Projet CSG_Particule

Vecteur

Écrire une classe *Array* template du nombre d'éléments et du type d'éléments
Pas d'allocation dynamique, stockage “en dur”
Elle devra au moins contenir

- Les constructeurs
- opérateur d'affectation
- operateur de test d'égalité
- les accesseurs au données
- échange de 2 Array
- etc..

Dériver *Array* en *Vector* (toujours template) en ajoutant les opérateurs mathématiques :

- opérateurs entre vecteurs: + - += -=
- opérateurs avec un scalaire * / *= /=
- Comment faire pour pouvoir écrire *V = 0.5f * V* ?

Dériver *Vector* en *Vec3f* et *Vec2f* en introduisant les opérations spécifiques :

- produit scalaire (*)
- produit vectoriel (^)

Remarques ;

on surchargera si nécessaire les constructeurs.
attention à la conversion automatique !

Matrices de transformation

Ecrire une classe *Matrix33f* représentant les matrices de transformations homogène 2D

- constructeur
- accesseurs
- inverse
- set & apply translation
- set & apply rotation
- set & apply homothétie
- multiplication par un vecteur (coordonnée homogène *Vec3f*)
- appliquer la transformation à un vecteur (*Vec2f*)

Image

Écrire une classe *Image* template de la dimension (1D,2D,3D, …) et du type des pixels
On écrira au moins

- les constructeurs (allocation dynamique de la mémoire de l'image)
- destructeur
- affectation
- échange entre deux images
- accès aux pixels
- accès au pointeurs sur la mémoire de l'image

Remarque : on utilisera Array pour passer les tailles et les coordonnées (dimension inconnue à la compilation!)

On dérivera en *image2D* template uniquement du type pour définir des contructeurs et accesseurs plus pratique (x,y)

On dérivera en *Image2Grey* (image 2D de unsigned char) en ajoutant
les fonctions de chargement et sauvegarde du format PGM (ascii)
sous-échantillonnage (diviser la taille de l'image par 2, renvoie une nouvelle image)
llissage (moyenne des pixels avec les $(2n+1)^2$ pixels voisins, renvoie une nouvelle image)
seuillage par une valeur (0 en dessous 255 au dessus)

Qu'impliquerait d'écrire le seuillage dans Image2D<T> ?
Comment contourner le problème ?

Écrire la fonction (ou classe) *GradientSobel* qui à partir d'une image à niveau de gris calcul une image de Vec2f. On utilisera les filtres 5x5 suivants :

Pour la direction horizontale :

```
{1, 2, 0, -2, -1,
 4, 8, 0, -8, -4,
 6,12, 0,-12, -6,
 4, 8, 0, -8, -4,
 1, 2, 0, -2, -1};
```

Pour la direction verticale :

```
{  1,  4,  6,  4, 1,
   2,  8, 12,  8,  2,
   0,  0,  0,  0,  0,
  -2, -8,-12, -8, -2,
  -1, -4, -6, -4, -1};
```

On dérivera *image2D* en *Image2RGB* (image 2D de Vec3uc) en ajoutant les fonctions de chargement et sauvegarde du format PPM (ascii)

Boîtes englobantes

Ecrire une classe BoundingBox décrivant des boites englobantes alignées sur les axes :

- constructeur
- accesseurs
- union
- intersection
- différence
- test si vide
- centre
- teste si un point est dans la boite
- ajout d'un point P dans la boite

CSG

Principe :
La scène est décrite sous la forme d'un ou plusieurs arbres binaire dont les nœuds sont des opérations (Union, Intersection, Différence) et les feuilles des primitives (ici disques et polygones) avec une transformation géométrique (*Matrix33f*). Chaque nœud contient une *BoundingBox*

Décrire le graphe avec les classes *CsgTree*, *CsgNode*, *CsgOperation*, *CsgPrimitive*, *CsgDisk*, *CsgPolygon*.

La classe *CsgTree* contiendra :

- un ensemble de nœuds racines
- un ensemble de nœuds feuilles
- une map permettant de retrouver un nœud (son adresse) à partir de son id (un entier)
- ...

Fonctions à programmer

- Chargement / sauvegarde du graphe (on utilisera les flux << et >>). Le format du fichier sera fourni.
- Rendu du graphe dans une image à niveau de gris
- Cloner un nœud
- Enlever un nœud et remettre ces 2 fils dans les racines
- Swap fils droit/ fils gauche

L'interface Qt sera fournie. On pourra déduire la signature de certaines méthodes à en étudiant le code de l'interface.