Urban Pattern: Layout Design by Hierarchical Domain Splitting

Yong-Liang Yang
KAUST

Jun Wang
KAUST

Etienne Vouga
Columbia University

Peter Wonka
KAUST

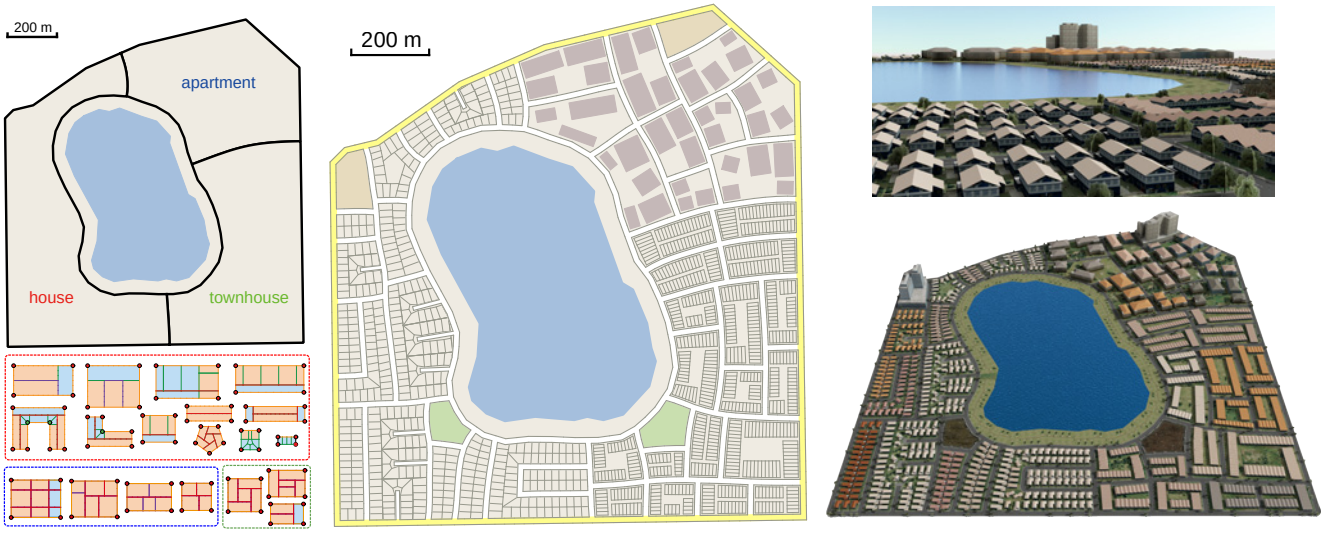

Figure 1: *Starting from a polygonal region and user-prescribed design elements (top left), we hierarchically subdivide the input region using streamline-based and template-based splitting operations. Selected templates are shown on the bottom left, the final layout in the middle, and a simulated 3D construction on the right.*

Abstract

We present a framework for generating street networks and parcel layouts. Our goal is the generation of high-quality layouts that can be used for urban planning and virtual environments. We propose a solution based on hierarchical domain splitting using two splitting types: streamline-based splitting, which splits a region along one or multiple streamlines of a cross field, and template-based splitting, which warps pre-designed templates to a region and uses the interior geometry of the template as the splitting lines. We combine these two splitting approaches into a hierarchical framework, providing automatic and interactive tools to explore the design space.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—;

Keywords: street layouts, parcel generation, computational design, mesh optimization, region splitting

Links: DL PDF VIDEO

1 Introduction

We present a framework for generating street and parcel layouts (see Fig. 1). Besides synthesizing virtual urban environments, our work is useful for planning future developments. In contrast to existing work in urban modeling, e.g., [Chen et al. 2008a; Weber et al. 2009], we aim to generate solutions with high geometric quality that respect some specified polygonal boundary conditions. Because of the slow and chaotic historical process of urban development, many of our oldest cities have inefficient layouts, including labyrinthine streets that meet at odd angles, and land parcels that are oddly shaped and proportioned. Existing work in urban modeling, focused primarily on synthesizing virtual urban environments, generates solutions that mimic many of these problems. For urban planning, these artifacts are not acceptable: we do not want to recreate the problems of the past when planning high-quality layouts for the future. We therefore propose a new framework that generates high-quality, user-controllable street and parcel layouts.

There are two reasons why we believe that this is an interesting problem. First, existing subdivisions are currently generated without computational design tools, which often leads to inefficient solutions in the form of undesirable parcel shapes, parcel sizes, or too much land devoted to roads. Given the limited availability and the high cost of land in most urban areas, the financial impact of better planning is significant. Second, the problem of urban layout generation is an interesting mesh generation problem in geometry that has not been solved with high geometric quality.

In Fig. 11, left (CE1), we visualize a street and parcel layout that was generated with a current state-of-the-art urban modeling tool (CityEngine). We observe two major problems that we wish to overcome. First, its street layout algorithm is based on growing street segments and this strategy fails when more complex bound-

aries are prescribed. Parts of the street network will grow together from different sides and they do not match up well. Second, the algorithm for parcel generation [Vanegas et al. 2012], while being very general, leaves no significant opportunity for user control of the style of a subdivision.

Problem Analysis: We discussed the problem with about 10 urban planners and architects (in this paper we simply call all practitioners urban planners or planners). As a result of these discussions we decided to focus on the layout of an area with up to a few thousand parcels. That is the approximate scale that would be developed within a single urban planning project. Further, almost all realistic projects come with fixed boundary conditions that have to be respected, e.g., defined by property rights. Additionally, it became clear that urban planners wanted to maintain full control over key design elements of a layout. We therefore include several interactive tools in our framework. While our framework admits a larger range of user interactions at all steps of the layout process, we focus the exposition of the paper on a single type of workflow that mimics a typical top-down design process employed in practice. In this workflow, the user interactively provides key design elements as constraints (e.g., main roads, a lake, a school, distinctively shaped parks, an opera house, etc.) and our framework will create multiple and different versions of complete designs by filling in the details (in current practice, a lead designer gives the key elements and a team of planners work on many variations of complete designs). By focusing on this top-down design, we can separate the interactive and automatic design steps and provide a more meaningful evaluation and comparison of our work.

The second part of the problem concerns the technical challenges on how to generate a nice layout. There is a considerable amount of literature on designing local subdivisions and street networks [Marshall 2005; Southworth and Ben-Joseph 2003; Robinson et al. 2004], considering aspects such as traffic, safety, visual quality, transportation efficiency, and noise. As solutions we can find many interesting prototypical urban patterns that are considered desirable. Examples include regular grids for efficient transportation, cul-de-sacs for privacy, or curved roads for visual quality. As the recommendations for good design differ, we do not want to limit ourselves to one planning philosophy, but instead give the planner the option to work with those patterns that the planner prefers (or that are required by a project).

Challenges and Contributions: The main idea of our proposed framework is to let the user specify (draw) prototypical templates and then deform these templates to tile the available space. Urban patterns are usually generated from basic building blocks such as rows of parcels along streets or parcels arranged around cul-de-sacs, called *spikes* (see Fig. 2). As we tried to draw a small set of initial templates, we quickly noticed that even a minimal set of templates for urban layouts is very large. This is due to the fact that there is a combinatorial explosion in arranging different discrete design elements (e.g., parcels). We therefore propose to use hierarchical templates and templates that can repeat design elements to combat this combinatorial explosion. The next question is how to fill the domain with templates. Here, we identified two different approaches. Our first attempt was to pack and deform the templates directly (similar in spirit to [Kim and Pellacini 2002]), but that leads to unrealistic street networks with many short continuous street segments connected by T-junctions. We therefore propose a solution based on hierarchical domain splitting that proves to be significantly more successful. If a region cannot be covered by a template, we split it using one or multiple streamlines of a cross field. The function of the field is to ensure proper spacing of the new streets with respect to the boundary and other streets and to give the user a global view of the design. Template-based splitting warps pre-designed templates to a region and uses the interior geometry of the template as splitting lines. Template-based splitting is useful for generating unique and interesting urban patterns, i.e., local street and parcel patterns according to the user's design preferences. The advantage of this solution is that the design process only consists of drawing prototypical examples rather than programming or writing grammars. To obtain a complete system, we build a hierarchical framework, automatically combine streamline- and template-based splitting using heuristic search and backtracking, provide a user interface for the integration of user feedback, and suggest metrics to evaluate the quality of urban layouts. We claim the following contributions:

- From the application side, we significantly improve the geometric quality of street and parcel layouts presented in previous work, so that our layouts can also be useful for planning purposes.
- From the methodology side, we make two contributions. First, we present an overall framework to combine streamline-based and template-based splitting operations. Second, we propose a novel template warping algorithm for hierarchical templates and templates with repeating design elements. This is a significant extension to existing template warping algorithms, e.g., Aliaga et al. [2008b].

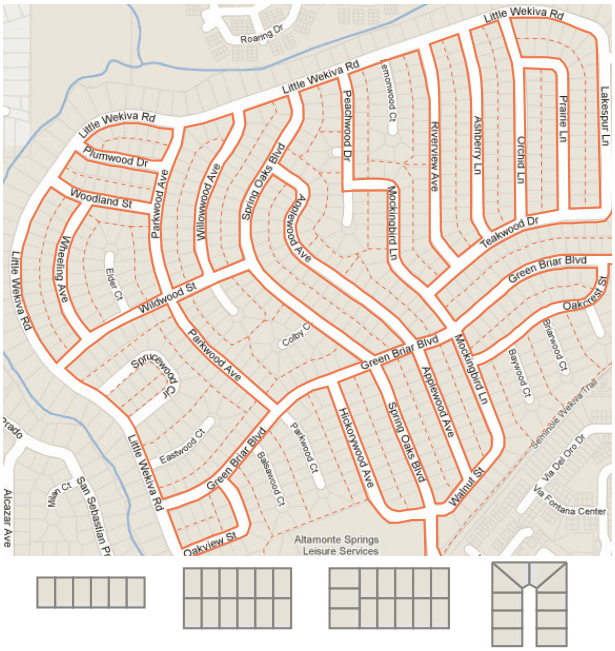

Figure 2: *Block subdivisions using parcel strips and spikes. We show a real-world layout on top and use red lines to highlight the structure and the breakdown of the region into typical parcel patterns. The four most common patterns are shown on the bottom.*

2 Related Work

Modeling streets and modeling parcels have been studied as separate problems such that all previous work first fixes the street network and then further subdivides the induced partition of land into parcels. The original version of CityEngine, inspired by L-systems [Prusinkiewicz and Lindenmayer 1990] suggested a growth-based system for street layouts that grows street segments from active seed points [Parish and Müller 2001] and has been refined and adapted by Weber et al. [2009]. Aliaga et al. [2008a] proposed connecting points seeded by a synthesis algorithm and Chen

et al. [2008a] proposed generating streets as streamlines in a tensor field. Chen et al. focused on coarser layouts with an eye toward highways and major roads, whereas we are interested in residential street and parcel layouts. In this problem domain, it is important that the spacing between two parallel streets remains approximately constant, and that land parcels are well shaped and proportional. By tracing streamlines of low-divergence cross fields and filling in the fine-scale design using template matching, we address for the first time these additional desiderata. Editing and combining street layouts has been studied by Lipp et al. [2011]. Existing parcel generation approaches heuristically mimic existing designs [Parish and Müller 2001; Vanegas et al. 2012]. In contrast to our work, these parceling techniques offer the user little control over the design and they can generate undesirable layouts, e.g., parcels without street access. An advantage of these alternative approaches is that they can (in theory) generate parcel subdivisions for any region. An important idea of Vanegas et al. [2009] was to include behavioral aspects of a city to control the geometric layout. The design philosophy of our work is similar in that we also consider functionality as a factor in geometric design.

At a larger scale, modeling roads connecting individual cities [Maréchal et al. 2010; Galin et al. 2011] also considers factors such as the topography, terrain, and the graph connectivity, but the shapes of the polygons between roads are not considered. Complementary to our work is procedural building generation [Wonka et al. 2003; Müller et al. 2006; Merrell et al. 2010; Talton et al. 2011] and sketch-based design [Paczkowski et al. 2008]. We refer the reader to a recent survey [Vanegas et al. 2010] for a broader review of urban modeling.

We are also aware that commercial civil engineering software, e.g., Autodesk's Civil3D [Civil3D 2013] or SiteOps [SiteOps 2013], has nice semi-automatic tools for street and parcel generation. However, the automatic part of the software relies on simple, deterministic construction algorithms. For example, streets and sidewalks can be generated by offset operations and parcels can be generated by splitting a region along lines orthogonal from the main road. This type of construction often creates undesirable leftover spaces that cannot be subdivided by nicely shaped parcels. The difficult part of the layout optimization is therefore left to the user.

3 Overview

Input: The first input to our system is a polygonal region, $\mathcal{R}_{input}$, together with user-specified key design elements (or constraints) of the layout, e.g., major roads, lakes, parks, landmark buildings, or other unique subregions. These design elements may split $\mathcal{R}_{input}$ into multiple subregions or specify holes in $\mathcal{R}_{input}$. The second inputs are user-designed (hierarchical) templates together with a specification of different aspects of their deformation behavior. Our framework provides interactive tools for modeling both inputs.

Hierarchical Splitting In the first stage, we use an automatic and hierarchical splitting framework. The core of the framework is two operations to split a region, $\mathcal{R}$, in the hierarchy:

1. **Template-based Splitting:** Given a library of templates, $\mathcal{T}_i$, $i = 1, \ldots, n_T$, we would like to select those that can be warped to a given region, $\mathcal{R}$, by evaluating the warping energy ($energy$) that is necessary to deform the template region of $\mathcal{T}_i$ to the region $\mathcal{R}$. The energy not only considers the deformation of the template boundary, but also the deformation of the interior geometry that specifies the split (see Sec. 4).

2. **Streamline-based Splitting:** A region, $\mathcal{R}$, is split using one or multiple streamlines of a *cross field*. We propose multiple solutions for cross field design and a scoring function to evaluate and rank streamlines automatically (see Sec. 5).

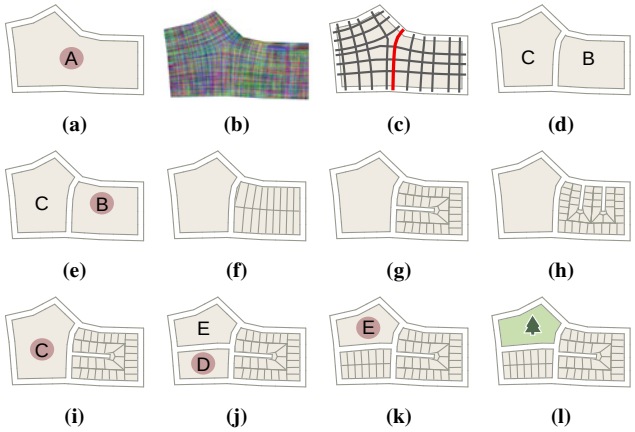

Figure 3: *Illustration of the splitting operations. The top row illustrates streamline-based splitting: (a) a region, A, is selected for splitting; (b) a cross field is computed; (c) streamlines are extracted and ranked (the best one is highlighted in red); (d) the region is split along the best streamline generating two subregions, B and C. The middle row illustrates template-based splitting: (e) the subregion B is selected for splitting and available templates are deformed to evaluate how well they fit. Three candidate templates are shown in (f), (g), and (h). Option (g) is selected as the best match. In the bottom row, the example is finished with streamline-based splitting (i,j) and template-based splitting (k,l).*

We combine these two splitting algorithms in a hierarchical framework as follows. The automatic algorithm always prefers template-based splitting. Only if a suitable template (with $energy < threshold$) cannot be found does the framework use streamline-based splitting. In streamline-based splitting, the splitting line(s) are randomly selected according to their quality function, $qual$. In template-based splitting, the templates are randomly selected based on a score that is the product of the deformation energy and a user-defined probability. The user-defined probability is used to encode the fact that some templates are less common than others and some templates are more general and can more easily be matched with lower error. Additionally, other constraints, e.g., a template should occur a maximum of k_1 times or a minimum of k_2 times, can be met by randomized search and backtracking. We typically generate 10 to 100 variations of a layout. If the framework is not successful in generating variations, we invoke a rule-based assignment that either uses the parceling algorithm based on the straight skeleton [Vanegas et al. 2012] or places a park or commercial area. In Fig. 3, we illustrate the two splitting operations using a simple example.

During modeling, we provide statistics (quality metrics) and visualization tools for assessing the layout quality. The most important quality metrics are described in Sec. 7. We also mention an implementation detail necessary for integrating the two splitting strategies. The streamlines conceptually correspond to road centerlines. Before template matching, we therefore offset region boundaries by half the street width if necessary (managed by flags stored with half-edges of the polygonal regions).

Global Optimization: All variations are post-processed by a global optimization algorithm (see Sec. 6) that improves the street network with the goal of reducing the distortion of the template warping.

Quality Metrics: The user can review the generated variations in

an interactive interface and analyze the layout quality using a set of quality metrics (see Sec. 7).

Interactive Post-processing: In this optional step, the user can change template assignments, delete subregions, recompute subregions, and modify streets. One important use of this step is to modify template assignments of difficult regions (e.g., select a region to be processed by the straight skeleton based parceling algorithm) or place additional key elements like parks and parking lots.

We present example layouts together with comparisons to the state-of-the-art and existing layouts in Sec. 8 and conclude in Sec. 9.

4 Template-based Splitting

In this section, we discuss templates, their selection, and best matching.

We build on Aliaga et al.'s idea of using deformations for urban templates [2008a; 2008b]. This previous solution leaves a lot of room for technical improvement, which we will provide: 1) We solve a non-linear optimization problem using known explicit solutions of various types of 2D registration problems. 2) The deformation can be controlled by specifying the type of admissible deformations locally within a template and enforcing it up to a given tolerance. For example, a row of parcels can be easily stretched in one direction where new parcels can be added, but it should not be stretched in the other direction where all parcels will become longer and have a different area as indicated by the template. 3) We have hierarchical templates so that one template represents hundreds to thousands of discrete variations. 4) We encode template compatibility, so that only compatible boundary edges and corners are allowed to match. Our nonlinear solver is easy to implement and efficient enough that there is no need to sacrifice accuracy and design quality through a reduction to a single linear system. By contrast, Aliaga et al. linearize a non-linear problem directly. In the best case, this result is comparable to one search step of a non-linear solver and therefore this approach would require an excellent initialization that cannot be guaranteed in practice.

Given a library of templates, $\mathcal{T}_i,\ i = 1, \ldots, n_T$, we would like to select those that can be warped to a given region, $\mathcal{R}$, while keeping the essential user-defined properties of the split (described below and illustrated in Fig. 4) stored with the respective template. The core of our solution to this problem is an optimization algorithm for the best warp, $\mathcal{T}_i \rightarrow \mathcal{R}$.

Templates. A template, $\mathcal{T}$, is a polygonal region, split into polygonal subregions, $\mathcal{P}^k$, where each $\mathcal{P}^k$ comes with an allowed range of shapes it can assume. Instead of describing the space of admissible shapes, we prefer to prescribe the type of transformations it can undergo. In our current implementation, these are rigid body motions, uniform scalings with a prescribed range of the scaling factor, and affine maps with tolerances on the principal distortions. The overall template may undergo a nonlinear deformation, while its parts, $\mathcal{P}^k$, deform within a tolerance according to the prescribed affine deformation class. For each $\mathcal{P}^k$, we store its vertices $\mathbf{p}_i^k$, its deformation behavior and tolerances, and possibly some alignment properties, expressing that certain vertices should map to corners, to the boundary of $\mathcal{R}$ or should lie on additional key elements of the split, e.g., a straight line segment, L_j, or a smooth curve, C_j (see Fig. 4 alignment curve). Hence, the *key elements of a template* are these alignment elements and the vertices, $\mathbf{p}_i^k$, of the subregions, $\mathcal{P}^k$. Note that the $\mathcal{P}^k$ may be templates themselves, as illustrated in Fig. 4 and Fig. 5.

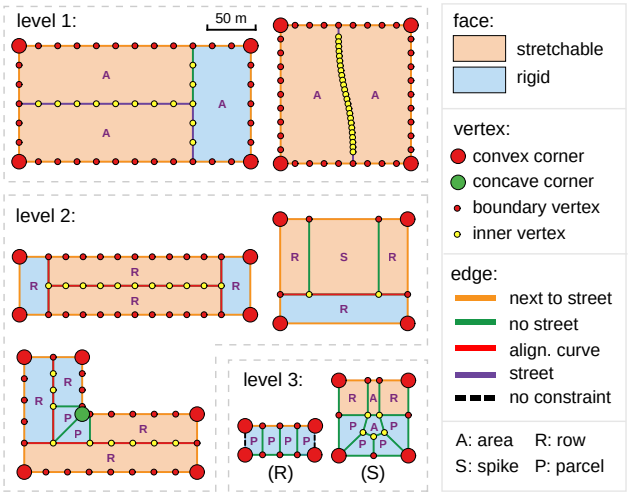

Figure 4: *Examples of frequently used templates along with semantics and notation. We show templates grouped into three approximate levels. In general, each template has a region label and all subregions (faces) of a template also have region labels. The first condition for a template to match is that the region labels agree. We structure the templates into three levels so that in general level-k templates are used to replace faces of level-$(k-1)$ templates. The exception is the level-three parcel template (R) that also occurs in the spike template (S). Regions labeled A are area templates that can be replaced by level-two templates. Edges can be constrained to have certain neighbors (street, no street, unconstrained) and they can also generate geometry (street). Vertices in a template are automatically classified as convex, concave, or regular (boundary), but the user can override this automatic classification. The complete set of templates and their prototypes from Google Maps can be found in the supplementary material. These templates are designed in a custom-made editor. Note that one pentagon-shaped template has a private park in the middle. This is an example of an intentionally designed region that has no street access.*

4.1 The Warping Algorithm

The unknowns of our optimization problem are the new positions $\mathbf{p}_i^{k*}, C_j^*, \ldots$, of the key elements, $\mathbf{p}_i^k, C_j, \ldots$, of a template.

Initialization. Templates possess boundary points marked as corners (convex or concave), which should map to corners of $\mathcal{R}$. Moreover, their boundary segments have a semantic meaning depending on the application, such as "street" or "no street". Thus, we first detect the convex and concave corners of $\mathcal{R}$. Then, for each template with the right arrangement of corners and boundary curve semantics, we look for compatible boundary matches and rank them according to the distortions of boundary curve lengths. This results in a number of candidate templates. To initialize the warping for a selected candidate template, we warp boundary curves via arc length scaling and subsequently map interior points using mean value coordinates.

Basic Iteration: In order to keep the algorithm as local as possible and its implementation simple, we decided to proceed with an alternating optimization:

- *Registration*: Register each subregion, $\mathcal{P}^k$, of the template within its allowed deformation type and constraints to the current key vertex positions, $\mathbf{p}_i^{k*}$.
- *Regression*: Find new positions of key elements, $\mathbf{p}_i^{k*}$, through

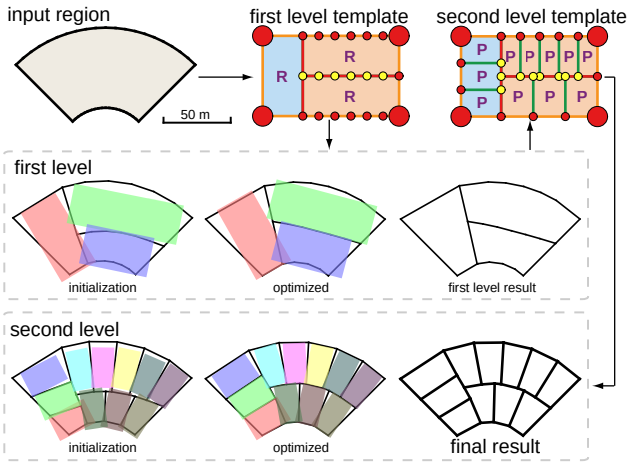

Figure 5: *Hierarchical template matching. The optimization result of one level determines the selection of templates for the next finer level. In this example, the number of parcels per row. The parcel rows are then optimally placed in the second level. The registered and colored subregions of the templates visualize the effect of optimization.*

weighted least squares regression of the vertices of the warped subregions $\mathcal{P}^{k*}$, respecting the alignment constraints.

We now provide the details on these two parts:

Registration. For each subregion, $\mathcal{P}^k$, we have to compute a transformation, α, within the allowed class such that the least squares error,

$$E^k := \sum_i (\alpha(\mathbf{p}_i^k) - \mathbf{p}_i^{k*})^2, \tag{1}$$

is minimized. All of our deformations can eventually be broken down into affine maps, $\alpha : \mathbf{p} \mapsto \mathbf{a} + A \cdot \mathbf{p}$. Affine registration is known to be simple: the barycenter of the vertices, $\mathbf{p}_i$ (dropping the upper index k for simplicity), maps to the barycenter of their corresponding points, $\mathbf{p}_i^*$, which determines the translational component, $\mathbf{a}$, and leaves us with registration problems for linear maps, $\mathbf{p} \mapsto A \cdot \mathbf{p}$. Now,

$$E^k = \sum_i [\mathbf{p}_i^T A^T A \mathbf{p}_i - 2\mathbf{p}_i^{*T} A \mathbf{p}_i + \mathbf{p}_i^{*T} \mathbf{p}_i^*],$$

is a quadratic function in the entries of the 2×2 matrix A, which therefore can be found by solving a linear system. Registration by a similarity is even simpler, with only two unknowns in A, namely $c = a_{11} = a_{22}$ and $s = a_{21} = -a_{12}$; the scaling factor is $\lambda = \sqrt{c^2 + s^2}$ and the rotational angle ϕ follows from $c = \lambda \cos\phi$, $s = \lambda \sin\phi$. A rigid body motion is characterized by the constraint $c^2 + s^2 = 1$; the corresponding registration problem can still be solved explicitly, since then E^k is linear due to $A^T A = I$.

Due to the iterative nature of the algorithm, it is not essential to solve the registration subproblem with high accuracy; one iteration of any appropriate numerical optimization algorithm suffices. This is used when we have other constraints on the affine map. One example is affine maps where the principal distortion directions and intervals on the distortion factors are given. This problem arises when a rectangle, $\mathcal{P}^k$, should stay (close to) a rectangle. We first perform the registration by a similarity and then append the registration by non-uniform scaling in the principal directions: this is affine registration with a diagonal matrix, A, and linear inequalities for the diagonal elements, resulting in a very simple quadratic program in just 2 unknowns. Other examples are similarities with bounded scaling factor and affine maps with area distortion constraints.

Regression. After the registration phase, vertices which are shared in the template by adjacent regions, $\mathcal{P}^k, \mathcal{P}^l, \ldots$, will not agree, nor will they lie on a boundary or a possibly prescribed alignment curve. Let us first discuss the case where several points, now simply called $\mathbf{p}_1, \ldots, \mathbf{p}_n$, shall be merged to a single point, $\mathbf{p}$. In view of the constraints and tolerances on the deformed subregions, $\mathcal{P}^k$, we define a tolerance ε_i for each $\mathbf{p}_i$, and let $\mathbf{p}$ be the barycenter of points, $\mathbf{p}_i$, with weights $w_i = 1/\varepsilon_i$. If $\mathbf{p}$ is constrained to lie on a given curve, we project it orthogonally onto the curve. If a series of points should lie on a simple non-fixed curve (straight line, circle, etc.) we compute a weighted fit of all involved points, $\mathbf{p}_i$, and project them onto the fitted curve. Finally, if the points should lie on an unknown smooth curve, we perform a weighted moving least squares projection.

Our matching algorithm shares some aspects with [Liu et al. 2008] who parameterize triangle meshes via constrained affine deformations of faces. It could be implemented in a non-alternating fashion, which would probably exhibit faster convergence. However, since subregions may transform in many different ways and in view of the presence of various types of alignment elements, our approach is much easier to implement, while still achieving the performance needed for the application. Currently, one matching takes about 30 iterations and we can approximately match 25 templates per second.

Hierarchical Matching. Since subregions of templates may be templates themselves, we proceed in a hierarchical fashion, starting at the coarsest level. Each level has an initialization phase, followed by optimization (see Fig. 5).

Best Warp Selection. In our implementation, the choice of a template is made in the initialization phase, where we randomly pick a template from the ones ranked most highly according to the deformation of the boundary. This may still result in a warp that exceeds certain tolerances on the deformation of its subregions. Although we did not find it necessary in our application, one could decide on the template after the complete matching algorithm has been performed on all templates that passed the initialization criteria.

Comparison. We compare the results of our algorithm to warping using only mean-value coordinates in Fig. 6. We can see that the distortion of our framework is significantly lower.

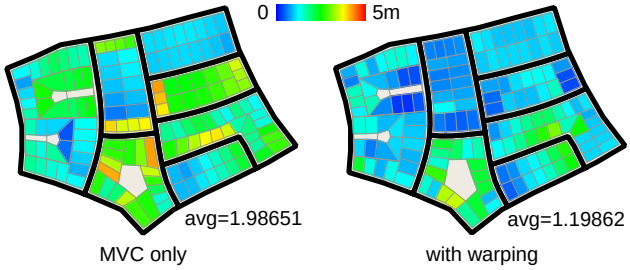

Figure 6: *Template matching based only on mean-value coordinates (left) and our warping algorithm (right). The parcels are color-coded based on the registration distance from the ideal parcel.*

5 Streamline-based Splitting

While template warping works well for designing street and parcel layouts at a fine scale, it is not ideally suited to coarse-scale design, since the region boundary can be highly irregular, and it is difficult to capture the more freeform layouts of major road networks with pre-designed templates. To accommodate common templates, streets in these networks need to meet at approximately right angles, with parallel streets staying a roughly constant distance apart. To meet these design goals, placement of streets at a coarse level is guided globally by a *cross field* over $\mathcal{R}$. A cross at point $\mathbf{p} \in \mathcal{R}$ is a pair of orthogonal straight lines through $\mathbf{p}$; streamlines through the cross field correspond to potential road placements.

We design the road network by choosing a cross field over $\mathcal{R}$, and then choosing a streamline of the field. The streamline partitions $\mathcal{R}$ into two subregions; recursively applying this approach yields a hierarchical street network over $\mathcal{R}$. In the supplemental materials, we show that finding cross fields that are smooth, have low divergence, and are adapted to the boundary of $\mathcal{R}$ satisfies the above design goals, and we propose three algorithms, with different advantages, for generating such cross fields. Our framework is not tied to any particular method of cross field design and many existing algorithms could be used. Once a cross field has been chosen, our streamline evaluation algorithm selects one of its streamlines along which to split $\mathcal{R}$.

The streamline evaluation algorithm takes as input a region $\mathcal{R}$ and a cross field and computes a set of candidate streamlines as well as a quality score for each streamline. Below, we discuss the generation of streamline candidates, the overall quality score, and the four individual components of the quality score.

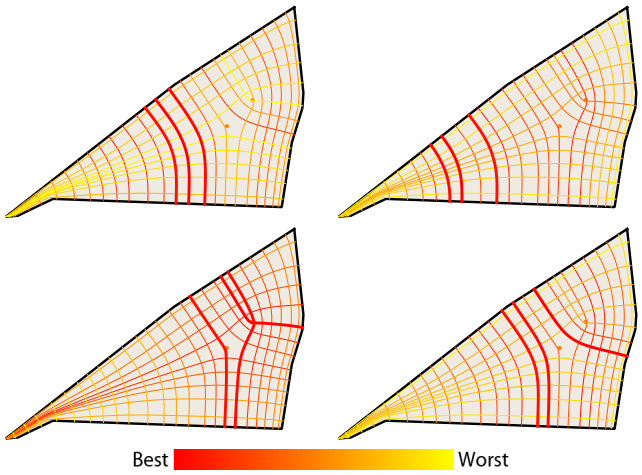

Figure 7: *A visualization of the quality scores for streamlines. The best three streamlines are highlighted by a thicker line width. Top left: The divergence quality score. In this example we can see that one direction is clearly preferred to the other. Top right: The distance-to-boundary quality score. Bottom left: The distance-to-singularities quality score. Bottom right: The combined (overall) quality score. Note that we use a much denser set of streamlines for the computation of the final results than in this visualization.*

Streamline Candidate Generation: All inner vertices, v_i, of the constrained Delaunay triangulation of $\mathcal{R}$ are considered potential seed points for a streamline. Each inner vertex has two associated orientations through the cross field and we store two binary flags to mark each of these orientations as active or inactive. We iteratively generate a streamline from a random seed point in a direction that is still active, until all seed points are inactive in both orientations. A streamline is computed from a seed point in both directions associated with an orientation until it hits the boundary of R, using Runge-Kutta as the streamline integration method (similar to Alliez et al. [2003]). After a new streamline is placed, we disable the orientation that is most similar to that of the new streamline for all seed points within a Euclidean distance of d_ε of the streamline. Streamlines that stop at singularities and self-intersecting streamlines are not considered as candidates. The parameter d_ε is a tradeoff between the computation speed and layout quality (we use $d_\varepsilon = 0.3 * parcellength$ for our results).

Quality Score: The quality score of a streamline $qual(s_i)$ is a combination of four components (see Fig. 7): divergence, $DIV(s_i)$, distance to the region boundary, $DB(s_i)$, distance to singularities, $DS(s_i)$, and continuity, $CT(s_i)$. The individual scores can be combined by optionally normalizing each score to the range $[0, 1]$ and by controlling linear weights $\lambda_1, \ldots, \lambda_4$ in the user interface. For our results, we normalized the scores and set all weights to one. Changing the weights from this default setting is one way to explore different design variations for the same input region.

Divergence: The divergence score of a streamline, s, is computed by integrating the divergence

$$DIV(s) = l(s) \int_s \nabla \cdot f(x)dx,\ l(s) = 1/length(s)^p, \qquad (2)$$

where $p = 0.9$ and $length(s)$ is the arc length of s. The parameter p is set to favor longer streamlines. In our implementation, the integral is discretized as a sum.

Distance to the Boundary: The distance to the boundary score tries to estimate how well rectangular templates with given widths, $W_1, \ldots, W_k$, can be placed parallel to the streamline. We precompute possible width combinations by an exhaustive search (up to a threshold) and compute a function $opt(y)$ that evaluates how well a value y can be explained by a sum of W_i's. We compute the pointwise quality score, $db(x) = opt(dist(x))$. The final score, $DB(s)$, is then the average over the streamline of the pointwise score, $db(x)$.

Distance to Singularities: The distance to singularities, $DS(s)$, is the smallest Euclidean distance to a singularity in the field. This component is based on the observation that streamlines close to singularities sometimes break the region $\mathcal{R}$ into two subregions ($\mathcal{R}_1$ and $\mathcal{R}_2$), so that after recomputing the field for both subregions the number of singularities in $\mathcal{R}_1$ and $\mathcal{R}_2$ is smaller than the number of singularities in $\mathcal{R}$. This is due to the fact that D-fields (the most often used fields in our work) push singularities away from the boundary to the center. Furthermore, we observed that we often selected streamlines close to singularities when we experimented with manual streamline selection.

Continuity: Continuity encourages streamlines that are the natural continuation of existing splitting streamlines. We store all existing streamline endpoints and their directions. An endpoint of a candidate streamline is considered to be continuous if it is within a Euclidean distance threshold, d_{dist}, and an angle threshold, d_{dir}, of an existing endpoint. A streamline with one continuous endpoint has $CT = 0.5$, with two continuous endpoints, $CT = 0$, and $CT = 1$, otherwise. In our implementation, we chose $d_{dist} = 4$ meters and $d_{dir} = 20$.

The streamline ranking algorithm results in a ranked set of streamlines where each streamline has a quality score.

6 Global Optimization

In the following, we describe our algorithm to optimize the street network. We propose an algorithm that iterates two phases: fitting strips to the blocks induced by the street network and optimizing the network using convex optimization. This type of iterative framework was also successfully used in the context of shape approximation [Cohen-Steiner et al. 2004] and deformation [Bouaziz et al. 2012].

Problem statement: The input of the optimization is a street network represented as polygonal mesh, $M = (V, E, F)$. The goal of the optimization is to improve the fairness of the roads and the regularity of the blocks, i.e., the block shape should be like a strip bounded by one pair of offset curves. Fig. 8 presents an illustration of what the global optimization aims to improve. After the global optimization, the same templates are warped again to fit the optimized regions. The energy function we want to optimize is defined as:

$$F(V) = w_s f_{shape} + w_f f_{fair} + w_c f_{close}, \tag{3}$$

and we describe the individual terms below. In this paper, we use $w_s = 1$, $w_f = 0.8$, and $w_c = 0.2$.

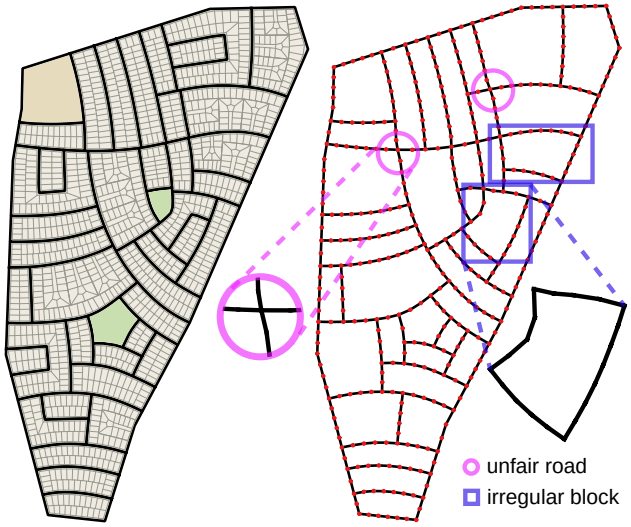

Figure 8: *An input road mesh for optimization (right) from hierarchical domain splitting (left). In order to improve the road network, we collapse short road edges. This avoids continuity problems at intersections, but the roads are no longer smooth (pink circles).*

Regularity. The main idea of regularity is to lower the deformation error introduced by warping templates. In the following, we describe the regularity term for the most common type of templates that are designed as rectangles and are mapped to quad-shaped blocks (faces $\in F$). Each such block has two pairs of opposing sides (p_1 and p_2) and we compute a target shape in the form of a *strip*. A strip is a quad where one of the two pairs of opposing sides is defined by two offset curves. We first need to decide which of the pairs to select for fitting offset curves. For each pair, we compute the offset deviation by uniformly sampling and then we select the pair of polylines with less distance deviation (called (c_1, c_2) in the following). We then construct the target strip as follows: (1) we uniformly generate the same number of samples on c_1 and c_2 and form a middle polyline, c_m, by linking the middle points m_i's; (2) the constant offset l is determined by the preferable template size or estimated by averaging all the sampled distances between c_1 and c_2; (3) the two target curves are generated by marching along the

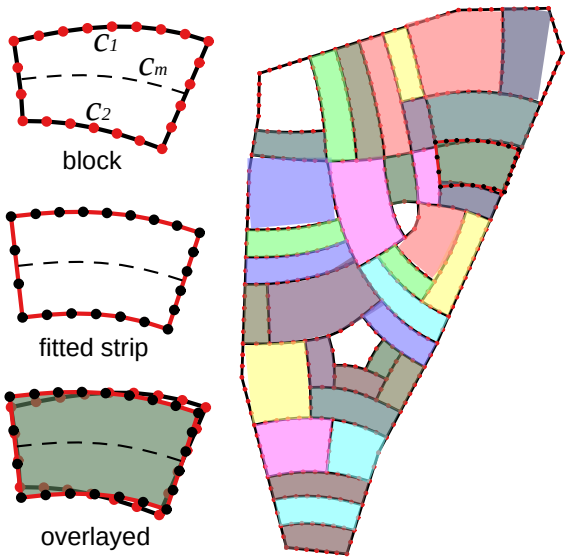

Figure 9: *Fitting a strip with two offset curves to a given block (left) and the input layout overlayed with fitted strips (right).*

normal of m_i with distance $l/2$, $-l/2$, and connecting the two sets of points; (4) the target strip is completed by connecting the end points of the two offset curves. Then, we re-sample the target strip so that it has the same number of vertices as the original block and each vertex has the same arc length parameter on the corresponding polyline. See Fig. 9 for an example.

It is easy to see that after strip fitting, a mesh vertex, $\mathbf{v}_i$, has a preferable location, $\bar{\mathbf{v}}_i^j$, on the target strip, $\bar{b}_j$. Suppose that $\mathbf{v}_i$ has multiple incident blocks. The preferable location, $\bar{\mathbf{v}}_i$, is the weighted average from all the target strips:

$$\bar{\mathbf{v}}_i = \sum_j w_j \bar{\mathbf{v}}_i^j, \tag{4}$$

where $\sum_j w_j = 1$. We use equal weights in our test. Finally, the regularity term is defined as the sum of squared distances to the preferable locations of the mesh vertices:

$$f_{shape} = \sum_i (\mathbf{v}_i - \bar{\mathbf{v}}_i)^2. \tag{5}$$

Fairness. We extract all roads, r_j, not fixed by boundary constraints. For each road, the fairness of a road vertex, $\mathbf{v}_i$, is measured by the squared second-order difference of three consecutive vertices, i.e., $(\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1})^2$. The total fairness term of the road mesh is:

$$f_{fair} = \sum_{r_j} \sum_{\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1} \in r_j} (\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1})^2. \tag{6}$$

Closeness. The closeness is defined as the sum of the squared distances to the original mesh vertices, $\mathbf{v}_i^0$, i.e.,

$$f_{close} = \sum_i (\mathbf{v}_i - \mathbf{v}_i^0)^2. \tag{7}$$

Optimization: We fix all mesh vertices defined in the input (e.g., on the boundary), except the ones that are road intersections (i.e.,

vertex valence > 2). The location of such road intersections can slide on the boundary without crossing neighboring boundary vertices. In each iteration, the energy function has a quadratic form w.r.t. the unknown vertex locations. Thus, the closed-form global optimum can be computed by solving a sparse linear system. We use the CHOLMOD sparse solver [Chen et al. 2008b] in our implementation. We then update the vertex locations, re-fit target strips, and iterate a fixed number of times (10 times for the results shown in the paper). After road mesh optimization, we hierarchically remap the same templates. Fig. 10 presents an illustration of the effect of the global optimization.

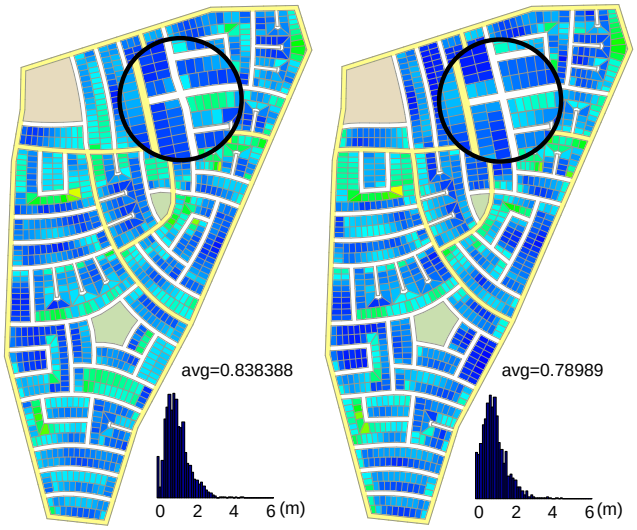

Figure 10: *Initial layout (left) and optimized layout (right) with parcels color-coded by the registration distance from the ideal parcel.*

7 Quality Metrics

We define several quality metrics to guide the layout design and evaluate the results. These quality metrics can be divided into four categories.

Land use metrics measure the land usage of the basic elements of the layout in different categories. We use the percentage of land used for streets (l_s), residential parcels (l_r), parks (l_p), and other uses including commercial (l_o).

Block metrics measure the properties of the blocks: 1) The number of blocks, b_n, per km^2; 2) the maximal boundary length of a block, b_{max}; 3) the minimal boundary length, b_{min}, of a block; 4) the average boundary length, b_{avg}, of all of the blocks.

Street metrics measure the properties of the street graph. Each of these metrics are densities, expressed per km^2: 1) the street length, s_l; 2) the number of street intersections, s_i; 3) the number of dead-end streets, s_d.

Parcel metrics measures the geometric quality of the parcel shapes: 1) the number of parcels, p_n, per km^2; 2) the average angle deviation from 90 degrees, p_{angle}, for all parcel corners; 3) the average area distortion w.r.t. the parcel's minimum bounding rectangle, p_{area}; 4) the average area distortion w.r.t. the ideal parcel shapes, p_{ideal} (measured for our results only); 5) the average similarity distortion w.r.t. to the ideal parcel shapes, $p_{similarity}$. This is computed as the average registration distance to the ideal parcel (also measured for our results only).

The land use and street metrics are adapted from the literature, e.g., [Southworth and Owens 1993]. The parcel metrics and block metrics stem from our own analysis. A parcel's shape should facilitate easy laying out of house and garden which suggests a rectangle as the optimal shape. We try to capture the usefulness of a parcel with p_{angle} and p_{area}. The block metrics are related to the accessibility of the parcels. If the boundary length of a block is too small, it means that too many streets are used to access it. If the boundary length is too high, it will take too long to walk (or drive) around and reach the other side. We also use and display other information, such as the total number of parcels, a histogram of parcel sizes, and a histogram of parcel deformations. We currently exclude distance-based considerations, e.g., average walking distances of residential parcels to parks and driving distances to reach parcels from the main roads.

8 Results

We show several qualitative and quantitative results and provide a discussion of important aspects of our system. All boundary constraints were digitized from real-world examples.

Comparisons: We compared our work with real-world layouts and state-of-the-art algorithms for generating street and parcel layouts. For this comparison, we selected regions dominated by single family homes and we only used a small subset of templates for these types of parcels. Please note that we only compute a single automatic layout for our result. Instead of random streamline selection, we always use the one with the highest quality score. Map: real-world layouts from internet mapping websites. CE1: results generated with CityEngine. CityEngine uses segment-based street growing similar to Parish and Müller [2001] and Weber et al. [2009] and a parceling algorithm from Vanegas et al. [2012]. We set the parameters of CityEngine to best match our street width and parcel size. CE2: our street layout together with the parceling algorithm from CityEngine. The results of the comparison for three regions are listed in Table 1 and one comparison for the Salisbury example is shown in Fig. 11 (see the supplemental materials for the other two). In these three generated results we aim for a parcel length of 30 m, parcel width of 18 m, and a street width of 15 m. The current focus of our work is the parcel quality. Looking at the metrics p_{angle} and p_{area}, we can confirm that we can significantly improve upon existing layouts and CityEngine. CityEngine uses a greedy growing algorithm with no global control of the layout. As a result, artifacts appear when two street networks grow together, or boundary constraints are given. In Fig. 12, we highlight some of these problems using color coding for parcel quality. We can also place more parcels than CityEngine, but it is hard to compare this number to existing layouts as we did not model all existing constraints (topographic, legal, special use, etc.). On the downside, we notice that our street network has some disadvantages compared with real layouts (see Limitations below). Additional comparisons (including all field-related comparisons) are presented in the supplemental materials.

Computation Time: Our experimental platform is a Windows 7 desktop PC with an Intel Xeon CPU clocked at 2.67GHz. Table 2 lists the time required for the results in Table 1.

data set	splitting	backtracking	optimization
Salisbury	28.0	1.7	3.7
Hempstead	14.8	1.1	2.4
Villagepark	28.6	9.4	4.7

Table 2: *Performance statistics. The timings are in seconds.*

Model	Method	Landuse(%)				Block				Street			Parcel		
		streets(l_s)	parcels(l_r)	parks(l_p)	other(l_o)	b_n	b_{max}	b_{avg}	b_{min}	s_l	s_i	s_d	p_n	p_{angle}	p_{area}
Salisbury	Map	28.67	59.12	2.90	9.31	30.92	1192	721	145	13.46	62.9	1.0	879	5.840	1.118
	CE1	34.24	54.23	11.53	0	77.29	1745	329	67	15.01	109.2	0	874	5.012	1.076
	CE2	26.37	58.49	3.71	11.43	55.65	1454	566	326	18.51	107.2	3.1	958	3.681	1.071
	Our	26.37	58.49	3.71	11.43	55.65	1454	566	326	18.51	107.2	3.1	1062	3.188	1.063
Hempstead	Map	27.69	66.93	3.96	1.40	46.38	2191	543	246	14.87	77.7	15.7	1152	7.502	1.112
	CE1	36.15	63.85	0	0	79.22	1034	366	66	16.56	110.6	3.0	965	4.650	1.082
	CE2	27.66	71.10	1.24	0	61.27	1108	559	308	19.47	115.82	2.24	1176	2.188	1.054
	Our	27.66	71.10	1.24	0	61.27	1108	559	308	19.47	115.82	2.24	1271	2.136	1.042
Villagepark	Map	29.10	46.26	9.22	16.42	22.14	7468	1025	184	12.77	41.5	8.8	577	4.932	1.102
	CE1	31.57	59.83	8.60	0	76.57	1100	349	47	27.37	107.0	1.8	986	4.321	1.076
	CE2	27.78	70.04	2.18	0	62.74	955	550	290	19.64	117.2	6.0	1094	2.874	1.058
	Our	27.78	70.04	2.18	0	62.74	955	550	290	19.64	117.2	6.0	1282	2.266	1.045

Table 1: *Result comparison with real-world maps and CityEngine. CE1 uses the same boundary input to generate layout in CityEngine and CE2 uses our streets to generate parcels in CityEngine.*

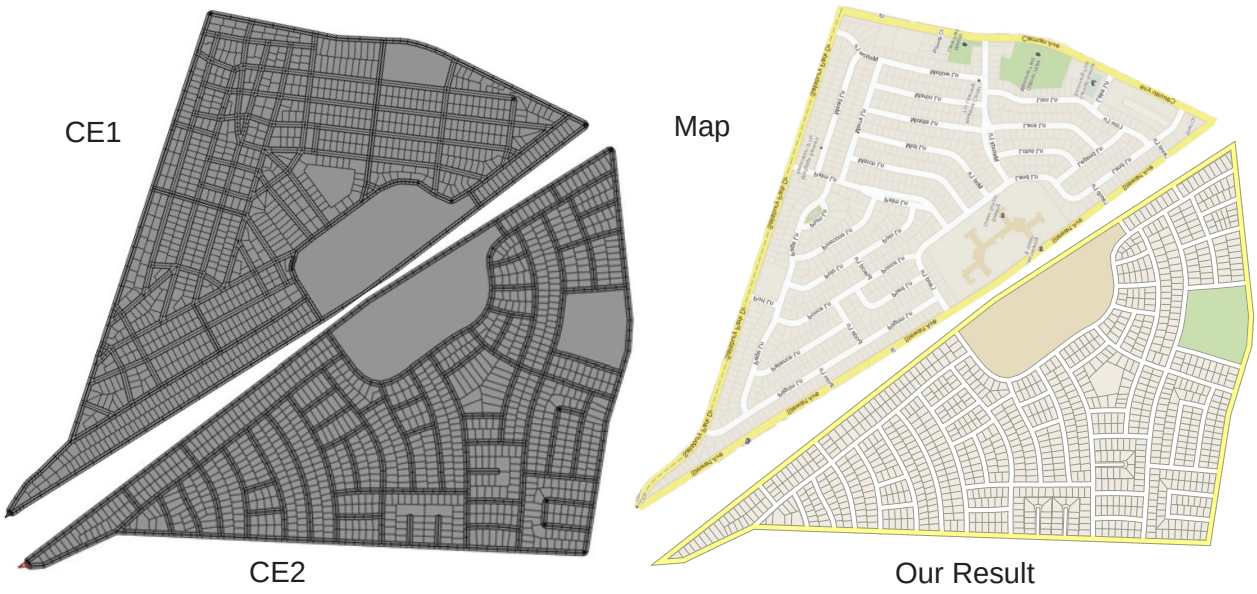

Figure 11: *A comparison of our results to CityEngine (CE1), our street network and CityEngine parcels (CE2), and the real-world layout (Map) for Salisbury. Results for CE1 and Map have been flipped.*

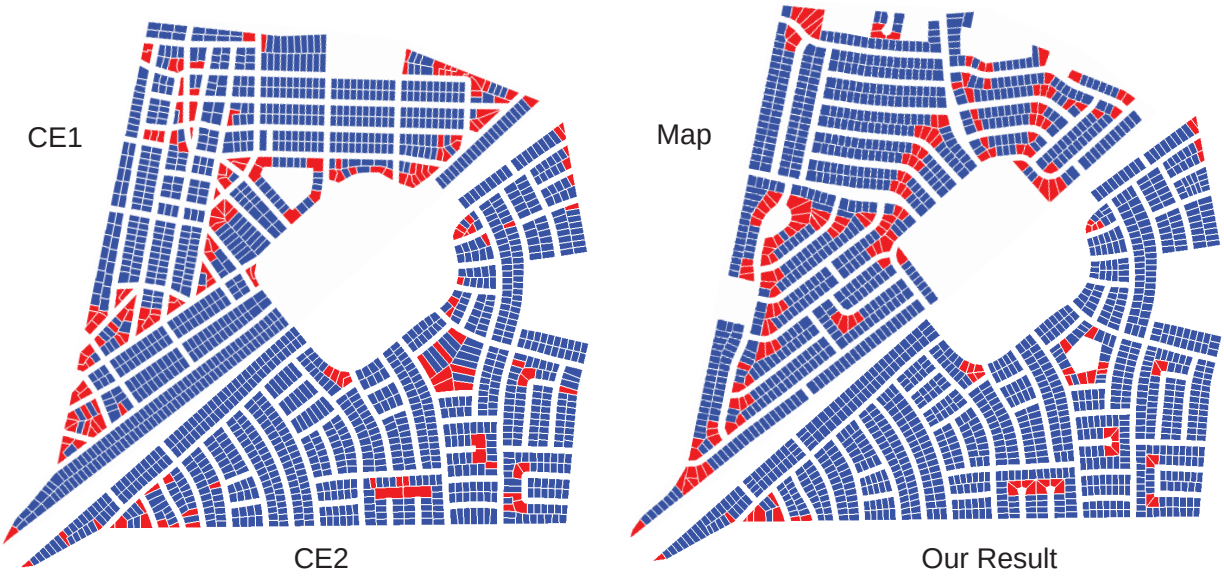

Figure 12: *Visualization of good (blue) and undesirable (red) parcels. An undesirable parcel has $p_{angle} > 15$ or $p_{area} > 1.2$. The number of undesirable parcels: CE1 (119), Map (138), CE2 (57) and our result (48).*

Figure 13: *A layout combining commercial areas, a school, single family parcels, townhouses, and apartments.*

Template Variety: In general, we can use a larger variety of templates and generate patterns that cannot be modeled by existing state-of-the-art methods. In Fig. 1 and Fig. 13 we show results with additional templates for townhouses and apartment buildings. Fig. 14 shows design variations generated by using different subsets of single family house templates. Finally, our framework is also suitable to model details of urban layouts, such as garden designs (see Fig. 15, video, and supplemental materials for the used templates). A gallery of four additional examples is shown in Fig. 16.

Design Variations: Our system can be used to generate different designs for one boundary by modifying the input constraints and using interactive editing (See Fig. 17). We use the following number of interactions for each of the three layouts: 2/2 (left), 2/1 (middle), and 1/1 (right). The former is the number of deleted subregions; the latter is for land use reassignment. In the supplemental materials and video, we show example variations generated by automatic computation for the same input constraints. In these materials, we also show layouts with different parcel sizes for one family houses, example input constraints other than major roads, and examples with different subsets of templates.

Limitations and Future Work: The major limitations of our work are related to transportation and the street network. Our comparison to existing layouts showed that our network might be more efficient, because our blocks are smaller, although the total lengths of roads and the number of intersections are higher (see Table 1). While this is partially a design trade-off, we do not have sufficient control over the street network. This area of design was also identified by our collaborators in planning as the most important area of future work. We were encouraged to consider elements, such as the generated traffic demand, different street designs, bike lanes, traffic safety, and public parking. For example, it would be interesting if the framework could control the number of T-junctions or minimize through traffic on most but a few selected roads. Additionally, we could incorporate the strategic placement of public places, parks, and community life in the optimization. Currently, the placement is done as a pre-process or a post-process and not a part of the optimization itself. Also, it would be interesting to consider lighting and heat generation by the sun in future work.

Figure 15: *Garden layouts, including bushes, trees, driveways, swimming pools, and building footprints can also be modeled in our framework (see the related templates in the supplemental material).*

9 Conclusions

We presented a solution for the geometric layout optimization of streets and parcels for a new development or a virtual city. Our results show that we can achieve higher quality layouts than can previous work. We believe that this work is important for two reasons. First, urban layout planning has a tremendous impact in everyday life and still currently almost no geometric optimization tools are available in this domain. Second, our layouts can be seen as a specific type of hierarchical quad mesh and we believe that their study is interesting from a geometry processing view point.

Acknowledgements. We are grateful to Helmut Pottmann for his many useful suggestions at various stages of the work; Mohamed Shalaby for valuable discussions, his help with Fig. 2 and the design of the garden templates; and the anonymous reviewers for their helpful comments. We thank Pascal Müller for providing the CityEngine results; Yoshihiro Kobayashi for the 3D rendering of the teaser image; Charlotte Rakhit for the video voiceover; Virginia Unkefer for carefully proofreading the paper; and John Peponis and Lars Hesselgren for discussions on urban planning and layout quality metrics. This project was funded by KAUST, NVIDIA, Google, and the National Science Foundation.

References

ALIAGA, D., VANEGAS, C., AND BENES, B. 2008. Interactive Example-Based Urban Layout Synthesis. *ACM Trans. on Graph. 27*, 5.

ALIAGA, D. G., BENEŠ, B., VANEGAS, C. A., AND ANDRYSCO, N. 2008. Interactive Reconfiguration of Urban Layouts. *IEEE Computer Graphics and Applications 28*, 3, 38–47.

ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LÉVY, B., AND DESBRUN, M. 2003. Anisotropic polygonal remeshing. *ACM Trans. on Graph. 22*, 3.

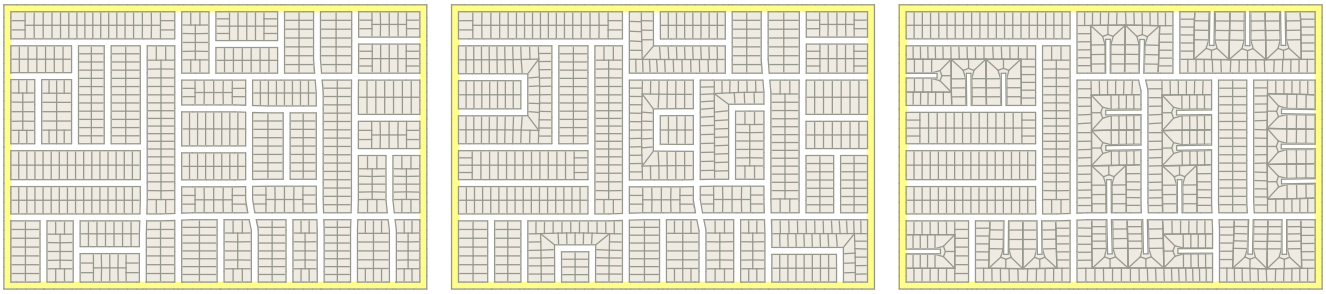

Figure 14: *Multiple layouts generated by using different subsets of templates.*

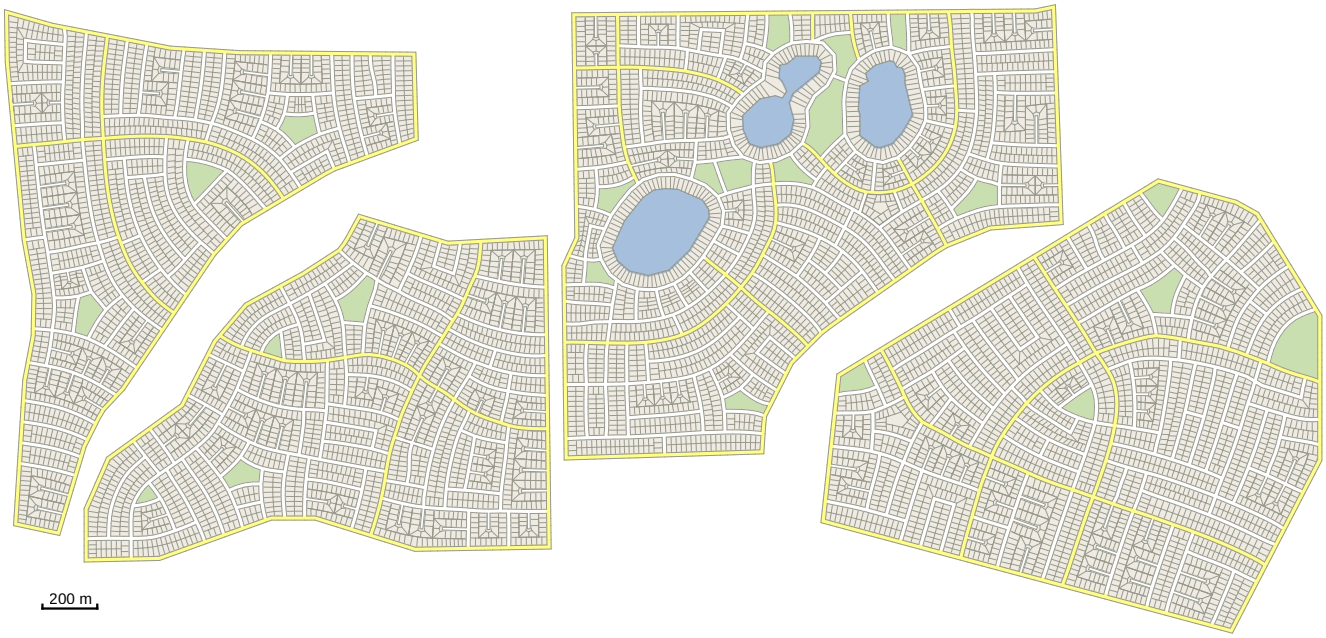

Figure 16: *Various automatic results generated by our framework. Initial constraints are shown in yellow. While we can solve all regions with high quality, the third layout from the left has too many parks. For this layout, interactive post-processing should be used to change some parks to residential areas.*

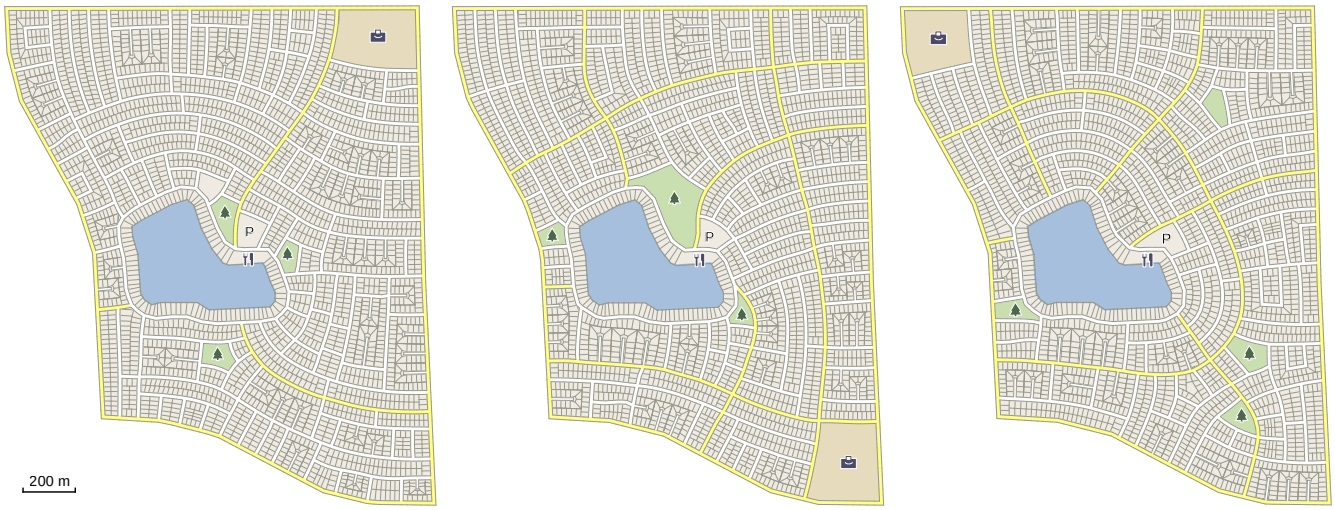

Figure 17: *Three design variations for the same region starting from different user constraints (highlighted by the yellow roads).*

BOUAZIZ, S., DEUSS, M., SCHWARTZBURG, Y., WEISE, T., AND PAULY, M. 2012. Shape-up: Shaping discrete geometry with projections. *Computer Graphics Forum 31*, 5.

CHEN, G., ESCH, G., WONKA, P., MÜLLER, P., AND ZHANG, E. 2008. Interactive procedural street modeling. *ACM Trans. on Graph. 27*, 3, 103:1–9.

CHEN, Y., DAVIS, T. A., HAGER, W. W., AND RAJAMANICKAM, S. 2008. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw. 35*, 3, 22:1–22:14.

CIVIL3D. 2013. http://www.autodesk.com/products/autodesk-autocad-civil-3d.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. on Graph. 23*, 3.

GALIN, E., PEYTAVIE, A., GUÉRIN, E., AND BENES, B. 2011. Authoring hierarchical road networks. *Computer Graphics Forum 29*, 7, 2021–2030.

KIM, J., AND PELLACINI, F. 2002. Jigsaw image mosaics. *ACM Trans. on Graph. 21*, 3, 657–664.

LIPP, M., SCHERZER, D., WONKA, P., AND WIMMER, M. 2011. Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum 30*, 2, 345–354.

LIU, L., ZHANG, L., XU, Y., GOTSMAN, C., AND GORTLER, S. J. 2008. A local/global approach to mesh parameterization. *Computer Graphics Forum 27*, 5, 1495–1504.

MARÉCHAL, N., GUÉRIN, E., GALIN, E., MERILLOU, S., AND MRILLOU, N. 2010. Procedural generation of roads. *Computer Graphics Forum 29*, 2, 429–438.

MARSHALL, S. 2005. *Streets & Pattern*. Spon press, New York.

MERRELL, P., SCHKUFZA, E., AND KOLTUN, V. 2010. Computer-generated residential building layouts. *ACM Trans. on Graph. 29*, 6, 181:1–181:12.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Trans. on Graph. 25*, 3, 614–623.

PACZKOWSKI, P., KIM, M. H., MORVAN, Y., DORSEY, J., RUSHMEIER, H., AND O'SULLIVAN, C. 2008. Insitu: Sketching Architectural Designs in Context. *ACM Trans. on Graph. 30*, 6, 182:1–10.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of SIGGRAPH 2001*, 301–308.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.

ROBINSON, R., LONG-JR., B. J., OSTERGAARD-JR., P., AND LEVINE, K. 2004. *The Architectural Pattern Book: A Tool for Building Great Neighborhoods*. W. W. Norton & Company Inc., New York.

SITEOPS. 2013. http://www.siteops.com.

SOUTHWORTH, M., AND BEN-JOSEPH, E. 2003. *Streets and the Shaping of Towns and Cities*. Island Press, Wasington DC.

SOUTHWORTH, M., AND OWENS, P. M. 1993. The evolving metropolis: Studies of community, neighborhood, and street form at the urban edge. *Journal of the American Planning Association 59*, 3, 271–287.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. on Graph. 30*, 2, 11:1–11:14.

VANEGAS, C. A., ALIAGA, D. G., BENEŠ, B., AND WADDELL, P. A. 2009. Interactive design of urban spaces using geometrical and behavioral modeling. *ACM Trans. on Graph. 28*, 5.

VANEGAS, C. A., ALIAGA, D. G., WONKA, P., MÜLLER, P., WADDELL, P., AND WATSON, B. 2010. Modelling the appearance and behaviour of urban spaces. *Computer Graphics Forum 29*, 1, 25–42.

VANEGAS, C. A., KELLY, T., WEBER, B., HALATSCH, J., ALIAGA, D., AND MÜLLER, P. 2012. Procedural generation of parcels in urban modeling. *Computer Graphics Forum 31*, 2.

WEBER, B., MÜLLER, P., WONKA, P., AND GROSS, M. H. 2009. Interactive geometric simulation of 4d cities. *Computer Graphics Forum 28*, 2, 481–492.

WONKA, P., WIMMER, M., SILLION, F. X., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. on Graph. 22*, 3, 669–677.