UFR de mathématique et d'informatique

Université de Strasbourg

Exa-MA WP1 - Terrain

Giulio CARPI LAPI

August 23, 2024

Introduction

- Internship took place at **Cemosis - IRMA**
- **PEPR Numpex** through **Exa-MA** project
- Part of the **HiDALGO2** project
- **Ktirio Urban Building application**
- Supervised by **Vincent Chabannes**

Figure: cemosis logo [?]

Figure: Numpex logo [?]

Figure: HiDALGO2 logo [?]

Introduction: Main objectives

- **Exa-MA WP1 - Terrain**

- **Exa-MA WP1 - Vegetation**
- **Exa-MA WP1 - Urban Building LOD-1**
- **Exa-MA WP1 - Urban Building LOD-2**

- Develop a **detailed 3D model** of the urban landscape

Introduction: Main objectives

- Source elevation data from **Mapbox**
- **Reduce mesh density** with **contour line constraints**
- **Parallelization**

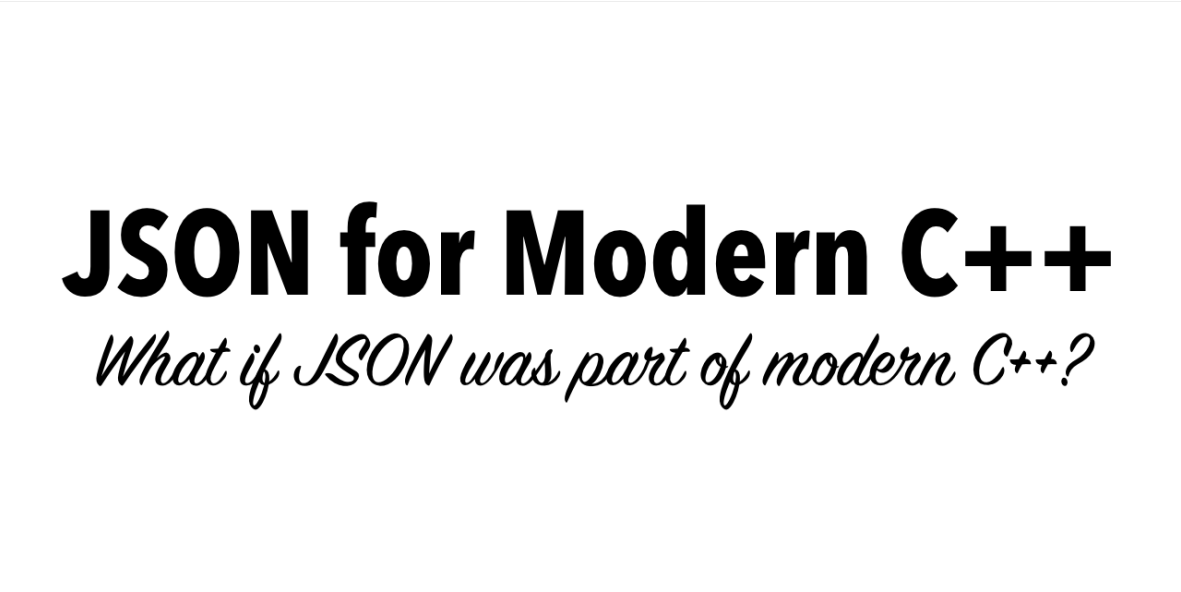

Figure: json nlohmann logo[?]

C++ library for **JSON** parsing

Figure: curl logo[?]

URL transfer library

Introduction: Software and libraries

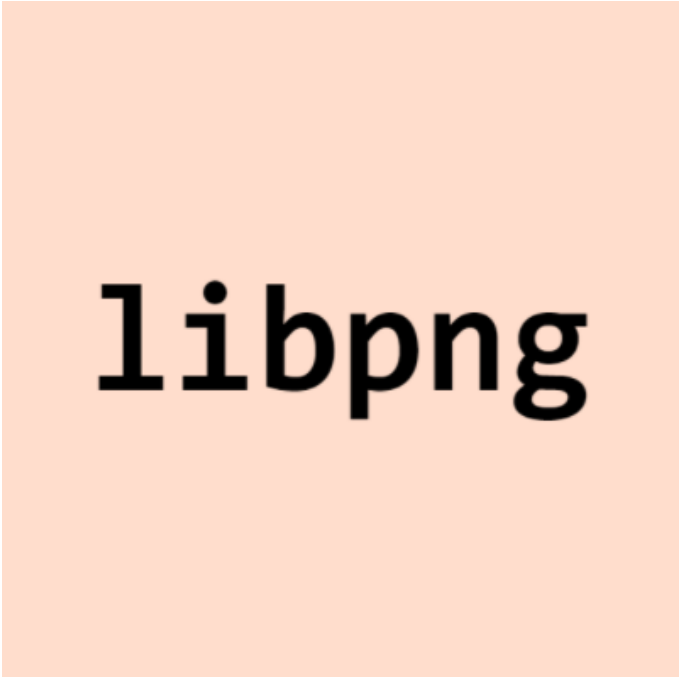

Figure: libpng logo[?]

PNG reference library

Introduction: Software and libraries

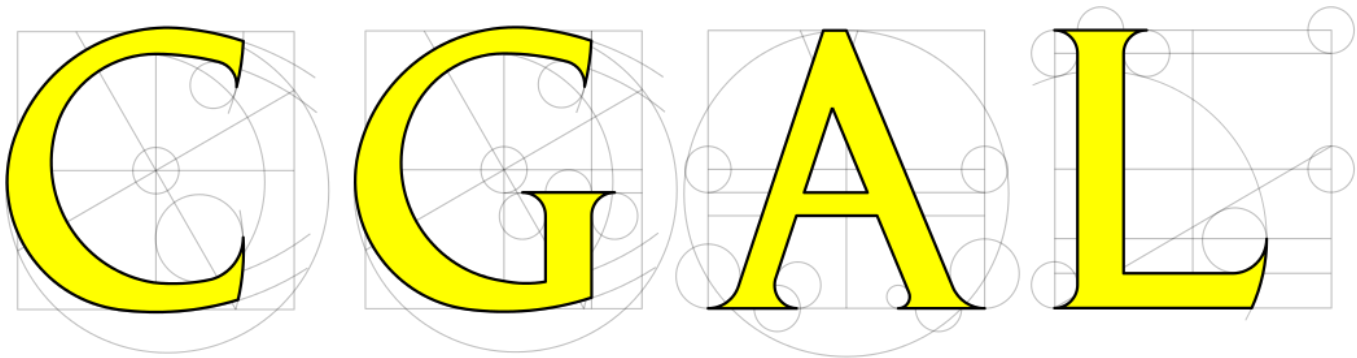

Figure: CGAL logo[?]

Open source software library for **computational geometry algorithms**

Methodology: Pre-existing codebase

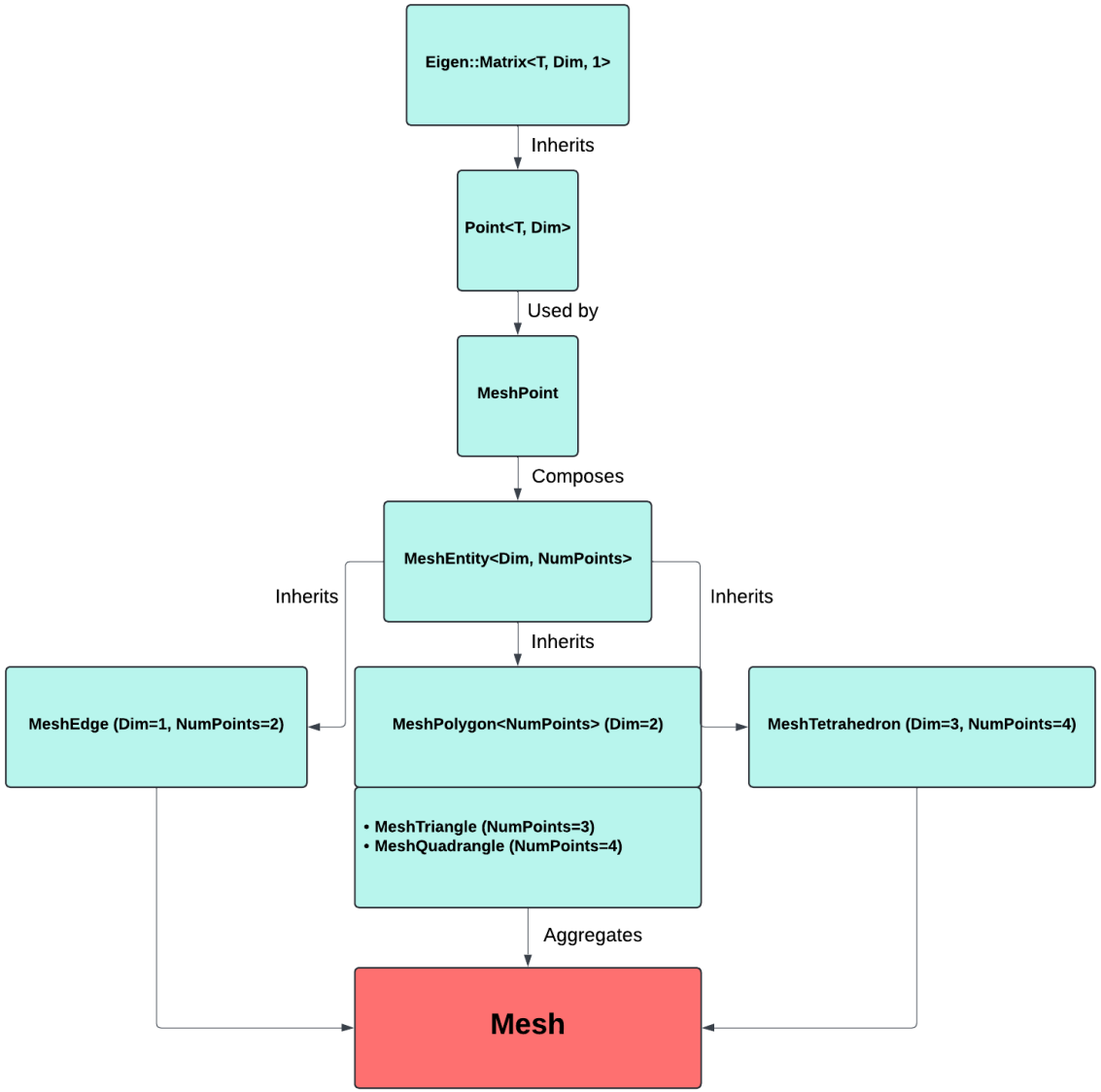

Figure: UML diagram of the codebase structure 1

Methodology: Pre-existing codebase

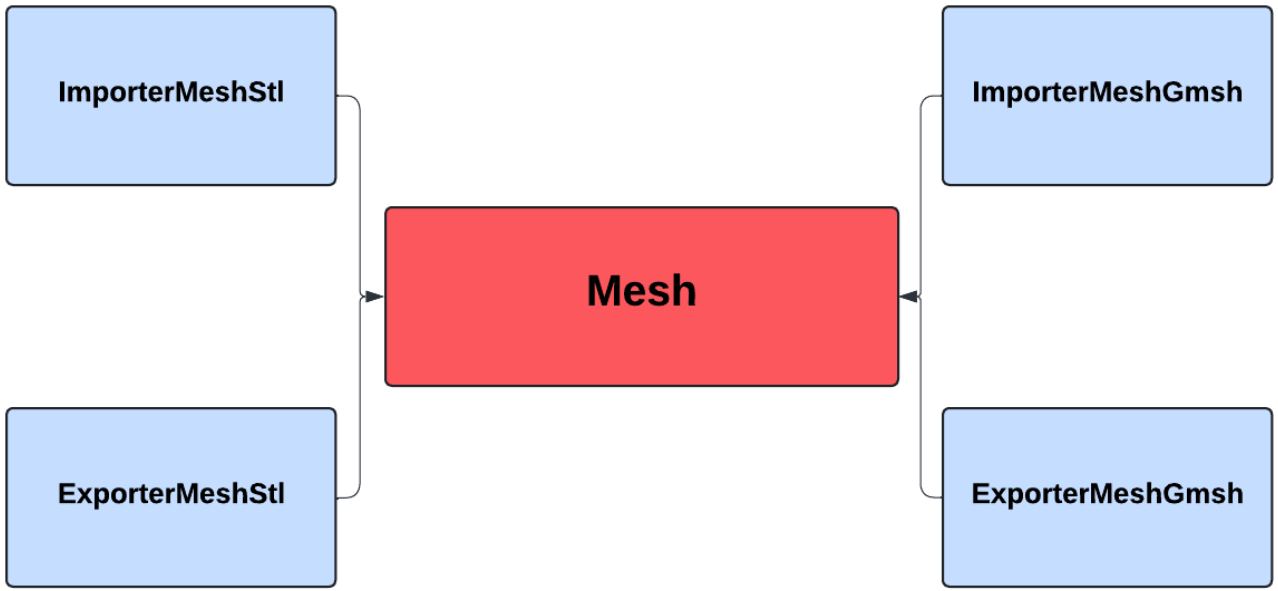

Figure: UML diagram of the codebase structure 2

Methodology: Configuration

```
{
  "useGPS": true,
  "precision": 0.5,
  "longitude": 5.7232,
  "latitude": 45.1835,
  "zoom": 16,
  "api_key": "[YOUR_API_KEY_HERE]"
}

```

Methodology: Latitude and longitude

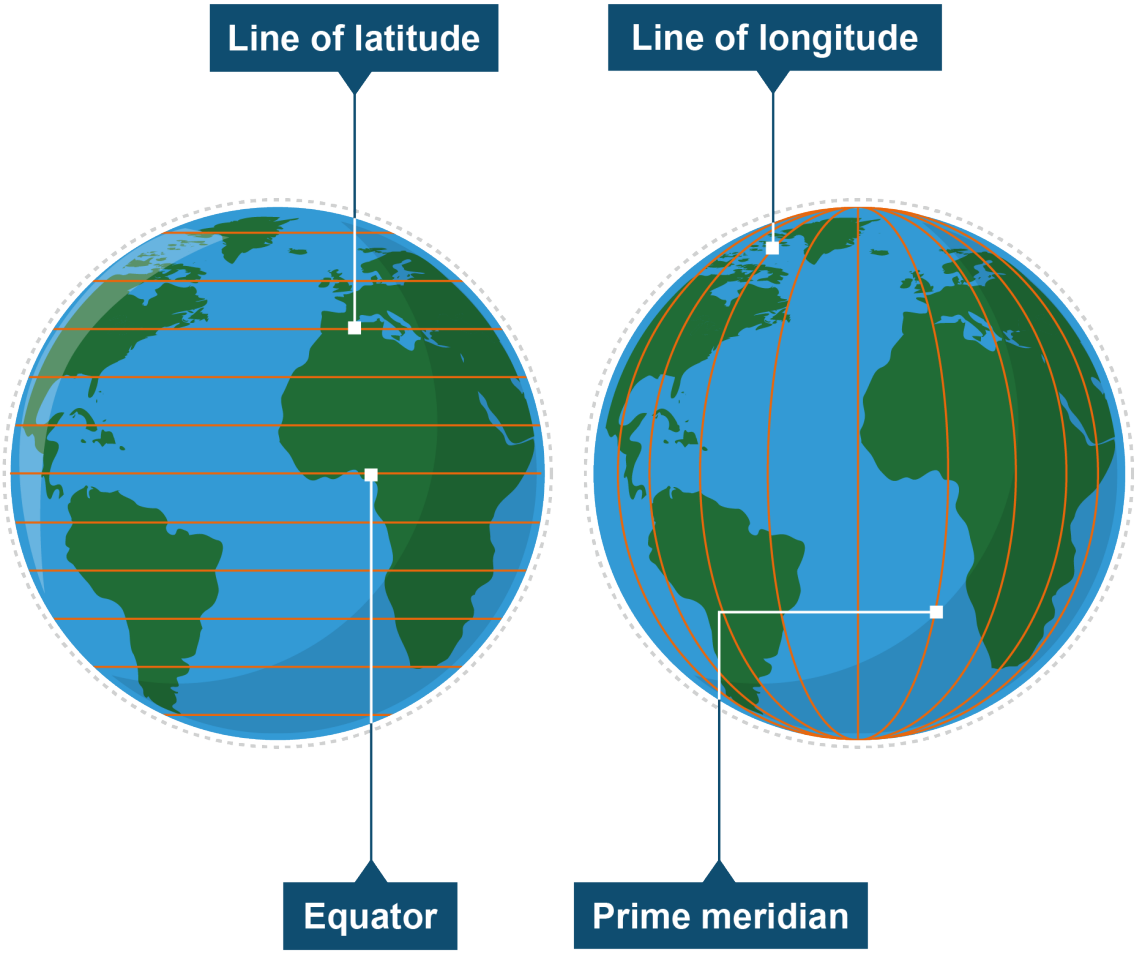

Figure: Latitude and longitude [?]

Methodology: Mercator projection

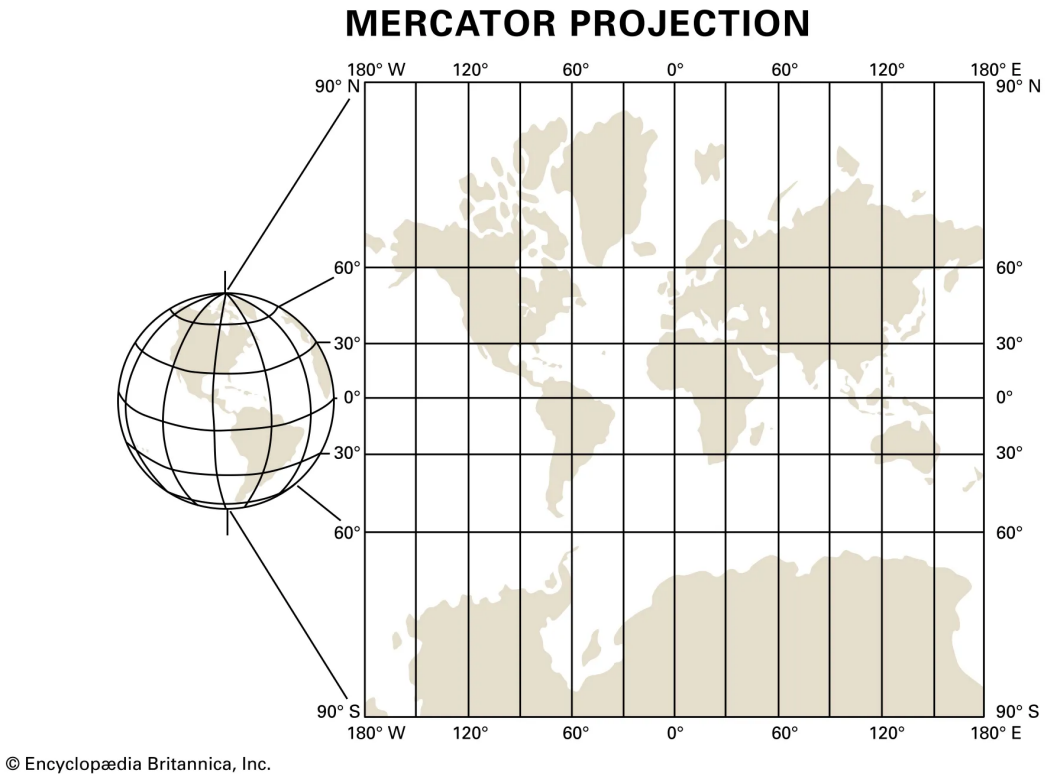

Figure: Mercator projection [?]

Methodology: Tiled web map

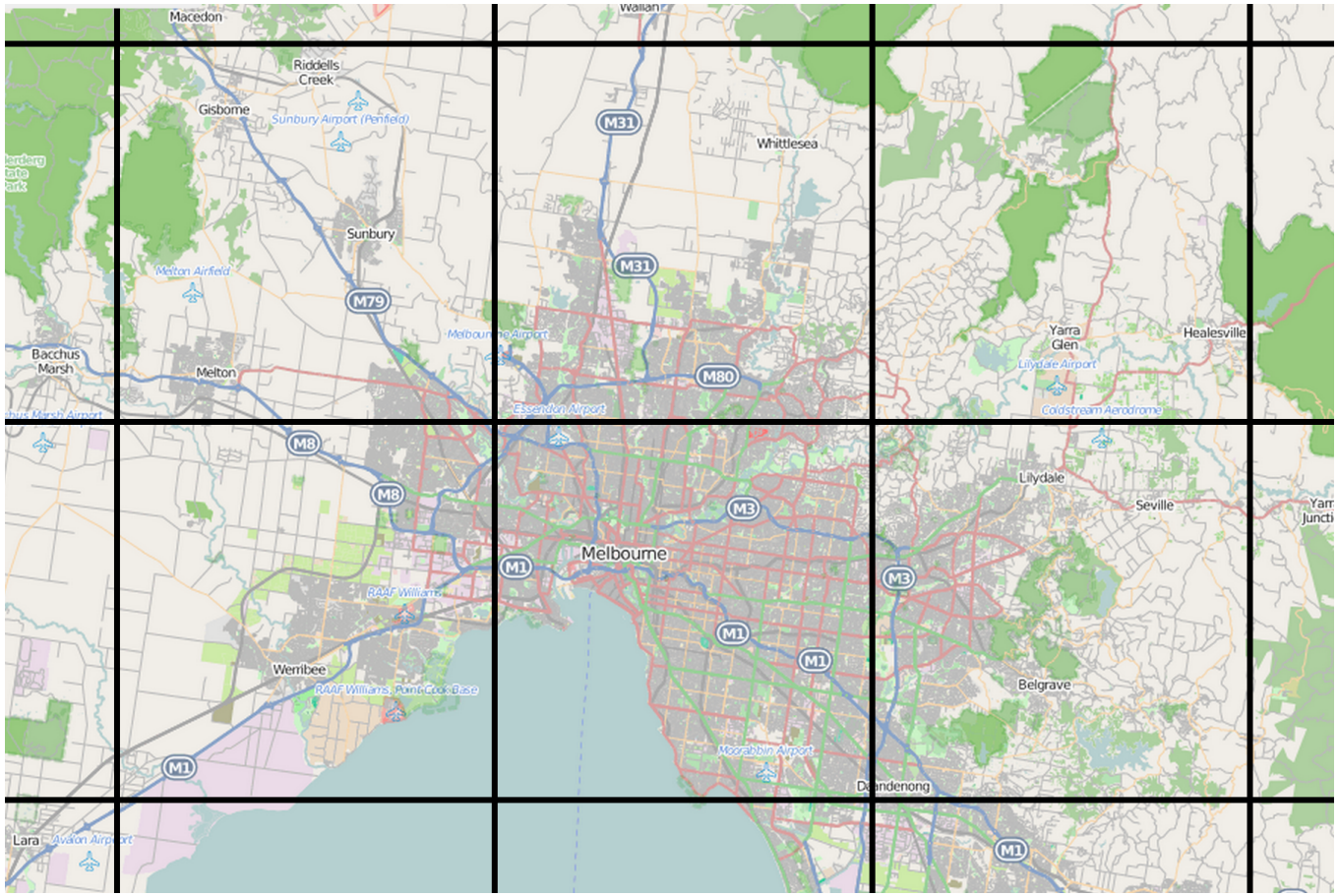

Figure: Tiled web map [?]

Methodology: Tiles coordinates

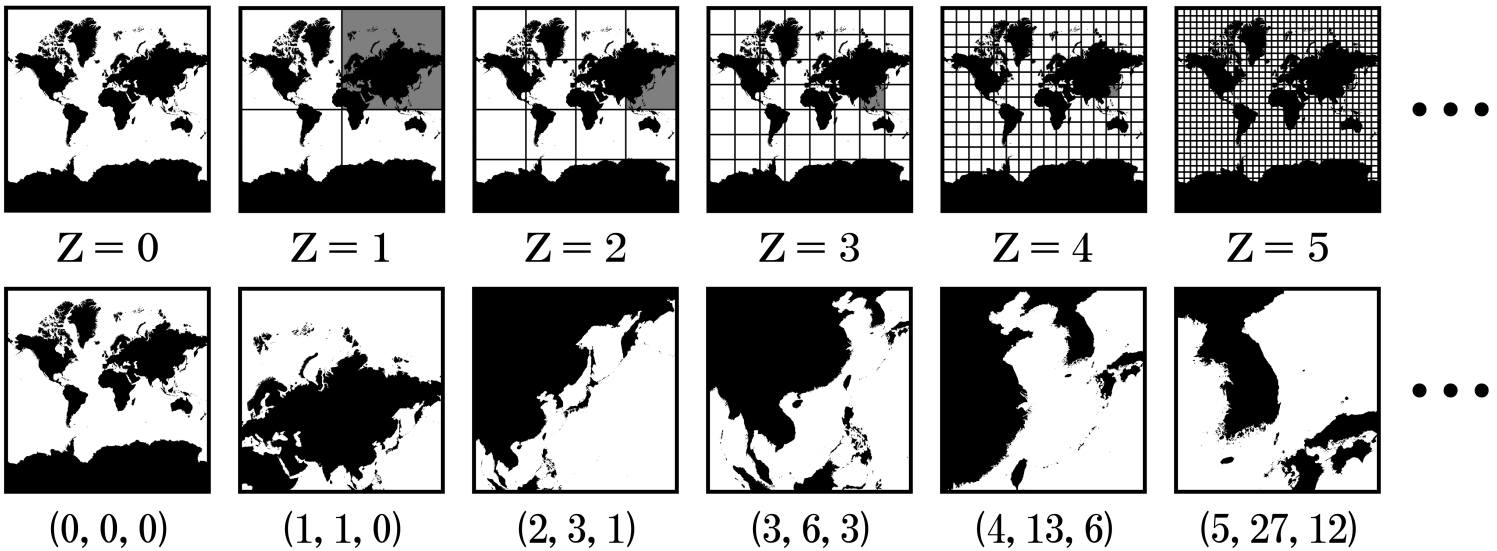

Figure: Tiles coordinates [?]

Methodology: Data acquisition

URL structure for **Mapbox Terrain-RGB v1** API:

```
1 "https://api.mapbox.com/v4/mapbox.terrain-rgb/" + std::
      to_string(zoom) + "/" + std::to_string(x) + "/" + std::
      to_string(y) + ".png?access_token=" + api_key;
2
```

Methodology: Geographic coordinates to tile coordinate

$$x = \left\lfloor \frac{\text{lon} + 180}{360} \times 2^{\text{zoom}} \right\rfloor$$

$$y = \left\lfloor \left(1 - \frac{\log\left(\tan\left(\frac{\text{lat} \times \pi}{180} \right) + \frac{1}{\cos\left(\text{lat} \times \frac{\pi}{180} \right)} \right)}{\pi} \right) \times 2^{\text{zoom}} \right\rfloor$$

Function to read PNG data:

```
1  std::vector<std::vector<std::tuple<int, int, int>>> readPNG(
       const std::vector<char> &data)
2  {
3    std::vector<std::vector<std::tuple<int, int, int>>> image;
4    std::vector<std::tuple<int, int, int>> row;
5    for (unsigned x = 0; x < image_struct.width; ++x)
6    {
7        int r = buffer[3 * (y * image_struct.width + x) + 0];
8        int g = buffer[3 * (y * image_struct.width + x) + 1];
9        int b = buffer[3 * (y * image_struct.width + x) + 2];
10       row.emplace_back(r, g, b);
11   }
12     return image;
13 }
14
```

Methodology: Mapbox Terrain-RGB v1

$$height = -10000 + ((R \times 256 \times 256 + G \times 256 + B) \times 0.1)$$

Methodology: Delaunay triangulation

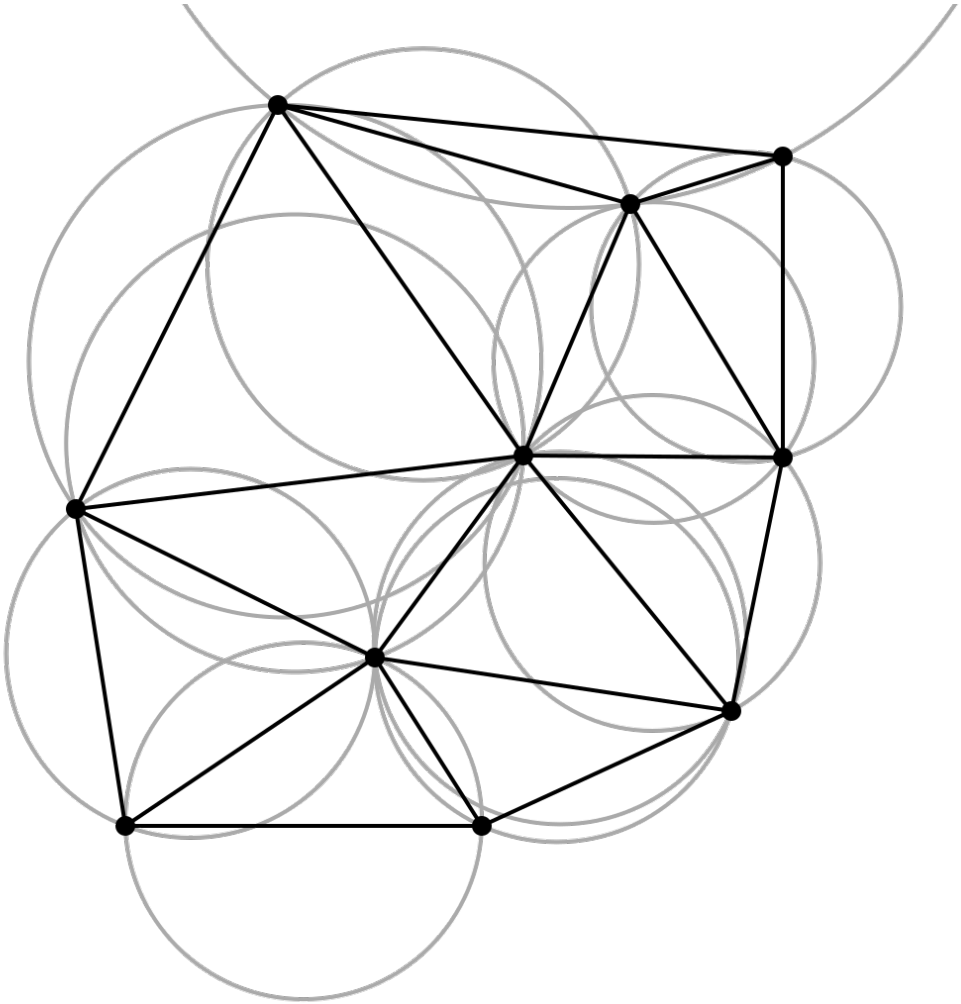

Figure: Delaunay triangulation [**?**]

Methodology: Constrained Delaunay triangulation

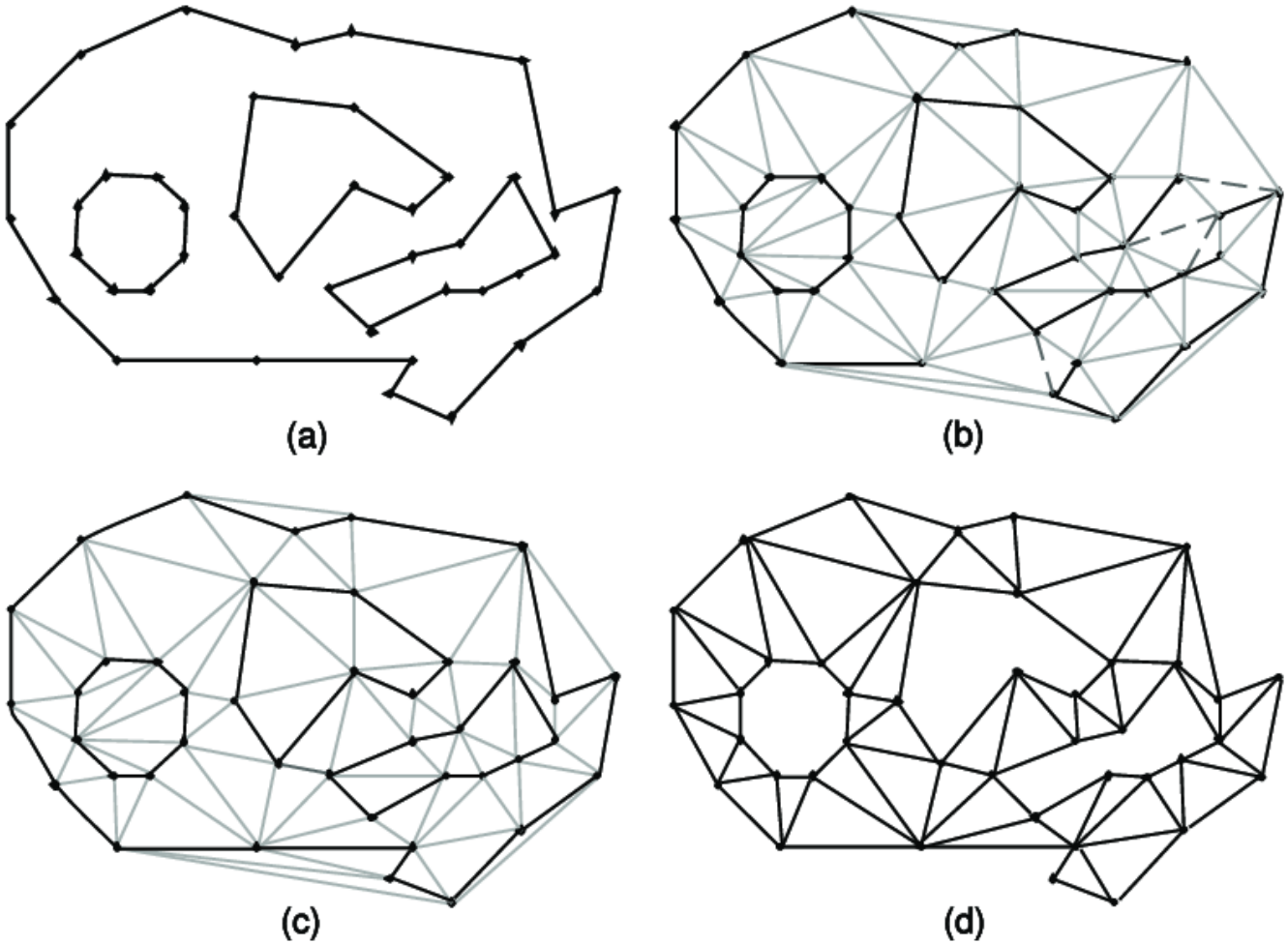

Figure: Constrained Delaunay triangulation [?]

Implementation: Project structure

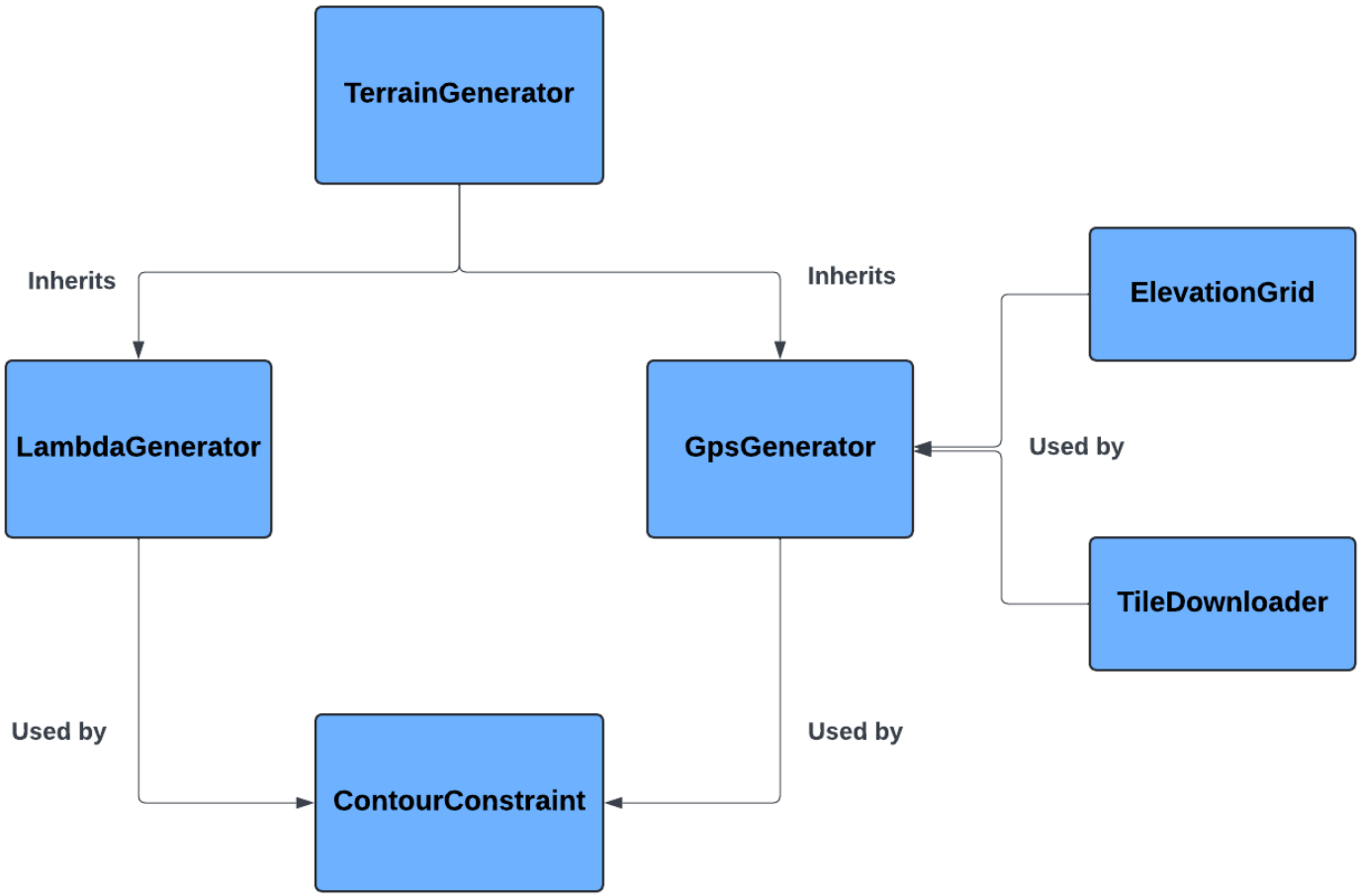

Figure: UML diagram of the project structure

Implementation: Mesh generation

TerrainGenerator class definition:

```
1  class TerrainGenerator
2  {
3  public:
4      using mesh_type = Mesh;
5      virtual std::unique_ptr<mesh_type> generate() = 0;
6      virtual ~TerrainGenerator() {};
7  };
8
```

LambdaGenerator class definition:

```
1  class LambdaGenerator : public TerrainGenerator
2  {
3  public:
4      using fn_type = std::function<double(double, double)>;
5      LambdaGenerator(int meshSize, fn_type const &fn);
6      virtual ~LambdaGenerator();
7
8      std::unique_ptr<mesh_type> generate() override;
9
10 private:
11     int M_meshSize;
12     fn_type M_fn;
13 };
14
```

Implementation: Mesh generation

$$z(x, y) = \sqrt{r^2 - (x - x_0)^2 - (y - y_0)^2}$$

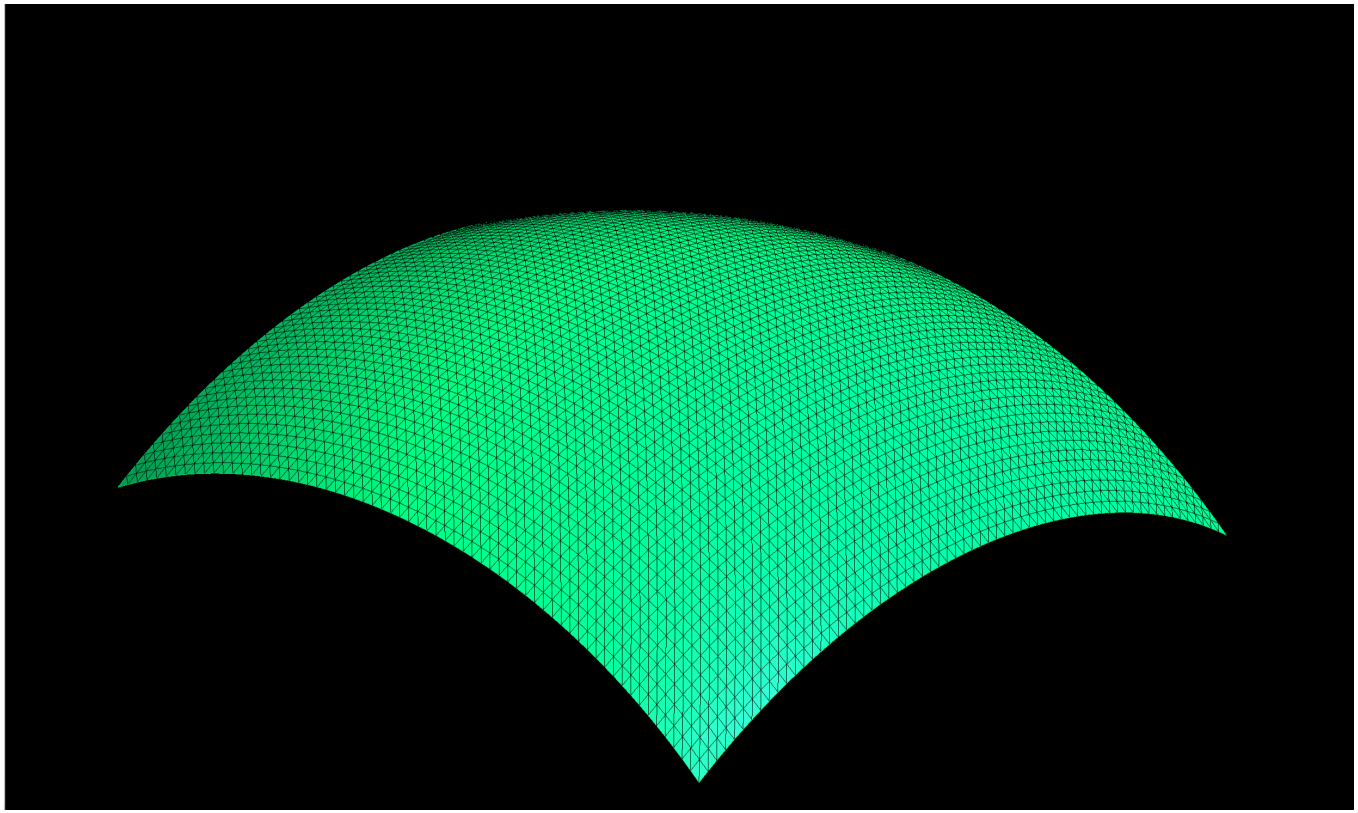

Figure: 3D mesh **LambdaGenerator** 1

Implementation: Mesh generation

$$z(x, y) = sin(x) + cos(y)$$

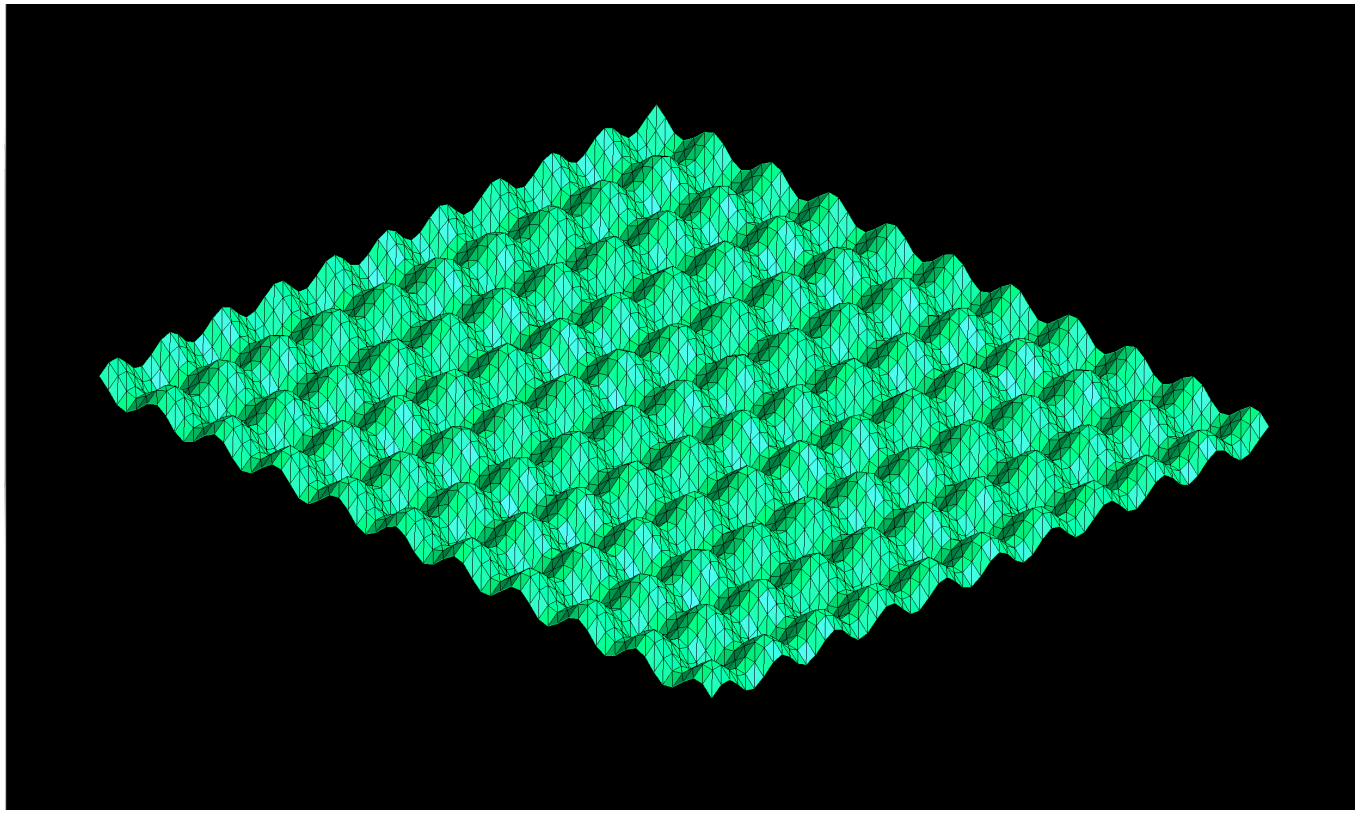

Figure: 3D mesh **LambdaGenerator** 2

GpsGenerator class definition:

```
1  class GpsGenerator : public TerrainGenerator
2  {
3  public:
4      GpsGenerator(int meshSize, double latitude, double
       longitude, unsigned zoomLevel, std::string const &apiKey
       );
5      virtual ~GpsGenerator();
6
7      std::unique_ptr<mesh_type> generate() override;
8
9  private:
10     int M_meshSize;
11     double M_latitude;
12     double M_longitude;
13     unsigned M_zoomLevel;
14     std::string M_apiKey;
15 };
16
```

Implementation: Mesh generation

$(x, y) = (33809, 23527)$ at zoom level 16

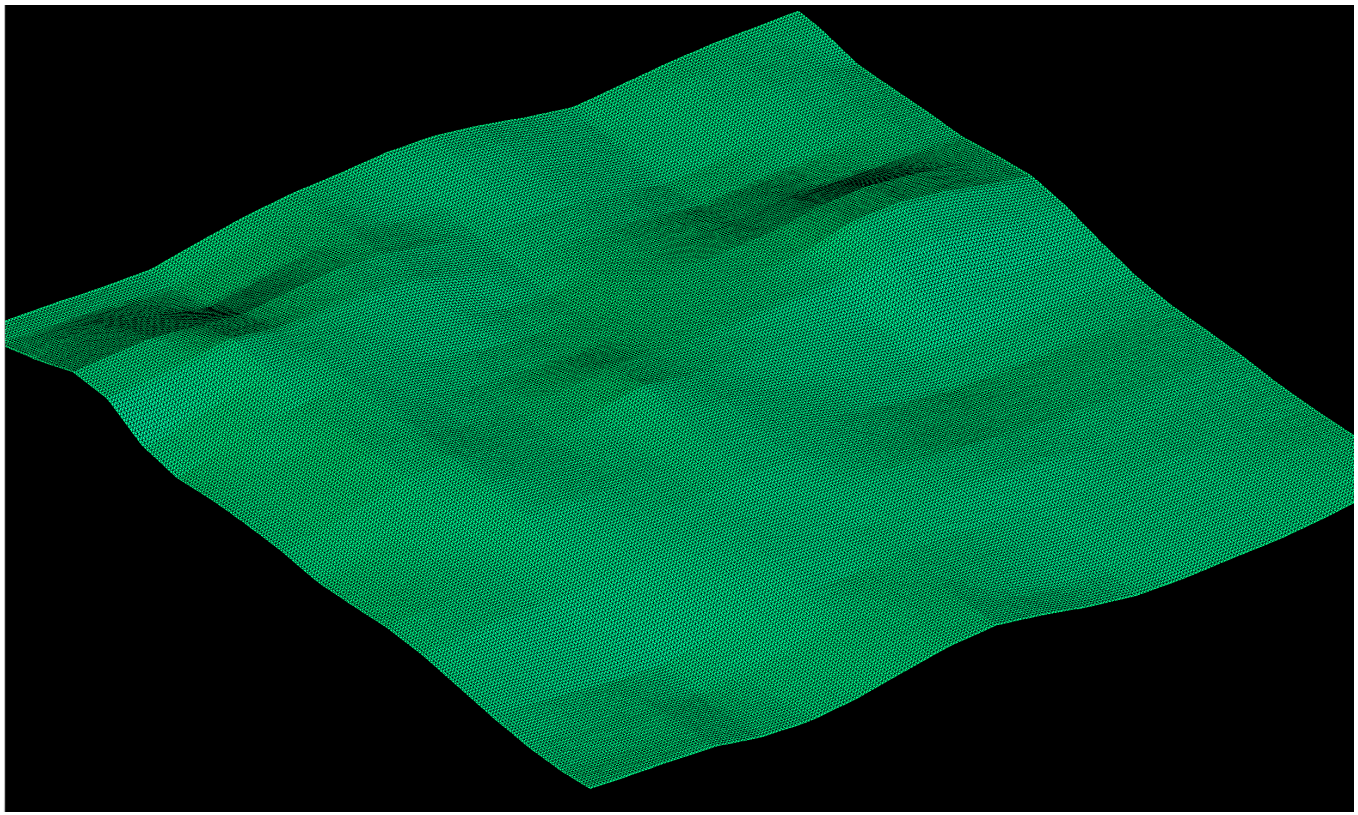

Figure: 3D mesh **GPS**

Implementation: Contour lines generation

ContourConstraint class:

- **computeAllContoursMeshes** function
- **getMinMaxZ** function
- **computeContourMesh** function

Implementation: Contour lines generation

$$z(x, y) = \sqrt{r^2 - (x - x_0)^2 - (y - y_0)^2}$$

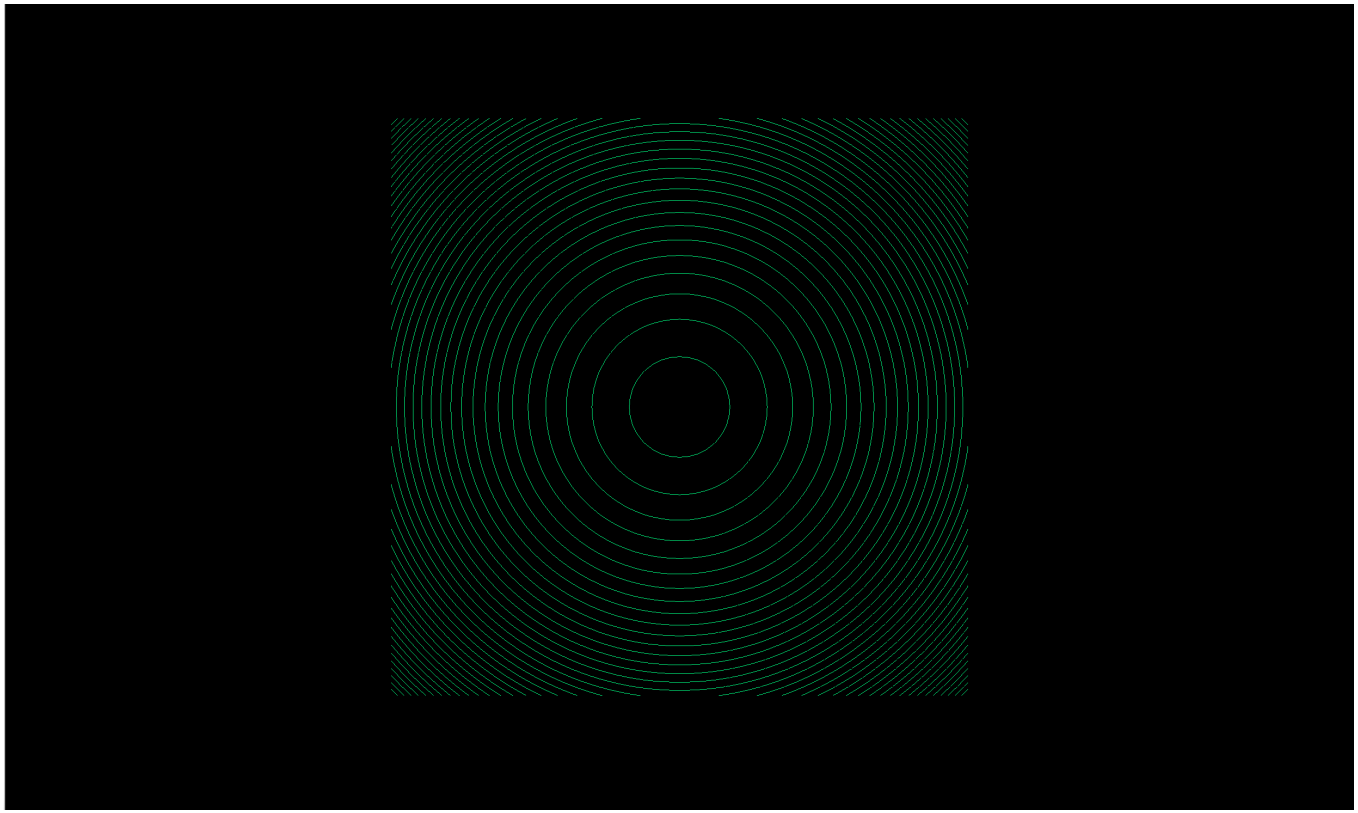

Figure: 3D mesh **contour lines** 1

Implementation: Contour lines generation

$$z(x, y) = sin(x) + cos(y)$$

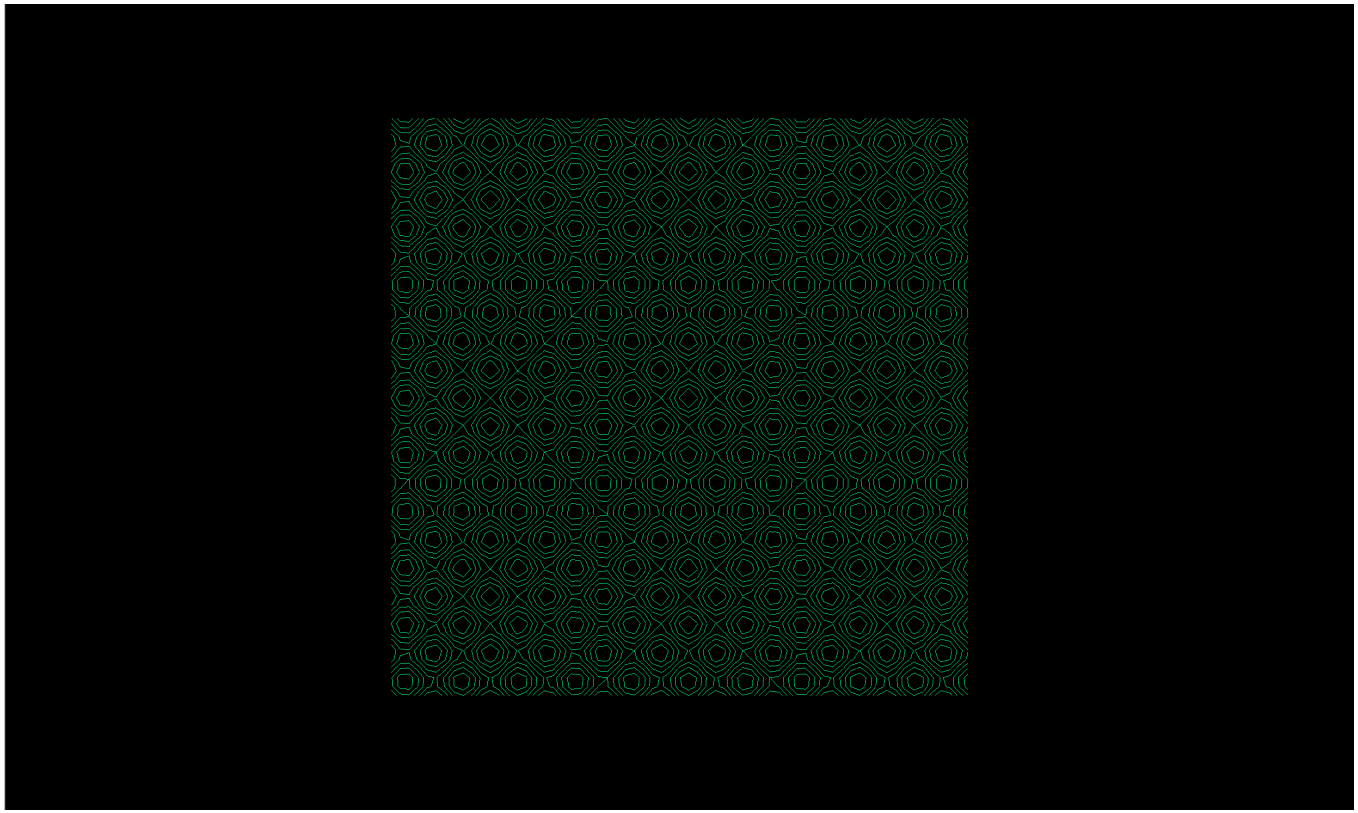

Figure: 3D mesh **contour lines** 2

Implementation: Contour lines generation

$(x, y) = (33809, 23527)$ at zoom level 16

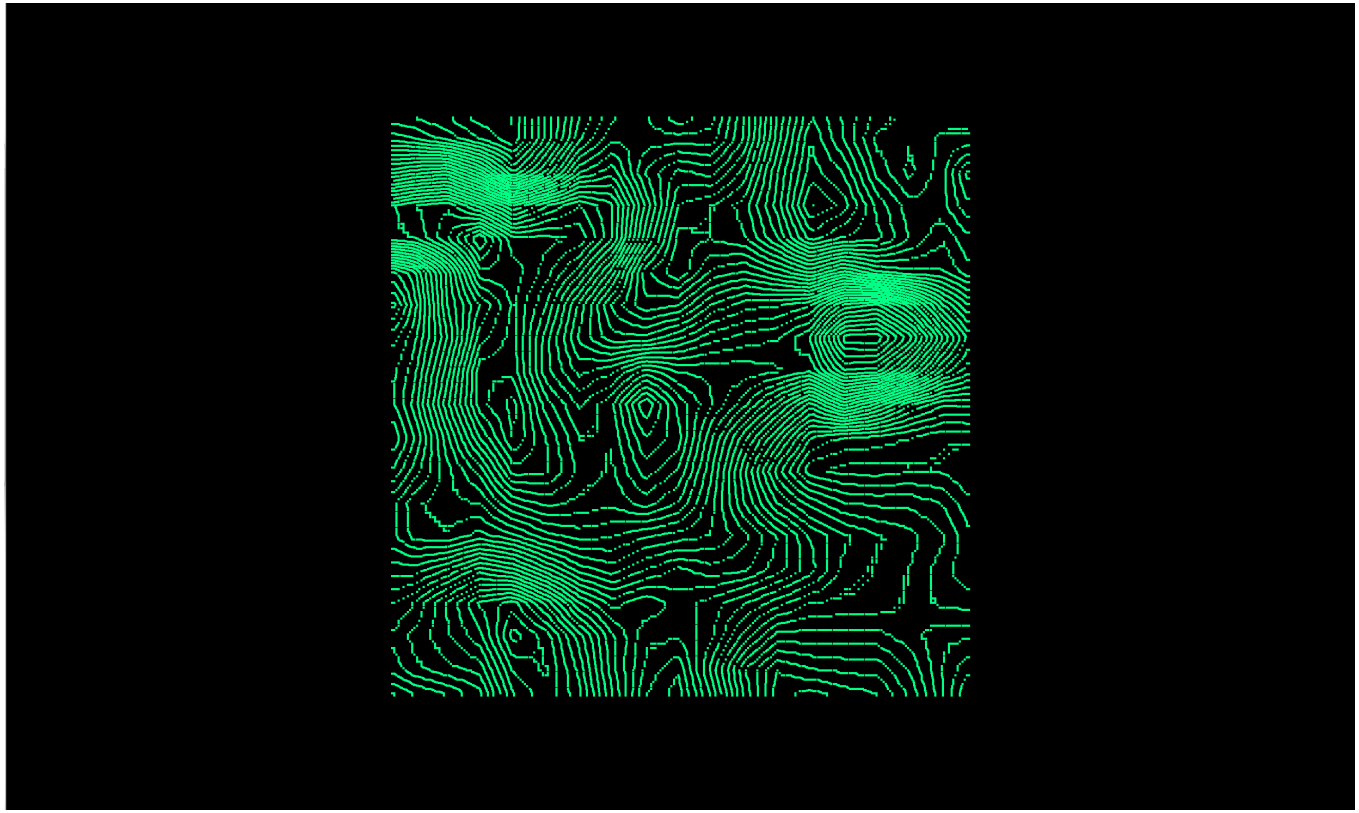

Figure: 3D mesh **contour lines GPS**

Implementation: Re-triangulation

triangulateAssembledMesh function:

- **Inserting constraints** into **Constrained_Delaunay_triangulation_2**
- **CGAL::make_conforming_Delaunay_2**
- **Generating** the **final mesh**

Results

$$z(x, y) = \sqrt{r^2 - (x - x_0)^2 - (y - y_0)^2}$$

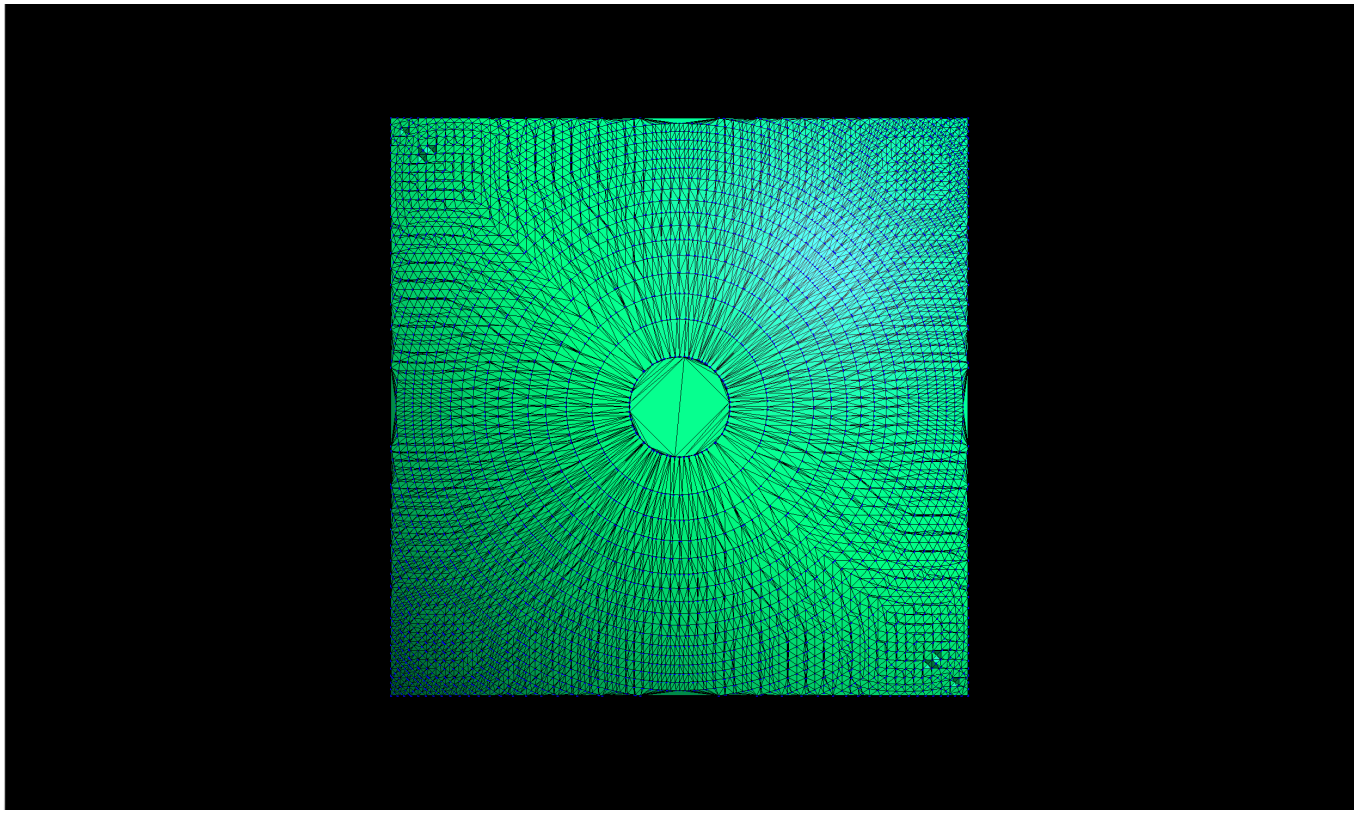

Figure: 3D mesh after re-triangulation (lambda function 1)

Results

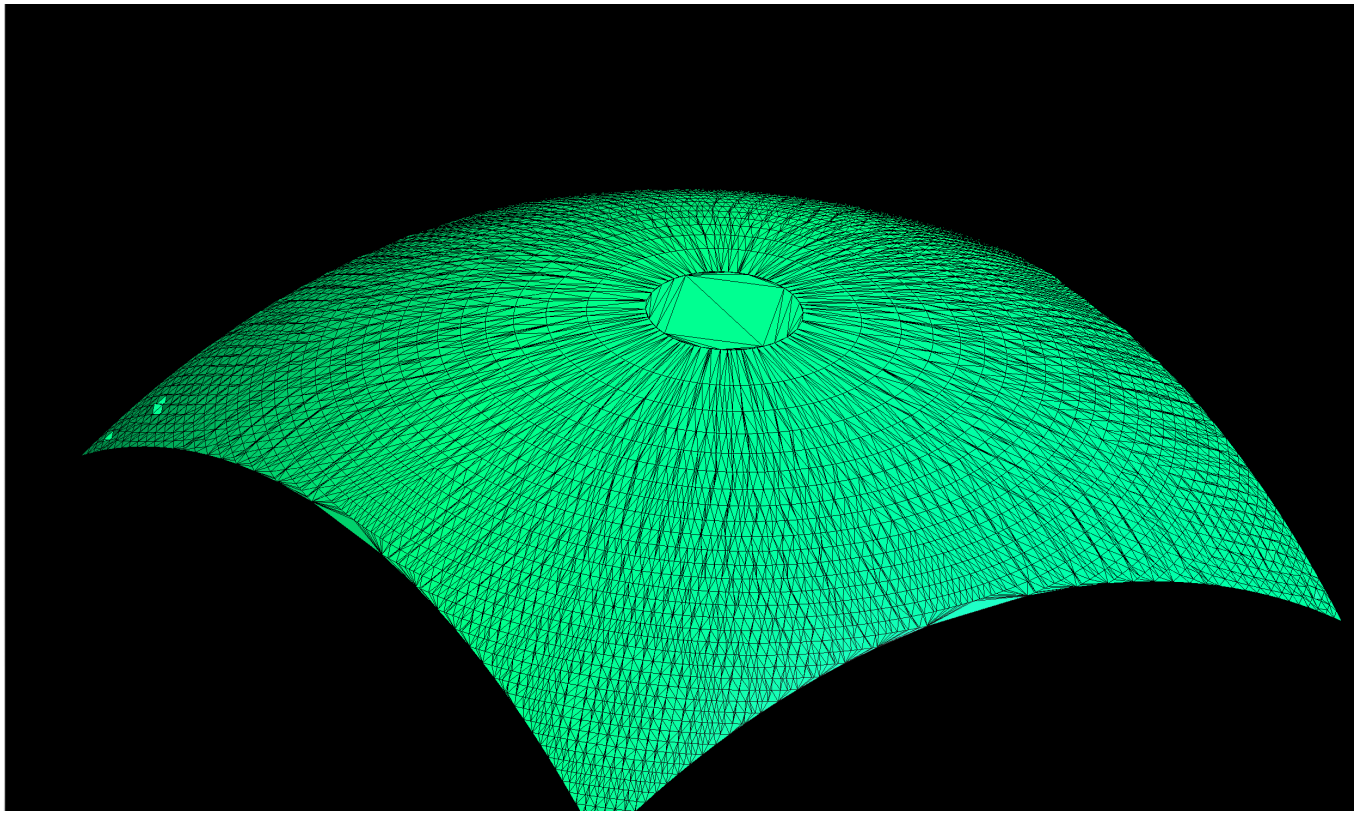

Figure: 3D mesh after re-triangulation (lambda function 1) side view

Results

$$z(x, y) = sin(x) + cos(y)$$

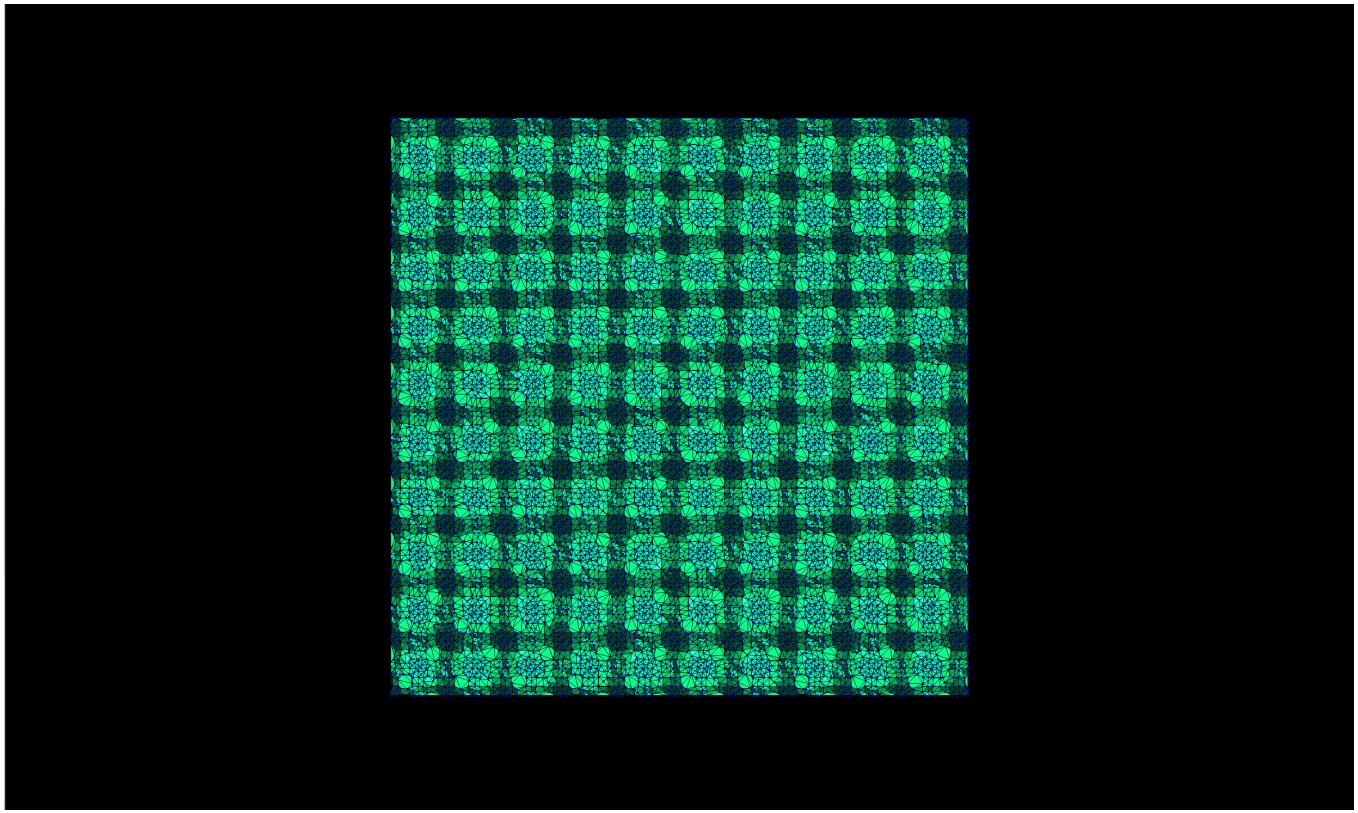

Figure: 3D mesh after re-triangulation (lambda function 2)

Results

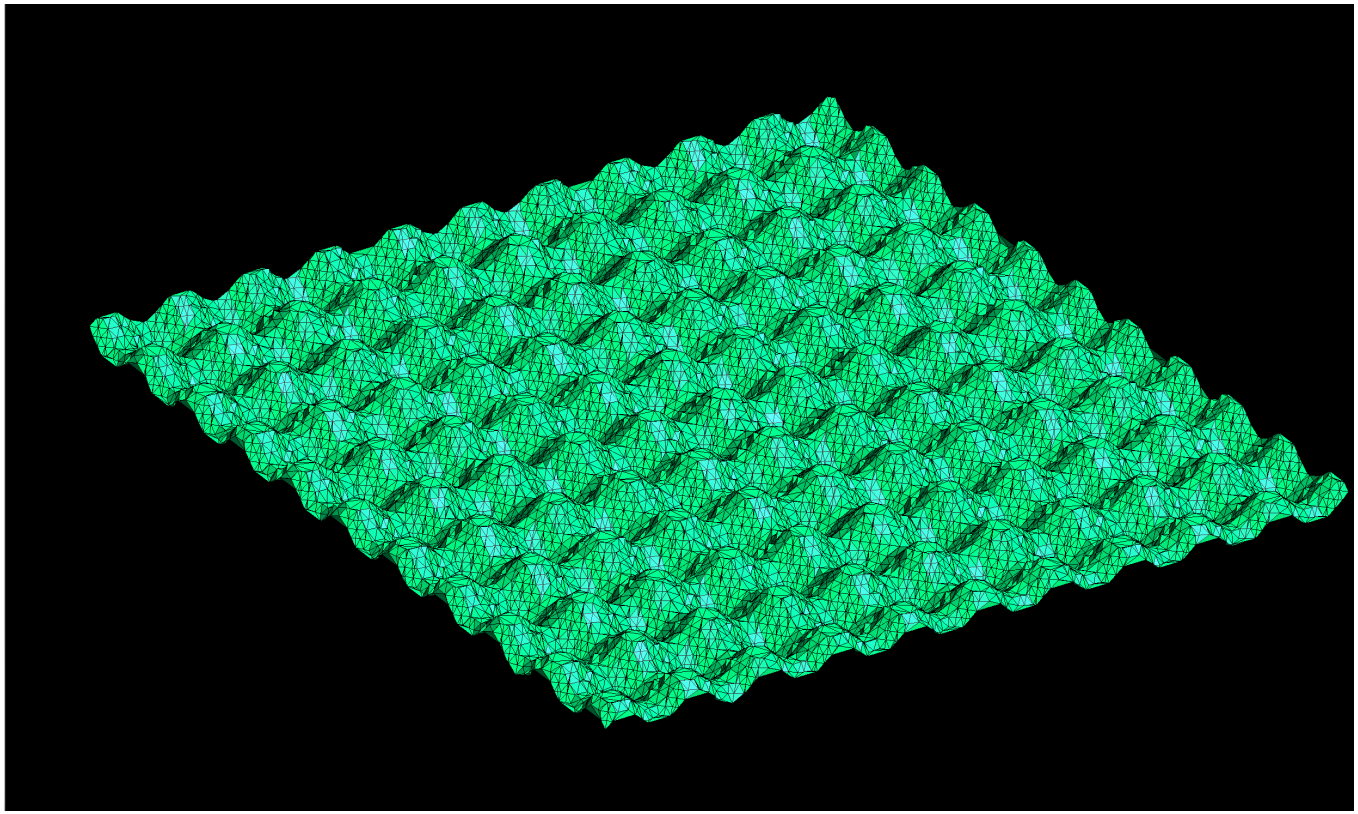

Figure: 3D mesh after re-triangulation (lambda function 2) side view

Results

$(x, y) = (33809, 23527)$ at zoom level 16

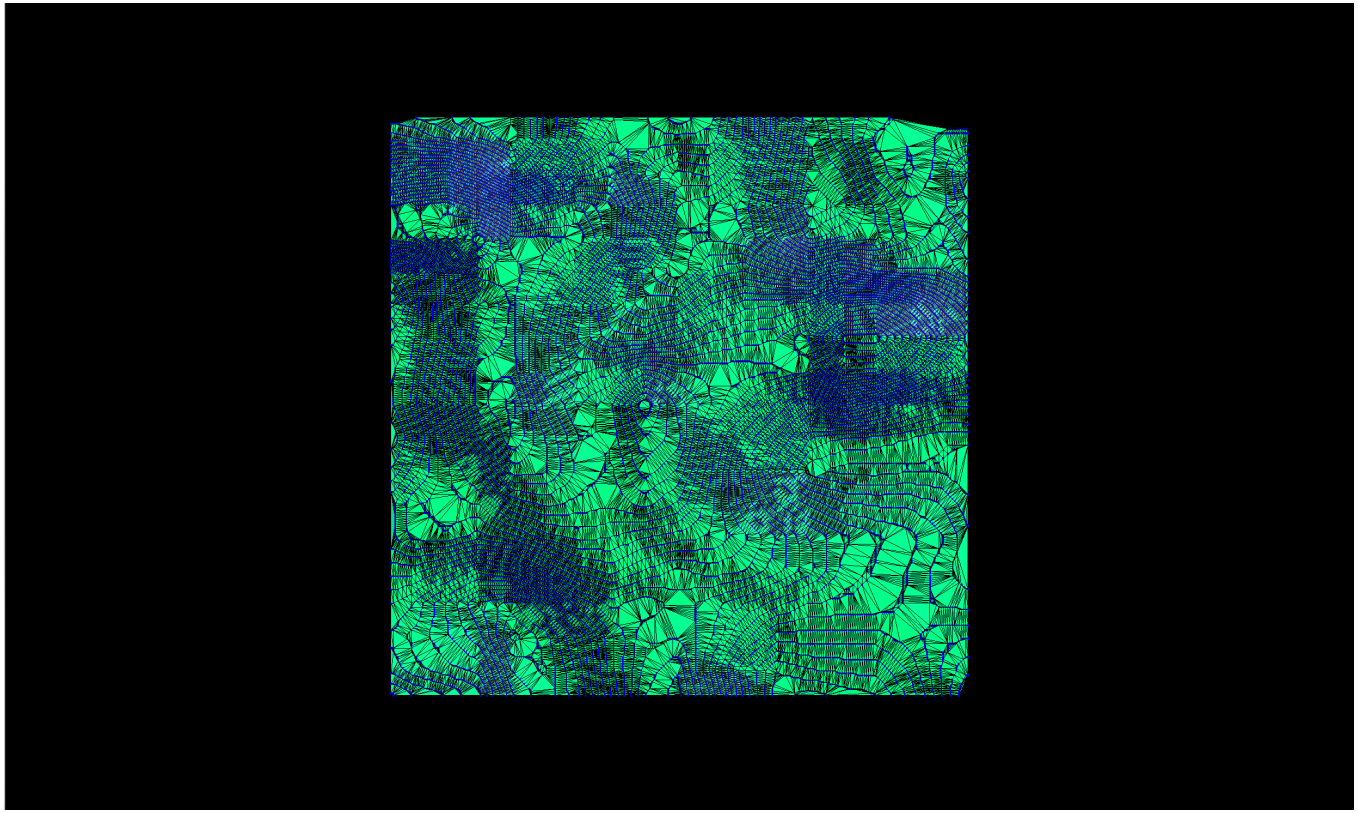

Figure: 3D mesh after re-triangulation (GPS data zoom 16)

Results

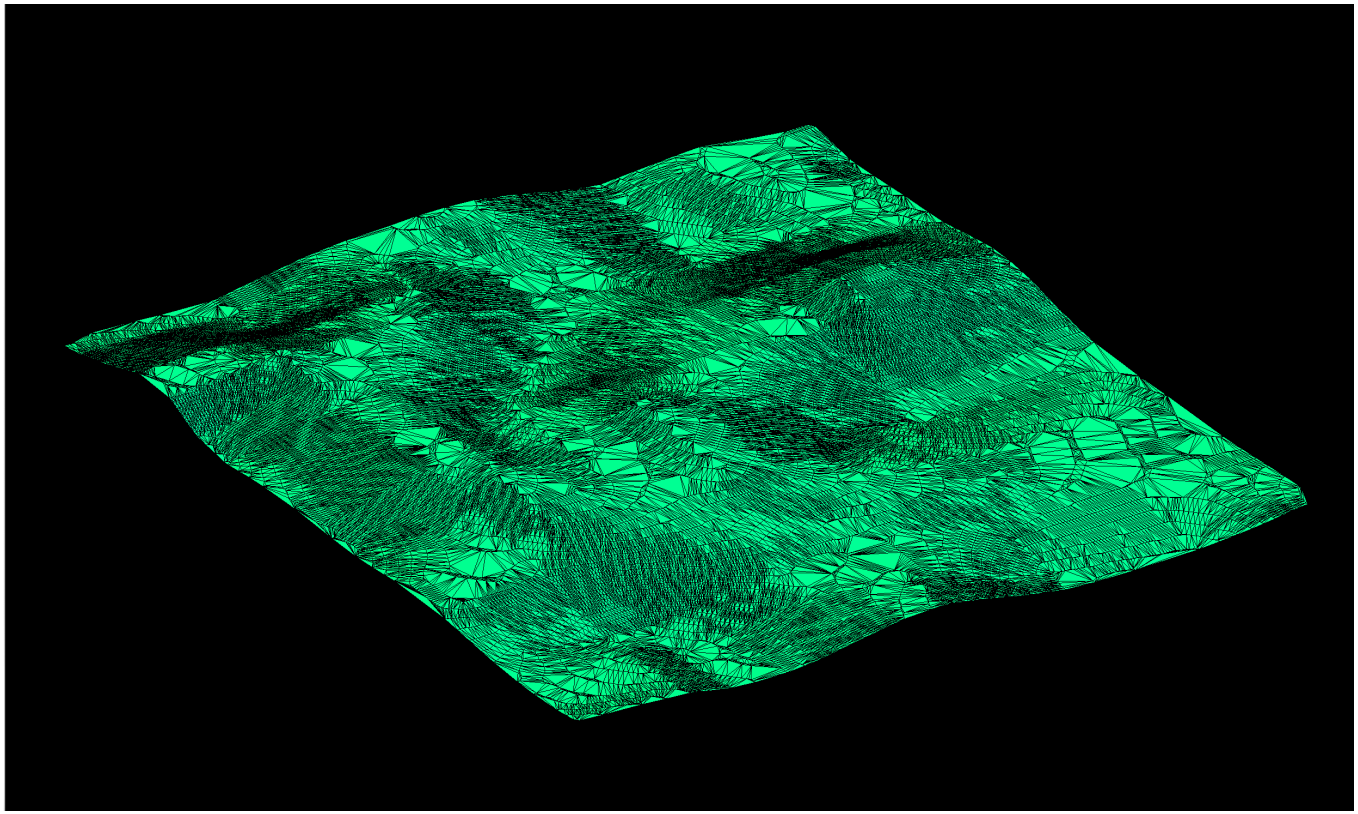

Figure: 3D mesh after re-triangulation (GPS data zoom 16) side view

Results

$(x, y) = (16904, 11763)$ at zoom level 15

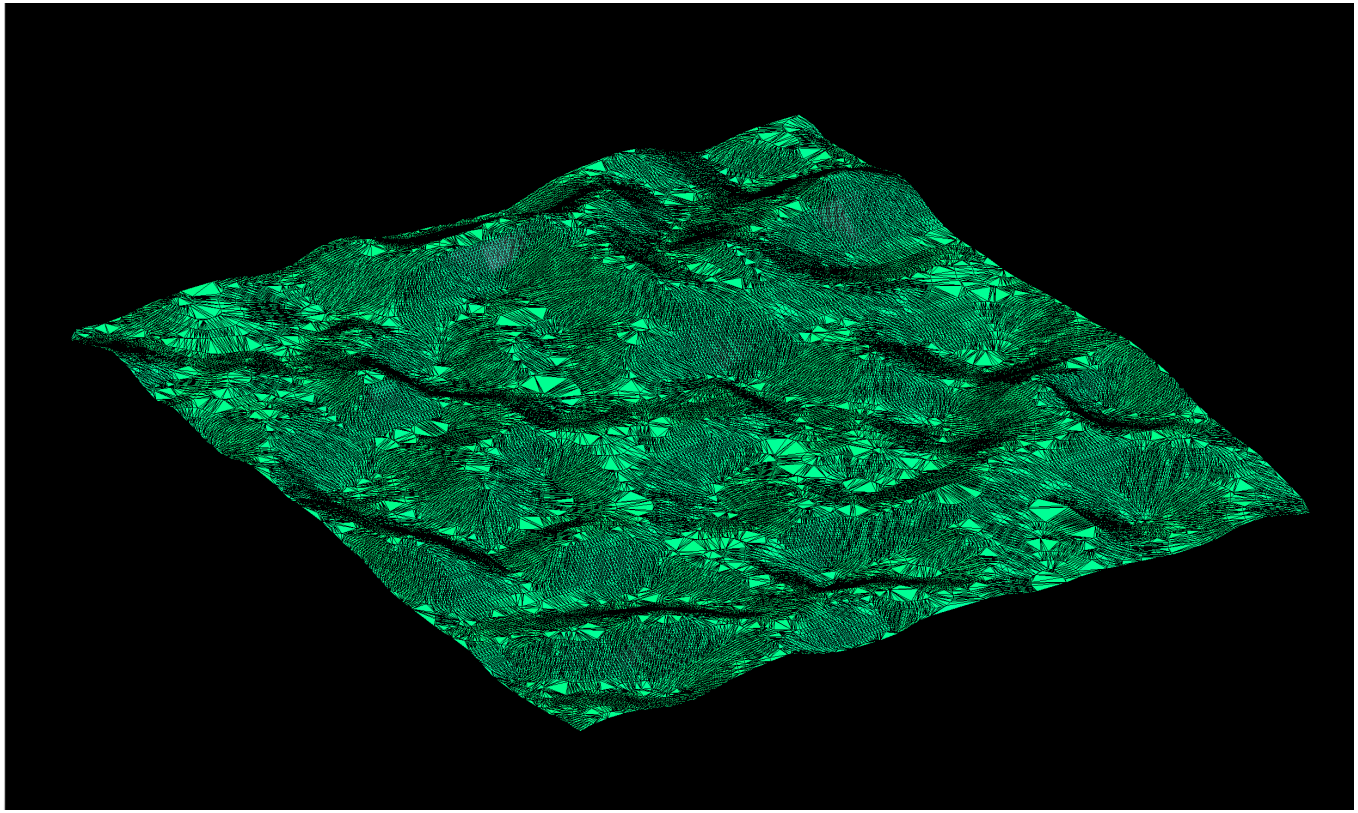

Figure: 3D mesh after re-triangulation (GPS data zoom 15)

Results

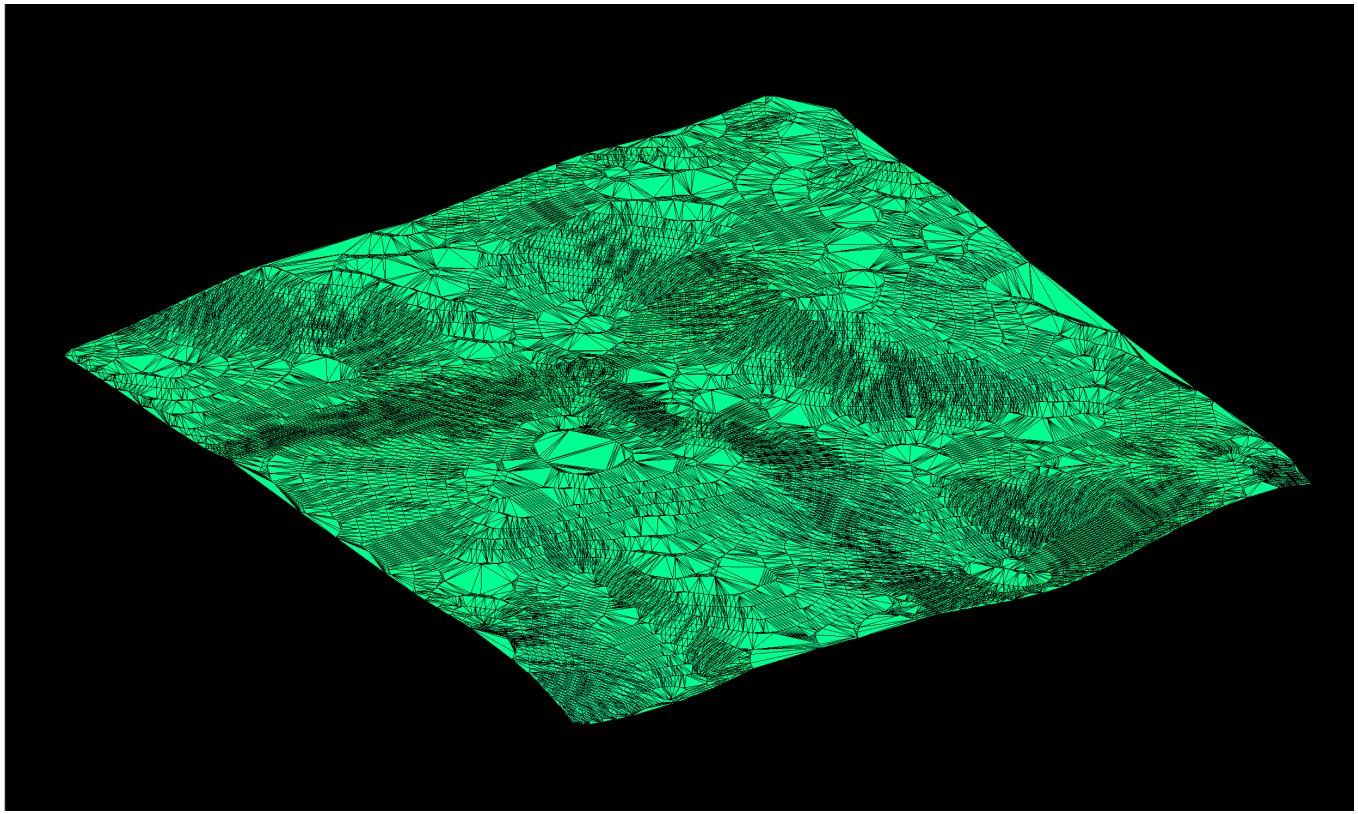

Figure: 3D mesh after re-triangulation (GPS data Strasbourg zoom 16)

Results

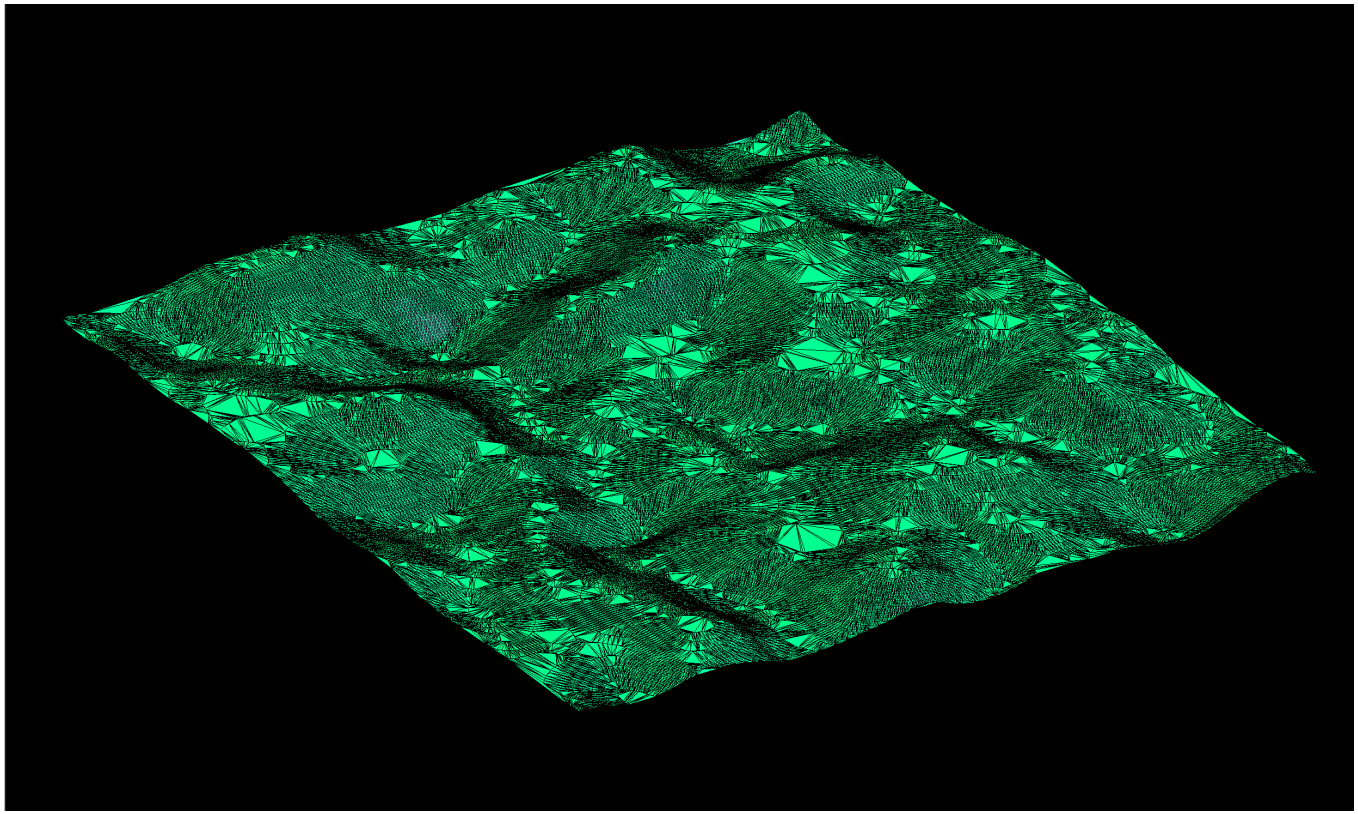

Figure: 3D mesh after re-triangulation (GPS data Strasbourg zoom 15)

Prospects: Handling and merging multiple tiles

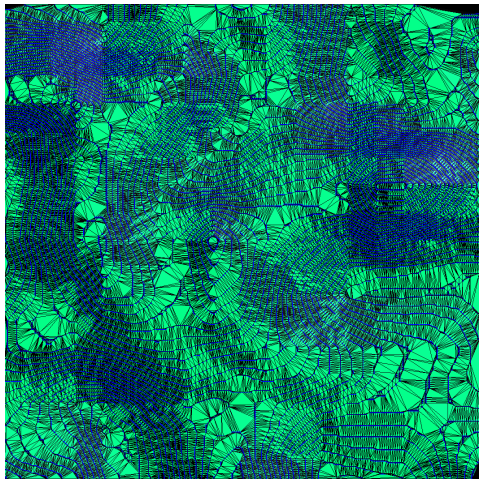

Figure: (33809, 23527)

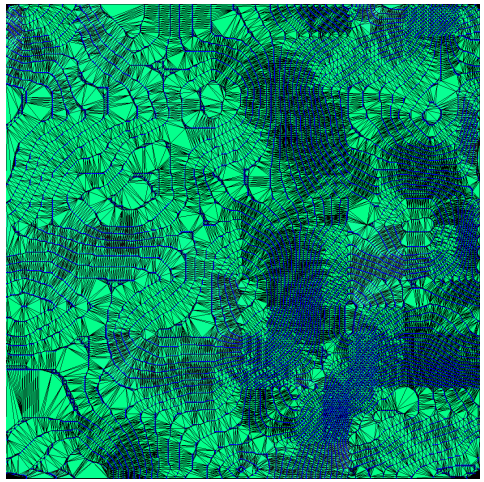

Figure: (33810, 23527)

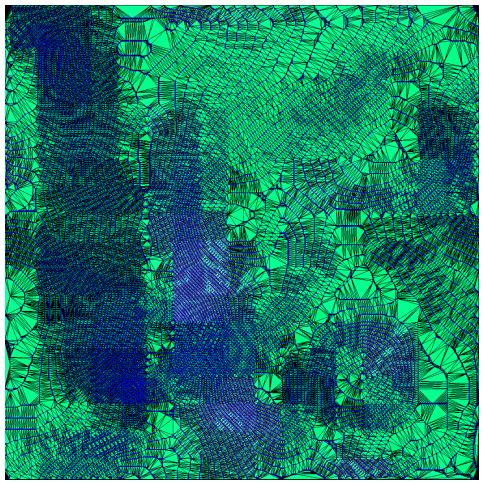

Figure: (33809, 23528)

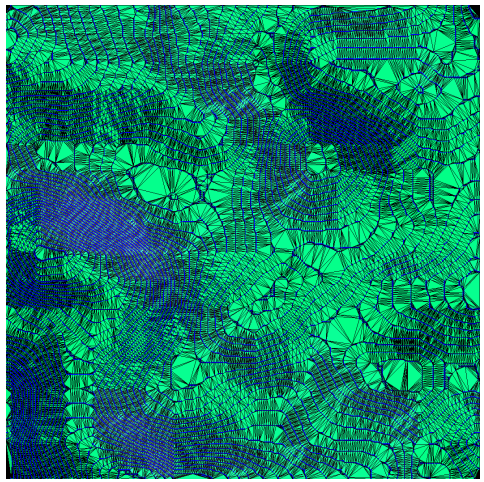

Figure: (33810, 23528)

Prospects: Handling and merging multiple tiles

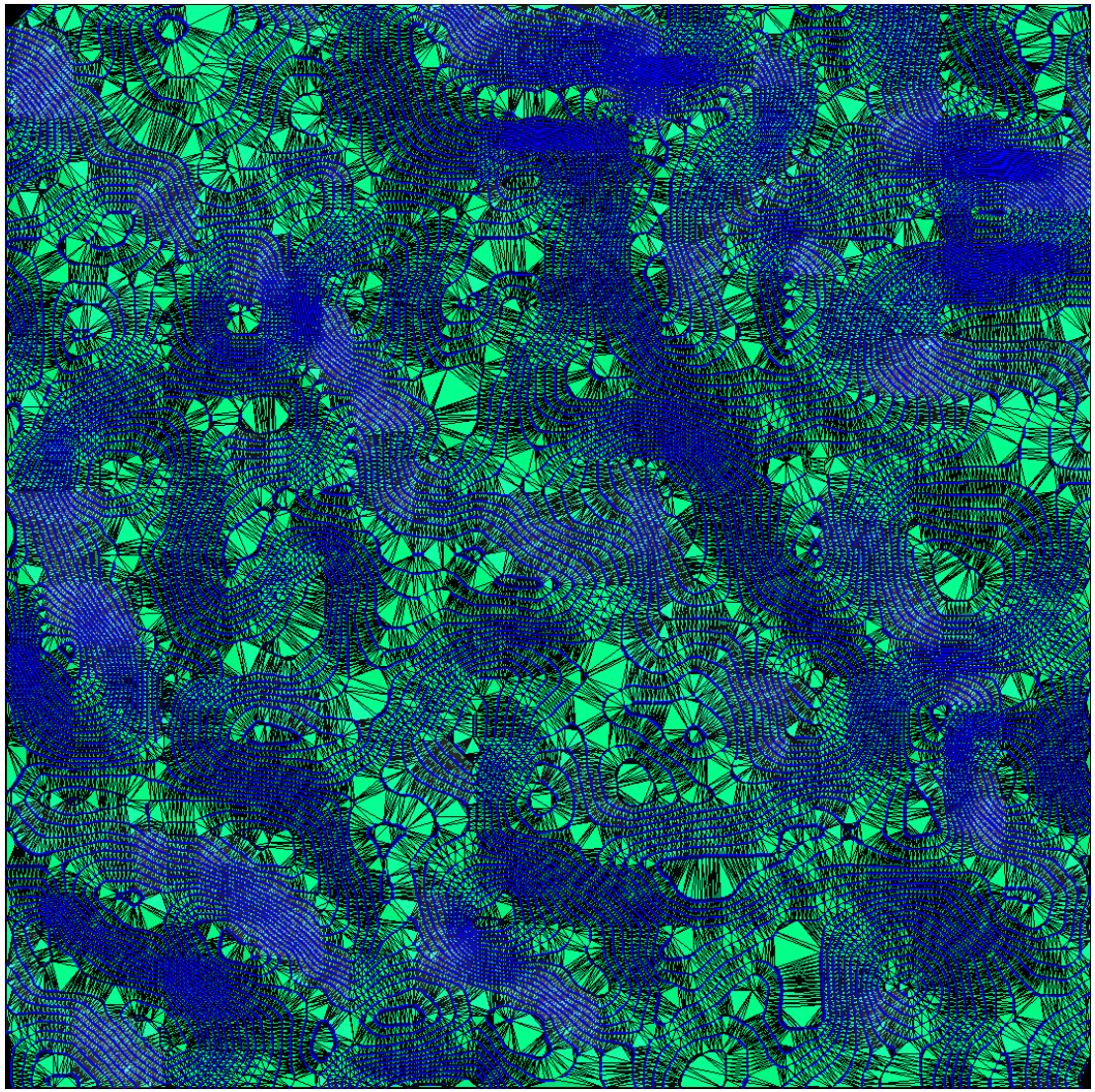

Figure: Resultig merged tile

Prospects: Mesh refinement

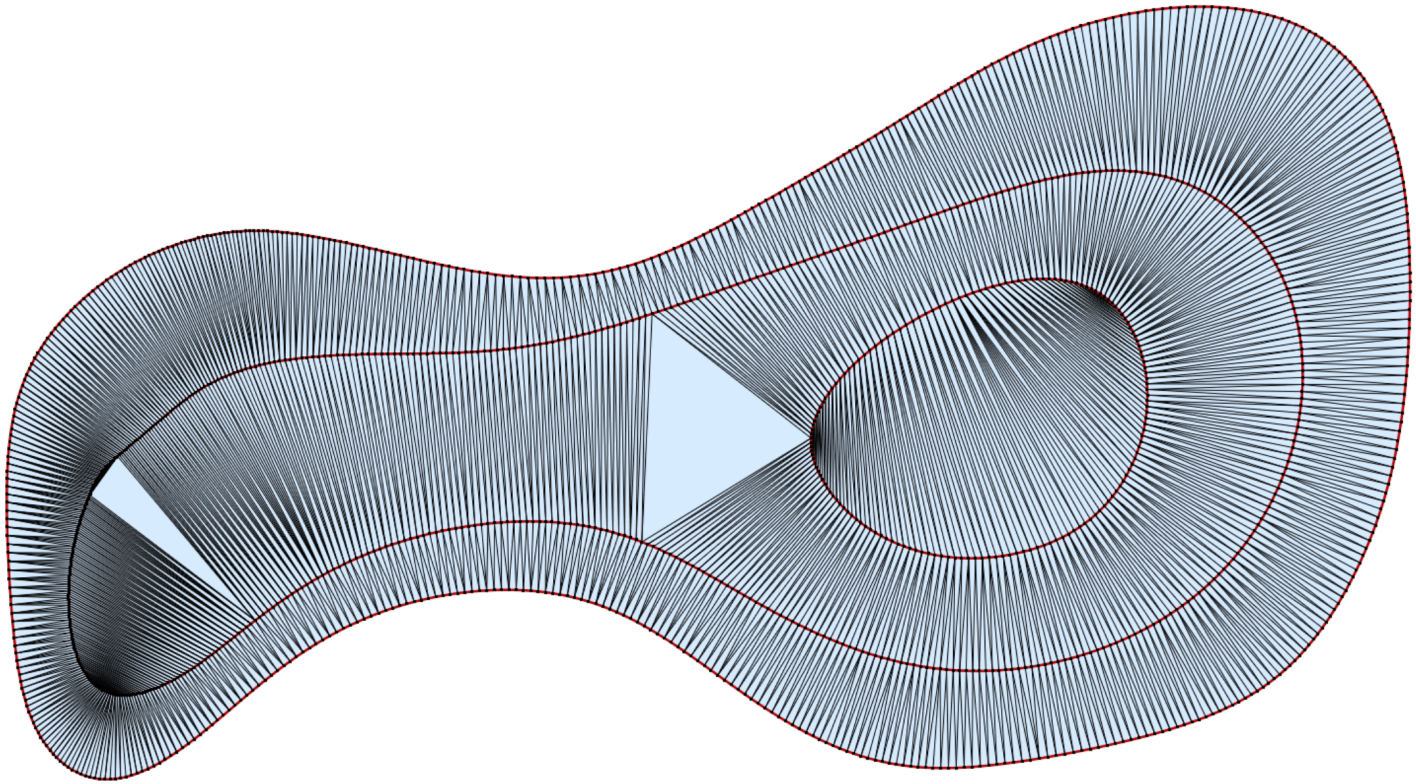

Figure: Mesh constrained to contour lines

Prospects: Mesh refinement

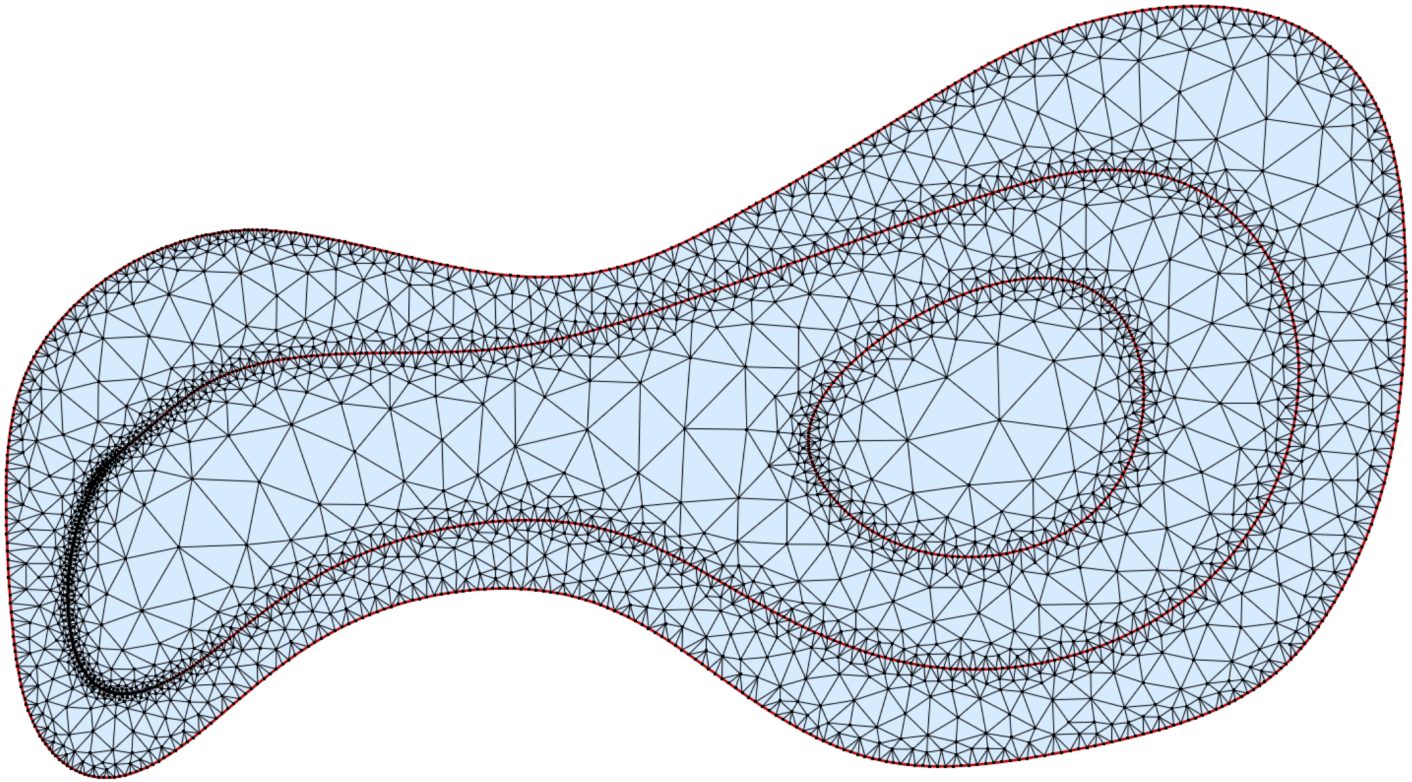

Figure: Mesh refined

Prospects: Parallelization

- **Parallel processing** of **multiple tiles**
- **Parallelizing mesh generation**
- **Parallel tile merging**

Prospects: Urban elements integration

- **Integrating urban** elements with **terrain models**

- **Detailed 3D model** of the **urban environment**

Conclusion

Flexible **terrain** generation:

- **LambdaGenerator**
- **GpsGenerator**

Conclusion

Contour lines generation:

- **ContourConstraint**

Conclusion

Triangulation:

- **triangulateAssembledMesh**

Conclusion

Framework developed provides foundation:

- **Future developments** and **applications**
- **Sustainable urban development** and **environmental conservation**

The end

Thank you all for your attention!