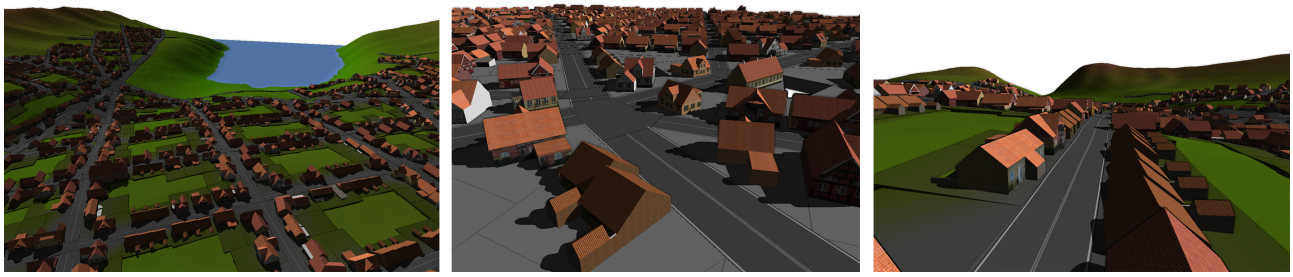
A Constraint Based System to Populate Procedurally Modeled Cities with Buildings

Johannes Scharl*
Supervised by: Daniel Scherzer†

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Austria



Abstract

Creating large-scale virtual environments for interactive applications such as computer games poses a demanding challenge for computer graphics. We present a system that procedurally creates urban environments including street networks, street geometry and building parcels. Our main contribution is a constraint based system that chooses the "best fitting" building for every parcel from a set of existing buildings. Building properties such as the footprint, area, and faces that should have street access are used to select the most suitable building.

Furthermore we introduce a robust technique to create 3D street geometry for streets that adapt to different terrain heights and describe a method to create a more realistic city shape and more detailed outer regions.

Keywords: Procedural Modeling, Urban Environments, Computer Games

1 Introduction

The usual way to create urban environments for computer games or movies is to use commercial modeling packages like Autodesk Maya. This is a very time consuming, tedious and expensive task and gets less and less suitable for modern applications that demand even larger and more detailed environments. A promising approach that has emerged in recent years is to create content procedurally.

Urban environments are mainly defined by their street network. Such a network forms the back bone of a city and determines its layout. Therefore it is the first thing that has to be generated when modeling a city.

Recent state-of-the-art techniques [10, 15] rely on extended L-systems to create such networks. Usually a top-down approach is used, meaning that major roads are created first, because they define the main routes and districts in the city. Regions surrounded with major roads are then filled by minor roads, creating the finer structures of districts and neighborhoods. When using this approach, areas surrounded by major roads are usually located at the city center. This often leads to sparse regions at the outskirts of the city, where no minor roads can be created. We propose an approach to generate cities where minor roads are also generated in the outer regions. This is illustrated in figure 1.

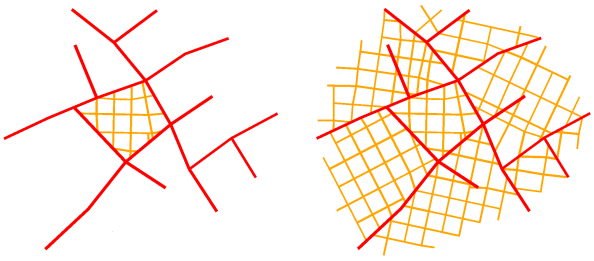


Figure 1: Left: In previous methods, no detailed neighborhoods are created in the outskirts of the city. Right: Outer regions created with our method.

After the street network is generated, street geometry is created and the blocks in between are subdivided into building parcels. Tessellating the street geometry is straightforward in a case where all streets are in a single plane, but gets complicated if streets adapt to three dimen-

*johannes@cg.tuwien.ac.at
†scherzer@cg.tuwien.ac.at

sional terrain. In this case, junctions have to be kept planar. This is usually handled by forcing the junction geometry to be parallel to the ground plane, resulting in unnatural steps in steeper roads. We introduce a way to adjust these junctions to adapt them to the underlying terrain.

Buildings are an essential part of every urban environment. Usually each model is hand-crafted in a modeling software and placed at its destination by a designer or artist. This approach is not feasible for larger urban environments. Typically only certain regions of the city are crucial for a computer game, while other regions do not contain a lot of individual detail. A method that fills these regions automatically greatly reduces the time needed to add building models to a road network. As our main contribution, we propose a method to select a building from a set of existing models that fits best for a certain building parcel and place it there.

2 Related Work

Our work is based on previous procedural modeling methods that employ *L-systems*. L-systems where originally developed as a formalism for plant modeling [12].

An extension to L-systems that allowed plants to communicate bidirectionally with their environment, called *Open L-systems*, was introduced later by Měch et. al. [9]. This method was developed further by Parish et. al. [10] to create city street maps using a set of production rules.

Weber et. al. [15] extended this method to simulate a three-dimensional urban model over time. They define a city hierarchy that guides the creation of such networks. Because we use this hierarchy definition, we will discuss it in greater detail in Section 3.1.1.

The CityEngine [11] is a commercial software package that is capable of procedurally generating complete cities. Input is provided in the form of many controllable parameters and various image maps: height maps can be used to model terrain, obstacle maps denote regions where no streets should be created and population density maps control the type and density of the streets and buildings in certain regions. The system is capable of creating large street networks, building parcels and even buildings. Streets are generated using L-systems, while buildings are created with a shape grammar technique.

Other methods to create street layouts and networks include interactive editing of a tensor field [2], a mixture of interactive and procedural techniques where main streets have to be created manually, while minor roads are generated automatically [5], and image based approaches that rely on aerial images to reconstruct a road network [1]

Recently, procedural generation of buildings and facades has been researched heavily, including the generation of facades using shape grammars [8, 16] and interactive editing of shape grammar rules [7].

An excellent review of various urban modeling techniques can be found in a recent survey paper by Vanegas et. al. [14].

Gebhart [3] describes a system that helps artists to create 3D street networks. Streets are "drawn" by an artist or designer, and detailed geometry is created semi-automatically by setting parameters such as the number of lanes, the radius of a curve, etc. Zimmermann [17] presented a technique to construct a fully polygonal 3D street representation out of centerlines. Although both methods produce visually impressive results, both fail to address the problem of maintaining a stable and realistic tessellation for 3D streets that adjust to terrain levels of different heights.

The problem of finding a model that fits into a certain environment has not been investigated very extensively. Kjølaas [6] presented a system that automatically places furniture into a given floor plan by selecting a template from a given set of default templates for common rooms and adapting it to the given room.

In the field of automated building placement a lot of work has been done in the direction of recognition and reconstruction of buildings from aerial images [4, 13]. This is usually done in the process of reconstructing existing cities, e.g. for mapping applications, but not for artist-created urban environments.

3 Our System

In this section we present our techniques to create street networks for urban environments, as well as a method to tessellate the street geometry in a simple and stable way. Additionally, we propose a technique to match the "most suitable" building to a parcel from a set of previously modeled buildings.

3.1 The Street Network

Our algorithm to create street networks is based on the work of Parish et. al [10] and Weber et. al. [15]. Streets are created using a system similar to extended L-Systems, although we chose not to implement a string rewriting system, but to apply the production rules directly to the street objects to avoid slow string operations, as proposed in [15]. We will first explain our city hierarchy definition in Section 3.1.1 and describe a set of important control parameters that are used for creating streets in Section 3.1.2. Finally, we will describe the algorithm based on the described hierarchy in Section 3.1.3.

3.1.1 City Hierarchy

We use a city hierarchy definition similar to the one introduced by Weber et. al. [15]:

A *street network* is a planar graph (V,E) with nodes V and edges E. A street consists of one or more edges $e \in E$, the *street segments*. Each street segment e connects two nodes $n(e)_1, n(e)_2 \in V^2$. Streets can be *major* or *minor*

streets and have different *widths*. A facet in the planar graph (V_{major}, E_{major}) is referred to as *quarter*, that is an area surrounded by major roads. A facet in (V, E) (an area surrounded by any street) is called a *block*. Each block can be subdivided into *building parcels*. This hierarchy is illustrated in Figure 2.

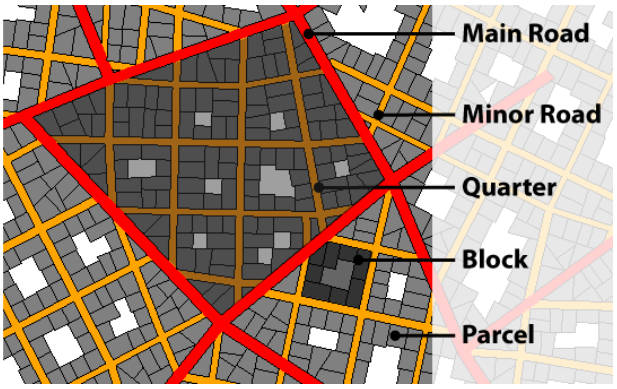


Figure 2: City Hierarchy. *Main roads* are displayed red, *minor roads* are orange. *Quarters* are areas surrounded by major roads. *Blocks* are surrounded by any road and divided into *parcels*.

To create a city obeying this hierarchy, major streets are created first, and spaces in between are filled with minor roads afterwards. This results in a planar street network and buildings blocks in between. A block consists, according to our hierarchy definition, of multiple building parcels, each defining the space a single building can take up. These parcels can be generated by repeatedly subdividing a building block.

3.1.2 Control Parameters

The creation of streets can be controlled with a lot of different parameters:

- Cities usually have different layouts. Streets in New York City are strictly rectangular, whereas Paris follows a loosely circular pattern. Cities with no superimposed pattern grow organically. In our system, major and minor roads can follow *different patterns*.
- Street *length*, *width* and *angles between adjacent street segments* can be controlled.
- To avoid street ends near existing junctions, a distance `snappingDistance` can be defined. If the distance between a street end and a junction is smaller than `snappingDistance`, the street end is snapped to this junction.
- A *height map* may be specified to create a terrain (see Figure 3). The streets will then adjust to this terrain. If the slope of the street is larger than a user defined threshold `maxSlope`, the street is rotated until it is plain enough, or removed if that is not possible.
- Urban environments may contain areas where no streets should be created, such as parks or water. Such areas can be denoted in an *obstacle map* (see Figure 3). This map is sampled regularly and streets will avoid any obstacles.
- Real cities grow after demand: Major roads connect centers of high population densities, while minor roads provide access to the major roads in populated areas. A *population density map* (Figure 3) can be set to control the development of major and minor streets.
- The average size of a building parcel can be controlled to adjust the parcels to the desired building size.

All of these parameters can be set using the user interface of our application.

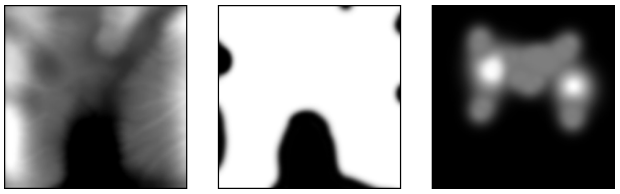


Figure 3: Input maps for a bay area environment. From left to right: (1) Height map, (2) Obstacle map, (3) Population density map.

3.1.3 Creating the Street Network

As explained above, the algorithm is divided in two stages: (1) creating major streets and (2) identifying quarters surrounded by major streets and filling them with minor streets. Quarters and blocks can be identified easily using a planar graph face traversal algorithm.

As discussed in [10] we use a 3-level hierarchy to evaluate parameters for a new street segment: First, an *ideal successor* is created. This is a new street segment without any parameters assigned. For this new street segment, the *global goals function* is evaluated. The location and orientation of the street is set according to the superimposed street pattern and the local population density. After the initial parameters have been evaluated, the street segment is adapted to its local environment by calling the *local constraints function*: This function changes the location and orientation of the street according to the parameters in Section 3.1.2: The new street segment is snapped to existing junctions, the street is adjusted to the local terrain slope or changed to avoid any obstacles such as parks. The procedure is the same for major and minor streets, although different parameter sets can be used.

One of the main problems of this approach is that minor streets are only created inside quarters that are surrounded by major streets. This leads to unrealistic results at the

boundaries of the street network, where no minor streets are created.

We have solved this problem by calculating the convex hull around the whole street network and using this hull to create additional quarters at the city boundaries.

If the points of the convex hull are connected by a straight line, outer regions of the city look ”cut off” and unnatural. To account for that, we bulge the hull by random amounts to create a more realistic city shape.

After all streets have been created, we use the same algorithm we used before for identifying quarters to now find all blocks surrounded by streets (major and minor). To create parcels of the size the user specified, each block is recursively subdivided into building parcels using a method similar as the one described in [15]: (1) Select the longest side of the polygon and create a perpendicular split line at its center. (2) Split the polygon along this line. (3) Repeat this for each new polygon if its area is larger than a predefined threshold `maxArea`.

In most cases, only parcels with street access should be created, because most buildings have an entrance that should face a street. If the boolean parameter `deleteInnerLots` is set, we delete all parcels that have no direct street access. The algorithm is illustrated in Figure 4.

The complete street creation process is illustrated in figure 5.

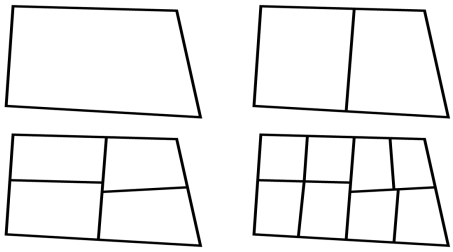


Figure 4: This figure illustrates how a block is recursively split into parcels until each parcel is below the area threshold `maxArea`.

3.2 Street Geometry Tessellation

Street tessellation is not the main scope of our work, so we wanted to implement a simple, but stable geometrical representation of the street network that enables simple shading of streets that adjust to the slope of the underlying terrain.

Tessellating a single street is straightforward, but some problems arise at junctions: If each street would be tessellated and rendered independently, discontinuities and z-fighting would appear where two streets meet. Therefore we need a special geometric representation for junctions.

Another problem that arises is how streets that adjust to multiple height levels should be tessellated so that junctions are planar, but do not form unnatural steps on slopes.

We will first describe our method for street tessellation in case of streets that are coplanar in Section 3.2.1. In Section 3.2.2, we will discuss how we solved the problem of non-coplanar street networks.

3.2.1 Geometry for Planar Streets

The street network is represented as a planar graph of edges that connect to each other at junctions. To draw this network, we need to construct a fully polygonal street representation. The original edges serve as centerlines for the street geometry.

The geometric representation of a street network consists of *junctions* and *street segments*. Each street has a certain `streetWidth` that was set in the street network creation process. We call the point where two street centerlines meet the *center point* of a junction.

To create a polygonal representation of the street, we use a similar approach as the one discussed in [17]: we offset lines from the centerline on both sides by $\frac{\texttt{streetWidth}}{2}$. These offset lines are called *street outlines*. Street outlines of two adjacent street segments intersect at the *corner points* that define the corners of the junction and separate the junction geometry from the street segment geometry. To create the junction geometry, the two corner points of one end of a street segment form a triangle together with the junction center point. This triangle is called a *street head*.

A junction consists of n street heads, where n is the number of street segments adjacent to the junction. This is illustrated in Figure 6. In cases where two junctions are too near to each other, so that junction geometries overlap, the two center points are merged to create a stable junction geometry.

Our method produces simple, but stable results, as can be seen in Figure 7.

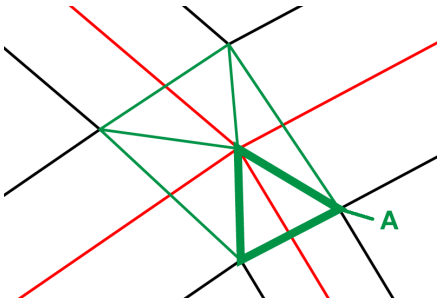


Figure 6: This figure illustrates how a junction is defined by 4 street heads. The street centerlines are displayed in red and meet at the center point. Street outlines are colored black. Street heads are shown in green, the lower right street head is highlighted for clarity (A).

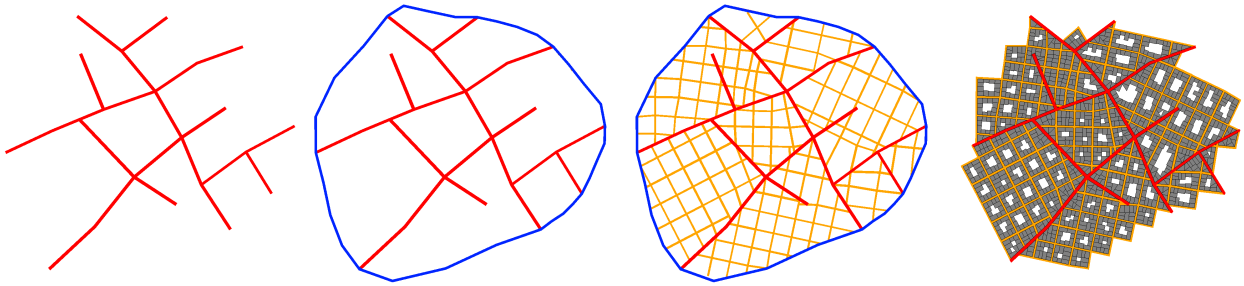


Figure 5: This figure illustrates the steps necessary to create a street network. From left to right: (1) Major streets are created (organic pattern, red). (2) the convex hull is calculated and bulged (blue). (3) Minor streets are created inside the quarters (grid pattern, orange) (4) The final street network with building parcels, convex hull and dead end roads removed.

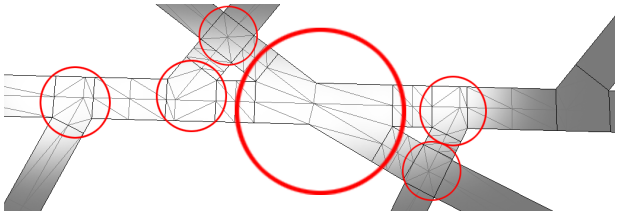


Figure 7: Even complex junctions are tessellated correctly with our method.

3.2.2 Geometry Displacement for 3D Streets

The method described in [17] creates good results as long as all streets are in one plane. When streets are not coplanar, a number of problems arise: The junction geometries described in Section 3.2.1 need to be flat, otherwise streets will twist unnaturally. We introduce a method that nestles street segments and junctions to the terrain underneath while preserving the planarity of junction geometries.

To create flat junction geometries, initially all the vertices of the junction geometry are placed at the same height as the center point of the junction. This leads to unaesthetic and unrealistic steps and extreme slopes, as can be seen in Figure 8. These steps can be smoothed by moving the junction geometry into the tangent plane to the terrain surface at the junction center point. This can be done by calculating how much the normal of the terrain is rotated against the up vector. Based on that, a rotation matrix is calculated around an arbitrary axis that is later applied to every vertex of the junction. As a result, the whole junction geometry is rotated into the tangent plane of the terrain surface. An illustration can be found in Figure 8.

Unnatural steps in steep streets are avoided this way, but we introduce a certain lateral grade. This lateral grade is limited by the longitudinal slope of all other streets adjacent to this junction. This is acceptable for interactive applications such as games, since the maximum allowed longitudinal slope (12%) is not much higher than the maximum allowed lateral grade (8%)[1].

We follow the approach in [17], where street segment

[1] In Austria, this may differ in other countries, and for mountain roads.

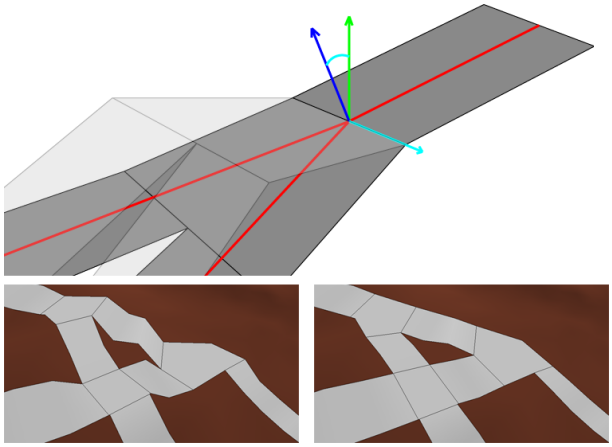


Figure 8: Smoothing of junction geometry. Top: The initial geometry is displayed transparent. It is then rotated into the tangent plane of the terrain surface. Bottom, left: Geometry before rotation. Right: After rotation.

ends are modified so that they connect to the junction at a line perpendicular to the direction of the segment. This is done for the following reason: If the two corner points of a street head do not lie on a line perpendicular to the street segment direction, the street may twist unnaturally or become uneven. Refer to Figure 9 for an illustration.

In between the street heads, the geometry of the street segment is subdivided regularly and displaced according to the terrain height to nestle against the underlying terrain surface.

3.3 Building Assignments

Our main contribution is a technique to assign buildings to parcels from a set of previously modeled buildings. After the street network creation process described in Section 3.1, building parcels of various size and shape have been created, most of them being rectangular. Modeling a suitable building for all of them by hand would be a huge and tedious effort. We propose a method that selects the ”best fitting” model for each parcel from a set of existing buildings. By ”best fitting”, we mean the model that

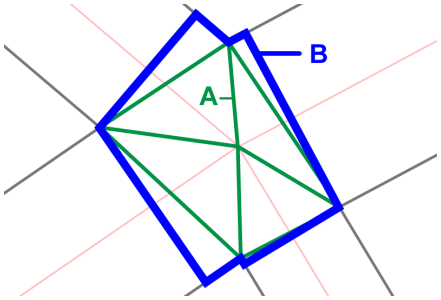


Figure 9: Street ends are modified to create connecting lines perpendicular to the street segment direction. The original junction geometry is shown in green (A), the modified geometry in blue (B).

occupies most of the building parcel, while satisfying various constraints, such as not protruding from the parcel. These buildings can be created using a commercial modeling software like Autodesk Maya, or may be procedurally generated.

The rest of this section is structured as follows: we first discuss some characteristics of building models in Section 3.3.1. Then we describe our method for selecting the "best fitting" building for a parcel in Section 3.3.2.

3.3.1 Building Properties

Each house has a footprint that can be defined as the convex hull of all vertices in its geometry projected onto the ground plane. This gives a good estimate for the area the building will need on a parcel. If this area exceeds the area of a certain parcel, the building will not be considered further for it (Constraint 1).

A building consists of sides that have to face a street side, e.g. because there is a door that needs street access. Also, there are sides that must not face a street, e.g. because there is a plain brick wall on this side. We refer to them as *Street Access Sides* and *Inaccessible Sides*. These sides pose constraints for our system that have to be met. *Street Access Sides* have to be aligned and placed next to streets to guarantee direct street access (Constraint 2), and *Inaccessible Sides* have to point away from streets so that they are not directly visible (Constraint 3). All other faces of the building may face a street, but they do not have to. See Figure 10.

All these properties are stored as meta information in a XML file for each building model.

3.3.2 Selecting a Building

The building with the largest footprint that meets all the criteria described above will be selected as the "best fitting" building for a parcel.

The set of previously modeled buildings is stored in a list ordered by footprint area size from largest to smallest. This list is enumerated for each parcel. All models

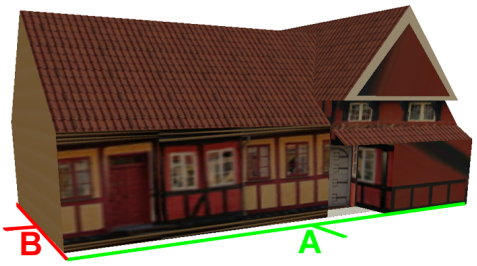


Figure 10: A simple building model. *Street Access Sides* are displayed by green lines on the ground plane (A), red lines denote *Inaccessible Sides* (B).

that have a larger footprint area than the current parcel are discarded. Also, models that contain more *Street Access Sides* than the parcel has adjacent street sides are not considered, because it is not possible to align them all correctly. For every remaining building, a series of transformations and tests are applied. The first model that passes all the tests is chosen for the current parcel. This guarantees that the building that occupies most of the parcel area while meeting all constraints is selected. The process is illustrated in Figure 11.

- The building footprint is moved into the center of the parcel.
- The largest *Street Access Side* of the footprint is aligned to the largest side of the parcel that is adjacent to a street to get an initial alignment for the building. This suffices for most of the buildings, since many of them only have one front side that needs to face the street.
- Rotate the building footprint until all *Street Access Sides* face a street and all *Inaccessible Sides* do not. A side faces a street if it is nearly parallel[2] and a ray cast perpendicular from its center directly hits a street.
- Move the footprint as near as possible to any streets adjacent to the parcel. Buildings usually adjoin directly to streets, but a minimum distance can be configured in the user interface.
- Check if all points of the footprint are inside the parcel.
- If any of the above tests failed, repeat the process with the next smaller building. If a valid solution was found, assign the building to this lot considering the found transformations.

If the parcel is located on a slope, the building is moved down so that every vertex is on the ground or beneath. If the slope of the gradient of the parcel exceeds a user defined threshold `maxParcelSlope`, the parcel is discarded and no building is assigned.

[2] we chose a max. deviation of $\pm$ 30 degrees

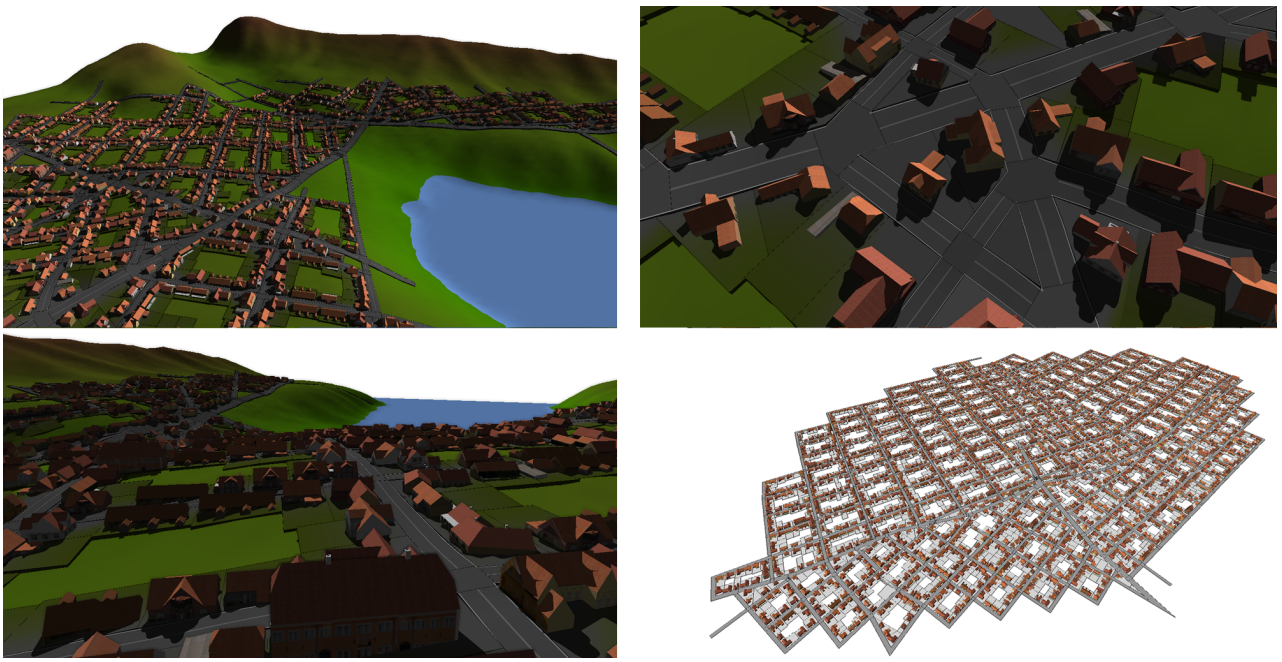


Figure 12: Some results from our system. Top left: a small village on a hillside next to a bay. Note how the roads adapt to different terrain heights and stop if they become too steep. If the ground is too uneven, no parcels are created. Top right: Close-up of a more complex junction in the village. Bottom left: Close-up with view over the bay to the hillside where streets spread over the plain of the terrace. Bottom right: a larger city without terrain with far over 100 streets and more than 2.500 buildings.

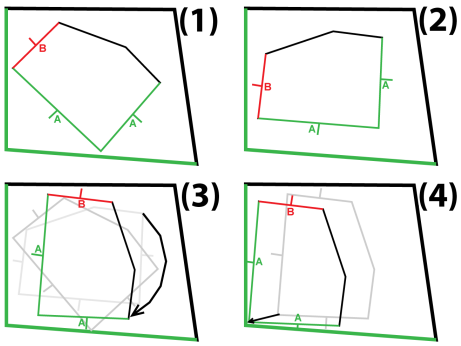


Figure 11: Steps for fitting a building footprint into a parcel. The outer polygon illustrates the parcel, the inner one the building. *Street Access Sides* and parcel sides adjacent to a street are colored green (A), *Inaccessible Sides* red (B). (1) The footprint is moved into the center of the parcel. (2) The largest *Street Access Side* is aligned with the largest parcel side with street access. (3) The footprint is rotated. (4) The footprint is moved near to the streets.

4 Results

Our system is capable of procedurally generating whole cities with dozens of streets and hundreds of buildings within seconds. Most parameters used for the creation of street networks and the assignment of buildings can be changed in our user interface. The assignment of buildings worked well in our tests, however the visual result depends on the quality and amount of models used. If only a few buildings are available, very uniform neighborhoods are created. The larger the set of models is, the more diverse the cities become. For our tests, we used 20 models available freely at Google 3D Warehouse and achieved pleasing results. Some screen shots of our results can be seen in Figure 12.

We implemented our system in C# using the XNA framework for rendering. All tests and images in this paper were made on a system consisting of a Core 2 Quad 6600 with 4GB RAM. At the moment our system is single threaded, but it could easily use multi threading, especially for the parcel subdivision and building assignments. See Table 1 for detailed results.

Task	Count	Time
Street creation	73 streets, 283 segments	0.65s
Street tessellation	73 streets, 283 segments	0.17s
Parcel Creation	811 parcels, 107 blocks	0.08s
Buildings	776 buildings assigned	2.24s
Total		3.14s

Table 1: Performance benchmarks of our system for a small city consisting of 4 main and 69 minor roads. The whole city was created in just over 3 seconds.

5 Conclusion and Future Work

We presented a system that procedurally creates urban environments including street networks, street geometry and building parcels. Very different types of cities can be created by changing parameters in the graphical user interface. Our main contribution is a constraint based system that choses the "best fitting" building for every parcel. Building properties such as the footprint, area and faces that should have street access are used to select the most suitable building.

Furthermore we describe for the first time a robust method to create 3D street geometry for streets that adapt to different terrain heights.

In existing street generation systems, no minor roads are created at the city borders. We addressed that problem by calculating a convex hull around the city and widen it by random amounts to create a more realistic city shape and more detailed outskirts.

5.1 Future Work

Image maps could be used to control more parameters of the building assignment process and give the user more control over the selection of buildings for the parcels.

An input map similar to a height map could be used to control the building height for different regions. Another map could be used to manage the type of buildings: A building can be of a certain type (residential, industrial, suburban, etc.), and each type is associated with a certain color in the map. This information could be used in the building assignment process to create districts with a different look and feel.

The main limitation of our current system is that it is static. Models placed automatically can not be moved or modified in any way. We want to give the user the power and flexibility to change the environment after it has been created automatically.

References

[1] D. G. Aliaga, C. A. Vanegas, and B. Beneš. Interactive example-based urban layout synthesis. *ACM Trans. Graph.*, 27(5):1–10, 2008.

[2] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang. Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3):1–10, 2008.

[3] Gernot G. Automatisches generieren von 3d-straßensystemen. Master's thesis, Vienna University of Technology, March 2008.

[4] C. Jaynes, E. Riseman, and A. Hanson. Recognition and reconstruction of buildings from multiple aerial images. *Comput. Vis. Image Underst.*, 90(1):68–98, 2003.

[5] G. Kelly and H. McCabe. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology*, pages 8–16, 2007.

[6] K. A. H. Kjølaas. Automatic furniture population of large architectural models. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, January 2000.

[7] M. Lipp, P. Wonka, and M. Wimmer. Interactive visual editing of grammars for procedural architecture. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, New York, NY, USA, 2008. ACM.

[8] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[9] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 397–410, NY, USA, 1996. ACM.

[10] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 301–308, August 2001.

[11] Procedural Inc. Cityengine, www.procedural.com, 2010.

[12] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[13] I. Suveg and G. Vosselman. Reconstruction of 3d building models from aerial images and maps. *ISPRS Journal of Photogrammetry and Remote Sensing*, 58(3-4):202 – 224, 2004.

[14] C. Vanegas, D. Aliaga, P. Wonka, P. Müller, P. Waddell, and B. Watson. Modeling the appearance and behavior of urban spaces. *Comput. Graph. Forum*. to appear.

[15] B. Weber, P. Müller, P. Wonka, and . Gross. Interactive geometric simulation of 4d cities. *Computer Graphics Forum*, 28(2):481–492, April 2009.

[16] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003.

[17] M. Zimmermann. Procedural construction of streets. Technical report, ETH Zürich, 2007.