EUROGRAPHICS 2009 / P. Dutré and M. Stamminger
(Guest Editors)

Volume 28 (2009), Number 2

Interactive Geometric Simulation of 4D Cities

Basil Weber[1,2], Pascal Müller[2], Peter Wonka[3], and Markus Gross[1]

[1]ETH Zürich, Switzerland
[2]Procedural Inc.
[3]Arizona State University, USA

Abstract

We present a simulation system that can simulate a three-dimensional urban model over time. The main novelty of our approach is that we do not rely on land-use simulation on a regular grid, but instead build a complete and inherently geometric simulation that includes exact parcel boundaries, streets of arbitrary orientation, street widths, 3D street geometry, building footprints, and 3D building envelopes. The second novelty is the fast simulation time and user interaction at interactive speed of about 1 second per time step.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Simulation and Modeling [I.6.3]: Applications—

1. Introduction

In this paper we tackle a new interdisciplinary research problem that has a significant geometric modeling component: geometric urban simulation. While urban simulation has been investigated for many years, even state-of-the-art micro-simulation systems such as UrbanSim [Wad02, WBN*03] do not simulate detailed geometric aspects. Instead, they use regular grids as spatial data structure. Each grid cell represents a collection of households and businesses, but no geometry is simulated. We aim at including detailed urban geometry in an urban simulation, such as exact parcel boundaries, streets of arbitrary orientation, street widths, 3D street geometry, building footprints, and 3D building envelopes. This will allow us to create 4D cities *i.e.* interactive three-dimensional urban environments that change over time. An example is shown in figure 1.

Our system was motivated by three applications of 4D cities. The most important application is the creation of dynamic content for interactive entertainment, especially for the emerging 3D Web. For example, metaverses such as Second Life and massive multi player games such as World of Warcraft all are experimenting with the inclusion of dynamic contents. The second application is educational gaming to teach students about sustainability in urban systems similar to the famous simulation SimCity. We can enrich virtual worlds by simulating time-depended phenomena such as urban growth, transitions in land use, changes in building density, changes in the wealth and cleanliness of urban neighborhoods, and changes due to transportation policies or infrastructure. The third application is to make urban simulation more comprehensible by visualizing alternative futures of urban environments.

We believe that the best approach to tackle the problem is to make the simulation inherently geometric. We therefore create an urban simulation system by integrating geometric rules and decisions for street expansion, traffic computation, land use simulation, lot subdivision, and building envelope construction. Our solution focuses on quickly recreating urban patterns such that medium size cities can be generated interactively (which is significantly faster than existing simulations). Please note that we limit the scope of this paper to show how to use urban simulation as an efficient modeling tool for computer graphics. The question that we try to answer is how to model the three-dimensional geometry of urban environments as it changes over time. Only in future work we want to evaluate how urban planning applications can benefit from our faster and more detailed simulation.

The main contributions of our work are: (1) We extend previous procedural urban modeling methods in computer graphics by modeling cities over time. Instead of creating a single static city, we can create a sequence of urban configurations. (2) We are the first to propose an urban simulation system that includes realistic geometric configurations that are not forced into a regular grid. This leads to a significant

Figure 1: *We present an interactive simulation system that can simulate a three-dimensional urban model over time. Here, we start the simulation by seeding with a single street segment to grow the urban environment. The user can interactively guide the simulation. Total simulation time: 51s.*

improvement of visual quality. (3) We present a fast simulation at interactive speed that can be edited during simulation.

Please note that this paper excludes the modeling of several changes to the three-dimensional geometry, such as the influence of weather on materials, facade renovations, or adding new rooms to existing buildings. The simulation is mainly performed in two dimensions, but our geometry-based approach (not grid-based) enables us to generate three dimensional cities developing over time.

1.1. Related Work

Our work is most closely related to *procedural urban modeling*, especially street network modeling through L-systems. The latter are a popular formalism for procedural modeling that was successfully applied to plant modeling [PL91]. Based on this pioneering work, several authors adapted the idea of L-systems to street network modeling [PM01, CEW*08, CBAN07]. After a street network is generated, shape grammars [Sti75, WWSR03, MWH*06] can be used to generate a city model including building geometry. While these approaches model one static city with high visual and geometric quality, none of these methods integrate simulation to obtain meaningful urban growth over time or land use.

Urban simulation in urban planning and social sciences became fairly complex and current systems combine several methods such as spatial interactions using gravity models [Gol71], cellular automata [Cou97, Ben97], agent-based modeling [LZ98a, LZ98b], random utility theory [Mcf74], and rule-based simulation [Klo99]. A good review of existing urban simulation can be found in [WU04]. The probably most advanced and comprehensive system for regional simulation is UrbanSim [Wad02, WBN*03]. UrbanSim can be seen as system of interacting urban simulations. Due to the complexity of urban environments, previous work in simulation uses drastic geometric simplifications and aggregation. For example, UrbanSim uses grid cells with a side length of 150m. While this is considered a micro-simulation it produces a gridded output and is still at a scale that does not capture any interesting visual geometric details for three-dimensional urban visualization.

Initial urban simulation work in computer graphics improves the visual quality of the output by adding geometry [HMFN04, Wat07, LWW*07] or image fragments [AVB08, VABW08] in a post process.

The creation of a geometric urban simulation requires studying *urban design* and *land use planning*. There are several interesting books that we can recommend [Pun99, Leu03, BGKR06]. The challenge is that geometric information and rules that could be used to create a geometric urban simulator are rare, even in Christopher Alexanders famous book about design patterns [AIS77] and Space Syntax [Hil96, Hil98]. The creation of geometric rules and decisions requires synthesis from many different sources and is a contribution of this paper. Architects and urban planners are typically very strong at drawing and visual presentations. Therefore, we found sources with many illustrations ultimately the most helpful (e.g. [GRC*04]) as the synthesis of geometric rules in the simulation had to be mainly our own work.

2. Overview

The major challenge of this design is to integrate various components into a coherent framework and to make sure that all necessary geometric quantities are simulated. One of our important insights was to let the simulation first create planned geometry and have the actual building lag by several time steps. That allows to simulate the effect of planning into the urban simulation.

2.1. City Hierarchy Definitions

An urban layout is a hierarchical geometric configuration, consisting of a street network and city regions such as quarters, blocks and lots (illustrated in figure 2). The *street network* is a planar graph (V,E) with nodes V and linear edges E. The nodes $node[i] \in V$ have the following attributes: a two-dimensional location $node[i].pos \in R^2$, a flag to denote the hierarchy $node[i].hierarchy \in \{major, minor\}$ and the growth flag $node[i].growth \in \{unfinished, finished\}$. The edges $street[j] \in E$ are line segments with the following attributes: the indices $street[j].nodes \in \mathbb{N}^2$, the status

flag $street[j].status \in \{planned, built\}$, the classification in the hierarchy $street[j].hierarchy \in \{major, minor\}$, and a street width $street[j].width$ in meters. The induced cycles of the street graph (V, E) give rise to *blocks* (represented as polygonal faces). The *quarters* are induced cycles of the graph (V_{major}, E_{major}) *i.e.* all nodes and edges are marked as *major*. The blocks can be further subdivided into building *lots*. The latter store a land use type $lot[k].lut$ and a scalar land use value $lot[k].luv$ describing its suitability.

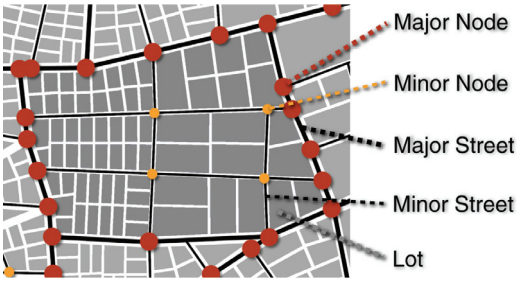

Figure 2: *The hierarchy of a city. In our definition, an urban layout consists of major and minor streets, quarters, blocks and lots. For example, the quarter shown in the center consists of eight blocks.*

2.2. System Pipeline

The input to our pipeline consists of: (1) the environment i.e. the topography given as a height map, a water map, and a forest map, (2) an initial urban layout configuration ranging from a single street up to a larger (potentially already existing) city, (3) land use type definitions (see section 4), and (4) user input data that can change over time to interactively control the simulation. In table 1 we list the most important user inputs. Our system pipeline is illustrated in figure 3.

Parameter	Description
height map	terrain as floating point image
$citycenter[i]$	one or multiple city centers $\in \mathbb{R}^2$
$growthcenter[i]$	one or multiple growth centers $\in \mathbb{R}^2$
$streetgrowth[t]$	percentage of new streets per year
$avgprice[t]$	average land price per year
street pattern[i]	patterns defining street expansion
$land\ use\ type_t$	a set T of land use type definitions
$goal_t$	land use percentages for all $t \in T$
$setback_t$	construction setback values
$shape\ grammar_t$	building generation rules

Table 1: *The most important user inputs in our system.*

First, the street network of main streets expands according to the centers and the growth rate $streetgrowth[t]$ (see section 3). New street segments $street[i]$ are generated by a stochastic growth process. For each street segment, we compute a traffic value $street[i].traffic$ and update the traffic

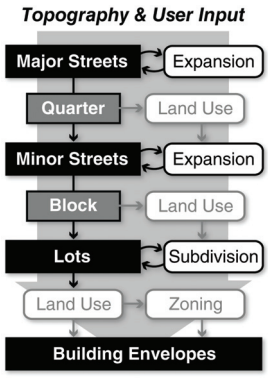

Figure 3: *Our system grows an initial urban layout according to the given environment and user input. It consists of the following components: street expansion including traffic simulation, land use simulation, building lot generation (subdivision) and zoning regulation. The output of the system is a city layout (consisting of streets, lots and building envelopes) that changes over time.*

in the local neighborhood by using a traffic demand model. This results in new *planned* major streets. And as soon as enough traffic is on a new street, its status flag $street[i].status$ is set to *built*.

When a new quarter is generated by street expansion, its dominant land use is estimated. The land use simulation searches for the optimal land use type according to the user-given definitions (see section 4). Note that our framework is highly *generic* and can easily be configured to work with a large number of land use types. Then, according to the quarter's land use, the minor streets are generated, again with the street expansion algorithm. This results in new minor streets.

When a new block is generated by the minor streets, its land use is estimated based on the same simulation algorithm as above. The block is also recursively subdivided according to its land use value resulting in one or several lots (see section 5). The land use simulation is performed again on the lot level. If the new lot is located near a built street, given zoning regulations are computed, (according to the land use type) resulting in the final building envelope. The latter describes the volume wherein the final building will be constructed.

Our pipeline is designed such that individual components can be executed in parallel or sequentially. In the parallel mode each component of the whole pipeline is executed after each street expansion. In the sequential mode each component is scheduled once after a given time step (usually one month for interactive application or one year for video production). Thus we first expand the major streets according to the $streetgrowth[t]$ parameter for one time step, then develop the emerging quarters, blocks, and lots and finally compute the traffic and land use accordingly. The outcome of the simulation is a 4D city. Figure 4 shows an example and in section 6 we demonstrate several case studies.

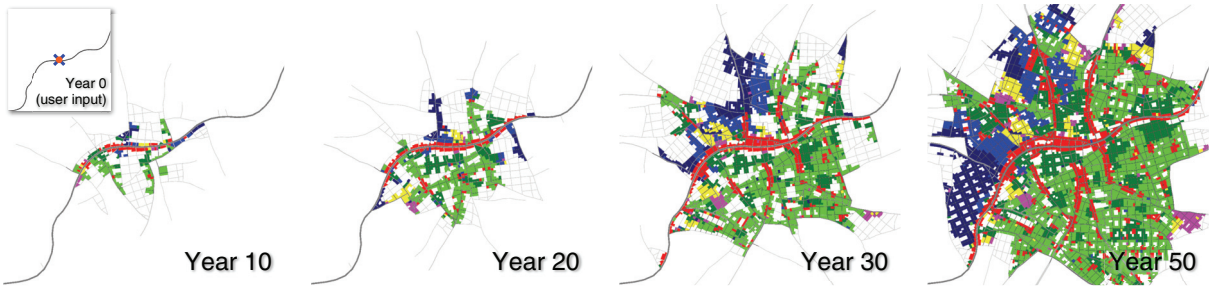

Figure 4: *Example simulation of a city on a given street with one center (city and growth center are identical). This initial configuration defined by the user is depicted in the small figure top left. The resulting lot land use is encoded in colors as in table 5, planned streets are colored light gray and built streets dark gray. Note* e.g. *how the industrial zones (blue) relocate over the years. Total simulation time: 58s.*

```
while (expanding) {
   node = sample(stage);
   street = expand(node);
   street = adapt(street);
   if (legal(street)) {
      add(street);
      updateBlocks();
   }
}
```

Table 2: *Pseudo code of the street expansion algorithm. Note that we use the same code for both stages of the algorithm, just the sampling strategy depends on the stage.*

3. Street Expansion

The expansion simulates growth of the street network. While the expansion strategy is inspired by [PM01], our approach has several differences: 1) In the original paper only the final result was meaningful, but the streets were not built in the correct order. In contrast, we use grow seeds (nodes which generate new street segments) in a temporally meaningful order. This can be done by selecting grow seeds via stochastic sampling of the existing nodes, rather than seed propagation typically used in context-sensitive L-systems. As a consequence we can incorporate user input specifying preferred expansion directions. 2) We chose not to embed the expansion in an L-system framework to make the implementation more efficient. 3) We present an incremental traffic demand model which can be integrated in our more general approach. This traffic demand model heavily influences the land use simulation and is necessary to derive the widths of the streets.

3.1. Expansion Strategy

Our solution to the street expansion has two stages: (1) Expanding the major streets and (2) filling the resulting quarters with minor streets. Table 2 gives a brief overview of the expansion. The two stages differ in the sampling strategy only. First, we compute the total number of streets to be generated by multiplying the current number of streets, $streetgrowth[t]$, and the length of a time step. In the first stage, we stochastically sample the street network's *unfinished major* nodes to select where a street expansion can be applied, as follows: (1) The probability of selecting a node is proportional to $e^{-f\|node[i].pos-growthcenter[j]\|^2}$ which depends on the distance from the nearest growth center j and the user-controlled variable f (defining how strongly focused the growth is). (2) The overall ratio of the number of major nodes with valence 2 (*i.e.* two incident major streets) and the number of major nodes with valence 4 should correspond to a user-defined ratio (per street pattern). This assures that the size of quarters is fairly uniformly distributed. If a new quarter originates, the algorithm switches to the second stage and samples nodes within this quarter only, until the quarter is completely filled out with *minor* streets. A node's *growth* flag switches to *finished* if its valence becomes more than three or if the expansion fails multiple times.

For each selected node we have to choose the following parameters to expand it: direction ($straight$, $left$, or $right$), angle deviation ϕ and length l. If the node has valence 1, the direction is always straight. If the node has valence 2, we randomly pick left or right, and if the node has valence 3, there is only one remaining expansion direction (see figure 5). The parameters ϕ and l are chosen depending on the street pattern. Similar to [PM01], we implemented three different street patterns (organic, grid and radial) which can be controlled globally via maps *i.e.* by painting weights, or - especially for minor streets within a quarter - are determined stochastically according to the user-defined land use types. The specific street patterns are also defined by the user: (1) For the organic street pattern, ϕ is chosen from a truncated Gaussian distribution with mean 0 and user-defined standard deviation ϕ_σ. The length is computed similarly. (2) For the grid pattern, the angle deviation is always 0 degrees so that only two lengths ranges for the long and the perpendicular short streets have to be defined. (3) For the radial street pattern, there are two angle deviations. Either the next node lays on a circle around the center with the same radius as the current node, or the next node is on a ray pointing in the di-

rection to or from the center. If no pattern-specific center is defined, the nearest city center is taken.

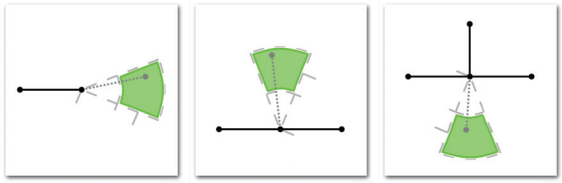

Figure 5: *General parameters for the street expansion. Nodes with valence 1 (left) are usually expanded in straight direction. For nodes with valence 2 (middle) we randomly select if a left or right expansion is applied. Otherwise the remaining direction is taken (right).*

If a height map $h(x,y)$ is given that maps every $node[i].pos$ to a height value z, we have to adapt a proposed street segment as soon as its slope is above a user defined *critical slope* s_c. In this case, the goal is to align the direction of the proposed street to either minimize or maximize its slope which is typical for streets in urban areas (*e.g.* San Francisco). Therefore, we use a simple steepest decent algorithm to align the proposed street segment (within angle deviation ϕ) into the direction of either the minimum or maximum slope of the terrain (whichever is the nearer).

Similar to the local constraints of [PM01], the proposed street segment is adapted to its local environment: (1) If an intersection is detected the street segment is shortened and the new node position is set to the intersection point. (2) To prevent nodes near to streets and short street segments in the future, we also enlarge the proposed segment by a factor 1.5 and test for a possible future intersection. If a future intersection is detected we slightly enlarge the street segment by setting the new node position to the future intersection point. (3) To prevent close-by intersections snapping can be used. A circle of user defined radius $snap_r$ (typically we set $snap_r = 25m$) is inspected for other street nodes. If there is at least one node within this area, the proposed node position snaps to the nearest node. (4) The length of street segment is shortened, if it intersects with obstacles such as water, forests, or external objects given as constraints to the simulation. If the length is below a *minimal street length* threshold s_{min}, the proposed street segment is rejected completely. (5) If the slope of the street exceeds a user defined *maximum slope* s_{max}, the direction is aligned until its slope is below the threshold (again via steepest descent and within angle deviation ϕ). If the alignment did not succeed, the proposed street is rejected. The five cases are illustrated in figure 6.

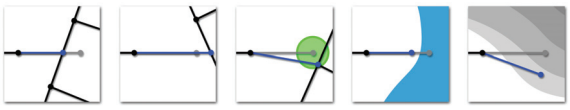

Figure 6: *Legality tests. Proposed streets (gray) are adapted (blue) to the local environment. From left to right: intersection, enlargement, snapping, avoidance and slope correction.*

When all legality tests are passed the new node and street segment are added to the existing graph data structure. If snapping or an intersection occurred, new quarters (major streets) or blocks (minor streets) may originate *i.e.* either a new quarter or block is generated, or an existing quarter or block is split into two. We can detect these cases by elementary algorithms from graph theory for detecting induced cycles (we used a half-edge data structure for the graph *i.e.* cycles can be easily found via traversal). Note that, although their location will not change anymore, the newly generated streets are just *planned*. A street will be only be built when enough traffic demand is computed for the street. The computation of the traffic is described in the following.

3.2. Traffic Simulation

The goal of the traffic simulation is to compute a traffic value $street[i].traffic$ for each street segment i. This value denotes the amount of people which are using a street segment per day (related to the average daily traffic volume [AAS04]), and is used in the computation of the street widths and the land use simulation. Traditional traffic demand models follow a four step procedure [McN00]. While modern systems, e.g. TRANSSIM [Tra06], include several refinements, existing traffic demand models are unfortunately not suitable for our purposes because: (1) The traffic demand models are complex and include many external variables and data tables. We want to simplify real traffic demand models by extracting the components that are responsible for generating interesting spatial distributions. (2) Existing traffic demand models use aggregated data and work with larger blocks while we need a traffic value for each individual street. (3) Existing models work only with static cities. We want to use the dynamic disequilibrium approach to incrementally compute the evolution of traffic over time in a fast way.

We propose a stochastic model that uses sampling to distribute a discrete number of trips in the street network. A trip k has a starting street segment $trip[k].start$ denoting the index of the start street segment, an end street segment $trip[k].end$, and a volume $trip[k].volume$ denoting the number of people transported on the trip. In the following we will explain our basic model in three parts: (1) The outline of a stochastic algorithm that selects new and existing street segments for trip generation. (2) The details on how to generate and compute trips from one $street[i]$. (3) How to maintain a matrix of estimated travel times $distance[i][j]$ between two street segments i and j for a fast incremental algorithm. Finally, we explain two extensions for increased realism.

The overall algorithm works by selecting a set of street segments for trip generation. For each selected street segment all trips previously generated for the street segment are

deleted and new trips are computed. For each of the deleted trips, the traffic values for all street segments that lie on the trip are decremented by $trip[k].volume$. For each of the new trips we compute all street segments that lie on the shortest path from $trip[k].start$ to $trip[k].end$ and store it with the trip. Then we increment the traffic value of all segments of the trip by its volume. In a simulation step we select the following street segments for trip generation: all newly generated street segments, all segments with a changed number of residents $street[k].residents$, and a specified percentage $traficresamplerate$ of street segments that see no change in the number of residents. The latter allows us to consider the dynamic traffic load and the growth of the system.

The number of trips starting from a street segment is proportional to the residents living on the land adjacent to a street ($street[i].residents = \sum lot[j].residents$). Please note that we not only compute residents for residential lots, but we also compute residents for lots of all other land uses depending on how many people will be there during a 24 hour period. For a street segment we compute n different one-person trips with $n = street[i].residents * tripsperresidents$. The destination of the trip is computed by sampling according to trip attractiveness. The attractiveness of a trip ending in street segment j depends on the distance (travel time) and the number of residents at the street, i.e $trip[k].attraction = street[j].residents * f(distance[i][j])$. The function f can be $1/x^2$ or an exponential function [McN00]. At the beginning of the simulation we compute trips with a volume of 1 person. At later stages we compute less discrete trips with a higher volume, so that the total number of simulated trips is approximately equal to a user specified constant. Figure 7 illustrates the trip generation and the resulting traffic.

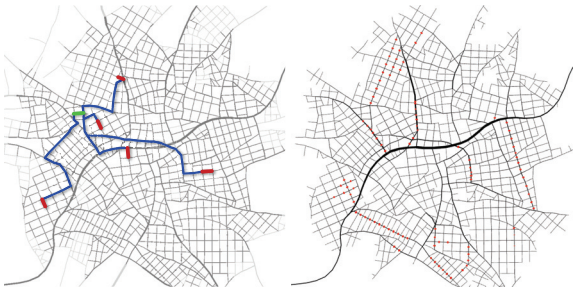

Figure 7: *We stochastically generate trips from each street to compute its traffic value. Left: an example of 5 trips starting at the green-colored street. Right: illustration of the traffic values per street after the complete simulation (red dots mark overloaded streets).*

A challenge of the simulation is that we compute the shortest path ($distance[i][j]$) between the starting and end point for each possible trip. The distance is affected by the length of street segments on the path, the type of street and its speed, and the angles between two street segments on the path. The angle is used to include results from space syntax [Hil98]; e.g. we rate a 90 degree turn as a straight 500m drive. The computation of $distance[i][j]$ for a fixed i and variable j is the shortest path problem and the computation of $distance[i][j]$ for variable i and j is the all pair shortest path problem [DE04]. The brute force approach requires computation time of $O(n^3)$ which is too slow. The main idea to accelerate the computation is to use incremental computation, and only updating the matrix when new streets are added to the street graph or when a street segment i exceeds its maximum volume $street[i].maxvolume$. The maximum volume is proportional to $street[i].width$. For this purpose we used a *dynamic all pair shortest path* implementation as described in [RR96], which is known to be very fast in practice ($O(n)$ in average and $O(n^{2.5})$ in the worst case [DE04]). The incremental implementation trades off computation speed for memory. Fortunately, a careful implementation can keep the memory overhead reasonable. In our case, we used n^2 bytes for the distances and $n^2/2$ bytes to save the paths (with n being the number of streets). For example, a street network consisting of 2000 streets needs approximately 6MB of memory.

We also included several extensions to the basic model. First, we included a trip generation model that considers land use values when selecting trip destinations. This requires to model probabilities on how likely a land use type j is going to be the destination of a trip from land use type i. This model can easily be incorporated using stochastic sampling. Second, if the city becomes too big we stop updating *all* shortest paths and instead compute only the shortest paths for the new trips with Dijkstra's algorithm. On our hardware, specific trip updating with Dijkstra's algorithm became faster than the dynamic all pair shortest path algorithm for cities with more than 10000 streets. Third, we simulated the delay in planning and actual construction. While all streets are generated as *planned* streets and are immediately included in the traffic simulation, we actually only build a street if the traffic value is above a user-defined threshold. Then the street is marked as *built* and a temporary street width can be computed according to the current $street[i].traffic$ value. The final street width is computed after a user specified time span. At this time, the maximum capacity of the street is also set. In Figure 4 we depict new streets in light gray to illustrate the delay in planning and construction.

4. Land Use Simulation

This section explains how to assign land use values to every lot in an urban layout. The land use values are needed for the traffic simulation (the number of residents per lot depends on the land use value) and the building construction (to determine which type of building should be built). The simulation tries to optimize a land use value function luv, that measures the value of the urban configuration. In general, land use simulation is very challenging because urban systems are inherently complex and the simulation considers time spans of several years. This leads to many uncertainties

and it is therefore still not clear if complex simulations considering many variables can achieve a better accuracy than very simple ones. In our simulation we first wanted to make sure that we can work with detailed geometry and that the simulation is much faster than existing solutions. In that context we want to highlight two ideas that made our system work well. Our simulation does not use an agent based approach that is prevalent in many social sciences to a simulation focused on polygonal regions picking their optimal land use. According to our initial experiments, this allows to speed up the computation by one to two orders of magnitude. Additionally, we control the proportions of land use via global settings. This allows to keep the simulation stable.

An important design choice of our system is to implement a *generic* land use simulation. We therefore allow a designer to define a set of possible *land use types* T. A land use type is defined as a convex combination of individual land use valuation functions. In most of our example simulations, we experimented with a set of seven land uses, that are typical for most cities (described in the additional material). Please note that new land use types can be added easily by a user.

4.1. Simulation Algorithm

The total system value luv is defined by the global land use goals of the simulation and the local land use goals:

$$luv = \lambda_{global} \cdot luv_{global} + \lambda_{local} \frac{\sum_{\forall i} lot[i].area \cdot lot[i].luv}{\sum_{\forall i} lot[i].area} \quad (1)$$

The global goals capture how well the configuration matches the optimal (user specified) percentage values for land use distribution over the whole city. Each lot has an assigned land use type $lot[i].lut$ and a land use value $lot[i].luv$ that describes how well the lot is suited for its land use. Hence, the local goals are evaluated as the sum of these land use values proportional to the lot areas. The weights are set to $\lambda_{global} = 0.5$ and $\lambda_{local} = 0.5$ in our results.

To initialize a newly created lot $lot[i]$ we set $lot[i].lut$ to the land use type which produces the highest system value luv. We can also perform the same computation for newly created quarters and blocks. While quarters and blocks are not included in the land use simulation itself, the computed land use value is used to determine the street pattern of a new quarter or a subdivision pattern of a new block. For example, a block in an industrial zone is divided into much larger lots than a block in a residential zone (see next section).

The algorithm to *update* the land use values of existing lots is inspired by stochastic sampling algorithms, such as Metropolis-Hastings [CG95] and Simulated Annealing [KGV83]. There are several goals that we considered when designing the update algorithm. First, algorithm speed is very important and we therefore opted against an agent-based simulation algorithm. Second, we want to adopt the dynamic disequilibrium approach. We do not want to find the optimal land use configuration, but we want to simulate the system striving towards better configurations. Third, we want to be able to configure the algorithm to consider time and different planning strategies. The resulting update algorithm is described in table 3.

```
repeat (m) {
   i = rand(0, |lots|);
   lut = lot[i].lut;
   lot[i].lut = rand(0, |T|);
   value = compSystemValue(lots);
   delta = lastValue - value;

   if (accept(delta)) lastValue = value;
   else lot[i].lut = lut;
}
```

Table 3: *The update algorithm in pseudo code. The acceptance function decides whether to keep a new land use type based on the difference* delta *between the new and the last system value.*

The algorithm is repeated after a time span s (typically equal to the simulation time step). However, setting the time span s to a larger period can help switching from a market driven dynamically changing city to a centrally planned city where land use is reevaluated every, say 10 years. The variable m is typically set to a percentage of the total number of lots in the city. For example, we set m to 50% of the existing lots in the city for a time step of 6 months. The acceptance function is modeled in such a way that changes are always accepted if $\Delta_{luv} > tresh_{\Delta}$ rejected with a probability proportional to $tresh_{\Delta} - \Delta_{luv}$ otherwise. If we do not accept any negative changes we would only perform local optimization. Please note that in contrast to simulated annealing we do not try to force convergence by lowering the probability of accepting negative changes over time. This is a consequence of the dynamic disequilibrium approach.

4.2. Global Land Use Goals

The simulation can be controlled by user defined global land use goals that specify the desired land use percentages of a city. The land use percentages used in our simulations are listed in the additional material. The land use criterion luv_{global} should be small if the lot's land use type (the current setting) makes the difference between the actual and the desired land use percentage larger, and it should be large if the lot's land use makes the difference smaller. The larger the difference between the actual and the desired land use percentage (positive or negative), the more articulate (very large or very small) the land use criterion value luv_{global} should be:

$$luv_{global} = -\sum_{t \in T} \left(\frac{percent_t - goal_t}{scale} \right)^2 \quad (2)$$

$$percent_t = \frac{\sum_{\forall i} (lot[i].lut == t) \cdot lot[i].area}{\sum_{\forall i} lot[i].area} \quad (3)$$

where $goal_t$ is the desired percentage of land use type t and $scale(\approx 0.05)$ is a parameter which can be used to control the allowed land use percentage deviation. The squared distance assures that the land use percentages stay close to the specified goals.

4.3. Local Land Use Goals

Each land use type is associated with a (user specified) combination of *valuation functions*. Accordingly, the land use value of $lot[i]$ is computed as sum over all valuation functions associated with the land use type $lot[i].lut$:

$$lot[i].luv = \sum_j \lambda_j \cdot valuation_j(lot[i]) \quad (4)$$

We found it practical to find a good trade off between generality and user-friendly modeling. Therefore, we opted to model the overall land use valuation function as a convex combination of individual valuation functions $valuation_j()$ that can assume values between 0 and 1. The restriction of the range brings some limitation, but it greatly simplifies the modeling, because the user can control the relative importance by setting the convex weights and can have some intuitive idea of the outcome of the system. As valuation functions, we decided to use the following mapping functions: (1) step, (2) linear up-ramp, (3) linear down-ramp, (4) gain up-ramp, and (5) gain down-ramp [PH89]. The ramp functions are depicted in figure 8. Thus, to define a valuation function, the user specifies which of the five functions to use, selects the lot attribute to apply it, and sets the two boundary values p_{min} and p_{max}. Hence, by using these functions, we can transform the individual lot attributes to a value between 0 and 1. The needed lot attributes $cluster$, $influence$, $centerdist$, $forestdist$, $waterdist$, $traffic$, $slope$, and $elevation$ are explained in the following.

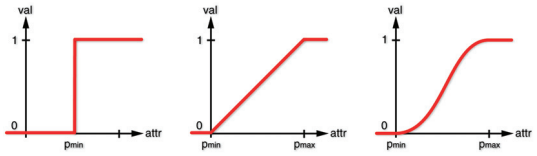

Figure 8: *A land use valuation function is specified by the mapping functions , the lot attribute to apply it, and 1 or 2 boundary values,* i.e. *the lot attribute is transformed to values between* 0 *and* 1. *The used functions are step (left), linear-up (middle), linear-down, gain-up (right) and gain-down.*

Clustering: For several land use types, such as residential land use, it is an advantage to be close to lots of the same land use. While some clustering occurs because close by regions attract the same land use, clustering explicitly encodes the advantage of clustered land use plans. Clustering $lot[i].cluster$ can be computed as the percentage of lots in the $n(\approx 10)$ nearest lots around the current lot that have the same land use.

Neighborhood land use: The influence of another land use type $lot[i].influence_t$, with $t \in T$ can be computed as follows. For every land use t it is the weighted percentage of lots which have the land use t inside a distance (also called neighborhood) of radius $r(\approx 256$ meters) around $lot[i]$. Lots which are close to the current lot are linearly weighted more than lots which are far away. The input can be computed using the following formula:

$$lot[i].influence_t = \frac{\sum_{v \in A} (lot[v].lut == t) \cdot dist(i,v)}{\sum_{v \in A} dist(i,v)} \quad (5)$$

$$dist(i,v) = \frac{||lot[i].ctr - lot[v].ctr|| \cdot lot[v].area}{r} \quad (6)$$

where A is the set of lot indices which are inside the distance with radius r around the current lot.

Distance to centers, forest, and water: The Euclidean distances to the closest city center ($lot[i].centerdist$), the nearest forest ($lot[i].forestdist$), and to the nearest water ($lot[i].waterdist$) can be computed. To have a similar range of values during the simulation we normalize the distance by dividing by the longest distance in the urban configuration.

Traffic: The traffic can have a positive or a negative influence on the suitability of a certain land use. We set $lot[i].traffic = \sum street[j].traffic$ with j being the adjacent streets of $lot[i]$.

Lot slope: A steep slope of a lot can hinder construction. The lot slope is evaluated at n_{slope} random locations within a lot and averaged. Given a height field $h(x,y)$, the lot slope is computed as the magnitude of the gradient, *i.e.*

$$lot[i].slope = \frac{\sum_{j=1}^{n_{slope}} ||\nabla h(x_j, y_j)||}{n_{slope}} \quad (7)$$

Elevation: Living on a hill slightly elevated above most parts of the city is often considered more prestigious. We can use the height map $h(x,y)$ to compute an average elevation value by sampling at $n_{elevation}$ random locations in the lot (normalized with the maximum elevation h_{max}):

$$lot[i].elevation = \frac{\sum_{j=1}^{n_{elevation}} h(x_j, y_j)}{n_{elevation} \cdot h_{max}} \quad (8)$$

5. Building Lot Construction

In this section we explain the subdivision of lots and the generation of geometric configurations for building construction. The latter includes the simulation of simplified zoning regulations and lot price. While these simulations are essential to produce visual results, they have not been considered in previous simulations.

5.1. Lot Subdivision

To subdivide blocks into lots, we use a recursive algorithm similar to [PM01]. Our main observation is that lots are dom-

Figure 9: *A land use type with associated valuation functions is assigned to each lot. The figure shows the resulting land use values (white = 1 and black = 0) of the example from figure 4.*

inantly rectangular and that lot boundaries are typically orthogonal to the street. The input of the algorithm is a polygon defined by the street boundaries. The land use simulation described in the previous section provides us with an estimated dominant land use type $t = block[i].lut$. The algorithm recursively splits the polygon and all resulting sub-polygons, until the area of all sub-polygons is below a user-defined threshold $maxarea_t$, depending on the land use type. If a (sub-)polygon exists that is bigger than $maxarea_t$ the algorithm performs the following steps: (1) simplify the polygon by merging all collinear edges, (2) the longest side on a street is selected, (3) a split candidate is stochastically defined approximately perpendicular in the middle region of the selected side, (4) if the width-length ratio of the two resulting sub-polygons is below $minratio_t$, the polygon is split along the split candidate, (5) if the ratio constraint cannot be fulfilled even after several iterations on step (3), we also test sides which are not on streets. See figure 10 for an illustration of the algorithm.

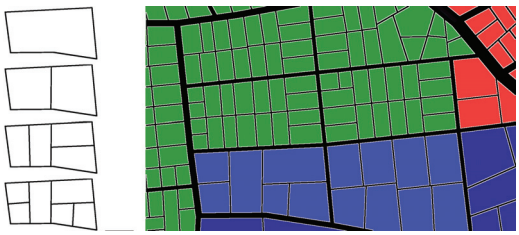

Figure 10: *Left: The initial block (top) is recursive split (approximately orthogonal to the longest side) until all sub-polygons are below a land use dependent area threshold. Right: Resulting lots. Note the different sizes and ratios of the lots in residential (green) and industrial zones (blue).*

5.2. Economy Model

Equitable construction and the maximization of profits is a dominant factor in architecture. According to the income approach [BM89], the buildings have to be of proportional value to the property in order to be reasonable investments. Therefore, we can compute the monetary value of a lot to be able to determine its building envelope. First, the price (in [$]) of a lot is computed:

$$lot[i].price = lot[i].area \cdot avgprice[t] \cdot \frac{lot[i].luv}{\sum_{\forall j} lot[j].luv/n} \tag{9}$$

where n is the current number of lots and $avgprice[t]$ is a time-dependent parameter specifying the average price per m^2 of the city (bigger cities are typically more expensive). Now we can compute the required floor space, *i.e.* the effective area in $[m^2]$ where residents live and work, that would make the building profitable:

$$lot[i].floorspace = lot[i].price \cdot margin_{lot[i].lut} \tag{10}$$

where $margin_t$ is a truncated Gaussian distribution defined per land use type t. The bigger the margin, the more floorspace is required. Analogous, we can compute the estimated number of residents $lot[i].residents$ on a lot with the $interestrate$. This economy model is easy to control and understand.

5.3. Building Envelope Generation

Detailed construction regulations typically include setbacks from the street, the back, and the sides of a building, a construction zone within the building, and height regulations. These construction regulations result in a building envelope. An example is given in figure 11. Each land use type t has their own range of setback values that can be determined stochastically for each block. Additionally, we use variations for how far a building can move back from the front setback. After determining the floor plan we can reuse results from the previous section to compute the height and number of floors.

$$lot[i].nFloors = \frac{lot[i].floorspace}{lot[i].envelope.area} \tag{11}$$

After the building envelope has been determined, the actual building geometry can be created. We use a library of configurable buildings for every land use using parametric shape grammars [MWH*06]. We orient the building according to the street with the highest traffic value and use the shape grammar to create models that respect the envelope and the land use type.

5.4. Building Substitution

Existing buildings may be substituted, depending on the building age $lot[i].age$ and by measuring the value discrepancy Δ_{price} of the actual $lot[i].floorspace/margin_{lot[i].lut}$

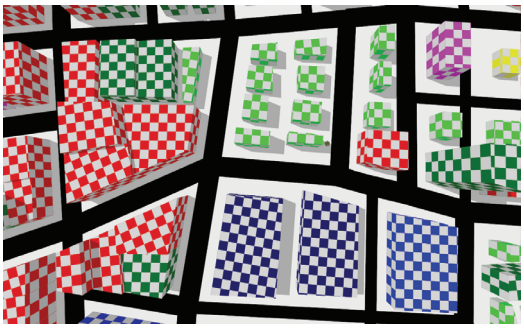

Figure 11: *Based on the land use type and price of a lot we can compute the final building envelope. Here, the colors indicate the land use types and one square corresponds to 5x5 meters.*

of the existing building and the price of a potential new building $lot[i].price$. We model a substitution probability $p_{sub}(lot[i]) = f_1(lot[i].age) + f_2(\Delta_{price})$ using user selected functions f_1 and f_2 (see previous section) and compute actual replacements via sampling.

6. Results

The goal of this section is to evaluate 1) simulation speed, 2) realism by comparing to a real environment, and 3) modeling expressiveness by demonstrating the simulation of time-dependent phenomena.

Speed: We recorded the speed as a function of the number of streets for the simulation shown in figure 4 on a 2Ghz PC. The final city has 3005 streets and the total simulation time is 58.3 seconds (see figure 12).

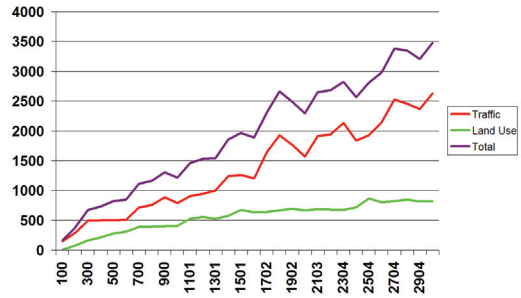

Figure 12: *For each simulation step of a growing city we show the computation speed. On the x-axis we list the number of streets in the city at a given simulation step and on the y-axis the simulation time in milliseconds for traffic simulation, land use simulation, and total simulation time.*

Realism: We conducted this test to evaluate the similarity of our land use patterns to real data from Las Vegas. This test should demonstrate that our simulation framework is flexible and can be configured to produce realistic growth patterns. We chose Las Vegas because it is one of the fastest growing cities in recent times and the growth is well documented. We started with an old map from year 1950 of Las Vegas and continued to simulate growth for 25 years and compared it to the real land use data (see figure 13). The result has an extent of 20 square kilometers. Our simulation was designed in about two hours by experimentally tuning parameters. During the simulation, we only made two interactive changes: After 10 years of development, we moved the city center in South direction and added two growth centers to invoke the elliptic East-West development. We implemented a version of Wei et al.'s texture similarity metric [WHZ*08] for a quantitative comparison of the simulation image IS to the ground truth image GT. Both images use integer values to encode different land uses. For each pixel in IS we construct an $n \times n$ pixel neighborhood and find the most similar neighborhood in GT. To compare two neighborhoods we compute the percentage of identical values:

$$Similarity(X,Z) = \sum_{p \in X} (sim(x_p, z_p))/NumPixelsInX \quad (12)$$

$$sim(x_p, z_p) = \frac{1}{n*n} \sum_{i=1}^{n*n} (x_p(i) == z_p(i)) \quad (13)$$

where X and Y are the images that should be compared, p is a pixel in X, x_p a pixel neighborhood around p, z_p is the best matching neighborhood for x_p in Y, and $==$ compares to values and returns 1 if they are identical (0 otherwise). Intuitively Sim measures the average similarity of two images. In table 4 we show the results for computing $Similariy(IS, GT)$ and $Similarity(GT, IS)$.

neighborhood size	4	8	12	16	25
Similarity(IS, GT)	0.99	0.93	0.87	0.82	0.74
Similarity(GT, IS)	0.98	0.92	0.86	0.8	0.72

Table 4: *The average similarity of the simulated city compared with the ground truth data. The best possible value is* 1.0 *which indicated* 100% *similarity. We used textures of size* 238×178 *for the comparison.*

Time-dependent Phenomena: The most important time-dependent phenomenon is urban growth. We show an example of a fictional European city in figure 1 and show the screen shots from our interactive simulation environment rendered in real-time. Figure 15 completes this series and shows a final close-up view of the same city. We simulated Las Vegas (figure 13) as an example of urban growth in a Western American city. In figure 14 we show two temporal phenomena on a small scale. On the left we show how lower density residential buildings are replaced by higher density residential ones. On the right we show a transition of a city based on sustainable development. A third transition is shown in the additional material from a residential area to a commercial one.

Editing: The fast simulation speed allows us to integrate user interaction with simulation. For example, our framework allows the user to change and insert streets and to

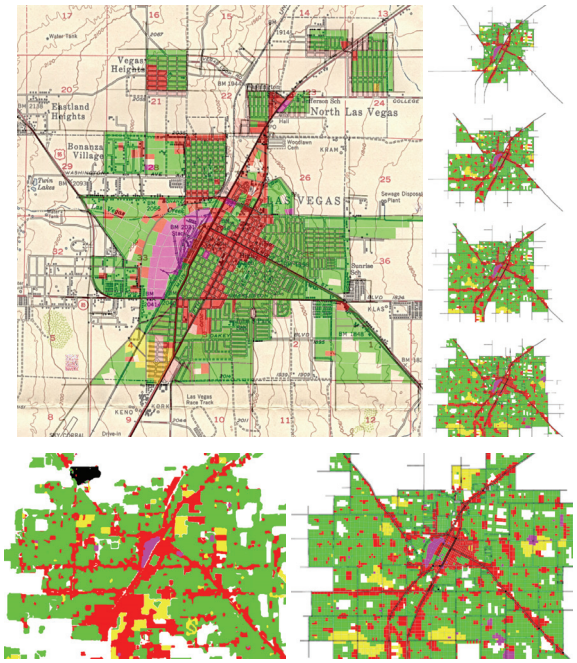

Figure 13: *Our system can grow real cities. Top left shows a map of Las Vegas from 1950 with our overlaid initial simulation data (reality-based). On the right side the steps after 5, 10, 15 and 20 years. The bottom row shows on the left the real land use map of Las Vegas after 25 years in 1975 and on the right our simulation in 1975. Total simulation time: 291 seconds.*

paint land-use values during simulation. This feature is highlighted in the video and the additional materials (figure 16).

7. Discussion

Comparison to urban planning simulations: In the current state we do not believe a formal comparison to existing urban simulation software, such as UrbanSim is useful, because the output and goals are too different. On the one hand, to the best of our knowledge we propose the first geometric urban simulation system and our main output are two-dimensional maps and three-dimensional cities that change over time. Other simulation systems cannot produce this output. On the other hand, we designed a simulation system that focuses on recreating visually realistic urban patterns of land use and transportation. We did not attempt to tune the simulation to match existing urban data nor do we simulate variables that have little visual importance (e.g. energy consumption). We plan to address this in future work.

Comparison to computer graphics simulations: While there is some initial work in urban simulation, we believe that the existing attempts are not developed far enough to be competitive. For example, the grid structure of the simulation could not be overcome by competing approaches [HMFN04, LWW*07, Wat07]. A key aspect for high quality visual output are meaningful street orientations and lot geometry that is not dominated by a regular grid of the simulation.

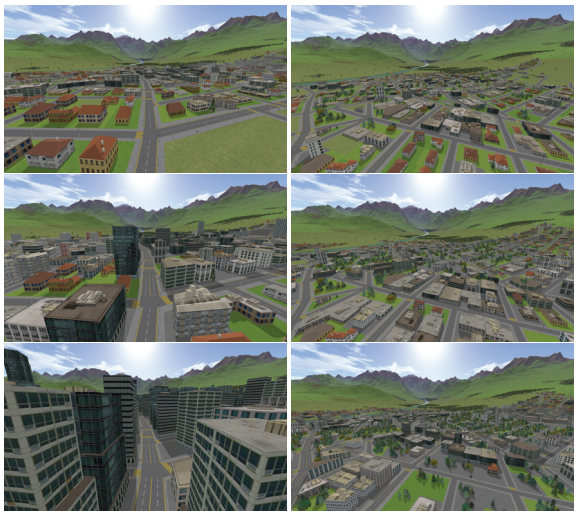

Figure 14: *Two time series are shown in the columns. The left series shows the transition from low density to high density in the city center. The right column shows a transition of a city based on sustainable development with sufficient green areas.*

Limitations: The main difficulty of such a large simulation system is to find a useful set of parameters to produce simulations of high visual quality. In practice, the most important inputs to the simulation are the grow centers and the two growth rates ($streetgrowth[t]$ and $avgprice[t]$). These parameters can be interactively controlled during the simulation. In our experience a reasonable setup can be achieved in about one hours, and the fast simulation times help giving quick visual feedback for tuning parameters. We expect that a new user needs to study the simulation and its parameters for about one full day to produce high quality results.

8. Conclusion

We presented a geometric urban simulation system. In contrast to previous work we included the geometric component of streets and lots in the simulation rather than working with data on a regular grid. We included geometric rules for the generation of streets, lots, and land use on a small scale. As a result we demonstrated that the visual output of the simulation can be drastically improved compared to previous work opening the door to new applications in computer graphics, visualization, virtual worlds, and education.

We would like to thank Urban Weber for his continuous support, and Andreas Ulmer (Procedural Inc.) for helping with the production of the three-dimensional urban models. Peter Wonka was supported by NSF and NGA.

References

[AAS04] AASHTO: *A Policy on Geometric Design of Highways and Streets, 5th edition*. American Association of Highway and Transportation Officials, 2004.

[AIS77] ALEXANDER C., ISHIKAWA S., SILVERSTEIN M.: *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977.

[AVB08] ALIAGA D., VANEGAS C., BENES B.: Interactive example-based urban layout synthesis.

[Ben97] BENATI S.: A cellular automaton for the simulation of competitive location. *Environment and Planning B: Planning and Design 24*, 2 (1997), 205–218.

[BGKR06] BERKE P. R., GODSCHALK D., KAISER E., RODRIGUEZ D.: *Urban Land Use Planning*. University of Illinois Press, 2006.

[BM89] BAUM A., MACKMIN D.: *The Income Approach to Property Valuation (Third Edition)*. Routledge, London, Routledge, London., 1989.

[CBAN07] COELHO A., BESSA M., AUGUSTO SOUSA A., NUNES FERREIRA F.: Expeditious modeling of virtual urban environments with geospatial l-systems. *Computer Graphics Forum 26*, 4 (2007), 769–782.

[CEW*08] CHEN G., ESCH G., WONKA P., MÜLLER P., ZHANG E.: Interactive procedural street modeling. *ACM Trans. Graph. 27*, 3 (2008), 1–10.

[CG95] CHIB S., GREENBERG E.: Understanding the metropolis hastings algorithm. *American Statistician 49* (1995), 327–335.

[Cou97] COUCLELIS H.: From cellular automata to urban models: new principles for model development and implementation. *Environment and Planning B: Planning and Design 24*, 2 (1997), 165–174.

[DE04] DEMETRESCU C., EMILIOZZI S.: Experimental analysis of dynamic all pairs shortest path algorithms. In *SODA* (2004), pp. 369–378.

[Gol71] GOLDNER W.: The lowry model heritage. *Journal of the American Institute of Planners 37*, 2 (1971), 100–110.

[GRC*04] GINGROZ R., ROBINSON R., CARTER D. K., JR. B. J. L., OSTERGAARD P.: *The Architectural Pattern Book: A Tool for Building Great Neighborhoods*. W. W. Norton & Company, 2004.

[Hil96] HILLIER B.: Cities as movement economies. In *Urban Design International* (1996), pp. 41–60.

[Hil98] HILLIER B.: The common language of space: A way of looking at the social, economic and environmental functioning of cities on a common basis, 1998.

[HMFN04] HONDA M., MIZUNO K., FUKUI Y., NISHIHARA S.: Generating autonomous time-varying virtual cities. In *CW '04: Proceedings of the 2004 International Conference on Cyberworlds (CW'04)* (2004), pp. 45–52.

[KGV83] KIRKPATRICK S., GELATT C. D., VECCHI M. P.: Optimization by simulated annealing. *Science 220* (1983), 671–680.

[Klo99] KLOSTERMAN R. E.: The what if? collaborative planning support system. *Environment and Planning B: Planning and Design 26*, 3 (May 1999), 393–408.

[Leu03] LEUNG H.-L.: *Land Use Planning Made Plain*. UTP, 2003.

[LWW*07] LECHNER T., WATSON B., WILENSKY U., TISUE S., FELSEN M., MODDRELL A.: Procedural modeling of urban land use.

[LZ98a] LANDIS J., ZHANG M.: The second generation of the california urban futures model. part 1: Model logic and theory. *Environment and Planning B: Planning and Design 25*, 5 (1998), 657–666.

[LZ98b] LANDIS J., ZHANG M.: The second generation of the california urban futures model. part 2: Specification and calibration results of the land-use change submodel. *Environment and Planning B: Planning and Design 25*, 6 (1998), 795–824.

[Mcf74] MCFADDEN D.: Conditional logit analysis of qualitative choice behavior. *Frontiers in Econometrics* (1974), 105–142.

[McN00] MCNALLY M. G.: *The Four Step Model*. Tech. Rep. UCI-ITS-AS-WP-00-51, Center for Activity Systems Analysis, 2000.

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural Modeling of Buildings. In *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics* (2006).

[PH89] PERLIN K., HOFFERT E. M.: Hypertexture. In *Computer Graphics (SIGGRAPH '89 Proceedings)* (1989), pp. 253–262.

[PL91] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001* (2001), Fiume E., (Ed.), ACM Press, pp. 301–308.

[Pun99] PUNTER J.: *Design Guidelines in American Cities*. Liverpool University Press, 1999.

[RR96] RAMALINGAM, REPS: An incremental algorithm for a generalization of the shortest-path problem. *ALGORITHMS: Journal of Algorithms 21* (1996).

[Sti75] STINY G.: *Pictorial and Formal Aspects of Shape and Shape Grammars*. Birkhauser Verlag, Basel, 1975.

[Tra06] TRANSIMS: Transims tutorial available on transims-opensource.org, 2006.

[VABW08] VANEGAS C., ALIAGA D., BENES B., WADDELL P.: Visualization of simulated urban spaces: Inferring parameterized generation of streets, parcels, and aerial imagery.

[Wad02] WADDELL P.: Urbansim: Modeling urban development for land use, transportation and environmental planning. In *Journal of the American Planning Association* (2002), vol. 68, pp. 297–314.

[Wat07] WATSON B.: Real and virtual urban design. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (2007), pp. 167–228.

[WBN*03] WADDELL P., BORNING A., NOTH M., FREIER N., BECKE M., ULFARSSON G.: Microsimulation of urban development and location choices: Design and implementation of urbansim. *Networks and Spatial Economics 3*, 5 (2003), 43–67.

[WHZ*08] WEI L.-Y., HAN J., ZHOU K., BAO H., GUO B., SHUM H.-Y.: Inverse texture synthesis. *ACM Trans. Graph. 27*, 3 (2008), 1–9.

[WU04] WADDELL P., ULFARSSON G. F.: Introduction to urban simulation: design and development of operational models. *Handbook in Transport 5* (2004), 203–236.

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Transactions on Graphics 22*, 3 (2003), 669–677.