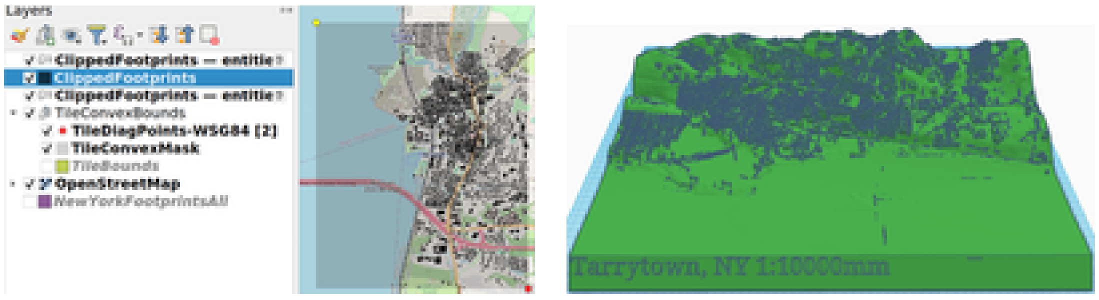
README

August 29, 2022

1 Efficiently construct a 3d printable topographic relief with municipal features

Jess's Tarrytown NY model notes



In this example, I am working entirely with free and *almost* exclusively open source software compiled for my linux distribution (the exception being Autodesk Meshmixer). Because this instance of geospatial modeling demonstrates such a common set of recurring operations preformed within the DLA Makerspace, I intend to build out much more thorough documentation and workshop curricula focused on this task using a variety of approaches and tools.

Software	Version
protobuf version	3.19.1
PROJ version	9.0.1
GDAL/OGR version	3.4.3
Qt version	5.12.8
GEOS version	3.10.2-CAPI-1.16.0
Python version	3.8.10
QGIS version	3.26.2-Buenos Aires
QGIS branch	Release 3.26
OpenSCAD version	2019.05
Inkscape	0.92.5 2060ec1f9f
Desktop Environment	Budgie GTK+
Operating System	Ubuntu 20.04.5 LTS
Windowing system	X11

Munging missing footprint data: Ilana, YuAng and I noticed a large chunk of Tarrytown's municipal footprint data is inexplicably missing or incorrect from common OSM CDNs and vector tile servers. This data is unfortunately missing from the Nextzen tile servers as well, so we'll use Microsoft's footprints classified using public orthoimagery.

First, lets grab a copy of these footprints:

```
wget https://usbuildingdata.blob.core.windows.net/usbuildings-v2/NewYork.geojson.zip
        -d ./assets/vector/
unzip assets/vector/NewYork.geojson.zip
```

We'll rely on tile data available via the Nextzen tile servers for roads and topography. Let's interact with these servers with using the nifty vectiler CLI.

```
# grab a copy of the vectiler interface:
git clone https://github.com/Jesssullivan/vectiler --depth=1
cd vectiler
git submodule update --init --recursive

# make sure toolchain depends are installed:
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install cmake libcurl4-openssl-dev # built in g++ should be fine
```

Now lets build vectiler:

```
cmake . -Bbuild
cmake --build build

# copy the executable from the build directory:
cp build/vectiler.out vectiler
chmod +x vectiler
```

Construct and execute a nextzen query for Tarrytown, NY:

```
# Builds 9659.12274.15.obj:
./vectiler --tilex 9659/9661 --tiley 12274/12278 --tilez 15
    --terrain 1 --roads 1
    --terrainExtrusionScale 1
    --pedestal 1 --pedestalHeight -20
```

Taking a look at Tarrytown at the resolution of the DEM data publicly available to us (level `15` using Google's map tile coordinates), the ROI is:

Upper left tile {x,y}: **Google:** `9659/12274` Upper left convex point: **WSG84:** `-73.88305400, 41.09591200`

Lower right tile: **Google:** `9661/12278` Lower right convex point: **WSG84:** `-73.85009500, 41.05450200`

We can take a look at these tile extents superimposed over Terrytown's OSM default map here.

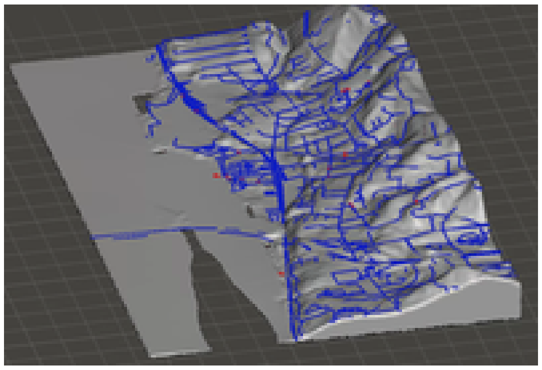
Using the WSG84 convex point coordinates above, we can now hop into QGIS. We add these two points and create a bounding box using the points as the diagonal extents as shapefiles. We can then use this bounding box both to limit the rendering extents (there are *a lot* of houses in New York!) and clip the footprints geojson file. These clipped footprints are can now be exported in place as a DXF.

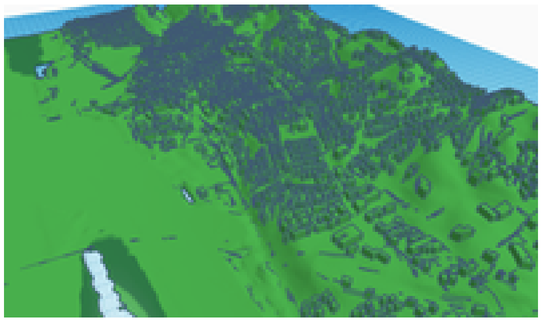
I then convert the resulting DXF to Inkscape SVG; this is not strictly necessary.

We can now extrude the clipped footprints using OpenSCAD:

```
# ./assets/vector/ExtrudeFootprints.scad
linear_extrude(height = 20)
import (file ="ClippedFootprints.svg");
```

We now just need to merge our footprint extrusions with the terrain and road data. To simplify this process, I first preform a plane cut, simplification remesh and solidify operations in autodesk Meshmixer and exaggerate the Z scale by a factor of 2. From there, we preform a union between the footprint extrusions and terrain mesh. I then create an inverted solid from the terrain mesh a few ~5mm above the original to clip roof heights.





Let's take a moment to consider our scale prior to the creation of a pedestal:

[4]:
```
## calculate model scale prior to the creation of a pedestal

# Spherical Mercator X google tile values @ 15
sm_XA = 8224624
sm_XB = 8220955

# Spherical Mercator Y google tile values @ 15
sm_YA = 5026499
sm_YB = 5022830

model_x_length_meters = sm_XA - sm_XB
model_y_length_meters = sm_YA - sm_YB
```

```python
assert model_x_length_meters == model_y_length_meters, "This is not a square, check your projection"

print("ROI extent is %smm wide and ~%smm long (using spherical mercator values!)" %
      (model_x_length_meters*1000, model_y_length_meters*1000))

# if we wanted this model to be printed no larger than 500mm on the short side:
model_x_length_mm = model_x_length_meters * 1000
print("\nThe printed model will be %s mm wide on the short side at 10000:1 in millimeters" % (model_x_length_mm / 10000).__str__())
```

```
ROI extent is 3669000mm wide and ~3669000mm long (using spherical mercator
values!)

The printed model will be 366.9 mm wide on the short side at 10000:1 in
millimeters
```

We can now add a pedestal and get ready to slice!