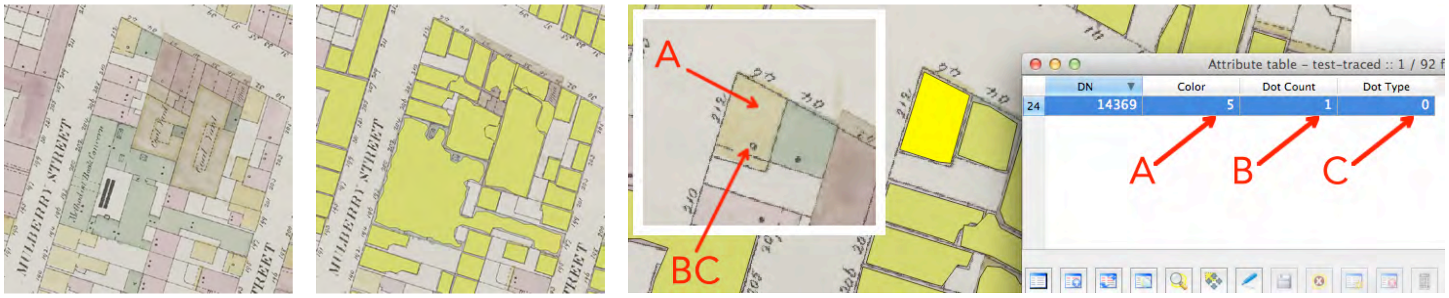
Historical map polygon and feature extractor

Mauricio Giraldo Arteaga
New York Public Library Labs
mauriciogiraldo@nypl.org



Attribute table – test-traced :: 1 / 92 f

	DN	Color	Dot Count	Dot Type
24	14369	5	1	0

Figure 1: *From left: original map input, vectorizer output shapefile superimposed on the original map and annotated individual polygon showing the additional attribute data being extracted.*

ABSTRACT

Polygon and attribute data extraction from historical maps such as US insurance atlases from the 19th and early 20th centuries has so far been a manual task. The New York Public Library (NYPL) currently relies on staff and volunteer work to manually extract polygons and other attribute data from its collection to create new public data sets for the study of urban history. This is a time-intensive task requiring up to several minutes to trace a shapefile and transcribe attributes for a single building. In this paper we propose an approach to automatically extract such attribute data from historical maps. The approach makes use of multiple image processing and statistics utilities to produce desirable results in a fraction of the time required to do by hand.

On average, a shapefile for an atlas sheet is generated in ~11.4 minutes for a total of 23.5 hours of processing time for a whole atlas that contains ~55,000 polygons; contrast this time frame to NYPL's current manual process that has taken three years to extract about 170,000 polygons across four New York City street atlases.

Even with some error rate in the proposed approach, the most cumbersome, time-intensive work (manual polygon drawing) has been reduced to a fraction of its original scope. This new workflow has promising implications for historical GIS.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – *Boundary representations.*

Keywords: map image tracing, attribute data extraction, alpha shapes.

1. INTRODUCTION

Vector polygon and attribute data extraction from raster images of historical maps (e.g. in GeoTIFF file format) such as US insurance atlases from the 19th and early 20th centuries has so far been a manual task. Although there exist tools that produce vectors from raster images (e.g. GDAL Polygonize [GDAL 2013]), these require either very clean and simple polygons or human intervention to assist the polygonization algorithm (e.g. clicking specific areas of an image, inputting numerous parameters, manual tracing of the image and other non-automatable processes). Historical maps such as those in the Lionel Pincus and Princess Firyal Map Division of the New York Public Library (NYPL) do not conform to the ideal raster image expected by those tools. The collection includes tens of thousands of sheets from 1853 to 1930 organized in 200 atlases and manual tracing is a time-intensive task requiring up to several minutes to extract data for a single building footprint.

NYPL has relied on volunteer and staff work to extract about 170,000 polygons (Figure 2) and their respective attribute data (e.g. street number, color, dot presence and type) during the past three years using a custom-made open source browser tool [NYPL 2011]. This effort covers only three of the hundreds of atlases in the collection. At this pace, NYPL will be unable to extract the bulk of the data in any reasonable amount of time.

One of the main roadblocks to automation is the nature of the images: most are very old maps that might be damaged or discolored in some way and each institution has different criteria for preserving and scanning them.

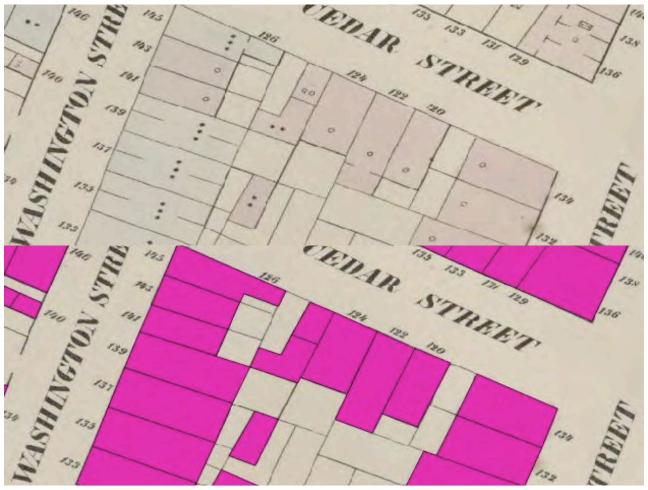


Figure 2: *Above: original map. Below: hand-traced polygons overlaid. Note the adjacency of polygons. This is the desired result.*

To address the above problems, we present a multi-step approach to map polygon and feature extraction (Figure 1) along with further explorations to be undertaken in the future. It is important to note that this is still work in progress. Any current and later explorations will be available for download and review in a public code repository [Map Vectorizer 2013].

2. RELATED WORK

Prior to pursuing this project we studied several existing alternatives. Our requirements include extracting building footprint as well as attribute data such as color, and dot or cross presence in the footprint.

The processes explored work by generating footprint polygons based on the pixel value of a given area of a raster image. This usually demands a previous thresholding step (Figure 3) since we are dealing with photos of hand-drawn maps and these contain paper texture, noise or just plain dirt where color areas are not homogeneous. The threshold process discards color values from the map and returns a two-value bitmap (black and white pixels). The expectation is that the polygonizing tool will then generate better polygons at the expense of color.

Work on building footprint extraction has been explored by Laycock et al [2011]. This includes a process similar to that described in 2.2 below with an extra polygon-simplification step. However, no actionable or open code could be found.

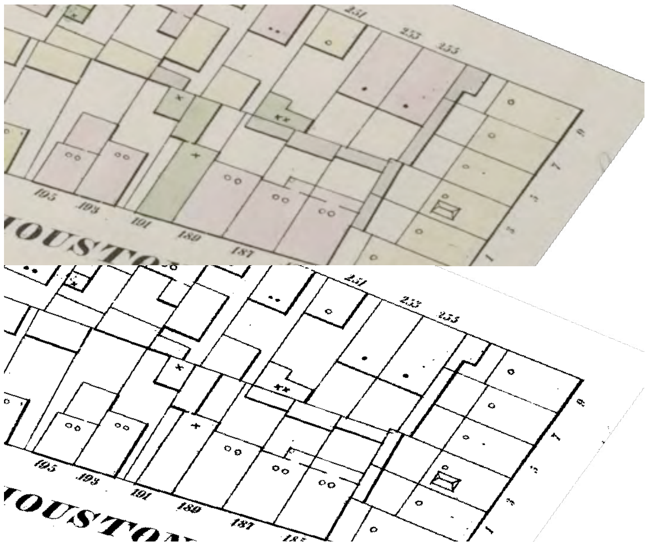


Figure 3: *Above, the original scanned, color-corrected and geo rectified map we are providing as input. Below, the threshold image produced by hand using the GIMP.*

2.1 ArcGIS

ArcGIS [ESRI 2012] is a commercial product that enables geospatial manipulation and analysis of data. Among these features are 'Convert Raster to Polygon' (CRP) and a 'Convex hull' geometric function. Because for most NYPL atlas sheets certain areas of the image end up yielding thousands of square "polygons" (and up to 1.3 million polygons in a single sheet), we first use the CRP function to produce a rough polygon layer and finally apply the convex hull function to simplify them (Figure 4).

This approach has two main drawbacks: on one hand it requires manual selection of non-building polygons in the rough shape layer in the first step and on the other hand the convex hull function does not satisfy the requirements of the desired final polygons to be produced (Figure 2). As of this writing we are unaware of a native function in ArcGIS that produces concave hulls and allows for automation for processing thousands of images.

2.2 GDAL Polygonize

GDAL [GDAL 2011] is a set of open-source utilities that can manipulate geographic raster and vector file formats such as GeoTIFFs and ESRI shapefiles. Among those utilities is Polygonize. This utility creates vector polygons for all connected regions of pixels in the raster image that share a pixel color value. An attribute indicating the color value is created for each polygon. [GDAL Polygonize, 2013]

This utility can be automated via the many wrappers available for various programming languages. The results using GDAL Polygonize are similar to those generated by ArcGIS's CRP but lacking a hull function.

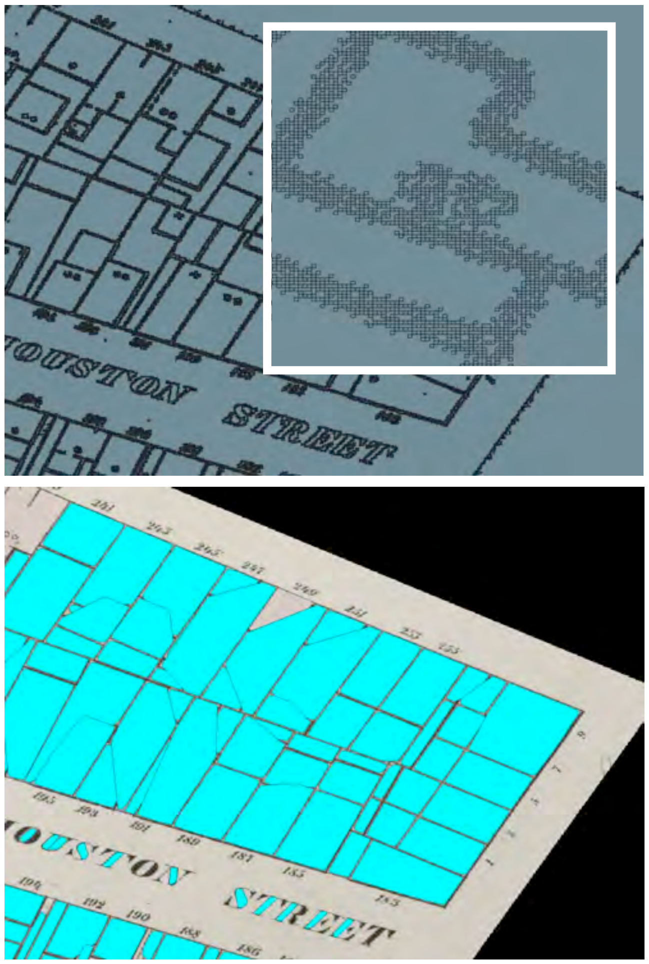


Figure 4: *Top: Detail of the output of ArcGIS 'Convert raster to polygon' function for the threshold file. Note the dozens of small polygons created, almost one for each pixel. Bottom: The resulting layer with convex hulls after manual cleanup. Note the problems that arise with using convex hulls such as losing L-shaped contours in many buildings.*

2.3 GRASS

GRASS [GRASS 2013] is a free and open source software suite that includes many features for geospatial analysis and manipulation and can also be automated using the Python programming language. It offers a large array of geometry and image processing functions including concave hull creation. Unfortunately, this concave hull function has just recently been added to the suite (Version 7, Spring 2013), still under development and considered unstable and experimental. We weren't able to get it to work properly for the needs of this project but are looking forward to any progress the GRASS development team makes.

3. PROCESS

It is worth noting that this process assumes a computer running the Mac OS X Lion operating system but should be applicable to any UNIX or Windows-based system. It also bears mention that, other than the operating system, this process involves only open source tools.

Below is the current process we are proposing to automate the extraction of polygons and attribute data from historical maps such as those available in the Lionel Pincus and Princess Firyal Map Division of the New York Public Library. These maps have a few consistent graphic qualities that allow for good automated polygon extraction:

- Dark lines describe building boundaries.
- The large majority of buildings are completely enclosed by these lines and do not contain interrupting elements (e.g. internal walls).
- All buildings are colored.
- Building attribute data is contained within the building boundary (e.g. use, material type).

These characteristics form the logic behind the automation process we present. Other types of maps may not be suitable.

In broad terms the automation consists of three steps:

1. Raster image thresholding
2. Rough polygon extraction
3. Polygon analysis and simplification

Several software utilities are used in each step to achieve the required goals.

3.1 Requirements

The following open source software needs to be present and functioning properly in the computer that will be used for polygon extraction:

- Python [Python 2013] version 2.7.1 or later with OpenCV [OpenCV 2013] version 2.4.6 or later. Python is a general scripting language available for many operating systems. OpenCV is an open source computer vision and machine learning software library.
- ImageMagick [ImageMagick 2013] version 6.8.6 or later. This is a software suite to create, edit, compose or convert bitmap images in multiple formats.
- R [R 2013] version 3 or later with the rgdal, alphahull, igraph and shapefiles libraries. R is a software environment for statistical computing and graphics.
- GIMP [GIMP 2013] version 2.8 or later. It is a software suite for photo retouching, image composition and image authoring.
- GDAL Tools [GDAL 2013] version 1.9 or later. This is a translator library for raster geospatial data formats.

It is also a recommended to install QGIS [QGIS 2012] to test and verify the results of this process.

The resulting process is a main Python script file that invokes all necessary utilities. Some code snippets will be presented below.

3.2 Calibration

This is a manual step that only needs to be applied once for each uniform set of map sheets. In the case of NYPL, all of the sheets in a single atlas are similarly colored and scanned, therefore color operations that work in one sheet will generalize to the whole set of sheets. This has been the case in our tests so far.

Calibration entails enumerating the distinctly unique colors present in the images that need to be extracted. Figure 4 shows the range of colors in the map collection we sampled. For each color that needs to be included in the attribute data (and its container treated as a polygon) a red, green, blue (RGB) value must be listed. The RGB value of the paper color also must be included. If colors have been applied inconsistently on the sheet or the sheet has just degraded over time (e.g. slightly darker or lighter hues) additional colors can be added to the enumeration. In Figure 5 there are two shades of red and two shades of blue. This enumeration can be done using the standard color picker present in image processing software. Multi-pixel color averaging is recommended for better results.

So far, this list is input directly in the main Python script file as follows:

```
basecolors = [
        [206,202,185] # paper
        ,[179,155,157] # dark red
        ,[149,156,141] # green
        ,[195,189,154] # yellow
        ,[187,194,192] # light blue
        ,[161,175,190] # "navy" blue
]
```

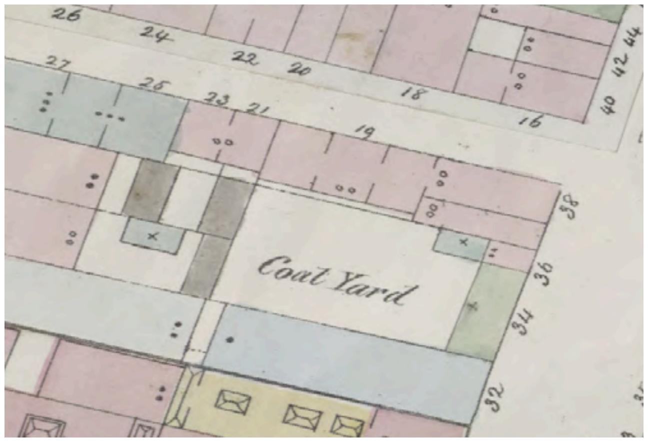


Figure 5: *Example of the color palette present in one map. The color in these maps can represent attribute data such as building material or use. Note the slightly darker red at the bottom compared to the top part. This will require two separate colors in the calibration process so as to extract as many polygons as possible. In a later stage these values could be consolidated in a database system.*

This list is overridden with the data in `vectorize_config.txt` if present.

A future improvement will be to create a simple tool that helps a user specify RGB color values by clicking in a base map, avoiding the hassle of inputting these values in code.

3.3 Thresholding

Similar to calibration, this is a manual step that only needs to be applied once for each uniform set of map sheets. This step provides the parameters necessary for a proper black and white threshold image (Figure 3) that the script can use for polygon extraction. It involves two basic image manipulation steps available in GIMP (a free image manipulation program): color brightness-contrast, color threshold. Included in the code repository [Map Vectorizer 2013] is a GIMP script file with parameters for these functions that work in an NYPL atlas:

```
(gimp-brightness-contrast drawable -50 95)
(gimp-threshold drawable 145 255)
```

The first line decreases the brightness by 50 and increases the contrast by 95 (both in a scale of -127 to 127). The second line applies a threshold to the resulting contrasted image where pixels between the values of 145 and 255 (in a scale of 0 to 255) are replaced with white, and all other pixels with black.

Another future development could be to eliminate the dependency on the GIMP program and only use the ImageMagick or similar utility. A simple interface could be created to test different contrast/threshold parameters and avoid dealing with additional script files.

3.4 Coarse polygonizing

This step makes use of the polygon extraction capabilities in GDAL Polygonize to produce a very rough shapefile (as described in 2.2 above). Since the output of this file can grow to millions of polygons for a single sheet, the shapefile is "split" into multiple smaller shapefiles, each with up to 50,000 polygons, in a process similar to that described by Garrard [Garrard 2009]. This process is later used to assemble the final shapefile.

3.5 Shape simplification

Each polygon produced in the previous step undergoes geometry analysis and modification using R. We need to exclude polygons that are either too small or too big to be buildings. Arbitrary area thresholds were set to 20 square meters in the low end and 3,000 square meters in the high end. These numbers can be modified to include or exclude more polygons but this might end up including extraneous polygons such as the letters forms of street names.

The polygons produced by GDAL Polygonize can have thousands of edges and may contain holes. The desired resulting polygon should resemble what a person would produce manually using traditional vector tools. As can be seen in Figure 6, the results so far are quite promising.

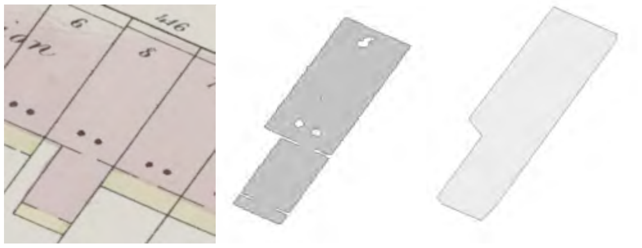


Figure 6: *From the input image a rough polygon is generated via GDAL Polygonize which is then simplified and closed using R.*

In order to simplify the rough polygon we generate a set of points inside it with a certain criteria (e.g. random, regular grid, hexagonal grid, etc) and then produce a surrounding shape, or *alpha shape*. Using the alphahull, shapefiles, rgdal and igraph packages in R we were able to produce a simplified polygon shape with satisfactory results for a large percentage of the buildings in a sheet. Proper false-positive and false-negative measurements are in process.

The steps taken to produce this simplified polygon are:

1. Choose a sample set of 1,000 points inside the rough polygon with one of these four types: hexagonal, regular, nonaligned, stratified.
2. Produce an alpha shape for these points (alpha=2).
3. If step 2 is successful, continue to step 4. Return to 1 otherwise and choose a different type.
4. Produce a circular graph from the alpha shape.
5. If step 4 is successful, continue to step 6. Return to 1 otherwise (or abandon attempting to simplify this polygon if no types are left and log this error).
6. Simplify the graph reducing the edge count.
7. Convert the graph into a shapefile.

The main challenges of shape simplification were:

- Finding the right parameters to produce an alpha-shape [Pateiro-López and Rodríguez-Casal 2011] that would work best with igraph (Figure 7).
- Modifying the resulting graph so as to produce a single circular graph.
- Producing a good enough simplified contour for concave polygons. Chamfers tend to appear in these buildings due to the alpha shape parameters (Figure 8). However, these chamfers do not detract from the generally acceptable result.

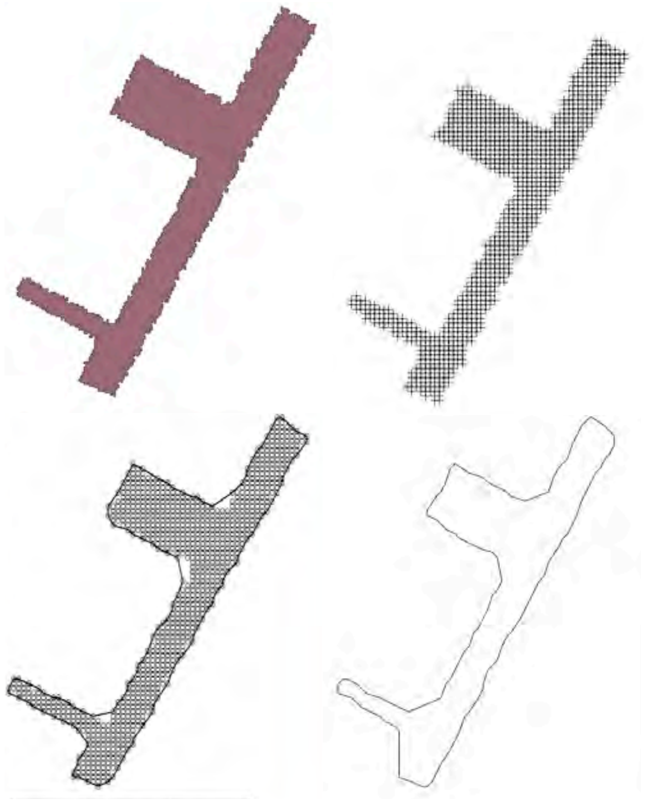


Figure 7: *From top left: rough concave polygon generated via GDAL Polygonize; regular point sampling; alpha shape for a regular point sampling (n=1000, alpha=0.75); final shape produced.*

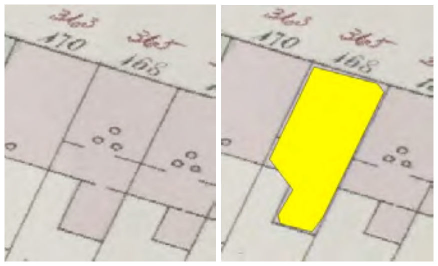


Figure 8: *The resulting chamfered polygon after alpha shaping and simplification. Note the diagonal "cuts" in the corners.*

3.6 Polygon exclusion and feature extraction

The previous step produces a list of simplified polygons that are likely to be buildings based only on their area. A building is also defined as a polygon whose color is not paper. The list of colors assembled in step 3.1 will serve as reference to measure the Euclidean distance between them and the average color inside the polygon.

3.6.1 *Polygon exclusion and color*

Firstly, we produce a new raster image by cropping the map to the boundaries of the polygon shapefile:

```
gdalwarp -t_srs EPSG:3785 -cutline polygon.shp -
crop_to_cutline -of GTiff map.tif polygon.tif
```

Next, a simplified averaging technique for calculating the average color is used. It is done by resizing the image down to a 1-pixel square using ImageMagick (Figure 9):

```
convert polygon.tif -resize 1x1 average.png
```

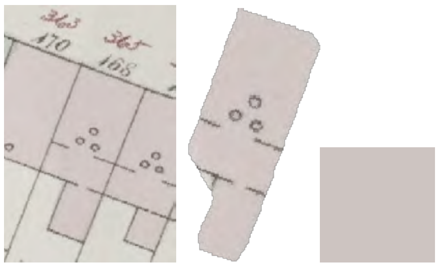


Figure 9: *The cropped image and the calculated average color pixel (red=194, green=183, blue=180).*

Finally the average color is compared to the base color list to determine the closest one. In case of paper being the nearest color, the polygon is ignored. Otherwise, the polygon is added to the list along with its nearest color.

Alternative ways of determining paper similarity could be explored.

3.6.2 *Dots*

Many buildings contain additional attribute data represented by circles, dots or crosses. Initial steps have been taken to determine the presence of this attribute data in polygons but the process still needs improvement. OpenCV includes many image-processing algorithms that allow for shape recognition in raster images. One of them is HoughCircles which finds circles in a gray scale image using the Hough transform. The original polygon cropped image is first converted to gray scale to then be searched for circles (Figure 10):

```
im=cv2.imread("polygon.tif")
gray=cv2.cvtColor(im,cv.CV_RGB2GRAY)
circles = cv2.HoughCircles(gray,
cv.CV_HOUGH_GRADIENT, 1, 2, np.array([]), 200, 8,
4, 8)
```

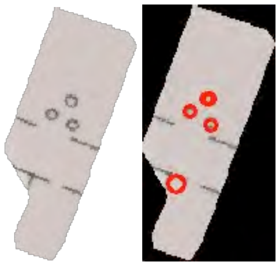


Figure 10: *The result of running HoughCircles in the cropped image. Note the false positive in the bottom part.*

Further work must be done in this area to extract attribute data more accurately.

3.7 Final shapefile assembly

A final shapefile is created with each individual polygon with its attribute data added as shapefile features.

4. RESULTS AND CONCLUSION

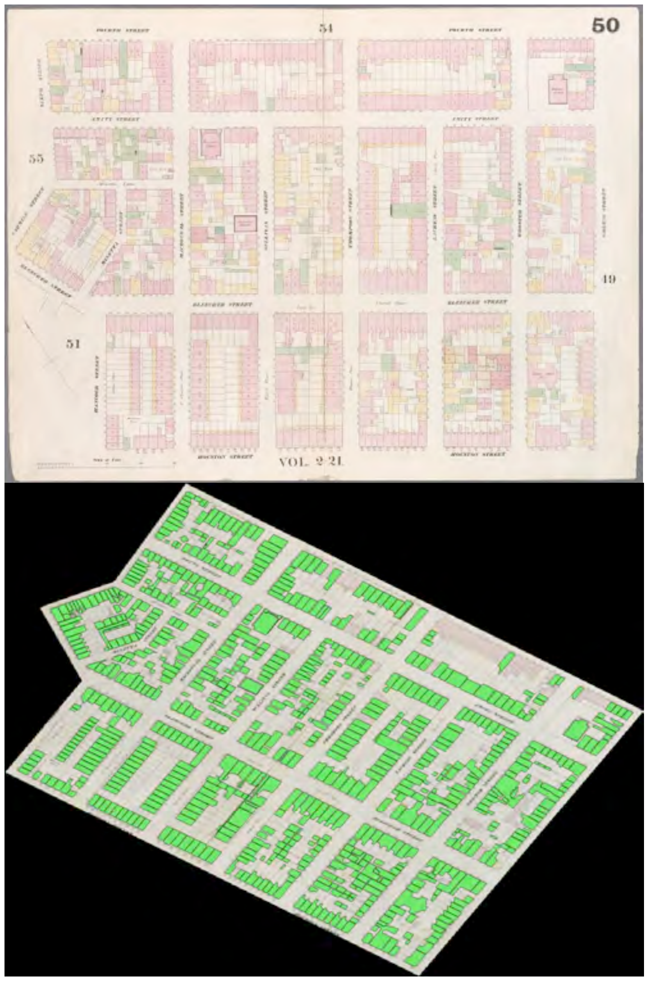


Figure 11: *The original scanned sheet (above) with the resulting shapefile (green) overlaid on the rectified GeoTIFF. Note some false negatives in the upper mid section.*

We have presented a process to automate polygon and attribute data extraction from historical atlases. We have tested this process in a Macintosh with an Intel Core i7 2.3 GHz CPU and 8GB RAM running OS 10.7.5 with an atlas that includes 123 sheets. On average, a shapefile for an atlas sheet is generated in ~11.4 minutes for a total of 23.5 hours of processing time for the whole atlas. Figure 11 displays one such shapefile overlaid on the original map. In total, 56,464 polygons were detected as possible buildings. Further evaluation will be made to determine the error rate. Also, additional work will be done in feature extraction (e.g. circles, dots, crosses) and other sources of attribute data.

A drawback of this process is that it does not guarantee that footprints will not overlap (e.g. when a smaller but large enough polygon is detected inside a larger footprint, both are kept). An additional step could be added to prevent overlapping in a certain threshold while also enforcing building adjacency. The open source nature of this project [Map Vectorizer 2013] enables improvements by anyone interested in this subject.

So far 170,000 polygons have been extracted from four New York City insurance atlases over three years using NYPL's current manual process (via staff and crowdsourcing efforts). Even with some error rate in the proposed approach, the most cumbersome work (manual polygon drawing) has been reduced to a fraction of its original scope. With this new workflow, NYPL's vast historical map collection can be vectorized in a matter of days, as opposed to decades. Subsequent, better targeted, crowdsourcing tools are being developed to validate this output and to capture features the automated process is not designed to extract such as addresses.

5. ACKNOWLEDGMENTS

We thank our reviewers for their valuable feedback and comments. We also thank Michael Resig for suggesting an initial workflow that led the way toward the automated process we describe here, Chris Garrard for his sample code to assemble and disassemble shapefiles [Garrard 2009] as well as Barry Rowlingson for his tutorial on converting alpha shapes to polygons [Rowlingson 2009].

6. REFERENCES

[1] GDAL, 2013. GDAL - Geospatial Data Abstraction Library: Version 1.9.2, Open Source Geospatial Foundation. http://gdal.osgeo.org

[2] Polygonizer, 2012. http://www.gis-support.pl

[3] Map Vectorizer, 2013. https://github.com/NYPL/map-vectorizer

[4] NYPL, 2011. http://www.nypl.org/blog/2012/06/13/nyc-historical-gis-project

[5] ESRI, 2012. http://www.esri.com/software/arcgis

[6] GRASS, 2013. http://grass.osgeo.org/

[7] Python, 2013. http://www.python.org/

[8] OpenCV, 2013. http://opencv.org/

[9] ImageMagick, 2013. http://www.imagemagick.org/script/download.php

[10] GIMP, 2013. http://www.gimp.org/

[11] QGIS, 2012. http://qgis.org/

[12] Garrard, 2009. http://cosmicproject.org/OGR/cris_example_write.html

[13] Rowlingson, 2009. http://rpubs.com/geospacedman/alphasimple

[14] Laycock, S. D., Brown, P. G., Laycock, R. G., & Day, A. M., 2011. Aligning archive maps and extracting footprints for analysis of historic urban environments. Computers & Graphics, 35(2), 242-249.

[15] Pateiro-Lopez, B., Rodriguez-Casal, A. 2011. Generalizing the Convex Hull of a Sample: The R Package alphahull. Universidad de Santiago de Compostela. http://cran.r-project.org/web/packages/alphahull/vignettes/alphahull.pdf