A Sketching Interface for Procedural City Generation

Matthew de Villiers
University of Cape Town

mat@mallardexpress.co.uk

Neilan Naicker
University of Cape Town
+27 72 191 2754
darric@gmail.com

ABSTRACT

In this paper, we present an application which uses a sketch and gesture interface to facilitate the procedural creation of three dimensional city environments. City boundaries, roads and region allocations are drawn directly onto a 3D terrain, and a city is generated using these inputs. Aspects of the city that can be modified include the population density, allocation of parks, and variances in building height. Once generated, the city is rendered in a non-photorealistic sketching style, to correspond with the nature of its creation.

Categories and Subject Descriptors

I.3.8 [**Computer Graphics**]: Applications, Methodology and Techniques

General Terms

Algorithms, Design, Experimentation, Human Factors.

Keywords

Sketching Interface, Gesture Recognition, Procedural Modelling, Non-Photorealistic Rendering.

1. INTRODUCTION

The entertainment industry, particularly in the fields of film and computer games, has seen a rapid adoption of increasingly complex digital environments. In the case of some films, extensive use of entirely virtual environments has been employed for scene backgrounds. Needless to say, this is also the case for every three-dimensional computer game. There is a tendency for these environments to become more and more complex with time, in parallel to the rising expectations of viewers, and the ever increasing capabilities of modern graphics hardware.

A city is a good example of a complex environment to recreate. Even if only a coarse level of detail is required, streets and buildings need to be placed realistically in order to achieve a believable aesthetic. Road patterns, despite their simplicity when examined individually, can display very complicated behaviour when forming a larger system. Typically, a large number of concept artists, modellers, texture designers and animators are needed in order to create such an environment. Thus, the resources required for such an undertaking – with regards to time, money and human resources – can be exceptionally high.

For this reason, it becomes beneficial to investigate a new means of procedurally modelling city environments. Sketch and Gesture interfaces have emerged as an easy to use means of easily creating content in a variety of applications. For instance, in Teddy by Igarashi *et al.* [4], a sketch and gesture interface is used for the rapid creation of rotund 3D models. Such an approach could be used for creating cities.

We present in this paper such an application for creating city environments, by means of a sketch and gesture interface. The *CitySketch* application allows the user to draw the properties of a city directly onto a 3-dimensional terrain. For instance, the user might sketch out the boundary of the city, as well as a number of desired roads. In addition, the application allows for regional allocation of properties such as population variances, park areas, and changes in building height. The city is then generated, complete with a spanning road network, and correctly placed buildings.

The city is rendered using a non-photorealistic "sketching" style rendered, to correspond with the nature of its creation. Modifications to the city are instantaneous, and the city's creation process is rendered to the user at all times. After the city has been generated, it can be modified by removing roads and changing the heights of buildings. If desired, more roads can be drawn, or additional cities can be drawn on the terrain.

This paper covers the methods and algorithms behind *CitySketch* as well as all of the features available to the user.

2. RELATED WORK

Procedural modeling is employed extensively in the generation of natural phenomena, such as plant life. For example, the use of Lindenmayer Systems ("L-systems") in producing natural-looking plants and trees [6, 12] clearly illustrates an effective means of procedurally generating that which would be unreasonable to do manually.

A fair amount of work has been done on the procedural generation of cities, but most of this requires little or no input. For instance, Lechner *et al.* generate 2-dimensional cities based only on an image map. Such a system could be easily enhanced

to produce 3D environments, if coupled with a procedural building modeler, as done by Martin [8, 9] and Greuter [2].

Image maps are used widely for generating cities, as they provide a very natural way of viewing the various contributions to the city's generation across the city area map. Perhaps the best example of this is the work done by Parish and Müller [10].

Their implementation uses multiple image maps as input (maps for population, water/parks, road patterns, terrain) and generates a fully 3-dimensional city environment based on these inputs. Image maps, however, are tedious to edit, and it is particularly hard to maintain consistency between the maps when altering them.

In the field of sketch and gesture interfaces, Teddy [4] has already been mentioned as an effective use of a sketch/gesture interface for 3D modeling. Another example of a 3D modeling application by means of sketches is "SKETCH", by Zeleznik *et al.* [15].

Other approaches to modeling which utilize sketch interfaces are Hughes' [3] method of interpolating 3D models from user created sketches of a number of cross sections of the model, and the system used by Cherlin *et al.* [1], which makes use of two phases: a constructive phase, where sketches determine the actual geometry, and a progressive editing phase, where strokes allow subtle changes to be made to the geometry.

A very good framework for the development of gesture recognition systems is provided by Rubine [13]. For instance, Landay and Myers [5] have described a means of extending any user interface to utilise gestures, as implemented by Rubine. A similar approach based on feature recognition was developed by Lipscomb [7].

Praun et al [11] present a method for a convincing hatching solution that runs in real-time. One of the core ideas presented in their paper is that of "tonal art maps". These are a series of hatching textures that can be blended across a surface to give the impression of shading. An appropriate combination of these textures is dynamically blended onto geometry according to surface tone.

Each texture level is a subset of the more densely-hatched textures, meaning that the textures smoothly blend from one level to the next.

In real-world non-photorealistic renderings, artists typically draw the strokes in a scene at a consistent size, regardless of their distance from the viewpoint: Praun *et al*. leverage texture-mapping hardware by using mip-maps to scale distant strokes appropriately. In order to achieve this effect, custom mip-map textures are created; these textures consist of strokes of the same size as those in the original texture.

Figure 1: "Tonal art maps" from Praun *et al*. [11]

Rusinkiewicz et al [14] present a technique for achieving exaggerated shading over terrain surfaces. Their technique involves the use of a "soft toon" shader in order to emphasise sloped (as opposed to horizontal) surfaces. They also create several levels of normals for the terrain surface; these are used to emphasise local detail, regardless of large-scale surface orientation.

3. METHOD

3.1 Sketching Interface

The sketching interface allows the user to draw directly onto the terrain whenever the left mouse button is held down (the right and middle mouse buttons are used for adjusting the view). To ensure that the users' intentions are interpreted correctly, the system needs to keep constant track of what is expected next.

Specifically, there are two main states. The first state is when the system is expecting the user to draw a region or road. A road is simply any drawn stroke which has its endpoint sufficiently far from its starting point.

By contrast, a region is any drawn stroke for which the end point is close to the start point. In addition, regions and roads can not self-intersect. If a road is drawn, the system does not change state – another region or road is expected next. However, if a region is drawn, the system moves to the second state.

The second state is when the system is waiting for the user to draw what is termed a region gesture. These are gestures which are used to indicate the purpose of a drawn region. They are the up and down arrow, for increasing and decreasing population in a region, the tree gesture, for allocating a park area, and the building heights gesture. This gesture requires three buildings to be drawn, which determine the mean heights and variability of buildings in that region.

Once a region has been drawn, and its property allocated, it is rendered to a texture and used to modify the system's internal image maps, as according to its property. For example, a region followed by the up arrow gesture increases the population in that region on the internal population density image map.

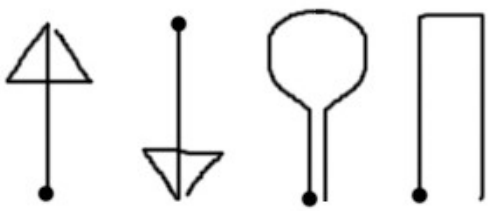

Figure 1. The available region gestures. From left to right: Population increase, population decrease, park allocation, building height. Dots indicate starting points of each gesture.

The other type of gesture is the command gesture, of which there are two types. The spiral gesture commands the system to generate the city based on the current roads and regions, while the undo gesture reverses the most recent action.

Because command gestures can be drawn when the system might otherwise expect a road or region, each has a property which uniquely determines it. The spiral gesture must be drawn in the sky in order to be interpreted correctly, while the undo gesture has a self-intersection – a property unacceptable for roads or regions.

Figure 2. The available command gestures. The spiral (left) generates the city, while the undo (right) reverses the last action.

All strokes are initially recorded as a sequence of points while the left mouse button is held down. To be of any use to the stroke and gesture parsers, these points need to be converted to segments. This is done by generating a new segment when the mouse moves beyond a certain distance from the endpoint of the last segment. Thus, a connected sequence of segments of uniform length is generated as soon as the left mouse button is released.

In the case of strokes (i.e. roads and regions) these segments are simply passed to the road generator (to be converted to an L-system string) or rendered to a texture. For the purposes of gesture recognition however, the process is more involved.

To aid gesture recognition, the sequence of segments is converted to a sequence of lines of unequal length. This is achieved by grouping segments with similar angles into a single line. Thus, the drawn stroke is interpreted as a minimal number of lines which represent it.

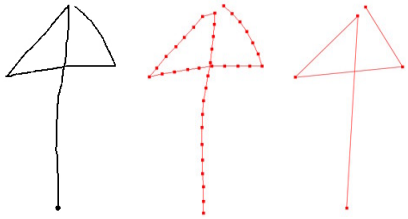

Figure 3. The stroke points are divided into segments, which are then grouped by angle similarity into lines.

A number of features are then determined from the gesture, as described by Rubine [13]. Features include properties such as the total length of the stroke, the number of lines comprising it, the total angle traversed, and the initial angle. In all, twelve distinct features, each a floating point value, are determined from each stroke.

These feature values are then compared to pre-computed values for each of the defined gestures, and the gesture for which the most features correspond is decided as the gesture intended by the user. However, a minimum number of features must be matched, or else the gesture is rejected and must be redrawn.

Finally, strokes can also be interpreted as scribbles if they cross the central axis of the stroke a sufficient number of times. Scribble strokes are used to remove roads which the user deems undesirable. This is implemented by checking which road segments are intersected by the scribble stroke more than a few times, and removing them from the road system.

3.2 City Generation

The city generation process takes place in two distinct steps. First, the road network is created based on the inputs from the sketch interface. The second step, which occurs once the road network is complete, is the placement of buildings in the city's street blocks.

The road generation system is based heavily on Parish and Müller's road system [10]. Their approach uses L-systems in order to generate the branching structure of the road network. The network is comprised of two types of roads, highways and streets, which behave slightly differently.

Highways rely heavily on the population map in order to determine their growth. Each highway projects an arc in front of it, and, for each angle in that arc, samples the population density at a number of points along a line in front of it. The angle which has the highest population density sum is chosen as the angle in which the highway continues.

Thus, there is a tendency for highways to seek out high population regions. This results in a concentration of highways in more populated areas. This fits the function of highways; to provide access to the more frequented areas of the city. Highways develop perpendicular branches every five segments, and between those branches, street networks generate, on a large delay. The delay allows the highway network to generate satisfactorily before the street network begins to generate.

The function of streets is therefore to fill in the gaps between the highway networks with a semi-regular grid pattern. Thus, streets generate outward from their origins, with perpendicular branches after every segment. Street networks continue until they reach a highway, or the population drops below a threshold.

To facilitate the road generation, frequent checks are made at every iteration. Newly formed roads are checked for intersections with existing roads, and shortened to the intersection point if one is found. Any road which ends close to an existing junction of roads is modified in angle and length to meet that junction. Also, roads which enter park or water areas are swiveled and/or shortened to keep them valid.

Roads are also checked against the terrain height map for validity. The height of the terrain at a road's start and end point is used to determine the slope of the road, which if too high, causes the road to be rejected. Also, highways tend to remain on areas of comparable height when heading towards high population zones.

The initial string which is passed to the L-system is based entirely on the road strokes which are drawn by the user. For a given feature road stroke, the middle segments are passed to the L-system as "final" roads, while the end segments are passed as developing roads. Thus, all feature roads develop at their end points. In addition, developing roads are placed perpendicular to the feature road along its length.

For the purposes of intersection checking, a secondary data structure – a quadtree – is used to store the road segment data while the road system is generating. This prevents having to check every road in the system; rather, only the roads that are close to a specific road need be checked, by using the quadtree to check closeness.

The road generation continues until all roads are final (when the road network has more or less filled the city boundary). At this point, the road segment data is converted to a connected graph of edges and vertices; each vertex has a list of attached edges, and each edge's end-points are indices into the vertex array.

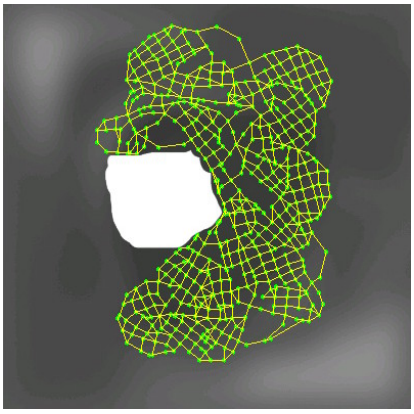

Figure 4: A road network graph generated on a height image map. The white area is a water feature.

Building generation is achieved through two steps: lot allocation, and building placement.

In order to determine the lots in which buildings can be placed, we find the smallest polygons in the graph. The centroids of these polygons are then calculated (see Figure 4), and each lot is scaled down towards its centroid.

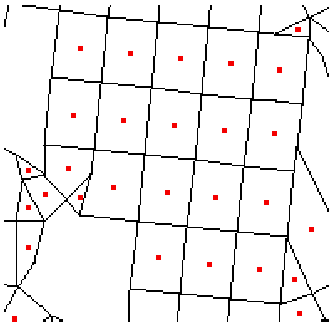

Figure 5. Centroids of lots.

After we have determined the lots, we place the buildings that populate these areas. Our approach in this problem differs from Parish and Müller's.

We place buildings along each edge, performing checks at each placement to avoid collisions between buildings. These checks are accelerated through the use of a quad-tree structure.

Building width and depth are randomised slightly at each step, and building height is a function of the image map defined through the sketching interface. This results in a collection of buildings that appear to follow the road structure while still maintaining an organic, humanised appearance.

3.3 Renderer

In order to render our scene in a non-photorealistic style, we primarily follow Praun *et al.*'s [11] cross-hatching approach.

To calculate the shading at each point in the scene – and therefore the combination of crosshatch textures – we use an exaggerated shading model as described in Rusinkiewicz et al. [14]. This method emphasises the non-photorealism of the scene by accentuating sloped terrain, and complements Praun *et al.*'s crosshatching technique.

We used this combination of non-photorealistic methods to render the terrain, which is a 512x512 heightmap. Frustum culling and a simple level-of-detail scheme are used to facilitate interactive frame-rates.

In order to visualise the buildings that make up the city, we render instances of a cube model for each building. Affine transformations are applied to each instance in order to position, scale and rotate the cube appropriately.

We found that the most effective approach for accentuating building outlines was to simply apply an "outline texture": one

which has a light interior and dark borders. This approach yielded faster frame-rates than other methods, and provided acceptable results (see Figure 6).

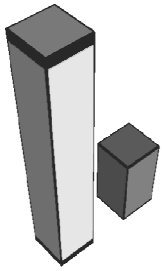

Figure 6. Buildings rendered with "outline textures".

The roads in our scenes are represented as textured 3D geometry. To achieve this, we create quadrilaterals from line segments. These quadrilaterals are formed by extending line segments in a direction perpendicular to their direction and the Y-axis vector.

Figure 7. Line segments are extended perpendicular to their direction and the Y-axis vector. (Realistic rendering mode depicted.)

In order to enable comparison, we have also implemented a realistic mode and a toon-shaded mode.

For the realistic rendering mode, three pre-made images are used to texture the terrain. These textures are blended smoothly based on a "coverage map", which is a hand-made RGB texture. The three channels of the coverage map specify how much of a contribution the first, second and third texture make to the final pixel colour.

Several additional elements were built into the realistic mode, such as water with reflection and refraction and a skybox.

The toon-shaded mode simply shades the terrain using standard lighting methods, and then reduces the tonal range; this results in an aesthetic that is typical of cartoon drawings.

4. RESULTS

A city generation application by means of a sketch and gesture interface was successfully implemented with the following features:

- Drawing city boundaries on the terrain
- Drawing regions in which to increase/decrease populations
- Drawing regions in which to allocate parks
- Allocating building height means and variance in a region
- Drawing feature roads from which the city road network generates
- Modification of the city by erasing roads

The application is rendered using a non-photorealistic sketching style renderer. Using a combination of existing techniques for real-time non-photorealistic rendering, it effectively displays scenes at over 30 frames per second and provides great visual clarity of the created environment. Figure 8, below, is a screenshot of a generated city in the application.

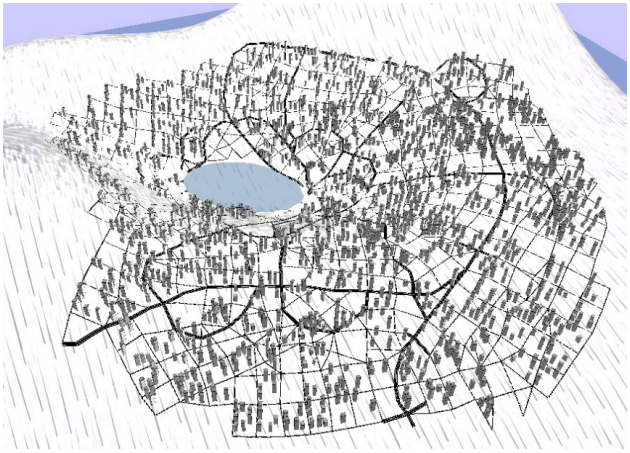

Figure 8. A generated city rendered in a non-photorealistic style in the *CitySketch* application

Our alternative to Parish and Müllers' [10] method for building placement provides acceptable results and is not computationally expensive.

In order to test the application, and compare it to the approach by which cities are created by editing image maps (as employed by Parish and Müller), an experiment was conducted with two groups of users. Each user was presented with two sets of images of cities to recreate.

The control group edited image maps in an image editor, which were then provided as inputs to the city generation system. The experiment group used *CitySketch* to recreate the cities. Their attempts were timed and assessed based on accuracy to the target cities.

It was found that the experiment group, using CitySketch, were able to more closely create the target cities, and did so in less

time. In addition, the users expressed a greater sense of control over the city's creation when using CitySketch (as assessed by questionnaire).

5. CONCLUSION

The aim of creating a 3D city-sketching application has been achieved, and the results are promising for future work in this area.

We found city generation to be very CPU-intensive. When using the NPR renderer, the frame-rate only dropped below our target minimum for very large cities; however, in order to generate the cities within a reasonable time, we had to keep them small and simple. Therefore, we found that the generation time would exceed our target maximum long before the cities became too complex to render at interactive rates.

Not all of the proposed features could be implemented, but the core functionality of the system was finished on time and works as required. Additionally, the renderer was extended to include some of our proposed "future work" by adding a realistic mode and several additional features, such as the graphical indications when gestures are recognised and the ability to draw gestures outside of the terrain area.

We have found the combination of systems to work very well: city generation is a particularly appropriate application for a sketching interface, combining the freeform nature of sketching with the level of detail only feasible through procedural methods. Additionally, the use of non-photorealistic rendering in order to represent a scene is a powerful method of immersing a user in a sketching experience.

Finally, the experiment showed that the sketch and gesture approach is more efficient and easier to use than an approach based on manually editing image maps. In addition users reported a greater sense of control with the sketch and gesture interface, and typically learnt the system in less than 15 minutes.

6. FUTURE WORK

There are numerous additions that could still be implemented in our solution.

At present, all buildings are represented solely by single rectangular blocks. More detailed building structures could be introduced, similar to the Parish & Müller method by means of L-systems [10].

In order to create more variation across the city, additional zoning properties could be introduced, for instance, commercial, residential and industrial areas. Each would have different building styles and sizes.

More complex road types could be introduced, such as highways, bridges and tunneled roads. In addition, the way in which the road network is inferred by drawn strokes could be improved.

To better allow for correction of errors, the ability to modify regions by scribbling, in the same way roads are scribbled out, could be introduced. Also, multiple levels of undo could be implemented rather than only a single step.

Similarly, to allow the user to make minor modifications to the city after it has been generated, a means of cutting or extending individual buildings could be introduced.

It would be interesting to know which of our rendering modes is more conducive to creative sketching of cities. This could be analysed by restricting test users to a single rendering option, asking them to create a specific city, and measuring the resulting cities with a closeness metric.

7. REFERENCES

[1] – Cherlin, J., Samavati, F., Sousa, M., Jorge, J. *Sketch-based Modeling with Few Strokes*. 21st Spring Conference on Computer Graphics, 2005

[2] – Greuter, S., Parker, J., Stewart, N., Leach, G. *Real-time Procedural Generation of `Pseudo Infinite' Cities*. Computer Graphics and Interactive Techniques (GRAPHITE), 2003

[3] – Hughes, D., *3D Model Creation by Sketching Cross-sections*. Yale University, 2005

[4] – Igarashi, T., Matsuoka, S., Tanaka, H. *Teddy: a sketching interface for 3D freeform design*. SIGGRAPH, 1999

[5] – Landay, J., Myers, B. *Extending an Existing User Interface to Support Gesture Recognition*. INTERCHI'93: Human Factors in Computing Systems, p. 91-92, 1993.

[6] – Lindenmayer A., Prusinkiewicz, P. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990

[7] – Lipscomb, J. *A trainable gesture recognizer*. Pattern Recognition, 24(9):895-907, 1991

[8] – Martin, J. *Procedural House Generation*. Symposium on Interactive 3D Graphics and Games, 2006

[9] – Martin, J. *The Algorithmic Beauty of Building: Methods for Procedural Building Generation*. Honours Thesis, Trinity University, 2004

[10] – Parish, Y., Müller, P. *Procedural Modeling of Cities*. SIGGRAPH, 2001

[11] – Praun, E., Hoppe, H., Webb, M., Finkelstein, A, *Real-time Hatching*, 2001

[12] – Prusinkiewicz, P., James, M., Mech, R. *Synthetic Topiary*. SIGGRAPH, 1994

[13] – Rubine, D. *Specifying Gestures by Example*. SIGGRAPH, 1991

[14] – Rusinkiewicz, S., Burns, M., DeCarlo, D., *Exaggerated shading for depicting shape and detail*. ACM Transactions on Graphics, 2006

[15] – Zeleznik, R., Herndon, K., Hughes J. *SKETCH: An Interface for Sketching 3D Scenes*. SIGGRAPH, 1996