Projet PFav 2016/2017
Rapport

Abderrazak ZIDANE

Hiérarchie Des Modules

Voici une description des modules que j'ai crée :

- **vect** : Vecteur
- **texture** : Texture
- **rotation** : Pour crée une rotation
- **ray** : Le programme principale
- **ppm** : Pour crée une image ppm
- **hit** : Le point d'intersection entre un rayon et un objet
- **command** : Pour parser les options données en ligne de commande au programme
- **color** : Une couleur
- **camera** : Objet camera
- **box** : Objet boite
- **light** : objet lumière
- **plane** : objet plan
- **sphere** : Une Sphère
- **scene** : contient toute la scène

Description du module Scene

Le type scene, est une structure de donnée, qui contient toute la scène. Elle est définit de cette façons :

```
type t = {
  ambiant  : float;
  camera   : Camera.t;
  lights   : Light.t list;
  spheres  : Sphere.t list;
  boxes    : Box.t list;
  planes   : Plane.t list;
}
```

L'idée est de crée une scène en parcourant l'arbre AST produit par le parseur fournit. En stocke tout sa dans un objet de type Scene qui sera utilisé plus-tard. Les fonctions importante de ce module sont : `val intersect  : Vect.t -> Vect.t -> t -> Hit.t option` qui permet de voir si il y a eu une intersection entre un rayon est un des objets de la scene, et de retoutner un objet de type Hit. `val ray_trace  : Vect.t -> Vect.t -> int -> t -> Color.t` qui permet de calculer une couleur en envoyons un rayon vers une direction, cette couleurs est calculé suivant l'équation fournit dans la doc du projet.

Description du module Hit

Le type Hit.t représente une intersection entre un rayon est un objet. Il contient toute les information pour le calcule de la couleur, voici une définition de ce type :

```
type t = {
  cam   : Vect.t;
  point   : Vect.t;
  normal   : Vect.t;
  refl   : Vect.t;
  color   : Color.t;
```

```
  kd  : float ;
  ks  : float ;
  phong : int ;
}
```

Description du module Rotation

Comme vu dans la doc du projet, une rotation est définit comme une matrice de réel, le calcule de cette matrice est tellement exigent qu'on a préféré le mettre dans un module a part pour le séparer du programme

Et pour La translation et la dilatation ?

La translation et la dilatation étais pas trop compliqué a réaliser, dans chaque définition d'objet (Box, Plane, Sphere) on a 3 méthodes qui permettent de réaliser les opérations de transformation :

```
val apply_rotation  : t -> Rotation.t -> t
val apply_translation  : t -> Vect.t -> t
val apply_dilatation  : t -> float ->  t
```

Exécuter notre programme

Pour exécuter notre programme nous devons d'abord parler du module Command

Description du module Rotation

En utilisant le module Ocaml Arg, on a crée un parseur de commande en ligne qui reconnait les options suivante :

```
-source               : Scenario source file
-v                    : Enables verbose mode
-depth                : Sets maximum number of rebonds
-hsize                : Sets width picture
-vsize                : Sets height picture
-maxProcessForVideo   : The maximum number of process used for creating video
-maxProcessForPicture : The maximum number of process used for creating picture
-size                 : Sets width and height picture
-anim                 : Create a movie with n pictures
-help                 :  Display this list of options
--help                : Display this list of options
```

Exemple pour exécuter notre programme

Après avoir exécuté le Makefile dans *"pfav-2016/src/Makefile"*, on exécute pfav-2016/src/ray.native comme ceci :

- ./ray.native -source ../scenarios/planets.ray -size 800 600 -anim 7 -maxProcessForVideo 100
- ./ray.native -source ../scenarios/planets.ray -size 3000 400 -anim 7 -maxProcessForVideo 8
- ...

Fonctionnalité implémenté et Piste d'extension

J'ai réussi a crée le ray traceur, mais j'ai pas encore implémenter le programme qui transforme l'arbre AST en objet scene, est dont pour tester le ray traceur, j'ai crée une scène manuellement dans la fonction **Scene.create**, j'ai crée la scène du scénario du system solaire.

test de performance

Voici un test de performance que j'ai réaliser pour avoir une idée de l'importance d'utiliser le parallélisme avec l'option **-maxProcessForVideo**. Le test est réalisé sur une VM debian sous VMware workstation, le processeur host est un i7-6700HQ a 2.60GHz :

- Réaliser une animation de **7 images** en résolution **800*600** :
  - Avec **1** processus : **15** seconds
  - Avec **7** processus : **5.7** seconds
  - Avec **8** processus : **4.8** seconds
  - Avec **20** processus : **4.5** seconds
  - Avec **100** processus : **4.6** seconds
- Réaliser une animation de **100 images** en résolution **800*600** :
  - Avec **1** processus : **196** seconds
  - Avec **7** processus : **70** seconds
  - Avec **10** processus : **60** seconds
  - Avec **20** processus : **63** seconds
  - Avec **100** processus : **63** seconds
- Réaliser une animation de **100 images** en résolution **3840*2160** :
  - Avec **1** processus : **3666** seconds
  - Avec **7** processus : **1678** seconds
  - Avec **8** processus : **1293** seconds
  - Avec **20** processus : **1283** seconds
  - Avec **100** processus : **1293** seconds

A faire

Il faudrait maintenant implémenter le programme qui traduit l'arbre AST en objet Scene, est relancer le test de performance sur d'autre scénario

Âpres sa, il faut essayer d'optimiser notre programme, en apportant des modification sur les fonctions utiliser de manière trop souvent, et de relancer le test de performance a chaque fois pour voir si les modification apporté ont réellement apporter un gain de performance