# Procedural Modeling

Lenka Pitonakova
*University of Bedfordshire*
*contact@lenkaspace.net*

## Abstract

*The paper provides an insight into procedural modelling techniques and uses. Procedural modelling is an alternative approach to modelling when high realism and fast real-time rendering of complex objects is desired. The paper first introduces common procedural modelling techniques including creation of fractals and L-systems. Discussion of various approaches to how the models can be created and how they are used by a number of games and software follows.*

## 1. Introduction

Today's modellers are searching for techniques which would automatically create natural or complex objects based on some formal definitions or sets of rules. The need for such techniques comes from current market demands for simple creation of models and fast real-time rendering.

Procedural modelling is an alternative approach to standard modelling using software like 3D Max or Maya. Procedural models are more mathematical descriptions and formulas than meshes. The new modelling approach is a way of creating models without manipulating 3D meshes themselves. Various software tools exist which can read, interpret and visualise given initial primitives. Then they apply rules and use pseudo-random numbers to create complex objects.

Software and games which use the procedural modelling technique are able to introduce new interesting features and detailed, realistic environments never seen before on screen.

## 2. Procedural Modelling Techniques

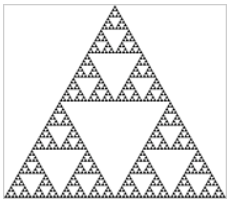
Procedural modelling uses mathematical models and abstract definitions of shapes rather than meshes created by usual modelling techniques. Procedural models are therefore easier to modify since only their formal definitions need to be changed. Most common mathematical structures used are fractals and L-systems.

### 2.1. Fractals

A fractal is "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole" [1], i.e. a self-similar structure [2]. They are similar on any scale, therefore considered as "infinitely complex" [3].

A typical example of a fractal shape is Sierpinski triangle [4] which starts with a single triangle and splits infinitely (Figure 1).

Figure1: Sierpinski triangle [4]



Fractals can be found in nature (clouds, mountains, rivers, coastlines, etc. [5]), can be created using vector iteration programmes (usually to model real-life objects) [3] and are a part of manufactured objects (antennas, fractal drums for absorption of sounds, fibre optics [5]).

### 2.2. L-systems

L-systems or Lindenmayer systems are fractal-like self-similar structures grown from an initial state based on a set of production rules [6], usually represented by strings. Similarly to fractals, their structure is the same both on a large and small scale [2].

L-systems are usually defined by the following components [6]:
**Alphabet:** a finite set V of formal symbols, usually represented as letters of the alphabet (a,b,c, etc).
**Axiom:** a string ω from the set V also denoted as V*. V* represents words such as aab, acbb, etc.

**Productions:** rules of mapping letters from V to V* denoted as

```
p: α --> ω
```

If there is no rule defined for a letter α from the set V, it is mapped to itself and called a constant.
An L-system is therefore a *tuple* which can be formerly represented as

```
G = {V,S, ω, P}
```

where S is a set of constants [2].

There are a number of uses of L-systems, especially for growing artificial plants. The rules are applied iteratively to an initial string which can be interpreted graphically by a computer programme. Figure 2 shows an artificial plant grown using the following definition of an L-system [2]:

```
variables: X F
constants: + −
start: X
rules: (X → F-[[X]+X]+F[+FX]-
       X), (F → FF)
angle: 25°
```

where F means "draw forward", - means "turn left 25°", and + means "turn right 25°".

Figure 2: artificial plant created as an L-system [2]

Simpler grammar can be used to get the Fibonacci numbers where a number in the set is a sum of string's letters in each iteration [2]:

```
variables : A B
constants : none
start  : A
rules  : (A → B), (B → AB)
```

This L-system produces the following sequence of strings:

n = 0 : A
n = 1 : B
n = 2 : AB
n = 3 : BAB
n = 4 : ABBAB
n = 5 : BABABBAB
n = 6 : ABBABBABABBAB
n = 7 : BABABBABABBABBABABBAB

The result obtained after 7 iterations is 1 1 2 3 5 8 13 21.

One can control growth of the string by simply altering the production rules. This provides great advantages as a modelling technique, since the graphical representation itself is done automatically by a software which uses the production grammar.

The results of using L-systems show that graphics produced are very similar to natural objects, such as the plant mentioned above. Moreover, a number of variations can be created by simply randomizing a small part of the production rule or the initial string.

## 3. Procedural Modelling of Urban Areas

Estate areas, industrial parks or large cityscapes all appear both in modern movies and computer games. Both industries have a common need to create large amount of interesting structures that can be quickly adjusted and rendered. Building cities using current 3D modelling software like 3DS Max or Maya is difficult. They provide great tools for creating the most realistic meshes with good-looking textures. However, it is hard and very time-consuming to model large areas with diversity of object, since each mesh has to be made and manipulated with by hand. Therefore, an alternative approach needs to be taken where the modeller can specify parameters and common features of objects while the actual models are created automatically and with randomised diversity.

CityEngine [7] is the leading tool on the software market [8] which makes city creation much faster. It is quite easy to learn to work with and it can generate city maps based on street shape patterns, grow buildings between the streets and provide other 3D creation tools as well as rendering engines with usable models.

### 3.1. Creating Streets

The first step in creating a city is to define suitable areas for streets and specify the city height map. To do so, CityEngine implements elevation, land/water and population density maps. These are all layers which help the street generation algorithms decide where and how to lay out the street junctions. These maps can be

based on real cities or created from scratch. Figure 3 shows examples of such maps.

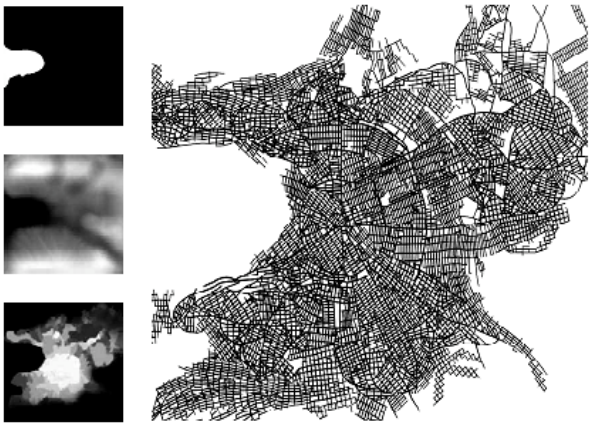


Figure 3: Left column: Water, elevation and population density maps of an imaginary virtual city of 20 miles diameter. Right: One possible roadmap generated from this input data. [8, p. 302]

The second step is to generate street networks. There is an interface dialogue where you can specify the area, density and shape of street network. A number of network patterns have been identified during the research in this field [8, 9]. CityEngine lets you choose one of the following for minor streets and major highways: *rectangular raster* (similar to New York), *radial to centre* (similar to Paris) and *branching* (with no superimposed pattern). Once all the parameters have been specified, the whole network is created by a simple click of a button. The streets have greater concentration in areas with high population density, avoid water while growing along coasts and snap to the terrain elevation (Figure 3). The creation process itself is based on L-systems and uses a fast trial-and-error method to provide streets shapes and connections.

### 3.2. Creating Buildings

After the streets have been generated, user can create structures between them. In City Engine, Each building stands on a *lot*. Lots are created automatically and can be subdivided to provide bases for extra buildings.

The procedural approach to modelling proves itself useful again when actual models of buildings are created. Instead of building the meshes by hand, user can write a CGA (Computer Generated Architecture) script which specifies how to 'grow' a building from its initial lot. Each lot therefore needs to have a script associated with it. Shape grammars were initially introduced by architect Stiny in 1971 as a tool for designing and analysing architecture [10].

CGA uses Shape trees in order to extract new geometry from the initial shape of a building lot. [11] Branches of a shape tree are created by applying one or more of the provided shape manipulation functions to a predecessor shape: scope transformation (scaling, rotation, translation), texture UV coordinates speciation, addition of geometry (e.g. attaching an imported model of a window, roof, etc.), extrusion, component split (which helps identify various parts of an object, e.g. front wall or roof) and subdivision split (which splits initial object into more, e.g. to create separate floors of a building). Complex buildings can be created by applying a pre-defined set of building rules on the initial lot and using a number of iterations to manipulate new shapes generated at each step.

Usually only a small amount (3 - 6) of CGA scripts is needed to create a city with a great diversity [12]. While one rule is applied on a number of lots, each building is slightly different thanks to randomness which can be incorporated when specifying attributes like height, number of floors, walls textures or used models of doors and windows. What's more, editing of the generated models is easy as well: user only needs to adjust one CGA file and reapply the rules to the selected slots. A new set of buildings can be created almost immediately.

## 4. Modelling of Natural Phenomena

The problem for urban modelling seems to be solved by CityEngine which is a great tool for creating naturally-looking streets as well as CGA-based models of buildings. However, there are more complex problems today's modellers encounter – modelling and animating natural phenomena. Everyone can very quickly recognize whether an image of a natural thing like a flower or water surface was shot in a real environment or created on a computer. While we can all *distinguish* between what is natural and what is artificial, very few yet attempted to *formally define* the term. The following section provides an insight into such attempts, specifically for modelling of cracks and fractures on 3D models and creating the ocean surface.

### 4.1. Cracks and Fractures

Modelling of cracks on a given model or a surface could be done simply by using a graphic editor like Adobe Photoshop of Adobe Fireworks. Cracks would have to be created by hand and have the emboss or bevel method applied to them to look as if a part of the texture. However, editing the shapes would be a lengthy process and creating original patterns on

hundreds of object seems almost impossible in human terms. Therefore, a tool for automating the cracks creation is required. The existing physically based techniques are computationally demanding and lack control over crack and fracture propagation. [13] Work done by Martinet et al. [13] addresses the problem and provides an effective solution based on procedural modelling.

Figure 4 shows the proposed simulation cycle of the cracking process. It starts by defining a cracking pattern in form of a graph which describes cracks branching features as well as their geometry. A designer then maps one point of the graph onto a surface and the whole graph is afterwards automatically transformed into a 3D skeleton.

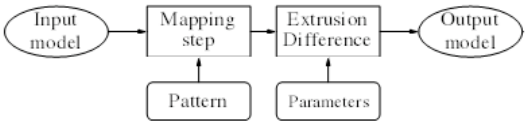



Figure 4 - simulation cycle of the cracking process [13, p. 2]

The automatic process can be managed via parameters like width and depth of cracks, their directions and angles between them. All the parameters are stored in the initial crack pattern graph. The results (Figure 5) seem very promising – the system is able to generate cracks which are very realistic in comparison to real objects.

Figure 5: a real vase (left) and a created model (right) [13, p. 2]

Creating fractures out of a given model is a more complex problem. Cracks are only visible on the surface of an object, therefore only its texture needs to be manipulated. However, breaking an object into smaller pieces involves manipulation with its 3D mesh. As with the previous case, a designer first creates a pattern of fractures called *fracture mask*. Fracture masks define the profile of fracture between two fragments. Since a mask is applied repetitively to smaller and smaller parts (note similarity to fractals), it defines both overall large scale pattern of cracks as well as small details which make fractures surfaces look smooth or rough. Afterwards, *fracture regions* are defined. These are volumes which define how an object is split into two parts, i.e. where cracks are being created.

The creation of fragments is controlled by an automated algorithm with two basic parameters: ΔV which characterises distribution of fragments sizes and α that defines whether fragments should be long thin shards or roughly round pieces (Figure 6).

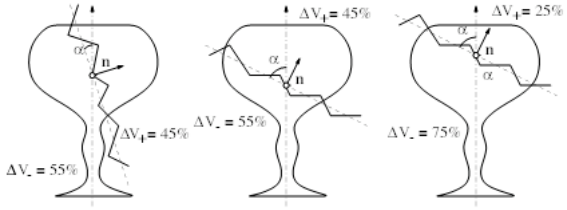



Figure 6: controlling the shape of fragments [13, p. 3]

At every step of the algorithm location and orientation of a fracture mask is selected randomly and adjusted so that the result matches with the two given parameters. Fracturing an object is a matter of converting the initial object into a point cloud representation. This is a pre-processing step which happens only once. The object is sampled using an octree decomposition and on each level of the octree, as many points as needed are created. Points are than classified as inside and outside of the mask to compute the volume of fragments.

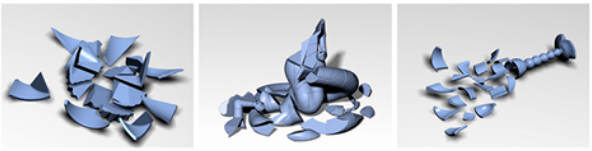


Figure 7: examples of fractured models [13, p. 1]

Figure 7 shows creation of fractures from various initial models. The times needed for generating the first set of fractures (18 and 48 pieces) were almost identical and didn't exceed 10 seconds. About 70 seconds were needed to produce 128 fractures. This implies that the time grows exponentially with the amount of created pieces.

### 4.2. Creating Realistic Ocean Surface Animation

When creating a realistic ocean surface animation, a number of factors need to be considered including time of the day and lighting, temperature and wind conditions. A number of approaches have been taken to solve this problem. Max's wave model uses approximate equations to simulate low-amplitude ocean waves [14]. The surface is quite periodic though which is a downside [15] because the result is not very realistic. Peachy [16] and Fournier et al. [17] took a particle-system approach where the particles form the surface of the ocean and perform circular or elliptical orbits in order to simulate wave breaks. However,

particle systems usually need more CPU power so they are not really an option for creating large virtual objects which need to be rendered in real-time.

The work of Pozzer and Pelegrino [15] presents a solution where the ocean surface is represented by a flat plane and bump mapping and noise functions are used to alter the surface's normal vector. The process uses a 2D matrix of pseudo-random numbers to influence pattern repetitions on the surface. The matrix is understood as a height field of the surface. The noise function defines the matrix values based on desired qualities of the surface: wave perturbation, speed and shape of waves. The algorithm is designed so that the values are not completely random but follow a pattern (Figure 8 c). Rather than calculating a height value for each point of the matrix individually, a common value is generated for a control point and applied to an associated block of matrix slots. The following algorithm is used:

```
value  = random()*perturb
length = random()*multiplicity
for count = 0 to length
   texture[x+count][y] = value
```

This produces textures with long waves. To make it more realistic, two texture layers are used and evaluated in different scales (Figure 8 d). This allows controlling the perturbation degree without need to generate new control points.

To produce smoothly looking surfaces, B-spline interpolation is used (Figure 8 a,c,d). However, great processing time is needed to create such a texture and this method is only used for texturing areas far from the observer.

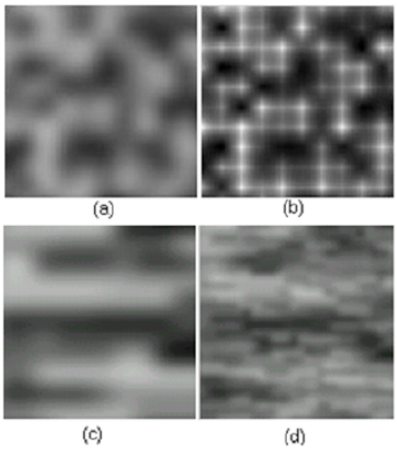



Figure 8: Texture examples: (a) random values and B-spline interpolation; (b) exponential interpolation; (c) multiplicity = 6; (d) two texture layers [15]

Animating a created ocean surface requires a method of manipulating the texture. Simply creating a new texture each frame would not produce realistically looking animation because frames would not be associated with each other. To overcome this problem, a 3rd dimension (Z) has been introduced to the texture matrix. This dimension represents movement of the shaded plane inside the 2D texture. Each element of this plane is calculated by interpolating four neighbouring control points (2 above and 2 below) located on the solid texture along the Z-axis whose X and Y coordinates are equal to the point of the plane being computed. The animation is than created by changing the X-Y plane texture over the time (along the Z-axis).

To control such an animation, three parameters can be used: X and Y for wave displacement and Z for shape change (the longer the z-axis, the more surface changes over time).

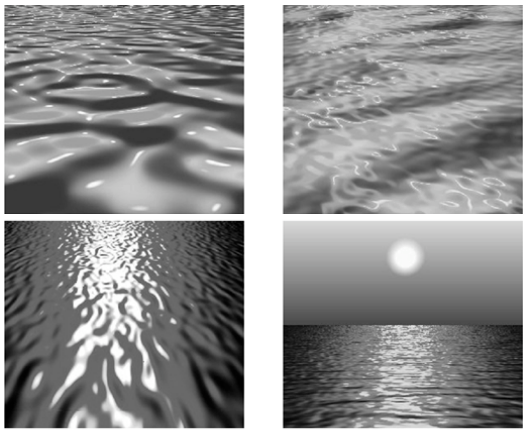


Figure 9: examples of created ocean surface [15]

To complete the animation, some lighting processing needs to be done. Diffuse and specular reflections are used to model the light. The initial water colour is combined with the diffuse reflection value and the source colour is multiplied by the specular reflection value. The two colours are added together on each point of the final surface matrix. This produces some realistically-looking results shown in Figure 9.

## 4. Procedural Modelling of Natural Phenomena Motion

After creating a procedural model of a natural phenomena (e.g. trees, wind, water) we usually end with a great number of primitives which a model consists of. If pseudo-random numbers are used to create these primitives, traditional key framing technique for creating motion would not be a suitable solution because an animator would need to manipulate each primitive [18].

Green and Sun [18] introduced *motion verbs* as a higher-level script-driven approach to modelling motion of such a great number of objects. Motion verbs are applied to each of the particles of a given type during the animations. The motion processes are generated at the same time as the primitives, which enables them communicate with the processes of connected primitives. Green and Sun mention an example where motion of tree branches is connected to the motion of their parent and sub branches. A motion process first determines states of the neighbour branches and then computes the motion.

### 4.1. The MML language

The MML language was introduced to encode the motion verbs into a working programme. The authors also provided an interactive interface for creating MML – based motion which uses a primitives model and another interface for experimenting with MML models.
The structure of a motion modelling script is the following:

1. *Primitives section* which defines names of primitives and the list of their attribute names and data types.
2. *Generate section* which describes how individual generations of primitives are created. The implementation is used on a rule-based model of the generation process.
3. *Motion section* where motion verbs are declared. Each motion verb has parameters and functions. Minimum and maximum possible values of each parameter (attribute) are defined. The motion functions are associated with certain particle names and their bodies are filled with C statements.
4. *Render section* which describes how the primitive are displayed. It is possible to define actions executed in the beginning and in the end of each frame using statements in C. These statements are associated with names of primitives and are fully responsible for their rendering.

An example of MML- based motion is a fountain where the falling water is a particle system (Figure 10). There are two types of particles: one for water and another one for the fountain source. To save the processing time, the particles which hit the ground or reach end of their life time for any other reason are deleted.

Figure 10: artificial fountain

A rather significant issue with the current system is that it doesn't allow for more motion verbs to be applied to a particle at the same time. In the current implementation version the motion processes are executed one after each other but are not able to communicate (e.g. if a branch was moving because of the wind and some other outer force, a realistically looking motion would be a combination of the two forces). Further research needs to be done in order to solve this problem.

## 5. Procedural Modelling in Practice

Procedural; modelling is used in games to generate large 3D environments in time shorter than required by usual modelling techniques. New games like Star Trek online tend to move towards this way of representing objects which allows for creation of distinctive and strange sights all over the game's universe [19].

An experimental game called Charbitat [20] uses procedural modelling and pre-modelled sample objects to create the world during the game play, based on player's actions. The game's goal is exploration and generation of space in the mind of a Chinese princess who fell into coma. The new modelling approach provides scalable, continuous and unique world for every game played.

MojoWorld developed by Pandromeda is another example of how procedural modelling can be used. It is a software where user can create their own planets based on a number of parameters like scale, atmosphere thickness, distance from the nearest star, level of gases in the atmosphere, terrain elevation, water distribution, etc. [21] Although MojoWorld is quite hungry for graphic card memory, it is a useful tool for creating interesting personalised environments.

The examples above show that software and games where procedural modelling is used introduced new interesting features and faster real-time rendering. The latter is especially necessary for online multiplayer games like Star Trek Online. These advantages are probably a trade-off for a more natural approach to creating 3D models.

## 6. Summary

In this paper I discussed various procedural modelling techniques and uses. First I provided an

overview of basic terms including fractals and L-systems. Then I introduced CityEngine, a leading software tool for creating procedural cities in a relatively short time. I also looked at modelling and animation of natural phenomena, namely generation of cracks on a surface, fractures creation and ocean surface modelling and animation. The following section provided information about the MML language, a grammar used for applying motion to procedural models with a high amount of primitives. In the end a number of games and software which uses procedural modelling was mentioned along with interesting features which procedural modelling allowed for.

## 7. Conclusion

Procedural modelling proves to be a much more efficient technique for creating complex structures and natural phenomena than usual modelling techniques. It provides control over the shapes and creation of primitives, while leaving space for randomness and variation. Procedural models can be used in science to represent complex structures or in games and movies for fast real-time rendering of very realistic models.

Providing that further research is undertaken in the field of formal definition of the things around us, there is no doubt that procedural modelling will not only provide more and more realistic and life-like models but also help us understand the way nature works and creates.

The fact that we can mathematically define real objects is fascinating - it proves that complexity in nature comes from iterated simplicity of its small elements. Fractals and other self-similar structures can be found all around us. Procedural modeller needs to find the simplicity in order to create real-life structures. If we proceed with and extend this way of thinking further, we will be able to understand objects and processes in the nature and maybe define the nature itself as a set of rules applied to the basic particles which form everything we see around us.

## 8. References

[1] Mandelbrot, B.B. *The Fractal Geometry of Nature*. W.H. Freeman and Company. 1982.

[2] Wikipedia, L-systems: http://en.wikipedia.org/wiki/L-system

[3] Wikipedia, Fractals: http://en.wikipedia.org/wiki/Fractal

[4] Wikipedia, Sierpinski triangle: http://en.wikipedia.org/wiki/Sierpinski_triangle

[5] Yale University, Fractals: http://classes.yale.edu/fractals/

[6] Oklahoma State University, L-systems: http://www.math.okstate.edu/mathdept/dynamics/lecnotes/node13.html#SECTION00041000000000000000

[7] Procedural Website: http://www.procedural.com/

[8] Y. I H. Parish and P. Müller. Procedural Modelling of Cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, August 2001, p.301-308

[9] K. Dietrich, M. Rotach, E. Boppart. *Strassenprojektierung.* Zurich 1993.

[10] G. Stiny and J. Gips, *"Shape Grammars and the Generative Specification of Painting and Sculpture"*, Proc. IFIP Congress 71 , North-Holland, 1972, pp. 1460 –1465.

[11] CityEngine Help Documentation: http://www.procedural.com:9099/help/index.jsp?topic=/com.procedural.cityengine.help/html/toc.html

[12] B. Watson, P. Müller, et al. Procedural Urban Modeling in Practice. *IEEE Computer Society*, May/June 2008, p. 18-26

[13] A. Martinet et al. Procedural Modeling of Cracks and Fractures. *IEEE Computer Society,* SMI'04, p. 1-4

[14] N.L. Max. Vectorized Procedural Models for Natural Terrains: Waves and Islands in the Sunset. *Computer Graphics*, v.15, n.3, Aug. 1981, pp. 317-324.

[15] C. T. Pozzer and S. R. M. Pellegrino. Procedural Models on Image Synthesis for Ocean Animation. *IEEE Computer Society,* SIBGRAPI'01

[16] D.R. Peachey. Modeling Waves and Surfaces. *Computer Graphics*, v.20, n.4, Aug. 1986, pp. 65-74.

[17] A. Fournier, W.T. Reeves. A Simple Model of Ocean Waves. *Computer Graphics*, v.20, n.4, 1986, pp. 75-84.

[18] M. Green and H. Sun. A Language and System for Procedural Modeling and Motion. *IEEE Computer Graphics & Applications,* Nov 2008, p. 52-64

[19] S. Hogarty. Star Trek Online. *PC Zone*, June 2009, p. 42-45

[20] M. Nitsche et al. *Designing Procedural Game Spaces: A Case Study*, Georgia Institute of Technology, Atlanta

[21] MojoWorld, Pandromeda: http://www.pandromeda.com/products/mojoworldstandard.php