COSMetyc : OpenStreetMap en OCaml

Timothé Baleras[1], Martin Bodin[1] et Ugo Comignani[1]

[1]Université Grenoble Alpes, Inria, LIG, 38000 Grenoble, France

Nous présentons COSMetyc, une bibliothèque OCaml pour manipuler les données OpenStreetMap (OSM), une base de données géographique collaborative. Face à l'hétérogénéité des usages, formats et représentations dans l'écosystème OSM, nous proposons une solution modulaire exploitant le système de types d'OCaml pour garantir statiquement la validité des données. Notre bibliothèque permet d'importer et d'exporter des données depuis différents formats (notamment GeoJSON et OSM XML) via différentes représentations typées, de convertir entre systèmes de coordonnées, et d'effectuer des requêtes spatiales efficaces. Nous présentons un retour d'expérience sur la conception de cette bibliothèque et illustrons comment les foncteurs, types fantômes et variants polymorphes d'OCaml permettent de gérer cette complexité.

1 Introduction

1.1 OpenStreetMap : un écosystème riche et hétérogène

OpenStreetMap (OSM) est une base de données géographique collaborative comparable à Wikipédia : n'importe qui peut contribuer en ajoutant, modifiant ou supprimant des données. Sa licence libre ODbL très permissive a conduit à son adoption massive par de nombreux acteurs publics et privés, générant un écosystème d'une grande diversité.

À partir de cette base de données, il est possible de générer des cartes, de construire des applications de routage, des statistiques, des simulations, et bien d'autres applications. La figure 1 illustre cette diversité : rendus généralistes pour le grand public, rendus artistiques, cartes thématiques techniques (voies cyclables, éclairage public), applications de routage, et même utilisations dans des domaines éloignés de la cartographie comme les jeux vidéos ou la simulation numérique.

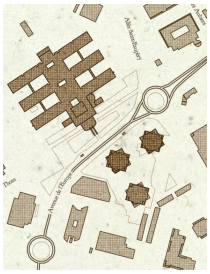
1.2 Problématique et contributions

Cette richesse d'usages s'accompagne d'une complexité importante : les bibliothèques existantes pour manipuler OSM [lib] utilisent une grande diversité de langages de programmation, mais les langages fonctionnels y sont presque absents. De nombreux projets utilisent ces bibliothèques, à des niveaux de maturité technologique [iso13] très variés : certains sont de simples scripts d'importation, d'autres des programmes avec une communauté établie.

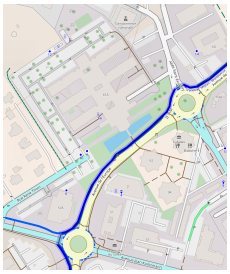
Dans cet article, nous présentons COSMetyc, une bibliothèque OCaml pour manipuler les données OSM. Son code source est disponible à `https://gitlab.inria.fr/mbodin1/logic-osm` avec 8 000 lignes de code et des tests variés. Nos contributions principales sont :

1. Une architecture modulaire basée sur des foncteurs permettant d'adapter la représentation des données aux besoins spécifiques de chaque usage ;

(a) Rendu généraliste

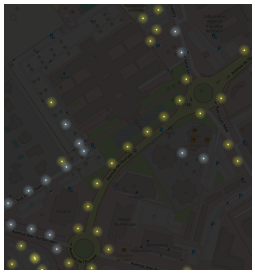


(b) Rendu artistique



(c) Thématiques cyclables

(d) Rendu 3D



(e) Rendu de lampadaires

(f) Routage sur un téléphone

Figure 1. Exemples de rendus générés avec des données OpenStreetMap

2. L'utilisation de types fantômes et variants polymorphes pour garantir statiquement la validité des fichiers OSM XML générés ;
3. Une implémentation complète de conversions entre systèmes de coordonnées (WGS 84, Lambert) basée sur les spécifications officielles ;
4. Un retour d'expérience sur l'utilisation du système de types d'OCaml pour gérer l'hétérogénéité des données géospatiales.

Plan de l'article. La partie 2 présente le contexte hétérogène d'OSM sous quatre aspects : usages, données, bibliothèques existantes et échelles spatiales. La partie 3 décrit comment OCaml permet de représenter et manipuler ces données de manière typée et modulaire. La partie 4 détaille les conversions entre systèmes de coordonnées. La partie 5 présente notre évaluation et un cas d'usage concret. Nous concluons en partie 6.

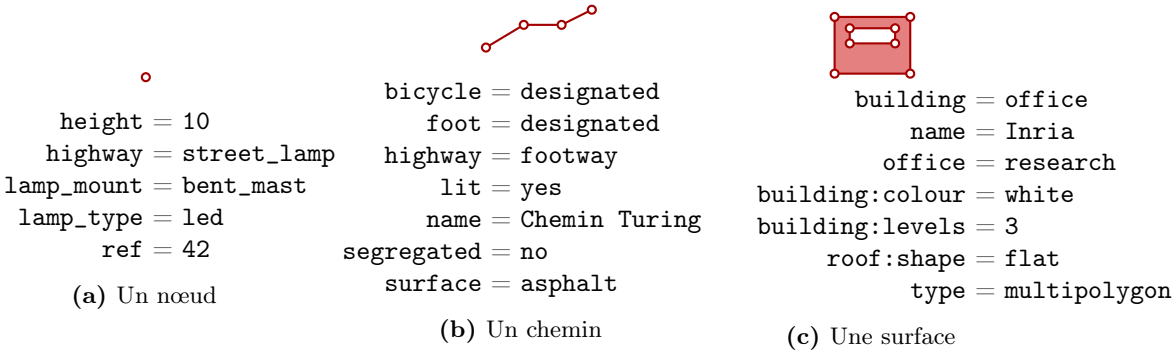
2 Un contexte hétérogène

2.1 Hétérogénéité des données

OpenStreetMap représente le monde via trois types d'objets de base : les nœuds (ponctuels), les chemins (une suite de nœuds), et les relations (une suite d'objets quelconques). Chaque objet est associé à des étiquettes clés/valeurs décrivant ses caractéristiques. La figure 2 montre trois exemples : un lampadaire (nœud), un chemin piéton et cyclable, et un bâtiment (surface, ici représentée par une relation).

Les clés décrivent les caractéristiques pertinentes pour différents usages. Par exemple, un rendu 3D (Figure 1d) utilisera `height` ou `building:levels` pour estimer la hauteur d'un bâtiment, tandis qu'un rendu 2D (Figure 1a) les ignorera. Pour le routage (Figure 1f), la clé `surface` permet de diminuer la priorité des routes avec des surfaces inadaptées.

Surfaces : 3 ou 4 types d'objets ? Le bâtiment 2c est représenté par une surface, qui n'est pas un type de base mais peut être encodée via un chemin fermé ou une relation. Cette



(a) Un nœud **(b)** Un chemin **(c)** Une surface

Figure 2. Exemples de données dans OpenStreetMap

ambiguïté crée deux visions : certains considèrent les surfaces comme un quatrième type de base (modèle à 4 types), d'autres comme une sous-catégorie des chemins et relations (modèle à 3 types). Cette distinction influence fortement la conception des outils : certains formats (comme GeoJSON) stockent les surfaces séparément, tandis que d'autres (comme OSM XML) les encodent via les types de base.

Les clés/valeurs ne sont pas contraintes : c'est l'usage qui prime, documenté sur `wiki.openstreetmap.org`[1]. Des outils comme Osmose [RJ] vérifient la cohérence des données. OSM donne aussi accès à son historique, exploitable via des outils comme Overpass [Rai].

2.2 Hétérogénéité des formats

De nombreux formats coexistent dans la communauté OSM, chacun avec ses dialectes. JOSM (un éditeur populaire) refuse les fichiers OSM XML sans attribut `version`sur les nœuds, mais Overpass [Rai] ne génère pas ces champs. GeoJSON a une spécification [BDD+16] précise sur la géométrie mais floue sur le stockage des clés, menant à des implémentations incompatibles.

Ces variations ne posent pas problème à la communauté car chaque générateur est utilisé dans des contextes différents. Pour COSMetyc, nous devons accepter le maximum de variantes pour ne pas limiter son usage.

2.3 Hétérogénéité spatiale et systèmes de coordonnées

La densité des données varie considérablement : les villes concentrent des milliers d'objets par kilomètre carré, tandis que les océans en contiennent très peu. Cette disparité nécessite des structures de données adaptatives (voir partie 3.3).

De plus, plusieurs systèmes de coordonnées coexistent : OSM utilise l'ellipsoïde WGS 84 [WGS], mais les administrations française et belge utilisent des projections Lambert [Deca, Decb]. La conversion entre ces systèmes est essentielle pour échanger des données (voir partie 4).

2.4 État de l'art des bibliothèques

Comme dit précédemment, les bibliothèques existantes [lib] utilisent une grande diversité de langages (Python, JavaScript, C++, Java, etc.) mais quasiment pas les langages fonctionnels. Ces bibliothèques visent différents objectifs : interfaçage avec l'API serveur, conversion de formats, génération de rendus, etc.

Nous n'y avons identifié qu'un seul fichier OCaml d'une centaine de lignes lisant un sous-ensemble d'OSM XML. La plupart des bibliothèques offrent une interface relativement

1. Les clés de la figure 2 sont cliquables et renvoient à leur documentation.

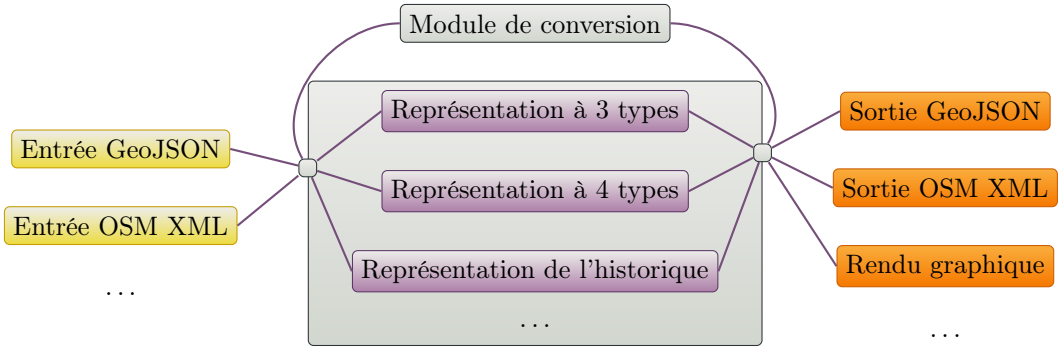


Figure 3. Architecture modulaire de COSMetyc

« bas-niveau » où l'utilisateur manipule directement les structures de données sans garanties de cohérence. COSMetyc se distingue de celles-ci par son approche fortement typée.

3 Forces de la représentation en OCaml

3.1 Modularité via foncteurs

L'hétérogénéité des usages rend difficile l'anticipation de l'utilisation de COSMetyc. Nous proposons donc une architecture modulaire (Figure 3) où la plupart des modules sont paramétrés par un module de représentation qui détermine :

— Le nombre de types d'objets (3 ou 4) ;
— La présence de métadonnées (historique, auteur, etc.) ;
— Le format de stockage des géométries.

Cette approche évite la surutilisation mémoire tout en proposant des structures adaptées à chaque usage. Une utilisation typique consiste en : entrée de données, manipulation sous une forme adaptée, puis sortie. Pour des usages complexes nécessitant plusieurs représentations, un module de conversion est fourni.

Voici un exemple simplifié de signature pour un module de représentation :

```
1  module type REPRESENTATION = sig
2    type node
3    type way
4    type relation
5    (* D'autres types peuvent être définis pour
6      représenter les surfaces, l'historique, etc. *)
7    type object_type  (* Union des types ci-dessus *)
8  end
```

Les foncteurs d'import/export sont ensuite paramétrés par ces modules :

```
9  module Import_GeoJSON(R : REPRESENTATION) : sig
10   val read : string -> R.object_type Seq.t
11 end
```

```
12 module Export_OSM_XML(R : REPRESENTATION) : sig
13   val write : R.object_type Seq.t -> string
14 end
```

Cette généricité permet de construire des traitements (ici import et export) qui s'adaptent à la représentation choisie. Le dossier test en illustre plusieurs instanciations.

Ainsi le fichier test_GeoJson.ml commence par les lignes ci-dessous. La représentation `Surface` est choisie afin de pouvoir sélectionner les surfaces pour un traitement à part, puis les entrées et sorties vers les fichiers GeoJSON sont construites en instanciant leurs foncteurs respectifs.

```
15 module Repr = Osm.Repr.Surface
16 module In = Osm.Input.GeoJson.Make (Repr)
17 module Out = Osm.Output.GeoJson.Make (Repr)
```

3.2 Garanties statiques via types fantômes

Le format OSM XML est utilisé pour communiquer avec le serveur OSM. Il est essentiel de garantir que les fichiers générés sont conformes à la spécification. Nous avons adopté l'approche de TyXML [RGa], qui utilise le système de types d'OCaml pour spécifier ce que chaque balise XML accepte.

Les contraintes sont encodées via des types fantômes (types paramétriques dont le paramètre n'apparaît pas dans la définition) :

```
18 type +'a attrib = Xml.attrib
19 type +'a elt = Xml.elt
```

Le paramètre `'a` n'est utilisé que pour encoder des restrictions. Chaque attribut y est associé à un variant polymorphe :

```
20 val a_date : string wrap -> [> `Date ] attrib
21 val a_uid : int wrap -> [> `Uid ] attrib
22 val a_user : string wrap -> [> `User ] attrib
```

La balise `<comment>` accepte les attributs `date`, `uid`, `user` et du texte comme contenu :

```
23 type ('a, 'b, 'c) star = ?a:('a attrib list) -> 'b elt list -> 'c elt
24
25 val text : string wrap -> [> `Text ] elt
26 val comment : ([< `Date | `Uid | `User ],
27                [< `Text ],
28                [> `Comment ]) star
```

Les types en arguments utilisent des bornes supérieures (`[< ...]`) car la balise peut accepter moins d'attributs. Les types de retour utilisent des bornes inférieures (`[> ...]`) pour permettre la composition.

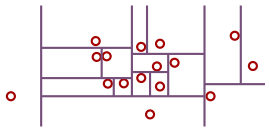
L'implémentation est simple, car les restrictions n'y apparaissent pas :

```
29 let a_date = string_attrib "date"
30 let a_uid = string_attrib "uid"
31 let a_user = string_attrib "user"
32 let comment = star "comment"
33 let text txt = Xml.pcdata txt
```

Un usage erroné provoque une erreur de type. Ce code est ainsi accepté :

```
34 let accepted =
35   comment ~a:[a_user "User"] [text "Text"]
```

Ce code est rejeté (impossible d'imbriquer des `<comment>`) :



(a) Représentation graphique

```
40 type tree =
41 | Leaf of Punctual.t
42 | Horizontal of
43     Spatial.t * tree *
44     Position.Degrees.t * tree
45 | Vertical of
46     Spatial.t * tree *
47     Position.Degrees.t * tree
```

(b) Représentation en OCaml

Figure 4. Les arbres k-d permettent de découper efficacement l'espace

```
36 let refused =
37   comment [
38     comment []  (* Erreur : [> `Comment] incompatible avec [< `Text] *)
39   ]
```

Cette approche garantit statiquement que notre sortie OSM XML respecte la spécification.

3.3 Requêtes spatiales avec arbres k-d

Convertir des fichiers en séquences d'objets est utile, mais les applications ont généralement besoin de requêtes spatiales (obtenir tous les objets dans une zone). Nous avons défini un foncteur Cache stockant les objets pour permettre ces requêtes.

Nous utilisons des arbres k-d [Ben75], arbres de recherche spécialisés pour des objets spatialisés. Chaque nœud représente une zone découpée horizontalement ou verticalement. Cette structure découpe fortement les zones denses (villes) tout en laissant simples les zones peu denses (océans). La figure 4 illustre un découpage de l'espace par cette structure.

Le foncteur Cache est générique sur la source de données :

```
48 module Cache(R : REPRESENTATION)(Input : INPUT with type t = R.t) : sig
49   val query : bbox -> R.object_type Seq.t
50   val add : R.object_type -> unit
51 end
```

À partir d'une séquence d'objet, ce foncteur construit une structure de données facile à requêter et stockant naturellement quelles sont les zones déjà connues des autres. Cette généricité permet d'utiliser le cache avec des fichiers locaux, des requêtes serveur, ou toute autre source, tout en évitant les requêtes redondantes.

3.4 Module simplifié

La quantité de foncteurs peut intimider les débutants. Nous proposons donc un module Basic offrant une interface simplifiée avec des choix par défaut :

```
52 (** Exporte vers OSM XML. *)
53 val write_OSM_XML_string : ?indent:bool -> object_type Seq.t -> string
54
55 (** Génère un rendu PNG.
56     Paramètres : nom de fichier, zone, échelle (pixels/mètre), objets *)
57 val render_png : string -> bbox -> float -> object_type Seq.t -> unit
```

Ce module sacrifie l'expressivité pour la simplicité, permettant une prise en main rapide.

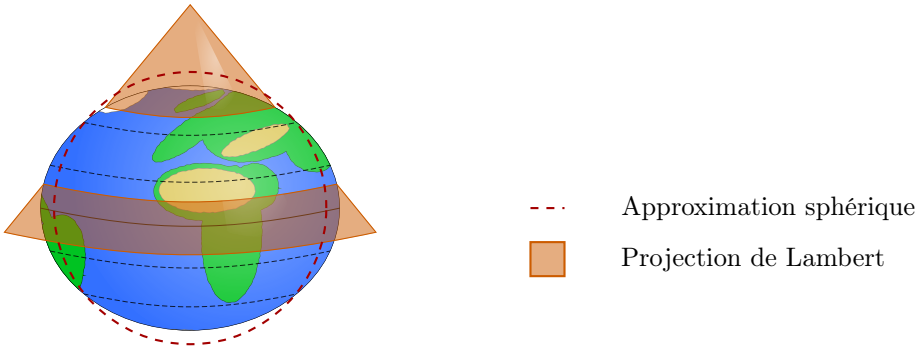


Figure 5. Illustration de différentes projections (excentricité exagérée)

4 Systèmes de coordonnées

4.1 Complexité des modèles terrestres

Choisir un système de coordonnées revient à choisir un modèle de la Terre. La Terre n'est pas parfaitement sphérique : une personne à l'équateur est 0,3 % plus éloignée du centre qu'une personne aux pôles (environ 20 km). Pour la cartographie, une précision de l'ordre du mètre est nécessaire.

Un modèle ellipsoïdal est plus précis mais insuffisant : les reliefs (montagnes, vallées sous-marines) et les variations du champ gravitaire créent des disparités importantes. Le géoïde (surface équipotentielle du champ de gravité) s'écarte de l'ellipsoïde de référence [WGS] jusqu'à 100 m en Inde ou 80 m en Nouvelle-Guinée [EPS].

4.2 Projections cartographiques

Pour représenter la Terre sur une carte plane, diverses projections existent. La projection conique conforme de Lambert [Lam72] (projection utilisée en France et en Belgique) projette la surface terrestre sur un cône (Figure 5), préservant les angles et limitant les déformations de distance sur une tranche de latitudes donnée.

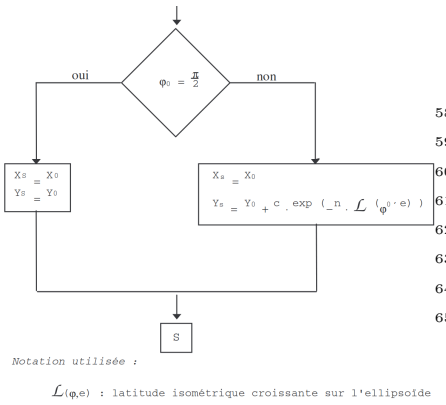
4.3 Implémentation des conversions

OpenStreetMap utilise WGS 84 [WGS], tandis que les administrations françaises et belges utilisent Lambert [Deca, Decb]. Nous avons implémenté les conversions en suivant les spécifications de l'IGN [IGN95].

La figure 6 compare notre implémentation à la spécification officielle. Nous avons privilégié la fidélité à la spécification plutôt que les optimisations, et validé notre implémentation avec les jeux de tests fournis par l'IGN.

4.4 Calcul de distance

Le calcul de distance dépend du modèle. Sur une sphère, la formule de Haversine [dM$^+$95] est simple. Sur un ellipsoïde, les formules de Vincenty [Vin75] sont précises mais coûteuses. Nous avons implémenté trois approches et Position.distance choisit automatiquement : approximation plane (< 100 m), Haversine (100 m à 15 km), Vincenty (> 15 km). Nous avons vérifié l'absence de discontinuités aux transitions.



(a) Spécification IGN [IGN95, Algorithme 54, p. 21]

```
58 let (x_s, y_s) =
59   if phi0 =~ pi /. 2. then
60     (base.x0, base.y0)
61   else
62     (base.x0, base.y0 +. c *.
63      Float.exp (-. n *.
64        isometric_latitude phi0
65          excentricity))
```

(b) Implémentation dans COSMetyc

Figure 6. Comparaison avec la spécification IGN

5 Évaluation et retour d'expérience

5.1 Validation des formats

Pour valider COSMetyc, nous avons produit des fichiers GeoJSON et OSM XML avec différents générateurs et vérifié qu'ils sont acceptés par notre bibliothèque. Réciproquement, nous avons vérifié que nos fichiers générés sont acceptés par JOSM.

Une limitation actuelle : nous ne pouvons pas accepter des fichiers référençant des objets non définis (cas d'extraction partielle d'une zone). Ceci découle de nos choix de représentation modulaire (section 3.1) qui privilégient la cohérence interne.

5.2 Cas d'usage : les lampadaires de Grenoble

En travaux futurs, nous avons reçu de Grenoble Lumières (la compagnie gérant l'éclairage public) une base de données avec l'ensemble des 17 000 luminaires de la ville, incluant leurs positions en coordonnées Lambert et leurs caractéristiques techniques (type de lampe, hauteur, puissance, etc.).

Nous comptons importer ces données dans OSM à l'aide de COSMetyc ce qui fournira à terme une évaluation de l'utilité de la bibliothèque sur un cas réel. Le processus comprendra :

1. Lecture du fichier CSV fourni par Grenoble Lumières ;
2. Conversion des coordonnées Lambert 93 vers WGS 84 (voir partie 4) ;
3. Transformation des attributs métier vers les étiquettes OSM standards ;
4. Vérification de cohérence avec les données OSM existantes ;
5. Génération d'un fichier OSM XML prêt à l'import.

6 Conclusion

Nous avons présenté COSMetyc, une bibliothèque OCaml pour manipuler les données OpenStreetMap, conçue pour gérer l'hétérogénéité inhérente à cet écosystème. Nos contributions principales sont :

— Une architecture modulaire permettant d'adapter la représentation des données (3 ou 4 types d'objets, métadonnées, etc.) aux besoins spécifiques ;
— L'utilisation de types fantômes et variants polymorphes à la TyXML pour garantir statiquement la conformité des fichiers OSM XML ;

— Une implémentation complète et validée des conversions entre les systèmes de coordonnées WGS 84 et Lambert.

Le système de types d'OCaml s'est révélé particulièrement adapté pour encoder les contraintes complexes d'OSM : les foncteurs gèrent l'hétérogénéité des représentations, les types fantômes garantissent la validité des formats, et les arbres k-d permettent des requêtes spatiales efficaces.

Travaux futurs. Plusieurs extensions sont envisagées : support de l'historique OSM complet, optimisation des performances pour de très gros volumes, support de formats additionnels (PBF, Shapefile), et outils d'analyse de qualité des données. Nous espérons que COSMetyc contribuera à renforcer la présence des langages fonctionnels dans l'écosystème OpenStreetMap et facilitera le développement d'applications de cartographie en OCaml.

Remerciements. Nous remercions Grenoble Lumières pour la mise à disposition des données d'éclairage public, ainsi que la communauté OpenStreetMap pour ses retours constructifs. Nous remercions également les relecteurs anonymes pour leurs commentaires détaillés et constructifs qui ont permis d'améliorer substantiellement la structure et la présentation de cet article. Merci aussi à Aurélie Kong Win Chang pour son aide avec LaTeX.

Références

[BDD+16] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub et Stefan Hagen : The GeoJSON Format. *RFC*, (7946), 2016. `https://www.rfc-editor.org/rfc/rfc7946.txt`.

[Ben75] Jon Louis Bentley : Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[Deca] Décret 2006-272. Légifrance. `https://www.legifrance.gouv.fr/loda/id/LEGIARTI000006344188/2006-03-10`.

[Decb] Passage vers le Lambert belge 2008. `https://geoportail.wallonie.be/home/ressources/outils/Lambert-belge-2008-LB08.html`.

[dM+95] José de Mendoza *et al.* : *Memoria sobre algunos métodos nuevos de calcular la longitud por las distancias lunares : y aplicación de su teórica a la solución de otros problemas de navegación.* En la Imprenta real, 1795.

[EPS] WGS 84 to EGM2008 height. EPSG. `https://epsg.org/transformation_3859/WGS-84-to-EGM2008-height-2.html`.

[IGN95] Projection cartographique conique conforme de Lambert : Algorithmes. `https://geodesie.ign.fr/files/geodesie/2025-02/NTG_71.pdf`, 1995.

[iso13] Définition des Niveaux de Maturité de la Technologie (NMT) et de leurs critères d'évaluation. *ISO*, 2013. ISO 16290. `https://www.iso.org/fr/standard/56064.html`.

[Lam72] Johann Heinrich Lambert : *Beyträge zum Gebrauche der Mathematik und deren Anwendung*, volume 3. Verlag des Buchladens der Realschule, 1772.

[lib] Software libraries. OpenStreetMap Wiki. `https://wiki.openstreetmap.org/wiki/Software_libraries`.

[Rai] Martin Raifer : Overpass turbo. `https://github.com/tyrasd/overpass-turbo`.

[RGa] Gabriel Radanne, Stéphane Glondu et al. : TyXML. `https://github.com/ocsigen/tyxml`.

[RJ] Frédéric Rodrigo et Jocelynj : Osmose (OpenStreetMap Oversight Search Engine). `https://osmose.openstreetmap.fr/`.

[Vin75] Thaddeus VINCENTY : Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey review*, 23(176):88–93, 1975.

[WGS] World Geodetic System 1984. EPSG. `https://epsg.org/ellipsoid_7030/WGS-84.html`.