2008

Interactive Reconfiguration of Urban Layouts

Daniel G. Aliaga
Purdue University, aliaga@cs.purdue.edu

Bedrich Benes

Carlos A. Vanegas

Nathan Andrysco

Report Number:
08-003

INTERACTIVE RECONFIGURATION OF URBAN LAYOUTS

Daniel G. Aliaga
Bedrich Benes
Carlos A. Vanegas
Nathan Andrysco

CSD TR #08-003
January 2008

Interactive Reconfiguration of Urban Layouts

Daniel G. Aliaga* aliaga@cs.purdue.edu
Bedřich Beneš+ bbenes@purdue.edu
Carlos A. Vanegas* cvanegas@cs.purdue.edu
Nathan Andrysco* nandrysc@cs.purdue.edu

*Department of Computer Science
+Department of Computer Graphics Technology
Purdue University

Abstract

The ability to create and edit a model of a large-scale city is necessary for a variety of applications. Although the layout of the urban space is captured as images, it consists of a complex collection of man-made structures arranged in parcels, city blocks, and neighborhoods. Editing the content as unstructured images yields undesirable results. However, most GIS maintain and provide digital records of metadata such as road network, land use, parcel boundaries, building type, water/sewage pipes and power lines that can be used as a starting point to infer and manipulate higher-level structure. We describe an editor for interactive reconfiguration of city layouts, which provides tools to expand, scale, replace and move parcels and blocks, while efficiently exploiting their connectivity and zoning. Our results include applying the system on several cities with different urban layout by sequentially applying transformations.

Keywords: urban reconstruction, image synthesis, procedural modeling, texture synthesis, geometric constraints, style.

1. Introduction

Modeling and visualizing large urban environments is a great challenge for computer graphics. Creating, extending, and changing a model of a large-scale urban environment is useful for a variety of applications. For example, it enables urban planning applications to simulate changes to city layouts or to newly proposed neighborhoods, to create hypothetical views of an urban area after applying development and growth algorithms, to show microclimate visualizations, to provide road planners with aerial views of new street networks, and to allow architects to see the results of using common building blocks to design a new city layout. Emergency response simulations can train personnel in current and speculative urban layouts, including planning evacuation routes for various catastrophes, and can suggest emergency deployments of communication networks, resources, and policing. Finally, previously-existing cities for which only partial information is known can be visualized and analyzed. The structural characteristics of a historically significant region are important aspects of urban visualization.

When the layout of an urban space is captured by images (e.g., satellite and aerial photographs), the structural information is lost and it cannot be further exploited. An urban layout consists of a complex collection of man-made structures arranged in roads, parcels, city blocks, and neighborhoods. On the one hand the content can be treated as unstructured images and recently

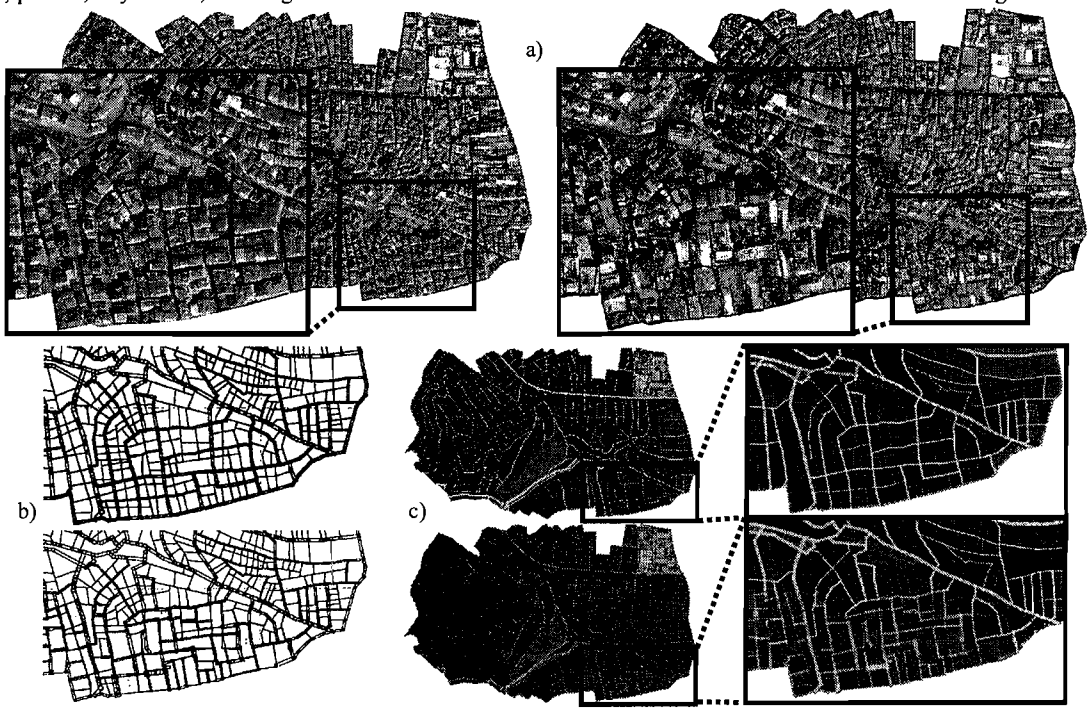


Figure 1. *(a) Satellite image of the original (left) and modified (right) layout of Istanbul. (b) The original topology of the affected area (top) has been modified (bottom). c) Selected tiles have been copied from the industrial zone of the city (red) and relocated to residential area (blue).*

developed image processing tools can be used to change the images (e.g., [1][2]) or to generate similar imagery synthetically (e.g., [3]). However, this yields to undesirable results, including deformations and structural discontinuities. On the other hand, the structure of the urban space can be assumed to be known, in the form of an explicit or procedural model (e.g., [4]), and be altered by directly changing model parameters. However, knowing all model parameters is challenging and even more so for large urban spaces. Representing an existing urban layout as a set of procedural rules typically involves significant manual labor.

Our inspiration comes from using the records of parcel boundaries, roads, and other metadata maintained by counties in digital form (e.g., such as that used by Google Maps) as a starting point for inferring higher-level structure automatically and for supporting urban layout editing. Moreover, urban spaces exhibit a significant degree of repetition on a global scale and individuality in local detail. This implies that the general "style" of a city can be extracted at a high-level and the resemblance between similarly-classified elements of the urban layout can be used to improve the editing and reconfiguration process.

Our approach is to decompose an urban layout into a collection of adjacent tiles, separated by road or parcel boundaries, to cluster tiles based on their similarity, and to support a variety of editing and reconfiguration tools that minimize the deformation of the tiles and minimize the distortion of the grid of tiles. The general problem of packing together arbitrary shapes is an NP-complete problem, but in our work we propose a group of heuristics that enable efficient solutions to this inherently complex geometric problem as applied to the specific case of the urban layout editing. We support intuitive notions such as stretching an urban layout to occupy a new space, partitioning and reassembling an urban map, and copying and replacing parts of the map in one area with parts from other areas. All the while, our system maintains a similarly connected street network and reuses the existing tiles of the urban layout with minimal or no deformations. We have applied our system to the editing and reconfiguration of portions of various real world cities as is seen in Figure 1 and show various examples of modifications that are difficult to achieve by previous techniques such as topology-changing cut and paste operations, moving areas of urban layout, redesigning roads, etc.

The main contributions of this paper are

- an interactive system for urban layout modification preserving local and global features and the style of the original layout,
- a constraint system for urban layout editing and reconfiguration, and
- derivation and classification of the high level urban structural information from a set of images and low level structural data and its application to urban layout editing.

2. Related Work

Our work situates itself between image-processing and procedural modeling and borrows concepts from computational geometry. Recently, several image processing algorithms have been proposed to enable editing images. For example, Avidan and Shamir [1] describe a method to resize and retarget images. Fang and Hart [2] propose a technique to re-synthesize image texture when reshaping a textured portion of an image and reduce unwanted deformations. These methods provide powerful tools for image editing but are not suitable for images with highly-structured information. An urban layout consists of building contours, road networks, and local details that cannot be treated the same as a patch of grass or sand. Important boundaries, angular relationships, and logical connectivity must be maintained and tiles should not be deformed.

Texture and image synthesis provide another group of techniques for image processing. In general, such synthesis algorithms support creating new image content similar to a provided sample. For example, Zhou et al. [3] provide a method to generate synthetic terrain following the style of a source image. However, there is no facility to modify the content of the source image; instead, they propose a mechanism to generate a completely different terrain. Hertzmann et al. [5] use a source image and a color coding combined with a target color coding to produce an image analogous to the source. They demonstrated rearranged terrain layouts as well as rearranged urban spaces. Nevertheless, the newly generated urban layouts recombined the patches by treating them as collections of pixels and used blending to yield new images. Thus, upon closer inspection, it is observed that the connectivity and close-up details are not maintained.

Procedural modeling methods have the advantage of exhibiting a high degree of detail amplification; e.g., significant details can be synthetically generated or modified using only a small number of attributes of rules. However, since a small change in the parameters can cause massive changes in the resulting model, it is difficult to use these systems and a great experience and insight is required to provide good results. Nonetheless, procedural techniques have been demonstrated in a variety of contexts including urban spaces. For instance, one of the first procedural techniques describing 3D city generation is found in [4]. Their City Engine system takes images of existing areas to generate road maps using L-systems. The road map is then divided into lots that are filled by buildings provided by another stochastic parametric L-system. More recently, the generation of buildings has been extended to fully procedural modeling of buildings [6] and to a hybrid procedural and computer-vision approach to generating building facades [7]. The methods for generating urban layouts require either the careful hand-crafting of a set of procedural rules or significant a priori information about the urban environment. Furthermore, they do not necessarily reproduce or start-with existing urban layouts. The challenge for procedural methods is the automatic generation of rules from a given input data. Our system presents a step in this direction as it derives high-level structural information from a given input data that is reused in the process of urban layout editing.

Also related to our system are several image and model reconfiguration methods. For example, Aliaga et al. [8] uses partially annotated images of buildings (analogous to our mapping data input) to edit and reconfigure buildings. This system takes uncalibrated images of a building, performs a photogrammetric reconstruction with user assistance, and then infers the parameters for a procedural model of the captured building. Interactive modifications, such as resizing and copy/paste are easily performed while maintaining the style of the building. Kim and Pellacini [9] introduce an image mosaic algorithm whereby arbitrary images of objects are composed together to form the final picture also of arbitrary shape. Our approach also has some similarity with parametric computer-aided-design (CAD) models [10], where an object is defined by a parameterized model whereby each parameter has a valid range. The set of valid ranges of all parameters defines the potential shapes of the object. The parameters of the urban layout are the location of the tiles and the width and orientation of the numerous streets. In general, our work draws inspiration from these papers and takes the concept to urban layouts. We enable similar operations but for much larger physical spaces and for generally rigid tiles of a functional city. Unlike parametric CAD, our method supports larger changes by adopting the strategy of joining (or removing) a group of tiles and replacing it with a similar-function tile of larger (or smaller) size. This provides significant more flexibility. To the contrary of the procedural models, once provided with the input data, our system is fully automatic and interactive.

3. Urban Layout Editing

Our system supports interactively editing an urban layout and visualizing the results. An urban layout of a city is formed by adjacent tiles (e.g., parcels of land), interconnected by a network of streets, and together forming a collection of neighborhoods. Our system supports *moving* (translate, rotate), *copying*, *cutting*, and *pasting* (group of) tiles in order to edit or create a new city arrangement. In the context of our system, moving implies selecting one or more tiles and displacing them. Tiles that must be excessively deformed to accommodate the overall changes are automatically replaced with other tiles that better fit in the deformed space and represent an equivalent urban structure. Copying, cutting, and pasting refer to the process of selecting a collection of tiles and placing them in another part of the city. The tiles in the source location may be copied or removed while the tiles in the destination location are replaced with the new ones. After any editing operation, the system attempts to accommodate the position and orientation of nearby tiles to the changes and maintain a consistent layout.

The input to our system is the same as that used by mapping applications: an array of high-resolution images, a set of lines identifying street layout, and per-tile information. This information is made available from geographical information systems (GIS) and pertinent features can be extracted from these data sets using an open source tool such as the Geospatial Data Abstraction Library (GDAL) or the Shapefile C Library. Each street or parcel boundary is defined by a polyline. In the case of streets, each polyline segment has an associated street width. For parcel boundaries (i.e., logical land boundaries), the polyline has no width parameter. The collection of intersecting polylines partitions the plane into adjacent polygons or tiles; a tile fully surrounded by street-polylines corresponds to a "block" and a tile with at least one street edge corresponds to a "parcel" (thus, by definition, there are no streets inside a block or parcel). Although not strictly enforced, we assume the use of an egress rule which implies that each tile has direct access to a segment of a street. Figure 2a shows an example urban layout and Figure 2b contains a diagram of the corresponding interconnected set of tiles, streets, and boundaries. In addition, each tile is annotated

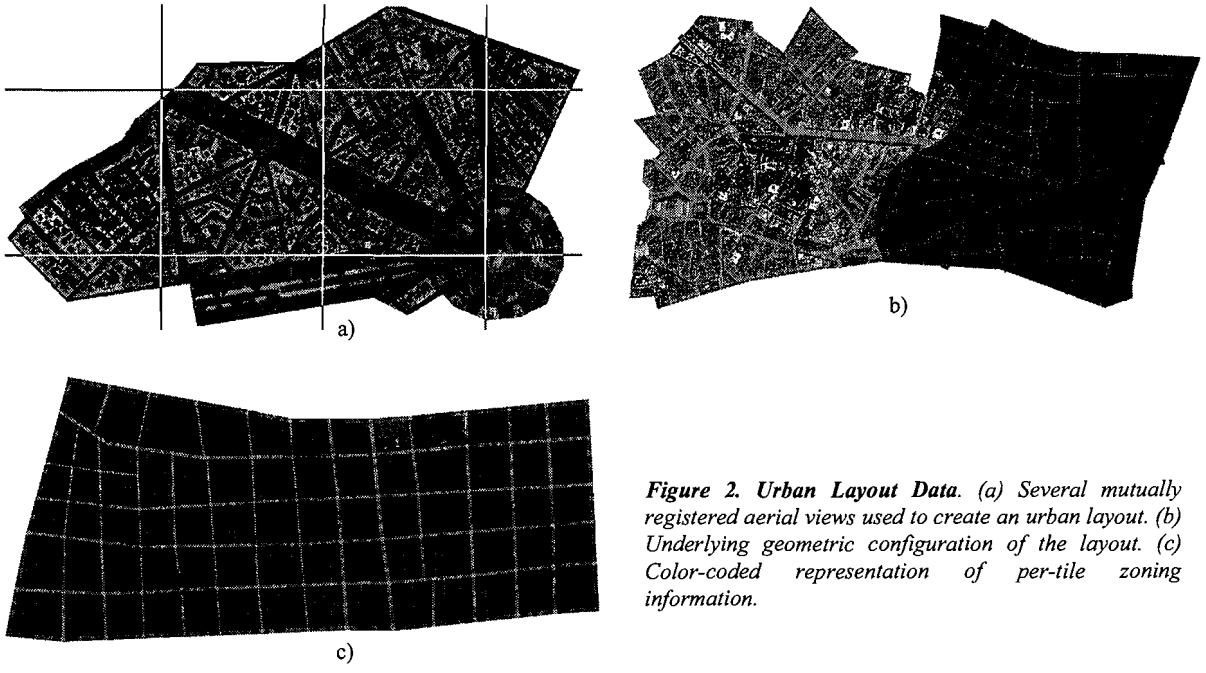


Figure 2. Urban Layout Data. *(a) Several mutually registered aerial views used to create an urban layout. (b) Underlying geometric configuration of the layout. (c) Color-coded representation of per-tile zoning information.*

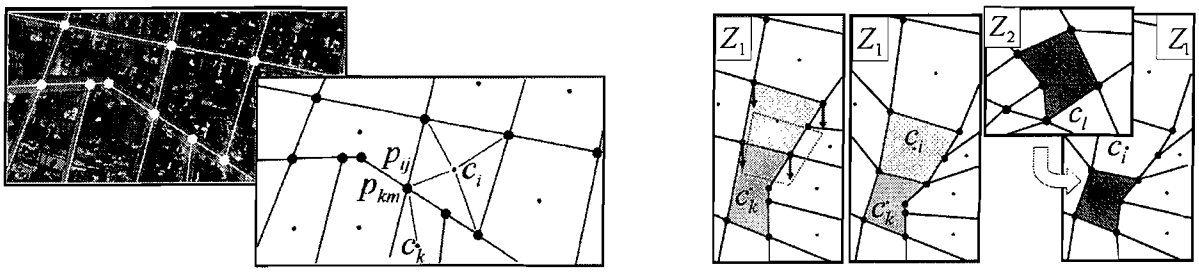


Figure 3. Layout Management. *(a) We show a set of polylines that partition an aerial image into tiles and a depiction of the quantities used by the constraint solver; point* p_{ij} *is attempted to be kept the same as point* p_{km}. *(b) To demonstrate replace operations, tile* i, *located in a zone* Z_1, *is moved to the south. Neighboring tile* k *is deformed as a result. The system automatically finds the union of tiles* j *and* l *(located in some other zone* Z_2 *of the city) to be a tile that, after a relatively smaller deformation, better fits the deformed space previously occupied by* k.

with its zoning classification. Without loss of generality, we choose a zoning classification of residential, commercial, industrial, park or other. Figure 2c depicts zoning information using color modulation of the tiles.

In order to execute the editing operations, the system performs (1) tile layout management and (2) tile similarity estimation. The layout management component ensures that the result after editing is a valid and sensible street network and that no tile is excessively deformed. To help obtain tiles of similar function (and shape), the similarity estimation component places each tile into one of a set of clusters and then compares tiles within each cluster. For example, houses are grouped together; then similar houses are further placed into subgroups, etc. In the following two sections we describe these two components in more detail.

4. Layout Management

Our layout management component uses a linear system of constraints that during editing simultaneously attempts to keep streets unchanged and to minimize the deformation of the tiles. While there are multiple approaches to managing an urban layout editing process, we focus on supporting the easy editing of an existing layout using only GIS-like data and a minimal user input. Furthermore, in order to prevent having to generate new and synthetic imagery, we reuse the imagery of the existing overhead views. This leads us to an approach whereby we must rearrange the original, arbitrarily-shaped tiles, and maintain a validly-connected and similarly-styled street layout. In order to accommodate changes to the layout, we observe that while the tiles are desired to be kept mostly rigid, there is some flexibility in the width and relative orientation of the segments of the street network. Thus, if a particular urban tile is to be inserted into another part of the layout, we accommodate the additional space needed for the insertion by mildly changing the width and angle of all nearby streets. However, if the required change is too large, we resort to small deformations of the tiles and eventually to collapsing (or removing) tiles from the layout so as to support the operation.

4.1 Constraint Solver

Our solver attempts to find a planar transformation for each tile that best accommodates the changes caused by the editing operations. Conceptually, we wish to keep each tile exactly touching its neighboring tiles and also exactly abutting the segment of road it touches. If an editing operation causes a tile to be displaced, then we seek to displace the neighboring tiles in such a fashion as to keep the tiles from overlapping and to keep the affected streets of the same width and size. Geometrically, tiles are polygons that partition the urban layout and thus each tile shares its vertices with the neighboring tiles. If a tile were to move, the vertices of the neighboring tiles need to move as well in order to maintain the connectivity of the tile layout. To attempt to keep tiles adjacent, we setup a collection of constraints whose nominal value is zero and we use an optimization to discover the transformation to apply to each tile in order to best satisfy the constraints. To formulate the constraints, we first place all tiles into a graph. A node in the graph corresponds to a tile. An edge in the graph corresponds to a vertex that is in common between two tiles. Each graph edge effectively becomes a constraint to keep abutting tiles exactly touching.

The constraint solver evaluates two types of error. First, *gap error* is defined to be the distance by which corresponding vertices from two neighboring tiles do not overlap. The common boundary between two tiles can be either a street or a logical property boundary. If the former, then there is some "play room" for the solution because the street can mildly change width without severely affecting the end result. For the latter, generating overlapping tiles across a property line might visually cause buildings to overlap, for example. Hence, the gap errors across streets are penalized less than those across property boundaries. The flexibility provided by the street widths allows us to re-position all tiles for a given editing operation without necessarily deforming them. Nevertheless, to support larger changes, we might need to deform the tiles using non-rigid transformations as well. This brings us the second type of error, *tile deformation error,* which is the amount by which a tile must be deformed to better fit into the destination location. In our system, we restrict ourselves to deformations resulting from affine transformations, because they are linear and provide a large space of editing operations. The constraint solver finds a global solution for the tiles that minimizes a weighted sum of the gap error and deformation error. Figure 3a depicts the constraint setup.

We setup a system of equations representing the constraints. A nonlinear system of equations would require significantly more computation time and potentially suffer from lack of stability. Hence, we have devised a linear system of equations to represent the constraints for N tiles. The equations divide into two major groups: i) the first group expresses the desire to keep the total gap error, as previously defined, at zero; ii) the second group expresses the desire to avoid having tile deformations, i.e., a deformation error of zero. A single global weighting parameter allows the user to choose between tile deformations that are rigid transformations or mild affine tile deformations.

The equations for minimizing gap error are:

$$\sum_i \sum_k \sum_j \sum_m \left\| \left(c_i + t_i + M_i d_{ij}\right) - \left(c_k + t_k + M_k d_{km}\right) \right\| = 0 \qquad (1)$$

where c_i and c_k are the centroids of the tiles i and k; t_i and t_k are the unknown tile translation vectors; and M_i, M_k are unknown 2x2 tile transformation matrices; $d_{ij} = p_{ij}-c_i$ and $d_{km} = p_{km}-c_k$ where p_{ij} and p_{km} are points j and m on the perimeter of the tile i and k. Perfectly abutting tiles would have the property that $p_{ij}=p_{km}$. When they do not agree, the computed translations and the matrix transformation must compensate as best as possible. To setup a linear system of equations for expression (1), we expand it to

$$\begin{aligned} \sum_i \sum_k \sum_j \sum_m \left(t_{i,x} + M_{i00} d_{ij,x} + M_{i01} d_{ij,y}\right) - \left(c_{k,x} + t_{k,x} + d_{km,x}\right) + \left(c_{i,x} + d_{ij,x}\right) = 0 \\ \sum_i \sum_k \sum_j \sum_m \left(t_{i,y} + M_{i10} d_{ij,x} + M_{i11} d_{ij,y}\right) - \left(c_{k,y} + t_{k,y} + d_{km,y}\right) + \left(c_{i,y} + d_{ij,y}\right) = 0 \end{aligned} \qquad (2)$$

One way to minimize tile deformation error is to keep the matrices M as close as possible to a 2D orthogonal rotation matrix. Thus, our system prefers that the matrices M have the general form $[a\ -b;\ b\ a]$ for arbitrary values of a and b. Thus, to also minimize tile deformation error we add the following additional set of equations:

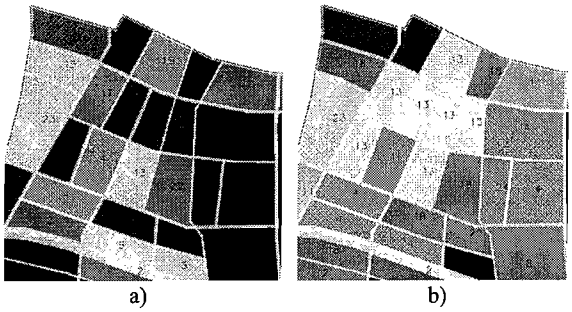
$$\sum_i (M_{i_{00}} - M_{i_{11}}) = 0\ ;\ \sum_i (M_{i_{01}} + M_{i_{10}}) = 0\ . \qquad (3)$$

These two groups of equations can be written as a system of linear equations in the form Ax=b, where x is constructed as an array of $6N$ unknown variables, namely $\{t_{i,x}, t_{i,y}, M_{i_{00}}, M_{i_{01}}, M_{i_{10}}, M_{i_{11}}\}$. The number of columns of A is $6N$ and the number of rows depends on the number of vertices per tile. Since all tiles have at least 3 vertices, there are at least $6N$ equations from the equation set (2) and $2N$ equations from the equation set (3). This yields a typically over-constrained and sparse linear system of equations that can be efficiently solved using linear least squares.

4.2 Editing Operations

Given the aforementioned graph and constraint system, editing operations amount to changing the graph and re-computing the transformation of the tiles so as to best satisfy the constraints. In general, editing operations fall into two categories: those preserving the graph topology and those changing the graph topology. An example of a topology-preserving operation is an interactive displacement of a set of tiles to a new location. The system adjusts the transformation applied to the nearby tiles so as to best accommodate the move. Gap error and tile deformation error are minimized as per the aforementioned linear constraint solver. Tiles that yield an excessively large gap and/or deformation error are attempted to be replaced by other tiles of similar classification chosen manually or automatically (see next section). Potential replacement tiles are those tiles that are most similar (non-deformed) to the deformed tile. A replacement is performed only if using the candidate tile results in a solution with lower overall error, as defined by the constraint equations. Thus, during editing the deformation error of each tiled is computed and those exceeding a threshold are attempted to be replaced by another tile. After several (or all) tiles exceeding the threshold have been replaced, the constraint solver is executed again yielding a new set of tile transformations. This process repeats in an iterative fashion until no more replacement operations can be carried out. Figure 3b shows an example of a move and replace sequence.

An example of a topology-changing operation is "copy/paste" or "cut/paste" which transfers a part of the urban layout from one location to another one. In this case, an interactively selected group of tiles is placed at a new location by replacing the most similar tiles with the selected tiles. The notion of similarity is used to naïvely find the best mapping between the selected tiles



a) b)

Figure 4. Similarity Clustering. *In the first part of similarity estimation, tiles can be partitioned into, for example, (a) clusters mostly-based on the area of the tile, or (b) clusters mostly-based on tile shape similarity. Subsequently, similarity between tiles in the same cluster is computed using a geometric similarity measure.*

and the destination tiles. Subsets of these computations are pre-computed and stored in a similarity matrix which helps to improve the system performance.

5. Similarity Estimation

An important building block for performing the editing operations is the quantification of the tile similarity. This problem is analogous to the polygon mapping problem (e.g., how to best correspond a n-vertex polygon with another m-vertex polygon). Sander et al. [11] introduce a texture stretch metric to minimize while constructing progressive meshes and Zhang et al. [12] propose a scheme to partition the surface into patches so as to minimize surface parameterization stretch. In our case, the decomposition is given (e.g., roads, parcels, blocks) and thus our focus is on keeping the shape minimally deformed in the context of an urban space. Thus, we setup a penalty function that discourages scaling and shearing of the parcels and permits a controlled amount of deformation of the streets, which follows the simple intuition that roads can twist, bend, and change width. Further, two tiles are considered alike if they serve a similar function in the urban layout and one can be substituted by another one through only a small geometric distortion. Hence, our system determines similarity in two phases. First, tiles are grouped into similarity clusters. Each cluster represents a set of tiles of similar function, style, and approximate shape. Second, a more expensive and accurate metric is used to measure the geometric similarity between tiles of the same cluster. A 2D similarity matrix is computed to represent the further similarity data between tiles in the same cluster.

To generate the first-level of similarity, we use a weighted k-means clustering algorithm to group tiles into clusters of similar function and form. The metadata defining a tile includes its zoning classification, construction date, building type and numerous other characteristics. The general shape is characterized, for instance, by the area, number of vertices, and interior angles of the tile's geometry. Without loss of generality, in our system we define a 5-dimensional vector $S_i=[z,a,n,\mu,\sigma]$ to represent a tile i, where z is the tile's zone (i.e., industrial, residential, commercial, park, or other), a is the area of the tile, n is its number of vertices, μ denotes the mean interior angle, and σ is the interior angle's standard deviation. The number of clusters is specified by the user as a percentage of the total number of tiles (Figure 4).

To generate the second-level of geometric similarity between two tiles (typically of the same cluster), we iterate through a subset of all vertex-to-vertex mappings and estimate a similarity value. For each mapping, we compute the best linear transformation to align the perimeters of the two tiles. The mapping that produces the least tile distortion and smallest perimeter mismatch is used as an indicator of the similarity between the tiles. To explain our method, we define tile A to contain n vertices and tile B to contain m vertices, with $n \geq m$. If both tiles have the same number of vertices, then $O(n)$ possible vertex mappings are tried by sequentially attempting to map the vertices of A to those of B with an offset varying from 0 to n-1. If tile A has more vertices than tile B, each vertex of A is individually removed and the similarity method is recursively called. While this may yield a very large number of potential mappings, in practice the difference between the number of vertices of A and B is small. Furthermore, several trivial-reject heuristics are used to significantly reduce the number of potential mappings: (1) if the removal of a vertex of A causes the area of the tile to be significantly reduced, that branch of potential mappings is ignored; (2) if the sequence of vertices around a pair of tiles of the same of number vertices does not match in the type of the vertex (e.g., street or parcel boundary vertex), the case is ignored.

Our system also finds the similarity between 1-to-many tiles -- this enables comparing, and thus replacing, one tile with multiple tiles (e.g., a large tile replacing a collection of smaller tiles). This computation is performed by recursively merging the smaller of the two tiles with an adjacent tile. The similarity estimation is performed analogously on the merged tiles. The merging process is not performed if the area difference between two tiles is too large and it stops when the merged tile is sufficiently larger than the other tile.

To choose the best mapping between a pair of tiles A and B, we distort tile A to tile B, estimate a measure of the *tile distortion error* $e_{distortion}$, and compute a difference between perimeter shapes. Tile distortion error is computed as a weighted sum of undesired shearing, scaling, and change in aspect ratio, namely $e_{distortion} = k_0 e_{scale} + k_1 e_{aspect} + k_2 e_{shear}$ and

$$e_{\text{scale}} = \left| \frac{\max(\|\mathbf{a}+\mathbf{b}\|, \sqrt{2})}{\min(\|\mathbf{a}+\mathbf{b}\|, \sqrt{2})} - 1 \right|, \qquad e_{\text{aspect}} = \left| \frac{\max(\|\mathbf{a}\|, \|\mathbf{b}\|)}{\min(\|\mathbf{a}\|, \|\mathbf{b}\|)} - 1 \right|, \qquad e_{\text{shear}} = \left| \frac{\max\left(\cos^{-1}(\mathbf{a}\cdot\mathbf{b}), \frac{\pi}{2}\right)}{\min\left(\cos^{-1}(\mathbf{a}\cdot\mathbf{b}), \frac{\pi}{2}\right)} - 1 \right|, \tag{4}$$

where a is the vector (1,0) transformed by the matrix computed for tile A and similarly b is the vector (0,1) transformed by the same matrix, and k_i are weights for establishing the relative importance of the components of the distortion error. While the angles between streets and the angles between the segments of the perimeter are not directly forced to remain unchanged, the inclusion of shearing and aspect ratio into the error metric does indirectly encourage the angular relationships of the original layout to be maintained. The perimeter mismatch error, $e_{perimeter}$, is computed straightforwardly as the summed distance between corresponding tile vertices. The total similarity error, e, is a weighted sum of the two aforementioned error metrics:

$$e=(1-\alpha)e_{distortion} + \alpha e_{perimeter}, \tag{5}$$

where α controls the relative influence of each error. The clustering and tile similarity information can be pre-computed so as to accelerate the performance of the paste and replace operations during interactive editing. To improve the performance of the move operation, a deformed tile that is desired to be replaced is compared against a representative from each cluster (e.g., the tile

City	Streets	Boundaries	Tiles
Buenos Aires	20	440	686
Istanbul	174	1073	2012
Madrid	79	568	802
Paris	33	373	535

Table 1. Information about our example datasets.

closest to the midpoint of the tile's description vector) and then further compared against the members of the cluster of the best-matched representative.

6. Results and Discussion

We have used our system to edit the urban layouts of subsets of several real-world cities. The software system is implemented in C++. All rendering is done using standard OpenGL, GLUT, and GLUI. To improve performance, our system makes a one-time simplification of the graph of tiles in order to reduce the number of constraints. Pairs of vertices that are very near each other are collapsed to one vertex; existing constraints between the affected tiles are merged. In addition, non-corner edges that are approximately collinear with their surrounding edges are removed as well. The similarity clustering and tile-to-tile similarity matrix is also pre-computed and stored to disk.

Our four example datasets are *Buenos Aires, Istanbul, Madrid,* and *Paris*. Each city consists of 9 to 16 1024x1024 images obtained from GoogleMaps and arranged in a 2D grid. Table 1 provides additional information about the cities and their layouts. The clustering and tile similarity information (Section 5) is pre-computed in order to accelerate the performance of paste and replace operations; this preprocessing varies from one to three hours for each of the datasets. During run-time, the tile-similarity information for newly created and significantly deformed tiles is computed on demand. The input data can either be parsed automatically from a GIS database/county-office or input manually. In our prototype system instead of focusing on parsing the mapping files, we used a simple interactive system to specify streets and parcel boundaries.

We demonstrate topology-preserving operations in Figures 5 and 6. We compare the results of bending the layout using a common image processing program to the results obtained by using our method. Figure 5a shows a view of the original urban space. Figure 5b shows the result using our linear constraint solver without any tile deformations and Figure 5c shows a similar result but enabling mild tile deformations. The average gap error in Figure 5c is smaller than in Figure 5b but a small average tile deformation is introduced. Comparatively, Figure 5d demonstrates an image after naïve mesh warping using Adobe Photoshop and without considering any structural information; large deformation error is evident. In Figure 6, we show two more examples of moving operations. In particular, Figures 6a-c depict a sequence of breaking an urban layout into pieces, displacing the pieces outwards, and reassembling a stretched version of the layout. The stretching of the entire layout has been distributed by rigid/affine transformations among hundreds of constituting tiles, and thus the size and aspect ratio of each block present only a very small variation. Figures 6d-f show another example where some parts of Paris are rotated around the Place de la Concorede. Both of these examples are generated interactively and comprise core topology-preserving operations.

The copy/cut and paste operations are depicted in Figure 7. Using the Buenos Aires layout, we replicate an urban structure in

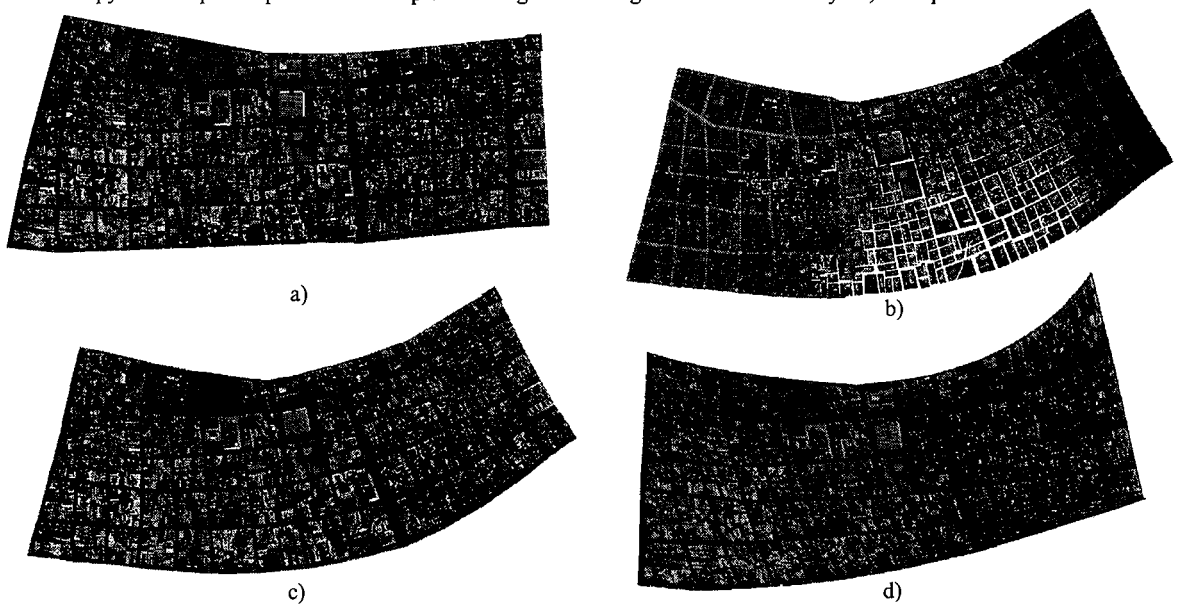


Figure 5. Image Comparison. *(a) A view of the Buenos Aires layout. (b) Layout deformed using our constraint solver with rigid and (c) with mild non-rigid transformations. (d) Layout deformed using a standard image processing program.*

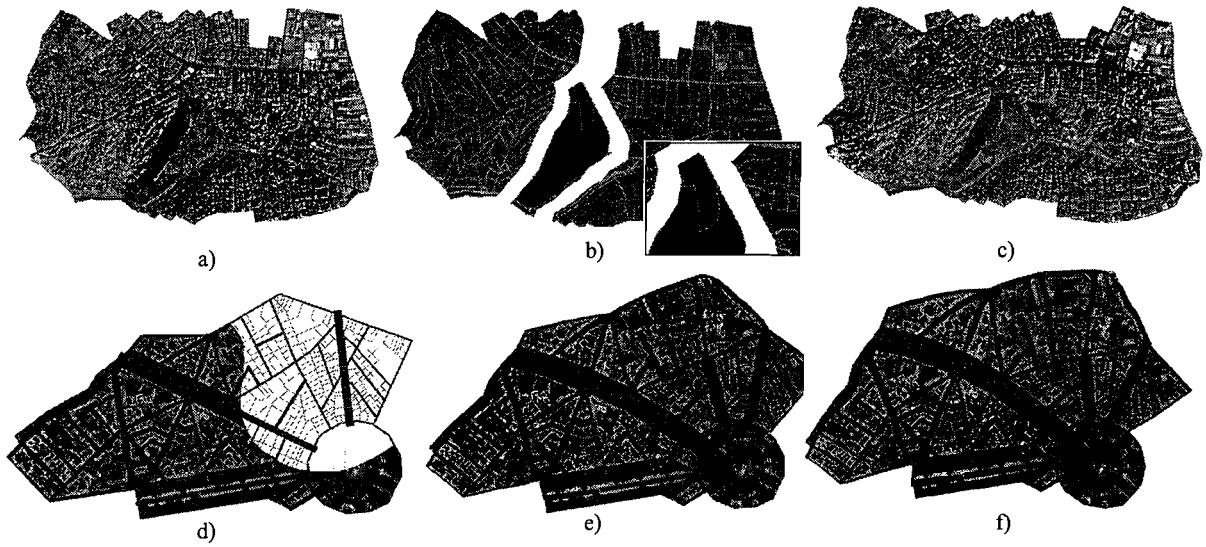


Figure 6. Topology-preserving operations. *(a) An original view of Istanbul, (b) the layout divided into distinct parts, and (c) reassembled with a stretch. (d) A view of Paris (e-f) with parts of the layout rotated by around the Place de la Concorde.*

several locations of the layout. For each placement, the constraint system is re-solved yielding a slightly perturbed layout that accommodates the new tile. Figures 7a-c show the major steps of this operation. Despite placing pieces of different connectivity and different size into the grid, the system is able to reconfigure the tiles and accommodate the changes. The position and the orientation of the pasted tile is calculated so that the matching to the tile or set of tiles that it replaces is maximized. Then, the pasted tile inherits its connectivity from the replaced tiles, and the gap error between the tile and its neighbors is recalculated and minimized by re-solving the constraint system. The type of transformation used to reduce gap error (rigid or affine) can be selected by the user.

Figure 8 demonstrates an example that combines multiple operations. Figure 8a shows the original zoning division and street network of downtown Madrid (top), and a satellite view (bottom). Figure 8b shows two zoomed areas: a curve in "Gran Via" Avenue (top), and the building of the Museum of the Royal Academy (bottom). Several copy/paste operations are performed. First, the museum is copied and pasted to a set of tiles in the southeast corner that were initially occupied by residential buildings. Then, the building of the Spanish Army is copied and pasted to (1) the former location of the museum (Fig. 8d bottom), and (2) a set of tiles with residential buildings (center of the map). Two specific rotations are then applied to some tiles in the northwestern corner of the city, aiming to eliminate the curve in "Gran Via" Avenue. Such rotations result in overlapping tiles with large gap errors. Finally, the constraint system is re-solved for the entire layout and a straightening of "Gran Via" is obtained (Fig. 8d top). The final result of the sequence of operations is depicted in Figure 8c. The process was completed in approximately 10-15 minutes (once the source and destination tiles had been identified). Getting similar solution using an image editor would require experienced user and several hours of editing.

7. Conclusions and Future Work

We have presented an efficient system for urban layout editing that effectively exploits low-level structural information such as the street layout and the parcel shape provided by GIS. We derive high-level structural information, such as tile similarity and

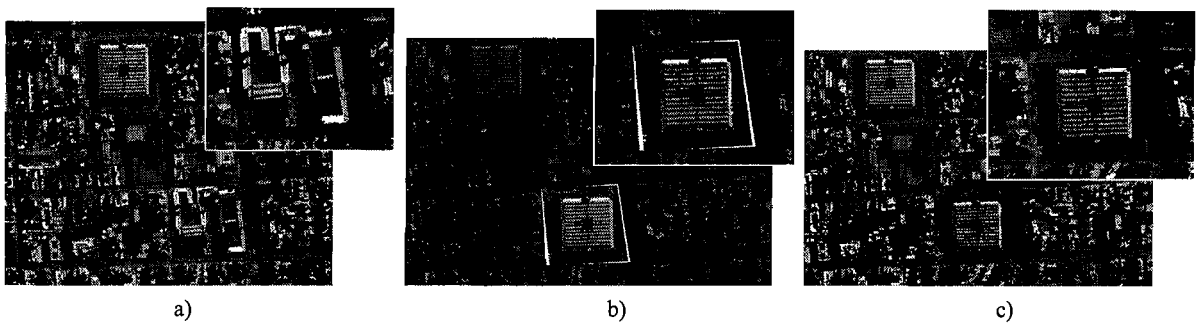


Figure 7. Copy/Cut & Paste. *(a) A view of Buenos Aires; (b) the result after performing a paste operation, and (c) after re-solving the constraint system.*

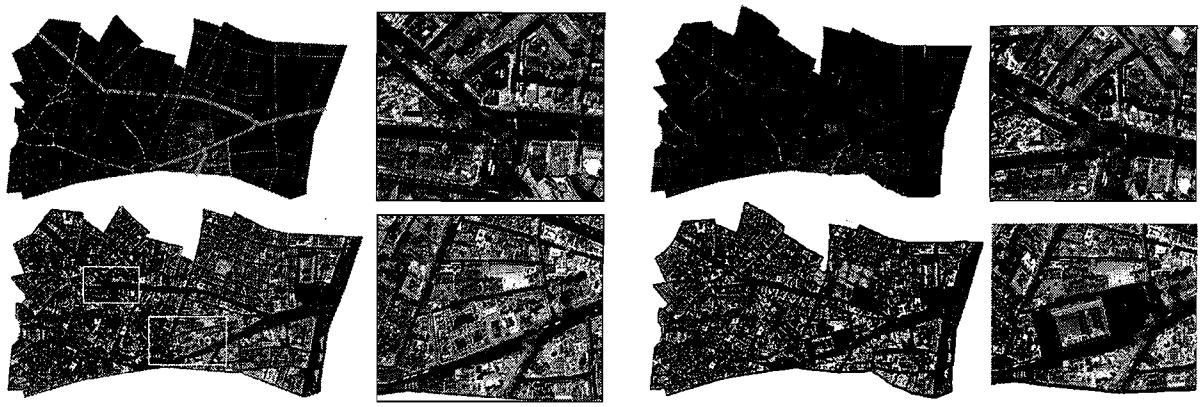


Figure 8. Combined Example. *(a) Zoning and Satellite image of the original layout of Madrid and two different zoomed areas. (b) Edited layout. Royal Academy building (lower left zoom box) has been moved to a different location in the southeastern corner of the map, and has been replaced by a copy of the Spanish Army building (lower right zoom box). Madrid's Gran Via (upper zoom boxes) has been straightened. Several other tiles have been also copied and relocated.*

street connectivity, and use it for efficient urban layout modifications. Our system allows for a wide variety of operations where pixel-oriented editors fail, such as moving city areas to new locations, topology changing copy and paste, straightening roads, etc. An efficient constraint solver maintains the deformation of the regions minimal and allows for easy replacement of parts that do not fit well with other parts that preserve similar style. In this way the visual plausibility of the new city layout is maintained coherent with the input data.

There are several possible improvements of the system. Our current similarity estimation method has some limitations as well as the edition of new urban layouts. The method we describe is efficient and accurate when comparing tiles that are relatively similar, as is common for urban layouts. However, the quantitative result is less stable for large dissimilarities. The actual bottleneck of the system is calculation of the tile distortion error. An efficient method for this calculation should be addressed as future improvement. While our system supports multiple operations, the future work should include developing the full spectrum of standard low-level editing procedures and interfacing with an urban simulation engine (e.g., UrbanSim, www.urbansim.org) to obtain high-level editing goals. The new editing tools could include creating a city from scratch perhaps using a style extracted from a database of urban layouts or one of a generic set of city templates.

References

[1] S. Avidan, A. Shamir, "Seam Carving for Content-Aware Image Resizing", *Proc. of ACM SIGGRAPH*, 26(3), 2007.

[2] H. Fang, J. Hart, "Detail Preserving Shape Deformation in Image Editing", *Proc. of ACM SIGGRAPH*, 26(3), 2007.

[3] H. Zhou, J. Sun, G. Turk, J. M. Rehg, "Terrain Synthesis from Digital Elevation Models", *IEEE Transactions on Visualization and Computer Graphics, 13(4),* 834-848, 2007.

[4] Y. Parish, and P. Müller, "Procedural Modeling of Cities", *Proc. of ACM SIGGRAPH*, 301-308, 2001.

[5] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, D. Salesin, "Image Analogies", *Proc. of ACM SIGGRAPH,* 327-340, 2001.

[6] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural Modeling of Buildings", *Proc. of ACM SIGGRAPH*, 614-623, 2006.

[7] P. Müller, G. Zeng, P. Wonka, and L. Van Gool, *"*Image-based Procedural Modeling of Façades*", Proc. of ACM SIGGRAPH, 26(3), 2007.*

[8] D. Aliaga, P. Rosen, and D. Bekins, "Style Grammars for Interactive Visualization of Architecture", *IEEE Transactions on Visualization and Computer Graphics,* 13(4), 786-797, 2007.

[9] J. Kim and F. Pellacini, "Jigsaw Image Mosaics", *Proc. of ACM SIGGRAPH*, 657-664, 2002.

[10] C. Hoffmann and K.-J. Kim, "Towards Valid Parametric CAD Models", *Computer-Aided Design*, Vol. 33, 81-90, 2001.

[11] P. Sander, J. Synder, S. Gortler, H. Hoppe, "Texture Mapping Progressive Meshes", *Proc. of ACM SIGGRAPH*, 409-416, 2001.

[12] E. Zhang, K. Mischaikow, G. Turk, "Feature-Based Surface Parameterization and Texture Mapping", *ACM Trans. on Graphics*, 24(1), 1-27, 2005.