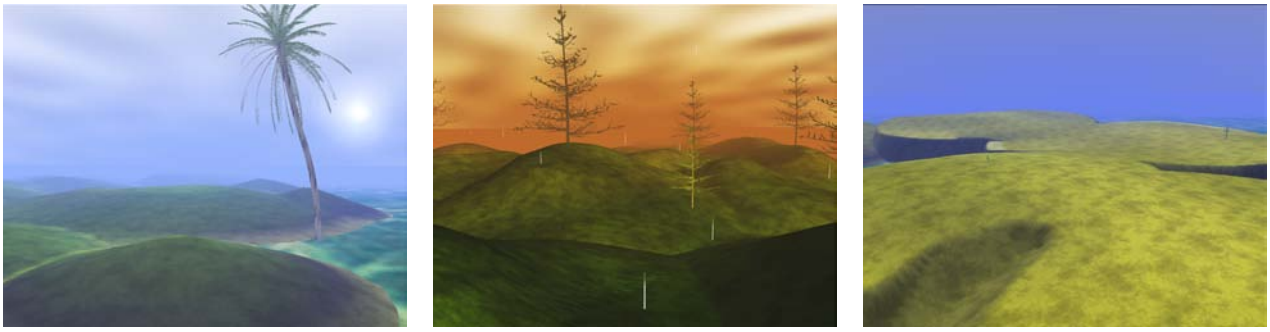
Affective Scene Generation

Carl Hultquist*
Department of Computer Science
University of Cape Town
Cape Town, South Africa

James Gain†
Department of Computer Science
University of Cape Town
Cape Town, South Africa

David Cairns‡
Department of Computing Science
and Mathematics
University of Stirling
Stirling, Scotland



(a) (b) (c)

Figure 1: Examples of scenes rated by users using our system. One user described these scenes using our rating system as: (a) -0.5 wet, 0.5 sparse, 0.7 tropical, 0.0 cloudy, 0.9 light, 0.1 mountainous and 0.9 undulating; (b) 0.3 wet, 0.3 sparse, 0.1 tropical, 0.5 cloudy, -0.8 light, 0.7 mountainous and 0.7 undulating; (c) -0.9 wet, 0.7 sparse, 0.8 tropical, -1.0 cloudy, 1.0 light, -0.3 mountainous and 0.1 undulating

Abstract

A new technique for generating virtual environments is proposed, whereby the user describes the environment that they wish to create using adjectives. An entire scene is then procedurally generated, based on the mapping of these adjectives to the parameter space of the procedural models used. This mapping is determined through a pre-process, during which the user is presented with a number of scenes and asked to describe them using adjectives. With such a technique, the ability to create complex virtual environments is extended to users with little or no technical knowledge, and additionally provides a means for experienced users to quickly generate a large, complex environment which can then be modified by hand.

CR Categories: G.1 [Mathematics of computing]: Numerical analysis— [G.1.2]: Numerical analysis—Approximation[Approximation of surfaces and contours] I.2 [Computing methodologies]: Artificial intelligence—[I.2.1]: Artificial intelligence—Applications and expert systems[Natural language interfaces] I.3 [Computing methodologies]: Computer graphics— [I.3.7]: Computer graphics—Three-dimensional graphics and realism[Virtual reality]

Keywords: Virtual environments, procedural models, function approximation, radial basis function networks

1 Introduction

With processor and graphics hardware performance increasing at such a rapid rate, the modern desktop machine is more capable of interactively displaying ever larger and more complex virtual environments (VEs). Such environments are particularly useful in the entertainment and education[1] sectors, and as a result of this increase in performance there is a demand for these sectors to make use of larger and more complex VEs.

Sadly, human "hardware" is not as capable of keeping up, which has a two-fold impact:

1. Creators of VEs are put under immense pressure by having to produce larger and more complex scenes.

2. Users with little or no technical knowledge but who may have considerable artistic talent are even less able to create compelling VEs. This aspect has a particular impact both in in the education sector and in the developing world.

[1]The reference to the education sector refers to the use of simulations for training purposes.

*e-mail: chultqui@cs.uct.ac.za
†e-mail: jgain@cs.uct.ac.za
‡e-mail: dec@cs.stir.ac.uk

AFRIGRAPH 2006, Cape Town, South Africa, 25–27 January 2006.
© 2006 ACM 1-59593-288-7/06/0001 $5.00

With regard to designing VEs, the majority of the tools available require that the user has a fairly sophisticated technical training not only in using the tools, but also in understanding other technical concepts related to the field of VEs, physics, and computer graphics in general. Experienced users are able to readily and rapidly use such tools, but most novice or non-technical users are typically at a complete loss. More generally, the *interface* to VE design prohibits such users from being able to design VEs.

With these thoughts in mind, an ideal technique that addresses these problems should provide the following features:

1. Allow large and complex VEs to be created quickly.
2. Provide an interface that supports non-technical users.

We present a new technique intended to solve these problems, after which we provide preliminary results on the use of this technique and outline details on how the technique could be improved. The work presented is still in progress, and more conclusive testing and improvements are planned.

2 Related work

Part of the overall problem has been addressed through the use of *procedural modelling* techniques. Such techniques take a set of simple inputs and, by applying a set of procedural steps, produce more complex output. The inputs to the techniques can take a variety of forms, including:

- **Scalar values:** some techniques such as Perlin noise [Ebert et al. 1994] require a few simple scalar parameters to control the scale, stretch and number of fractal levels in the generated noise. Most techniques use scalar values in conjunction with other more complex inputs.
- **Sets of rules:** these are used by L-systems which were first conceived by Lindenmayer [1968], and later used for computer graphics by Lindenmayer and Prusinkiewicz [1990]. An L-system is a parallel string rewriting technique that iteratively modifies a string based on the rules that are passed as input, and the string is later interpreted to produce graphical output. Such systems are used extensively for modelling plants.
- **Image maps:** Parish and Muller use image maps to control elevation, land and vegetation types, population density, street patterns and other parts of their procedural city generator [2001]. Image maps have also been used by Deussen et al. to specify terrain and plant distribution in their modeling and rendering of plant ecosystems [1998].
- **Photographs:** Shlyakhter et al. make use of photographs for reconstructing tree models [2001], whilst Debevec et. al. use photographs in order to model and render architechture [1996].
- **Freehand sketches:** Ijiri et al. [2005] and Okabe & Igarashi [2003] make use of freehand sketches to guide the modelling of flowers and trees respectively.

Procedural techniques allow for *components* of a virtual environment such as plants, terrain and clouds to be generated by the computer, subject to a set of parameters. However, the number of parameters required by these techniques is, in general, very large — large enough that it is not feasible for a user to simply manipulate the parameters in order to obtain a desired output. The mapping of parameters to perceived results may also not always be intuitive to the user. Furthermore, it is often necessary for parameters from different components to interact in order to achieve truly realistic output: this interaction can be complex and difficult to control when adjusting the parameters by hand.

One possibility for addressing this issue is the use of *genetic algorithms* [Holland 1995]. The user is shown several scenes, and chooses which scene is "best" or closest to what they desire. Merry et al. [2003] investigated this approach but their study produced a negative result, inferring that users preferred being able to manually select values for the parameters controlling the procedural models. They hypothesise that more parameters[2] are required before the use of genetic algorithms will be better than manually specifying parameters.

Polichroniadis [2001] makes use of a similar technique to ours for human figurine animation. Adjectives are used to describe different animations, and then new styles of animation can be synthesised by choosing several adjectives and interpolating between their animations.

3 Overview of technique

As was outlined in Section 1, our aim is to provide a technique that allows for VEs to be created quickly, and which provides an interface that is usable by non-technical users. We propose the following components for achieving each of these goals:

1. *Allow large and complex VEs to be created quickly.* Procedural modeling methods (also known as procedural models) provide a useful and effective means of generating a large variety of different objects automatically. That is, procedural models eliminate the typically high cost of modeling objects for a VE, thus drastically increasing the speed at which a VE can be generated. It should be noted that procedural models are not without their own problems: they can be too restrictive, and determining the correct set of parameters to achieve a certain "look" can be extremely difficult. The first issue (that of procedural models being restrictive) is completely dependent on the modeling techniques used, and so for our purposes can be ignored (since arbitrarily more parameters and complexity could be added to improve flexibility). The latter issue (that of determining the "correct" parameter set for a particular "look") will be addressed shortly as part of our technique.

2. *Provide an interface that is accessible to non-technical users.* Suppose a user wishes to create a VE: they have a mental picture of what this VE looks like, but how can they convey this to the computer? Since our aim is to provide an interface that is usable by non-technical users, a better approach would be to leave the computer out of this question for the moment, and simply phrase it as "how can they convey this to another entity?". In everyday life, a non-technical user would convey their ideas to another person, which means the core question being dealt with is "how can they convey this to another person?".

   When one person wants to convey the idea of a VE to another person, the natural way that they do this is to *describe* important features of the VE, so that the

[2] They used 21 parameters.

other person can hopefully build as close as possible a picture of the VE in their mind. The act of describing, and in particular of describing specific details, involves interspersing the description with *adjectives* that qualify certain aspects of the description.

With this in mind, we propose an interface in which the user can choose from a number of adjectives that describe the VE they wish to create. Such an interface should be familiar to the user, as it mimics the way in which they interact with other people. Associated with each adjective is a scalar value that can be adjusted to either enhance or reduce the impact of the adjective, if desired (in this way, users can quantify abstract concepts such as the use of the word "very").

The remaining problem now is to link the two ideas discussed above: to find some way in which the adjectives input by the user can drive the procedural model generation by mapping to correct procedural parameter values. Our proposed solution is now discussed.

3.1 Adjective space and parameter space

Suppose that the user can choose adjectives from a set $\boldsymbol{A}$ to describe their VE. We define *adjective space*, $\mathcal{A}$, to be a subset of $|\boldsymbol{A}|$-dimensional real values, $\mathcal{A} = [-1; 1]^{|\boldsymbol{A}|}$ (where $|\boldsymbol{A}|$ denotes the number of adjective in the set $\boldsymbol{A}$). Each dimension in $\mathcal{A}$ thus relates to a unique adjective in $\boldsymbol{A}$, and the value x in any dimension of $\mathcal{A}$ is a real value in the range $[-1; 1]$ which is the *scalar value* associated with the relevant adjective (as discussed in point 2 above). Hence an element of $\mathcal{A}$ describes a specific set of scalar values associated with the adjectives in $\boldsymbol{A}$, and $\mathcal{A}$ represents the set of all possible descriptions that a user could make.

Suppose that $\boldsymbol{P}$ represents the set of available procedural models, and that $\boldsymbol{P}_m$ is the set of parameters controlling procedural model m, $m \in \boldsymbol{P}$. Let

$$\boldsymbol{Q} = \bigcup_{m \in \boldsymbol{P}} \boldsymbol{P}_m$$

be the set of all parameters controlling all the available procedural models.

Without loss of generality, assume that each parameter is represented by a single, scalar, real value. This assumption is well founded since we can establish mappings for other parameter types (such as boolean, integer and enumerant parameters) to and from real values[3]. We then define *parameter space*, $\mathcal{P}$, to be the $|\boldsymbol{Q}|$-dimensional reals, viz. $\mathcal{P} = \mathbb{R}^{|\boldsymbol{Q}|}$. Hence an element of $\mathcal{P}$ describes a specific set of real values associated with the parameters in $\boldsymbol{Q}$, and $\mathcal{P}$ represents the set of all possible scenes that could be generated by the procedural models in $\boldsymbol{P}$.

Recall that our aim was to establish a means for mapping the user's description into numerical parameters that can be evaluated by the computer. With the definitions of adjective space and parameter space above, this is mathematically expressed as the need to find a function

$$f : \mathcal{A} \to \mathcal{P}$$

[3]For example, to convert from a real to an integer value, one could simply round the real value to the nearest integer; to convert a real into a boolean, one could simply divide the reals into two subsets and say that a real value less than 0 corresponds to the boolean value *false*, whilst a real value greater than or equal to 0 corresponds to the boolean value *true*.

However, determining f is a non-trivial task, and moreover different users will express themselves differently meaning that if f accurately models one user's descriptions, it is unlikely to accurately model the descriptions of another user.

Our solution to these problems is for the system to "learn" f for each user through a training process. A number of quasirandom[4] elements are chosen from $\mathcal{P}$, and for each the corresponding VE is generated. The user is able to explore the VE, after which they describe the VE by choosing adjectives and associated scalar values, giving an element of $\mathcal{A}$. The challenge now is to determine f — that is, to find the relationship between $\mathcal{A}$ and $\mathcal{P}$ — by using the data acquired. This can be approached using several classes of techniques, including *scattered data interpolation*, *scattered data approximation* and *supervised learning*.

3.2 Function approximation

A widely adopted technique for scattered data interpolation and supervised learning is the use of *radial basis function networks* [Orr 1996; Wendland 2005]. A traditional radial basis function network (RBFN) is a special case of the more general *neural network*, with the following defining characteristics:

- An RBFN has exactly one hidden layer.
- Each unit h_j in the hidden layer is modelled by a *radial basis function* (RBF), and all components of the input vector $\vec{x}$ are fed forward into every unit in the hidden layer.
- The outputs from the hidden layer are linearly combined with weights to form the function's output.

A graphical illustration of an RBFN is shown in Figure 2.

If p training pairs $(x_i; y_i)$ are used to train a RBFN with m RBFs, then using a modified least squares derivation the weights w_j are solved for as

$$\vec{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{H}^{\top}\vec{\mathbf{y}}$$

where $\mathbf{H}$, the *design matrix*, is

$$\mathbf{H} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \dots & h_m(x_1) \\ h_1(x_2) & h_2(x_2) & \dots & h_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(x_p) & h_2(x_p) & \dots & h_m(x_p) \end{bmatrix}$$

and $\mathbf{A}^{-1}$, the *variance matrix*, is

$$\mathbf{A}^{-1} = (\mathbf{H}^{\top}\mathbf{H} + \mathbf{\Lambda})^{-1}$$

The elements of $\mathbf{\Lambda}$ are all zero except along the diagonal, which is filled by regularisation parameters associated with the RBFs. This solution minimises the mean-squared error between $f(x_i)$ and y_i over all the training pairs $(x_i; y_i)$.

More details on the underlying mathematical and algorithmic details of RBFNs and their applications can be found in the work of Orr [1996].

[4]By quasirandom, we simply mean elements are chosen in such a way as to adequately sample the space covered by $\mathcal{P}$. The exact method that would be used to accomplish this has not yet been determined.

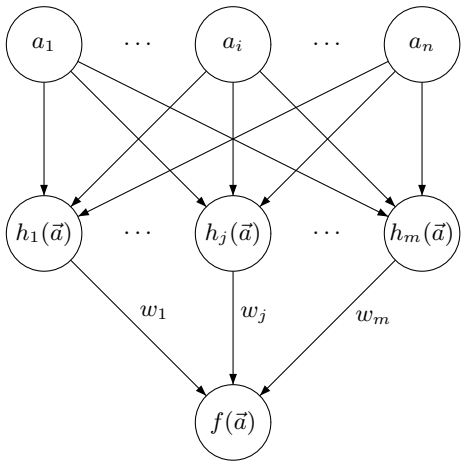


Figure 2: A traditional RBFN accepts input from an n-dimensional vector, $\vec{x}$, which is fed forward to the m hidden units h_j ($j \in 1 \ldots m$). The outputs from the hidden units are then each weighted by w_j and summed to give the result, $f(\vec{x})$.

4 Implementation

We have completed a simple implementation that successfully demonstrates a complete system, from acquiring user descriptions to using these as training data for function approximation, and then allowing the user to specify adjectives and generate new scenes using the resulting function.

The scenes generated are presently being controlled by 16 parameters that depict the following procedural elements:

- **Landscape:** by using Perlin noise and a number of pseudo-randomly placed Gaussian functions with varying properties, an island is created that can mimic flat beaches, rolling hills, rocky cliffs, or even combinations of these.
- **Trees:** an implementation [Diestel 2003] of the techique of Weber and Penn [1995] for generating realistic trees is used. For speed (both of scene generation and of rendering) we currently use pre-generated models of 3 different types of tree (palm, cactus and fir), and these are placed pseudo-randomly over the landscape according to the landscape type (for example: palms on beach areas, cactii on desert and firs on grassland).
- **Sky:** based on the time of day, the sky changes using the scattering properties described by Preetham [1999].
- **Clouds and rain:** Perlin noise is used to generate a cloud texture, and parameter thresholds control rain that is rendered using a particle simulation.

For function approximation, we have used a RBFN as described in Section 3.2. Our RBFN has the following specific properties:

- An RBF is placed at each point in adjective space supplied by the user. This is a widely accepted and adopted approach to placement of the RBFs [Orr 1996].
- All of the RBFs in each network have equal and fixed radii. In general, it is possible for the RBFs to have different radii (and even for the individual RBFs to be different radial functions) but exploring these possibilities is beyond the scope of this paper, and most applications of RBFNs also employ equal and fixed radii for their RBFs.
- Associated with each RBF is a regularisation parameter. This is used in a local regularisation step [Orr 1996] which seeks to minimise the *general cross validation* (GCV) criterion. The GCV is an estimated measurement of how much the RBFN will deviate from the theoretically perfect function on other future unknown inputs. The resulting regularisation parameters are used to better control the weights in the network, as well as to remove RBFs from the network which are not required (and which may be causing, for example, over-sampling in certain regions).

To simplify implementation, there is an individual RBFN for each procedural parameter. This is potentially limiting, as it does not allow for parameters to interact with and affect each-other: some possibilities for overcoming this limitation are discussed in Section 6.

5 Testing and results

Thus far, some explorative testing has been conducted by selecting 15 elements of $\mathcal{P}$ and obtaining descriptions of these from 8 different users. 7 adjectives were chosen before the testing that reflected some obvious differences in the generated scenes: the adjectives chosen were *wet*, *sparse*, *tropical*, *cloudy*, *light*, *mountainous* and *undulating*. In practice, more adjectives could be useful and the possibility of allowing a user to specify their own adjectives needs to be explored: these issues are given more thought in Section 6.

Using the collected data, RBFNs are constructed for each user, which then allows the user to describe a desired scene and have the RBFNs determine the corresponding values in parameter space, which are used to generate the scene.

In general, the results of this technique are quite mixed: in some instances, the generated scene does indeed match the user's expectations, but in others the result is not at all what the user had in mind. We have found that the best results are achieved when a user chooses an element of $\mathcal{A}$ which lies within the "effective radius" of several RBFs (that is, the contribution from each of these RBFs is not very close to 0, and so they all have an effect on the output of the RBFN for that element). Conversely, when a user chooses an element that happens to lie within the effective radius of only 1 RBF or, worse, does not lie within the effective radius of any RBFs, then the output is biased towards one element of $\mathcal{P}$ and so the output is not as expected. This could be corrected by increasing the radii of all RBFs — and hence increasing their effective radii — but this makes overfitting more likely, meaning that the function will fit the training data extremely well but will not generalise to other data. Additionally, trying to control this by insisting on an even sampling in the function's domain, $\mathcal{A}$, is also impossible since the points in $\mathcal{A}$ are provided by the user, and so we have no control over these.

6 Conclusions and future work

For regularly sampled descriptions in $\mathcal{A}$, the technique proposed produces good results and the resulting scenes meet the users' expectations. More conclusive testing needs to still be done to ascertain whether users prefer our technique to that of manually choosing values for the procedural parameters. There are also still several issues that need to be addressed before the system could be considered practical and usable, and areas for future research include the following:

- An alternative approach to having equal radii for all RBFs is to allow the RBFs to have different radii, according to the change of density in training data over $\mathcal{A}$. In doing this, one would need to ensure that all of $\mathcal{A}$ is covered by some minimum number of RBFs, as well as avoiding overfitting in denser areas.

- RBFNs were used because they are the most widely accepted form of scattered data interpolation; however, there are many other techniques available, which may produce better results and so should be considered. It could also be argued that, since we rely on data collected from users, the data we are interpolating is inherently noisy, and so techniques for scattered data *approximation* would be much more appropriate.

- The issue of adjective selection needs to be addressed in more detail, as certain users may feel more comfortable with their own adjectives rather than predetermined ones.

- Presently, the user must assign a value to *every* adjective when they wish to generate a scene. In practice, a user may wish to omit some adjective, when they do not mind what value the adjective takes on. Means for dealing with this situation would be desirable.

- Currently, each procedural parameter is assigned its own RBFN and so there is no interaction between the parameters. Several ways of approaching this are possible and should be explored, such as a separate function which takes as input all the outputs from the RBFNs and produces new output parameters, or possibly by passing the procedural parameters as inputs to the RBFNs and then repeatedly re-evaluating the RBFNs (and updating their inputs with the most recent RBFN outputs) until the system stabilises.

7 Acknowledgements

We would like to thank the Harry Crossley Foundation and the Postgraduate Funding Office at the University of Cape Town, without whose support this research would not have been possible.

References

Debevec, P. E., Taylor, C. J., and Malik, J. 1996. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH 96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 11–20.

Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz, P. 1998. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH 98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 275–286.

Diestel, W., 2003. Arbaro — tree generation for povray. `http://arbaro.sourceforge.net`.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. 1994. *Texturing and modeling: a procedural approach*. Academic Press Professional, Inc.

Holland, J. H. 1995. *Hidden order: how adaptation builds complexity*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Ijiri, T., Owada, S., Okabe, M., and Igarashi, T. 2005. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. *ACM Transactions on Graphics 24*, 3, 720–726.

Lindenmayer, A. 1968. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18, 280–315.

Merry, B., Giacchetta, G., Thwaits, B., and Gain, J. 2003. Genetic selection of parametric scenes. Tech. Rep. CS03-12-00, Department of Computer Science, University of Cape Town.

Okabe, M., and Igarashi, T. 2003. 3d modeling of trees from freehand sketches. In *Proceedings of the SIGGRAPH 2003 conference on Sketches & applications*, ACM Press, New York, NY, USA, 1–1.

Orr, M. J. L. 1996. Introduction to radial basis function networks. Tech. rep., University of Edinburgh, April.

Parish, Y. I. H., and Müller, P. 2001. Procedural modeling of cities. In *SIGGRAPH 2001: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 301–308.

Polichroniadis, T. P. 2001. *High Level Control of Virtual Actors*. PhD thesis, University of Cambridge.

Preetham, A. J., Shirley, P., and Smits, B. E. 1999. A practical analytic model for daylight. In *SIGGRAPH 99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 91–100.

Prusinkiewicz, P., and Lindenmayer, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA.

Shlyakhter, I., Rozenoer, M., Dorsey, J., and Teller, S. 2001. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications 21*, 3 (May/June), 53–61.

Weber, J., and Penn, J. 1995. Creation and rendering of realistic trees. In *SIGGRAPH 95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 119–128.

Wendland, H. 2005. *Scattered data approximation*. Cambridge University Press.