TP 6 : Lancer de rayon primaire

L'objectif du TP est de mettre en oeuvre les différents éléments abordés lors des 5 premiers TP pour créer des visualisations de formes 3D éclairées par un lancer de rayon primaire. Les visualisations seront générées et lancées à partir d'exécutables compilés.

Question 1. Récupérez et explorez les fichiers de code dans la Dropbox.

Création d'une bibliothèque de fonctionnalités communes

Question 2. Créez un fichier *tp6_commun.ml* et placez à l'intérieur tout le code de *tp6_code.ml*, sauf ce qui se trouve dans la section Formes (la déclaration des 3 formes, l'équation du tore et les fonctions pour construire le cube de Menger).

Nous allons transformer *tp6_commun.ml* en bibliothèque commune pour tous les exécutables que nous construirons ensuite. Chaque exécutable sera dédié à une seule forme donnée et contiendra uniquement le code qui la concerne. Avant cela, dans le fichier *tp6_commun.ml* :

Question 3. Modifiez la fonction *genere_images* pour la rendre récursive terminale, puis encapsulez-la pour ne pas avoir à renseigner les valeurs par défaut des accumulateurs à chaque fois.

Question 4. Créez un type pour le paramètre *scene* (que vous pouvez appeler de la même manière). Ce type possède deux attributs *vue* et *forme*. Les types des attributs sont déclarés dans le fichier *ray_tracer.ml*, à vous de correctement les identifier. Le premier attribut doit pouvoir contenir *vue_generique*, déclarée dans la section Initialisation de *tp6_commun.ml*, et le second doit pouvoir contenir n'importe quelle forme, que ce soit *cube_violet*, *tore_jaune* ou *cube_menger*.

Question 5. Créez une fonction *diaporama* dont la signature est $val\ diaporama : string\ list \rightarrow unit = < fun >$. La fonction utilise *List.iter* pour parcourir une liste de noms de fichiers images, puis pour chaque élément :

1) charge le fichier avec la fonction *read_img* du module Image ;
2) crée l'image avec la fonction *make_image* du module Graphics ;
3) affiche l'image dans l'interface graphique avec la fonction *draw_image* du module Graphics.

Question 6. Créez une fonction récursive terminale *lancer_animation* qui prend en entrée une liste de noms de fichiers images et qui appelle en boucle la fonction *diaporama* jusqu'à ce que l'utilisateur entre le caractère *q* dans la console.

Question 7. Créez une fonction *generation_puis_animation* qui appelle successivement la fonction *genere_images* puis la fonction *lancer_animation*.

Question 8. Certaines fonctions du fichier *tp6_commun.ml* ont besoin d'autres fonctions comme *tracer_rayons* ou *read_img*, qui sont déclarées dans les fichiers *ray_tracer.ml* et *image.ml* et qui sont importées via le module *Ray_top*. Lorsque nous importerons les fonctions du module *Tp6_commun* dans nos futurs exécutables, celles du module *Ray_top* ne seront plus accessibles si importées avec la directive *open*. Trouvez la bonne directive pour pouvoir ensuite garder l'accès à ces fonctions.

Indice : Comment des fonctions comme tracer_rayons ou read_img, qui sont déclarées dans deux fichiers différents, peuvent bien être accessibles via un même module, lui-même déclaré dans un autre fichier (ray_top.ml) ?

Compilation de la bibliothèque

Il existe deux compilateurs pour le langage OCaml, tous les deux accessibles via la commande **ocamlfind**. Le premier, **ocamlc**, compile du code portable en bytecode. Le second, **ocamlopt**, compile en code natif. Le second compilateur permet d'obtenir des exécutables bien plus optimisés que le premier.

Le fichier *makefile* fourni avec le TP permet pour le moment de compiler le fichier *ray_top.ml* en tant que bibliothèque grâce au *-a*, à la fois avec **ocamlc** (qui crée un fichier en .cma) et avec **ocamlopt** (qui crée un fichier .cmxa). Le paramètre qui suit le *-o* correspond au nom à donner à la bibliothèque, et tous ceux qui suivent sont les noms des dépendances à inclure.

Question 9. Modifiez le *makefile* pour que le fichier à compiler en tant que bibliothèque ne soit plus *ray_top.ml* mais *tp6_commun.ml*. Attention aux dépendances, un fichier A qui utilise une fonctionnalité déclarée dans un fichier B ne doit pas être placé avant lui dans la liste. Le fichier compilé *tp6_commun.cmo* doit également faire partie de cette liste (tout à la fin).

Question 10. Vérifiez que votre code compile bien pour le moment.

Création des exécutables

Question 11. Créez trois fichiers *tp6_cube.ml*, *tp6_tore.ml* et *tp6_menger.ml*. Insérez dans chaque fichier les parties du code correspondantes de la section Formes de *tp6_code.ml*.

Question 12. Importez la bibliothèque précédemment créée avec la directive *open*. Le nom du module correspond au nom du fichier en .cma (ou .cmxa), sans l'extension et avec une majuscule.

Question 13. Ajoutez un appel à la fonction *generation_puis_animation* dans chaque fichier.

Compilation des exécutables

Les parties commentées dans le *makefile* fourni avec ce TP montrent comment compiler, avec **ocamlc** et **ocamlopt**, un exécutable dont le code se trouve dans un fichier *tp6_exemple.ml* et qui a besoin des bibliothèques *graphics* et *tp6_commun* pour fonctionner.

Question 14. Décommentez ce code, modifiez-le et dupliquez-le pour pouvoir créer trois exécutables *ex_cube*, *ex_tore* et *ex_menger* à partir des trois fichiers créés à la question 11.

Question 15. Lancez le *makefile*, vérifiez qu'aucun problème de compilation ne se présente.

Question 16. Lancez vos exécutables générés avec **ocamlc**, puis ceux générés avec **ocamlopt** pour constater la différence d'optimisation.

Question 17. (Pour aller plus loin). Créez plus d'exécutables avec d'autres formes plus ou moins complexes. Tentez des animations différentes, voire manuelles.