Procedural Feature Generation for Volumetric Terrains

Rahul Dey
Bournemouth University & Sony Interactive Entertainment Euro R&D
13 Great Marlborough Street
London, UK W1F 7HP
i7697909@bournemouth.ac.uk

Jason G. Doig
Sony Interactive Entertainment Euro R&D
13 Great Marlborough Street
London, UK W1F 7HP
Jason.Doig@sony.com

Christos Gatzidis
Bournemouth University, Faculty of Science and Technology, Creative Technology Department
Poole House, Talbot Campus
Poole, Dorset, UK BH12 5BB
cgatzidis@bournemouth.ac.uk

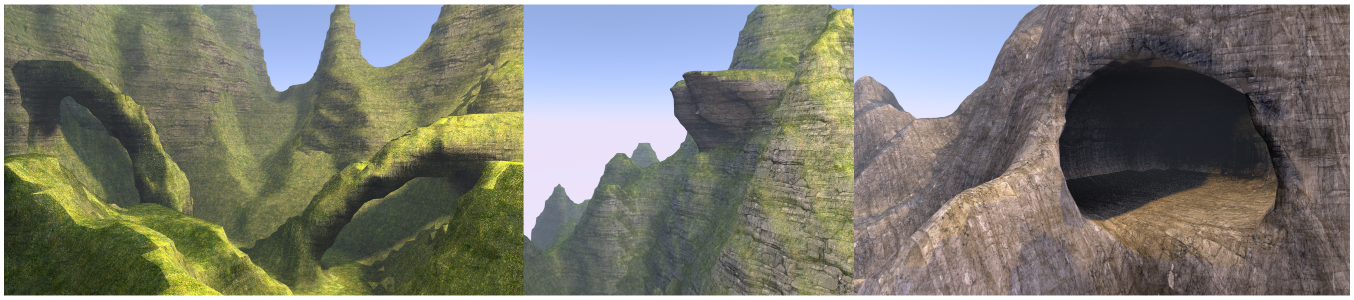

Figure 1: Procedurally generated features with our method: arches (left), an overhang (middle) and a cave (right).

ABSTRACT

In this work we present separate procedural methods to generate features that are found in natural terrains which are difficult to reproduce with heightmap-based methods. We approximate overhangs, arches and caves using procedural functions and a reduced set of parameters. This produces visually plausible terrain feature topologies as well as a high degree of artistic control. Our approach is more intuitive and art-directable than other existing volumetric methods that are more complex to integrate into existing voxel engines, due to the framework changes necessary, or rely on automatic procedural generation, thus reducing the ability to provide creative input.

CCS CONCEPTS

• **Theory of computation** → **Computational geometry**; • **Computing methodologies** → **Real-time simulation**; **Volumetric models**; *Shape modeling*;

KEYWORDS

Procedural generation, Terrain, Volumetric

ACM Reference format:
Rahul Dey, Jason G. Doig, and Christos Gatzidis. 2017. Procedural Feature Generation for Volumetric Terrains. In *Proceedings of SIGGRAPH '17 Posters, Los Angeles, CA, USA, July 30 - August 03, 2017,* 2 pages.
DOI: 10.1145/3102163.3102216

1 INTRODUCTION

Terrains are a key feature for portraying realistic virtual outdoor environments. Current methods often use heightmap-based terrain representations, where a 2D displacement map perturbs vertices on a polygonal grid. However, heightmaps do not allow for specific features of terrains such as cliffs, naturally-formed arches and caves. In contrast, volumetric representations of terrains are not limited in any such way and are gaining traction in practical applications such as procedurally generated computer games. This work presents procedural methods for generating features that are found in real-world terrains and can be complex to represent using heightmap-based methods.

Existing methods for creating overhangs have applied vector displacement to output vertices of heightmap-based terrains [Gamito and Musgrave 2001], though this work relies on unintuitive manipulation of the vector field data to achieve desired results. A hybrid approach introduced by [Peytavie et al. 2009] uses a signed-distance field to construct the terrain mesh. While implicit surfaces allow for many operations to be performed on the mesh, generating implicit functions can be computationally expensive. Recently, [Becher et al. 2017] presented the concept of voxelizing input feature curves to form the terrain mesh, allowing the formation of features such as overhangs and arches. However, our method generates a set of local delta values that can be applied directly to the volumetric data, which eliminates the need for a voxelization step.

2 OUR APPROACH

This work has been produced for integration into *Sony Interactive Entertainment's* proprietary game engine *PhyreEngine*^{TM} and proposes a procedural method for each of the three main features found in terrains by directly acting on the underlying volume representation. It extends prior work with voxel grammars where terrains are generated using a set of transforms [Dey et al. nd]. These transforms are a set of voxel replacements and the features described in this work here have been developed as more intuitive extensions. Overhangs and arches are functions that generate a set of local voxels that can be additively blended with the existing data to create the desired topologies. The cave generation method is a subtractive function utilising a particle-based approach that produces a set of negative density values removing voxels in the terrain (when blended). The user begins with either an empty or initialised voxel dataset. They can then add either overhangs, arches or caves which are layered on top of the existing data to ensure the editing is non-destructive. The parameters for the features are set, an optional transformation matrix (translation and rotation) is applied to the feature and the resulting voxel values are submitted to the GPU for surface extraction and rendering. Each feature has user-defined dimensions in voxels (F_{dim}) and the index of each voxel (I) being processed is calculated as the unit cube coordinates of the current voxel (V_{index}) within the feature's bounding box ($I = \frac{V_{index}}{F_{dim}}$).

2.1 Overhang Generation

Cliffs and overhangs are constructed by approximating their topology as a bicubic Bézier surface. A Bézier surface ($S(u, v)$) uses a 4x4 geometry matrix (G) of vectors containing the control points of the surface. The second and third rows of control points act as erosion parameters to the resulting overhang. The final surface function is combined with a parabolic function ($P(u)$) using a user-defined exponent (k) to emulate realistic plateaus and is shown in Equation 1, where $u = I_z$ and $v = 1 - I_y$.

$$\begin{aligned} U &= \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \\ V &= \begin{bmatrix} 1 & v & v^2 & v^3 \end{bmatrix} \\ B &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \\ S(u,v) &= U \cdot B \cdot G \cdot B^T \cdot V^T \\ P(u) &= (4u \cdot (1-u))^k \\ D_{Cliff}(u,v) &= \begin{cases} 1, & \text{if } I_x < min(S(u,v), P(u)) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

2.2 Arch Generation

Arches are created by sweeping a sphere along a cubic Bézier spline, with the four control points governing the resultant shape. Each end of the arch has optional tapering parameters (radius multipliers (M_i) and exponents (E_i)), as well as interpolants (I_i) to define where the tapering occurs. This results in feasible natural arches simulating greater erosion towards the middle of the arch. At each step of the sweep (t), the radius (r) is updated using the tapering parameters (T_i). This uses the function in Equation 2 where V is the voxel being processed and P is the position of a voxel after the radius has been applied at the current step.

$$\begin{aligned} T_i &= r \cdot M_i \cdot (1 - I_i^{E_i}) \\ radius &= r + T_0 + T_1 \\ D_{Arch}(V,P) &= \begin{cases} 1, & \text{if } \|P - V\| < radius \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

2.3 Cave Generation

In order to generate cave-like structures, we adopt a particle-based approach where a sphere is traced in a user-specified direction for a set number of iterations. A decay parameter (λ) reduces the radius of the sphere for each iteration. At each step (t), the direction (D) is further affected by gravity (G) and a velocity vector (U) generated by curl noise [Bridson et al. 2007]. This efficiently computes a viable estimation of speleological erosion, reducing the need for a complex fluid dynamics system sculpting the internal structure of the cave. The position (P) and radius (r) of the particle are used to check whether a voxel is within the sphere, and therefore subject to removal, using Equation 3.

$$\begin{aligned} P(t) &= P + t \cdot (D + G + U) \\ r(t) &= r - t\lambda \\ D_{Cave}(V,P,r) &= \begin{cases} 0, & \text{if } \|P - V\| < r \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{3}$$

3 EXPERIMENTAL RESULTS

We have implemented these methods using a sparse voxel buffer. However, any underlying volumetric representation can be used as our methods are independent of any data structure, meaning that integration into existing procedural volumetric engines is trivial assuming the voxel data can be read from and written to. The mesh is extracted from the volumetric data using *surface nets* [Gibson 1998]. Initial results can be seen in Figure 1 and show features integrated with a terrain generated using Perlin noise. For future work, we are developing GPU accelerated feature generators to create more detailed, large scale features at higher voxel resolutions and maintain interactive rates in order to maximise design efficiency.

4 ACKNOWLEDGEMENTS

We thank the EPSRC-funded *Centre of Digital Entertainment* project (EP/G037736/1) and *Sony Interactive Entertainment Euro R&D* for their support plus *NVIDIA Corporation* for hardware donations.

REFERENCES

Michael Becher, Michael Krone, Guido Reina, and Thomas Ertl. 2017. Feature-based Volumetric Terrain Generation. *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Article 10, 10:1–10:9 pages. DOI: https://doi.org/10.1145/3023368.3023383

Robert Bridson, Jim Hourihan, and Marcus Nordenstam. 2007. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics (TOG)* 26, 3, 46.

Rahul Dey, Jason G. Doig, and Christos Gatzidis. n.d.. Procedural Terrain Generation with Voxel Grammars. *Submitted to Computer Animation and Virtual Worlds* (n.d.).

Manuel N Gamito and F Kenton Musgrave. 2001. Procedural landscapes with overhangs. *10th Portuguese Computer Graphics Meeting* 2, 3.

Sarah Gibson. 1998. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. *Medical Image Computing and Computer-Assisted Intervention MICCAI98* (1998), 888–898.

Adrien Peytavie, Eric Galin, Jérôme Grosjean, and Stéphane Mérillou. 2009. Arches: a Framework for Modeling Complex Terrains. *Computer Graphics Forum* 28, 2 (Apr 2009), 457–467. DOI: https://doi.org/10.1111/j.1467-8659.2009.01385.x