

LUND UNIVERSITY

Procedural Generation of 3D Caves for Games on the GPU

Mark, Benjamin; Berechet, Tudor; Mahlmann, Tobias; Togelius, Julian

2015

Link to publication

Citation for published version (APA):
Mark, B., Berechet, T., Mahlmann, T., & Togelius, J. (2015). *Procedural Generation of 3D Caves for Games on the GPU*. Paper presented at Foundations of Digital Games, United States.

Total number of authors:
4

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 02. Nov. 2026

Procedural Generation of 3D Caves for Games on the GPU

Benjamin Mark
ITU Copenhagen
bmar@itu.dk

Tudor Berechet
ITU Copenhagen
tbde@itu.dk

Tobias Mahlmann
Lund University
tobias.mahlmann@lucs.lu.se

Julian Togelius
NY University
julian.togelius@nyu.edu

ABSTRACT

Procedural Content Generation in Games (PCG) is a thriving field of research and application. Recent presented examples range from levels, stories and race tracks to complete rulesets for games. However, there is not much research to date on procedural 3D modeling of caves, and similar enclosed natural spaces. In this paper, we present a modular pipeline to procedurally generate underground caves in real-time, to be used as part of larger landscapes in game worlds. We propose a three step approach, which can be fully implemented using General-Purpose Computing on Graphics Processing (GPGPU) technology: 1) an L-System to emulate the expanded cracks and passages which form cave structures in nature, 2) a noise-perturbed metaball approach for virtual 3D carving, and 3) a rendering component for isosurface extraction of the modeled voxel data, and further mesh enhancement through shader programming. We demonstrate how the interaction between these components produce results comparable to real world caves, and show that the solution is viable for video game environments. For this, we present the findings of a user study we conducted among indie-game developers and players, using our results.

1. INTRODUCTION

Nearly every game contains some sort of landscape. Sometimes it is an integral and functionally crucial part of gameplay, constraining and affording player action. Other times it is mere backdrop, or something in between. Large and costly efforts are therefore made during game production to provide believable, functional and enjoyable environments for the player.

The automatic generation of landscapes is relatively well-studied within procedural content generation (PCG). PCG refers to the algorithmic creation of game content with no or little human input; game content here could mean anything that is not part of the game engine itself, textures, levels, characters and stories, or the rules of the game itself [24].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FDG '15 Pacific Grove, CA USA
Copyright 2015 ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Here, we normally distinguish between *offline* (while the game is being developed) and *online* (while the game is being played) content generation. While the offline creation often allows more control over "what" procedural content makes it into the game, online creation augments the game with a wider variety of game content, adding replayability to the game, and offers the opportunity to make the creation process adaptive to the gameplay.

A special case of landscapes are underground worlds, i.e. caverns. While most our actual lives happen overground, it is surprising how many plots in literature, film, and especially games take place in underground locations. Underground cities, cavernous dungeons, or "the cave behind the waterfall" are common tropes, especially in the role-playing game genre. Recent notable examples, which make prominent use of caves, are *Skyrim* and *Dragon Age 2*. These games feature a range of caves, crafted by a human a designer, which are being re-used as dungeons or hideouts throughout the game world.

Though there have been notable publications concerning the procedural generation of 3D geometry (including landscapes), and also of maze-like angular environments ("dungeons") [23], little attention has been given to generating caves as game content. There are important reasons for including runtime generation of life-like caves in games, in particular introducing variety. Reusing caves or parts of caves puts a strain on the suspension of disbelief of the player. While the creation of a larger variety of caves by human designers is clearly possible, this would raise production time and cost significantly. Instead, we propose an algorithmic solution to this problem, providing an easy method for game makers to enrich their game worlds with a large variety of believable and interesting caves.

In this paper, we address the procedural generation of caves as part of a game world. In contrast to mimicking actual caves with all their properties and details, we focus on providing "believable" caves in the context of games, i.e. caves are believable as long as they "work" in a game, which in return is not that afar from actual caves. Our generated caves can stand as a game world on their own or be integrated into a larger context. In the following, we will outline in more detail the paradigms after which our caves are generated, and then present our methodology in the following. We will conclude this paper with presenting our findings and discussing the implications of our result.

1.1 Design choices

When approaching a PCG task, there are several prop-

erties to consider. From a recently proposed model for assessing content generators [26], we would like to discuss a relevant subset of properties for our solution. We choose believability and expressivity over reliability, i.e. we consider a wider range of caves, which don't break the immersion of the game, more desirable than hard constraints of control on the generator. The challenge therefore is to balance the procedures such that the results remain visually coherent and never cease to suspend disbelief in the portrayed environment. Artificial and "weird" shapes will exist and may be even desired, as long as they do not break the immersion. To maintain believability we will attempt to constrain the generator to a spectrum of shapes identified from a set of natural phenomena, which will be described in more detail in section 2.1.

Despite our focus on believability and expressivity, speed and controllability are also important to some degree. While very quick generation of caves is not required per se, online generation would require a high computational efficiency to produce the cave environment faster than the player can explore it. Furthermore, as not all types and shapes of caves are fit for every game environment, it is desirable to be able to tweak various parameters of the generation process to achieve just the style of cave that fits a specific application and style.

2. RELATED WORK

As we try to generate "believable" caves, it seems natural to review literature regarding the actual generation (in nature) of caves. Furthermore, we would like to set our work into context of previous attempts of 3D terrain generation.

2.1 Natural Caves

As mentioned in the introduction, one of our main goals is the creation of believable caves for games. Here, "believable" refers to a resemblance of natural caves, so that the player accepts the game world as "natural" in the context of the game they are playing. It is useful to compare the method we propose here with the features that compose a natural cave, and the processes by which natural caves develop. While it is beyond the scope of this paper to representatively survey the geological literature on caves, we would like to mention on two phenomena here: the creation of passageways in cracks, and the creation of stalagmites and stalactites.

While a general introduction into the creation of Phreatic- and Vadose-passages can be found in [3], we would like to mention the work by Boggus and Crawfis [2]. The authors presented a method to procedurally create caves that realistically mimic the overall structure of real cave networks. Their initial work used emulate acidic water erosion, but only used noise and 2D heightmaps, which are not sufficient for our problem of generating 3D caves. However, interesting here is their research on natural phenomena in the formation of natural caves. They identify that approximately 99% of caves are formed due to expansion of large pre-existing fractures or partings, and that the dissolution process, which creates the caverns, can form either Phreatic-passages, Vadose-passages, or a combination of both (Figure 2). Furthermore, patterns of these expanded passages in fractures can be seen in Figure 1. Another aspect of caves that we would like to recreate are stalagmites and stalactites. Again, a thorough discussion of the geological phenomenon is beyond the scope of our work. In short, stalagmites and

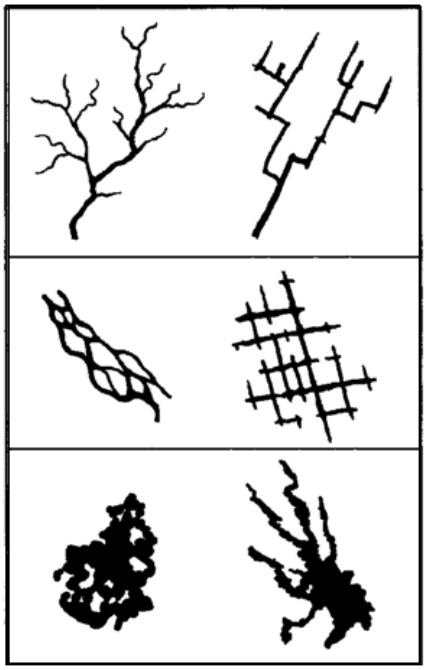


Figure 1: Common natural cave patterns [2, 20]

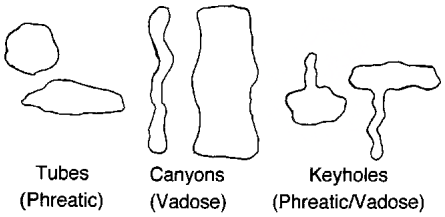


Figure 2: Front view slice of the main types of passages found in caves [7].

stalactites are accumulations of material through water drippings that grow over time from cave ceilings or floors. For a more thorough description, please refer to [10].

2.2 3D Terrain Generation

In contrast to 3D cave generation, the generation of 3D landscapes and flora is a well-studied problem. While the original L-System model [13] is certainly a form of landscape modelling already, discussing all approaches on the general topic would be beyond the scope of this paper. As a starting point on procedural terrain generation, we would like to refer to a recent survey paper by Smelik et al. from 2009 [25]. In summary, there exist three main approaches for terrain generation: fractal landscape modelling, physical erosion simulation and terrain synthesis from images or sample terrain patches [21]. We would like to point out though, that our work follows more the needs of the entertainment industry, which differ from geoscientific domain substantially: although they represent similar natural phenomena, the former one puts emphasis on interactive realistic visual appearance (or playability), the latter one focuses on the correctness from the geoscientific point of view [17].

The most notable approaches are probably heightmap-based methods, and many of the algorithms that are dedicated to generating heightmap-based terrain are based on a fractal approach [8, 1, 16]. But while applying heightmaps to two-dimensional meshes offer a compact and easy way of storing terrain data, a 3D terrain generator would require a voxel-based approach. In a voxel-based approach, data about the world is stored in a set of voxels that are positioned in a 3D grid. The main two benefits of this is that it is possible to effortlessly create caves and overhangs, and that data is available about each cube on the world grid.

Landscapes and even flora or buildings can be represented in volumetric textures created through fractals, grammars, or noise functions [18]. While this technique was popular in the early 90s in the gaming industry for rendering game worlds, it became a rather academical approach with the saturation of the consumer market with 3D accelerator hardware later that decade, which relies on 3D polygon geometry instead. However, the results of using noise to make high quality terrain, even with overhangs, has previously been documented [19]. However, specific arches and overhangs prove to be difficult to produce with just noise.

A more sophisticated solution is presented by Greeff [9]. This solution estimates erosion on landscapes, and generates rivers and mountains. However, to generate more advanced geometry like caves and overhangs, a mixed initiative PCG terrain system is developed, essentially leaving these complex features up to the designer to define. It is argued that a pure fractal or noise approach offers little control over the structure and desired features, and can only provide superficially realistic results.

Greeff also presents a method to allow a designer to model the overall flow of the landscape or to build a specific landmark at a specific location. The principle is inspired by how an artist or a designer would build the initial rough skeleton of a clay sculpture out of metal wires, to provide an overall structure onto which to apply the finer details.

While there are many publications on using erosion modelling to generate 3D landscape, we would exemplary like to mention the work by Kristof et al [12]. There, the authors used particle based hydrodynamics to model hydraulic erosion. But although the use of fluid simulation seems a potent technique to generate realistic looking caves, their design choice of using a 2D world model prevents the generation of actual underground caves.

Publications on 3D cave generation exist fewer, but are mainly voxel and grammar based [5, 4, 27] with the exception of Johnson et al. [11], who used cellular automata. However, Johnson focussed more and playability aspects than generating believable caves, which is closer to our work even though we're not generating assets with a specific game type in mind.

3. METHODOLOGY

The overall structure of the cave generation pipeline consists of 3 major components as follows:

1. **The structural component** uses an L-system to generate a set of structural points.
2. **The tunnel generation** builds the actual cave tunnels around the structural points.
3. **The renderer** extracts a mesh from the voxel data and applies materials and shaders.

(a) The basic alphabet

F	Move forward
R	Yaw clockwise
L	Yaw counterclockwise
U	Pitch up
D	Pitch down
O	Increase the angle
A	Decrease the angle
B	Step increase
S	Step decrease
Z	The tip of a branch
0	Stop connecting other branches
[ ]	Start/End branch

(b) The macros used

C	A curve
H	A vertical ascent that returns to horizontal
Q	A branching structure that generates a room
T	Similar to the H symbol, but splits into two curves
I	Represents a straight line

Table 1: The L-System

The pipeline loads structural points into a voxel volume in a video card's memory, where it is processed by a set of DirectX 11 Compute Shaders [15]. This includes a voxel carving shader and a stalactite growing shader. The latter runs over multiple frames as it uses Cellular Automata.

The split between the overall layout of the tunnels and the actual carving, allows both components to run autonomous from each other. This simplifies the overall process, and allows for interesting emergent behaviour when the two components interact.

3.1 Generating the Overall Structure

As seen in Figure 1, natural caves generally have an organic structure similar to that of vegetation. Due to this similarity, L-systems are the go-to method for generating vegetation, and were chosen to generate the structural framework of the cave. The L-system we used is a basic bracketed L-system, with the symbols seen in Table 1(a). Its alphabet is interpreted as drawing instructions for a virtual turtle (turtle graphics), with symbols controlling direction and movement. This turtle generates a number of structural points, which are traversed as seen in 3.2. Furthermore, the alphabet also allows for an alteration of the turtle's behaviour without the need for a fully fledged parametric L-system. On top of this basic alphabet, there is a set of macro symbols (Table 1(b)). These symbols encapsulate a string of basic alphabet symbols. Before each rewriting step, any macro symbol within a production rule is substituted by its sequence of axiomatic characters, which allows for a very simple representation of a long production rule. Each of these macros represent a different type of structure such as a curve, a straight line, a room or similar. The macros also add a stochastic element to the expansion of the L-system axiom, as each macro can encapsulate several different strings of symbols. This is shown in Figure 3, where two different strings, both encapsulated within the macro representing a room (Q), are drawn. When a macro encapsulates several strings like this, the string that replaces the macro at any given time is determined by a weighted randomised process.

The main purpose of macro symbols, apart from introducing a stochastic element, is to allow the representation

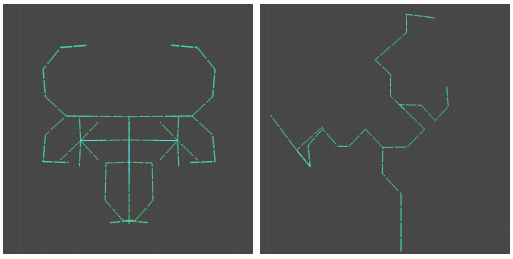
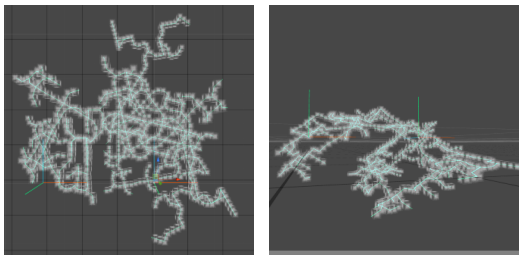


Figure 3: Two different room structures seen from above. Both are represented by the macro Q



(a) (b)

Figure 4: An L-system expanded from the randomly generated rule: $Z \rightarrow I[Q[C[T[TQ]]]]$. Seen from above (a) and the side (b)

of complex shapes with a single symbol. This prevents bloated production rules, while still retaining a certain level of control over the structure of the L-system. We favoured long production rules with few rewrites, generally 2-3, over shorter production rules: shorter rules with many rewrites appeared very ordered and self-similar in nature and resemble structures of plants. While caves also somewhat resemble vegetation, their structure can be much more chaotic and unstructured in nature, as seen in Figure 1. With a longer production rule and fewer rewrites, the self-similarity of the cave is reduced, but not eliminated, allowing for a more chaotic structure.

Even with the stochastic element described above, it is not possible to achieve the desired expressivity of the generator. To overcome this, the ability to randomly generate production rules was introduced. These randomly generated rules consist entirely of macro symbols, which are randomly chosen based on their associated weights, and brackets which are placed randomly at a frequency specified by the user. However, randomly generated rules expand the expressive range but at the cost of reliability. Nevertheless, with the way that the macro system is structured, we retain a certain level of reliability. As each macro symbol represents a set of structures and every randomly generated cave is simply a set of these macros, caves are guaranteed to be constructed out of meaningful building blocks (rather than a random set of instructions).

A problem with using L-systems to create the structure is that they're inherently tree-like in nature. This potentially results in a lot of dead ends, which could be a problem for gameplay (e.g. player frustration). Our approach to this is to connect a certain percentage of the dead ends with each other. The percentage of ends to connect is a user defined parameter, while the endpoints are selected randomly. Connections here are done by drawing a distorted line between them. The distortion is applied through curl noise, and turns the unnatural straight line into a winding tunnel 5.

To allow for control over the structure of the cave, the behaviour of the elements described above can be adjusted: production rules and macro strings and parameters such as the turtle's turning angle or the number of connected dead ends, can be configured. It is also possible to constrain the structure to a specific volume and to influence its direction. As mentioned, the user may specify the number of dead ends a cave may have. However, a location base selection of tunnels to connect is part of ongoing-work.

3.2 The warping Meta-Ball approach

A common approach, with the procedurally modelling of terrain in real-time on the GPU, is to apply various noise functions to the voxel data. In our approach, the voxels store values between -1 and 1, but the values are not determined by a noise function. The voxel values are instead populated by applying noise-distorted metaballs. The metaball moves through the voxel volume, jumping from one structural point to the next. The behaviour is analogous to a user applying various intersecting 3D brush strokes to a solid block of voxel terrain.

A metaball here is a smooth energy field, represented by a gradient of values between "empty" at its centre and "full" at its outer horizon. A spherical metaball however would not yield a satisfactory simulation of cave walls. Therefore, a warping function was created to perturb the metaball to match the outlines of the patterns identified from the studied natural phenomena.

This function runs in parallel on the GPU: for every voxel found under the radius of a metaball. It smoothly distorts the measured distance between the metaball's centre and the current voxel in order to artificially lower or increase the metaball's influence on the voxel. The function provides consistent distortion by using a combination of Simplex noise and Voronoi noise, applied to the worldspace coordinates of the voxels. A precomputed Curl noise value, stored inside each structural point, is also used to provide overarching variation, and to create very short or very tall tunnels. Figure 6 illustrates (in mesh form) some of the shapes that a metaball can take after being distorted by our function. Simplex noise and a layer of high frequency Voronoi noise were chosen to simulate scallops, and two more Voronoi noise layers of lower frequency and higher amplitudes were added to simulate sharp rocky outlines of cave walls.

The structural points are never further apart from each other than the minimum diameter of a warped metaball. This ensures tunnel connectivity, and intersecting metaballs further contribute to the emergence of a more advanced topology. Figure 7 shows an example of a voxel volume after a number of metaballs have partially been applied to the same region.

Parts of the L-System were designed to provide patterns which exhibit paths that intersect or go around each other, as well as spontaneously intersecting with other branches. This, coupled with the varying shape and size of the metaball, allows for various landmarks (e.g. columns, hoodoos) and more advanced shapes to emerge from the procedural carving.

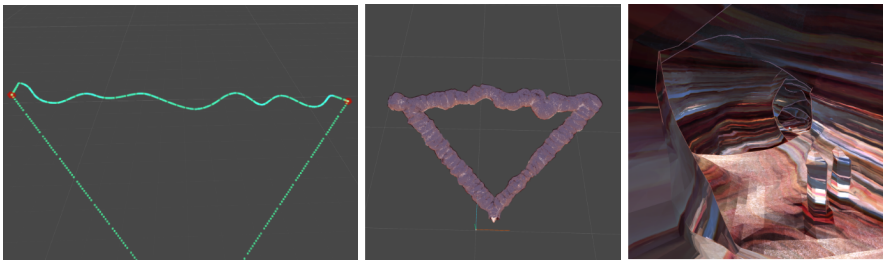


Figure 5: Two endpoints (red) connected via a curl noise in the 3D view and from inside the cave.

3.3 Populating the environment with objects based on noise and world position

Caves in games are rarely empty, hence we developed a method to populate them with objects (stalactites and stalagmites in this example):

1. A low frequency noise value is computed for each voxel, on the GPU, and any voxel's value that falls within a predefined noise range is picked as a potential spawn point. This essentially marks worldspace volumes as placement sites.

2. A high frequency noise value is computed for the voxels chosen above, and voxels falling within another predefined range are chosen as actual spawn points. The height of the frequency of this noise computation determines how densely populated the feature-pockets will be.

3. The object or feature is placed based on its type. For stalactites and stalagmites this involves using Cellular Automata for detecting the floor and ceiling of the cave, and then using CA again for growing these features from there.

The result of this method can be seen in Figure 8. While we only used this method to place stalactites, it could easily be extended to other types of objects, such as game relevant objects (e.g. treasures, monsters, etc.). A similar approach can be used on the fragment shader to display procedural decals or to provide worldspace progression between one type of texture and another.

3.4 Rendering

To display the cave voxel volumes in a state-of-the-art game engine, they ideally have to be converted into 3D geometry data. To achieve this, voxels created by the metaballs and exhibiting a sign change are processed by a *Marching Cubes* algorithm [14]. This was done to retain simplicity

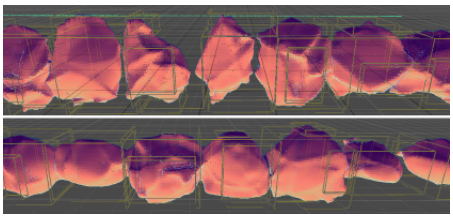


Figure 6: Voxel data created by warping metaball.

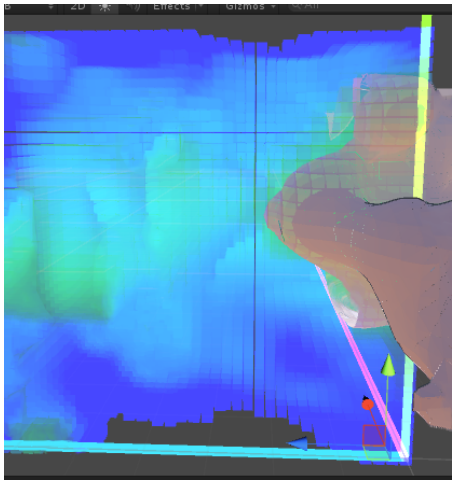


Figure 7: Volume of voxel data, after a series of metaball "brush strokes" have been applied to it. Green represents empty space, dark blue "filled" space.

and flexibiliy for our method, but could be replaced with a Dual Grid method, such as Dual Contouring [22].

Then, the mesh normals are calculated by sampling the density values of the neighbouring voxels. The flat normal of each mesh vertex inside the current Marching Cubes cube, is bent on each axis based on how "full" or how "empty" the neighbouring cubes on that axis are. This requires us to just sample the 6 neighbours of each cube, instead of 26. Further smoothing is achieved by averaging the normals of the current cube, by a factor of the face normals found in the neighbouring cubes already being analysed for the aforementioned method.

To apply textures to the resulting mesh, triplanar projection is used to texture the uv-less procedural meshes. To achieve a canyon stratification appearance, a tileable texture with parallel lines is used as a base, and is procedurally perturbed into a more interesting Solid Texture [6] before the lookups. This is done by perturbing each fragment's world position with a vertex-interpolated Curl noise value before sampling for the current planar projection.

As a final step, bump mapping is added by using the same principles and perturbing the interpolated normals before applying the Lambert lighting equation.

Parameters of the shader can be tweaked to dramatically modify the results, demonstrating versatility. For example, an ice texture can be achieved by just tweaking the frequency and amplitude of the Curl noise, reversing the bumpmap

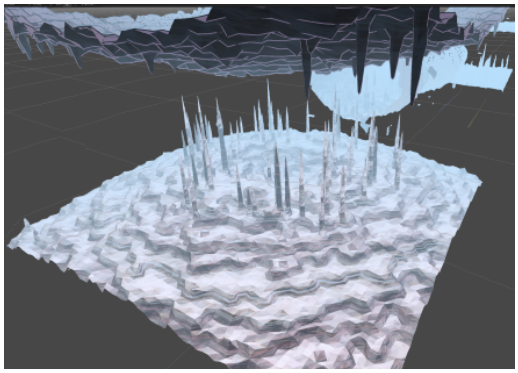


Figure 8: Stalactite and Stalagmite distribution and construction test run.

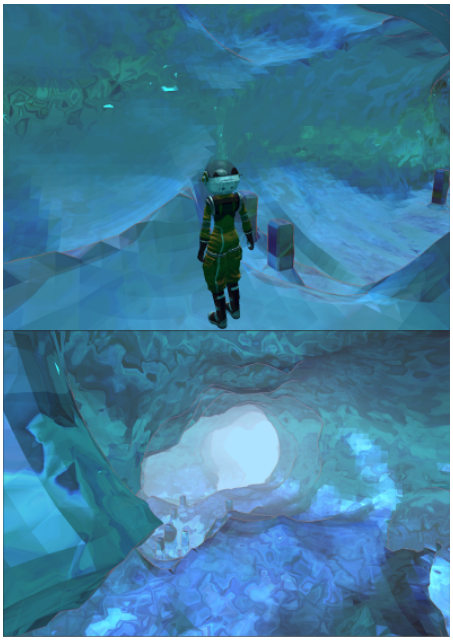


Figure 9: Changing the shader parameters can easily redress the caves.

and texture scale and colours, changing the scene's lighting. Furthermore, a simple refraction simulation was achieved by animating the seed value of the Curl noise based on camera rotation and movement (Figure 9).

4. RESULTS AND EVALUATION

To evaluate our work, we would like to discuss two points: the expressivity of the generator and its controllability, and how users perceived the generated caves, i.e. are our caves believable "enough".

4.1 Expressive Range of the Generator

A quantitative evaluation over the range of expressed features proved to be extremely difficult. Ideally, we would employ pattern recognition and comparing our caves with 3D data from real caves, but this is beyond the scope of our current work. Instead, we chose to discuss a few significant features below.

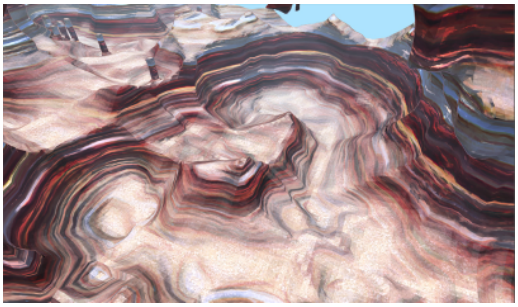


Figure 10: A generated feature which mimics overarching riverbed networks and islands.

The strength of the presented solution is the interaction between the structural and detailing components. Due to the differing shapes of a metaball on neighbouring structural points, multiple landscape patterns can emerge easily, e.g. hills or sharp peaks, plateaus or small mesas. Additionally, depending on whether the structural guidelines generated by the L-System overlap, or diverge, chasms, walkways, arches, and hoodoos appear, as well as cracks or windows through thin walls. As overlapping caves are explicitly desired, we implemented no measures to prevent crossing L-System branches. This however would require the L-System to be aware of the metaball size to limit the allowed symbols upon branch extension.

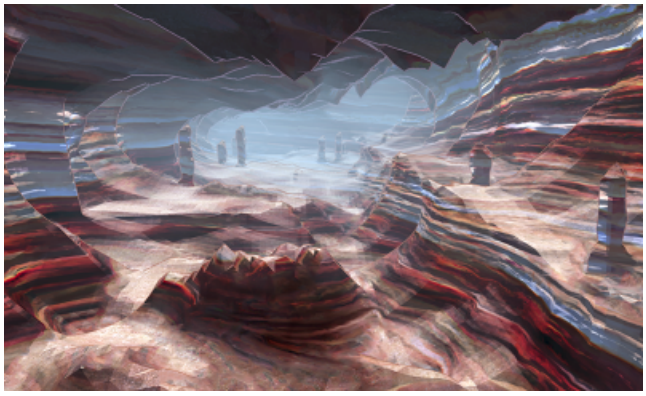
We have designed the L-system to augment the metaball detailing, by creating patterns for columns, skeletal structures for rooms etc. Figure 11, shows an example of this behaviour, where a relatively dense structural representation allows metaballs to overlap and create a long room structure. Additionally, by raising a single structural line slightly, a walkway is created on one side of the room. One might argue, that with the addition of more macros, a more skilled designer may even achieve more advanced results and a greater variety.

4.2 User Evaluation

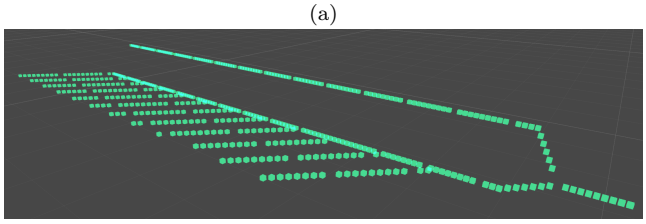
Ultimately, we are most interested in how well the generated caves stand as game content. To ascertain the viability of the generator as a part of an actual game, we gathered feedback from 30 individuals using an online survey. We recruited our participants from social media and game related websites, and users were presented with random cave environments they could explore. As the data was generated on the users' computer, each user saw a different set of caves.

The self-reported data indicated a general positive reception of our caves. Most participants found the caves natural looking. Furthermore, participants generally appreciated the shapes and art style. Also, the majority of users would be interested in playing a game using our environments as a part of it. Yet, it requires further research to isolate the impact of the caves' geometry from other aesthetic aspects like textures, control scheme, etc.

Even though most of the feedback was positive, some users suggested adding more variety, with a few users mentioning specifically what elements they perceived as repetitive. The first issue was the lack of very tight passages, as although local variation in tunnel size exists, the metaball has a certain minimum radius to prevent blockages. This limited its ability to create deep cracks and tiny corridors. The second



(a)



(b)

Figure 11: A room with a raised walkway (by the right wall) (a), created by the interaction between the structural representation (b) and the metaball.

issue identified by the users, was that the same texture was used throughout the entire cave. This lowers the perceived variety even if the shapes and patterns are varied. This issue can be minimized by including texture generation into the procedural content generation pipeline.

However, we theorize that further factors might have contributed to the generally lack of game resemblance, starting with the sparseness of our caves. Our environments were generally empty, despite a player's natural expectations of seeing a populated game world. Secondly, for the sake of a clear topology showcase, everything was evenly lit by a directional light. Certainly, these factors differentiate our experiment from an actual game, where for instance the play of light and shadow over the walls, might help make the caves appear more interesting and mysterious. Also, our caves' geometry aren't very complex compared to geometry found in modern AAA titles with billions of polygons.

5. DISCUSSION AND FUTURE WORK

Our solution presented in this paper is an effective way to generate believable detailed 3D underground landscapes. With our approach of designing a virtual 3D brush, and guiding it to carve out a cave, we can effectively immitate the shapes and patterns found in natural caves.Furthermore, the generally positive responses to our survey indicate that our approach is suitable for actual game projects, either as part of an asset pipeline or as part of a procedurally generated world.

As part of a standard asset pipeline ("offline generation"), our cave generation tool could speed up creation of caves for all types of games, in a similar fashion to what *SpeedTree* does for vegetation. As with most PCG solutions, this would increase the efficiency and variety in the content creation

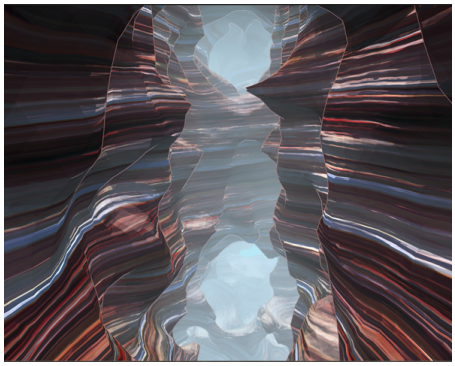


Figure 12: Vadose passage created by stacked parallel L-System lines.

process. In a mixed-initiative approach, a human designer could draw a guiding graph to edit the structural points currently generated by our L-System, that are then processed by the metaball, and would probably achieve best results.

If we think beyond the design process, our tool is also suitable for online landscape generation, in generally more procedural games such as *Minecraft*. So far, the world generation in Minecraft is composed of relative simple methods, as it is designed a discrete world made of cubes. For a high fidelity 3D game, a tool like the one presented in this paper, would be required for a more immersive experience.

However, depending on the particular use case a deployed version of our tool might require controllability beyond what we have implemented so far: while our caves follow some constraints, e.g. a limit of verticality through the restriction of the maximum possible slope that a tunnel can have, or the number of dead ends by connecting branches together, a human game designer may want to have more control over things such as difficulty of navigation, speed of traversal, and other gameplay related aspects.

On the implementation side, marching Cubes, even though it is a flexible and convenient mesh extraction solution, does not accurately represent the voxel topology, especially on fine details. Our future work will therefore focus on adapting our solution to Hermite data and a Dual Contouring or a Dual Marching Cubes method. We see two possible solutions with regard to handling Hermite data. The first, is to attempt to procedurally represent the expressive range of the noise-distorted metaball, into a set of constructive solid geometry operations. This would easily provide normals without intensive gradient computation. The second method is to pre-compute static volumes of the types of noise which are plugged into our metaball warping equation, as well as to pre-compute their corresponding gradient values. The metaball would then be distorted at runtime by 3D texture lookups, and the normals would similarly be looked up and then blended into the final warped surface used to carve the voxels with. Furthermore, to allow efficient online generation, the system has to rely on faster data structures such as octrees like used by Cui et al. [5].

As described above, several users found the caves to be lacking in variety. One solution for this is to introduce context-awareness into the structural generation. This could, for instance, be done by changing the L-system to be parametric, and introduce parameters that change various aspects of the detailing and rendering components. An ex-

ample is to change the texture gradually throughout the cave, to create different looking environments as the caves progress. A further enhancement would be to change tunnel size on a larger scale from within the L-system, instead of completely relying on the metaball's noise-based size variation. This would provide areas with very narrow tunnels and areas with cavernous tunnels without the risk of blockages.

Finally, a designer-friendly frontend for the metaball warping equation should be built. A user should be able to plug in their choice of perturbation noise types, and tweak their relationships, frequencies and amplitudes, while having a live preview of the resulting metaball's range of shapes. Similarly, an L-System frontend would be desirable, to visually customize the rules, macros and various parameters such as end-point connectivity, range of angles, volume and direction constraints etc.

6. REFERENCES

[1] D. A. Ashlock, S. P. Gent, and K. M. Bryden. Evolution of l-systems for compact virtual landscape generation. In *Proceedings of IEEE Congress on Evolutionary Computing*, pages 2760–2767, 2005.

[2] M. Boggus and R. Crawfis. Procedural Creation of 3D Solution Cave Models. The Ohio State University, 2009.

[3] J. H. Bretz. Vadose and phreatic features of limestone caverns. *The journal of Geology*, L(6):675–811, August 1942.

[4] J. Cui, Y.-W. Chow, and M. Zhang. Procedural generation of 3d cave models with stalactites and stalagmites. *IJCSNS*, 11(8):94, 2011.

[5] J. Cui, Y.-W. Chow, and M. Zhang. A voxel-based octree construction approach for procedural cave generation. *International Journal of Computer Science and Network Security*, 11(6):160–168, 2011.

[6] D. S. Ebert. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

[7] S. A. Engel and A. S. Engel. A geology and archeology field guide. *Rockcastle Karst Conservancy*, 1998.

[8] A. Fournier, D. Fussel, and L. Carpenter. Computer Rendering of Stochastic Models. Communications of the ACM, 25:371-384, 1982.

[9] G. Greeff. Interactive voxel terrain design using procedural techniques. Master's thesis, Stellenbosch University, 2009.

[10] F. L. Hicks. Formation and mineralogy of stalactites and stalagmites. *National Speleological Society Bulletin*, 12:63–72, 1950.

[11] L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 10. ACM, 2010.

[12] P. Krištof, B. Beneš, J. Křivánek, and O. Šťava. Hydraulic erosion using smoothed particle hydrodynamics. In *Computer Graphics Forum*, volume 28, pages 219–228. Wiley Online Library, 2009.

[13] A. Lindenmayer. Mathematical models for cellular interactions in development 1. filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280–299, 1968.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

[15] Microsoft Corporation. Direct Compute API.

[16] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The Synthesis and Rendering of Eroded Fractal Terrains. *Computer Graphics*, 23(3), 1989.

[17] M. Natali, E. Lidal, J. Parulek, I. Viola, and D. Patel. Modeling terrains and subsurface geology. *Proceedings of EuroGraphics 2013 State of the Art Reports (STARs)*, pages 155–173, 2013.

[18] F. Neyret. Synthesizing verdant landscapes using volumetric textures. In X. Pueyo and P. Schröder, editors, *Rendering Techniques '96*, Eurographics, pages 215–224. Springer Vienna, 1996.

[19] H. Nguyen. *GPU Gems 3*, chapter 1. Addison-Wesley Professional, 2007.

[20] A. N. Palmer. *Cave geology*, volume 454. Cave books Dayton, OH, 2007.

[21] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou. Arches: a framework for modeling complex terrains. *Computer Graphics Forum*, 28(2):457–467, 2009.

[22] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 70–76. IEEE, 2004.

[23] N. Shaker, A. Liapis, J. Togelius, R. Lopes, and R. Bidarra. Constructive generation methods for dungeons and levels. In N. Shaker, J. Togelius, and M. J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2014.

[24] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2014.

[25] R. M. Smelik, K. J. de Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen. A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009.

[26] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis. What is procedural content generation? mario on the borderline. In *Proceedings of the 2nd Workshop on Procedural Content Generation in Games*, 2011.

[27] T. Zawadzki, S. Nikiel, and K. Warszawski. Hybrid of shape grammar and morphing for procedural modeling of 3d caves. *Transactions in GIS*, 16(5):619–633, 2012.