Conduite de projet - CPROJ6 OpenStreeMap Viewer

Perrachon Quentin, Gourgoulhon Maxime, Boufedji Belkacem, *Université Paris Diderot*

Ce document est un rapport pour un projet de programmation, mené en janvier-mai 2016 par Perrachon Quentin, Gourgoulhon Maxime et Boufedji Belkacem, dans le cadre des études en licence d'Informatique à l'Université Paris Diderot.

Table des matières

sources : https://github.com/XAMEUS/CPROJ6

1 Introduction

L'objectif de ce projet est de réaliser en trinôme un logiciel de rendu (renderer) de cartes OpenStreetMap en C. Le logiciel doit être capable de lire un fragment de carte au format XML standard d'OpenStreetMap et de l'afficher à l'écran en utilisant des primitives portables de dessin.

2 Techniques

Cette section explique les détails des différentes techniques (ou algorithmes) utilisés pour ce projet.

2.1 Affichage : vue, déplacements et agrandissement

En OpenGL, `glOrtho`[1] permet de définir la vue de la caméra :

```
void glOrtho(GLdouble left,
             GLdouble right,
             GLdouble bottom,
             GLdouble top,
             GLdouble nearVal,
             GLdouble farVal);
```

Où `left` représente la coordonée à en abscisse minimale (donc à gauche) de la fenêtre, `right` la maximale (à droite) ; de même `bottom` et `top` permettent de définir les valeurs respectivement minimale et maximale en ordonnée ; `nearVal` et `farVal` définissent une notion de profondeur (pour permettre de superposer des plans).

La map au format `.osm` précise quels sont les longitudes latitudes observées :

```
<bounds minlat="39.7492900"
        minlon="-104.9737800"
        maxlat="39.7525610"
        maxlon="-104.9693810"/>
```

2.1.1 Vue

Peut-on associer :

```
void glOrtho(GLdouble minlon,
             GLdouble maxlon,
             GLdouble minlat,
             GLdouble maxlat,
             -1.0f,
             1.0f);
```

Si les dimensions de la fenêtre n'ont pas le même ratio $\frac{width}{height}$ que celles de la map donnée $\frac{maxlon-minlon}{maxlat-minlat}$, l'image sera alors écrasée ou étirée.

Une solution est de définir la hauteur comme limitante $bottom - top = maxlat - minlat$, autrement dit on s'assure que la map prenne toujours toute hauteur possible. Il reste ensuite à trouver quel doit être la largeur correspondante.

Il est possible maintenant de calculer la taille d'un pixel avec $pixelsize = \frac{maxlat-minlat}{height}$; la largeur de la fenêtre doit conserver ce ratio, donc $right - left = width \times pixelsize$.

Il faut aussi implémenter les déplacements et un système pour pouvoir zoomer ; ça va compliquer un peu les choses.

2.1.2 Zoom

Un facteur global *zoom*, par défaut à 1, précise quel est le niveau de zoom ; si $zoom > 1$ c'est un agrandissement, sinon c'est un dézoom.

Gérer le zoom, c'est modifier les valeurs `glOrtho` ; si c'est un agrandissement, il faut les rapprocher, sinon les espacer.

Si vue était simple :

1. https://www.opengl.org/sdk/docs/man2/xhtml/glOrtho.xml

```
void glOrtho(
             -1.0f,
             1.0f,
             -1.0f,
             1.0f,
             -1.0f,
             1.0f);
```

Le facteur zoom peut alors facilement être introduit (car les bornes sont symétriques, l'une est l'inverse de l'autre) :

```
void glOrtho(
             -1.0f*zoom,
             1.0f*zoom,
             -1.0f*zoom,
             1.0f*zoom,
             -1.0f,
             1.0f);
```

Mais dans le cas présent, les bornes ne sont pas symétriques ; il faut recentrer le repère à 0, ensuite appliquer le zoom, puis déplacer le repère obtenu vers sa position d'origine.
$\frac{minlon+maxlon}{2}$ calcule le centre du repère en x et $\frac{minlat+maxlat}{2}$, le centre du repère en y.
$\frac{pixelsize \times width}{2}$ calcule la distance entre le centre du repère et le bord gauche ou droit de la fenêtre.
$\frac{maxlat+minlat}{2}$ calcule la distance entre le centre du repère et le bord haut ou bas de la fenêtre.

```
glOrtho(
        (minlon + maxlon) / 2 - zoom * (pixelsize * width / 2),
        (minlon + maxlon) / 2 + zoom * (pixelsize * width / 2),
        (minlat + maxlat) / 2 - zoom * (maxlat - minlat) / 2,
        (minlat + maxlat) / 2 + zoom * (maxlat - minlat) / 2,
        -1.0,
        1.0
    );
```

2.1.3 Déplacements

Il suffit simplement d'ajouter 2 variables dx et dy (respectivement le décalage en x et en y).

```
glOrtho(
        (minlon + maxlon) / 2 + dx - zoom * (pixelsize * width / 2),
        (minlon + maxlon) / 2 + dx + zoom * (pixelsize * width / 2),
        (minlat + maxlat) / 2 + dy - zoom * (maxlat - minlat) / 2,
        (minlat + maxlat) / 2 + dy + zoom * (maxlat - minlat) / 2,
        -1.0,
        1.0
    );
```

Il faudra faire attention à bien calculer le déplacement en fonction du zoom.

```
dx -= pixelsize * event.motion.xrel * zoom;
dy += pixelsize * event.motion.yrel * zoom;
```

2.2 Projection cartographique

L'image obtenu à partir des données n'est pas parfaite, du moins ne ressemble pas à ce qu'on imagine ; il faut faire une projection cartographique, dans le cas présent ce sera une projection cylindrique ; il s'agit de projeter l'image de la sphère (la terre) sur un cylindre, qu'il est ensuite possible de dérouler (mettre à plat).

2.3 Hashmap

Les `node` doivent être stockés en mémoire après avoir parsé le fichier `.osm`.

3 Compilation

4 Utilisation

5 Liens

1. Liste des projections cartographiques : https://en.wikipedia.org/wiki/List_of_map_projections
2. Projection de Mercator : https://en.wikipedia.org/wiki/Mercator_projection
3. Projection de Miller : https://en.wikipedia.org/wiki/Miller_cylindrical_projection