe au début du projet, avec chaque "trou" attribué à l'un d'entre nous, en fonction de notre compréhension du sujet et de la méthode à implémenter. Bien évidemment, le travail s'effectuant en binôme, il y a eu des moments où nous nous sommes entraidés : si l'un d'entre nous éprouve des difficultés à compléter sa fonction, il demande de l'aide à l'autre. Une fois que nous avions une partie "fonctionnelle", nous rendant une image assez proche de ce que nous désirions avoir, nous nous somme penchés sur ce que nous voulions ajouter, et nous nous sommes chacun penchés sur ce que nous voulions faire.

Parmi les effets que nous avons réalisés, on peut voir notamment la réflexion : que ce soit dans la grosse sphère ou sur le sol, la réflection est apparente. La réfraction est également visible, encore une fois, en regardant au travers de la grosse sphère principalement. On peut également voir la présence d'ombres douces, sur chacune des ombres. On peut également remarquer la présence d'un éclairage global qui accompagne notre éclairage diffus, principalement au fait que les ombres ne sont pas noires, mais juste de couleur sombre.

Nous avons également créés une série de tests nous premettant de valider nos résultats quant aux méthodes concernant les vecteurs. Ceux-ci se trouvent dans le répertoire test et sont composés de 4 fichiers : 2 headers et 2 fichiers .cpp.

1. Une fonctionnalité que nous souhaitions ajouter est de gérer la génération d’image de manière ‘propre ‘ au lieu de tout mettre dans le main. Nous avons donc créé une classe ObjectScenario qui permet de regrouper l’ajout des objets, le calcul de l’image et son enregistrement. Cette classe pourra par exemple être utilisée pour générer plusieurs images à la suite et de rendre le code plus lisible. Sa principale utilité cependant est de grandement faciliter l’utilisation du code. Quelqu’un souhaitant s’interfacer sur notre code n’aura potentiellement que cette classe à utiliser pour générer des images.

Dans notre ‘simple’ cas pour générer plusieurs images, nous créons une classe ‘ScenarioPersoX’ héritant de ObjectScenario. Nous redéfinissons son constructeur afin d’ajouter tous les objets que nous souhaitons à l’instanciation de cette classe dérivée. Nous appelons ensuite la méthode toImage qui permet de calculer l’image liée au scenario et de l’enregistrer.

Le calcul d’un scenario personnalisé est de la forme :

ScenarioPerso(int largeur, int hauteur).toImage(char[] nom, bool afficherTexte, int profondeur);

Une autre fonctionnalité qui nous semble importante en comparant les images que nous avons générées et celles que nous pouvons obtenir est l’implémentation d’une lumière qui ‘adoucit’ les ombres. Il suffit pour réaliser cela que la lumière vienne de plusieurs ‘points’ situés relativement proche les uns des autres.

Nous avons donc choisi de créer une classe «Lumiere\_Smooth» héritant de «Lumiere\_Ponctuelle». Nous avons ajouté 2 attribut dans cette nouvelle classe : un flottant représentant la « largeur » de la lumière et un entier proportionnel à la qualité du rendu.

En pratique cette Lumiere\_Smooth se comporte comme un «cube» de lumière ponctuelle. L’ entier nbLums représente le nombre de lumières par face du cube. Au total 3\*\*nbLums lumière ponctuelle sont donc générées lors du calcul. En pratique, nbLums à 3 ou 4 produit un rendu suffisamment réaliste. Augmenter nbLums est très couteux en temps de calcul, il faut donc faire attention à ce paramètre.

Il aurait été possible de faire cette classe en disposant les lumières en sphère autour du centre pour un rendu plus réaliste, ou en carré afin de diminuer le temps de calcul pour un résultat relativement similaire.