HAL
open science

Urban Brush: Intuitive and Controllable Urban Layout Editing

Xiaochen Zhou, Pascal Chang, Marie-Paule Cani, Bedrich Benes

To cite this version:

Xiaochen Zhou, Pascal Chang, Marie-Paule Cani, Bedrich Benes. Urban Brush: Intuitive and Controllable Urban Layout Editing. UIST '21: The 34th Annual ACM Symposium on User Interface Software and Technology, Oct 2021, Virtual Event, United States. pp.796-814, 10.1145/3472749.3474787 . hal-03431693

HAL Id: hal-03431693

https://hal.science/hal-03431693v1

Submitted on 16 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Urban Brush: Intuitive and Controllable Urban Layout Editing

Xiaochen Zhou*
Purdue University
USA
zhou1178@purdue.edu

Pascal Chang*
LIX, Ecole Polytechnique/CNRS, IP Paris
France
pchang@student.ethz.ch

Marie-Paule Cani
LIX, Ecole Polytechnique/CNRS, IP Paris
France
marie-paule.cani@polytechnique.edu

Bedrich Benes
Purdue University
USA
bbenes@purdue.edu

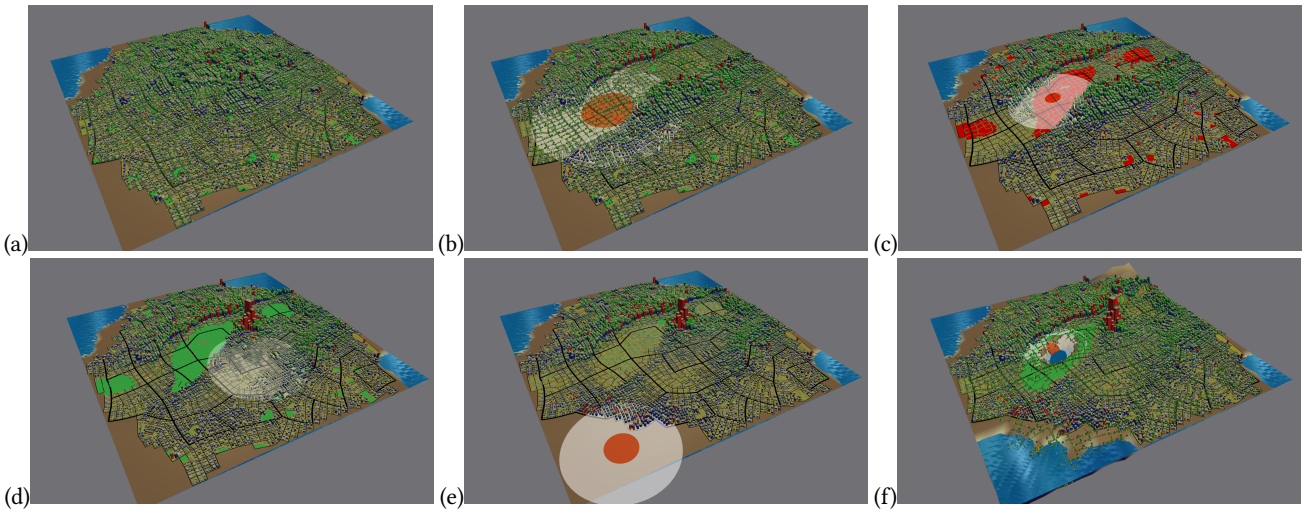

Figure 1: Starting from an initial urban area (a), first, the user pushes the jobs and population in the red circle area to the white area by the repulsor brush (b), and then creates the new land use to build parks marked as red (c). Next, a drag-drop brush is used to re-allocate the job, and population, which forms a downtown area (d), and a break brush distributes all properties and removes the blocks, parcels, and roads in the target area (e). Finally, the user changes the terrain, and the system automatically calls brushes to create the mountain area, park, and coastline with the consistency of the population and jobs (f).

ABSTRACT

Efficient urban layout generation is an interesting and important problem in many applications dealing with computer graphics and entertainment. We introduce a novel framework for intuitive and controllable small and large-scale urban layout editing. The key inspiration comes from the observation that cities develop in small incremental changes *e.g.,* a building is replaced, or a new road is created. We introduce a set of atomic operations that consistently modify the city. For example, two buildings are merged, a block is split in two, etc. Our second inspiration comes from volumetric editings, such as clay manipulation, where the manipulated material is preserved. The atomic operations are used in interactive brushes that consistently modify the urban layout. The city is populated with agents. Like volume transfer, the brushes attract or repulse the agents, and blocks can be merged and populated with smaller buildings. We also introduce a large-scale brush that repairs a part of the city by learning style as distributions of orientations and intersections.

*Shared first authors

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

UIST '21, October 10–14, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8635-7/21/10.
https://doi.org/10.1145/3472749.3474787

CCS CONCEPTS

• **Computing methodologies** → **Shape modeling**; Interactive simulation.

KEYWORDS

Urban modeling, procedural models, interactive modeling, geometry

ACM Reference Format:
Xiaochen Zhou, Pascal Chang, Marie-Paule Cani, and Bedrich Benes. 2021. Urban Brush: Intuitive and Controllable Urban Layout Editing. In *The 34th*

Annual ACM Symposium on User Interface Software and Technology (UIST '21), October 10–14, 2021, Virtual Event, USA. ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3472749.3474787

1 INTRODUCTION

Large-scale urban layouts on arbitrary terrains are essential in many Human-Computer Interaction (HCI) and Computer Graphics (CG) applications. However, their generation is not a straightforward nor simple task. Urban layouts can be generated by procedural modeling (*e.g.,* [29]) that uses rules to describe the generation of roads, blocks, parcels, and buildings. Close to procedural generation are simulations (*e.g.,* [43]) that use real-world rules and behavior to create urban layouts. Reconstruction (*e.g.,* [26]) brings real-world assets to the virtual world by converting data from sensors (cameras, LiDAR, depth maps) into formats suitable for 3D rendering and further processing. Urban layouts can also be generated by using interactive modeling (*e.g.,* [6]), that is probably the best way of controlling the output and creating models that closely express the user intent.

However, there is a disconnect of the methods for urban layout generation. Procedural methods and simulations are notoriously difficult to control. Although they can quickly provide visually plausible layouts, they tend to be repetitive and may include errors such as missing egress or other architectural problems. Reconstruction algorithms are limited to the input and often need additional manual effort to make the CG models usable. Interactive methods are often tedious and slow to provide large-scale output, while the user receives no help towards consistency and realism.

One of the critical problems of urban layout design in HCI and CG is therefore consistently modifying an existing layout, which would enable users in seek of control not to start from scratch. Existing approaches either focus on the full design of the entire urban layout [43] or allow only global changes [38] by changing simulation parameters. Very few works attempt to modify an urban layout in a consistent way [19, 32].

Our first inspiration is the observation that cities grow in increments [4]. It is uncommon that an entire city will be built from scratch. Instead, a building is often removed and replaced by a different one, or a small area of a city is remodeled. In all cases, the affected area must be carefully integrated into the existing urban layout. An important aspect of such localized changes is that they need to respect the function of the city. People living in the affected area must be relocated, lost jobs should be replaced, etc. Our second inspiration comes from interactive design. Users prefer intuitive and straightforward operations for content creation and modification. Several approaches focused on intuitive operations that mimic brushes, allowing a wide range of operations that hide the underlying algorithms' parameters. Brush and sculpting metaphor have been used in shape modeling [5], constant volume deformation [9, 41], and landscape modifications [11], but they have not been used for editing of highly structured urban layouts.

We present a novel approach to interactive modeling of urban layouts in which we combine user-controlled editing with localized, context-sensitive changes. We introduce a set of brushes that implement localized atomic operations on urban layout. We use brushes as the interactive tools since the atomic operations can be interactively applied to any urban layout. Each brush is parameterized by its area of influence and its function. The brushes are also context-sensitive, and affected jobs or inhabitants are relocated. In this way, each modification of the city is consistent and does not modify its function. In addition, the changes are localized and fully controllable by the user.

We show our approach on interactive examples and large-scale edits of various urban layouts. An example in Figure 1 shows extensive global changes to an urban layout that are consistently handled by low-level operations.

We claim the following contributions. (1) We introduce a set of intuitive brushes for urban layout modifications. (2) We introduce a novel space colonization algorithm that allows for the intuitive generation of urban layouts with different styles. (3) We introduce a non-homogeneous brush analogy that considers the urban layout to be a material with vessels. (4) We show how the brushes can be combined into fully automatic global operations on urban layouts.

2 RELATED WORK

This work is related to urban procedural modeling and simulations. It is also inspired by interactive editing methods such as virtual sculpting. We do not review works on urban reconstruction, referring readers to [25]. We also do not focus on terrain modeling, surveyed in [14], nor cover virtual worlds creation, in general, that was reviewed in [35].

Procedural methods generate a model from initialization and a set of rules. Parish and Müller [29] were the first to combine various procedural methods into a procedural city generation pipeline. They use a set of input layers like population, density, or terrain map, and they extend L-systems [31] to grow street networks and generate buildings at large scales. The purely procedural models for large cities were further extended by [21, 22]. Wonka et al. [45] introduced instant architecture, *i.e.,* procedural modeling for buildings using split grammars. This approach was further expanded by [23] and recently by [34] who introduced advanced internal communication of procedural modules. On a smaller scale, a large amount of work has been dedicated to the automatic generation of façades (*e.g.,* [24]).

Road networks have been generated from a regular grid pattern [15] and pattern-based templates [36]. However, these methods ignore the behavioral variables like population density on the road network's shape. Vanegas et al. [38] combine behavioral and geometrical modeling that adapt to the underlying population, jobs, terrain, and local transportation demand. This approach successfully captures the road density variations induced by the population distribution, but it only offers limited control over the network's appearance. Agent simulation was used to generate road networks [18], and an example-based method was introduced in [3]. Good global controllability of planar urban layouts can be achieved by modeling the overall layout in a dual space of tensor fields [6]. Road networks define *blocks and parcels* that have been generated by a subdivision scheme in Vanegas et al. [37]. This approach was extended in [40] by using the straight skeleton of a block's contour. Lastly, roads and parcels generation were adapted to rough terrains and sparser urbanization [10].

The main problem of procedural models is user control since rule parameters only enable indirect tuning, which affects the whole result. Moreover, they may also fail to generate valid urban layouts since they do not account for the city's function, namely hosting a population and giving them access to jobs.

Urban simulation methods, in contrast, focus on the functional modeling of a city. Early work focused on behavioral modeling, using cellular automata [1, 7], agent-based simulation [30], or micro-simulation discrete choice models [42] exploiting agents that make decisions to locate and move within the generated city, including land use, activity, population, and jobs. A recent method use GANs to reconstruct a 3D city from a photograph [17]. We build on the urban simulation algorithm of Vanegas et. al [38] that generates consistent global urban layouts from macro parameters such as jobs, land use, and population. However, this approach only allows for global modifications, making it harder for urban editing, local control, and user interaction.

Urban editing has been addressed by Lipp et al. [19] who introduced transformation operators based on graph cuts that, combined with a layering system, allow intuitive manipulation of urban layouts like drag and drop, translation, rotation, etc. However, their work does not integrate behavioral modeling (*i.e.,* simulation of population, jobs, and their movement), so the edited city's consistency may be lost. Vanegas et al. [39] introduced a high-level control over an existing urban model, thanks to an inverse procedural model that generates urban layouts fulfilling high-level criteria on sunlight exposure, the ratio of parks, or landmark visibility. Finally, deep-learning sketch-based method for urban editing that infers user sketches and converts them into a consistent procedural model has been introduced in [27].

Local control and user interaction require a solid and flexible system designed for both urban simulation and user-friendly editing metaphors. Existing approaches (*e.g.,* SketchUp and AutoCAD) use the region selection feature for local control and editing. However, this can only select regular shapes (rectangles), which is insufficient for flexible urban editing. Our system is inspired by the work [44], where multiple layers were used for local feature editing and, together with image combination, allow users to precisely edit the targeted elements. To design the layers, we also borrow the idea of a hierarchy system [46], where a tree structure is used for the hierarchy of attributes. It allows the edits on low-level layers to automatically affect the high-level layers (*e.g.,* edit on road layer will affect the building), which keeps the system's consistency when the user edits the lowest layers.

Sculpting and painting metaphors: Our work borrows from the interactive methods enabling consistent, intuitive authoring of virtual worlds. While sketching and sculpting techniques were introduced for shape modeling Cani et al. [5], where consistency was expressed in terms of geometric constraints such as constant volume deformation [9, 41], they were extended to virtual worlds editing. Emilien et al. [12] presented a framework to populate virtual worlds with distributions of objects (rocks, trees, houses, and roads) from user-defined examples, using a painting metaphor. The user is provided brushes that store statistical distributions of scene elements and their correlation with terrain slope instead of colors. They can be used to paint locally consistent distributions of objects. Moreover, statistical consistency is maintained when selected parts of the layout are moved or deformed over the terrain. This approach was extended to dedicated brushes enabling to author consistent plant ecosystems over eroding or large-scale terrains [8, 13]. In contrast, such methods were never applied to the consistent urban layout editing, which we tackle here.

3 URBAN LAYOUT

3.1 Input and Structure

The input to our algorithm is an urban layout that includes jobs and population jointly referred to as agents in this work. The urban layout $\mathcal{U}$ is a set of 2D layers

$$L = [h, R, k, p, b, S, P],$$

where h denotes the terrain height, R the road network, k blocks, p lots (parcels), b buildings, S jobs, and P population (see Table 1). Height is stored as a high-resolution image. Jobs, and population are input in discrete layers of fixed resolution $w \times h$ (100m ×100m in our system) and re-calculated per lot as a weighted sum of the number of agents in cells that intersect the lot (see Appendix A). The roads are stored as a graph. Blocks and lots are polygons, and buildings are either 3D meshes or procedural models that generate them.

Table 1: Layers L used in an urban layout $\mathcal{U}$. Jobs and population are computed per each lot.

Layer	Symbol	Type
Height field	$h(i, j)$	Grid
Roads	$R = [V, E]$	Graph
Blocks	k	Polygon
Lots	p	Polygon
Buildings	b	3D mesh
Jobs	$S(p_i)$	Integer
Population	$P(p_i)$	Integer

Roads are represented as an oriented graph $R = [V, E]$ with $v \in V$ vertices corresponding to intersections, and edges $e \in E, e = v_i \rightarrow v_j$ that correspond to road segments. We follow the road classification from [16] and divide the roads into highways e^h, arterial roads e^a, and streets e^s. Highways are the highest capacity roads that connect cities. Arterial roads are inter-city roads for fast and high volume traffic, and streets connect arterial roads and other streets with individual lots. Our urban layouts do not include highways because they are used for inter-city connections.

The loops in the road network $[v_i, v_{i+1}, \ldots, v_i]$ may define blocks denoted by k, and each block can be subdivided into lots p. Lots may include buildings b, which are represented either as a mesh or as a procedural model that generates the said mesh. Empty lots are marked as parks. Each road also optionally carries information about the traffic flow.

The urban layout *i.e.,* the distribution of roads and geometry of buildings, can be given explicitly. In our framework, we use the simulation from [38] to generate it. Similarly, population and jobs, detailed next, may either be generated through simulation (*e.g.,* [38, 43]) or provided by the user as 2D maps.

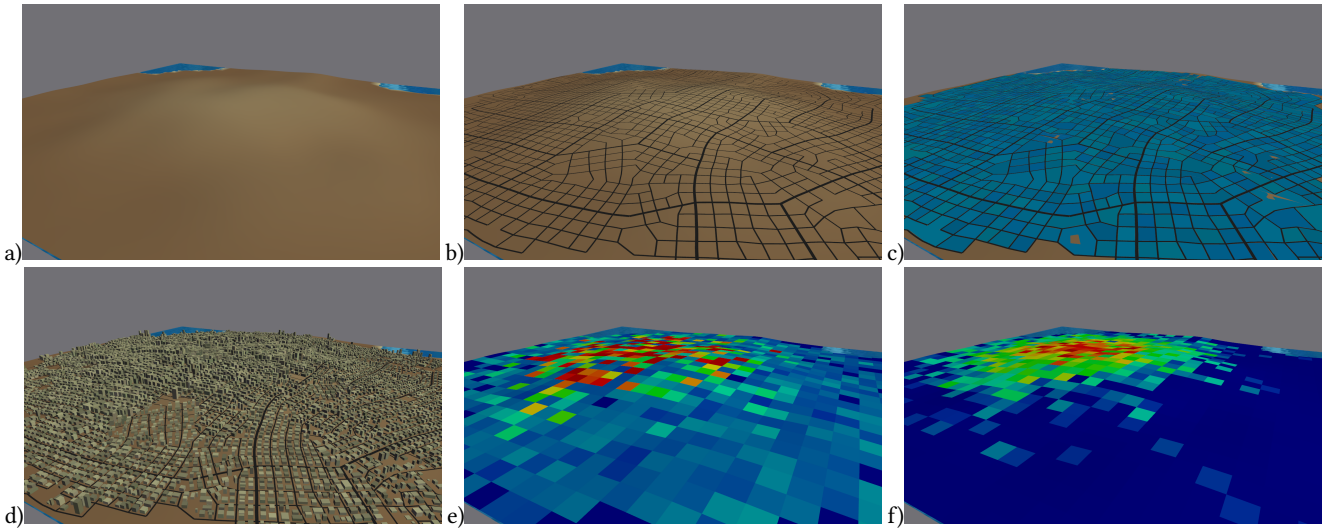

Figure 2: Input layers of an urban layout: Terrain height (a), roads (b), blocks and lots (c), buildings (d), and optionally population (e) and jobs (f).

3.2 Lot Structure and Validity

A lot (a parcel) is the smallest unit in our urban model. A lot belongs to a unique block and can be characterized by its contour geometry and the number of jobs and population (agents) it carries. A lot contains at most one building, whose volume always reflects the total number of agents of the underlying lot. If the lot is empty, there is no building, and the lot is marked as a park. For physical plausibility, buildings' height is not allowed to exceed a specific maximal bound that depends on its base area. This defines a maximal amount of population and jobs $M(p)$ a parcel p can host.

Each lot also includes two additional non-negative variables, *population-to-handle* $\Delta P \geq 0$ and *jobs-to-handle* $\Delta S \geq 0$, which serve as temporary buffers during operations. This allows us to write operations on lots systematically: any amount of population or jobs that are supposed to leave a given lot is first put into these temporary buffer variables before they can be distributed, relocated to, or absorbed by other lots.

With these considerations, a lot p is said to be *valid* if 1) the capacity of the lot imposed by the maximal height of its building is not exceeded (*i.e.,* $P(p)+S(p) \leq M(p)$), and 2) the temporary buffers are empty ($\Delta P(p) + \Delta S(p) = 0$).

3.3 Consistent City Editing

A functional city model is characterized by a distribution of agents that matches the building volume and reflects some reasonable proportion between population and jobs. A second key feature is the consistency of the road network, making each district accessible while accounting for the topography of the underlying terrain features, including slope and presence of water bodies. Our goal is to maintain the input city model functional throughout editing.

We use the analogy of painting with 2D brushes and sculpting clay material, where the constant volume maintains consistency throughout changes. We need to design brushes that maintain both the population and their jobs to constant amounts. Moreover, contrary to clay, a city embeds a consistent inner structure: the road network. Thus, deforming it is similar to sculpting complex material, such as reinforced concrete, or organic shapes, such as leaves that embed a vessel's network. To our best knowledge, no sculpting method has been proposed yet to achieve this. The city is located on terrain, and we also need to ensure that the road network remains consistent with the slope. Lastly, the urban layout includes the city's division in lots, which, together with the way streets are oriented and branch together, gives a unique style to each city. We seek to define editing operations that preserve such style.

We define basic atomic operations on buildings, lots, and roads that can be combined into a variety of consistent tools, *i.e.,* tools that preserve the total amount of population and jobs in the city, the validity of all lots, and road network (defined in Section 3.2).

4 ATOMIC OPERATIONS

The interactive brushes in Section 5 are designed bottom-up from a set of consistent atomic operations which we describe in this section. There is a logical succession of these operations given by the semantic dependencies of the city layout. For instance, operations like merging or splitting blocks affect the impacted blocks and the dividing road segment, and all the lots and buildings contained in the blocks.

4.1 Buildings

Because the height of a building (rounded to a closest integer number of floors) is defined such that its volume corresponds to the actual number of agents in the underlying lot, operations that only affect the behavioral attributes of lots (amount of agents) without modifying their geometric attributes (contour shape of the lot) can be seen as operations on buildings.

The **Transfer** operation only affects buildings and has no other dependencies. Given two buildings and the amount of population and jobs to relocate (P_t, S_t), it tries to transfer as much of these

amounts as possible from one building to another, while maintaining the validity of the two underlying lots (see Section 3.2).The desired amount to transfer might not be feasible due, for instance, to target lot reaching maximum capacity, or source lot not having enough agents.

4.2 Lots

We introduce two operations on lots: *Merge* and *Split*, which use the operations on buildings. Unlike the *Transfer* operation (Section 4.1), *Merge* and *Split* are atomic operations, and they modify the geometry of the affected lots.

Merge joints two adjacent lots, destroys their buildings, and creates a new and larger building that includes all the agents from the input lots. Suppose the newly created building cannot absorb all agents from the original buildings due to the maximum capacity of the population and jobs in the lots. In that case, the operation fails and is canceled to maintain consistency (otherwise, we are either invalidating a parcel or violating the constant-agent constraint). *Merge* can be used, for instance, during a *Transfer* operation, where the target building reaches maximal capacity: instead of reducing the transferred amount, the target building can merge with its neighbor to absorb more.

The **Split** operation divides a lot in two. As in [40], we calculate the lot's PCA-aligned bounding box and split it according to its larger side. The two new lots are then populated with buildings that attempt to absorb the agents from the original lot. Similar to *Merge*, if the split causes an excess of agents, it is not executed because the city would become inconsistent. This operation prevents tall buildings with large foundations from becoming small buildings with large foundations, which is not plausible. The split can be controlled indirectly. For example, a lot will split every time the included building's height goes beneath a given threshold.

4.3 Roads and Blocks

A block is an agglomeration of lots surrounded by roads. Similar to lots, we define atomic operations *Merge Blocks* and *Split Block*. However, contrary to lots, blocks and roads are mutually dependent, an operation on two adjacent blocks affects the separating road segment between them. Similarly, operations involving road segments affect adjacent blocks.

Operation **Merge Blocks** affects two adjacent blocks with a road segment in the middle. It erases all lots in both blocks and stores all agents and jobs in temporary buffers. Then it removes the road in the middle and merges the blocks into one. The newly created block is subdivided into lots, buildings are created and occupied by the agents and jobs. Buildings have different sizes because each building's size is proportional to the lot it occupies.

Similarly, the **Split Block** operation takes a block, finds its PCA-aligned bounding box, and divides the block by inserting a road segment. It first removes all the agents into a temporary buffer and deletes all lots and included buildings. It then adds the road and creates two smaller blocks. The blocks are then populated by the agents and jobs in the same way as the merge operation.

5 BASIC BRUSHES

The atomic operations are combined into user-controlled parameterized brushes. The brushes modify the city geometry locally, just like the sculpting tools induce spatial transformations on localized areas of the sculpted material. They can push away, attract, or even relocate some of the city's volume within itself. The brushes can be applied to population, jobs, to their sum, or to the total building volume. We will describe them on population.

We introduce three local brushes in our framework: *Attractor*, *Repulsor*, and *Drag & Drop* reallocate a certain number of agents and manipulate the underlying blocks, lots, and buildings.

5.1 Brush Parameters

The brushes have several shared intuitive parameters (see the menu from our implementation in Figure 15 and Table 2) that control their influence.

Table 2: Brush common parameters

Affects	Action
Impact region	concentric circles
Population	adds/removes agents
Height	adds/removes building height
Amount	adds/removes an absolute amount
Percentage	adds/removes a percentage
Target	max saturation value
Continuous	adds/removes continuously or per click
Jobs/Population/Both	what is affected
Allow Merge	allow buildings to merge if needed
Allow Split	allow buildings to split if needed
Travel time distance	reallocate based on road or Euclidean distance

General properties of the brushes include user-specific options regarding the number of agents to be moved (see *Repulsor* and *Attractor*). Each brush has the influence region and the impact region (Figure 3). The brush (Table 2) is specified as two concentric circles. The inner circle is the impact region, where the immediate effect (for example, for removing agents) is applied. The influence region specified as the outer circle defines where the brush can make the modifications to compensate for its action to maintain overall consistency. For instance, it can be where the affected quantities (*e.g.,* agents) are deposited or drawn from, depending on the type of the brush. They can be moved either using Euclidean distance or traveling time distance along the roads (Figure 3). Note that the influence radius can be infinite, allowing agents to be relocated anywhere in the urban layout.

Changing the **Stroke Style** affects the distribution weights when the brush computes the amount to remove or add. *Uniform* manipulates the same quantity (absolute amount or percentage) over the brush's inner region. *Gaussian* specifies a falloff from the center of the brush (absolute amount or percentage). The weights follow a Gaussian distribution of parameters $\mathcal{N}(c, r_{inner}/2)$ where c is the brush's center and r_{inner} the inner radius. In the same way, the outer circle can be set to uniform or Gaussian.

The brush transfer mode can either be discrete, *i.e.,* controlled by mouse clicks, or continuous (Figure 3).

5.2 Repulsor

The *Repulsor* brush removes a specified amount of population from the lots in the brush center. It moves it to the influence area without modifying the road and block geometry (Figure 4). This is achieved by iteratively transferring agents between pairs of buildings in the two regions (sorted by distance to the brush center) using the *Transfer* operation (see Appendix B.1 for details).

There are two possible extreme cases. On the one hand, the user may set the influence region to zero, and no change will be made to the city. On the other hand, if the influence region's radius is very large or infinite, the excess of agents is distributed to many lots, leading to visually imperceptible changes. In both cases, the city's population and jobs are constant through the brush application so the city remains consistent.

Merge and Split: The *Repulsor* brush can be alternatively used with an option allowing for the use of the atomic, merge, and split operation (split in the impact region, merge in the influence region). These two regions will then attempt to adapt to the new amounts of agents by either splitting their lots or merging neighboring ones, thus decreasing or increasing the number of agents that can be consistently handled.

5.3 Attractor

The *Attractor* brush acts intuitively as the opposite of *Repulsor* by pulling agents in (Figure 5). Similar to *Repulsor*, it iterates simultaneously over the lots in the impact and the influence region and transfers agents based once again on the *Transfer* operation (see Appendix B.2). Note that the brush may pull fewer agents than expected if there were not enough of them in the influence region.

Merge and Split: The *Attractor* brush can be alternatively used with the option allowing for the use of the atomic operations *Merge* in the impact region and *Split* in the influence region. The impact area will attempt to accommodate more agents by merging the lots as shown in Figure 6, where small and abundant buildings are combined into larger and taller ones. Similarly, the influence region will try to accommodate fewer agents by splitting blocks.

5.4 Drag & Drop

This brush allows moving agents from one area of the city to another. It does not use two concentric circles. Instead, the user selects a region from where agents will be drawn (impact). Then, as the user brushes through the influence region, the agents are added to the buildings under the brush as buildings in the first region get flatten out. The algorithm is similar to the *Repulsor*, except that it computes the lots' distances to the brush center twice (at the first click and then on the second selection).

Merge and Split can be alternatively used in the influence and impact areas similar to the *Repulsor* and *Attractor*.

5.5 Mask

The mask brush locks the lots in the impact regions. Locked lots will not be affected by the brushes above. Additionally, users can choose to lock specific lot types, such as parks, etc.

5.6 Road Break and Connect

Here we define two simple brushes applied to roads that expand the previous brushes.

The **Break** brush can be viewed as an extension of the *Repulsor* (Section 5.2) that also deletes roads in empty regions and merges the affected blocks. It can either remove vertices or edges of road segments, depending on the selected mode. A valid block should be sealed by roads and should not obtain any dead-end roads or arterials. When using brush removal, some nodes and roads are eliminated, making the original closed blocks unsealed, leading to invalid blocks. In this case, the blocks are removed as well as all lots and buildings in them. The affected buildings' population is moved to the influence area using the *Repulsor* brush. An example is the top part of Figure 8 shows this brush in action.

Similarly, the **Connect** brush is an extension of the *Attractor* (Section 5.3) that attracts agents from the influence region, but also splits blocks, adding roads between them when applied to blocks.

6 STYLE-PRESERVING BRUSH

Until now, sculpting a city had strong similarities with clay sculpting. However, road connectivity makes the layout anisotropic that needs to be treated differently, for example, as a non-homogeneous material that includes vessel network, which connectivity needs to be maintained. We are not aware of any previous work that solves this problem.

6.1 Concept of City Style

The road connectivity and geometry are essential for the overall look and feel of the urban layout. We want to make sure that the brushes do not alter their visual consistency. While the basic brushes did not globally affect the road network, style needs to be considered for more extensive changes.

The urban layout appearance is predominantly determined by the spatial distribution of intersections that, in effect, determines the road layout [2, 38]. Therefore, we capture the urban layout style by a careful categorization of the intersections and statistics of the intersection types' distribution and orientation, the junction angles, and distances between them.

6.1.1 Style extraction from a urban layout. Let us recall that the road layout is an oriented graph $R = [V, E]$ with a set of vertices $v \in V$ that represent the intersections and edges edges $e \in E, e = v_i \rightarrow v_j$ that represent roads. Edges are further classified into arterials e^a and streets e^s. Each intersection v_i is classified according to the edges it connects. We will call an edge connecting a vertex its *connection*.

Each intersection v also has its *arity* that corresponds to the number of connecting edges. We consider intersections of arity two, three, four, etc. but we do not consider the intersection of arity one (dead-ends). We define the vertex identifier (vID) as a sequence of letters identifying its connectivity in the counter-clockwise order starting with the positive direction of the $x - axis$ as indicated by the arrow in Figure 9. Moreover, the *directions* of each connecting road is stored as the angle towards the $+x$ axis. The middle example in Figure 9 has the first connection to a street road and the next two to arterials. Its arity is three, and the vID is s(15)-a(85)-s(230).

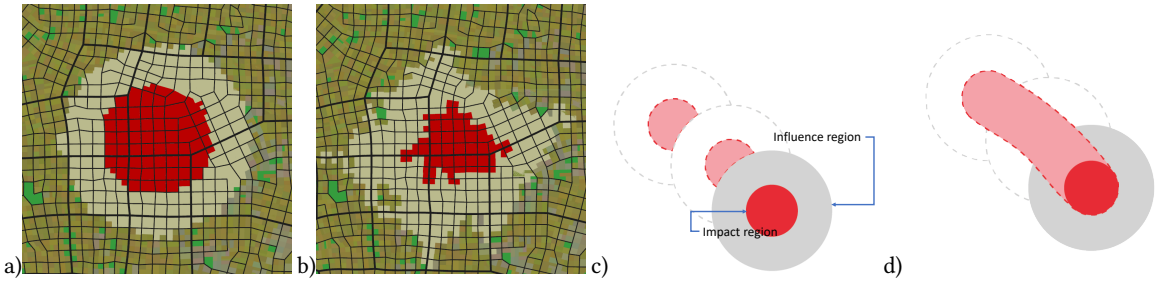

Figure 3: Brush impact region (red) and influence region (light green) over the same area using: (a) Euclidean distance (b) Travel time distance. (c) In the discrete mode, the affected lots are not stored, and the influence region of the next step may cover the impact region of the previous steps; (d) In the continuous mode, we store the lots in a separate data structure that allows next steps not to change agents distribution in the lots that were previously in the impact region.

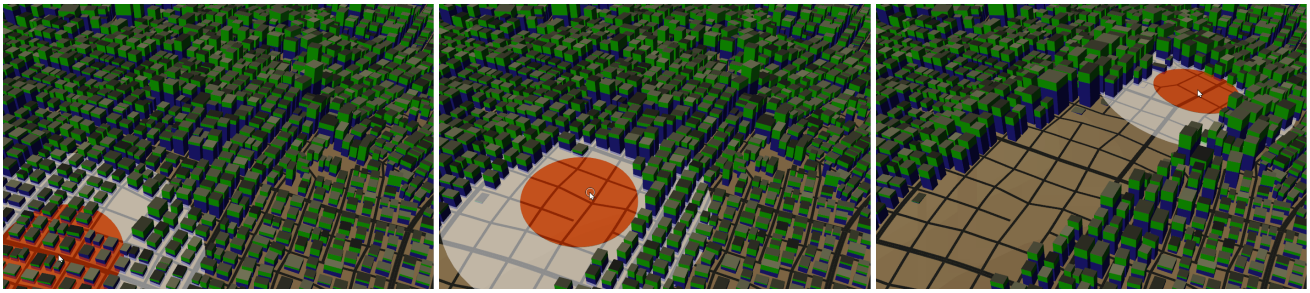

Figure 4: Applying the repulsor brush to remove 100% of the agents.

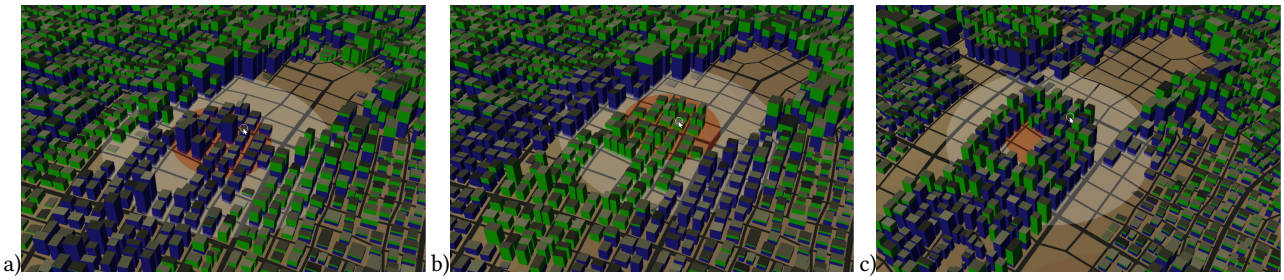

Figure 5: Using the Attractor brush to pull in: (a) population only (blue), (b) jobs only (green), (c) or both. The brush only takes agents from the influence region (white circle).

The *vID histogram* is denoted by vID_H, and it stores the frequency of each vID in a given urban layout U or in a selected area. An example in Figure 7 show an urban layout and its vID. The *junction histogram* is denoted by J_H and it stores the directions for each item from the *vID histogram*. The top graph shows the vID, the middle graphs are branching angles, and the bottom graph shows the average distance between intersections. The directions are quantized 72 bins by binning the angles from 0° to 360° by 5° per each vID. Each value is stored as the mean and the standard deviation. Last, the *intersection distance histogram* D_H stores per each vID the distance to the nearest intersection as the mean and the standard deviation.

The example in Figure 7 shows a pretty regular city layout and an organic one. The regular layout has more pronounced bins around integer multiples of $\times 90^o$ than the organic one, as visible in its J_H.

6.2 Re-Build Brush

The goal is to re-build an empty part of a city or a region destroyed by the Break brush. More precisely, the first input is the usual brush parameters, where the impact region of the brush defines the region that should be populated by a new urban layout, and the influence region is the region from which the population and jobs to be moved to the newly generated buildings are extracted.

The second input is the set of histograms vID_H, J_H, and D_H that characterize the desired city style. Note that the style can

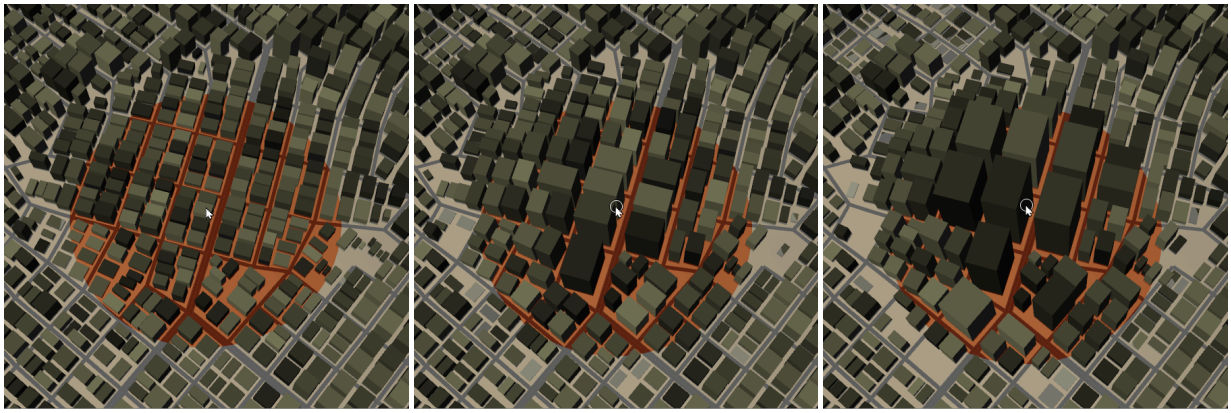

Figure 6: Small buildings merge, for example when the user adds agents. The underlying lots are then merged.

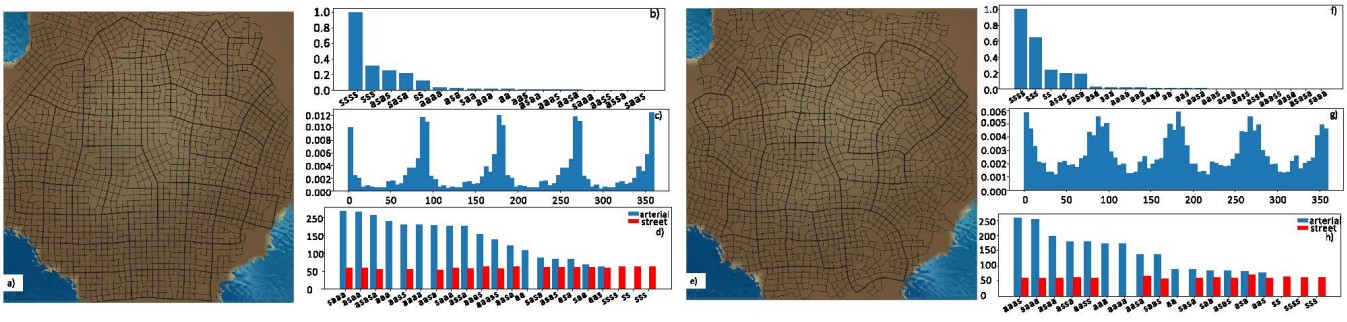

Figure 7: Urban layout histograms The top graph is the vID frequency, middle graph shows the branching angles, and the bottom graph distances between intersections. A regular urban layout (a) has a dominant presence of branching angles that are $k \times 90^o$ (c) and a high number of streets of the same length (d). Organic pattern (e) has more randomized street angles (g) as well as the distances of intersections (h). The distributions of vIDs (b) and (f) are similar. Note a high number of roads of type s-s-s-s.

be captured from an arbitrary city, or district, thus allowing style transfer.

We introduce an algorithm for generating the road pattern using space colonization. The roads are generated hierarchically (see Figure 10). First, we generate the arterials, then the streets. As each road is expanded, it attempts to connect and replicates the layout style by sampling the distributions. Each road also generates intersections that act as *oriented markers* that define not only where, but also *how* the roads connect.

6.2.1 Space Colonization of Urban Layouts. We extend the space colonization algorithm for tree growth [28, 33] and interactive tree modeling [20] to allow for the generation of urban layouts. Space colonization populates some volume with markers that act as attractors. The tree develops by extending branches that compete for the markers by growing towards them and consuming them. The resulting emergent phenomenon is the shape of the tree.

We include three important modifications to this algorithm. First, we consider the *orientation* and the type of each marker represented as vID. The markers are the intersections, and the roads attempt to connect them. Second, instead of filling the area with the markers before the simulation, the markers are generated by the expanded roads during the simulation. In other words, each road generates new markers. This allows for the generation of closed graphs as opposed to only trees. Third, we generate the layout hierarchically. Arterial roads first cover the area while generating markers for streets that are connected in the second pass (see Appendix C).

7 LARGE-SCALE MODIFICATION

The atomic operations and brushes can be applied interactively as shown in Section 5, 6 and through the examples in Section 9. However, because each operation is consistent, it can also be used in an *automatic* mode to modify the urban layout. This approach's key idea is to identify the problematic part that needs attention after a large-scale operation. We demonstrate this approach on three examples: city expansion, shrink, and changing relief.

7.1 Geometric Inconsistencies

Inconsistencies are localized and small parts of the city where the consistency has been violated. We have already defined consistency from the viewpoint of agents and jobs in Section 4, and through the notion of style in Section 6. Here we expand this definition by adding geometric terms. Following the hierarchical subdivision

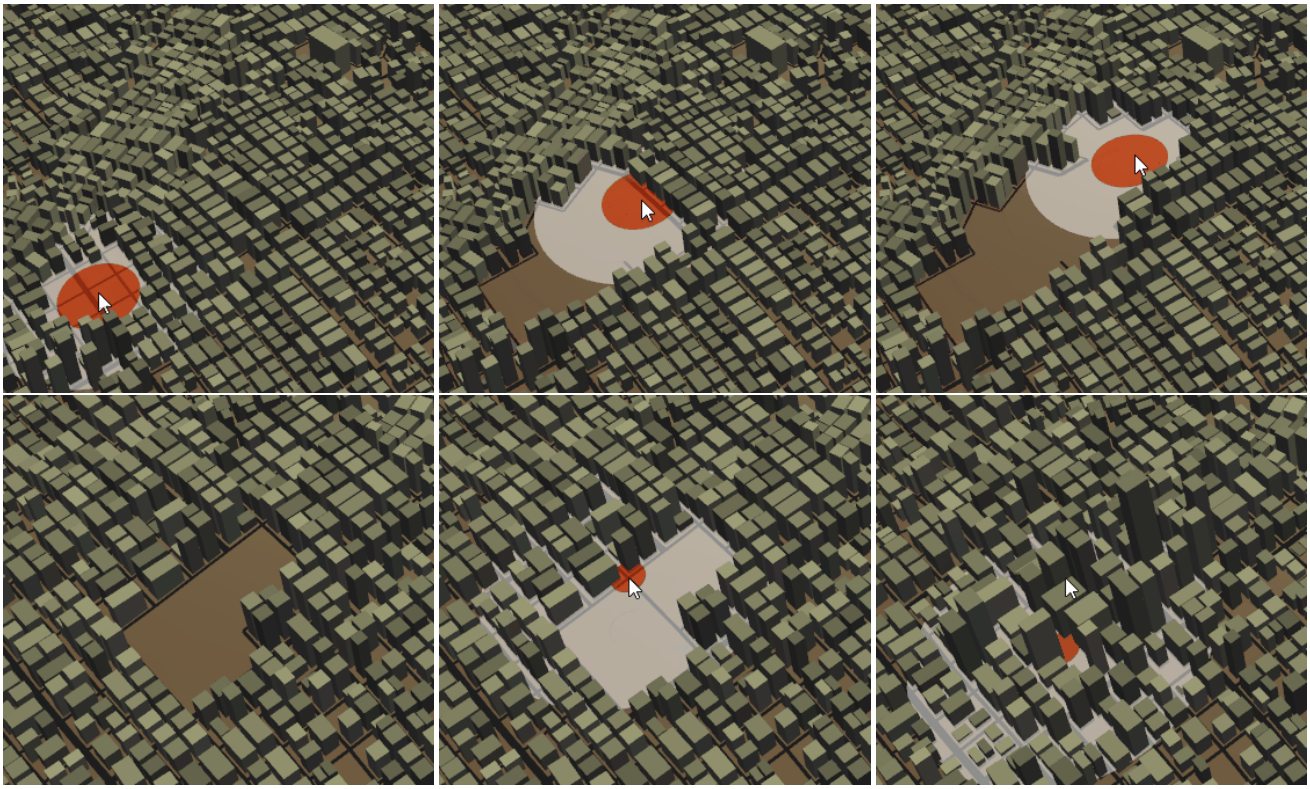

Figure 8: Sequence of brush Breaking roads (top). The population is relocated to the neighborhood by using Repulsor brush. Similarly, the Connect brush (bottom) splits blocks and lots into two by adding road segments, and occupies them by using the Attractor brush.

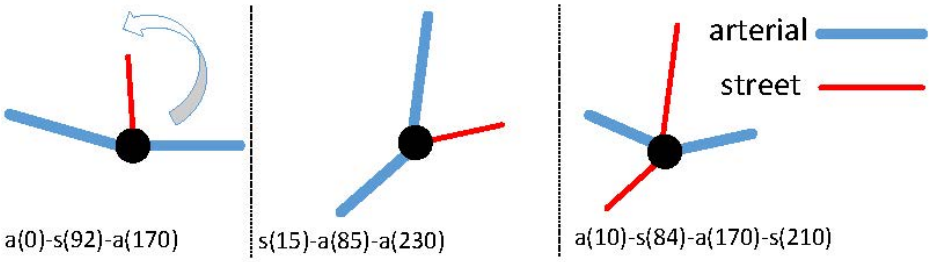

Figure 9: Vertex ID identifies each intersection. Counting from the positive direction of the $+x$ axis counter-clockwise, we store the types of connecting roads and their angle. The left example starts with an arterial, the next intersection is a street, and it ends with another arterial resulting in vID a(0)-s(92)-a(170).

of the city from Section 3, we define an inconsistent building, lot, block, and road segment.

An inconsistent **building** is either taller than its maximum height or smaller than the minimum height. It is handled implicitly by the atomic operations from Section 4.

An inconsistent **lot** is either too small or large or has a high aspect ratio. A small lot will be merged with its neighbors, and a large lot will be split by using atomic operations. A lot that has its aspect ratio large (we use 1.7× in our implementation) is also marked as inconsistent. Such a highly asymmetrical lot will be merged with the neighboring lot on its large side. If there is no lot, it is removed, and the agents from its building are transferred.

Similarly, an inconsistent **block** has either a too small or too large area and will be merged with its neighbor or split in two. A high-aspect-ratio block is merged with a block on its larger side, but only if it is not adjacent to an arterial road. The block merge operation (Section 4.3) removes the road segment between the two blocks. We do not want to affect arterial roads because they are essential for the overall city connectivity and appearance. The extreme lot and block sizes parameter can be extracted from the city style since the size of blocks is related to the length of the street

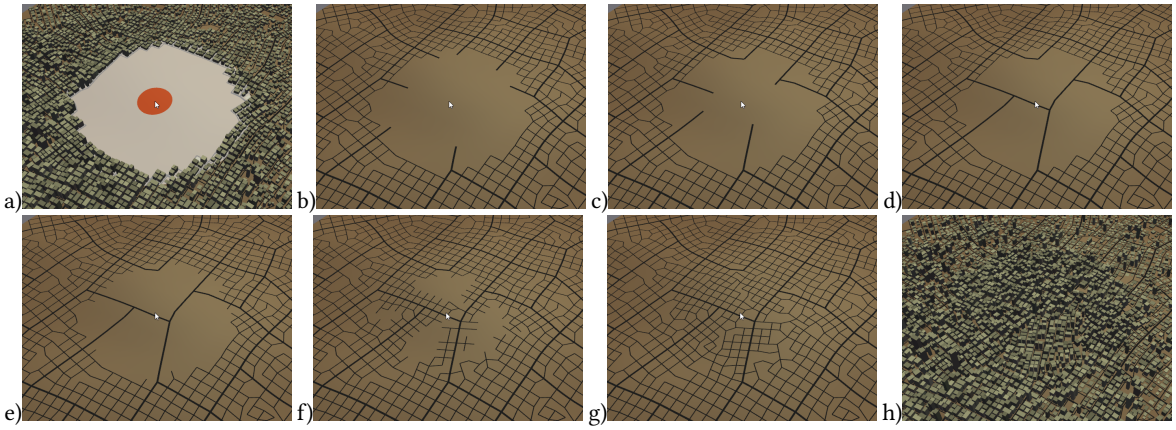

Figure 10: Urban layout generation An urban layout with a missing part (a) is completed with arterial roads (b-d). Once the arterial roads have been generated the streets are generated (e-g). Then the lots are occupied and people relocated leading to a complete layout (h).

segments that surround them. The geometric inconsistencies can be learned from style, except for road slope, which is a physical constraint.

Finally, a **road** segment is inconsistent if it is located in a water body (*e.g.,* a lake), or its slope is greater than a maximum allowed value, indicating that the road is too steep. Such a road segment is removed, and the adjacent blocks are deleted while removing their lots, buildings and relocating their population. Note that we do not define operations for merging and splitting road segments because they are handled implicitly by merging and splitting the blocks.

7.2 Handling Inconsistencies

When a large change is applied to a city, such as uniformly scaling the entire urban layout or editing the terrain to add a mountain or a lake, the current framework will apply the operation directly to the urban layout, leading to local geometric inconsistencies.

The lists of inconsistencies are then created, one for road segments, blocks, lots, and buildings. Each list is ordered by the amount by which it violates the layout. For example, a lot inside a lake has $+\infty$, a lot with high aspect ratio stores the value of the aspect ratio, roads are sorted by their slope, etc.

We then process the lists in their hierarchical order. First, we eliminate all roads with a high slope, which also causes the blocks and the associated lots and buildings to disappear. This is performed by automatically calling the Repulsor brush, with block split enabled. Then the list of inconsistent blocks is processed, starting with the highest value of inconsistency. Next, we use the block merge operation to fix the blocks. Eventually, all inconsistent lots are corrected, starting with the highest value of inconsistency.

After all the inconsistencies have been fixed, we get a consistent urban layout. However, some parts may have become empty (*e.g.,* is an existing lake was moved elsewhere) or have disconnected, (*e.g.,* if we grow a uniform table mountain in the city, the upper part may end-up disconnected from the down position). The newly emptied areas are checked, and the Re-Build brush is automatically called to fill them, using a city style learned from the surroundings. The connectivity is then verified, and the smallest disconnected components are removed, and the agents and jobs they hosted are relocated.

Several parameters are left under the user's control during global operations: Each brush has its influence area. For example, when we add a hill in the city and some blocks become invalid, it is unclear how far the agents should be relocated. The user can control this by setting a global parameter for the entire city or by providing a map of varying parameter values.

8 USER STUDY

We have performed a detailed user study to validate the usability of our approach.

8.1 Population

The study was performed by 12 participants, aged from 20 to 50, seven male and, five female. Four participants are graduate students majoring in computer graphics, six of them are graduate students with no or little experience of graphics where one of them is an expert in urban planning. One undergraduate student in computer science and one college staff. They self-identified their 3D modeling skills as novices (2 participants), familiar with 3D modeling (4), and experienced with 3D modeling, previously working on Unity Engine and SketchUp (4). When we asked about their experience in computer graphics, they reported: no computer graphics experience (3 participants), little experience (1), familiar with computer graphics (5), and an expert (1).

8.2 Study Protocol

The user study lasted about one hour, and it was divided into three parts. 1) We introduced our system to each participant, described each brush's functions, and showed them some generated results using our system. 2) We assigned each participant two tasks. a) For the first one, we showed them an urban layout and urban model of San Francisco from Google map and Google earth (shown in Figure 11). We asked the user to compare the initial urban model

Table 3: User study responses and time on tasks. Q1: "I could easily achieve the task", Q2: "The tool is intuitive for city modification", Q3: "The model is intuitive for creating urban models". Scale: 0 strongly disagree, 1 disagree, 2 neutral, 3 agree, 4 strongly agree.

User ID	**Modeling Exp.**	**CG Exp.**	**Age**	**Time [min] on Task1**	**Time [min] on Task2**	**Q1**	**Q2**	**Q3**
#1	0	0	23	28	32	2	3	3
#2	0	0	22	22	28	4	4	3
#3	0	0	24	30	35	4	3	4
#4	0	1	25	29	42	3	4	4
#5	1	0	25	19	42	3	2	4
#6	1	1	48	28	33	4	3	4
#7	1	2	24	12	20	3	4	2
#8	1	2	26	16	22	3	4	4
#9	2	2	26	24	50	3	4	4
#10	2	2	25	15	44	3	4	4
#11	2	2	26	19	47	3	2	3
#12	2	3	24	25	33	3	2	4
avg.	1.00	1.25	26.50	22.75	35.67	3.17	3.25	3.58
stdev.	0.85	1.06	6.88	6.75	9.54	0.58	0.87	0.67

with references and modify the urban city to resemble the reference. b) For the second task, we gave the user a template of a city and asked them to create their own. They were asked to modify the geometry of the provided template terrain and make at least one large-scale change, such as adding a river, creating a mountain, etc. 3) Eventually, we asked the participants to complete a survey querying their experience with the system.

8.3 Results and Observations

The main differences between the provided starting model and the target model are land use and job and population distribution. Therefore, to create the target layout, the participants needed to modify the land use and distribute the jobs and populations to achieve visual similarity. Our system is an urban editing tool that maintains the city's consistency when atomic brushes are used. Our goal was to evaluate user's experience, which we did by evaluating: (1) The user's ability to successfully edit the urban layout towards the specified target; (2) User's satisfaction about their city editing experience with our system. (3) User's satisfaction with the city layouts they have built.

All participants successfully modified the San Francisco urban layout and created a downtown model (shown in Figure 12). The participants changed the land use (adding a park, beach, etc.) and built a downtown via distributing the jobs and population to the target area by re-allocating the job and population distribution with given brushes.

In the second task, all participants created their city by using our system. Although creating a new urban city with new terrain requires the users to be more familiar with all the brushes, participants could easily carve the new landscape, built a new road network, and re-allocate the job and population after the basic training. Figure 13 shows the initial template urban given to the participants and the cities created by the participants.

Figure 14 presents the frequency and timing of usage of each brush by each user for the first task. The *Repulsor* brush was used most frequently since creating parks and jobs, and population reallocation can be done by *Repulsors* with different attributes quickly. The second frequently used operation was *Drag and Drop*. An interesting point through the statistics is that the user #5 used drag and drop brush much more than *Repulsor* while other users use *Repulsor* most. The usage of the brushes from participants varies, which shows that there is no universal solution that fits all participants to complete their task. The feedback from the user study does not indicate a preference for specific brushes. Users can choose to edit the city layout or the distribution of the population and jobs at their own will. Our system will allow the users with different strategies to use our system easily with no restriction and encourage users to edit the city with their creativity. All participants finished the first task in under 30 minutes (20.8 in average, between 12 and 28 minutes). The amount of time for the second task varied, while all the participants completed the second task in under one hour the minimum time on task was 20 and the maximum 50 minutes with the average time 35.1 minutes. Table 3 shows each participant's information and the time used on each task.

To ensure low latency and real-time operation, we have dedicated a significant amount of work to optimize our system for speed. It uses GPU computations, and it is also optimized in C++ with OpenMP to use all CPU cores. The users did not mention any latency when using the brushes, even when the influence area was infinite. Achieving interactive feedback required some restrictions on brush operations, *e.g.,* we do not allow users to use brush removal in continuous mode.

The qualitative questions show that the participants thought our system is useful for urban planning and urban modeling. Also, they mentioned that the system is user-friendly. To be specific, we asked three questions Q1: "I could easily achieve the task" (responses 2,3,3,3,3,3,3,3,3,4,4,4 $\mu = 3.17, \sigma = 0.58$), "The tool

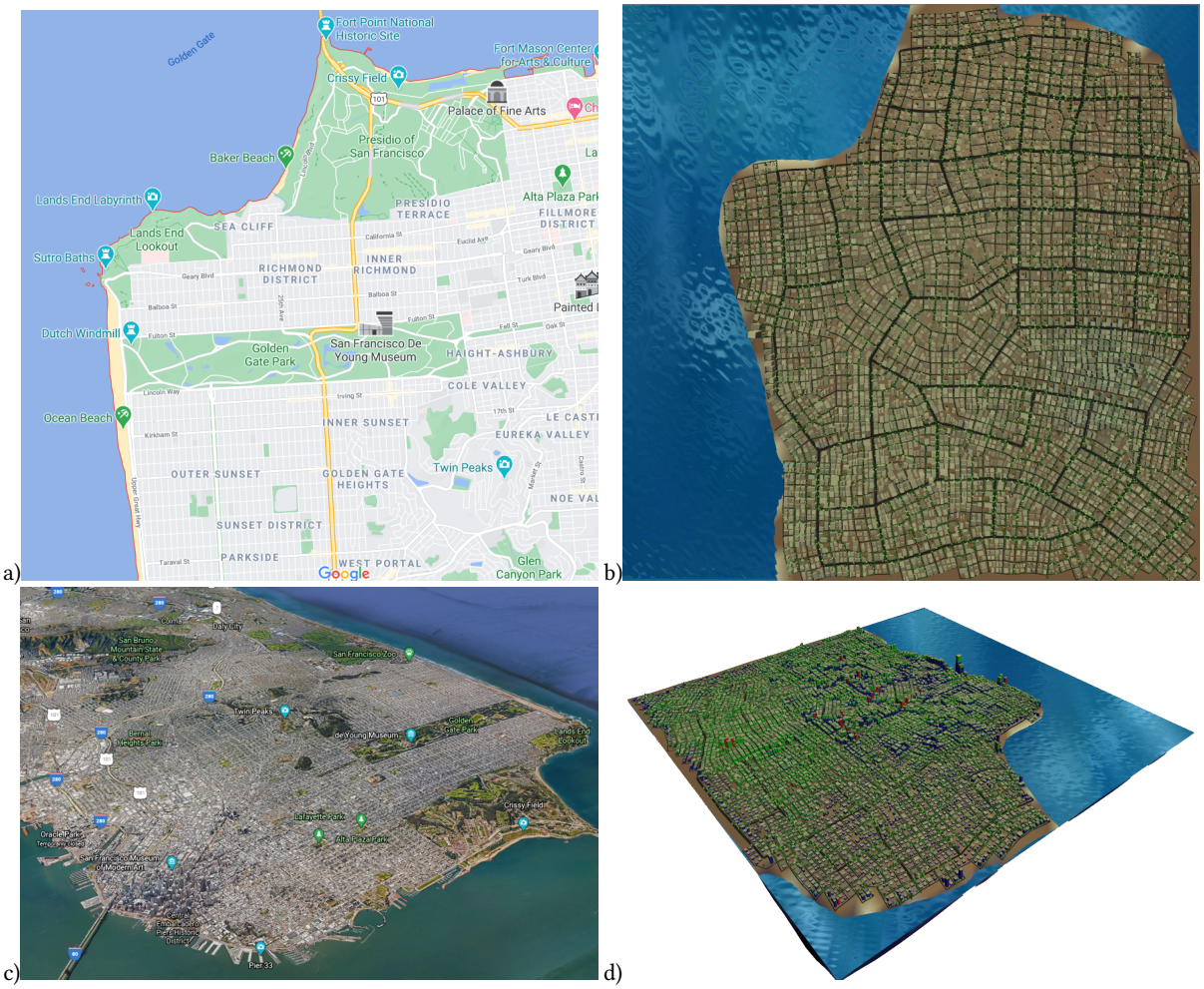

Figure 11: a) Reference San Francisco image from Google maps, b) initial urban model, c) urban distribution from Google earth, and d) initial urban distribution. The green color of the building represents jobs, and blue represents the population. The red color represents a saturated building.

is intuitive for city modification" (responses 2,2,2,3,3,3,4,4,4,4,4,4 $\mu = 3.25, \sigma = 0.87$) and "The model is intuitive for creating urban models" (responses 2,3,3,3,4,4,4,4,4,4,4,4, $\mu = 3.58, \sigma = 0.67$), where we use the scale is: 0 strongly disagree, 1 disagree, 2 neutral, 3 agree, 4 strongly agree. We did not encounter failure cases during the user study, and no obvious and unacceptable artifacts occurred. The users can correct the artifacts created by mistakes through brushes and edit the city's local detail by zooming in/out the terrain. All users were pleased when they successfully built their urban cities and enjoyed the experience of using our system. One of the participants, a graduate student majoring in urban planning, showed great interest in the urban modeling performance and visualization of the urban city. A young professional 3D artist who tested the system stated that "it feels like a Substance Painter for cities." Besides, one of the main goals of our tool is to encourage creativity. In our user study, participants with different skill levels could build their own city: the capital of Congo (#User 2) was created by an urban planning expert, Venice (#User 10) was recreated by a computer graphics expert, and Taichi (#User 9) was designed by a non-experienced user.

The participants also shared some other suggestions. Although the urban planning expert said that our system outperformed the system she was using in the lab in efficiency and visualization, she suggested that more features could be added as further work. In particular, we could add bridges that would connect isolated terrains and highlight the layout of the group of the same land-use area. Also, some participants would like to see brushes that can switch the terrain, population, and jobs between two regions. One participant wanted to have more refined and more direct control over the city layout. They also gave us some suggestions on improving the brush's performance, which can sharply increase the user experience when using the system. We will optimize the brushes and add more features in our further work.

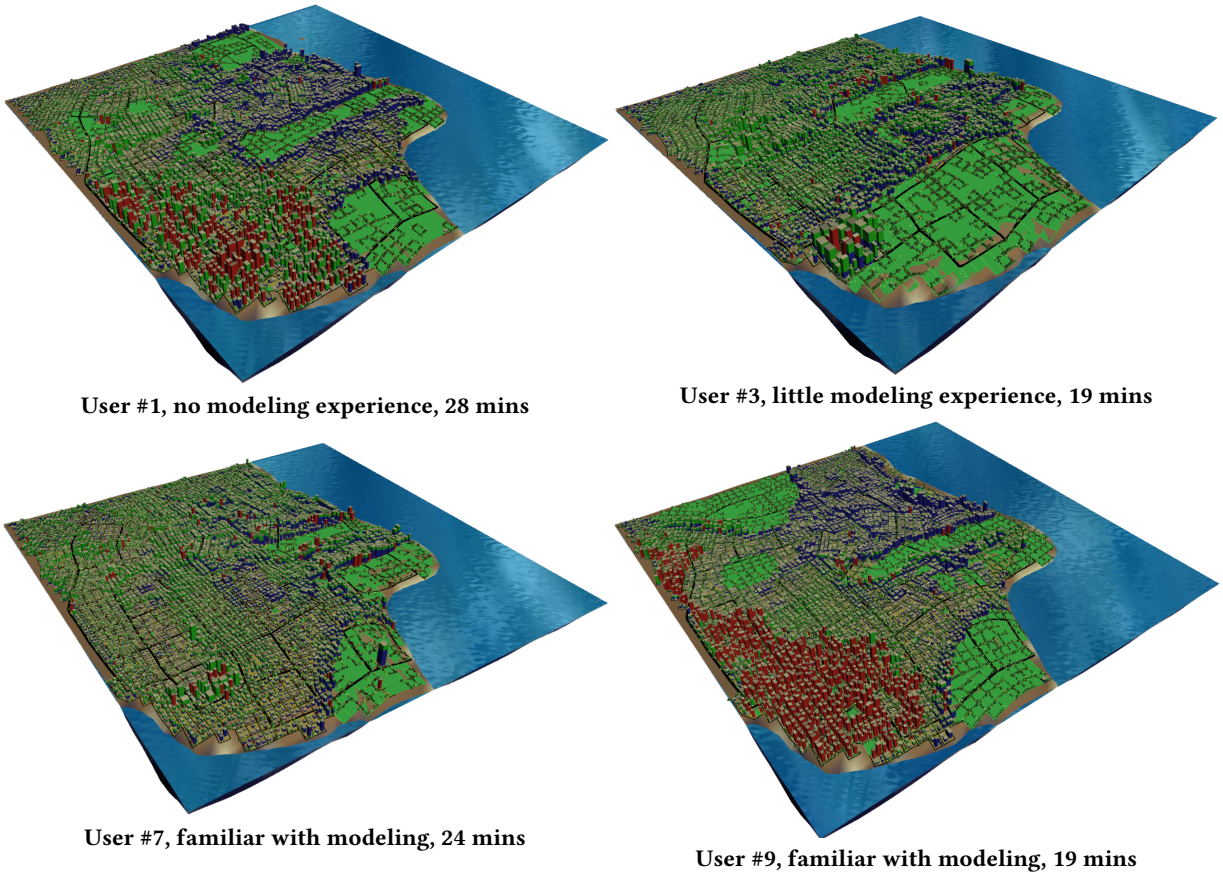

Figure 12: The urban models generated by participants for the first task.

9 IMPLEMENTATION AND RESULTS

Our system was implemented in C++ with Qt to define the user interface (see Figure 15 for the parameters of the brushes from Table 2). Our tests were performed on a desktop computer equipped with Intel i9-9900k at 4.2 GHZ and Nvidia RTX 2080 GPU.

9.1 Results

In addition to the results generated by user study participants, in this section we will show more results for specific functions that will be useful for urban modeling.

Style Transfer is simple with the style-preserving brush from Section 6. An example in Figure 16 shows a regular urban layout that has been cleared by several continuous moves of the block *Break* brush (visible in Figure 16 b)). The style-preserving brush then takes parameters from an irregular city and, with a single click, connects all affected intersections from Section 6 c) leading to a new layout. This operation takes a few seconds.

Sculpting: An example in Figure 17 shows how using the *Repulsor* and *Attractor* can be used to manipulate the overall height of buildings. A letter "A" has been embossed into the buildings by adding agents to certain buildings.

Shrink and Expand: We applied multiple global shrinking and expansion of an urban layout to show the consistency of the global operations. An input layout is shrunk globally by 1/3 in the x direction, causing some blocks to be invalid because of their size or aspect ratio (marked in red). The invalid blocks are locally and consistently fixed, resulting in a new layout. The layout is then shrunk in the y direction resulting in a new layout in Figure 18. The new layout has much higher buildings, but their overall height is relatively uniform, which was also the case for the input. We then expand the city in the x and y directions resulting in a new city similar to the input Figure 18 g). This example shows that we achieve reversible deformations in terms of the global aspect of an entire city.

Sea level rise: an example in Figure 19 shows an urban layout divided in two by an ocean. The flooded area is automatically cleared by the system using the *Repulsor* and break brush, excess of the population and jobs are relocated nearby, causing the buildings to grow.

10 LIMITATIONS AND FUTURE WORK

While all participants enjoyed their experience with the system, some pointed out aspects to be improved. In particular, users lacked an undo function when some unexpected results occurred due to their selection of the wrong brush or mistakes in the setting of attributes, such as buildings growing too tall or too short or

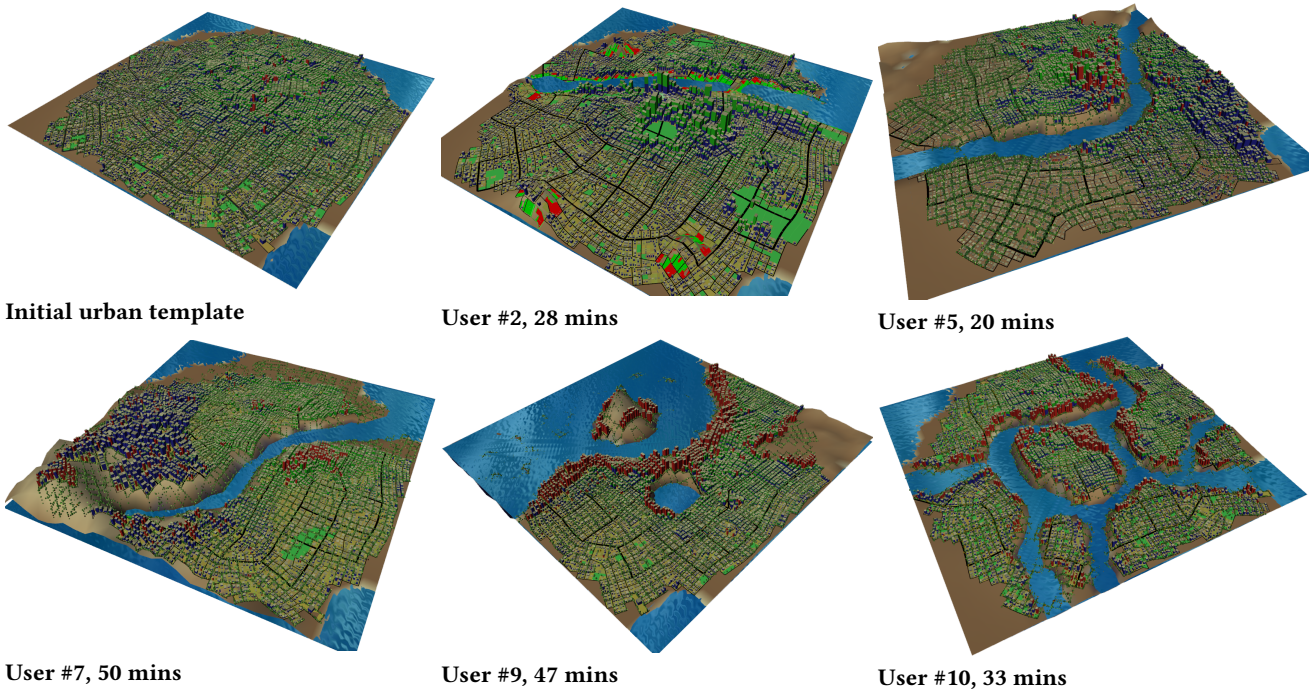

Figure 13: Urban models created by participants for the second task. The first figure shows the initial urban template, and the others are the users' cities.

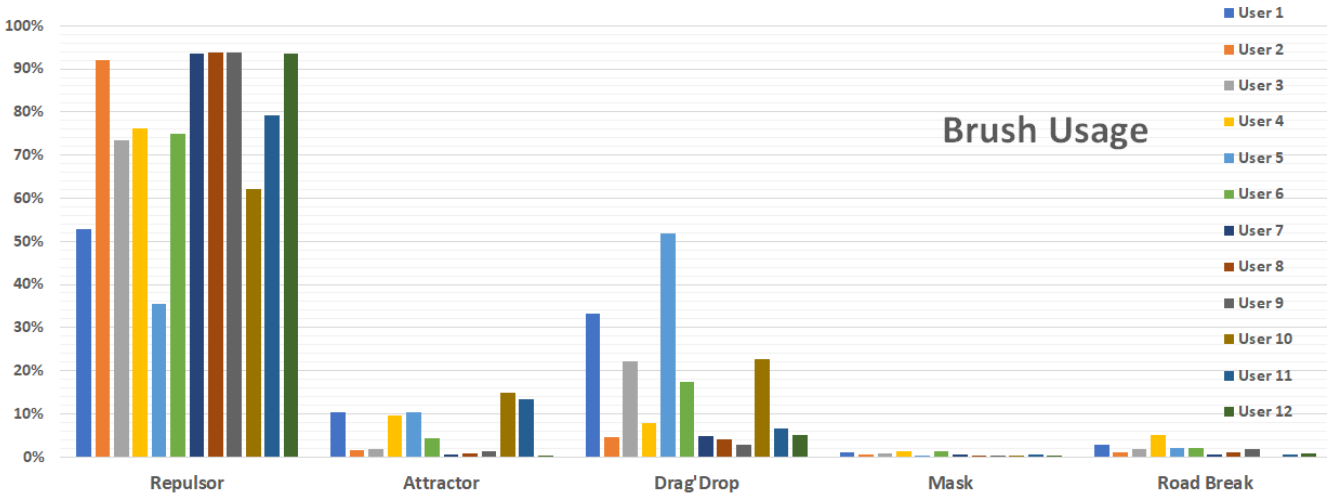

Figure 14: User Study Statistics: Frequency and usage of individual brushes in the user studies.

downtown areas growing larger than expected. Indeed, fixing these with a series of brush operations takes more time and effort than using undo/redo, which would be an easy addition to our system. To allow precise tuning, offering a detailed visualization of the object's attributes in the tools' influence area, *e.g.,* height of buildings, area of blocks, the number of population and jobs, would also be a valuable improvement.

In the future, the current algorithm should be extended towards more accurate city design. To achieve this, we could specify local land-use, *e.g.,* use for school, industry, etc. The associated attributes and relationships with the local layout could also be explored. For instance, the acceptable ratio of jobs vs. population and their max values could vary according to land use. Moreover, connectivity rules and restrictions between areas of different uses are the next interesting topic that could be explored in future work. For instance, school and residency should not be close to heavily polluted industrial areas. More brushes can be designed for such land-use partition, together with user-editable attributes. In addition, more

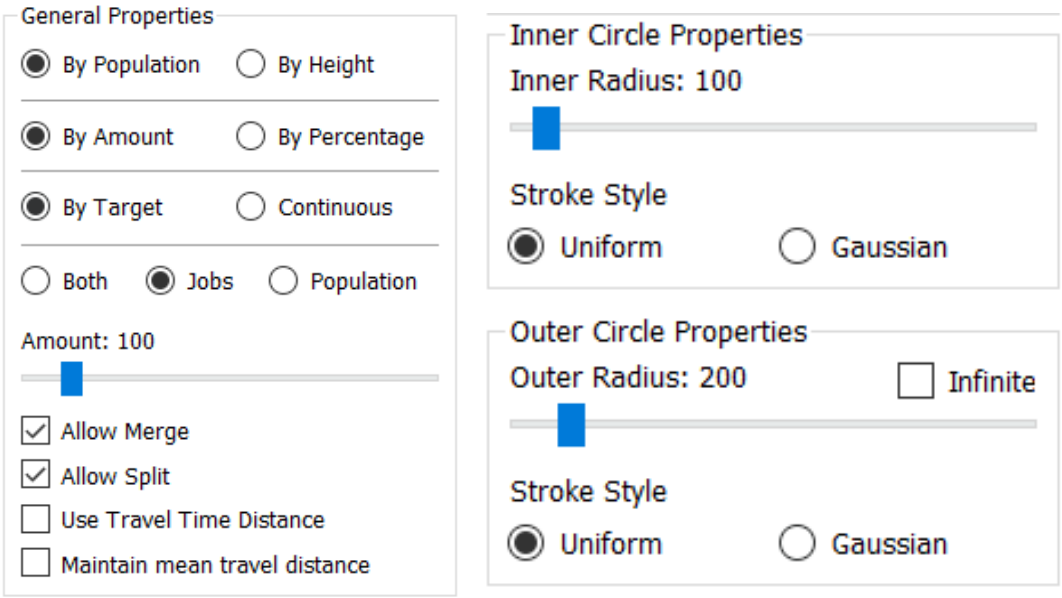

Figure 15: Brush parameters: (a) General Properties (b) Inner/Outer Circle Properties

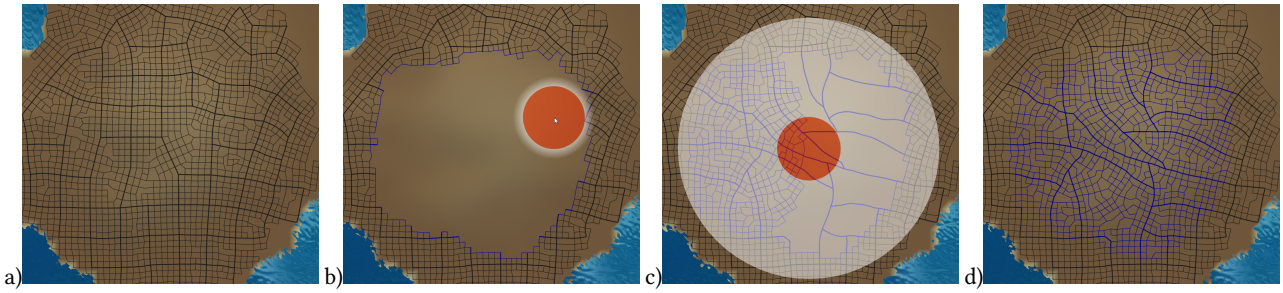

Figure 16: Style transfer example starts with a regular city (a) where the central part is deleted (b). Then the rebuild brush is used with irregular histograms from another city (c) to reconnect the layout (d).

diverse urban structures could be implemented, such as brushes for terrain changes, bridges, and highways. Other features such as building rendering and climate simulation could be added to improve completeness. Lastly, we could investigate the consistency between the attributes in the system (road size and type, building size, etc.), which can be learned from examples to improve the plausibility of the resulting city.

11 CONCLUSIONS

We presented the first expressive design method for urban layouts. Inspired by an analogy with clay sculpting, it provides the user with various editing brushes that allow maintaining the city functionality and visual style. Moreover, the brushes can be automatically triggered by larger-scale user interaction (from shrinking or expanding the city to moving a lake or sculpting the terrain), enabling seamless application of any kind of global change. Finally, thanks to its ability to keep the city consistent and functional, our method could be an excellent complement to current city reconstruction and generation techniques, which results are difficult to edit.

Among our contributions, the notion of city style, introduced in this paper, brings a wide range of applications: It could be used to analyze and cluster existing cities, for instance, for adapting the style of buildings to one of the layouts, or to build new cities which layout interpolates between predefined styles (*e.g.,* using optimal mass transport for histogram interpolation as in [11]).

Our work aims at mid-level operations, and its limitations stem from this objective. As observed in the user study, very low-level control over individual roads and buildings is provided. Also, more global edits, such as in [19] is not possible because we do not allow operations on large road segments. As future work, it would be interesting to combine such low-level and large-scale operations with our brushes and see how they perform in authoring large- and small-scale cities. Also, extra brushes could be introduced to address style editing while minimizing the current layout changes.

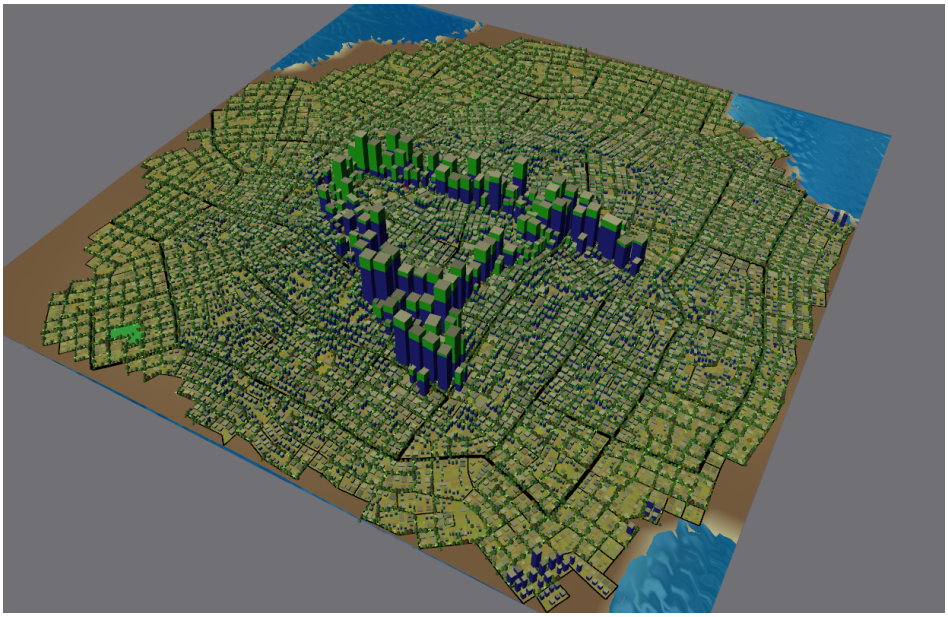

Figure 17: The *Repulsor* and *Attractor* brushes allow for intuitive changes of building height. A letter "A" was embossed in the building heights in this way.

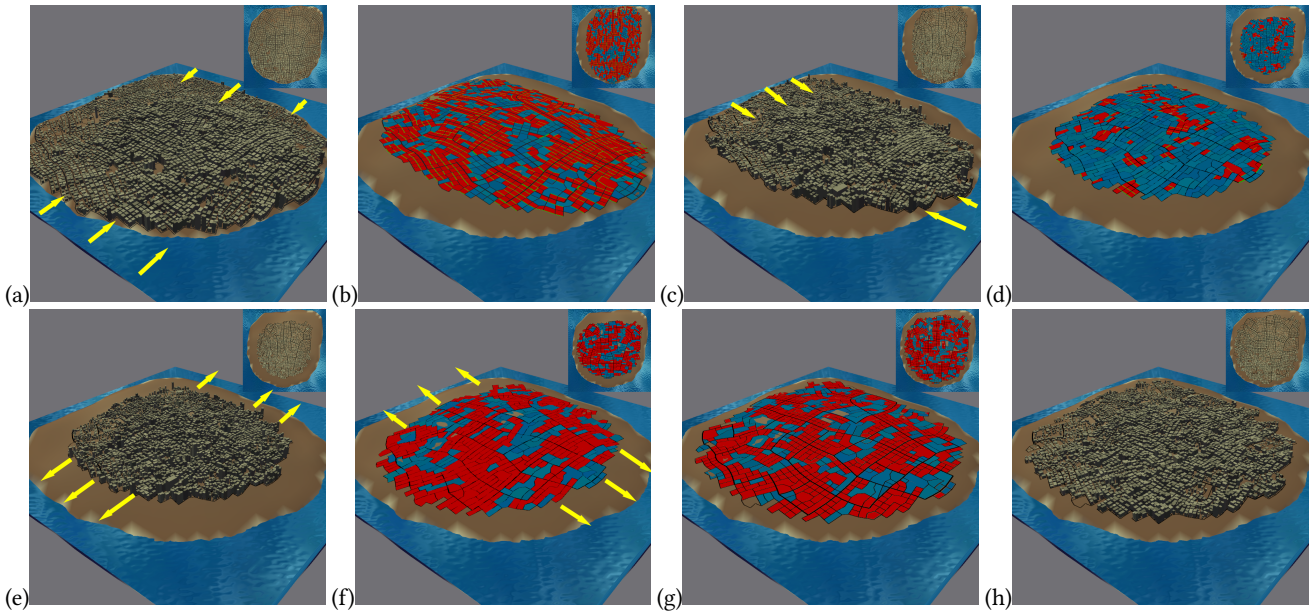

Figure 18: Starting from an initial urban area (a), the user shrinks the city globally by 1/3 in x direction. All intersections are moved by the corresponding distance that makes some of the roads, blocks, and lots inconsistent. All invalid blocks (marked in red) are then merged (b), resulting in a new, more elevated layout (c). The same process is then repeated in the y direction (d-e). We then expand the city back by 1/3 in the y direction (e) and the x direction (f), resulting in a city similar to the input (g). Note that the city is different, yet the layout is similar.

Figure 19: An urban layout is divided in two by a mass of water, causing the building close to the coast to grow.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their valuable and constructive comments, and to Romain Pascual for his help with an early version of the system. This research was funded in part by National Science Foundation grant #10001387, *Functional Proceduralization of 3D Geometric Models.*

REFERENCES

[1] Sharaf Al-kheder, Jun Wang, and Jie Shan. 2008. Fuzzy inference guided cellular automata urban-growth modelling using multi-temporal satellite images. *International Journal of Geographical Information Science* 22, 11-12 (2008), 1271–1293.

[2] Sawsan AlHalawani, Yong-Liang Yang, Peter Wonka, and Niloy J Mitra. 2014. What makes London work like London?. In *Comp. Graph. Forum*, Vol. 33. Wiley Online Library, 157–165.

[3] Daniel G Aliaga, Carlos A Vanegas, and Bedrich Benes. 2008. Interactive example-based urban layout synthesis. *ACM Trans. on Grap.* 27, 5 (2008), 1–10.

[4] Michael Batty. 2007. *Cities and complexity: understanding cities with cellular automata, agent-based models, and fractals.* The MIT press.

[5] Marie-Paule Cani, Takeo Igarashi, and Geoff Wyvill. 2008. *Interactive Shape Design.* Morgan & Claypool Publishers, ISSN:1933-8996. 78 pages. https://hal.archives-ouvertes.fr/hal-00336304

[6] Guoning Chen, Gregory Esch, Peter Wonka, Pascal, Müller, and Eugene Zhang. 2007. Interactive procedural street modeling. *ACM Trans. on Grap.* 27, 3 (2007), 35. https://doi.org/10.1145/1278780.1278822

[7] Keith C Clarke and Leonard J Gaydos. 1998. Loose-coupling a cellular automaton model and GIS: long-term urban growth prediction for San Francisco and Washington/Baltimore. *International journal of geographical information science* 12, 7 (1998), 699–714.

[8] Guillaume Cordonnier, Eric Galin, James Gain, Bedrich Benes, Eric Guérin, Adrien Peytavie, and Marie-Paule Cani. 2017. Authoring Landscapes by Combining Ecosystem and Terrain Erosion Simulation. *ACM Transactions on Graphics - Siggraph 2017* 36, 4 (2017).

[9] Guillaume Dewaele and Marie-Paule Cani. 2004. Interactive global and Local Deformations for Virtual Clay. *Graphical Models* 66, 6 (2004), 352–369.

[10] Arnaud Emilien, Adrien Bernhardt, Adrien Peytavie, Marie-Paule Cani, and Eric Galin. 2012. Procedural Generation of Villages on Arbitrary Terrains. *The Visual Computer* 28, 6-8 (June 2012).

[11] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. 2015. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *ACM Trans. on Grap.* 34, 4, Article 106 (July 2015), 11 pages. https://doi.org/10.1145/2766975

[12] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. 2015. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Trans. on Grap.* 34, 4 (2015), 1–11.

[13] James Gain, Harry Long, Guillaume Cordonnier, and Marie-Paule Cani. 2017. EcoBrush: Interactive Control of Visually Consistent Large-Scale Ecosystems. *Computer Graphics Forum, Eurographics 2017* 36, 2 (2017).

[14] Eric Galin, Eric Guérin, Adrien Peytavie, Guillaume Cordonnier, Marie-Paule Cani, Bedrich Benes, and James Gain. 2019. A review of digital terrain modeling. In *Comp. Graph. Forum*, Vol. 38. Wiley Online Library, 553–577.

[15] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. 2003. Real-time procedural generation of pseudo infinite cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia.* 87–ff.

[16] Michael W Hancock and Bud Wright. 2013. A policy on geometric design of highways and streets. *American Association of State Highway and Transportation Officials: Washington, DC, USA* (2013).

[17] Suzi Kim, Dodam Kim, and Sunghee Choi. 2020. CityCraft: 3D virtual city creation from a single image. *The Visual Computer* 36, 5 (2020), 911–924.

[18] Thomas Lechner, Pin Ren, Ben Watson, Craig Brozefski, and Uri Wilenski. 2006. Procedural modeling of urban land use. In *ACM SIGGRAPH 2006 Research posters.* 135–es.

[19] Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. 2011. Interactive modeling of city layouts using layers of procedural content. In *Comp. Graph. Forum*, Vol. 30. Wiley Online Library, 345–354.

[20] Stephen Longay, Adam Runions, Francois Boudon, and Przemyslaw Prusinkiewicz. 2012. TreeSketch: interactive procedural modeling of trees on a tablet. In *Proc. of the Intl. Symp. on SBIM.* 107–120.

[21] Paul Merrell and Dinesh Manocha. 2008. Continuous model synthesis. (2008), 1–7. https://doi.org/10.1145/1457515.1409111

[22] Paul Merrell, Eric Schkufza, and Vladlen Koltun. 2010. Computer-generated residential building layouts. In *ACM SIGGRAPH Asia 2010 papers* (Seoul, South Korea) *(SIGGRAPH ASIA '10).* ACM, New York, NY, USA, Article 181, 12 pages. https://doi.org/10.1145/1866158.1866203

[23] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural modeling of buildings. *ACM Trans. on Grap.* 25, 3 (July 2006), 614–623. https://doi.org/10.1145/1141911.1141931

[24] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. 2007. Image-based Procedural Modeling of Facades. *ACM Trans. on Grap.* 26, 3, Article 85 (July 2007). https://doi.org/10.1145/1276377.1276484

[25] Przemyslaw Musialski, Michael Wimmer, and Peter Wonka. 2012. Interactive Coherence-Based Facade Modeling. *Comp. Graph. Forum* 31, 2pt3 (May 2012), 661–670. https://doi.org/10.1111/j.1467-8659.2012.03045.x

[26] Przemyslaw Musialski, Peter Wonka, Daniel G Aliaga, Michael Wimmer, Luc Van Gool, and Werner Purgathofer. 2013. A survey of urban reconstruction. In *Comp. Graph. Forum*, Vol. 32. Wiley Online Library, 146–177.
[27] Gen Nishida, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive sketching of urban procedural models. *ACM Trans. on Grap.* 35, 4 (2016), 1–11.
[28] Wociech Palubicki, Karl Horel, Stephen Longay, Adam Runions, Brendt. Lane, Radomír Měch, and Prezemyslaw Prusinkiewicz. 2009. Self-organizing Tree Models for Image Synthesis. *ACM Trans. on Grap.* 28, 3, Article 58 (2009), 10 pages.
[29] Yoav I. H. Parish and Pascal Müller. 2001. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, 301–308. https://doi.org/10.1145/383259.383292
[30] Juval Portugali. 2009. *Self-Organization and the City*. Springer New York, New York, NY, 7953–7991. https://doi.org/10.1007/978-0-387-30440-3_471
[31] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomir Mech. 1996. L-systems: from the theory to visual models of plants. In *Proceedings of the CSIRO*, Vol. 3. Citeseer, 1–32.
[32] Oriol Pueyo, Albert Sabrià, Xavier Pueyo, Gustavo Patow, and Michael Wimmer. 2020. Shrinking city layouts. *Computers & Graphics* 86 (2020), 15–26.
[33] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. 2007. Modeling Trees with a Space Colonization Algorithm. *EG Nat. Phenom.* (2007), 63–70.
[34] Michael Schwarz and Pascal Müller. 2015. Advanced Procedural Modeling of Architecture. *ACM Trans. on Grap.* 34, 4 (2015), 107:1–107:12.
[35] Ruben M Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. 2014. A survey on procedural modelling for virtual worlds. In *Comp. Graph. Forum*, Vol. 33. Wiley Online Library, 31–50.
[36] Jing Sun, Xiaobo Yu, George Baciu, and Mark Green. 2002. Template-based generation of road networks for virtual city modeling. In *Proceedings of VRST*. 33–40.
[37] Carlos A Vanegas, Daniel G Aliaga, Bedrich Benes, and Paul Waddell. 2009. Visualization of simulated urban spaces: Inferring parameterized generation of streets, parcels, and aerial imagery. *IEEE Trans. on Vis. and Comp. Graphics* 15, 3 (2009), 424–435.
[38] Carlos A. Vanegas, Daniel G. Aliaga, Bedřich Beneš, and Paul A. Waddell. 2009. Interactive design of urban spaces using geometrical and behavioral modeling. (2009), 1–10. https://doi.org/10.1145/1661412.1618457
[39] Carlos A Vanegas, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Paul Waddell. 2012. Inverse design of urban procedural models. *ACM Trans. on Grap.* 31, 6 (2012), 1–11.
[40] Carlos A Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel G Aliaga, and Pascal Müller. 2012. Procedural generation of parcels in urban modeling. In *Comp. Graph. Forum*, Vol. 31. Wiley Online Library, 681–690.
[41] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. 2006. Vector Field Based Shape Deformations. *ACM Transactions on Graphics - SIGGRAPH 2006* 25, 3 (2006).
[42] Paul Waddell. 2002. UrbanSim: Modeling urban development for land use, transportation, and environmental planning. *Journal of the American planning association* 68, 3 (2002), 297–314.
[43] Basil Weber, Pascal Mueller, Peter Wonka, and Markus Gross. 2009. Interactive Geometric Simulation of 4D Cities. *Comp. Graph. Forum* (April 2009). http://www.procedural.com/publications/2008_EG_Urban_Simulation/2008.EG.Weber.UrbanSimulation.Paper.pdf
[44] Nora S Willett, Rubaiat Habib Kazi, Michael Chen, George Fitzmaurice, Adam Finkelstein, and Tovi Grossman. 2018. A mixed-initiative interface for animating static pictures. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 649–661.
[45] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. 2003. Instant architecture. *ACM Trans. on Grap.* 22, 3 (2003), 669–677. https://doi.org/10.1145/882262.882324
[46] Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor. 2016. Object-oriented drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 4610–4621.

APPENDICES

A JOBS AND POPULATION PER LOT

Population and jobs are initially represented as distributions of discrete values on a fixed resolution grid (see Figure 2 (e-f)). However, to interact with them locally, it is necessary to convert them to a distribution over the city lots. We use the same algorithm described below for population to calculate the exact amounts of population $P(p_i)$ and jobs $S(p_i)$ in each lot p_i.

Let N_c and N_p be respectively the total number of grid cells and lots. Let the cells be denoted by $c_i, 1 \leq i \leq N_c$ and the lots by $p_i, 1 \leq i \leq N_p$. We use $C(x)$ to refer to the contour polygon of x, $P(x)$ for the amount of population in x and $A(C)$ to refer to the area within a closed contour C (see Figure 20).

A lot is habitable if it is not a park (*i.e.,* it includes a building). The total habitable area (THA) of each grid cell (Figure 20) is:

$$\forall i \in [1, N_c], \qquad THA(c_i) = \sum_{p_j | p_j \neq \text{park}} A(C(p_j) \cap C(c_i)).$$

Under the hypothesis of locally uniform distribution, the population $P(p_i)$ of a lot p_i is then:

$$P(p_i) = \sum_{c_j} P(c_j) \frac{A(C(p_i) \cap C(c_j))}{THA(c_j)}.$$

If a grid cell has a positive population but has no habitable lot inside or intersecting its contour, the transfer is impossible, and we discard it. By treating the distribution of the jobs in the same way, we obtain an urban layout where the amount of people and jobs in each building (or a lot) is known.

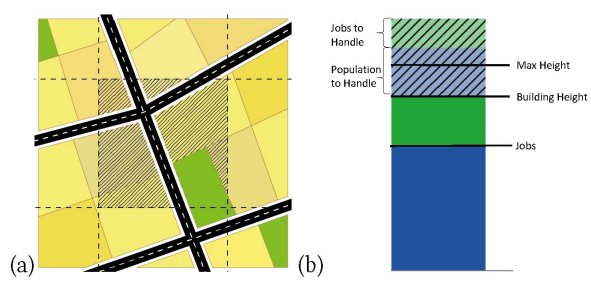

Figure 20: For a given grid cell (dotted lines), the total habitable area is the total area covered by lots that are not parks (shaded area) (a). We represent a building's four variables: population (solid blue), jobs (solid green), population to handle (shaded blue), jobs to handle (shaded green) (b).

B BRUSH ALGORITHMS

B.1 Repulsor

The *Repulsor* brush removes a specified amount of population from the lots in the brush center and moves it to the influence area without modifying the road and block geometry. The algorithm is as follows.

First, the brush sorts all affected lots by distance to the brush center (using Euclidian or traveling-time distance). Then, the total number of agents to relocate is computed, as well as the amount each lot in the influence region should receive to preserve the total

agent-volume (determined based on user-defined weighting scheme *uniform* or *gaussian*). Naturally, at this stage, if a lot in the influence region has already reached its maximal capacity or is locked (using the *Mask* brush), no amount will be assigned to it.

We start the transferring by pairing the first sorted impact lot and the first influence lot. If the influence lot absorbs all population from the impact lot, we pair the current influence lot with the next impacted lot. Otherwise, if the building in the influence lot achieves its max height or absorbs the max amount of population, the influence lot will be iterated to the next in the sorted queue.

While the city is consistent after the brush has visited all lots in both regions, we cannot guarantee that every lot in the influence region absorbed its part of the impact region's population excess. Indeed, a lot could have been assigned more agents to receive than it can, so there might be more population remaining in the impact region than desired after the process (since by construction of the *Transfer* operation, any not handled agent is returned into its original lot). As a solution, we reiterate the whole process until no significant mobility between both regions is observed.

B.2 Attractor

The *Attractor* brush acts intuitively as the opposite of *Repulsor* by pulling agents in. The *Attractor* first sorts the lots by distance from the brush center (Euclidean or traveling-time). It computes the total number of agents to be added to the buildings in the impact region from a user-specified value. Then it calculates the number of agents that each lot in the influence region has to provide based on the previous total amount. It iterates simultaneously over the lots in the impact and the influence region and transfers agents based on the *Transfer* operation in a way similar to the *Repulsor*. Also similar to the *Repulsor*, the process is iterated if necessary.

C HIERARCHICAL ROAD GENERATION

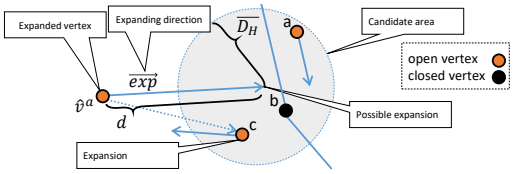

Figure 21: Single arterial node expansion: Starting from $\hat{v}^a$, a possible expansion is determined by choosing as a new connection, among compatible nodes (a, b, c) which are open (a, c), the most aligned one (c).

The road generation is hierarchical. We first expand the arterial roads and then the streets. All vertices from the impact area of the brush are marked as **open** (denoted by the red circles in Figure 21). Vertex that cannot be further expanded is denoted as **closed** (a black dot in Figure 21). A vertex is closed when it is already fully connected by four roads, or it has an invalid geometry attribute that makes it unable to be connected, like a vertex on the river.

We keep two lists of open vertices. A vertex that includes vertices with an open connection of type e^a is stored in a list for open arterial roads $\hat{V}^a$. An open vertex that includes *only* open connections to streets e^s is stored in open streets $\hat{V}^s$. Note that $\hat{V}^a$ may include open nodes also for streets. They will be relocated to $\hat{V}^s$ once the arterial connections have been made.

If a road segment has been previously disconnected from a vertex, we keep the original direction, because it may be used for a future re-connection as shown in Figure 22.

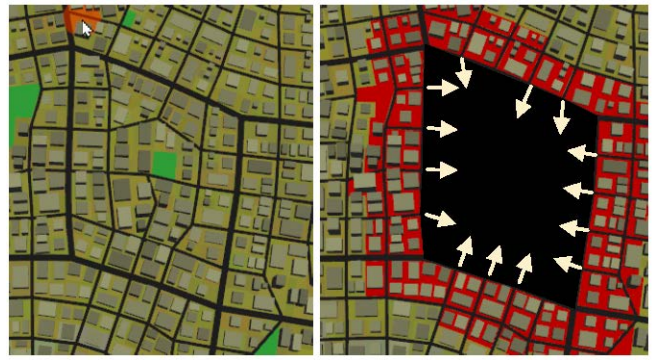

Figure 22: Each intersection stores the direction of the connecting roads. If the area and the roads are removed, the directions are kept for potential reconnect to keep the layout consistent.

Arterial roads generation: We randomly chose one vertex $\hat{v}^a$ from $\hat{V}^a$ and pick one of its open connections $\hat{e}^a$. First we determine the direction of the expansion $\overrightarrow{exp}$ (see Figure 21). If the connection comes with the direction information, *i.e.,* it has been disconnected before, and the direction is stored as shown in Figure 22, we use it. If the direction is unknown, we sample the direction histogram J_H for the compatible type. We then sample the distance histogram D_H to determine the distance of the expansion d. The new possible expansion is then located at $\hat{v}^a_{new} = \hat{v}^a + d\ \overrightarrow{exp}$.

Before expanding the road to $\hat{v}^a_{new}$, we check if there are other compatible intersections in the vicinity. The candidate area is a disk at $\hat{v}^a_{new}$ with radius $\overline{D_H}$. The radius is calculated as the average of all distances between intersections of the same type (a-a in this case). If compatible nodes are present (a and c in Figure 21, because node b is closed), we do not generate a new one, but instead, we connect to an existing one. Any node inside the disk is suitable, but we chose the node with the most aligned direction (node c in Figure 21) to preserve style. If no compatible node is present, we generate a new open node at $\hat{v}^a_{new}$. Its type is determined by randomly sampling vID_H.

If the expanding segment intersects with another road, it is removed, and the node expansion is terminated. If the expanded node does not have any open connections e^a, but it has open connections e^s, it is moved from $\hat{V}^a$ to $\hat{V}^s$. All closed nodes are removed from $\hat{V}^a$. The expansion of arterial nodes ends when $\hat{V}^a$ is empty. If dead-ends are present, they are removed.

Streets generation: At the end of the arterial expansion, $\hat{V}^s$ still includes nodes to be expanded. They are processed as street nodes by using the same algorithm as for arterial expansion.

However, we do not leave any open node during this pass because there is no sub-category, the streets being the lowest roads in the hierarchy.