USING PROOF-OF-CONCEPT FEEDBACK TO EXPLORE THE RELATIONSHIP
BETWEEN ARTISTS AND PROCEDURAL CONTENT GENERATION IN
COMPUTER GAME DEVELOPMENT TOOLS

by

Robert Ota Dieterich
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Arts
Computer Game Design

Committee:

___________________________________________  Director

___________________________________________

___________________________________________

___________________________________________

___________________________________________

___________________________________________

___________________________________________  Director of the School of
Computer Game Design

___________________________________________  Dean, College of Visual and
Performing Arts

Date:  ________________________________  Spring Semester 2017
George Mason University
Fairfax, VA
Using Proof-Of-Concept Feedback to Explore the Relationship Between Artists and Procedural Content Generation in Computer Game Development Tools

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Arts at George Mason University

by

Robert Ota Dieterich
Bachelor of Science
College of William & Mary, 2002

Director: Scott Martin, Professor
Department of Computer Game Design

Spring Semester 2017
George Mason University
Fairfax, VA
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>Abstract</td>
<td>vii</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Structure of this Paper</td>
<td>2</td>
</tr>
<tr>
<td>Chapter One: On Procedural Content Generation and Artistic Intent</td>
<td>4</td>
</tr>
<tr>
<td>Discussing Procedural Content Generation from a Design Standpoint</td>
<td>6</td>
</tr>
<tr>
<td>PCG for Games and PCG for Game Tools</td>
<td>11</td>
</tr>
<tr>
<td>The Relationship Between Artist and Algorithm</td>
<td>12</td>
</tr>
<tr>
<td>Chapter Two: A Primer on Terrain Mesh Generation</td>
<td>17</td>
</tr>
<tr>
<td>Terrain Meshes in Video Games</td>
<td>17</td>
</tr>
<tr>
<td>Using PCG to Generate Terrain</td>
<td>19</td>
</tr>
<tr>
<td>Customizing PCG Terrain with Manual Edits</td>
<td>21</td>
</tr>
<tr>
<td>Chapter Three: Overview of Selective Terrain Expression</td>
<td>23</td>
</tr>
<tr>
<td>Chapter Four: Strengths and Weaknesses of Selective Terrain Expression</td>
<td>29</td>
</tr>
<tr>
<td>Chapter Five: Feedback On Selective Terrain Expression</td>
<td>36</td>
</tr>
<tr>
<td>From Reddit</td>
<td>37</td>
</tr>
<tr>
<td>From 3D Artists</td>
<td>39</td>
</tr>
<tr>
<td>Conclusion</td>
<td>44</td>
</tr>
<tr>
<td>Improvements to Selective Terrain Expression</td>
<td>45</td>
</tr>
<tr>
<td>Importance of Granular Output Control</td>
<td>46</td>
</tr>
<tr>
<td>Are Particle Effect Editors Procedural Content Generation?</td>
<td>47</td>
</tr>
<tr>
<td>Procedural Game Tools in Theoretical Frameworks</td>
<td>48</td>
</tr>
<tr>
<td>Appendix: Technical Details of Selective Terrain Expression</td>
<td>52</td>
</tr>
<tr>
<td>Proof-of-Concept Implementation</td>
<td>52</td>
</tr>
<tr>
<td>Procedural Terrain Generation used in Proof-of-Concept</td>
<td>52</td>
</tr>
</tbody>
</table>
Mapping the Procedural Terrain to the Authored Mesh ................................. 54
Controlling the Influence of the Procedural Terrain on the Authored Mesh ........ 57
Preparing the Result Mesh for Use in Unreal Engine 4 .................................. 59
References ........................................................................................................ 61
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>8</td>
</tr>
<tr>
<td>Table 2</td>
<td>9</td>
</tr>
<tr>
<td>Table 3</td>
<td>50</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4</td>
<td>19</td>
</tr>
<tr>
<td>Figure 5</td>
<td>24</td>
</tr>
<tr>
<td>Figure 6</td>
<td>25</td>
</tr>
<tr>
<td>Figure 7</td>
<td>26</td>
</tr>
<tr>
<td>Figure 8</td>
<td>28</td>
</tr>
<tr>
<td>Figure 9</td>
<td>30</td>
</tr>
<tr>
<td>Figure 10</td>
<td>33</td>
</tr>
<tr>
<td>Figure 11</td>
<td>34</td>
</tr>
<tr>
<td>Figure 12</td>
<td>37</td>
</tr>
<tr>
<td>Figure 13</td>
<td>40</td>
</tr>
<tr>
<td>Figure 14</td>
<td>53</td>
</tr>
<tr>
<td>Figure 15</td>
<td>56</td>
</tr>
<tr>
<td>Figure 16</td>
<td>57</td>
</tr>
</tbody>
</table>
ABSTRACT

USING PROOF-OF-CONCEPT FEEDBACK TO EXPLORE THE RELATIONSHIP BETWEEN ARTISTS AND PROCEDURAL CONTENT GENERATION IN COMPUTER GAME DEVELOPMENT TOOLS

Robert Ota Dieterich, M.A.

George Mason University, 2017

Thesis Director: Dr. Scott Martin

This paper explores the relationship between 3D artists and procedural content generation (PCG) in computer game development tools by presenting a proof-of-concept implementation of a new tool to artists with game development experience and examining the feedback. The tool presented for feedback implements a method, dubbed Selective Terrain Expression, by which artist-created 3D models can be selectively combined with procedurally-generated terrain. By analyzing the feedback provided, the author gains suggestions for improvements that can be made to the tool itself as well as insight into the nature of PCG in computer game development tools and the methods used to discuss them in academic research on PCG.
INTRODUCTION

Consider the scenario wherein a computer game artist is tasked with creating the virtual environment for the opening sequence in a computer game. In this scene, the cradle-like spaceship containing the last child of an alien species has crash-landed in a field in the American Midwest, carving a path through the dirt and grass. The artist creates a mesh representing the cradle-ship, generates a landscape and trees using state-of-the-art procedural tools, and then lovingly carves the impact crater into the generated landscape before placing the alien cradle-ship. Later, the artist finds out that, due to a miscommunication by her manager, the setting for the alien child’s crash-landing was not supposed to be a Midwestern farm but a Siberian snow field instead. While the cradle-ship mesh is salvageable, the customized farm terrain must now be thrown out. New parameters (switching from temperate farm to Siberian winter) for the procedural terrain generator creates a new terrain mesh. This new mesh forces the artist to discard the customizations she had made to the original terrain mesh and repeat her work.

Now, consider how this scenario would have played-out had the artist been able to change settings on a procedural terrain generator without discarding her customization of the generated mesh. The shock of sudden re-work would have been highly mitigated if all she had to do was tweak procedural parameters while leaving her cradle-ship and its impact crater where they were.
This thesis explores the relationship between artists and tools that use procedural content generation by presenting a tool that implements the features described in the above hypothetical scenario and collecting feedback from 3D artists with professional game development experience.

The technique implemented for this paper, dubbed Selective Terrain Expression, is a process by which manually modeled 3D geometry may be selectively combined with procedurally generated terrain. This technique allows a 3D artist to create a flat terrain with arbitrary architecture, use a procedural terrain generator to perturb the vertices in the mesh, and then control the strength of the influence from the procedural terrain on a per-vertex level. Once generated, the artist is then able to freely change parameters on the procedurally generated terrain to create different geometry while preventing the newly generated terrain from adversely affecting hand-crafted geometry. The name Selective Terrain Expression is derived from the artist’s ability to granularly control the degree to which the procedurally generated terrain is expressed in the artist-created geometry.

By presenting a proof-of-concept implementation of Selective Terrain Expression to 3D artists and receiving feedback, the author receives insight into, not just improvements to Selective Terrain Expression itself, but into the way 3D artists think about PCG in computer game development tools and how this way of thinking differs from the methods used to discuss PCG in academic research.

**Structure of this Paper**

This paper is structured in three primary sections. The first section provides background information that aids in understanding procedural content generation (PCG)
and terrain meshes (a common form of geometry used to depict terrain in 3D virtual environments.) The first chapter of the paper identifies ideas and language useful for classifying various forms of PCG as well as discussing them from a game design perspective. It also examines the relationship between procedurally generated content and the human artists who create it. The second chapter describes terrain meshes as they pertain to computer games, PCG techniques commonly used to generate them, and an overview of techniques that allow human artists to control the results of procedural terrain mesh generation.

The next two chapters, which make up the second section of this paper, describe Selective Terrain Expression. Chapter Three provides an overview of the technique and the prototypical implementation created for this paper. Chapter Four describes Selective Terrain Expression’s strengths and weaknesses.

The final section of the paper constitutes an analysis of the feedback provided regarding Selective Terrain Expression. Chapter Five examines the feedback received by the author after presenting Selective Terrain Expression to 3D artists and online communities. The paper concludes with lessons gleaned from the feedback conversations regarding improvements to Selective Terrain Expression and the relationship between artists and PCG in tools for computer game development.

An appendix describes Selective Terrain Expression at a greater level of technical detail and should provide suitably skilled individuals with the knowledge they need to implement the technique themselves.
CHAPTER ONE: ON PROCEDURAL CONTENT GENERATION AND ARTISTIC INTENT

Procedural Content Generation (PCG), as a term, covers myriad techniques related to the algorithmic creation of content for a variety of uses. For this paper, I borrow the definition provided by Shaker, Togelius, & Nelson (2016) in Procedural Content Generation in Games: A Textbook and an Overview of Current Research. In this book, the authors define PCG as “the algorithmic creation of game content with limited or indirect user input” (Shaker et al., 2016) which they synthesize from Togelius, Kastbjerg, Schedl, & Yannakakis (2011), an earlier work by one of the authors.

The book continues to refine its definition of PCG by using the taxonomy for dissecting the role of search-based PCG described in Togelius, Yannakakis, Stanley, & Browne (2010). The taxonomy identifies five factors for distinguishing PCG: (a) Online versus Offline, (b) Necessary versus Optional Content, (c) Random Seeds versus Parameter Vectors, (d) Stochastic versus Deterministic Generation, and (e) Constructive versus Generate-and-test. Online versus Offline distinguishes between systems that generate content online during the runtime of a game, or offline during its development. Necessary versus Optional Content distinguishes between systems which generate content necessary to player progression (e.g. dungeons) versus that which is not (e.g. background scenery.) Random Seeds versus Parameter Vectors identifies the degree to which a procedural content generator uses random number generators versus sets of
parameters to determine their resulting content. Stochastic versus Deterministic Generation identifies algorithms that will produce varying (stochastic) versus identical (deterministic) output over multiple runs with the same parameters. Constructive versus Generate-and-test identifies PCG that directly generate final output versus those that generate candidate output that is tested against some sort of fitness criteria and regenerated on failure. Daily challenges in Spelunky (Yu, 2013), shown in Figure 1, is an example of Online, Necessary Content, Deterministic procedural content generator. In Spelunky, daily challenges are procedurally generated levels that are created using a random seed of the day. Because the levels are generated in a deterministic way, players with the same seed value can experience the same daily challenge level.
Daily challenges in Spelunky (Yu, 2013) are an example of Online (levels are generated at runtime) Necessary Content (the players must play the levels to advance) Deterministic (all players get the same daily challenge level) PCG.

**Discussing Procedural Content Generation from a Design Standpoint**

While the taxonomy described in the preceding section provides a model for identifying the operational features of a given procedural content generator, other models help understanding PCG in terms of its relationship with a game’s design and its designers. In other words, they help us understand the relationship between procedural content generators and the people who use them. Khaled, Nelson, & Barr (2013) present four metaphors that categorize the relationship of PCG to the designer. Smith (2014) proposes a framework for understanding PCG based on the Mechanics, Dynamics, Aesthetics (MDA) framework established by Hunicke, LeBlanc, & Zubek (2004).
The four design metaphors proposed by Khaled et al. (2013) are TOOL, MATERIAL, DESIGNER, and EXPERT. Under the TOOLS metaphor, PCG implementations are viewed as being designed to optimize the performance of their users and thus aid in achieving said users’ design goals. MATERIALS are understood as substance without specific purpose or context. The output of SpeedTree (2016) is a MATERIAL because, while it’s ostensibly designed to create forests, the context and actual use of the output is determined by the artist after-the-fact. The DESIGNER metaphor represents PCG algorithms that are tasked with solving game design problems with little artist intervention. Finally, EXPERTS encode knowledge about a particular domain and apply it by monitoring and interpreting data resulting from gameplay. Khaled et al. (2013) distinguishes EXPERT from DESIGNER with the latter encoding expertise in game design and the former encoding expertise in domains outside of game design. It behooves someone designing a PCG solution to carefully think about the relationship between the solution and its user and how it can be identified within Khaled et al. (2013)’s metaphor.

The MDA framework, proposed by Hunicke et al. (2004), establishes three major lenses through which to examine the design of games: Mechanics, Dynamics, and Aesthetics. Building upon MDA, Smith (2014) defines a framework for understanding the role of PCG in games. Within the Mechanics domain, Smith identifies four aspects related to both how a generator works as well as how a designer is expected to use it. The four aspects and their corresponding values are summarized in Table 1.
Table 1
Summary of the mechanical aspects of Procedural Content Generation.

<table>
<thead>
<tr>
<th>Building Blocks</th>
<th>Game Stage</th>
<th>Interaction Type</th>
<th>Player Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential Chunk</td>
<td>Offline</td>
<td>None</td>
<td>Indirect</td>
</tr>
<tr>
<td>Large, human-authored</td>
<td>Before game</td>
<td>No human influence</td>
<td>No direct experiential control</td>
</tr>
<tr>
<td>Template</td>
<td>Online</td>
<td>Parameterized</td>
<td>Compositional</td>
</tr>
<tr>
<td>Computer fills in the blanks</td>
<td>During game</td>
<td>Indirect, human sets values</td>
<td>Human influences available components</td>
</tr>
<tr>
<td>Component Pattern</td>
<td>Preference</td>
<td>Human selects good products</td>
<td>Experiential player experience</td>
</tr>
<tr>
<td>Small, human-authored</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subcomponent</td>
<td>Direct Manipulation</td>
<td>Human manipulates product</td>
<td></td>
</tr>
<tr>
<td>Internal representation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note. Reprinted from "Understanding procedural content generation: a design-centric analysis of the role of PCG in games," by Gillian Smith, Copyright 2014 by Gillian Smith.

In the Dynamics domain, Smith (2014) identifies aspects which help us identify the way game rules interact and how PCG enables these interactions. The Other Mechanics aspect, in particular, identifies the degree to which PCG is central to the dynamics of a particular game. Table 2 summarizes the dynamics aspects of the framework. The first two take on set values while the remaining four represent potential properties of the procedural content generator.
The combination of mechanics and dynamics serves to produce certain aesthetic experiences. Out of the eight “kinds of fun” identified by Hunicke et al. (2004), Smith (2014) highlights three: Discovery, Challenge, and Fellowship. When used to create varied environments, PCG supports Discovery as players explore a near infinite selection of configurations. By using PCG to generate new tests of player skill, a game can support the Challenge aesthetic. This is particularly the case for a game that uses dynamic difficulty adjustment (Hunicke, 2005) to guide the outputs of its content generators. Finally, PCG can encourage Fellowship as players discuss successful strategies for managing the varying challenges that a PCG game may present.
By examining PCG solutions through Smith’s framework, we can identify the aspects of the solution that most affect its interaction with artists and designers. Paying attention to how aspects of the framework, particularly those in the Mechanics domain, are engaged can provide PCG designers with a window into how their solutions affect the artists who use them.

While the taxonomy presented in Togelius et al. (2010) provides a useful vocabulary for describing PCG in technical terms, conceptual frameworks like Khaled, Nelson, & Barr (2013) and Smith (2014) facilitate conversations about PCG in terms of the people that use them. While Togelius et al. speaks to what a given procedural content generator does, Khaled et al. and Smith help us speak about PCG in terms of what we (as human designers) want them to do and how we relate to them. As it is described in this paper, Selective Terrain Expression falls under Khaled et al.’s MATERIAL metaphor. The prototypical implementation on which this document is based focuses on taking an artist-authored mesh and selectively expressing a procedural terrain on parts of the mesh. Such a mesh could then be exported to produce a suitable asset for use in a 3D game engine. While a technique like Selective Terrain Expression may be usable in an online (as defined by Togelius et al. and Smith) context, it is arguably more suited for offline use. The ability of an artist to define the degree of procedural influence on a per-vertex basis in Selective Terrain Expression indicates that its interaction model falls under the Direct Manipulation category as defined in Smith’s framework.
PCG for Games and PCG for Game Tools

When considering how PCG is designed, a useful distinction can be found in considering whether PCG is intended for use in the tools used to develop computer games or in the computer games themselves. When used in games themselves, PCG is often used in service of the player to produce near-limitless variations on the game’s content. The goal is such cases is generally providing entertainment through novelty for the player.

On the other hand, when using PCG in game tools, the audience served by the PCG is not the player but the computer game artists that create the game instead. In such a case, the goal of the PCG can be considered as providing convenience and expressive range to the artist. Consequently, the requirement of control over outputs is greatly increased when considering PCG for tools. Novelty remains a factor as an artist may cycle through random seeds to find appealing outputs, but the primary measure of a PCG game tool is in its ability to aid the artist in expressing their vision.

The design metaphors proposed by Khaled et al. (2013) of TOOL, MATERIAL, DESIGNER, and EXPERT provide some help in understanding this distinction. PCG that match the TOOL and MATERIAL metaphors may be commonly found in tools for creating games, but are less commonly found in games themselves. On the other hand, Togelius et al (2010) and Smith (2014), while providing several useful vectors for classifying PCG, do not present explicit factors for distinguishing between PCG used in games and PCG used in tools for making games.
The Relationship Between Artist and Algorithm

In 1997, Ken Perlin received a Technical Achievement Award from the Academy of Motion Picture Arts and Sciences for the development of the Perlin Noise algorithm (Perlin, n.d.). Perlin Noise is a well-known technique for producing data that simulates many naturally-occurring phenomena and is widely-used in a variety of PCG. While getting an Oscar may be considered a great acknowledgement of artistic merit, the relationship between artists and PCG algorithms is not always so harmonious.

In his Game Developer Conference 2015 talk, “Art Direction Bootcamp: How I Learned to Love Procedural Art”, No Man’s Sky (2016) art director Grant Duncan, in reference to learning about PCG from the programmers at his company, states: “I was quite reluctant. I think a lot of artists are a little bit intimidated by procedural generation. It's very alien and it takes a lot of control away from us.” (Duncan, 2015). As one may surmise from the title of the talk, Duncan eventually embraced using PCG to create No Man’s Sky’s content (pictured in Figure 2), but his statement highlights one of the major points of contention artists have when working with PCG. This is further emphasized later in the talk when he lists a series of common artist complaints with regards to PCG, saying: “Taking control away from artists was the number one thing that people seemed to be scared of.” (Duncan, 2015).
Figure 2
The variety of flora, fauna, and terrain present in *No Man’s Sky* represent a combination of human artistic direction and procedural content generation.

While the cost benefits of using an algorithm to do the work that would have taken days of labor by a team of human artists may be obvious (assuming the cost of developing / licensing an appropriate solution is less than the wages saved), there are costs to replacing human artistic work with PCG that go beyond the financial. In particular, the loss of artistic control intrinsic in surrendering data production to PCG may be beyond what is acceptable to the developers of a game.

Naturally, this depends on the nature of the assets being created. Consider the case of *The Witcher 3: Wild Hunt* (2015), a game in which the player controls an adventurer named Geralt as he travels through large and very detailed virtual environments. In order to reduce the human labor cost of producing these environments, the developers used tools like SpeedTree (2016) to automate processes like populating...
forests with trees. While the developers of *The Witcher 3: Wild Hunt* surrendered a certain degree of control of where individual trees were placed to SpeedTree, they were willing to make this sacrifice, as it did not compromise their artistic vision for the game in a particularly significant way. It is very much less likely that they would have accepted a similar sacrifice when dealing with their main cast of characters, who represent the face of the game’s artistic vision.

It is worth noting that this tension between artistic control and the benefits of procedural content generation is greater felt when using PCG in an offline context. When using PCG in an online context, i.e. at runtime, PCG presents an advantage that cannot be replicated through artist effort: the production of novel content while a game is running. While online use of PCG can be thought of a place of natural advantage for PCG, it does not eliminate the conflict between artistic control and automatic content generation. Even while using PCG in an online context, there can be a strong desire to execute artistic control over the results.

At times, the relationship between artist and PCG can be antagonistic. *Mass Effect: Andromeda* (2017), a tent-pole game title developed by BioWare and published by Electronic Arts, met with significant criticism of its animations, facial animations in particular (Hernandez, 2017). In a video claiming to have first-hand information from BioWare staff, Liam Robertson identifies over-reliance on automated technology as a scape goat for *Mass Effect: Andromeda*’s lackluster facial animation (Robertson, 2017). According to Robertson's sources, outputs produced through cyberscanning, a "process by which an extremely thin, low-energy laser beam is used to scan an actor's face or
body" (Colmenarez, Xiong, & Huang, 2013), were not sufficiently edited by 3D artists to produce ideal results. Robertson’s sources continue by suggesting that the impetus for the reliance on automated technology stemmed from the desires of BioWare management for cost reduction. This represents a case in which the way PCG was used stood in direct conflict with the artistic intention of the game’s artists.

Despite the potentially contentious loss of artistic control, PCG is used significantly in many areas of game design. This suggests the significant value of PCG to game development that, in certain cases, overrides the cost of lost artistic control. And yet, there are myriad instances of game development where designers opt to abhor PCG in favor of hand-crafted content and the direct artistic control it allows. This suggests a delicate tug-of-war that occurs between the potential benefits of PCG to a game’s production and the loss of artistic control inherent in the use of PCG. In other words, the choice to use PCG to generate assets for a game is tempered by the need of the human artists to exert control over their artistic voice.

Selective Terrain Expression aims to be an example of a tool that allows artist use of PCG while preserving their ability to control the output on a granular level. As a technique that serves to combine procedurally generated geometry with hand-authored geometry, Selective Terrain Expression lends itself to use in offline contexts. In essence, it allows the artist to choose between modeling manually and using procedural-generated terrain on a per-vertex level. Considering the tension between artistic vision and procedural content, we might expect Selective Terrain Expression’s granular output control to be a welcome addition a procedural tool. By examining feedback from 3D
artists to Selective Terrain Expression, we can confirm this theory and gain insight into the interplay between PCG and artistic control.
CHAPTER TWO: A PRIMER ON TERRAIN MESH GENERATION

As Selective Procedural Terrain Expression deals with the combination of artist-created meshes with procedurally-generated terrain meshes, it is useful to have a working knowledge of terrain meshes, their relation to computer game development, and the procedural techniques used to generate them. This chapter presents a brief overview of this subject.

Terrain Meshes in Video Games
Considering the great number of computer games that implement 3D environments, it is no surprise that the creation of the assets that make up such environments comprises a significant amount of a computer game’s development effort. This is evidenced in the frequency with which roles such as “environmental artist” appear on game development job listing sites, like Gamasutra Jobs. Within the realm of game environments, outdoor environments are often implemented with a technology called a terrain mesh. For this paper, I define terrain mesh as a mostly contiguous array of quads laid out on a plane where the perpendicular height values of individual vertices relative to the plane vary in such a way as to express ground topology. An example of such a quad array is shown in Figure 3.
A terrain mesh is a contiguous array of quads whose vertices represent the height of terrain features.

In general, terrain meshes are convex with features that require concave topology like caves or holes being introduced after modeling of the terrain mesh is completed. As terrain editors can be found in a variety of game development software, including Unity 5 (2017) and Unreal Engine 4 (2017), it is safe to say that game developers find terrain meshes to be useful and expressive tools for creating their games’ environments. These terrain editors provide a suite of tools for shaping terrain meshes into a variety of shapes as can be seen in Unreal Engine 4’s terrain editing tool shown in Figure 4.
Using PCG to Generate Terrain

In addition to defining metaphors for understanding PCG from a design perspective, Khaled, Nelson, & Barr (2013) identifies two major trends that motivate game PCG research: 1) the development of systems that facilitate game development, and 2) adapting game content to the skills and preferences of players in both offline (prior to play) and online (during play) contexts. This first trend can be seen in the use of PCG in tools designed to procedurally generate terrain meshes.

Given the ubiquity of terrain meshes in 3D games, it’s no surprise that there is a significant amount of literature concerning generating terrain meshes using PCG. Chapter 4 of Procedural Content Generation in Games: A Textbook and an Overview of Current
Research by Shaker, Togelius, & Nelson (2016) describe several techniques for generating heightmaps using stochastic and agent-based methods. These heightmaps (2D arrays of height values) can be applied to a plane of quads to generate a terrain mesh. Many game development tools, like Unreal Engine 4, support the import of externally generated heightmaps as an input for generating terrain meshes. Other approaches, such as Terrain Sketching (Gain, Marais, & Straßer, 2009) and Declarative Terrain Modeling (Smelik, Tutenel, de Kraker, & Bidarra, 2010) introduce models by which artists can create ambiguous data suggestive of terrain features that a computer can then use to create concrete terrain mesh data.

Gain, Marais, & Straßer (2009) present a terrain sketching technique by which a user draws a silhouette that the algorithm then extrapolates into a landscape feature, like a mountain or island. This technique allows an artist to provide a vague suggestion of their intent that the PCG is then able to flesh out into a fully-formed asset. Like SpeedTree (2016) does to the placement of foliage, this tool simplifies artist effort at the cost of fine-grained control over the produced terrain mesh.

Smelik, Tutenel, de Kraker, & Bidarra (2010) propose a technique for integrating PCG and manual editing when creating virtual worlds with a technique called Declarative Modeling. Through Declarative Modeling, a designer expresses their intent through semi-abstracted notions. For example, a designer may define swathes of rocky terrain mixed with flat terrain through which a river runs. They would then define a rough path a road would take through the terrain. Next, they would outline areas that contain a city. Having
done this, procedural content generators produce virtual terrain that matches these specifications.

In spirit, this is similar to the work of Gain et al. (2009) but executed on a much larger scale with more complex outputs. Similarly, this approach trades ease-of-modeling against fine-grained control of the final product.

**Customizing PCG Terrain with Manual Edits**

Even with the added artistic control provided by Declarative Terrain Modeling, Smelik, Tutenel, de Kraker, & Bidarra (2011) note that many designers, game designers in particular, express a desire for more control over the output, stating:

Although the approach gives designers proper control at a high level, especially game designers commented that they miss even more fine-grained user control, for instance to tune generated objects (e.g. individual buildings or trees) to precisely match their intent or artistic vision. (Smelik, Tutenel, de Kraker, & Bidarra, 2011)

In a similar vein, but from a different perspective, game developer Robin-Yann Storm, in his talk at the Game Developers Conference 2016, laments about the inherent limitation of heightmap-driven landscapes:

I’m still wondering why some time in the past we decided we didn’t need cliffs or caves anymore and settled with using models to cover them all up. And it just increased the works for artists; saddling a designer to deal with the problem.

(Storm, 2016)
These comments suggest an appetite for methods that allow the editing of terrain meshes, particularly those that are procedurally generated, with custom mesh data that goes beyond the expressive bounds of the PCG techniques that created them, or even the limits of the terrain mesh data structure itself.

While there is certainly financial value in replacing human artist work with a reasonably priced procedural solution, the effect this has on artistic value is debatable. In some cases, an automated solution may be accepted because it does not compromise the artistic vision of the game developers. However, depending on the process being automated, the loss of artistic control intrinsic in surrendering data production to PCG may go beyond what is acceptable to the developers of a game. I suggest that enhancing PCG solutions with techniques that enable human customization of the output would encourage their use in such solutions.

Based on this reasoning, a technique that eases the mixing of artist-created meshes with procedurally-generated terrain would surely be welcome. By providing a system that allows an artist to non-destructively combine procedural landscapes with custom meshes, such a technique provides utility by allowing an artist to experiment more freely with procedural parameters without having to worry about the destruction of customized mesh features. Through a proof-of-concept implementation of Selective Terrain Expression, we can substantiate this reasoning by examining artist feedback.
CHAPTER THREE: OVERVIEW OF SELECTIVE TERRAIN EXPRESSION

This paper opens with a story about a hypothetical 3D artist creating a virtual environment for an equally hypothetical computer game project. In that story, a 3D artist is forced to re-create the environment due to a miscommunication in the development process. While the general thrust of the environment, that it involved a crashed spaceship that left a path of displaced ground in its wake, remains the same, the setting for the environment shifts from temperate farmland to snowfield. The introduction continues by suggesting that the amount of effort required to rework the virtual environment would be reduced if the 3D artist could selectively combine hand-crafted geometry (the spaceship and the scars its crash-landing leave in the ground) with procedurally generated terrain. Selective Terrain Expression attempts to provide such a solution.

Selective Terrain Expression is a technique by which a specially prepared artist-created mesh is combined with a procedurally-generated terrain mesh to produce a result mesh that incorporates features of both the hand-authored mesh and the procedural mesh. The process allows the artist to choose the degree to which any vertex in the artist-created mesh is affected by vertices in the procedural mesh, hence the use of the word “selective” in Selective Terrain Expression.

The artist-created mesh, often referred to as the authored mesh, is a mesh that has been prepared by a 3D artist. This mesh contains arbitrary geometry created by the artist.
as well as a lattice of vertices specifically prepared to express the geometry of the procedurally-generated terrain. Figure 5 presents an example of such an authored mesh.

![Figure 5](image)

*Figure 5*
An artist created mesh featuring a detailed tower, a simple road leading to the tower, and a plane of vertices prepared to express features of the procedural terrain

Selective Terrain Expression automatically generates a terrain mesh that will be combined with the authored mesh. While Selective Terrain Expression does not specify what method should be used for procedural terrain generation, it does assume the generated terrain is a contiguous set of connected vertices with no two vertices occupying the same coordinates on the ground plane. Any terrain mesh generated by a conventional heightmap fits this description. The proof-of-concept implementation of Selective Terrain Expression uses a rudimentary fractal heightmap to generate its procedural terrain mesh as depicted in Figure 6:
The proof-of-concept implementation generates a procedural terrain mesh by first creating a plane of quads that matches the dimensions of the provided authored mesh. It then generates a heightmap that it uses to displace the height of the vertices of the plane to create the procedural terrain. The resolution of the procedural terrain is exposed to the user as a parameter. Values used to determine the height of terrain features and the random seed used in heightmap generation are also provided as parameters.

Selective Terrain Expression combines the procedural terrain with the authored mesh by modifying the positions of select vertices in the authored mesh based on the positions of relevant vertices in the procedural terrain. Conceptually, this is similar to 3D animation via blend shapes, a process by which the position of vertices in one mesh are morphed to the positions of corresponding vertices in a target mesh. Unlike blend shapes, Selective Terrain Expression does not attempt to exactly match the positions of vertices.
in the procedural terrain. Instead, it attempts to generally express features of the procedural terrain in the vertices of the authored mesh. In doing so, Selective Terrain Expression avoids requiring a one-to-one mapping of vertices in the authored mesh to vertices in the procedural terrain. This also allows for multiple vertices in the procedural terrain to influence the final position of vertices in the authored mesh. The result of this technique can be seen in the smoothing of procedural terrain features as they are expressed in the result mesh shown in Figure 7:

![Figure 7][1]

A mesh resulting from an authored mesh being modified via Selective Terrain Expression

Comparing the result mesh in the Figure 7 with the original authored mesh in Figure 5 reveals that, while many vertices are changed by the expression of procedural terrain, some vertices are unchanged. The tower and the path leading to the tower are not affected by the expression of procedural terrain. The degree to which any vertex in the
authored mesh is affected by the procedural mesh is determined by values in a UV channel. While UVs are traditionally used to map vertices to a texture, Selective Terrain Expression repurposes them by using the value of the U channel to determine how much influence each vertex in the authored mesh receives from the procedural terrain. Values closer to 0 increase influence from the procedural terrain while values closer to 1 reduce the influence. A value of 1 negates the influence entirely. In the example data, much of the flat plane has U values of 0 while features like the tower and the road have U values of 1. Vertices that transition between the flat plane and the tower/road have values between 0 and 1 to ease the transition from expressed terrain to authored geometry. The proof-of-concept implementation uses UV channel 3 so as not to override values in lower UV channels used for texture mapping.

The proof-of-concept implementation of Selective Terrain Expression is implemented as a 3ds Max (2017) plugin. By leveraging features of this software, it was trivial to export the geometry produced by Selective Terrain Expression in a format suitable for runtime use in a commercial game engine. Figure 8 shows the example data used in this chapter running in Unreal Engine 4 (2017), a one such game engine.
The goal of Selective Terrain Expression is to broaden the options available to 3D artists when creating virtual environments, particularly for computer games. By allowing an artist to specify areas of geometry that should or should not be affected by procedural content generation, this technique helps 3D artists by allowing them to concentrate their attention on features that they feel they should hand-craft while leaving the generation of less important features up to a procedural algorithm. While the technique presented in this paper focuses solely on the combination of hand-authored geometry with procedural terrain data, there is much room for exploration into applying a similar technique to a wider range of geometry.
CHAPTER FOUR: STRENGTHS AND WEAKNESSES OF SELECTIVE TERRAIN EXPRESSION

Selective Terrain Expression is a technique that encourages the combining of procedurally generated content and hand-authored content. As mentioned earlier in the paper, procedural content generation (PCG) highlights an intrinsic conflict between authorial control and the convenience of automated content generation. Selective Terrain Expression serves as an example of a class of tools that seek to reduce this conflict by providing the artist with explicit, fine-grained control of the resulting data while still exploiting the advantages of procedural content generation.

With regards to defining the amount of influence vertices in the authored mesh receive from the procedural terrain, the proof-of-concept implementation provides a solution that does not introduce an unreasonable amount of extra work. In the example data, most vertices can trivially be set to values of 0 or 1 using standard 3ds Max UV mapping tools. Only vertices that transition from the procedural terrain to the authored mesh require fine tuning of their influence values. This fine-tuning can be performed in 3ds Max’s UV editor with the modifier stack’s Show End Result feature enabled to provide real-time feedback to the artist.

The procedural terrain generation in Selective Terrain Expression’s proof-of-concept implementation uses a seed value to determine the output of its random number generation. Because of this, changing the seed value produces a new variation of the
procedural terrain. More importantly, the terrain produced is the same given the same seed value, a crucial property for enabling artistic control. The artist can change the seed value to quickly produce new variations of the procedural terrain and instantly combine them with their hand-authored data. One such alternate variation on the example data used in this paper is shown in Figure 9.

![Figure 9](image)

**Figure 9** Changing the random seed value quickly produces new variations of the procedural terrain combined with the authored mesh.

With this quick variation changing enabled by the random seed value, an artist can quickly try multiple variations until they find formations that most appeal to their artistic vision. As the influence control data exists independently of the procedural terrain generation, the artist can change terrain generation parameters without fearing data loss to their authored mesh.

In a similar fashion to changing seed values, an implementation of Selective Terrain Expression that supports multiple procedural terrain algorithms would allow the
artist to easily change terrain generation technique. Again, the artist would be able to do this without risking the integrity of their hand-crafted geometry.

By using a mapping method based on searching for vertices in the procedural terrain within a certain distance of vertices in the authored mesh, Selective Terrain Expression manages to partially decouple the structure of the authored mesh from the procedural terrain. This frees the authored mesh from being required to have a one-to-one mapping (of vertices that receive influence) with vertices in the procedural mesh. This increases flexibility in the how the artist can structure their authored mesh and configure the procedural terrain generation.

That being said, the mapping method used in the proof-of-concept implementation does require the authored mesh and procedural terrain to meet some minimal structural requirements before producing acceptable results. As mentioned before, an authored mesh must provide a lattice of vertices suitable for expressing procedural terrain features. As Selective Terrain Expression neither adds nor removes vertices from the authored mesh, the meshes ability to express procedural terrain features is limited by how many vertices are available to be translated. It is also limited by the initial location of said vertices. Using the methods implemented in the proof-of-concept, this requires the authored mesh to provide a plane of evenly spaced quads. The proof-of-concept implementation also requires that the lattice have the same dimensions and number of subdivisions along the X and Y axes, i.e. a square lattice. This is due to a limitation in the procedural terrain generator’s implementation and not a core property of Selective Terrain Expression itself.
While producing the lattice of vertices itself is a relatively simple task, it does require the artist to have foreknowledge of what geometry is best suited for expressing the desired procedural content. A shift from one procedural content generation technique to another may require a shift to a different lattice geometry. For example, while a plane of quads effectively expresses heightmap-based terrain, an alternate PCG technique, like a Voronoi diagram ("Voronoi diagram," n.d.), may be better expressed through a lattice of hexagons. A late-stage change in lattice could represent a significant amount of work as the artist is responsible for merge the lattice with their hand-authored geometry. It may be possible to mitigate this by pairing procedural generation of lattices that are appropriate to the desired procedural mesh generation technique. For lattices composed of simple geometry, it may be feasible to automate the welding of vertices from authored meshes with those of the lattice to create contiguous meshes. The controlled nature of lattice vertices would likely provide a simpler case to solve than general welding of arbitrary meshes. Naturally, such an automatic welding feature would need to be optional in order to allow the artist final control over how vertices in their authored meshes are handled. Procedural lattice generation with automatic welding would also have the added benefit of allowing the artist to move their authored geometry around more freely within the lattice as they would not have to manually redo welding.

In addition to the authored mesh providing a suitable lattice for terrain expression, the procedural terrain generator must be configured to produce terrain meshes of a sufficient resolution for effective expression by the authored mesh. Failure to provide enough vertices in the procedural terrain results in degenerate meshes. Figure 10 shows
the result of mapping example authored mesh to procedural terrain of an insufficient resolution.

Figure 10
A procedural terrain with an insufficient resolution produces a degenerate result mesh.

In addition to the structural requirements on the authored mesh and the procedural terrain described above, the XY planar mapping used in the proof-of-concept implementation places significant limits on the kind of procedural meshes that can be successfully expressed by the plugin. Because Z values are ignored while searching during mapping, vertices in the procedural terrain will, by necessity, be mapped to the same vertices in the authored mesh. While this does not present a problem for heightmap-based terrain like that used in the proof-of-concept, it limits the domain of procedurally-generated meshes that can be expressed. For example, procedural content generators that produce buildings that rise straight along the Z-axis would not be expressible using the current method.
Selective Terrain Expression does not provide any facility for dynamically changing the height of hand-authored features in the result mesh. This results in vertices defined as taking no influence from the procedural terrain always keeping the same position regardless of what features are being expressed around them. If a large hill, for example is generated around the tower in the example data, this results in the tower being buried in a crevice cut into the hill as shown in Figure 11.

This carving effect may or may not be desirable depending on the artist’s intention. Were Selective Terrain Expression to include a feature allowing the position modification process to optionally raise or lower the Z coordinate of non-influenced vertices in a consistent manner, this would increase the artist’s ability to define features that should exist on the peaks of procedurally generated hills. Unfortunately, the proof-of-concept implementation does not currently support this sort of functionality.
In its current state, Selective Terrain Expression only deals with the merging of vertices between the authored mesh and the procedural terrain. It does not handle the merging of combining of UV values, materials, or other data related to the texture mapping of the result mesh. These factors are obviously important for the appearance of result meshes presented to end-users. Considering how such data should be best handled when combining meshes will be an important to the refinement to the technique.
CHAPTER FIVE: FEEDBACK ON SELECTIVE TERRAIN EXPRESSION

The interaction between artists and algorithm lies at the heart of Selective Terrain Expression and this paper. By allowing an artist to granularly control a mesh’s vertices and the degree to which they receive influence from procedurally generated data, Selective Terrain Expression invites artists to mix hand-modeled features with PCG when creating terrain meshes. It is therefore crucial to consider the opinion of artists when presenting a solution like Selective Terrain Expression. Any tool, regardless of its potential, is useless until adopted by the craftspeople who would use it. Likewise, a tool built in isolation from the desires of those it seeks to serve runs the risk of serving no purpose at all.

This paper refers to two major sources for feedback on Selective Terrain Expression and its underlying ideas. The first source is Reddit, a major social news aggregation site with sub-channels devoted to specific topics. More specifically, the author posted to the /r/proceduralcontentgeneration category of Reddit, a gathering place for individuals interested in applications of procedural content generation. The second major source of feedback is informal interviews with 3D artists who have had a significant degree of experience in game development and have been involved in the development of multiple shipping titles (i.e. games that have been released for sale to the public.) These sources serve to provide perspective that ranges from PCG enthusiasts to
those who primarily evaluate PCG based on its ability to solve problems encountered during game development.

**From Reddit**

The author approached the /r/proceduralcontentgeneration community by posting links to two videos demonstrating the Selective Terrain Expression proof-of-concept plugin at different levels of completion. These videos were first uploaded to YouTube and then posted to Reddit for comment from the community. The first video (Dieterich, 2017a) presents a simplistic building-like structure on a low-resolution grid being modified using an early iteration of the proof-of-concept plugin. This video, a frame of which is shown in Figure 12, served as an initial public announcement of the project.

![Figure 12](image.png)

*Figure 12*

A frame of the first video presenting Selective Terrain Expression to /r/proceduralcontentgeneration for feedback.

After implementing several optimizations to increase the performance of the proof-of-concept plugin, the author produced a second video demonstrating the Selective
Terrain Expression plugin (Dieterich, 2017b). While the first video has no sound, this second video includes narration describing how the plugin works and some of the ideas presented in this paper. Due to the optimizations, the version of the proof-of-concept plugin present in the second video handles a more complicated, higher-resolution mesh. The mesh demonstrated in the second video is the same mesh featured in the Selective Terrain Expression plugin screenshots shown earlier in this paper (see Figure 7 for an example).

The videos of the proof-of-concept received a modest amount of attention on /r/proceduralgeneration. In terms of upvotes, the system by which Reddit users vote for content that they believe should be shown higher in the Reddit user interface, the first video earned 49 upvotes and the second earned 6 upvotes. This level of attention likewise translated into views on YouTube with each of the videos having hundreds of views at the time of writing this paper. Also at the time of this writing, comments on the Reddit posts were largely composed of follow-up questions and requests for clarification. These comments did not express a particularly strong opinion about the solution.

While the conversation on Reddit did not result in significant direct feedback on the Selective Terrain Expression project itself, it did provide exercise for the rhetoric used to describe it. For example, one commenter compared Selective Terrain Expression to blend shapes, a technique in which mesh vertices are transformed to match the positions of matching vertices in a target deformation mesh. This technique is commonly used to animate mouth shapes for facial expressions and lip synching. Unlike Selective Terrain Expression, blend shapes often rely on the mesh topology of the deformation
mesh exactly matching the topology of the base mesh. The exercise of explaining distinctions such as this proved useful in shaping some of the arguments used in this paper.

**From 3D Artists**

Once the Selective Terrain Expression proof-of-concept plugin reached a demonstrable level of functionality, the author conversed with multiple experienced 3D game artists to attain feedback on the plugin and the ideas leading to its creation. Each of the individuals interviewed possess multiple years of experience in producing 3D art for professional game development companies. As such, they provide a perspective on Selective Terrain Expression and the problems it attempts to solve that is steeped in a pragmatic understanding of the needs and desires of professional 3D game artists.

The interviewed artists largely agreed with the notion of there being an intrinsic tension between the convenience of procedural content generation and a desire for artistic control. This coincided with an idea that a key aspect of a game artist’s job is to be particular about the data they present to the player. With regards to 3D meshes, this manifests itself in a desire to control (or at least have the option to control) every vertex that the player may see. Like any human, 3D artists are bound by constraints of time and energy and so must focus their resources on areas that will produce greater returns for their efforts. To appeal to an artist, a PCG solution must offer suitable returns for the effort it requires and the limitations it entails. To paraphrase one of the interviewed artists: PCG is a tool; the artist’s job is to make cool looking things. From their
perspective, PCG, like any other tool in their palette, is only as useful as its ability to enable the artist to create evocative game art.

An interesting corollary that arose from conversation about PCG as tool is the categorizing of which tools regularly employed by game artists constitute PCG. One interviewed artist suggested that visual effects (VFX) tools, such as the Unreal Engine 4 (2017) particle effect tool pictured in Figure 13, can be considered PCG tools. From the artist’s perspective, such a tool procedurally produces novel content based on the parameters (texture to be displayed, particle emission rate, particle lifetime, etc.) provided to the tool by the artist.

![Unreal Engine 4’s particle effect editor can be considered an example of a tool that uses procedural content generation.](image)

Considering particle effect editors as a form of PCG tool opens a variety of other tools used in game development to consideration as PCG. Physics simulations, which can be used to generate dynamic behavior for a variety of objects in game development, are
particularly apt for inclusion. Such simulations are crucial for programatically producing movement in procedural animation staples like cloth and hair physics. They are equally important in the implementation of ragdolls, collections of joints and rigid-bodies whose simulations are used to drive the behavior of limp characters. Taken to the extreme, interpolation between key-frames may also be considered a form of PCG but this may stretch the definition beyond the point of utility.

This inclusion of particle effect editors and physics-based animations under the umbrella of PCG is also notable as they are not mentioned in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research* by Shaker, Togelius, & Nelson (2016). Perhaps the authors would consider the degree of input required by most particle effect editors to go beyond the bounds of “limited or indirect user input” as per their definition of procedural content generation. This may likely be the case considering arguments present in Togelius, Kastbjerg, Schedl, & Yannakakis (2011) where the authors argue that offline player-created content does not constitute PCG. Still, this may indicate a point of disconnect between what is considered PCG by artists and researchers. It is also worth noting that the content procedurally produced by particle effect editors and physics simulations is motion, placing them in a category sometimes identified as procedural animation. Much of the PCG discussed in Shaker, Togelius, & Nelson’s textbook can be described as producing outputs that can exist statically in virtual space, i.e. 3D geometry, 2D pixels, and game levels. Animation, procedural or otherwise, gets little mention in the book. This may indicate a bias away from
considering animation in procedural content research that does not align with the instincts of 3D artists when defining what constitutes a PCG tool.

The core feature of Selective Terrain Expression, allowing the artist to control the use of PCG terrain on a per-vertex level, was positively received. The interviewed artists expressed the idea that they are ultimately responsible for the quality of the visual data in their games. This responsibility occurs regardless of whether the data was produced painstakingly by-hand or algorithmically through PCG. As such, they appreciate the ability to adjust results on a highly granular level. One artist mentioned that while PCG may provide convenience, it can also be creatively limiting.

Selective Terrain Expression’s ability to allow the artist to continue to change procedural parameters while maintaining existing edits was also interesting to the artists. In many offline workflows using procedural terrain, an artist may generate several variations of terrain and then choose one to customize. Once this customization happens, the artist rarely revisits the original terrain’s procedural parameters.

One of the artists interviewed relayed his experiences on working with a system in a past project that sought to solve similar problems to Selective Terrain Expression. This system allowed artists to markup up areas of terrain as requiring certain geological features (e.g. flat lands, plateaus) required for the placement of certain in-game buildings while other terrain would be generated procedurally. They then used this system at runtime to generate a variety of environments for the players to enjoy. In a similar vein, Selective Terrain Expression can start with an artist mesh created at authoring-time.
perform its procedural terrain generation and blending at runtime to achieve similar results.

A comment of note by one of the interviewed artists was that, in his experience, game designers are more open to PCG than game artists. This may come as little surprise when one considers the notion of the role of PCG in games versus that of PCG in game creation tools. Game designers, who are arguably more aligned with the desires of players, may be naturally drawn to PCG’s ability to create runtime novelty. Meanwhile, a game artist must consider how PCG will aid or hinder them in expressing their artistic intent. Bear in mind that this is not to say that game designers do not share an equally strong compulsion to express artistic intent in their work. Instead, it is better to consider runtime novelty to more likely align with the creative intent of game designers than of game artists.

As a proof-of-concept, there was little expectation that the reviewed implementation of Selective Terrain Expression would compare to the level of refinement found in professionally developed tools. Be that as it may, the interviewed artists expressed interest in the solution and its strategy of providing granular control of procedural output. In these conversations with 3D game artists, flexibility and choice were common themes. The mission of these artists is to create compelling content. Mastery of a variety of tools, procedural or otherwise, is key to this mission. As such, they are willing to try various tools and ultimately judge them on their ability to enable the artist to express their vision.
CONCLUSION

This paper began by identifying a potential use-case for procedural content generation (PCG) in a game development tool in the form of a hypothetical scenario where a 3D computer game artist avoided re-work thanks to a PCG tool. To test the reasoning behind this scenario, the author developed a proof-of-concept demonstration of the technology, dubbed Selective Terrain Expression, described in the hypothetical scenario. He then reached out to multiple experienced 3D game artists to receive feedback on the technology.

On one hand, this feedback process served to gather suggestions for improvements on Selective Terrain Expression. On another, it served as a vehicle to explore the relationship between 3D artists and the procedural content generation in their toolsets. In exploring this dynamic between game artists and procedural content generation, differences between how artists and researchers view PCG became apparent.

This final chapter of the paper synthesizes lessons learned from the feedback sessions with regards to Selective Terrain Expression itself as well the relationship between artist and PCG. It also examines that relationship’s place in a theoretical understanding of PCG and suggests improvements to conceptual frameworks for understanding PCG as it pertains to tools.
**Improvements to Selective Terrain Expression**

Many suggestions for improvements to Selective Terrain Expression came out of the feedback sessions. This section covers some of the more salient suggestions.

A common suggestion was to allow for alternate mapping methods when determining the per-vertex influence from the procedural terrain. While mapping via UV channel is effective, alternate methods such as vertex colors can potentially provide a better user experience depending on the operations allowed by the 3D modeling tool being used. This flexibility is doubly important as the availability and usage of vertex data may vary depending on the runtime solution employed. For example, vertex colors may be used at runtime for their intended purpose of tinting vertices, but they are often repurposed for a game’s specific needs.

In its current state, the Selective Terrain Expression plugin requires the artist to provide a suitable lattice of vertices for the procedural terrain to perturb. As such, creation of a suitable lattice requires planning and some level of knowledge of the plugin’s internal logic. Adding a feature that automatically generates a suitable lattice mesh for the plugin’s settings would reduce the work required at this stage. On a more ambitious note, adding logic that automates stitching of the artist-created geometry into the lattice mesh would significantly decrease work. That being said, programmatically merging arbitrary geometry can be a non-trivially complicated task. This is somewhat mitigated by the relatively simple nature of lattice geometry. This simplicity serves to simplify the geometry merging problem space.

One of the interviewed artists suggested allowing for a setting allowing a variable amount of height influence on a per-structure (i.e. per-building) basis. The purpose of this
would be to allow buildings to be raised evenly in accordance to terrain should results like those depicted in Figure 11, where a tower is embedded into a procedurally-generated hill, were deemed unacceptable.

The proof-of-concept implementation of Selective Terrain Expression maps the authored mesh to the procedural terrain in a planar fashion on XY. This makes it impossible for the authored mesh to express features in the procedural terrain that exist solely on the Z axis. This is not a problem for the current procedural terrain generation method, but it limits the domain of geometry expressible with Selective Terrain Expression. Exploration into alternate mapping methods and procedural terrain generation strategies is necessary to expand the range of output possible with the technique.

**Importance of Granular Output Control**

Selective Terrain Expression’s principal feature of allowing granular control of outputs was positively received by the interviewed game artists. This is not a surprise considering the importance of artistic control and expression to their jobs. This suggests that designers of PCG tools would do well to consider how artists can exercise precise granular control of the outputs from their systems.

For some applications of PCG it may suffice customize outputs after generation with no expectation of returning to generation. For example, an artist may produce several random terrain meshes, choose the one that best suits their needs, and then customize the terrain with the specific features they desire. In such a scenario, the artist
cannot regenerate the terrain without destroying their edits but this may be acceptable to them.

Alternatively, an artist may wish to preserve their manual edits through multiple regenerations of procedural content. This is particularly true if the manual edits involve a large amount of work or if they need to support generating the procedural content at runtime.

**Are Particle Effect Editors Procedural Content Generation?**

In reading contemporary PCG research and receiving feedback from 3D artists, it becomes clear that the delineation of what one may or may not consider to be “procedural content generation” is fluid. In Togelius, Kastbjerg, Schedl, & Yannakakis (2011), the authors explicitly identify what they do and do not consider to constitute PCG. Building on that work, they arrive at the definition present in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*: “the algorithmic creation of game content with limited or indirect user input” (Shaker, Togelius, & Nelson, 2016). However, in feedback discussions with 3D game artists, tools like particle effect editors and physics engines enter the conversation as procedural tools. These tools which produce motion programmatically are often classified as procedural animation tools and are a common staple of game development toolsets. Given the input requirements of such tools, even a lax reading of Shaker et al.’s definition finds such tools excluded from the category of PCG. Does this mean that a particle effect editor is definitely not PCG? Or does this perhaps suggest a need to broaden the definition of what constitutes procedural content generation?
While tools like particle effect editors may not fit into the formal definition of PCG, it can be detrimental to ignore them with regards to PCG research. Likewise, it can be detrimental to ignore PCG research when examining such tools. Factors identified in theoretical frameworks for understanding PCG, like the Togelius, Yannakakis, Stanley, & Browne (2010) taxonomy and the Smith (2014) framework, can help identify particularly useful features of such tools. For example, Jonathon Blow’s time-bending game *Braid* (2008) implemented particle effects based on a random seed and a deterministic simulation (Blow, 2010). This structure is unusual for particle effects and not implemented in many solutions. However, due to this non-standard implementation, Blow could support particle effects in *Braid*’s rewind mechanic without exceeding the game’s memory budget; a feat that would have been impossible with a standard implementation. An analysis based on Togelius et al.’s taxonomy brings the use of random seed and determinism into relief when compared against other particle systems and illuminates why Blow’s particle system was uniquely suited to the problem it addressed.

Given the applicability of PCG research to procedural animation tools, it behooves the researcher to give said tools at least some consideration in the study of PCG. Whether it is more useful to consider procedural animation tools as falling under the umbrella of PCG or to consider them a sibling area of study is open for debate.

**Procedural Game Tools in Theoretical Frameworks**

Should one opt to extend the bounds of PCG study to include procedural animation tools or not, it is prudent to examine how PCG tools are referenced in
theoretical frameworks. Khaled, Nelson, & Barr (2013) provides explicit mention of tools in its TOOL and MATERIAL metaphors for understanding the role of PCG in game design. The framework proposed by Smith (2014), while it makes some reference to AI-based tools (i.e. mixed-initiative tools), is more oriented to understanding PCG in games than in game tools. This is natural as the Mechanics, Dynamics, Aesthetics framework (Hunicke, LeBlanc, & Zubek, 2004) on which it is based is primarily player-oriented.

The taxonomy presented by Shaker, Togelius, & Nelson (2016), because it focuses on structural behavior of PCG, can be read as being ambivalent as to whether PCG is being used in a computer game or a game development tool. The contents of the textbook, however, suggest a focus similar to Smith (2014). Most of the specifically tool-related content focuses on mixed-initiative tools. This is not to suggest that Shaker et al. (2016) or Smith (2014) have purposefully ignored PCG in game tools. Rather, it is a sign that game-related PCG research is largely focused on in-game PCG employed for game player consumption rather than PCG used in tools for game artist consumption.

To more fully consider PCG as it is used in game development tools, it may be useful to extend existing theoretical frameworks with tool-centric values. For example, identifying “expression” as an important experience within the Aesthetics domain of Smith’s framework can serve to identify a crucial experiential need of artists using PCG in tools. This use of “expression” is a slight repurposing of the term from the original MDA term which is described as “game as self-discovery” (Hunicke, LeBlanc, & Zubek, 2004). The term can easily be read to signify expression of the user’s (i.e. artist’s) intent and therefore serve as an identifier for this crucial artistic need. With such an extension in
place, one can reason that PCG that does not serve “expression” in its aesthetic experience would be ill-suited to use in a game tool.

Considering the importance of granular output control to artists, it would be useful to consider the degree to which a PCG tool supports it on a mechanical level. Just as the addition of Aesthetics domain of the framework established in Smith (2014) can be extended to support the concept of expression, the Mechanical domain can be extended to support the concept of granular output control. I propose adding an aspect with the values listed in Table 3 to express this concept.

<table>
<thead>
<tr>
<th>Granular Output Control</th>
<th>None</th>
<th>Post Generation</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>No granular control of outputs</td>
<td>Granular modifications not preserved on subsequent content generation</td>
<td>Granular modifications preserved through subsequent content generation</td>
<td></td>
</tr>
</tbody>
</table>

By considering the Granular Output Control aspect as part of a PCG tool’s analysis, one can reason effectively as to a PCG tool’s ability to serve artist needs. PCG that supports no granular control of outputs would likely be considered unsuitable for use in a tool. PCG tools that only support granular modification on may only be suitable if the PCG output is generated in an offline context. PCG tools that preserve modifications through multiple content generations may allow the content generation to be performed in
an online context as well as encourage the artist to more freely change procedural parameters.

Procedural content generation is an important aspect of many games. As such, research on PCG is largely oriented towards the study of PCG as it occurs in games. However, it is also an important aspect in many of the tools that are used to create games. By studying the relationship between these tools and the artists that use them, we can get greater insight into their vital features.
APPENDIX: TECHNICAL DETAILS OF SELECTIVE TERRAIN EXPRESSION

This appendix discusses various technical details related to Selective Terrain Expression and its proof-of-concept implementation.

Proof-of-Concept Implementation
The proof-of-concept implementation of Selective Terrain Expression was developed by the author of this paper. The proof-of-concept implementation was developed as a 3ds Max (2017) plugin in order to leverage 3ds Max’s rendering and editing toolset. It was developed as a native plugin using C++. The plugin derives from the SimpleMod2 class which in turn derives from SimpleMod which derives from Modifier. As a child of Modifier, the plugin operates by modifying the vertex data passed up through the 3ds Max modifier stack. It does not add or remove vertices. Transformations are performed in object space. The plugin takes the geometry it receives through the modifier stack as the artist-created authored mesh. The procedural terrain mesh is generated by the plugin itself and is never explicitly shown to the end user. Only the procedural terrain’s effect on the authored mesh is shown.

Procedural Terrain Generation used in Proof-of-Concept
The procedural terrain used in the proof-of-concept implementation of Selective Terrain Expression is a plane of vertices whose Z values are distorted based on the values of a heightmap. The heightmap is a grayscale image generated procedurally using the
diamond-square algorithm ("Diamond-square algorithm," n.d.). An example of a heightmap generated using this technique is shown in Figure 14.

![Figure 14](image)

A heightmap generated using the diamond-square algorithm

Diamond-square produces its output by gradually subdividing the texture while averaging values diagonally (the diamond step) then horizontally (the square step) and adding a random value. The diamond-square averaging and randomization process continues using increasingly smaller subdivisions of the texture until some minimum subdivision size is reached. The implementation of the diamond-square algorithm used for this paper generates consistent output given the same seed value for random number generation. This consistency is key to providing the 3D artist with a degree of control over the procedural terrain generation.
Once a heightmap is generated, the value is applied to the Z value of each vertex in a programmatically generated plane of quads. The number of quads in the plane is determined by the resolution of the generated heightmap. The dimensions of the plane are set based on the extents of the authored mesh in the X and Y axis. The final Z-value of each vertex is determined by multiplying the corresponding value from the heightmap with a height parameter provided by the user. The result of this process is a procedurally generated terrain mesh as shown earlier in Figure 6.

While the proof-of-concept implementation uses diamond-square to generate its procedural terrain, Selective Terrain Expression is not limited to this one algorithm. In fact, any terrain generated using a heightmap should be compatible with the techniques used in the current implementation.

**Mapping the Procedural Terrain to the Authored Mesh**

Because Selective Terrain Expression does not enforce a one-to-one correspondence of vertices between the authored mesh and the procedural terrain mesh, it is necessary for the technique to establish a mapping between the two meshes. Naturally, this mapping must establish a relationship that is acceptable and reasonably intuitive to the 3D artist using the Selective Terrain Expression plugin.

The proof-of-concept implementation maps vertices of the authored mesh to vertices of the procedural terrain in a planar fashion on the XY plane. In other words, the Z coordinate of the vertices are ignored when determining their mapping. For each vertex in the procedural terrain, the implementation searches for vertices within the authored mesh that are within a set radius. Finding relevant vertices is supported by using a k-d
tree ("k-d tree," n.d.) to increase search speed. The found vertices are recorded as vertices that can be influenced by the current vertex in the procedural mesh. The amount of influence a vertex in the authored mesh receives is determined by its distance from the vertex in the procedural terrain. The smaller the distance, the larger the influence of the procedural terrain vertex. The distance between vertices in the authored mesh and the procedural terrain is calculated using their planar XY positions. Their Z coordinates are ignored when calculating distance. For optimization purposes, authored mesh vertices that take no influence from the procedural terrain are ignored during mapping.

The mapping process described in the preceding paragraph produces a list of 0 or more influencing vertices in the procedural terrain for each vertex in the authored mesh. Each influencing procedural terrain vertex is marked with the degree of influence it asserts on a given authored mesh vertex based on distance. The influencing vertices are then combined and averaged relative to their influence to produce a combined influence point for each authored mesh vertex. These averaged influence points are used to determine the modified position of the authored mesh vertex when creating the result mesh.

The mapping of multiple vertices in the procedural terrain to vertices in the authored mesh produces a smoothing, or blurring, effect of features from the procedural terrain as they are expressed in the result mesh. This smoothing effect can be increased by increasing the radius used to find matching vertices in the procedural mesh. Figure 15 demonstrates the smoothing effect of doubling the search range used in Figure 7.
Likewise, a reduction in search range reduces the number of vertices that are found, thus reducing the smoothing in terrain expression. Figure 16 displays the effect of reducing the search range to the point where only one vertex in the procedural terrain is mapped to any given vertex in the authored mesh.
Decreasing the search range reduces the smoothing effect and produces a more literal expression of the procedural terrain.

It is worth noting that the reduced search range results in a more literal expression of the procedural terrain shown in Figure 6. Depending on the 3D artist’s intention this may be desirable over the smoothing effect provided by a larger search range. The proof-of-concept plugin supports this choice by parameterizing the search range for use by the 3D artist.

Controlling the Influence of the Procedural Terrain on the Authored Mesh

Supporting explicit artistic control of vertices is an intrinsic property of Selective Terrain Expression. The proof-of-concept implementation does this by allowing the 3D artist to specify a per-vertex U value in a particular UV channel. This U value specifies the degree to which a given vertex in the authored mesh receives the influence of vertices in the procedural terrain. This influence control variable specifies a value between 0.0 and 1.0 where 0.0 indicates complete influence from the mapped procedural terrain vertices and 1.0 indicates an authored mesh vertex that is completely unaffected by the
procedural terrain. This allows the artist to, in effect, protect certain vertices from modification by the procedural terrain. The artist is also able to specify partial influence values which aid in controlling transitions between the procedural terrain and the authored mesh.

The proof-of-concept implementation uses the U value of UV channel 3 to specify influence control. As 3ds Max allows for 99 UV map channels, they are not a significantly limited resource. While the proof-of-concept uses channel 3 as a hard-coded constant, it would be trivial to parameterize the channel used for greater artist control.

There is no requirement that vertex selection be handled through a UV channel. It is not uncommon for meshes to repurpose vertex color data or encode data in mapped textures for a variety of rendering applications. Encoding values in a texture may sacrifice the ability to encode unique values per-vertex depending on texture resolution. Repurposing vertex colors provides a way to specify values per-vertex at the cost of using vertex colors for other purposes, including its literal purpose of coloring vertices. Encoding techniques such as these could be useful in a scenario where Selective Terrain Expression is performed in an online PCG context, i.e. at runtime.

The proof-of-concept implementation opts for the UV channel approach as it is producing its result mesh in an offline PCG manner. Once the data is prepared for display in a game engine, the UV channel-based influence control setting is baked into the result mesh and no longer used.
Preparing the Result Mesh for Use in Unreal Engine 4

The conceptual scenario around which the proof-of-concept implementation is built is that a 3D artist, after using the 3ds Max plugin to combine an authored mesh with procedural terrain, would convert the result mesh into a format suitable for their target game engine and export the data for later import into a game engine. Once imported into a game engine, the result mesh is handled no differently than meshes produced through typical means. Should the 3D artist desire to rework the data, they would be expected to return to 3ds Max to perform their edits. After performing edits, they would once again export their mesh for reimport into their target game engine. This is workflow common in typical asset production pipelines.

For a mesh produced by the proof-of-concept implementation to be displayed in Unreal Engine 4 as demonstrated in Figure 8, the following procedure was followed. First, the Selective Terrain Expression plugin and the preceding modifiers used to create the authored mesh are collapsed into a single Editable Poly. This is done most as a precaution against unintentional edits lower in the modifier stack invalidating edits higher in the stack, particularly in the UV channels required by Unreal. Next, two UV channels are added to the mesh: one for materials applied in Unreal and one for lightmaps calculated by the game engine. These UV channels are 1 and 2, respectively. This additional work of adding UV channels could be avoided if the Selective Terrain Expression implementation were extended to blend the existing UV channels of the authored mesh and the procedural terrain in an acceptable manner. This feature, however, does not exist in the proof-of-concept implementation.
After UV channels are created, a Turn to Poly modifier is added to limit the polygon size to 3. Unreal Engine 4 manages meshes in terms of triangles and so polygons with greater than 3 sides must be converted to triangles at some point in the export/import process. The Turn to Poly modifier makes the triangle conversion explicit and allows for artist verification of the result before export. Finally, the mesh is exported in Autodesk FBX format with smoothing groups enabled.

With an FBX version of the result mesh available, the artist can import the mesh into Unreal via a drag-and-drop interface. Once the mesh is imported, it can be handled in the Unreal Editor in the same manner as any other mesh asset.
REFERENCES


Dieterich, R. (2017a). GMU Grad Thesis Project Demo #1 (Selective blending of an arbitrary mesh with procedural terrain) [Video file]. Retrieved from https://www.youtube.com/watch?v=0BOjoQp-OiM


Robert Ota Dieterich graduated high school from The Maret School, Washington, DC, in 1998. He received his Bachelor of Science with a concentration in Computer Science from the College of William & Mary in 2002. He is a veteran game developer who spent 10 years of his career working in Tokyo, Japan. He is credited on multiple games including *Elite Beat Agents* for the Nintendo DS, *Lips* for the Microsoft Xbox 360, and *Infinity Blade Cross* on iOS. He is currently president of Skyboy Games and received his Master of Arts in Computer Game Design from George Mason University in Spring 2017.