Framework pour la visualisation d'algorithmes interactifs

Technologie utilisée

- C++11
  - Windows: `MinGW 4.9.1`
  - Debian sid: `g++ (Debian 4.9.1-19) 4.9.1`
- Qt5.4
  - Windows: `QtCreator 3.3.1`
  - Debian sid: `qmake version (5.4.0)`
- OpenGL3.3
  - `GLSL 330`
- GLM 0.9.6
  - Manipulation matricielle, pour matrice projection/model/view
- GSL 1.16
  - Pour le calcul des valeurs/vecteurs propres pour la méthode spectral graph drawing

Exemple de structure de fichier de graphe

```
nSommets 12
oriente 0
value 0
complet 0
debutDefAretes
0 1
0 5
1 2
1 5
2 3
3 6
3 7
4 5
5 6
5 9
9 10
10 11
finDefAretes
```

- `nSommets` : nombre de sommets total
- `oriente` : indique si le graphe est orienté
- `value` : indique si les arêtes du graphe sont pondérées
- `complet` : indique si le graphe est complet
- `X Y` : arête entre le nœud X et Y
- `finDefAretes` : fin de la définition des arêtes

NB: Les informations d'en-tête ne sont pas tout gérés, elles sont présentes mais pas nécessairement prises en compte dans l'avancement actuel du projet.

Compilation

Linux

```
qmake --version
```

Doit être > 5.4 pour l'utilisation de `QOpenGLWidget` entre autres

```
qmake compile.pro
make
```

Génération du makefile puis compilation

```
export LC_NUMERIC=C
```

Au cas où Qt aurait modifié le locale

Les warnings de compilation que l'on peut voir sur cette compilation sont liés à `GLM`.

Windows 7

Pour compiler sur Windows, il faut télécharger la version de QCreator cité au début. Puis importer le fichier `compile.pro` pour y créer un nouveau projet.

Il faut ensuite avoir une version de la GSL portée sur Windows. Beaucoup de problèmes surviennent lors du portage de notre code sur Windows.

Exemple de bug rencontré : la moitiée des arêtes de n'affichent pas correctement Exemple ici

Ajout d'une structure de donnée dans le framework

Pour ajouter une nouvelle structure de donnée au framework, il faut créer une nouvelle classe héritant de la classe mère `Structure`. Il faut ensuite impérativement redéfinir les méthodes de chargement (dans lequel il faudra placer dans l'espace les différentes cellules de donnée de la structure).

Exemple pour la structure tableau

```
class Tableau : public Structure {
private:
    int tableau[10]; // La structure que l'on veut visualiser
    // herité de la classe Structure :
```

```
    // std::vector<SceneVertex> _vertices;
    // std::vector<Etat::Case>  _vertex_to_etat;
public:
    Tableau(std::string path, TextBox* textbox);
    void charger();
    void parcours();
};
```

Les valeurs de l'objet `SceneVertex` du vecteur `_vertices` à la position `i` correspond au placement spatiale de la case `i`. La valeur `Etat::Case` du vecteur `_vertex_to_etat` à la position `i` correspond au type associé à la case `i`.

Dans la méthode `charger`, il faut donc initialiser ces deux vecteurs à partir du fichier d'entrée. Pour placer les cases de notre tableau dans l'espace, il suffit de placer la case `i` sur l'abscisse `i` tout en laissant à 0 sur les deux autres axes.

```
void Tableau::charger()
{
    _vertices.resize(10);
    for(int i(0); i < 10; i++)
    {
        SceneVertex vertice_courant = { i , 0 , 0 };
        vertices[i] = vertice_courant;
    }
}
```

Chaque valeur de `Etat::Case` est associé à une structure de donnée de visualisation (donnée 3D, couleurs, etc).

Le vecteur associant l'état à une case est modifiable durant l'exécution de l'algorithme que l'on veut visualiser.

Modification du rendu durant l'exécution

Pour visualiser un algorithme, il faut donc se placer dans la classe correspondant à la structure utilisée, puis y implémenter une méthode correspondant à notre algorithme.

Exemple en implémentant un simple algorithme de parcours d'un tableau.

```
void Tableau::parcours() {
    for(int i(0); i < 10; i++)
        _vertex_to_etat[i] = Etat::Case::Parcouru;
}
```

Ainsi, durant l'exécution de cet algorithme, nous pourrons voir au fur et à mesure la visualisation des cases parcourus dans la boucle. À chaque itération, nous verrons la case `i` avec les données de visualisation 3D (couleur/modèle 3D etc) correspondant au type `Etat::Case::Parcouru`.

Ajout et modification des données de rendu associés aux types

Pour ajouter et/ou modifier les données 3D associés aux types, il faut se placer dans `SceneGL::charger_contenu_graphique` se trouvant dans le fichier `scene.cpp`, cette fonction va charger au démarrage toutes les données 3D associés aux types enum.

Par exemple, il est possible d'ajouter la valeur enum `Etat::Case::Parcouru_Paire` et `Etat::Case::Parcouru_Impaire`. Puis d'y associer les différentes données de visualisation.

```
init(Etat::Case::Parcouru_Paire, "carre3d.dae", glm::vec3(0.45, 0.45, 0.45),
Qt::white, QFont::Monospace);
init(Etat::Case::Parcouru_Impaire, "mario3d.dae", glm::vec3(0.45, 0.0, 0.0),
Qt::blue, QFont::Monospace);
```

Dans ce cas de figure, les cases qui seront associées au type `Etat::Case::Parcouru_Paire` se verront être visualisé avec le modèle `carre3d.dae` de couleur gris, et leurs valeurs seront affichés en blanc avec une police de type Monospace. Pour les parcourus impaires, ce sera avec le modèle 3d `mario3d.dae` de couleur rouge, et leurs valeurs seront affichés en bleu avec une police de type Monospace.

Il est ensuite possible de modifier la méthode de parcours en conséquence.

```
void Tableau::parcours()
{
    for(int i(0); i < 10; i++)
        if(i%2 == 0)
            _vertex_to_etat[i] = Etat::Case::Parcouru_Paire;
        else
            _vertex_to_etat[i] = Etat::Case::Parcouru_Impaire;
}
```

Fenêtre de log

Pour avoir un suivi personnalisé de l'exécution de l'algorithme, il est possible d'afficher du texte dans une partie de la fenêtre de visualisation.

La classe `Tableau` hérite de la classe mère `Structure` qui a un objet de type `TextBox`, cette classe permet de manipuler le texte se trouvant dans une petite fenêtre de la fenêtre principale.

```
void Tableau::parcours()
{
    int num_ligne = _text_box->newline();
    for(int i(0); i < 10; i++)
    {
        // nouvelle iteration
        _text_box->ajouter("texte ajouter sur la ligne \"num_ligne\"", num_ligne);
```

```
        if(i%2 == 0)
            _vertex_to_etat[i] = Etat::Case::Parcouru_Paire;
        else
        _vertex_to_etat[i] = Etat::Case::Parcouru_Impaire;

        _text_box->remplacer("texte remplaçant la ligne \"num_ligne\" à partir du
caractere 0", num_ligne, 0);
    }
}
```

Source de données pour les graphes

Banque de jolie graphe artistique

- http://www.cise.ufl.edu/research/sparse/matrices/HB/index.html
- http://www.cise.ufl.edu/research/sparse/matrices/groups.html

> récupérable sous format MTX, que l'on peut ensuite adapter à notre structure de fichier de graphe.

Autres banques de graphe représentant divers tas d'informations

- http://konect.uni-koblenz.de/networks/
- http://networkrepository.com/index.php
- http://snap.stanford.edu/data/
- http://www-personal.umich.edu/~mejn/netdata/
- http://law.di.unimi.it/datasets.php