REAL-TIME FORCE-BASED SOUND SYNTHESIS USING GPU PARALLEL COMPUTING

Ryoho Kobayashi

Faculty of Environment and Information Studies,
Keio University SFC
Kanagawa, Japan
ryoho@sfc.keio.ac.jp

ABSTRACT

In this paper we propose a real-time sound synthesis method using a force-based algorithm to control sinusoidal partials. This synthesis method can generate various sounds from musical tones and noises with three kinds of intuitive parameters, which are attractive force, repulsive force and resistance. However, the implementation of this method in real-time has difficulties due to a large volume of calculations for manipulating thousands or more partials. In order to resolve these difficulties, we utilize a GPU-based parallel computing technology and precalculations. Since GPUs allowed us to implement powerful simultaneous parallel processing, this synthesis method is made more efficient by using GPUs. Furthermore, by using familiar musical features, which include MIDI input for playing the synthesizer and ADSR envelope generators for time-varying parameters, an intuitive controller for this synthesis method is accomplished.

1. INTRODUCTION

A force-based sound synthesis [1] is an application of sinusoidal partial editing techniques. This synthesis method can generate various sounds from musical tones and noises with three kinds of intuitive parameters, which are attractive forces to a reference spectrum, repulsive forces between partials, and resistances for decreasing the speeds of the partials. However, this method requires a large volume of calculations due to a need of some thousands or more partials to control.

By recent developments of graphics processing units (GPUs) [2] and APIs for handling these processors, an implementation of an efficient simultaneous parallel processing has been realized. GPUs are originally developed for computer graphics and visual image processing, however, they have begun to be used for general purpose due to a popularization of general-purpose computing on graphics processing units (GPGPU) [3]. And they recently have been utilized for sound processing and syntheses [4, 5, 6]. A GPGPU framework is considered of value for a acceleration of force-based sound synthesis, because this synthesis method needs many simple calculations simultaneously.

In this paper we propose a process to accomplish a real-time sound synthesis using a force-based algorithm by utilizing a GPU-based parallel computing technique. All programs presented in this paper are written in Swift [7] and use Metal API [8] for GPU processing.

2. FORCE-BASED SOUND SYNTHESIS

The sound synthesis method presented in this paper uses a force-based algorithm, which is commonly known as a graph drawing algorithm [9, 10]. The fundamental process for the synthesis is described in this section.

2.1. Analysis of a Reference Sound Source

The first step is the analysis of a reference sound source. The Short-Time Fourier Transform (STFT) analysis [11] is used for this step, where amplitudes $A(k)$ for each frequency $F(k)$ are detected by

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n / N} \tag{1}$$

$$A(k) = |X(k)| \tag{2}$$

$$F(k) = \frac{kR}{N} \tag{3}$$

where $x(n)$ consists of N samples of a windowed waveform and R represents the sampling rate.

2.2. Distribution of Partials

The synthesis phase for the proposed method begins by generating partials in a specific range of frequencies. The amplitudes of the partials are unchangeable. The user specifies the maximum number of partials and the number of active partials is determined in proportion to the amplitude of the reference sound as (4).

$$\nu = \nu_{max} \sum_{k=0}^{N-1} \alpha A(k) \tag{4}$$

In this equation, α represents a constant number for scaling the amplitude. The active or inactive partials are randomly chosen. In this step, since the frequencies of the partials are random, an unpitched sound is typically created.

2.3. Attractive Force

Attractive forces, which are applied to the partials, are generated from the spectrum detected from the reference sound. A partial is attracted to neighboring frequency components. The force is inversely proportional to the square of the distance between the target frequency component and the frequency of the partial.

$$f_a(P(i)) = \sum_{0<|F(k)-P_F(i)|<\tau} \frac{\mathrm{sgn}(F(k) - P_F(i)) g_a A(k)}{|F(k) - P_F(i)|^2} \tag{5}$$

$f_a(P(i))$ represents an attractive force for partial $P(i)$ of which the frequency is $P_F(i)$, g_a is a constant value to adjust the strength of the force, and τ corresponds to the range of the effective frequency components.

2.4. Repulsive Force

To avoid congestion of partials at a small peak in the spectrum, repulsive forces are generated between every pair of partials. The force is inversely proportional to the square of the distance between the partials.

$$f_r(P(i)) = \sum_{P_F(j) \neq P_F(i)} \frac{\mathrm{sgn}(P_F(i) - P_F(j)) g_r}{|P_F(i) - P_F(j)|^2} \tag{6}$$

$f_r(P(i))$ represents a repulsive force for partial $P(i)$. By using all pairs of partials for the calculation, partials depart from condensations.

2.5. Resistance

When the reference sound has a static frequency component, the partials have the risk of periodic vibration around a spectral peak. This is because the attractive forces convert back and forth between potential and kinetic energy. Therefore, the oscillations are inhibited by implementing resistance.

$$f(P(n,i)) = f_a(P(n,i)) + f_r(P(n,i)) \tag{7}$$

$$v(P(n,i)) = r(v(P(n-1,i)) + f(P(n-1,i))) \tag{8}$$

In this equation, $v(P(n,i))$ represents a current speed for partial $P(i)$, n is the current time frame, and r is a resistance value between 0 and 1.

2.6. Synthesis

The forces, which are derived above, are applied to partials at every frame by addition of the forces.

$$P_F(n,i) = P_F(n-1,i) + v(P(n-1,i)) \tag{9}$$

The sound synthesis is accomplished using a common oscillator bank synthesis technique [12, 13] which is realized by

$$y(t) = \sum_{\forall i} A \cos[2\pi P_F(n,i) t + \phi_i] \tag{10}$$

where A represents a constant amplitude for each partial, and ϕ_i is an initial phase.

3. GPU-BASED PARALLEL COMPUTING

3.1. General-purpose computing on graphics processing units

General-purpose computing on graphics processing units (GPGPU) refers to the use of GPUs for general purpose parallel computing, and performs computation in applications traditionally handled by the central processing unit (CPU), outside of computer graphics and image processing. GPU has ability to process multiple tasks simultaneously by having thousands of cores. OpenCL [14] and NVidia's CUDA [15] are two popular frameworks to implement general purpose computations on GPUs. The synthesis method this paper proposes uses Apple's Metal API [8] for low-overhead access to the GPUs.

3.2. Metal API

Metal [8] is a graphics application programming interface (API), which allows low-level and low-overhead access to GPUs. Metal is developed and provided by Apple, and is available to use on iOS and Mac OS X. The previous implementation of this synthesis method was written in Objective-C and ran on Mac OS X. The real-time version, which is proposed in this paper, is rewritten in Swift, and is still implemented for Mac OS, thus, Apple's Metal API is expected to deliver high performance and has good prospects for the future. Furthermore, further possibilities of implementation for mobile devices are expected, because Metal is compatible with iOS.

By using Metal framework, a low-overhead interface, a memory and resource management, integrated support for both graphics and compute operations, and precompiled shaders are provided. For the force-based sound synthesis, the handling of the current frequency, the resistance for speed, and the attractive/repulsive forces are required for acceleration to manipulate each partial component. Therefore, the current frequency and speed need to be stored until the calculations for the next time frame are performed.

4. IMPLEMENTATIONS

We present implementations of GPU-based parallel computing technologies and conceptions for the efficiencies of calculations for force-based sound synthesis.

4.1. Attractive and Repulsive Forces

The attractive force for a partial is proportional to the amplitude for a target frequency component, and inversely proportional to the square of the distance between the target frequency component and the frequency of the partial. The repulsive force is inversely proportional to the square of the distance between the partials.

These two forces are possible to calculate by using the convolutions of the functions below with the reference spectrum for the attractive forces and the spectrum of current partials for the repulsive forces.

$$h_a(F) = \frac{-\mathrm{sgn}(F) g_a}{F^2} \tag{11}$$

F is a frequency, g_a is a value to adjust the strength of the force. and $h_a(F)$ is a calculated function for attractive forces.

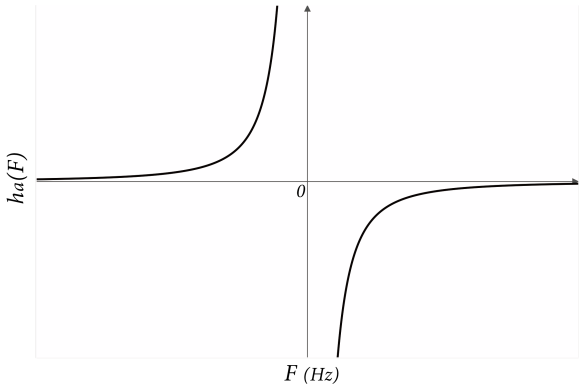

Figure 1: *Function for calculating attractive forces.*

$$h_r(F) = \frac{\mathrm{sgn}(F) g_r}{F^2} \tag{12}$$

F is a frequency, g_r is a value to adjust the strength of forces. and $h_r(F)$ is a calculated function for repulsive forces.

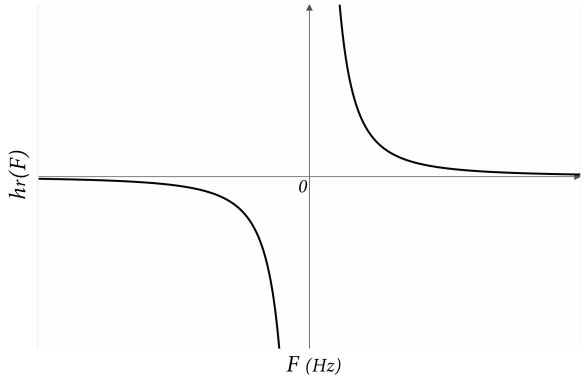

Figure 2: *Function for calculating repulsive forces.*

The summation of these forces are described the following equation 13.

$$\alpha(F) = h_a(F) * A_d(F) + h_r(F) * A_c(F) \tag{13}$$

where $\alpha(F)$ is a summation of the attractive and repulsive forces, $A_d(F)$ is an amplitude for a frequency F from a reference source, and $A_c(F)$ is an amplitude from current partials.

4.2. Updating of Frequencies of Partials

The speed of the frequency change $v(P(n,i))$ and the frequency of a partial $P_F(n,i)$ for the newer frame are calculated by the equasion 8 and 9 in section 2. Since these calculations are achieved by using simple summations and multiplications, parallel computing is applied without difficulties.

5. MIDI INPUT

To use the force-based synthesis method for a musical performance, MIDI note messages [16] are available to input. The fundamental frequency for a reference source spectrum is calculated from the "note number", and the maximum number of partials is proportional to the "velocity".

5.1. Preset Reference Spectra

Efficient computations are accomplished by using GPU-based parallel computing presented in previous sections, however, the volume of calculations is still large. Therefore, a preparation of reference spectral sources and precalculations of the attractive forces are valuable for the achievements of a stable real-time synthesis environment.

The force-based synthesis is possible to generate a variety of timbres by controlling the combination of attractive and repulsive forces and a resistance, thus, simple harmonic series are useful enough. By preparing the reference source, the attractive forces $h_a(F) * A_d(F)$ in equation 13 are also available to prepare.

5.2. ADSR Envelopes

This synthesis method can generate various sounds by adjusting the parameters. In particular, the coefficients for attractive force g_a in the equation 5, repulsive force g_r in the equation 6, and resistance r in the equation 8 are important for controlling the similarity to the reference sound and the quickness of transitions.

By implementing time-varying controls for these parameters, users can dynamically synthesize various sounds. The synthesis method allows users to assign attack time, decay time, sustain value, and release time (ADSR) functions, and minimum and maximum values to the three parameters. The ADSR envelope generator model is generally used in existing synthesizers through the ages, and familiar to musicians. These dynamic controls are simultaneously activated with MIDI note messages and adjusted with MIDI control messages.

5.3. Synthesized Result

In this section, we present a synthesized result of this method.

A spectrogram of a reference sound source for this example is made from sawtooth waves as shown in Figure 4. In this source sound, a fundamental pitch increase, and every overtones are contained.

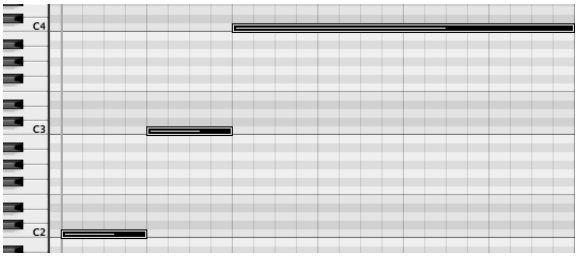

Figure 3: *Input MIDI notes.*

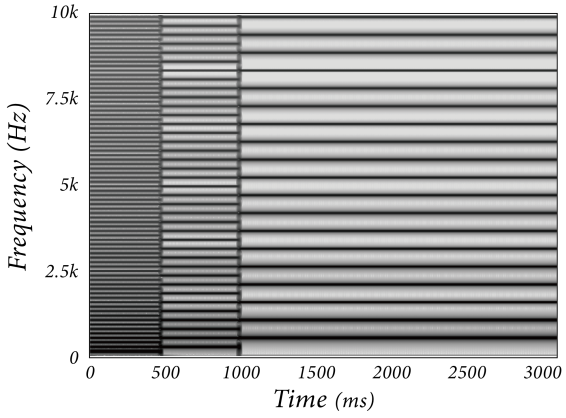

Figure 4: *Reference sound source (sawtooth wave).*

By applying ADSR envelopes in Figure 5 for 5000 partials, a result as shown in Figure 6 is generated. This result shows that a strong repulsive force at an attack makes noise and the partials are gradually attracted to the reference source. At the first note,

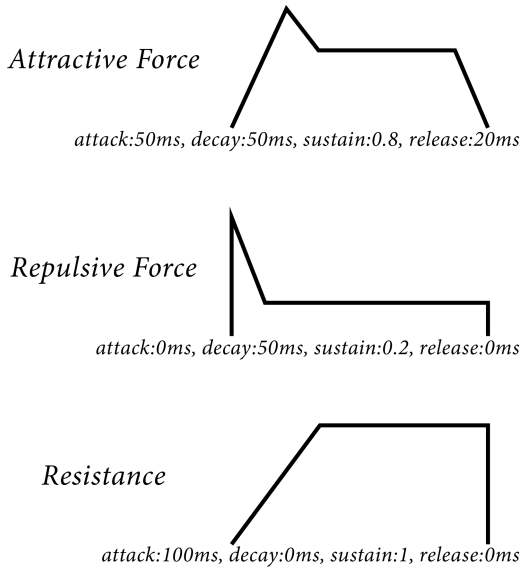

Figure 5: *ADSR envelopes.*

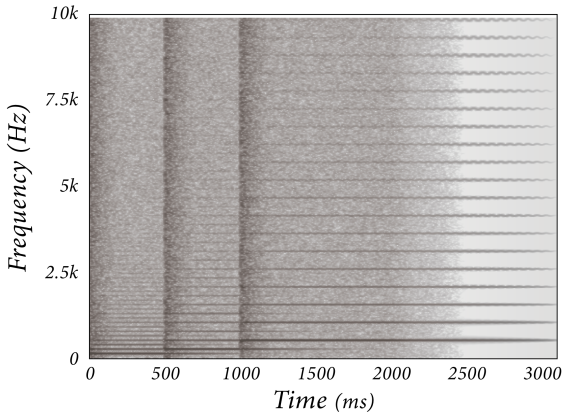

Figure 6: *Synthesized result.*

randomly distributed partials cannot converge on reference overtones during the short duration of 500ms. Since the second and third notes start from partials with density fluctuation, which are made by the previous note, they are easily attracted on the reference overtones.

6. CONCLUSIONS

In this paper, a real-time application for force-based sound synthesis, which is accomplished by using GPU-based parallel computing, is proposed. The utilization of GPUs is powerful and efficient for this synthesis method. Although it is difficult for off-the-shelf personal computers to generate high-quality results in real-time in this time, thus, some ideas for reducing the calculation cost are implemented. In this section, we provide the process for the real-time synthesis.

6.1. Preparations

1. Select one reference source sound which is previously detected frequency components
2. Calculate attractive forces from the reference spectrum
3. Setup the maximum number of partials and prepare the corresponding shader

6.2. Real-Time Processing

1. Receive MIDI note message
2. Scale the attractive force function corresponding to the note number.
3. Distribute partials corresponding to the velocity and ADSR function.
4. Calculate attractive and repulsive forces for each partial
5. Calculate speeds and frequencies for each partial
6. Store the new speeds and frequencies of the partials
7. Synthesize the sound for the current frame by using inverse fast Fourier transform (IFFT) [11]

6.3. Future works

The major limitation for this synthesis method is caused by the limit of the frame-rate, which depends on the capabilities of GPUs. GPUs are generally designed for manipulating visual data, therefore, the upper limit of the frame-rate is not enough for a high-quality sound synthesis. We consider that the development of the interpolation techniques between frames has value, though the improvement of these kind of processors are expected.

7. REFERENCES

[1] Ryoho Kobayashi, "Sinusoidal synthesis using a force-based algorithm," in *Proceedings of the 17th International Conference on Digital Audio Effect (DAFx-14)*, Erlangen, Germany, September 1-5 2014, pp. 19–22.

[2] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[3] Mark Harris, "GPGPU: General-purpose computation on GPUs," *SIGGRAPH 2005 GPGPU COURSE*, 2005.

[4] Lauri Savioja, Vesa Välimäki, and Julius O. Smith III, "Audio signal processing using graphics processing units," *Journal of the Audio Engineering Society*, vol. 59, no. 1/2, pp. 3–19, 2011.

[5] Lauri Savioja, Vesa Välimäki, and Julius O. Smith III, "Real-time additive synthesis with one million sinusoids using a GPU," *Audio Engineering Society Convention 128*, 2010.

[6] Pei-Yin Tsai, Tien-Ming Wang, and Alvin Su, "GPU-based spectral model synthesis for real-time sound rendering," in *Proceedings of the 13th International Conference on Digital Audio Effect (DAFx-10)*, Graz, Austria, September 6-10 2010, pp. 1–5.

[7] Apple Inc., "Swift - apple developer," Available at https://developer.apple.com/metal/, accessed March 15, 2016.

[8] Apple Inc., "Metal for developers," Available at https://developer.apple.com/swift/, accessed March 15, 2016.

[9] Thomas M. J. Fruchterman and Edward M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[10] John Adrian Bondy and Uppaluri Siva Ramachanda Murty, *Graph Theory with Applications*, The Macmillan Press Ltd., London, 1976.

[11] Jont B. Allen, "Short term spectral analysis, and modification by discrete fourier transform," *IEEE Transactions on Acoustics, Speech, and Processing*, vol. 25, no. 3, pp. 235–238, 1977.

[12] G. DiGiugno, "A 256 digital oscillator bank," in *Proceedings of the International Computer Music Conference*, MIT, Cambridge, 1976, pp. 188–91.

[13] F. Richard Moore, *Elements of Computer Music*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.

[14] John E. Stone, David Gohara, and Guochun Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science and Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[15] Jason Sanders and Edward Kandrot, CUDA *by Example: An Introduction to General-Purpose* GPU *Programming*, Addison-Wesley Professional, 2010.

[16] MIDI Manufacturers Association, *The Complete MIDI 1.0 Detailed Specification*, 1996.

[17] Pat Hanrahan and Jim Lawson, "A language for shading and lighting calculations," *ComputerGraphics*, vol. 24, pp. 289–295, 1990.

[18] Alécio P. D. Binotto, Joao L. D. Comba, and Carla M. D. Freitas, "Real-time volume rendering of time-varying data using a fragment-shader compression approach," in *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2003, pp. 69–75.

8. APPENDIX: SOUND EXAMPLES

Sound examples are available online at the following address.

http://www.ryoho.com/software/sinfba/