SpeedRock: procedural rocks through grammars and evolution

Isaac M. Dart, Gabriele De Rossi and Julian Togelius
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
{imda, gdro, juto}@itu.dk

ABSTRACT

We present an approach to procedurally generating diverse and believable rocks for usage in games and virtual worlds. The basic idea is to evolve rulesets for three-dimensional L-systems. The fitness calculation involves expanding these rulesets a number of times, collapsing the resulting structure and evaluating how well the collapsed structure conforms to a user-specified shape. Texture is then applied through ray-casting from a sphere around the evolved "skeleton". The result is a lightweight, stand-alone tool for rock generation capable of exporting assets to mainstream modelling programs.

1. INTRODUCTION

This paper reports on the development of a software tool aimed to provide 3D artists with a method for rapidly generating different types of rocks using a search-based PCG technique based on an indirect representation.

Rapid development of realistic 3D rocks in game levels and movie scenes is a priority as rock generation is currently something of a bottleneck in 3D graphic artists' production pipelines. The development of realistic 3D rocks can take considerable time and skill, as seen in Sascha Henrich's 3DS Max tutorial [3] on environment modeling.

Although not the most glamorous type of game object, the humble rock can play a vital role in a game. For example, rocks can be used:

- To provide cover-spots in conflict based games
- As natural barriers during exploration
- As potential challenges (i.e. when rolling or falling)
- As enemies, such as in the classic video game Asteroids
- To enhance aesthetics for a more immersive gaming experience.

In order to provide artists with control over the shape of a desired rock we propose a novel approach based on a form of 3D L-system, artificial evolution and an implosion mechanism. In the SpeedRock system, an artist specifies certain parameters about how a rock should look and then the system will evolve a rock meeting these given shape specifications. L-systems have previously been used as genotype-to-phenotype mappings for search-based PCG [7] in e.g. [2, 4], but we believe that neither the specific form of L-system we are using here nor the implosion and skinning mechanisms have been used in a similar context before. We hope the method presented in this paper will enable anyone involved in 3D development to easily and quickly produce unique and interesting 3D rocks for use in their games and 3D scenes. We also urge the reader to download and try out the SpeedRock software in conjunction with reading this paper, as it is freely available online[1].

PCGames 2011 June 28, 2011, Bordeaux, France

1.1 Other approaches to procedural rock generation

There already some tools for procedural rock generation available.

The *aa_rockGen 1.0* [1] tool is a plug-in for 3D Studio Max. This tool gives the artist a rock of a preset shape which can then be adjusted manually to meet the shape requirements of an artist. In contrast, our tool creates the rock from scratch, thereby allowing greater variety and originality. It is also an independent tool rather than 3DS Max plugin.

In [5], Peytavie et al. describe a method for generating very impressive scenes composed of rocks and rock formations using a mixture of aperiodic tiling and erosion. This method is however more suited to rock piles than individual rocks, and there is no mention of a complete tool available for artists to use.

2. METHODS

SpeedRock is a simple application built using the game engine/development environment *Unity3D*. It presents the user with a simple interface allowing them to specify how a PGC rock should look. It also provides simple file output functionality, allowing the user to save the generated rock as an OBJ file for easy import into mainstream modelling programs.

The rocks are represented as axioms and rulesets for L-Systems as such representations have previously been shown to induce search spaces with high locality, permitting evolutionary algorithms to search for structures with a shape

[1]http://logicartists.com/logicartists/tool_links/speedrock/description.html

specified by the user [4]. The most common way of using L-systems for content generation and computational art is to interpret strings generated by L-system rewriting as instructions for a turtle mechanism that draws in 2 or 3 dimensions [6]; in contrast, we are here using the 3D structure resulting from rewriting directly as the "skeleton" for our rocks.

Each chromosome with the evolutionary algorithm consists of one axiom and four rules.

The SpeedRock chromosome is inspired by L-system grammar, expanded to 3 dimensions. The actual representation of the L-system rule is a 3 dimensional array of bytes, with the size of each dimension being 2. This represents a cube made up of 8 sub-cubes. Each byte value in a sub-cube holds a value representing another rule, an empty space (rule 0), or referencing itself.

For evolution to work well, the evolutionary algorithm needs a large amount of variation within the population. The population is therefore seeded with uniformly randomly generated chromosomes.

2.1 3D L-systems

The algorithm we created works by starting with a 3D cubic matrix as the L-system axiom, along with four randomly generated rules. Figure 1 shows an example 3D Matrix Division rule set.

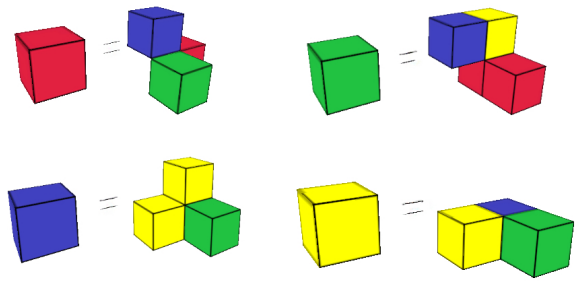

Figure 1: A sample ruleset for 3D Matrix division. The results of a single expansion of a block of each type (red, green, blue, yellow) is shown.

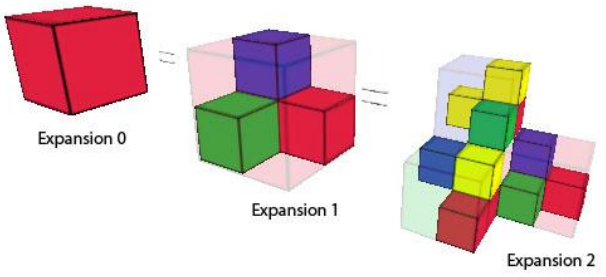

Figure 2: The results of two expansions of a red block of the same rule set.

We then rewrite the 3D matrix six times. In each iteration, each cube (cuboid) in the matrix is sub-divided along their 3 axis into 8 smaller cuboids, reflecting the rule of the parent cube. We call the approach of expanding the axiom according to the ruleset "3D matrix division". Figure 2 illustrates two expansions of one of the sets shown in figure 1.

After each expansion, if a cuboid is set as "Rule 0", it is treated as a gap in the rock and invisible, in any other case where the rule is non-zero it is treated as solid and interpreted as rock during rendering.

During initial testing of this concept, we found the results to be interesting, yet the variation in shapes was leaning towards cubic or triangular structures. The results also indicated that this technique may be suited for creating PGC Buildings, perhaps some crystalline silicate rocks, and man-made structures. See figure 3 for examples of structures created through repeated expansions of random rulesets.

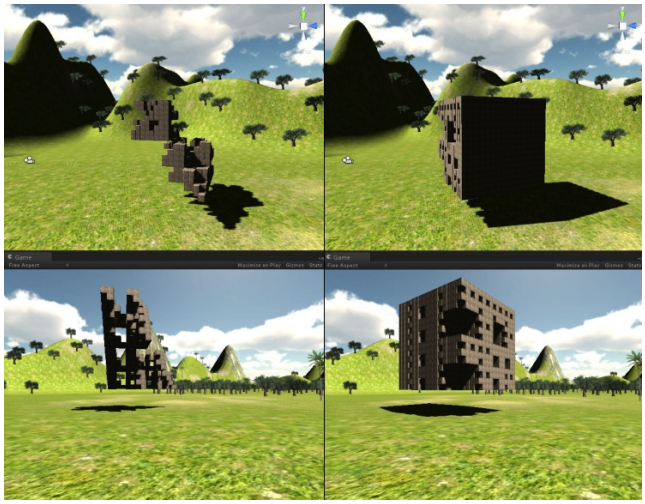

Figure 3: Some structures created through repeated expansion of random rulesets.

2.2 Rock implosion

After the rewriting/expansion phase, SpeedRock applies an "implosion" algorithm to the rock, forcing all cubes to move towards the centre of the cube until no internal empty space exists. Implosion works by repetitively searching through the expanded 3 dimensional rock and upon finding an empty space between a brick and the current center plane, pulling the brick into the emprty space. The 3 center planes, one for each dimension, run along the centre of the particular dimension being operated on. This method is entirely deterministic.

See figure 4 for examples of structures created through expansion of random rulesets followed by implosion. In the eyes of the designers, the imploded structures look less man-made than the correspinding unimploded structures.

2.3 Rock evolution

The variability of structures attained through the simple expansion and implosion processes described here is quite considerable, suggesting that the underlying representation of four rules and an axiom is well suited for evolutionary search. In SpeedRock, the user controls the generation chiefly through specifying parameters for the fitness function.

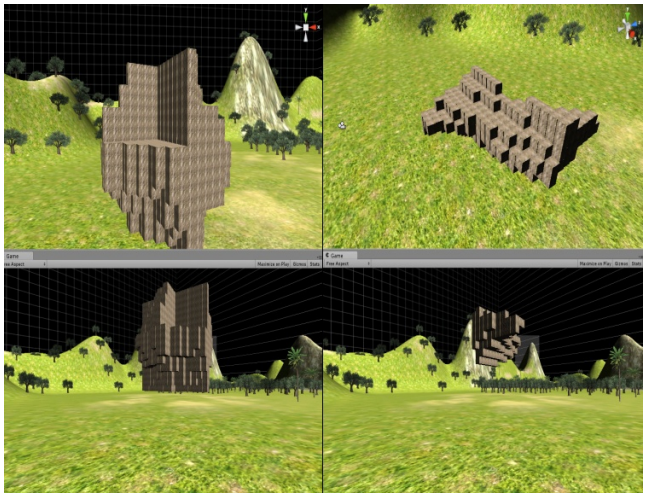

Figure 4: Some structures created through repeated expansion of random rulesets followed by implosion.

In initial experiments, a pre-existing shape and mesh was used as an input, with the fitness function analysing the difference between it and the candidate rock to determine its fitness. After further consideration, we decided that using a known shape would restrict creativity too much, and the requirement of having a "target mesh" would also place unnecessary demands on the user.

In the current version the user specifies the desired dimensions of the generated rock, getting an x:y:z ratio. For example, a flat stone would have a low y value, and higher z and x values, such as 5:1:4. The fitness function is simply defined as $1-d$, where d is the average normalised difference between desired and actual ratio in all dimensions.

Additionally, we decided to let the user specify some general criteria about the type of rock that they want before examining the fittest results. These criteria are are:

- Chunkiness value (size of each cuboid)
- Shape of individual cuboids: cubic / spherical
- Stochastic erosion percentage.
- Whether to implode the rock or not

The following evolutionary algorithm was used:

- Population size: 200
- Rank-based roulette wheel selection, where each gene gets the same number of wheel places as its ranking in descending order (i.e. fittest gene gets 200 places)
- One-Point random crossover.
- Mutation rate: 0.5% chance / rule
- Mutation: Chose a random cuboid within a rule. If cuboid is empty: flip it to a random rule. If cuboid is not empty: set it to empty.

We hypothesised that it would take longer to evolve a rock where fitness was determined after implosion or erosion. After running multiple simulations and finding the average fittest in each generation over 25 generations, we found little difference between methods. The algorithm automatically stops when no progress has been observed for the last few generations (usually after about ten generations).

2.4 Rock erosion

Erosion is an optional feature (the amount of erosion can be controlled with a slider between 0 and 100%) which is applied to the structure after evolution. The algorithm is inspired by [5] and resembles a one-step cellular automaton. The number of neighbours for each cuboid is calculated, and the probability of deleting the cuboid depends on the number of neighbours (45% if 4 neighbours, 65% if 3 etc) multiplied by the overall erosion amount.

2.5 Turning rock to mesh

Once the basic rock skeletal structure is evolved, it is converted into a mesh in order to be a exportable to a standard 3D file format. As the saying goes, there are many ways to skin a cat, and skinning a rock is no exception to this. Techniques we considered ranged from finding the centre of all cuboids and looking at the nearest neighbours to using external tools such as Autodesk and Blender.

The approach we settled on was to place the evolved rock "skeleton" inside a large sphere. We then iterate through each vertex on the sphere's perimeter and cast a ray towards the very centre of the sphere. Once the ray hits the rock skeleton we move the current vertex onto the hit location and add a small amount of displacement noise to the position. An analogy to this would be vacuum sealing a rock by putting it into a plastic bag and removing the air.

Finally, once all vertexes on the sphere's surface have been wrapped over the skeletal structure, we recalculate the normals of each surface. To give a sense of the kind of rocks produced by SpeedRock, figures 5 and 6 show two rocks that have been evolved with differing chunkiness values.

3. THE TOOL

Figure 7 shows a screenshot of the final tool, SpeedRock 1.0. The various options for the generator are available from a graphical user interface, as is an option to export the rock as an OBJ file for import into e.g. 3DStudio Max.

4. CONCLUSIONS

We have described a method for generating diverse and believable rocks using a combination of L-systems, evolution and an implosion mechanism. The method is embedded in a standalone tool which can export generated assets in a format suitable for modern game production pipelines. No user tests have been performed, but we believe the rocks we have generated to be reasonably believable. We believe that the direct usage of 3D L-systems is novel in the context of game content generation, and also that the implosion mechanism, while not an advanced feature in itself, has not been combined with structures resulting from grammar rewriting.

Acknowledgments

This research was supported in part by the Danish Research Agency project "Adaptive Game Content Creation using Computational Intelligence" (*AGameComIn*, 274-09-0083).

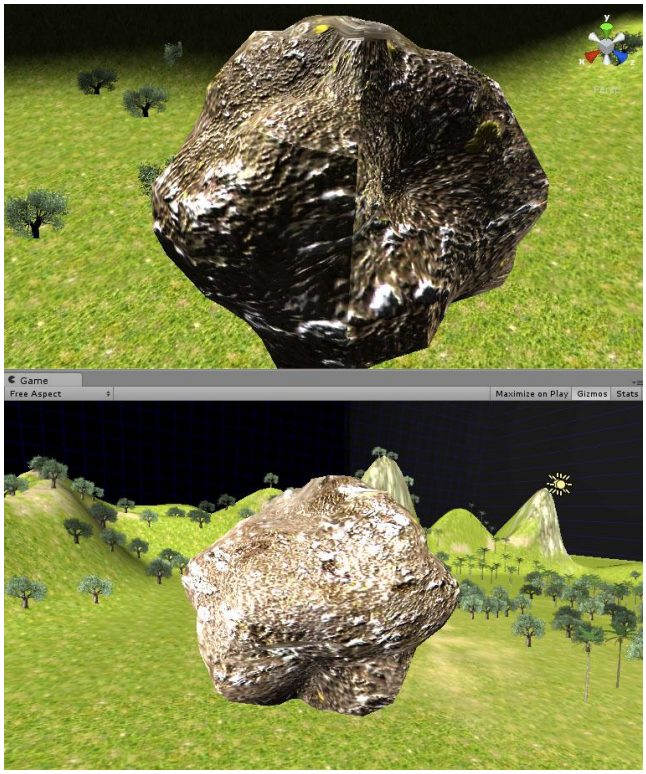

Figure 5: Skinned rock using high chunkiness value.

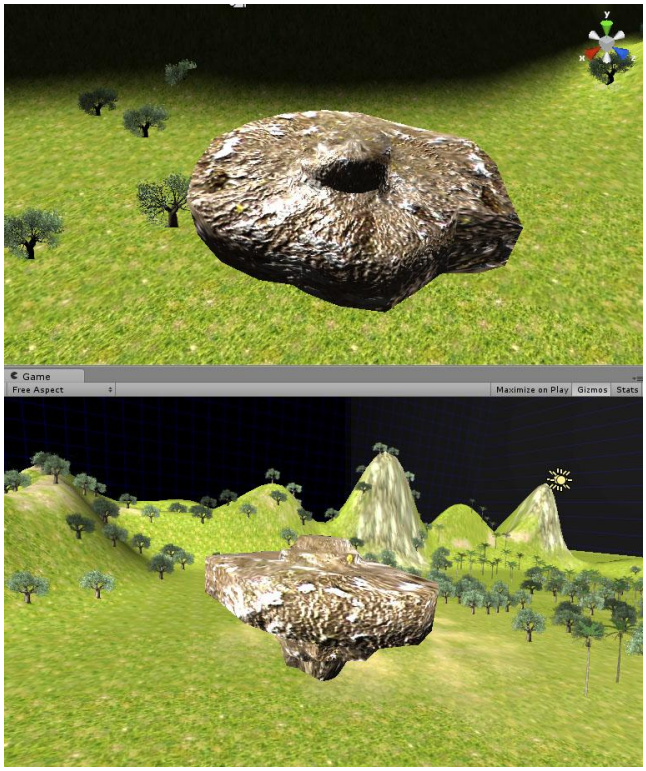

Figure 6: Skinned rock using medium chunkiness value.

5. REFERENCES

[1] A. Ardolino. Rock generator, 2010. http://www.scriptspot.com/3ds-max/scripts/rock-generator.

[2] D. A. Ashlock, S. P. Gent, and K. M. Bryden. Evolution of l-systems for compact virtual landscape generation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

[3] S. Henrichs. 3ds max environment modeling #1: Procedural stone, 2010. http://saschahenrichs.blogspot.com/2010/03/3ds Max-environment-modeling-1.html.

[4] G. Ochoa. On genetic algorithms and lindenmayer systems. In *Parallel Problem Solving From Nature*, pages 335–344, 1998.

[5] A. Peytavie, E. Galine, J. Grosjean, and S. Merillou. Procedural generation of rock piles using aperiodic tiling. *Computer Graphics Forum*, pages 1801–1809, 2009.

[6] P. Prusinkiewicz. Graphical applications of l-systems. In *Proceedings of Graphics Interface / Vision Interface*, pages 247–253, 1986.

[7] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation. In *Proceedings of EvoApplications*, volume 6024. Springer LNCS, 2010.

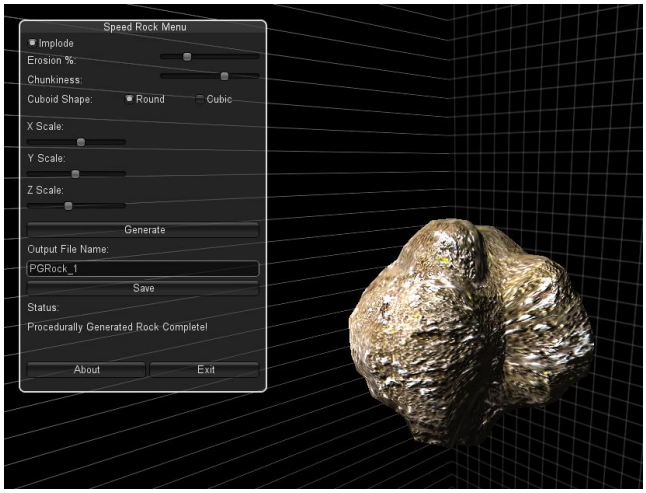

Figure 7: Screen shot of the final tool: SpeedRock v1.0.