Sebastien Klasa

Glaw

Documentation - Français

Version alpha 0.1.0

Dernière modification le 11/02/15

Sommaire

Introduction

Glaw est une librairie écrite en C++ visant à simplifier la création d'interfaces graphiques dans ce langage. Les outils utilisés par cette librairies sont la SDL et OpenGL. La SDL pour l'ouverture de la fenêtre et la gestion des événements (clavier, souris) et OpenGL pour dessiner dans cette fenêtre.

Glaw est actuellement en version alpha et peut être utilisé sous Linux.
Son fonctionnement sous Windows n'est pa garenti.

Installation

L'installation de Glaw se fait via git en copiant le repository dans le dossier de votre programme.

```
git clone https://github.com/skeggib/Glaw
```

Le fichier *hpp* à inclure est donc :

```
#include "Glaw/glaw.hpp"
```

Préparation

Tout d’abord assurez vous d'avoir bien installé la version de la SDL pour développeurs, pour cela :

Sous Ubuntu :

```
sudo apt-get install libsdl1.2-dev
```

Code source minimal

Voici la code source minimal pour un programme, il comprend :

- L'initialisation des variables nécessaires
- L'initialisation de la SDL et d'OpenGL
- L'ouverture d'une fenêtre
- Une boucle principale minimale
- La fermeture de la SDL

```cpp
#include <iostream>

#include <SDL/SDL.h>
#include <GL/gl.h>
#include <GL/glu.h>

#include "Glaw/glaw.hpp"

using namespace std;

#define MULTISAMPLING true

int main()
{
    /* Variables */

    bool run = true;
    SDL_Surface *window; // Fenêtre

    Input input; // Contient l'état de tout les événements SDL
    memset(&input, 0, sizeof(input)); // Initialise les événements

    Vector windowSize(800, 600); // Taille de la fenêtre

    /* Initialisation */

    SDL_Init(SDL_INIT_VIDEO);

    // Si MULTISAMPLING est égal à true
    // on initialise l'anti-crénelage
    if (MULTISAMPLING)
    {
        SDL_GL_SetAttribute(SDL_GL_MULTISAMPLEBUFFERS, 1);
        SDL_GL_SetAttribute(SDL_GL_MULTISAMPLESAMPLES, 8);
        glEnable(GL_MULTISAMPLE);
        glHint(GL_LINE_SMOOTH_HINT, GL_NICEST );
        glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST );
        glEnable(GL_LINE_SMOOTH);
        glEnable(GL_POLYGON_SMOOTH);
    }

    // Ouverture d'une fenêtre
    window = SDL_SetVideoMode(windowSize.x, windowSize.y, 32, SDL_OPENGL);

    /* Boucle principale */

    while (run)
    {
        updateEvents(&input); // Actualisation des événements

        // Si on appui sur la touche échap ou sur la croix
        if (input.key[SDLK_ESCAPE] || input.quit)
            run = false; // On sort de la boucle principale

        // On efface l'écran
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_TRIANGLES);
```

```
            // On dessine ici

        // On affiche à l'écran ce que l'on vient de dessiner
        glEnd();
        glFlush();
        SDL_GL_SwapBuffers();
    }

    /* Fermeture */

    SDL_Quit();

    return 0;
}
```

L'affichage

Pour afficher des formes en OpenGL il faut le faire via des triangles, une forme complexe est donc un ensemble de triangles qui le forment. Pour organiser ces formes Glaw utilise des conteneurs : un conteneur est un objet qui contient des triangles pour pouvoir les afficher et modifier ensembles.

La structure Vector

Cette structure sert à définir un point sur l'écran, elle contient deux variables : x et y.
Elle peut être initialisée et modifiée en accédant directement à ses variables :

```
Vector point;

point.x = 50.7;
point.y = 12.2;
```

Mais aussi lors de sa déclaration :

```
Vector point(50.7, 12.2);
```

La structure Color

Cette structure permet de définir une couleur RGBA et contient les variables r, g, b et a.
Comme Vector elle peut être initialisée et modifiée en accédant à ses variables mais aussi lors de sa déclaration.

La classe Triangle

Chaque triangle est défini par un objet de type Triangle, il peut être créé, utilisé et affiché de manières indépendante des conteneurs mais peut aussi être à l'intérieur de l'un d'eux.

Coordonnées

Les coordonnées du Triangle sont exprimées en pourcentage du conteneur dans lequel est le triangle. Si le Triangle n'est pas dans un conteneur, ses coordonnées seront un pourcentage de la fenêtre. *Vector(0, 0)* sera donc le coin supérieur gauche de la fenêtre et *Vector(100, 100)* sera le coin inférieur droit.

Déclaration

On peut créer un Triangle de 3 manière différentes :

```
Triangle();
Triangle(Vector apex1, Vector apex2, Vector apex3, Color color);
Triangle(Vector apex1, Vector apex2, Vector apex3, Color color1, Color color2,
Color color3);
```

La première créé un Triangle vierge, le deuxième un Triangle de couleur unie (*apex1*, *apex2* et *apex3* étant les trois sommets du triangle et *color* sa couleur) et la troisième créé un Triangle avec une couleur par sommet (*color1* est la couleur de *apex1* etc...).

Par exemple :

```
Vector a1(20, 40), a2(10, 10), a3(50.5, 70);
Color c1(255, 180, 40, 255);

Triangle t1(a1, a2, a3, c1);
```

Initialisation

Un triangle peut être (ré)initialisé avec sa méthode *set()* :

```
void set(Vector apex1, Vector apex2, Vector apex3, Color color);
void set(Vector apex1, Vector apex2, Vector apex3, Color color1, Color color2,
Color color3);
```

Affichage

Pour afficher un triangle, on appel la méthode draw() entre *glBegin et glEnd* :

```
glBegin(GL_TRIANGLES);

    t1.draw();

glEnd();
```

La classe Object

La classe Object peut contenir des Triangles pour pouvoir les grouper et créer des formes.

Déclaration

```
Object();
Object(Vector pos, Vector size);
```

pos étant la position de l'objet sur l'écran et *size* sa taille en pourcentage de l'écran.

Affichage

L'affichage se fait de la même manière qu'avec les Triangles avec sa méthode *draw()*. Lors de l'affichage d'un Object, tout les Triangles qu'il contient sont affiché en fonction de la position et de la taille de l'Object.

Ajout de Triangles

Pour ajouter des Triangles à un Object, deux méthodes sont possibles :

- L'ajout d'un Triangle existant
- La création d'un nouveau Triangle en l'allouant dynamiquement

L'ajout d'un Triangle existant se fait avec la méthode *add()* :

```
void add(Triangle *t);
```

Il faut passer en paramètre l'adresse du triangle que l'on veut ajouter.

Pour créer un Triangle dans l'Object :

```
Triangle* create(Triangle t);
```

Le Triangle est alloué dynamiquement et ajouté à l'Object. La méthode renvoi l'adresse du Triangle créé.

Il faut bien comprendre que seules les adresses des Triangles sont ajoutés à l'Object, donc si un Triangle que l'on a ajouté est modifié après, cette modification sera visible lors de l'affichage.

Suppression de Triangles

Pour supprimer un Triangle d'un Object il suffit d'appeler la méthode *rem()* de cet Object en lui passant l'adresse du Triangle en paramètre.

```
void rem(Triangle *t);
```

Transformations

Translations

La translation des Triangles peut se faire avec la méthode *translate()* en passant en paramètre un Vector ou deux variables *x* et *y* :

```
void translate(Vector a);
void translate(double x, double y);
```

La translation des Object est à venir dans une prochaine version.

Rotations

La rotation des Objects et des Triangles se fait de la même manière :

```
void rotate(Vector axis, float angle);
```

Avec *axis* le point de rotation, sur un Triangle ce point correspond à une position en pourcentage de l'écran et pour les Objects un pourcentage de cet Object.

L'angle est a donner en degrés.