Procedural Content Generation for Games: A Survey

Article *in* ACM Transactions on Multimedia Computing, Communications and Applications · February 2013

DOI: 10.1145/2422956.2422957

Alexandru Iosup
VU University Amsterdam, Amsterdam
250 PUBLICATIONS **8,392** CITATIONS

Procedural Content Generation for Games: A Survey

MARK HENDRIKX, SEBASTIAAN MEIJER, JOERI VAN DER VELDEN, and ALEXANDRU IOSUP
Delft University of Technology, the Netherlands

Hundreds of millions of people play computer games every day. For them, game content–from 3D objects to abstract puzzles–plays a major entertainment role. Manual labor has so far ensured that the quality and quantity of game content matched the demands of the playing community, but is facing new scalability challenges due to the exponential growth over the last decade of both the gamer population and the production costs. Procedural Content Generation for Games (PCG-G) may address these challenges by automating, or aiding in, game content generation. PCG-G is difficult, since the generator has to create the content, satisfy constraints imposed by the artist, and return interesting instances for gamers. Despite a large body of research focusing on PCG-G, particularly over the past decade, ours is the first comprehensive survey of the field of PCG-G. We first introduce a comprehensive, six-layered taxonomy of game content: bits, space, systems, scenarios, design, and derived. Second, we survey the methods used across the whole field of PCG-G from a large research body. Third, we map PCG-G methods to game content layers; it turns out that many of the methods used to generate game content from one layer can be used to generate content from another. We also survey the use of methods in practice, that is, in commercial or prototype games. Fourth and last, we discuss several directions for future research in PCG-G, which we believe deserve close attention in the near future.

Categories and Subject Descriptors: A.1 [**General Literature**] Introductory and Survey; K.8.0 [**Personal Computing**]: General—*Games*; I.2.4 [**Computing Methodologies**]: Knowledge Representation Formalisms and Methods—*Representations, procedural*; *Representations, rule-based*; H.4.0 [**Information Systems Applications**] General

General Terms: Theory, Design, Standardization, Algorithms

Additional Key Words and Phrases: game content generation, procedural, survey

ACM Reference Format:

Hendrikx, M., Meijer, S., van der Velden, J., and Iosup, A., 2012. Procedural Game Content Generation: A Survey. ACM Trans. Multimedia Comput. Commun. Appl. -, -, Article 1 (January 2012), 24 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Computer games are increasingly present in our lives. Every day, hundreds of millions of players around the world are entertained by games such as FarmVille, World of Warcraft, Call of Duty, The Sims, and StarCraft. The US Entertainment Software Association (ESA) reports [ESA 2010] that, in 2009, over 68% of the American households play computer or video games; inside these households, the average game player age is 35, and the average experience with computer or video games is 12 years. The same ESA reports show a growing use of computer games in US households over the past decade. Game content is an important factor in keeping players engaged in gaming worlds. We take for granted the ability of computer games to present players with engaging content, whereas demand for

Author's address: A. Iosup, Mekelweg 4, 2628CD, Delft, the Netherlands. Email: A.Iosup@tudelft.nl (Corresponding author.)

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

new and even player-customized content keeps increasing while manual content production is already expensive [Takatsuki 2007] and unscalable [Iosup 2011]. In contrast to manual content production, Procedural Content Generation for Games (PCG-G) is the application of computers to generate game content, distinguish interesting instances among the ones generated, and select entertaining instances on behalf of the players. PCG-G is difficult: generating most types of game content requires from a computer not only computational power, but also the ability to judge the technical and cultural values of the generated instances. It is not surprising that, despite over three decades of research and development, the community has yet to design a general-purpose procedural game content generator. In this article we conduct a systematic survey of PCG-G techniques and their application in practice.

The most popular commercial games get larger, prettier, more atmospheric, and more detailed with each generation. For example, the massively multiplayer online role-playing game (MMORPG) World of Warcraft (WoW) immerses its players into a virtual fantasy world of great complexity. Each of the two main continents in this world is about the size of Manhattan and includes a detailed landscape, a diverse ecosystem, complex renditions of fantasy towns, and a comprehensive road network. The game features many types of content, such as sound, textures, terrains, buildings, cities, object behavior, and game scenarios. In total, WoW was comprised in 2008 [ComplexityGaming.com 2008] of over 1,400 interesting geographic locations, 30,000 items, 5,300 interactable creatures, 7,600 quests, and 2,000,000 words of text; this content was produced during almost five years of development.

Today, the production of high-quality commercial games may require the work of a few hundred people, including artists, designers, programmers, and audio engineers, many of whom work on producing game content. As a consequence, content production has grown to the point at which it has become a bottleneck in both game budgets and product time-to-market [Kelly and McCabe 2007; Lefebvre and Neyret 2003; Smelik et al. 2009; Iosup 2009]. In the early-to-mid-1990s, the successful game development company `id` created popular commercial games such as Commander Keen, Wolfenstein 3D, and Doom with a single full-time content developer in a team of five or six game developers [Kushner 2003]. In the early 2000s, game production teams already employed hundreds of people [Krueger et al. 2005]. An increasing growth trend for both production cost and team size with time was observed as early as 2005 [Krueger et al. 2005]. The actual production budgets remain unknown for many of the popular commercial games, but educated guesses place the budgets of contemporary games such as WoW at $20,000,000–$150,000,000 [Johnson 2006; Reynolds 2010b; Video Game Sales Wiki 2009]. Anecdotal evidence suggests an increase in the fraction of the game production costs spent on generating content, toward 30-40% of the game budget [Irish 2005, p.231] [Irish 2005, p.281] [Elas 2010]. Apart from the cost, finding qualified people has become a major problem as early as 2005 [Krueger et al. 2005], even when considering the global workforce [Fields 2010, p.99]. Fast-growing production costs have already been at least partially responsible for the dissolution of once successful game companies such as Interplay and Factor 5. Ultimately, the current situation of game content production means that fresh content cannot be not produced anymore at the rate and player-customization that gamers would want, and for the cost that would keep games affordable for their current gamer communities.

Procedural techniques are an alternative to making complex game worlds in a limited amount of time without putting a large burden on the game content designers. The main idea behind procedural content generation is that game content is not generated manually by human designers, but by computers executing a well-defined procedure. To avoid losing control over the design process, it is desirable that artists and designers can still influence the final product by adjusting the parameters of the procedure.

Following three decades of research, many methods exist for procedurally generating many types of game content. One of the simplest and earliest approaches to procedural game content generation is based on pseudo-random number generation (PRNG). The space exploration game Elite used this approach in the 1980s to generate a very large universe. A similar approach can be used to generate textures for game objects [Perlin 1985]. The Demoscene, a loose community of artists and coders, has used procedural techniques since the mid-1980s to create vast virtual environments with small disk space requirements. The game ".kkrieger" [Farbrausch Prod. 2006] is a 3D first-person shooter, similar in genre to Halo 3 (a $20,000,000-plus budget game). Having Demoscene-like goals, ".kkrieger" uses procedural techniques to generate textures, meshes, and sounds that are used to create a complex, immersive game. This game requires under 100 KB of disk storage—three to four orders of magnitude less than a similar game. Can the techniques employed by ".kkrieger" be used by other games? Research such as 3D model streaming [Mondet et al. 2009] may benefit from the techniques employed by ".kkrieger", and vice-versa. What other techniques can be used to generate the content present in ".kkrieger"? Some techniques may focus on stochastic outcome [Ashlock 2010; Edwards 2011], rather than the deterministic outcome present in Elite and ".kkrieger"; other [Chen et al. 2008; Smith et al. 2010; Smelik et al. 2010] may focus on human-aided content generation.

Despite an abundance of PCG-G techniques, their application in commercial products is not mainstream. PCG-G methods have been successfully applied to generate many game content types (see Section 2), but the existing solutions are not general-purpose. The literature on procedural game content generation is scattered across numerous disciplines (computer graphics, image processing, artificial intelligence, computer-human interfaces, psychology, linguistics, social sciences, ludology, etc.) and publication venues. In this sense, PCG-G is co-evolving with many other areas in science, and in particular of computer science. We found relevant material for our survey in *IEEE Trans. on Visualization and Computer Graphics*, *ACM Trans. on Graphics*, *ACM Trans. on Multimedia Computing, Communications and Applications*, *IEEE Trans. on Computational Intelligence and AI in Games*, the *J. of Visualization and Computer Animation* (recently renamed *Computer Animation and Virtual Worlds*), the *Int'l. Conf. on Game Design and Technology*, the *IEEE Conf. on Computational Intelligence and Games*, the *ACM Int'l. Conf. on Foundations of Digital Games*, the *Artificial Intelligence and Interactive Digital Entertainment Conference*, the *Int'l. Workshop on Procedural Content Generation in Games*, etc. While there currently exists no textbook on the subject of general procedural game content generation, we point out one book [Ebert et al. 2002] and two surveys [Kelly and McCabe 2006; Smelik et al. 2009]. Ebert et al. introduce a variety of methods for procedural generation of textures and materials, including methods for noise, solids, gases, fire, water, earth, and cloud generation. Kelly and McCabe survey techniques for procedurally generating cities, focusing on the generation of cityscapes, individual buildings, and road networks. For five procedural city generation methods, they analyze the realism, the scale, and the variation of the generated content. Last, they introduce and use several evaluation criteria for user-control and efficiency in procedural city generation. Smelik et al. survey procedural generation of terrain and urban environments. They investigate over 50 papers that address the generation of terrain elevation, bodies of water, road networks, and urban environments. It is symptomatic for the state of the field that there exists no structural overlap in these two surveys, despite the topical overlap and time difference between them. Structuring the entire field of research in procedural game content generation is the main goal of this work.

In this work we survey the methods used for procedurally generating game content for whole spectrum of game content types, with an emphasis on methods that have been used and demonstrated in commercial applications, or that are new trends or ideas. In Section 2 we present several game content types, including game bits, game space, game and systems. We introduce a taxonomy of general PCG-G methods, which, for reasons of space, we summarize in Section 3 and detail in Appendix A. In Section 4 we survey various applications of PCG-G methods, by game content type, and map these applications to our taxonomy of methods. In Section 5, we provide an overview of games that use PCG-G techniques. Finally, we provide recommendations for future research in Section 6, and conclude in Section 7.

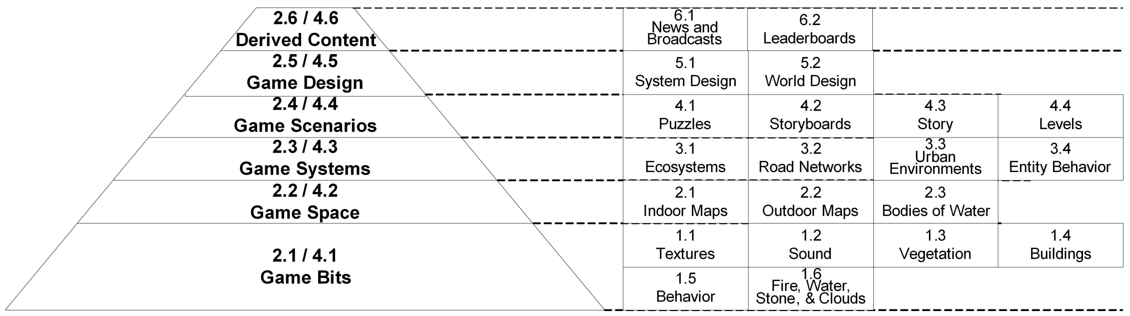

Fig. 1. Types of game content that can be procedurally generated. The numbers in each box refer to the subsection dedicated to the content type in Sections 2 (definition) and 4 (application of procedural generation methods).

2. WHAT'S IN A GAME?

In this section we review six main classes of game content that can be generated procedurally. In addition to classes of content present strictly inside games, we consider derived content, that is, content derived from the content or the state of a game with the goal of immersing the player further into the game world. We structure the six classes as a virtual pyramid (see Figure 1), in which classes of content closer to the top may be built with elements from the classes closer to the bottom.

2.1 Game Bits

Game bits are elementary units of game content, which typically do not engage the user when considered independently. We identify six main types of game bits, which we describe in turn.

2.1.1. *Textures* are images used in games for adding detail to geometry and models, and for giving a visual representation to game elements such as menus. Textures often define the art style of a game. Often, the texture has to additionally represent the material of the object and the way the environment is reflected on, by, and through the material. Stimulated by the degree of skill and amount of work necessary to produce realistic textures—shading and material representation have often formed the advanced part of drawing courses for the past 50 years [Loomis 1951; Edwards 1989; Hillberry 1999]—many games now use artists only for contour generation and difficult to generate materials, and procedural techniques for most general and material-specific textures.

2.1.2. *Sound* has an important function within games [Bartle 2003]; for example, music is used to set game atmosphere and pace, and sound effects are used to give feedback to the player on actions and environment change. Current game production relies mainly on pre-recorded sound clips that are triggered by game events [Manocha et al. 2009, p.160], which would be akin to the use of video clips instead of visual object representations, a decade ago. There is considerable resistance to algorithmic compositions from musicians and the general public [Edwards 2011], but when it comes to procedural generation, the importance of computational thinking about sound can not be ignored. However, pre-recording under financial constraints leads to several limitations, including unrealistic sounds due to unexpected or complex scene configurations.

2.1.3. *Vegetation* is used in many games for a more realistic and thus immersive look. For example, trees can cover the ridge of a mountain and reeds can be found along river banks. The presence of vegetation is not merely aesthetic; it may serve as hiding place or raw material for inventive players, and clarify the climate in which the player operates and thus lead to changes in gameplay.

2.1.4. *Buildings* are essential to represent urban environments in games. Even more so than for vegetation, buildings may be significant to players and often influence gameplay—players often plan their game activities in relation with buildings, for example by ending resource collection activities closer to warehouses or marketplaces. Modern game buildings need to be diverse yet belong to a unitary architectural style; even fictional environments such as World of Warcraft's require interesting and believable buildings to maintain immersion.

2.1.5. *Behavior* is the way in which objects interact with each other and the environment, for example by breaking or exploding when hit. The behavior of objects makes a game more lively and interesting. Procedural behavior is often employed in games to create the illusion of complexity; in response, players may find creative ways to bend or use object behavior.

2.1.6. *Fire, Water, Stone, and Clouds* are often used in games to create a more believable world. In early games, the role of these elements of nature was purely decorative, and only textures and sounds were necessary for this type of content. However, recent advances in computation and modeling [Ebert et al. 2002; Dorsey and Rushmeier 2009] have made more common the use of detailed, realistic, and interactive representations of these elements.

2.2 Game Space

The game space is the environment in which the game takes place, and is partially filled with game bits among which players navigate. The game space plays a major role in creating dramatic experience, as players often construct their interpretation of the game starting from the game space [Nitsche 2009]. In contrast to [Nitsche 2009], we consider space as only the one, two, or three-dimensional area where residing artifacts have relative position and direction.

Our distinction between indoor and outdoor maps stems from aesthetic, cultural, and technical considerations. Aesthetically, outdoor maps of natural and even urban areas have consistent themes, which is less likely for indoor maps. Culturally, indoor maps increasingly feature *many* inter-related, possibly one-of-a-kind artifacts. Technically, outdoor maps require different algorithms and data structures than indoor maps to generate and operate.

2.2.1. *Indoor Maps* are depictions of the structure and relative positioning of indoor space partitioned into rooms. Rooms may be interconnected by corridors, overlapped in layers interconnected by stairs, and grouped altogether in dungeons. Another form of indoor maps are caves, which can have varying and unusual geometry. Yet another form of indoor maps focuses on human-like buildings, where the position and size of rooms, and sometimes their content, are important. The game genre Multi-User Dungeons (MUDs), the ancestor of today's popular MMORPGs, is a type of game in which indoor map navigation is central to gameplay. The platform game Prince of Persia, which is set inside a multi-leveled dungeon, is one of the few to make extensive use of indoor maps.

2.2.2. *Outdoor Maps* are depictions of the elevation and structure of an outdoor terrain. It is common for games with outdoor maps to also have indoor maps; due to important technological differences in the representation and rendering of outdoor and indoor maps, the transition between the two is often made discrete. For example, World of Warcraft has large outdoor areas and numerous indoor areas; players transition between them through the use of special entrance areas and teleportation portals. Many commercially-successful platformer games, such as Nintendo's Super Mario World and Number None Inc.'s Braid, make extensive use of outdoor, albeit stylized, maps; other platform games, such as the Commander Keen series, mix outdoor and indoor maps.

2.2.3. *Bodies of Water* such as rivers, lakes, and seas are often used as map obstacles or even as interactive game space. *Other map features*, such as teleportation areas, etc. and mountains, ridges, ravines, grottoes, etc. may also be part of game space.

2.3 Game Systems

The use of complex systems theory and modeling to generate or simulate parts of a game is not uncommon. The use of game systems can make games more believable and thus appealing. Many systems used in games are similar to textbook complex systems and models [Alexander 1977; Strogatz 1994; Edelstein-Keshet 2005].

2.3.1. *Ecosystems* govern the placement, evolution, and interaction of flora and fauna through algorithms and rules. The designers of Ultima Online, a game which established the game genre of MMORPG [Loguidice and Barton 2009], focused on generating large ecosystems that included complex food chains [Barron 1999, p.21] [Bartle 2003, Ch.4].

2.3.2. *Road Networks* form the basic structure of an outdoors map, serving different purposes such as transportation between points of interest, and structuring of and transportation within cities. The main difficulties in generating road networks are finding the right balance between randomness and structure, and conveying the view of the game designer about places of interest, such as remoteness (through interrupted roads), difficulty (labyrinthine roads), and importance (broader roads).

2.3.3. *Urban Environments* are large clusters of buildings where many people live together and interact with their surroundings. Realistic cities take centuries to grow, and evolve by influence of the people that live there. Procedural algorithms often take a different approach by first generating the road networks, then dividing the terrain between the roads in building lots, and at last generating buildings on these lots. The realism of these generated cities can be improved by applying effects like erosion caused by people and weather.

2.3.4. *Entity Behavior* Many types of complex player-environment interaction need to be possible to make the player experience that a virtual world is life-like. Entities such as non-playable characters (NPC) that interact with the player are a powerful tool to achieve this illusion. Procedurally generating entity behavior based on player action and interaction has the potential to create immersive and realistic experiences. Not only player interaction requires complex entity behavior; Group movement patterns are examples where procedural algorithms could achieve more realistic results.

2.4 Game Scenarios

Game scenarios describe, often transparently to the user, the way and order in which game events unfold.

2.4.1. *Puzzles* are problems to which the player can find a solution based on previous knowledge or by systematically exploring the space of possible solutions embedded in the problem [Colton 2002]. For puzzles, the process of finding the solution *is* the game and thus a rewarding experience. Many quests present in commercial games may be expressed as a sequence or graph of puzzles. Examples of puzzles are riddles, crosswords, and chess endings. The size of the solution space and the previous experience of the player largely determine the difficulty of a puzzle.

2.4.2. *Storyboards* are design aids for the game developer or player. Storyboards are often presented as comics, with sequential panels describing scene events through a visual/textual hybrid. Storyboards can also be used to entertain and guide players, for example through cut-scenes interleaved with normal gameplay. Depending on how they are produced and used, storyboards may also be an example of derived content (see Section 2.6 and 4.6).

2.4.3. The *Story* of a game is often key in creating a good gaming experience. It keeps the player motivated, presents a logical basis for the events that unfold in the game, and provides a goal for the player to accomplish. Besides revelation through cut-scenes and quests, the game story may be embedded entirely in the game universe. Albeit rare in commercial games, perhaps mostly in text-based adventures such as Skoto's Castle Marrach and in artistic games such as Façade, story-intensive quests may not be easily represented as graphs of individual puzzles; for these quests, the dramatic arc plays an important role.

2.4.4. The concept of *Levels* is used in nearly every game as a separator between gameplay sequences. For example, a level in a platform game, such as Nintendo's Super Mario World, Rare's Donkey Kong Country, and Sega's Sony the Hedgehog, would consist of a separate, playable game space in which the player may be required to move from the start to the end position through a series of movements on and jumps to/from platforms, while completing (optional) tasks and avoiding obstacles.

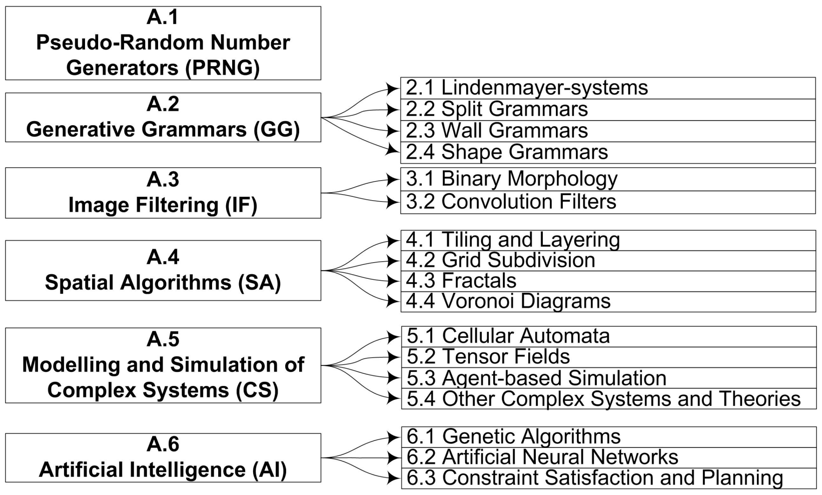

Fig. 2. Taxonomy of common methods for generation game content. The label in each box (such as "A.1" in the "Pseudo-Random Number Generator" box, at the top) refers for each method to its section in Appendix A.

2.5 Game Design

The design of a game is comprised of content such as rules (what can be done in the game?) and goals (what is the player trying to achieve?); an aesthetic component, such as a dramatic arc or a graphical theme, are also important elements in design [Norman 2002]. A game can be seen as an instance of a game design, in which the parameters of the rules and of the goal have been set. A game design can refer to game content of all the types described in Section 2, including the recursive reference to other game design content. Game design can be complemented by (semi-)automatic game generation or by providing tools that help the designer convert ideas into game design content.

2.5.1. The *System Design* of a game entails "the creation of mathematical patterns underlying the game and game rules" [Brathwaite and Schreiber 2008]. An example of generating game rules is the generator by [Pell 1993] which generates symmetric, chess-like games. One of the main challenges in system design generation is ensuring that the rules are balanced between all players.

2.5.2. The *World Design* of a game is "the design of a setting, story, and theme" [Brathwaite and Schreiber 2008]. An example includes the generation of novel games based on beforehand unknown story structures by [Hartsook et al. 2011].

2.6 Derived Content

We define derived content as content that is created as a side-product of the game world. Generation of derived content can greatly increase the feeling of immersion the player has with the game world, as players record their in-game experiences for review inside or outside the game; this "game beyond the game" [Garfield 2000] was the basis of the "metagame" notion (ibid.)

2.6.1. *News and Broadcasts* A game may show its players news items based on their actions and other changes in the game's universe; the same news items could then be presented as television broadcasts and newspaper articles. Similarly, game sessions may be broadcasted—a popular part of television schedules in Asia [Rossignol 2008, Ch.2] and, more recently, US and Europe.

2.6.2. *Leaderboards*—player ranking tables—are popular for a variety of game genres and are used by fan-sites to serve millions of players [Iosup et al. 2010].

3. A TAXONOMY OF METHODS FOR PROCEDURAL CONTENT GENERATION

The usefulness of our survey depends partially on the existence of a set of methods for PCG-G that can be used to generate content across the different content types introduced in Section 2. These methods, which we call *fundamental*, could then be part of a generic content generator, and our survey could help alleviate overlapping efforts in this direction.

We identify five groups of methods for PCG-G, which we discuss in this section and discuss in Appendix A. An overview of the fundamental methods discussed in this work is depicted in Figure 2. The methods are grouped in classes such as Pseudo-Random Number Generators (PRNG), Generative Grammars (GG), Image Filtering (IF), Spatial Algorithms (SA), Modeling and Simulation of Complex Systems (CS), Artificial Intelligence (AI), etc. We show in Section 4 how methods belonging to these groups can be used across different content types.

4. A SURVEY OF PROCEDURAL CONTENT GENERATION FOR GAMES

In this section we survey the state-of-the-art and use of PCG-G techniques. We focus, in turn, on each of the six game content types introduced in Section 2. For each game content type, we begin with the techniques discussed in Appendix A, then look at other PCG-G techniques.

4.1 Game Bits

4.1.1 *Textures.* can be generated procedurally through a variety of techniques, including PRNG, IF, SA, and CS techniques. Perlin noise and other PRNG-based techniques are commonly used for texture generation. Noise-generated textures can be mapped easily on complex objects, unlike raster 2D images. The implementation of noise is relatively simple, and is present in many software shaders and hardware graphics cards, such as NVIDIA's. Pattern-based procedural texturing is an IF-based technique to imitate the level of detail that a large resolution texture can provide [Lefebvre and Neyret 2003]; this technique also requires a procedural algorithm for breaking the regularity of the pattern, without putting limitations on the mesh. Convolution filters such as sharpening or smoothing can improve game textures, even dynamically. SA techniques such as tiling and layering are common in games; their small computational and memory requirements lead to their adoption by many commercial RTS games. To imitate different ornamental styles, geometric decorative patterns can be developed using a range of mathematical algorithms [Whitehead 2010]. From the AI techniques, Genetic Algorithms have been used [Sims 1991] to generate textures procedurally. Chromosomes describe texture-generating functions, such as noise, as expression trees. Crossover occurs by combining branches from the expression trees. The fitness of a gene (texture) is evaluated against a goal set by the user.

4.1.2 *Sound.* Procedural sound can be used for most types of sound [Farnell 2007], for example through PRNG, CS, and GG techniques. A variety of procedural techniques for generating sound, including GG, CS, AI, and mathematics have been surveyed by [Edwards 2011]; some of them date from the 1950s and 60s. CS techniques have been used to generate sound corresponding to different level of detail by simulating sound attenuation [Farnell 2007] or propagation through a liquid [Moss et al. 2010]. Procedural sound can be obtained using GG techniques like a context-free language [Smith, Adam M. 2009] or other rule-based systems [Farnell 2007] specified by a composer or sound modeler.

4.1.3 *Vegetation.* Real vegetation is often self-similar or has a structure. Thus, vegetation can be procedurally modeled [Kelly and McCabe 2006] using GG, SA, and CS techniques. For example, self-similar plants such as ferns and cauliflower can be generated using fractals, and L-systems can be used to model non-self-similar vegetation–symbols may represent plant parts such as the trunk or a leaf, and rules may determine the "growth" of the plant from the initial trunk. An advanced GG-based technique that uses a graph of GGs can be used to generate complex vegetation [Deussen and Lintermann 1999]. A complex system based on plant growth under weather was used [Weber and Penn 1995] to generate realistic trees. Genetic Algorithms have been used [Sims 1991; Reynolds 2010a] to generate vegetation procedurally, similarly to texture generation. SpeedTree is an example of commercial software [Re et al. 2009] to generate vegetation.

4.1.4 *Buildings.* Procedural generation of buildings aims at generating many different buildings from a limited set of rules or user-generated content. GG techniques have been used to generate buildings. For example, the extended L-systems used by CityEngine [Parish and Müller 2001] starts from a symbol representing a bounding box for the building, then iteratively transforms the current building or generates new buildings; the generated buildings belong to the same architectural style. Split grammars [Wonka et al. 2003] and shape grammars [Müller et al. 2006] have been similarly used to generate buildings.

Building floor plans can be procedurally generated using PRNG techniques. A simple process based on floor plans that are extruded upwards can be used [Kelly and McCabe 2007] for real-time building generation. Similar yet more realistic buildings have been generated [Greuter et al. 2003] by randomly changing and merging bi-dimensional polygons, and by extruding the resulting polygons upwards.

4.1.5 *Behavior.* The behavior of many game objects, such as fireworks or patches of wheat, is determined by both the object characteristics and its surroundings. This behavior can be procedurally modeled by, for instance, using GG and CS techniques. From the GG techniques, context-sensitive L-systems, which differ from ordinary L-systems in the ability to represent and manipulate context-dependent variables as symbols, can be used [Hidalgo et al. 2008] to specify complex object behavior such as the explosion of fireworks without a full physical model. Ordinary L-systems and cellular automata can be used to generate plants in a soil patch. The game Galactic Arms Race uses genetic algorithms [Hastings et al. 2009] to adapt the player's weapons to play style, over time.

4.1.6 *Fire, Water, Stone, and Clouds.* The procedural generation of a variety of complex elements, in particular fire, water, stone, and clouds, has received much attention in the past [Ebert et al. 2002; Dorsey and Rushmeier 2009]. For brevity, we focus here on the procedural generation of clouds. Perlin noise and other PRNG techniques, and IF and CS techniques can be used [Roden and Parberry 2005] to generate a realistically looking sky.

4.2 Game Space

4.2.1 *Indoor Maps.* Procedurally-generated dungeons were introduced early in Multi-User Dungeons (MUD) games—the open-source MUD Angband had them in 1992 [Doull 2007a]. Inside the dungeons, mazes were generated using a wide variety of algorithms, mainly based on PRNG and CS techniques [Pullen 2011]. Sangband, an Angband variety, uses PRNG-based techniques to generate mazes and rooms. Gonzalez introduced a fractal-based algorithm for generating caves for NPPAngband, another Angband variant. Large caves can be generated using CS techniques such as cellular automata [Johnson et al. 2010a], as was done in NPPAngband [Babcock 2005; Johnson et al. 2010b]. A much more advanced dungeon generation algorithm, which combines PRNG and GG techniques and also generates other game content types, has been proposed [Adams 2002].

Because of their lax spatial constraints, MUD space generation techniques may not be useful for generating realistic indoor spaces. The combination of PRNG and GG techniques for generating building indoors–through a graph of rooms where the graph is generated using a GG technique, and each room is generated through PRNG techniques–has been proposed [Martin 2006] instead. Other techniques, notably general PRNG combined with advanced parameter space search such as genetic algorithms, have been proposed [Togelius et al. 2010; Frade et al. 2010; Sorenson and Pasquier 2010] for generating indoor space with constraints.

Many outdoor maps rely on a grid-based structure, for example a height map–a matrix where each cell represents the height of the terrain. Many height map generation algorithms based on PRNG, IF, SA, and CS techniques have been described extensively [Ebert et al. 2002; Smelik et al. 2009]. Digital elevation models [Smelik et al. 2009] combine the SA technique of grid subdivision with the use of real data—the terrain is subdivided into regions having each an elevation profile, then digital elevation models of real-world terrain that match the elevation profile of the region are used to create a new, realistic height-map.

4.2.2 *Outdoor Maps.* Different approaches have been suggested to give the designer finer control of the produced environments. Declarative modeling and procedural sketching [Smelik et al. 2010] allow the designer to efficiently create assets while staying within an interactive workflow. Terrain synthesis using procedural brushes allow for local control, up to full automatic terrain generation [de Carpentier and Bidarra 2009]. Software agents are another technique for creating different types of terrain while leaving the designer in control [Doran and Parberry 2010].

4.2.3 *Bodies of Water and Other Map Features.* Map features such as mountains and sea shores are important to players, and have led to the development of specific procedural generation techniques. Among these features, we focus in this survey on bodies of water; although mountains may be another feature of interest, in contrast to water they are usually un-traversable map features.

Procedural generation of bodies of water, such as rivers, lakes, and oceans, is often done during or directly after the generation of the outdoor height map. The shape of the river or sea is either determined by the height map (for example, water starts in mountains and follows the elevation gradient), or the water shapes the height map [Smelik et al. 2009]. The generation of lakes and oceans is usually done after height-map generation by a flooding algorithm, which "floods" the terrain starting from the lowest point, or by simply setting the water level to a specific height–all heights below the water threshold are considered to be under water. SA techniques [Prusinkiewicz and Hammel 1993] and CS techniques that model true water flow and erosion [Clyde 2004] have also been used.

4.3 Game Systems

4.3.1 *Ecosystems.* Using CS techniques, an ecosystem can be modeled [Flake 1999] as a producer and consumer system, in which multiple producers and consumers interact in cooperative and competitive ways. Complex behavior can be modeled to represent the interaction between members of the same species; when multiple species exist, the inter-species interaction obeys much simpler rules. Thus, for multiple species CS and PRNG techniques may be combined for more variation. [Deussen et al. 1998] and [Hammes 2001] have developed ecosystem simulation algorithms to generate vegetation placement maps, which combine an ecosystem with game space information. The algorithm of Deussen et al. is iterative, with plants competing with each other for resources to mimic evolution at each pass of the algorithm; the computational complexity of this algorithm makes it unusable for real-time generation of ecosystems [Smelik et al. 2009]. Hammes uses a single-pass algorithm, which first determines the areas in which plants are to be placed, and then places plants randomly in these designated areas; this algorithm is usable in real-time. An extensive generation process that combines PRNG (Perlin noise), IF (erosion, smoothing), SA (Voronoi diagrams), and CS techniques has been shown [Patel 2010] to generate a complete outdoor map, including an advanced ecosystem.

4.3.2 *Road Networks.* Many methods have been developed to generate road networks procedurally. L-systems (GG), agent simulations (CS), and tensor fields (CS) have been used [Smelik et al. 2009] to generate road networks. For example, L-systems have been used [Parish and Müller 2001] to create road networks with control parameters such as population density, road patterns, elevation constraints, and local constraints such as nearby surface water or (non-)placement near other roads. As an example of agent simulation, two types of agents–one to explore the terrain and create roads, another to connect existing roads to each other into a terrain spanning tree–can be used [Lechner et al. 2003]. A method [Sun et al. 2002] that combines GG and SA techniques is based on Voronoi diagrams as locus for intra-city roads and inter-city highways.

4.3.3 *Urban Environments.* [Kelly and McCabe 2006] give an excellent overview of the different methods that can be used to generate urban environments. Generating these urban environments often starts with a dense road network, where available space near roads are subdivided into building lots [Kelly and McCabe 2007]. Vectorization can be used to find the boundaries of these gridded blocks [Sexton and Watson 2010]. Then, buildings are generated according to pre-defined rules.

Recently, more attention has been paid to the realism of the city, as most techniques fail to generate cities with a realistic structure [Smelik et al. 2009]. [Weber et al. 2009] describes an evolutionary model for city growth, which yields realistic results in a reasonable amount of time. [Groenewegen et al. 2009] generates in seconds, on commodity systems, cities that resemble real cities in Western Europe or the United States.

4.3.4 *Entity Behavior.* Complex artificial intelligence techniques have been used to make game entities react to the wide variety of actions a player can perform in a virtual world with a seeming intelligence. An early example of such an approach is Eliza (1962), a conversational robot that has been analyzed elsewhere [Murray 1997, p.69]; a contemporary, more advanced example is Façade [Mateas and Stern 2005], in which two characters react to what the player does and says. Concerning group behavior, Reynolds [Reynolds 1987] proposes a model for flock behavior using CS techniques.

4.4 Game Scenarios

4.4.1 *Puzzles.* While many algorithms for creating specific types of puzzles are known [Alt et al. 2009], most puzzles are still man-made. An approach to generate puzzle instances procedurally can be based on PRNG techniques, where puzzle instances are randomly generated so that the entire space of puzzle instances is searched [Iosup 2011]. Another approach is to use genetic programming; for example [Ashlock 2010], to generate chess and chromatic mazes. One of the main challenges in the field is to generate interesting puzzles for a large amount of users. While the problem of scaling puzzle generation is shown to be feasible for simple puzzles [Iosup 2009], determining the complexity for most puzzles remains challenging [Colton 2002].

4.4.2 *Storyboards.* Procedural storyboard generation was developed to aid developers in the design process of the game, rather than dynamically generating the story in real time; Story Canvas is an example whose description surveys much related work [Skorupski and Mateas 2010]. The storyboard, in the form of comic strips, would help assess the final gameplay for a given level design [Pizzi et al. 2010]. The player would act as the main agent of the level, with the gameplay actions acting as basic elements in constructing a solution. Overall, every solution should reach the same set of goals (for example, the assassination of a computer-controlled actor), yet achieve these goals through various ways.

4.4.3 *Stories.* Generating a story based on the player's actions could require a complex artificial intelligence engine that can generate a cohesive storyline for various player inputs [Nareyek 2007]. At the same time the engine also needs to abide to guidelines that exist for creating structured storylines. For example, one of these guidelines is that the story tension must peak about every ten minutes to keep the player's attention.

Procedural storytelling may start from the structural approach to interactive storytelling of Vladimir Propp [Propp 1968]. Propp developed a morphological method of classifying (describing) folk tales through the use of story functions. The functions represent character actions, such as "departure" and "guidance"; a limited set of actions turns out to be sufficient to describe the folk tales of many nations [Gervás et al. 2004]. Although Propp's work is limited to folk tales, a number of authors noted that its structural analysis is applicable to many popular modern stories [Fairclough and Cunningham

2003]. Coupled with natural language processing techniques (from the AI field), Propp's morphology has been used to generate (game-like) stories using GG techniques.

[Riedl et al. 2011] uses the artificial intelligence technique of partial-order plans (POP) to represent a story. They extend the standard POP representation by including author goals: intermediate states that the story must contain. Other authors also extended POP, for example [Chen et al. 2010] used formal models of roles to expand the expressiveness of stories. A problem with standard POP solvers is that, although they can find valid solutions, these are not necessarily interesting. Actions can break the structure of the story, in which case the planner of [Riedl et al. 2011] tries to find a new narrative trajectory while taking a player model into account to maximize satisfaction. The planner assumes that humans make the best story lines, and therefore tries to stay as close as possible to a human narrative trajectory. [Riedl and León 2009] also use POP, but takes a different approach in which they try to generate novel stories by transforming existing stories based on analogical reasoning to find points in the stories where parts can be interchanged. As noted by [Li and Riedl 2010], another important problem with POP-based planners is that, currently, they can only be used for offline generation and adaption of storylines.

4.4.4 *Levels.* Level generation is currently one of the most popular types of PCG-G. Although nearly every genre can benefit from generated levels, 2D platformers and puzzles have especially attracted attention. We have already discussed the generation of puzzles in Section 4.4.1.

[Compton and Mateas 2006] and [Smith et al. 2009] generate levels for 2D platformers based on the concept of rhythm: the pattern of hand movements of a player when playing the game. Related, [Shaker et al. 2010] and [Jennings-Teats et al. 2010] demonstrate that personalized levels can be generated online for platform games based on a player model. More general than platforms, [Dormans 2010] uses a grammar to create a mission structure using graphs, which is then translated to a 2D level by using a shape grammar.

4.5 Game Design

4.5.1 *System Design.* Few automatic game generation systems focusing on systems design have been created. The Metagame generator [Pell 1993] can generate symmetric, chess-like games with various piece strength, ability, and positioning. Another generator [Nelson and Mateas 2007] can generate games starting from few words and textures as input. The generated games are very simple, but the generator creates a logically correct game from just a few words, by exploring the thematic space–the generator infers which subjects and type of game match the words input by the user by using common-sense knowledge bases. For example, if the supplied words are "shoot" and "pheasant", a game could be created where the player has to shoot a duck (a game where the player has to avoid being shot by a pheasant is equally plausible to be generated). The EGGG generator [Orwant 2000] takes a different approach by using the rules of a board-game as input. The EGGG generator focuses on abstract game mechanics (rules), which it combines using PRNG techniques. Another generator by [Togelius and Schmidhuber 2008] evolves the game rules of a simple PacMan-like game. The possible rules are limited. Using evolutionary programming the rules are evolved based on a fitness function based on how difficult the game is learn, since it is assumed that an interesting game is easy to learn. The hardness of a game is measured by using controllers which are evolved. Identically, [Hom and Marks 2007] uses genetic algorithms to evolve games, however board games are evolved and the fitness is based on the balance of the games. Another approach that automatically tests the generated games through a constraint satisfaction technique is Variations Forever [Smith and Mateas 2010].

4.5.2 *World Design.* One of the rare game generators of this type is created by [Hartsook et al. 2011], which discusses a system that, based on computer or human generated story and a player preference profile, generates Computer Role-Playing Games (CRPGs). In their approach first the user

Table I. Use of PCG-G techniques in games. We did not find commercial games that generate procedurally Game Designs or Derived Content.

Games (with year of release)	**Game Bits**	**Game space**	**Game Systems**	**Game Scenarios**
Borderlands (2009)	x			
Diablo I (2000)		x		
Diablo II (2008)		x		x
Dwarf Fortress (2006)		x	x	x
Elder Scrolls IV: Oblivion (2007)	x			
Elder Scrolls V: Skyrim (2011)				x
Elite (1984)		x	x	x
EVE Online (2003)	x	x		x
Facade (2005)				x
FreeCiv and Civilization IV (2004)		x		
Fuel (2009)		x		
Gears of War 2 (2008)	x			
Left4Dead (2008)				x
.kkrieger (2004)	x			
Minecraft (2009)		x	x	
Noctis (2002)		x		
RoboBlitz (2006)	x			
Realm of the Mad God (2010)	x			
Rogue (1980)		x		x
Spelunky (2008)	x	x		x
Spore (2008)	x	x		
Torchlight (2009)		x		
X-Com: UFO Defense (1994)		x		

or computer provides a story presented as a partial-order plan. Next, the game plot adaption algorithm modifies the story based on a player's preference model. Following the designer defines a game world model: a model of which transitions between environment maps are realistic. Finally, based on the player's preference profile and the game world model, a genetic algorithm creates a 2D CRPG with the story generated by the game plot adaption algorithm.

4.6 Derived Content

Creating derived content has largely been a task for players that wanted to record their daily gaming experiences. While numerous videos and screenshots are manually produced and shared through sites such as Machinima [Machinima.com 2011], video authoring is difficult and screenshots often lack the context to convey a story.

Chan et al. [Chan et al. 2009] developed a system that creates comics based on key moments in the player's game session. The system picks the most significant frames out of a collected set of screenshots and generates a matching comic layout, accompanied by speech bubbles to indicate sound effects and events. Similarly, an existing [Nelson and Mateas 2007], automated game generator could be used to create news-related games. Cheong et al. [Cheong et al. 2008] describe a system that summarizes game experiences based on game logs, and creates videos based on these.

5. USE OF PROCEDURAL CONTENT GENERATION IN GAMES

In this section we provide an overview of games that use PCG-G techniques. Our selection of games is not exhaustive–too many games use PCG-G techniques. Instead, we focus on games that have been included in the Vintage Games [Loguidice and Barton 2009] (including the web chapters) selection, won industry awards, or have been very recently released and became popular. From these, we select games that cover many game genres, including Strategy (both Real-Time and Turn-Based), simulation (flight), RPG (including roguelike), and FPS.

Elite, a space exploration and trading game, is one of the earliest games to generate a full world–space, systems, and scenarios–procedurally [Loguidice and Barton 2009, Web Chapter]. Inpired by Elite, EVE Online, a popular massively multiplayer online game, generates its planets' visuals [TenTonHammer.com 2009], and its universe and scenarios [Procedural Content Generation Wiki 2009]; EVE Online uses CS techniques to generate its universe and PRNG techniques to place hand-generated scenarios. Other strategy games such as X-Com: UFO Defense, and FreeCiv and Civilization IV (the Civs) generate their game space procedurally. For X-Com, the tactical ground operations take place in randomly-generated, urban-like environments. For the Civs, the terrain and the resources are randomly generated using PRNG and CS techniques, some of which are provided by the two games' respective communities [Civilization Fanatics 2005; FreeCiv Community 2005].

Rogue [Loguidice and Barton 2009, Web Chapter] is an open-source, free-to-change game that spawned popular free-to-use games such as Moria (1983), NetHack (1987), Angband (1990) and Unangband, Thomas Biskup's Ancient Domains of Mystery (ADOM) (1994), and Dwarf Fortress; all these games are "roguelike". Rogue also inspired many popular commercial games, including the Diablo series [Loguidice and Barton 2009, Ch.4], Torchlight, and Realm of the Mad God. Rogue and its descendants generate the indoor game space–dungeons, caves, etc.–procedurally using PRNG, SA, and CS techniques [Doull 2007a; 2007b; Pullen 2011]. Similarly to ADOM, which is "the most complex of the lot" [Loguidice and Barton 2009, Web Chapter], Diablo II extends [IGN Australia 2008] the procedural generation of dungeons with random adventures (scenarios). Dwarf Fortress and Realm of the Mad God generate complete worlds, including the outdoor maps and the ecosystem, through a combination of PRNG, IF, SA, and CS techniques [Patel 2010]. Spelunky [Yu 2008] is a roguelike-platform game that generates "levels, items, monsters, and so forth" (`http://pcg.wikidot.com/pcg-games:spelunky`).

Other games use PCG-G techniques. Noctis and Spore generate planets (visuals and characteristics). Left4Dead generates scenarios by creating enemy encounters dynamically, based on the computer-analyzed stress level of the players. Elder Scrolls IV: Oblivion and Gears of War 2 are just two of the many games [SpeedTree.com 2011] that use the SpeedTree middleware [Re et al. 2009] to generate vegetation. Elder Scrolls V: Skyrim generates scenarios using the Radiant Story system, which fills in quest templates based on game conditions (`http://www.uesp.net/wiki/Skyrim:Radiant`). Façade is an interactive drama game that reacts on text input by the player. RoboBlitz uses procedural techniques to store textures.

6. RECOMMENDATIONS FOR FUTURE RESEARCH IN PCG-G

Prompted by the significant progress in the field of PCG-G and by the uptake of research results by the industry, in this section we make five recommendations for future research.

(1) *Generating content at the top of the content pyramid* (layers "Game Scenarios", "Game Design", and "Derived Content".) Although many concepts and designs for storytelling systems have been created [Göbel et al. 2004], they have yet to find their way into successful commercial games; their main failing may still be the lack of capabilities for producing authentic characters and drama [Fairclough and Cunningham 2003]. Procedural game design is a starting research field, with early results such as the formalization of game design [Nelson and Mateas 2007]. For complete game generation, a generator that focuses on creating news-based games may be academically challenging (Which news are the most appealing to gameplayers? How to generate games from them?) and commercially attractive. Creating derived content has traditionally been a task for the game communities themselves, but authoring and event-selection tools should provide fertile research ground. These automated systems would have to act like movie directors, journalists, or educators. Generating levels for more game genres, in addition to extensions to the early ap-

proaches for generating content for 2D platformers and puzzle games, is a very promising area for the future.

(2) *More detailed generators* (especially for the "Game Space" and "Game Systems" layers) As players are getting more used to realistic games, and as new platforms become increasingly able to use more detailed content, the demand for more detailed content is increasing. Generating high-definition content, both realistic and non-realistic, is already increasing content production costs faster than the increase in game revenues [Takatsuki 2007]. In the future, research should adapt the existing generators to the new insights in the modeling and simulation of a variety of game-related topics: building indoors [Taylor and Parberry 2010], object behavior, river deltas, coastal regions, mountain crevasses, ecosystem formation and evolution, climate, etc. Research should also focus on the interaction between generators, for example water-soil and fauna-urban build-up.

(3) *Generating the "missing" game bits* (the "Game Bits" layer) More types of game bits may become the focus of researchers as content generators for other layers become more realistic. Procedurally generating game bits such as animals, vehicles, and humans [van Welbergen et al. 2010] is complex and requires much future research; the complexity stems from the additional importance the players put on moving game objects. As an example, human motion synthesis, that is, the procedural combination of walking sequences, is an active field of research in 2010 [van Basten et al. 2010].

(4) *The detail-performance trade-off (generating content at scale)* Increasing detail is possible even today, for example by using the modeling and simulation results of decades of Earth science research. However, these models are not easily tractable, and supercomputers and computing grids are necessary today to compute on these models. The study of the detail-performance trade-offs could lead to new game techniques and tools. The possible innovation could also follow two directions in computer systems design: *scale-in*, that is, using multi-core computers (especially graphics cards) for real-time content generation; and *scale-out*, that is, using multi-node computers such as supercomputers and computing grids for massive content generation. Scale-in research may lead to technology that will be embedded in every gamer's environment; scale-out may affect the operation of massively multiplayer online games and of games that operate entirely as a service. The scale-out approach also requires novel techniques for content distribution [Mondet et al. 2009].

(5) *Evaluation of generated content* Research in techniques for content generation should be complemented by research in (semi-)automatic evaluation of generated content. Although several approaches have been proposed, much work remains to be done in characterizing the quantity [Smith and Whitehead 2010], playability [Smith et al. 2010], learnability [Togelius and Schmidhuber 2008], difficulty [Iosup 2011; Smith and Whitehead 2010], freshness [Iosup 2011] and interestingness [Schmidhuber 2002], utility [Li and Riedl 2010], and other elements that may be important for the experience of users. User-centric design [Norman 2002] is an established field in which game-specific techniques are starting to be developed by large companies [Kim et al. 2008; Wixon and Pagulayan 2008], but it is unclear if the proposed techniques can work for the majority of game development studios that cannot afford large-scale, technology-heavy, real-user studies. Increasing the detail of generated content may lead to more parameters needed to control the process of procedural game content generation. In turn, this increases the cost of content production, as more experimentation and expertise are required to produce the desired results. The study the trade-off between detail and content generation controllability [Smelik et al. 2010] (and expressiveness [Smith and Whitehead 2010]) could lead to the development of new procedural techniques.

7. CONCLUSION

The research field of procedural game content generation is evolving rapidly, driven by the increasing demand from game development companies. In this work we have presented a comprehensive survey of Procedural Game Content Generation (PCG-G). First, we have created a taxonomy of game content with six layers: game bits, game space, game systems, game scenarios, game design, and derived content. These layers have been ordered by complexity, with each layer using the techniques and methods from the layer before it. Second, we have created a taxonomy of PCG-G techniques that can be used to generate content from different layers in our content taxonomy. Third, we have surveyed the state-of-the-art in PCG-G techniques and the use of these techniques in (real) games. We have found an inverse relationship between the position of the layer in the content taxonomy and the maturity of the layer's procedural generation techniques. We have also found that many real games use PCG-G techniques, but that usage if often limited to a particular type of game content. Last, we have made five recommendations for future research on PCG-G: generating content from the high-complexity layers, developing more realistic generators, generating the missing game bits, investigating the realism-performance trade-off, and investigating the realism-control trade-off.

We intend to continue our work on the survey of PCG-G by focusing more on commercial games. We would like to conduct a survey or interviews on this topic with commercial game studios.

Acknowledgments

We thank our reviewers for their useful comments. Thanks to them, we have discovered a new home! The authors also thank Rafael Bidarra, Tim Tutenel, and Ruben Smelik, for their help in improving this paper. This work was supported by the STW/NWO Veni grant @larGe (11881).

REFERENCES

ADAMS, D. 2002. Automatic generation of dungeons for computer games. B.Sc. Thesis, University of Sheffield, UK. `www.dcs.shef.ac.uk/intranet/teaching/projects/archive/ug2002/pdf/u9da.pdf`.

ALEXANDER, C. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, NY, USA.

ALT, H., BODLAENDER, H. L., VAN KREVELD, M. J., ROTE, G., AND TEL, G. 2009. Wooden geometric puzzles: Design and hardness proofs. *Theory Comput. Syst. 44,* 2, 160–174.

ASHLOCK, D. 2010. Automatic generation of game elements via evolution. In *IEEE Symposium on Computational Intelligence and Games (CIG), 2010*. 289 –296.

AURENHAMMER, F. 1991. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv. 23*, 345–405.

BABCOCK, J. 2005. Cellular automata method for generating random cave-like levels. `roguebasin.roguelikedevelopment.org/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels`.

BARRON, T. 1999. *Multiplayer Game Programming*. Prima Publishing.

BARTLE, R. 2003. *Designing Virtual Worlds*. New Riders Games.

BRATHWAITE, B. AND SCHREIBER, I. 2008. *Challenges for game designers*. Charles River Media, Inc. Rockland, MA, USA.

CHAN, C., THAWONMAS, R., AND CHEN, K. 2009. Automatic storytelling in comics: a case study on World of Warcraft. In *International Conference on Human factors in computing systems*. ACM, 3589–3594.

CHEN, G., ESCH, G., WONKA, P., MÜLLER, P., AND ZHANG, E. 2008. Interactive procedural street modeling. In *SIGGRAPH Annual Conference on Computer graphics and Interactive Techniques*. ACM, 103.

CHEN, S., SMITH, A. M., JHALA, A., WARDRIP-FRUIN, N., AND MATEAS, M. 2010. Rolemodel: towards a formal model of dramatic roles for story generation. In *Intelligent Narrative Technologies III Workshop*. ACM, 17:1–17:8.

CHEONG, Y., JHALA, A., BAE, B., AND YOUNG, R. 2008. Automatically generating summary visualizations from game logs. In *International Conference on Artificial Intelligence and Interactive Digital Entertainment*.

CHOPARD, B. AND DROZ, M. 1998. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press.

CIVILIZATION FANATICS. 2005. [map script] full of resources. Community discussion. `forums.civfanatics.com/showthread.php?s=7ef168705a794e2c328217a5de2e8589&t=151629`.

CLYDE, D. 2004. Adding realistic rivers to random terrain. GameDev technical article. Accessed via `archive.org` on 24 January 2011. `http://www.dcs.shef.ac.uk/intranet/teaching/projects/archive/ug2002/pdf/u9da.pdf`.

COLTON, S. 2002. Automated puzzle generation. In *AISB'02 Symposium on AI and Creativity in the Arts and Science*.

COMPLEXITYGAMING.COM. 2008. Incredible wow stats. ComplexityGaming.com community post. `www.complexitygaming.com/forums/showthread.php?p=25334`.

COMPTON, K. AND MATEAS, M. 2006. Procedural level design for platform games. In *International Conference on Artificial Intelligence and Interactive Digital Entertainment*.

DAVIDSSON, P. 2001. Multi agent based simulation: Beyond social simulation. In *Multi-Agent-Based Simulation*, S. Moss and P. Davidsson, Eds. LNCS Series, vol. 1979. Springer Berlin / Heidelberg, 141–155.

DE BERG, M., CHEONG, O., VAN KREVELD, M., AND OVERMARS, M. 2008. *Computational geometry: algorithms and applications*. Springer. 3rd Ed.

DE CARPENTIER, G. J. P. AND BIDARRA, R. 2009. Interactive gpu-based procedural heightfield brushes. In *International Conference on Foundations of Digital Games*. FDG '09. ACM, New York, NY, USA, 55–62.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MĚCH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH Annual Conference on Computer graphics and Interactive Techniques*. ACM, 275–286.

DEUSSEN, O. AND LINTERMANN, B. 1999. Interactive modeling of plants. *IEEE Comp. Graphics and Applications 19*, 56–65.

DORAN, J. AND PARBERRY, I. 2010. Controlled procedural terrain generation using software agents. *IEEE Transactions on Computational Intelligence and AI in Games 2,* 2, 111–119.

DORMANS, J. 2010. Adventures in level design: generating missions and spaces for action adventure games. In *Workshop on Procedural Content Generation in Games*. ACM, 1–8.

DORSEY, J. AND RUSHMEIER, H. 2009. Advanced material appearance modeling. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Courses*. ACM, New York, NY, USA, 1–134.

DOULL, A. 2007a. Unangband dungeon generation, parts 1–9. roguelikedeveloper.blogspot.com/2007/11/unangband-dungeon-generation-part-one.html.

DOULL, A. 2007b. Wilderness generation using voronoi diagrams - part i. roguelikedeveloper.blogspot.com/2007/07/wilderness-generation-using-voronoi.html.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach* 3rd Ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

EDELSTEIN-KESHET, L. 2005. *Mathematical Models in Biology*. Classics in Applied Mathematics Series, vol. 46. SIAM.

EDWARDS, B. 1989. *Drawing on the Right Side of the Brain*. The Putnam Publishing Group, New York, NY, USA.

EDWARDS, M. 2011. Algorithmic composition: computational thinking in music. *Commun. ACM 54*, 58–67.

ELAS, T. 2010. The cost of creating and maintaining an MMORPG. Tham's Blog. www.thamelas.com/2010/02/12/the-cost-of-creating-and-maintaining-an-mmorpg/ Last retrieved via archive.org, Nov 2011.

ESA. 2010. Essential facts about the computer and video game industry: Sales, demographics, and usage data. Annual Report, series 2003–2009. [Online] Available: http://www.theesa.com.

FAIRCLOUGH, C. AND CUNNINGHAM, P. 2003. A multiplayer case based story engine. In *4th International Conference on Intelligent Games and Simulation (GAME-ON)*. 41–46.

FARBRAUSCH PROD. 2006. Website of .kkrieger producer, .theprodukkt. *http://www.theprodukkt.com/*.

FARNELL, A. 2007. An introduction to procedural audio and its application in computer games.

FIELDS, T. 2010. *Distributed Game Development: Harnessing Global Talent to Create Winning Games*. Focal Press.

FLAKE, G. 1999. *The Computational Beauty of Nature*. MIT press.

FRADE, M., DE VEGA, F. F., AND COTTA, C. 2010. Evolution of artificial terrains for video games based on accessibility. In *Applications of Evolutionary Computation, EvoApplicatons 2010*. LNCS Series, vol. 6024. Springer, 90–99.

FREECIV COMMUNITY. 2005. Map comparison. Community discussion. www.samiam.org/freeciv/.

GARFIELD, R. 2000. Metagames. Horsemen of the Apocalypse: Essays on Roleplaying. Reproduced as Lost in the Shuffle: Games Within Games, http://www.wizards.com/Magic/magazine/Article.aspx?x=mtg/daily/feature/96.

GERVÁS, P., DÍAZ-AGUDO, B., PEINADO, F., AND HERVÁS, R. 2004. Story plot generation based on CBR. *Applications and Innovations in Intelligent Systems XII 12*, 33–46.

GÖBEL, S., SPIERLING, U., HOFFMANN, A., IURGEL, I., SCHNEIDER, O., DECHAU, J., AND FEIX, A., Eds. 2004. *Technologies for Interactive Digital Storytelling and Entertainment, Second International Conference*.

GOLDBERG, D. 1989. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional.

GREUTER, S., PARKER, J., STEWART, N., AND LEACH, G. 2003. Real-time procedural generation of 'pseudo infinite' cities. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. 87–95.

GROENEWEGEN, S., SMELIK, R., DE KRAKER, K., AND BIDARRA, R. 2009. Procedural City Layout Generation Based on Urban Land Use Models. In *Proceedings of the 30th Annual Conference of the European Association for Computer Graphics (Eurographics' 09)*. 45–48.

HAMMES, J. 2001. Modeling of ecosystems as a data source for real-time terrain rendering. In *Digital earth moving: first international symposium, DEM 2001, Manno, Switzerland, September 5-7, 2001: proceedings*. Vol. 2181. Springer Verlag, 98.

HARTSOOK, K., ZOOK, A., DAS, S., AND RIEDL, M. 2011. Toward supporting stories with procedurally generated game worlds. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE, 297–304.

HASTINGS, E., GUHA, R., AND STANLEY, K. 2009. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games 1,* 4, 245–263.

HAYKIN, S. 1994. *Neural networks: a comprehensive foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA.

HIDALGO, J., CAMAHORT, E., ABAD, F., AND VICENT, M. 2008. Procedural Graphics Model and Behavior Generation. In *ICCS '08*. LNCS Series, vol. 5102. Springer, 106–115.

HILLBERRY, J. D. 1999. *Drawing Realistic Textures in Pencil*. North Light Books, Cincinnati, OH, USA.

HOM, V. AND MARKS, J. 2007. Automatic design of balanced board games. In *International Conference on Artificial Intelligence and Interactive Digital Entertainment*. 25–30.

IGN AUSTRALIA. 2008. The ten commandments of diablo iii. Technical Article. `uk.pc.ign.com/articles/888/888189p1.html`.

IOSUP, A. 2009. Poggi: Puzzle-based online games on grid infrastructures. In *Euro-Par*. LNCS Series, vol. 5704. Springer, 390–403.

IOSUP, A. 2011. POGGI: generating puzzle instances for online games on grid infrastructures. *Concurrency and Computation: Practice and Experience 23,* 2, 158–171.

IOSUP, A., LASCATEU, A., AND TAPUS, N. 2010. CAMEO: Enabling social networks for massively multiplayer online games through continuous analytics and cloud computing. In *ACM/IEEE Symposium on Network and Systems Support for Games (NetGames 2010)*. 1–6.

IRISH, D. 2005. *The Game Producer's Handbook*. Course Technology PTR.

JENNINGS-TEATS, M., SMITH, G., AND WARDRIP-FRUIN, N. 2010. Polymorph: dynamic difficulty adjustment through level generation. In *Workshop on Procedural Content Generation in Games*. PCGames '10. ACM, New York, NY, USA, 11:1–11:4.

JOHNSON, L., YANNAKAKIS, G. N., AND TOGELIUS, J. 2010a. Cellular automata for real-time generation of infinite cave levels. In *Workshop on Procedural Content Generation in Games*. PCGames '10. ACM, New York, NY, USA, 10:1–10:4.

JOHNSON, L., YANNAKAKIS, G. N., AND TOGELIUS, J. 2010b. Cellular automata for real-timee generation of infinite cave levels. In *PCGames*. ACM, New York, NY, USA, 1–8.

JOHNSON, S. 2006. The long zoom. The New York Times. `www.nytimes.com/2006/10/08/magazine/08games.html`.

KELLY, G. AND MCCABE, H. 2006. A survey of procedural techniques for city generation. *ITB Journal 14*, 87–130.

KELLY, G. AND MCCABE, H. 2007. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology*. 8–16.

KIM, J. H., GUNN, D. V., SCHUH, E., PHILLIPS, B., PAGULAYAN, R. J., AND WIXON, D. R. 2008. Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. In *CHI*.

KRUEGER, B. D., BRAND, O., AND BURTON, D. 2005. Reinventing your company without reinventing the wheel. Game Developers Conference.

KUSHNER, D. 2003. *Masters of Doom: How two guys created an empire and transformed pop culture*. Random House, New York.

LARIVE, M. AND GAILDRAT, V. 2006. Wall grammar for building generation. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*. ACM, 437.

LECHNER, T., WATSON, B., AND WILENSKY, U. 2003. Procedural city modeling. In *In 1st Midwestern Graphics Conference*.

LECKY-THOMPSON, G. W. 2001. *Infinite game universe: Mathematical techniques*. Advances in Computer Graphics and Game Development. Charles River Media.

LEFEBVRE, S. AND NEYRET, F. 2003. Pattern based procedural textures. In *I3D 2003 Conf. Proc*. 203–212.

LI, B. AND RIEDL, M. 2010. An offline planning approach to game plotline adaptation. In *Proc. of the 6th AI and Interactive Digital Entertainment Conference*.

LOGUIDICE, B. AND BARTON, M. 2009. *Vintage Games: An Insider Look at the History of Grand Theft Auto, Super Mario, and the Most Influential Games of All Time*. Elsevier Focal Press.

LOOMIS, A. 1951. *Successful Drawing*. The Viking Press, New York, NY, USA.

MACHINIMA.COM. 2011. Machinima.

MANOCHA, D., CALAMIA, P., LIN, M. C., MANOCHA, D., SAVIOJA, L., AND TSINGOS, N. 2009. Interactive sound rendering. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Courses*. ACM, New York, NY, USA, 1–338.

MARTIN, J. 2006. Procedural house generation: A method for dynamically generating floor plans. In *Symposium on Interactive 3D Graphics and Games*. 1–2.

MATEAS, M. AND STERN, A. 2005. Procedural authorship: A case-study of the interactive drama facade. In *Digital Arts and Culture: Digital Experience: Design, Aesthetics, Practice (DAC 2005)*. 1–8.

MONDET, S., CHENG, W., MORIN, G., GRIGORAS, R., BOUDON, F., AND OOI, W. T. 2009. Compact and progressive plant models for streaming in networked virtual environments. *TOMCCAP 5*, 21:1–21:22.

MOSS, W., YEH, H., HONG, J.-M., LIN, M. C., AND MANOCHA, D. 2010. Sounding liquids: Automatic sound synthesis from fluid simulation. *ACM Trans. on Graphics 29*, 21:1–21:13.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*. ACM, 623.

MURRAY, J. H. 1997. *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. The Free Press, New York, NY, USA.

NAREYEK, A. 2007. Game AI Is Dead. Long Live Game AI! *IEEE intelligent Systems 22*, 9–11.

NELSON, M. J. AND MATEAS, M. 2007. Towards automated game design. In *Congress of the Italian Association for Artificial Intelligence on AI*IA 2007*. Springer-Verlag, Berlin, Heidelberg, 626–637.

NITSCHE, M. 2009. *Video Game Spaces: Image, Play, and Structure in 3D Worlds*. MIT Press.

NORMAN, D. A. 2002. *The Design of Everyday Things*. Basic Books.

ORWANT, J. 2000. Eggg: automated programming for game generation. *IBM Syst. J. 39*, 782–794.

PARISH, Y. AND MÜLLER, P. 2001. Procedural modeling of cities. In *SIGGRAPH Annual Conference on Computer graphics and Interactive Techniques*. ACM, 301–308.

PATEL, A. 2010. Polygonal map generation. Blog. `www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/`.

PELL, B. 1993. METAGAME in symmetric chess-like games. Tech. Rep. UCAM-CL-TR-277, University of Cambridge, Computer Laboratory.

PERLIN, K. 1985. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 287–296.

PERLIN, K. 1990. Making noise. `http://www.noisemachine.com/talk1/`.

PI, X., SONG, J., ZENG, L., AND LI, S. 2006. Procedural terrain detail based on patch-lod algorithm. In *Edutainment*, Z. Pan, R. Aylett, H. Diener, X. Jin, S. Göbel, and L. Li, Eds. LNCS. Springer, 913–920.

PIZZI, D., LUGRIN, J., WHITTAKER, A., AND CAVAZZA, M. 2010. Automatic generation of game level solutions as storyboards. *IEEE Transactions on Computational Intelligence and AI in Games 2,* 3, 149–161.

PROCEDURAL CONTENT GENERATION WIKI. 2009. Eve online. Technical Article. `http://pcg.wikidot.com/pcg-games:eve-online`.

PROPP, V. 1968. *Morphology of the Folktale*. University of Texas Press.

PRUSINKIEWICZ, P. AND HAMMEL, M. 1993. Fractal model of mountains with rivers. In *Proceeding of Graphics Interface (Toronto, Ontario, May 19-21, 1993)*. 174–180.

PULLEN, W. 2011. Think labyrinth: Maze classification, creation algorithms, and solving algorithms page. `www.astrolog.org/labyrnth/algrithm.htm`.

RE, A., ABAD, F., CAMAHORT, E., AND JUAN, M. C. 2009. Tools for procedural generation of plants in virtual scenes. In *ICCS 2009: International Conference on Computational Science*. Springer-Verlag, Berlin, Heidelberg, 801–810.

REYNOLDS, C. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*. Vol. 21. ACM, 25–34.

REYNOLDS, C. 2010a. Using interactive evolution to discover camouflage patterns. In *SIGGRAPH Posters*. ACM.

REYNOLDS, D. 2010b. The cost to make a quality MMORPG. Self-published. `www.whatmmorpg.com/cost-to-make-a-quality-mmorpg.php`.

RIEDL, M. AND LEÓN, C. 2009. Generating story analogues. In *International Conference on Artificial Intelligence and Interactive Digital Entertainment*.

RIEDL, M., THUE, D., AND BULITKO, V. 2011. *Game AI as storytelling*. Vol. 125. Springer Verlag.

RODEN, T. AND PARBERRY, I. 2005. Clouds and stars: efficient real-time procedural sky rendering using 3d hardware. In *ACE '05: ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACM, 434–437.

ROSSIGNOL, J. 2008. *This Gaming Life: Travels in Three Cities*. U. Michigan Press.

RUSSELL, S., NORVIG, P., CANNY, J., MALIK, J., AND EDWARDS, D. 1995. *Artificial intelligence: a modern approach*. Vol. 74. Prentice hall Englewood Cliffs, NJ.

SCHMIDHUBER, J. 2002. Exploring the predictable. In *Advances in Evolutionary Computing*, A. Ghosh and S. Tsuitsui, Eds. Springer, 579–612.

SEXTON, C. AND WATSON, B. 2010. Vectorization of gridded urban land use data. In *Workshop on Procedural Content Generation in Games*. PCGames '10. ACM, New York, NY, USA, 5:1–5:8.

SHAKER, N., YANNAKAKIS, G., AND TOGELIUS, J. 2010. Towards automatic personalized content generation for platform games. In *International Conference on Artificial Intelligence and Interactive Digital Entertainment*. 63–68.

SIMS, K. 1991. Artificial evolution for computer graphics. In *SIGGRAPH*. ACM, 319–328.

SKORUPSKI, J. AND MATEAS, M. 2010. Novice-friendly authoring of plan-based interactive storyboards. In *International Conference on Artificial Intelligence and Interactive Digital Entertainment*. (poster).

SMELIK, R., DE KRAKER, K., TUTENEL, T., BIDARRA, R., AND GROENEWEGEN, S. 2009. A survey of procedural methods for terrain modelling. In *CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*. 25–34.

SMELIK, R., TUTENEL, T., DE KRAKER, K. J., AND BIDARRA, R. 2010. Integrating procedural generation and manual editing of virtual worlds. In *Workshop on Procedural Content Generation in Games*. PCGames '10. ACM, New York, NY, USA, 2:1–2:8.

SMITH, A. M. AND MATEAS, M. 2010. Variations Forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *IEEE Symposium on Computational Intelligence and Games (CIG)*. IEEE, 111–118.

SMITH, G., TREANOR, M., WHITEHEAD, J., AND MATEAS, M. 2009. Rhythm-based level generation for 2d platformers. In *International Conference on Foundations of Digital Games*. ACM, 175–182.

SMITH, G. AND WHITEHEAD, J. 2010. Analyzing the expressive range of a level generator. In *Workshop on Procedural Content Generation in Games*. PCGames '10. ACM, New York, NY, USA, 4:1–4:7.

SMITH, G., WHITEHEAD, J., AND MATEAS, M. 2010. Tanagra: a mixed-initiative level design tool. In *International Conference on the Foundations of Digital Games*. FDG '10. ACM, New York, NY, USA, 209–216.

SMITH, ADAM M. 2009. cfml: the context-free music language.

SORENSON, N. AND PASQUIER, P. 2010. Towards a generic framework for automated video game level creation. In *Applications of Evolutionary Comp., EvoApplicatons 2010*. LNCS Series, vol. 6024. Springer, 131–140.

SPEEDTREE.COM. 2011. Speedtree list of games.

STROGATZ, S. H. 1994. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books Publishing, LLC.

SUN, J., YU, X., BACIU, G., AND GREEN, M. 2002. Template-based generation of road networks for virtual city modeling. In *ACM Symposium on Virtual Reality Software and Technology*. VRST '02. ACM, New York, NY, USA, 33–40.

TAKATSUKI, Y. 2007. Cost headache for game developers. `news.bbc.co.uk/2/hi/business/7151961.stm`.

TAYLOR, J. AND PARBERRY, I. 2010. Computerized clutter: How to make a virtual room look lived-in. Tech. Rep. Technical Report LARC-2010-01, University of North Texas. April.

TENTONHAMMER.COM. 2009. Incarna Incarnate: An EVE Online Q&A with Torfi Frans Olafsson. `www.tentonhammer.com/node/75367`.

TOGELIUS, J. AND SCHMIDHUBER, J. 2008. An experiment in automatic game design. In *IEEE Symposium on Computational Intelligence and Games (CIG)*. IEEE, 111–118.

TOGELIUS, J., YANNAKAKIS, G. N., STANLEY, K. O., AND BROWNE, C. 2010. Search-based procedural content generation. In *Applications of Evolutionary Computation, EvoApplicatons 2010*. LNCS Series, vol. 6024. Springer, 141–150.

VAN BASTEN, B. J. H., PEETERS, P. W. A. M., AND EGGES, A. 2010. The step space: example-based footprint-driven motion synthesis. *Journal of Visualization and Computer Animation 21,* 3-4, 433–441.

VAN VERTH, J. M. AND BISHOP, L. M. 2008. *Essential mathematics for games and interactive applications: a programmer's guide*. Elsevier Morgan Kaufmann Publishers. 2nd Ed.

VAN WELBERGEN, H., VAN BASTEN, B. J. H., EGGES, A., RUTTKAY, Z., AND OVERMARS, M. H. 2010. Real time animation of virtual humans: A trade-off between naturalness and control. *Comput. Graph. Forum 29,* 8, 2530–2554.

VIDEO GAME SALES WIKI. 2009. Video game costs. `vgsales.wikia.com/wiki/Video_game_costs`.

WEBER, B., MÜLLER, P., WONKA, P., AND GROSS, M. 2009. Interactive Geometric Simulation of 4D Cities. In *Computer Graphics Forum*. Vol. 28. Blackwell Publishing, 481–492.

WEBER, J. AND PENN, J. 1995. Creation and rendering of realistic trees. In *SIGGRAPH Annual Conference on Computer graphics and Interactive Techniques*. ACM, 119–128.

WHITEHEAD, J. 2010. Toward proccedural decorative ornamentation in games. In *Workshop on Procedural Content Generation in Games*. PCGames '10. ACM, New York, NY, USA, 9:1–9:4.

WIXON, D. R. AND PAGULAYAN, R. J. 2008. That's entertainmnt - Halo 3: the theory and practice of a research-design partnership. *Interactions 15,* 1, 52–55.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. on Graphics 22*, 669–677.

YU, D. 2008. Spelunky. Game and Source code. `http://www.derekyu.com/games/` Accessed 14 July 2011.

A. MORE DETAILS ON THE TAXONOMY OF METHODS FOR PROCEDURAL CONTENT GENERATION

In this section we discuss the classes of methods introduced in Section 3.

A.1 Pseudo-Random Number Generators (PRNG)

Nature often gives the illusion of randomness, for example in the shape of a mountain, a cloud, or a flower. Pseudo-random number generators (PRNGs) can, therefore, be used for mimicking the randomness found in nature.

Perlin noise [Perlin 1985; 1990] is a PRNG-based noise generator with wide use in media and entertainment. This noise generates maps of data points (random values). The map data points are generated by a seeded PRNG through interpolation. More detail can be added to the noise map by combining multiple layers of Perlin noise and by using scaling. The Perlin noise is representative for the many PRNG-based techniques used in games [Van Verth and Bishop 2008]; other popular PRNGs exist [Lecky-Thompson 2001].

A.2 Generative Grammars (GG)

Generative grammars, stemming from Noam Chomsky's study of languages in the 1960s, are sets of rules that, operating on individual words, can generate only grammatically-correct sentences. They can be used to create correct objects from elements encoded as letters/words. In this section we discuss L-systems, split grammars, wall grammars, and shape grammars, which have been used for content generation in entertainment.

A.2.1 *Lindenmayer-systems.* (L-systems) consist of a grammar consisting of symbols which describe the characteristics of an object. A string generated by the grammar describes the structure or the behavior of an object.

A.2.2 *Split Grammars.* Similarly to L-systems, split grammars [Wonka et al. 2003] work on string-encoded shapes. New shapes are generated from a basic set of shapes by applying rewriting rules governing shape-to-shape conversion, the split grammar generates a new shape. For example, using a split grammar an initial wall-shape can be divided into two smaller shapes, which in turn can be rewritten (converted) into a window frame shape and a window shape. Split grammars are context-free, meaning that the rewriting process always yields the same result given a set of rewriting rules, no matter in which order the string of symbols is evaluated.

A.2.3 *Wall Grammars.* [Larive and Gaildrat 2006] are specifically designed for creating building exteriors. Shapes are manipulated to form a building exterior similarly to split grammars, but wall grammars can generate more advanced shapes–for example, the wall extrusion rule can lead to complex three-dimensional shapes like balconies and fire escapes.

A.2.4 *Shape Grammars.* [Müller et al. 2006] are context-sensitive and sequential grammars originating from the work of Stiny in the 1970s. For each rewriting step, the symbol and its neighbors in the string determine what symbol(s) replace the original symbol. Therefore, the rewriting process of shape grammars is different from L-systems or split grammars. Similarly to wall grammars, shape grammars can generate more complex structures.

A.3 Image Filtering (IF)

Image filtering has as main goal to improve an image with regard to a (subjective) measure, or to emphasize certain characteristics of an image to display (partially) hidden information. Many techniques have been developed for image filtering; the yearly image processing tutorials at SIGGRAPH are good surveys of the state-of-the-art with application in multimedia-related fields. In this section we present two fundamental image processing techniques, binary morphology and convolution filters.

A.3.1 *Binary Morphology.* is a set of techniques used for binary operations on images. The binary image often required by binary operations can be obtained through thresholding, a process where pixels below a certain intensity are set to zero and the rest to one, effectively creating a binary image. Typical examples of binary morphology operations include dilation, in which pixels are added to the edge of an element in an image, and erosion, which does the opposite. By combining basic binary operators, more useful complex operations can be achieved. For example, by first dilating an image and then subtracting the original from the result, the final result depicts the edges of each element in the original image.

A.3.2 *Convolution Filters.* are filters which can be represented by a simple image, function, or discrete dataset). Convolution is a mathematical operator on two signals, where one signal is used to modify the other thereby creating a new signal. These type of filters can be used for example to remove noise, smooth, sharpen, detect edges of objects, or even detect the movement direction of objects in an image. Using convolution filters, a simple texture can be manipulated to create a whole new texture, thereby saving storage space.

A.4 Spatial Algorithms (SA)

Spatial algorithms manipulate space to generate game content. The output is created by using an input with structure, for example a grid, or self-recurrence. In this section we discuss tiling and layering, grid subdivision, fractals, and Voronoi diagrams.

A.4.1 *Tiling and Layering.* Tiling is a technique used to create a game space by decomposing a map into a grid. The grid is not limited to a rectangle-size–hexagonal shapes are also common. Grids are 2D-data structures, but isometric projections can be coupled to grids to create the illusion of a 3D map.

Layering is a technique that integrates several grids, called layers, into the same map. A tile is then constructed by overlapping parts from each layer, some of which may contain transparent parts. This approach enables the creation of overlay effects, such as running water, and of 3D-looking game space by using only a limited amount of source terrain textures.

A.4.2 *Grid Subdivision.* is an iterative and dynamic technique for object generation. An object is first divided into a uniform grid with the appropriate textures. A grid subdivision algorithm, for example the Patch-LOD algorithm [Pi et al. 2006], is used to iteratively add detail to the object. The dynamic part of this technique links the iterative generation to the point of view–only the grid cells that are closer to the current point of detail must be subdivided into smaller cells to create the required level of detail. An example using this technique is the rendering of a procedurally generated terrain: only the cells near the player are detailed (generated), while the rest of the terrain is more coarse, thereby saving computational resources.

A.4.3 *Fractals.* are recursive figures which consist of copies of themselves, for example a Koch snowflake. With fractals, a few parameters control a wide range of possible results. An advantage of fractals is that objects with seemingly infinite detail can be stored as a simple recursive function. The generation of fractals is a resource-intensive process, due to recursiveness. Using Iterated Function Systems, the resulting image can be generated faster, using an iterative rather than a recursive process.

A.4.4 *Voronoi Diagrams.* [Aurenhammer 1991] are decompositions of metric spaces into parts whose size and shape is determined by the position of seed points (points of interest) in the metric space. The decomposition establishes borders of points equally distant from the closest seed points, and territory containing exactly one seed point and all the locations for which the territory's seed point is the nearest point of interest. For a small map with a few points, a diagram can easily be calculated using a iterative approach. However, when the collections of points increases in size, an improved computational approach is required. A variety of algorithms have already been proposed to make Voronoi diagrams usable in near-real-time [de Berg et al. 2008, Ch.7].

A.5 Modeling and Simulation of Complex Systems (CS)

It is impractical in some cases to describe natural phenomena with mathematical equations. Models and simulations can be used to overcome this problem. In this section we describe cellular automata, tensor fields, and agent-based simulation.

A.5.1 *Cellular Automata.* [Chopard and Droz 1998] A cellular automaton is a discrete computational model based on cells aligned in a grid, where each cell has a state and is subject to a common set of rules. The computational model is applied at discrete time steps. The rules of the board determine how cell neighborhood and state influence the next state of the cell. The resulting behavior can be random, but also periodic.

Cellular automata can be combined with other systems to form new computational models. An example of a combined system is a open L-system, a combination of cellular automata and L-systems. In open L-systems, the behavior of the objects is largely determined by interaction with the environment the object is in and by interaction with other objects in that environment [Hidalgo et al. 2008]. This allows for the modeling of spatial constraints, such as the minimum distance between two objects.

A.5.2 *Tensor Fields.* [Chen et al. 2008, Sec.4] are two-dimensional generalizations of vectors, which can be used to specify the shape of a game space. The tensors describe the direction of the elevation of the map. Because tensor lines can be visualized, they are suitable for interactive design and manipulation of road networks.

A.5.3 *Agent-based Simulation.* (ABS) [Davidsson 2001] is based on modeling a complex situation using individuals, called agents. Emergent behavior – complex behavior that arises out of relatively simple agent interactions – is a feature of ABS that contrasts with the average behavior observable through traditional modeling techniques. Agents can be added, removed, or replaced during the simulation; agents may also learn in time.

A.5.4 *Other Complex Systems and Theories.* Many other complex systems and theories have and may still make their way in procedural content generation for games, including theories of the dramatic act (for story generation), of the cognitive process (for entity behavior), etc.

A.6 Artificial Intelligence (AI)

Artificial Intelligence is a large field in computer science that tries to mimic animal or human intelligence. Examples include speech recognition, planning, and execution of physical tasks by robots.

A.6.1 *Genetic Algorithms.* [Goldberg 1989] are used to solve optimization problems by mimicking biological evolution. Possible solutions are coded as a strings (chromosomes), and a fitness function is used to evaluate the quality of a solution. A mutation and crossover function are applied to create new solutions. The mutation function converts a solution in a new one. The crossover function specifies the exchange of chromosome parts between a set of parent chromosomes. The mutation rate and crossover rate determine how frequent these operations occur.

For optimization problems, a pool of N initial candidate solutions is first generated. Next, each solution is rated by using the fitness function, after which new N solutions are created by applying the mutation and crossover functions on randomly selected sets of parents based on fitness. The process continues until a satisfactory solution is found or until a predefined round count.

A.6.2 *Artificial Neural Networks.* [Haykin 1994] are computational models with the ability to learn the relationship between an input and output by minimizing the error between the output and expected output. ANN's can be used for finding patterns, and classifying, remembering, and structuring data. An ANN consists of computational units called neurons, which are connected by weighted edges. A single neuron can have several incoming and outgoing edges. When a neuron receives an input, it first combines the inputs of all incoming edges and tests if it is triggered by this input. If the neuron is triggered, it sends the combined signal over the output lines. The ANN functions in an environment, which provides the input signals, processes the output signals, and calculates the error which the ANN can use to adjust the weight on the edges and thereby learn.

A.6.3 *Constraint Satisfaction and Planning.* [Russell et al. 1995] entails finding a path from an initial state to a end state by applying actions. A planning problem consists of an initial state, actions, and a goal test. Planning Domain Definition Language (PDDL) is commonly used to express planning problems. An action can be executed when the initial condition is satisfied. The effect of an action can be the addition or deletion of variables resulting in a new state. Forward state-space search algorithms start planning from the initial state. In contrast, backward state-space search algorithms start in the final state. Despite both types of algorithms, planning is NP-hard in general, which explains the importance of heuristics in planning.

Received February 2011; revised July and December 2011; accepted —