Procedural 3D Caves, Clouds and Architecture Generation Method Based on Shape Grammar and Morphing

Tomasz Zawadzki, Slawomir Nikiel, *University of Zielona Gora*
Eraldo Ribeiro, *Florida Institute of Technology*

***Abstract*— This paper presents a new procedural 3D model-construction algorithm that benefits from a combination of discrete and continuous modeling approaches. Our algorithm models complex scene components such as caves, architectural buildings, and clouds. The method combines the discrete descriptiveness of shape grammars with the continuous flexibility of shape morphing. This combination allows for a modeling approach that can be controlled by a morphing parameter to produce various types of geometry. In the paper, we focus on the description of the algorithm while also showing its capabilities in generating complex scene components.**

***Index Terms*— algorithms, procedural modeling, shape grammar, computer graphics, feature-based models, morphing.**

I. Introduction

Complex structures are necessary elements of visually convincing virtual scenes. Buildings [1], whole urban structures [2],[3], terrains [4],[5], clouds [6]-[11], plants [12] or caves [13]-[17] can be modeled with help of systems based on automated shape construction. The algorithms that enable full automation of the modeling process help to achieve large savings in the digital media production time and budget. Procedural systems are also used in the CAD systems such as: *City Engine* – procedural cities, *Houdini* – procedural animation, *Terragen* – procedural landscapes, *Art of Illusion* – procedural textures. We can observe a constant development of new methods i.e. merging technology and dynamical systems [18], [19]. The problem of automated shape modeling constitutes an important area of computer graphics activity and has drawn attention of digital media industry for several years. Digital movies have created constant demand for pleasing visual effects in three-dimensional graphics. In addition to entertainment applications, shape modeling has the practical use ranging from CAD engineering applications, through scientific visualization to advanced game programming and Virtual Environments. As far as we consider real-time simulations, it is very hard to easily satisfy the above-mentioned demand. Moreover, it is almost impossible to do ~~so~~ without the use of procedural modeling systems. We propose to extend the set of currently available procedural methods with a hybrid of shape grammar and morphing, offering better performance and versatility.

Manuscript received March x, 2013. The main author is a scholar within Sub-measure 8.2.2 Regional Innovation Strategies, Measure 8.2 Transfer of knowledge, Priority VIII Regional human resources for the economy Human Capital Operational Programme co-financed by European Social Fund and state budget under Grant nr DSF.VI.052.4.43.1.2012.

Tomasz Zawadzki is a PhD student at Faculty of Electrical Engineering and Computer Science, University of Zielona Góra, Poland. He is now with the Department of Computer Sciences at Florida Institute of Technology, USA (e-mail: t.zawadzki@my.fit.edu).

Slawomir Nikiel, is an Assoc. Prof. at Institute of Control & Computation Engineering, University of Zielona Gora, Poland (e-mail: s.nikiel@issi.uz.zgora.pl).

Eraldo Ribeiro, is an Assoc. Prof. Department of Computer Sciences at Florida Institute of Technology, USA (e-mail: e.ribeiro@cs.fit.edu).

II. Previous works

A. Shape grammars

Stiny and Gips are precursors of the shape grammars. Their work supported the design process using a "linguistic model of the generational system" [20], [21]. The definition of shape grammars is analogous to the one of formal grammars, and is graphically expressed in terms of words composed of symbols with and a set of grammatical rules called productions [22].

Definition 1. Shape grammars are defined as follows:

$$SG =< V_T, V_M, R, I >, \quad (1)$$

where:

$\boldsymbol{V_T}$ - is a set of terminal shapes,
$\boldsymbol{V_M}$ - is a set of non-terminal shapes ($V_T^* \cap V_M = \emptyset$),
$\boldsymbol{R}$ - is a finite set of ordered pairs *(u, v), and*
$\boldsymbol{I}$ - is an initial shape.

Pairs *(u, v*) are such that *u* is a shape consisting of an element of V_T* combined with an element of V_M, and *v* is a shape consisting of: (A) the element of V_T* contained in *u,* or (B) the element of V_T* contained in *u* combined with an element of V_M, or (C) the element of V_T* contained in *u* combined with an additional element of V_T* and an element of V_M.

B. Morphing

DOI: 10.5176/2251-3043_3.1.249

The idea of shape metamorphosis commonly known as morphing forms a wide and important area of computer graphics. Generally, morphing can be defined as a both continuous (i.e., over time) and smooth process of transformation of one shape into another. So-called key shapes (by analogy to key-framed animation) may have different topologies, and transformation smoothness does not have to a homeomorphism [23]. Typically, morphing is a two-step problem, namely, the step of determining 'features' of key shapes to be morphed, and the interpolation of shapes according to 'trajectories'. The second step results in intermediate shapes that have some topological characteristics of both the key shapes (i.e., the 'beginning' and the 'final' shapes). The morphing intermediate shapes are used in this paper as an alternative to CSG Boolean productions (e.g., the 'beginning' AND 'final' shapes). There are various methods for metamorphosis for 2D shapes represented mostly by polygons [24] but also images [25]. The problem has also been investigated in 3D domain, and according to Lazarus [26] they span the methods based on polygonal mesh representation [27], [28] and the methods using voxel representations [29].

III. SG-M Hybrid Algorithm

A. Construction of the hybrid system

Our hybrid method combines two independent approaches. We use a hierarchical shape representation along with a set of operations that can be applied to the shapes (i.e., shape rules). In addition to discrete shape operations, we incorporate a continuous morphic shape rule.

The representation consists of three main elements: *the root* – a place where the modeling process ends, *the nodes* – stores information about actual shape appearance, and *leafs* – determines the possibility of rules application (Fig. 1).

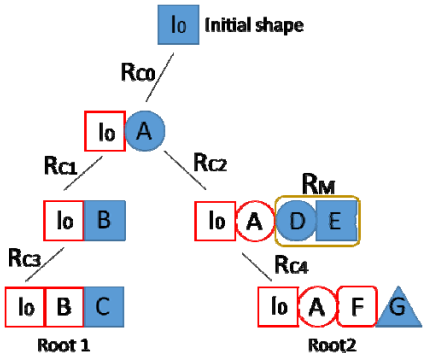

Fig. 1. Hierarchical representation with two possible modeling ways. Specifies terminal shapes are shown in red. Non-terminal shapes are shown in blue.

The example shown in Fig. 1 can be described as follows:

$$R_0: I_0 \Rightarrow I_0 \cup A \quad (2)$$

$$R_1: I_0 \cup A \Rightarrow I_0 \cup B \quad (3)$$

$$R_2: I_0 \cup A \Rightarrow I_0 \cup A \cup (D\ morph\ E) \quad (4)$$

$$R_3: I_0 \cup B \Rightarrow I_0 \cup B \cup C \quad (5)$$

$$R_4: I_0 \cup A \cup (D\ morph\ E) \Rightarrow I_0 \cup A \cup F \cup G \quad (6)$$

$$R_0 \Rightarrow R_1 \Rightarrow R_3 \quad (7)$$

$$R_0 \Rightarrow R_2 \Rightarrow R_4 \quad (8)$$

Shape rules contain Boolean operations such as sum, difference or union. We call these *classic shape rules* (R_C). We then provide another type of rule - *morphic shape rule* (R_M), which works by morphing two input shapes into one output shape using a linear-interpolation parameter (Fig. 2). Figure 3 shows a flow diagram of the modeling process. Our system can apply matching rules only to non-terminal shapes that are recognized during modeling process (Fig. 4).

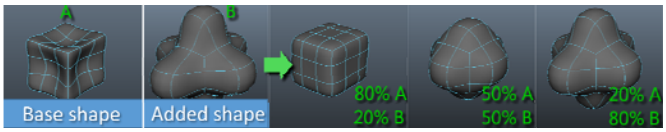

Fig. 2. An example of morphing rule (R_M) for another contributions of shape A and B.

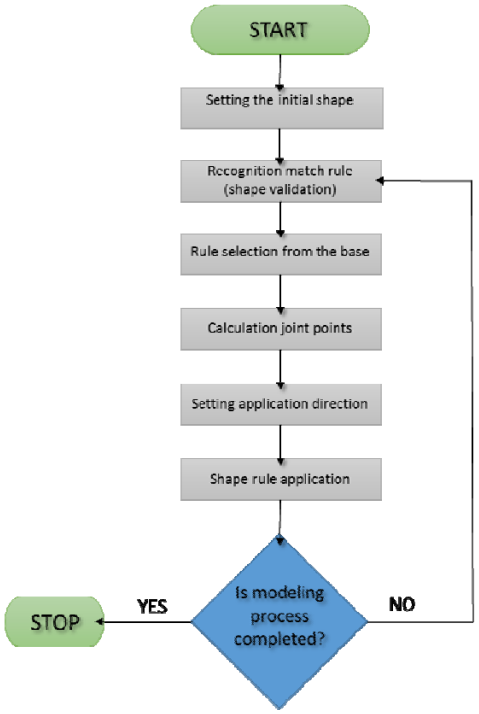

Fig. 3. Main SG-M block diagram.

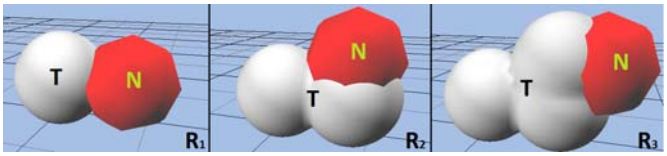

Fig. 4. Terminal and non-terminal shapes on simple rule example: sphere $_{(T)}$ ∪ sphere $_{(N)}$ ⇒ sphere $_{(T)}$ ∪ sphere $_{(T)}$ ∪ sphere $_{(N)}$ directly from editor.

B. *Procedural cave generation using SG-M concept*

The number of iterations is equal to the sum of all rules R_C and R_M. In each rule, the scene base shape is recognized and subsequently added to another shape formed by morphing a sphere and a cube (Fig. 5).

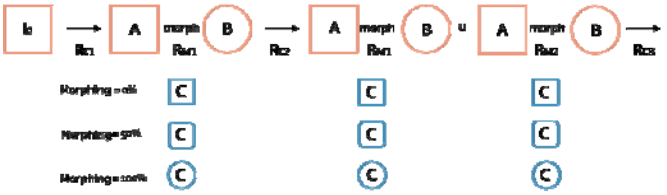

Fig. 5. Application of applying shape rules and controlling the modeling process by morphing parameter in cave mode. Letters A, B, C define shapes.

It is possible to select the direction of the joint points under which new rules will be added (+X/-X, +Y/-Y, +Z/-Z). It is also possible to control the S_J parameter (Fig. 6, 7).

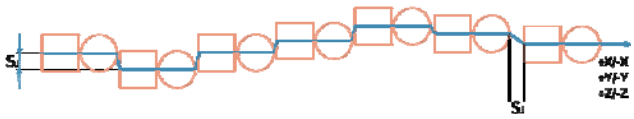

Fig. 6. Controlling the modeling process using the S_J parameter – shift of the joint points for shapes.

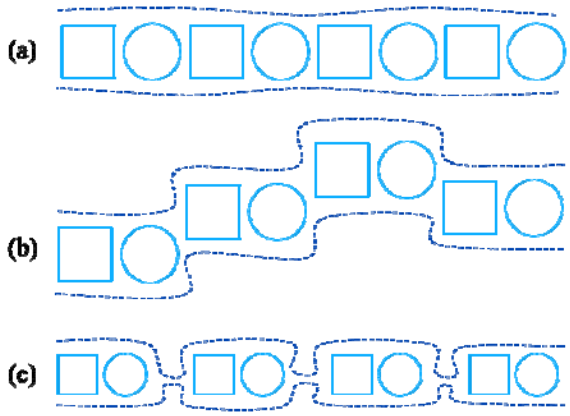

Fig. 7. S_J parameter characteristic: (a) regular cave, (b) non-regular cave in Y axis, (c) non-regular cave in Y or Z axis.

C. *Procedural clouds generation using SG-M concept*

Cloud modeling can be done by summing many spheres in successive rules constrained to specified directions. In our method, we create the base shapes and shape rules that will be used during modeling process (Fig. 8).

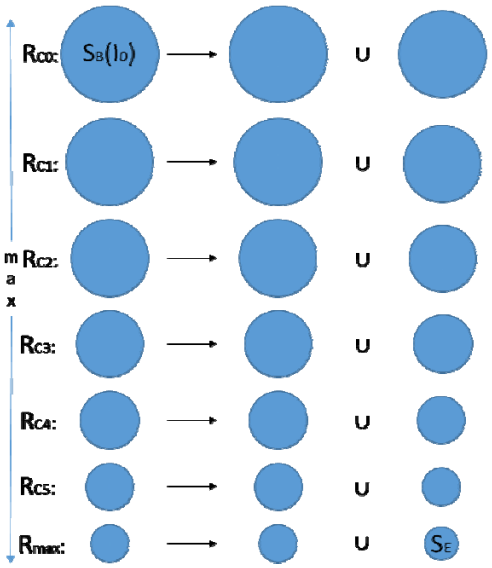

Fig. 8. Rule-creation process using the *Max* parameter for a *specified axis (+X, +Y, +Z, -X, -Y, -Z). S_B – defines shape begin radi*us, S_E – defines shape end radius.

The number of rules is calculated as the maximum of the value *Max* for each direction, i.e. Max(X, -X, Y, -Y, Z, -Z).

For each rule, all *Max* values are reduced by 1. Then, new connections are created for those rules that have a positive value. In the *n-th* rule, n spheres are added to the object. The radius of the sphere in the *n-th* iteration is interpolated between the initial and the final radius. In this step, the amount of required shape rules is determined. For shapes connecting points, some random offset is introduced. After performing all classic rules, we can use the morphic rule to morph our structure with the selected shape (e.g., sphere, torus) (Fig. 9).

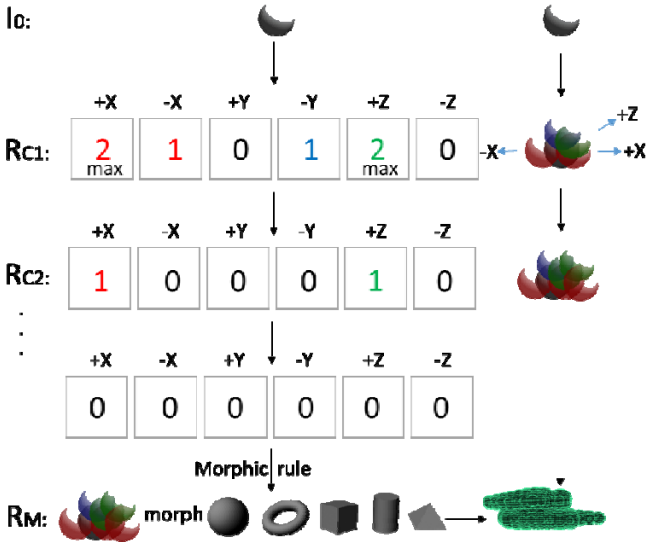

Fig. 9. Cloud-modeling algorithm. Colors show directions of adding spheres.

D. *Procedural architecture generation using SG-M concept*

We assume that the buildings are multistory. We start by setting the first rule for the ground floor (i.e., the initial shape)

and the last rule for last floor. Here, intermediate rules may be created using linear interpolation. For each level, we can determine characteristics such as height, width, and convexity. For example, a floor is a cube that can be convex (i.e., we apply 4 sum rules), concave (i.e., we apply 4 difference rules). After the application of the last rule, we can model the roof shape by morphing from a sharp (i.e., a pyramid) shape to a spherical (i.e., sphere). All modeling directions are selected automatically according to basic building principles, i.e., axes –X/X, -Z/Z are selected to adding/cutting sides, axis +Y is chosen for building up floors and adding a roof (Fig. 10).

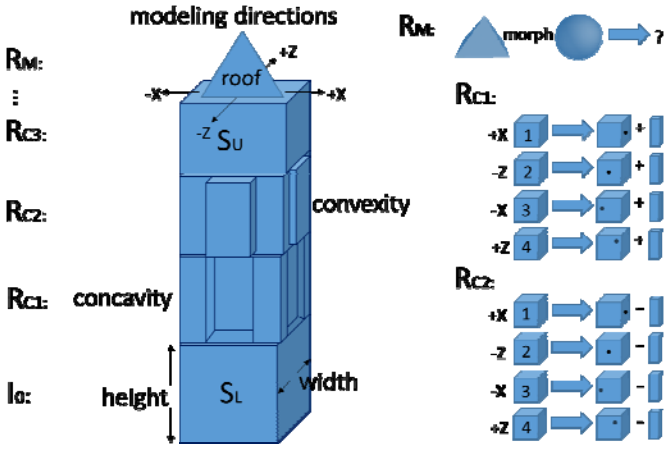

Fig. 10. Multistory architecture construction with possible application rules. S_L – defines the upper floor, S_L – defines the lower floor .

E. Functional Description of Three-Dimensional Shapes

We propose an alternative functional description of shapes. In contrast to classical algorithms, we are not interested in a polygon mesh object, but only in the actual function that describes space where solids are located. The algorithm performs operations only on functions or scalar functions describing the field for selected shapes.

Definition 2. A subset $A \subset \mathrm{R}^n$ *(in our case* R^3*)* is called an *implicit object* if there exists a function $f : U \rightarrow \mathrm{R}^k$, $A \subset U$, and a subset $V \subset \mathrm{R}^k$, such that $A = f^{-1}(V)$[30]:

$$A = \{P \in U : f(P) \in V\} . \tag{9}$$

Definition 3. The *distance* from a point p to a surface M in R^n *(in our case* R^3*)* is the minimum of the Euclidean distance d_E (P, s), where $s \in M$ [30]:

$$d(P, M) = \inf_{s \in S} d_E(P, s) . \tag{10}$$

Definition 4. When f is a real-valued function, that is $k = 1$, then f is a *point-membership classification function* that returns a value according to the relationship of a point $P = (x_1, \ldots, x_n)$, given as its argument, with the implicit object A defined by f [30], i.e.,

$$f(P)\begin{cases} > 0 & \text{then } P \notin A \text{ (point is outside surface)} \\ = 0, & \text{then } P \in A \text{ (point is on surface)} \\ < 0, & \text{then } P \in A \text{ (point is inside surface)} \end{cases} \tag{11}$$

We define primitive shapes (e.g., cube, sphere, torus, cylinder) using standard implicit objects (based on functional description) [30]. For example, a sphere f is given by:

$$f(P) = P.x^2 + P.y^2 + P.z^2 - r , \tag{12}$$

where P is a point in R^3 and r is the sphere's radius.

Definition 5. An implicit CSG solid is defined by any set of points in R^n (in our case R^3) that satisfies $F(x) \leq 0$ for some $F \in S_j$. [30]. An example for a sum operation is given by:

$$f(P) = f \bigcup g = \max(f(P), g(P)) , \tag{13}$$

while a morphing controlled by a morphing parameter a is defined by the following linear interpolation:

$$f(P) = f * g = f(P) * (1 - a) + g(P) * a . \tag{14}$$

Shapes are described by the final function as a composite of the above functions.

F. Joint points, bonds, and directions

An important problem that we had to solve was how to determine the joint points of the grid for the main shapes and for the objects created during modeling process.

Definition 6. The *joint point* J defines the way in which one shape can be combined with another. The joint point is described as a point P and a *direction* D in 3D space

$$J = (P, D) . \tag{15}$$

The *bond* B combines two shapes into one by using appropriately selected joint points. Two bonds make a joint point for overlapping points P and opposite directions D. This can be seen as the "gluing" of two walls facing each other in opposite directions. This operation helps avoid unnecessary connections (Fig. 11). The bond is given by:

$$B = (J_1, J_2) \Leftrightarrow (J_1.P = J_2.P \wedge J_1.D = -J_2.D) , \tag{16}$$

where B is the bond, J_1, J_2 are joining points, P is a point in R^3, and D is a direction for the shape. If the bond does not have a direction, i.e., when $D = (0, 0, 0)$, then a joint point can be created with any other point.

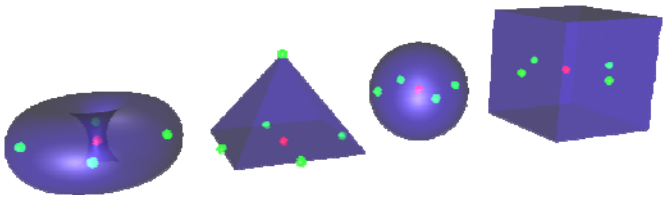

Fig. 11. Joint points for base shapes. Red color shows possibility to join with central point of the shape, green – shows another possible connections.

G. *Grid Display Algorithm – Marching Cubes*

Conceptually, the object surface can be described by a function called a *density function*. For a point $P \in$ 3D, the function produces a single floating-point value. These values can be positive, negative, or zero. The value of the function is positive when P is located inside the solid. If the value is negative, then P is located in the empty space. The boundary between positive and negative values - where the value of the density function is zero - forms the surface of the solid.

We wish to construct a polygonal mesh that spans that surface. We use the GPU to generate polygons for a "block" of structures at a time, but we further subdivide the block into 32x32x32 smaller cells, or voxels. Inside these voxels, we construct polygons (i.e., triangles) that represent the solid surface. The marching-cubes algorithm [31] allows us to generate polygons within a single voxel, given as input the density value at its eight corners. As the output, this algorithm produces anywhere from zero to five polygons. If all the densities at the eight corners of a cell have the same sign, then the cell is entirely inside or outside the solid, so no polygons are defined. In all other cases, the cell lies on the boundary of the solid and one to five polygons are generated. We use space partitioning with cubes. Once we determine that the shape intersects all the boxes, we can read incidental edges, i.e., those colliding with the surface of the shape volume. By using linear interpolation, we can choose exactly the point of intersection of the solid with each edge of the cube. We take the density values at the eight corners and determine whether each value is positive or negative. Each value is assigned one bit in a binary representation. If the density is negative, we set the bit to zero; if the density is positive, we set the bit to one. From the marching-cubes' pre-defined table of intersections (256 combinations), the system reads the triangles that form the points of intersection and displays them. Interpolation determines exactly where a vertex is placed along an edge. The vertex should be placed where the density value is approximately equal to zero. For example, if density at the end A of the edge is 0.1 and at the end B it is -0.3, the vertex is placed at 25 percent in the way from A to B (Fig 8.).

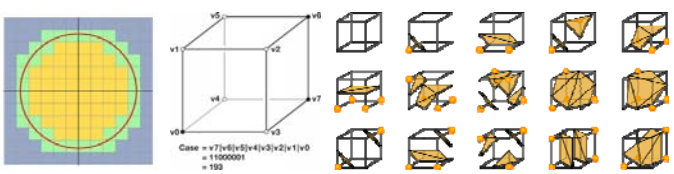

Fig 8. Left – cross sections of cubes intersected by a solid (green color depicts the selected cubes). Middle - a single voxel with known density values at its eight corners. Right - the 15 fundamental cases in *Marching Cubes*.

IV. Results

Figure 13 shows some results obtained using our method.

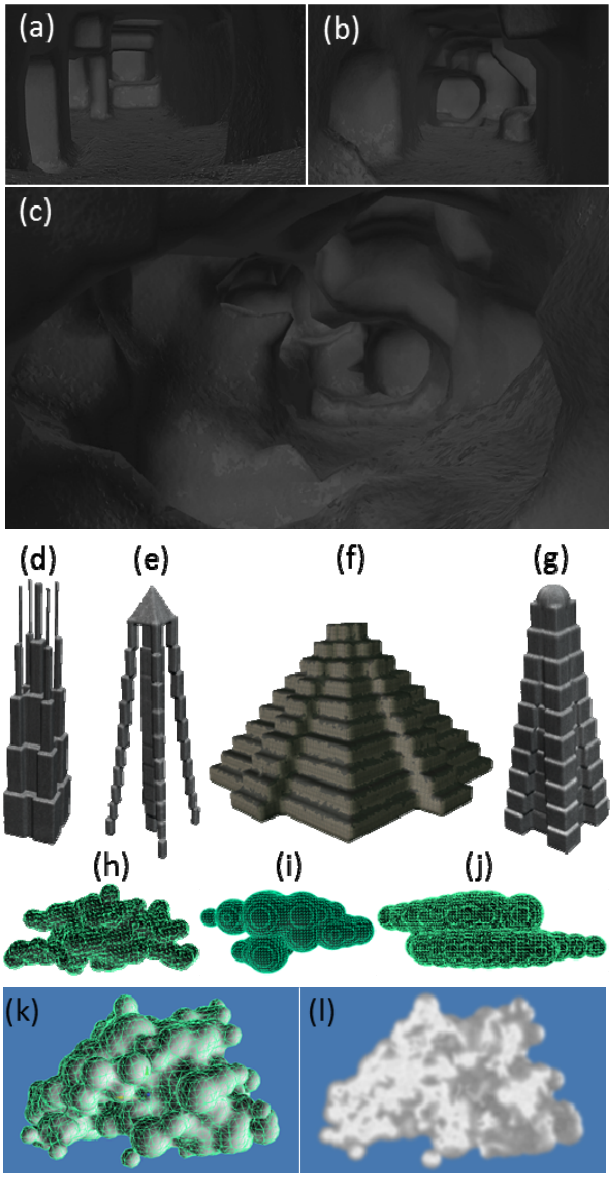

Fig. 12: Possible results (mesh and final renders): (a-c) caves, (d-g) – architecture, (h-l) clouds. For objects in (b), (c), (e), (g), (j), (k), and (l), a morphic rule was applied with a different value of morphing parameter.

The platform used for simulations consisted of - nVidia GeForce GTX 460M GPU, i7-2630QM CPU and 12 GB RAM. Processing times: from 500 ms to 10 sec (caves: 500 ms – 2 sec, buildings: 3 sec - 10 sec, clouds: 1,5 sec – 9 sec)

Summary

Computer graphics systems are a key part of modern information systems. Nowadays, there has been a noticeable trend of applying new methods of modeling 3D objects in

virtual-reality systems. This trend is motivated mainly by market demand from areas such as digital entertainment, simulation for 3D gaming, and 3D VFX industries. Our paper presented an innovative method for real-time procedural modeling of three-dimensional geometry of caves, clouds, and buildings. By adding morphing capabilities to the classical formalism of shape grammars, our method is able to synthesize objects with greater variety and geometric complexity, which are characteristics that highly influence visual realism. In addition to the advantages of shape grammar, the morphing parameter allows us to establish the continuous percentage contribution of two input shapes to produce a single output object, and in turn help fine-tuning control of the modeling process. Our further research will focus on the development of shape rules by providing intelligent expert system to steering rule selection.

REFERENCES

[1] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, Instant architecture. ACM Transactions on Graphics 2003; 22(3), pp. 669–77.

[2] Y. I. Parish, and P. Muller, Procedural modeling of cities. Proceedings (*SIGGRAPH'01*), ACM Press, E. Fiume, 2001, pp. 301–308.

[3] S. Greuter S, J. Parker, N. Stewart, and G. Leach, Real-time procedural generation of pseudo infinite cities. Proceedings (*GRAPHITE'03*), ACM Press, 2003, pp. 87-95.

[4] A. Peytavie, E. Galin, J. Grosjean, and S. Merrilou, Arches: a Framework for Modelling Complex Terrains, Computer Graphics Forum, Proceedings *EUROGRAPHICS 2009*; 28(2), pp. 457-467.

[5] K. Warszawski, and S. Nikiel, A proposition of particle system-based technique for automated terrain surface modeling. In *Proceedings of the 5th International North American Conference on Intelligent Games and Simulation* (Game-On-NA '09), 2009, ISBN 978-9077381-49-6, pp. 17-19.

[6] A. Bouthors, Neyret F. Modelling Clouds Shape, Proceedings EUROGRAPHICS, 2004.

[7] J. Schpok, J. Simons, D. S. Ebert, and C. Hansen, A real-time cloud modeling, rendering, and animation system. Symposium on Computer Animation'03, 2003, pp. 160-166.

[8] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita, A simple, efficient method for realistic animation of clouds. *Proceedings of ACM SIGGRAPH 2000*, 2000, pp. 19-28.

[9] D. S. Ebert, Volumetric procedural implicit functions: A cloud is born. SIGGRAPH 97 Technical Sketches Program, Whitted T., (Ed.), *ACM SIGGRAPH*, Addison Wesley, 1997, ISBN 0-89791-896-7.

[10] P. Elinas, and W. Sturzlinger, Real-time rendering of 3D clouds. Journal of Graphics Tools. 2000; 5(4), pp. 33-45.

[11] T. Nishita, E. Nakamae, and Y. Dobashi, Display of clouds taking into account multiple anisotropic scattering and sky light. SIGGRAPH 96 Conference Proceedings, Rushmeier H., (Ed.), *ACM SIGGRAPH*, Addison Wesley, 1996, pp. 379-386.

[12] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants. Springer-Verlag, 1991: 101–107. ISBN 978-0387972978.

[13] B. A. Am Ende, 3D Mapping of Underwater Caves, *IEEE Computer Graphics Applications,* 2001; 21(2), pp. 14-20.

[14] M. Boggus, Crawfis R. Procedural Creation of 3D Solution Cave Models, Proceedings of the 20th IASTED International Conference on Modelling and Simulation, 2009, pp. 180-186.

[15] M. Boggus, and R. Crawfis, Explicit Generation of 3D Models of Solution Caves for Virtual Environments, *Proceedings of the 2009 International Conference on Computer Graphics and Virtual Reality*, 2009, pp. 85-90.

[16] P. Schuchardt, and D. A. Bowman, The Benefits of Immersion for Spatial Understanding of Complex Underground Cave Systems, *Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology* (VRST '07), 2007, pp. 121-124.

[17] L. Johnson, G. N. Yannakakis, and J. Togelius, Cellular Automata for Real-time Generation of Infinite Cave Levels, *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (PC Games, 10), 2010, pp. 1-4.

[18] J. B. Clempner, A. S. Poznyak, Convergence method, properties and computational complexity for Lyapunov games, *The international journal of Applied Mathematics and Computer Science*, 2011; 21(2), pp. 349-361.

[19] L. J. Di Trapani, and T. Inanc, NTGsim: A graphical user interface and a 3D simulator for nonlinear trajectory generation methodology, *The international journal of Applied Mathematics and Computer*, 2010; 20(2): 305-316.

[20] G. Stiny, and J. Gips, Shape grammars and the generative specification of painting and sculpture. Information Processing 71, *North-Holland Publishing Company*, 1972, pp. 1460-1465.

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

[22] G. Stiny, Introduction to shape and shape grammars. In Environment Planning B. 1980; 7(3), pp. 343–361.

[23] T. Martyn, A new approach to morphing 2D affine IFS fractals. Computers & Graphics 2004; 28, pp. 249-72.

[24] M. Alexa, D. Cohen-Or, and D. Levin, As rigid as possible polygon morphing. Computers Graphics (*SIGGRAPH '2000*) 2000; 34:157-64.

[25] G. Wolberg, Image morphing: a survey. The Visual Computer 1998; 14(8-9), pp. 360-72.

[26] Lazarus F, Verrous A. Three-dimensional metamorphosis: a survey. The Visual Computer 1998; 14(8-9), pp. 373-89.

[27] J. R. Kent, W. E. Carlson, and R. E. Parent, Shape transformation for polyhedral objects. Computer Graphics (*SIGGRAPH '92*) 1992; 26, pp. 47-54.

[28] A. W. F. Lee, D. Dobkin, W. Sweldens, and P. Shroeder. Multiresolution mesh morphing. Computer Graphics (*SIGGRAPH '99*) 1999; 26, pp. 43-6.

[29] G. Turk, J. F. O'Brien, Shape Transformation using variational implicit functions. Computer Graphics (*SIGGRAPH '99*) 1999; 33, pp. 335-42.

[30] L. Velho, J. Gomes, and L. H. Figueiredo, Implicit Objects in Computer Graphics. Springer, 2002, ISBN: 978-0387984247.

[31] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3D surface construction algorithm*, Computer Graphics*, 1987, vol. 21, pp. 163 -169.

Tomasz Zawadzki received his degree and specialization – Software Engineering in Computer Science in 2006 at Faculty of Electrical Engineering and Computer Science, Univeristy of Zielona Góra in Poland. From 2006 he is a PhD student at the same faculty and his interests focused on computer graphics and virtual reality.

Slawomir Nikiel is currently the Professor at the Institute of Control and Computation Engineering, Department of Electrical Technology, Computer Science and Telecommunication, University Of Zielona Góra, Poland. His research interests include virtual reality systems, game programming and multimedia.

Dr. Eraldo Ribeiro is an Associate Professor of Computer Sciences at Florida Institute of Technology. In 2001, he was awarded a Ph.D. degree in Computer Vision in the Department of Computer Science at the University of York, U.K. under the supervision of Professor Edwin R. Hancock. Prior to this, he gained a Master of Science Degree with distinction in Computer Science (Image Processing) at the Federal University of Sao Carlos (UFSCar- SP), Brazil in 1995.