City Sketching

James Gain
University of Cape Town
jgain@cs.uct.ac.za

Patrick Marais
University of Cape Town
patrick@cs.uct.ac.za

Rudolph Neeser
University of Cape Town
rudy.neeser@gmail.com

ABSTRACT

Procedural methods offer an automated means of generating complex cityscapes, incorporating the placement of park areas and the layout of roads, plots and buildings. Unfortunately, existing interfaces to procedural city systems tend to either focus on a single aspect of city layout (such as the road network) ignoring interaction with other elements (such as building dimensions) or expect numeric input with little visual feedback, short of the completed city, which may take up to several minutes to generate.
In this paper we present an interface to procedural city generation, which, through a combination of sketching and gestural input, enables users to specify different land usage (parkland, commercial, residential and industrial), and control the geometric attributes of roads, plots and buildings. Importantly, the inter-relationship of these elements is pre-visualized so that their impact on the final city layout can be predicted. Once generated, further editing, for instance shaping the city skyline or redrawing individual roads, is supported. In general, City Sketching provides a powerful and intuitive interface for designing complex urban layouts.

Keywords

sketching interfaces, procedural modeling

1 INTRODUCTION

Procedural methods for simulating terrain, buildings, plants and cities have been used effectively in computer games, visual effects and virtual environments for training and simulation. This loose family of computational methods includes L-systems [1], noise functions [2], shape grammars [3] and other algorithms characterised by recursive self-affine behaviour. Their benefit is that with little manual effort, complex and realistic content can be generated. In the most extreme instance an entire virtual world can burgeon from a single pseudo-random seed.

However, there is a tension between the productivity that arises from extensive automation and the control provided by user involvement. Ideally, users should be able to dictate key aspects of a scene with as fine a granularity as desired, and the attendant procedural method should follow these specifications. There are several strategies for achieving this, some more successful than others.

The most prosaic procedural interface is a set of numeric control parameters. Unfortunately, this type of interface tends to expose the internals of the procedural method and fails to foster any geometric intuition on the part of the user. More effective are image maps, where regions of a landscape are painted with different properties, such as landforms with particular frequencies [4], the distribution and density of plants or categories of land usage in cities [5]. Here there is a direct visual mapping to the domain but the context of use is somewhat limited. Another option is a textual approach in which adjectives are used to describe a scene and then mapped via machine learning to procedural parameters. This approach is generally preferred to parameter specification by inexperienced users [6] but it does require an extensive training phase. Recently, procedural techniques have begun to borrow from real-world data using example-based synthesis. As long as such real-world data-sets are readily available, as is the case for city [7] data, and the intended results do not diverge significantly from the exemplars, this can lead to unprecedented realism.

Recent work has resulted in some early direct manipulation of procedural models for buildings and trees, which has been extended to the direct manipulation of road networks [8].

Another alternative is to develop a sketching interface. Inspired by pencil-and-paper illustration, these interfaces are accessible to non-experts and enable rapid iterative design, particularly where absolute precision is not required. They have been applied with success to the procedural modeling of terrain, trees, flowers and clothing.

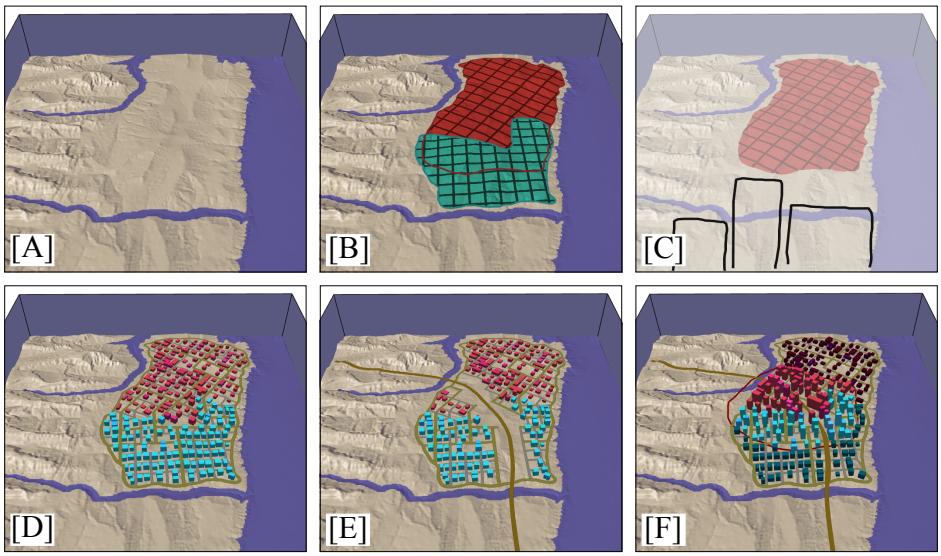


Figure 1: Generating urban layouts by City Sketching: [A] Given an existing landscape; [B] the user sketches road pattern and land usage zones, and [C] a gesture interface is used to interpret the type of zone and its statistics; [D] a corresponding city is generated; further iterative editing of roads [E] and buildings [F] is supported.

Turning to the particular case of procedural cities, the purpose here is to create an urban layout composed of a road network, plot subdivisions and building envelopes. This may extend to procedurally modeling the facades of individual buildings, but, of late, this has devolved to a separate subfield [3]. While seminal work [5] uses broad proxy parameters (road pattern type and population density) coupled to image map inputs, the field has since adopted both example-based methods [7] and simulation [10] in the interest of increased variety and realism. With some exceptions (as noted in Section 2) interfaces have not kept pace with the increasing sophistication of city simulation and are generally still limited to direct parameter specification and image maps.

Motivated by the failings of existing procedural city interfaces and the ease-of-use, familiarity and speed of sketching, we have developed *City Sketching*. The fundamental operation in our system is marking out regions on a landscape (see Figure 1[A, B]) to represent road patterns and land usage zones. There are two important considerations here: firstly, a full set of statistics is automatically calculated for each region (for example, land usage will not only have a type — commercial, industrial, residential or parklands — but also a mean, minimum and maximum for plot size, building base and height, or appropriate equivalents), and, secondly, the final city is determined by the interaction between the different regions (roads conform to the landscape, plot sizes determine separation between roads, building heights influence road widths). We provide visual feedback of these interrelationships using procedural textures projected onto the landscape. Further to the high-level control provided by such region drawing, users are also able to directly sketch individual roads at a low-level.

Once a user is satisfied with their initial layout, the region statistics and explicit road constraints are passed to a procedural system that creates road networks, plot subdivisions and individual building placements (Figure 1[D]). After the city has been created it can be iteratively modified by sketching new roads and regions (Figure 1[E]), followed by a localized procedural update. Furthermore, the characteristic skyline of the city can be shaped by raising and lowering buildings to fit a side-view sketch (Figure 1[F]), or more directly by interactively manipulating building heights in a region.

To summarize, the key contribution of this work is a holistic approach to procedural city generation that allows users to specify and visualize not only road networks, but also categories of land usage, dimensions of plots and buildings and their interrelationships.

The system has a number of novel components: (a) A gesture-based component that extracts both zone information and statistics for roads and buildings from sketched exemplars. (b) Interactive pre-visualization, which shows the interrelationship between road patterns and region usage and their effect on the final city. (c) A procedural engine that grows the entire city and supports interaction between all elements, and the inclusion of user specified roads as constraints.

The remainder of the paper is structured as follows: Section 2 provides more detailed coverage of previous interfaces to procedural city systems; Sections 3 and 4 address the interface design and associated procedural engine; Section 5 examines the system's performance

in terms of usability and versatility; and, finally, Section 6 provides a summary and recommendations for future work.

2 RELATED WORK

As previously mentioned, most procedural city systems employ a combination of numeric parameter input and image maps [5, 12, 10]. These image maps are often generated using a raster paint application and supplied as separate input files, making it difficult for a user to create an integrated mental map of the final layout. Some systems do support the drawing of simple road constraints [10] but, in any event, rarely go far enough in allowing interactive specification and previsualization of city layout. This is less problematic for fast system, where the generated city serves as its own visualization, but can cause frustrating design cycles for simulation-heavy methods, which typically take several minutes to execute.

It is also worth considering other procedural domains, such as plant and terrain modelling, be they painting [4] or sketching [9] interfaces. The key issue is how regions are specified in these interfaces. Sketching interfaces generally allow the user to draw a closed loop, which works well for large contiguous regions without holes. In contrast, painting interfaces require longer to demarcate a region but allow complex topology. While these interfaces are a source of inspiration, our system goes beyond marking out regions and visualizes how regions overlap and interact, the specification of constrained linear features such as rivers and roads, and the sketch-based input of geometric region parameters for roads and buildings.

There are two recent techniques that counter this trend. Aliaga et al. [7] develop an example-based approach that allows a city to be assembled from fragments consisting of vector data (attributed point sets representing the junctions of road) and aligned images (georeferenced aerial photography of city blocks). From an interface perspective, fragments can be copied and pasted and then synthesized into a coherent urban layout via *join*, *blend* or *expand* operations depending on their spatial arrangement. Although the technique focuses on mimicking aerial imagery, there is no reason, in principal, that it could not be extended to create a fully geometric realization of a virtual city. The only weakness with this kind of structural synthesis is the reliance on existing annotated and registered data.

We were also inspired by the road network design of Chen et al. [13]. In their system a 2D tensor field is shaped by boundary, pattern, heightfield and density constraints, and edited with smoothing, noise, rotation and brush operations. Finally, roads are traced along the streamlines of the major and minor eigenvector fields. These results are passed to a separate engine for splitting into block, plots and buildings, which precludes any real feedback between building data and the road network, as is achieved by our system.

Lipp et al. [8] show how road networks can be edited using direct manipulation, including copy and paste functionality. These operations preserve road network validity (such as preventing unwanted intersections), and persist even after the road network is regenerated.

Galin et al. [11] address road networks between cities using a hierarchical approach, starting from highways and progressing to secondary roads, with graph optimization that respects terrain and water features. Their interface relies on the now familiar image map painting.

For a more complete overview of approaches to the modelling and simulation of urban environments, see Vanegas et al. [14].

3 INTERFACE

There are two aspects of city design that receive separate consideration in our sketching interface: (a) closed region strokes are drawn onto the landscape to represent *usage zones and road patterns* with a gestural interface for capturing various region statistics; and (b), once an initial city has been generated, further editing is supported, such as drawing in extra roads, constraining skylines from separate viewpoints using sketched polylines, and altering building heights. Distinct forms of sketching are appropriate in each case. Care is taken in the interface to provide support for iterative refinement. For instance, users are able to refine a particular portion of a curve or loop by oversketching or direct manipulation, with the option of shifting to a more favorable viewpoint during the process. Furthermore, users can "undo" previously committed sketches using a scratch gesture. In our system, city and landscape features are generally sketched from an aerial viewpoint with strokes and points in the 2D image plane projected onto the 3D terrain. The resulting features are then variously interpreted as usage zones, road patterns or road constraints, depending on the selected interface tool.

3.1 Usage Zones and Road Patterns

City Sketching, in common with other procedural city systems, controls city attributes, such as zoning and road arrangement, through the shaping and placement of contiguous regions. However, we diverge from previous approaches in several respects. First, our system exposes a broad spectrum of detailed geometric parameters for roads, buildings and plots. We favor a geometric approach because it promotes visual depiction and thus has greater predictability than abstract quantities such as population density and wealth. Second, these parameters are controlled through a combination

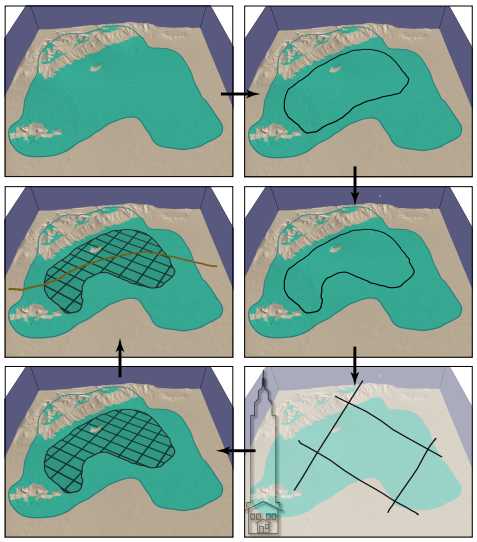


Figure 2: Sketching of zones and roads: inside a previously sketched commercial usage zone (blue-green) the user draws a closed region stroke onto the terrain; this can be modified by oversketching; the road pattern (a grid) and statistics (orientation of major and minor gridlines) are inferred from gestures; the zones and their interactions are visualized using procedural textures; roads (such as a highway) can also be sketched directly.

of sketching (to specify the outlines of regions) and gestures (to capture specific region information) — Figure 2.

City Sketching users mark out two types of regions on the landscape. *Usage Zones* represent typical urban zoning classifications (commercial, industrial, residential or parklands) and also capture the size and density of buildings (or trees in the case of parklands); specifically the minimum, maximum and mean for building height, building length and plot length. *Road Patterns*, on the other hand, represent the dominant layout of roads in a region (grid, radial or random) and also encode various style parameters, such as the angular separation between radiating avenues in the case of the radial style, or the aspect ratio of blocks in the grid style.

The final city layout is determined by the interaction of usage zones and road patterns. Thus, inter-road spacing is determined by plot size, road widths by building volume and density (approximating traffic intensity), and the incidence at road junctions by the road pattern. This enables nuanced control over an urban layout. For example, a rich residential suburb with relatively narrow roads and large single-story buildings on larger plots can be specified with our system — a level of control difficult to achieve using less detailed parameters such as population density.

However, two challenges arise: the need for a visual depiction of the interaction between regions, and for a simple, fast and effective means of entering region statistics. The problem of visualization is overcome by creating procedural textures that are stencilled appropriately to the intersection of a usage zone and road pattern region and mapped onto the landscape. While these textures cannot capture the full complexity of a procedural system, as presented in Section 4.2, they are sufficient to convey the directionality, distribution and style of the final road layout. We use gestures to address the problem of capturing region information. While the field of gestural input is broad and encompasses aspects such as communication through hand motion and body language, one commonality is the compact encoding of symbolic information. In our context a set of sketched symbols is used to indicate the type of each region (see Figure 3). If that was the limit of our gesture functionality then a simple button or menu interface would suffice. However, our gestures also serve as exemplars from which region statistics are derived.

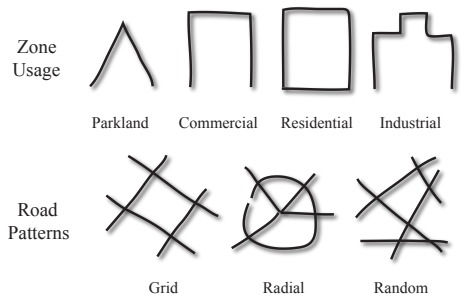


Figure 3: Gestures: simple gestures are used to indicate zone usage (park, residential, commercial, or industrial) or road pattern (grid, radial, or random). Variation in the gestures, such as spacing and aspect ratio are also used to capture zone statistics.

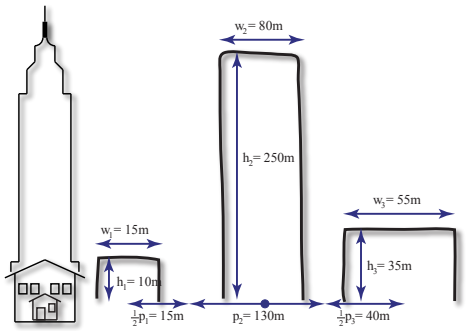


Figure 4: Extracting statistics from gestures: three exemplars are drawn for each land usage zone (in this case commercial) and their aspect ratio and relative separation are used to derive statistics (minimum, maximum and mean) for building height (h_1, h_2, h_3), building width (w_1, w_2, w_3) and plot width (p_1, p_2, p_3). Scaling in the vertical dimension is piecewise linear in order to allow both skyscrapers and single-story buildings on the same gesture canvas.

For usage zones we expect users to draw three gestures of the same type. As shown in Figure 4, the dimensions of the bounding boxes around each gestures and their relative separation are then used to derive statistics for building length, height and plot length. Specifically, the values for a particular parameter (e.g., plot length) are sorted from smallest to largest and then mapped to the minimum, mean and maximum, respectively. One complication is that the full range of building heights (from 5m to 400m) cannot reasonably be drawn to a linear scale on a single canvas. Instead, we utilize a piecewise linear mapping with a relatively small gradient up to 20m and a steeper slope thereafter. The key on the left of Figure 4 serves users as a visual indicator of this scale.

Road pattern layout statistics are derived as follows: for grids, the direction of major/minor axes and the aspect ratio of city blocks; for radial, the angular distribution of the radial spokes and their center; for random, the range of angles between intersecting roads. Of course, the exemplars could be followed more explicitly, but this would require a more deliberate and less natural style of interaction.

Finally, while road pattern gestures provide the template for an entire region, sometimes users wish to precisely control the placement of roads. For this purpose, City Sketching provides the capability to draw primary, secondary or tertiary roads directly onto the terrain, as demonstrated in Figure 2. When the procedural city is generated road constraints, corresponding to the user-drawn roads, are stitched into the prevailing road pattern.

3.2 Building Editing

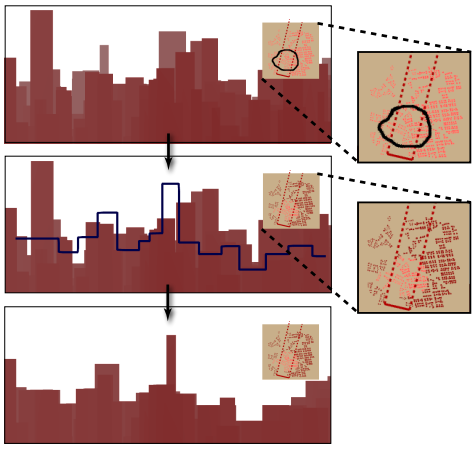


Figure 5: Sketching skylines: from an orthographic side-on view the user selects a subset of buildings on an inset mini-map; then draws a characteristic skyline, which is automatically converted into a piecewise representation; finally, building heights are adjusted to conform to the skyline.

Roads can be drawn directly onto the terrain and then passed as explicit constraints to the procedural engine, and it is desirable to have analogous control over buildings. This is provided in two ways: in *skyline* mode where the characteristic skyline of the city can be sketched from a side-on orthographic view, with building heights conforming to the resulting constraint envelope (see Figure 5), and in *elevation* mode, by demarcating collections of buildings from an aerial perspective and then directly raising or lowering them.

In skyline mode, users first select a group of buildings to edit by drawing an enclosing loop on an inset contextual map; they can then draw and refine a silhouette, which is automatically converted into a step-wise representation (effectively, a piecewise constant function). Once committed, building heights are adjusted to the minimal extent necessary to meet the skyline constraint, while taking cognizance of the region statistics (as discussed further in section 4.3). Note that building footprints remain unaltered, because to do otherwise would require re-organising the street layout.

From a rendering perspective we chose flat-shading, so that users would focus predominantly on building silhouettes, and a fog effect to provide some depth context. We found an orthographic projection to be essential in preserving the relative scale of buildings.

While this approach works well for moderate changes in terrain altitude, it fails for cities built on or around hills and mountains (such as San Francisco). In extreme cases buildings might be entirely culled, since their foundations lie above the sketched skyline. We considered a number of alternatives — projecting buildings onto a plane for the purposes of Skyline editing or allowing users to select buildings in bands of altitude — but ultimately we opted to simply filter out buildings with a base situated above a user-defined threshold. Skyline mode is primarily intended for shaping the city vista from a particular view direction and this is damaged by alternative implementations where there is only an indirect correspondence between the sketched skyline and the final results.

In elevation mode, the user selects a set of buildings either by radial distance from a pick point or by drawing a region loop (using the same process as land usages and road patterns). The selected buildings can then be raised or lowered by a constant amount, with a smooth tailing away of elevation change towards the edges of the selection, so that the alterations blend with the surrounding city (see Figure 1[E]). Any building whose new height falls outside the bounds specified by the region statistics is highlighted.

4 IMPLEMENTATION

4.1 Gesture Recognition

The gesture recognition subsystem is responsible for categorizing one or more user-sketched input strokes and extracting appropriate parameters. For our purposes, each stroke is a sequence of point samples drawn in one continuous motion. Gesture classification is usually based on a set of extracted features, with machine learning applied to deal with inevitable variability. Extensive training on example gestures is usually required and, although the recognition rates are generally high (95% or greater), these systems can also produce non-negligible false positive rates [15].

However, since our interface need only differentiate between a limited set of 4 zone usage gestures, methods based on machine learning were deemed an unnecessary overhead. Instead, we have developed a simple and robust ad-hoc approach with satisfactory performance (see Section 5). Specifically, we assume that gesture strokes are piecewise linear curves defined on the unit square, and that the interface designer will supply a set of *gesture exemplars* for each zone usage type. To determine if a set of point-sampled strokes matches a valid usage exemplar, we proceed as follows: First, we filter the point set to remove drawing jitter. We then apply a simple line aggregation algorithm to collapse multiple consecutive line segments, which results in either a single larger line segment or a poly-line, depending on the curvature of the stroke. Finally, we compute a simple *feature vector*: a list of angles between adjacent line segments. A similar vector is also computed for each exemplar during initialization. Classification entails first matching the number of segments (since each usage zone gesture is unique in this respect) and then testing the absolute difference between components of the feature vector to see if the gesture falls within the broad range of permitted angular variation. If a gesture is not recognized as a usage zone we test it against the set of road patterns.

A road pattern template also consists of a series of strokes arranged in specific geometric relationships. For example, a Manhattan grid has two sets of roughly parallel lines, which are approximately orthogonal, while a Paris-style radial road network has concentric ring-roads with spoke-avenues radiating from a central area.

In order to classify a road pattern, each input stroke is first categorised as a line, polyline or (approximate) circle. We then compare the set of input strokes against each road pattern template in turn, until a match is found. If no match is found, the collection of strokes is flagged as unclassifiable. The tests required for a match depend on the template under consideration. So, for the radial pattern, the stroke set must contain both lines and at least one circle, but no polylines. If multiple circles are present, the circles are tested to see if they are roughly concentric. Finally, lines are examined to see if they start within the innermost circle and cross outwards. At least two lines are required to estimate angular separation. If all these tests are passed, the set of strokes is matched and the appropriate parameters extracted. Otherwise we attempt to match against the next pattern template.

4.2 Procedural City Generation

Our goal is to create a simple and efficient means of generating road networks, parcel allotments and building envelopes, based on the statistical information provided by a city sketch. The system outlined here is just one of several possible procedural back-ends that could be fed by our sketching front-end. Our approach is inspired by the principles of L-system road generation [5], including context awareness and non-determinism. However, in practice, we found L-systems to be very slow for large cities, because they require production matching over the entire string, which grows exponentially. Instead, we use *growth seeds* to represent potential road segments. These are placed initially at the centroids of each unique combination of road pattern and usage zone. Should the centroid not lie inside the corresponding region, we use the center of the largest inset square instead. A growth seed derives its direction, length and road width from the underlying region statistics. For example, road length within a grid pattern depends on average plot width, block aspect ratio, and alignment with the major or minor axis. As a further example, road width varies according to the average building volume normalized by plot area as a proxy for population density. Road growth also adapts to the terrain, either avoiding or conforming to slope contours depending on the severity of the incline. To lessen the impact of high frequency fluctuations we first smooth the terrain before calculating slope. In regions with a slope greater than 2.86 degrees from horizontal, roads are aligned with the terrain contours (the so-called San Francisco pattern [5]) and we prevent road placement altogether where the slope is greater than 5.7 degrees, corresponding to a 10% grade limit used in international road codes.

Growth seeds are reoriented as they approach existing roads, using the heuristics outlined by Parish and Müller [5], in order to connect at junctions and avoid malformed blocks. As a seed lays down roads it may, depending on the needs of the road network, spawn further growth seeds at junctions.

There are two issues with this approach: handling user-drawn road constraints and merging road networks at region boundaries. It is important that individual roads sketched by users do not unduly distort the underlying road pattern. Unfortunately, the obvious strategy

of growing patterns outwards from seeds placed along a sketched road is foiled by any form of curvature in the sketch. Instead, we stitch in user-drawn roads by first growing the road network to completion, then deleting existing roads within an offset distance of the user constraint, followed by laying a new road along the constraint curve, and finally initiating growth seeds at the newly formed deadends in the existing network, with parameters copied from incident deleted roads. A memory of the prior road pattern is thus retained, so that it is not unduly disrupted during regrowth. In this way the network regrows with a similar pattern that adapts where necessary by snapping to the road constraint. A similar process is used for filling new regions in an existing city.

The second challenge lies in correctly switching between road patterns at region boundaries. It is not sufficient to reorient seeds as they transition between regions because, even if their directions align with existing roads in the current region, this does nothing to match road position and spacing. Rather, we delay growth seeds as they cross region boundaries. In this way existing growth within a region can continue undistorted and the approaching fronts of two road patterns will meet correctly at the boundary. We also place different categories of road into separate queues and process each to completion in priority order (highways, arterials, then streets), ensuring that higher priority roads can extend to completion without being blocked by those with lower priority.

Once the road network is in place, we extract blocks by a simple winding process. We then generate buildings by first subdividing an input block into parcels and the placing a building bounding box within each parcel. The bounding box is a cuboid which can be replaced with the appropriate building geometry.

The parcel splitting procedure must run quickly and produce sub-plots which span the range of plot sizes specified by the usage zone sketches (see Figure 4). This provides a single axis for building width and spacing (and thus, implicitly, plot size), from which infer a range of values for plot size that constrain the allowable size of bounding boxes generated by parcel splitting. No explicit provision is made to produce or control irregular sub-plots; they arise naturally along curved boundaries. The plot parameters do not obey a specific distribution: they are merely bounds which constrain how large or small a plot should be. Since we employ a recursive splitting procedure, explained below, and this will tend to split plots as much as possible, we allow a random chance (50%) of terminating splitting early, as long as the current plot under scrutiny is not bigger than the largest specified plot size.

Our parcel generation phase is a modification of the method used by Parish and Müller [5]. First, we build an oriented bounding box for the input polygon. We then split the input polygon in the direction of the longest axis of the bounding box, allowing a small random perturbation in the range (0.4,0.6) for the origin of the split line along the shortest box axis. The splitting process continues recursively: each new polygon is split orthogonally to an edge of the polygon which maximizes our splitting criterion. This criterion gives high weight to long edges as well as edges which have immediately adjacent edges which are near orthogonal to the edge under consideration. The origin of the splitting line is jittered to ensure that we do not always split from the centre of an edge. We effectively ignore concavity by taking the first two line intersection points with the polygon loop. This is guaranteed to yield two polygons, regardless of boundary complexity. This allows concave plots if the initial polygons is concave, but since these occur in reality this is not an issue. We disallow splits which would generate new polygons with oriented bounding boxes that fall below our minimum parcel size requirement. A new parcel must be large enough to accommodate the smallest building as well as a prescribed margin around each building. We also disallow splits which would generate parcels with no street access.

For each we estimate the largest interior box, using a sampling approach: we look inwards from each polygon edge and try to find the largest contained box using a number of ray intersections centred on each edge. While this is a rather crude approximation, it is reasonably cheap (for small numbers of rays) and yield boxes oriented along edges, promoting street access. We then shrink the box to fit the constraints supplied by the sketch system. We do this by generating a random plot margin for each box axis and determining whether the resulting (reduced) box obeys the constraints on building size. If it does, we accept the reduced envelope as the building base; otherwise we try with the minimal margin constraint. If this fails, we flag that parcel as being too small to contain a valid building. Once we have generated a suitable building, we impose a global aspect ratio check to shrink plots which have exploited the maximal plot size to aggressively. Currently we do not allow buildings with an aspect ratio above 3. Finally, we generate a uniform random height between the height constraints supplied for that input block.

4.3 Building Editing

The skyline subsystem takes a sketched skyline curve and must then ensure that the final orthogonal projections of all selected buildings conform to the implied envelope. Our solution obeys the following principles: (a) the fewest possible buildings are altered, (b) heights after modification remain in the min/max range attached to buildings, (c) buildings above the skyline constraint are lowered, while those lying below are only

raised as necessary to cover the skyline, and (d) the procedure executes at interactive rates.

To enforce the skyline constraint, we decompose the piece-wise constant skyline envelope into a sequence of zones on the projection plane. Each zone is simply a disjoint rectangular region in the plane with a constant height constraint that impacts all buildings projecting into that zone. Of course, buildings often project across adjacent zones. In keeping with principle 3, such a building is lowered, if necessary, to the minimum height of the zones it spans.

In order to satisfy principle 1, we develop the idea of *zone coverage*. This refers to the proportion of a zone's width that is covered by the union of the bases of buildings whose heights have been fitted to that zone's skyline. Thus the potential coverage of a building is the intersection of its projected extent with the zone's extent, but this coverage is only realized if its height is altered to fit the zone. To keep building edits to a minimum, we first raise those buildings which contribute a large proportion of unique coverage to any given zone. To do this we order the height edits for each zone by potential coverage. We then select legal edits for each zone until either we have no edits remaining or some prescribed coverage fraction has been attained. In practice the piece-wise constant height constraint can lead to skylines that are too sharply defined to be natural. We address this by allowing a certain "fuzziness" in associating buildings with zones. More precisely, we shrink the building extent by an ε in either direction for the purpose of computing zone associations. The resulting skyline has a rougher, more approximate appearance while, in essence, still conforming to the sketched constraint. To further enhance variation, we add a small random offset to building height edits, which allows them some variation below the zone height threshold.

By default, the skyline is enforced on all buildings in the selected region. Sometimes, however, a user may wish to edit only those buildings in the front-line closest to the view. We find these buildings using a 1D z-buffer. The base extent of each building (in the XZ-plane) is rasterized into a 1D depth buffer such that the identity of the closest buildings along the base line can be retrieved.

For elevation mode we need to ensure that height changes fall off smoothly towards the boundary of the selection. To achieve this the change in height is scaled by a weighting factor:

$$w = \begin{cases} 1 & \text{if } r > r_1 \\ f(1 - \frac{r}{r_1}) & \text{if } 0 \leq r \leq r_1 \\ 0 & \text{if } r < 0 \end{cases} \quad (1)$$

where $f(x) = (x^2 - 1)^2$, $x \in [0,1]$; r is the shortest distance to the boundary of the selection (with points inside having positive distance and those outside negative), and r_1 is the distance after which a constant height change is applied.

5 EXPERIMENTS AND RESULTS

Since the primary focus of this work is the development of a sketching and gestural interface, we concentrate in this section on the utility and ease-of-use of the final system. In this regard, we administered a number of usability experiments:

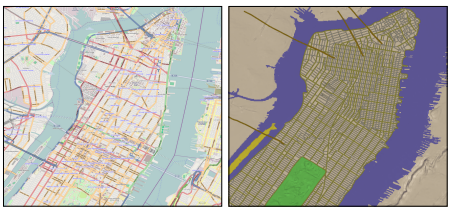
Comparing Sketching and Input Maps: We tested an early version of our sketching interface against the dominant alternative of image map input created with a paint program. In the painting interface each layer (usage, road pattern, parkland) is displayed separately and created using a typical image painting toolset with variable width paintbrushes, erasers and bucket fill. This represents a common paradigm in procedural city modelling.

The experiment consisted of a between-groups study with 20 subjects. Users were asked to reproduce 2 previously created cities (drawn randomly from a sample of 5) and were assessed on the basis of speed, accuracy and a post-test usability questionnaire. While sketching was faster than image maps, this was not significant ($p = 0.15$ on a t-test). We attribute this to the disparity in experience with comparable interfaces (users scored a mean of 3.2 for painting and 1.4 for sketching on a 5-point Liekert scale of self-reported experience). With training we expect sketching to be significantly faster. Accuracy was assessed using a scoring scheme which considered the approximate shape of region boundaries, placement of road constraints and the heights of buildings. The sketching interface was found to be significantly more accurate ($p = 0.008$). Finally, users scored the usability of sketching more highly than image maps, with a mean score of 4.35 against 2.10 on a simple 5-point Liekert scale. In summary, we found that sketching wins in general over image maps with respect to accuracy and usability, with a non-significant trend towards improved speed.

Gesture Calibration: In order to deal with the wide range in building scales (from 5m high single-story homes to 400m tall skyscrapers) and the limited size of the gesture canvas, we originally intended employing an exponential scale. After running an exploratory experiment with 10 subjects, however, we discovered that most users found this to be confusing and counter-intuitive. Given drawings of various building gestures along with a scaling key (as shown on the left of Figure 4) and the task of attaching heights and widths, the majority of users (8 out of 10) used a piecewise linear scale for the vertical, with a change in slope at about the $3-4$ story level, and a width dictated by the

aspect ratio of the building. These findings motivated our final approach.

Gesture Recognition: We undertook an experiment to test the robustness of our simple gesture recognition scheme, since it can be frustrating for a user to have to undo false positives or redo false negatives. Our experiment was something of an acid test: the 12 subjects were given no training except a briefing sheet, received no feedback as to the success or failure of gestures (since we wanted to minimize learning effects) and they used a mouse, which, although ubiquitous, is rather deficient as a drawing device. Subjects were presented with 3 examples of each gesture, in a randomized order. The results were encouraging, with a mean recognition rate of 84.1% ($s = 7.4\%$) on the first attempt. There were no misclassified gestures (false positives) and the mean unclassified gesture rate (false negatives) was thus 15.9%. The gesture that failed most often was the simple Manhattan road pattern: two sets of near orthogonal lines that cross each other. Users sometimes drew overly long lines with high curvature, which were incorrectly classified as polylines, an issue that is easily corrected with remedial training. Overall, this experimental setup represents worst-case performance, since, in practice, users are given feedback, are allowed to practise, and are not restricted to using a mouse, leading to substantially higher first-attempt recognition rates.



[A] [B]

Figure 6: Results: comparison of real and virtual Manhattan - [A] real street map (©OpenStreetMap contributors), [B] virtual recreation. City Sketching is able to reproduce the alignment and spacing of road networks and the placement of features, such as Central Park, Broadway and Brooklyn Bridge but is not suited to exactly reproducing local road layouts.

We have also undertaken some informal validation tests to see whether an existing cityscape can be roughly reproduced by an experienced designer within a fixed time limit (45 minutes). A procedural version of Manhattan (New York City) with 7660 buildings and 3006 roads is shown in Figure 7 and alongside a road network from the corresponding real city in Figure 6. Also shown is a hypothetical city on the edge of the Grand Canyon (11794 buildings and 5605 roads). The procedural city generation completed in under 30 seconds for both examples on a 2.4 GHz Intel Core 2 Duo with 2 Gb RAM.

We have found that sketched gestures are an effective input mechanism when absolute precision is not required and parameters have a direct geometric interpretation. In such circumstances they can be both meaningful and compact. For instance, our usage zone gesture captures a selection and 9 float values in a single sketch. However, gestures are less useful for indirect parameter specification. For example, our system assumes usage zones with a dominant type, but in many cases a mixture is more realistic. Using a gesture to encode the exact proportion of industrial, residential and commercial buildings within a zone would likely be clumsy in the extreme. In cases like this, traditional GUI elements, such as slider bars, are more appropriate.

6 CONCLUSION

This paper presented City Sketching, a procedural system that employs a hybrid sketch and gesture interface, enabling users to control, in detail, the generation of a virtual city, including the layout and interaction between road networks, categories of land usage, and the dimensions of plots and buildings. Our system can be applied to improve scene modelling for visual effects, computer games and simulation. Our results show that it is possible to have a single procedural framework, controlled through sketch-based interaction alone, which draws together all the necessary components for high-level city design. Furthermore, our approach is both more usable and more accurate when compared to conventional image map or numeric constraint specification.

There are a number of directions in which the system could usefully be extended. First, it would be interesting to connect our interface to a separate simulation-oriented city generator that is capable of exploiting the statistics and constraints that we produce. Second, there is scope to extend detailed editing of the city. For instance, the placement of individual “hero” buildings with specific dimensions is possible but not currently supported. Finally, the example-based aspect of our system could be extended by substituting real-world road map images for road pattern gestures. However, deriving statistics or even explicit road constraints in this case would require extensive image processing.

ACKNOWLEDGEMENTS

This research was supported by an NRF/THRIP grant and the Centre for High Performance Computing.

7 REFERENCES

[1] Prusinkiewicz, P., and Lindenmayer, A. The algorithmic beauty of plants. Springer-Verlag, 1996.

[2] Ebert, D., Musgrave, F., Peachey, D., Perlin, K., and Worley, S. Texturing and Modeling: A Procedural Approach. Morgan Kaufmann; 3 ed., 2003.

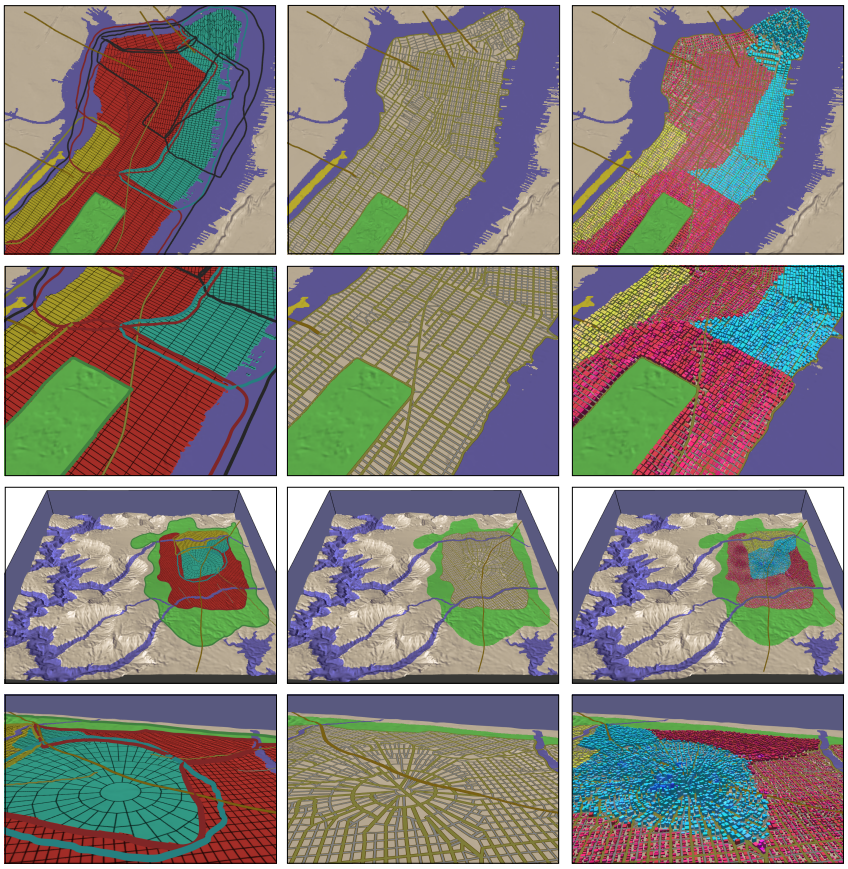


Figure 7: Results: two procedural city sketches. By row from top to bottom - Manhattan, Manhattan in close up, A hypothetical city on the edge of the Grand Canyon and its close up. By column from left to right - sketch, road network, procedural city. Both examples were designed in under 45 minutes by an experienced user.

[3] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings. ACM Trans Graph, 25(3): p. 614-23, 2006.

[4] Perlin, K., and Velho, L. Live paint: painting with procedural multiscale textures. In: SIGGRAPH '95, p. 153-60, 1995.

[5] Parish, Y., and Müller, P. Procedural modeling of cities. In: SIGGRAPH '01, p. 301-8, 2001.

[6] Hultquist, C., Gain, J., and Cairns, D. An adjectival interface for procedural content generation. In: Intelligent Computer Graphics 2009; v. 240, p. 143-65, 2009.

[7] Aliaga, D., Vanegas, C., and Benes, B. Interactive example-based urban layout synthesis. In: SIGGRAPH Asia '08, p. 1-10, 2008.

[8] Lipp, M., Scherzer, D., Wonka, P., and Wimmer, M. Interactive modeling of city layouts using layers of procedural content. Computer Graphics Forum, 30(2), p. 345-54, 2011.

[9] Gain, J., Marais, P., and Strasser, W. Terrain sketching. In: I3D '09, p. 31-8, 2009.

[10] Vanegas, C., Aliaga, D., Benes, B., and Waddell, P. Interactive design of urban spaces using geometrical and behavioral modeling. In: SIGGRAPH Asia '09, p. 1-10, 2009.

[11] Galin, E., Peytavie, A., Guerin, E., and Benes, B. Authoring hierarchical road networks. Computer Graphics Forum, 30(7), p. 2021-2030, 2011.

[12] da Silveira, L., and Musse, S.. Real-time generation of populated virtual cities. In: VRST '06, p. 155-64, 2006.

[13] Chen, G., Esch, G., Wonka, P., Müller, P., and Zhang, E. Interactive procedural street modeling. In: SIGGRAPH '08, p. 1-10, 2008.

[14] Vanegas, C., Aliaga, D., Wonka, P., Müller, P., Waddell, P., and Watson, B. Modelling the appearance and behaviour of urban spaces. Computer Graphics Forum, 29(1), p. 25-42, 2010.

[15] Dadgostar, F., Sarrafzadeh, A., Fan, C., Silva, L., and Messom, C. Modeling and recognition of gesture signals in 2d space: A comparison of nn and svm approaches. IEEE Conference on Tools with Artificial Intelligence, p. 701-4, 2006.