Generative Methods

Kate Compton
Expressive Intelligence Studio
UC Santa Cruz
kcompton@soe.ucsc.edu

Joseph C. Osborn
Expressive Intelligence Studio
UC Santa Cruz
jcosborn@soe.ucsc.edu

Michael Mateas
Expressive Intelligence Studio
UC Santa Cruz
michaelm@soe.ucsc.edu

ABSTRACT

The field of procedural content generation continues to grow in scope and in technology, but the term "procedural content generation" awkwardly suggests that the field's output be defined by its ability to produce game "content", a term that fails to capture the breadth of artifacts produced by PCG researchers. There are many parallel fields of research on using algorithmic means to generate what we often call "content", but due to differences in terminology and industry they have remained invisible to the mainstream of games research. In this paper, we propose the term "generative methods" to refer to all such generative systems. This term relates our practice to similar work in generative art, generative music, generative design, and other fields, and reconnects game artifact generation problems to their historical (but non-game-specific) instances that have often been overlooked.

Categories and Subject Descriptors

Software and its engineering [**Software organization and properties**]: Contextual software domains Virtual worlds software [Interactive games]

General Terms

Algorithms, Design

Keywords

procedural content generation generative methods parametric

1. INTRODUCTION

From the (eventually abandoned) random star-fields of *Spacewar!*[2] to the automatic dungeon layout of *Rogue*, designers and programmers have often devised and deployed algorithms to synthesize in-game artifacts. Until recently isolated practitioners and academics lacked the consistent language necessary to discuss their practice. When a useful term was found in the field of computer graphics and popularized by an influential report, it allowed a community to form around the newly identified field of "procedural content generation".

Computer graphics had historically used the term "procedural" to describe process-derived artifacts, even including the process of rendering of 3D scenes [5]. Eventually, the term "procedural" came to refer, in the graphics community, to various approaches for synthesizing e.g. textures, models, and animations. When Intel published a technical report directed at game developers entitled "Procedural 3D Content Generation"[4], the phrase (with "3D" dropped) stuck, aided by the historical overlap between the games and graphics communities. This popular article surveyed the generative techniques of the past few decades, from fractals and L-systems to Perlin noise, and informed subsequent academic[3] and popular[14] discourse, even giving a name to this workshop.

As PCG has grown as a field, we have expanded outside the motivating problem of generic landscape synthesis provided by the Macri and Pallister report. PCG now encompasses—as a few examples—puzzle generation, level design, animation, quest design, and storytelling. The "content" of PCG has become problematic: the term is nebulous. Is a level content? Are enemy behaviors? What if they're defined in XML? It is also a notion which has, in the games literature, long been at odds with "rules". This is especially troubling when content is produced by systems of rules, or if, as in [10], rule systems are the artifacts being generated.

When the authors spoke with colleagues and practitioners about their respective conceptions of PCG, pinning down what was and was not PCG was a persistent issue. Often, merely changing the context of use could take the same algorithm from "not PCG" to "PCG": a physics simulation is not considered PCG, but is if it generates terrain. Even fields that clearly overlap with PCG—classical and game AI, storytelling and narrative synthesis—are often counted out, while automatic UV skinning is in.

In this paper, we contend that these problems are not inherent to the task of writing algorithms to produce artifacts; instead, they are incidental to a particular *way of seeing*: "Procedural content generation" privileges certain types of instantial asset—textures, models, text, terrain—over other sorts of things that could be generated (e.g. game rules or

player models). Further stifling interdisciplinary research, its exclusive use as a game- or graphics-centric term ignores the clear connections to dozens of academic and industrial traditions that exist outside of games and graphics.

PCG may remain useful as an industry term to describe a particular solution to game asset production (like its sibling "user-generated content"), but we argue that as an academic term it troubles and frustrates inquiry.

So what term do we substitute? Since all PCG approaches could usefully be described as *methods* which *generate* some artifact, we propose the term *Generative Methods*.

2. DEFINING GENERATIVE METHODS

Abstractly, a generative method is a function which produces artifacts (Fig. 1). Generative methods often begin with some set of *inputs* which may consist of user-provided tuning values, static assets (such as 3D models, story segments, or audio samples), or higher-level inputs like partially-configured artifacts or scripts. The *generator*, a core component of every generative method, then manipulates, composites, or combines these inputs to create a set of artifacts.

When speaking of artifacts, we refer to two relevant portions of Herbert Simon's influential definition: "Artificial things can be characterized in terms of functions, goals, adaptation" and they present an "interface" between interior substance and an exterior environment[6]. Put simply, an artifact is something which contains non-trivial structure (an interior) and a context in which it is used (an exterior).

The sine function is not a generative method since it produces only uninterpreted scalars, which have no interior substance and are not situated in an exterior environment: that is, we can project no goals onto the generation—the sine function has no inherent context or purpose. However, a generative method for determining camera movements might use a sine function as a core component, but situate the output in a "camera movement" of which the sine function's scalar output forms only a part. This movement can be interpreted in the context of visual expression: though the camera's movement is driven by a sine wave, its context of use demands an evaluation of framing, composition, and our interpretation of its "intent" when it gives an unexpected closeup of a particular character.

Very often a generative method will include a *critic* which validates the generator's proposed artifact, releasing it into its surrounding context or providing it as input to another generative method. The critic can also give the generator an explicit accept/reject decision, a reward value, or a corrected artifact to accommodate generators which incorporate feedback. This computational critic could range in complexity from a simple collision detector to a heuristic optimizer capable of returning an improved version of the artifact.

The generator, assets and critic may themselves be composed of smaller systems, including other generative methods; for example, a level generator's critic could apply a set of generated player models to the level under consideration. Critics can be human as well as computational: Human critics are exceptionally good at providing aesthetic heuristics

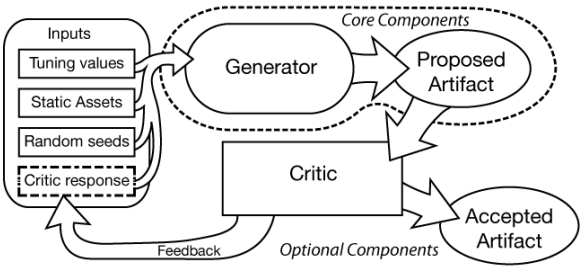

Figure 1: A generator combines several inputs into an artifact which is then evaluated by a critic. This evaluation is provided as additional input to the generator.

(e.g. beauty, uniqueness, interestingness) that would otherwise be inaccessible to a generator. Human generators combined with computational critics could form the basis of novel approaches to user-generated content.

Our purpose in replacing "procedural content generation" with "generative methods" is not to encompass every way that an algorithm can produce output, but to construct a useful framework for understanding the synthesis of artifacts. The definitions and examples given for generative methods are not meant to draw a hard line between generativity and non-generativity. Instead, "generative methods" provides a lens for examining systems that emphasizes the *interplay* between inputs, algorithms, and the expressivity and correctness of the outputs.

"Generative methods" integrates better into existing scholarly and industrial discourses, many of which already use the term "generative" to indicate algorithmic approaches to their respective fields. This makes it easier to search for and find relevant work. Searching the literature for any combination of the terms "procedural X generation" is not likely to return many results prior to the year 2000; that would miss decades of study in generative music, parametric architecture, generative art, generative methods in programmed instruction, generative programming, generative grammars for computational linguistics, and generative models for computer-aided industrial design—not to mention the non-digital traditions of Dada, Pollack, cut-up, and other human-enacted algorithms. In fact, among the approaches which address computation, "generative" is the term of choice—not "procedural"—in nearly every discipline.

3. FINDING NEW COMMUNITY

If the term "generative methods" includes disciplines outside of the discourse on games, how does the diligent researcher make an inquiry into these newly-available fields?

From the perspective of generated artifacts, our hypothetical researcher could seek out systems that produce artifacts similar to the ones desired. This can be an easy task for game artifacts with analogues in non-game contexts: Generative game music is already well-linked to the generative music community, and much of the generated terrain work in games can trace its roots to generative art and computer graphics traditions of fractals and context-free gram-

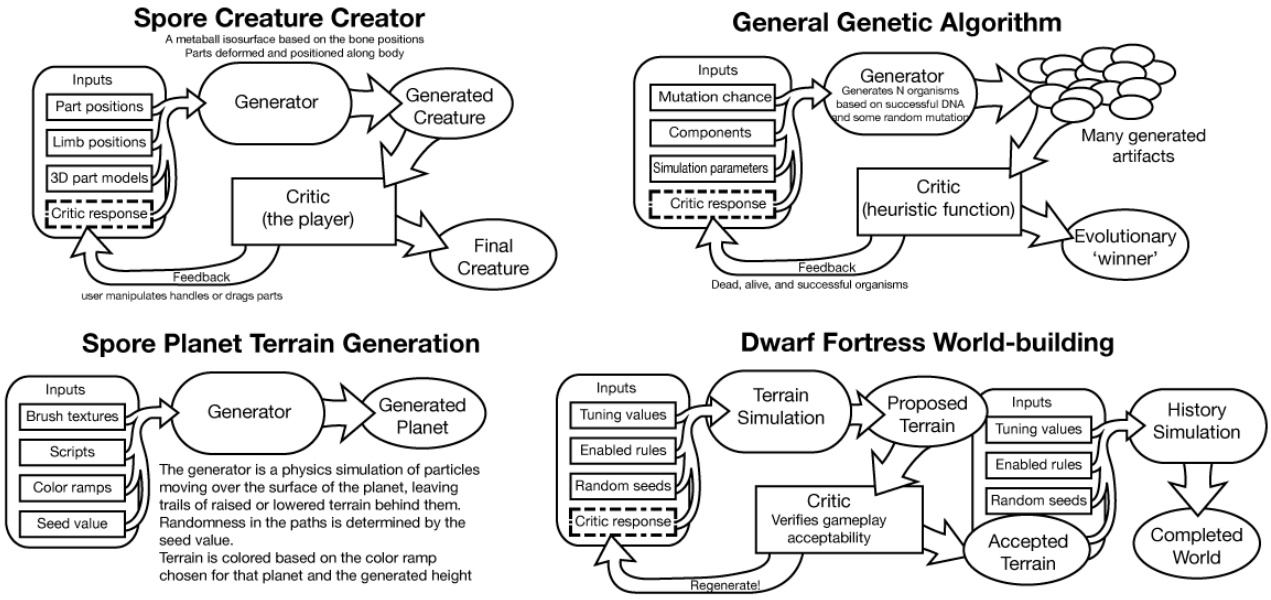

Figure 2: Several generative methods. Each follows the schema of Fig. 1.

mars; but there remain many under-examined connections between research areas in generative methods.

Some areas of PCG research may not have clear parallels in existing generative methods traditions; in those cases, we can work from the perspective of shared constraints. Considering "procedural level design", there is no tradition of level design outside of games as there are no "levels" in the natural world; but there are already established subfields in architecture, urban planning, and puzzle generation for computer-based learning (which coincidentally has a lot to say about user-adaptation). Generative puzzle design probably has at least as much in common with the hard constraints of industrial design as it does with the softer constraints of 3D model generation and texturing. The lens of generative methods produces useful connections to other areas of inquiry, even when the generated artifacts are different.

Examining mature fields of research in non-game generative methods can also suggest clear links to current PCG research. Consider space planning, a category of problems concerned with placing objects in the correct spots in a room (or rooms in a building, or buildings on a city block) such that the desired objects are in the room, that they have some physical relationships with respect to each other, and that other constraints like the traversability of the room are maintained. Generative methods in space planning have been an active area of research from the 1970s[1] to the modern day[12], but similar parallel research in games and graphics since 2005 has used the term "procedural interior generation"[11] and barely refers to this established discourse. The new term isn't only harder to search the literature for: it cuts the concept off from its own past and present.

We argue that by acknowledging both the similarities and differences in generating particular types of artifacts across two fields, practitioners and academics from both sides can make informed decisions on borrowing techniques and approaches from each other. Furthermore, understanding the values of another discipline could lead to new ways of thinking about both. By sharing knowledge across fields of practice, we begin to discover common threads that run through all generative methods.

4. COMMON THREADS

The lens of generative methods helps us see a broader and more interesting set of candidate systems for analysis and synthesis—including systems which do not fall within traditional definitions of PCG[9] (Table 1). By viewing these examples as generative methods (i.e., in the schema of Fig. 1), a researcher can make a quick visual comparison of what the systems' respective inputs are, what sorts of generators they use, what artifacts they produce, and what forms of critics and feedback loops they employ (Fig. 2).

Diablo dungeons	**Rogue-likes**	**Spore**
Galactic Arms Race	**Elite**	WolframTones
Mirrorgram	Cell Cycle	Sodaplay
Conway's Game of Life	Revit	Eliza
Context Free Art	Electric Sheep	Bubble Harp

Table 1: Systems in bold are traditional "PCG", but all of these examples employ generative methods.

Besides comparing the topologies of generative methods, researchers can also compare the respective algorithms driving their generators or critics. Every implementation strategy—e.g. a satisfiability solver, a production grammar, or a genetic algorithm—carries a distinct set of affordances. Specific artifact use-cases often have distinctly hard or soft constraints: a system producing artifacts to satisfy soft constraints can more readily use optimization techniques like genetic algorithms, but systems with hard constraints (where a single mistake will result in a broken artifact) are better

off using constraint solvers.

There are many concerns and issues that appear in nearly all practical implementations of generative methods, and shared problems can suggest shared approaches:

- Granularity. How fully-formed are the inputs to the generative method? Does this method assemble hundreds of small and generic units, or does it piece together large and complex components that have already been partially authored or configured? Is a text generator starting from letters, words, sentences, or hand-written scenes?
- Expressive range. It is a common goal of generative methods to generate a wide and surprising range of artifacts. What factors determine how broad an expressive range will be? What parameters are tuned to eliminate bad artifacts without pruning away desirable ones? Granularity is a relevant factor, but other properties are involved; research here is only just beginning[7].
- Adaptability. A key feature of generative systems is that, with sufficient computing power, they can be run in real-time (or at least on demand), and therefore can generate artifacts in response to user actions. As tantalizing as a responsive and adaptable system is, we often forget how hard it is to define *how* such a system should adapt, a difficulty pointed out in one of the earliest dynamic difficulty papers[13].
- Hard and soft failures. We rarely talk about the failures of our generative methods, instead using only the most flattering examples of output in our papers. But generative systems often make bad, broken, or suboptimal work, and an ugly artifact and a broken artifact can have very different effects depending on their context of use. Understanding the system's resilience to failure and its coping mechanisms in case of failure will allow more generative methods to be deployed in unsupervised situations like games—and, potentially, new application domains in other fields.

5. CONCLUSIONS

We hope that by adopting the more useful phrase "generative methods" in place of "procedural content generation", game developers and academics can discover and reach out to generative methods practitioners in other fields. We can work across fields, with computational linguists working alongside game dialog systems experts; and across types of artifact, with rhythm-based level generators (like Launchpad[8]) borrowing the rhythm generators invented by generative music researchers. We would like to see more game developers attending generative methods conferences in other fields, and vice-versa.

One way to measure the success of this project is whether this workshop is, in the future, named "Generative methods for games". More seriously, we hope that generative methods can be seen as worthy of inquiry in their own right, so that generative methods papers in games no longer have to begin with an appeal to cost-saving or workflow improvement. There is much more to generative methods than merely optimizing traditional development processes, and the vibrant academic discourse in other fields echoes that assertion. Generative music composition, for example, is not grounded in the economical production of music, but in the basic theory of the aesthetics and mathematics of music.

Using the right terms is only the first step. By deciding now to integrate ourselves into existing discourses on generative methods, we can grow with these other fields and multiply our effort, rather than continually reinventing the discoveries of the past fifty years.

6. REFERENCES

[1] C. M. Eastman. Representations for space planning. *Communications of the ACM*, 13(4):242–250, 1970.

[2] J. M. Graetz. The origin of spacewar. *Creative Computing*, 7(8):56–67, 1981.

[3] S. Greuter, J. Parker, N. Stewart, and G. Leach. Real-time procedural generation of 'pseudo infinite' cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '03, New York, NY, USA, 2003. ACM.

[4] D. Macri and K. Pallister. Procedural 3d content generation. Technical report, Intel Developer Service, 2000.

[5] D. F. Rogers et al. *Procedural elements for computer graphics*, volume 103. McGraw-Hill New York, 1985.

[6] H. A. Simon. *The sciences of the artificial*. MIT press, 1969.

[7] G. Smith and J. Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 4. ACM, 2010.

[8] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha. Launchpad: A rhythm-based level generator for 2-d platformers. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):1–16, 2011.

[9] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation. *Applications of Evolutionary Computation*, pages 141–150, 2010.

[10] M. Treanor, B. Schweizer, I. Bogost, and M. Mateas. The micro-rhetorics of game-o-matic. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 18–25. ACM, 2012.

[11] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker. Rule-based layout solving and its application to procedural interior generation. In *CASA Workshop on 3D Advanced Media In Gaming And Simulation*, 2009.

[12] M. Verma and M. K. Thakur. Architectural space planning using genetic algorithms. In *The 2nd International Conference on Computer and Automation Engineering*, volume 2, pages 268–275. IEEE, 2010.

[13] J. Wexler. A teaching program that generates simple arithmetic problems. *International Journal of Man-Machine Studies*, 2(1):1–27, 1970.

[14] W. Wright. The future of content. In *Game Developers Conference*, 2005.