Professional paper http://doi.org/10.24867/JGED-2016-2-025

A method for generating stochastic 3D tree models with Python in Autodesk Maya

ABSTRACT

This paper introduces a method for generating 3D tree models using stochastic L-systems with stochastic parameters and Perlin noise. L-system is the most popular method for plant modeling and Perlin noise is extensively used for generating high detailed textures. Our approach is probabilistic. L-systems with a random choice of parameters can represent observed objects quite well and they are used for modeling and generating realistic plants. Textures and normal maps are generated with combinations of Perlin noises what make these trees completely unique. Script for generating these trees, textures and normal maps is written with Python/PyMEL/NumPy in Autodesk Maya.

Nemanja Stojanović

RT-RK, Novi Sad, Serbia

Corresponding author:
Nemanja Stojanović
nemanja.stojanovic007@gmail.com

First recieved: 18.09.2016.
Accepted: 21.11.2016.

KEY WORDS
L-system, Perlin Noise, Procedural, Modeling

Introduction

Two main reasons for this research are:

1. Popular software for generating L-system trees like L-Studio and xLinden use small amount of parameters that can be changed. The idea was to write scrypt in one of the most popular software for 3D modeling with as many variables as possible for tweaking our result.
2. Autodesk Maya makes it possible to modify models even further and to export them in all possible formats used in 3D.

3Modeling natural objects with a computer have been a very difficult task for decades. Modeling 3D trees from the real world is very difficult due to their variations and complex geometry. Plant modeling requires a combination of biological knowledge, mathematical formalism and computer graphics programming.

There has been a great deal of research on modeling trees, predominantly using procedural approaches (Lindenmayer, 1990) and reconstruction approach, mostly by photographs (Reche-Martinez, Martin & Drettakis, 2004). Complexity and visual appearance have been enhanced over the years in such a way that today many tree models appear photo-realistic to us.

Many approaches have been proposed to model plants and trees, and they can be roughly classified as either rule-based or image-based.

Image-based methods. Image-based methods directly model the plant using image samples. Models generated by these approaches are only approximate and have limited realism.

Rule-based methods. (Prusinkiewicz, James & Měch, 1994) developed an idea of the generative L-system. Rule-based techniques make use of a set of generative rules or grammar to create branches and leaves. Plant models in computer graphics are commonly created using procedural methods, which generated branching structures with a limited user input. Our approach is rule-based.

Modeling in three dimensions

For modeling in three dimensions turtle algorithm is used. Three vectors $\vec{H}$, $\vec{L}$, $\vec{U}$ indicate the turtle's heading, the direction to the left, and the direction up.

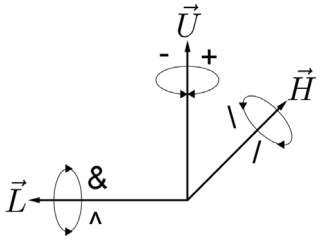
These vectors have unit length, are perpendicular to each other, and satisfy the equation $\vec{H} \times \vec{L} = \vec{U}$ (Prusinkiewicz, 1986; Prusinkiewicz & Lindenmayer, 1996). Rotation of the turtle is expressed by the equation:

$$\left[\vec{H}' \quad \vec{L}' \quad \vec{U}'\right] = \left[\vec{H} \quad \vec{L} \quad \vec{U}\right] R \tag{1}$$

is rotation matrix with dimensions 3x3. It represents rotation by angle α around the vectors: $\vec{U}$, $\vec{L}$ i $\vec{H}$:

$$R_U(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad R_L(\alpha) = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

$$R_L(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$



» **Figure 1:** *Controlling the turtle in three dimensions*

The following symbols control turtle orientation in space:

F(α)	Move forward a step of length α > 0.
f (α)	Move forward a step of length α without creating a branch.
+ (α)	Rotate around by an angle of α degrees.
- (α)	Rotate around by an angle of -α degrees.
& (α)	Rotate around by an angle of α degrees.
^(α)	Rotate around by an angle of -α degrees.
/F(α)	Rotate around by an angle of α degrees.
\(α)	Rotate around by an angle of -α degrees.
[	Stores information about turtle's position and orientation in an array (branch vectors and rotation angle).
]	Restore information from last position in an array.

Modeling of Trees

All trees generated by the same deterministic L-system are identical. In order to prevent this artificial regularity it is necessary to introduce variations that will preserve the general aspects of a tree but will modify its details. If the same L-system was used again, with different seed values for the random number generator, a variation of this image would be generated. The geometric parameters, such as the length and diameter of an internode, as well as branching angles, are calculated according to stochastic laws. Width of branches in every iteration is equal to width of the branches in previous iteration multiplied by factor $w_r = 0.707$. This constant satisfies a postulate by Leonardo da Vinci, according to which "all the branches of a tree at every stage of its height when put together are equal in thickness to the trunk below them."

Perlin Noise

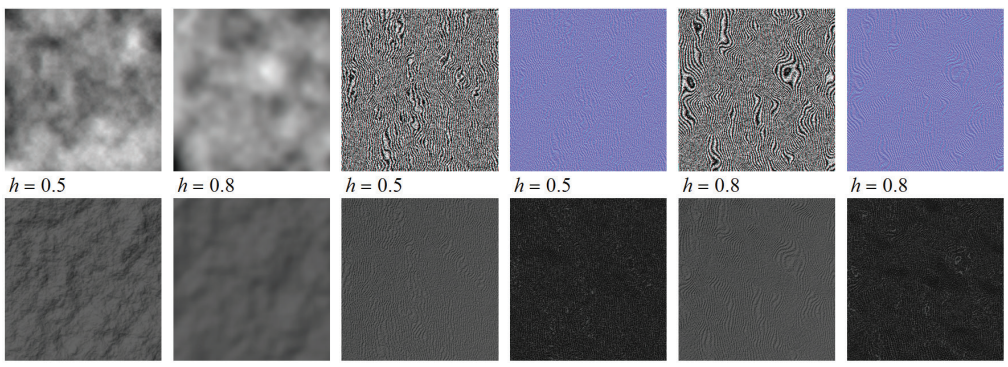
Noise is the random number generator of computer graphics. It is useful wherever there is a need for a source of extensive detail. Since its introduction more than two decades ago, Perlin noise has found wide use in graphics. Perlin noise is generated with fractal summation of basic noise functions. Each noise is multiplied by a weight controlling its contribution to the final result (Ward 1991; Perlin, 2002). This idea was introduced by Perlin. The final pattern can be presented as:

$$n_p(x,y) = \sum_{i=0}^{k} \frac{n_i(x, y, p \cdot (1/a)^i)}{b^i}, \quad p \in \mathbb{N}^+ \tag{2}$$

$$n_p(x,y) = \sum_{i=0}^{k} n_i(x, y, p \cdot H^i) \cdot H^i, \quad p \in \mathbb{N}^+ \tag{3}$$

$$n_p(x,y) = \sum_{i=0}^{k} n_i(x, y, p \cdot (1/lac)^i) \cdot per^i, \quad p \in \mathbb{N}^+ \tag{4}$$

Where $n_p(x,y)$ is Perlin noise and $n_i(x,y)$ is basic noise i. Parameter α defines how irregular will the generated



» **Figure 2:** *Perlin noise and its modifications. In bottom row we can see images of plane with applied textures from first row*

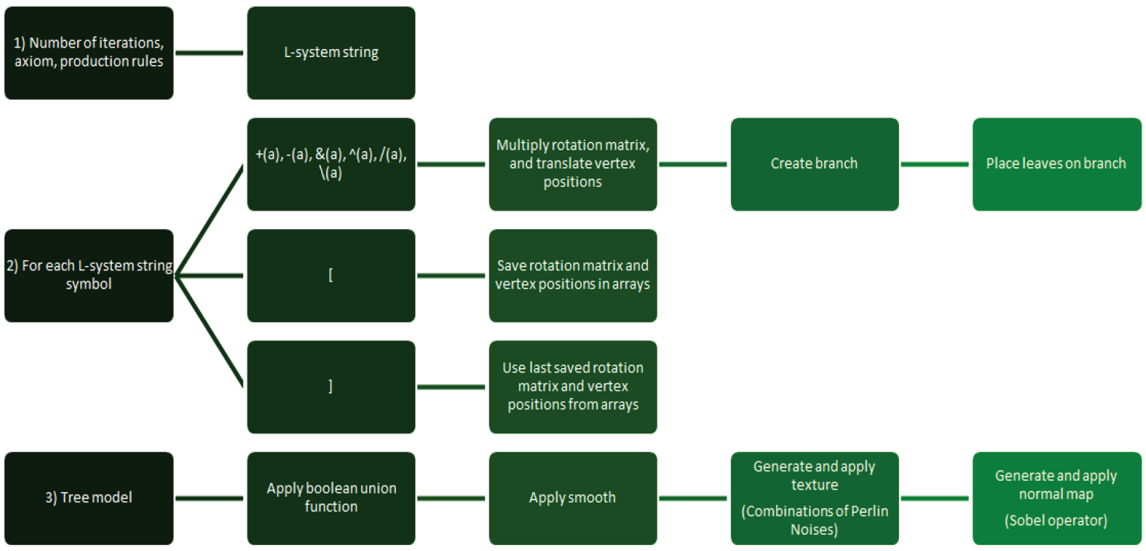
noise be. In practise parameters *a* and *b* are usually equal and it is common that they are both equal to 2. In this case, when a and b are equal, we can replace them with *1/H*, where *H* can be treated as Hurst parameter in fractional Brownian surfaces (Mishura, 2008). Each noise $n_i(x,y)$ is called octave and Perlin noise is commonly formed by twofold decrease in amplitude and twofold increase in frequency of these octave noises. Frequency is input parameter in our noise functions and it is calculated as $1 / (p \cdot (1 / a)^i)$ or $1 / (p \cdot H^i)$, which means it will increase by *1/H* for each octave. This value *1/H*, with which frequency is multiplied, is called lacunarity. On the other hand, amplitude has a constant value in first noise function and it is decreased by dividing whole noise function with b^i or H^i. This value *H* with which we decrease our amplitude is called persistance. Typically *k* has value between 5 and 10 and in our case *k* is equal to 7.
Images in the first row of Figure 2 are created with Python/PyOpenGL, and images in second row represent render of XY plane with textures from above applied on them, in Autodesk Maya. Images 1, 2, 3, and 5 are used as heightmaps that generate a terrain, while images 4 and 6 represent applied normal map on a plane.

Images 3 and 5 in Figure 2 are created by modifying Perlin Noise with sin wave. This can be seen in next pseudo code:

```
for y in range(0, imageWidth):
   for x in range(0, imageHeight):
      noise1 = noise(x*scale, y*scale, seed, 100)
      noise2 = noise(x*scale, y*scale, seed, 100 * H)
                        ...
      noise7 = noise(x*scale, y*scale, seed, 100 * (H^6))
      #Perlin Noise - Figure 2, image 1 and image 2
      int k = int(noise1) + int(noise2 * H) + int(noise3 * (H^2))
            + int(noise4 * (H^3)) + int(noise5 * (H^4)) + int(noise6 * (H^5))
            + int(noise7 * (H^6))
      # 1) - Perlin Noise in RGB values - Figure 2, image 1 and image 2
      data[x][y][0] = k
      data[x][y][1] = k
      data[x][y][2] = k
      # 2) - Perlin Noise in RGB values - Figure 2, image 3 and image 5
      data[x][y][0] = sin(x*x + k*k) + sin(y + k)
      data[x][y][1] = sin(x*x + k*k) + sin(y + k)
      data[x][y][2] = sin(x*x + k*k) + sin(y + k)
```

As can be seen, modifying Perlin noise images is quite easy. Combining different images generated with Perlin noise, doesn't require much effort either. For generating normal maps from textures Sobel operator is used. Source code for noise function as well as more in depth explanation about combinations and modifications of Perlin noise was covered in our previous research (Stojanović, 2016).

Algorithm



» **Figure 3:** *Algorithm for generating 3D tree models in Autodesk Maya*

To generate L-system tree model we first have to create L-system string with defined axiom, distribution rules, probabilities for each distribution (if we use stochastic L-systems) and number of iterations. After the L-system string is created we go through every string symbol, as shows in step 2 of Figure 3, and generate our branches. Idea of generating 3D tree model after L-system string is done is to use two vectors that change with every branch generated. First we start with vector one in (0,0,0), and vector two with (0,*y*,0) (Height of the tree trunk is *y*).

After the first branch is created, in this case tree trunk, first vector becomes second and next calculations with transformation matrix are applied on vector two, so we again create branch between vectors one and two.

Every branch should have smaller and smaller radius, so we generate our model with truncated cones rather than cylinders. In case of symbols [or] we save our vectors in a list. For every next symbol [we append our list, and for every symbol] last elements of a list are deleted and we use new ending elements for further calculations.

After the L-system tree is generated, textures and normal maps are applied. On tree trunk and starting branches Boolean union function is applied, and then the trunk is smoothed. This newly generated trunk won't have any leaves on it.

Boolean union function doesn't work very well in Maya 2012, Maya 2015 and Maya 2017. It tends to hide branches on which it was applied, so script for Boolean union function was written by copying 3Ds Max ProBoolean idea. Results of our program before the creation of leaves is shows on Figure 4.

Next, we have to generate leaves on tree.
Every leaf is generated with usage of random variable *X*, defined like this:

$$X:\begin{pmatrix} A & B & C & D \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

$$A: \quad \vec{b}_p = \vec{a}_p \cdot \frac{1}{4}$$

$$\vec{L}_{1,p} = \vec{b}_p,\ \vec{L}_{2,p} = \vec{b}_p + \vec{a}_p/4,\ \vec{L}_{p,3} = \vec{b}_p + k \cdot \vec{N}_{p+Sx},\ \vec{L}_{p,4} = \vec{b}_p + \vec{a}/4 + k \cdot \vec{N}_{p+Sx},\ k \in \mathbb{R}^+$$

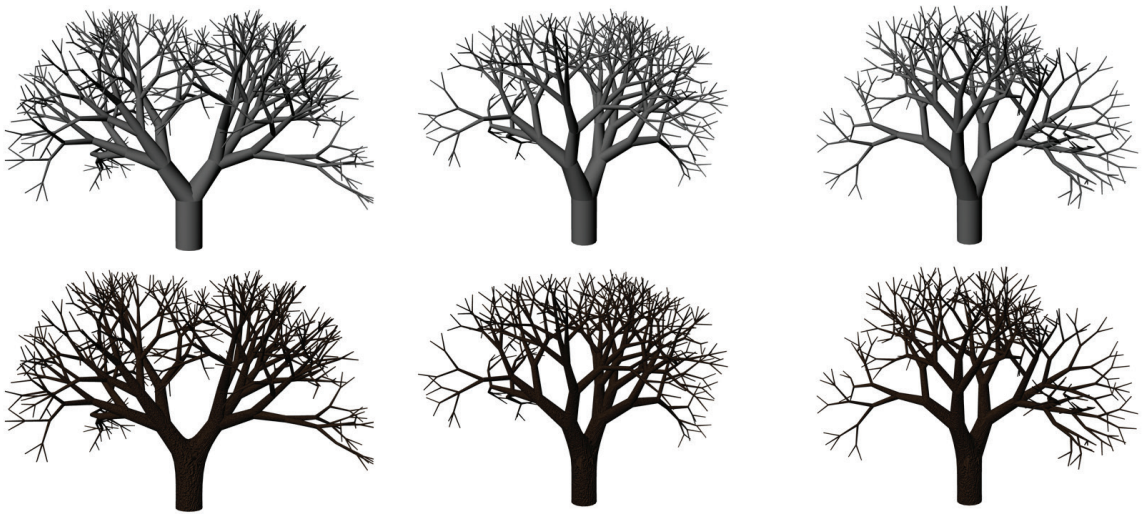
$$B: \quad \vec{b}_p = \vec{a}_p \cdot \frac{1}{2}$$

$$\vec{L}_{1,p} = \vec{b}_p,\ \vec{L}_{2,p} = \vec{b}_p + \vec{a}_p/4,\ \vec{L}_{p,3} = \vec{b}_p + k \cdot \vec{N}_{p+Sx},\ \vec{L}_{p,4} = \vec{b}_p + \vec{a}/4 + k \cdot \vec{N}_{p+Sx},\ k \in \mathbb{R}^+$$

$$C: \quad \vec{b}_p = \vec{a}_p \cdot \frac{3}{4}$$

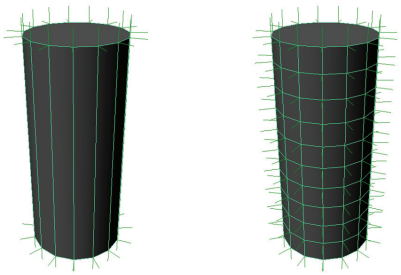
$$\vec{L}_{1,p} = \vec{b}_p,\ \vec{L}_{2,p} = \vec{b}_p + \vec{a}_p/4,\ \vec{L}_{p,3} = \vec{b}_p + k \cdot \vec{N}_{p+Sx},\ \vec{L}_{p,4} = \vec{b}_p + \vec{a}/4 + k \cdot \vec{N}_{p+Sx},\ k \in \mathbb{R}^+$$

$$D: \quad \textit{Don't draw leaf!}$$



» **Figure 4:** *Three examples of trees before and after boolean union function, textures and normal maps*

Where $k \epsilon \mathbb{R}^+$ presents width of the leaf. In our case k is equal to $\vec{a}_p/4$, so width and height of the leaf are equal. $p \epsilon (0,Sx)$ represents edges in cylinder and Sx number of subdivisions of cylinder by x axis. $\vec{a}_p = \vec{V}_{p+Sx} - \vec{V}_p$ is edge vector for every edge p. $\vec{b}_p$ defines starting position of leaf on the edge, and $L_{i,p}$, $i=\{1,2,3,4\}$, defines positions of leaf vertices. N_m, $m \epsilon \{0, 1,...2p-1,2p\}$ defines normal vector in vertex m. Normal vectors on boundary vertices have 2 normals as can be seen on Figure 5, and this feature was used to position leaves facing the new branches with ease.



» **Figure 5:** *Vertex normals on cylinders in Autodesk Maya*

Four examples created with two given L-systems are shown below. Tree models are made of about 4724 (image2) – 6120 (image1) polygons. Subdivision Sx for each cylinder is 20, so amount of polygons can be reduced drastically without any loss of details. Tree generation takes 5 (image2) – 13 (image1) seconds on i5/2.20Ghz/8GB RAM/64 bit windows 10, and low quality maya render is used for images below.

$$n = 7$$
$$\omega : FA$$
$$p_1 : A \xrightarrow{1/3} [\backslash(r)\&(r)FA]/(r)\&(r)FA$$
$$p_2 : A \xrightarrow{1/3} [/(r)\hat{}(r)FA]\backslash(r)\hat{}(r)FA$$
$$p_3 : A \xrightarrow{1/3} [FA][/(r)\hat{}(r)FA]\backslash(r)\hat{}(r)FA$$
$$r = randInt(25,40)\ .$$

r represent random integer, and it is generated every time our programs runs over it.
As can be seen on previous examples, branch lengths depend on number of iteration. Lengths of branches in previous example are 10, 9, 8, 7, 6, 5 and 4, so further changes were made. Length of branches should be stochastic too. Also, idea was to make production's double for every production that will continue to grow our previous branch with a slightly changed angle.
This idea provided more to realism to tree models.
On next example results of our program are shown. Trees are completely stochastically generated.

$$n = 6$$
$$w : F(10)A$$
$$p_1 : A \xrightarrow{3/12} [\backslash(r_1)\&(r_1)/(r_1)\hat{}(r_1)F(10-j+0.5)A][\backslash(r_2)\&(r_2)F(10-j-r_3)A]/(r_2)\&(r_2)F(10-j-r_3)A$$
$$p_2 : A \xrightarrow{1/12} [\backslash(r_2)\&(r_2)F(10-j-r_3)A]/(r_2)\&(r_2)F(10-j-r_3)A$$
$$p_3 : A \xrightarrow{3/12} [\backslash(r_1)\&(r_1)/(r_1)\hat{}(r_1)F(10-j+0.5)A][/(r_2)\hat{}(r_2)F(10-j-r_3)A]\backslash(r_2)\hat{}(r_2)F(10-j-r_3)A$$
$$p_4 : A \xrightarrow{1/12} [/(r_2)\hat{}(r_2)F(10-j-r_3)A]\backslash(r_2)\hat{}(r_2)F(10-j-r_3)A$$
$$p_5 : A \xrightarrow{3/12} [\backslash(r_1)\&(r_1)/(r_1)\hat{}(r_1)F(10-j+0.5)A][/(r_2)\hat{}(r_2)F(10-j-r_3)A]/(r_2)\&(r_2)F(10-j-r_3)A$$
$$p_6 : A \xrightarrow{1/12} [F(10-j-r_3)A][/(r_2)\hat{}(r_2)F(10-j-r_3)A]/(r_2)\hat{}(r_2)F(10-j-r_3)A$$
$$r_1 = randInt(1,8) \quad r_2 = randInt(25,40) \quad r_3 = randFloat(0,1)$$

Conclusion and future work

The purpose of this paper is to show how to easily generate realistic trees in Autodesk Maya. Idea was to procedurally generate 3D tree models with textures and normal maps in Maya, so models can be easily modified and exported for further use. Results demonstrate clearly that 3D tree models can be generated quite fast and easily. The possibility of defining other stochastic L-systems, with different textures, that represent kinds of plants in nature should and will be explored in the future work. Further work will also cover optimization for generating these models faster. Reduction of subdivisions, what implies reduction of polygons is already an option, but the goal is to stop using PyMEL and its PyNodes completely. Even though they give massive amount of possibilities, treating every branch as PyNode reduce program's speed drastically. Original script for generating 3D tree models was made so that realistic representations of growing plants can be easy to modify with all parameters included in process: textures (their colors, width, height, combinations), trees (width, height and subdivisions of branches, L-system and probabilities of distributions) and leaves (where and how many will be on each branch).

References

Mishura, Y. (2008) *Stochastic Calculus For Fractional Brownian Motion And Related Processes.* Berlin, Springer.

Perlin, K. (2002) Improving noise. *ACM Transactions on Graphics.* 21 (3), 681-682. Available from: doi: 10.1145/566654.566636 [Accessed 10th June 2016].

Prusinkiewicz, P. (1986) Applications of L-systems to computer imagery. In: Ehrig, H., Nagl, M., Rozenberg, G. & Rosenfeld, A. (eds.) *Graph Grammars: Proceedings of the 3rd International Workshop on Graph Grammars and Their Application to Computer Science,*

Graph Grammers 1986, 2 -6 December 1986, Warrenton, Virginia, USA. Berlin, Springer. pp. 534-548.
Prusinkiewicz, P., Hammel, M., Hanan, J. & Mech, R. (1996) L-systems: From The Theory To Visual Models Of Plants. In: Michalewicz, M.T. (ed.) *CISRO 1996: Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences, CISRO 1996, 5 - 7 Febrary 1996, Melburne, Australia.* Clayton South VIC, CSIRO Publishing. pp. 1-32.
Prusinkiewicz, P., James, M. & Měch, R. (1994) Synthetic topiary. In: Glassner, A. (ed.) *SIGGRAPH 94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques, SIGGRAPH 1994, 24 - 29 July 1994, Orlando, Florida*. New York, ACM Press. pp. 351-358.
Prusinkiewicz, P. & Lindenmayer, A. (1996) Modeling of cellular layers. In: Prusinkiewicz, P. (ed.) *The Algorithmic Beauty of Plants. The Virtual Laboratory.* Springer, New York, pp. 145-174.
Reche-Martinez, A., Martin, I. & Drettakis, G. (2004) Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics.* 23 (3), 720-727. Available from: doi: 10.1145/1186562.1015785 [Accessed 10th May 2016].
Stojanović, N. (2016) Applications of two-dimensional Perlin Noise wit C++. *Serbian Science Today.* 1 (1), 157-168.
Ward, G. (1991) A recursive implementation of the perlin noise function. In: Glassner, A.S. (ed.) *Graphics Gems II. The Graphic Germs Series: A Collection of Practical Techniques for the Computer Graphic Programmer*. Academic Press, San Diego, pp. 396-401.