Université Paris Diderot
L3 Informatique

Conduite de Projet
Année 2015-2016

Rendu OpenStreetMap

version 1.0

1 Introduction

> *OpenStreetMap (OSM) est un projet qui a pour but de constituer une base de données géographiques libre du monde (permettant par exemple de créer des cartes sous licence libre), en utilisant le système GPS et d'autres données libres.*
>
> — http://fr.wikipedia.org/wiki/OpenStreetMap

Dans l'esprit des wikis, les utilisateurs d'OpenStreetMap [6] souhaitant contribuer peuvent fournir des données afin de former progressivement une carte numérique mondiale de plus en plus précise. La façon la plus simple de contribuer est d'enregistrer des itinéraires parcourus dans le monde réel avec un récepteur GPS et, ensuite, de restituer les *traces GPS* correspondantes sur le serveur de données d'OpenStreetMap.

Toutes les mentions utiles (noms, largeur, nature du revêtement, sens uniques, parcs, zones résidentielles et d'activités, barrières, pistes cyclables, boîtes aux lettres, cabines téléphoniques, commerces, fontaines, etc.) sont appelées des *points d'intérêts* (POI, pour « Point of Interest »). Les POI peuvent être ensuite associés, grâce à des logiciels d'édition de carte, aux parties existantes de la carte OpenStreetMap.

Plusieurs types de logiciels sont utilisés pour le bon fonctionnement de l'initiative OpenStreetMap :

- des logiciels d'extraction de traces du récepteur GPS ;
- des logiciels de rendu de cartes ;
- des logiciels d'édition de cartes ;
- des logiciels collaboratifs pour la gestion des données communautaires.

But du projet. Le but de ce projet est de réaliser en trinôme un logiciel de rendu (ou *renderer*) de cartes OpenStreetMap en C. Votre logiciel devra 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.

La progression régulière de votre travail au cours du semestre, de même que le bon usage des outils et des pratiques de conduite de projet présentés dans ce cours, seront au moins aussi importantes pour votre évaluation que le résultat final. Les informations pratiques utiles vous seront fournies sur la page Didel du cours, http://didel.script.univ-paris-diderot.fr/claroline/course/index.php?cid=CPROJ

2 Cartes OpenStreetMap

Les données de la carte XML sont encodés dans un dialecte XML. Chaque document dans le format XML OpenStreetMap contient *un fragment* de la carte OpenStreetMap du monde, qui peut être utilisé indépendamment du reste de la carte. Votre renderer devra être capable de lire un de ces fragments et d'en faire un rendu.

Les informations complètes sur la syntaxe du format XML OpenStreetMap sont disponible sur le site [12]. La DTD complète du format XML est disponible en [11]. Vous trouverez ci-dessous une introduction aux concepts de base et aux éléments fondamentaux du format.

2.1 Map

Un fragment de carte OpenStreetMap est un document XML ayant comme racine un élément `<osm>`, puis un certain nombre de sous-éléments tels que `<node>`, `<way>` ou encore `<relation>`.

Example 1 (squelette de carte en format XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="...">
  <!-- nodes -->
  <node ...> ... </node>
  ...
  <!-- ways -->
  <way ...> ... </way>
  ...
  <!-- relations -->
  <relation ...> ... </relation>
  ...
</osm>
```

2.2 Propriétés

Il existe deux manières d'associer des propriétés aux différents éléments d'un document XML OpenStreetMap.

Attributs. Un certain nombre de propriétés sont directement des attributs XML de l'élément. Pour chaque élément, la DTD précise les attributs possibles et ceux obligatoires. Vous rencontrerez fréquement des attributs communs tels que `id`, `user`, `uid`, `timestamp`, `visible`, `version`, `changeset`. Parmi ceux-ci, les seuls que vous aurez à manipuler sont `visible` et `id`. Si un élément possède l'attribut `visible="false"`, alors il ne doit pas être affiché sur l'écran lors du rendu. Quant à l'attribut `id`, il associe à l'élément un identifiant unique utilisé pour référencer cet élément ailleurs dans la carte.

Tags. Par ailleurs, certains éléments XML d'une carte peuvent aussi avoir un certain nombre d'éléments fils `<tag>`. Les attributs de ces éléments fils vont permettre d'associer des propriétés supplémentaires *clé=valeur* à l'élément XML père. Pour cela, un élément XML `<tag>` doit forcément posséder un attribut `k="clé"` et un attribut `v="valeur"`. L'espace de nom de ces propriétés est a priori libre car non-spécifié par la DTD OpenStreetMap, même si en pratique des conventions à l'intérieur de la communauté OpenStreetMap en fixent l'usage. Il en est de même avec l'éventail des valeurs possibles pour chaque propriété. Un ensemble de propriétés fréquemment utilisées est disponible en [15].

2.3 Nodes

En général, une carte contient plusieurs noeuds (*nodes*). Chaque noeud (élément XML `<node>`) correspond à un point géographique et est associé à une valeur de latitude (attribut `lat`) et à une valeur de longitude (attribut `lon`). Un noeud possède aussi obligatoirement un attribut `id`.

Example 2 (noeud en format XML)

```
<node id="25496583" lat="51.5173639" lon="-0.140043"
    version="1" changeset="203496" user="80n" uid="1238"
    visible="true" timestamp="2007-01-28T11:40:26Z">
  <tag k="highway" v="traffic_signals"/>
</node>
```

2.4 Ways

En général, une carte contient aussi plusieurs chemins (*ways*). Un chemin (élément XML `<way>`) est une séquence ordonnée de noeuds (au minimum deux). Chaque chemin donne une description d'une caractéristique linéaire du paysage, comme par exemple une rue, une fleuve, un chemin de fer, etc.

Example 3 (chemin en format XML)

```
<way id="5090250" visible="true" timestamp="2009-01-19T19:07:25Z"
    version="8" changeset="816806" user="Blumpsy" uid="64226">
  <nd ref="822403"/>
  <nd ref="21533912"/>
  <nd ref="821601"/>
  <nd ref="21533910"/>
  <nd ref="135791608"/>
  <nd ref="333725784"/>
  <nd ref="333725781"/>
  <nd ref="333725774"/>
  <nd ref="333725776"/>
  <nd ref="823771"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Clipstone Street"/>
  <tag k="oneway" v="yes"/>
</way>
```

L'attribut `id` donne un identifiant unique à chaque chemin. Les fils `<nd>` donnent les noeuds qui forment le chemin : pour chaque `<nd ref="`v`">`, il doit exister dans la carte un noeud correspondant `<node id="`v`">`.

2.5 Areas

Les aires sont un cas particulier de chemins. Une aire (aussi appelée *chemin fermé*) est un chemin dans lequel le premier point et le dernier sont identiques.

Example 4 (aire en format XML)

```
<way id="63640943" user="Pieren" uid="17286" visible="true"
    version="2" changeset="8256602" timestamp="2011-05-26T19:44:28Z">
  <nd ref="787355503"/>
  <nd ref="787399257"/>
  <nd ref="787371856"/>
  <nd ref="787407516"/>
  <nd ref="1301125739"/>
  <nd ref="787355503"/>
  <tag k="building" v="yes"/>
  <tag k="source" v="cadastre-dgi-fr Mise a jour: 2010"/>
</way>
```

Dans l'exemple ci-dessous, le premier élément `<nd>` et le dernier ont des attributs `ref` contenant la même valeur, à savoir `787355503`.

La plupart des objets "fermés" qu'on trouve habituellement dans des cartes géographiques (bâtiments, lacs, parcs, aéroports, etc.) sont représentés dans le format de OpenStreetMap par des chemins fermés.

2.6 Relations

NB : les relations peuvent être ignorés dans un premier temps, ils ne sont pas nécessaires pour la réalisation du travail de base (voir Section 3.1).

Pour décrire des objets plus complexes, le format OpenStreetMap propose également des élements `<relation>`, qui permettent de grouper plusieurs chemins et/ou noeuds dans une même entité.

Example 5 (relation en format XML)

```
<relation id="411405" user="Coyau" uid="150478" visible="true"
    version="1" changeset="3904985" timestamp="2010-02-17T21:56:38Z">
  <member type="way" ref="50591602" role="inner"/>
  <member type="way" ref="50591687" role="inner"/>
  <member type="way" ref="50591613" role="outer"/>
  <tag k="type" v="multipolygon"/>
</relation>
```

Une relation permet par exemple de représenter une zone dont la frontière ne peut pas être décrite à l'aide d'un unique chemin. Cela peut être causé par la présence d'un ou plusieurs "creux" dans la zone, ce qui sera marqué par la présence d'éléments `<member>` ayant l'attribut `role="inner"`. Un bâtiment peut ainsi avoir des cours intérieures. La frontière extérieure d'une zone peut également être décrite à l'aide de plusieurs chemins (cf. l'attribut `role="outer"`), par exemple lorsque ces chemins deviennent trop longs pour être facilement gérables d'un seul tenant.

Les relations qui concernent ce projet sont toutes taggées avec la propriété *type=multipolygon*, voir [13] pour plus d'information.

2.7 Encodage des éléments géographiques

Une **rue** est encodée comme un chemin taggé avec la clé `"highway"`. Cette clé peut prendre de multiples valeurs, comme par exemple `"motorway"`, `"road"` ou `"pedestrian"`, *cf.* [15] pour

une liste complète. Une rue doit être rendue comme une ligne graphique ouverte, formée par plusieurs segments qui connectent les points du chemin OpenStreetMap. L'épaisseur de la ligne graphique doit dépendre de l'importance de la rue : plus la rue et importante, plus la ligne correspondante doit être épaisse.

Un **bâtiment** est encodé comme un chemin fermé taggé avec la clé `"building"`. La valeur associée est la plupart du temps un simple `"yes"`, mais elle peut également être plus précise : `"school"`, `"hotel"`, `"church"`, etc. Un bâtiment doit être rendu comme un polygone fermé rempli d'une couleur uniforme, reliant les points du chemin OpenStreetMap correspondant (*cf.* également les améliorations proposées à la Section 3.2.1 lorsque ce bâtiment est muni d'une ou de plusieurs cours intérieures).

Un **cours d'eau** est encodé comme un chemin taggé avec la clé `"waterway"` et des valeurs telles que `"river"`, `"canal"`, etc. Un cours d'eau doit être affiché comme une rue, mais avec une couleur différente pour pouvoir les distinguer.

Un **lac** est encodé comme un chemin fermé taggé avec la clé `"natural"` et la valeur `"water"`. Un lac doit être affiché comme un bâtiment, mais avec une couleur différente pour pouvoir les distinguer.

Un **rivage** (ou **littoral**) est encodé comme un chemin taggé avec la clé `"natural"` et la valeur `"coastline"`. Comme nous travaillons avec des fragments de cartes au lieu d'utiliser la carte du monde entier, les mers et océans sont rarement représentés par des aires complètes, mais plutôt sous forme de chemins séparant terre et eau. Ces chemins de rivages peuvent être fermés (par exemple autour d'une île), mais ils peuvent également ne pas l'être, et juste déborder de part et d'autre de la zone géographique qui nous intéresse, puis s'arrêter.

Par convention, lorsque l'on parcourt un chemin XML représentant un littoral du premier point vers le dernier, l'eau se trouve toujours *à droite* de la direction de parcours courante. Un rivage devra être affiché comme une aire d'eau s'étendant (du bon coté) du littoral jusqu'aux contours de la carte à afficher, ou bien jusqu'aux autres rivages environnants.

Un **espace vert** est encodé comme un chemin fermé taggé avec des clés telles que `"landuse=grass"`, `"landuse=forest"`, ou encore `"leisure=park"`. Un espace vert doit être rendu comme un polygone fermé, avec une couleur distinctive.

2.8 Exemples

Plusieurs exemples de cartes au format XML OpenStreetMap sont disponibles sur *didel*, ainsi que des images de rendus correspondantes. D'autres exemples peuvent être facilement obtenus à partir de l'interface web de OpenStreetMap [6] en utilisant la fonctionnalité d'*export*.

3 Travail demandé

Le projet est à réaliser par **groupe de trois** élèves, qui devront tous aller au même TP.

3.1 Travail minimal

Votre *renderer* devra afficher à l'écran les éléments suivants de la carte OpenStreetMap donnée en entrée :

— rues
— bâtiments
— cours d'eau (fleuves, canaux, etc.)
— lacs

— rivages (de l'océan, de la mer, etc.)
— espaces verts

Pour un renderer minimal, on pourra ne considérer que les objets constitués d'un unique chemin, et ignorer tout autre objet (en particulier les éléments `<relation>`). Les cours d'eau et les rivages pourront être affichés comme de simples lignes de couleur.

La zone à afficher est spécifiée au début des cartes OSM via l'élément `<bound>`, qui donne les coordonnées des coins de la carte. Lors du rendu, vous devrez préserver les proportions : un carré sur le terrain, comme par exemple la place des Vosges à Paris, devra bien apparaître carré à l'écran. Dans une carte OSM, les coordonnées sont exprimés en degrés de latitude et longitude. Les degrés de latitude ont toujours la même dimension (périmètre terrestre divisé par 360). (Nous vous rappelons aussi qu'un parallèle ayant degré de latitude ϕ a une longueur de $c \cdot \cos\phi$, ou c est la circonférence de la Terre.) A l'inverse, les degrés de longitude sont de taille variable selon l'endroit : il y a un facteur correctif à appliquer selon que l'on est près du pôle ou de l'équateur. Comme nos cartes sont de faible dimension, on pourra utiliser le même facteur correctif d'un point à l'autre d'une même carte.

Le programme devra accepter par défaut un nom de fichier `.osm` en argument, et afficher son rendu à l'écran.

3.2 Extensions du travail minimal

Afin de consolider votre note, voici une liste d'extensions possibles pour votre renderer, approximativement triées par ordre de difficulté croissante. Cette liste n'est pas exhaustive, et vous pouvez nous soumettre vos propres idées d'améliorations ou de nouvelles fonctionnalités. Les éléments que vous aurez traités devront être mentionnés dans votre rapport.

3.2.1 Améliorations de l'affichage

Ordre d'affichage. Lors d'un croisement d'une rue principale et d'une rue secondaire, le carrefour devra avoir l'aspect de la rue principale. De même un pont sur une rivière devra apparaître comme une rue et non comme de l'eau. Plus généralement, on prendra soin de gérer les différents types de chevauchement pouvant se produire.

Rivières larges. Les rivières larges telles que la Seine ne sont pas seulement des chemins `"waterway=river"` : chaque tronçon de la rivière est représenté dans sa largeur par des chemins fermés `"waterway=riverbank"` qui suivent une berge un moment, puis la berge opposée. Prendre en compte ces éléments dans l'affichage.

Feuilles de styles. Concevoir une manière de sauver dans un fichier des préférences de style, comme par exemple :
— les couleurs choisies pour chaque catégories d'objets
— les épaisseurs respectives des différents types de rues et de cours d'eau.
— l'affichage ou non de certaines catégories d'objets

Marquages des noms. Permettre l'affichage des noms d'objets (propriété `"name"`). On devra déterminer une manière de choisir une position "au milieu" de l'objet pour afficher son nom. Certaines cartes étant très riches en noms, on pourra essayer de hiérarchiser les objets pour n'afficher que les noms essentiels. Une autre extension possible consiste à anticiper la taille du nom à écrire pour s'efforcer de le placer à un emplacement vide de texte.

Bâtiments creux. Un certain nombre de bâtiments ont des cours intérieures. De tels bâtiments sont encodés sous la forme d'un chemin fermé pour l'exterieur du bâtiment, un chemin fermé par cour intérieure, et enfin une relation de type `"multipolygon"` listant ces différents chemins, en précisant s'ils sont extérieurs (`"outer"`) ou intérieurs (`"inner"`). Permettre un affichage correct de ces cours intérieures.

Aires formés de multiples chemins. Pour pouvoir réutiliser des portions de chemins dans plusieurs éléments adjacents, et/ou pour permettre de manipuler des portions de chemins de tailles plus modestes, certains chemins sont divisés en plusieurs morceaux. Ces morceaux sont eux-mêmes encodés comme des chemins, tandis que la succession de ces chemins est encodé comme une relation de type `"multipolygon"`. Pour de tels éléments subdivisés, reconstituer le chemin global et afficher l'objet correspondant.

Affichage des mers et océans. Pour afficher correctement les rivages de mers et d'océans, il va falloir fermer les `"coastline"` ouvertes, en évitant que ces fermetures artificielles n'apparaissent dans la zone affichée à l'écran. Ces fermetures devront se faire de manière correcte vis-à-vis de la place respective des terres et des mers.

3.3 Nouvelles fonctionnalités

Déplacement et zoom. Le programme n'affiche plus en entier la carte mais seulement une partie. Le choix de la portion affichée se fait donc en fonction d'une résolution et d'un point de référence. Ceci permet d'ajouter les fonctionnalités de zoom (agrandissement et rétrécissement) et de déplacement dans la carte. En fonction de l'echelle courante, la carte est affichée avec plus moins de détails.

Recherche locale. Beaucoup de données sont incluses dans le fichier xml décrivant une carte, en particulier le nom et le type de certains éléments. On peut donc permettre à l'utilisateur de rechercher un lieu suivant ces critères.

Export du rendu. L'export du rendu sous forme d'image pixelisée ne présente guère de difficulté. On pourra par exemple utiliser un format d'image élémentaire [4, 9], puis dériver d'autres formats plus efficaces à partir d'utilitaires de conversion [10]. La production d'une image au format SVG [7], un format vectoriel dérivé de XML, demande un peu plus de documentation et de travail, mais l'avantage de ce format est sa relative insensibilité aux changements d'échelle.

Téléchargement à la volée. OpenStreetMap possède une API (dite *overpass*) qui permet de télécharger des données grâce à des requêtes HTTP (voir [14]). La requête la plus simple est de recevoir l'intégralité des données comprise entre deux latitudes et deux longitudes. **Attention**, la réponse du serveur et le téléchargement de la carte peut prendre un certain temps ; pour permettre une navigation fluide il vaut mieux télécharger à l'avance une carte plus grande et n'en afficher qu'une partie (voir paragraphe *déplacement et zoom*).

Recherche globale. L'API des requêtes *overpass* (voir ci-dessus et [14]) permet de faire rechercher au serveur des lieux dans sa base de donnée, circonscrites à une zone ou non. Ceci permet notamment de permettre à l'utilisateur de rechercher quelque chose et d'en afficher les environs.

Calcul d'itinéraire. Permettre à l'utilisateur de choisir un point de départ, un point d'arrivée et un moyen de déplacement (en voiture ou à pied) et d'afficher un itinéraire adéquat, de préférence *pas trop mauvais*. Il sera donc nécessaire d'approximer le point cliqué par l'utilisateur en un point de la rue la plus proche donc de calculer la distance d'un point à un segment (portion de rue) [3]. Pour calculer un plus cours chemin dans un graphe orienté, voir par exemple l'algorithme de Dijkstra [2] ou l'algorithme A* [1]. Attention, certaines rues sont à sens unique en voiture et d'autres ne peuvent pas être prise à pied (autoroute).

4 Bibliothèques externes

Pour la réalisation du projets, il est nécessaire d'utiliser au moins deux bibliothèques externes : une pour lire le format XML et une pour l'affichage graphique. Les deux bibliothèques mentionnées ci-dessous sont librement utilisables.

Le but de ce cours est aussi de vous laisser une certaine autonomie : vous pouvez donc choisir d'intégrer à votre projet d'autres bibliothèques, à la condition impérative de faire valider votre choix par l'un de vos enseignants.

4.1 Analyse syntaxique du XML : libxml2 [5]

Libxml2 is the XML C parser and toolkit developed for the Gnome project (but usable outside of the Gnome platform), it is free software available under the MIT License. [...] Though the library is written in C a variety of language bindings make it available in other environments.

— http://xmlsoft.org

4.2 Affichage graphique : SDL [8]

Simple DirectMedia Layer (SDL) est une bibliothèque et son API utilisées pour créer des applications multimédias en deux dimensions comprenant si besoin du son comme les jeux vidéo, les démos graphiques, les émulateurs, etc. Sa portabilité sur la plupart des plateformes et sa licence zlib, très permissive contribuent à son succès.

— http://fr.wikipedia.org/wiki/Simple_DirectMedia_Layer

5 Évaluation de votre projet

La notation prendra en compte tous les points suivants.

Utilisation des outils standard. Les cours magistraux présenteront tout au long du semestre des outils standard de développement (Makefile, git, etc.) qui doivent être *effectivement* utilisés pendant le projet. Par exemple, une utilisation frauduleuse de *Makefile* pourrait être un appel unique à un script externe.

Contrôle continu. Le travail doit être réalisé tout au long du semestre grâce à un répertoire *git* par trinôme, répertoire auquel vous donnerez accès à tous les enseignants dès les premières semaines. Ceci permettra notamment de garder la trace et les dates de contributions de chaque étudiant. Vous pouvez choisir entre le service d'hébergement GitLab offert par l'université

http://moule.informatique.univ-paris-diderot.fr:8080/ et tout autre service tiers, toujours à la condition d'y donner accès à vos enseignants.

Lisibilité et clarté. Il est fortement recommandé de séparer les différents modules du programme, d'indenter systématiquement, d'éviter les lignes trop longues (80 caractères), de choisir des noms parlants pour les variables et fonctions, et de commenter le code souvent, mais en restant concis.

Esthétisme. Il vous est demandé d'implémenter un *renderer* : la qualité du rendu est donc importante. Un projet respectant les spécifications techniques d'affichage mais ayant un rendu affreux sera sanctionné.

Références

[1] Algorithme A* pour calcul de chemin raisonnablement court. http://fr.wikipedia.org/wiki/Algorithme_A*.

[2] Algorithme de Dijkstra pour calcul de plus court chemin. http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra.

[3] Calcul d'un point à une droite. http://fr.wikipedia.org/wiki/Distance_d%27un_point_%C3%A0_une_droite.

[4] Formats d'image. https://en.wikipedia.org/wiki/Image_file_formats.

[5] libxml2. http ://xmlsoft.org/html/index.html.

[6] OpenStreetMap web interface. http://www.openstreetmap.org/.

[7] Scalable vector graphics (svg).

[8] Simple directmedia layer (sdl). http ://www.libsdl.org/.

[9] Spécification du format `ppm`. http://netpbm.sourceforge.net/doc/ppm.html.

[10] Imagemagick. Convert et autres commandes. http://www.imagemagick.org/script/index.php, http://www.imagemagick.org/script/convert.php.

[11] OpenStreetMap. API v0.6, DTD. http://wiki.openstreetmap.org/wiki/API_v0.6/DTD.

[12] OpenStreetMap. Data primitives, specification du format XML OpenStreetMap. http://wiki.openstreetmap.org/wiki/Elements.

[13] OpenStreetMap. multipolygon relation. http://wiki.openstreetmap.org/wiki/Relation:multipolygon.

[14] OpenStreetMap. Overpass API. http://wiki.openstreetmap.org/wiki/Overpass_API.

[15] OpenStreetMap. Recommended OpenStreetMap map features. http://wiki.openstreetmap.org/wiki/Map_Features.