Procedural Modeling of Cities

Yoav I H Parish
parish@imes.mavt.ethz.ch
ETH Zürich, Switzerland

Pascal Müller
pascal@centralpictures.com
Central Pictures, Switzerland

ABSTRACT

Modeling a city poses a number of problems to computer graphics. Every urban area has a transportation network that follows population and environmental influences, and often a superimposed pattern plan. The buildings appearances follow historical, aesthetic and statutory rules. To create a virtual city, a roadmap has to be designed and a large number of buildings need to be generated. We propose a system using a procedural approach based on L-systems to model cities. From various image maps given as input, such as land-water boundaries and population density, our system generates a system of highways and streets, divides the land into lots, and creates the appropriate geometry for the buildings on the respective allotments. For the creation of a city street map, L-systems have been extended with methods that allow the consideration of global goals and local constraints and reduce the complexity of the production rules. An L-system that generates geometry and a texturing system based on texture elements and procedural methods compose the buildings.

CR categories: F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems: Parallel Rewriting Systems, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism, I.6.3 [Simulation and Modeling]: Applications

Keywords: L-system, software design, developmental models, modeling, urban development, architecture

1 INTRODUCTION

Modeling and visualization of man-made systems such as large cities is a great challenge for computer graphics. Cities are systems of high functional and visual complexity. They reflect the historical, cultural, economic and social changes over time in every aspect in which they are seen. Examining pictures of a large-scale city such as New York reveals a fantastic diversity of street patterns, buildings, forms and textures. The modeling and visualization of large-area cities using computers has become feasible with the large memory, processing and graphics power of todays hardware. The potential applications for a procedural creation range from research and educational purposes such as urban planning and creation of virtual environments to simulation. Especially the entertainment market such as the movie and game industry have a high demand for the quick creation of complex environments in their applications.

Visual modeling of large, complex systems has a long tradition in computer graphics. Most of these approaches address the appearance of natural phenomena. Much of the appeal of such renderings lies in the possibility to depict the complexity of large-scale systems, which are composed of simpler elements. Some of these systems include: the simulation of erosion [23], particle based forests [28] and cloud modeling [25].

Grammar-based generation of models (especially L-systems) are employed in computer graphics mainly to create plant geometry [26]. L-systems have evolved into very sophisticated and powerful tools [20, 27] and have been extensively used in the modeling of plant ecosystems described in [8].

1.1 Related Work in Urban Modeling

One approach to modeling existing cities is the use of aerial imagery to extract the buildings and streets thereof, using computer vision methods. There are various research projects that rely on this approach, e.g. [15]. This method can be used to rebuild cities, but is not designed to create new models without photographic input data.

The existing research work concerning the visualization of cities [6, 7, 13, 29] focuses on techniques for data management, fast real-time visualization and memory-usage optimization.

In [1], Alexander et al. describe a pattern language, which consists of over 250 relevant patterns for the successful construction of cities, buildings and houses. They range from very general patterns like “Ring Roads” to very specific ones like “Paving with cracks between the stones”. Since these patterns are not formalized, they cannot be used in the automatic creation process of an urban environment.

Space Syntax has been developed by Hillier [16]. Space syntax can be viewed as a set of theories analyzing the complexity of large-scale spaces, such as cities. It tries to explain human behaviors and activities from a spatial point of view and has been used in the analysis of city pedestrian flows [17] or way-finding processes [24]. This approach is analytical and relies on the availability of city-maps. In the field of architectural interactive design one approach might be mentioned: the *shape grammar* developed by Stiny [31]. This method uses production rules, but instead of operating on strings, a shape grammar defines rules directly on shapes. Shape grammars have been used to generate two-dimensional patterns and in interactive design applications.

1.2 Our approach

We present a system called `CityEngine` which is capable of modeling a complete city using a comparatively small set of statistical and geographical input data and is highly controllable by the user. To our knowledge, there is no such system available, although one very similar project outline is presented under [34]. In [5], a method for generating urban models is presented by refinement of existing geometry. However, in this approach a basic model of the city buildings has to be created manually. Other systems [4, 32] rely on aerial pictures as the main input for building and road placement. Our `CityEngine` creates urban environments from scratch, based on a hierarchical set of comprehensible rules that can be extended depending on the users needs.

In [33], Wegener splits the urban model into subsystems. He states that the subsystems *networks, land use* and *housing* are among the slowest changing elements in an urban environment. Therefore, in our system `CityEngine`, the creation of the city is reduced to generating a traffic network and buildings. Land use data is provided by the user in form of image-maps. When studying maps and aerial photographs of large cities, it is obvious that the streets follow some sort of pattern on different scales. Roads are the transportation medium for the urban population distributed over the area. L-systems have been used in a similar application [20], support branching very well and have the advantage of database amplification [30]; this suggests their potential use to generate convincing large-scale road patterns. We have adapted the model of L-systems to enable the creation of large cities, based on the data that has been collected in [11] on four huge cities around the world, i.e. New York, Paris, Tokyo and London.

Although we simplified the underlying model of the virtual city, the main design goal for the system is easy extensibility. We want to be able to add new subsystems, such as different transportation networks (underground, train) and alternative land uses. To achieve this, we extended the L-system with a higher-level mechanism that makes the addition of new rules very easy.

1.3 Overview

In Chapter 2 we describe the concept and the pipeline of the `CityEngine` system, and the methods used therein. In Chapter 3 the idea of extended L-systems, which allow the implementation of global goals and local constraints is introduced. We demonstrate the use of this extended concept on the creation of the roadmap. Chapter 4 gives a brief overview of the generation of allotments and buildings and explains our proposed mechanism to simplify the texturing of facades. Finally, the results we achieved are shown and discussed in Chapter 5.

2 SYSTEM ARCHITECTURE

The `CityEngine` system consists of several different tools which form the pipeline that is shown in figure 1. In a first step, the input data is fed to the road-generation system, using an extended L-system described in 3.4. The areas between the roads are then subdivided to define the allotments the buildings are placed on. In a third step, by applying another L-system, the buildings are generated as a string representation of boolean operations on simple solid shapes. Finally, a parser interprets all the results for the visualization software. The visualization software should be able to process polygonal geometry and texture maps. This is the case for practically any 3D renderer. Additionally, most scanline renderers support procedural textures, so the proposed mechanism to generate facades of buildings can be incorporated into the pipeline.

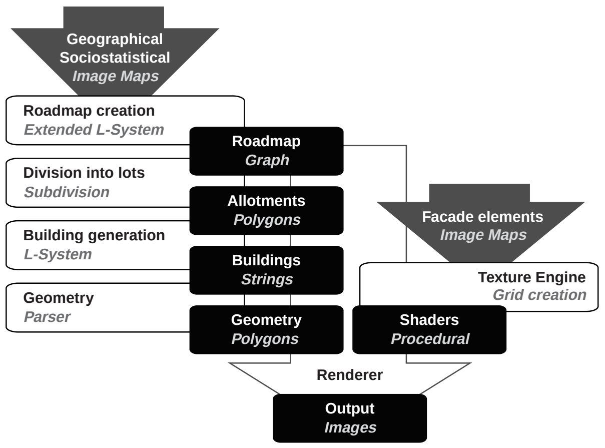


Figure 1: The pipeline of the city creation tool. The dark boxes list the results and data structures of the indiviual tools in the white rectangles.

Most of the input data to build up the virtual city is represented by 2D image maps which control the behavior of the system. Those images can be easily generated either by drawing them or by scanning from statistical and geographical maps, as found in [11]. The data can be categorized into two general classes:

- Geographical Maps
  - Elevation maps
  - Land/water/vegetation maps
- Sociostatistical maps
  - Population density
  - Zone maps (residential, commercial or mixed zones)
  - Street patterns (control behavior of streets)
  - Height maps (maximal house height)

Control of the various parameters within the particular tools can be changed by the user interactively or by providing parameter files. For example, statistical measures such as the approximate area size of a block or the average number of intersections per square mile, such as in [18] can be used to change the resulting street map. In figure 2 the top pictures are examples of a geographical image showing land-water boundaries and another image depicting the distribution of the population over the area.

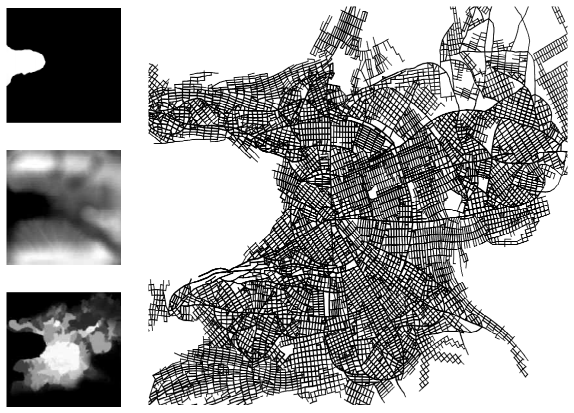


Figure 2: Left column: Water, elevation and population density maps of an imaginary virtual city of 20 miles diameter. Right: One possible roadmap generated from this input data.

Two different L-systems are invoked for the creation of the complete city, one for street, the other for building generation. The population density of a city is influenced by the creation of streets. Through streets, people are transported by the system to the next highway [12]. We reflect this by using an approach similar to the Open L-system model [20] when creating streets. The L-system mechanism has been modified in such a way that various different road patterns can be visualized using the same production rules.

According to [12], not all roads change the population density of their immediate local surroundings, e.g. roads connecting two cities. We therefore chose to consider two types of roads: *highways* and *streets*. They differ in their purpose and behavior: Highways connect areas with highly concentrated population globally, by scanning the population density input map for peaks. Streets cover the areas between highways according to local population density, giving all neighborhoods transportation access to the nearest highway. Together, these two classes form the road map of the virtual city.

Once the road map is generated, the land is divided into smaller areas surrounded by streets. These areas can be geometrically subdivided to define the allotments for the individual buildings. The buildings themselves are generated by a stochastic, parametric L-system. In our system the buildings are composed by extruding and transforming an arbitrary outline.

For the final visualization in the renderer facade textures are generated using a semi-procedural approach. Every facade is tiled into superimposed and nested grid structures. The grid cells resulting from this subdivision can then be assigned textures or procedural textures.

3 CREATING THE STREET MAP

3.1 Extended L-systems

An L-system is a parallel string rewriting mechanism based on a set of production rules. Each string consists of a number of different modules which are interpreted as commands. The parameters for these commands are stored within the modules. When writing a complex rule system to create a street map, there are a large number of parameters and conditions that have to be implemented to the L-system. The number of productions and their complexity grows very quickly. Every time a new constraint is implemented, many rules have to be rewritten. This makes extensibility a very difficult task. Thus, instead of trying to set the parameters of the modules inside the productions, the L-system creates only a generic template at each step. We call this generic template the *ideal successor.*

Therefore, the setting and the modification of parameters in the L-system modules have been ported to external functions. These functions follow a loose hierarchy to distinguish between higher-level tasks and environmental constraints (fig. 3). We call these functions *globalGoals* and *localConstraints*, respectively. Every time the rules are applied to an existing string of modules, the following actions are executed:

- Return the *ideal successor* by the L-system. The parameters of the modules are unassigned.
- Call the *globalGoals* function. This function determines the parameter values of the dominant global goals. All parameter values are set.
- Call the *localConstraints* function. The parameters are checked within the local constraints of the environment. The parameter values are adjusted to fit these constraints. If this function cannot find suitable parameters, it sets the state flag to FAILED and the module is subsequently deleted.

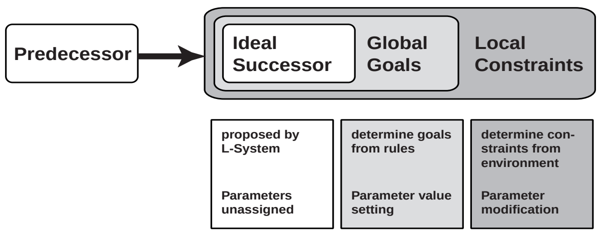


Figure 3: Functions applied to a successor.

When invoked, *globalGoals* and *localConstraints* consider several influences that can set or change the values of the parameters, respectively. The roadmap creation tool distinguishes between the following global goals and local constraints.

- global: street patterns and population density
- local: land/water/park boundaries, elevation, crossing of streets

Through the separation of module succession and parameter calculation into other tasks, the rules for this extended L-system become much simpler. This makes it easy to add new goals and constraints without changing the production rules at all. Thus, for the creation of all our examples of roadmaps only the following set of production rules is used (we follow the notation style used in [20] and [27]):

```
ω:  R(0, initialRuleAttr) ?I(initRoadAttr, UNASSIGNED)
p1: R(del, ruleAttr) : del<0  → ε
p2: R(del, ruleAttr) > ?I(roadAttr,state) :  state==SUCCEED
      {globalGoals(ruleAttr,roadAttr) creates the parameters
        for: pDel[0-2], pRuleAttr[0-2], pRoadAttr[0-2]}
         → +(roadAttr.angle)F(roadAttr.length)
             B(pDel[1],pRuleAttr[1],pRoadAttr[1]),
             B(pDel[2],pRuleAttr[2],pRoadAttr[2]),
             R(pDel[0],pRuleAttr[0]) ?I(pRoadAttr[0],UNASSIGNED)
p3: R(del, ruleAttr) > ?I(roadAttr, state) : state==FAILED → ε
p4: B(del, ruleAttr, roadAttr) : del>0 → B(del-1, ruleAttr, roadAttr)
p5: B(del, ruleAttr, roadAttr) : del==0
         → [R(del, ruleAttr)?I(roadAttr, UNASSIGNED)]
p6: B(del,ruleAttr,roadAttr) : del<0  → ε
p7: R(del,ruleAttr) < ?I(roadAttr,state) : del<0 → ε
p8: ?I(roadAttr,state) : state==UNASSIGNED
      {localConstraints(roadAttr) adjusts the parameters for:
       state, roadAttr}
         → ?I(roadAttr, state)
p9: ?I(roadAttr,state) : state!=UNASSIGNED → ε
```

The axiom ω initializes the L-system with a road module *R* and an insertion query module *?I*. This initial segment needs to be located inside a legal area of the user input data, i.e. not on water or in parks.

The first three productions control the *R* module. The production *p2* controls the road and branch creation. Two branch modules, *B* and a road module *R* plus the insertion query *?I* are created. Their attributes are initialized according to the global goals which returns an array of attributes (*pDel[0-2]* for branch delay and deletion, *pRuleAttr[0-2]* for rule-specific attributes and *pRoadAttr[0-2]* for road data, e.g. length, angle, etc).

The *R* module uses the *del* parameter as a break flag: The *globalGoals* function can set this to a negative value and the R module is deleted in the next iteration by production *p1*. In *p3* any *R* module is deleted if its state is set to *FAILED*, by the *localConstraints* function thereby removing the road segment.

The production *p5* creates a new road segment at the branch points after the delay count (*del*) has reached zero. The *globalGoals* function can inhibit branches by setting their *del* parameter to a negative value. Production *p7* deletes the query module *?I* in cases where the *globalGoals* in *p2* set the delay of the resulting *R* module to a negative value. The last two productions determine the actual creation of the road, checking if all the local constraints are met by calling the *localConstraints* function which adjusts the values in the road attributes *roadAttr.* The state variable is modified by *localConstraints* to either *FAILED* or *SUCCEED* and determines if the road segment is created.

3.2 Global Goals for road creation

Global goals are primarily considered to set the parameters of the modules to their initial values. The system weighs the influences of all active global goals and chooses the appropriate values for the parameters. The influence of a particular goal at any point is controlled by the input image maps.

3.2.1 Population density

As mentioned above, the highways build the main connection medium between different highly populated areas of a city whereas streets develop into the residential areas and give the habitants access to the next highway [12]. The determination of the parameters of a highway segment are described as follows:

Highways connect centers of population. To find the next population center, every highway road-end shoots a number of rays radially within a preset radius. Along this ray, samples of the population density are taken from the population density map. The population at every sample point on the ray is weighted with the inverse distance to the roadend and summed up. The direction with the largest sum is chosen for continuing the growth. This mechanism is illustrated in figure 4.

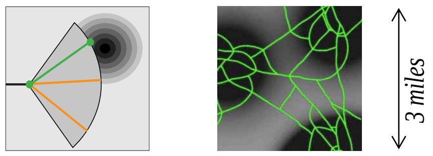


Figure 4: Left: The road-end of every highway shoots radial rays to find the next population peak (dark area). Right: A possible highway net, connecting population centers.

On the contrary, streets do not change their direction by following the steepest gradient of the population density map. Typically they follow the dominant street pattern. This is reasonable, since most streets in urban areas are built following a superimposed pattern plan. The created street ends still lower the population density in their environment, usually within the area of the block, according to the mechanisms proposed for Open L-systems [20]. When an area with no population is reached, the streets stop growing.

3.2.2 Road patterns

The other important global goal is the compliance of roads with the dominant patterns in that area. This applies for both highways and streets although in our urban model, streets are limited in the possible patterns they can form. In figure 5 various frequent road patterns are listed [9].

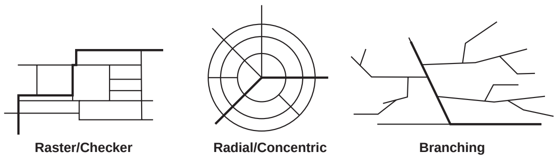


Figure 5: A selection of frequent road patterns.

Most real cities display a number of road patterns due to historical growth and different phases of city creation. In [12], Fuesser categorizes streets into two classes: main streets and settlement-oriented side-roads. In his analysis, he compares raster with radial patterns. In polycentric cities raster patterns prevail, whereas in monocentric cities radial patterns dominate. Looking at the road types, Highways can represent both patterns, streets usually appear in raster or raster-related forms. In [11] the distinction of different zones in the city leads to the conclusion that the central zone (Zone 1) is of such high population density that it can be viewed as polycentric city. Therefore, in high density zones the raster pattern is dominant.

We have adopted a selection of patterns to apply on the street map and group them into different rules.

- *Basic rule*: This is the simplest possible rule. There is no superimposed pattern and all roads follow population density. This may also be referred to as the natural growth of a transportation network. Mainly older parts of cities show such patterns. All other rules are based on restrictions of this rule by narrowing the choices of branch angles and road segment length.
- The *New York* (or checkers) *rule* follows a given global or local angle and maximal length and width of a single block. This is the most frequent street pattern encountered in urban areas, where all highways and streets form rectangular blocks.
- *Paris* (or radial) *rule*: The highways follow radial tracks around a center that can be either calculated from the input data or set manually.
- *San Francisco rule*: This pattern lets streets and highways follow the route of the least elevation. Roads on different height levels are connected by smaller streets, that follow steepest elevation and are short. This pattern is usually observed in areas with large differences in ground elevation. See figure 6 for an example.

The following table summarizes the goals that the road creation follows:

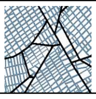
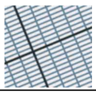
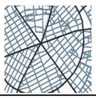
Pattern name	Pattern	Example
Basic	No superimposed pattern.	
New York	Rectangular Raster	
Paris	Radial to center	
San Francisco	Elevation min or max	see figure 6

Table 1: An overview of the street pattern used in the CityEngine system with a short description and an example.

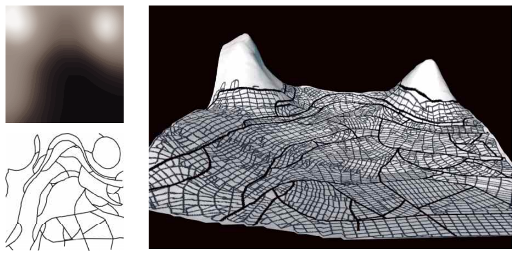


Figure 6: Left: Elevation map and the resulting highway map using the San Francisco rule. Right: Street map projected onto the elevated geometry.

If more than one pattern rule is active at a given location, all of them are evaluated. The proposed parameter values are summed up and weighted according to the value in the input image grey scale map. This allows an easy blending of different street patterns over a defined area as shown below in figure 7.

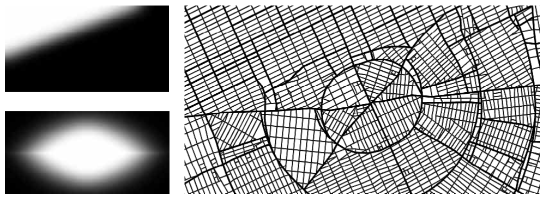


Figure 7: Left: Two different street pattern control maps for New York and Paris rule. Right: The resulting street pattern of the two maps overlaid.

3.3 Local constraints

The *localConstraints* function adjusts the parameter values proposed by the *globalGoals* function to the local environment. Whenever a special situation in the environment is encountered, these values are changed. If there is no possibility to find a valid set of parameters, the function sets the state of the module to *FAILED*. It is then deleted from the module string by another production in the L-system. As an extension to existing L-systems our implementation allows the creation of closed loops and intersections of road branches. In the `CityEngine` system, streets and highways are subordinate to the same local constraints.

The *localConstraints* function executes the two following steps:

- check if the road segment ends inside or crosses an illegal area.
- search for intersection with other roads or for roads and crossings that are within a specified distance to the segment end.

If the first check determines that the road end is inside water, a park or another illegal area, the system tries to readjust the values for the road segment in the following ways.

- Prune the segment length up to a certain factor so that it fits inside the legal area of the starting point.
- Rotate the segment within a maximal angle until it is completely inside the legal area. This allows the creation of roads that follow a coastline or a park boundary.
- Highways are allowed to cross illegal area up to a specified length. The generated highway segment is flagged. At the geometry creation stage it can then be replaced by e.g. a bridge, or two tunnel entrances on both sides.

Once all road ends are checked for being inside legal territory, the system now scans the surrounding of the road ends for other roads to form crossings and intersections.

3.3.1 Self-sensitive L-Systems

The idea of L-system branches growing together and forming closed loops could not be found in existing applications, although a similar approach to the one we propose can be found in the procedural generation of blood vessels [21]. In traffic systems the dead end road is the exception. Most roads end when crossing other roads or circling back to themselves [9]. The forming of loops changes the topology from a tree-like to a net-like structure. Since we do not have an underlying model of transportation flow simulation on our roadmap, we can ignore the implications on street capacity.

At first the segment is checked for intersection with other segments. For this, all existing street segments in the surrounding area are checked for intersection with the new segment. If one is found, the road is pruned and a crossing is generated.

If the *localConstraints* function finds a street within the given radius of the end of a segment it can modify the parameters for the following events illustrated in figure 8:

- two streets intersect $\rightarrow$ generate a crossing.
- ends close to an existing crossing $\rightarrow$ extend street, to reach the crossing.
- close to intersecting $\rightarrow$ extend street to form intersection.

If a crossing is generated, a new node and two edges in the existing street graph have to be created. After a crossing is generated, a road stops growing.

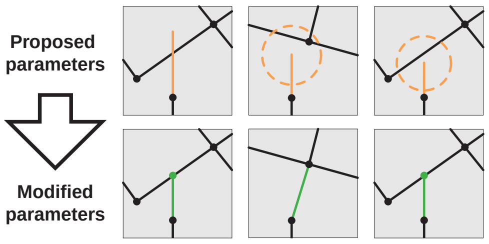


Figure 8: Examples of local constraints. Upper row: Proposed parameters by global goal. Lower row: after parameter correction.

Once the values of the parameters are finalized, the street map can be rendered or output to the next step in the pipeline. Since the system only creates straight road segments a method is needed to smooth roads with a high curvature. The road creation tool does this by implementing a simple subdivision scheme based on [3].

Figure 9 below shows an example of the creation of a roadmap using scanned maps of Manhattan island. The oldest part of Manhattan is first generated following no generation pattern. All newer parts of the city are created using the New York rule. The local constraints change the direction of the highway to follow the coastal line along the island. Note how the proposed bridges cross the water very close to the locations of where the real bridges are built.

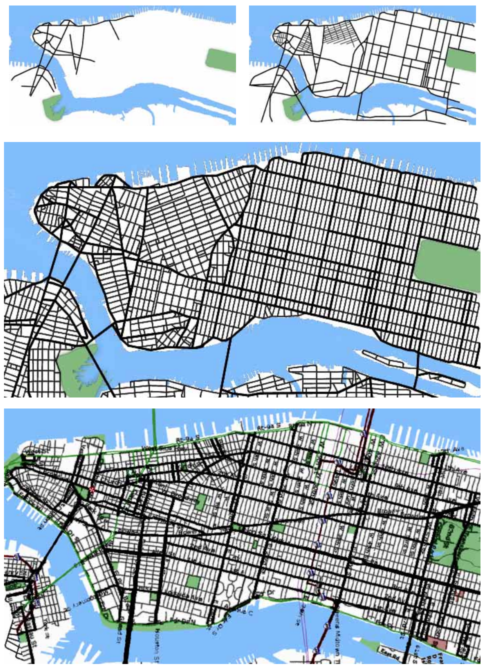


Figure 9: Street creation system applied to Manhattan. Top row: The network after 28 and 142 steps. Middle: The final roadmap. Lower: A real map of Manhattans streets for comparison.

4 MODELING OF BUILDINGS

Once the road map is generated, the system creates the allotments for the placement of the buildings. A stochastic, parametric L-system then generates the geometry for the buildings. Every building is assigned a texture. To keep memory usage small, a semi-procedural approach to texturing has been taken.

4.1 Division into lots

After the creation of highways and streets the populated area of the city is subdivided into many small areas which we call *blocks*. Those have to be divided into lots for the placement of buildings, as shown in figure 10. We assume that most of these allotments are convex and rectangularly shaped. The system therefore forbids the creation of concave allotments. A block is divided into smaller units using a simple, recursive algorithm that divides the longest edges that are approximately parallel until the the subdivided lots are under a threshold area specified by the user.

In most cities there are regulations that control the maximum height of a building through the ground area of the allotment and zoning plan. Therefore, the user can control the maximum height of a building through the usage of an image map and restrict the generation of skyscrapers to certain zones. After the subdivision of the blocks, all allotments that are too small or do not have direct access to a street are discarded from the system.

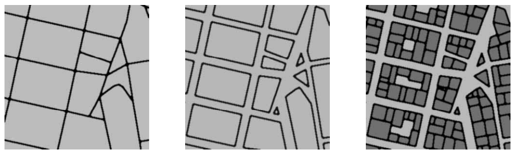


Figure 10: Left: Resulting street map. Middle: Blocks created by scaling from street crossings. Right: The generated lots. Allotments too small or without street access are removed.

4.2 Geometry

All buildings in our virtual cities are modeled with a parametric, stochastic L-system. For every allotment one building is generated. To follow the different styles, we consider three types of buildings: skyscrapers, commercial buildings and residential houses. These are determined by the zoning rules, controlled through the use of image maps. For every type of building a different set of production rules is executed.

Buildings are created by manipulating an arbitrary ground plan. The modules of the L-system consist of transformation modules (*scale* and *move*), an extrusion module, branching and termination modules, and geometric templates for roofs, antennae, etc. The final shape of the building is determined by its ground plan which is transformed by interpreting the output of the L-system. Although a large variety of building types can be generated this way this is a limitation of the system, as the functionality of the buildings can not be represented using only these simple rules. Nevertheless, a high degree of visual complexity can be reached as illustrated below in figure 11.

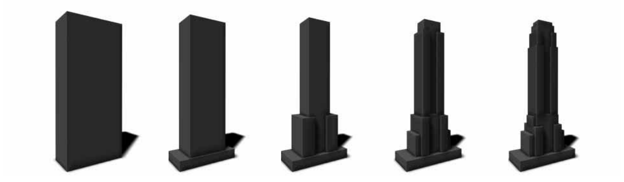


Figure 11: Five consecutive steps of the generation of a building. The axiom of the L-system is the bounding box of the building, allowing easy LOD-generation.

To allow for an automatic level-of-detail model creation, a restricted ‘decreasing apices’ L-system class as described in [14] can be used. Since the generation of the building starts with its bounding box as the axiom, the output of each iteration can be interpreted as a refining step of the geometry.

The output of the L-system is fed to another parser, which translates the resulting string into geometry readable by the visualization systems.

4.3 Textures

A high degree of scene detail and complexity can be achieved through the use of detailed textures on the buildings. In existing applications, pictures of actual buildings are scanned, modified and projected onto the surfaces of the building geometry. Although this method reproduces the most detailed facade, the amount of work to prepare the textures is too high compared with our geometry generation time. Also, for a large number of buildings, memory limitations pose a major problem on many systems. Most of these difficulties can be addressed by the use of procedural textures [10]. Unfortunately, not all the smaller details that make up the appearance of a facade can be modelled by such textures. Certain patterns like stone and brick walls can be analyzed and synthesized ([22], [19]) but the description of a general house facade cannot be modelled by these approaches. Therefore we decided to design a tool for the semi-automatic creation of facades using layering and a simple functional composition technique as discussed in [10] we call *layered grids*.

Our design is based on some observations of facades. To simplify our model, the following assumptions about facade textures have been made:

1. Facades show one or several overlayed or nested grid-like structures where most grid cells accomplish the same function, i.e. typically windows and doors for openings.
2. Particular grid cells influence the positions and/or sizes of the surrounding grid cells e.g. windows at ground level or above a door have different sizes.
3. Irregularities in the grid structure mostly affect complete rows and columns of the grids, not single grid cells.

Our goal was to create procedural textures that enable the user to capture a certain style of facade by creating a generic *style texture*. Every single style texture should produce a rich variety of different textures in the defined style regardless of the facade dimensions. Since images of facade elements can capture the intricate detail very well, the texturing system should still be able to use scans of facade elements like bricks, doors and windows.

The hierarchical grid system is based on interval groups. An interval groups is a set of non-overlapping, ordered intervals. The advantage of using one-dimensional intervals as the basis of texturing is that rows and columns can be changed by modifying single intervals in the interval group. This corresponds to the third assumption listed above. Nevertheless, we still have the possibility to access particular grid cells.

Two arbitrary interval groups can be combined to form a two dimensional layer as shown in figure 12. A layer is defined by two interval groups, an evaluation function *eval* between the interval groups and a color evaluation function *col*. For each point *(s,t)* where the function *eval(s,t)* is evaluated to 1 the point is considered as an *active* point in the layer. All active points in a layer are the *active area* of a layer. If this set is partitioned, the partitions are called *active grid cells*.

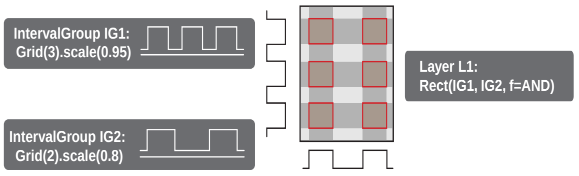


Figure 12: Left: Two interval groups define a layer via a function (logical AND in this example).

If a point *(s,t)* is considered active, the color function *col(s,t)* is called and returns the color (or bump or reflectivity) value of the point. Non-rectangular active areas can easily be created by assigning functions to the intervals in the interval groups. In all the examples presented here, we assigned a simple PULSE function [10] to the interval groups and logical functions (*AND, OR*) for the evaluation function *eval*.

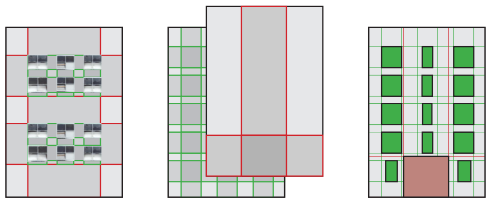


Figure 13: Left: A nested layer evaluated as a random image map. Middle: A stack of layers. Right: The red layer influences the scales of active cells in the green layer.

The evaluation function of a layer can be another procedural texture, an image map or another nested layer or layerstack described below. For example, when using an image map of a window we used a number of pictures showing the window open, half-opened and closed. By randomly applying the different window to each cell, a great visual complexity of the facade is easily established.

Layers can be stacked which means that if a point *(s,t)* on layer *l* is evaluated as not active the same point is evaluated on layer *l-1*. Different superimposed grid-like structures can be easily modelled this way. To establish the influence of different layers on each other, functions between layers can be established. The principle of this mechanism is outlined in figure 13 above and shown in an example in figure 14 below.

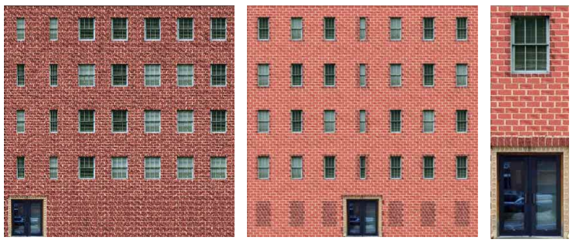


Figure 14. Left and middle: Brickhouse facade rendered with different parameters. Right: Differently sized and colored bricks around the window and the door, as an example of functions between layers.

A limitation of the system at the moment is that each style texture has to be defined manually. As shown in figure 15, this is done by visually determining the regularities and measuring facade element sizes. Once a shader is defined the texture can scale to any width or height.

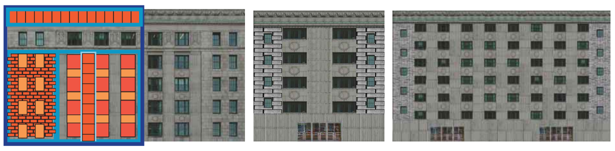


Figure 15. Left: The original facade picture partially overlaid with the grid structure, where blue frames and red grid cells are nested and random image layers, respectively. Middle and right: Two facades in that style at different sizes.

5 RESULTS

Here we present a couple of images that result as the output from the `CityEngine` system. The first visualization is performed on a real-time platform, whereas the second example was rendered using a raytracer.

Shown in figure 16 are screenshots of a 'virtual' Manhattan, displayed in the real-time visualization software Division *dvreality* 5.0. The model covers Manhattan island and consists of approximately 13'000 buildings. The creation of the street graph took less than ten seconds, the division into lots and the creation of buildings approximately 10 minutes. The buildings were extruded from the shape of the allotment and automatically textured, therefore we have not used level-of-detail representations of the buildings in this visualization. Since the software does not support the dynamic generation of textures we used a set of conventional facade textures.The bridge was added to the city model manually.

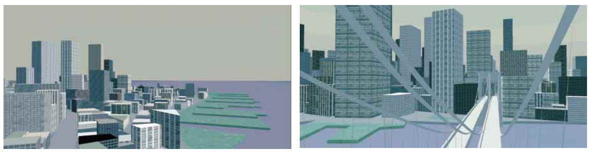


Figure 16. Screenshots of the flythrough visualization.

Our second example is a different automatically generated model of Manhattan and a city generated from the data in figure 2. In this example we use geometry generated by the second L-system and textures created through layered grids. The final pictures in figure 17 and figure 18 were rendered using Alias/Wavefront's Maya 3.0.

6 FUTURE WORK

We have presented a system that is capable of generating the layout of a large-scale city based on 2D-input data. In contrast to existing similar systems our approach does not depend on aerial photography of streets and buildings and is able to generate an infinite number of cities using the same image input very quickly. The system generates street maps through extended L-systems which were extended to handle many different requirements on a global and local level without increasing the complexity of the core system and creates building geometry on the subsequently generated allotments. A basic texture mechanism was introduced that captures the nested grid-like structures of facades in procedural textures.

With such a complex system as a city creation system there are many interesting problems that could lead to more realistic results. Many of these were not addressed in this work and might prove interesting extensions to the software.

Global Goals for road creation: Many more rules can be added to let the road creation mechanism behave more realistically. The selection from a pool of goals could follow a behavioval scheme using inhibition and level-of-interest [2].

Simulation and analysis of growth: Showing the evolution of a city poses new problems. Transportation flow simulation could be implemented into the system and trigger the roadmap growth process.

Buildings generation: A better mechanism for the automatic generation of buildings could be developed by dividing the space of a house into functional units and combining them to generate new buildings by means of a similar production system as used for the road creation.

Visualization: The creation of procedural texture styles could be automated by assisting the user to find grid structures and facade elements through methods of computer vision and automatically creating the texture style shader.

Acknowledgments

We would like to thank Prof. Markus Gross for his continous support of our work. Furthermore, we would like to thank Hanspeter Brunner and Christian Iten for helping with the models and Jessica Bernatschek for working with us on the text.

REFERENCES

[1] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel. *A Pattern Language*. Oxford University Press, New York, 1977.
[2] B. M. Blumberg and T. A. Galyean. Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments. In *SIGGRAPH 95 Conference Proceedings*, pages 47-54, August 1995.
[3] E. Catmull and J. Clark. Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, 10(6):350-355, 1978.
[4] CGSD. Parametric Planets Software. http://www.cgsd.com/ParametricPlanets.
[5] Max Chen. Generation of Three-Dimensional Geometry for Night Illumination and Urban Visualization. http://graphics.lcs.mit.edu/~maxchen/Boston.html, May 1999.
[6] D. Davis, W. Ribarsky, T.Y. Jiang, N. Faust and S. Ho. Real-Time Visualization of Scalably Large Collections of Heterogeneous Objects. *IEEE Visualization '99*, pp. 437-440, October 1999.
[7] X. Decoret, G. Schauffler, F. Sillion and J. Dorsey. Multi-layered Impostors for Accelerated Rendering. *Eurographics* 18(3), 1999.
[8] O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr and P. Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. In *SIGGRAPH 98 Conference Proceedings*, pages 275-286, August 1998.
[9] K. Dietrich, M. Rotach, E. Boppart. *Strassenprojektierung*. Zurich 1993.
[10] D. Ebert, F.K. Musgrave, D. Peachey, K. Perlin, S. Worley. *Texturing & Modeling. A Procedural Approach.* 2nd. Edition, Academic Press, 1998.
[11] C. Focas (ed.) *The Four World Cities Transport Study.* London Research Centre, The Stationery Office, London 1998.
[12] K. Fuesser. *Stadt, Strasse & Verkehr (City, Roads and Traffic)*, Vieweg Verlag, 1997.

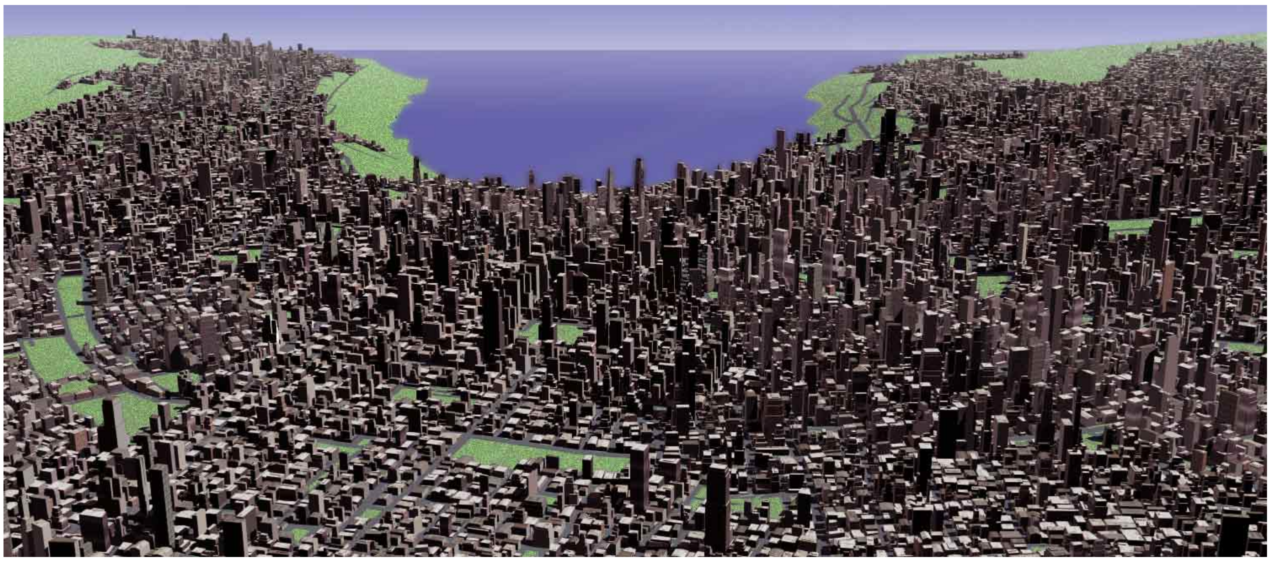


Figure 17. A virtual city modelled using the data from figure 2. Approximately 26000 buildings were created.

[13] T. Fujii, K. Imamura, T. Yasuda, S. Yokoi and J. Torikawi. A Virtual Scene Simulation System for City Planning. *Computer Graphics International*, 1995.

[14] J.C. Hart. The Object Instancing Paradigm for Linear Fractal Modeling. In *Proceedings of Graphics Interface 92*, pages 224-231, 1992.

[15] O. Henricsson, A. Streilein and A. Gruen. *Automated 3-D Reconstruction of Buildings and Visualization of City Models*. Bonn, Oct. 1996.

[16] B. Hillier. *Space is the Machine: A Configurational Theory of Architecture*. Cambridge University Press, Cambridge, UK, 1997.

[17] B. Hillier, A. Penn, J. Hanson, Grajewski and J. Xu. Natural Movement: or, Configuration and Attraction in Urban Pedestrian Movement. *Environment and Planning B*, Vol. 20, pp. 29-66, 1993.

[18] A.B. Jacobs. *Great Streets*. The MIT Press, Cambridge Massachusetts, 1993.

[19] L. Lefebvre and P. Poulin. Analysis and Synthesis of Structural Textures. In *Graphics Interface 2000 Proceedings*, pages 77-86, May 2000.

[20] R. Mech and P. Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. In *SIGGRAPH 96 Conference Proceedings*, pages 397-410, August 1996.

[21] V. Meier. *Realistic Visualization of Abdominal Organs and its Application in Laparoscopic Surgery Simulation.* Dissertation, ETH Zurich, 1999.

[22] K. Miyata. A Method of Generating Stone Wall Patterns. In *SIGGRAPH 90 Proceedings*, pages 387-394, 1990.

[23] F.K. Musgrave, C.E. Kolb and R.S. Mace. The Synthesis and Rendering of Eroded Fractal Terrains, In *SIGGRAPH 89 Proceedings*, pp. 41-50, July 1990.

[24] J. Peponis, C. Zimring and Y.K. Choi. Finding the Building in Wayfinding. In *Environment and Behavior*, Vol. 22, pp. 555-590., 1990.

[25] K. Perlin. An Image Synthesizer. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3): 287-296, 1985.

[26] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*, Springer, 1990.

[27] P. Prusinkiewicz, M. James and R. Mech. Synthetic Topiary. In *SIGGRAPH 94 Conference Proceedings*, pages 351-358, July 1994.

[28] W.T. Reeves and R. Blau. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3): 313-322, 1985.

[29] S.M. Rubin and T. Whitted. A 3-dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics* 14(3), pages 110-116, 1980.

[30] A.R. Smith. Plants, Fractals and Formal Languages. *Computer Graphics (SIGGRAPH 84 Proceedings)*, 18(3):1-10,1984.

[31] G. Stiny. *Pictorial and Formal Aspects of Shapes and Shape Grammars*. Birkhauser, Basel, Switzerland, 1975.

[32] Virtual Terrain Project. http://www.vterrain.org.

[33] M. Wegener. Operational Urban Models: State of the Art. *In Dortmunder Beiträge zur Raumplanung* No. 84, University of Dortmund, 1998.

[34] C.Yap, *The Other Manhattan Project, Project description*. http://www.cs.nyu.edu/visual/home/proj/manhattan, 1998.

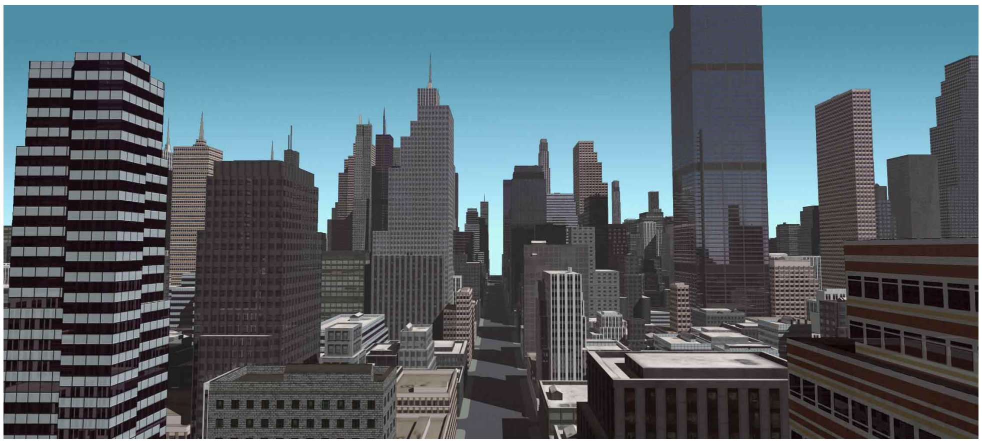


Figure 18. Somewhere in a virtual Manhattan.