Real-time Synthesis of Footfall Sounds on Sand and Snow, and Wood

Chirag Mehta
chirag@chir.ag

Advisor: Dr. Dinesh Pai

Abstract

Since the early first-person shooter video games, prerecorded Foley sounds have imparted the desired aural realism. However, static sound clips have reached their limits and with the demand for increasingly realistic environments, sound clips must be rendered in real-time akin to 3D graphics. This paper proposes a rudimentary method to synthesize the sound of footfalls on various surfaces like sand, snow, and wood in real-time. Additionally, the paper describes a simple algorithm to generate footfalls given the weight and walking/running speed of the human. No digitally sampled files are used in the synthesis of any sounds. We propose a black box that generates the sound of walking, running, and jogging on sand, snow, and wood with minimal input parameters like weight, speed, crunchiness, grain-size, woodenness, creakiness etc.

1. Introduction

One of the most common methods to synthesize sound is by physical modeling. By applying their modal algorithm, Pai, van den Doel [1] et al., were able to render the sound of a ball dropping and rolling within a three dimensional virtual frying pan. With their JASS package, Pai, van den Doel [2] showed how modal algorithms can be used to render sounds like creaks and bells. We propose a simpler and less CPU-intensive algorithm to synthesize sounds such as knocks on wood, carpet, as well as wooden creaks that require only a fading sine wave function.

> *"All sound is an integration of grains, of elementary sonic particles, of sonic quanta."*
>
> -Xenakis (1971).

Another common method, which is often employed to enrich the texture of notes generated by music synthesizers, is granular synthesis. “Granular synthesis of sound involves generating thousands of very short sonic grains to form larger acoustic events” [Roads, 3]. “These sound ‘grains’ that are less than 50 ms in duration and typically in the range of 10-30 ms with typical grain densities from several hundred to several thousand grains per second; the grain itself may come from a Wavetable (e.g. sine wave), FM synthesis or sampled sound” [Traux, 4]. Granular synthesis is the most appropriate technique for rendering the sound of footfalls on sand, snow, and other granular surfaces. We will see how sound grains can be created using a simple white-noise generator.

While more complex methods can be implemented to produce even more realistic sound, the algorithms proposed herein have minimal processing demands and hence can be incorporated into CPU-intensive video games without major performance loss. In addition to games, footfall synthesis can be applied to virtual reality and film production.

2. Architecture

Figure 1 shows the basic architecture of the system. FootFallStream generates footfalls via live input from hardware, data files recorded previously, or using the FootModel footfall generator. FootFallSynth continuously retrieves the latest footfall from FootFallStream and synthesizes the sound using functions within the SoundGrainModel. SoundGrainPlayer renders the sound using Java Sound API. The sound rendered by SoundGrainPlayer may be directed to a standard ‘wav’ file format with minimal coding.

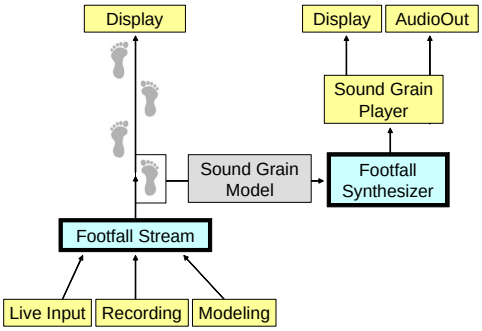

Figure 1: Footfall Synthesis Architecture

The Graphical User Interface (GUI) provides the user with the ability to change the walking/running speed, weight, surface crunchiness, compression (compactability), grain-size, carpet-thumps, woodenness, and creakiness in real-time. For debugging purposes, FootFallStream displays the footfall preview and FootFallSynth displays the waveform via an oscilloscope on the screen.

3. Synthesis Resolution

A single footfall is generated at a resolution of 16 cells x 44 cells (32x44 cells for two feet) as shown in Figure 2. Pressure values per cell approximately range from 0 (white) to 50 (bright red). Depending on the intensity of these pressure values, FootFallSynth generates the appropriate sound.

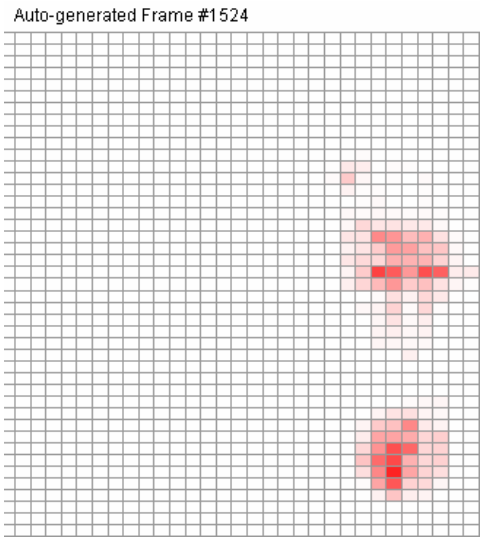

Figure 2: Footfall Preview

There are two ways to render sound given the array[32x44] of pressure cells. The first option, fast but less precise of the two, is implemented in FootFallSynthByFrame. It calculates the maximum, average, variance, and standard deviation of the pressure cells in the current frame and renders the compound sound by the algorithms that follow. The simpler, more accurate, but quite processor intensive technique is to generate an individual sound grain for each cell in the frame. FootFallSynthByCell (not implemented yet) can render granular sound in this manner.

Given the current state of technology and available processing power on a typical PC, we shall restrict our attention to FootFallSynthByFrame.

4. Synthesis Engine

At the core of the synthesis engine, SoundGrainModel, lie two sound grain generators: White Noise and Fading Sine Wave. Input parameters for White Noise are length/duration, frequency, and amplitude. For Fading Sine Wave the inputs are length, frequency, and the starting amplitude; ending amplitude is zero.

Sounds created by friction, i.e. scratching, rubbing, walking on sand and snow etc., seem to have almost no distinguishable wave patterns or frequencies. Being a mix of thousands of frequencies, we show that synthesis of such sounds can be approximated using white noise set at the appropriate frequency and amplitude. Lower frequencies of white noise generate sounds similar to winds and ocean waves, whereas higher frequencies sound akin to bad reception on FM radios. Gaussian white noise and pink noise yielded results very similar to pure white noise and hence were not implemented in the generators.

The Fading Sine Wave starts at given amplitude and frequency and fades out linearly to silence, zero amplitude. Lower frequencies of fading sine waves sound such as thumps, knocks, and creaks while higher frequencies sound like pops, electric bells, and water drops.

For the following algorithms, the FootFallStream returns the latest frame to render, given the user-selected speed and weight parameters. The sound is generated at 44.1 kHz bit rate. The FootFallStream speed (frames per second) is in the range of 25fps - 75fps.

Model Parameter:

```
Grain Length: Bit Rate / FPS
```

Consequently, the number of sound grains per second is 25-75 with respective durations between 40ms and 13ms. Depending on the speed of the processor, multiple grains may be generated for a single frame to enrich the sound texture.

4.1. Synthesizing Sand

The smaller the grain size is, the closer the sound is to being pure white noise. As grains increase in size, physical models approximate their sound with greater accuracy. The process

of rendering sand grains mostly consists of playing white noise at the appropriate frequency and amplitude.

Model Parameters:

```
Pressure : Frame Max ÷ Stream Max
HeelToe  : (1-(1÷StdDev))

Amplitude: Grain-size
           * Pressure
           * HeelToe

Frequency: 44.1 * 0.4 = 17.6 kHz

WhiteNoise(Duration, Amp, Freq)
```

Pressure (from 0 to 1) is used to determine how strongly the foot is being pressed onto the surface in a given frame, compared to the maximum value possible for the stream. The amplitude is the product of user-selected grain-size (from 0 = silence to 1 = loudest) with pressure and a measure of heel-toe impact. When the whole foot is pressed against the surface, variance is low and HeelToe is generally small. When heel or toes are strongly pressed, variance is high and HeelToe is large. The sound of footfalls on sand is generated whenever the heels or toes hit the surface in a walking/running motion (high HeelToe) or when feet are firmly on the ground (high pressure), as if sinking in the sand. For sand, the most realistic white noise frequency was found to be at 17.6 kHz by experimentation.

4.2. Synthesizing Snow

Crunching is defined as the high-pitched sound made when one steps on snow, breaking through its different layers. Compression is the low-frequency sound created due to compacting of these layers into ice or icy slush. Compression takes places as long as more pressure is being applied in each frame, while crunching happens throughout the duration of the footfall.

Model parameters for Compression:

```
Pressure : Variance ÷ Total Cells
Amplitude: Compression * Pressure²
Frequency: 1.1 kHz to 2.2 kHz
           (50 grains)

If Pressure has increased:
  WhiteNoise(Duration, Amp, Freq)
```

Model parameters for Crunching:

```
Pressure : Variance ÷ Total Cells
Amplitude: Crunchiness * Pressure²
Frequency: 0.8 kHz to 44.1 kHz
           (50 grains)

If Pressure has changed:
  WhiteNoise(Duration, Amp, Freq)
```

Calculate the pressure (0-1) as the Variance ÷ Total number of non-blank cells. Here the heels and toes are even more pronounced than Frame Max ÷ Stream Max. Amplitude is the product of user-selected compression or crunchiness parameters and pressure, squared for increased emphasis. Low values of pressure result in almost no sound whereas high values result in loud sound. Instead of rendering just one grain (like for sand), snow sounds best when between fifty and a hundred grains are used for rendering compression and crunchiness, each generated within a range of frequencies.

4.3. Rendering Wood

A StreamAnalyzer is used to detect peaks in the variance of FootFallStream frames. At every peak a single sound grain for wood knocking and carpet thump sound is generated as follows.

Model Parameters for Knocks:

```
Pressure : Frame Max ÷ Stream Max
Amplitude: Woodenness * Pressure

Frequency: 150 Hz
           + 250 * Woodenness
           + 350 * Pressure

Range    : 150 Hz - 750 Hz

If Peak detected
  FadeSineWave(Duration, Amp, Freq)
```

Model Parameters for Carpet:

```
Pressure : Frame Max ÷ Stream Max
Amplitude: Carpet * Pressure

Frequency: 50 Hz
           + 50 * Carpet
           + 50 * Pressure

Range    : 100 Hz - 300 Hz

If Peak detected
  FadeSineWave(Duration, Amp, Freq)
```

Creaking is handled differently, without peak-detection, since it is a continuous sound. The primary difference between creaking and knocks or carpet is the duration.

Model Parameters for Creaking:

```
Duration : Bit Rate / 400

Pressure : Frame Max ÷ Stream Max
HeelToe  : (1-(1÷StdDev))

Amplitude: Creakiness
           * Pressure
           * HeelToe

Frequency: 4 kHz to 9 kHz

If Pressure has changed:
  FadeSineWave(Duration, Amp, Freq)
```

Creaks are very short duration sounds generated by stick-slip oscillations. We simulate rudimentary creaking by the same FadeSineWave function as knocks and carpet thumps.

5. Generating footfalls

Taking the footprints in Figure 3 as the original frame, a very simple algorithm was applied to generate sequence of footfalls with each call to GetFrame(), depending on the speed and weight of the human.

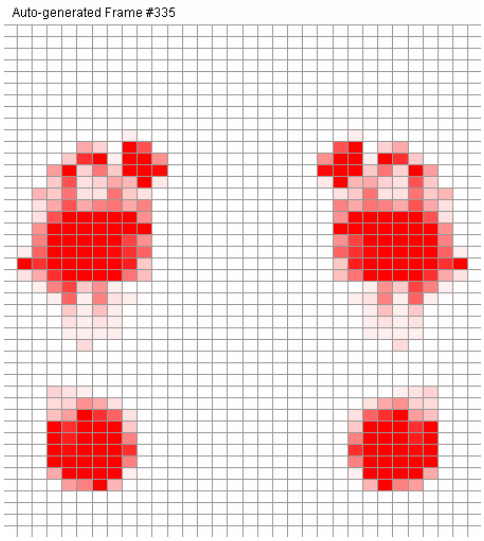

Figure 3: Original Footprints

```
Initialize():

  Load FootOriginal
  CurrentPosition = Heel

End Initialize()

GetFrame():

  Weight       = User-selected (0-1)
  HeelPressure = Proximity to Heel

  FallOff      = Distance from
                    CurrentPosition

  MidPoint     = Distance from
                    Center-axis of Foot

  CellPressure = FootOriginal[Cell]
                  * Weight
                  * HeelPressure
                  * FallOff
                  * MidPoint

  Render Each Cell

  CurrentPosition = Move Up With Speed

  If CurrentPosition = Top Then
     CurrentPosition = Heel

  Return FootFall Frame

End GetFrame()
```

6. Demo

Full source-code with Java-implementation of this algorithm is available at the following URL:

http://chir.ag/493

Require JRE 1.4.2 or above with soundcard. JavaWebStart preferable but not required.

7. Conclusion

More work is required to improve the sounds of creaking. The system doesn't render sound correctly at very low values for weight. With the completion of the FootFallSynthByCell algorithm, both these issues can be resolved.

References:

[1] K. van den Doel, P.G. Kry and D.K. Pai, "FoleyAutomatic: Physically-based Sound Effects for Interactive Simulation and Animation", in Computer Graphics (ACM SIGGRAPH 01 Conference Proceedings), Los Angeles, USA, 2001, August 12-17.

[2] K. van den Doel and D.K. Pai, "JASS: A Java Audio Synthesis System For Programmers", in Proceedings of the 2001 International Conference on Auditory Display, Espoo, Finland, 2001, July 29-August 1.
http://www.cs.ubc.ca/~kvdoel/jass/jass.html

[3] C. Roads. Granular synthesis of sound. In Foundations of Computer Music. C. Roads and J. Strawn, eds. MIT Press, Cambridge, pp. 145—159, 1985.

[4] Truax, B. (1988) Real-time granular synthesis with a digital signal processor. Computer Music Journal, 12(2), 14-26.
http://www.sfu.ca/~truax/gran.html

Bibliography:

P. Cook, "Modeling Bill's Gait: Analysis and Parametric Synthesis of Walking
Sounds," Proc. Audio Engineering Society 22 Conference on Virtual, Synthetic
and Entertainment Audio, Helsinki, Finland, June 2002.

Bencina, R. (2004), "Implementing Real-Time Granular Synthesis." In Greenbaum (Ed), Audio Anecdotes, A.K. Peters, Natick.

Castle G., Adcock M., Barrass S., "Integrated Modal and Granular Synthesis of Haptic Tapping and Scratching Sounds.