A Different Manhattan Project: Automatic Statistical Model Generation*

C. K. Yap, H. Biermann, A. Hertzman, C. Li J. Meyer H. K. Pao T. Paxia

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
New York, NY 10012

ABSTRACT

We address the automatic generation of large geometric models. This is important in visualization for several reasons. First, many applications need access to large but interesting data models. Second, we often need such data sets with particular characteristics (e.g., urban models, park and recreation landscape). Thus we need the ability to generate models with different parameters.

We propose a new approach for generating such models. It is based on a top-down propagation of statistical parameters. We illustrate the method in the generation of a statistical model of Manhattan. But the method is generally applicable in the generation of models of large geographical regions. Our work is related to the literature on generating complex natural scenes (smoke, forests, etc) based on procedural descriptions. The difference in our approach stems from three characteristics: modeling with statistical parameters, integration of ground truth (actual map data), and a library-based approach for texture mapping.

Keywords: automatic model generation, large geometric model, visualization, urban model, statistical model, procedural model generation

1. INTRODUCTION

We consider the problem of generating very large geometric models for visualization applications. There are many potential applications. A major challenge in current visualization research arises from the largeness of the geometric models that must be visualized in realtime (E.g., Funkhouser et al[1,2]). This arises in walk-throughs of complex architectural building or detailed model of an airplane. Such large data sets are either proprietary or very expensive to acquire. Nevertheless, researchers want access to a variety of such data sets. For instance, in testing and tuning a walkthrough visualization system, we need large data sets with different statistical properties. Thus there is a general need for the ability to generate large geometric models with various parameters.

In this paper, we propose an approach based on statistical parameters. Our method is particularly suited for generating geometric models of landscape or cityscape. There are many partial models of cities and urban areas: these are found distributed with city planners, conservation groups, historical societies, real estate developers, utility companies, etc. Indeed, many local government are actively† trying to bring together such partial data sets into public domain. In any case, it is currently virtually impossible to come by any reasonably complete model of an entire city or a large region. Fortunately, for many visualization applications, the exact model is less important than certain statistical properties such as building density and the ratio of residential to commercial properties. For instance, in a web-based tourist industry, it may be enough to generate look-alikes of a city, with only critical landmarks or buildings in a small neighborhood being true-to-life. This is quite acceptable if, for instance, the streets are map-based and the user is aware that not all buildings are real, and furthermore, the user can query for this information. Other uses for large landscape models include simulation, games or computer-aided education and instruction.

* This work is supported in part by NSF/ITR Grant #CCR-0082056.

†Examples are the New York State GIS Clearinghouse at `http://www.nysgis.state.ny.us`, and the New Jersey Spatial Data Clearinghouse at `http://njgeodata.state.nj.us`.

The problem of generating visually realistic models is better known in the simulation of natural phenomenon or environments such as clouds, mountains, erosion, forests, etc. For example, Reeves and Blau[3] describe the use of particle systems to generate a forest environment with stochastic variations. These methods are based on procedural knowledge, either encoded in formal systems such as L-systems.[4] or in statistical properties ("noise functions"). However, there is a distinction in the modeling of natural environments versus man-made environments. In natural environments, the look-and-feel and statistical properties of the final model may be the only basis for judging the quality of the method. For man-made environments, there are some hard ground-truths (such as the information found in a map of the area) ought be preserved. A purely procedural approach does not suffice or unsatisfactory. We need modeling techniques that can incorporate the special information, both geospatial and man-made facts that are important for the user. One of our main contribution is to raise and propose a solution to this issue.

There are applications such as computer games and entertainment where the external factual constraints are less stringent. In such cases, it is quite acceptable to use a purely procedural methods. Thus, Parish and Mueller[5] address the problem of modeling cities by using procedural techniques based on L-systems. Our statistical approach is still applicable in this setting. Our generator begins with a map, but this map could now be procedurally generated as in.[5]

2. A DIFFERENT MANHATTAN PROJECT

This paper describes "A Different Manhattan Project" whose goal is to automatically generate statistically-correct models[‡] of Manhattan from high level scripts. Landscape models have a constraint, as well as an advantage, in our statistical approach. The constraint is that the the model must ultimately be correlated to the earth surface. This is also its advantage because map data is widely available. So we can begin with a map of the region. Moreover, many maps have auxiliary geocoded data, including population density, average income, land use. For example, this type of information is available in the well-known TIGER data set.[6] By exploiting such information, we can randomly construct statistical look-alikes.

Let us illustrate the importance of statistical ground truth. Manhattan is an long narrow island, roughly 3 kilometers wide and running 20 kilometers from North to South. This map we use is shown in figure 7. The tall buildings of Manhattan are concentrated in the midtown and downtown sections (roughly, the middle and southern tip of the island). Any simulated model of Manhattan ought to preserved this information. Current procedural techniques are not equipped to handle this (though it would be of interest to explore this connection). In figure 1, we see a simulated skyline of Manhattan generated quite simply and naturally by our statistical approach. This is a view from the West (New Jersey side).

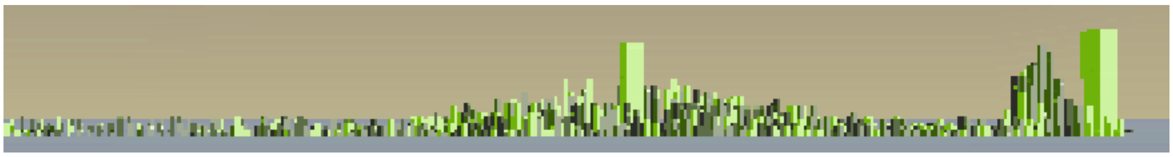

Figure 1. Simulated Manhattan Skyline: View from the West

The Manhattan skyline is punctuated by two skyscrapers that stand out among the tall buildings: the Empire State Building and the twin towers of the World Trade Center[§]. No believable model of Manhattan can ignore this isolated but striking fact. In modeling natural phenomenon, isolated features are generally non-existent. In our script-based model generation, such landmarks can be naturally and easily handcoded. These two landmarks of the skyline of Manhattan are clearly visible in figure 1. In figure 2, we see the same skyline from the South.

[‡]It is different, not only from the real thing, but also from the other, more famous Manhattan Project.

[§]On September 11 of 2001, a terrorist act removed this second landmark.

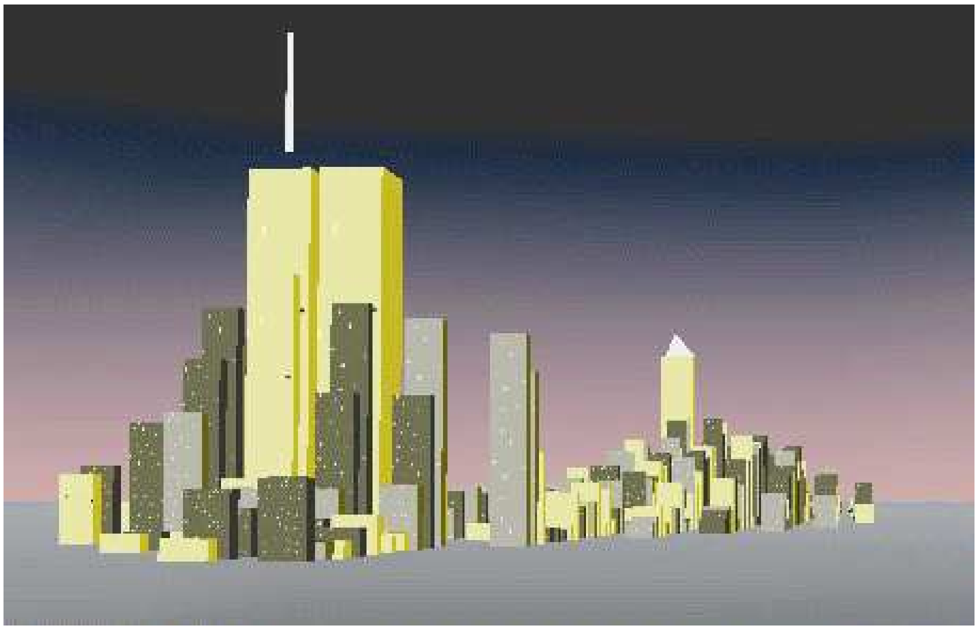

Figure 2. Simulated Manhattan Skyline: View from the South

In our work, the starting point is a map. The advantage of this map is that it can be used to encode common knowledge about the nature of city neighborhoods. New York city, like every city, is the unique sum of its distinct neighborhoods: from the canyon-like streets of the financial (Wall Street) district, through the bright lights of Broadway Theatre district to ethnic neighborhoods such as Harlem and Chinatown. Rule-based approaches would be awkward at best to try to capture such unique properties. Just as the Empire State Building is an isolated fact which cannot be ignored, the uniqueness of neighborhoods must somehow be captured.

In our approach, each neighborhood has its own script. The script is basically a collection statistical parameters. This collection is somewhat open ended and data dependent; we will list those that are important for Manhattan. The most obvious physical parameters are average heights and sizes of buildings. The variance of these parameters is also important. Next comes zoning classification of land use in a neighborhood: commercial, residential, park, public facility and industrial use. For instance, we may model Greenwich Village as 60% residential and 35% commercial and 5% park. The distribution of age of buildings and dominant architectural styles are also important parameters: the Upper West Side has a predominance of large prewar apartment buildings, Greenwich village has a relatively high percentage of brownstone buildings from the 19th century, and SoHo has its distinctive cast iron buildings. The architectural flavor of each neighborhood is thus captured at a high level through these parameters.

Based on the above statistical parameters and a map of the neighborhoods, our algorithm takes over: the parameters for a neighborhood are randomly propagated down to individual city blocks. Each city block is randomly subdivided into individual lots, where the lots are either building or have other use. Building lots are given the parameters of height, age, zoning classification, architectural style, etc. This random propagation of parameters can be refined in several ways: there can be rules for prohibiting or enhancing certain values. Thus commercial buildings prefer corner lots and major streets. Clustering behavior dictates that buildings with the same zoning type occur in close proximity, and shun isolation. Spill over effects should be modeled at the boundary of neighborhoods.

Finally, when every lot has been classified with specific parameter values we appeal to a library of textures and models to provide the visual effect. For each building, a suitable call to the library with its parameters (height, age, zone classification, etc) will be randomly instantiated with a texture for the building facade. Additional random elements include street details (trees, hydrants, street signs, etc) and street life (pedestrians, cars). The existence of an rich texture library is vital to the success of this approach.

3. GROUND TRUTH

It is useful to take maps as our starting point for two reasons: they are relatively easy to obtain, and they greatly constrain the possible models (this *is* a good thing). Having decided on maps, we could have chosen to use the maps found under the TIGER data set.[6] However, we choose to take advantage of the street maps from the New York City Department of City Planning. `http://www.ci.nyc.ny.us/html/dcp/html/bytes/blguide.html`. These maps are represented by a list of polygons representing the detail shapes of city blocks; the common exterior of these polygons thereby provide fully dimensioned details of the streets. See figure 3 for a sample.

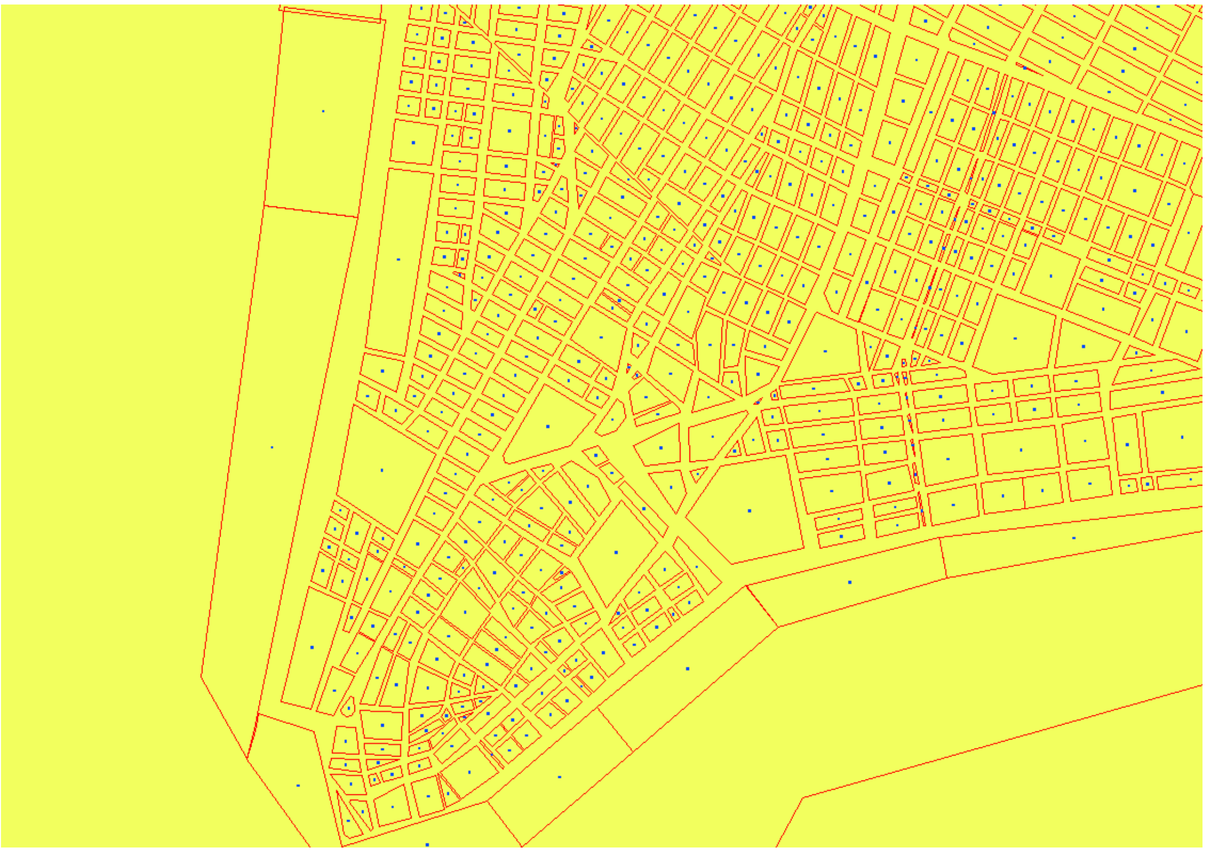

Figure 3. Street Map of Lower Manhattan, with Fully Dimensioned City Blocks

Note that TIGER data set lack such fine details. But we must now solve several problems that would be non-existent if we begin from TIGER data. Our street maps originated in the Adobe Illustrator file format. The format has no explicit geometric information beyond a listing of the city blocks. From these polygons, we have the task of determining the individual streets. Due to the nature of our generator (see below), we want to partition each street into **street sections** and **street junctions**. Street junctions are shared among the streets that crossing that intersection. Figure 4 illustrates a sample output from the partitioning process: the white polygons which are marked with an interior point are city blocks. The street junctions are white polygons without such markers. The grey polygons represent street sections, and these are generally rectangular.

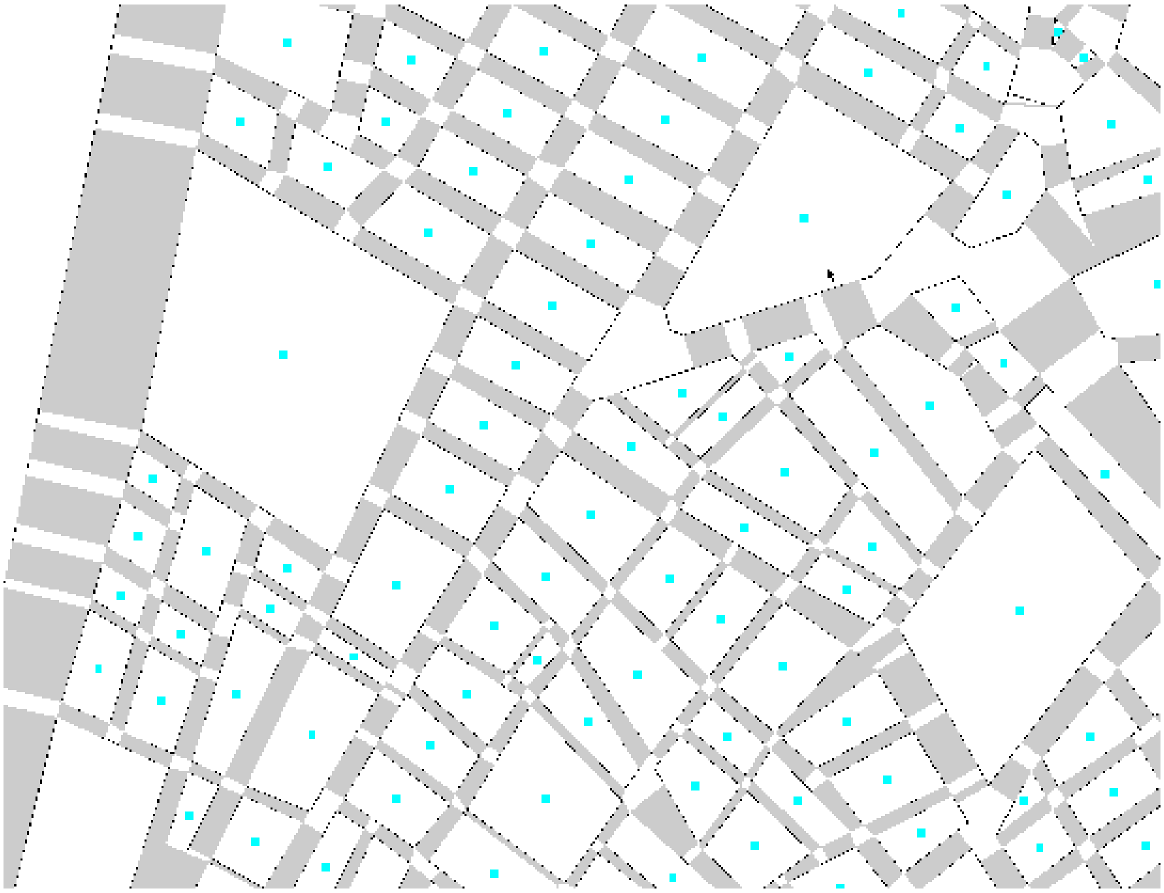

Figure 4. Partition of streets into sections and junctions.

We see from figure 4 that most street junctions are the intersection of two streets (either a T-junction or a 4-way junction). But there are complicated shapes whose partition and classification are not obvious. Towards the upper right corner of figure 4 we see several junctions that are odd shaped and non-convex. Because of dirty data and the approximate nature of this partitioning, a variety of heuristics is necessary.

We first outline our heuristic approach. To begin, we convert all the city blocks into convex polygons. Each edge of the resulting city block then attempts to match itself to an opposite edge from another block. This matching process is illustrated in figure 5, where we draw a line segment (let us call them "m-segments") from the midpoint of each edge to the edge that it attempts match. Often, the feeling is mutual so we see many almost parallel pairs of m-segments between pairs of edges. These matched pairs are candidates for street sections. But as seen in the upper left and upper right corners of the figure, some m-segments are "long" in the sense that they cross other shorter m-segments. We remove the long ones. From what is left, we form the street sections; the left over parts form the street junctions. This procedure is relatively simple to implement and runs in $O(n^2)$ time.

The Corridor Detection Problem. We can formulate a corresponding abstract computational problem: given a collection of pairwise disjoint polygons (called **blocks**), we want to partition the common exterior Ω of these blocks into **corridors** and **junctions**. Thus corridors correspond to our street segments. We define a corridor to be a polygon $C \subseteq \Omega$ whose boundary is partitioned into four connected portions: one portion is the boundary of some block B, another is the boundary of another block B', and the remaining two boundary portions, β and β', are line segments that cross the interior of Ω. We call C a corridor "between B and B'" and

β, β' the "exits" of the corridor. The connected components of the non-corridor parts of Ω are the junctions. We sketch an $O(n \log n)$ solution based on Voronoi diagrams of the blocks. It is well-known that this Voronoi diagram partitions Ω into Voronoi cells whose boundaries are either block boundaries or Voronoi edges. The Voronoi edges are curves in the interior of Ω that are either line segments or pieces of parabolas. We define a **Voronoi corridor** corresponding to a maximal connected sequence of Voronoi edges, such that there are two blocks B, B' such that each point on these Voronoi edges are closest to both B and B'. Suppose p is an endpoint of this edge sequence. Then the largest disc $D(p)$ centered at p inside Ω touches a third block B''. Let us assume $D(p)$ touch no other blocks. If $D(p)$ touch B, B', B'' at b, b', b'', respectively, then the segment $[b.b']$ is an exit for the corridor between B and B'; the other exit is similarly defined. The triangle $\Delta(b, b', b'')$ forms a junction. If the input were in general position, all junctions would be triangles. We can compute the Voronoi diagram of Ω in time $O(n \log n)$. The actual corridor detection can be carried out in $O(n)$ time be walking along the Voronoi edges to form the above corridors.

Figure 5. Matching of segments to form potential street sections.

Kernighan and van Wyk[7] addressed an analogous problem of deriving geometry from the collection of line segments that constitute a drawing in the DXF file format. The DXF format, like the Adobe Illustrator format, is mainly an drawing file format with little explicit geometry. Another problem which we did not address in the above is to identify the streets – the Adobe Illustrator files contain street names for some street sections but these must be extrapolated to the other sections.

4. A SCRIPT-BASED GENERATOR

Beginning with a street map of Manhattan, we subdivide it into neighborhoods and specify statistical parameters for each neighborhood. Hard-coded landmark information, such as the Empire State Building is also part of this initial input. Based on these inputs, the final outcome of our generator is a model that is complete with urban model complete with textures and street details.

For our purposes, we divide Manhattan into 10 neighborhoods: Downtown (Financial District, Chinatown, Tribecca, etc), East Village (including Lower East Side), West Village (including Soho), Gramercy Park (including Murray Hill), Chelsea, Midtown East, Midtown West, Upper East Side, Upper West Side, Uptown. It is clear that a finer subdivision would be appropriate for a more accurate model.

We identify five tasks, and corresponding generator units: the **neighborhood generator**, the **block generator**, the **street generator**, the **texture generator** and the **integrator**. The interfaces between each of these tasks are clearly defined so as to be suitable for a team project. The data flow for these units is depicted in figure 6.

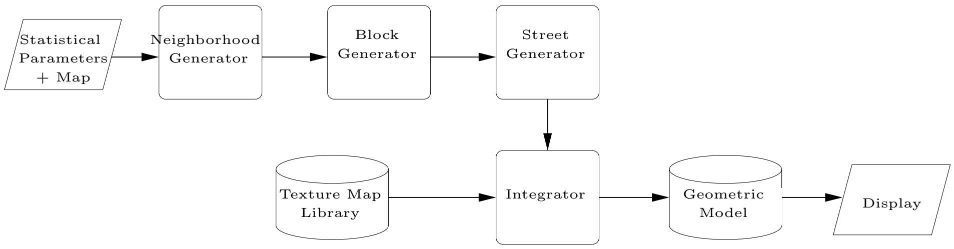

Figure 6: Data Flow Chart for Generator Units

The main inputs and outputs for each of the units are text files that "scripts" in some high level description. The advantage of using text files is that they can be viewed and modified using an ordinary text editor. In the context of our team project, this allowed each team to hand-code input scripts to test their own units.

The neighborhood generator acts on the user-given statistical data for each neighborhood, to generate parameters for a set of blocks and streets for each neighborhood. The block generator takes this input, and generates the detail parameters for each lot of a block. Most of these lots are buildings. But the block generator is also responsible for park areas or green plots. The street parameters are basically passed through unchanged. Next the street generator takes the output of the block generator to produce streel details (but it passes through the lot parameters). Thus it is conceivable that the block generator and street generator be arranged in parallel in the flowchart of figure 6. On the other hand, a more sophisticated generator may want to coordinate the street and block generator: for instance, if a lot is a restaurant establishment, it may want to place tables and chairs on its side walk. Since street life is under the control of the street generator, some coordination between these two units is necessary. This justifies our linear flow of data in figure 6.

The texture generator team has two tasks: first, it has to collect a large library of building textures and other images (parks, trees, fire hydrants, cars, etc), and to systematically catalog them according to the relevant statistical parameters (height, size, zoning, age, architectural style, etc). Second, this team provide function calls to randomly choose textures based on these parameters. For example, the parameter specify "turn-of-century brownstone 4-storey". For robustness, we design the function call to always return some texture, in a best effort manner, rather than choke on unusual parameters. The integrator group takes the output of the street generator, and uses it to call up texture maps from the library. The result is a geometric model, that is stored in database for subsequent visualization.

When the model has been generated, our current integrator also generates a WRL script for visualizing the model. This is rather very slow. Since it is the model generation that we are interested in, we did not consider this to be critical. Any other visualization software could have been substituted.

Issues in Texture Library. The design and construction of a large building texture library is a topic of independent interest. Collection of texture images is a nontrivial task: it is not enough to point a camera at a building and click. The image collected in this way suffers from perspective distortion. Details of the facade may be obscured by trees, vehicles, pedestrians, or by shadow. There is considerable interest and current research on the topic of reconstructing facades and models from multiple images. Some of these techniques can be used to solve our texture collection problems. In any case, our current effort did not extend in this direction (so we simply point and click our camera to collect textures).

Buildings may have one to four (or even more) facades or exposures. Corner buildings must be textured with facades that has at least two exposures. But even a non-corner building, if it is taller than its immediate neighbors, may need up to four exposures. Buildings have crowns, some of which are fairly complicated in shape. How should the library be organized to receive such variations, and still allow easy parametrized retrieval? Should we have a separate collection of building crowns or should they be part of the facade? What would it take to organize a texture library so that we can parametrically generate a bustling neighborhood with a variety of businesses, from restaurants, supermarkets, cafes, drug stores, etc?

For the current project, all these questions are left unanswered. For instance, we only create crownless buildings with flat roots. In figure 2, the two landmarks of Empire State Building and World Trade Center has non-flat flourishes at the top; but these are hand-coded of course.

While our texture library approach could give outstanding realism, it seems to be too labor intensive to be practical. A combination of our library with the procedural texture approach of Parish and Müller[5] should be much easier to achieve.

5. RESULTS AND CONCLUSION

Our work represents a preliminary experiment on statistical generation of large urban landscapes. It raises many interesting research issues that remains to be explored.

Our main contribution is to introduce the statistical propagation technique to complement known procedural methods such as L-systems or noise functions. We plan to explore the combination of our statistical approach with the procedural knowledge method in the future.

This research was originally organized as a class project in a Visualization Class taught by Professor Yap in the Spring of 1998, with teaching assistant Chen Li. Based on this experience, we would recommend this project for its pedagogical value. The end result is a useful artifact, namely software to freely generate large cityscape models.

The quality of the generated model ultimately depend on the the design and choice of appropriate statistical parameters. This should be further investigated.

A further direction which we have not explored is the simplification of such models. This refer to the hierarchical Level-Of-Detail (LOD) models (e.g.,[8]). This would be important when we actually want to perform visualization in real time.

For further information, see project web page `http://cs.nyu.edu/visual/home/proj/manhattan/`. The software for processing street maps in Adobe Illustrator format iis available from this site. This software includes visualization tools based on OpenGL.

ACKNOWLEDGEMENT

The class participants, whose individual contributions were absolutely essential to the final realization of this project, are: H. Biermann, X. Duan, P. Edward, V. Gilinsky, J. Golec, A. Hertzmann, N. Kaushik, J. Meyer, R. Mistry, I. Nemcova, H. Pao, S. Paxia, X. Qian, F. Schneider, J. Tan, H. Wang, G. Wasif, Y. Yeshurun, J. Yung, X. Zhang, Z. Zhang, T. Zhao, and J. Zheng.

Figure 7. Street Map of Manhattan

REFERENCES

1. T. Funkhouser and C. Séquin, "Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments," in *Computer Graphics (SIGGRAPH '93 Proceedings)*, **27**, pp. 247–254, Aug. 1993.
2. T. Funkhouser, S. Teller, C. Séquin, and D. Khorramabadi, "Database management for models larger than main memory during visualization of complex virtual environments," in *Interactive Walkthrough of Large Geometric Databases, Course Notes 32, SIGGRAPH '95*, pp. E15–E60, 1995. Also appeared as "The UC Berkeley System for Interactive Visualization of Large Architectural Models", in *Presence: Teleoperators and Virtual Environments*, Vol.5, No.1, Winter 1996.
3. W. T. Reeves and R. Blau, "Approximate and probabilistic algorithms for shading and rendering structured particle systems," *Computer Graphics Proceedings (SIGGRAPH 85)* **19**(3), pp. 313–322, 1985.
4. P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer, 1990.
5. Y. I. H. Parish and P. Müller, "Procedural modeling of cities," *Computer Graphics Proceeedings (SIGGRAPH 2001)* , pp. 301–308, 2001.
6. "Topologically integrated geographic encoding and referencing system (tiger) homepage," 1995. US Census Bureau. URL http://www.census.gov/ftp/pub/geo/www/tiger/.
7. B. W. Kernighan and C. J. van Wyk, "Extracting geometric information from architectural drawings," in *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, eds., pp. 167–176, Springer, 1996. Lecture Notes in Computer Science No. 1148; Proceedings of Workshop on Applied Computational Geometry (WACG), Federated Computing Research Conference 1996, Philadelphia, USA.
8. A. Varshney, P. Agarwal, J. F.P. Brooks, W. Wright, and H. Weber, "Automatic generation of multiresolution hierarchies for polygonal models," in *Workshop on Simulation and Interaction in Virtual Environments*, pp. 25 – 28, July 1995.