Procedurally Generated Genetic Keys

Senior Project

Adam Levasseur

Advised by Prof. Zoë Wood

Abstract

This project presents a method for creating multi-part models based on input keys to generate new, variant models via genetic algorithms. By utilizing 3D models as modular parts, this method allows for the generation of a unique, compound model based on one or multiple input keys. This paper explains the process of creating and testing such generation styles using simple geometry to create more complex, compound models.

Introduction

Repetitive loading of 3D models for the sake of small changes, or variants, can consume valuable time. By loading fewer, modular 3D models and procedurally combining their geometry, an application can use less time spent loading models and more time spent on the rest of the application. When considering the application for video games, this style of procedural generation fits to create new and often interesting content. The purpose of this project is to explore the process of creating a procedural generation application and designing an evaluation function to automatically produce successor models.

The system designed for this project uses C++ as the language of choice and implements OpenGL graphics and simulates physics based on the Bullet Physics engine. The OpenGL code used is an adaptation of base code supplied by Professor Wood for the "Introduction to Computer Graphics" course offered at CalPoly. This base code was originally designed to be able to display single, non-partitioned .obj models and to utilize GLSL shaders for Phong-reflection shading—as designed during the course. For this project, the base OpenGL code has been adapted to display multi-partitioned .obj models and prepared in C++ classes for Bullet Physics implementation. The Bullet Physics engine was originally implemented to simulate collisions and offer physics calculations for the simulated models. Since the engine has been implemented with the modified OpenGL code, collisions can now be simulated, but physics calculations and retrieving certain physics-based data for the simulated models proved too limited to offer significant enough information. Therefore, the Bullet Physics engine is only partially used, in this system, to handle mass and contact calculations while manual calculations are used to fill most of the calculation voids [2].

Figure A: Original Concept for Procedurally Generated Melee Weapons

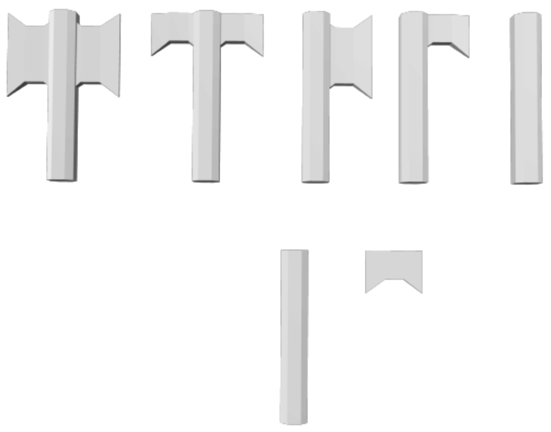


Figure B: Original Early Evaluation Criteria for Generated Swords

- Center of mass within lower third of model to simulate a balanced sword
- Top of model must merge to a single vertex
- Lower third of model must have at least one separate material/mass to simulate a handle

Optional portions of model include:

- <u>Pommel</u>: separate material than handle to bring center of mass closer to handle
- <u>Guard/hilt</u>: aesthetic portion that may also bring center of mass closer to handle
- <u>Handle Guard</u>: aesthetic portion that alters the center of mass to give the sword a direction use; thus altering the performance of the sword if implemented

The original objective and theme of this project was to procedurally create and evaluate 3D models of melee weapons (e.g. axes, swords, shields, etc.). The original concepts and evaluation criteria can be seen in **Figure A** and **Figure B**. However, due to the limitations of the Bullet Physics engine and the lack of mechanical physics knowledge for measuring aspects like "sharpness" and blunt force, the project was re-oriented to produce a simpler genetic algorithm for 3D keys. The rest of this paper focuses on this new theme and the evaluation process.

Related Work

"Borderlands 2"

The game "Borderlands 2" was the initial inspiration of this project. In the game, the player adventures through a post-apocalyptic style world seeking an ancient treasure while fending off countless foes that block their way. One particularly mechanic that the game used was the randomization of weapons. Each gun found in the game is procedurally generated based on a set of possible attributes, model parts, materials, ammo type, and even the game's internal gun brands. Through this mechanic, a game that would usually have a simple, linear adventure path becomes more interesting as the player searches for better weapons and further engaging with the gameplay mechanics. The original objective of this project was to recreate this style of procedural generation while comparing the generated models against each other to determine the better model.

Borderlands 2 Concept Art [1]

"Evolving Virtual Creatures" [3]

One paper that inspired the development of this project is the 1994 SIGGRAPH paper by Karl Sims [1]. In this paper, Sims explains a method for procedurally generating 3D virtual creatures using connected graphs, L-systems, and neural networks to generate both morphologies and control behaviors. L-systems and graph-based models for procedural generation utilizes better programming techniques as well as produces a cleaner, more organized structure for generating new 3D models. L-systems, like the ones discussed in Sims' conference paper, allow for greater repetition of procedural generation in a modular format. Sims' development created modular appendages, each entailing their own style and magnitude of mechanical movement (i.e. rotation, sliding, etc.). This type of development for procedural models allows for more configurable generation where set limits can be used to produce less randomized results and more specific model generation to satisfy a task or evaluation function. This particular strategy for developing 3D models opens the way to more physically, practical tests as well as allowing the combination of artificial intelligence techniques to enhance its procedural generation.

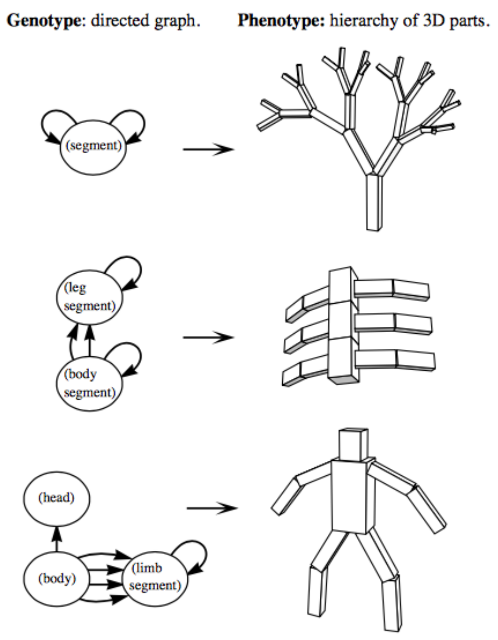


Figure 1: Designed examples of genotype graphs and corresponding creature morphologies.

Sims' Diagram for Connected Graphs to Model Phenotypes [3]

Algorithm

Often geometry modelled in editing programs such as Blender and Maya are partitioned into separate shapes or meshes, rather than one continuous mesh. These multi-part models allow for selective graphical features like color, texture, and animation. This project utilizes the selectivity of multi-part models to procedurally generate a multi-part model from an individual model.

This project uses one strategy for procedurally generating multi-part models using two input keys to seed a pseudo-random number generator. The two keys are designated as the Trait and Mutation Master seeds. The Trait seed determines both the number of components introduced to the original mesh or model, and the number of transformations or traits that apply to each component. The Mutation Master seed determines each component's Mutation seed for applying transformations (e.g. translation, rotation, or scale) as well as the magnitude of each of the applied transformations/traits. Together, these two seeds are used with an initial mesh or model to generate a new multi-part model as seen in **Figure C**.

Figure C: Model Creation using Pseudo-Random Seeds

Trait Seed A Produces	Translation to Model Creation
4 (N)	4 Components to add
3 (T_0)	Component #0 has 3 traits
1 (T_1)	Component #1 has 1 trait
0 (T_2)	Component #2 has 0 traits
4 (T_3)	Component #3 has 4 traits
…	*unused*

Mutation Master Seed B Produces	Translation to Model Creation
8 (M_0)	Component #0 Mutation seed
17 (M_1)	Component #1 Mutation seed
47 (M_2)	Component #2 Mutation seed
24 (M_3)	Component #3 Mutation seed
…	*unused*

M_0 Seed (8) Produces	Translation to **Component #0** Mutation(s) (T_0 = 3 traits)	
2	Choose attachment point #2	
8	Trait #0:	Scale X
39	Trait #0:	Magnitude: 39
12	Trait #1:	Scale Z
73	Trait #1:	Magnitude: 73
16	Trait #2:	Rotation X
34	Trait #2:	Magnitude: 34
…		*unused*

M_1 Seed (17) Produces	Translation to **Component #1** Mutation(s) (T_1 = 1 trait)	
4	Choose attachment point #4	
2	Trait #0:	Scale Y
31	Trait #0:	Magnitude: 31
…		*unused*

Trait Seed

The first key, or Trait seed, is shown in the upper-left table of **Figure C**. It uses the first number generated, N, to determine the number of additional model geometries, or components, for the initial model. The next N numbers generated by the first seed determine the number of geometric transformations, considered as traits, for each component (T_0, T_1, …, T_{N-1}). Finally, for aesthetic purposes, these N numbers generated are also used to color the component model.

Mutation Master Seed

The second seed is shown in the upper-right table of **Figure C**. It generates the next generation of seeds that are used to determine which transformations are applied and their magnitudes (M_0, M_1, …, M_{T-1}). This determination is done by using this second generation of numbers to seed the pseudo-random number generator for randomly determining where and how to attach the new components (numerically seen in the lower tables of **Figure C**). First, every component is allowed one trait to randomly translate the component to one of the available attachment points on the currently generated multi-part model (as seen in **Figure D**). After the component's attachment point is determined, the second generation of numbers are used to apply transformations with random magnitudes (as seen in **Figure E**). Finally, after all trait transformations have been applied, the algorithm translates the component to the attachment point.

Figure D: Parent model with attachment points displayed

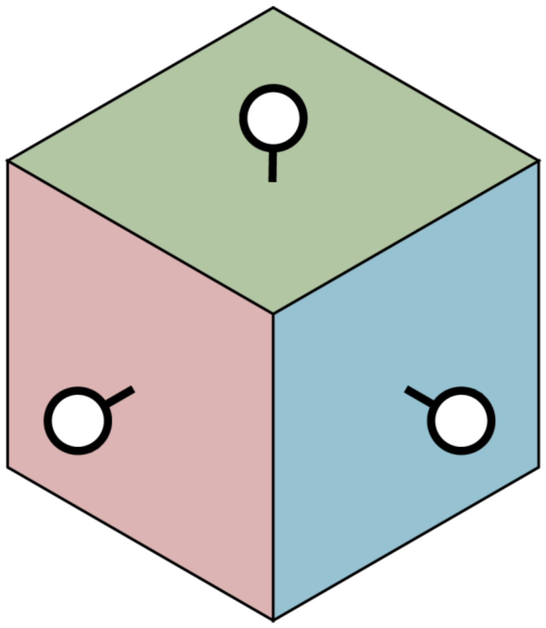
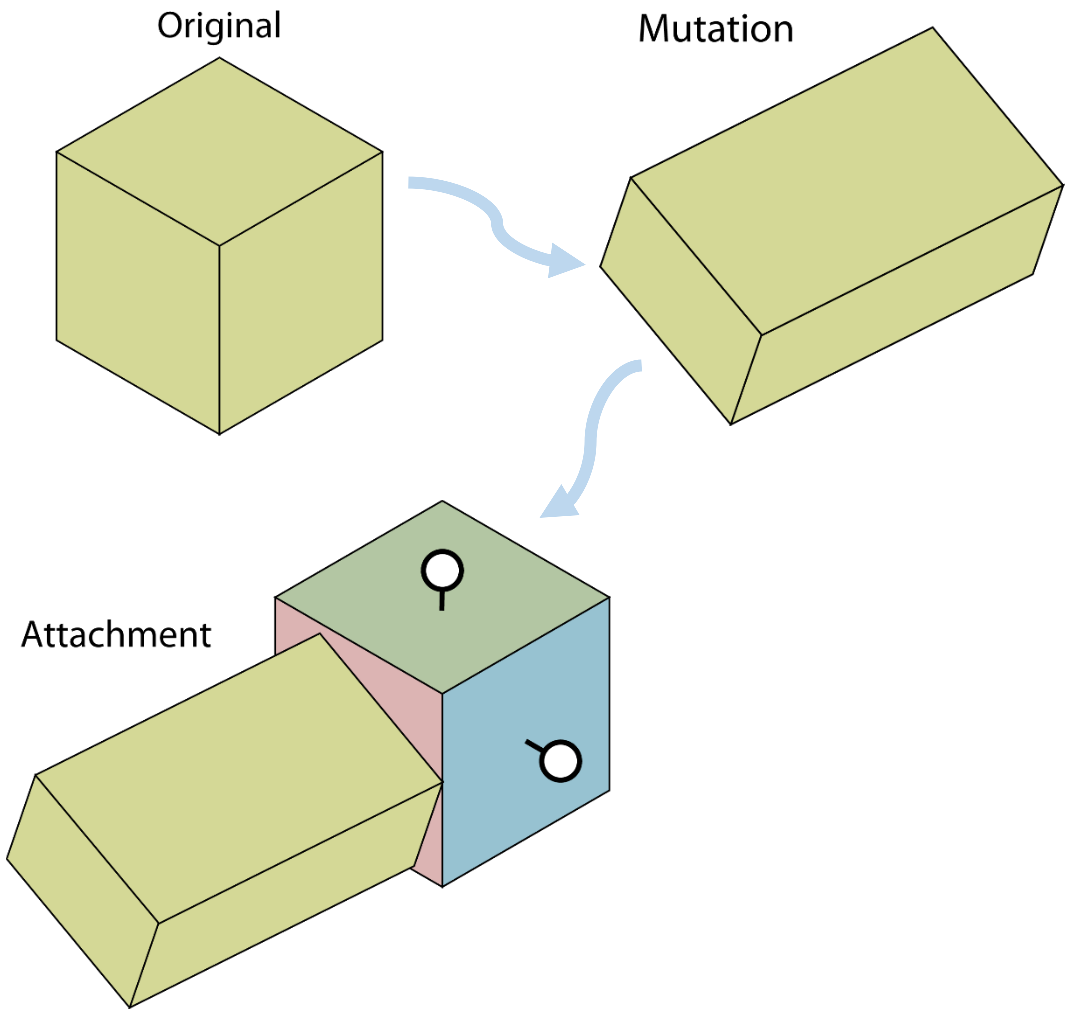


Figure E: Component application to parent model



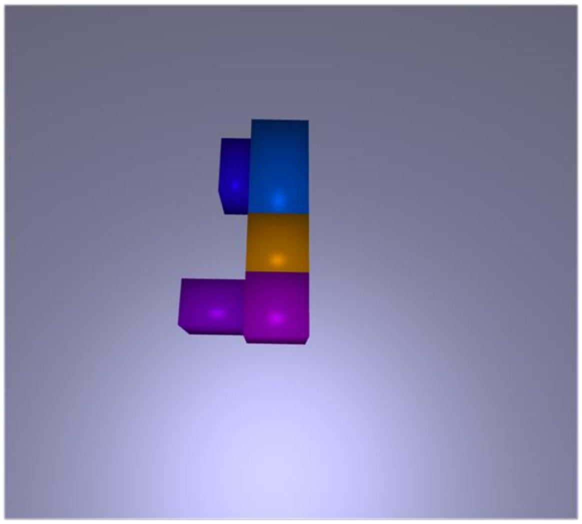
Genetic Mixing

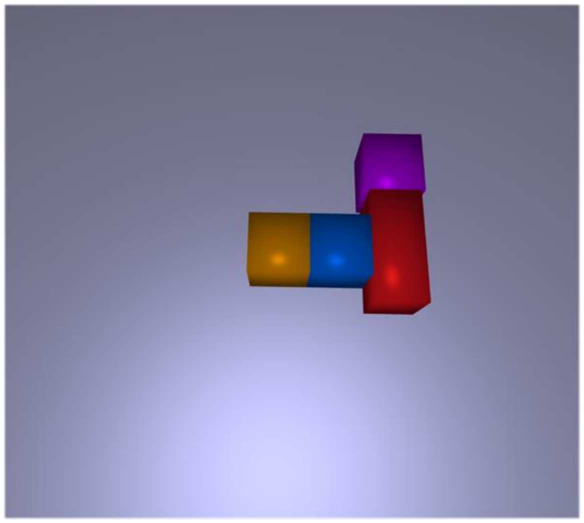
Once a model is constructed, it stores the original seed keys (Trait seed and Mutation seed) to allow for the passing of "genes" to offspring models. When using two generated parent models, the offspring model is created using a mixture of their seed keys (i.e. Trait seed from parent A and Mutation seed from parent B). **Figures F** and **G** present a two generations of procedurally generated models using a single-gene model to produce two variant, child models.

Parents

In **Figure F**, Parents A and B have been generated with different Trait and Mutation Master Seeds. Looking at the **Parent A** image, an initial model (shown as the orange cuboid) with four components (shown as the two purple and two blue cuboids). Similarly, in **Parent B**, an initial model (shown as the orange cuboid) with three components (shown as the blue, red, and purple cuboids) is generated.

Figure F: First (Parent) Generation of Keys

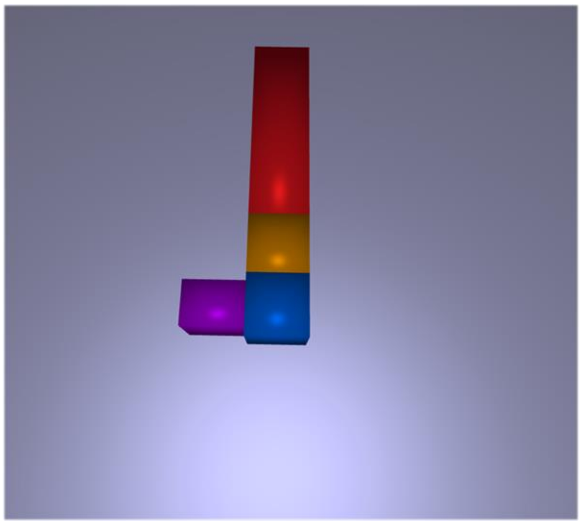

Parent A
Trait Seed: **208**, Mutation Master Seed: **176**


Parent B
Trait Seed: **243**, Mutation Master Seed: **133**

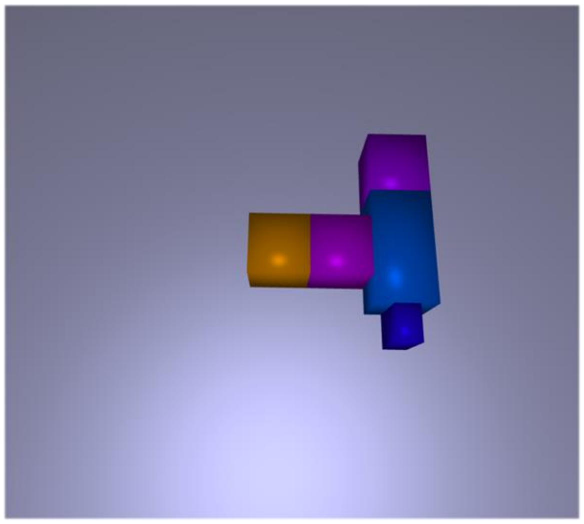
Children

In **Figure G**, Child B/A and A/B have been generated using the genotype seeds of the previous generation (Parent A and B). In these images, **Child B/A** is seen to have a similar genotype as Parent A while showing a similar phenotype as Parent B. The genotype of Parent A is shown as the Mutation Master Seed is inherited and produces the similar model transformations and placement as Parent A's components. On the other hand, the phenotype of Parent B is shown through the inheritance of the Trait Seed to produce the same number of components (three components) and to generate the same random colors. Likewise, in **Child A/B**, the phenotype of Parent A (through the number of components their colors) and the genotype of Parent B (through the components' mutation transformations and placements) are shown.

Figure G: Second (Child) Generation of Keys



Child B/A
Trait Seed: **243**, Mutation Master Seed: **176**



Child A/B
Trait Seed: **208**, Mutation Master Seed: **133**

Results

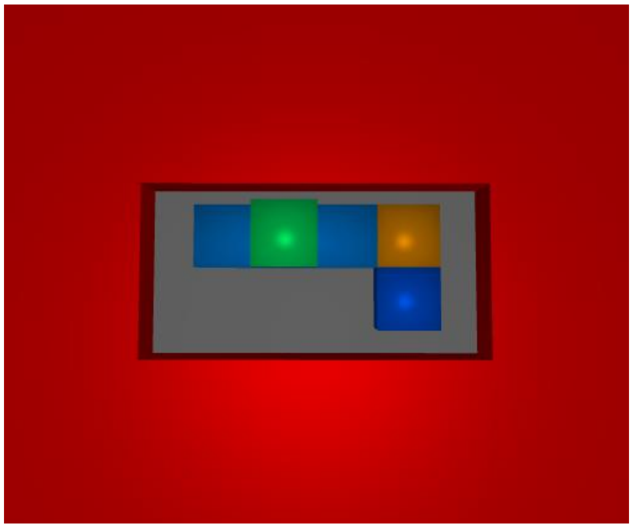
Genetic Testing

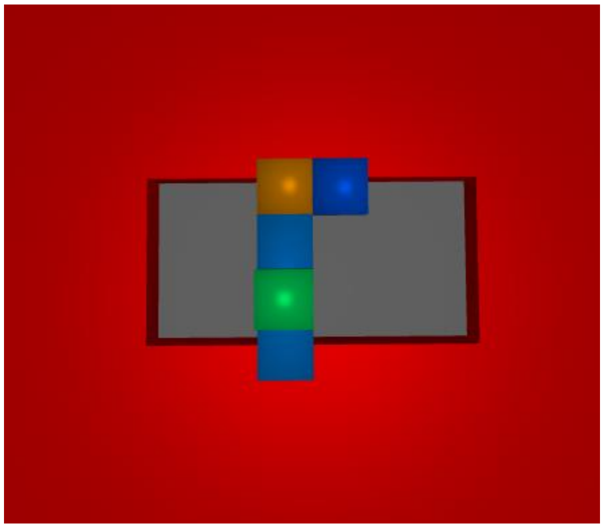
Overall, this method of procedural generation is successful in creating new 3D multi-part models. Given two input keys, the same model will always be produced. Similarly, given four input keys, two parent models will be produced using the first two keys for one model and the last two for the other. In this four-key method, the two models are then used to breed a new combination or offspring model based on a mixture of the parents' keys.

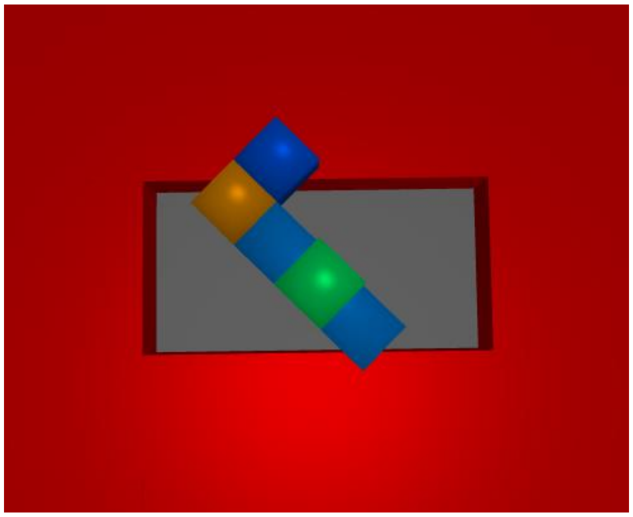
Evaluation

Finally, to evaluate and compare these models, a simple collision test was designed using the Bullet Physics engine for the application of procedurally developing a key model. In this test, a keyhole model is previously prepared and placed in the 3D space with an appropriate collision mesh. If the generated model passes far enough passed the keyhole model, the test is successful. The model is then tested using several orientations and given a score based on how many tests were successful. **Figure H** shows examples of these orientations for one model generation. In these images, the red background is the keyhole model whereas the grey center is the hole for the key to pass through. On the other hand, **Figure I** shows the actual results after running the evaluation iterations using single rotations in steps of 45 degree angles. This table shows failure results only in Y-axis rotations since the generated object is significantly longer in one dimension than the other two. Using an improved evaluation algorithm, testing can be optimized to avoid future tests with similar rotations.

Figure H: Example Evaluation Orientations for Trait Seed: 129, Mutation Master Seed: 39


Test #0, Rotation 0° about –axis, PASS


Test #2, Rotation 90° about Y–axis, FAIL


Test #3 , Rotation 135° about Y–axis, FAIL

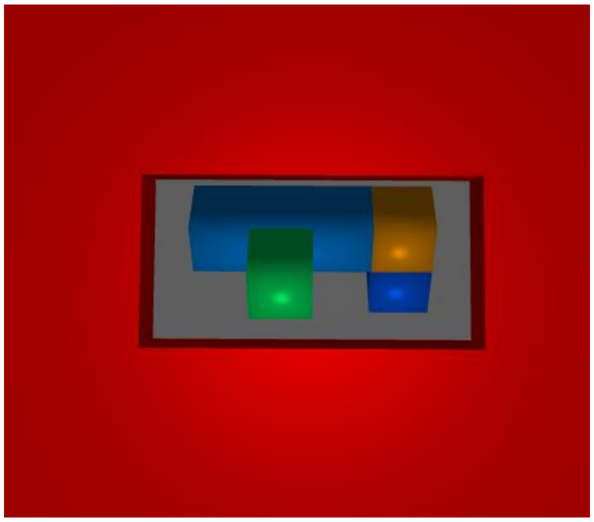

Test #9, Rotation 45° about X–axis, PASS

Figure I: Evaluation Results for Trait Seed: 129, Mutation Master Seed: 39

Test Number	*Pass/Fail*	*Rotation Axis*	*Rotation Angle (degrees)*
0	PASS	Y	0
1	FAIL	Y	45
2	FAIL	Y	90
3	FAIL	Y	135
4	PASS	Y	180
5	FAIL	Y	225
6	FAIL	Y	270
7	FAIL	Y	315
8	PASS	X	0
9	PASS	X	45
10	PASS	X	90
11	PASS	X	135
12	PASS	X	180
13	PASS	X	225
14	PASS	X	270
15	PASS	X	315
16	PASS	Z	0
17	PASS	Z	45
18	PASS	Z	90
19	PASS	Z	135
20	PASS	Z	180
21	PASS	Z	225
22	PASS	Z	270
23	PASS	Z	315

Reflections and Future Work

Generation

In this system, there are two major aspects to be improved: the variety of the procedural generation and the evaluation function. As the implementation stands now, the procedural generation generates much like a single gene and is, in fact, forced to output a mixture of the two parents. By expanding the number of seeds used to generate a unique model, the genetic variance would increase significantly, but the model class/data structure would become increasingly more complex. Better programming practices, like stricter Object-Oriented Programming, is advised for both code clarity and data organization. While C++ provides and available class structure, further organization within the designed classes can improve quality and speed of generations.

Evaluation

Further work on this project will include analysis of this first evaluation to continue testing the models until the best possible orientation is found. In the current implementation, the key models are tested at eight rotations about each axis independently using a binary, pass/fail function. A total of twenty-four current tests are implemented as a simplified collision-fitting algorithm. In these tests, if the model can fall completely through the keyhole model, the test passes and the next test is executed. If the model, instead, stops without passing through the keyhole mode, then the test fails and the remaining tests are executed to determine the best, passing orientations of the key model. This testing process is similar to blindly attempting to insert a new key into a keyhole/lock.

3D Modelling Software Plugin

This implementation uses a previously designed OpenGL program that was created to learn basic OpenGL and general computer graphics concepts. This program has been further modified to introduce random, multi-component models, implement the Bullet Physics engine, and produce new, procedurally-generated genetic models based on model data. While this series of modifications has proved beneficial for learning low-level implementations, using higher-level languages like MEL and Python via Blender and Autodesk Maya could produce similar, if not better results at a faster development rate. Implementing this algorithm within an interface where physics engines and modelling functions are already established and refined would reduce low-level focus and introduce a greater avenue for learning and understanding Maya or Blender plugins.

Physical Calculations

Originally, this project was designed in tandem with the implementation of the Bullet Physics engine to calculate and evaluate the physical properties of the generation models. For example, the colors depicted in the program were designed to indicate the density of the model in the Bullet Physics engine (from 0 BulletMass/volume to 10 BulletMass/volume). These calculations proved difficult to query from the Bullet Physics engine and, instead, were calculated using the research paper written by C. Zhang and T. Chen for extracting physical properties from 2D/3D meshes [2]. These methods proved useful for designing more realistic models, however the evaluation function was simplified to enable the feature of basic genetic generation. Although these methods are no longer necessary for the current implementation of the procedural generation, future work could utilize these numerical calculations in order to determine more physically realistic scenarios.

References / Bibliography

[1] BradyGames. *The Art of Borderlands*. DK Games, 2012.

[2] C. Zhang and T. Chen. *Efficient Feature Extraction in 2D/3D Objects in Mesh Representation* [Online]. Available: http://chenlab.ece.cornell.edu/Publication/Cha/icip01_Cha.pdf

[3] K. Sims, “Evolving Virtual Creatures,” in *Proc. SIGGRAPH 94*, Orlando, FL, 1994, pp. 15-22.