Generating Virtual Guitar Strings Using Scripts

Luka Kunić and Željka Mihajlović
University of Zagreb, Faculty of Electrical Engineering and Computing,
Zagreb, Croatia
luka.kunic@fer.hr, zeljka.mihajlovic@fer.hr

***Abstract*—Artists who create 3D models usually rely on the traditional method of direct mesh manipulation using basic operations such as translation, rotation, scaling, and extrusion. In some cases, creating a model in this manner requires performing many repetitive and precise actions, which makes a fully manual approach suboptimal. This paper explores an alternative concept of 3D modeling using scripts, which aims to automate parts of the modeling process.**

Existing procedural algorithms use scripts to generate objects based on mathematical models (e.g. generating realistic terrains using fractals), or to build complex structures using simple template models (*model synthesis*). Unlike those methods, which rely on stochastic behavior to generate pseudo-randomized shapes, the goal of this method is to write scripts that generate parametric objects which would be either difficult or time-consuming to model by hand. The example described in this paper is a script that generates animated strings for musical instruments. Although the basic shape of a string is quite simple, animating string vibrations and bending can be quite a tedious task, especially because of the various shapes and sizes of string instruments.

I. Introduction

With the advancements in computer graphics, 3D modeling has become a major part of many industries. Artists who create 3D models most commonly use dedicated software solutions that include a 3D viewport, which allows them to directly modify the mesh data using basic operations such as translation, rotation, scaling and extrusion. While this traditional method can be used to produce very good results, those results are often difficult to achieve due to large amounts of manual work required. This is especially true for large-scale environments such as terrains or entire cities used in movie and video game industries, which are practically impossible to generate manually in a reasonable time frame. However, many tasks involved in this process can be entirely or at least partially automated, allowing for a faster and less repetitive workflow.

Automating a task can be as simple as creating a custom tool that performs some commonly used or repetitive actions. This allows users to focus on the creative aspect of modeling instead of being hindered by the lack of software features. On the other hand, scripts can be used to generate complex objects based on a set of parameters and constraints, which can be given by the user or determined by the choice of the algorithm being used. Obviously, these scripts must be tailored to each specific problem, which brings into question whether the gains of using a script justify the time spent on development. However, if a task can be generalized or parametrized, writing a script to perform that task can greatly reduce time spent working on different areas of the model. Furthermore, variations of algorithms developed for such generic tasks can be used for solving various modeling problems in different domains.

This paper describes a method for automatically generating and animating 3D models of musical strings, which will be used for creating interactive virtual string instruments. Existing procedural modeling techniques, briefly covered in Section II, mostly rely on some form of stochastic behavior in order to achieve the final shape of the generated model. We present a more deterministic approach, in a sense that running the algorithm repeatedly with the same set of parameters will always result in the same model. The primary gain of this method is not in the model creation, since the actual mesh is quite simple, but rather in automatically adding complex behavior to the model through morphing and animations. Another significant advantage over a manual process is the ability to easily generate arrays of objects using various parameters. This is quite important for rapid prototyping, where the user iteratively creates variations of the model until they are satisfied with the result. Further discussion about the motivation for this method can be found in Section III.

The string generation process, described in detail in Section IV, includes: creating the string mesh based on a set of input parameters, adding vertex animations to the mesh in order to simulate string bending, adding and attaching an armature to the mesh, and animating the string vibrations. The results are presented in Section V.

II. Related work

The following subsections present several valuable methods for automatically generating 3D content: procedural modeling, template-based methods and interactive simulations.

A. Procedural modeling

Procedural techniques are algorithms that are used to determine the characteristics of an object or effect [3]. An algorithm is implemented as a procedure which contains an abstract representation of the desired features for the output model. Executing the procedure calculates the features based on input parameters and constraints set by the user and generates a finished model which satisfies those constraints. This allows a high level of control and flexibility. The procedures often incorporate a form of stochastic behavior in order to introduce a level of randomization into the output models. This is useful when modeling objects which are represented in nature, so that the finished product appears more organic and realistic.

Procedural modeling techniques have been used in computer graphics since the early beginnings of the field. At first, procedures were used to generate images and textures resembling materials found in nature, as it was found that many natural shapes follow distinct patterns which could be described by mathematical models. A prominent example of such textures are fractals, introduced by Mandelbrot in 1982 [7]. Fractals can be used to create very natural looking objects and structures

because of their self-similarity property, which is why they are widely used in computer graphics.

A different approach to procedural modeling uses formal languages and grammars to describe the output model. One such model was introduced by Lindenmayer who used his L-systems [6] to formally describe the growth of plants. The system consists of an alphabet of symbols and set of production rules, which are used to generate strings of symbols that define a plant. This technique is often used in computer graphics because of its high versatility, and because it can be easily extended with additional functionality [8], [13], [14].

Procedural techniques are also commonly used to create and animate natural volumetric phenomena that cannot realistically be modeled with mesh geometry, such as gasses, clouds or fire [3], [5]. Attributes like color and transparency can be controlled using procedures that simulate air turbulence and noise in the space occupied by the model volume. Similar techniques can also be applied to particle systems which are also guided algorithmically and can be affected by some external influence.

B. Model synthesis and template-based modeling

Certain objects cannot entirely be described using mathematical models. This applies to most man-made structures such as buildings or machines that are often found in 3D environments. For example, when creating a large virtual city, it is important to include enough diversity so that the city does not seem artificial, but at the same time most buildings should have similar features so that they visually fit together. One of the solutions would be template-based modeling, a set of techniques that take simple objects as input and use the objects' distinct features to generate various other objects that incorporate those features.

One such algorithm named *model synthesis* is presented by Merell and Manocha [10]. Their algorithm takes an object and uses it as a template to create complex structures. For a given point in space for the model being generated, their algorithm looks at the surrounding area of the point and calculates the possible geometry samples which can be generated at that point. Their algorithm also takes into account various constraints: predetermined dimensions of an object, ratios between dimensions, connectivity constraints, and the general macroscopic shape and scale of the output model. Furthermore, this algorithm can take multiple objects as input, which results in output models with a high degree of variety.

A template-based approach has also been used by Zhou et al. for generating terrain using their *terrain synthesis* algorithm [18]. As input for their algorithm, they used height maps of real-world terrains and mountain ranges, combined with user-made sketches. The algorithm would identify important features from the terrain height map and the sketch. It would use the sketch to determine the general shape for the output model and would apply the extracted terrain features from the height map to that general shape. The result, as shown in Fig. 1, would be a terrain that resembles the real terrain used for the height map, but has the shape of the user-made sketch.

C. Interactive simulations

Simulating how virtual objects would behave in the physical world is a common task. Typical examples include cloth simulations [1], [17], simulating collisions between objects, simulating fluids [12] and force fields, determining how objects interact with natural forces like wind, etc. These types of simulations are often used for animating objects in video games and other virtual environments. Simulations used for animating the objects typically have many parameters and obey certain laws of physics which require performing complex calculations. These calculations can either be done in real-time, which often sacrifices accuracy for computation performance, or they can be precomputed, in which case the calculated simulation will not be able to adapt to the surrounding objects in the virtual environment during execution.

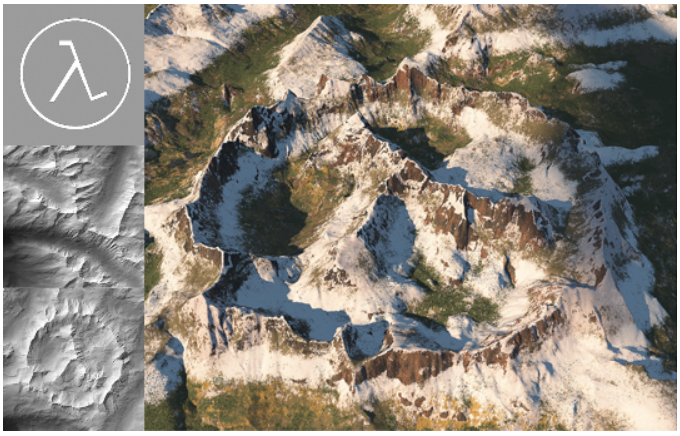

Fig. 1. The result of the terrain synthesis algorithm. On the right is the finished terrain which was generated based on the height map of Mount Jackson in Colorado, USA (center left) and a user defined sketch (upper left). The combined height map generated by the algorithm is shown in the lower left corner.

III. Motivation

Throughout history, music has been a prevalent source of entertainment in our society. Professional musicians have always been praised for their high technical abilities which they had developed over years of dedication and daily practice. Some of those musicians were also engineers, so they started using their expertise in both areas to create robotic instruments like self-playing pianos, guitars, and other instruments [2], [15], [16]. In the modern era of technology, with continuous advancements in the field of computer graphics, it is natural to explore the possibilities of placing musical instruments into virtual environments.

In order to create a self-playing virtual instrument, the 3D model of the instrument must first be created and animated. In the case of a string instrument like a guitar, this includes modeling the body of the instrument, creating and animating the strings, and adding fretting and picking mechanisms which will be used to press the strings onto the fretboard and to pick the strings as notes are played.

The most challenging part of this process is modeling and animating the strings. This task can be approached in several ways. The animations could be done manually, but that would require an immense amount of work so it is better to find an automated alternative. Because the strings obey certain laws of physics while vibrating, their behavior can be simulated at runtime. This approach offers most flexibility and most realistic behavior, but it can be computationally demanding, which might decrease runtime performance. A good compromise would be to precompute a set of animations and store them, so that they can be played back as required. This preserves the realism gained by the simulation without

sacrificing performance. The main idea is to write a script that will generate the mesh of the string, attach an armature to the mesh and automatically compute the animation keyframes and morphing data that will be used in the final animations.

Reusability and the drastic reduction of time required for modeling are the primary advantages of using scripts as part of the modeling process. The main aspect of reusability is the use of input parameters in order to define the characteristics of the generated model. In the case of a guitar string, parameters are used to define string length, diameter, mesh complexity (number of vertices), and the number of frets.

IV. Generating Musical Strings

This section describes the *StringGenerator* script. The script was developed as a Blender add-on using Blender's Python scripting API.

A. Blender scripting API

Blender is a popular open-source 3D software suite that provides tools for the entire 3D model creation pipeline – modeling, texturing, animation, rendering, compositing and sequence editing. Due to its open-source nature and powerful Python scripting API [19], it is continually being improved by members of the community, both by revising features of the core system and by developing add-ons which add new functionality.

Blender has an integrated Python console that allows users to write and execute scripts directly in the viewport. This is useful for testing and debugging, but also for writing short commands that can easily accelerate the workflow. The API provides references to all objects in the scene, as well as functions for performing operations on those objects. This can be used to perform the same operation on multiple objects in the scene at once, instead of repeating the operation manually for each object. Python scripting in Blender can also be used for generating and modifying meshes, adding textures and animations, implementing new tools and operations, as well as automating tasks involved in other steps of the model creation pipeline, such as rendering or compositing.

Add-ons in Blender are packaged pieces of Python code that can be loaded into Blender in order to seamlessly include additional functionality. Of course, while users can achieve the same functionality by executing the script directly from the integrated console, add-ons provide the convenience of having the functions easily accessible from the UI menus, which greatly increases usability. Also, they can easily be distributed and shared among users, which facilitates collaboration.

B. Planning and design decisions

The model guitar strings should be animated, which refers to animating string vibrations when a string is plucked, and bending the string towards a fret depending on where it is pressed. It is immediately clear how difficult it would be to create these animations by hand because of the complexity of realistic string vibration and the number of frets on the guitar, so the only viable alternative would be to automate these tasks. This will be done by implementing a script which (1) generates a string based on given parameters such as length and diameter, (2) creates shape keys[1] which are used for bending the string, and (3) animates the string vibration by calculating vertex positions for each keyframe. The script will be packaged as a Blender add-on.

[1]Shape keys are used to deform the mesh into a new shape without the need for an armature. A shape key saves vertex positions for a deformed mesh and allows interpolation between the initial vertex positions and the deformation.

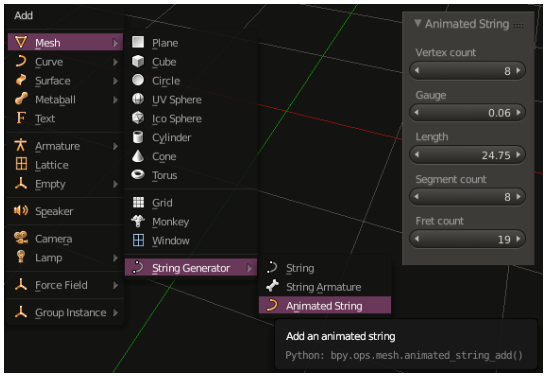

Fig. 2. The custom operation for generating animated strings incorporated into the Blender's *Add mesh* menu. Clicking the menu item calls the custom *animated_string_add* function which generates and animates a string with the given parameters.

The fretboard for the guitar can also be generated using a script because exact fret positions depend on string length and are determined using a mathematical formula. The same formula is already used for calculating the string bending shape keys, so a part of the code can be reused.

C. Generating the string mesh

The *StringGenerator* add-on provides a new tool for generating models of strings for musical instruments. Each generated string consists of two components: the string mesh that defines the visible geometry of the string, and the armature which is used for animating string vibration.

As shown in Fig. 2, the tool allows users to modify several properties of the generated string: length, gauge, fret count, vertex count and segment count. String length and gauge are both given in inches, and fret count is used for generating the shape keys required for bending the string. If the fret count is set to zero, only string vibration is animated and no shape keys are created. Vertex count determines the number of vertices in the cross-section of the string and segment count determines the number of segments along the length of the string. These two properties are used to control the complexity of the model. Higher values give better quality and smoother animation which is suited for high-detail offline rendering, but cause slower rendering times due to the increased polygon count. On the other hand, lower values give fast rendering times required for real-time execution, at the expense of reduced quality. However, the reduction in mesh quality is not very noticable, even when the model is being viewed from close distance, because the string shape itself is very simple and proper shading can provide the necessary detail required for a more realistic appearance.

The mesh for the string is generated in three phases. The first phase creates string segments based on the given parameters. Each segment is a closed loop of vertices connected to the neighboring segments by 4-sided faces, forming a cylinder. The segment count and string length properties are used to determine the distance between neighboring segments. The second phase is used to create vertex groups for each of the segments. Vertex groups are simple collections of vertex indices that will be used to easily map vertices to their corresponding bones in the armature.

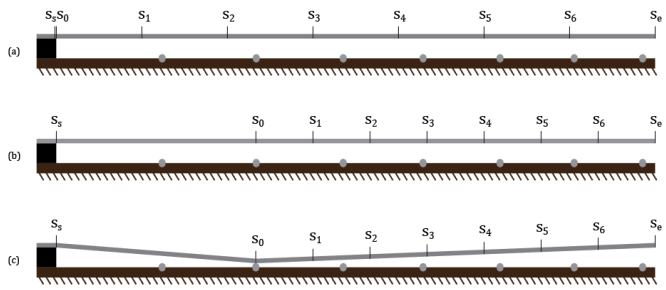

Fig. 3. Repositioning the segments s_i using shape keys. Figure (a) shows the initial setup where the segments are equally distributed along the length of the string. When a fret needs to be pressed, the segments are translated along the string so that s_0 lines up with the fret (b), and when the string is pressed the segments are translated towards the fretboard (c).

The third phase of mesh creation is creating the shape keys. The goal is to have a robust system for animating string bending on each fret. In order to achieve this goal, shape keys are used to reposition the segments based on the fret at which the string is bent. The idea is illustrated in Fig. 3. Segments s_s and s_e indicate the fixed endpoints of the string and will not be affected by the shape keys. The other segments will be used for animating the string bending and vibration and therefore need to be repositioned for each fret. A shape key for each fret translates the segments along the length of the string so that the initial segment s_0 lines up with the corresponding fret. All other segments are equally distributed between s_0 and s_e. Bending the string towards a fret is controlled by one additional shape key which can be used in combination with any of the other shape keys. It bends the string by placing s_0 directly on top of the fret and distributing the other segments to form a straight line from s_0 to s_e.

D. Determining fret positions

A guitar string produces a note of a certain frequency depending on the length of the string and the string tension. Tension is used to set the root note of a string, which is the note produced by playing an open string without pressing down on any fret. This means that the only way to play a different note on the same string is to shorten the string. The chromatic scale consists of 12 notes, so the string pressed down on the 12th fret should produce the note that is an octave higher than the root, which means that the string should be shortened to half of its original length. Similar logic applies to the positions of other frets. The exact position for the fret i is given by

$$d_i = l - \frac{l}{2^{i/12}} \tag{1}$$

where d_i is the distance from the nut to the fret and l is the length of the string.

This calculation can also be used to generate the fret models. The script uses a manually modeled fret as template for generating the entire fretboard. It calculates the position for each fret in the fretboard, duplicates the template and places it in the calculated position. An additional consideration when generating frets is the neck width. In order to compensate for the wider frets towards the head of the guitar, the neck becomes narrower as it goes from the guitar body towards the head in order to improve playability. This means that the frets also need to be scaled to the width of the neck at the calculated position. The fretboard with the generated frets is shown in Fig. 4.

Fig. 4. The finished fretboard for the model. A script was used to calculate the fret positions and to scale the frets to match the width of the neck.

E. Animating the string

The second component created by the add-on is the armature used for animating the string. The armature consists of a set of bones, each of which is used for controlling a group of vertices. A bone is created for each segment of the mesh, and the vertex group containing the vertices of each segment is attached to the corresponding bone.

String vibration is caused by a stationary wave traversing the string, which produces a sound based on the frequency of the wave. In order to simulate a realistic vibration, the wave needs to be broken down into its base components, which are called harmonics. Each harmonic of the wave can be represented by

$$y = A \sin(\omega x) \tag{2}$$

where A is the amplitude of the wave, ω is the radial frequency and x is a longitudinal position on the string. String vibration for a single harmonic can be simulated by varying the amplitude over time, using a sine function to control the oscillation

$$y(t) = A \sin(k\pi t) \sin(\omega x) \tag{3}$$

where k determines the oscillation frequency. The harmonics can then be summed up to get a more realistic result. Lastly, dampening needs to be added so that the vibration fades over time. This is done using the δ factor. The resulting equation is given by

$$Y(t) = \sum_{i=1}^{N} y_i(t) = A \sum_{i=1}^{N} \sin(k_i \pi t) \sin(\omega_i x) \tag{4}$$

$$Y_d(t) = Y(t) e^{-t/\delta} \tag{5}$$

Frame rate is set to 60 frames per second to allow a more detailed simulation. Two harmonics are used to calculate the positions for each bone over 5 seconds (300 frames). Higher degree harmonics are not used for this simulation because the frame rate would need to be much higher for them to make a significant impact on the result. Also, for simplicity, string vibration is only animated along a single axis, which produces very good results despite not being entirely realistic[2]. The final position of each bone at the time t is given by

$$y(x,t) = A(\sin\frac{t\pi}{4}\sin\frac{x}{2} + \frac{1}{4}\sin\frac{t\pi}{2}\sin x)e^{-t/\delta} \tag{6}$$

[2]A musitian who plays the guitar would never pick the string perfectly vertically, so the string would also receive a horizontal vibration component.

The second harmonic was multiplied by the factor $1/4$ in order to reduce its maximum amplitude, resulting in a more believable animation.

Two animations are created, one for the downstroke and one for the upstroke when picking the string, and both are added as animation sequences to the armature. One animation is given directly by the equation (6) and the other contains the opposite movement, so the equation result is merely negated. These animations can be used in combination with the shape keys do simulate vibration when the string is pressed to a fret. This is enabled by the fact that bones only control the vertical movement of segments and shape keys translate the segments along the length of the string.

F. Texturing the strings

Each of the strings was UV-unwrapped and assigned a simple striped texture in order to mimic the look of wound[3] strings. While the texture contributes to the realistic appearance of the string, an issue arises when using the shape keys to simulate bending of the textured string. Because the base mesh of the string was UV-unwrapped, each vertex was assigned a fixed (u, v) coordinate of the texture. When the mesh is deformed using the shape keys, the vertices move in 3D space, but not in texture space. This causes stretching in the texture, which is of course unrealistic.

This issue can be solved by splitting the mesh material into three slots and assigning the same texture to each of these slots. One slot would contain all static vertices which form the string endings that were added manually. The second slot would contain the the cylinder made by the first two segments of the string (s_s and s_0 in Fig. 3), and the final slot would contain all faces of the vibrating section of the string (s_0 to s_e in Fig. 3). Splitting the material in this manner allows the texture to be scaled individually for each of those sections, in turn enabling the use of a texture even when the string mesh is deformed by shape keys. When a shape key is used to press a fret, the texture of the second material would be scaled down to increase the level of detal, while the third material would have its texture scaled up. This would preserve the overall textured appearance of the string mesh and eliminate undesirable stretching.

V. Results

Fig. 5 shows the finished model of a 5-string bass guitar. The strings were generated using the *StringGenerator* add-on, and the frets were placed using the script described in the previous section. The remaining components of the model were mostly created manually in Blender.

The parameter values used for generating the strings were based on actual values for common bass strings. For this model, the string diameters were set to values ranging from .040 to .130 inches, and the string length was set to 34 inches. Since the script only generates the vibrating portion of the string, the missing geometry on both ends – the start of the string at the bridge of the guitar, and the string ending which wraps around the tuning post – were modeled manually for each string and appended to the generated models. This had no effect on the animation data because the generated geometry had not been modified.

Fig. 5. The finished bass guitar model.

The fretboard was generated using a manually modeled fret template, which was duplicated and placed using the previously described script. The string length was used to determine the exact fret positions, and each fret was scaled vertically to match the neck width at the calculated position.

VI. Future work

As stated in Section III, the main motivation behind making this model was building self-playing virtual musical instruments. The idea is to create a 3D model of an instrument, add specific mechanisms that would be used to visualize playing the instrument, such as picking and fretting fingers for string instruments, and finally implement the required functionality for playing a given set of notes.

The process described in this paper can easily be used to create various models of string instruments, but in order for those instruments to become 'self-playing', they need to be programmed. This task can be accomplished using one of the popular game development engines such as *Unreal Engine* [21] or *Unity* [20]. These powerful tools are primarily designed for creating 3D and 2D games, but they can also be used for developing quite complex, realistic, and interactive visualizations.

As a way to feed note data into the self-playing instrument visualization, one could define a custom data stream that would be interpreted and used to play the required animations corresponding to each given note. This method would allow fast playback and high flexibility for defining the exact way each note should be played. However, building an entire song in that manner would be quite time-consuming, mainly because one would need to manually define how to play each note. For a string instrument, this would mean defining which left-hand finger presses which string on which fret, and which right-hand finger picks the string. An alternative method would be to use a common data source like a MIDI file to provide the note data, and then perform an optimization task over the entire song in order to determine the optimal fingering and picking sequences to be used during playback.

Fig. 6 shows the bass model imported into Unreal Engine. The right-hand picking mechanism consists of five mechanical fingers whose behavior has been programmed using a state machine. The current state of each finger is used to determine which finger will be used to pick the next note in the sequence, with the aim to reduce the cumulative movement of all fingers. The left-hand fretting mechanism could have been modeled as a human hand [4], but the note positioning optimization

[3]Wound strings consist of a round wire wrapped in a tight spiral around either a round or hexagonal core. This type of construction allows the string to produce a much lower pitch than regular plain strings, which allows them to be much thinner and easier to play.

Fig. 6. The bass model with picking and fretting mechanisms imported into Unreal Engine. The strings are colored red and blue in order to help visualize the material change when a string is bent to a fret using a shape key.

algorithm would have to be more complex and require many more contraints related to physical limitations of a human hand. That is why the left hand was implemented as four sliding fingers, each of which has five moving pins – one for each string. The fingers slide across the fretboard and their pins are used to press the strings, bending them at the fret where the finger is currently positioned.

VII. Conclusion

The introduction of this paper mentioned several interesting techniques that make use of scripts to generate virtual objects. It was shown how procedural techniques can be good for generating complex, organic and natural-looking geometry, which is not easy to achieve by manual modeling. These techiques served as inspiration for using scripts to generate parametrized 3D models of guitar strings. A generated string includes shape keys used for deforming the mesh when bending the string, as well as an armature with vibration animations. Using a script for this task provides a fast and repeatable method which can be used for modeling various musical string instruments.

It was shown how this method can be used to easily create strings for a 3D model of a bass guitar, and an idea has been presented on how that model can be used to implement a self-playing instrument visualization. These types of visualizations can be quite useful for educational purposes, especially if the fretting and picking mechanisms can be modeled as actual human hands performing the movements. This would open doors for further investigation regarding visualization of optimized fretting sequences, and automatic creation of human-playable tablature from music sheets or structured audio formats such as MIDI files.

References

[1] D. Baraff, A. Witkin, *Large steps in cloth simulation*, ACM SIGGRAPH, 1998.

[2] R. B. Dannenberg, B. Brown, G. Zeglin, R. Lupish, *McBlare: A Robotic Bagpipe Player*, Proceedings of the International Conference on New Interfaces for Musical Expression, 2005.

[3] D.-S. Elbert, F.-K. Musgrave, D. Peachey, K. Perlin, S. Worley, *Texturing & Modeling: A Procedural Approach*, 3rd ed., Morgan Kaufmann Publishers, 2002.

[4] G. ElKoura, K. Singh, *Handrix: Animating the Human Hand*, Eurographics/SIGGRAPH Symposium on Computer Animation, 2003.

[5] J. Kniss, S. Premoze, C. Hansen, D. Ebert, *Interactive Translucent Volume Rendering and Procedural Modeling*, IEEE Visualization, 2002.

[6] A. Lindenmayer, *Mathematical Models for Cellular Interaction in Development*, Journal of Theoretical Biology, 1968.

[7] B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman, 1982.

[8] R. Mech, P. Prusinkiewich, *Visual Models of Plants Interacting with Their Environment*, ACM SIGGRAPH, 1996.

[9] P. Merrell, *Example-Based Model Synthesis*, Symposium on Interactive 3D Graphics (i3D), 2007.

[10] P. Merrell, D. Manocha, *Model Synthesis: A General Procedural Modeling Algorithm*, IEEE Transactions on Visualization and Computer Graphics, 2010.

[11] F. K. Musgrave, C. E. Kolb, R. S. Mace, *The Synthesis and Rendering of Eroded Fractal Terrains*, ACM SIGGRAPH, 1989.

[12] M. Müller, D. Charypar, M. Gross, *Particle-Based Fluid Simulation For Interactive Applications*, ACM SIGGRAPH 2003.

[13] P. Müller, P. Wonka, S. Haegler, A. Ulmer, L. Van Gool, *Procedural Modeling of Buildings*, ACM SIGGRAPH, 2006.

[14] Y. Parish, P. Müller, *Procedural Modeling of Cities*, ACM SIGGRAPH, 2001.

[15] F. A. Saunders, *The Mechanical Action of Violins*, Journal of Acoustic Society of America, 1937.

[16] E. Singer, K. Larke, D. Bianciardi, *LEMUR GuitarBot: MIDI Robotic String Instrument*, Proceedings of the International Conference on New Interfaces for Musical Expression, 2003.

[17] P. Volino, N. Magnenat Thalmann, *Implementing Fast Cloth Simulation With Collision Response*, Computer Graphics International, 2000.

[18] H. Zhou, J. Sun, G. Turk, J. Rehg, *Terrain Synthesis from Digital Elevation Models*, IEEE Transactions on Visualization and Computer Graphics, 2007.

[19] Blender Foundation, *API documentation*, built October 2nd, 2016, https://www.blender.org/apiblender_python_api_2_78_release, February 12th, 2017.

[20] Unity, *About the Unity Editor*, https://unity3d.com/unity/editor, February 12th, 2017.

[21] Epic Games, *What is Unreal Engine 4*, https://www.unrealengine.com/what-is-unreal-engine-4, February 12th, 2017.