A Visual Framework for Digital Reconstruction of Topographic Maps
A. Thabet, N. Smith, R. Wittmann, J. Schneider

Technical Report
September 30, 2014

Computer, Electrical and Mathematical Sciences & Engineering
King Abdullah University of Science and Technology
23955 Thuwal, Kingdom of Saudi Arabia

Abstract: We present a framework for reconstructing Digital Elevation Maps (DEM) from scanned topographic maps. We first rectify the images to ensure that maps fit together without distortion. To segment iso-contours, we have developed a novel semi-automated method based on mean-shifts that requires only minimal user interaction. Contour labels are automatically read using an OCR module. To reconstruct the output DEM from scattered data, we generalize natural neighbor interpolation to handle the transfinite case (contours and points). To this end, we use parallel vector propagation to compute a discrete Voronoi diagram of the constraints, and a modified floodfill to compute virtual Voronoi tiles.

Our framework is able to handle tens of thousands of contours and points and can generate DEMs comprising more than 100 million samples. We provide quantitative comparison to commercial software and show the benefits of our approach. We furthermore show the robustness of our method on a massive set of old maps predating satellite acquisition. Compared to other methods, our framework is able to accurately and efficiently generate a final DEM despite inconsistencies, sparse or missing contours even for highly complex and cluttered maps. Therefore, this method has broad applicability for digitization and reconstruction of the world’s old topographic maps that are often the only record of past landscapes and cultural heritage before their destruction under modern development.

Keywords: Data acquisition and management, Geometry-based techniques, Geographical/geospatial visualization.
A Visual Framework for Digital Reconstruction of Topographic Maps

Ali Thabet, Neil Smith, Roland Wittmann, Jens Schneider

Abstract—We present a framework for reconstructing Digital Elevation Maps (DEM) from scanned topographic maps. We first rectify the images to ensure that maps fit together without distortion. To segment iso-contours, we have developed a novel semi-automated method based on mean-shifts that requires only minimal user interaction. Contour labels are automatically read using an OCR module. To reconstruct the output DEM from scattered data, we generalize natural neighbor interpolation to handle the transfinite case (contours and points). To this end, we use parallel vector propagation to compute a discrete Voronoi diagram of the constraints, and a modified floodfill to compute virtual Voronoi tiles. Our framework is able to handle tens of thousands of contours and points and can generate DEMs comprising more than 100 million samples. We provide quantitative comparison to commercial software and show the benefits of our approach. We furthermore show the robustness of our method on a massive set of old maps predating satellite acquisition. Compared to other methods, our framework is able to accurately and efficiently generate a final DEM despite inconsistencies, sparse or missing contours even for highly complex and cluttered maps. Therefore, this method has broad applicability for digitization and reconstruction of the world’s old topographic maps that are often the only record of past landscapes and cultural heritage before their destruction under modern development.

Index Terms—Data acquisition and management, Geometry-based techniques, Geographical/geospatial visualization.

1 INTRODUCTION

Fig. 1. Left: Working with physical maps is cumbersome due to their size and wealth of detail. Shown are two of 24 sheets covering Mecca, Saudi Arabia (dating to c. 1968). Middle: After scanning, contours and isolated points are extracted semi-automatically using our interactive framework. Right: From this representation, a digital elevation map is reconstructed using our method. Extracted contours and data points are overlaid in blue.

The necessity to convert historical maps into accurate digital representations has thus become a real concern, driven by the desire to preserve and document cultural heritage, and the current state-of-the-art commercial software is inadequately prepared.

The need to reconstruct DEMs from sparse sets of iso-contours has been recognized since the late 1970’s. There are two main challenges in this reconstruction process. Firstly, reliably extracting contours, especially in the presence of noise or complex topology, is a cumbersome process and existing commercial tools are rarely capable to scale to mid- or even large-scale data sets. This currently precludes the use of topographic maps as a data source for areas covering more than a few square kilometers. Assuming the contour extraction problem to be solved, the reconstruction can be formulated as a scattered data interpolation process. Here, the second challenge arises. A successful interpolation method has to deal with both contours and isolated data points. Hence, the interpolator is required to be transfinite. Furthermore, extremely inhomogeneous sampling rates of data constraints can be present in different areas of the map, requiring algorithms based on constrained Delaunay tessellations to be implemented with extreme care.

Contributions

In this paper, we present a full DEM reconstruction pipeline of large-scale historical topographic map collections using a semi-automatic, visually-oriented software framework. This visual framework integrates robust contour extraction and interpolation for complex topographic maps and was developed out of the lack of any current method or commercial software to digitize large sets of entire map collections accurately and efficiently.
Our contour extraction algorithm represents a novel contribution to the field of contour line recognition that is applicable to a variety of topographic maps that previously could not be processed without manual digitization. The reason is that the maps we are concerned with, unlike contemporary USGS maps, do not use colors to encode certain topological informations such as iso-contours vs. annotations and land usage symbols. Being purely grayscale turns the contour extraction problem into one of separating layers of information as well, and the literature on this particular problem is very sparse to say the least. Since the fully automatic separation of these layers seems futile (and this belief is supported by our experiments conducted with state-of-the-art commercial software), we instead opted for a semi-automated method in which the user extracts large batches of contours in one step and gets immediate visual feedback of the result.

We show that our interpolation algorithm performs at the same level of accuracy as competing methods but on considerably larger datasets, it also performs well in the presence of both locally dense and locally sparse contour information.

As a test data set, we consider a collection of 24 historical topographic maps of the city of Mecca in the Kingdom of Saudi Arabia. Since the early 1900’s, Mecca has undergone dramatic changes as the Saudi authorities seek to cope with the ever increasing influx of pilgrims to the holy sites to perform the two pilgrimages (Hajj and Omrah). Aside from creating additional infrastructure (roads, hotels, railway stations), these changes also include flattening mountains and hills to make room for the planned expansion of the Grand Mosque around the Ka’abah. In this light, the collection of historical topographic maps are a unique piece of testimony dating back to their creation between 1964 and 1971 upon request of the Saudi government. After scanning and rectification, we use our visual processing pipeline. The user can interact in each step to maintain a high degree of accuracy while maintaining minimal time consumption. Our visual framework is easily scalable and the interface allows crowd-sourcing techniques to be employed to increase output. We demonstrate how it can be easily distributed to a team of unexperienced volunteers to digitize an entire collection of maps in about half an hour of labor per volunteer.

In particular, we present:

- A novel semi-automatic contour vectorization module that requires only minimal user interaction.
- A parallelized, transfinite and discrete interpolation scheme based on natural neighbor interpolation.
- A reconstruction of a 16km × 9km area on a 1m × 1m grid, comprising most of the city of Mecca from a selection of 1:2,500 scale maps.

2 Related Work

Corresponding to the two main modules in our framework, we first discuss prior work related to the contour extraction problem before addressing prior work relevant to our reconstruction scheme.

2.1 Contour Extraction

Contour line recognition of multi-layered topographical maps remains an ongoing research problem. Initial work on digital contour extraction started in the 1980’s [2, 26]. Current research has focused on contour line recognition of complex topographical maps that contain overlapping layers of densely spaced contours, roads, buildings, geographical features, and labels. In addition, digital scans of these maps usually produce aliasing effects and fail to reproduce colors accurately. This in turn makes segmentation very challenging since algorithms are necessary that are robust to the aforementioned flaws in the digital scans.

Arrighi and Soille were the first to develop a systematic approach for the extraction of colored contour lines [2]. Colored maps, such as provided by the USGS, provide a significant advantage in that they allow contours to be separated from the background based on distinct colors. Building on this work, Khotanzad and Zink address the problem of aliasing and false color reproduction during scanning [26]. They propose the use of a color key set extracted automatically using a linear regression in RGB space. To vectorize contour lines, they propose a valley seeking algorithm that relies on false color elimination and the closing of large gaps.

Later methods of contour line extraction focus on colored topographical maps, with improvements or changes primarily made to the algorithms applied to the different steps originally set out by Arrighi and Soille [2]. For color segmentation, parameter-less models have been developed, such as automatic iterative thresholding of the color map within small regions [40], contrast-limited adaptive histogram equalization (CLAHE) [32], and selective Gaussian filtering to reduce noise during color segmentation [4]. After color segmentation, approaches proceed to apply thinning algorithms to the contours [40, 33, 39, 47, 36, 37]. The way single pixels are connected to form line contours differ from method to method: A geometric approach using a constrained Delaunay triangulation takes the “crust” of a line boundary between Voronoi points to represent each contour [39, 4]. Alternatively, a line tracing algorithm is used [8]. Other approaches involve calculation of gradient vectors to refine each contour’s centering and to close gaps [47, 36]. Poudreou and Spinello first calculate gradient orientation fields and then smooth contours with B-spline curves [35]. As a final step, all the aforementioned methods must close large gaps in the contours that arise due to the removal of other layers, symbols and, most commonly, labels. These gaps are partially resolved by methods similar to the aforementioned ones [2, 35].

Even though contour line recognition as presented in [2] has undergone many improvements, four significant drawbacks remain problematic to this approach. Firstly, this approach is highly dependent on the presence of color-coding of layers in the topographic maps. The treatment of gray-scale or even black-and-white maps is generally not addressed. While USGS-made color maps clearly demarcate contour lines from other layers, a full discussion on whether methods related to [2] are able to process maps not adhering to USGS standards is missing. Secondly, the fidelity of the digital scan is crucial. Even for maps scanned at high resolutions, sophisticated segmentation algorithms are required that are robust against aliasing and color aberrations. Thirdly, contour information is lost during segmentation due to interference with other layers. This results in gaps that have to be closed artificially—a problem that continues to be highlighted in the literature. Such gaps may also arise when labels that intersect contours and have a color similar to the contour are removed [36]. OCR is suggested as a possible remedy but has not been implemented by any of the papers. Finally, an issue that has received relatively little attention is that, once such problems have occurred, the only remedy is manual editing—a process that, depending on the severity of the problem, may take as long as or even longer than manual digitization.

In contrast, we abandon the notion of a fully automated contour extraction process altogether and propose a semi-automatic approach that robustly handles the majority of issues arising from the layer separation problem. It provides the user with simple selection and editing metaphors to aid the process and further provides immediate visual feedback. Our approach, in particular, does not depend on color maps and an OCR module automatically extracts contour labels.

2.2 DEM Reconstruction

Methods to reconstruct digital elevation maps can be roughly classified into purely geometric, tesselation-based and general scattered data interpolation methods.

**Tesselation-based methods** start with a variant of a constrained Delaunay triangulation and add additional constraints, such as the me-
dial axis / skeleton [16, 44, 31] or the crust [1], to achieve a practically feasible triangulation despite the extremely ill-conditioned triangles otherwise produced by naïve approaches.

A sub-class of methods that shows good results assumes closed and nested contours. In this case, neighboring contours are tessellated using a triangle strip. A first solution to branching contours was proposed by Christiansen and Sederberg [9]. Considering neither skeleton nor slope information, they achieve only $C^0$-continuity. Hormann et al. [25] improve this idea by estimating slopes on contours, which are then used in a $C^1$ Hemite-interpolation between contours along the line of steepest descent. Following the idea of interpolating along one-dimensional paths, Soille [43] uses linear interpolation along the geodesics between pixels and their closest contours. Similarly, Gousie [18, 19] interpolates intermediate contour curves. While the results of these methods are in general among the best in terms of reconstruction fidelity, the restriction to nested contours is a luxury that cannot be afforded in all cases. For instance, at map boundaries, nested contours are rarely available and thus boundaries require special treatment that is not discussed in the literature. Furthermore, the aforementioned challenge of closing contour gaps due to interference with other layers affects the applicability of these reconstruction methods severely.

**General scattered data interpolation methods** usually define a continuous interpolation which is then evaluated at discrete sampling points. Kriging [27], for instance, is a maximum likelihood estimator originating in geosciences. It is well established and close to parameter-free—the user only has to provide a prior in form of a semi-variogram model. Radial basis functions (RBFs) [6, 34] have also been used to reconstruct DEMs. While also being virtually parameter-free (except for the choice of a kernel), RBF fitting and evaluation requires sophisticated algorithms to overcome their computational cost [6], that still might prove prohibitive.

**PDE-based approaches** formulate a partial differential equation (PDE), in which boundary conditions (height and, optionally, slope) are defined along contours. After discretization, the PDE is then solved and yields a smooth reconstruction. Chai et al. [7] propose a PDE inspired by electric fields considering both height and slope information. In a different context, Hnaïdi et al. use the heat kernel to interpolate between features such as ridge and valley lines to author terrain [23]. In the context of free-form editing of vector graphics, Finch et al. use a biharmonic PDE that minimizes the thin-plate spline energy [14], and they also show terrain authoring applications. While PDE-based methods ensure global smoothness and treat branching naturally, they can be involved both numerically and computationally. Typically, multi-grid solvers [5] are used as a remedy, but their performance is not well-established for biharmonic PDEs, while resorting to well-established harmonic PDEs produces visually unpleasant and implausible artifacts.

In this paper, we use a variation of the natural neighbor interpolation [42]. This method uses a Voronoi diagram to define natural neighbors, of which a convex combination is used to result in the interpolant. Although this method can achieve $C^\infty$ continuity locally, at data points and on Voronoi circles the reconstruction is only $C^0$. Hiyoshi et al. [21, 22] develop methods to improve the global continuity by using derivative information at the data points (see also [3, 10]).

Natural neighbor interpolation is appealing to our application, since it is fast and can thus deal well with mid- to large-scale reconstructions. One problem is that contour curves either have to be sampled by discrete points or a generalized Voronoi diagram has to be computed. Since taking point samples along contours can result in highly anisotropic sampling densities and may affect stability, we opted for the second option of a generalized Voronoi diagram. The resulting method presented in the following can be discretized and parallelized easily. To alleviate potentially visible artifacts due to the use of a discrete generalized Voronoi diagram, we propose to use a biharmonic post-smoother.

### 3 Framework Overview

Our framework begins by scanning physical topographic maps. The map repository at our disposal consists of multiple semi-transparent grayscale sheets at scales ranging from 1,000:1 to 10,000:1. Each sheet measures 80cm × 60cm, excluding legend and print margins (also see Fig. 1). In this work, we use 24 scale 2,500:1 sheets. Each sheet covers an area of 2km × 1.5km. Together, the maps cover the central area of the city of Mecca. The maps were issued by the Kingdom of Saudi Arabia’s government in 1971 and were commissioned to a UK-based surveying company. Surveys were undertaken between 1964 and 1968, thus predating digital surveys. Sections of the maps are topologically extremely complex and hard to read even by humans (also see Fig. 2). As a side-note, traditional exploration of printed maps is also hindered by their physical extent as shown in Fig. 1.

These maps represent an important testimony to the past, as Mecca’s surroundings have undergone severe landscaping projects, altering the elevation profile irreversibly. It is therefore of importance that this rare set of maps is preserved for future generations, in part to serve for the purpose of cultural heritage as well as planning documentation.

**Input.** Each sheet is then scanned at 600dpi to a grayscale image...
comprising about 268M pixels, followed by a rectification process that aligns the map to cardinal image axes and also removes the legend. The set of rectified maps, plus meta-data (location of the map, scale, resolution, etc.) is the input for the following stages of our algorithm.

**Contour Extraction & Labeling.** To extract contours, we have developed an interactive user interface, described in detail in the next section. It allows the user to traverse the map tile by tile. In each tile, the user clicks on two points which are connected by a straight line. All contours intersecting this line are then automatically extracted and available labels are read using an OCR module. Here, we exploit the fact that the labels are laid out curvilinear to—and on top of—the contour. Clearly, in very complex areas unsupervised extraction is very unlikely to succeed in all cases, as our contour tracer may get diverted into layers depicting houses or ridge and valley lines. Therefore, our framework offers the user editing tools such as shortening contours or splitting one contour into two. Isolated data points are segmented manually, since the anchor to these points is too easily lost in clutter. Finally, we fit a cubic spline using a chordal parameterization to the discrete data points comprising each contour. Since this reflects the smoothness of human drawing motions, we never observe problems due to over- and undershoots.

With the help of volunteers in our lab, we extracted and labeled more than 20,000 contours and more than 1,000 data points within a few hours.

**Interpolation.** To reconstruct the DEM, we generalize the natural neighbor interpolation in Section 5 to the use of iso-contours. This requires the use of a discrete, generalized Euclidean distance transform (EDT) and a variation of the flood fill algorithm. The entire interpolation module can be easily parallelized, for which we used OpenMP.

We acknowledge that the discretization of the natural neighbor interpolation may result in visible and distracting quantization noise. To combat this effect, we first evaluate the EDT at a higher resolution than the reconstruction would mandate. Finally, we perform a post-smoothing based on a bi-harmonic PDE in unconstrained areas. This rapidly results in a smooth reconstruction, since the natural neighbor interpolation is sufficiently smooth in most regions. It serves therefore as an adequate first guess to the solution of the otherwise costly biharmonic PDE.

**Output.** The final output on our framework is either a DEM on a regular grid for further use in off-the-shelf GIS software or a triangle mesh using a restricted quadtree mesh [20], optionally including multiple resolutions. The latter one is routinely used in efficient terrain rendering engines [28, 29, 13] and, by offering this particular type of output, a costly meshing process can be entirely avoided prior to rendering.

4 CONTOUR EXTRACTION

Our contour extraction method provides an easy-to-use interface to extract, edit, and label iso-contours and isolated points requiring only minimal user interaction. Figure 3 shows our contour extraction package.

To start the extraction process, the user clicks on two points, defining a seed line that intersects several contours. At each line-contour intersection, a seed point is placed. Seed points are then used to initialize a trace in both directions, thereby fully extracting each contour. The two initial directions are obtained by a local analysis of the contour pixels vs. the background, and it is the user’s obligation to avoid overly cluttered regions for seed placement. In the presence of background structure such as houses, roads, etc. along the trace, unwanted points might be included in the contour. To clean up such contours, we provide the user with editing tools to shorten/reduce a contour, split a contour into two disjoint parts, merge two contours, and to delete a contour altogether.

Fig. 3. Contour extraction package. The interface has 6 interaction panels. 1: IO tools for loading and saving of maps and contours. 2: Extraction tools to add, reduce, split, merge, and delete contours. 3: Information about map. 4: Labeling tools to automatically and manually add elevation labels to contours. 5: Image split panel to process smaller portions of the map. 6: Portion of the map being processed.

Algorithm 1 Contour tracing algorithm

```
1: procedure TRACE_CONTOUR(seed_point)
2:   previous_point ← seed_point
3:   direction ← perpendicular to line direction
4:   direction ← NORMALIZE(direction)
5:   while (1) do
6:     CREATE_ELLIPSE(direction)
7:     in_points ← pixels satisfying condition 1
8:     valid_points ← pixels satisfying condition 2
9:     if valid_points is empty then
10:       break;
11:     end if
12:     next_points ← MEAN_SHIFT(valid_points)
13:     direction ← next_point − previous_point
14:     direction ← NORMALIZE(direction)
15:     previous_point ← next_point
16:   end while
17:   previous_point ← seed_point
18:   direction ← (perpendicular to line direction)
19:   direction ← NORMALIZE(direction)
20:   back to 5
21: end procedure
```

It is trivial to check for condition 2 since background pixels are all colored white. Once a set of valid pixels is defined, i.e., pixels covered by the ellipse that meet conditions 1 and 2, the next point on the contour is calculated using the mean of the valid points. This process is known as a mean shift. The last step before repeating the process is to update the trace direction as the normalized difference between the next point and the previous one (which is initialized as the seed point). Once we repeat the process, the updated direction, and therefore the long axis of the ellipse, reflects the direction of movement from the previous point to the current one. Thus, a smooth trace that does not allow for abrupt change in the shape of the contours is obtained. The
trace will stop once the algorithm cannot find any points that satisfy both condition 1 and condition 2.

Clearly, the shape of the ellipse should reflect the local behavior along the contour. The short axis should reflect the thickness of the contour, a measure that we estimate as follows. Whenever the decision is made to place a seed point, a circular, local neighborhood of pixels on the contour is considered. A principal component analysis of the contour pixels’ position then yields the two trace directions as the eigenvector corresponding to the largest eigenvalue. The eigenvector corresponding to the smallest eigenvector is sufficiently aligned with the cross section of the contour to measure the contour thickness as the amount of collinear pixels in that direction. Under the assumption that the seed line does not intersect the contour at an overly acute angle, an ellipsoid neighborhood may be used for the principal component neighborhood instead. Note that while contour thickness varies across the map, there is usually a set of a few, narrow thickness “bands” (e.g., one band for contours every 10m vs. one for contours every 2m). This information can be used to tune the size of the initial region for the entire set of maps, while the principal component analysis compensates for local variations between adjacent contours.

For the tracing ellipsoid, we chose to define the length of the larger axis to be a factor of two of the shorter one. This provides the algorithm with sufficient incentive to move along the contour with reasonable progress and without missing high frequency details of the contour. Once the trace in one direction is finished, the algorithm restarts tracing from the seed point in the opposite direction. Fig. 5 illustrates the first cycle of the tracing algorithm.

Using this method, few clicks on a given map suffice to extract all the iso-contour curves. To avoid multiple extraction of the same contour, our algorithm does not place seed points on the intersection between the seed line and contours that have been previously extracted. This particular feature removes restrictions on the user while placing seed lines and facilitates his/her interaction with the tracer.

A distinctive advantage of our algorithm is that the trace will always follow a smooth path. Abrupt changes in direction automatically result in termination of the trace, which is particularly useful when a contour passes through background structures such as houses. Fig. 6 shows an example of a contour that crosses the floorplan of a house (red). The traced contour (blue) yields the correct curve, despite passing through the building.
4.1 Contour Labeling

In order to create terrain data from the extracted contours, we need to obtain the elevations associated with each contour. A large number of contours have labels printed somewhere along their length. The first step of the elevation assignment for these contours is to localize their labels. This is done by creating label candidates from image portions that could contain label information. Since the elevation numbers are aligned with and placed atop the contour, we look at rectangular portions between contour end points that fit a size criterion that we obtained empirically from the map data. As shown in Fig. 7, this results in both valid and invalid candidates. We send these image portions to an Optical Character Recognition (OCR) algorithm and retain only those that provide meaningful numerical data. While any OCR module can be used for this task, we chose to use the freely available Tesseract OCR engine [17].

The vast majority of contour lines are separated by a fixed and known elevation difference. This difference, also known as contour-spacing, does not generally vary across a set of maps. In our case, the legend lists this value (2m for the 2,500:1 scale) but it can otherwise be obtained by a single glance at the map. Elevations of remaining contours can thus be obtained as follows. Since the seed lines defined by the user covers all available contours, we start tracing along each line until we find two labeled contours. For the majority of cases, these two labels determine whether height is increasing or decreasing along the seed line. In the rare case that the two labels are the same height value, we cannot infer intermediate heights, and the user is asked to assign labels manually. This case may occur when a seed line crosses either a ridge or a valley in the map.

In case that the two labels in fact define increasing or decreasing slope, all contours between this pair of labeled contours are updated accordingly. A basic “sanity check complements this operation: if contour labels are not plausible due to missing contours along the seed line, the user is alerted and prompted to update heights manually if needed. This feature requires knowledge of the contour-spacing value, as our algorithm infers the amount of contours to expect between labels. Note that the opposite case, in which more contours would be encountered, cannot happen with our method, since contours are always fully labeled one after the other. It is therefore irrelevant how often the seed line crosses the same contour.

For contours that are not enclosed between labeled pairs, we continue calculating height values along the seed line, but instead of labeling contours automatically, we suggest these values to the user. The reason is that contours might be crossing a ridge or a valley and, thus, the sign of the increment would change. Note that we also provide tools to click and manually change any of the contour labels.

5 Transfinite Interpolation

In this section, we first review the concepts behind natural neighbor interpolation. Then, we generalize this concept to cover the transfinite case and we discuss implementation details.

5.1 Natural Neighbor Interpolation Revisited

Natural neighbor interpolation reconstructs a continuous function $\Phi$ over a domain $\Omega$, given data locations

$$X := \{X_i\}_{i=1}^N \subseteq \Omega$$

and function values

$$\phi_i := \Phi(X_i), \ i \in \{1, \ldots, N\},$$

$\Phi$ is evaluated at any point $X_0 \in \Omega$ as a convex combination of the function values $\phi_i$:

$$\Phi(X_0) := \sum_{i=1}^N w_i(X_0) \phi_i \text{ with } 0 \leq w_i \leq 1 \text{ and } \sum_{i=1}^N w_i = 1.$$  \hspace{1cm} (3)

To compute weights $w_i(X_0)$, a Voronoi diagram is computed. Given a distance metric $d : \Omega^2 \to \mathbb{R}_0^+$, $\mathcal{V}(X) = \{B_i\}_{i=1}^N$ tessellates $\Omega$ into Voronoi tiles around Voronoi sites $X_i$:

$$B_i := \{x \in \Omega : d(x, X_i) \leq d(x, X_j) \ \forall j \neq i\}.$$  \hspace{1cm} (4)

A virtual tile $\mathcal{B}_0$ is then computed, but not added to $\mathcal{V}$, using a query point $X_0$ as a new site:

$$\mathcal{B}_0 = \{x \in \Omega : d(x, X_0) \leq d(x, X_i) \ \forall i \in \{1, \ldots, N\}\}.$$  \hspace{1cm} (5)

The actual weights are then obtained as the fractional area

$$w_i(X_0) = \frac{|\mathcal{B}_i \cap \mathcal{B}_0|}{|\mathcal{B}_0|},$$  \hspace{1cm} (6)

where $|\cdot|$ denotes the area of a compact subset of $\Omega$. Sites $X_i$ with $w_i \neq 0$ are called the natural neighbors of $X_0$ (see also Fig. 8 for an illustration of the general process).

The concept of natural neighbor interpolation can be generalized to interpolate not only data points but also iso-curves $X_i(t)$ parameterized over $t$, with $\Phi(X_i(t)) = \phi_i = \text{const}$. If $X_i(t) = \text{const}$ this generalization
methods, we chose [41] for our application. Finally, Schneider et al. [41] parallelized the idea of vector paradigm called Jump Flooding. Rong et al. present a general parallel information propagation algorithm of Hoff et al. [24] follows the sweep-line algorithm [15][24, 38, 41] that all handle the aforementioned generalization. The

\[ x \in \Omega \]

constraint and \( C^2 \) continuous contours. Note that only those Voronoi tiles \( \mathcal{K} \) are guaranteed to be convex for which the site is a point. Bottom: A virtual tile \( \mathcal{R} \) is inserted for the blue query point. Natural neighbor weights are computed as the fractional area \( |\mathcal{K} \cap \mathcal{R}|/|\mathcal{R}| \).

specializes to simple data points.

It is clear that in the continuous case this generalization is equivalent to finding the roots of higher order polynomials, which may have no analytic solution. We therefore resort to a discrete Voronoi diagram over \( \Omega \subseteq \mathbb{Z}^2 \). To this end, we use the equivalent computation of a discrete Euclidean distance transform (EDT), in which we store both the ID of the closest site \( X \), as well as \( X \)’s distance for each point \( x \in \Omega \). For this task, several parallel algorithms have been proposed [24, 38, 41] that all handle the aforementioned generalization. The algorithm of Hoff et al. [24] follows the sweep-line algorithm [15] and requires the potentially costly extrusion of curves into 3D geometry. Rong et al. present a general parallel information propagation paradigm called Jump Flooding to compute distance transforms. Despite being highly efficient, it can produce spurious and unbounded errors. Finally, Schneider et al. [41] parallelized the idea of vector propagation [11]. The latter method has strict and generally negligible error bounds. Considering the advantages and disadvantages of these methods, we chose [41] for our application.

Let \( D(x,y) \) denote the squared distance from pixel position \( x,y \) to the closest Voronoi site. Beginning with a seed pixel \( p_s = (x_s,y_s) \), we initialize an interval \( I(y) = [x_l \leftarrow x_t, x_t \leftarrow x_r] \). We then expand this interval to the left as long as \( \|(x_l,y_l) - p_s\|^2 \leq D(x_l,y_l) \) and similarly to the right. Moving up one line, we initialize a new interval \( I(y+1) = I(y) \). If \( \|(x_l,y_l+1) - p_s\|^2 \geq D(x_l,y_l+1) \) we shrink the interval to the left, otherwise we try to expand it. Repeating this process first to the right and then upwards yields intervals \( I(y_1), \ldots, I(y_{\max}) \) and, in the same fashion, \( I(y_{\min}), \ldots, I(y_{\max}) \) in another pass to the bottom. This essentially tracks the boundary of the region.

Traversing the pixels in the discrete region then permits computing the weights \( \omega_i \) by counting the multiplicities of Voronoi tiles \( \mathcal{K} \) covered by the region. Note that since floodfills and weight computations for neighboring pixels are mutually independent, they are trivially parallelized. Since our weights are essentially quantized to \( 1/|\mathcal{K}| \), we typically use a higher resolution for the EDT than for the final interpolation. This naturally affects the performance of the floodfill since more pixels have to be traversed, hence the need for an efficient implementation.

5.3 Post-Smoothing

Despite using a higher resolution EDT, quantization noise may still be present on the reconstructed surface. To remove these artifacts, we use a biharmonic post-smoothing instead of the more common Laplace-based smoothing, as the latter one produces tent-like artifacts around isolated points.

We therefore embed \( \Phi \) into \( \Omega^2 \times \mathbb{R}^d \) with \( \Phi(\cdot, t = 0) \leftarrow \Phi(\cdot) \) by adding an artificial time (or smoothness) parameter \( t \). The PDE to be solved is then

\[
\left( \frac{\partial^4}{\partial x^4} + 2 \frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial y^4} - \frac{\partial}{\partial t} \right) \Phi = 0. \tag{7}
\]

In order to maintain the interpolation property \( \Phi(X_i) = \varphi_i \), we introduce a cutoff radius \( \tau \) around each site \( X_i \) and we smooth locations \( x \) for which \( d(x,X_i) > \tau \). This information is readily available from
the EDT, and we set $\tau$ to the equivalent of $\sqrt{2}$ pixels. Using Gauss-Seidel relaxation and the stencil from [14], only about 100 iterations are necessary to reduce noise on the surface (see also Fig. 10).

6 Applications

6.1 Flood simulation

As an example application for the terrain reconstruction, we simulate constant rainfall over the complete domain to determine areas which might be affected by potential floods. That such flooding due to occasional but heavy rainfall is indeed a source of concern for authorities is illustrated in Fig. 11 showing two photographs taken during the flooding of 1941.

We use an approach based on the shallow water equations using the software package FullSWOF2D [12], which stands for “Full Shallow water equations for Overland Flow”. Compared to most of the other available shallow-water equation solvers, this one is specifically designed to handle the dry-wetting effects and thus fits perfectly for our simulation scenario. Further, this transforms the original problem from a three-dimensional into a two-dimensional one and hence reduces the required computational work. The shallow-water equations are solved by the software package on an initially refined regular grid without local time stepping.

Unfortunately, the inability to adapt time steps locally is not very practical for simulating longer time periods of rainfall due to the Courant-Friedrichs-Lewy (CFL) condition and the resulting tiny time steps. Therefore, we integrated the solver into the PeanoClaw [45] framework which is built upon the Peano framework [46]. By doing so, not only does the original solver gain adaptive local time stepping, but the framework also dynamically performs adaptive mesh refinement and parallelization. Both distributed memory and shared memory systems are supported in the following way: PeanoClaw partitions the original domain into different smaller domains which exchange their data with neighboring domains in form of ghost layers. Thus each domain is solved almost independently and hence can progress in time as much as numerically possible.

However, to maintain a valid and accurate solution there are certain constraints, e.g., one domain may not overtake its neighbors with respect to its current simulation time. Domains which cannot advance further in time are excluded from the ongoing computations and the available computational resources are then focused on remaining domains, which are the ones with a comparable small time step. Clearly, a full discussion of the entire simulation framework is beyond the scope of this paper and we refer the reader to [45].

The simulation itself was run on the complete domain around Mecca with a resolution of the about 6m and with constant rainfall of 50mm/h. Figure 12 shows the distribution of water after a couple of minutes of simulated rainfall. One can clearly see how the rivers and lakes are forming in the mountain area and how the domain is decomposed due to PeanoClaw.

6.2 Documentation

The DEM reconstruction can also be used for digital documentation, both for planning purposes as well as for cultural heritage. In Fig. 13, a side-by-side view of the current Grand Mosque area and the area dating back to 1968 is depicted. Changes readily apparent include the newly built hotels to the south of the mosque, the mosque extension site to the north, and flattened hills to the north-east. Regarding scale, the Grand Mosque (white in the top picture) covers 356,800 square meters and the expansion site comprises roughly twice this area.

7 Evaluation & Results

7.1 Evaluation of Contour Extraction

In order to parallelize the contour extraction process of the 24 maps, the semi-automatic contour extraction software was distributed to 19 volunteers. Since the contour extraction program’s visual framework was written in a Matlab GUI, it could be installed on volunteers’ computers using an included MCRInstaller. In a short session, the authors
presented the goal of the project to the volunteers and also the GUI and its use. None of the volunteers had any previous experience using cartographic/GIS software or were familiar with best practices in topographic digitization. The volunteers were given a week to digitize 1-2 maps in between their other activities. The majority of volunteers finished the contour extraction of their provided maps within an hour of the same day and all handed in their results by the end of the allotted period with none reporting having spent more than half an hour on an individual map. The volunteers’ work was inspected by the authors and minor mistakes made by the volunteers were corrected. After inspection the automatic OCR contour label extractor was run in Matlab on each map and the final vectorized contour lines were exported as shapefiles for evaluation in ArcGIS and as an ascii format for integration into our DEM interpolation framework (see Section 5).

For evaluation purposes we chose one of the most complicated maps of the set (Fig. 2). This map has many different types of overlapping layers, incomplete, sparse and dense contour areas, and missing data in the middle and upper right corner of the map. We used several commercial software tools (ArcGIS, R2V, WinTopo Pro, and Didger5) for comparison with the method proposed here. The softwares chosen implement the state-of-the-art in research with respect to contour line extraction as outlined before. Therefore, being the most broadly used software for topographical digitization, they are the best litmus test for determining the utility of new automatic and semi-automatic methods of contour line extraction. All of these software packages offer automatic and manual vectorization of topographic maps and range in price from $389-$4000. They provide automatic and interactive methods for setting color thresholds. In particular, Didger5 offers preset options to extract the brown colored contours of USGS maps and options for other types of colored maps or CAD drawings. WinTopo Pro specifically references its thinning algorithms and offers B-spline smoothing of vectorized contours.

In Fig. 14, we present the results of the commercial software in comparison to our proposed method. In general, all of the commercial options perform poorly since they are designed for color maps where overlapping layers can be differentiated by means of color-based segmentation. Beyond the obvious inability to distinguish contours from other layers, the softwares perform differently in their ability to trace the detected lines. All of the commercial examples except Didger5 perform well in creating smooth vector lines. However, each software varies in thickness and quality of the lines traced. In contrast, the proposed method is unique in being very robust to follow contours despite the overlapping of other features. In cases where contours cross through buildings, gaps typically occur only in places where the contour is not visibly clear. Additionally, contours are rarely side tracked

Fig. 13. Mecca Now and Then. Top: current Google Earth View. Bottom: Our reconstruction of data from c. 1968. Note the massive extension plan north of the Grand Mosque (red arrow), newly built hotels (green arrow), new roads and razed mountains (blue arrow).

Fig. 14. Magnified view of commercial and proposed method’s contour line extraction results. R2V (a), ArcMap (b), WinTopo Pro (c), Didger5 (d), Proposed Method (e), Proposed Method (f) with OCR detection and single user edit. Note that all commercial methods fail to isolate contours from other features due to the lack of color information. With only minor user interaction the initial extraction in (e) is able to isolate contours without creating false contours of buildings, labels, and grid lines. In (f) we show the automatic joining of broken contours using OCR detection.
contrast to the majority of previous work, our approach does not rely on previous surveys, but it reflects how the terrain looked fifty years ago. In the visited city in the world), this resolution is higher than typical satellite working day. For the central area of Mecca (the most traveled and creation for the entire scanned map set was completed within one week. Relying on volunteers of our lab to work in parallel, DEM construction from a large set of historical topographic maps. We have presented a visual framework to enable the digital reconstruction of terrain from a large set of historical topographic maps. We showed how contours can be extracted semi-automatically and with minimal user interaction from 24 grayscale maps. Furthermore, we demonstrated that the extracted DEM can be used to simulate the interpolation of scanned Mecca map 08SW1, our method produces contours that are fast and does not produce invalid tags at the same time. Comparing intervals, all methods perform within expectations, with ArcGIS’ TIN methods fare best, followed by our method. For the 100m contour intervals, all methods perform within expectations, with ArcGIS’ TIN performing best and Surfer’s NN and our method tied for second place. It is worth noting, though, that some of the methods unnecessarily produce samples tagged as invalid values. Our method is the only one that is fast and does not produce invalid tags at the same time. Comparing the interpolation of scanned Mecca map 08SW1, our method produces smooth continuation outside the convex hull of the contours.

8 CONCLUSIONS & Future Work

We have presented a visual framework to enable the digital reconstruction of terrain from a large set of historical topographic maps. We showed how contours can be extracted semi-automatically and with minimal user interaction from 24 grayscale maps. Furthermore, we reconstructed a DEM comprising 16km × 9km at a grid spacing of 1m. Relying on volunteers of our lab to work in parallel, DEM creation for the entire scanned map set was completed within one working day. For the central area of Mecca (the most traveled and visited city in the world), this resolution is higher than typical satellite surveys, but it reflects how the terrain looked fifty years ago. In contrast to the majority of previous work, our approach does not rely on color maps and can handle challenging maps with alternating areas of densely packed and sparse contours, significant overlap between layers, isolated data points, and the absence of color. To the best of our knowledge it is also the first to integrate an OCR module to automatically extract contour labels and assign elevation information to each contour and elevation point. Our interpolation method is fast and compares favorably with commercial software representing the pinnacle of research, while at the same time being able to cope with mid- to large-size data. Furthermore, our visual framework reduces the complexity of accurate topographic digitization, making it easily accessible to an untrained volunteer workforce. As an application, we demonstrated that the extracted DEM can be used to simulate and visualize past floodings of Mecca, an information valuable to authorities for the purpose of future planning. Finally, we apply the DEM as a comparison to document changes of Mecca over the last fifty years.

In the future, we would like to to scan an even larger set of more than 100 sheets of 1:1,000 topographical maps comprising the same area, plus additional sheets to cover the hinterland, in order to improve the accuracy and coverage of our DEM. In order to fully facilitate crowd sourcing methods, we would like to make our visual framework available—either as an online tool or a plug-in to commercial software—to the millions of users interested in participating in the preservation of the cultural heritage of Mecca.

Additional directions for future research include the application to other large historical map sets or even other data modalities such as core drillings, the extraction of building floor plans or road networks from these maps, etc. The latter ambition is of course supported by the availability of the extracted contours, which can now be removed digitally to expose the next layer of information in these maps.

In the context of the DEM reconstruction, we would furthermore like to extend our method by estimating and utilizing slopes in order to achieve higher continuity across contours, and we would like to further accelerate our method using a GPU-based parallelization.

Acknowledgements

This research was conducted in collaboration with Al-Makkiyah Al-Madaniyah Institute, Jeddah, Kingdom of Saudi Arabia and the Chair of Scientific Computing in Computer Science at the Technische Universität München, Germany. In particular, we would like to thank Dr. Sami Angawi and his two sons, Ahmed and Ammar, of Al-Makkiyah Al-Madaniyah Institute for providing the maps used in this paper. We would also like to express our immeasurable gratitude to the Angawi family for frequent and very valuable discussions and guidance, as well as for the active help in scanning, organizing, understanding and analyzing the Makkah data set.
Fig. 15. Overview of 19 out of the 24 maps used in this work. For reference, the region tinted green is covered by a single map.

Ali Thabet was funded through KAUST OCRF Award No. KSA-C0069. Neil Smith and Jens Schneider were funded by the Visual Computing Center at KAUST. Roland Wittmann was partially funded through KAUST Grant UK-c0020; KSA-c0069.

REFERENCES

[17] Google. tesseract-ocr—an ocr engine that was developed at hp labs between 1985 and 1995... and now at google., 2014. accessed Sep.2014.
Fig. 16. A visual comparison of various reconstruction methods. Kindly also refer to the digital version of this report. Note that the synthetic Puget Sound data is visually very close for most methods, as confirmed by Table 1. In contrast, differences are readily apparent for sheet 08SW1.