Animation

Projet commun LC4 / LS4

2009-2010
Version du 30 mars 2010

Introduction

Le but du projet est d'implémenter un programme permettant de créer et de visualiser un dessin animé rudimentaire.

Le programme sera composé d'une interface graphique pour rentrer les images décrivant l'animation, écrite en Python/Tkinter. On engendrera des images complémentaires permettant d'avoir une animation fluide par interpolation. Cette partie du programme et l'enregistrement des images se fera en C.

Pour vous guider, nous vous proposons trois étapes d'avancement :

1. implémenter une interface permettant de créer une image et de l'enregistrer,
2. étendre l'interface pour créer, en modifiant l'image initiale, une séquence d'images décrivant l'animation ; enregistrer cette séquence et la visualiser,
3. calculer les images complémentaires pour aboutir à un dessin animé que l'on visualisera.

1 Création et enregistrement d'une image

1.1 L'interface graphique (Python)

L'interface graphique doit être implémentée en Python en utilisant la bibliothèque `Tkinter`[1]. Elle doit permettre de dessiner à la souris des cercles, des triangles équilatéraux et des carrés remplis en noir, le tout sur fond blanc. Le programme devra garder trace des figures rentrées par l'utilisateur.

1.2 Enregistrement d'une image en format PBM (C)

Le but de cette sous-section est d'implémenter en C une fonction permettant d'enregistrer une image en format PBM. Ce format est décrit en annexe B.

Pour cela, implémentez la fonction suivante :

```
void enregistrer_image(
                        char *nom_fichier,
                        int largeur, int hauteur,
                        int nombre_cercles, int *cercles,
                        int nombre_triangles, int *triangles,
```

[1] Une description détaillée de cette bibliothèque se trouve sur le site `http://infohost.nmt.edu/tcc/help/pubs/tkinter/`

```
int nombre_carres, int *carres
 );
```

qui crée une image PBM de nom *nom_fichier* et d'extension `.pbm` correspondant aux spécifications (dimensions, figures géométriques) données en argument.

Vous pourrez implémenter des fonctions auxiliaires permettant de tester si un point est dans un cercle, un carré ou un triangle équilatéral.

Il est recommandé de ne pas écrire au fur et à mesure dans un fichier, ce qui ralentit considérablement l'exécution du programme, mais de se servir d'un tableau bi-dimensionnel d'entiers représentant les pixels de l'image comme tampon.

1.3 Liaison entre l'image dessinée et l'image enregistrée

Interfacez la fonction `enregistrer_image` pour l'appeler depuis votre interface. L'annexe D pourra vous aider, notamment pour la manipulation des tableaux d'entiers.

Complétez votre interface afin de permettre :
– l'enregistrement de l'image en format PBM,
– la visualisation de cette image.

Pour la visualisation vous pourrez lancer la commande `display` du logiciel libre `imagemagick`.[2]

2 Création et enregistrement d'une séquence d'images

À la fin de la première partie, vous avez un programme permettant de dessiner une image à la souris et de l'enregistrer en format PBM sur le disque dur.

L'objectif de cette partie est d'avoir un programme où l'utilisateur peut :
– dessiner l'image initiale et des images intermédiaires régulièrement espacées décrivant l'animation souhaitée,
– enregistrer ces images sur le disque,
– visualiser ces images sous forme de `gif` animé.

2.1 Interface graphique (Python)

Adaptez votre interface graphique pour permettre à l'utilisateur de déplacer les figures. Les méthodes `.find_closest` et `.coords` de la classe `Canvas` et `.bind` et `.unbind` de la classe `Event` pourront vous être utiles.

L'interface devra permettre à l'utilisateur de marquer la fin de la construction d'une image intermédiaire. Il faudra garder trace du contenu de toutes les images rentrées par l'utilisateur.

2.2 Enregistrement des images (C)

Dans cette partie, il n'y a pas une seule image mais une séquence d'images à enregistrer. Implémentez la fonction suivante :

[2] `http://www.imagemagick.org`

```
void enregistrer_sequence(
                      char *nom_fichier,
                      int nb_images,
                      int largeur, int hauteur,
                      int nb_cercles, int *seq_cercles,
                      int nb_triangles, int *seq_triangles,
                      int nb_carres, int *seq_carres
                       );
```

qui enregistre `nb_images` images `.pbm` dont le nom est `nom_fichier` suivi d'un nombre indiquant son ordre.

Le tableau `seq_cercles` peut être considéré comme une juxtaposition de tableaux contenant respectivement les données concernant les cercles de chaque image. Ainsi les données concernant les cercles de la première image seront stockées dans les $3 \times (\texttt{nb\_cercles})$ premières cases, les données concernant les cercles de la deuxième image dans les $3 \times (\texttt{nb\_cercles})$ suivantes, etc.

Il en va bien entendu de même pour les autres figures géométriques.

2.3 Création et visualisation d'un gif animé

Interfacez la fonction `enregistrer_film` avec SWIG de sorte qu'elle soit appelable depuis votre programme Python. L'enregistrement de la séquence d'images doit se faire lorsque l'utilisateur indique qu'il a terminé de rentrer les images.

Ajoutez un bouton à votre interface graphique permettant de visualiser la séquence d'images sous la forme d'un `gif` animé. Pour cela, utilisez les commandes `animate` (visualisation sans création) ou `convert` (création du gif animé) puis `display` (visualisation) fournies par `imagemagick`.

3 Création du dessin animé

À la fin de la deuxième partie, le programme permet à l'utilisateur de dessiner des images décrivant une animation, de les enregistrer, et de les visualiser.

Pour obtenir une véritable animation fluide, il reste à générer des images complémentaires.

Nous vous suggérons pour cela les étapes suivantes :
– calculer par interpolation les coordonnées intermédiaires des points spécifiques des figures (3.1),
– puis générer les images abstraites correspondantes (3.2),
– enregistrer ces images (3.3) et les visualiser sous forme de gif animé (3.4).

3.1 Interpolation (C)

Pour cela, il faut dans un premier temps calculer par interpolation (cf. annexe C) les coordonnées intermédiaires des points spécifiques des figures de l'image. Ces calculs se feront en C. Notez que pour des raisons de précision il vaut mieux faire les calculs avec des `double`.

Une interpolation permet de créer une courbe passant par les points des images intermédiaires créées, cette courbe étant représentée par une fonction. Cela suppose donc que le mouvement de chaque figure se fait sans changement de direction horizontale.

Ecrivez une fonction qui permette de tester qu'une suite finie de points est horizontalement monotone (c'est-à-dire que la suite des abscisses est monotone).

Nous allons calculer l'interpolation de trois manières différentes.

Interpolation linéaire

Il s'agit de lier deux points consécutifs par un segment de droite, comme expliqué dans l'annexe C.1.

Ecrivez la fonction `double interpolation_lineaire(double *donnees, int n, double x)` qui calcule l'ordonnée du point d'abscisse `x` dans la trajectoire passant par les `n` points de coordonnées `donnees`.

Interpolation polynômiale (implémentation naïve)

Son principe est expliqué dans l'annexe C.2.

Créez un type structuré `polynome` pour représenter un polynôme.

Ecrivez la fonction `polynome *polynome_lagrange(double *points, int n, int j)` qui calcule le `j`-ème polynôme de Lagrange à partir des `n` points du tableau `points`, et renvoie un pointeur vers ce polynôme.

Ecrivez la fonction `double interpolation_lagrange(double *donnees, int n, double x)` qui calcule l'ordonnée du point d'abscisse `x` dans la trajectoire polynomiale passant par les `n` points de coordonnées `donnees`.

Interpolation polynômiale (Algorithme de Neuville)

Son principe est expliqué dans l'annexe C.3.

Ecrivez les fonctions suivantes :

- `double** initialisation_tableau(double *d, int n)` qui créé et initialise un tableau dont les coefficients diagonaux sont les valeurs des ordonnées des données,
- `void calcule_tableau(double *d, int n, double **m, double x)` qui calcule les coefficients du tableau `m` selon la formule de Neville au point `x` et aux données `d`,
- `double interpolation_neville(double *donnees, int n, double x)` qui calcule l'ordonnée de la courbe passant par les `n` points en donnée à l'abscisse `x` en utilisant la formule de Neville.

3.2 Génération des images abstraites complémentaires (C)

Dans la section précédente 3.1, nous avons interpolé les coordonnées d'un point spécifique. Dans cette section, nous cherchons à calculer les images abstraites.

Vous devez coder, pour *chacune* des manières d'interpoler, une fonction C qui prend en argument une séquence d'images et un nombre d'images intermédiaires entre deux images consécutives et produit ces images intermédiaires.

3.3 Enregistrement des images concrètes (C)

Pour chacune des trois manières d'interpoler, coder une fonction globale qui, à partir des images rentrées par l'utilisateur, génère les images abstraites complémentaires par interpolation, puis enregistre les images concrètes (format PBM) de l'animation.

3.4 Création et visualisation d'un gif animé

Faites en sorte que l'utilisateur puisse choisir un type d'interpolation, engendrer des images complémentaires interpolées et lancer la visualisation d'un gif animé au niveau de l'interface graphique.

4 Consignes

Les projets sont à faire par groupes de deux étudiants.

Chacun devra être capable de répondre aux questions concernant la partie réalisée par l'autre. Par ailleurs, la répartition ne devra pas se faire selon le langage (C / Python), puisque ces deux parties servent à vous évaluer pour deux cours différents.

Le projet devra pouvoir être compilé (grâce à un Makefile) et exécuté sur les machines du script.

Le code devra être documenté correctement. Il sera joint, ainsi que la documentation automatiquement extraite (grâce à **pydoc -w** pour la partie python) à une rapide présentation écrite du projet.

Les étudiants non inscrits aux deux cours doivent le plus vite possible se faire connaître auprès de leur enseignant.

Appendice

A Images abstraites : description de la structure de données en C

L'image peut contenir des cercles, des triangles équilatéraux et des carrés.

A.1 Cercles

Un cercle sera défini par trois entiers représentant respectivement coordonnées du centre et rayon.

L'ensemble des cercles d'une image sera représenté par :
– un entier indiquant leur quantité,
– un tableau d'entiers regroupant les données de tous les cercles.

A.2 Triangles équilatéraux

Un triangle équilatéral sera défini par quatre entiers représentant respectivement les coordonnées d'un de ses sommets et celles de son centre.

Pour rappel, dans un triangle équilatéral, la longueur c du côté est égale à $\sqrt{3}l$, où l est la distance entre un sommet et le centre. Si on connaît un sommet A et le centre Ω du triangle, les deux autres sommets peuvent être obtenus comme intersection du cercle de centre A et de rayon c et du cercle de centre Ω et de rayon l.

De même que pour les cercles, l'ensemble des triangles sera représenté par un entier indiquant leur nombre et un tableau d'entiers regroupant les données de tous les triangles.

A.3 Carrés

Un carré sera défini par quatre entiers représentant les coordonnées d'un de ses sommets et celles de son centre.

L'ensemble des carrés sera représenté de la mêmes façon que l'ensemble des cercles ou des triangles.

B Images concrètes : le format Portable BitMap (PBM)

Ce format permet de stocker une image non compressée noir et blanc, en décrivant la liste des pixels qu'elle contient. Un fichier PBM est constitué d'un en-tête suivi de la liste des couleurs des pixels de l'image. Nous nous intéressons à la version P1 de ce format. Dans P1, le fichier est en *texte brut* : chaque couleur de pixel est désignée par un caractère (`'1'` si noir, `'0'` si blanc).

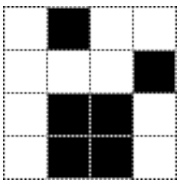
L'en-tête contient trois valeurs, séparées par des espaces ou des sauts de ligne. Les trois valeurs sont les suivantes :

1. la version du format, sur deux caractères (ici P1),
2. la largeur de l'image, en pixels, écrite en texte brut (la chaîne `1024` pour la valeur 1024),
3. la hauteur de l'image, en pixels, écrite en texte brut.

Dans l'en-tête, le dièse (`#`) peut être utilisé pour insérer des commentaires jusqu'à la fin de la ligne.

Les couleurs de pixels sont écrites successivement les unes à la suite des autres, séparées par un espace ou un saut de ligne, que les pixels appartiennent à la même ligne d'image ou non.

Voici une image et le fichier PBM correspondant :



```
P1
4 #largeur
4 #hauteur
0 1 0 0
0 0 0 1
0 1 1 0
0 1 1 0
```

C Interpolation

L'interpolation est une opération consistant à construire une courbe passant par un nombre fini n de points connus $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$, la suite $(x_i)_{0 \leq i \leq n-1}$ étant ici strictement monotone.

Cela équivaut à rechercher une fonction f telle que pour chaque $0 \leq i < n$, $y_i = f(x_i)$.

C.1 Interpolation linéaire

L'interpolation la plus simple est de considérer que le trajet entre deux points est un segment de droite.

C.2 Interpolation polynômiale

Limites de l'interpolation linéaire

L'interpolation linéaire est un algorithme rapide qui ne nécessite pas beaucoup de calculs. Toutefois les trajectoires obtenues ne sont pas très élégantes : les changements de direction sont brutaux aux points de passage. Pour obtenir une meilleure courbe, nous allons définir la trajectoire par une fonction polynomiale.

Plus précisément, étant donnés n points distincts $(x_0, y_0), (x_1, y_1) \ldots (x_{n-1}, y_{n-1})$, l'interpolation polynomiale consiste à trouver un polynôme P de degré minimal (ce degré sera nécessairement inférieur à $n-1$) passant par tous les points : pour i de 0 à $n-1$, on a $P(x_i) = y_i$.

Un tel polynôme existe toujours et est unique. Nous vous présentons maintenant une méthode pour le trouver.

Interpolation Lagrangienne

Polynômes de Lagrange Étant donnés n points $(x_0, y_0), (x_1, y_1) \ldots (x_{n-1}, y_{n-1})$, les polynômes de Lagrange sont n polynômes $l_j(X)$ (pour $0 \leq j < n$) tels que $l_j(x_j) = 1$ et $l_j(x_i) = 0$ si $i \neq j$. Ils sont définis par :

$$\begin{aligned} l_j(X) &= \frac{X - x_0}{x_j - x_0} \times \ldots \times \frac{X - x_{j-1}}{x_j - x_{j-1}} \times \frac{X - x_{j+1}}{x_j - x_{j+1}} \times \ldots \times \frac{X - x_{n-1}}{x_j - x_{n-1}} \\ &= \prod_{i=0, i \neq j}^{n-1} \frac{X - x_i}{x_j - x_i} \end{aligned}$$

Ils sont de degré $n-1$.

Polynôme interpolateur Il est alors aisé de définir un polynôme $L(X)$ tel que pour tout $0 \leq j < n$ $L(x_j) = y_j$. Il suffit de choisir la combinaison linéaire suivante de polynômes de Lagrange :

$$\begin{aligned} L(X) &= y_0\, l_0(X) + y_1\, l_1(X) + \ldots + y_{n-1}\, l_{n-1}(X) \\ &= \sum_{j=0}^{n-1} y_j\, l_j(X) \end{aligned}$$

C.3 Algorithme de Neuville

Une implémentation naïve du calcul du polynôme d'interpolation de Lagrange conduit à de nombreux calculs redondants et n'est donc pas efficace.

L'algorithme de Neuville permet de faire ce calcul de manière beaucoup plus efficace. Au lieu de calculer directement le polynôme de la trajectoire $\mathcal{T}(X)$ puis de l'évaluer ensuite sur le bon point x, on calcule directement la valeur de $\mathcal{T}(x)$ sans construire le polynôme.

Algorithme Étant donnés n points $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$ d'abscisses deux à deux distincts, on note $d_{i,j}(X)$ le polynôme minimal de degré inférieur à $j - i$ qui passe par les points $(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_j, y_j)$.

$d_{0,n-1}(X)$ passe par tous les points. Il est donc identique au polynôme d'interpolation $\mathcal{T}(X)$.

Le principe de l'algorithme de Neuville consiste à calculer $\mathcal{T}(x) = d_{0,n-1}(x)$ par l'algorithme récursif suivant :
– si $0 \leq i < n$:

$$d_{i,i}(x) = y_i$$

– si $0 \leq i < j < n$:

$$d_{i,j}(x) = \frac{(x - x_j)\, d_{i,j-1}(x) + (x_i - x)\, d_{i+1,j}(x)}{x_i - x_j}$$

Optimisation Dans l'algorithme précédent, on est amené à calculer plusieurs fois les mêmes coefficients intermédiaires $d_{i,j}(x)$. Pour éviter cela, une astuce consiste à utiliser une matrice m pour stocker les résultats intermédiaires : la case $m[i][j]$ correspondra au résultat du calcul de $d_{i,j}(x)$.

D Interfacer des programmes C en Python avec SWIG

Il existe diverses manières d'interfacer C et Python. Nous vous proposons d'utiliser `SWIG`. Cet utilitaire est disponible sur `www.swig.org`.

On rappelle que `SWIG` permet de créer à partir d'un fichier de configuration `.i` des modules d'encapsulation pour Python. L'exécutable **swig** utilisé avec l'option **-python** crée, à partir d'un fichier `module.i`, deux fichiers : `module_wrap.c` et `nom_du_module.py`. Le fichier `module_wrap.c` peut alors être compilé en un module `_nom_du_module.so`.

Syntaxe des fichiers d'interface. Un fichier `module.i` se présente comme suit :

```
%module nom_du_module
%{
/* Declarations supplementaires */
```

```
%}

/*Liste de fonctions, variables ou constantes
  a inclure dans le module */
void ma_fonction(void);
```

Le nom du module à créer est indiqué après la déclaration `%module`.

Le contenu du bloc `%{%}` est copié dans le fichier `module_wrap.c` généré, toute inclusion d'entête est donc à déclarer à cet endroit. De même, il est important d'ajouter le ou les fichier(s) d'entête où sont stockés les prototypes des fonctions utilisées.

Enfin, une liste d'éléments (fonctions, variables ou constantes) visibles dans le module doit être écrite. Cette liste peut inclure un entête où une sous-partie est déclarée avec la directive `%include` (semblable à la directive du préprocesseur `#include`). Souvent il suffira donc d'écrire :

```
%module nom_du_module
%{
#include "module.h"
%}

%include "module.h"
```

Passage de tableaux en paramètre. Pour pouvoir passer des listes d'entiers comme paramètres à des fonctions SWIG manipulant des tableaux d'entiers, il est nécessaire d'utiliser un module de traduction, car ces deux types de tableaux ne sont pas de la même nature.

SWIG fournit un utilitaire qui permet de réaliser cette traduction. On l'utilise en incluant dans l'en-tête du fichier `.i` les lignes :

```
%include "carrays.i"
%array_class(int, intArray);
```

Ceci a pour effet de définir dans le module cible la fonction `intArray` prenant en argument un entier n et fournissant en sortie un objet qui s'apparente à un tableau d'entiers de taille n et qu'il est possible de passer en argument à une fonction SWIG comme s'il s'agissait d'un pointeur vers un tableau C. Attention l'objet récupéré n'est pas vraiment un tableau, mais on peut accéder à ses cases avec la notation classique `[ ]`.

Compilation. Voici un exemple de `Makefile` (l'emplacement du répertoire `python` est évidement à adapter) :

```
all: _nom_du_module.so

module.o: module.c module.h
	gcc -c -fPIC module.c

module_wrap.c: module.i module.h
	swig -python module.i

module_wrap.o: module_wrap.c
	gcc -c -fPIC module_wrap.c -I/usr/include/python2.5

_nom_du_module.so: module.o module_wrap.o
	gcc -shared module.o module_wrap.o -o _nom_du_module.so
```