ON-DEMAND CREATION OF PROCEDURAL CITIES

M. Banf, M. Barth, H. Schulze, J. Koch, A. Pritzkau, M. Schmidt, A. Daraban, S. Meister,
R. Sandhöfer, V. Sotke, C. Rezk-Salama, A. Kolb
Institute for Vision and Graphics, University of Siegen, Germany

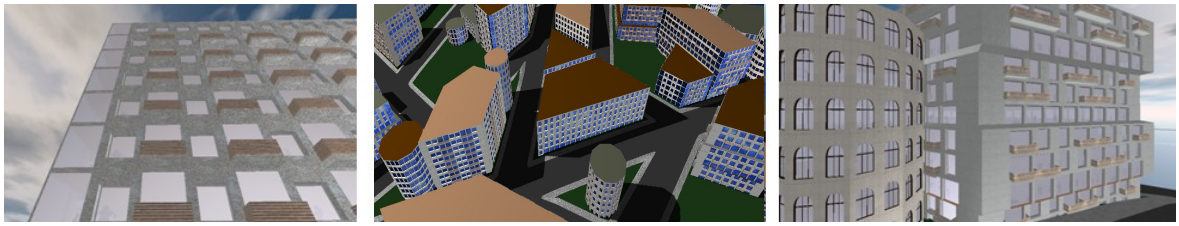

Figure 1. *On-Demand Creation of Procedural Cities* in action. Create infinite streets, terrain and houses on the fly.

ABSTRACT

We report about our student project with the objective of a procedural generation of pseudo-random cities, streets and terrains. The focus of the project is primarily the procedural modelling, real-time rendering and modelling on demand at run-time. This paper discusses the generation of terrain, road network, different approaches to build houses like random methods or shape grammars, and additional graphical effects. The framework is designed in away that a user without any previous knowledge is able to create infinite worlds on the fly as easy and interactively as possible while still having a wide influence on the appearance.

KEYWORDS

Procedural Architecture, Procedural Modeling, Street Networks, Street Modeling, Terrain Generation

1. INTRODUCTION

Cities are essential components of environments in computer games or films. A specific location is near by not relevant, but the visualization of fictional but preferably realistic scenes. The creation of such environments was so far the result of time-consuming manual modeling of one or more level designers. The trend of increasing size and more realistic cities, causes increasing modeling costs. To avoid problems like these, our student project group addresses to procedural generation of cities. The generation contains terrain, road network and buildings (Figure 1). Our algorithm firstly generates and samples the visible part of the road network within a 2D plane (see Chapter 3). The resulting 2D-coordinates are passed to our terrain module (see Chapter 4), where height values for all sampling points are calculated. At last our algorithm is able to use different approaches, using randomized rules and shape grammars, to build houses placing them into appropriate areas (see Chapter 5). One of the fundamental principles of the project is the generation on demand, allowing the creation of ever new parts of the city on the fly as the user moves through the scene. Using a pseudo-random approach for each object within the scene, depending on position coordinates, guarantees a constant appearance of the city.

2. RELATED WORK

Promising ways to reconstruct urban models but quite labor intensive are methods using photographs [Debevec et al., 1996; Dick et al., 2001; Jepson et al., 1996; REALVIZ, 2002] or videos and range scanning [Karner et al., 2001; Ribarsky et al., 2002; Teller, 2001].

The idea of modeling urban environments procedurally, including streets and buildings, was recently explored by [Parish and Müller, 2001; Wonka et al. , 2003; Müller et. al., 2006; Chen et. al., 2008]. [Parish and Müller, 2001] first note that street network is the key to creating large urban models, presenting a solution based on L-Systems [Prusinkiewicz and Lindemayer, 1991], that were originally applied to simulate growth processes of plants. To enhance user-control of such street networks [Chen et. al., 08] provide an extended approach using tensor fields.

In contrast to L-Systems shape grammars, as introduced by [Stiny, 1975], are more appropriate to create buildings as these grammars do allow a more powerful control using rules. [Parish and Müller, 2001] showed how to generate large urban environments where each building consists of simple mass models and shaders for facade detail. [Wonka et al., 2003] demonstrated how to generate geometric details on facades of individual buildings. A combination of both techniques can be found in [Müller et. al.,2006].

Good overviews over the creation of urban environments are given by [Mitchell, 1990], about grammars in architecture in particular, as well as other notable works related to urban design [Hillier, 1996; Alexander et al. 1977; Gingroz et. al., 2004]. Automatic modeling techniques are already available for enhancing existing architectural models, e.g. using cellular textures [Legakis et al. 2001] and texture synthesis [Wei and Levoy, 2000].

Compared to previous techniques, the aim of this work was to build a procedural city which is constructed on-the-fly while the user moves through the scene. If the user is about to enter a new region, it will be automatically generated. Likewise, if the user has left a region for a given distance, the geometry will be discarded, and rebuilt on-the-fly in exactly the same way as before if the user decides to re-enter this part.

3. GENERATION OF STREETS

Figure 2. Left: Normal and Orthogonal street generation and parts of the original street structure of Manhattan; Right: Storage of subdivided polygons using tree structure.

Though being able to create infinite cities on-the-fly, our road network makes use of a base area, adjustable in size. Infinity is simulated allowing the user to walk out of the area re-entering it on the other side representing another part of our city. An adjustable seed point as initialization of our pseudo-random algorithms is used, which defines and guarantees the consistent appearance of our city, as the user walks forth and back between parts of the city.

The generation of the streets is restricted to a specified radius around the user position. Streets within that radius are generated while others outside might be postponed until the user moves into that area.

3.1 Street Division Algorithm

Recursive subdivision generates an amount of polygons from the initial base polygon. Because individual polygons do not share information, each edge of a polygon represents a *half-street*.. A complete street will be formed in combination with the edge of an adjacent polygon. To avoid incomplete polygons, each polygon, that lies even partially within range, is completely computed.

Storage of the initial and all subdivided polygons is realized using a tree structure (Figure 2), where each polygon stores a pointer to its parent as well as its left and right child node. Whenever a polygon is divided we check whether further divisions are possible or whether the resulting polygons serve as leaf-nodes. Leaf-polygons represent areas for positioning houses. To be as close as possible to reality our algorithm allows more than one building to be placed onto each polygon. Based on a user definable threshold it calculates

further subdivisions of leaf-polygons, called *parcel subdivisions*, smaller areas within the leaf-polygon for a certain amount of houses.

We implemented two different approaches to divide a polygon using an arbitrary straight or an orthogonal line, both based on a pseudo random algorithm. Using this approach it is possible to create more planned street networks like e.g. downtown New York as well as more evolved road systems (Figure 2). The algorithm guarantees that streets will not become too short when dividing polygons, based on a minimum length condition, adjustable by the user. Additionally, it is possible to create areas using different minimum length conditions, resulting in more or less denser road networks, to simulate e.g. a city centre and surrounding suburbs.

3.2 Sampling the Road Network

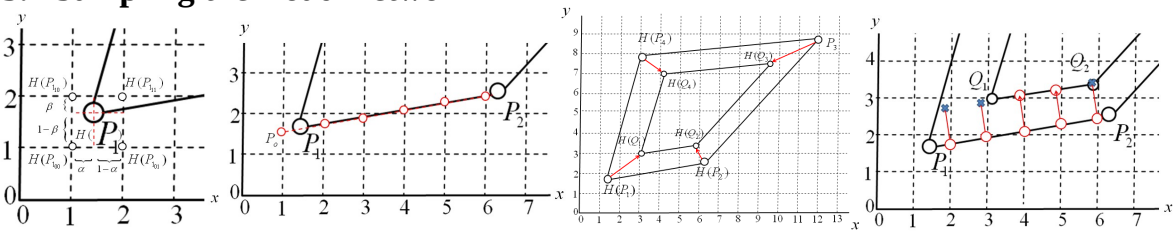

Figure 3. Left: Calculation of corners and appropriate height values using linear combination; Middle: Sampling points lying on the grid (for illustration purposes the distance between grid lines is set to 1); Right: Inner polygon, scaled from the outer polygon, height values of the corners are mapped. Projection of sampling points and their height values.

The base terrain in our approach is based on a combination of street networks and a uniform grid for the height field and the occlusion culling and rendering. Height field generation is discussed in Section 4.
The foundation of sampling the computed road network is a global, regular grid. The resolution of that grid is specified in advance by the user. The higher the resolution, the more sampling points are computed resulting in higher visual quality. On the other hand computing costs increase along with the resolution compromising real-time capabilities.

3.2.1 Sampling Corners

To create an individual polygon (Figure 3, e.g. having 4 corners) within 3D space, guaranteeing a seamless and waterproof mesh structure of all adjacent polygons, as they are not directly linked during triangulation, we approximate the height value of each corner doing a bilinear interpolation of the height values of its four surrounding, nearest neighbour grid points (Figure 3). Therefore these four points as well as the appropriate interpolation weights can easily be calculated using the vertex position within the grid plane:

$P_{1_{00}} = (\lfloor x_1 \rfloor, \lfloor y_1 \rfloor)$, $P_{1_{01}} = (\lceil x_1 \rceil, \lfloor y_1 \rfloor)$, $P_{1_{11}} = (\lceil x_1 \rceil, \lceil y_1 \rceil)$, $P_{1_{01}} = (\lfloor x_1 \rfloor, \lceil y_1 \rceil)$

$\alpha = x_1 - \lfloor x_1 \rfloor$, $\beta = y_1 - \lfloor y_1 \rfloor$ with $\lfloor x \rfloor = \max\{n \in \mathbb{Z} \mid n \le x\}$, $\lceil x \rceil = \min\{n \in \mathbb{Z} \mid n \ge x\}$

The height values of these grid points are calculated utilizing our Terrain Generator (see Chapter 4). As a result, for each corner we get:

$H(P_1) = \beta * H_a - (1-\beta) * H_b$ with $H_a = \alpha * H(P_{1_{00}}) - (1-\alpha) * H(P_{1_{01}})$, $H_b = \alpha * H(P_{1_{10}}) - (1-\alpha) * H(P_{1_{11}})$

3.2.2 Sampling along the Edges

Calculating sampling points on the edges of our polygons again involves making sure not to create gaps at the transitions of polygons, as they are not linked and do not know anything about their neighbours. The presented algorithm verifies the compliance of this condition, sampling at positions on the grid. Therefore we define a line equation within grid plane, starting from a certain position $P_o = (x_o, y_o)$ on the grid, which has to be calculated at first: $P = \begin{pmatrix} x_o \\ y_o \end{pmatrix} + \gamma * a$ with $a = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$; $\gamma = 1,2,...,n$

To guarantee proceeding from point to point on the grid the gradient a is normalized in x respectively y direction within two calculation steps. The number of sampling points along a particular edge is given by the number of intersected grid points.

3.2.3 Building Streets, Sidewalks and Housing-areas

As the outer edges of the polygons represent the centrelines of the surrounding streets, we create new polygons to build a sidewalk and the borders of the final area to place the houses. To get streets that might rise or fall along the run but stay constant in profile we project the height values of the outer polygon lines onto newly created points, that define the inner polygon, based on a distance w_{street}, representing the width of a street. Before acceptance, we evaluate the projected sampling point to actually lie on the inner edge or outside its borders. If so, it is accepted as new point on the inner edge and rejected otherwise.

Intuitively, the inner polygon is formed by scaling the outer polygon lines according to w_{street} (Figure 3). Due to this method we guarantee a uniform street structure that is built of two polygons, not knowing each other. Disadvantage of polygons not "knowing each other" might be the redundancy in calculating height values at neighbouring edges.

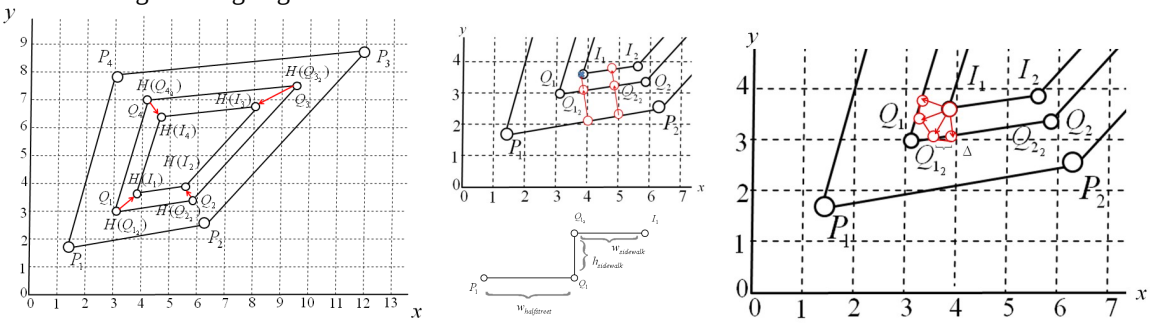

Figure 4. Left: Creating sidewalk and housing polygons; Right: Creation of the round sidewalks as parts of an approximated circle.

To create a sidewalk we define two additional polygons. One that equals the polygon, originally described as the inner polygon (Figure 5: defined using $Q_{1...4}$), using a constant $h_{sidewalk}$, the height of our sidewalk (which is constant throughout our application) to be added to all height values and another one created the same way as the previous inner polygon line, using a constant $w_{sidewalk}$ as uniform width of our sidewalks (Figure 4). To create street scenes as realistic as possible, our algorithm constructs rounded sidewalk corners, which look much more natural than sharp borders (Figure 5).

Figure 5. Left: Adjacent half-streets; Right: Rounded sidewalk corners.

3.3 Triangulation of Streets and Terrain

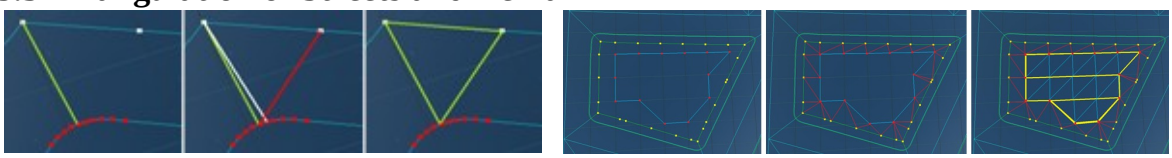

Figure 6. Left: Triangulation of a segment – Starting from the corners find and select the best possible edge to triangulate; Right: Extraction (blue) and triangulation (red) of the *ring* and line-based triangulation of the remaining area based on the grid.

For triangulation, all edges of an individual polygon are divided at their corners to be used as separate segments. Due to the grid, the topology within a segment can easily be set. Regularly each segment contains

different height values, so we cannot render them as planar polygons. To involve the different heights into rendering of streets, we triangulate each particular segment, using a fast greedy algorithm.

As for the segments, a greedy algorithm is utilized for the inner areas of each polygon as well. Due to small angles in some of the polygons edges as well as sampling points lying on or too close to these edges we had to extend our implementation: We calculate a kind of border structure, called *ring*, to be triangulated at first, to be able to a do a line-based triangulation, accounting for the grid points (Figure 6). Therefore some points have to be moved or in some cases additional points must be inserted.

4. CREATION OF LANDSCAPES

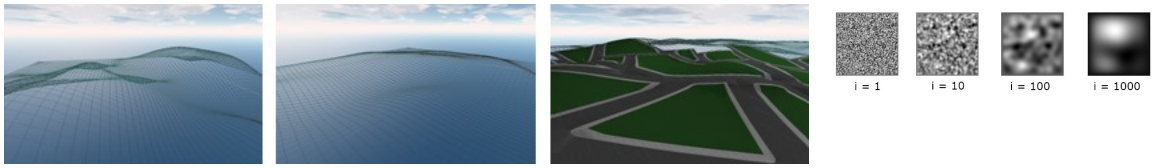

Figure 7. Left: two different results created by the terrain generator. Middle: Streets and Terrain based on the created terrain geometry; Right: Different iteration steps to use with our fluid simulation.

To not simply create flat cities we encountered different possibilities to build up a natural looking terrain geometry to embed our streets and cities into. Examining several approaches like *pure randomness*, *simulations of erosion and filtering*, *noise functions* [Ebert et. al., 1998], we finally choose a *fluid simulation* [Lengyel, 2001; GameDev.Net, 2003] as it is more appropriate for the creation of natural looking environment under real-time conditions.

The algorithm starts, filling a field with pseudo random numbers, being treated like the surface of a fluid. Applying fluid simulation equations to that surface, we smooth randomness, gaining realistic looking structures. More iterations of those equations lead to smoother height fields (Figure 7).

5. BUILDING HOUSES

Consistent with the road network or terrain geometry, we did not intend storing predefined models for the buildings. There is a need to create the structure, appearance and shape of buildings dynamically as they are to be drawn. As just a small region of our city is drawn, it is necessary to delete building geometry that is not seen any more as the user moves.

To ensure reproducibility, which means that exactly the same buildings must be created at the same position repeatedly, we use pseudo random numbers depending on the individual position within the grid plane. We implemented two separate approaches described below to address all problems. At this stage, it was not our intention to create highly detailed buildings, like the ones presented in [Parish and Müller, 2006], but we focused on on-the-fly generation. Nevertheless, creating more complex buildings based on an extension of the building rules used within our framework may be straight forward.

5.1 XML Grammars

Our approach draws from the work on split grammar and control grammar presented by [Wonka, P. et al., 2003]. Our grammar is developed on the basis of XML. This allows our system to be expanded by different grammars without the need for recompilation. We achieve this by the use of a data binding mechanism. The semantics of the grammar rules are defined in a schema file. The *CodeSynthesis XSD* compiler is used to establish a mapping between the XML schema and the target programming framework. The result is a ready-to-use domain-specific XML parsing framework.

The grammar itself is made up of grammar rules to specify the spatial layout of the buildings. The assigned probabilities can be applied to choose between competing derivation rules. One can choose between transformation rules such as translation, rotation, scaling as well as push and pop operations to preserve a specific transformation state. By means of subdivision rules the overall building scope can be split into

subparts to be derived in a different manner, e.g. alternating floor styles of a building. Furthermore the XML approach is also used to specify texture configurations, loaded at startup. At the end of the derivation process each scope is decomposed into its faces. According to their type corresponding textures are mapped on these faces.

5.2 Area-Dependent Random Shapes

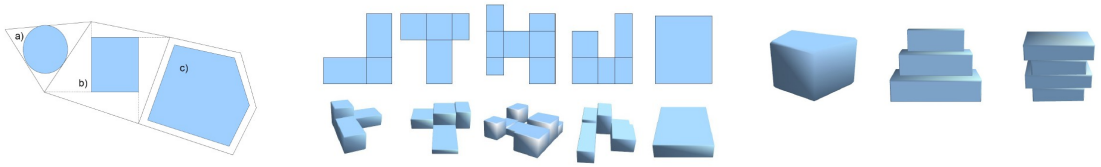

Figure 8. Left: Different types of parcel subdivisions; Middle: Quadric building areas can be split into different shapes; Right: Area filling as well as basic shape approaches allow generation terraces or alternating floors.

Our novel *area-dependent random shapes* approach allows us to change appearance of our city during runtime accordingly to changing the random seed value. Based on that value as well as the position coordinates, we define texture files of facades, roofs or other house components, like balcony or window frames as well as their shapes. This is a significant benefit compared to previous approaches, as the city will be generated interactively.
Shapes of houses in general depend on the areas they are build upon, since the parcel subdivisions, created by the road network may have different forms (Figure 8). In cases of triangles, we favor round buildings, like towers. In other cases, it is possible to generate a building filling an area or a quadratic basic shape building. Generating area filling houses or buildings with a circular platform involves the creation of walls, as single planes with a specific height and width as well as an iterative process separating walls into floors and floors into further tiles. Each tile may contain a window, balcony as well as just being part of concrete wall. Based on the random seed the algorithm further customizes any final tile, choosing out of predefined algorithms, e.g. as for defining the final form of a specific window.
The quadratic basic shape approach starts by defining the borders of the later building, computing a quad within the base area. In dependency of a pseudo random number, the algorithm select a L-, T-, H-, U- or quad basic shape (Figure 8). Shaped objects are split into single rectangles with different height, width and length values (called *Basic Shapes*). E.g. an H-shaped building contains seven basic shape objects. For each Basic Shape we compute the wall objects regarding whether walls are hidden by adjacent shapes. All wall objects will be put together to our house for further algorithms to tile them.
Both, area filling as well as basic shape approaches allow the generation of terraces and alternating floors, differing in width and height based on our random seed value (Figure 8).

5.3 Plugin Management

To be able to add more elements into our scenes like blend shapes or vegetation, without touching the code or varying the building rules, we created a plugin interface into our framework. Therefore using a pseudo-random algorithm, the system decides whether a specific housing area would be build or remains free. The percentage of build and free areas can be controlled interactively. For each free area we create a dataset from its corners as well as the grid points lying within. The plugin has to implement three methods. One to verify that a particular plugin is written to use with our framework, one for initialization issues, which is called during loading the plugin and a render method, which is called during the render procedure of our application, utilizing the previous mentioned dataset. The plugin is compiled and linked as dynamic link library (dll). Plugins can be loaded automatically at startup or manual during runtime.

6. VISUAL EFFECTS

Terrain and urban calculation is done on CPU. To enhance overall visual quality without losing too much performance for calculations, we embedded common computer graphics techniques such as *Environment Mapping*, *Shadow Mapping*, *Normal Mapping*, *Distance Fogging* as well as for further realism a shader controlled day – night period using the CG Shading language [Fernando R. and Kilgard M.J., 2003].

Figure 9. Left: *Environment Mapping* using a dynamic Cube Map, rendered depending on the viewers position. Opposite buildings, streets and not occluded parts of the clouds texture are reflected within windows; Right: Simple Texture Mapping compared to *Normal Mapping*. 3D Impression depends on radiance of light.

Figure 10. Left: *Shadow Mapping* not only enhances image quality but also increases the depth impression of the scene. Right: *Distance Fogging* adds realism and character to the scene.

7. CONCLUSION

Strengths of our framework is first of all the ability to compute and build real infinite worlds on the fly, as the user moves through the scene, including algorithms to create realistic looking road networks embedded into a natural environment and an intelligent creation and placement of buildings, given the user a high interaction possibility as well as an as easy as possible overall control. Furthermore there is the modular structure, which allows an easy extension of the system as well as a simple exchange of different parts (e.g. the terrain generator) as well as the plugin structure that allows an easy import of blend shapes and other elements to extend our environment. Besides the modular structure, due to using shape grammars it is straight forward to build much more complex buildings or village styles.

Limitations are that our framework does not, as originally planned, create round streets and different street widths, which would enhance realism, as well as the redundant calculations during sampling due to the polygon structure. To that point our framework does not make use of threads which might be helpful to compensate these redundant calculations and speed up overall computations of streets and houses. Another problem is, that houses intersect with the terrain at their bottom planes, due to their current positioning algorithms.

Possible applications could be the entertainment, especially the game and movie industry to help decreasing modeling time – therefore it might be worthwhile to extend the modeling algorithms recognizing historical developments of cities as well as the effects of nature catastrophes.

REFERENCES

Alexander C.,Ishikawa S. and Silverstein M., 1977. *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, New York.

Chen G., Esch G., Wonka P., Müller P. and Zhang E., Interactive Procedural Street Modeling, , *Proceedings of ACM SIGGRAPH 2008*, 29(3), ACM Press, 9 pages.

Debevec P.E., Taylor C.J. and Malik J., 1996, Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach, *Proceedings of ACM SIGGRAPH 96*, ACM Press, 11 - 20.

Dick A., Toor P., Ruffle S. and Cipolla R. , 2001, Combining single view recognition and multiple view stereo for architectural scenes, *Proceedings of the Eighth International Conference On Computer Vision*, IEEE Computer Society, Los Alamitos, CA, 268 – 274.

Duarte J., 2002, *Malagueira Grammar – towards a tool for customizing Alvare Siza's mass houses at Malagueira*, PhD Thesis, MIT School of Architecture and Planning.

Ebert D. et al, 1998, *Texturing and Modeling: A Procedural Approach (2nd Edition)*, Morgan Kaufmann Publishers, San Francisco, USA.

Fernando R. and Kilgard M.J., 2003, The *Cg Tutorial*, Addison-Wesley.

GameDev.Net, 2003, *Terrain Generation Using Fluid Simulation*, http://www.gamedev.net/reference/articles/article2001.asp

Gingroz R., Robinson R., Carter D.K., and Ostergaard P., 2004, *The Architectural Pattern Book: A Tool for Building Great Neighborhoods*, W. W. Norton & Company

Hillier B., 1996. *Space Is The Machine: A Configurational Theory Of Architecture*, Cambridge University Press.

Jepson W., Liggett R. and Friedman S., 1996, Virtual modeling of urban environments, *PRESENCE* 5(1), 72 – 86.

Karner K. Bauer J., Klaus A., Leberl F. and Gräbner M., 2001, Virtual habitat: Models of urban outdoors. *Third International Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Imaging*, 393 – 402.

Legakis J., Dorsey J. and Gortler S.J., 2001, Feature-based cellular texturing for architectural models, *Proceedings of ACM SIGGRAPH 2001*, ACM Press, 309 – 316.

Lengyel E., 2001, *Mathematics for 3D Game Programming and Computer Graphics*, Charles River Media, Hingham, USA.

Mitchell W.J., 1990, *The Logic of Architecture: Design, Computation and Cognition*, MIT Press.

Müller P., Wonka P., Haegler S., Ulmer A. and Gool L.V., Procedural Modeling of Buildings, *Proceedings of ACM SIGGRAPH 2006*, 25(3), ACM Press, 614 – 623.

Parish Y. I. H. and Müller P., 2001, Procedural modeling of cities, *Proceedings of ACM SIGGRAPH 2001*, ACM Press,301-308.

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

REALVIZ, 2002, Image modeler product information. http:///www.realviz.com

Ribarsky W., Wasilewski T. and Faust N., 2002, From urban terrain models to visible cities, *IEEE Computer Graphics & Applications 22*(4), 231 - 238

Stiny G., 1975. *Pictorial and Formal Aspects of Shape and Shape Grammars*, Birkhauser Verlag,Basel

Teller S., 2001, MIT city scanning project: Fully automated model acquisition in urban areas. http://city.lcs.mit.edu//city.html

Wei L.-Y., Levoy M., 2002, Fast texture synthesis using tree-structured vector quantization. *Proceedings of ACM SIGGRAPH 2000*, ACM Press, 479 – 488.

Wonka P., Wimmer M., Sillion F. and Ribarsky W., 2003. Instant Architecture. *ACM Transaction on Graphics*, 22(3), 669 - 677.