Procedural Natural Systems for Game Level Design

Remco Huijser Jeroen Dobbe Willem F. Bronsvoort * Rafael Bidarra

Cannibal Game Studios
2629 JD Delft, The Netherlands

Delft University of Technology
2628 CD Delft, The Netherlands

Abstract

Due to the increase in magnitude and level of detail of next-gen games, the time required to manually design a game level has increased dramatically. This paper introduces *procedural natural systems*, a novel approach aimed at reducing the time needed to design large-scale natural phenomena for game levels. The concept of natural systems separates the shape of a natural phenomenon from its footprint, allowing a designer to edit either of them separately. Various procedural techniques are used to combine the shape and footprint of a natural system, as well as to tweak these in real-time in a game world. We conclude that natural systems provide a solid foundation for intuitive, flexible and efficient procedural generation of significant portions of a game level.

Keywords: procedural content generation, natural systems, game level design

Authors' contact:
R.Huijser|J.Dobbe@cannibalgamestudios.com
W.F.Bronsvoort|R.Bidarra@tudelft.nl
*corresponding author: tel. +31 15 278 4564

1. Introduction

As predicted by Moore [1965], computational performance has roughly doubled every two years, each new generation of computer hardware opening doors to endless new possibilities. Likewise, new generations of game hardware, often referred to as next-gen, have enabled game developers to create games that are more realistic and immersive than ever before. Along with this increase in possibilities, customer expectations and demands have also increased, forcing game developers to deliver whatever becomes technically possible.

Although technological growth has been exponential, game content production efficiency has not grown at an equal pace. Most digital content, even today, is created manually. With ever more detailed game content, the time and money it takes to develop games has increased dramatically.

One of the most promising ways for game developers to cope with this problem is to increase the efficiency of content creation. This can be achieved by automating the content creation process as much as possible, reducing the amount of work done manually. Here, a prominent role is reserved for procedural content generation methods. Although these methods have already been successfully applied to create specific game content, the design of large-scale natural phenomena like rivers and mountains ridges is still far from optimal. The main reason for this is that editing natural phenomena often involves changing many different aspects of the game world, e.g. the height and texture of the terrain and the 3D models that populate the world. Because these modifications still often have to be done manually, this makes designing natural phenomena a very tedious activity, and making changes afterwards a prohibitive practice.

Basically, current procedural techniques developed for level design fall short in a number of areas. In Section 2 we take a look at the current state of tools for level design and procedural content creation methods.

To address the current issues, and allow for large natural phenomena to be placed and modified in a game level, we propose *procedural natural systems*: an approach to assist designers in creating large natural phenomena. A natural system consists of a *footprint* that contains the 'appearance', a *shape* that controls the placement, and a *procedure* that combines the footprint and shape to generate an instance of the natural system in the game world.

The principal contribution of our work lies in the flexibility of this separation of footprint and shape, in their efficient reusability, and in the intuitivity of supporting intermixed manual tweaks in real-time.

The main concepts of procedural natural systems are described in more detail in Section 3. In Sections 4 and 5 we show how we put this concept to practical use in a prototype level design tool. Some results we achieved with this prototype are presented in Section 6, and our conclusions are discussed in Section 7.

2. Background

To explain the genesis of the concept of procedural natural systems, we describe how level design is currently being done (Subsection 2.1), what can be achieved with procedural techniques (Subsection 2.2), and finally how natural phenomena can be structured and classified (Subsection 2.3).

2.1 Current game level design

Over the last decade, the importance of level design has grown significantly. It combines inputs from most departments of a game development studio, and because of this central role it is a potential bottleneck within the overall process.

The most serious weakness of current level design practices is that it is very labor intensive. A lot of time is spent on low-level operations like dragging vertices, applying textures the right way, and adding decoration objects one by one. Because these operations are so low level, making changes to previous design work is also very time consuming. As a result, changing 'early stage choices' later on in the process should be avoided at all costs, making the process of level design rigid and very linear.

We can illustrate these problems with the example of creating a river in current level editing tools. Roughly, the following steps are involved:

- Carve out the river using simple tools that change the height of a terrain until we get the shape of the river. In some modern level editing tools we get a curve-based height modification tool to somewhat ease this process;
- Manually paint soil colors (**textures**) on the river and riverbed to ensure that the right colors are displayed to represent the type of river we want;
- Add individual **objects** (such as trees and bushes) to get the vegetation we need;
- Since we are modeling a river, we also need to add something to represent the water level, which is usually following the shape of the river. In some modern level editing, we can use for this the same curve-based tool as above.

When a change needs to be made to the shape of the river, e.g. as a result of game-play considerations, many of these operations will have to be undone and redone manually.

The general consensus is that, in order to keep up with the increasing amount of work that has to be done, the future of level design holds two possibilities. The first one [Byrne 2009] is super specialization where, for example, a single person is highly specialized in adding grass and small plants to a level. This approach increases the overall productivity, but makes level design a very repetitive task. It also suffers from the law of diminishing returns due to management overhead, and does not really solve the flexibility problem.

The second possibility [de Jong 2009] is that better tools are developed, allowing the designer to work more efficiently. This can be done using procedural content creation. In this approach, level designers are still in charge of the *design* of the level, but let the computer do the heavy lifting. Here Moore’s law is actually working to our advantage: since computers become faster, they can do a lot more construction based on a design automatically than before. Since the problem (higher demands) and solution (automation) grow at the same pace, procedural content creation poses a far more scalable solution than super-specialization.

2.2 Procedural content generation

Procedural content generation has been an active research topic for over thirty years, resulting in high-quality models and procedures for specific terrain features, such as landscapes [Miller 1986, Musgrave 1993], plant models [Prusinkiewicz and Lindenmayer 1990] and vegetation distribution [Deussen et al. 1998], road networks [Sun et al. 2002], urban environments [Parish and Müller 2001], and building facades [Wonka et al. 2003, Müller et al. 2006, Finkenzeller 2008]. The reader is referred to the recent survey [Smelik et al. 2009] for a more detailed analysis of the pros and cons of these techniques, as well as for current trends and further developments in the area.

Procedural content generation has found its way in a variety of commercial products, which in turn are being profitably used in the game industry for specific purposes. Examples of these are the procedural generation of textures [Allegorithmic 2010], and of trees and other types of vegetation, e.g. [Speedtree 2010] and [Xfrog 2010]. Xfrog allows you to procedurally generate a model of a tree by designing an abstract version of the tree with some simple parameters. Yet other tools have been proposed that generate either specific types of virtual worlds, such as GeoControl for terrains [Rosenberg 2010] and City Engine for cities [CityEngine 2010], or fully-integrated virtual worlds, such as SketchaWorld [SketchaWorld 2010]. GeoControl is an example of a tool that allows the user to generate very detailed terrain shapes based on all sorts of geological settings and parameters.

To underpin our design choices, we derived a set of guidelines for designing procedural tools, which we can illustrate based on two of the tools mentioned above: GeoControl and Xfrog.

Since GeoControl generates very detailed results, it becomes very slow to probe the influence of different input settings. This makes it hard to interpret what certain inputs and settings do, as it simply takes too long to play around with the different input values. Xfrog has a more interactive interface, where a change in the input is reflected rather quickly in the output. This allows the user to play around with the settings to get the desired results. This playing around with settings and possibilities is paramount to creativity. Our first guideline for a procedural content generation tool is therefore that it should be **real-time** or, at least, close enough to be interactive in changing the settings.

Another desirable aspect of a procedural content generation tool is that the settings are not too abstract. Artists and designers are usually not biologists or physicists. Therefore we need to make sure that there is some logical **mapping** between the input and output, which the artists and designers can intuitively understand. GeoControl has a lot of detailed settings for creating realistic erosion and other real-world effects, Xfrog provides a more abstract set of inputs, but with a closer mapping to the end-result, i.e. a good visual correspondence between the input and the output.

Since design is a creative process, designers should have full control over what they create. When using a procedure to automate construction of the content, it is highly unlikely that this process at once produces the end-result exactly as envisioned by the designer: as we reduce the input to a few parameters, we typically give away some control over output details. Manual **tweaking** of the output can be used to overcome this issue without over-complicating the procedure itself. However, it is very desirable that manual tweaking and changing the input set can occur intermixed, in any order, e.g. changing the construction after some manual tweaking has already taken place. Simply generating the content once before allowing manual tweaking will not be good enough. Neither Xfrog nor GeoControl provide such intermixed editing features.

2.3 Earth sciences

Since our aim is to 'artificially generate natural phenomena', we now take a quick look at the science behind actual natural phenomena, without going into detail.

In earth sciences, many natural phenomena are classified according to certain properties. These classifications and groupings of natural phenomena allow us so have a structured view on nature, which we can use to create the abstractions necessary for our procedural natural systems.

Two relevant classifications are *biomes* and *landforms*. Biomes are a classification of natural systems based on similarities in vegetation, climate and location (e.g. tundra, tropical, grassland). In practice this means that the same type of plants, animals and soil organisms are present. Since these factors are mostly related to both color and aspect of small objects, biomes largely determine the **appearance** of a certain natural phenomenon.

Landforms are categorized by characteristic physical attributes such as elevation, slope, orientation, stratification and rock exposure. Since these properties relate mostly to the height and form of a terrain, we can largely interpret this classification as describing the **shape** of natural phenomena (e.g. mountains, gullies).

The interesting feature about these two classifications is that they are able to break down complex natural phenomena into independent and easy to understand concepts. By combining the appearance description with the shape description, we should be able to fully describe a natural phenomenon.

3. The concept of natural systems

As with biomes and landforms, natural systems separate the appearance of natural phenomena from the kind of shape they have in the world. With procedural content creation, we separate the design of content from its construction. If we apply this to natural systems, we can observe that the appearance and shape together represent the design of a natural phenomenon. By procedurally combining the appearance and shape, we can create an instance of that natural phenomenon. This activity represents the construction part.

The appearance of a natural system is captured by the *footprint*, which describes the “cross section” of a natural system. The *shape* of a natural system can be either a fat curve or a freeform area. Footprint and shape are procedurally combined to create each natural system instance in the game world. This relationship is shown in Fig. 1. A major advantage of the concept of natural systems is that the appearance of natural phenomena becomes a reusable component to create multiple (different) instances.

3.1 Footprint

In current level design practices, creating natural phenomena involves changing the height of the terrain, the texture that is applied to the terrain, and the 3D (vegetation) models that are added to the game world. In our approach, *environmental features* are used to describe how these aspects are defined in the footprint,

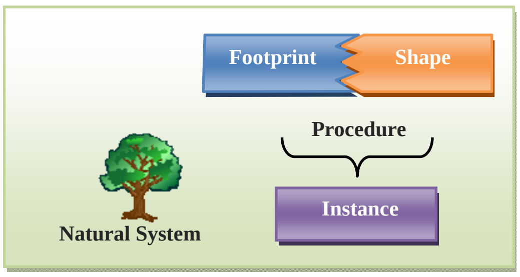


Fig. 1. A natural system consists of a footprint and a shape which are combined using a procedure to create an instance.

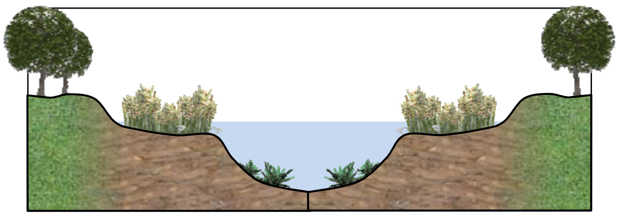


Fig. 2. Example of a natural system footprint.

i.e. the “cross section” of a natural system. An example of a natural system footprint is shown in Fig. 2.

We have identified four kinds of environmental features:

- the ***height feature*** indicates how the height of the terrain is modified by a natural system. Depending on the landform we want to replicate, we can raise or lower the terrain to, for example, model a ridge or canyon;
- the ***soil features*** indicate how the surface of a natural system looks like. Depending on the biome we are trying to replicate, this can, for example, be rock, moss, mud or sand. Soil features essentially influence the texture that is placed on the terrain;
- the ***vegetation features*** indicate which kind of vegetation grows where in a natural system and how this flora is distributed. Natural systems belonging to a forest biome will contain various vegetation features, ranging from large trees to smaller shrubs, due to the large biodiversity of these biomes;
- the ***water height feature*** indicates which height the level of water has for that natural system. This feature is of practical use only for natural systems that belong to a freshwater biome.

3.2 Shapes

When we look at a river and a lake, we see that a river can be described by something that looks like a curve, but that for a lake it is more appropriate to use a closed shape that describes its area. From a reusability perspective, it is desired that both shapes are interchangeable with one another. It should, for example, be possible to apply the same footprint of a stream to both a curve and an area. The former should result in a river, the latter in a lake. Furthermore, in some cases, the width of a natural system changes along the shape: a river has sections where it is broader or narrower. This means that a shape should also contain a width that determines over which range the footprint is applied. These requirements lead to the following shape definitions.

A **fat curve** is defined as the trace left by a moving circle of variable radius along a curve [Mestetskii 2000]. In our case, a fat curve consists of a number of connected control points, with each of these control points also having a width value that corresponds to the radius of the tracing circle at that point. The curve running through the control points is called the *base curve* of the shape. Because the width of the fat curve is only defined at its control points, the width at locations between two control points is defined by interpolating their width values. Examples of landforms that can be described with a fat curve are: rivers, glaciers, valleys, canyons, atolls and dunes.

We define an **area** as the trace left by a moving circle of variable radius along a closed curve. As with the fat curve, the area shape also consists of a sequence of connected control points, in which the last control point is connected to the first. The curve running through these control points is the base curve of this shape. The area enclosed by the base curve defines the *inside* of the area. As with the fat curve, each control point also has a width that corresponds to the radius of the tracing circle at that point. The difference between a fat curve and an area is that for the area shape only the outside of the trace is used. The reason for this is that for creating a natural system instance with an area shape, only half of the natural system footprint is used, as will be explained in Section 3.3. Examples of landforms that are defined by an area are: lakes, mountains, buttes and hills.

3.3 Combining footprint and shape

The approach for combining a footprint with the two shapes is basically by “sweeping” the footprint along the shape. As previously stated, it is desired that we are able to apply the same footprint to both types of shapes, without changing anything to the footprint. In this subsection, we will describe how the footprint shown in Fig. 2 is combined with a fat curve and an area shape.

To instantiate a natural system from a footprint and a fat curve, the footprint is swept along the base curve in a single direction; see Fig. 3. Conceptually, this means that a copy of the footprint is positioned at each control point of the base curve, perpendicular to the direction. To make sure the sides of the footprint are placed at the sides of the fat curve, the footprint is scaled to match the local width of the fat curve. After this, the environmental features of the footprint are used to instantiate a natural system. For the terrain

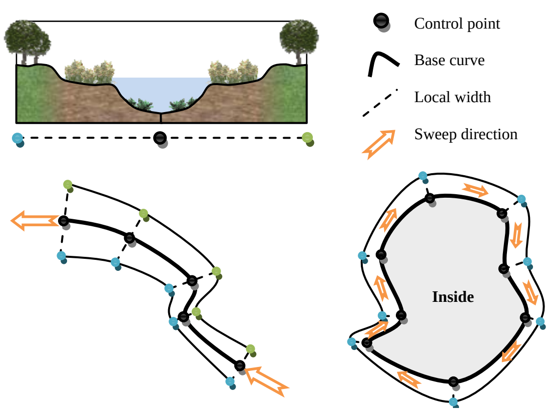


Fig. 3. “Sweeping” the footprint of a stream along a fat curve (left) and an area shape (right).

height feature, the sweeping operation results in a terrain along the fat curve that has the same height profile as the footprint. Sweeping the soil features works in much the same way; for example, the mud and grass texture are placed on the terrain that lies along the fat curve. Because the footprint is aligned with the width of the fat curve, the mud texture will be centered along the base curve. Further from it, towards the sides, the grass texture will be increasingly more visible, as defined by the soil features of the footprint. To evaluate where vegetation models need to be placed in the game world, while sweeping vegetation features, a distribution of points is used. For each of these points that lie along the fat curve, it is evaluated whether a vegetation feature dictates that vegetation grows at that part of the footprint. If it does, the procedure places the appropriate vegetation model at that location on the terrain in the game world. To avoid a very grid like distribution of the vegetation, pseudo-random offsets can be introduced. The water level height feature is represented by a (flat) water surface model in the game world.

Combining a footprint with an area shape is done in a similar way as combining a footprint with a fat curve. The only difference is that, for an area shape, only half the footprint is used. By sweeping half the footprint of the natural system along the area shape, only the outer area is changed (the area between the outer border of the area and the base curve). The inside area of the shape is defined by extrapolating the environmental feature values at the middle of the footprint. In the case of the stream footprint, the terrain height at the inside of the area is equal to the terrain height defined at the middle of the footprint. Furthermore, the inside area is completely covered by the mud texture and aquatic plants. The water surface model is extended to cover the entire inside of the area.

4. Natural systems put to work

To design the appearance of a natural system, the user describes the footprint of the natural system using the four kinds of environmental features mentioned above. This footprint can then be applied to a shape to create an instance of the natural system in the game world. While designing the approach for working with natural systems, we kept two aspects in mind. First of all, editing environmental features should be **intuitive**. This means that the user should have a clear understanding of what he is editing and how this adds to the final result. Secondly, the system should provide enough **flexibility** for the user to design a broad range of different natural phenomena.

4.1 Editing the footprint

Fig. 4 shows how a footprint is defined by combining the different environmental features. The top of the figure shows the height and water features. As one can see, the height feature is determined by a number of control points. The height of each control point indicates the terrain height at that point. The water height feature indicates the height of the water for this natural system.

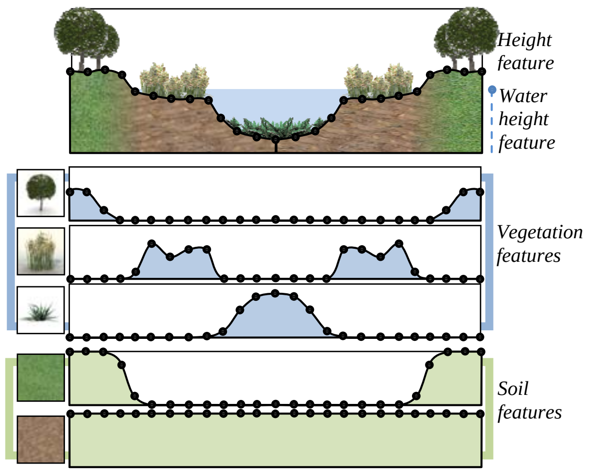


Fig. 4. Example of how environmental features describe the footprint of a natural system.

At the bottom of the figure, one can see the definition of the soil features, indicating which kind of texture is placed on the terrain. Soil features use the same curve-like editing as the height feature, but in this case the height of a control point represents the alpha value of the texture. Alpha blending or compositing is used to combine soil features, in a proportion determined by their alpha values. The lowest soil feature represents a mud layer, which is visible over the entire footprint. On top of the mud soil feature, a grass soil feature is defined. In the middle, the vegetation features are shown. For the footprint of the river, three vegetation features are used: trees, tall grass and aquatic plants. For vegetation features, the height of the control points indicates the vegetation density. An area with a higher density means that more vegetation is placed there.

4.2 Designing the shape

To design the shape of a natural system, the user is able to move around the control points of the shape, change the width of a control point, and add/remove control points to and from the shape. A major goal for designing the user interaction model for shape editing was that a user should be able to make changes to a shape in a fast and intuitive way, enabling him to quickly try out different configurations and rapidly work towards the shape that looks best. In order to facilitate this kind of behavior, all editing operations are performed directly on the shape and visualized at once.

4.3 Shape features

A shortcoming of the approach described so far is that the footprint will be exactly the same along the entire natural system. This is obviously not desired, as it does not capture, for example, the “natural” flow of a river. In the real world, cross sections of a river are

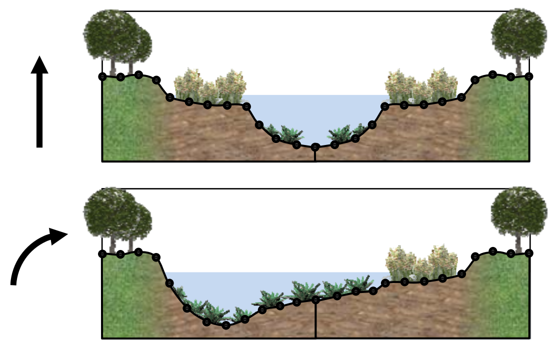


Fig. 5. Two footprints for a river, defined for straight river segments (top) and for river segments bending to the right (bottom).

different under different situations. For example, depending on the width of the river, the river bank might be steeper. Furthermore, when the river bends, its cross section is typically not symmetrical. To overcome this, we introduced *shape features*. Shape features describe a shape in terms of its local *width*, *curvature* and *slope*. By allowing the user to define different variants of the footprint for different shape feature values, we can conceptually position a different variant of the footprint at different positions along the shape, depending on its shape features, and in this way create a much richer description of the appearance of a natural system. Fig. 5 illustrates this. The footprint at the top describes the river at locations with zero curvature (where the river runs along a straight line). The footprint below describes the river when it sharply bends to the right. By mirroring the footprint horizontally, we obtain the footprint for the river when it bends to the left. To obtain the footprint at intermediate curvature values, the values of the environmental features can be linearly blended.

4.4 Terrain layers

The final point of attention is that the users should be able to further edit the game world after an area has been created using natural systems; natural systems should be available as an integrated part of level editing. An approach for this is to allow the user to freely mix editing the game world using natural systems and manual editing. Furthermore, it is undesired that a natural system overwrites manual edits made earlier. For example, when the shape of a river is changed, the area where the river previously ran should be restored to its original content. Our solution to this challenge is to edit the game world using layers. Each layer contains its own information regarding the height of the terrain, the texture that is placed on the terrain, and the vegetation models that are placed in the game world.

To get the final result, all layers are combined from bottom to top. Because each layer contains its own information, changes to a layer never change or overwrite the information stored in other layers. Because the process of combining terrain layers is independent of the actual content of the layers, this concept enables us to freely combine different ways of editing the game world. Terrain layers thus enable a user to combine natural systems and manual editing with each other. This allows him to, for example, make small changes to the natural system after it has been procedurally generated. An example of how two layers are combined using an add operation is shown in Fig. 6. Other operations, like subtract, min, max and average can also be used for the ways layers are combined.

4.5 Procedural generation of a natural system

The prototype system we developed allows the user to apply a natural system footprint to a fat curve. In this process, a procedure combines the information of the fat curve with the footprint to embed the natural

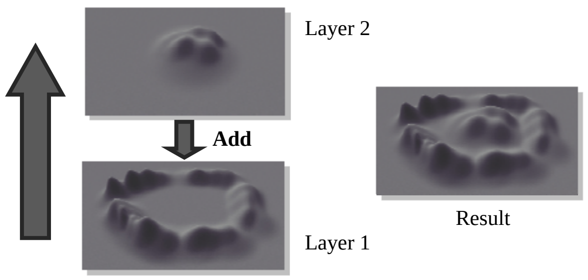


Fig. 6. Example of how the terrain height of two terrain layers is combined using an add operation.

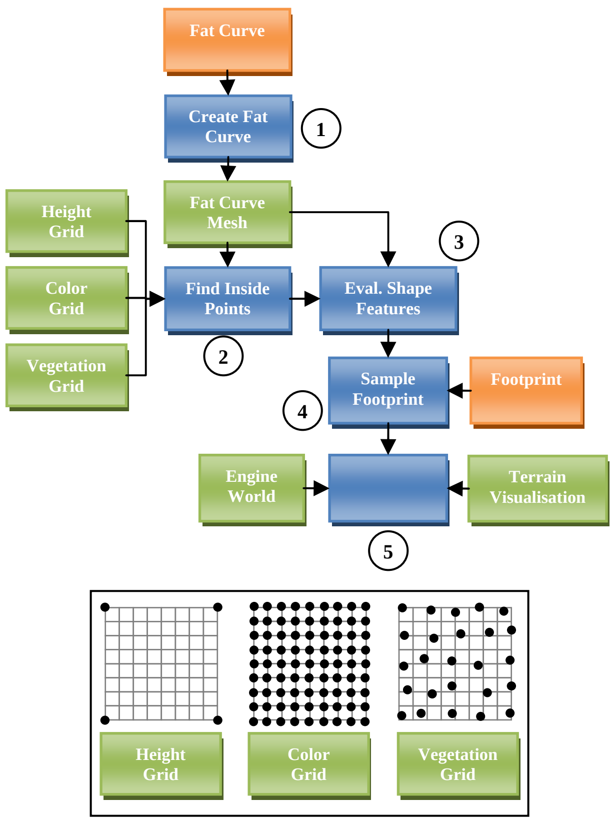


Fig. 7. Diagram (top) of the procedure in which the natural system footprint is combined with the fat curve. At the bottom the terrain height, terrain color and vegetation grids are shown.

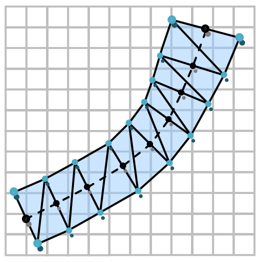


Fig. 8. Fat curve mesh representation

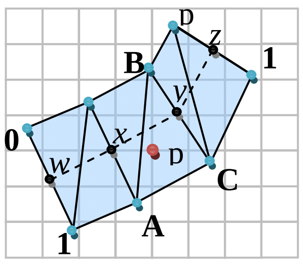


Fig. 9. Evaluating the normalized distance and local shape features for point p.

system instance in the game world. The steps of this procedure are shown in Fig. 7. The procedure for combining an area with a footprint is similar.

In short, the steps involved are the following:

1. The first step is to create a discrete mesh representation of the fat curve, which makes performing calculations on the fat curve a lot easier. The discrete mesh representation is created by evaluating each segment of the fat curve at a number of points and by connecting these points. Fig. 8 shows an example of a single discrete curve segment.
2. For the terrain height, terrain color and vegetation, three different grids are used. These grids are shown at the bottom of Fig. 7. The terrain height and color grids have resolution 1 and 8 samples per world unit, respectively. For the vegetation grid, a resolution of 2 samples per world unit is used. To create a more random effect, the vegetation grid points are given a pseudo random offset. For all grid points, we determine whether they are inside the area defined by the discrete mesh representation of the fat curve.
3. For each of the inside grid points, we evaluate the normalized distance of the point along the width of the fat curve. This represents the normalized position of the point on the footprint of the natural system. To evaluate the normalized distance, we use barycentric coordinates [Weisstein 2010]. To calculate the normalized distance, we first assign the normalized distance value of 0 to all vertices on one side of the curve and value 1 to vertices on the opposite side. How this works is shown in Fig. 9. To calculate the normalized distance at point p, we multiply the barycentric coordinates of p in the triangle defined by A, B and C with the normalized distance values of each of these vertices (A = 1, B = 0, C = 1). Calculating the shape features width, curvature and slope at each grid point is done in a similar way.
4. Once the shape features and normalized distance are known, they can be used to sample the natural system footprint for terrain height, terrain color, water height and vegetation density. In this step, the different variants of a footprint are blended to obtain the footprint at the desired shape feature value.
5. With the footprint information obtained above, the actual instance of the natural system is created. This activity involves:
   a) Applying the height and color information to the terrain visualization.
   b) Placing a water surface in the model of the game world.
   a) Placing the appropriate vegetation models at the right position in the game world.

5. Implementation

To demonstrate the feasibility and advantages of natural systems, a prototype has been developed. The goal of the prototype is primarily to show that the concept of natural systems has the potential to work as a useful extension to current level design practices. During the development of the prototype, we tried to adhere to the guidelines presented in Subsection 2.2 as much as possible.

The first guideline states that the application should be *real-time*. Even though our procedure, for simplicity, has been implemented entirely on the CPU, it is able to update the instance of a natural system at an interactive rate on current mid-end computer hardware. The prototype also offers a clear *mapping* between the provided input and the procedurally generated output. The main reason for this is that while editing the footprint and shape, the user is interacting with a representation that is close to the end result. Finally, we allow the user to manually *tweak* the end result by combining procedural with manual editing by using the layer system.

Our prototype has been developed using Cannibal Composer, which is a proprietary editor framework. For visualization purposes, the Cannibal Engine has been used. Both products have been developed in-house by Cannibal Game Studios. Together, Cannibal Composer and the Cannibal Engine offer a lot of standard editor and visualization functionality that makes the development of a prototype level editor a lot easier.

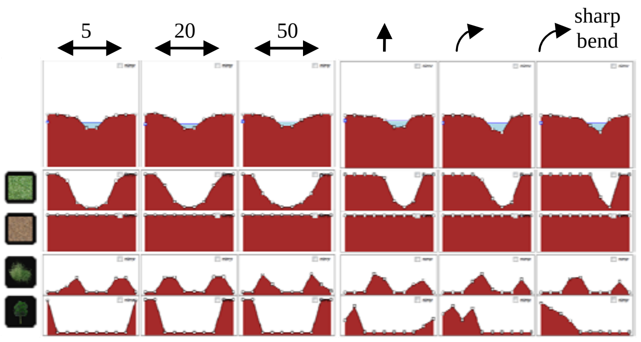


Fig. 10. Variants of a meander footprint for three width and three curvature values.

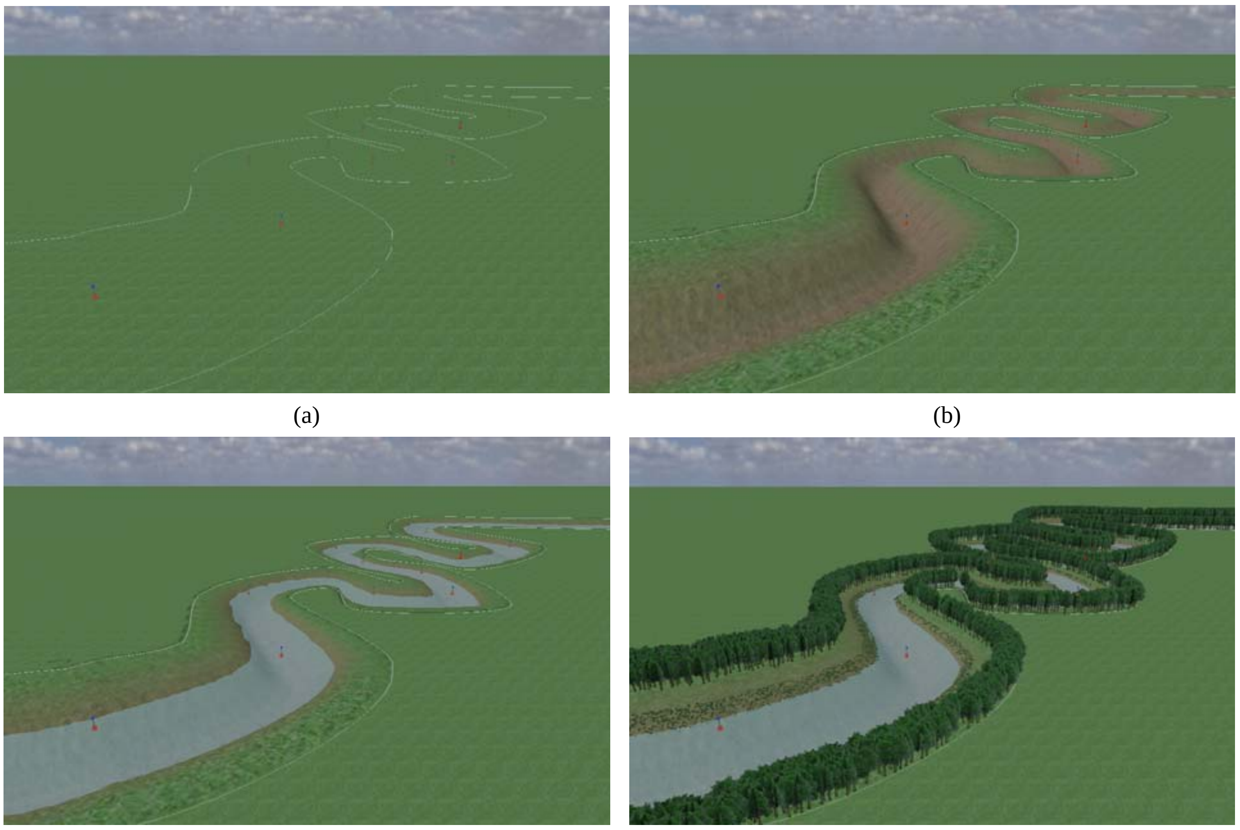

(a) (b) (c) (d)

Fig. 11. Creating a meander using natural systems: (a) designing the shape in the game world; (b) applying the height and soil features; (c) applying the water height feature; (d) applying the vegetation features.

6. Results

We now describe the process of creating a meander using nothing but natural systems, with the main goal of showing the ease with which something like a meander can be created in a fraction of the time it takes compared to achieving the same result manually.

With any natural system, the first step in this process is to design the footprint using the environmental features. Fig. 10 shows a screenshot of the footprint user interface. Designing the six footprint variants shown in Fig. 10 takes less than 15 minutes for a novice user.

After the design of the footprint, the shape of the natural system needs to be designed in the game world. This generally takes only a few minutes. Once the shape is finished, the footprint can be applied to the shape, after which the procedure can create an instance of the natural system in the game world. Fig. 11 shows the shape of the natural system and different steps of the generation procedure.

A major benefit of a natural system is that both its footprint and its shape can be freely and independently edited after they have been combined. The procedure that generates the natural system instance makes sure that any changes are immediately shown in the game world. Whereas designers need to redo a lot of work by hand in traditional level editors, with natural systems they are able to change large parts of the game world in a very simple and intuitive way. Changing, for example, the number of bends of the meander, as

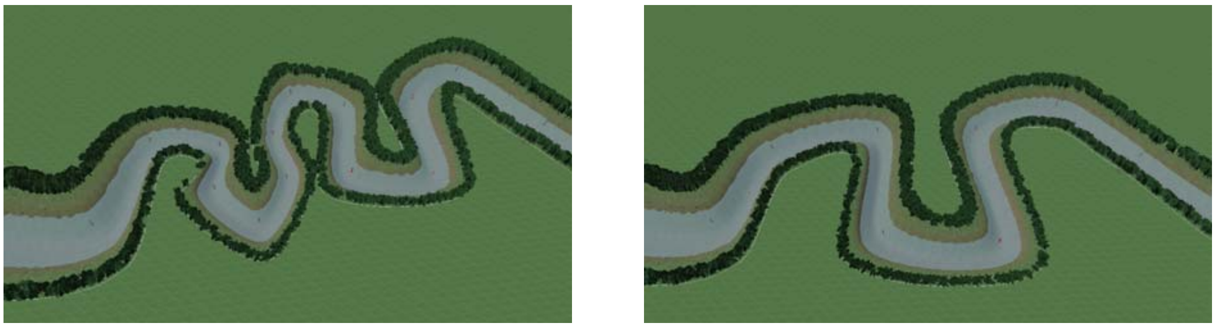
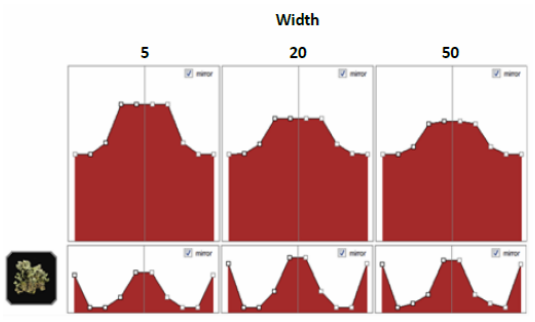
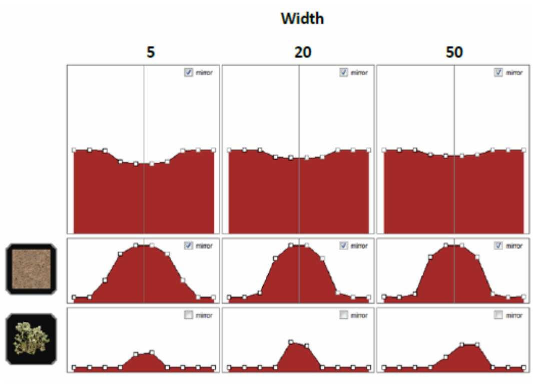

(a) (b)

Fig. 12. Editing the shape of a meander: (a) before and (b) after.



(a)



(b)

Fig. 13. Environmental features for (a) a butte and (b) a dried out creek.

shown in Fig. 12, takes only a few minutes. Depending on the complexity of the scene, manual editing might take a couple of hours.

In a similar way, other types of natural systems can be quickly specified and created following this approach. Fig. 13 depicts the definition for the environmental features of a butte, an eroded steep rock formation, and a recently dried out creek, consisting of a shallow groove with a muddy creek bed where some vegetation still grows. Fig. 14 shows two images of one such dried out creek, surrounded by a few buttes, both specified as natural systems.

7. Conclusions

Procedural methods are becoming more and more attractive to solve the increasing size and complexity of next-gen game levels. However, much research is required before these techniques can offer intuitive control parameters, provide powerful editing and visualization facilities of their results, and all this operated in real-time. In this paper we have introduced *natural systems*, a novel procedural approach to assist level designers in creating large-scale natural phenomena, e.g. rivers, canyons and ridges. The main feature of the natural system concept is the separation of the *shape* of a natural phenomenon (e.g. the trajectory of a river) from its *footprint* (e.g. the appearance of the river bed and banks). For each natural system instance desired in the virtual world, the designer interactively specifies its shape and its footprint, which are then combined by a procedure.

Our approach presents two main advantages: (i) it significantly reduces the time needed to generate complete natural systems, and (ii) it greatly increases the ease of editing the natural system's properties and attributes at any time throughout the design process. These advantages mainly arise from the fact that designers are given facilities to separately specify and edit the footprint and the shape of a natural system, which become therefore very convenient reusable components. As such, natural systems provide a solid foundation for intuitive, flexible and efficient procedural generation of significant portions of a game level.

References

ALLEGORITHMIC, 2010. Allegorithmic website. [Online] www.allegorithmic.com.

BYRNE E., 2009. Game-Artist.net, Interview Ed Byrne. [Online] www.game-artist.net/forums/vbarticles.php?do=article&articleid=9.

CITYENGINE, 2010. Procedural Inc., City Engine website. [Online] www.procedural.com.

DEUSSEN O., HANRAHAN P., LINTERMANN B., MECH R., PHARR M., PRUSINKIEWICZ P., 1998. Realistic Modeling and Rendering of Plant Ecosystems. In: *SIGGRAPH '98:*

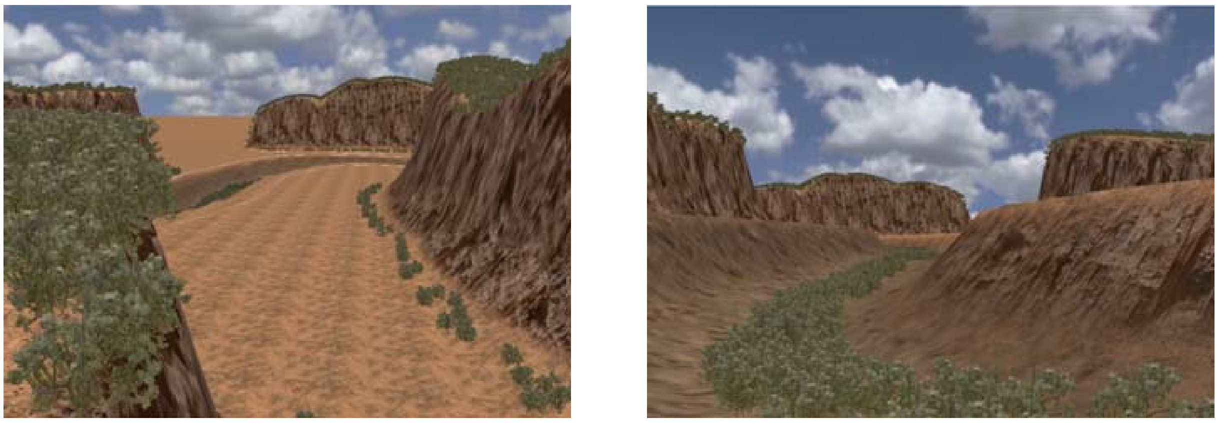


Fig. 14. Desert scene containing a few buttes and a dried out creek.

Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, NY, USA, 1998, pp. 275–286.

FINKENZELLER D., 2008. Detailed Building Facades. *IEEE Computer Graphics and Applications* 28(3) (2008) 58–66.

JONG S. DE, 2009. Game-Artist.net, Interview Sjoerd de Jong. [Online] www.game-artist.net/forums/spotlight-articles/1048-interview-level-design-sjoerd-hourences-de-jong.html.

MESTETSKII L.M., 2000. Fat Curves and Representation of Planar Figures. *Computers & Graphics* 24(1) (2000) 9–21.

MILLER, G.S.P., 1986. The Definition and Rendering of Terrain Maps. In: *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques,* ACM, New York, NY, USA, 1986, pp. 39–48.

MOORE G., 1965. Cramming More Components onto Integrated Circuits. *Electronics* 38(8) (1965).

MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L., 2006. Procedural Modeling of Buildings. In: *SIGGRAPH '06: Proceedings of the 33rd Annual Conference on Computer Graphics and Interactive Techniques,* ACM, New York, NY, USA, 2006, pp. 614–623.

MUSGRAVE F.K., 1993. Methods for Realistic Landscape Imaging. Ph.D. thesis, Yale University, New Haven, CT, USA (1993).

PARISH Y.I.H., MÜLLER P., 2001. Procedural Modeling of Cities. In: *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques,* ACM, New York, NY, USA, 2001, pp. 301–308.

PRUSINKIEWICZ P., LINDENMAYER A., 1990. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, NY, USA, 1990.

ROSENBERG J., 2010. GeoControl website. [Online] www.geocontrol2.com.

SKETCHAWORLD 2010. SketchaWorld website [Online] www.SketchaWorld.com.

SMELIK R.M., DE KRAKER K.J., TUTENEL T., BIDARRA R., GROENEWEGEN S.A., 2009. A Survey of Procedural Methods for Terrain Modelling. In: *Proceedings of the 2009 CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS),* Amsterdam, The Netherlands, pp. 25–34.

SPEEDTREE, 2010. IDV Inc. Speedtree website. [Online] www.speedtree.com.

SUN J., YU X., BACIU G., GREEN M., 2002. Template-based Generation of Road Networks for Virtual City Modeling. In: *VRST '02: Proceedings of the ACM Symposium on Virtual Reality Software and Technology,* ACM, New York, NY, USA, 2002, pp. 33–40.

WEISSTEIN E.W., 2010. Barycentric Coordinates. MathWorld - A Wolfram Web Resource. [Online] mathworld.wolfram.com/BarycentricCoordinates.html.

WONKA P., WIMMER M., SILLION F., RIBARSKY W., 2003. Instant Architecture. In: *SIGGRAPH '03: Proceedings of the 30th Annual Conference on Computer Graphics and Interactive Techniques,* ACM, New York, NY, USA, 2003, pp. 669–677.

XFROG, 2010. Greenworks website. [Online] www.xfrogdownloads.com.